A Performance Prototype for Soundcool Online

Huan Zhang

May 2021

Music and Technology
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements
for the degree of Bachelor of Science.

A Performance Prototype for Soundcool Online

Abstract

Soundcool Online is a Web Audio re-implementation of the original Max/MSP implementation of Soundcool, a system for collaborative music and audiovisual creation. Running on a centralized server, *Soundcool Online* allows saving and sharing projects between users. Another advantage provided by the cloud-based implementation is the potential to enable collaborative synchronized module updates for users from different physical locations. Thus, in this thesis we define a new feature for *Soundcool Online*: *performance*. We present the motivation and design for a performance prototype and describe the basic functionality of collaborative performance achieved through Open Sound Control and Websockets. Experiments with latency measurements and user feedback from a demo performance are reported.

1 Introduction

Soundcool (1) is a free system for collaborative audiovisual creation. It consists of a set of modules such as audio and video players, effects and mixers, video switchers, and virtual instruments that run in Mac or PC computers as part of a Max/MSP stand-alone application. These modules can be controlled with the Soundcool OSC app for iOS or Android smart phones and tablets through local Wi-Fi or at a distance over the Internet. One of the original motivations to use mobile devices as controllers in Soundcool is the simple fact that many students already have mobile phones they can use. The original implementation assumed the availability of a Windows or macOS personal computer to act as "server" and user interface, while mobile devices act as controllers.

However, due to the limitation of computing resources and the difficulty of running the personal computer version of Soundcool on multiple platforms, we considered other implementation options and decided to use Web Audio. With Web Audio, we have a wider range of operating systems, a fairly device- and system-independent execution environment (web browsers and HTML5), and thus greater and simpler portability.

We have previously reported on our implementation of *Soundcool Online* (2), which includes functionalities of main audio modules implemented under Web Audio, notion of projects editing and saving, as well as a simple backend and database for storing user data. In this thesis, we extend the functionalities of *Soundcool Online* with a basic implementation of collaborative performance, where we introduce the notion of performance where multiple browser frontends can witness the same module update through OSC control from the Soundcool moble app. Essentially, this gives multiple performers real-time access to all the controls of a shared modular digital music performance system, as well as the ability to all hear the same sound, which is (re)computed locally with high quality for each performer.

In this paper, we presents our design and implementation of the performance prototype of *Soundcool Online*. Related work is discussed in the next section. Then, an overview and design goals of *Soundcool Online* is presented in (Section 3). In Section 4, we describe the design, control flow and implementation of the performance prototype in order to support collaboration. In Section 6, we give some performance measurements of latency and a performance experiment with player feedback. Finally, Sections 7 and 8 describe future work and present our conclusions.

2 Related Work

There are many music audio projects with an educational objective, such as EarSketch (3) and BlockyTalky (4). In the Web Audio sphere, iMuSciCA (5) offers web-based musical activities to support STEM subject learning, and Scott Fradkin's Snap Music (www.fradkin.com/snapmusic-0.3/tone-snap.html) is a music programming system for children. WebPd (https://puredata.info/downloads/webpd) is an experimental release of Pure Data (6), a visual programming tool for multimedia, that allows a subset of Pure Data audio patches to run in the browser.

A number of virtual synthesizers exist for Web Audio as well, including Synth Kitchen (7) and Zupiter (z.musictools.live/), and more can be found at Web Audio Modules (www.webaudiomodules.org/wamsynths/).

Soundcool Online mainly differs from this other work in its emphasis on collaborative performance, versatile high-level sound modules, and a large body of supporting materials and experience in primary and secondary music education.

3 Soundcool Online Design

3.1 Graphical User Interface

In *Soundcool Online*, the Project panel provides a list of available modules, which users can connect and control. Users are able to instantiate modules by clicking on the icon from the panel (shown in Figure 1). In the Project panel interface, modules are stacked in columns with even margins. The arrangement of the modules is flexible, and users are free to drag and reorder modules among columns. Rather than using lines or "wires," modules are interconnected by clicking on an output ("Out"), then clicking on an input ("In"). The output and input boxes indicate the connection status, and hovering over a connection box highlights all the connected modules to make inspection easy. This approach was created for the original Soundcool system to simplify the implementation, but we found it works well even for novice users. As a large project usually involves a lot of interconnected modules, explicit wires significantly increase the visual complexity on the interface and confuse the user.

Underlying each module, we employ the Web Audio API to implement audio processing. In Figure 2, we show the Web Audio nodes composing a Soundcool delay module, where users are allowed to control delay time and feedback. Most modules have input (button In generally at the top left) and output (button Out generally at the top right).

Figure 1: A sample project with Player, DirectInput, and SignalGen modules.

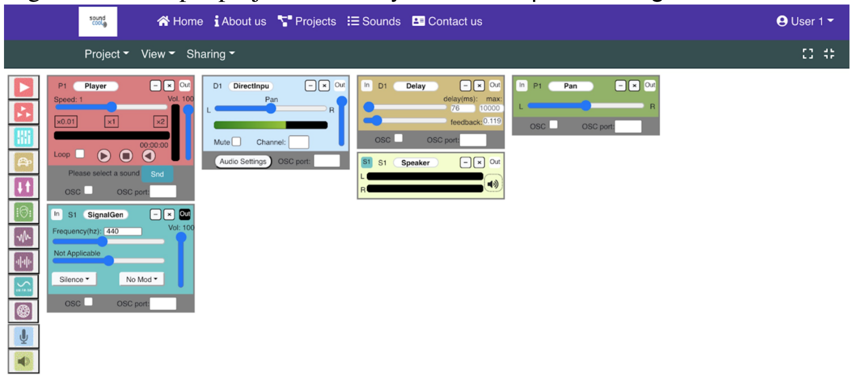
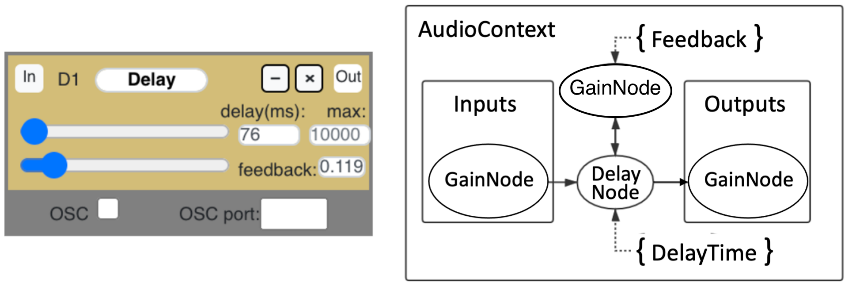


Figure 2: Soundcool Online's Delay module (left) and its Web Audio implementation (right).



For *Soundcool Online*, we have (so far) implemented the following modules: SignalGen, Player, SamplePlayer, Mixer, Delay, Transposer, Pan, Oscilloscope, Spectroscope, GranSynth, DirectInput (microphone input), and Speaker (audio output).

3.2 Patching vs Control

The tunable parameters of a project can be placed into two categories: Patching and Control.

Patching As discussed in Section 3.1, module and module connections are essential parts of the project. Patching refers to the overall interconnections of a project, and it can only be changed through the graphical interface by clicking on ("In") or ("Out") ports.

Controls Besides the patching, other module parameters such as the delay feedback and delay time are considered as Controls. Controls are typically dynamic and can be manipulated through both GUI and OSC.

As we are going to discuss in the Performance section, the patching will remain static during performances, while controls are the parameters that are manipulated through server-directed OSC messages.