Fluid Music

A New Model for Radically Collaborative Music Production

Charles Joseph Holbrow

Master of Science in Media Arts and Sciences at Massachusetts Institute of Technology, 2015

Bachelor of Music at University of Massachusetts Lowell, 2008

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning, in partial fulfillment of the requirements for the degree of Doctor of Philosophy at the Massachusetts Institute of Technology

September 2021

Author

Charles Joseph Holbrow
Program in Media Arts and Sciences
20 August 2021

Certified by

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

Accepted by

Tod Machover
Academic Head, Program in Media Arts and Sciences

Fluid Music

A New Model for Radically Collaborative Music Production

Charles Joseph Holbrow

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning, on August 20, 2021, in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

Twentieth century music recording technologies were invented to capture and reproduce live music performances, but musicians and engineers used these new tools to create the art of music production, something distinctly different from – and well beyond – simple archiving. Are current technologies poised to bring about new kinds of musical experiences yet again? Could these new kinds of music be every bit as transformative and impactful as recorded music was in the 20th century? Fluid Music proposes one possible trajectory by which this could happen: harnessing the internet's power for massive asynchronous collaboration in the context of music production.

This dissertation articulates how Fluid Music proposes new tools for computational music production, prioritizes human agency via audio production paradigms found in digital audio workstations, and rejects existing collaborative processes like remixing and crowdsourcing. It describes the Fluid Music framework, a software toolkit which provides the foundation to build, experiment, and study Fluid Music workflows. It discusses the long-term goals of Fluid Music, including the construction of a massive, open repository of music production Techniques that sound designers, producers, and composers can use to share malleable sound worlds without reimplementing complex music production processes, demonstrating how design choices in the Fluid Music framework support the project's larger objectives. One consequence of these design choices is that Fluid Music encapsulates the art of music production in the same way that recorded music encapsulates the art of music performance.

The dissertation lays out next steps that are required for Fluid Music to grow into an entirely new art form which is clearly distinct from the recorded music that is pervasive today.

Thesis Advisor: Tod Machover
Muriel R. Cooper Professor of Music and Media
Program in Media Arts and Science,
Massachusetts Institute of Technology

Fluid Music

A New Model for Radically Collaborative Music Production

Charles Joseph Holbrow

This dissertation has been reviewed and approved by

Thesis Reader:

Tristan Jehan
Founder, CEO
Elis OS, Inc.

Fluid Music

A New Model for Radically Collaborative Music Production

Charles Joseph Holbrow

This dissertation has been reviewed and approved by

Acknowledgments

I would like to thank Tod Machover for over a decade of support, guidance, and encouragement, and for sharing his musical thinking, which influenced every part of Fluid Music. I will do my best to pass it on.

I would like to thank committee members Tristan Jehan and Nancy Baym, for crucial guidance, support, and perspectives, and for their part in ensuring that Fluid Music is something I can be proud of.

I would like to thank my family, who made my time at the Media Lab possible in countless ways, and especially Helen, who also made it possible for me to graduate Helen Corless, Mark and Gwen Holbrow, Charlie H. and Mary Holbrow, and Hilary, Giles, and Felicity Holbrow.

I would like to thank my Opera of the Future family for welcoming me in 2008, carrying me over the finish line in 2021, and filling the journey with inspiration, creativity, and friendship: Ben Bloomberg, Rébecca Kleinberger, Nikhil Singh, Aarón Montoya-Moraga, Karsten Schuhl, Manaswi Mishra, Nicole L'Huillier, Alexandra Rieger, Akito Van Troyer, Priscilla Capistrano, Sizi Chen, Kelly Egorova, Simone Ovsey, Peter Torpey, Elly Jessop, Zhi Wei Gan, Catherine Zeng, Ethan Nevidomsky, Hane Lee, David Su, Bryn Bliska, Sarah Platte, Thomas Sanchez Lengeling, David Nuñez, and Janice Wang.

I would like to thank colleagues, friends, and community members, who offered guidance when I asked, and shared both the most difficult and the most joyful parts of the Media Lab: Joe Paradiso, Deb Roy, Ethan Zuckerman, Brian Mayton, Spencer Russell, Harpreet Sareen, Sunanda Sharma, Sang-won Leigh, Udayan Umapathi, Pedro Reynolds-Cuellar, Cindy Hsin-Liu Kao, Joy Buolamwini, Tim Savas, Alexis Hope, Savannah Niles, Shannon Johnson, Paula Aguilera, Jonathan William, Jimmy Day, Bianca Datta, Caroline Jaffe, Jon Ferguson, Sarra Shubart, Linda Peterson, Keira Horowitz, Monica Orta, Lily Zhang, Rebecca Cuscaden Marvin, and Becca Bly.

I would like to thank many others who trusted, aided, and supported me in big, small, or unexpected ways, including: Larry and Terry Sass, David Singer, The MacGregor House Team and the E-Entry Chairs and students, Meghan Moriarty and the KTCP, Mashi Hasu, Alex Case and the UMass Lowell SRT program, The Audio Engineering Society, Shawn Drost, Hack Reactor, The Recurse Center, Sudbury Valley School, Jo and Richard from Interfacio, David Rowland from the JUCE forums, Justin Frankel of Reaper fame, the Lynch Family Skatepark and the Lynch Roller Rats, and, finally, Ben, Gene, and Jeff for the best screams I ever recorded.

Table of Contents

1 Introduction

In the future, when we can look back on our present time and draw conclusions about the implications of the internet and connected digital media technologies, how will we understand and describe the impact that these technologies had on music and musical creativity? What are the major implications of digital networking and connected media technologies for music? Large scale digital networking technologies have existed for approximately 50 years. The World Wide Web was created just over 30 years ago. Is a half century long enough for a new communications medium to mature? The implications of sound recording technologies were still emerging when The Beatles released *Sgt. Pepper's Lonely Hearts Club Band* in 1967, a full 90 years after Edison's first phonograph, which was developed in 1877. Connected media technologies are still changing quickly, just as music production technologies were still changing quickly in the 1960s. We do not yet understand what connected media technologies can do for music and media. This dissertation describes a new connected music technology, Fluid Music, and shows how this technology can transform the way that we create and consume music by empowering music creators with fundamentally new capabilities.

New media technologies present both an opportunity and a threat to practicing musicians, composers, and media organizations. Consequently, when new media technologies are introduced, they are often received by their contemporary audiences with a mix of measured anxiety and hyperbolic enthusiasm. This was the case for the telegraph, the player piano, music recording technologies, radio, the internet, mp3s, artificial intelligence, and others.

Fears about new media technologies are not unjustified. New technologies can shift power toward artists, entrepreneurs, and early adopters who are unusually lucky or insightful and who can take advantage of the affordances of new media before their contemporaries. Musicians and music professionals who do not (or cannot) adopt new technologies can be outmaneuvered or simply obscured by the many artists competing for audiences and their attention. However, enthusiasm for emerging media technologies and for their potential to create entirely new forms of art, media, and music is equally valid, as the history of recorded music illustrates.

In the past, music studio production technologies enabled new processes for creating music, which in turn enabled musicians, composers, and engineers to develop a new art form – the art of music production. Music production gave us the ability create instantly recognizable sound worlds using the tools in a music production studio. It is the art of music production that allows you to recognize a song on the radio within a second or a fraction of a second – long before you parse harmonic, melodic, or rhythmic material. However, these sound worlds are locked inside the static recordings that contain them. Once a recording is produced, it is immutable, and there is no way to access, change, or reuse the core components that make up its sonic fingerprint without reverse engineering and repeating the production process, which is difficult, time-consuming, and error-prone. Fluid Music is a new media technology that can be used to describe produced music in terms of the core components and processes that make up a sound world. It enables musicians to create, share, and mutate sound worlds. This dissertation describes the underlying Fluid Music technology and shows how musicians, composers, and engineers can use this technology to create a new form of musical art yet again.

This dissertation distinguishes between three parts of Fluid Music: the concept, the framework, and the art form. Conceptually, Fluid Music is about using the collaborative affordances of the internet to create a notational language for music production and sound design that anyone can contribute to, and then using this notational language to compose music. The framework is a software toolkit that makes it possible to explore and study the Fluid Music concept by creating new collaborative and computational systems for music production. This document describes the framework as it exists today and shows how this framework opens a path to a fundamentally new art form. This path is partially predictive and partially prescriptive. It is in part a prediction of where music will go in the future, based on a study of how music and media technologies changed in the past in addition to the trajectories that are emerging in the present. However, Fluid Music is also prescriptive insofar as it is the result of my own attempt to nudge the trajectory of musical media in a way that I perceive to be favorable. If, in the coming decades, produced music becomes more *fluid* (dynamic, adaptable, malleable), *who* will decide *how* music adapts and changes? The current trajectory suggests that organizations with the resources necessary to train neural networks with billions of parameters will have an outsized influence on what we listen to and how it sounds. Fluid Music aims to nudge this current trajectory in a way that gives individuals and music creators more influence and control. It leans toward the collaborative model used by Wikipedia, which allows many people to work together to create something that is greater than the sum of its parts. This is the radically collaborative model for music production. However, encyclopedia entries are very different from music production processes, and it would not be effective to try to copy the collaborative model used by Wikipedia and apply it to music. Fluid Music gives us the tools to create and study new collaborative processes that are tailored for music and music creators.

1.1 Configuration, Computation, and Collaboration

Existing music production technologies are configurable. In the conventional music production model, an audio engineer adjusts the configurable parameters of audio production software and hardware to achieve certain musical results. Fluid Music extends this model by adding a layer of computation and a layer of collaboration. The layer of computation makes music production processes more reusable by enabling them to perform complex behaviors and adapt to different contexts. The layer of collaboration enables users to publish music production processes to a global repository and reuse the processes published by others.

The Fluid Music software framework described in this thesis provides a foundation for users to collectively experiment and discover what kinds of production processes can be automated with computation; what kinds can be published, shared, and reused; and what kinds of processes we want to continue to configure using a conventional approach. The framework also supports the development of hybrid combinations of computation, configuration, and collaboration, and this dissertation describes some initial experiments in this area.

Fluid Music has roots in the history of media; it embraces the future without disregarding the past. Fluid Music enables advances in machine listening and artificial intelligence (AI) to play an important role in music production, either by automating portions of the music production process, i.e., by creating new sounds, timbres, and instruments, or by generating portions of a digital musical score. However, it does not depend on AI, nor does it exclude other innovations from the last 70 years of digitized algorithmic composition and sound design. Importantly, Fluid Music ensures that users can depend on the familiar tools in a professional Digital Audio Workstation (DAW) to study and override the results of algorithmic and computational production processes.

Fluid Music recognizes the important role that DAWs play in modern music production, but it does not depend on DAWs – you do not need a DAW to create Fluid Music. Fluid Music borrows six conventions from DAWs: tracks, clips (audio and MIDI), routing, automation, plugins, and sessions. The Fluid Music software framework exposes computational access to these core DAW components through a simple but powerful application programming interface (API). This API enables developers to write code that performs the music production tasks that would normally be accomplished in a DAW with a graphical user interface (GUI). However, because Techniques (see below) are written with code, they can also perform complicated and elaborate processes that would not be feasible when working exclusively with a DAW through a GUI.

In the Fluid Music API, a music production task is called a Technique. A Technique is a configurable process which performs computation and actions within a DAW session. Unlike plugins, which are hosted inside a DAW and have limited access to the contents of a DAW session, Techniques can affect every part of a session. Techniques have similarities with the macros that are available in some DAWs but are much more powerful.[1] From the perspective of the Fluid Music framework, all Techniques are invoked with the same API function call. This enables users to share each other's Techniques without needing to understand the full details of how a Technique is implemented. Users can create derivative Techniques that modify or extend existing Techniques. Users can also publish collections of Techniques via the internet.

[1] Unlike DAW macros, Techniques can embed logic, inspect and modify the full underlying session, and take advantage of IDEs and external libraries. Techniques are covered in more detail in sections 3.1 and 4.2.2.

Every new Technique Library that is published expands the sonic vocabulary of the Fluid Music language.

Writing Techniques is not the only way to extend Fluid Music. The framework also provides the foundation needed to build interfaces that can be used to make Fluid Music. A Fluid Music interface is simply a way to create a DAW-like session instance, and then sequence Technique invocations which populate that session instance with content. The framework includes one such interface, which is a dedicated score API for sequencing Techniques on a musical timeline using pure JavaScript. This code-based interface is not intended to be the only way to create Fluid Music, but the score API is expressive and powerful enough to write compelling music with, and it was used for experimental collaborative music production, as well as a workshop and a user study, all of which are documented in this dissertation.

The name "Fluid Music" comes from two important characteristics: its flexibility and its dependency on the collaborative affordances of the internet. At a high level, Fluid Music enables users to author and share Techniques, and to author and share interfaces for sequencing Technique invocations. Because Fluid Music Techniques describe configurable *processes*, they are flexible and reusable in ways that static audio files, DAW sessions, presets, or templates – which store *state* – are not. The word "fluid" extends the "liquid" metaphor in "streaming" music. It also hints at an underlying hypothesis about the future of connected music technology: In the coming decades, music will become more dynamic, adaptable, malleable, and more *fluid* than the static stereo recordings that we still take for granted today.

1.2 Overview

Following this introductory chapter, chapter 2 shows how I arrived at the Fluid Music concept. It covers history, technology, music, and scholarship that motivated and influenced Fluid Music. It is framed with the history of magnetic tape recording, and the ways that magnetic tape and studio recording co-evolved in the twentieth century. It covers the origins of the recording studio in radio and briefly highlights some similarities between the impact of the radio technologies that enabled recorded music and the internet technologies that enabled Fluid Music. It describes early experiments with tape and studio technologies, and then covers a history of the idea that technology will engender transformative new music and media by reviewing the writing and thinking of historical figures like Thomas Edison, the Italian futurists, Daphne Oram, Glenn Gould, and others. Finally, it discusses the stream of developments in digital technologies that make Fluid Music possible.

Chapter 3 describes what Fluid Music is in more detail. It reviews remixing, crowdsourcing, and the collaborative processes that are possible with conventional DAWs, showing how Fluid Music is different from these existing options. It also clarifies the distinction between the concept of Fluid Music, which pertains to the ideas and goals of the project, and the Fluid Music framework, which is my software implementation. The chapter ends with analysis of the production used in two compositions. These examples illustrate the kinds of music production Techniques that can be described with Fluid Music.

Chapter 4 is dedicated to the Fluid Music framework, which is the main technical contribution of this dissertation. The Fluid Music framework has similarities with other code-based languages for computational sound like CSound and SuperCollider. However, similarities with other code-based languages for sound design are deceptive. Fluid Music is different from these other languages in its design, goals, and capabilities. This chapter shows what makes the framework a uniquely suited foundation for Fluid Music.

Chapter 5 describes the music that I composed using the Fluid Music framework. It also describes my music collaboration with composer Nikhil Singh and discusses what we learned from our experiments with some of the unique collaborative features of the framework.

Chapter 6 describes a Fluid Music workshop I ran, which taught others how to use the Fluid Music framework. It also describes the process and results of a user study which evaluated how audio professionals respond to both the Fluid Music concept and the Fluid Music Framework. In the long-term, Fluid Music will only work if it is adopted by users. The user study was an opportunity to evaluate my assumptions about what would make Fluid Music useful. It covers both where my assumptions were validated and where they were contradicted.

Finally, chapter 7 covers the next for lead Fluid Music that will lead it to a fundamentally new form of art.

2 Background

In the 1930s, German engineers developed a portable reel-to-reel audio recorder known as the magnetophone.[2] At the time, most reel-to-reel audio recorders used steel tapes as a recording medium, but the magnetophone recorded to reels of plastic or strong paper that were coated or embedded with microscopic iron particles (Morton 114). The choice to avoid steel was a practical one; paper and plastic could be manufactured domestically, while steel tapes had to be imported. When German forces invaded Poland in 1939, instigating the Second World War, the magnetophone, which was manufactured with German components, became valuable war-time communications technology, in part because it did not depend on imported components.

Early models of the magnetophone did not sound better than other recording technologies, but the device benefited from many technical improvements during the war (Morton 115) when it was used extensively for surveillance and propaganda (Morton 116; Bergmeier and Lotz 215). By the end of the war, the most advanced magnetophones could record and reproduce music that sounded as good as any other recording technology, including the disc recorders that were used for recording music at the time. When U.S. Army Signal Corps member Jack Mullin[3] heard the magnetophone first-hand in a German radio facility just as the war was ending, he was amazed by the quality. He wrote about the experience in a Billboard magazine article published in 1972:

> *The technician placed a roll of tape on one of the machines and started it. Suddenly, out of complete silence, an orchestra blossomed into being with fidelity such as I had never heard in my life. From deep resonant brass to the shimmering of the flute, it was all there. It was clean! It was free from any noticeable distortion. And if that were not enough, the dynamic range was fantastic compared with anything I had ever previously experienced (Mullin).*

Mullin was assigned the job of seizing German technology and sending it to the United States through a US Army agency called Field Intelligence Agency Technical (FIAT). He arranged to have two of the machines and 50 reels of magnetic tape sent to America. In late 1945, the US Department of Commerce declassified FIAT technical reports (Morton 120), giving American manufactures access to the German trade secrets that were effectively spoils of war. Magnetic recorders based on the German design soon made their way into recording studios across the country and beyond.

The new generation of tape recording had many advantages over the direct-to-disc recording processes that were used by music recording studios at the time. The low signal-to-noise ratio

[2] In German, the device is spelled Magnetophon. I will use the English spelling, magnetophone.
[3] Jack Mullin's full name is John T. Mullin, but many sources refer to him by his nickname, Jack.

made these machines useful for accurately capturing and reproducing music, but there was something else about the technology that made it more flexible, more hackable than the alternatives. As the technology continued to develop in the decades after the war, the sonic malleability it enabled increasingly affected music itself. By the end of the 1960s it was clear that recorded music was something more – or perhaps something different – from the capture and reproduction of a musical performance.

The history of studio recording is longer and more complicated than this story implies. However, the significance of analog tape in twentieth century music is not overstated. Analog tape machines enabled multitracking, overdubbing, editing, and variable speed playback. Coupled with the improved sonic fidelity of the magnetophone, these traits were essential to the art of sound recording as it developed in the years following World War II, and the influence of these technologies can still be heard today in contemporary music of all kinds.[4] The history of magnetic tape illustrates how a new technology can be instrumental in unlocking a new artistic medium with its own unique affordances and creative vocabulary.

Current and emerging technologies can have an equally transformative effect on music. The result will be what I call Fluid Music. What can the current generation of media technologies do *for* music and musicians? It is too soon to know for certain, but there are several areas of opportunity. My vision of Fluid Music focuses on one of these areas: the potentials of the internet and connected media technologies. My long-term vision is for Fluid Music to become an open repository of reusable music production tools and techniques. When fully realized, this will enable musicians and music producers to create, share, and reuse malleable "sound worlds" as easily as software engineers who share and reuse software modules on the internet.

Fluid Music is built on and around the affordances of studio music recording, which itself was built on and around the affordances of radio broadcast technologies. Section 2.1 discusses radio and its influence on studio recording. Section 2.2 discusses the origins of studio recording and production in avant-garde composers, and engineers, who after World War Two, experimented with new possibilities brought about by new recording technologies.

The Fluid Music framework is a step toward a larger goal. The framework is practical and tangible. It can be used and evaluated today. My larger goal, however, is more speculative. This kind of speculation about media technologies has a long and interesting history. In the past, artists, journalists, and practitioners speculated about the potential impact of new media technologies. In some cases this speculation was exaggerated and hyperbolic, and on more than one occasion it led to unrealistic expectations for the kinds of creative or cultural transformation that the new media technologies could introduce. The review in section 2.3 of

[4] For history and context about the interplay between music and technology in the twentieth century, see Warner; Braun; Morton.

some of the historical speculative writing about new music and media technologies of the past aims to guard against making techno-utopian predictions about the potential of Fluid Music and other new media technologies.

2.1 Radio

In the first half of the twentieth century, radio profoundly shaped cultural discourse in America and Europe. A book titled *Voices in Ruins: West German Radio Across the 1945 Divide* explains why this was the case in Germany, claiming “Almost without interruption from the mid-1930s to the late 1950s, the radio was not only the primary source of information, but also one of the cheapest sources of entertainment and one of the wealthiest supporters of culture” (Badenoch 1). German composer Kurt Weill corroborated the cultural significance of radio in a 1926 essay titled *Radio and the Restructuring of Musical Life.* A translated version of the essay (Hermand and Steakley 262) opens with this strong statement:

> *Within a remarkably short period of time, radio has become one of the most essential elements of public life. Today, it is one of the most frequently discussed topics among all segments of the population and in all organs of public opinion.*

Weill went on to predict that radio would bring about new kinds of instruments, music, and musical experiences. He believed these instruments and experiences would shape and be shaped by the affordances of radio technology, writing:

> *A special technique of singing and playing for radio purposes will develop; and sooner or later we will begin to find special instrumentations and new orchestral combinations suited to the acoustic requirements of the broadcast studio. And we can’t yet foresee what new types of instruments and sound producing devices may develop on this foundation.*

In retrospect, Weill’s predictions were largely accurate, although the evolutionary path that music and music technology would follow had other influences overlapped and interlaced with radio. One way that radio affordances influenced music was through the Hörspiel artform (Cory and Haggh). The term was coined in 1924 to describe a kind of theatrical radio play narrated by voice actors with music and sound effects sparsely interwoven to support the dramatic narrative. Jennifer Iverson (37) points out that while Hörspiel is not typically included in histories of electronic music, multiple well-known electronic composers like Luciano Berio and John Cage had their first experiences with audio technology while creating sounds for radio dramas. Additionally, Iverson points out that Pierre Schaeffer’s writing reveals a direct connection between Hörspiel and Schaeffer’s early experiments with audio sampling. In his book *In Search of a Concrete Music*, originally published as *A la recherche d'une musique* *concrète* in 1952, Schaeffer describes how he conceived musique concrète as, essentially, Hörspiel without speech:

> *I go to the sound effects department of the French radio service. I find clappers, coconut shells, klaxons, bicycle horns. I imagine a scale of bicycle horns. There are gongs and bird calls. It is charming that an administrative system should be concerned with birdcalls and should regularize their acquisition on an official form, duly recorded. I take away doorbells, a set of bells, an alarm clock, two rattles, two childishly painted whirligigs. The clerk causes some difficulties. Usually, he is asked for a particular item. There are no sound effects without a text in parallel, are there? But what about the person who wants noise without text or context? (4)*

By borrowing the foley equipment from the sound effects department at the *Radiodiffusion Française* (French national radio) studio and using that equipment to create musical collages of sound, Schaeffer may be the first composer to explore the (now common) field of "sampling" – that is using recorded sounds as malleable ingredients in a new composition. His intention was to broaden the meaning of "music" to include recorded and manipulated sounds.

On October 5, 1948, Schaeffer's *Cinq études de bruits* (Five studies of noises) was premiered via radio broadcast. The five tape compositions mixed sounds of trains, saucepans, singing, speech, and canal boats with sounds made with extended playing techniques on traditional musical instruments. Audiences were largely bewildered by the unconventional, noisy compositions. However, due to the political and social atmosphere described in section 2.2 below, contemporary composers of the time were eager to use similar techniques and push those techniques further.

2.1.1 Similarities Between Radio and the Internet

The history of radio is relevant to Fluid Music in part because radio technologies were appropriated and augmented for recording studios, and it was in recording studios that recorded music became the art form that we are familiar with today. However, there is an additional reason that radio is particularly relevant to Fluid Music. Fluid Music draws on the potential of the internet for music. As media of mass communication, radio and the internet have interesting commonalities. Kurt Weill commented on how quickly radio integrated into everyday life. Among U.S. Households, internet adoption went from 10% to over 80% in the twenty years between 1993 and 2013, and radio adoption followed in this same pattern, jumping from 10% to over 80% between 1925 and 1945 (Ritchie and Roser). Like the internet, radio was initially seen as a public service akin to a library. As radio became popular, it also became more commercialized, and powerful and influential media corporations emerged leading to pushback over the commercial adoption, and finally, regulation (Wu). In the process, radio became a powerful and effective tool for propaganda (Adena et al.; Badenoch). The internet followed a similar trajectory, both in terms of commercial adoption and its use for disseminating propaganda. Finally, both technologies opened new creative possibilities for music and media.

2.2 New Music After World War II

During World War II, the Nazi regime aimed to portray classical music by white, Aryan European composers as the most advanced form of music. They did this in part by controlling the music that was performed and played on the radio, and suppressing any music that they deemed a threat, especially music by Jewish composers. This was part of the Nazi propaganda campaign and was intended to justify notions of racial and cultural superiority by positioning the German and Austrian composers at the top of a musical hierarchy. In the words of Alex Ross, Nazi party rallies "were so immaculately choreographed to Beethoven, Bruckner, and Wagner that the music seemed to have been written in support of the pageantry" (343).

When the war ended, the U.S. embarked on a systemic re-education campaign, attempting to discredit the Nazi party in Europe and foster public support for liberal democratic values. This campaign sought to influence public opinion by sponsoring concerts, composers, and musical events that presented an alternative to the music that had become associated with the Nazi Party. The efforts by the U.S. government achieved equivocal results (Ross 373-385). However, one institution initially backed with American support was influential among composers. This was the annual *Darmstädter Ferienkurse* or *Darmstadt Summer Course for New Music*, a conference series held in the town of Darmstadt. The name "Darmstadt School" came to refer to a group of composers, including Karlheinz Stockhausen and Pierre Boulez, who participated in these courses and (at least in the initial years of the conference) saw serialism as the answer to (and the rejection of) the Nazi's conservative and authoritarian policies surrounding music.

For composers in the Darmstadt School, and other young composers working to establish themselves in the aftermath of the war, electronic music was also seen as a necessary break from the traditional thinking that enabled Nazis to come to power in the first place. In the words of Iverson:

> *As a result of the Nazi history and the Socialist Realist prescriptions, many young West German composers found it unthinkable to reengage with traditional musical forms and aesthetically conservative idioms. In such a climate, electronic music appeared to be a necessity. The only morally acceptable position was on the avant-garde (4).*

It was in this environment that the *Westdeutscher Rundfunk* (WDR), or West German Radio studio, was established in Cologne. The facilities at WDR were deliberately designed to rival the studio that Pierre Schaeffer was using at the French National Radio studio in Paris (Iverson 38–

39). To distinguish itself from the French National Radio Studio, the WDR was outfitted with the equipment for recording synthetic sounds in addition to the equipment for recording and manipulating acoustic sounds (Cross). The studio opened in 1953 and quickly became a hub of electronic music.

The best-known composition created at the WDR studio was *Gesang der Jünglinge* by German composer Karlheinz Stockhausen. Stockhausen had previously worked under Pierre Schaeffer at the French National Radio Studio, and he also attended the Darmstadt Summer Courses for the first time in 1951. Karlheinz Stockhausen finished *Gesang der Jünglinge* in 1956. He and his assistant, Gottfried Michael König, had been working on the piece for over a year, meticulously layering the output of a sine tone generator in harmonic series to create dense harmonic clusters, and splicing these clusters with filtered noise and impulses, all of which were generated with radio test equipment. Taking advantage of the technical capabilities at the WDR studio, the piece blends the synthesized timbres with melodic vocal recordings of 11-year-old Josef Protschka.

Stockhausen described his approach explicitly, saying *"Sprache kann sich Musik, Musik kann sich Sprache nähern bis zur Aufhebung der Grenzen zwischen Klang und Bedeutung"* (Cory and Haggh), which translates to "Speech can become closer to music, and music can become closer to speech, right up to the border between sound and meaning." In *Gesang der Jünglinge: History and Analysis*, John Smalley described the results:

> *Gesang der Jünglinge, is arguably the first real masterpiece of electronic music, a piece whose complexity and drama far surpassed anything the medium had yet produced. Indeed, the composer's unique synthesis of vocal and electronic sounds marked a quantum leap beyond the musique concrète techniques practiced in Paris and elsewhere.*

2.3 Rhetoric and Prognostication

Every new media technology invites speculation about its creative impact. In hindsight, some speculation sounds exaggerated and hyperbolic. Sometimes it is visionary and prescriptive. Often it is a mix of the two. By studying how historic media technologies were portrayed when they were still new, we gain points of comparison which are helpful for understanding Fluid Music, which also aims to understand the potentials of new media technology.

2.3.1 The Telegraph and the Internet

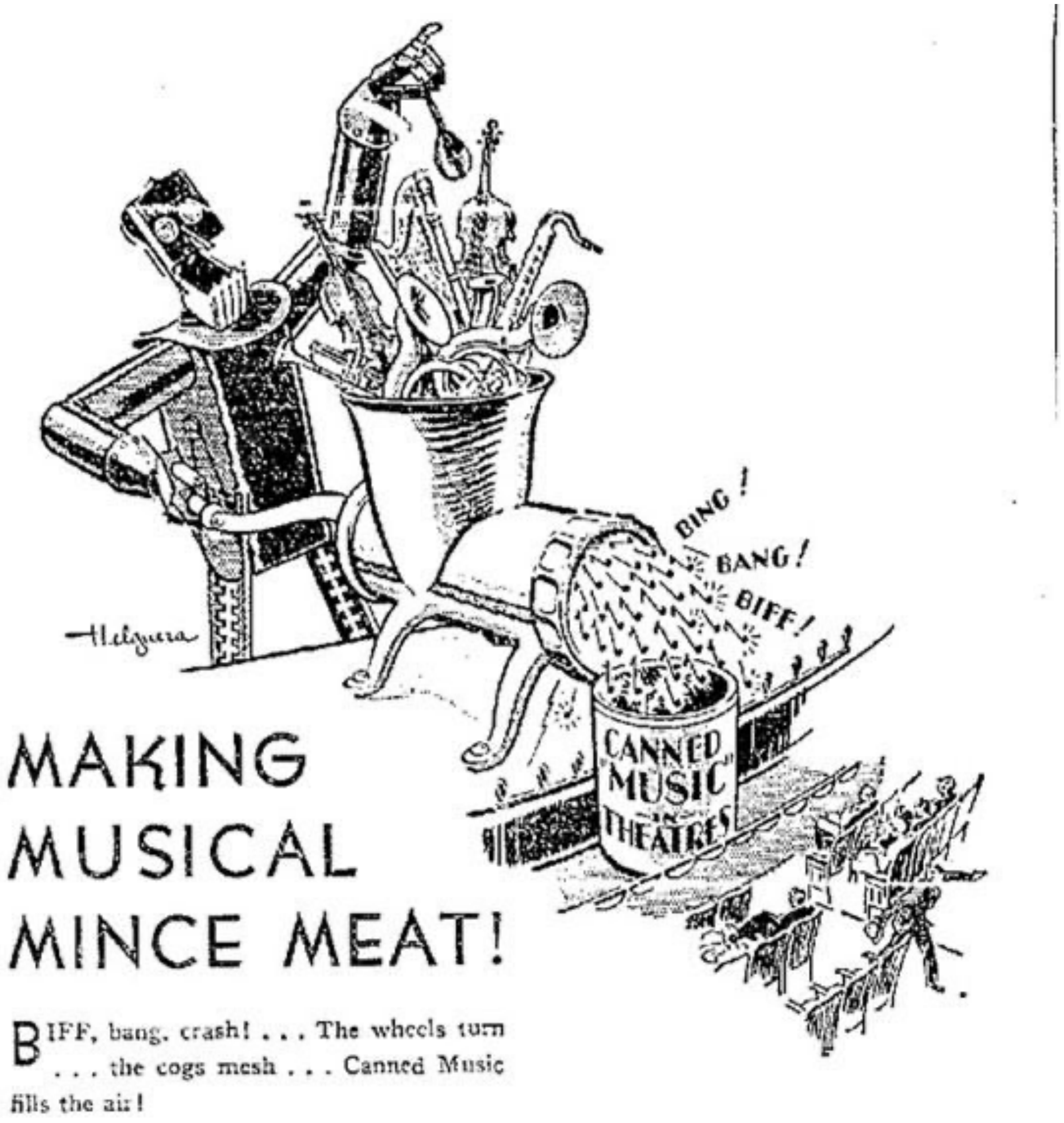
Americans were fascinated by the Telegraph in the 1840s, when the technology was becoming practical for uses like railway signaling and long-distance communication. In 1847, The New York Sun described the telegraph as "the greatest revolution of modern times and indeed of all time, for the amelioration of society," and hypothesized that the technology could bring peace and prosperity to nations (Phalen 15). At the time, it seemed like the telegraph might solve all kinds of social issues. In 1853, The Utica Gazette predicted an "immense diminuation" in crime, because criminals would give up trying to escape justice, writing "Fly, you tyrants, assassins and thieves, you haters of light, law, and liberty, for the telegraph is at your heels" (Phalen 16). In 1846, the Philadelphia *North American* predicted that because of the telegraph, "the absent will scarcely be away," and "The mother may, each day, renew her blessing upon her child a thousand leagues away; and the father, each hour, learn the health of those around his distant fireside" (Phalen 16). Journalists at the time summarized the anticipated capabilities of the telegraph with the phrase "Annihilation of space," which appeared in different sources (Phalen xxxviii).

From a modern perspective, techno-utopian predictions about the effects of the telegraph sound naive at best, but that did not stop writers like Kevin Kelly, John Perry Barlow, Esther Dyson, Stewart Brand, and others from promulgating strikingly similar narratives about the long-term impact of the internet in the 1980s and 1990s (Turner). A 1997 article in Wired Magazine titled *The Long Boom: A History of the Future, 1980–2020* predicted that technological innovations and shifting cultural attitudes would bring about a sustained period of economic growth and international prosperity, including "the beginnings of a global civilization, a new civilization of civilizations, that will blossom through the coming century" (Schwartz and Leyden). Some techno-utopian content still exists in projections about how blockchain technologies or AI technologies created by scraping the internet will be 'disruptive' or 'democratizing' in ways that yield social or economic benefits. However, today the internet is increasingly used to disseminate misinformation (Bradshaw and Howard). It is also increasingly effective as a tool for surveillance (Zuboff) and propaganda (Bay; DiResta et al.), and could be facilitating the rise in fascism and authoritarianism (Lewis). Social media usage is tied to social isolation (Turkle). The long-term prospects of the internet look less optimistic than they once did when *The Long Boom* was the Wired Magazine cover story.

Could we still be the beneficiaries of the internet and media technologies like Wired predicted? Or will we be the victims of it? With Fluid Music, I take a measured, but optimistic, stance and consider the detractors seriously. Fluid Music is both an internet technology *and* a music technology, so historical predictions about each are relevant. Historically, new music technologies have been met with skepticism and enthusiasm on different occasions and in different contexts.

2.3.2 Mechanical Music

Music technology detractors have criticized some technologies for being "mechanical." An 1813 essay discussing contemporary opinions about the chronometer (a predecessor to the metronome) presents a fictionalized dialog between a composer and a music director, wherein the composer soothes the music director's fear of being replaced by the machine, saying: "I was never of the opinion that a chronometer would be suitable for directing music. No lifeless or insensitive machine would ever be suitable for that” (Jackson). This argument against musical machines is often invoked when new technology threatens, or appears to threaten, the livelihood of musicians. For example, when recorded music began to replace live musical performance in the 1920s, an organization called the *Music Defense League* spent over $500,000 on advertisements like the one in Figure 2.1 that aimed to convince the public to demand live musicians accompany films in movie theaters (Novak).



BIFF, bang, crash! . . . The wheels turn . . . the cogs mesh . . . Canned Music fills the air!

Figure 2.1 This advertisement ran in the New York Syracuse Herald on November 3, 1930. It was part of a campaign organized by the Music Defense League, which invested $500,000 in similar advertisements, aimed to convince the public to demand live music in theaters by portraying recorded music as mechanical and inferior.

The Music Defense League was correct about the threat recorded music posed to musicians who performed in movie theaters. In the 1930s, films with synchronized pre-recorded audio, including speech, music, and sound effects, all but replaced silent films with live musical accompaniment. The appeal to the audience's sense of injustice and indignation on behalf of live music also belied another ploy that is subtle in Figure 2.1, but much more overt in other advertisements produced by the same organization. In images released by the Music Defense League, a dark, cartoonish robot consistently represents an existential threat to light-skinned audiences and working musicians (Novak), appealing to, and reinforcing racist notions and caricatures. When the argument arises, that music created with machines is inferior because of a perceived mechanical, lifeless, inhuman qualities, we should pay close attention to who is making the argument and why. This same argument about lifelessness was used to disparage disco music as it gained popularity in the 1970s. However, more recently, scholars and journalists exposed deep strains of racism and homophobia embedded in criticisms of disco music (Frank; Waleik; Danielsen).

2.3.3 The Liberation of Sound

Reactions against music technologies often focus on inhuman and machine-like qualities. I found a different mentality in optimistic responses to new musical machines. This mentality was best summarized by composer Edgard Varèse, who spoke enthusiastically about a hypothetical machine that could exactly reproduce the sounds in his mind, free of human "interpretation." In a lecture given at Princeton University in 1939, he said:

> *Personally, for my conceptions, I need an entirely new medium of expression: a sound-producing machine (not a sound-reproducing one). Today it is possible to build such a machine with only a certain amount of added research. If you are curious to know what such a machine could do that the orchestra with its man-powered instruments cannot do, I shall try briefly to tell you: whatever I write, whatever my message, it will reach the listener unadulterated by "interpretation." It will work something like this: after a composer has set down his score on paper by means of a new graphic notation, he will then, with the collaboration of a sound engineer, transfer the score directly to this electric machine. After that, anyone will be able to press a button to release the music exactly as the composer wrote it – exactly like opening a book. And the advantages I anticipate from such a machine: liberation from the arbitrary, paralyzing tempered system; the possibility of obtaining any number of cycles or, if still desired, subdivisions of the octave, and consequently the formation of any desired scale; unsuspected range in low and high registers; new harmonic splendors obtainable from the use of sub-harmonic combinations now impossible; the possibility of obtaining any*

> *differentiation of timbre, of sound-combinations; new dynamics far beyond the present human-powered orchestra; a sense of sound-projection in space by means of the emission of sound in any part or in many parts of the hall, as may be required by the score; cross rhythms unrelated to each other, treated simultaneously, or to use the old word, "contrapuntally," since the machine would be able to beat any number of desired notes, any subdivision of them, omission or fraction of them – all these in a given unit of measure or time that is humanly impossible to attain (Schwartz et al. 200).*

This was a lifelong interest of Varèse. In 1916, he was quoted in the New York *Morning Telegraph*, saying "Our musical alphabet must be enriched. We also need new instruments very badly" (Schwartz et al. 202), and in 1959, he also spoke of his "fight for the liberation of sound," taking care to emphasize that this aimed to complement acoustic instruments, not replace them (Schwartz et al. 201–02).

Varèse's words reveal a desire to use technology to exert control over sound. He was not the only artist with this inclination. The Italian futurists were a group of like-minded artists who, in the early twentieth century, advocated for artistic disruption that embraced aggression, industrial machines, and violence, all of which they associated with the future. Poet Filippo Tommaso Marinetti documented these beliefs in *Manifesto of Futurism,* published in a newspaper *Le Figaro* in 1909. Marinetti's manifesto celebrated nationalism, misogyny, and industrialism, and connected these ideas to themes of machines, dominance, and control. In a 1913 letter published as *The Art of Noise*,[5] Futurist composer Luigi Russolo extended these ideas to music and sound. The letter argued that conventional music had lost its ability to excite listeners (Russolo, *The Art of Noise*). Russolo believed that to move forward, new noisy, mechanical instruments and practices were required that embraced the industrial sounds of the future, as opposed to the orchestral sounds of the past. Russolo named these instruments Intonarumori, and managed to have some of them developed, shown in Figure 2.2.

In addition to his problematic association with authoritarianism and sexism, two different, but not unrelated, aspects of Russolo's writing stand out. First, Russolo correctly predicted that technology would enable exciting new instruments and sounds, and that these sounds would be created by machines. Today almost all music passes through machines before it is heard, although these machines are primarily electronic, and not mechanical as Russolo predicted. Second, it appears that there is a temptation to associate themes of control, freedom, and liberation to new media technologies. This is perhaps unsurprising, because, historically, bodies and nations used technological superiority to achieve dominance and subjugation. However, we

[5] Russolo wrote this letter in 1913; the similarly named book was published in 1916 (Russolo, *The Art of Noises*). A translation of the 1913 letter is available: Luigi Russolo, *The Art of Noise*, Robert Filliou, trans. (New York: Something Else Press, 1967).

should be apprehensive when proponents of a new technology claim that it will be a means to achieve liberation.[6]

Interestingly, in an 1878 article titled *The Phonograph and its Future*, Thomas Edison used the opposite metaphor from Varèse, claiming that the phonograph *captured* sounds instead of liberating them. He wrote that his newly invented phonograph enabled “The captivity of all manner of sound-waves heretofore designated as ‘fugitive,’ and their permanent retention” (4). Perhaps the reason for Edison’s choice of metaphor, was that at the time it was not clear that recording technologies could be used to manipulate sounds in addition to archiving them. In the same publication, Edison accurately predicted many applications of recording technology, including recorded books; talking dolls, clocks, and toys; recordings of family members and celebrities; advertisements; and, of course, recordings of musical performances. However, he did not anticipate that the legacy of recorded music would be found in the technology’s capability to synthesize and manipulate sounds in addition to capturing them.

[6] See Attali; Hesmondhalgh.

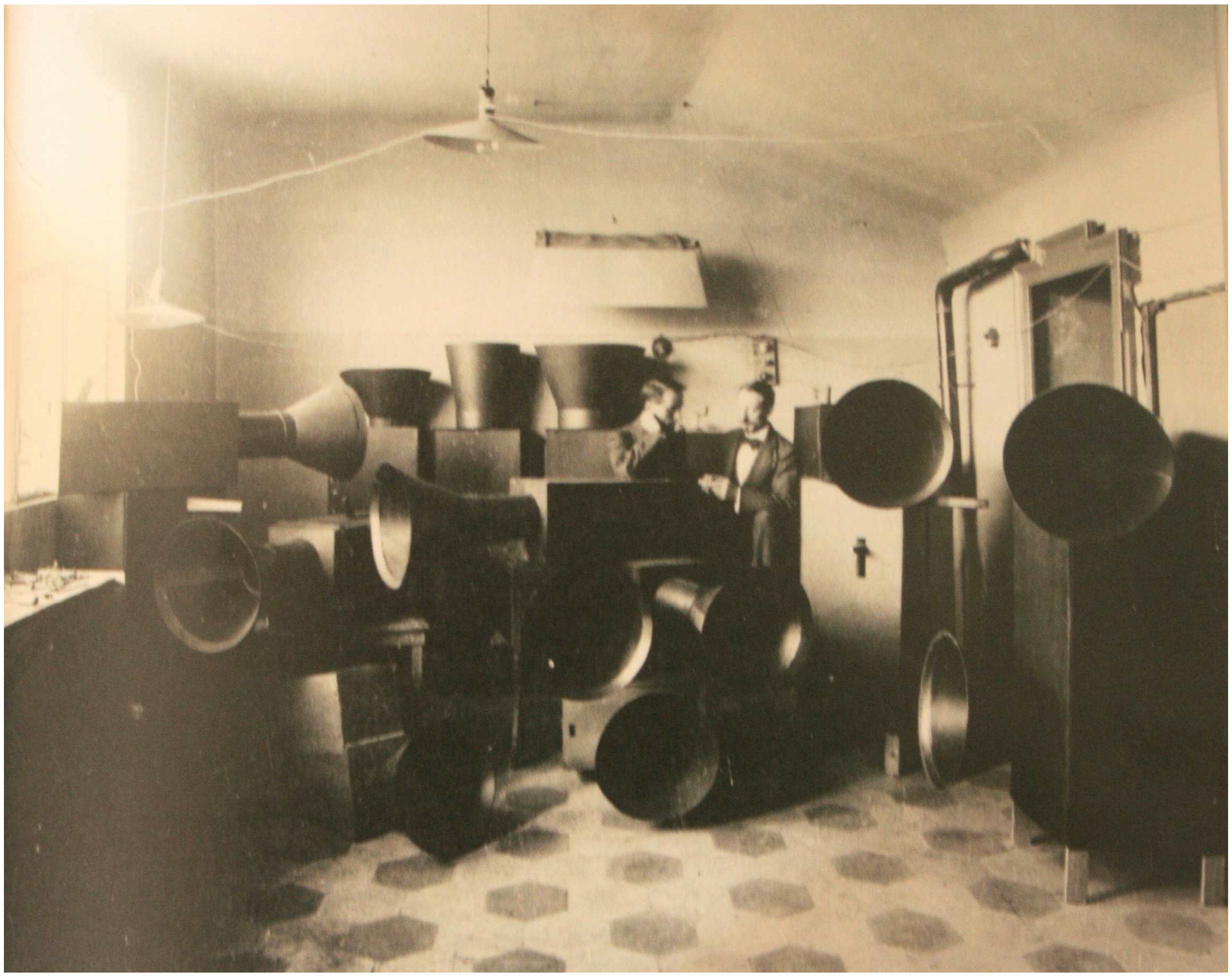


Figure 2.2 Luigi Russolo and his assistant Ugo Piatti in their Milan studio with their Intonarumori (noise machines), L'Arte dei rumori (The Art of Noises), 1913. Image Source: Wikimedia Commons.

The potential for sonic manipulation was predicted in an article by artist Laszlo Moholy-Nagy in a 1923 article titled *New Form In Music: Potentialities of the Phonograph*, originally published in the German magazine *Der Sturm* and translated to English in 1985 (Moholy-Nagy). Moholy-Nagy cited the instruments created by Italian futurists but observed that the mechanical nature of the Intonarumori limited their flexibility. He pointed out that the phonograph and amplification tube could be used to create and replay sounds by directly reading and writing to the microscopic grooves on a wax sound-recording plate. He wrote: "By establishing a groove-script alphabet, an overall instrument is created which supersedes all instruments used so far." Today waveform editing and waveform synthesis are common, albeit digitally, and not on physical records as Moholy-Nagy imagined. Still, it was the malleability of sound enabled by recording technologies that left the most significant imprint on music.

Before digital technologies were capable of editing waveforms, several engineers developed analog techniques for drawing waveforms. One such engineer was composer and electronic musician Daphne Oram, who was one of the founders of the BBC Radiophonic Workshop, a division of the British Broadcasting Corporation created to produce sound and music for radio. In 1957, Daphne conceived a tool that could draw both waveforms and envelopes, shown in Figure 2.3. She named the process Oramics*,* describing it in a book titled *An Individual Note: of Music, Sound and Electronics*, which broadly expressed her thinking about music and technology. Oram was very clear that, for her, Oramics was about the joy of musical experimentation. She explicitly noted an etymological relationship between the words “music” and “amuse” (Oram 1). She also hesitated to make predictions about the long-term implications of Oramics or music technology in general, writing “It is very difficult to assess which of the technical developments will be relevant in the future, for we usually have only a vague idea of the possibilities. Inventions can provoke many false and stupid prophesies” (110). To illustrate how predictions can go awry, Oram cited an encyclopedia article about the phonograph, which was written 25 years after the phonograph’s invention. The article predicted that the phonograph would amount to nothing more than “an interesting scientific toy.” When Oram published *An Individual Note* in 1972, enough time had elapsed for major commercial music empires built on recorded music to rise and fall multiple times. She observed “It seems incredible that, even 25 years after Edison's ideas were made known, the possibilities were still so unrealised” (111). When making predictions about technologies and their impact, we should be cognizant of Oram’s observations.

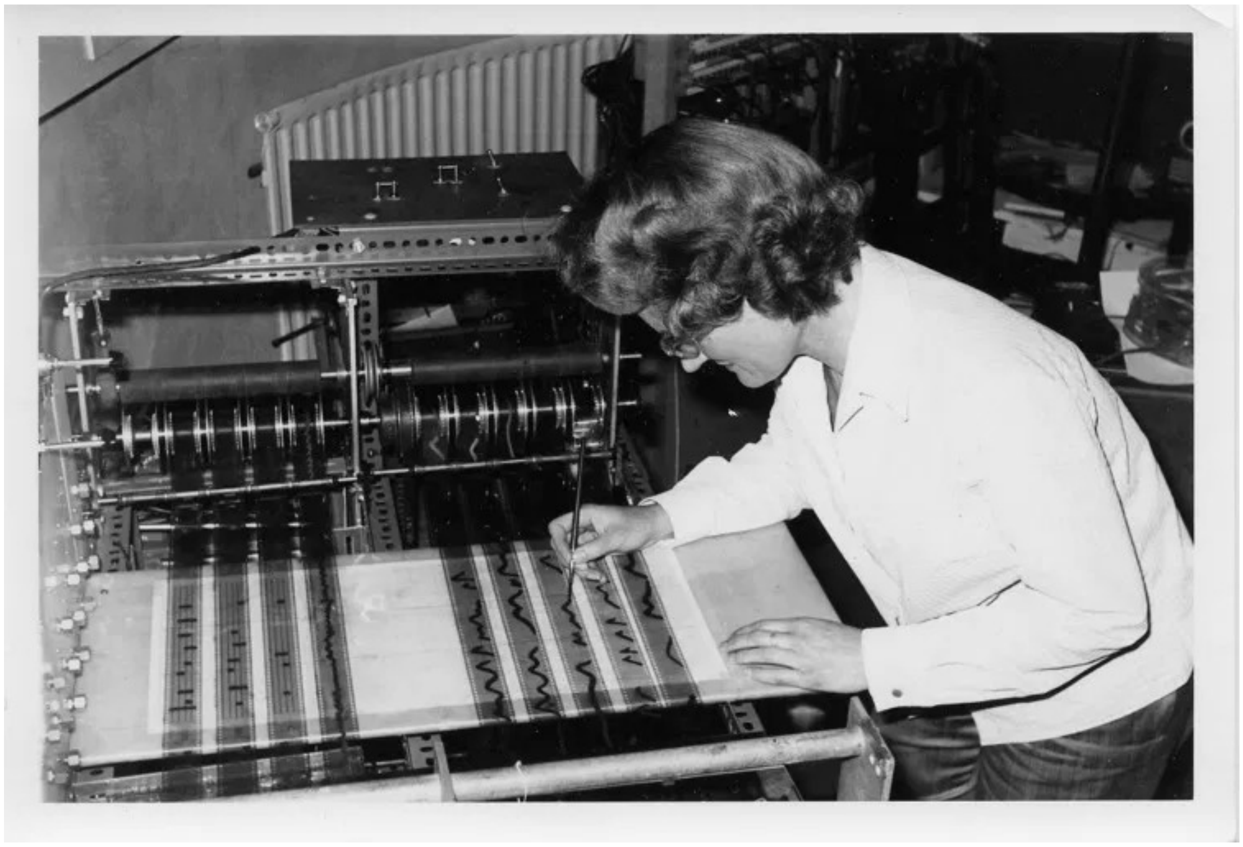


Figure 2.3 Daphne Oram working on the Oramics machine at Oramics Studios for Electronic Composition in Tower Folly, Fairseat, Wrotham, Kent. Source: https://120years.net/oramicsdaphne-oramuk1959-2/

Digital technologies like the Fairlight CMI, first released in 1978, would make the process of drawing waveforms and parameter envelopes more practical (Howell), and today, the automation lanes that are ubiquitous in DAWs make parameter envelopes widely accessible to DAW users. DAW automation lanes have the same purpose as the analog automation envelopes that Oram is shown drawing in Figure 2.3.

However, in the 1960s, before digital technologies became practical for manipulating audio, the dialectic pertaining to the roles of humans and machines in music reemerged in public discourse. This time, the debate centered around the merits of live music in comparison with the merits of recorded music. One side argued that there was an intangible quality associated with risk, mystery, or humanity that was erased by the music recording process. The other side argued that music created in the studio had many advantages over live music, resulting in a vastly superior musical experience. The debate was best summarized in a lengthy essay by pianist Glenn Gould, who had recently retired from performing live to focus on studio recording instead (Bazzana 229). Gould's essay, *The Prospects of Recording*, was a cover story in *High Fidelity* magazine in 1966 (46–63). In the essay, Gould outlined both arguments. His writing was also accompanied by perspectives from his contemporaries, including Aaron Copland,[7] Milton Babbitt,[8] Marshall McLuhan,[9] Goddard Lieberson,[10] and many others. These were presented as quotes, insights, and counterpoints adjacent to Gould's words.

At the time, advances in analog tape and other recording studio technologies were increasingly affecting popular music. Splicing, overdubbing, editing, and mixing were common in all kinds of recorded music. Recording and mixing techniques enabled engineers to cut, paste, and sculpt musical performances, creating new music with artificially enhanced clarity and precision. Were these new techniques enhancing music? Or destroying it? For Gould, the answer was clear: The sonic precision enabled by recording technologies gave recorded music one major advantage over live performance. The escape of music from the concert hall was another. In his words:

> *I herewith reaffirm my prediction that the habit of concert-going and concert-giving, both as a social institution and as chief symbol of musical mercantilism, will be as dormant in the twenty-first century as, with luck, will*

[7] Aaron Copland (1900-1990) was an American composer. He was quoted adjacent to Gould's to piece in *High Fidelity*, saying: "For me, the most important thing is the element of chance that is built into a live performance. The very great drawback of recorded sound is the fact that it is always the same. No matter how wonderful a recording is, I know that I couldn't live with it – even of my own music – with the same nuances forever" (47), and "If the listener does eventually come to the point where he makes the ultimate performance by splicing tapes from other musicians' recordings, he will eventually become just as bored with it as with other recordings, for it will still always be the same. Look, for instance, at electronic music. The boys are already becoming bored with what they do because they put it irrevocably on tape. The best indication of this is that more and more they are mixing the live performance element with their tapes" (60).

[8] Milton Babbitt (1916-2011) was a composer known for working with serialism and electronics. He was quoted adjacent to Gould's piece in *High Fidelity*, saying: "I can't believe that people really prefer to go to the concert hall under intellectually trying, socially trying, physically trying conditions, unable to repeat something they have missed, when they can sit home under the most comfortable and stimulating circumstances and hear it as they want to hear it. I can't imagine what would happen to literature today if one were obliged to congregate in an unpleasant hall and read novels projected on a screen" (47).

[9] Media theorist Marshall McLuhan (1911-1980) was quoted adjacent to Gould's piece, saying: "I think there is a parallel to tape recording in the technique of xerography. All the centuries of centralized mass production of books and printed material has suddenly become decentralized. The reader of any book can, with the aid of a Xerox machine, assume the role of author and publisher simply by snipping hither and thither from a multitude of sources. The tendency of this technology, and of electronic circuitry in general, is to tailor the response to the exact needs of the reader, the viewer, the listener" (59). McLuhan seemed to anticipate the world of personalized media content that emerged on the internet decades later.

[10] Lieberson (1911-1977) was the president of Columbia Records. Adjacent to Gould's piece in *High Fidelity*, he was quoted: "Personally, I don't like the present fashion of close-up miking, not even for the piano. I prefer perspective. I don't believe the engineer should intrude between the composer, or performer, and the listener and suddenly make you hear a flute or trumpet. I think the next step will be a regression back to the old days, with fewer microphones placed further away both to give perspective and to let the ears listen on their own. If a composer wants to write the other way, he should frankly call his piece a String Quartet for Four Instruments and Four Microphones; that is quite a different sound than for instruments alone" (49). Composers did write music for microphones just as Lieberson suggested. For example, Steve Reich (born 1936) composed *Pendulum Music* in 1968. Alvin Lucier (born 1931) recorded *I Am Sitting in a Room* in 1969.

> *Tristan da Cunha's Volcano; and that, because of its extinction, music will be able to provide a more cogent experience than is now possible* (1–2).

Gould expected recording technology to bring about the demise of the live concert and believed that by moving music out of concert halls and into our lived experiences in the wider world, music would be improved forever.[11]

In 2021, concerts have not become "extinct" as Gould predicted, although technologies for creating live shows increasingly apply characteristics and capabilities of the recording studio to live music (Bloomberg). It is hard to imagine live, acoustic music performances being effectively replaced by recorded music in the twenty-first century, but because the technology that mediates live performance is still evolving, it might be too soon for us to understand the extent to which live music performances and recorded music will continue to merge in the future. However, today we can better understand the *literal* prospects of recording, because the music that best illustrated the affordances of the medium did not yet exist when *High Fidelity* published Gould's essay. The magazine published *The Prospects of Recording* in their July 1966 issue. In August of the same year, at San Francisco's Candlestick Park The Beatles held their final public performance, after which the band permanently retired from touring to focus all their efforts on studio recording and production.

2.3.4 The Beatles, Strawberry Fields, and Sgt. Pepper's

The Beatles' producer George Martin described his mindset after the band's final public performance, writing "The time had come for [us to] experiment. The Beatles knew it, and I knew it. By November 1966, we had an enormous string of hits, and we had the confidence, even arrogance, to know that we could try anything we wanted" (Martin and Hornsby 199).

By February 1967, the band had recorded and released *Strawberry Fields Forever* and *Penny Lane* on the two sides of a 45-rpm vinyl record. The two tracks illustrate the kinds of sonic experiments that the band would push further when recording their subsequent album *Sgt. Pepper's Lonely Hearts Club Band*. *Penny Lane* featured a prominent and unusual piccolo trumpet orchestration and a piano recorded through a guitar amplifier with added reverb (Womack 165). The band recorded two versions of *Strawberry Fields*, and each version was in a different musical key. John Lennon preferred the first half of one recording and the second half of the other. George Martin adjusted the playback speed of the tape machine, increasing the speed of one, and decreasing the speed of the other, so that the two recordings could be spliced together without a jarring musical transposition (Julien 5).

[11] Glenn Gould was notoriously private and detested social functions. The two characteristics may have influenced his thinking.

On May 26, 1967, ninety years after Edison created the first phonograph, the Beatles released their eighth studio album, *Sgt. Pepper's Lonely Hearts Club Band*. The band spent nearly five months in the studio meticulously composing, performing, and overdubbing. The album blended avant-garde studio engineering practices with the Beatles' rock and roll instrumentation and producer George Martin's classical arrangement and orchestration. George Martin described *Sgt. Pepper's* as "songs which [the Beatles] had written which couldn't be performed live: they were designed to be studio productions" (Julien 6).

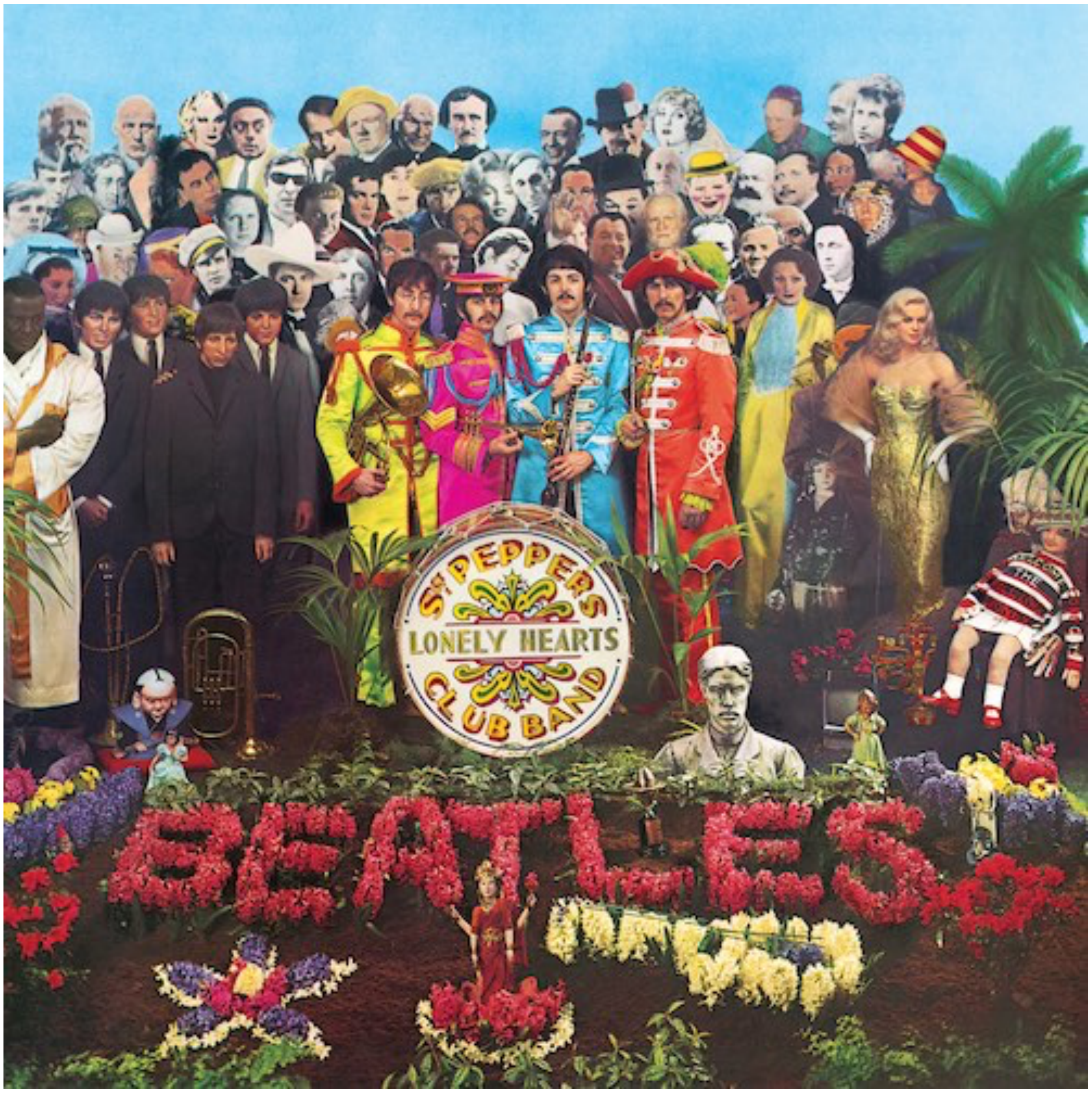


Figure 2.4 Sgt. Pepper's Lonely Hearts Club Band album cover. Karlheinz Stockhausen is in the back row, fifth from the left.

Working with producer George Martin and engineer Geoff Emerick, the Beatles spent months experimenting with performance and production techniques made possible by the tools in the studio. In *The watchamucallit in the Garden: Sgt. Pepper and fables of interference*, John Kimsey describes some of the production processes:

> *In numerous aspects, Sgt Pepper draws attention to itself as an artifact that has been shaped by human hands. The key gestures include the highlighting of the recording process per se through the emphasis on overdubbing; the deletion of rills between tracks; the manipulation of tape speed and direction; the use of editing to achieve startling juxtapositions; the layering of sounds to create exotic textures; the conspicuous deployment of signal processors like compressors, limiters and reverb units; the furthering of studio techniques such as close-miking, phasing and ADT;[12] the imposition of the 'show' frame[13] (along with the periodic breaking of that frame); and the sustained orchestration of these elements across 39 minutes* (133).

Please Please Me, the Beatles first album, was recorded in about 20 hours. Recording S*gt. Pepper's* took about 700 hours (Julien 5). Why was there such a large discrepancy? At the time *Please Please Me* was recorded, the convention was to have the band enter the studio having already written and practiced the music. They would spend a few hours performing the songs, and then leave the studio, while the engineers would mix the recordings. John Lennon recalled this earlier recording process, saying: "You just went in, sang your stuff, and went to the pub" (Lewisohn 10). With *Sgt. Peppers*, all the songs were written while the band was in the studio. The band's success allowed them to spend essentially unlimited time and money experimenting with many different timbral effects, performances, and arrangements for each layer of sonic material before committing to a particular sound.

The *Sgt. Pepper's* album is a useful example because of the turning point it illustrates. The release of the album merged the kinds of production processes used by Stockhausen and Pierre Schaeffer into popular music and popular consciousness. The Beatles acknowledged the influence of earlier tape music by including the likeness of Karlheinz Stockhausen on the *Sgt. Pepper's* album cover, shown in Figure 2.4.[14]

[12] ADT stands for Automatic Double Tracking, which is a recording technique developed for the Beatles by Engineer Ken Townsend (*Automatic Double Tracking - Wikipedia*), and used extensively on *Rubber Soul* and later albums including *Sgt. Pepper's*. In the process of ADT, the engineer layers a delayed copy of a signal over the unaffected signal. The result sounds similar to the double tracked recordings. Double tracking is the process of using overdubbing to layer two copies of an acoustic performance.

[13] Kimsey is referring to the "show" that is being performed by The Beatles' fictional alter egos – the members of Sgt. Pepper's fictional band. This fictional band is pictured on the album cover and is referenced in the album's song lyrics. See *"Nothing is Real": The Beatles as Virtual Performers* (Auslander and Ingils) for details.

[14] According to Kimsey (136), Paul McCartney liked Stockhausen's music.

Today, most music production depends on digital tools, and the analog equipment used by George Martin, Geoff Emerick, and The Beatles, has become costly and difficult to maintain by comparison. However, the newer digital production technologies can be used to simulate or re-create all the same production processes and effects used in *Sgt. Pepper's*. These production techniques are prevalent in many kinds of music today.

2.4 The Transition to Digital

The Beatles' time in the studio dovetails with the dawn of the internet. The band released *Abbey Road* on September 26, 1969. Five days later, a new digital device called the Interface Message Processor (IMP) was installed at the Stanford Research Institute (Hafner and Lyon 151). At the time, there was one other IMP, which was at the University of California, Los Angeles. These two IMPs were the beginning of ARPANET, the first long-distance computer network. The U.S. Advanced Research Projects Agency[15] had contracted the Cambridge, Massachusetts company Bolt Beranek and Newman (BBN) to develop ARPANET, a nationwide computer network that solved what was a very difficult problem at the time: reliable, long-distance digital communication. Over the next two decades, BBN continued to add IMPs to the network, and by 1979 ARPANET had 61 nodes (Hafner and Lyon 241).

As the number of IMP nodes connected to ARPANET grew, so did the complexity of maintaining the network and keeping all the IMPs up-to-date and running smoothly. In March of 1970, BBN added the fifth IMP node to their own research offices in Cambridge, MA. This enabled BBN to remotely monitor the active IMP nodes which were being installed at universities across the country. To monitor the network, BBN developed the first tools for remote maintenance and diagnostics. Their engineers introduced a feature that enabled IMPs in the network to update their internal operating software by downloading updated code from neighboring IMP nodes in the network (Hafner and Lyon 168). This feature, and other innovations developed at BBN, enabled the BBN team to propagate software updates to all IMPs in the network and to remotely restart or reload a specific IMP that was malfunctioning.

Today, sharing code and firmware updates over the internet is common – it is now unusual to distribute software any other way. However, before ARPANET, the best way to share software or code outside of a local area network was to carry or mail a stack of physical punch cards, so the ARPANET project was a major step toward the effortless digital content distribution that we expect from the internet today. Like Edison's phonograph, the project is a useful historical reference because it illustrates the start of a new era in communication technology with profound cultural implications for music and media.

[15] The Advanced Projects Research Agency (ARPA) was renamed to Defense Advanced Research Projects Agency (DARPA) in 1972.

The story about how IMPs could update their own code is particularly relevant to Fluid Music because the ability to effortlessly distribute code is probably the first new and useful capability that ARPANET introduced. This makes code distribution one of the oldest practical affordances of global-scale computer networking. In the years following ARPANET, other large-scale networks emerged that copied and improved the packet switching technology that ARPANET pioneered. Gradually these networks evolved and merged with each other, resulting in the internet as we know it today. Throughout the evolution of the internet, software engineers continued to create new and increasingly sophisticated tools for writing and distributing code collaboratively. Fluid Music takes inspiration from these sophisticated collaborative models that make open-source software engineers the beneficiaries of massive networks of global asynchronous collaboration.

2.4.1 Modern Technology for Massive Collaboration: Code and Music

While the internet makes it easy to distribute code, audio files, and digital media of all kinds, systems for sharing and collaborating with music and other media are less capable than their equivalents for collaborating with code. The combination of three collaborative paradigms, give code developers unprecedented flexibility in how they develop, share, and reuse code collaboratively. The three paradigms are version control, dependency management tools, and open-source collaboration platforms. Version control tools like `git` manage changes and updates to a code base, allowing users to track the history of these changes. `Git` enables many developers to work on the same code base simultaneously and eases the process of merging changes that overlap or conflict with each other (Hamano). Dependency management tools like `pip` and `npm` allow users to easily write code that incorporates external libraries. These tools work by creating a dependency graph, which is a map of all dependencies of a particular program or library, including any dependencies of dependencies.[16] The dependency manager can download and link all dependent libraries for a given software project. Open-source collaboration platforms like those hosted at https://github.com/ and https://www.npmjs.com/ provide centralized platforms that enable software library discoverability and facilitate collaborative workflows like forking, branching, and merging.

While there can be some overlap between the features offered by these tools, standardized interfaces that exist between these three collaborative paradigms allow interoperability between different tools and software engineers. For example, a software library that uses `git` for version control may be hosted on GitHub.com, which allows other developers to access the

[16] Dependencies of dependencies are known as "transitive dependencies." Robust dependency managers build dependency graphs recursively, so they identify all dependent software in a process called "dependency resolution" or "solving" the dependency graph. Part of the complexity in package management arises because package managers must track different versions of dependencies and identify or solve version conflicts when a dependency graph includes multiple incompatible versions of the same library.

code, suggest changes or updates to the original authors, or create a derivative version (known as a “fork”). If GitHub fails, the original developers can still host the code (including its development history) on a different collaborative platform that supports the `git` format, e.g., https://bitbucket.org/. Similarly, while the `npm` package manager is coupled with the platform hosted at https://npmjs.com, it is also possible to use `npm` with the open-source alternative available via https://verdaccio.org/.

In contrast, internet-based tools for sharing and collaborating with music are inflexible. In the early 2000s, peer-to-peer networks like Napster and Gnutella made sharing audio files widely accessible. These networks also re-introduced the rhetoric about liberating music, as illustrated by Stephen Witt's 2015 book about Napster, titled *How Music Got Free: A Story of Obsession and Invention*. However, sharing audio files is just one small part of collaborative music creation, and the peer-to-peer services were designed to be useful for music and media consumers, not music creators.

While more recent platforms like Blend (hosted at https://blend.io/) and Splice (hosted at https://splice.com) were designed for music creators, they are still inflexible compared to tools for sharing code. In part, this is because their roles, interfaces, and capabilities are less clearly defined than tools for collaborating with code. For example, the Splice platform offers a combination of services, including DAW session version control, an asset store, and a collaboration platform. However, Splice’s version control offers limited integration for a subset of DAWs. Their asset store has some resemblance to a package manager, but only supports static assets and does not support dependency management. Finally, Splice’s tools for collaboration are essentially automated cloud backups and do not support branching and merging. As a result, collaborative practices enabled by the Splice platform are limited to the synchronous exchange of static assets and DAW sessions.

Social media platforms like ccMixter (http://ccmixter.org/) and HITRECORD (https://hitrecord.org) are designed to facilitate collaboration by helping users to find collaborators and share static assets (Jarvenpaa and Lang). These platforms provide some of the functionality that open-source collaboration platforms provide to code developers, however they are focused on social networking. They might help users find collaborators or royalty-free assets to remix with, but their users must still use conventional tools during the composition or creation process.

Newer “online” DAWs like SoundTrap (https://www.soundtrap.com), Faders (https://www.faders.io/), Amped Studio (https://ampedstudio.com/), and BandLab (https://www.bandlab.com/) allow multiple users to play and edit a session simultaneously. The collaborative experience offered by these tools resembles Google Docs insofar as it allows multiple users simultaneous access DAW sessions in the same way that Google Docs allows multiple users to simultaneously edit a text document. However, these online DAWs limit collaboration to a single DAW session. They do not provide a way to flexibly reuse compositional or sound design ideas and, as a result, can only be collaborative at a small scale. They do not provide a pathway to leverage internet-scale collaboration. They also lock users into a proprietary platform, so, unlike tools for editing text and code, they are not interoperable with other tools and platforms.

In summary, tools for online collaborative audio and music production are primitive in comparison to tools for collaborating with code and text. Internet-scale collaboration is possible when sharing code and software libraries, but it is not currently possible in the domain of music production. In part, this is because collaborative audio tools are not widely compatible with each other. Additionally, tools for collaborative audio production provide different options for sharing static assets such as audio and MIDI files at a massive scale, but no viable path for sharing processes or Techniques that leverage the scale of the internet. In music production, what you do *with* assets is as important as the assets themselves, so existing tools are not poised to unlock internet-scale music collaboration.

2.4.2 Digital Audio

Just as radio and studio recording shaped and co-evolved with culture and music throughout the twentieth century, digital media and digital networks affect and shape music today. However, scholarly conclusions on the meaning, importance, and value of digital media technologies for music are still wide-ranging and inconsistent. In some cases, these conclusions resemble the discourse about recorded versus live music described by Glenn Gould in *The Prospects of Recording* in 1966, with proponents saying that the new technology fails to capture the essence of music in the way that the previous format did.

In his 2018 book *Perfecting Sound Forever*, Greg Milner claims that music lost something when we transitioned to digital recording production and distribution. Milner argues that meticulously editing and polishing a musical performance in a DAW removes the imperfections that gave the performance its emotional impact, and by doing this, they also enable artists to be content in turning lackluster performances into records. Milner writes that the ability to create perfect lossless copies of digital audio was a net loss for music, observing that every time we copy an analog recording, the quality of the copy is slightly degraded from the original, while digital copies are exact duplicates of each other.

In contrast, Lawrence Lessig argues that the ability to create exact copies of digital content is one of the key affordances that grants digital content its potential to be truly transformative for music and for media more generally. In his 2008 book *Remix: Making Art and Commerce Thrive in the Hybrid Economy*, Lessig argues that digital networking and digital software could enable everyone to become a content creator in addition to being a content consumer, correcting power imbalances by eliminating an anachronistic distinction between who gets to consume media content and who gets to create it. At the time, it appeared that file sharing would be the predominant way that music would be consumed on the internet, and Lessig thought that an updated interpretation of copyright law was key to unlocking the transformative potential of the internet for music and media. Lessig sought a compromise that preserved copyright as an incentive for content creators, while also legalizing the reappropriation of copyrighted source material in the production of content that significantly transformed the original material. Coincidentally, Spotify launched its music streaming service to the public in October of 2008, the same month and year that Lessig published his book on remixing. Unlike file sharing, streaming services do not currently help music consumers create new content by remixing existing content. In the years following Lessig's publication, the popularity of music streaming services surpassed the popularity of music file sharing services, and today Lessig's predictions appear dated.

In her 2017 book *This is Not a Remix: Piracy, Authenticity and Popular Music,* Margie Borschke identifies further weaknesses in Lessig's argument. According to Borschke, Lessig succumbs to the technological determinist pitfalls identified by David Schulz in *The Rhetorical Contours of Technological Determinism*; and the techno-utopian pitfalls identified by Ted Friedman in *Electric dreams: Computer culture and the utopian sphere*. Specifically, Lessig proposes a version of copyright law that allows both content copyright protection and content reuse in an attempt to get the best of free markets *and* free speech without fully acknowledging the tension between them (Borschke 53–56). In his book on remixing, Lessig implies that it was the digital revolution that enabled remixing. Borschke also shows how Lessig did not consider the origins of remixing in analog music technologies or the history of remixing, which began long before online digital tools were practical for repurposing music and media content (53).

One researcher who did recognize the long history of digital audio is Jonathan Sterne, who, in a book titled *MP3: The Meaning of a Format*, traces the origin of the `mp3` file format (and perceptual encoding more broadly) to research in psychoacoustics and information theory beginning in the early twentieth century. Sterne shows how this research was motivated and funded by the commercial interests of the telephone industry (19, 33). His historical analysis helps to set music and the internet in the context of file sharing services, but it does not attempt to look forward. It was published in 2012 and did not foresee the rise of streaming platforms as a viable alternative to distribution of static `mp3` files – although in the final chapter, he does recognize that "it is still unlikely that MP3s will remain the most common audio format in the world forever" (212).

While it was written before Sterne's book about the `mp3` format, Paul Théberge's journal article titled *The Network Studio: Historical and Technological Paths to a New Ideal in Music Making* includes some notable and timely insights about the future of digital audio. Théberge observed how academic literature about globalization prescribed the rhetoric surrounding internet technologies and music recording studios. At the time, scholars of economics, media studies, and anthropology were writing about globalization and were attempting to understand and describe how network communications were affecting their respective fields. Théberge identified themes in academic literature about how network technologies were affecting physical spaces, virtual spaces, and the digital connections between them. He discussed how the "spaces" where music was being created were also increasingly a combination of virtual and physical spaces mediated by digital communication technologies. He claimed that the trajectory of music production technologies was moving toward virtual spaces, writing that while the process of recording sound in an acoustic space would continue to be an important part of music production, "[because of] the increasing emphasis on the use of studio tools as a means of music composition, the status of sound as an acoustic event, to which, in many ways, the architecture of the larger studios still lent itself, has become largely irrelevant."

While Théberge primarily used the word "space" to describe physical spaces, he also left room for a more open-ended interpretation. We can think of a "space" for music production as an artificial acoustic space created by a digital reverb effect, the acoustic signature imprinted in a digitally sampled instrument, or even a virtual online space for music collaboration or social networking in addition to the architectural spaces in music recording studios. From this perspective, the timing of the article anticipated a turning point in popular music that corroborated his theory. This turning point was identified in the 2015 book *The Song Machine: Inside the Hit Factory* by journalist John Seabrook. In November of 2004, a month after Théberge's article was published in *Social Studies of Science*, artist Kelly Clarkson released *Since U Been Gone*. The song sounded similar to the pop-infused rock that was popular at the time, but unlike similar sounding music on the radio, *Since U Been Gone* was created by producers Max Martin and Lukasz Gottwald in a digital studio without an actual band, albeit with Gottwald's "intentionally amateurish-sounding" electric guitar (Seabrook 136). According to Seabrook, the song's release embodied a new era of pop music, wherein popular music charts were dominated by songs that producers had written using digital tools of the studio, instead of by bands or by the artists who perform them.

In *The Network Studio: Historical and Technological Paths to a New Ideal in Music Making*, Théberge correctly identified the changing role of digital production as popular music shifted from being *mostly performed* with a layer of production to being *mostly produced* with a layer of performance.[17] However, the implications for the potential of connected media production were more difficult to anticipate. The paper cited a failed dotcom era startup, Rocket Network,

[17] From this perspective *Sgt Pepper's* is an album that was mostly performed, but had a thick layer of production on top, insofar as the Beatles and various session musicians performed and recorded the underlying source material. In contrast, *Since U Been Gone* was mostly produced, albeit with Gottwald's guitar performance and Clarkson's vocals. Still, both The Beatles and Kelly Clarkson blur the lines between performance and production.

that aimed to connect physical recording studios together so that artists and engineers in different geographic locations could collaborate in "near real-time." The Rocket Network technology did not turn out to be useful or effective enough to sustain a business. In 2004, the underlying technology was still changing too fast to understand the potential of digital collaboration for audio and music production. Today, we have more information. Currently, it is possible to connect geographically separated recording studios, synchronize DAW sessions, and remotely control professional digital recording consoles from afar. All of these can be accomplished as fast as the underlying networking infrastructure allows, effectively delivering on the promise that Rocket Network made in the early 2000s.

Could this kind of real-time collaborative interactivity be an important part of what digital networks can offer for music and for musicians? Possibly, but while real-time synchronous collaboration is now feasible, I have not been able to find an example where it provided a clear artistic value. As a result, for the making of Fluid Music I assume that the best real-time music collaboration happens in physical acoustic spaces and that the larger the geographical area, and the greater the number of collaborators, the slower we can usefully collaborate. To discover if and how new collaborative workflows can be musically valuable, we need a new kind of digital music technology that enables us to build and experiment with new collaborative workflows. This is the aim of the Fluid Music framework. The following chapters explain conceptually and technically how this framework works and how I propose we use it to explore new collaborative possibilities that embrace the technology and lessons of the past while looking optimistically but cautiously to the future.

3 Fluid Music

In his 1964 book *Understanding Media: The Extensions of Man*, Marshall McLuhan wrote:

> *The 'content' of any medium is always another medium. The content of writing is speech, just as the written word is the content of print, and print is the content of the telegraph. (8)*

Like much of McLuhan's writing, the meaning of this statement is delphic, yet it also carries a useful insight. Early examples of a new media format can encapsulate an existing medium. Initially, recorded music was essentially an encapsulation of a musical performance, before it grew into something new and altogether different. McLuhan's observation is relevant because recorded music is the content of Fluid Music. More specifically, Digital Audio Workstation (DAW) sessions are the content of Fluid Music. In this chapter, I describe DAWs, Fluid Music, and discuss how they relate to each other. First, I would like to distinguish the concept of Fluid Music from the Fluid Music software framework.

3.1 The Concept

Conceptually, Fluid Music involves massive asynchronous collaboration as it relates to the sound design and the core components of a DAW. In a conventional music production workflow, a producer, composer, or sound designer might use a known production technique, but they would reimplement and customize that technique each time it is needed. For Fluid Music I posit that composers, sound designers, and music producers should be able to share and reuse each other's techniques to create new content. If each technique has a unique name and is made available through a repository of techniques, then users can asynchronously collaborate to develop a comprehensive notation system for contemporary music production.

Fluid Music is not made by sharing and reusing DAW templates and plugin presets. Presets and templates are static configurations, while Fluid Music techniques are computational processes that can be programmed to have different effects in different situations. Additionally, a template or preset is only applicable to the specific DAW or plugin it was created with, whereas Fluid Music techniques apply to DAWs more generally.

Fluid Music is also different from remixes. In the context of music, remixing is when a creator uses an existing composition as source material for a new or derivative composition. Fluid Music is about sharing and reusing the processes used to create content, not just reusing the content itself. This also distinguishes Fluid Music from audio sample repositories for sharing audio content like ccMixter.org, splice.com, and freesound.org, among others.

Fluid Music is also different from crowdsourcing. With crowdsourced content, participants are working toward a particular project or goal, resulting in a hub-and-spoke shaped dependency graph. Fluid Music techniques are more general than crowd-sourced contributions, and when a user shares a technique in the context of Fluid Music, that technique may be used by any number of different projects or parent techniques.

Modern DAWs can host plugins that process and synthesize sounds, but sound design techniques incorporate much more than just processing and synthesis. To really share sound design techniques, those techniques must be able to manipulate the DAW content (such as audio and MIDI clips) that feeds into plugins, in addition to the configuration of those plugins. As a result, the ability to share and reuse audio plugins is also different from Fluid Music. To create Fluid Music, we need to be able to share modules that encapsulate the DAW itself, as opposed to sharing plugins that are encapsulated *by* a DAW. In other words, and as posited above, "recorded music is the content of Fluid Music."

3.2 The Framework

The Fluid Music framework is a software toolkit that enables developers to describe music production processes with code. It was designed to fill a specific gap in the landscape of computational sound design tools, languages, and libraries, and provides a robust foundation to build and study Fluid Music workflows. This chapter (chapter 3) describes the motivation and design from the framework. On its own, the framework is capable enough to implement and study Fluid Music workflows among users who are able to code. Chapter 0 describes my first experiments in this space and shares the results of a user test. In the long-term, Fluid Music will not reach a wide audience without a more accessible interface. Options and precedents for the development of a graphical user interface are discussed in chapter 7.

*Technique*s are a foundational concept in the Fluid Music framework. In the context of the framework, a Technique provides a mechanism to perform one or more sound design procedures akin to the actions a sound designer might perform in a DAW. For example, a simple Technique could insert an audio file into a DAW session at a particular time or gradually adjust the gain of an audio track at a particular time. In the Fluid Music framework, Techniques are written in code and can perform any action that can be described with code.

Complex aggregate Techniques can invoke many sub-Techniques. Advanced Techniques can incorporate machine listening or audio feature extraction to select, contextualize and parameterize sub-techniques. The Fluid Music framework comes bundled with many Techniques for common actions. Most importantly, it defines the Technique API. This makes the Fluid Music language extensible by enabling users to create their own Techniques.

Because the Fluid Music framework is built around the Node.js software ecosystem, Fluid Music piggybacks on the Node Package Manager (NPM) which makes it trivial to publish, share, and reuse Techniques that extend the vocabulary of the Fluid Music language. It also neatly handles Technique versioning and dependency management. This allows third-party Technique developers to incorporate each other's Techniques as dependencies and build new Techniques that extend or augment existing ones.

Fluid Music comes with a simple but powerful score language for sequencing Techniques on a musical timeline. This score language is built with JavaScript language primitives like objects and strings. The score language enables users to write Techniques and sequence them in musical scores in the same programming language. It also makes it straightforward to generate scores computationally.

The score language uses strings to describe musical rhythms. This rhythm notation system can describe both simple rhythms and complex musical timing with any combination of rhythmic subdivisions. Strings can also describe sequences of techniques arranged on a musical timeline. The language was designed to be used with machine learning models that operate on data sequences such as recurrent neural networks (RNNs) and transformers. Existing projects that use transformers and RNNs to model music only work with symbolic notation akin to MIDI. Such symbolic notation formats cannot effectively capture and describe studio production techniques that emerged in the twentieth century, so existing sequence-based models for describing music are not practical for modeling contemporary composition. At the same time, the score language is robust and expressive enough to enable composition within a text editor, assuming that the user is comfortable working with JavaScript and the Node.js software ecosystem. This format limits the potential users to people who can code. Developing a graphical user interface for working with Fluid Music is an important part of future work as described in chapter 7.

Unlike other tools for computational sound design, compositions created with Fluid Music can be exported and opened with a professional digital audio workstation. This feature enables the user to study and understand the output of content that was created procedurally, and it is valuable for debugging and verifying computational processes that create music. It also provides a viable pathway for incorporating computation into polished compositions that can sound competitive next to music created within a conventional DAW workflow. In music production details matter. To make audio content that sounds polished, we still need to be able to manually edit things like edits, fades, and equalization. Some of these sound design details can be performed or approximated computationally, but most can still be executed more effectively by a skilled audio engineer. We may want to offload some of the more monotonous sound design techniques to algorithms, however we cannot get the data that we need to automate these kinds of actions without a system like Fluid Music. Fluid Music provides the foundation required to move toward a future where we can fully take advantage of computation for sound design and music production without sacrificing the flexibility and control granted by a conventional DAW-based workflow.

A long-term goal for Fluid Music is to create an extensible notation system that enables music producers and sound designers to document, share, and reuse the kinds of music production techniques that, together, make up a creative vocabulary for the art of sound recording. Stepping toward this goal, the framework provides a practical foundation for building next-generation collaborative music technologies. In this way, it allows researchers and composers to explore questions like: What parts of the sound design process can we automate? What parts do we want to automate and what parts do we want to give to the composer or sound designer? What parts of the process do we want to share freely? What parts do we want to belong to composers? To study these questions, I ran a workshop where I taught 14 users how to use the Fluid Music framework. I then conducted semi-structured interviews with participants who completed the workshop and volunteered to be interviewed.

3.3 What is a Digital Audio Workstation?

To explain what the Fluid Music framework does (and does not) do, and what it may do in the future, I split DAW capabilities into two categories: the core and the extended features. This conceptualization of the core and the extended features of a DAW is useful for understanding how the design of the Fluid Music framework supports the project's long-term goals.

3.3.1 The Core

The core includes the six following functional components that distinguish a DAW from other audio software: tracks, clips, routing, automation, plugins, and sessions. For our discussion of Fluid Music, a DAW is simply any audio software that allows the user to manipulate sounds with these familiar interface paradigms. All DAWs have some concept of a "track," which is a skeuomorphic digital representation of a track on an analog tape recorder like the ones used by Pierre Schaeffer, Les Paul and Mary Ford, and The Beatles. Audio software like Max, SuperCollider, or Pure Data that does not include a comparable implementation of a "Track" would not be considered a DAW.

DAWs use different names to describe these core components. For example, in Pro Tools, an "audio clip" is the name for a non-destructive reference to a region within an audio file that is at a particular position on an audio track. Reaper uses the name "audio item" to describe its (functionally similar) component. Core components tend to be largely similar between different DAWs despite their different names.

All DAWs have subtle differences in the implementation of their core components. Digital Performer implements routing with "sends" and "busses" which behave like the components in an analog mixing console to which their names refer. Reaper's routing model doesn't distinguish between tracks and busses, but it allows arbitrary sends and receives between tracks.

The core DAW components are modeled after their analog ancestors that were used in the recording studios that emerged in the twentieth century. Conceptually, “plugins” are modeled after outboard signal processors used in recording studios and are another example of this pattern. Examples of plugins include dynamic compressors, equalizers, synthesizers, and reverb units. Each plugin has some combination of inputs, outputs, and automatable configuration parameters. Crucially, plugins do not need to be bundled with DAW software and may be developed by third parties. Like all the core DAW components, the behavior and capabilities of digital plugins extend beyond what was originally possible in purely analog studios while maintaining a clear lineage and clear similarities with their ancestral counterparts.

Automation was also first developed using analog studio technology (Gradstein 37–42). In popular music production, automation was one of the first practical applications of digital technologies in recording studios (Clark). As digital technologies became more pervasive, the power and flexibility of automation also grew proportionally.

3.3.2 The Extended Features

Commercial DAWs build complex and powerful features that distinguish different DAWs from each other and add workflow enhancements and layers of abstraction that extend beyond the core functionality described above. Extended features include audio effects and synthesizers that are integrated into the DAW and, unlike plugins, cannot be used with other DAWs. They also include built-in sample libraries or methods to automatically modulate plugin parameters with Low Frequency Oscillators (LFOs).

Most extended features are effectively layers of abstraction that manipulate the core elements of a DAW. For example, a built-in sampler is just a way to computationally trigger and manipulate audio samples and then sequence those samples on a timeline. An “auto panner” can modulate stereo panning with an LFO. However, the same effect can be achieved with panning automation.

Extended features also include enhancements like Ableton Live’s clip launcher, an interface for “playing” audio and midi clips like a live instrument. Like most extended features, the clip launcher mode depends on core DAW features like tracks, clips, and plugins, and allows the user to sequence, layer, and trigger audio clips in a live performance. Live performance features like this may fall into the Extended Features category, but they are also less relevant to Fluid Music. While the Fluid Music framework supports real-time audio playback, it is mainly designed for composition and notation, not improvisation.

3.3.3 DAW Sessions

All DAWs define a project format, which is a sound design document that uses structured data to describe the state of a particular project in terms of the core and extended features. For example, every time a user creates a track in a DAW, the software updates its internal state by adding a serializable digital identifier that the software recognizes as a track. When the user saves a project to disk, this state is serialized and stored as a file. Fluid Music uses the word *session* to name the generic structured data format that DAWs use to describe a project.

Each DAW has its own proprietary session format for modeling and serializing projects. DAW sessions all share some common characteristics which correspond to the core components described in section 3.3.1. However, session formats also support proprietary features that correspond to extended components described in section 3.3.2.

While session formats for different DAWs have some overlap in the way that they describe the underlying content, they also have considerable differences. Despite efforts to create generic DAW formats like AAF and OMF, there is currently no feasible way to describe a DAW document generally ("Turning the DAW Inside Out", Holbrow). In the words of the author of the AATranslator software, which aims to convert session documents between different DAW formats, DAW "capabilities vary so enormously that a complete conversion between every session format is impossible" (Rooney). Existing DAW session formats are good for describing the exact data model that is used by a particular DAW but are not an effective way of describing a sound design document generally.

The fragmentation in the extended features and capabilities of different DAWs presents a challenge for composers who want to de-couple their documentation from the specific technology that was used to create a particular composition. It also obstructs one of the main objectives of Fluid Music – the development of a useful notation system for the kinds of production processes that distinguish recorded music from live music. If each DAW has a slightly different feature set, and works in a slightly different way, how can a Fluid Music score be interpreted without access to the exact combination of audio software it was written for? The way that Fluid Music handles session objects is an illustrative case study of how to address this challenge.

The Fluid Music framework defines a class called `FluidSession`, which is described in detail in section 4.2.1. The `FluidSession` data model only supports core DAW components like tracks, clips, plugins, routing, and automation. It does not support any extended features. However, this does not mean that the kinds of extended techniques that define modern audio software are not available when using Fluid Music. Instead, the way to access extended DAW features in Fluid Music is to write and publish a Technique or extension. Because most extended features are effectively layers of abstraction that build on core DAW functionality, sound designers can use an existing Technique and insert it into a project, or write and publish their own extended Techniques if a suitable one does not already exist.

Because there are so many different tools used for music production, there are exponentially more possible ways to use those tools in combination with each other. As a result, it is impractical (at best) to create a notational language that can describe every musically useful application of studio technology. Fluid Music addresses this challenge by making a language that is extensible and allowing users to add new tools and techniques to the notational language vocabulary. While the scope of the language is necessarily much larger than existing notational systems, so is the collective effort of potential contributors. In this way, Fluid Music can use one of the great strengths of the internet – massive asynchronous collaboration – to meet a longstanding musical challenge.

3.4 Music Production Analysis

At its core, Fluid Music is a system for defining, sharing, and reusing sound design Techniques. Each Technique is a series of steps that operate on a portion of a sound design document (such as a track or time range in a DAW session file). Each Technique defines a sound design pattern, which contains a sequence of sound design steps that would conventionally be made inside a sound design document. Like software design patterns, Techniques can build on each other in layers of abstraction, allowing complex effects to be described concisely.

This section includes the results of an analysis of two compositions. It shows the kinds of Techniques that Fluid Music is intended to describe. The first example is an excerpt from Ariana Grande's song *No Tears Left to Cry.* The second is from Tod Machover's symphony *Between the Desert and the Deep Blue Sea: A Symphony for Perth.*

3.4.1 Analysis: No Tears Left to Cry

The following analysis is the result of reverse engineering the first 46 measures[18] of *No Tears Left to Cry*, omitting a description of the vocal production. This is a useful example, because the production is relatively simple, but also tasteful and professional.

Intro (mm. 1–10)

- Intro begins with vocals and a pad synth. The synth oscillator is set to slightly detuned copies of a rich waveform, likely a sawtooth, and a lightly resonant low-pass filter, set at a low frequency for a 'dark' timbre. The filter cutoff frequency is modulated to increase very slowly over a few seconds. The detuned sawtooth copies are panned for a wide panorama in the stereo field.
- m. 5 - Bass enters.
- m. 7 - The tempo abruptly increases from ~100bpm to ~122 bpm (with the lyrics "pickin' it up").

[18] This analysis extends through the first chorus, which is sufficient for this example. The subsequent sound design is quite similar to what is described above, although some subtle changes prevent the second verse and chorus from sounding identical to the first. For example, in the second verse the bass enters immediately instead of waiting four measures before entering. It also adds a woodblock after 6 measures which was not present in the first verse. The second chorus adds an additional pad synth that was not present in the first chorus.

- m. 7 - The reverb, which was initially exaggerated, is attenuated.
- m. 9 – A drum pattern is introduced, with snare, hat, and headless tambourine. The drums are band limited with a filter.

Verse 1 (mm. 11–30)

- The verse begins with vocals, full drum kit, and synth.
- The main synth uses a similar patch as was used in the intro, but slightly increases the cutoff frequency of the low-pass filter for a brighter sound.
- Drums become full bandwidth (band pass filter deactivated).
- m. 15 - Bass enters.
- m. 23 - The low-pass filter on the main synth slightly increases the cutoff frequency and resonance parameters.
- m. 23 - A low level pad enters playing a single note (A3), which sustains until the end of the verse. Aside from the reduced amplitude, the patch is similar or identical to the one used in the intro: it starts out very dark and the low-pass filter gradually opens, with the resonant peak of the filter sliding across the upper harmonics of the audio produced by the oscillators.
- m. 30 - A noise burst, possibly a rimshot, begins on the fourth beat of measure 30. It is fed through a long, bright, reverb with a large room size. The reverb decay is audible for exactly four beats, before being muted on beat four of the following measure. This effect fills the role of a crash cymbal at the end of a drum fill by anticipating the transition to a new musical section.

Chorus 1 (mm. 31–47)

- The chorus begins with vocals, drums, bass, and a new synth sound.
- The new “chorus synth” is another subtractive patch, but it has sparser harmonic content and is likely a triangle waveform with very light distortion. It is a monophonic patch with portamento, and the voicing loosely harmonizes with the vocal melody.
- mm. 37 - A vocal pad enters, following the current chords in the chorus (initially C major). This pad continues throughout the chorus, but “ducks” the other accompaniment and is not always audible.
- mm. 38 - A noise burst with reverb, like the one used in measure 30, begins on beat four of measure 38, coinciding with the transition to the second half of the 16-measure chorus.
- mm. 40 - On beat two of measure 40, the chorus synth shifts up a register for a brief melodic accent. It returns to its original place at the beginning of measure 41.
- mm. 41 - A xylophone part fades in over the next two measures. It is compressed with a fast attack and medium release, de-emphasizing note attacks and exaggerating the

sustained portions of the sound. The rhythmic structure is nearly identical to that of the bass in the verse.

Notes

Each point in the analysis above is a creative choice that represents some intention or objective by the sound designer. Together with the musical content (which was intentionally omitted from the analysis above), these choices define the sonic identity, the sound world of the song. For the purposes of the Fluid Music, each bullet point can be captured as a Technique and applied to any Fluid Music session. A high-level Technique can wrap a collection of sub-Techniques that define the song structure and arrangement and be used to generate a full song from a few musical fragments. In the case of the Ariana Grande song, like many pop songs, the musical content is very simple and secondary to the sound design and vocal production. For example, up until the chorus only a few musical fragments are used, as shown in Figure 3.1 and Figure 3.2.

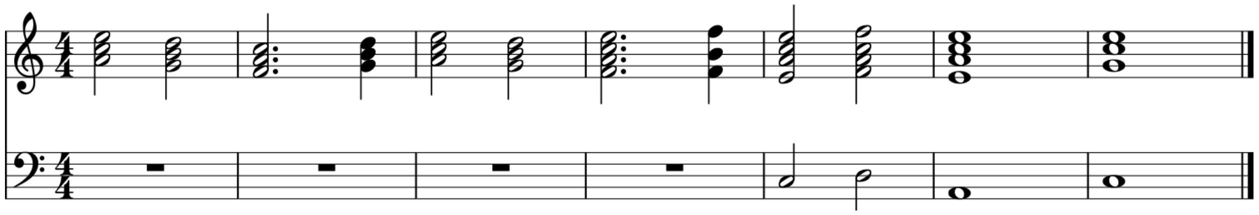


Figure 3.1 No Tears Left to Cry intro synth and bass transcription

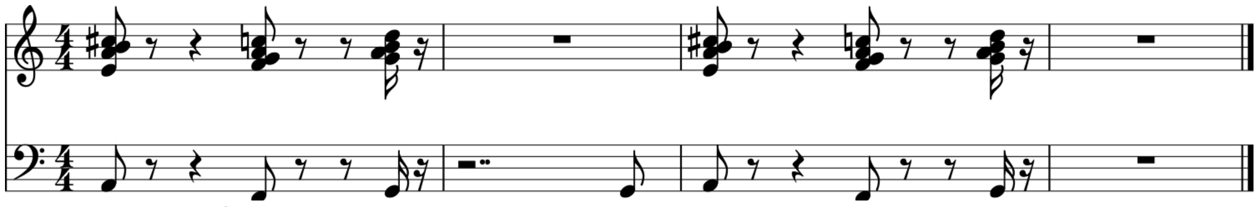


Figure 3.2 No Tears Left to Cry verse synth and bass transcription. This pattern repeats throughout the verses, with the bass part coming in and out as specified by the sound design analysis.

One advantage of packaging sound design Techniques is that they can be applied to any musical content. With the Fluid Music framework, a designer can compose different musical fragments, select different synth timbre Techniques, and reuse the musical structure and arrangement Techniques to generate an altogether different song. Alternatively, a sound designer can select combinations of other arrangement Techniques and build more elaborate musical structures while reusing the sound world defined by the existing Technique collection. Fluid Music enables a sound designer to easily create many mutations of an existing sound world and to share, remix, and reuse the sound worlds created by other sound designers.

3.4.2 Report: Between the Desert and the Deep Blue Sea: A Symphony for Perth

The analysis of Ariana Grande's song shows how sound design choices can be encapsulated as Techniques, and these Techniques can dictate both a musical arrangement, such as when a new instrument enters the mix, and adjust effect parameters, such as the modulation of a low-pass filter cutoff frequency or resonance control. In the Ariana Grande example, these recipes were easy to specify because of the formulaic structure of pop music. Will the technique work with other musical genres that are less formulaic?

To understand this better, I will review a sound design session I worked on for Tod Machover's *Between the Desert and the Deep Blue Sea: A Symphony for Perth*.[19] In this example, I was processing the electronic layers that Tod Machover composed, adding movement, blending, and working to make things 'sparkle.' I did not change the arrangement; so, my observations here apply to audio effect manipulation and not to the arrangement. Further experiments are needed to understand how choices of musical structure and arrangement could be made into recipes for genres that have more complex structures than pop music.

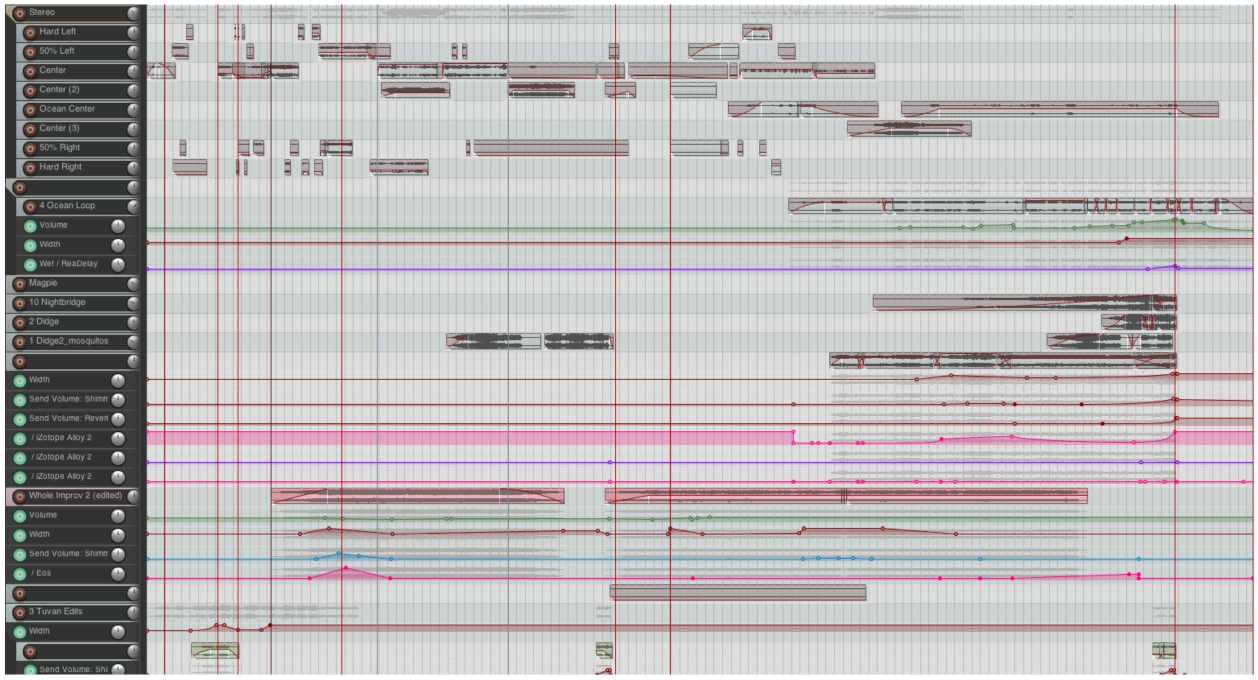


Figure 3.3 Reaper (https://reaper.fm) was used for editing the electronics in Movement III of Between the Desert and the Deep Blue Sea: A Symphony for Perth. This screenshot shows a subset of the 57 active tracks and automation lane parameters used in the session.

The DAW session used for the sound design has 57 active tracks and parameter automation lanes. Of the many edits that were made in the session (see Figure 3.3), some were practical changes, like automating the volume to blend multiple audio sources and prevent one of them from sounding too much louder than the others. The process of blending multiple sounds can be automated and packaged as a Technique. Developing a Technique for this purpose involves "automated mixing," an interesting research area. While the Fluid Music framework provides

[19] See http://citysymphonies.media.mit.edu/perth.html. I looked at my DAW session for the electronics in Movement III: Within, which begins at 13:26 in the recording available at the link above.

useful tools for this research area, it outside the scope of my dissertation. There are many edits in the session visible in Figure 3.3 that can be packaged as Techniques. The edits shown in Figure 3.4 are a good example.

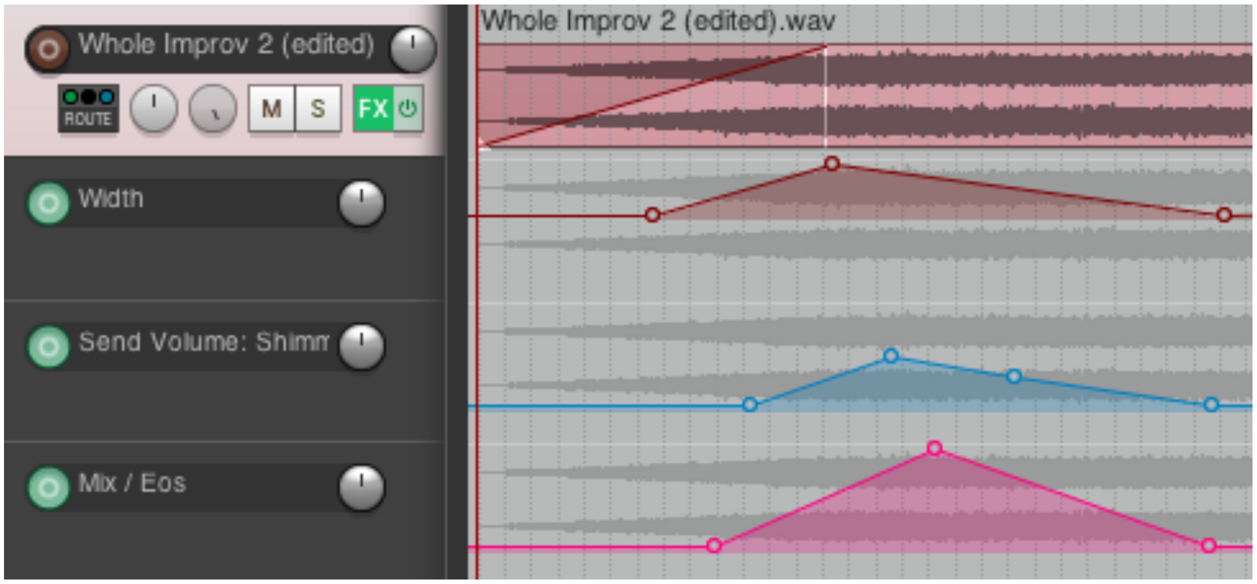


Figure 3.4 Close up of effect automation used in the A Symphony for Perth recording. The timeline here shows approximately sixty seconds of automation, which occurs between 14:24 and 15:24 in the symphony recording. The three lanes beneath the audio waveform show

One minute into the third movement of *A Symphony for Perth*, one of the sonic layers comes from a recording of musicians from Perth improvising (the waveform shown at the top layer of Figure 3.4). The three automation lanes (labeled Width, Send Volume: Shimmer, and Mix/Eos) were added with particular sound design goals in mind:

1. When the "Whole Improv 2 (edited)" clip initially begins to fade in, it is monophonic. The red "Width" envelope shows how, over approximately 12 seconds, the stereo field opens, creating a wide stereo image before collapsing back to mono over approximately 30 seconds. A wide stereo image sounds pleasing, but in this case, other stereo sound sources in the project become more important than this one. Collapsing the stereo field allows this sound to not distract from other sources and helps the result sound clear and uncluttered despite the many sonic layers.
2. The blue "Send Volume: Shimmer" envelope shows the amount of signal that will be sent to a long stereo delay of 480 milliseconds and 355 milliseconds on the left and right channels, respectively. Normally, a delay like this would be audible as a distinct echo, but in this case, the audio content does not have distinct transients, and the volume level of the delay is set so that, rather than sounding like echoes, the delay adds a "shimmer" of spaciousness to the soundscape.
3. The "Mix / Eos" envelope in pink affects the balance between direct sound and the output of a reverb plugin. This is an extreme version of a common technique:

temporarily adding reverb for dramatic effect (this technique was also used in the intro of the Ariana Grande song). This augments the spacious crescendo created by the first two envelopes. Reverb can sound beautiful, but too much reverb for too long becomes fatiguing to the ear and obscures sonic clarity. By only temporarily adding reverb, we can elicit the thrilling spaciousness of a giant hall of worship, and then direct the listener's focus to new sonic material without losing their attention or sacrificing clarity. By the time that the reverb has faded out, other sonic layers (visible in Figure 3.3, but not in Figure 3.4) have entered the mix, moving our attention away from the "Whole Improv" clip shown in Figure 3.4.

Here each envelope described above can be thought of as a sound design Technique, and together they form a composite Technique. Combined, these three patterns create a crescendo that sounds spacious and huge in a way that is both more subtle and more beautiful than when we create an artificial crescendo by directly automating a track's volume or gain. In this Technique the three automation peaks occur within a few seconds between each other, causing the peak of the crescendo to evolve over time instead of culminating at one precise moment.

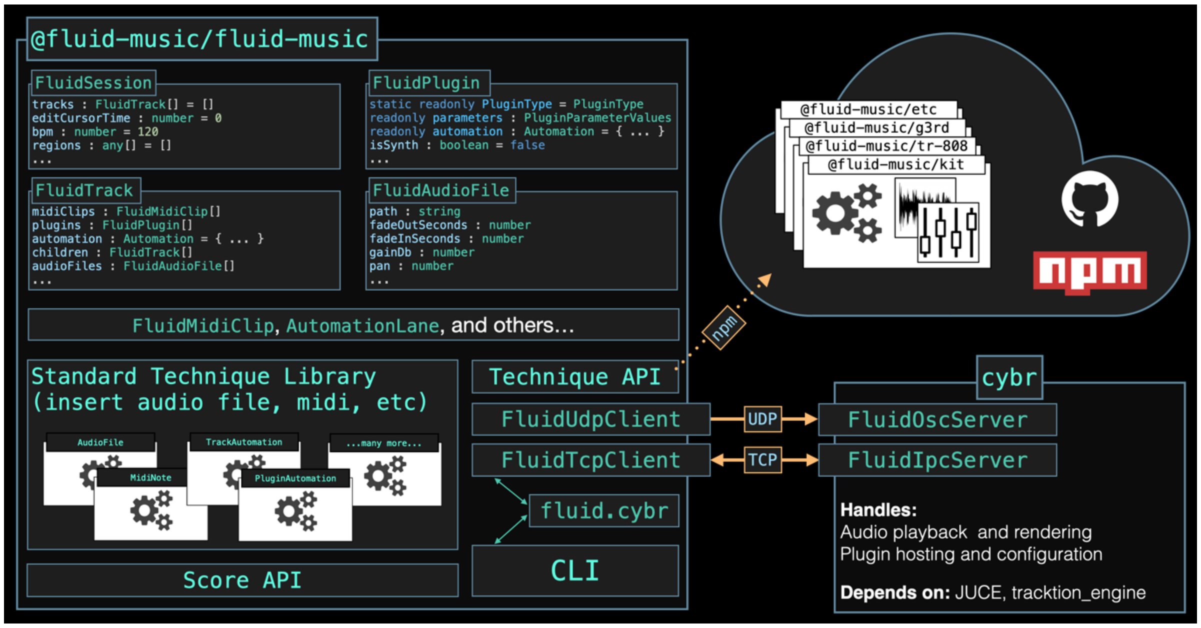
Most digital sound design follows this pattern: the engineer directs listener attention by automating and editing. This can be accomplished with filters, EQs, reverb units, panning, balances, or splicing and editing audio clips directly. Each adjustment to the DAW session is part of a plan to accomplish some creative objective. The Fluid Music framework can encapsulate these plans in Techniques and give sound designers a way to customize and re-apply them in different contexts.

4 The Fluid Music Framework

Chapter 4 describes the design and the technical implementation of the Fluid Music framework. Section 4.1 covers the design requirements that are unmet by existing languages for computational sound design. This section shows why Fluid Music could not be created with existing tools.

The framework implementation includes two parts: a client library and an audio server. The client library is written in TypeScript, which compiles to JavaScript. It is a lightweight library designed to make describing and mutating a DAW session with code as accessible as possible. The Fluid Music client is described in detail in section 4.2. This client includes an API for creating "session" objects which are covered in section 4.2.1. A session is an audio design document which resembles (and exports to) the structured data format saved by a digital audio workstation. The client includes an API for developing Techniques, which mutate sessions by adding or configuring core DAW components. The Technique API is described in section 4.2.2. The client includes a score API for sequencing Techniques in tracks along a musical timeline, which is described in detail in section 4.2.3. Finally, the client includes robust support for instantiating and configuring VST plugins. Plugin support is covered in section 4.2.4.

The Fluid Music audio server, covered in section 4.3, is written in C++ and handles audio playback, rendering, and processing, as well as VST plugin hosting and configuration.


```
FluidSession
tracks : FluidTrack[] = []
editCursorTime : number = 0
bpm : number = 120
regions : any[] = []
...

FluidPlugin
static readonly PluginType = PluginType
readonly parameters : PluginParameterValues
readonly automation : Automation = { ... }
isSynth : boolean = false
...

FluidTrack
midiClips : FluidMidiClip[]
plugins : FluidPlugin[]
automation : Automation = { ... }
children : FluidTrack[]
audioFiles : FluidAudioFile[]
...

FluidAudioFile
path : string
fadeOutSeconds : number
fadeInSeconds : number
gainDb : number
pan : number
...
```

Figure 4.1 Simplified architecture of the Fluid Music Framework. The @fluid-music/fluid-music package is written in Typescript and includes a data model for core DAW components, the standard Technique library, the Technique API for creating custom Techniques, the score API for sequencing techniques, a command line interface, and cybr module for communicating with the cybr server. The cybr server is written in C++ and handles all audio processing, playback, and rendering. Two commercial cloud services, mpm and GitHub, are also used for publishing Techniques and managing dependencies.

Finally, section 4.4 discusses what it is that makes this Fluid Music implementation a framework, and not simply a software library. This section shows how this distinction between a framework and a library is important for the project's high-level goals.

4.1 Requisites

Because the long-term goals for Fluid Music are such a dramatic departure from the way the music is currently composed and produced, the Fluid Music framework (as it exists today) is not intended to fully realize the long-term conceptual goals because there are too many interdependent outstanding questions about how to best make it work. Instead, the framework is intended to provide a robust foundation for building and experimenting with Fluid Music. The immediate goal is to enable creators to begin studying and exploring outstanding questions such as:

- What are the boundaries around what is useful to share and reuse?
- How rigid should the boundaries be?
- In what ways should reusable techniques be configurable?
- In what ways should reusable techniques attempt to configure themselves through machine listening or audio feature detection?
- How portable should Fluid Music techniques be? Should a reverb technique work with exactly one reverb plugin or algorithm? Or should it be able to parameterize several different reverb units to achieve similar results?
- How similar or different should a Fluid Music interface be to contemporary DAWs?
- How should we handle intellectual property? What kinds of content do we want users to "own" or keep as a trade secret?

At present, there are too many unanswered questions to try to fully realize Fluid Music. The framework aims to provide the tools necessary to study and answer the outstanding questions.

The remainder of section 4.1 enumerates the features and capabilities I identified as necessary to explore the possibilities of Fluid Music. Other code-based tools and libraries for manipulating sound have some of these features, but no existing tool has *all* these features. Together, the following sections describe the set of design choices that motivated the structure and capabilities of the Fluid Music framework. Without all these features in place, it would not be feasible to use this framework to step toward my long-term vision of Fluid Music.

4.1.1 Idiomatic Extension API in a General-Purpose Scripting Language

Fluid Music aims to welcome user contributions that extend its sonic vocabulary. To be welcoming to both experienced and beginning programmers, the process of writing an extension or Technique must offer a good user experience. The easiest way to accomplish this is to use an existing language that already provides robust tooling and a polished experience for developers.

Several existing code-based languages like Sonic Pi, ChucK, and SuperCollider use a domain specific language (DSL) that is explicitly intended for sound design. The advantage of this approach is that it allows that language designer to tailor the language syntax for the purpose of sound design. For example, ChucK syntax has a dedicated “patch” operator for routing signals between audio processing nodes. However, the developer experience for DSLs will never rival that of a widely used general-purpose programming language. Because the Fluid Music client API is written for Node.js, developers who are working to extend the Fluid Music vocabulary can take advantage of the many thousands of hours of effort that has already been invested in the Node.js software ecosystem to support features like syntax completion, type introspection, debugging, and comprehensive testing libraries. Audio and sound design DSLs will never be able to provide the polished experience available to users of widely used general-purpose languages.

Choosing to use a general-purpose language instead of a DSL has more advantages. When a developer spends time learning to write extensions in Fluid Music, they are also practicing and improving their JavaScript and Node.js. Learning a DSL can be an educational cul-de-sac. Writing Fluid Music extensions with the Node.js client uses all the same programming idioms as writing a web server in Node.js or writing JavaScript that runs in a web browser. Because the Fluid Music client is idiomatic, developers can translate their existing JavaScript skillset to and from the Fluid Music client language.

Another essential advantage of building the Fluid Music Client API in Node is the Node Package Manager (NPM). Package managers give developers a practical way to version and share code packages. NPM makes it very easy to publish a code package to the web under a unique name. It then becomes very easy to write new packages that depend on existing packages. NPM streamlines dependency graph resolution for a given project by recursively identifying all project dependencies and their dependencies. This kind of *transitive* dependency management is central to the Fluid Music concept described in section 3.1. Techniques need to be able to encapsulate and depend on each other, and because the user-facing portion of the Fluid Music API is written for Node.js, Fluid Music developers can take advantage of the existing tooling for managing dependencies. This is essential for testing and studying the questions outlined in 4.1 above. In the long term, as Fluid Music shifts from a software framework towards a tool for composers and musicians, it may benefit from a dedicated package manager. For now, using NPM for dependency management allows users to reliably depend on each other’s packages, while providing a well-defined specification for users to name and version their packages. This specification provides a standard on which a dedicated package manager can be built in the future, if (for some reason) it turns out that NPM does not meet the developing needs of the

Fluid Music ecosystem. The standard ensures that Fluid Music can migrate to a new platform without breaking compatibility with existing packages and Techniques.

The final reason that Fluid Music avoids a DSL relates to integration with hardware and the outside world. It is not yet clear what kinds of user interfaces (UIs) will be most useful to composers and lead to creative and interesting results. The ideal way to discover what kind of UI best suits Fluid Music is to make it easy for developers to build a variety of different kinds of interfaces. It should be welcoming to developers who want to create a UI of their own. NPM already has packages for creating user interfaces and connecting many kinds of different hardware peripherals. If Fluid Music were written with a DSL, developers would have to write a custom compatibility layer for each different UI framework or hardware interface before that interface could be used together with Fluid Music. Because Fluid Music is written to be compatible with the Node.js software ecosystem, developers can take advantage of the many existing software libraries for creating UIs.

For example, the React framework for building graphical user interfaces has seventy-three thousand dependent packages and over ten million weekly downloads from NPM as of June 2021. The node-midi package provides cross platform support for MIDI devices, and the kinect-azure package provides support for Microsoft’s latest “Kinect” depth camera model. Additionally, the Node.js standard library includes support for reading from a filesystem in a cross-platform manner, networking, and much more. To experiment with many kinds of workflows, developers need to be able to access these features without writing and maintaining adapter code for each one. As a result, by depending on the Node.js and NPM ecosystem, the Fluid Music framework can offer much more flexibility and extensibility than any DSL-oriented audio library.

A determined developer can work around some of the constraints associated with DSLs by embedding the language inside another software development environment. For example, the developers of ChucK embedded it in the Unity Game Engine. The CSound library runs in the browser, on mobile devices, or natively on major operating systems. Embedding audio libraries like CSound can mitigate some of the problems associated with DSL-oriented software. Embedding a DSL can make it easier (if still awkward) to call between the DSL and underlying host for hardware access and user interface libraries. By calling into the CSound library, users can access all the sound design primitives that CSound provides at the expense of an idiomatic API, modern package management, and modern development tooling.

4.1.2 DAW Integration

Another important requisite is the ability to inspect and edit the output of Fluid Music in a professional Digital Audio Workstation. Like DAWs, Fluid Music aims to be useful for composition. This is a fundamentally different goal from computational sound design languages like SuperCollider, Tidal Cycles Max, Pure Data, and Sonic Pi, which are focused on real-time synthesis, processing, and interaction. This distinction between composition and interaction underscores the most significant difference between the Fluid Music framework, and other languages for computational sound design. Fluid Music is built around the core DAW components and is focused on creating and sequencing these components computationally.

Today, essentially any music created *without* a DAW is an unusual exception. Why are DAWs so central to contemporary music production? Drawing on my own career of over a decade working in music production, studio recording, and sound design, the user interviews detailed in chapter 7, and anecdotal observations, I believe there are multiple reasons for this, which are covered in this section.

DAWs are designed for recording and overdubbing, neither of which is a goal of – or even the possible in – some code-based sound design languages. However, recording is also not the focus of Fluid Music, which is intended for sequencing, composition, and production.

Another reason for the importance of DAWs is that in music, details matter, and DAWs are still the best tool for detailed sound design work. Clean edits, subtle variations between a first chorus and the final chorus, and careful bus compression are among the kinds of details that make a mix sound professional and competitive (Case). The tools in a DAW let a sound designer audition and sculpt content on a timeline so that every note is special. Existing code-based tools, and graphical tools that do not use tracks and a timeline to sequence content like Max and Pure Data make this kind of precise editing difficult.

For a recent example of this kind of effect that is easy to achieve in a DAW, but more difficult to create with real-time audio software, listen to Post Malone's 2019 song *Circles*. During the verse the last word of certain vocal phrases is processed by an exaggerated reverb effect so that a diffuse, wet echo of the lyric is artificially sustained in the mix before the next line begins. This technique performs a strategic musical function – listeners can enjoy the rich, reverberant decay which blends with the harmonic texture that underpins the song. If the exaggerated reverb were applied to the entire vocal track, the reverberant signal would mask other lyrics, fatigue the ear, and make the overall mix sound indistinct. One way to create this effect is to automate the portion of the vocal track that is sent to a reverb unit and the output gain level from the reverb itself. This effect works best when the mix engineer intentionally tailors the precise envelope, gain levels, and reverb configuration for a song or composition so that the reverb supports the music rather than interferes with it. The tools in a DAW are the easiest way to precisely automate the gain, routing, and plugins to achieve this kind of effect.

By integrating with existing DAWs and professional audio plugins, Fluid Music enables us to begin the process of automating techniques like the one above. It allows us to step toward a world where instead of manually configuring tracks, automation, and plugins, a sound designer can invoke a Reverb Automation Technique that applies all the underlying configurations in one step. DAW and plugin integration are not strictly necessary for this. However, what we get from DAW and plugin integration is the ability to inspect and edit the output of automated techniques. This enables a mix engineer to override the choices made by an algorithm, which is important, because professional engineers can create much more polished results than automated mixing algorithms, as evidenced by the non-use of automated mixing in professional music production. Dependence on automated mixing techniques is likely to increase in the future as the technology improves. However, for automated mixing to become a useful technology, we need a viable mechanism to collect data that shows how skilled mix engineers achieve polished results. Fluid Music offers a viable path to begin collecting that data, while ensuring that the automated mixing decisions can be inspected and overridden by a human in the loop.

4.1.3 Plugin Integration

Audio plugins are another important component of Fluid Music. Full-featured plugin support is part of what distinguishes DAWs from other types of code-based tools and graphical patching environments like Max and Pure Data. Tools like SuperCollider, CSound and Max are capable of hosting DAW-centric audio plugins including VST plugins, but they tend to favor a workflow that involves building effects and synthesizers from scratch with primitive digital signal processing (DSP) nodes like oscillators, filters, and buffers. One way that this preference manifests itself is in parameter automation. Most audio plugins have automatable parameters that configure how that plugin processes sound. For example, the open-source Dragonfly Hall VST2 reverb plugin (Willis) has eighteen automatable parameters including a setting that adjusts the size of the simulated reverberant room, and settings to adjust the balance of early and late reflections in the computed reverberant signal. To computationally configure the “Size” parameter, which is a value in the range of 10 to 60 meters, the user must reverse engineer an unspecified function that maps to a normalized value in the range [0, 1]. This is potentially a non-linear or piecewise function. When setting a VST parameter from existing code-based tools for computation sound design, the user must iteratively guess and check parameter settings, making plugin configuration time-consuming and impractical.

While a determined developer can work around some of the obstacles associated with hosting professional audio plugins outside of a DAW, non-DAW environments consistently work best for programming audio effects with the DSP primitives that are integrated into computational sound design toolkits. In contrast, Fluid Music commits to first-class support for hosting and configuring VST plugins. The motivation for this choice is simply that it is essential for Fluid Music users to be able to inspect and edit their compositions using a DAW. The simple DSP primitives that are the foundation of existing computational sound design toolkits are not core DAW components and cannot be hosted or edited by most DAWs.

There are still disadvantages to depending on audio plugins. While there are many high-quality free and open-source audio plugins, there are also many popular commercial plugins, which are not available to everyone and can be expensive. A Fluid Music Technique that depends on a particular plugin can only be used by users who have access to the plugin. Additionally, as operating systems update, formats evolve, and software dependencies fall out of sync, older plugins are less and less likely to run correctly. Commercial plugin creators are incentivized to drop support for older hosts and operating systems so that users are required to purchase a newer version when they update their production host.

Additionally, commercial plugins are typically closed source, which can make it difficult to study and understand exactly what a plugin does. Common plugin types like reverb units may be configured to sound similar and be somewhat interchangeable, but a closed source niche synthesizer may have no viable alternative. When a closed source plugin with copy protection is not updated and no longer runs on modern operating systems it can be very difficult or impossible to use that plugin and any software or DAW sessions that depend on the unsupported plugin will no longer sound correct.

Despite the detractors, audio plugins are an important part of contemporary music production, and an important part of Fluid Music. As a result, the Fluid Music framework employs several strategies for mitigating the disadvantages, which are covered in section 4.2.4. Robust support for computational plugin hosting and configuration in the Fluid Music framework was enthusiastically welcomed by Fluid Music users in the user tests described in chapter 0, with multiple users claiming that this is a highly desirable feature that is missing from other computational sound design workflows.

4.1.4 OS Compatibility and Commercial Dependencies

Finally, the Fluid Music must be able to run headless (without a graphical user interface), and it must be able run natively on Windows, MacOS, and Linux. A headless Linux option enables users to build applications around Fluid Music that can be deployed at the scale of the internet. Support for MacOS and Windows ensures that the software is accessible to as many users as possible. Additionally, Fluid Music must not depend on any closed-source commercial software, which would also limit its accessibility and potential user-base.

This constraint led to an unusual compromise between the Fluid Music framework and commercial software. The framework supports integration with commercial DAWs, but it does not depend on them. You can use Fluid Music to create Techniques, compose music, play audio, and render audio files without any external commercial software. However commercial software is important in music production. In a space as competitive as recorded music, even a very small advantage can be important. Fluid Music aims to be useful for making music that people care about in a variety of genres. This is quite difficult to do without using any commercial software.

I considered building the Fluid Music framework around an existing digital audio workstation. Some DAWs have built in scripting support, and in the case of Reaper, that support is comprehensive enough to build reasonably elaborate applications with. I decided that even a comparatively inexpensive and accessible DAW like Reaper would too severely limit the range of applications for which Fluid Music would be useful, as well as limiting the range of users who could contribute to the Fluid Music language. It would also limit long term support of Fluid Music to the lifetime of the underlying DAW. As a result, I have designed the Fluid Music framework to have open-source dependencies that can be used for free. At the same time, it must not prevent sound designers from using the professional software they depend on for creating musical content.

4.2 The Client: fluid-music

The Fluid Music client is a software library for computational sound design and music composition. It is available under the name "fluid-music" via the Node Package Manager (NPM) or on GitHub as https://github.com/fluid-music/fluid-music. The library exposes a simple but powerful API for manipulating core DAW components. It is written in TypeScript which compiles to JavaScript. The motivation for using a general-purpose scripting language is to make it welcoming to developers with a wide variety of different backgrounds and competencies. As a comparatively high-level language, JavaScript applications are not ideal for computationally expensive tasks like rendering audio or hosting plugins. The Fluid Music framework handles computationally expensive tasks on a dedicated audio server which is written in C++ and described in section 4.3. As a result of this client/server architecture Fluid Music users can author sound design Techniques with the conveniences of JavaScript and its robust development tooling, share Techniques via NPM, and playback or render audio results using the high-performance audio server. The JavaScript classes and idioms covered in the following sections explain how the fluid-music client API is designed to ease the process of computational composition and satisfy the design requisites covered in section 4.1.

4.2.1 FluidSession

The Fluid Music client library defines a TypeScript class called `FluidSession` which models the core DAW components described in section 3.3.1. A `FluidSession` instance can contain any number of tracks. Each track can contain audio clips, MIDI items, plugins, and automation. Because `FluidSession` only models core DAW components that work similarly between different DAWs, it is comparatively simple to write exporters that target essentially any existing DAW session format, as well as new formats that may emerge in the future. The fluid-music package includes exporters to both the Reaper `.RPP` session format and the Tracktion Waveform `.tracktionedit` session format.

For modeling an audio graph, `FluidSession` has a tracks property, which is an array of `FluidTrack` instances. Each Track object has a configurable gain, pan, and width control, as well as the ability to host VST plugins, and send audio to other tracks (with the option to act as a sidechain input to another track). Each track may also act as a sub-mix track folder which contains one or more child tracks. A parent track sums the output of all its child tracks, thereby organizing tracks in nested groups which can be processed together. The example session below shows how to create a `FluidSession` instance in JavaScript that includes tracks, a reverb plugin, sends, and sidechain compression.

```
const fluid = require('fluid-music')

const compressor = new fluid.plugins.RoughRider3Vst2({ externalSidechainEnable: 1 })
const reverb = new fluid.plugins.DragonflyHallVst2({ decaySeconds: 4 })

const session  = new fluid.FluidSession({ bpm: 120 }, [
  { name: 'main', children: [
    { name: 'bass', plugins: [compressor.sidechainWith('kick')] },
    { name: 'kick' },
    { name: 'vox', gainDb: -3, sends: [{ to: 'verb', gainDb: -12 }] },
    { name: 'keys', gainDb: -6, pan: 0.6 },
    { name: 'verb', plugins: [reverb]}
  ]}
])
```

The `FluidSession` data model also supports sequencing content on the `FluidTrack` instances contained in a `FluidSession`. Each track may contain automation for gain, panning, and width. VST plugins contained on a track may contain automation for all automatable parameters. Tracks can contain MIDI clips or audio clips that reference audio files. Audio clips also have many parameters for fading, trimming, tuning, and stretching audio content.

The `FluidSession` data model is designed to be general enough to export to virtually any DAW, but comprehensive enough to accurately describe the precise edits, fades, and automation that sound designers need to make content that sounds professional. It includes all the core DAW features described in section 3.3.1. However, the core Fluid Music framework does not attempt to support extended DAW features articulated in section 3.3.2. Instead, the Fluid Music framework is itself extensible. It enables users who want access to extended DAW behaviors to write or reuse custom Fluid Music Techniques that mutate the core DAW components.

This design choice trades the convenience of high-level abstractions available in some DAWs for other behaviors that orient the Fluid Music towards its larger goal of creating an extensible language for sound design. This choice reduces the sound designer's dependence on monolithic closed-source DAWs. Users who want a particular feature do not have to petition the DAW creators for support. Instead, they can author their own Technique, and optionally share that Technique so that others can reuse it or reproduce their results.

This extensible nature of the Fluid Music framework also requires sound design sessions to explicitly declare which Techniques or features it relies on. For example, a sound design document that requires a particular auto-panning Technique must specify the exact Technique that was used. As a result, each Technique must have a globally unique name or identifier so that it can be unambiguously referenced when it is used as part of another project or parent Technique. Additionally, Techniques must conform to a versioning standard so that Technique authors can publish updated versions without breaking projects that rely on an existing version. NPM comes with built-in solutions for naming and versioning.

While this extensibility lessens the dependence on proprietary DAWs (that may or may not exist a decade from now), archiving sound design sessions is not the main goal of the Fluid Music framework. Still, compared to a traditional DAW, the Fluid Music system has non-trivial advantages over conventional DAW-based sessions when it comes to preserving sound design sessions for long-term digital permanence.

If a DAW is not maintained by the developers, it will become difficult to open, play, or edit session files that were created in that DAW. Hardware and software standards that evolve over time make it increasingly difficult to use unmaintained software. Similarly, Fluid Music depends on the availability of JavaScript interpreters and the maintenance of the NPM ecosystem for long-term stability. However, NPM's "pack" command will automatically find and download all the dependencies of an NPM package and create an archive that bundles that package along with all its dependencies. NPM "pack" and other dependency management tools that come with NPM have powerful advantages over the monolithic archival challenges faced by users of conventional DAWs, especially when copy protection measures require that software contact an authorization server to run.

Because the `FluidSession` data model encapsulates only core functionality that is common to different DAWs, the burden of maintenance is also proportionally smaller. It is comparatively easy to create a `FluidSession` exporter that targets a new DAW format – especially when juxtaposed with the monolithic task of converting a session file from one conventional DAW format to another.

Like DAWs, the Fluid Music code base must be maintained for long-term archival quality support. Unlike DAWs, the Fluid Music framework offers a viable path toward long-term session archival. It does this by setting clear boundaries around the session data model, limiting this model to the core DAW components, and necessitating that extended techniques be described in terms of the core components. Additionally, because JavaScript is an interpreted language, user-created Techniques are open source by default. As a result, sharing a Technique means also sharing the code that generated that Technique. However, currently, the primary goal of Fluid Music is enabling many users to share and reuse the kinds of music production techniques that distinguish the art of sound recording and studio music production from the capture and reproduction of a live musical performance. This is different from long-term session archival, and a user who wants to future-proof a `FluidSession` instance still needs to take extra steps to do so.

4.2.2 Techniques

The `FluidSession` constructor lets a user create a session and configure its audio graph by adding tracks and routing audio between them. The idiomatic way to add content to tracks in Fluid Music is to use Techniques, which were introduced in section 3.2. The Fluid Music client explicitly defines the Technique API which is described with the following TypeScript interface.

```
export interface Technique {
  use : {(context : UseContext) : any }
  finalize? : {(session : FluidSession) : any }
}
```

The interface definition above specifies that a Technique is any JavaScript object that contains a function named “use” which must accept a UseContext object as its first argument. The full UseContext documentation is available at https://fluid-music.github.io. Techniques may optionally contain a function named “finalize” which accepts a `FluidSession` class instance. This is the main interface that developers need to understand to author custom Fluid Music Techniques.

Below is a very simple example Technique written in JavaScript. The JavaScript object defined below has a single property called “`use`,” which is a function that accepts a context object, and simply logs the name of the track on which the Technique was invoked, in addition to the time at which it was invoked relative to the start of the `FluidSession` instance. The example also shows how to use the technique on a `FluidSession` instance.

```
const fluid = require('fluid-music')

// Create a simple custom Technique
const logTechnique = {
  use: (context) => console.log(context.track.name, context.startTimeSeconds)
}

// Create a FluidSession instance
const session  = new fluid.FluidSession({ bpm: 120 }, [{ name: 'bass' }])
```

```
// Invoke logTechnique on the 'bass' track with a start time of 3 seconds
session.useTechnique(logTechnique, { track: 'bass', startTimeSeconds: 3 })
```

The example above is simple, and the fluid-music package documentation covers many features that are skipped over this guide. However, the example above illustrates the most important parts of the Technique API. Techniques are simply JavaScript objects with a `use` method. When a Technique is "used" it is invoked on a track in a session at a particular time. It is not shown in the example above, but the `UseContext` object passed to the Technique's `use` function has many other parameters including that can be used to customize the behavior of the Technique.

Unlike the `logTechnique` example above, most practical Techniques do more than simply print contextual information. The Fluid Music client includes techniques for common actions like inserting and editing an audio sample, adjusting a plugin parameter, or inserting automation points on a particular Track. The standard Techniques that come bundled with the Fluid Music client are implemented as classes. JavaScript class constructors provide an often useful way to customize the behavior of a technique. For example, the `AudioFile` Technique constructor shown in the following code example has many optional parameters for things like gain, variable speed playback, and pitch shifting.

```
const fluid = require('fluid-music')

// Create a FluidSession instance
const session  = new fluid.FluidSession({ bpm: 120 }, [{ name: 'drums' }])

const tambourine = new fluid.techniques.AudioFile({
  path: '/samples/tambourine.wav',
  info: { duration: 3 },     // duration of tambourine.wav
  durationSeconds: 1.8,      // truncate playback after 1.8 sec
  startInSourceSeconds: 0.1, // skip over first 0.1 seconds
  fadeInSeconds: 0.01,       // fade in time
  fadeOutSeconds: 0.25,      // fade out
  gainDb: -3,                // adjust playback gain
  pan: -0.2,                 // pan slightly to the left
  playbackRate: 0.5,         // playback at half speed
  pitchSemitones: -7,        // pitch shift
})

session.useTechnique(tambourine, { track: 'drums', startTimeSeconds: 10 })
```

One important characteristic of the Technique API is that it makes it easy to create techniques that invoke other techniques. The next example creates a derivative technique using the `AudioFile` technique from the example above. The code below exemplifies a common Technique pattern. By forwarding a `UseContext` object, custom Techniques can invoke sub-techniques. By calling many sub-techniques, custom Techniques can perform complex composite actions.

```
const randomizedTambourine = {
  // Insert a tambourine sample with random pitch and gain
  use: (context) => {
    tambourine.pitchSemitones = Math.random() * 2 - 1
    tambourine.gainDb = Math.random() * -6
    tambourine.use(context)
  }
}
```

Each time a Technique's `.use` function is called, the full underlying session is available to that function via `context.session`, which allows Techniques to mutate any DAW components contained in the session, including components that are not part of the track that the technique was invoked on. Essentially, Techniques can perform any sequence of actions that a DAW user could perform with a graphical user interface.

4.2.3 Fluid Music Scores

The previous section demonstrated the `FluidSession`'s `useTechnique` method, which is useful when you want to insert one Technique at a particular time. However, this is impractical for music composition. When composing sheet music with pencil and paper you can quickly add a chord within a measure on the page. In contrast, typing `session.useTechnique(someTechnique, { startTimeSeconds: x, track: 'y'})` for every musical event is very time consuming. The Fluid Music client package includes a code-based score language designed to ease the process of expressing musical ideas in a text editor. Where other code-based systems for composition use a domain specific language for this purpose, the Fluid Music score language is built with JavaScript primitives like strings and objects. This makes it easy to embed musical sequences in software projects that are too large or complex to be written in a domain specific language. The example below shows a drum pattern written in the Fluid Music score language. It shows how the score language is optimized for invoking many Techniques on a musical rhythm.

```
const fluid = require('fluid-music')

const tLibrary = {
  t: new fluid.techniques.AudioFile({ path: '/clap.wav', info: { duration: 2 }}),
  D: new fluid.techniques.MidiNote({ note: 36 }),
  s: new fluid.techniques.MidiNote({ note: 40 }),
}

const score = {
  tLibrary: tLibrary,         // describe the score vocabulary (s/D/t)
  r:     '1 + 2 + 3 + 4 + ', // describe the score rhythm (16th notes)
  snare: '    s       s   ',
  kick:  'D            D  ',
  tamb:  't t t t t t t t ',
}
```

The remainder of section 4.2.3 breaks down how scores like the one above work and explores more advanced features of the score language.

The example above begins by creating a Technique Library which is simply a JavaScript object that maps single characters to Fluid Music Techniques. Keys in a technique library can be any single Unicode character including emoji, which allows Technique Libraries to be incomprehensibly large. Values in a technique library must be Fluid Music techniques (JavaScript objects with a `use` function). Technique Libraries offer developers a way to curate a collection of Techniques that can be referenced in a score.

There are many ways to include a Technique Library in a score. The example above inserts the Technique Library into the score object using the “`tLibrary`” key. In the Fluid Music score language, most object keys refer to track names. In the example above, “`snare`”, “`kick`”, and “`tamb`” all refer to tracks in a `FluidSession`. However, some keys like “`tLibrary`” have a special meaning and are interpreted differently by the Fluid Music score parser.

Just after the “`tLibrary`” key, the score object contains an “`r`” key which is another score object property with a special meaning. An “`r`” signifies the presence of a Rhythm String, which encodes a sequence of musical durations with a sequence of string characters. In the example above, the sixteen characters (including spaces) in `'1 + 2 + 3 + 4 + '` indicate sixteen consecutive sixteenth notes. Rhythm strings are best explained with some examples:

```
'1234'      // has four characters, and represents a sequence of four quarter notes
'w'         // is one character, and represents a single whole note
'hh'        // has two characters, and represents a sequence of two half notes
'h h '      // represents a sequence of four quarter notes (spaces may subdivide beats)
'h.h.'      // is equivalent to 'h h ' (dots may subdivide beats)
'1..2..'    // has 6 characters, and represents six eighth-note triplets
'1+2+3+4+'  // is eight characters, and represents eight consecutive 8th notes
'1 2 3 4 '  // is identical to '1+2+3+4+'
'1 + 2 + '  // is eight characters and represents eight consecutive 16th notes
'1e+a2e+a'  // is identical to '1 + 2 + '
'1....'     // represents five quintuplets (any number of subdivisions are possible)
'h34'       // represents a half note and two quarter notes. 'h' always represents a
half note, and 'w' always represents a whole note. Notice that this works a little
differently than quarter notes: the duration of a '1' symbol may be shortened by the
character that follows it ('1+' represents two eighth notes, not a quarter and an
eighth note). While 'w' and 'h' may be divided into subdivisions like any other value
```

```
('w...'), the total duration incurred by 'w' and 'h' is never affected by subsequent
characters.
```

The remaining properties in the score object example are Pattern Strings found in the score object’s “`snare`”, “`kick`”, and “`tamb`” properties. Pattern Strings reference Techniques found in any associated Technique Libraries, and invoke them on a particular track at a time indicated by their position in the Pattern String matched with the same position in an accompanying Rhythm String. As a result, the `'t t t t t t t t '` string in the score object represents eight calls to `session.useTechnique` which invoke a technique on successive eighth notes within the “`tamb`” track. The Technique that will be invoked is determined by the value of the “t” property in the accompanying Technique Library.

The previous score example is contrived. A more practical and musical example is below. This following example demonstrates some additional score language features and shows how to insert a score into a `FluidSession` instance.

```
const fluid = require('fluid-music')
const kit = require('@fluid-music/kit')

// Create a Session, specifying the beats-per-minute and audio/midi tracks
const session = new fluid.FluidSession({ bpm: 96 }, [
  { name: 'drums', gainDb: -6, tLibrary: kit.tLibrary, children: [
    { name: 'snare', gainDb: -3 },
    { name: 'kick' },
    { name: 'tamb', pan: 0.2 },
  ]}
])

// Insert the score object into our session.
//
// Each character (s/D/t) in the score below invokes a function which
// inserts an audio sample into the session. These functions and samples
// are bundled within the '@fluid-music/kit' npm package.
session.insertScore({
  r:      '1 + 2 + 3 + 4 + ', // describe the score rhythm (16th notes)
  snare: ['    s       s   ', '    s       s   '],
  kick:  ['D               ', '          D  D  '],
  tamb:  ['t t t t t t t t ', 't t t t t t t t '],
})

// Save the session as a Reaper file
session.finalize()
session.saveAsReaperFile('beat.RPP')
  .catch(e => console.error('Error:', e))
  .then(() => console.warn('Exported: beat.RPP'))
```

This second line of the example above imports the "@fluid-music/kit" npm module. This module defines some custom drum kit Techniques and exports an accompanying Technique Library. Some of the techniques in "@fluid-music/kit" are more elaborate than the simple AudioFile and MidiNote techniques we have seen so far. For example, the tambourine Technique identified by "`t`" randomly selects an audio file from a collection of different tambourine samples each time it is used. This is a common production technique that makes the tambourine sound less mechanical when it is played repetitively. Most DAWs include a sampler which would provide a similar sample randomization effect. However, the idiomatic way to create this kind of behavior in Fluid Music is to write a package like "@fluid-music/kit." The important takeaway is that anyone can create a package like this one, give it a unique name, publish it to NPM, and consume it in their JavaScript code. Like "@fluid-music/kit," packages can include specific audio samples. Alternatively, a package might include a general-purpose randomization technique that can be reused in combination with any other Techniques or audio files.

The example above also shows how JavaScript arrays can contain multiple elements. When the score parser encounters an array, it treats the array elements as a sequence. Any Pattern String can also be replaced by another score object or an array of Score Objects, which makes it possible to write large complex scores using only JavaScript language primitives. The score parsing algorithm is recursive, so it can step through deeply nested score objects. When the score parser encounters a pattern string, it matches with the nearest Rhythm String. This allows child objects in deeply nested scores to override the parent's Rhythm String when, for example, a small region of a composition is written in thirty-seconds notes while all other sections of the score are written in sixteenth notes.

Technique resolution also works recursively. When the score parser encounters a character in a Pattern String within a deeply nested score, the parser searches all the parent JavaScript objects for Technique Libraries, starting with the innermost object containing the pattern string, and ending with the outermost score object. The recursive score parsing behavior ensures that nested score objects can override parent properties when needed.

The score language is useful for expressing musical ideas with code, but it is not intended to be the only way (or even the primary way) to create Fluid Music. The Fluid Music framework gives developers a foundation that more elaborate interfaces can be built on. Importantly, custom Techniques that follow the Fluid Music specification can be used by many different user interfaces, including any interfaces that emerge in the future.

4.2.4 VST Plugin Support

Section 4.1.3 explains why robust plugin support is essential for Fluid Music and why plugin support also comes with disadvantages. However, it does not cover the steps that Fluid Music takes to mitigate the disadvantages. An example code block in section 4.2.1 instantiates a Dragonfly Hall Reverb VST2 plugin, sets the “decay” plugin parameter value to 4 seconds, and inserts that plugin into a `FluidTrack` within a `FluidSession` instance. The example shows that VST plugins can be instantiated from JavaScript but does not explain how this is accomplished.

This section covers technical details of how the Fluid Music Client supports plugins, and the motivations for the underlying design choices. It also shows how these underlying design choices mitigate some of the downsides of using plugins.

Currently, Fluid Music only supports audio plugins in the VST2 format. The choice to support VST2 plugins was a difficult one based on many factors. The VST2 format offers several advantages:

- All major DAWs support VST2, so Fluid Music sessions using VST2 plugins can be interoperable with DAWs. DAW interoperability is an essential feature of Fluid Music. It is also part of what distinguishes it from other code-based tools for sound design and music production.
- There are many high-quality, free VST2 plugins available.
- Most of the high-quality, popular plugins that producers and sound designers depend on are available in VST2 format. These plugins are often not available in other plugin formats that have fewer commercial ties or legal restrictions.
- VST2s are well supported on Windows, Linux, and MacOS.
- The Fluid Music server uses the JUCE C++ framework which has robust support for hosting VST2 plugins.

However, there are also disadvantages:

- The VST2 standard is not open. Steinberg, the company that created the format, does not allow anyone without an existing license to distribute VST2 plugins or VST2 hosts that depend on the official SDK. Fluid Music avoids this problem by using the GPL licensed open-source FST headers, which are the product of reverse engineering the format’s binary interface.
- VST plugin state is not always fully configurable from the host. Some VST2 plugins have parameters that can only be modified from the plugin's graphical user interface (GUI) or by saving and loading a preset.
- The format is not well defined, and different plugin hosts support slightly different feature sets.

- Sometimes simple modular audio primitives like a buffer or oscillators are useful for sound design, and while these are readily available in languages like CSound, SuperCollider, and Sonic Pi, VST plugins are not ideal for creating modular, minimalist audio processing nodes.

Configuring VST2 plugins from JavaScript also presents a challenge. The VST2 SDK is written in C++ and is designed for real-time performance. It is therefore not well suited to JavaScript's weakly timed event-loop architecture. The Fluid Music client offers lightweight tools for creating audio sessions, but it is not intended for audio processing, which is handled by the Fluid Music server.

Because of these factors, the Fluid Music client handles plugins the same way that it handles audio files. Instances of the AudioFile class wrap around audio files that are available to the server. AudioFile instances store audio file metadata that the client uses when inserting references to the audio file into a session. If the client has access to an audio file's metadata, the client can insert references to that file into a `FluidSession` even if the audio file is not available to the client. This separation of concerns allows the Fluid Music server to handle all audio file loading and processing.

Similarly, the `FluidPlugin` class encapsulates plugin metadata, including information about a plugin's parameters, the range of values supported by those parameters, the number of plugin inputs and outputs and their associated data types. As a result, the client instantiates the `FluidPlugin` class, sets parameter values, creates automation, and loads presents from binary data all without access to the underlying plugin.

This means that to fully support a plugin, the user must extract all the plugin's metadata, including all its inputs, outputs, and parameters, and then use the metadata to create a new class that extends `FluidPlugin`. The derived class will include all the details about that plugin's interface so that users can instantiate and manipulate it from the JavaScript client. In the Fluid Music client library, classes that extend `FluidPlugin` to support a particular plugin are called Plugin Adapters.

Creating a Plugin Adapter manually is complicated and error prone, so the Fluid Music client library comes with tooling to streamline the process. The Fluid Music client library is bundled with a command line interface (CLI) with helper commands for common Fluid Music development tasks. When creating a Plugin Adapter, the first step is to request a plugin report from the Fluid Music server using the `fluid vst2 <pluginName>` command. This command asks the Fluid Music server to load an instance of a particular plugin and send a report containing as much as possible about that plugin back to the client.

The Fluid Music client tooling can use the report to automatically generate a Plugin Adapter class, but limitations in the VST2 format make this difficult. From the perspective of a VST2 plugin host, all plugin parameters are set as normalized values in the range [0, 1]. When a plugin parameter value is set using this normalized scale, the plugin usually maps the normalized value to some arbitrary range determined by the specific plugin. The mapping function could be continuous, linear, non-linear, piecewise, or discrete. An unfortunate limitation of the VST2 plugin format is that plugins report mapped values as a string, not a number. Additionally, plugins do not report the mapping function that converts a normalized input value in the range 0-1 to the value shown in the plugin user interface. This makes configuring VST plugins from code difficult, because there is no easy way to set a plugin parameter to a specific value because the parameter's mapping function is unknown.

The Fluid Music tooling addresses this issue by iterating over all the plugin's parameters and making a series of guesses about the mapping function. For each parameter, the Fluid Music tooling compares the normalized input values with parsed output values and attempts to extract the units of the parameter (for example, seconds or Hz) and the range (for example [20, 20000]). The tooling also attempts to identify if the mapping function is linear or non-linear, and discrete or continuous. In the case of discrete functions, it attempts to identify the discrete choices, for example ["sine", "saw", "triangle"]. The guesser also suggests a JavaScript friendly camel case parameter name for all parameters. For a parameter named "OSC2: pan" the guesser will suggest "osc2Pan." After iterating over all plugin parameters, the guesser generates a secondary report about the plugin that can be used to auto-generate a TypeScript Plugin Adapter class that extends `FluidPlugin`.

Most of the time the tooling extracts plugin parameter metadata accurately. However, it is not unusual for the guesser to get the mapping for some parameters wrong. In this case, a user can manually edit the secondary report produced by the guesser before auto-generating an Adapter Plugin.

The tooling for generating Plugin Adapters is complex, but it comes with some big payoffs. First, adapter classes only need to be generated once. Anyone can create a Plugin Adapter and publish it to NPM where everyone can access it. Plugin Adapter classes provide an ideal combination of audio quality and developer experience, especially compared to the plugin support that is available in other code-based tools for computational sound design. The generated TypeScript adapters work well with JavaScript code editors like Visual Studio Code and IntelliJ IDEA to provide advanced features like syntax suggestions and auto completion when configuring plugins. When configuring a VST2 plugin from code, developers can rely on the fuzzy search feature built into modern development environments to suggest plugin parameter names. This way developers do not need to remember names like "filter1CutoffHz" every time they want to get or set a parameter value. This makes it easy to write code that is consistent and readable.

The following preset example was copied directly from the Fluid Music client library. The example shows how to create a Moog-like bass sound using the Podolski VST plugin Adapter. Notice that the plugin parameter units are the same units used by the plugin's user interface.

```
/** Almost pure sine tone with a very short attack and release */
export function sine() {
  return new PodolskiVst2({
    // Set the wave shape to a triangle
    osc1WaveWarpPercent: 100,
    // Tune the oscillator so note 69 (the A above middle C) is 440hz.
    // This is needed because the default patch has 'transpose' set to -12.
    osc1Tune: 12,
    // Filter out harmonics above the fundamental. Set via trial and error.
    // The result is not a pure sine tone, but it is close.
    vcf0Cutoff: 16,
    // Full Key Tracking
    vcf0KeyFollowPercent: 100,
    // In mono and legato modes, Podolski outputs an audible 'click' when the
    // onset of a note EXACTLY coincides with the release of the previous note.
    // While overall, it makes the more sense for a sub-bass oriented patch to
    // be monophonic, setting it to poly avoids triggering the 'click' artifact.
    vccMode: PodolskiVst2.parameterLibrary.vccMode.choices.poly,
    // Release just long enough to avoid clipping at low frequencies. A value
    // of 16 results in approximately 25ms of release.
    env1ReleasePercent: 16,
    // The filter cuts the fundamental, so boost to full volume
    mainOutput: 200,
  })
}
```

By making plugin configuration human readable in addition to machine readable, Fluid Music helps to strengthen some of the long-term fragility of sessions that depend on DAW plugins. Most plugins save their state in proprietary binary blobs that obscure their configuration. The preset configuration above exposes the choices and intent of the sound designer that created the preset. With this information, future digital archeologists have a chance of understanding how certain sonic results were achieved and reproducing those results with a similar synthesizer if the Podolski VST2 plugin ceases to be maintained.

Unfortunately, some plugins have non-automatable parameters that can only be modified from that plugin's GUI. Currently, the only way to access non-automatable plugin parameters is by opening the plugin in a DAW with a GUI, adjusting the desired parameter, saving a preset file, and then loading that preset file in a Plugin Adapter instance on the Fluid Music client. The Fluid Client supports loading `.fxp` VST2 preset files, thanks to the vst2-preset-parser package which developed for this purpose and is available at https://github.com/CharlesHolbrow/vst2-preset-parser. Using a `.fxp` preset allows Fluid Music users to edit plugins in a GUI and then load them into a `FluidSession` at the expense of the readability of the plugin configuration. A hybrid approach is also possible. In the hybrid approach the sound designer will load a preset to configure non-automatable parameters and then use the Plugin Adapter's parameters to further customize the plugin preset.

The Fluid Music client library comes bundled with Plugin Adapters for several existing VST plugins. These high-quality plugins are ideal for Fluid Music because they are available for free on MacOS, Windows, and Linux, and because they have no copy protection.

- U-He Podolski, a Virtual Analog/Digital hybrid synthesizer
- U-He Tyrell N6, a virtual analog synthesizer modeled after the Roland Juno 60
- U-He Zebralette, a modern digital wavetable synth
- Dragonfly Hall Reverb
- Dragonfly Room Reverb
- Audio Damage Rough Rider 3 Compressor
- Valhalla Freq Echo Delay

Fluid Music also includes Plugin Adapters for the following commercial plugins. These were originally chosen because of their quality, their usefulness, and because they are available on MacOS, Windows, and Linux. However, I plan to move these Plugin Adapters into a dedicated NPM module because they are not available for free.

- Tracktion #TCompressor
- Tracktion #TEqualiser
- Tracktion #TStereo Delay

Finally, a Plugin Adapter for Native Instruments Battery 4 plugin is available via the "@fluid-music/battery-4" NPM Package. This package is not included in the main Fluid Music client library because it is a commercial plugin that is not free and is only available for MacOS and Windows. However, it can be a useful example for developers who want to create their own Plugin Adapters for plugins that are not already supported.

4.3 The Server: cybr

In their software implementation, all DAWs work essentially the same way. Each DAW manages an audio graph and an event sequencer. The audio graph is a network of DSP modules called nodes. Each node has some combination of data inputs and outputs, and performs some action or mutation on the data (often audio or MIDI) that passes through it. For example, a simple gain node could accept a single audio input, adjust the gain of that audio input by multiplying every audio input sample by a constant scaling factor and forward the resulting value to its output. A mix bus node could have several audio inputs that are summed together, and then forwarded to the mix bus node's audio output. When a user creates a new track in a DAW, under the hood the DAW will instantiate one or more nodes and then connect those nodes to the audio graph. When the user inserts an audio file into a track, the DAW will add a node to the graph. That audio node is responsible for reading and parsing the audio file and forwarding the output to the parent track's input bus node according to the audio sequencer. The DAW is responsible for updating the audio graph as the user modifies the underlying content.

The Fluid Music server, named cybr, is effectively a headless DAW. The C++ source code and precompiled binaries are available on GitHub at https://github.com/fluid-music/cybr. It depends on two existing open-source audio libraries, JUCE and tracktion_engine. JUCE is a general-purpose library for high-performance audio processing. The tracktion_engine library also depends on JUCE, and implements classes for DAW components including sessions, tracks, audio clips, and plugins. It is the same C++ engine used by the Tracktion Waveform DAW, and it handles much of the heavy lifting involved with maintaining an audio graph and adding and removing audio nodes in response to changes in an underlying session, in addition to audio playback and rendering. Instead of building a graphical user interface on tracktion_engine, cybr opens TCP and UDP network ports, and listens for messages from the client, which may be requests for information or remote procedure calls. Cybr exposes the DAW behavior via fifty-six parametric request handlers. Requests may ask cybr to create an empty session, add a new track to the session, insert audio files or MIDI clips into tracks, load a plugin, set a plugin parameter, start real-time playback, or one of many other possible actions. The Fluid Music client code has complementary functions – one for each of the fifty-six server-side message handlers. To play or render a `FluidSession`, the Fluid Music client will iterate over the contents of that session object and build a large composite message that instructs the server how to create the session object locally and build the audio graph required to play or render it.

Cybr is a layer of abstraction around tracktion_engine, and the Fluid Music client adds additional layers of abstraction around the cybr server. These multiple layers enable Fluid Music's client API to be lightweight and accessible, while offloading computationally expensive tasks like audio playback and rendering to the server.

4.4 The Framework

The architecture of the Express.js web framework directly inspired the Fluid Music framework. Like Express, Fluid Music exposes a coordinated collection of idioms and JavaScript components that aim to be useful for building libraries and applications. The strength of the Express framework is that it standardized the language that dependent libraries use to communicate with each other. As a result, many developers can contribute packages that work not just with the Express framework, but also with each other.

Specifically, Express defined standardized representations of HTTP request and response objects. It also defined a “middleware” API which standardizes the interface for creating extensions to the Express framework, enabling developers to write and share code that *processes* the HTTP request and response objects. This means that there are two different types of packages that depend on Express. First, developers can write and share middleware, which extend the capabilities of Express applications. Second, developers can write web applications that use the Express framework to connect middleware with custom content and logic, resulting in user-facing web experiences.

Fluid Music aims to accomplish a similar goal, but for computational sound design and music production. It defines standardized representations of the core DAW components introduced in section 3.3.1. It also defines the Technique API, which standardizes the interface for creating extensions to the Fluid Music language. As a result, there are distinct ways to use the Fluid Music framework. First, developers can create new Techniques that extend Fluid Music’s sonic vocabulary. Second Developers can create software applications for music production and sound design that expose Fluid Music Techniques to end users. Finally, producers and sound designers can use Fluid Music software applications to create music content.

The Fluid Music score API which is bundled with the Fluid Music client is a simple but powerful example of an application for composing with Fluid Music Techniques. In addition to reviewing the results of a Fluid Music user study, chapter 5 describes some music created with the score API and discusses my experience sharing code and Techniques with another composer to create a derivative composition that could only be achieved using the Fluid Music toolset.

5 Fluid Music Projects

Using the tools described in chapter 0, I created several compositions to test, evaluate, and iterate on the capabilities and the Fluid Music framework user experience. The first compositions I created are short and show that the software works as intended and that it is possible to write code that generates professional sounding results. These examples and some of the underlying Techniques are described in sections 5.1 and 5.2. I also wrote a 3-minute composition, described in section 5.3 that shows that the framework can be used to notate more elaborate music. Because the Fluid Music concept is about sharing and reusing, I also provided the Techniques I wrote for one of my compositions to composer Nikhil Singh, and he leveraged those techniques to create a derivative work. Singh's work with the Fluid Music framework is described in section 5.4.

5.1 The example repository: A pure Fluid Music example

The first composition I created with the Fluid Music system is published in a repository available at https://github.com/fluid-music/example. The composition is short – just 17 seconds long. However, it demonstrates several unique capabilities of the Fluid Music system, such as transitive dependency management and studio quality production without a DAW. An audio rendering of the result is available at https://web.media.mit.edu/~holbrow/project/fluid-music-examples/media/fluid-experiment.mp3.

5.1.1 Transitive Dependency Management

The composition source material can be found in three NPM packages that contain audio samples and Fluid Music Techniques:

- @fluid-music/kit is a small collection of drum samples that are useful for creating rhythmic patterns. The package exports a Technique Library that resembles a drum sampler preset.
- @fluid-music/rides is a collection of ride cymbal samples and techniques for playing them back.
- @fluid-music/g3rd is a small collection of dyads (two note chords) I recorded on a classical guitar and then packaged for Fluid Music. This package also includes some electronically manipulated versions of the same sounds.

Collections of audio samples packaged as ingredients for composition are a common part of contemporary musical production. These are often sold as "sample packs" and are available from many conventional sample marketplaces like splice.com or loopmasters.com. Samples packaged for Fluid Music have major differences from conventional sample packs. For example, the logic of how to play those samples can be bundled directly with the samples. The @fluid-music/kit package has two different Techniques for inserting tambourine samples. The package includes multiple different tambourine samples played with increasing performance intensity, from pianissimo to fortissimo. One of the tambourine Techniques bundled with the library has a configurable intensity parameter. Invoking this Technique will select and insert a different audio sample according to the intensity parameter of the Technique invocation. This resembles the “Velocity Layer” feature found in sampler plugins like Native Instrument’s Kontakt and MOTU Machfive. However, commercial samplers bundle sample playback logic (including velocity handling) within the sampler software and configure this sampler with a proprietary preset format that depends on the commercial sampler. As a result, the sampler must re-implement algorithms for pitch shifting, stretching, trimming, and fading samples, which are part of the core DAW functionality. Fluid Music sample packages eliminate this redundancy.

A second tambourine Technique uses the same samples, but adjusts them with trimming, gain, and careful fades so that they sound as if they were played with a similar performance intensity. However, this Technique inserts a random sample, so the Technique sounds slightly different each time it is invoked. Sample randomization can produce results that better resemble an acoustic performance. Sample randomization is also a common feature in commercial sampler plugins. In the @fluid-music/kit package, the sample selection logic is open-source and is bundled in the package with the samples instead of obfuscated in a closed-source sampler plugin or extended DAW component.

If a user wants to reuse the same underlying audio samples, but invoke them with different playback logic, that user can write their own logic and publish a new package which includes @fluid-music/kit as a dependency. The new package does not need to copy the underlying samples. If the @fluid-music/kit package is listed as a dependency, the NPM tooling neatly handles all dependency resolution. This kind of transitive dependency management is common in open-source software development but is new to music production.

5.1.2 DAW-less Composition

The repository shows how to use the Fluid Music framework without an external DAW. It features a real-time audio playback via the cybr server. It is set up so that the user can make a change to the underlying JavaScript based score, and when they save the JavaScript score file, cybr updates the underlying audio graph and plays the new audio content without skipping a beat. As a result, composers who are comfortable writing JavaScript can use a text editor in place of a DAW, editing the code, auditioning the sonic results, and making changes in real-time. The live-reload feature was written as a proof-of-concept and is not currently part of the Fluid Music client library. A full integrated environment for composition, playback, and editing is left to future work.

5.1.3 Example Production Techniques

To make the audio sound polished, I packaged the following music production techniques using the Fluid Music framework. A sub-bass synthesizer adds low end to the kick drum, giving it extra OOMPH! This music production technique originated in analog recording technologies and is pervasive in popular music today. The Technique works by layering a sine tone with each kick drum. The sine tone is somewhere in the lowest two octaves of perceptibility, approximately 20-60hz. The sine is typically tuned to the key of the song or the current bass note. Sub-bass can be positioned slightly after the kick drum onset so that it does not mask or interfere with kick onset. It is most effective when the sustain, decay, and frequency are tailored to the accompanying musical content. A lower, slower pitch may work better with a slow musical composition, while a higher, more staccato sound may work better with a faster musical composition. Sub-bass gain should be set just high enough so that it can be felt, but not so high that it is perceptually distinct from kick and bass timbres present in the mix. For this example composition, I tuned the sub-bass parameters by ear. Automatically adjusting the same parameters for different musical content could be a useful research project in future work.

Another production Technique employed in this composition, time-aligned reversed audio, also draws from pop music production. This technique can be heard on percussion in The Beatles' 1967 single Strawberry Fields, for example. The technique is still widely used, and the core Fluid Music client library includes a dedicated technique, `AFReverse`, which reverses an audio sample and positions a particular point within that sample (such as a transient) at a particular musical time in a composition. The @fluid-music/g3rd package exports several Technique Libraries, and some of these use the `AFReverse` Technique, which positions the samples so the timbral onset at the end of the sound aligns with a downbeat, and the reversed decay anticipates that downbeat. The example composition begins with a stretched and reversed sample from the @fluid-music/g3rd package that resolves with a kick drum on the first downbeat beginning at 3.636 seconds.

Two custom techniques invoked at the beginning of the piece insert stereo width automation. At the very beginning of the piece, the reversed guitar sample is in stereo. Over the first 2.7 seconds, the stereo image merges into mono. Then, on the first downbeat, the stereo image suddenly opens back to stereo, as if the force of the initial kick drum pushed open the mix. This effect is accomplished with two custom Techniques that are defined in the example score repository. One is a fade-to-mono Technique, and the other is a fade-to-stereo Technique. Both techniques insert two points in a stereo width automation track. The first is positioned according to the `UseContext`'s `startTimeSeconds` property, and the second technique is positioned according to the `UseContext`'s `durationSeconds` property. When using the Fluid Music score language, Techniques' start time and duration are aligned with the musical grid indicated in the score’s Rhythm Strings. As a result, the expanding and contracting stereo image in this example composition is precisely synchronized with the musical content.

One final touch is a subtle reverb which glues the mix together. This might not exactly be considered a Technique, because it does not use the Technique API, and in this case, a similar result could be achieved with a plugin preset. However, the kinds of Techniques described above are a significant part of what distinguishes produced music from the simple capture of a live performance. Having written them once with the Fluid Music API, it is much easier to reuse them in the future. This is especially true for things like stereo width automation, which can be very time-consuming to implement in a DAW but is easily invoked with a single Technique by using the Fluid Music language.

The custom Techniques for sub-bass, time-aligned reversed audio, and stereo width modulation described above encapsulate common music production processes for use with the Fluid Music framework. Now that these Techniques have been written, they can be reused in other compositions. Used with care, Techniques like these are a big part of what make produced music stand out and sound professional.

5.2 The example-trap repository

The two examples described in this section feature trap music production Techniques written for Fluid Music. Musicians and producers like Shawty Redd, T.I., and Gucci Mane devised this musical genre in the early 2000s. There are two short musical examples available at https://github.com/fluid-music/example-trap. Both examples share some identical techniques, showing how mutable musical ideas can be encapsulated with Fluid Music and reused in different compositions. Sections 5.2.1 covers some of the Techniques and applications used in the first example. It can be found in the file named `example-1.js` within the example-trap GitHub repository. Section 5.2.2 does the same for the second example, and it can be found in the file named `example-2.js` in the same GitHub repository. This second example builds on the content and techniques used in the first to create a longer and more complex composition.

Some of the Techniques used in both trap example compositions are found in the @fluid-music/tr-808 NPM package, which contains audio samples recorded from a Roland TR-808 drum machine (Warner 154). TR-808 bass drum samples play an important role in trap music. The original TR-808 was an analog drum synthesizer which used analog circuitry to produce drum sounds. Most of the sounds that the machine produces have parameters which adjust the drum timbre. The bass drum has a 'Decay' parameter and setting this to the maximum value yields a long, low, resonant sound that is a staple of bass in trap and other hip-hop subgenres. The original TR-808 was not tuned to a particular scale. The bass drum sound on the TR-808 has an adjustable 'tuning' parameter, but that parameter controls the pitch at the onset of the sound, and not the decay portion. Trap producers adjusted the bass drum parameter so that its dynamic envelope resembles the dynamic envelope of a bass guitar. By sampling the sustained TR-808 bass drum sound and adjusting the pitch of the sample, the bass drum could be used as a pitched bass instrument instead of an unpitched percussion instrument. The @fluid-music/tr-808 package includes an `AudioFile` Technique that is tuned for this purpose.

TR-808 drum machines are famously inconsistent. Two TR-808 machines can sound quite different from each other due to inconsistencies in the analog circuitry that synthesizes the drum sounds. The TR-808 machine that was recorded for these samples (and generously distributed without licensing restrictions via http://machines.hyperreal.org/) is about 22 cents sharp of G1. The fundamental frequency of a G1 pitch is approximately 49hz, assuming equal tempered tuning, and an A4 of 440hz. The @fluid-music/tr-808 package includes both a bass drum Technique with a long decay that was tuned down 22 cents to G1, and an unmodified bass drum Technique. The unmodified Technique is suited to unpitched percussion, while the tuned Technique can be used as a pitched bass instrument. The fundamental frequencies of both Techniques are in the sub-bass range, and their upper harmonics extend into the bass range. The tuned `AudioFile` Technique's `pitchSemitones` property can be used to pitch-shift the sample to other equal-tempered pitches, effectively creating the pitched bass sampler that is appropriate for Trap music production. I used this Technique for the bass sounds in both trap example compositions.

5.2.1 Trap Example #1

The first trap example is found in the `example-1.js` file which is contained in the fluid-music/example-trap GitHub repo. This file constructs a new Technique Library which extends the Technique Library exported by the @fluid-music/tr-808 package. The new Technique Library is stored in the `trapKit` variable in the code excerpt below. The `trapKit` Technique Library employs another music production Technique that is common in sample-based music. Micro-timing adjustments on percussion samples position those samples so that their transients do not exactly overlap with each other. These timing adjustments can be used to create a more pronounced rhythmic impact (Snoman) and to create new "rhythmic feels" (Danielsen).

```
const tr808 = require('@fluid-music/tr-808')
const kit = require('@fluid-music/kit')

const copy = fluid.techniques.AudioFile.copy
const trapKit = { ...tr808.kit }
const trapKick = copy(kit.tLibrary.D, {
  fadeOutSeconds: 0.2,  // cross fade with bass
  pitchSemitones: -1.59 // this just sounds good ¯\_(ツ)_/¯
})

// timing adjustments
```

```
const clap = trapKit.p
trapKit.B = new techniques.AFOnset(trapKick, 0.01)
trapKit.p = new techniques.AFOnset(clap, 0.03)
trapKit.o = copy(trapKit.h, { gainDb: -3.5 })       // offbeat hi-hat
trapKit.r = new techniques.AFReverse(clap, 0.03)    // reverse clap
```

This kind of timing adjustment is common enough so that the Fluid Music client library includes a dedicated Technique named `AFOnset`, which adjusts the timing of AudioFile Techniques. The `trapKit` Technique Library above uses the `AFOnset` technique from the Fluid Music client to slightly offset percussion samples from the musical grid.

Part of what gives trap music its sound is staccato hi-hat patterns in modulating rhythmic subdivisions which often switch between triplets and duplets. This trap example uses a combination of dedicated Techniques and the Fluid Music score language to notate trap-inspired hi-hat patterns. The following score object is an excerpt from score in `example-1.js`. It shows the trap-style hi-hat patterns heard in the composition. It also shows how the score language (described in section 4.2.3) can use Rhythm String in combination with Pattern Strings to create complex rhythmic patterns that shift between different triplet and duplet rhythmic subdivisions. Note that in the score example below, both '`o`' and '`h`' Techniques insert a TR-808 hi-hat sample; however, the '`o`' character has its gain reduced by 3.5 decibels, as indicated in the Technique Library created in the previous code example above.

```
const variations = [
  {
    r:   '1 + 2 +...3 + 4.e.+ ',
    hat: 'hohoh     hohohoh-ho'
  },
  {
    r:   '1 + 2 +...3 + 4 + ',
    hat: 'h h h h hohohohoho',
  },
  {
    r:   '1 e + a 2 e + a 3.....e...+..a.4 e + a ',
    hat: 'hohohohoh       hohohhohohohohoh o h o ',
  },
  {
    r:   '1 e + a 2 e + a..3.....e + a 4 e + a ',
    hat: 'h   h   h h hohoohoohooh o h o h hoho',
  },
]
```

The unabridged score is available at https://github.com/fluid-music/example-trap.

5.2.2 Trap Example #2

The second trap music example can be found in `example-2.js` file which is also contained in the fluid-music/example-trap GitHub repo. The second example builds on Techniques described in the previous section to create a longer and more elaborate composition. It introduces a new custom Technique called `StutterSoftOddEvents`, which can be parameterized with an integer and any sub-technique. When used, the `StutterSoftOddEvents` subdivides the event duration, and repeatedly invokes the sub-Technique that was passed into its constructor. Each time the sub-technique is invoked it is passed a modified Dynamic object via the `UseContext`, resulting in a dynamic rhythm that alternates between forte and mezzo-forte. This single technique can then be used to achieve a similar effect as the juxtaposed '`h`' and '`o`' hi-hat Techniques used in `example-1.js` (covered in section 5.2.1).

Tracks in a Fluid Music session may specify a default Technique Library. When inserting a score into a `FluidSession`, the `insertScore` method will find Techniques that are contained in a `FluidTrack`'s default Technique Library, allowing the score author to reference techniques in a Pattern String without explicitly specifying a Technique Library in the score. In the `example-2.js` score, the following Technique Library is set as the default for the tambourine track. This sets the default sonic vocabulary for the track, which includes Techniques that trigger single, double, triple, and quadruple tambourine hits.

```
const tambLibrary = {
  t: kit.tLibrary.t,
  2: new StutterSoftOddEvents(2, kit.tLibrary.t),
  3: new StutterSoftOddEvents(3, kit.tLibrary.t),
  4: new StutterSoftOddEvents(4, kit.tLibrary.t),
}
```

In the score excerpt below, the `tamb` track uses both single and double hits from the Technique Library above. This example illustrates one of the fundamental contributions of Fluid Music framework, which is abstracting the score language from the underlying DAW-based interpretation of the score. A score sequences Techniques on a musical timeline, and the Fluid Music framework interprets that score using those Techniques. Techniques can be organized into Technique Libraries, so that the framework can be used to curate Techniques that collectively define a musical style or genre.

```
const excerpt = {
  r:       '1 + 2 + 3 + 4 + ',//1 + 2 + 3 + 4 + '
  snare: ['    K       K   ', '    K       K   ', '    K       K   ', '    K       K   '],
  clap:  ['    p       p   ', '    p       p-r-', '    p       p   ', '    p       p-r-'],
  kick:  ['B  B         B B', 'B  B B   B B    ', 'B  B         B B', 'B  B B   B B    '],
  bass:  ['g-------        ', 'g-------        ', 'g-------        ', 'g-------        '],
  hat:   ['h               ', 'h               ', 'h               ', 'h               '],
  tamb:  ['t   t   t   t   ', 't   t   t   t   ', 't   t   t   t   ', 't   t  2t   t tt'],
  pads: {
    r: '1 + 2 + 3 + 4 + ',
    padA:  ['        ', '            ', '          ', 'd-      '],
    padB:  ['A-------', 'b-----------', 'a-b-c---  ', '        '],
    padC:  ['        ', 'B-----------', 'C---------', 'A---B-E-----E---'],
  }
}
```

The full score implements some other common production strategies. For example, it uses reverb and bus compression to ‘glue’ the mix together. It features an expanded set of pad synthesizers which are sidechain compressed with muted kick drums resulting in a subtle “pumping” effect found in many styles of electronic music.

5.3 Seven and Five

Traditionally, sound designers and producers use the tools in a DAW to create polished, professional musical content. The two Trap Examples from section 5.2 show how these kinds of results can be achieved with the Fluid Music framework. This section describes my efforts to create a longer, more sophisticated composition. When creating this composition with the Fluid Music framework, I intentionally avoided trying to reproduce any existing musical style.

The piece is named *Seven and Five*. The name refers to the repeating patterns of seven notes that are organized in groups of five, resulting in the unusual time signature of 35 over 32. The JavaScript score is available in the GitHub repository at https://github.com/fluid-music/example-seven-and-five.

Two JavaScript files at the root level of the repository implement the musical idea behind *Seven and Five* with a combination of custom Fluid Music Techniques and JavaScript functions: `techniques.js` and `components.js`. The verbosely named `makeArp6TLibraryFromMidiChords` function defined in `components.js` accepts an input chord and other parameters, computationally generates Techniques, and returns a Technique Library that contains these Techniques. These “Arp6” Techniques generate the melodic, harmonic, and rhythmic material that make up the core of *Seven and Five*.

Every time one of the computationally generated Arp6 techniques is invoked, it inserts up to 70 MIDI notes distributed over 10 differently configured DAW tracks. Each of the 10 different tracks contains a Podolski[20] synthesizer plugin. Each of the 10 plugins has a slightly different filter configuration, so MIDI notes on different tracks play slightly different timbres. The idea behind the Arp6 generator starts with an arpeggiated musical chord. The

[20] Podolski is a hybrid digital/virtual analog synthesizer plugin developed by U-He. It is available for free download. It is available for free via the developer’s website at https://u-he.com/products/podolski/.

`makeArp6TLibraryFromMidiChords` function accepts chords with any number of notes, but the chords used in this composition all have 7 notes. The generated Technique inserts the seven arpeggiated notes with decreasing velocity on one DAW track. Then a delayed copy of those arpeggiated notes is inserted on the next DAW track, and this process is repeated until the Technique has inserted four distinct "echoes" of the original arpeggio on the subsequent tracks, resulting in five repetitions of the seven-note pattern. Instead of each echo being a direct copy of the initial arpeggio, it is computationally mutated, so instead of hearing an exact echo, we hear a variation of the original arpeggiated melody. The timbre of each subsequent synthesizer has a slightly "darker" timbre, meaning it is configured with a slightly lower low-pass filter frequency. This entire process is then duplicated on another set of five tracks, this time with a different synthesizer preset.

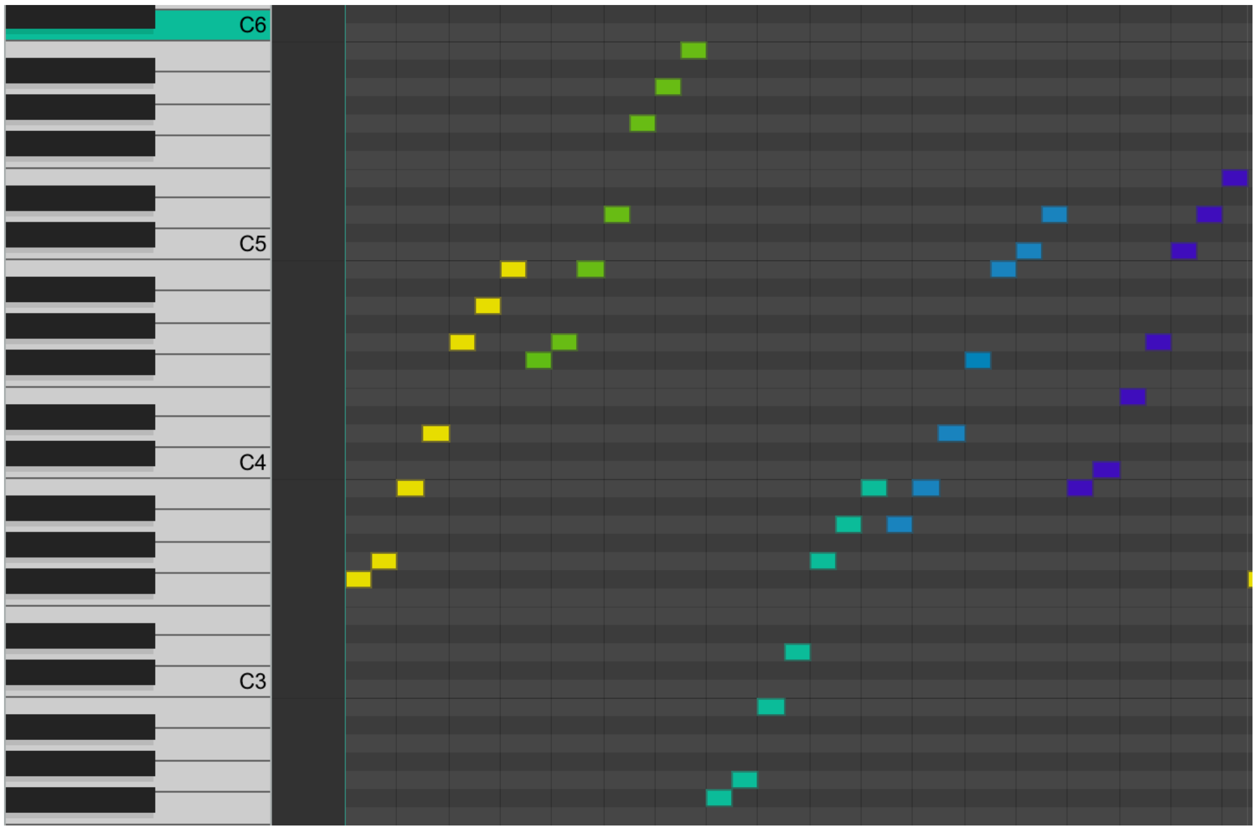


Figure 5.1 An Arp6 arpeggio with seven notes, and its five mutated "echoes" shown in Reaper's MIDI piano roll view. *This shows a basic mutation, which simply transposes the underlying scale degree of the notes in the initial chord. Each color represents a different track. A single invocation of one of the Arp6 Technique would insert this identical sequence on two separate tracks, resulting in 70 midi notes.*

Figure 5.1 shows a basic sequence of variations that a single Arp6 Technique inserts. This is the result of a basic parametrization of the `makeArp6TLibraryFromMidiChords` function which simply transposes the scale degree of the input chord. The function can also generate Techniques that mutate the input chord in other ways, for example, by adjusting the timing of the delayed variations and omitting and re-ordering notes. An example of a more elaborate

Technique is shown in Figure 5.2. The exact process by which delayed copies are mutated is best understood by reading the source code, beginning with the `makeArp6TLibraryFromMidiChords` entry point.

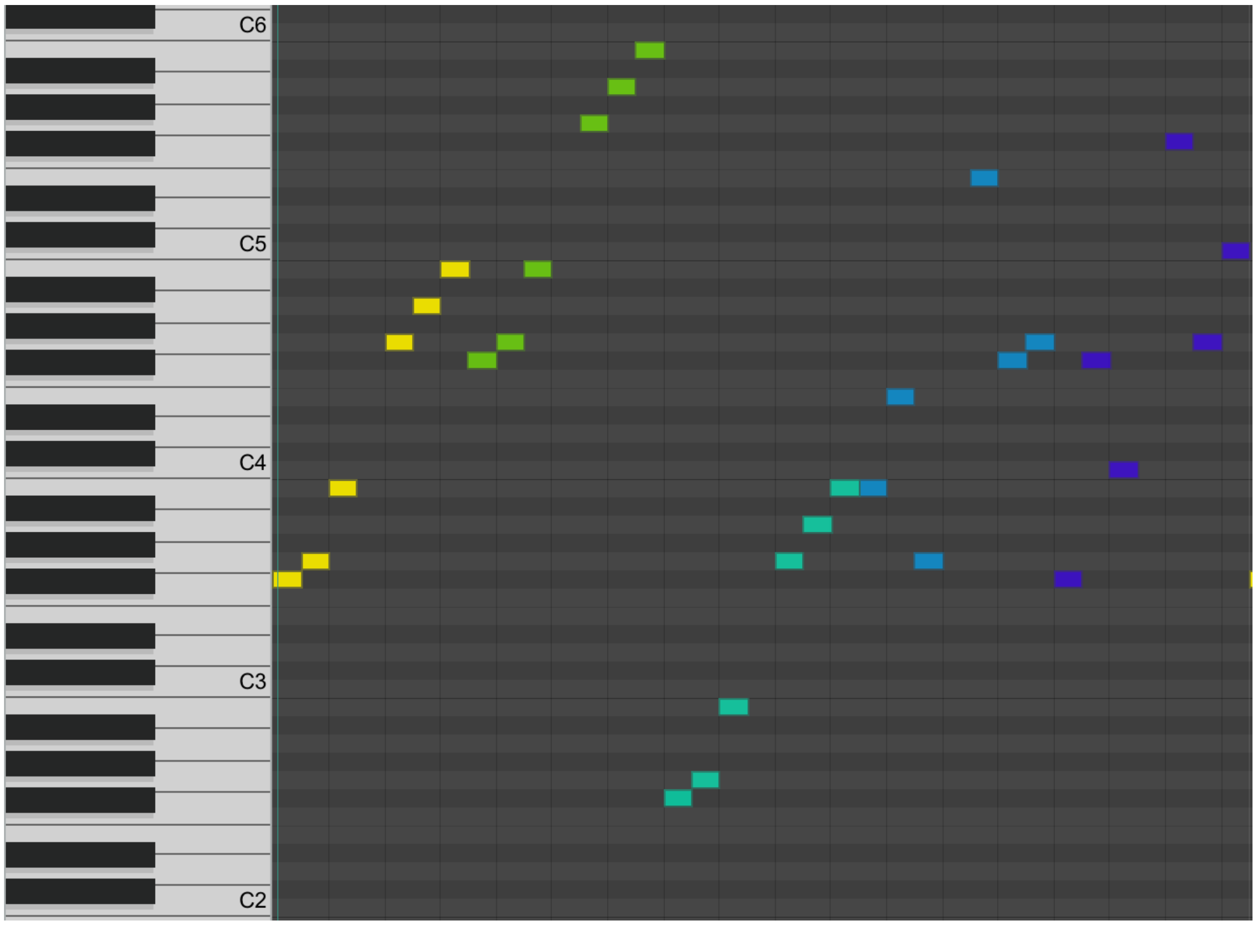


Figure 5.2 The output of a parameterized Arp6 Technique. This Technique was created by the makeArp6TLibraryFromMidiChords function.

Seven and Five builds around the harmonic and melodic mutations created with the approach described above. However, the full score includes a combination of explicitly notated layers, computational content, and custom embellishments. This composition demonstrates how the Fluid Music framework enables these different computational approaches to co-exist within a single score and how a range of different kinds of Techniques add small details and shape the high-level structure of a composition. Because the score can generate a Reaper session or a Tracktion Waveform session in addition to raw audio, the output can be further manipulated using all the tools available in a DAW.

While using the Fluid Music client API to compose *Seven and Five,* I encountered a limitation with the API, which led me to the addition of `FluidSession`'s `useTechnique` method in the Fluid Music client API. This limitation made it difficult to make changes to the composition at the structural level. The best compositions are interesting and creative at multiple different levels.

At the structural level, DAWs are useful for creating music with an intentional shape. Whether measuring time in minutes, verses, choruses, or movements, DAW users can sculpt the balance of musical tensions and resolutions along a musical timeline. At the same time, DAWs let the user zoom in to an individual phrase, note, or millisecond and carefully edit and shape sounds at these detailed levels. Computational tools for composition and sound design are rarely effective at many different levels. Computational compositions sometimes sound interesting at the phrase level, but sound bland or aimless at the structural level. Patch-based systems like Pure Data and Max have limited support for sequencing musical events on a timeline and do not have anywhere near the flexibility of audio or MIDI tracks in a DAW. Code is discrete by nature, so code-based scoring systems must define the musical resolution that individual units of code refer to. As a result, it can be difficult for a code-based score system to reach between units of resolution. Musical composition benefits from the ability to make precise edits from the microscale to the macroscale, and at musical units of time like beats, measures, and verses, to invariable units of time like minutes, seconds, and milliseconds.

Fluid Music provides several different ways to describe events on a musical timeline. These attempt to mitigate the detractors of discretizing musical time. The Rhythm String notation allows users to vary the temporal resolution within a sequence of events, as shown in the score examples in section 5.2.1. It also enables the user to adjust the temporal resolution of a discrete sequence of events from whole notes to thirty-second notes, and any integer subdivision thereof. The `Nudge` and `AFOffset` Techniques that are bundled with the Fluid Music client library provide a way to wrap around `AudioFile` and other Techniques, offsetting them from the musical grid by any number of seconds that can be represented with a floating-point value.

When I started writing *Seven and Five*, the only way to use a Technique in a composition was to put it into a score object, and then call `FluidSession`'s insertScore method. Parsing a score object requires a Rhythm String, a Pattern String, and a Technique Library. This puts the responsibility of ensuring that the score parser has access to all three components on the user. It is easy enough to create a score object that contains an "`r`" property, a "`tLibrary`" property, and a Rhythm String; however this is too much overhead when the only goal is to use one Technique at one specific time on a particular track. While composing *Seven and Five,* I found that in addition to the score language, which is useful for using many Techniques at once and applying them in with a musical rhythm, I wanted an easy way to apply a technique at a particular time. As a result, I added the `useTechnique` method to the `FluidSession` class. The method conveniently constructs a valid `UseContext` object and passes it into the Technique's `use` method. It also automatically populates `UseContext` properties that are not specified by the user, further reducing the overhead associated with explicitly invoking a single Technique.

In *Seven and Five*, `FluidSession`'s `useTechnique` method is used to insert plugin parameter automation points at specific times in the composition. Long, gradual parameter changes created by automating the oscillator filter and sync parameters help to give the composition its long-term structure by continuously modulating the synthesizer timbres that provide the main melodic and harmonic content of the piece

5.4 Nikhil Singh's Derived Composition

The main goal of Fluid Music is to enable producers, sound designers, and composers to share and reuse music production Techniques. How do we know if the framework is effective for sharing music production processes? To explore the potential for sharing music production processes with Fluid Music, I shared the source code I wrote for *Seven and Five* with composer Nikhil Singh, who kindly offered to use it to produce a derivative composition that would borrow music ideas and material contained in the source code, while modifying them to create a new and fundamentally different composition. The source code I shared with Singh includes custom Techniques I wrote for the composition, references to Techniques that are available via NPM, custom JavaScript routines, and the score itself. Importantly, sharing this source code is different from sharing a DAW session. The source code describes step-by-step how to create a session imperatively, where a DAW session describes only a session's state. As a result, Singh was able to make deep structural changes to the composition that extend beyond what would be possible with traditional collaborative remix workflows. His composition keeps the same harmonic material but mutates the existing timbres and adds new ones. His score also converts the 35/32 time signature to 4/4, a change that would be difficult or impossible to do without the Fluid Music framework. Nikhil's full score is available on GitHub via https://github.com/nikhilsinghmus/fluid-demotrapremix. The audio is linked from https://web.media.mit.edu/~holbrow/project/fluid-music-examples/.

5.4.1 Visual Studio Code Extension

While writing the score, Singh found that he really valued the ability to control session content computationally with the JavaScript language. In some ways, he found this preferable to using the GUI options in a DAW. However, he also found reading and writing Pattern Strings unintuitive compared to reading and writing sheet music. He observed that with sheet music, you can insert a symbol (such as a whole note rest) which represents a disproportionately long pause between the events that surround it. The Fluid Music score language also makes this possible; for example in the "`.a.b`" Pattern Strings matched with a "`w123`" Rhythm String, the first period signifies a whole note rest while the period signifies quarter note rests. However, this splits the meaning across two different strings, and in the words of Singh, makes it difficult to "feel the rhythm by looking at the string."

Singh found that he wanted to write the outline of Rhythm and Pattern Strings at a low rhythmic resolution such as quarter notes or eighth notes, but then visualize and refine them at fine-grained, faster rhythmic resolutions. His initial solution was to extend the JavaScript's String prototype to add the `halfTime()` method, which automatically processes a Pattern String, returning a new Pattern String that contains the same content, but plays that content at half the original speed. This allowed Singh to write Pattern Strings at coarser rhythmic resolutions than the matching Rhythm String, but it still had some limitations. An ideal system would allow the user to write a pattern string at a coarse rhythmic resolution, and then make edits at a higher resolution. Traditional DAWs have a zoom feature that lets the user edit and view events at different temporal resolutions, but this is not practical with the score language, even when using the `halfTime()` string method.

To address this problem, Nikhil chose to write a Visual Studio Code extension that lets the user write a Pattern String at a coarse temporal resolution, and then edit that pattern string at a higher temporal resolution. This extension is available under the name "fluid-scoretools" in the Visual Studio Code Extension Marketplace. It adds a minimal DSL to the Visual Studio IDE for manipulating Pattern Strings. Specifically, if the user can enter below, the "`halfTime:`" command invokes Singh's extension.

```
const hat = "h h h h" :halfTime:
```

In the example above, the ":`halfTime:`" code acts as an operator which opens a popup hint next to the line of code in the text editor. If the user presses the return key on the keyboard while the popup is active, the extension modifies the previous Pattern String, stretching it out so that it lasts twice as long, without otherwise changing its contents. This process results in the code shown in the example below.

```
const hat = "h-  h-  h-  h-  "
```

This process allows the user to write a Pattern String in a coarse temporal resolution, and then edit it in a finer resolution. The ":`halfTime:`" function can be applied as many times as needed to achieve the desired zoom level.

In addition to the "`:halfTime:`" code mutator, the "fluid-scoretools" extension provides a framework for creating other code generators specifically for the Fluid Music score language. It also includes a tool for computationally producing procedural rhythms in the Fluid Music score language. This tool can be invoked with the "`:dsbsd <size> <patternKey>:`" string. "dsbsd" stands for "Discounted Stochastic Binary Subdivision", and is Singh's adapted version of the stochastic binary subdivision technique for generative composition originally introduced by Langston. Singh's approach of generating Fluid Music score components directly in the text editor provides some of the advantages of a DSL without sacrificing any of the features of working with JavaScript.

Singh was able to use the tools provided by the Fluid Music framework to create a custom interface for composing with Fluid Music Techniques. He tailored the interface to his preferences while maintaining compatibility with the larger ecosystem of Fluid Music Techniques. In the long term, this is the process that will help to find the answers to the design questions outlined in section 4.1. The framework enables many Fluid Music users to create their own interfaces for working with Fluid Music Techniques. Each user-created interface is another experiment with how Fluid Music interfaces should be similar or different from existing tools for music production. The collective efforts of open-source developers borrowing and building on each other's best ideas is how Fluid Music can reach beyond a niche audience and be useful to musicians and producers of all kinds.

6 Workshop, User Study, and Evaluation

To gather feedback about the Fluid Music framework, I hosted a two-part workshop for interested sound designers, composers, and software engineers that is described in section 6.1. Attendees who completed the workshop series were then invited to participate in a qualitative user study. Section 6.2 provides an overview of the user study. Section 6.3 presents high-level analysis of the user study results, while section 6.4 goes into more detail about how users responded to specific questions and topics. Finally, section 6.5 discusses limitations of the research method.

6.1 Workshop

In January of 2021, I conducted a two-part virtual Fluid Music workshop. The workshop was designed to familiarize users with the Fluid Music framework so that I could solicit feedback, corroborate the efficacy of my design choices, and course-correct if necessary. To advertise the workshop, I built a web page with information about the Fluid Music framework and the workshop itself, which can be found at https://web.media.mit.edu/~holbrow/project/fluid-music-beta/. I recruited workshop participants by sending this web page to my network and asking for recipients to forward the request to anyone they thought might be interested in participating. A total of 20 potential participants filled out the signup survey, 14 users participated in the first workshop, and 12 completed both the first and the second parts of the workshop. Of these, 10 users volunteered to be interviewed for the user study.

The workshop was recommended for users who were comfortable with Node.js and npm, and who had at least a little familiarity with music production. The web page also included a link to a signup form, which asked participants to self-report their comfort level with DAWs. Of the 10 users who participated in the user study, six selected “I have used DAWs extensively (ex. in an academic or professional setting)”, one selected “I have used a DAW in the past, but not frequently (or I am out of practice),” and the remainder left the question unanswered.

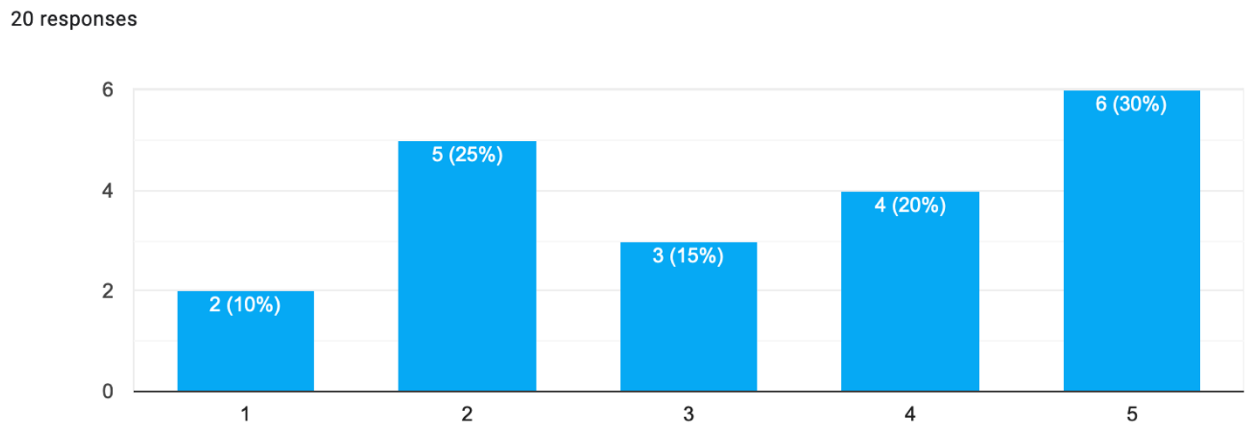


Figure 6.1 Results of the workshop signup question "What is your level of comfort when using JavaScript, Node, and npm? You need to know JavaScript basics to participate. However, there is no 'best' answer to this question; we would love to have participants from a range of different backgrounds." In the signup form, a response of "1" indicated "I can get by with a little help from Google/StackOverflow" and "5" indicated "I am very comfortable with the JavaScript/Node ecosystem."

I asked participants to complete the steps in an introductory tutorial prior to attending the first workshop. This tutorial was linked from the web page about the workshop. Following the steps in the guide enabled participants to begin the workshop with all the prerequisite software installed. It also gave me a chance to troubleshoot potential installation issues before the workshop began. Fortunately, setup issues were minimal, and all participants were able to begin the workshop with Node.js, NPM, the cybr server, the Reaper DAW, and the Podolski VST synthesizer plugin installed. The introductory tutorial also led users through the process of creating a simple `FluidSession`, rendering that session to audio and exporting it to a .RPP file that can be opened in Reaper. These steps ensured that all the major parts of the Fluid Music framework were working together as intended on participants' computers.

Both workshops were held via the Zoom video conferencing service, which enabled me to use a combination of screen sharing and live coding to show how the Fluid Music client library works, write example code, and answer questions. In the first workshop, we began where the introductory tutorial finished: with a JavaScript file that constructs a `FluidSession` instance and inserts a score object into the session. This example showed how to create a session, but it was not clear in the example how scores work, or what the characters in score Pattern Strings are referring to.

In the workshop, we covered the relationship between score objects, Techniques, and Technique Libraries, and walked through an example of creating a custom Technique. I built a minimal score example, added a breakpoint inside our custom Technique's `use` function, and used Visual Studio Code's debugging tools to pause code execution during the Technique invocation. This enabled us to visualize the call stack, wherein the `insertScore` method invokes the score parser, a function that calls the `use` method for each technique referenced in the score, passing in a tailored `UseContext` object. The debugger built into Visual Studio Code enabled us to interactively inspect the `UseContext` object passed to our custom Technique's `use` method. We also reviewed the `afactory` and `transcribe-chords` command line utilities from the @fluid-music/utils NPM package, which are useful for automatically generating `AudioFile` and `MidiChord` Techniques, respectively. Since all the examples we had seen previously used very simple scores that were intended to be demos and did not show how the framework could be used to create more complex music, we then inspected a slightly more complex score from the example repository available at https://github.com/fluid-music/example. We concluded with a discussion of different kinds of projects and tools that could be built with Fluid Music.

For most of the second part of the workshop, I answered questions that came up following our discussions on the previous day and proceeded slowly through a coding example so users could duplicate the coding process locally. I conducted both parts of the workshop twice, so users who could only come on the weekend could attend, as could users who could only come during the week. As a result, the exact content of the second part of the workshop was slightly different between the two instances. In one instance we listened to some popular music by Skrillex and Miley Cyrus, and I showed how production techniques used on those recordings can be re-created with custom Fluid Music Techniques. In another iteration we also took a more detailed look at the Fluid Score language, exploring how nested objects can be used to create complex musical scores, and how child objects can override or extend Rhythm Strings or Technique Libraries in parent objects. In both instances, we discussed different ways to use the Fluid Music framework for different kinds of projects and interfaces.

6.2 User Study Overview

The workshop signup form asked participants if they would be willing to be interviewed as part of a user study following the workshop. A total of 10 participants volunteered to be interviewed for this study and, after the workshops concluded, I conducted a semi-structured interview with each of them. During the interviews, I solicited a variety of feedback about Fluid Music, including both the Fluid Music framework and the concept of Fluid Music itself.

The MIT Committee on the Use of Humans as Experimental Subjects (COUHES) process reviewed the study design, and determined the study was exempt from a full review process, because the study design met the criteria for an exempt category of research called "Educational Testing, Surveys, Interviews or Observation."

One of the main goals of the study was to evaluate whether users felt like they could employ the Fluid Music framework to effectively describe musical ideas and production techniques. The Fluid Music framework, covered in chapter 0, provides the infrastructure required to experiment and study new collaborative workflows. Before Fluid Music can be widely used by musicians, it will need a graphical user interface for working with user-generated Techniques.

However, in its current state, the Fluid Music framework provides the capabilities required to begin experimenting with new kinds of collaborative workflows like the one described in section 5.4. The example in section 5.4 offers anecdotal evidence that the framework can be used for a new kind of musical collaboration. The user study, however, aimed to ask more generalizable questions about the framework, such as: What kinds of Techniques and musical ideas can be effectively captured with the Fluid Music language, and what kinds cannot? In what ways can the tools be improved?

I also solicited feedback about the Fluid Music concept. By asking users to discuss what kinds of musical ideas they are interested in sharing, I evaluated whether users would be willing to share their techniques with others, and whether they were interested in using techniques created by others in their own compositions. The Fluid Music concept, described in section 3.1, connects massive asynchronous collaboration with the core components of a DAW. The Fluid Music framework is built to enable radically collaborative music production that challenges traditional cultural attitudes about composition, which, since the sixteenth century, have attributed economic and moral ownership of a composition to the sole author of the score (Théberge, *Any Sound You Can Imagine: Making Music/Consuming Technology* 181). This step toward collaborative composition, sound design, and production also represents a step away from conventional notions of musical ownership. If musicians, composers, and sound designers are not interested in sharing or reusing musical content, Fluid Music is less likely to be widely adopted. For this reason, the user study also aimed to solicit responses to questions about sharing content, for example: How do composers and creators react to the prospect of sharing their composition and production Techniques, and how do they react to the prospect of using Techniques created by others?

6.2.1 User Study Process

I began each interview by asking the interviewee their age, and any identities or occupations related to software engineering and music including, but not limited to, composer, producer, audio engineer, student, or software engineer. Prior to the interviews I prepared a template which included the list of interview questions in Appendix A. The questions inquired about users about their relationship with music and technology, their creative process, and prompted them to react to the Fluid Music concept and the Fluid Music framework based on their experiences in the workshop. For the remainder of the interview, we followed this question structure. We took detours where interesting and relevant topics arose. I also occasionally skipped questions to stay within time constraints.

Interviewees were between the ages of 21 and 36, and all had either some professional or academic experience working with music, software engineering, or both. All interviewees had a musical background, either professional, academic, or hobbyist. The interviewees were evenly split between people who were directly interested and involved in creating music and media for consumption, and those who were tool developers or more engaged with music as a process than as a product. However, there was not always a clear distinction between these two categories. The users in the study were all familiar with one or more existing languages for computational sound design and learned to use the Fluid Music framework before the study.

When designing the study, I chose not to collect identifying information, so I refer to users by pseudonyms to preserve their anonymity. Quotes in this chapter have been lightly edited for clarity.

All interviews took place through video conference software. I recorded the interview audio, but not the video. Most interviews took between 35 and 75 minutes. Some of the questions from the list above inquired about the interviewee's process for creating music and media. In cases where users had several different production processes for different kinds of media production, we chose to focus on just one. For example, one interviewee was an industrial designer, artist, and experimental composer. When discussing that user's creative process, we chose to focus exclusively on their music composition process.

6.3 User Study Themes

Following the interviews, I listened to the recordings and took notes, extracting information, sentiments, and quotes. I analyzed the notes by looking for patterns and emergent themes across the question categories and holistically. The subsections below present the results of this analysis organized by theme.

6.3.1 Initial reactions to Fluid Music

Initial reactions to the Fluid Music framework were largely effusive, with many users calling out plugin and DAW integration as especially desirable and drawing comparisons with other tools for computational sound design. Users consistently recognized that Fluid Music exposes a different set of tools than existing languages for computational sound design. Users were enthusiastic about the computational access to the DAW components afforded by Fluid Music, especially when that access is coupled with the ability to inspect and edit computed output in a professional DAW. Identifying the core DAW components described in section 3.3.1 was essential to designing the Fluid Music client API that makes this possible.

Users came from a wide variety of different media professions, and had very different processes for creating tools, media, and music. Despite their very different production processes, users consistently believed that those processes could be described with the Fluid Music API. This is perhaps unsurprising, because music producers are familiar with DAWs, and Fluid Music exposes core DAW components through its client API. The ability to access and manipulate DAW primitives with an API was consistently welcomed by users with enthusiasm.

Sean, an electronic music producer and software engineer, said they saw potential in Fluid Music: "I've explored a lot of different electronic, kind of computer music combinations of sorts, and the only other time that I felt this was with CSound… that was the only other time I thought 'okay this is useful, this isn't just a little trick for programmers to make music with.' If applied the correct way, things could happen with this." Alex, a composer of music for film and video games, was excited about the prospect of being able to create musical layers for interactive content using DAW paradigms. Alex did not feel that existing tools for computational sound design were very effective, saying "I've tried several others – WISE is still the best right now, but Fluid Music – we could do pretty cool stuff with that." Sean and Alex both expressed apprehension about Reaper, saying they would prefer to work with Ableton Live. Ben, a drummer and web engineer said "I really liked how you wrapped up a lot of different features and functionalities in one place – all of the connective tissue is there. It is well organized and well structured." Pat, a musician who frequently uses Pure Data, said they were excited about procedural session generation, saying "that to me is really, really amazing… I haven't seen that anywhere else." Pat, who was just learning JavaScript, also felt that their skill with the language was not as strong as they would have liked, and that made it difficult to take full advantage of the framework. This sentiment was shared by Sam, a User Experience Designer who is experienced with the KSP scripting DSL, but new to JavaScript. Sam also said they thought the DAW support was "valuable," and, commenting about Fluid Music generally, added, "It really made my imagination roam free with 'Where could this go?'"

Jean, a software engineer and musician, observed that the score language was difficult to read. An individual character in a score can indicate different things depending on the Technique Libraries in the score and in the `FluidSession`. When writing a score, the composer will both choose the Technique Libraries and write the Pattern Strings that make up a score, easing the process of identifying the available keys, and inserting them into the score. However, when reading a score, the reader must know how Technique Libraries are used for Technique resolution and must successfully navigate the score code base to identify exactly which Techniques are invoked by which keys. Jean had worked on developing a computational sound design framework at their last industry job and said the engineering team found it difficult striking a balance between being accessible to musicians and useful to engineers.

Other interviewees also observed that in its current state, Fluid Music has a narrow potential user base, which excludes creators who do not wish to write code. While the code-based approach does limit Fluid Music's potential audience, several users also expressed excitement about the ability to organize and manipulate musical structure with code. Casey, a musician and composer with experience in classical, jazz, and electronic music, said Fluid Music "gives me a way to iterate quickly." Casey was excited to use code "in a way that is hierarchical and expressive... so I could change the number of repeats by just changing one number rather than dragging things around [in a DAW]."

Multiple users also expressed interest in using Fluid Music to mix and match different Techniques as a strategy for creating "happy accidents" that could serve as source material for new compositions. Some users brought this up directly, while other users alluded to the idea. One user, Jean, was excited by the prospects of combining different techniques to create new content but was also unsure it would be possible to make Techniques that were flexible enough to be used effectively in different contexts. Sam expressed a more optimistic position, saying: "You could get these different perspectives and combine them in these really interesting ways."

6.3.2 Iterative Exploration

Many users described an iterative music and media production process wherein some source content serves as inspiration for variations and production of complementary new content. That new content then becomes a new source of inspiration. Users expressed this theme with varying language, and one interviewee used the phrase "iterative exploration" to describe the process. Interviewees described many different approaches to such iterative exploration. The initial input for inspiration could be a sound in the user's mind, a specific element from an existing composition known to the user, or even just an idea for an algorithm. Often, users valued technology for its ability to inspire by creating results that the user might not independently imagine or create. Generally, the Fluid Music users I spoke with were excited by the prospect of using Fluid Music inside of the iterative feedback loop for creating content.

6.3.3 Combinatorial Creativity

Can music be created with reusable musical objects that can be combined in different ways? It is a compelling idea with a long history. I designed the Fluid Music score language so that it is possible to write a musical rhythm but change the set of Techniques that are invoked in that musical rhythm. I did not mention this design during the workshop or the interviews. Still, multiple users talked about different ways that Fluid Music could be used to experiment with different combinations of Techniques. Sam correctly hypothesized that the Framework was designed to make this possible, saying:

> *Fluid Music reminds me of ethnomusicology... It seems like its own world where you could get these different perspectives and combine them in really interesting ways... I get the impression that Fluid Music has an intention of being able to do that, which is really cool to me.*

Ideally, each new Technique that is published adds another potential sonic ingredient that can be used together with all existing Techniques. In this ideal scenario, a linear increase in the number of published Techniques results in an exponential increase in the number of possible permutations of Technique combinations. I use the phrase “combinatorial creativity” to describe this possibility.

In its current stage, Fluid Music does not show if combinatorial creativity is a viable path to creating compelling new musical content. However, the Fluid Music framework does provide the necessary tools for us to begin to experiment with combinatorial creativity that uses core DAW components as malleable ingredients.

6.3.4 Enthusiasm for DAW and Plugin Integration

Fluid Music users consistently expressed enthusiasm for DAW and plugin integration in the Framework. For example, Pat said, "That's something that I'm genuinely stoked on!"

It might seem surprising that no existing language for computational sound design comes with similar support for DAWs. Popular existing tools and libraries for computational sound design do support limited interaction with DAWs and plugins. For example, CSound is designed to be extensible, and integrates with the Blue software environment (*Blue - A Music Composition Environment for Csound*), which offers many DAW-like features. Popular languages like CSound, SuperCollider, and Max can host VST plugins. The DawDreamer and RenderMan Python libraries are also designed for hosting VST plugins, although none of these options support de-normalized parameters or syntax completion when configuring plugin parameters in an integrated development environment.

I suspect that the enthusiasm that users expressed for DAW and plugin support in the Fluid Music framework derives from the intersection of all the design goals enumerated in section 4.1 – not just the DAW and plugin support. By focusing on the core DAW components from the beginning, Fluid Music ensures that when you export a `FluidSession` instance to an external DAW like Reaper, the session is malleable in the context of that external audio workstation. As a result, it is possible to create a session computationally, and then edit that session using the high-level tools available in a professional audio workstation. In any case, users responded favorably to these features, and I am optimistic that DAW and plugin integration will be key distinguishing features of the Fluid Music framework.

6.3.5 The Client API

The Fluid Music client API aims to be welcoming to developers from a wide range of backgrounds and experience levels. The workshop and subsequent user study included both beginners and very experienced programmers, so it was also an opportunity to find out who feels capable of using the API. The two users who were just beginning to learn JavaScript understood the client basics, and could follow along with my coding examples, but also felt that they needed to strengthen their JavaScript abilities to really take advantage of the framework. I would like the Fluid Music framework to be able to serve as a gateway to programming for musicians, so making the framework and documentation more accessible to beginners will be a helpful next step.

Responses to the score language were varied. The two interviewees who were experienced with live coding musical performances had different reactions. Kerry saw how the language was useful, but preferred to use algorithms that generated rhythms, and said they would probably not want to specify rhythms explicitly using the score language. Meanwhile, Joran found the score language very welcoming, *because* of the way it was explicit. Jean and Alex said they would be more inclined to use a GUI based environment for composition and felt that DAWs were probably better suited for the task. At the same time, Casey appreciated the way that the score language enabled them to avoid a GUI altogether. My takeaway is that the score language is useful for some users and purposes, and probably is not the ideal interface for creating Fluid Music projects in the long term.

A DAW GUI has many hidden menus, interfaces, and options which are buried in context menus and detachable user interface panels. The number of hidden parameters makes it difficult or impossible to open a session in a DAW and understand how it was created. I hoped that the Fluid Music score language would alleviate this issue. However, after running the workshop and user studies, I concede that while the Score language can be a practical way to *write* a composition, it is not a practical way to *read and understand* a composition for the reasons identified in 6.3.1.

However, the Technique API which is central to the Techniques development process proved to be successful. The structure of a Technique and the role of the Technique `use` method and the `UseContext` object are essential for users who write their own Techniques that extend the overall sonic vocabulary of Fluid Music. Users consistently understood how the Technique API works and the complex role of Techniques and the `UseContext` object.

6.3.6 Sharing Techniques

Questions 5 and 6, about reusing Techniques and Publishing Techniques respectively, were posed in the context of the Fluid Music framework but are central to the Fluid Music concept. If sound designers are not interested in sharing their production Techniques, Fluid Music will not work as it is currently imagined. Fortunately, Fluid Music users were consistently enthusiastic both about publishing their Techniques with others, and using Techniques published by others in their own projects and compositions. All users who participated in the workshop and user study had at least a little background in both music production and software engineering, so it was an ideal group to study Technique publishing. Fluid Music is most likely to be widely adopted if there are many useful, high-quality Techniques available to users. It is encouraging that the users I interviewed consistently expressed interest in publishing and sharing their Techniques.

Some users expressed more hesitancy about sharing musical notes, samples, or chord progressions, than about sharing processing, synthesis, or production Techniques. This is perhaps not surprising, when considering that for centuries a musical score has been associated with musical ownership. Traditional scores were not designed for notating studio production techniques. Fluid Music takes a clear stance, treating all core DAW components as valid source material for musical scores. This stance is one of two fundamental design decisions that sets Fluid Music apart from other tools and languages for computational sound design.

The second design decision that distinguishes Fluid Music is the focus on sharing and reusing techniques. Because Fluid Music scores are inherently digital, the score language can take advantage of the collaborative affordances of the internet. Specifically, Fluid Music appropriates the distributed asynchronous collaborative capabilities that are only made possible by the internet. Wikipedia is a good example of distributed asynchronous collaboration. Package managers like NPM also exemplify distributed asynchronous collaboration and add package versioning and transitive dependency management. Versioning and dependency management are both important for maintaining projects with complex dependency graphs like Fluid Music scores. In its current form, Fluid Music uses NPM infrastructure for versioning Techniques and tracking and resolving Technique dependencies. In the future, Fluid Music may benefit from adopting a dedicated package manager. However, NPM provides a robust standard which guarantees a path to backwards compatibility if NPM cannot support the long-term needs of Fluid Music.

The next stage for Fluid Music is to develop an interface that lets musicians *use* custom techniques without writing code. While developing this interface, care should be taken to ensure that it is useful to practicing composers, especially because the opinions of users who make their living by writing music are underrepresented in this user study. With the Fluid Music client API approaching version 1.0.0, the process of developing an accessible user interface should happen in parallel with the research questions outlined in section 4.1. Some possible ways to proceed are outlined in chapter 7.

6.4 User Study Questions and Responses.

While analyzing the data, I organized the notes in a spreadsheet based on the question template in Appendix A. This helped with the analysis, because reading the spreadsheet horizontally summarizes the responses of a single user. Reading the spreadsheet vertically shows how all interviewees responded to a particular question or category of questions. The subsequent sections present a vertical reading of the spreadsheet. Each subsection below goes into more detail about how users responded to a particular question or group of questions.

6.4.1 Preferred Tools

Ableton Live was the most popular DAW, followed by Logic. All interviewees had experience with at least one computational sound design language. There were many different computational sound design languages represented, including Max, CSound, Pure Data, Tidal, Orca, Tone.js, Gibber, WISE, and the Native Instruments KSP scripting language. There were also many different applications of these technologies represented, including live coding (creating music by writing code as a live performance), music composition for film, art installations, interactive music for video games, electronic music production, experimental instrument, and sequencer design, writing pop songs, and tool development for other sound designers. One user described working on specialized audio feedback tools for use in physical therapy.

6.4.2 Finished and unfinished music

When music was primarily released on physical media, it had to be finished or finalized before it could be released. Now that music is less tied to physical media, musicians and producers can easily distribute content that is just a draft, and then change, polish, or abandon it. My curiosity about how workshop participants thought about releasing music and media in this context was the motivation for question 1b from the list in Appendix A, "When you release something, is it finished? Or is it just in its current state?" Users who created music for consumption, were mostly, but not entirely, apprehensive to call that music finished. Casey, a composer and audio researcher, said of music they released that it was not finished, but "abandoned." Casey said that they would often go back and rework or reuse old compositions to create entirely new compositions. Jessie, an artist, engineer, and graduate student, said "At the time that I release it, it is in its current state. It's not... never really finished… at least it never really feels finished." When users did say that the content they released was finished, the notion of "finished" usually came with qualifications. Ben, who creates generative artwork and identifies as a hobbyist musician, said they did consider released content as finished, but with the stipulation that it was meant to be "a proof of concept." Jean, a software engineer who worked in digital media and liked to write pop music, said that when they released music, it was finished because the music was intended to be a "demo," rather than a polished final product.

6.4.3 Processes and Fluid Music

Question 3 is a multi-part question which inquires about the user's creative process, asks if that process could be described with the Fluid Music language, and then asks if the user *desires* to describe the process with computation. Collectively, users described a diverse range of processes for creating music and for creating tools for music production.

When composing, Casey described starting with some source material, which could be an audio recording, a plugin or synthesizer, or a "synthesized sound in my mind I wanted to create," and then using that source material as “a central theme around which everything else is constructed.” Casey avoids thinking about melodic or harmonic ideas, using the sonic source material to create “a really specific texture” and then “the goal is to try to find the things that do that, and find the things that complement that.”

While users described very different compositional approaches, Casey’s answer embodied a common pattern in users who were creating musical content. In this pattern, the composer listens to some inspirational material, reacts to it by making changes to the score, and iteratively repeats this process, listening for inspiration, and then mutating the content in different ways. Pat described a similar process as “iterative exploration,” saying it was important that it happen in an environment that he was comfortable enough with to “be improvisatory,” implying that if the technology involved was unfamiliar, it impeded the creative process.

Kerry is an accomplished musician whose musical performances are based on live coding. They prefer making music with computers because it allows them to sit and “absorb the sound,” where traditional live musical performance requires the musician to continuously interact with their instrument to produce sound. For Kerry, “Making music with computers has been about having the time to sit and listen.” When composing, Kerry likes to start with a rhythm, and then writes algorithms for processing, stretching, manipulating, and chopping that rhythm. Kerry said that using code allows them to manage variations of a musical fragment that maintained an explicitly documented relationship to the original material, noting, “I don't think that any of these traditional DAW systems let you do that, which is one of the things that attracted me to Fluid Music.”

Jordan uses a similar approach to Kerry for live coding. Jordan calls out two distinct phases in the process. The first is an exploratory phase where Jordan, either alone or together with another co-composer, develops algorithms that make rhythmic loops often using Pure Data or the Tidal Cycles DSL. Once Jordan finds an interesting process, they either memorize how it works, or save a file with the relevant code. Either way, that process can be recalled during the second performance-oriented phase.

Ben, a drummer and programmer who creates procedurally generated audio and video content, values the complexity that can be achieved when creating with code. Ben usually starts a project with a procedural rhythm that generates sound or animations. They said:

> *I have an idea for an algorithm – this could generate a two or three-dimensional sequence of sorts that could be rendered visually. What I enjoy about the generative stuff is that I don't actually know what's going to come out, and the complexity of it is more than I would have been able to imagine.*

For Ben, creating with code was specifically tied to the process of iterative exploration, saying "rather than starting with a composition in my mind and working backwards, it's more of a case of iteration and experimentation with the algorithm."

Sean described two different approaches. For the first approach, Sean records a 20 to 30 minute musical improvisation, listens to the recording, extracts the most interesting parts, and then uses those excerpts as a foundation for new compositions. In the second approach, Sean adapts a commercial music production process for personal use. In Sean's words:

> *Figure out what the client wants. Figure out their favorite song. Figure out their least favorite song. Talk them into laying out exactly what they want and then make it. I use that same approach with myself. I listen to a bunch of my favorite pop music and favorite electronic music, and I say it's going to be 40% that song by Dua Lipa, 30% that song by Excision, and then 30% that song by Deadmau5, and it's going to sound nothing like any of them. Then I create the little ingredients, and then once I have the ingredients that are more-or-less carbon copies of the original, I change the key, change the tempo, and change the sound.*

Most of the users I spoke with enjoyed using code for creative production. However, there were a few exceptions. Alex, a professional composer and producer who writes orchestral scores for film and video games, does all their composing with sheet music before moving to a DAW for production. Alex typically starts a composition with a melody, and then adds harmony, and finally orchestration.

Jean, who wrote pop songs, likes to begin a composition with a chord progression written on paper, or a guitar lick notated with sheet music, and then builds the composition around that source material. Jean liked to avoid screens and computers when composing, preferring an interface "where when you play something, sound comes out, and there's no setup."

For each Fluid Music user that I spoke with, after asking about their process, I then asked if they thought that same production process could be captured with the Fluid Music framework. A critical distinction is that I was not asking about whether an algorithm could intelligently make the same edit or decision that the user made. When creating a track, a producer might have several reverb effects in the mix, and any audio source in the mix can be sent to any of the reverb effects in varying amounts. The producer must prepare the reverb unit and decide how much of each separate audio source should be sent to each reverb. The question to Fluid Music users is not "can your algorithm set the gain on each reverb send?" The question is "can you write a function that *exposes* the decisions about gain to the consumers of your code?" If it is possible to write the function that exposes the question to the user, then your consumers can decide if they want to parameterize that function automatically using another algorithm, or if they want to specify the send-gain parameters manually. All study participants responded affirmatively, saying that the Fluid Music framework did provide the tools needed to describe their sound design or compositional process.

I followed by asking what parts of their process users *want* to capture with code. Every user had parts of their workflow that they wanted to control computationally with Fluid Music. Users expressed many different reasons for this.

Sean was excited to write code that generated music in his own compositional style and to use the generated content as source material for more polished music. Sean said, “I would create a system that literally makes music in the style of Sean. But I would never in a million years create a bounce and release that. I would always have 500 tweaks I need to do.”

Casey also wanted to use Fluid Music in the exploratory part of the composition process, having found GUIs to be inefficient. Casey said, “When you're tweaking a GUI, you are limited to the range of tweaking that you have time to do… If you can iterate over 200 parameters and then listen to them and identify the things that are interesting, you can actually get a lot more mileage out of it.”

Some users had hesitations about handling certain parts of their workflow with the Fluid Music framework. For example, Kerry, who often creates music with code in a live context, said they would probably not use the score language to transcribe rhythms note-by-note, as they prefer to write algorithms that output notes, as opposed to writing the notes themselves.

Jessie and Alex both thought that the Fluid Music framework would be valuable for configuring DAW templates with tracks, routing, and plugins. However, Jessie and Alex also agreed that they would be more likely to use the traditional DAW to insert content into templates than they would be to use the Fluid Music score language for composing and sequencing content. Alex thought the framework would be most useful for creating interactive music, because it can computationally produce variations of musical content and swap variations in and out depending on the user input.

Alex described how, when working in film scoring, there is a need to manage many DAW templates and reuse those templates in slightly different configurations. For example, Alex has a DAW template for romantic scenes that starts with a slow tempo and has a string ensemble, two woodwind instruments, and three beats in a bar. If there is just one such template, then maintaining it is easy, but Alex has many variations of that template. To make a change to one of the tracks in the template, Alex must manually update all the templates that use that track. The Fluid Music framework could streamline the process of updating one setting, and automatically applying that update to many derived sessions. Jessie described a similar need, but in the context of interactive art installations that use a DAW for interactive audio synthesis. Like Alex and Jessie, Jean thought the score language would be less useful than existing graphical tools, but did think that the framework would be useful for arranging songs by sections (verse, chorus, etc.).

Kerry doubted that the Fluid Music framework would be better for live code-based musical performance than the DSLs that are specifically designed for that purpose. In contrast, Jordan, who also likes to perform music with live coding, appreciated the score language as a quick way to build rhythms with code that was also easy to understand.

6.4.4 DAW and plugin support

All users I spoke with strongly welcomed both DAW and plugin support. Kerry wanted to use plugins in live coding performance, but found it difficult to achieve with existing tools for live coding. Describing past experiences with plugins, Kerry said “Man, it's such a pain. It's SUCH a pain. I've been trying to make that work… I haven't dared to do a performance with plugins yet.” Alex had found it difficult to produce audio that sounded satisfactory without the tools in a DAW. Describing Fluid Music, Alex said “It directly goes into the DAW, which means that I have all the functionalities of the DAW, which means it could sound amazing. It could sound perfect." Other users expressed a similar sentiment.

6.4.5 Using Techniques created by others

Broadly, all users expressed interest in employing user-generated Techniques in their own compositions. Different users had different stipulations for what kinds of Techniques they wanted to use. Without being prompted, half of the 10 users I interviewed expressed a desire to make sure that when they used a Technique in a composition, there was some way to credit the original Technique author.

Casey, Ben, and Jean said that it was important that Techniques be configurable, meaning that they wanted to be able to adjust their behavior and the sonic results they produced. Packages distributed on NPM are open source by default, and Ben said that as a result, using an NPM package is “a lot less like downloading someone else’s template and ripping them off, or ending up sounding like another producer or another artist.”

Because Fluid Music Techniques can perform any action that can be described with code, they can insert different kinds of content into a session including plugins, audio files, MIDI notes, automation, and audio track configuration. Some users felt inclined to avoid Techniques with certain content types. For Sean, this was a practical choice. If a Technique inserts an audio file into a session, and that audio file is under copyright and ends up in a mix, YouTube’s algorithms might detect the copyright infringement. If a Technique uses any audio files, Sean would avoid that Technique unless there was a way to be absolutely sure that the audio files could be used legally without consequences.

Alex was interested in using Techniques that contained production, effects, processing, or plugin configuration, but did not want to use premade chord progressions, saying "That feels the most violating of someone else's property." Alex's aversion to using chord progression Techniques extended to procedurally generated chords, because in their words "The composer's genius is in the code... I wouldn't feel like it's my composition."

Sam mentioned feeling "uneasy" about reusing a musical motif but doesn't feel uneasy when an artist like Girl Talk[21] creates a mashup from many different songs because of the extent to which a Girl Talk mashup transforms the source material that was used to create it. Generally, Sam expressed interest in using other's Techniques provided there is a way to credit the original Technique author.

Kerry was broadly in favor of sharing Techniques of all kinds, saying "I totally support sharing. I do that all the time," and "If somebody has made something cool, I would much rather take that forward instead of spending time reimplementing it. I mean that's the whole idea of open source itself, right? Just standing on the shoulders of giants – and then passing it on"

Some users were actively opposed to keeping Techniques private. Recognizing that all content was inspired by *something*, Pat said "I think it's extremely distasteful for someone to take something of somebody else's, find success with it, and act like [the original creator] had nothing to do with it."

6.4.6 Sharing Techniques with others

Wikipedia would not work if no one wanted to share their writing and expertise. Similarly, Fluid Music will only work if users want to share their Techniques. Question 6 asks users how they feel about writing a Technique and publishing it to NPM so that anyone can use it. This question is especially relevant to the group of users I talked to, because for the foreseeable future, *creating* Techniques will require coding skills. The group of users that I assembled for the workshop had an interest in music production and software engineering, both of which are necessary for Technique creation.

Users were consistently in favor of publishing their Techniques. Some users valued the experience of seeing others use their Techniques. Jordan said, "I get more ideas when I see other people using my ideas, so of course I want to give it away." Other users did not see any value in coveting Techniques. Jessie explained, "It's okay to share these techniques and the process behind them. Because I think the process itself – it is not magical."

Some users vocally opposed keeping Techniques secret. Casey said:

[21] Girl Talk is an "mashup artist," who makes music by splicing and layering hundreds of short clips, most of which are sampled from pop music. Each sample might be as short as a single beat, or as long as a few measures. (Schuster; Mongillo)

> *I strongly believe that anyone who relies on keeping their techniques a secret doesn't have a leg to stand on as a producer... I think any good engineer or producer has a lot more than just a few secrets. They have a lot of experience and really amazing ears – those are things that you can't just get from a preset. Those are things you have to develop over time.*

More than one user thought the established practices surrounding open-source software might make developers inclined toward publishing and sharing in comparison to musicians, who may be more incentivized to covet their work. Ben said, "As a developer, I live in that world. We rely on open source, and I contribute to open source, and so I think the mentality that I have, and the mentality of a lot of my peers, is that you share everything." Ben recognized that they made a living as a developer, implying that if they made a living as a music producer, they might be less inclined to describe the work that they do with code. Jordan also speculated that if they made their livelihood from music production, they might feel less inclined to publish compositional Techniques.

Sean made a distinction between sharing Techniques that produced musical notes and studio-production oriented Techniques. Speaking as a music producer, Sean felt that the author of the musical notes in a composition would likely be more strongly associated with authorship than with studio production techniques. Sean expressed unwillingness to share Techniques that included musical notes but felt happy to share Techniques for processing or synthesizing those notes, saying, "I don't give a f***. My product is my music. My product is not my production techniques."

Alex also felt hesitant to share Techniques that produced musical notes. While considering if they would release an algorithm that described their own musical composition process, Alex said:

> *I feel like people use the mystery of composition as a way to restrict the amount of people that can compose. And so, as an educator I would love to share everything and have everyone else also share everything. But as a professional who needs to make money, I feel like maybe it's better if the people I make music for cannot decipher it code line by code line.*

Alex was the only user who spoke about making a living from composing, while several other users reflected that their feelings about sharing were influenced by the fact that they did *not* make a living from composing. As a result, we should consider Alex's reaction and the questions raised by their reaction as important areas for future research.

In more than one casual conversation about Fluid Music outside of the workshop and user study, some people told me they were skeptical that sound designers or audio engineers would be willing to share their Techniques. Why was there such a strong pro-sharing attitude among user study participants? Ben cited one reason explicitly, noting developers were already familiar with the idea of sharing open-source software. Because Fluid Music techniques are expressed with code, sharing them may feel like sharing any other kind of code. There are also many more ways to share music production ideas than there where were two decades ago, and these could be helping to change people's attitudes. There are countless music production tutorials and audio engineering tip videos on YouTube, and the /r/synthrecipes and /r/audioengineering subreddits are filled with more advice, tutorials, questions, and answers about how to achieve certain musical results. In the past, software engineers were more skeptical about open-source code, but now many of us rely on it as Ben observed. As tools for collaborative media production develop, will lingering skepticism subside? One study found that code-intensive collaborations receive higher peer ratings than media-intensive collaborations in the graphical programming language Scratch (Hill and Monroy-Hernández). However, as the authors of the study acknowledged, Scratch is an unusual environment, and attitudes about sharing content in Scratch may not be consistent with attitudes about other types of content. More research is needed to understand present attitudes about collaborative media production, and their trajectories into the future.

6.5 User Study Limitations

Ten of the Fluid Music workshop participants chose to join the user study. The gender balance of participants in the workshop and in the user study skewed heavily male. However, gender representation in the workshop and study were more diverse than the Audio Engineering Society membership statistics referenced by Buckingham and Ronan in 2019.

Study participants skewed toward coders, expert users, and "technology first" professionals, who have experience producing music and developing software. This was expected because the prior workshop that was attended by all user study participants was advertised as using code to produce music. Experience with both software engineering and music production is required for Fluid Music Technique creation, so the background and skill set of users that participated in the study reflect the expected background and skill set for Technique creators.

Currently, the only way to use the Fluid Music Techniques is by writing code. As a result, the backgrounds represented in the user study do reflect current Fluid Music users. However, future goals for the Fluid Music project include making Fluid Music Techniques accessible to musicians who cannot or do not write code.

When interpreting the user study results, we should be cautious when drawing conclusions because of the small sample size, and because the user backgrounds represented in the study do not represent the expected backgrounds for all Fluid Music users in the long term. In particular, the participants in the user study may better reflect the expected background of Technique creators than Technique consumers in the long-term.

Finally, user study participants understand that Fluid Music is a personal project. They knew that Fluid Music is closely tied to my PhD work, and that I am personally invested in its success. When discussing Fluid Music with me, study participants were likely less critical than they would have been in a more neutral setting.

7 Next Steps and Future Work

In a 2011 interview, Max Mathews described how he developed the MUSIC software library, which was among the first tools to create audio from a digital computer, saying:

> *The crucial thing here is that I didn't try to define the timbre and the instrument. I just gave the musician a tool bag of what I call unit generators, and he could connect them together to make instruments that would make beautiful music timbres. I also had a way of writing a musical score in a computer file, so that you could, say, play a note at a given pitch at a given moment of time, and make it last for two and a half seconds, and you could make another note and generate rhythm patterns. This sort of caught on, and a whole bunch of [software] programs in the United States were developed from that* (Dayal and Mathews).

This "bag of tools" approach that "caught on" was adopted by other computational sound design tools including CSound, SuperCollider, Pure Data, Max, ChucK, and others. I found these tools were useful for some purposes, but when using these tools for music production I found myself struggling to produce polished results without access to the slightly higher-level tools that are available in DAWs. As a result, Fluid Music abandoned the minimalist "bag of tools" approach in favor of treating the core DAW components as the primitives for computational sound design, as described in chapter 3. Fluid Music users enthusiastically welcomed this approach as described in chapter 6, which reflects favorably on the Fluid Music design process and results. However, the real validation will come in time if the Fluid Music concept or framework becomes widely adopted.

Except for the involvement of user study participants, the design process was largely a solitary experience, and was more informed by my background in audio and software engineering than it was by interaction or collaboration with other users. The primary next step for Fluid Music is the transition from a dissertation project to a community project. The objective of the next stage of the work is to enable as many people as possible to develop their own Fluid Music workflows, Techniques, and interfaces. This approach leans on the power of the internet as a tool for collective exploration, experimentation, and action, with the optimistic view that at least one user (and probably many users) will create powerful and useful Fluid Music tools that I would not imagine on my own. If those users choose to share their Fluid Music tools and Techniques with each other, we all benefit from each other's work.

A secondary step is the development of a user interface to make Fluid Music Techniques accessible to musicians, composers, and audio engineers who cannot code. As discussed in chapter 6, this is one of the main limitations of Fluid Music in its current state. While a minority of the participants in the user study preferred to use Fluid Music with code, more users expressed a preference for a graphical user interface. The obvious GUI would resemble a conventional DAW (or perhaps extend an existing DAW) so that users could invoke Fluid Music Techniques from a familiar interface for the purposes of composition. My inclination is that the ideal Fluid Music interface would have some similarities with conventional DAWs, but also some non-trivial differences. At the same time, I suspect that *not* resembling a DAW could be an advantage in a Fluid Music interface. In my experience editing and mixing in many different DAWs, I have often found them to be a very tedious way to interact with sound. For me, the potential to automate many of the tedious and time-consuming aspects of working in a DAW is a big selling point of the Fluid Music Technique API.

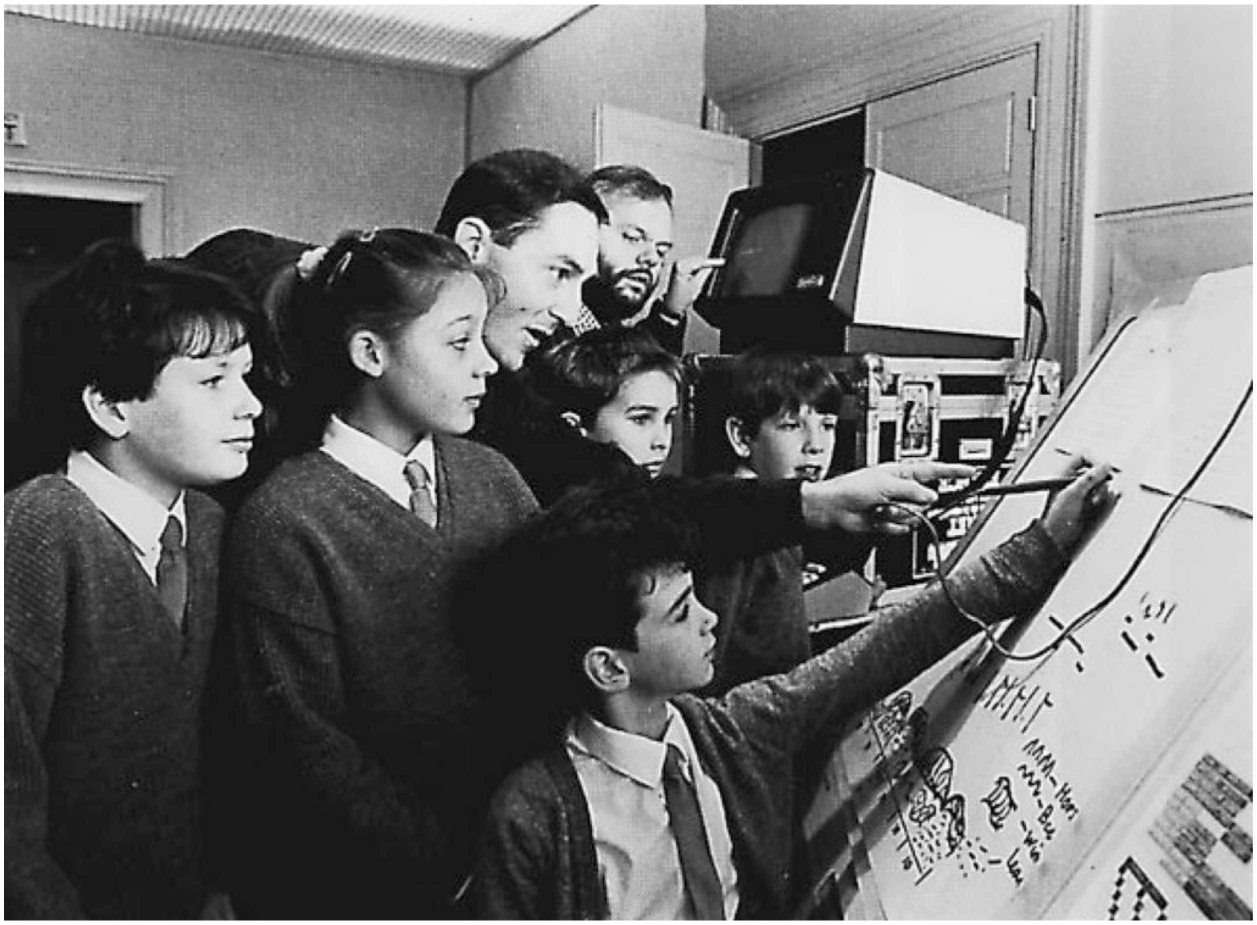


Figure 7.1 Music students composing with the UPIC system in 1983. Composer Iannis Xenakis conceived the UPIC system in the 1970s. The digital system for composition allowed users to draw notes and automation curves with a tablet and stylus interface. Today it is comparatively simple to digitize hand-drawn automation curves created with a tablet interface and use optical note recognition to recognize hand drawn notation on staff paper. Image from "Huddersfield Contemporary Music Festival Review" in the November 26, 1987 publication of the Huddersfield Daily Examiner (Bourotte and Delhaye).

One approach to creating a dedicated Fluid Music interface is to take inspiration from Oramics (Figure 2.3) or the UPIC system (Figure 7.1). In the past, stylus-based systems for composing electronic music have not gained traction with musicians. However, existing stylus-based systems operate as closed environments. By combining a system like this with Fluid Music, hand-drawn symbols and automation curves could invoke customizable Fluid Music Techniques, enabling users to collectively expand the notational language and its sonic vocabulary. However, this is just one possible approach and there are many others. For example, another interface might focus on injecting computational composition or machine learning into certain parts of the composition process or be tailored to a certain musical genre. The Fluid Music framework provides the foundation to build and experiment with many alternatives, and the best results will emerge by supporting users to create the interfaces that suit their interests. Because the framework standardizes the Technique API, custom techniques created by users will be forward compatible with Fluid Music interfaces developed in the future and maintain compatibility with the code-based score API that is bundled with the `fluid-music npm` package.

7.1 Research questions

In the 1950s, the Westdeutscher Rundfunk radio studio where Karlheinz Stockhausen created *Gesang Der Jünglinge* was outfitted with simple radio test gear like a sine wave generator, a noise generator, and a band pass filter (Chang). To create the pitched timbres used in the piece, Stockhausen and his assistant meticulously generated and overdubbed sine tones in a harmonic series to build harmonically rich waveforms. The gear in the studio was flexible enough to enable this early form of additive synthesis, but it was also primitive insofar as it took over a year to finish the composition. Today, dedicated synthesizers make it easy to compose with harmonically rich synthetic timbres. However, the gear at the WDR studio was a necessary step toward the modern musical technology that is popular today. To arrive where we are today, we had to discover what technology was useful to musicians and recording engineers, and then figure out how to create it.

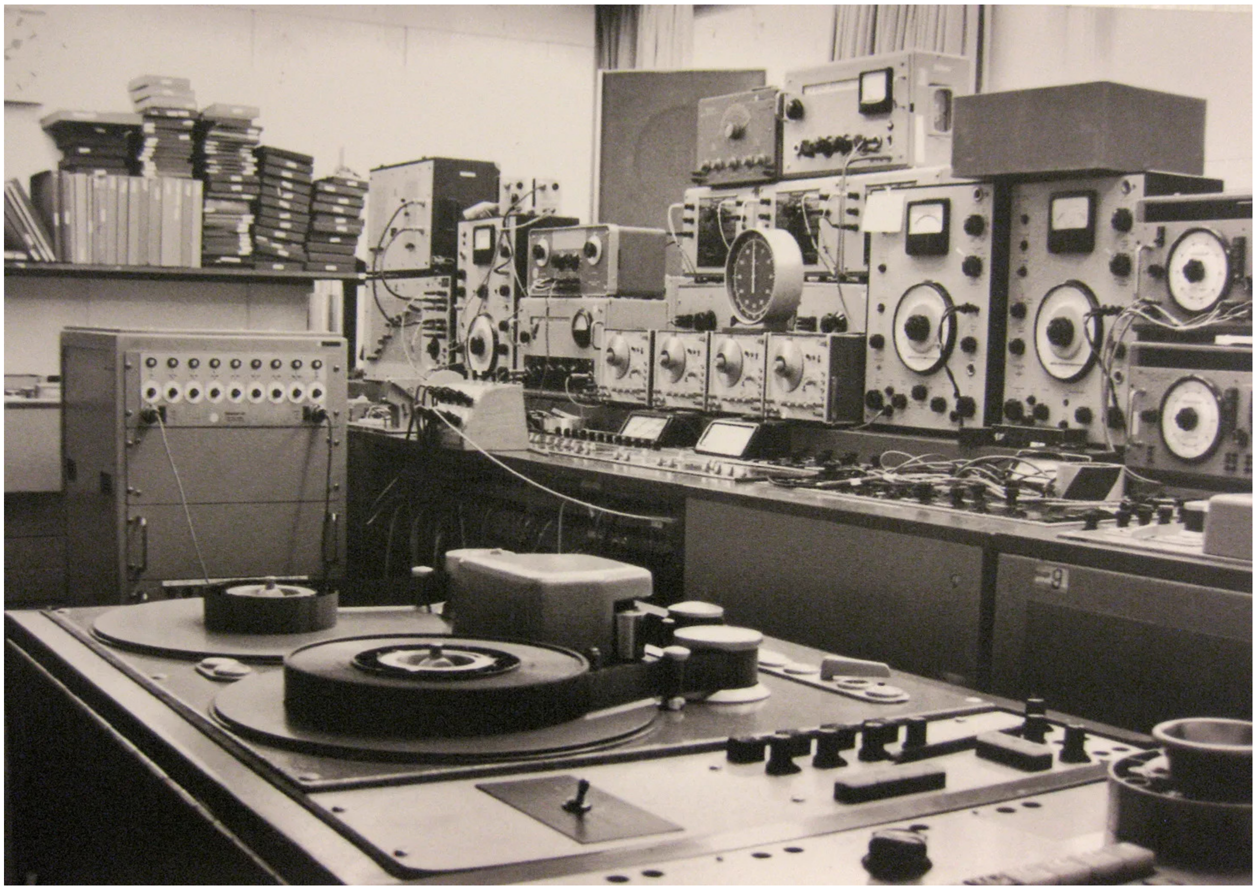


Figure 7.2 A four-track tape recorded and patched wave generators at the Westdeutscher Rundfunk studio. Date unknown ("WDR Electronic Music Studio, Werner Meyer-Eppler, Robert Beyer & Herbert Eimert, Germany, 1951").

The concept of a Fluid Music Technique is approximately as mature as sound synthesis was in the 1950s. Today, a Fluid Music Technique is flexible, insofar as it can do anything that you can describe with code. The Techniques that are bundled with the `fluid-music` `npm` library describe simple actions like inserting an audio file, MIDI note, or automation point into a DAW session. Users currently need to combine these primitives to achieve musical sounding results.

In Stockhausen's time, "How do you synthesize musical timbres?" was a fundamental research question. The technology at the WDR studio enabled him to explore the possibility space. With Fluid Music, the high-level questions might be "What kinds of Techniques are musically useful to share?" and "What parts of music production processes can be generalized with computation?" The Fluid Music framework enables us to begin exploring this possibility space. In the following paragraphs I will identify some more targeted areas for future research.

For Fluid Music to work optimally, we need to understand the boundaries around what is useful to share and reuse. How rigid should the boundaries be? In what ways should reusable techniques be configurable? Fluid Music Techniques and Technique Libraries can encapsulate chord libraries, automated and computational production, MIDI content, and any combination of production processes that involve core DAW components. Techniques are very open ended, and only a subset of possible Techniques will be musically useful. For example, a technique that generates a chord progression can be combined with another technique that generates a harmonic rhythm. This allows us to mix and match different combinations of chords with harmonic rhythms. Is that a musically useful paradigm?

Perhaps composers are not interested in reusing specific chord patterns, and Techniques and interfaces should focus on making it easy for a composer to author their own harmonic material note by note. In that case, a Technique that orchestrates harmonic material might also be useful. By allowing many different users to author their own orchestration algorithms, composers could experiment with different orchestration Techniques contributed by different users, and then fine-tune the results. In this example, Fluid Music can provide the framework that ensures compatibility between different orchestration Techniques.

Techniques in the standard library consistently perform simple actions. In the standard library, JavaScript class constructors are used to customize the Technique behavior.[22] Once instantiated, Techniques in the standard library are "dumb" – they will perform the exact same action each time they are invoked. However, "smart" Techniques can use computation or machine listening to customize their behavior and perform different actions in different contexts. What kinds of decisions do we want smart Techniques to be making on behalf of Fluid Music users? The Fluid Music framework enables us to explore the possibilities, while ensuring that we can inspect and override the results in a DAW.

The standard library contains Techniques for setting and automating plugin parameters. To use these techniques, the user must specify a plugin and parameter value. As a result, any user who wishes to reuse a Technique must have access to the identical plugin as the Technique author. This enforces consistency in the signal chain, but it also means that, for example, Techniques that depend on a particular reverb plugin cannot be used with a different reverb plugin, even when the two plugins are very similar in their behavior. How portable should Fluid Music techniques be? The current solution is practical, and it also allows us to use Fluid Music with many different plugins that exist today without worrying that another user who reuses our Technique may have results that sound very different from our own. However, it also limits the long-term viability of Techniques that depend on plugins which may or may not be supported in

[22] For example, when instantiating a `PluginAutomation` class from the standard library, the user supplies a value to the class constructor and, when the resulting Technique is invoked, it will always insert an automation point at the specified value. The track and time of the automation point will be determined by the Technique invocation context, and the plugin, parameter, and parameter value are chosen when instantiating the Technique. After instantiation, the technique can be invoked on any track (assuming the track contains the correct plugin) at any time. For each Technique invocation, the technique will insert the same automation value. To insert a different value, the user should create a new instance of the `PluginAutomation` class which creates a new Technique. See chapter 4 for details.

the future. When composing for acoustic instruments, we expect that different musicians will produce different interpretations of the same musical score. Should scores for digital audio production work the same way?

Fluid Music currently depends on the `npm` platform. So far, `npm` has met the needs of the project. Fluid Music users can publish and reuse packages that contain Techniques and Technique Libraries. It handles dependency management and Technique versioning. Most engineers who are familiar with JavaScript are also familiar with `npm`, so new Fluid Music users typically do not have to learn a new tool to access existing Fluid Music packages. However, `npm` was designed for sharing code, and the long-term needs of Fluid Music may diverge from the needs of `npm`. A platform for publishing and sharing Fluid Music techniques would enable dedicated tools for Technique organization and discovery.

Intellectual property laws for traditional music scores and music recordings emerged from the evolving interests of artists, composers, performers, and the music industry. While Fluid Music is still a new idea, we have an opportunity to nudge the trajectory of the legal issues that surround sharing and reusing music production Techniques. How should we handle intellectual property? What kinds of content do we want users to “own” or keep as a trade secret?

Some of these research areas are more pressing than others, but all of them are relevant to the development of a fully realized Fluid Music interface. How should a Fluid Music interface be similar or different from existing tools? There are too many questions for me to study alone. It is time to focus on supporting a wider community of researchers, engineers, and musicians.

7.2 A Notation Language for Music Production

Music recording and music production are new art forms. Considering the long and slow evolution of established notation systems, we should not be surprised that we do not yet have a robust notation language for music production. The proliferation of tools for music production introduces a new notational challenge: Every new effect, processor, and synthesizer introduces new capabilities and a unique interface. We cannot anticipate the new production Techniques that musicians, composers, and engineers will discover in the future. No notational system can account for all the different ways that music production technologies can interact with each other.

Fluid Music makes two contributions that address this challenge. First, the Fluid Music concept proposes that we make a notational language that anyone can contribute to. This uses the strength of the internet as a tool for facilitating collective action, enabling Fluid Music users to document and share new production Techniques as they are discovered. Second, the Fluid Music software framework provides the necessary foundation to study and create Fluid Music.

The final contribution of Fluid Music as it exists today is a history of the idea that technology will profoundly shape music, as covered throughout chapter 2. I want to emphasize that this is *a* history of the idea, not *the* history. It is not exhaustive. However, to my knowledge, no other source aggregates predictions about how technology can (or should) profoundly change music in the future. This history draws from the writing and lectures about music by inventors, composers, and theorists, including Thomas Edison, Kurt Weill, Edgard Varèse, Daphne Oram, Luigi Russolo, László Moholy-Nagy, Paul Théberge, Glenn Gould, and various others who shared analysis and predictions on the impact of the telegraph, radio, or the internet. This history includes some striking insights, some falsified theories, and hope and hyperbole in equal measure. The one person who was wholly correct in her predictions was Daphne Oram, who simply said that we cannot know what the future will bring. With that in mind, I would offer my own possibly hyperbolic hope for Fluid Music.

In the future, when you hear music of any kind, you will be empowered to deconstruct the sound world it contains, and fluidly combine, blend, compose, and recreate it with any combination of other sound worlds, resulting in a *fluid* score that everyone can reinterpret just as easily.

8 References

Adena, Maja, et al. "Radio and the Rise of Nazis in Pre-War Germany." *SSRN Electronic Journal*, June 2013, doi:10.2139/ssrn.2242446.

Attali, Jacques. *Noise : The Political Economy of Music*. University of Minnesota Press, 1985.

Auslander, Philip, and Ian Ingils. "'Nothing Is Real': The Beatles as Virtual Performers." *The Oxford Handbook of Music and Virtuality*, edited by Sheila Whiteley, Oxford University Press, 2016, pp. 35–51.

Automatic Double Tracking - Wikipedia. https://en.wikipedia.org/wiki/Automatic_double_tracking. Accessed 15 July 2021.

Badenoch, Alexander. *Voices In Ruins*. Palgrave Macmillan, 2008.

Bay, Sebastian. *Social Media the Black Market for Social Media Manipulation*. 2018.

Bazzana, K. *Wondrous Strange: The Life and Art of Glenn Gould*. McClelland & Stewart, 2010.

Beatles, The. *Abbey Road*. Apple Records, 1969.

---. *Penny Lane*. Parlophone Records Limited, 1967.

---. *Please Please Me*. Parlophone Records Limited, 1963.

---. *Rubber Soul*. Parlophone Records Limited, 1965.

---. *Sgt. Pepper's Lonely Hearts Club Band*. Parlophone Records Limited, 1967.

---. *Strawberry Fields Forever*. Parlophone Records Limited, 1967.

Bergmeier, Horst J. P., and Rainer E. Lotz. *Hitler's Airwaves: The Inside Story of Nazi Radio Broadcasting and Propaganda Swing*. Yale University Press, 1997.

Bloomberg, Benjamin. *Making Musical Magic Live: Inventing Modern Production Technology for Human-Centric Music Performance*. MIT, 2020.

Blue - A Music Composition Environment for Csound. https://blue.kunstmusik.com/. Accessed 29 June 2021.

Borschke, Margie. *This Is Not a Remix: Piracy, Authenticity and Popular Music*. Bloomsbury Publishing, 2017.

Bourotte, Rodolphe, and Cyrille Delhaye. "Learn to Think for Yourself: Impelled by UPIC to Open New Ways of Composing." *Organised Sound*, vol. 18, no. 2, Cambridge University Press, 2013, pp. 134–145, doi:10.1017/S1355771813000058.

Bradshaw, Samantha, and Philip N. Howard. *The Global Disinformation Order 2019 Global Inventory of Organised Social Media Manipulation*. 2019.

Braun, H. J., and International Committee for the History of Technology. *Music and Technology in the Twentieth Century*. Johns Hopkins University Press, 2002.

Buckingham, Shelley Ann McCarthy, and Malachy Ronan. "Factors Contributing to Gender Imbalance in the Audio Industry." *Audio Engineering Society Convention 146*, 2019.

Case, Alex. *Sound FX: Unlocking the Creative Potential of Recording Studio Effects*. Focal Press, 2007.

Chang, Ed. *Stockhausen: Sounds in Space: GESANG DER JÜNGLINGE*. http://stockhausenspace.blogspot.com/2015/01/opus-8-gesang-der-junglinge.html. Accessed 8 Aug. 2021.

Clark, Rick. "Roy Thomas Baker: TAKING CHANCES AND MAKING HITS." *MixOnline.Com*, 1999, https://web.archive.org/web/20050426041352/http://mixonline.com/mag/audio_roy_thomas_baker/.

Cory, Mark, and Barbara Haggh. "Horspiel as Music: Music as Horspiel: The Creative Dialogue between Experimental Radio Drama and Avant-Garde Music." *German Studies Review*, vol. 4, no. 2, 1981, p. 257, doi:10.2307/1429272.

Cross, Lowell. "Electronic Music , 1948-1953." *Perspectives of New Music*, vol. 7, no. 1, 1968, pp. 32–65.

Danielsen, Anne. "Glitched and Warped." *The Oxford Handbook of Sound and Imagination, Volume 2*, edited by Mark Grimshaw-Aagaard et al., no. February, 2019, pp. 593–609, doi:10.1093/oxfordhb/9780190460242.013.27.

Dayal, Greeta, and Max Mathews. "Max Mathews Interview with Geeta Dayal." *Frieze Publishing*, 2011, https://web.archive.org/web/20120616210338/http://blog.frieze.com/max-mathews/.

DiResta, Renee, et al. *The Tactics and Tropes of the Internet Research Agency*. 2018.

Edison, Thomas. "The Phonograph and Its Future." *North American Review*, 1878.

Frank, Gillian. "Discophobia: Antigay Prejudice and the 1979 Backlash against Disco." *Journal of the History of Sexuality*, vol. 16, no. 2, 2007, pp. 276–306.

Friedman, Ted. *Electric Dreams: Computer Culture and the Utopian Sphere*. Duke University, 1999.

Gould, Glenn. "The Prospects of Recording." *High Fidelity*, no. April, Apr. 1966, pp. 46–63.

Gradstein, S., editor. "The Philips Pavilion at the 1958 Brussels World Fair." *Philips Technical Review*, vol. 20, Philips Technical Review, 1959, pp. 1–36.

Grande, Ariana. *No Tears Left To Cry*. Republic Records, 2018.

Hafner, Katie, and Matthew Lyon. *Where Wizards Stay Up Late: The Origins of the Internet*. Simon & Schuster, 1996.

Hamano, Junio C. "GIT — A Stupid Content Tracker." *Linux Symposium*, vol. 1, 2006, pp. 386–94.

Hermand, Jost, and James Steakley, editors. *Writings of German Composers*. Continuum, 1984.

Hesmondhalgh, David. *Why Music Matters*. 2013.

Hill, Benjamin Mako, and Andrés Monroy-Hernández. "The Cost of Collaboration for Code and Art: Evidence from a Remixing Community." *Proceedings of the ACM Conference on*

Computer Supported Cooperative Work, CSCW, 2013, pp. 1035–45, doi:10.1145/2441776.2441893.

Holbrow, Charles. “Turning the DAW Inside Out.” *Audio Engineering Society Conference: 146th Convention*, 2019.

Howell, Steve. “The Lost Art Of Sampling: Part 1.” *Sound On Sound*, 2005, https://www.soundonsound.com/techniques/lost-art-sampling-part-1.

Iverson, J. *Electronic Inspirations: Technologies of the Cold War Musical Avant-Garde*. Oxford University Press, 2018.

Jackson, Myles W. “From Scientific Instruments to Musical Instruments: The Tuning Fork, the Metronome, and the Siren.” *The Oxford Handbook of Sound Studies*, edited by Trevor Pinch and Karin Bijsterveld, Oxford University Press, 2011, pp. 201–23, doi:10.1093/oxfordhb/9780195388947.013.0056.

Jarvenpaa, Sirkka L., and Karl R. Lang. “Boundary Management in Online Communities: Case Studies of the Nine Inch Nails and Ccmixter Music Remix Sites.” *Long Range Planning*, vol. 44, no. 5–6, 2011, pp. 440–57, doi:10.1016/j.lrp.2011.09.002.

Julien, Olivier, editor. *Sgt. Pepper and the Beatles: It Was Forty Years Ago Today*. Routledge, 2009.

Kimsey, John. “The Watchamucallit in the Garden: Sgt. Pepper and Fables of Interference.” *Sgt. Pepper and the Beatles: It Was Forty Years Ago Today*, edited by Olivier Julien, Ashgate Publishing Limited, 2008, pp. 121–38.

Langston, Peter S. *Six Techniques for Algorithmic Music Composition*.

Lessig, L. *Remix: Making Art and Commerce Thrive in the Hybrid Economy*. Penguin Publishing Group, 2008.

Lewis, Rebecca. “Alternative Influence: Broadcasting the Reactionary Right on YouTube.” *Data & Society*, 2018.

Lewisohn, Mark. *The Beatles Recording Sessions: The Official Abbey Road Studio Session Notes 1962-1970*.

Machover, Tod. *Between the Desert and the Deep Blue Sea: A Symphony for Perth*. 2014.

Martin, George, and Jeremy Hornsby. *All You Need Is Ears*. St. Martin’s Press, 1979.

McLuhan, Marshall. *Understanding Media: The Extensions of Man*. First MIT, MIT Press, 1994.

Milner, G. *Perfecting Sound Forever: The Story of Recorded Music*. Granta Books, 2018.

Moholy-Nagy, Laszlo. “New Form In Music: Potentialities of the Phonograph.” *Der Sturm*, July 1923.

Mongillo, David. “The Girl Talk Dilemma: Can Copyright Law Accommodate New Forms of Sample-Based Music.” *Pittsburgh Journal of Technology Law & Policy*, vol. 9, 2009, pp. 1–32.

Morton, David. *Sound Recording: The Life Story of a Technology*. Greenwood Press, 2004.

Mullin, John T. "The Birth of the Recording Industry." *Billboard*, 1972, pp. 56–79.

Novak, Matt. "Musicians Wage War Against Evil Robots." *Smithsonian Magazine*, 10 Feb. 2012, https://www.smithsonianmag.com/history/musicians-wage-war-against-evil-robots-92702721/.

Oram, D. *An Individual Note: Of Music, Sound and Electronics*. A Galliard, Galliard Limited, 1972.

Phalen, William J. *How the Telegraph Changed the World*. McFarland & Company, 2014.

Ritchie, Hannah, and Max Roser. "Technology Adoption." *Our World in Data*, 2017.

Rooney, Michael. *AATranslator.Com.Au*. https://www.aatranslator.com.au/. Accessed 28 May 2021.

Ross, Alex. *The Rest Is Noise: Listening to the Twentieth Century*. 2007.

Russolo, Luigi. *The Art of Noise*. 1913.

---. *The Art of Noises*. 1916.

Schaeffer, Pierre. *In Search of Concrete Music*. University of California Press, 2012.

Schulz, David Paul. *The Rhetorical Contours of Technological Determinism*. The Pennsylvania State University, 2002.

Schuster, W. Michael. "Fair Use, Girl Talk, and Digital Sampling: An Empirical Study of Music Sampling's Effect on the Market for Copyrighted Works." *Oklahoma Law Review*, vol. 67, no. 3, pp. 443–518.

Schwartz, E., et al. *Contemporary Composers On Contemporary Music*. Hachette Books, 2009.

Schwartz, Peter, and Peter Leyden. "The Long Boom: A History of the Future, 1980 - 2020." *Wired*, July 1997.

Seabrook, J. *The Song Machine: Inside the Hit Factory*. W. W. Norton, 2015.

Smalley, John. "Gesang Der Jünglinge: History and Analysis." *Masterpieces of 20th-Century Electronic Music: A Multimedia Perspective., Columbia University*, 2000, pp. 1–13.

Snoman, Rick. *The Dance Music Manual*. Focal Press, 2009.

Sterne, Jonathan. *MP3: The Meaning of a Format*. Duke University Press, 2012.

Stockhausen, Karlheinz. *Gesang Der Jünglinge*. 1956.

Théberge, Paul. *Any Sound You Can Imagine: Making Music/Consuming Technology*. Wesleyan University Press, 1997.

---. "The Network Studio: Historical and Technological Paths to a New Ideal in Music Making." *Social Studies of Science*, vol. 34, no. 5, 2004, pp. 759–81, doi:10.1177/0306312704047173.

Turkle, S. *Reclaiming Conversation: The Power of Talk in a Digital Age*. Penguin Publishing Group, 2015.

Turner, Fred. *From Counterculture to Cyberculture: Stewart Brand, the Whole Earth Network, and the Rise of Digital Utopianism*. University of Chicago Press, 2006.

Waleik, Gary. "Forty Years Later, Disagreement About Disco Demolition Night." *Wbur.Org*, 2019, https://www.wbur.org/onlyagame/2019/07/12/disco-demolition-dahl-veeck-chicago-white-sox.

Warner, Dan. *Live Wires: A History of Electronic Music*. Reaktion Books, 2017.

"WDR Electronic Music Studio, Werner Meyer-Eppler, Robert Beyer & Herbert Eimert, Germany, 1951." *120 Years of Electronic Music*, https://120years.net/wdr-electronic-music-studio-germany-1951/. Accessed 8 Aug. 2021.

Willis, Michael. *Dragonfly Reverb*. https://michaelwillis.github.io/dragonfly-reverb/. Accessed 11 June 2021.

Witt, Stephen. *How Music Got Free: A Story of Obsession and Invention*. Penguin Publishing Group, 2015.

Womack, Kenneth. *Long and Winding Roads: The Evolving Artistry of the Beatles*. Continuum, 2007.

Wu, Tim. *The Attention Merchants: The Epic Scramble to Get Inside Our Heads*. Alfred A. Knopf, 2016.

Zuboff, Shoshana. "Big Other: Surveillance Capitalism and the Prospects of an Information Civilization." *Journal of Information Technology*, vol. 30, no. 1, Nature Publishing Group, 2015, pp. 75–89, doi:10.1057/jit.2015.5.

Appendix A User Study Interview Questions

During the user interview, I loosely followed the question template below. We took detours to pursue interesting tangents, and I occasionally skipped questions to stay within time constraints.

1. Could you tell me about your relationship to music, media, and technology?
   a. In what ways do you interact with them? For example, what audio languages or DAWs do you use? What kinds of things do you do with them?
   b. When you release something, is it finished? Or is it just in its current state?
2. After spending a little with Fluid Music, what did you notice, or take away from the workshop?
3. Conventional sound design documents (like a ProTools session file) describe the product, while Fluid Music aims to provide a language for describing the sound design process that results in a session. Can I ask you to think about your process?
   a. Walk me through your process. When you pull out your instrument, or open your code editor, can you describe what you think and do?
   b. For each step, can the Fluid Music language describe that step? Could you parameterize it, and create a NPM package with Fluid Music?
   c. For each step, would you want to parameterize it?
4. How did you respond to the DAW and plugin integration in Fluid Music?
5. When it comes to creating recorded music, the production process is typically considered an art form, wherein a producer may work to create a signature sound. How would you feel about finding and re-using other people’s techniques in your own projects?
6. Some producers treat recording or production techniques like trade secrets. Meanwhile, Fluid Music is built around the idea of freely sharing and reusing production techniques. Do you have any mixing techniques that you’ve come to rely on, and how would you respond to the prospect of publishing or sharing your own production techniques?
7. Any final thoughts about Fluid Music?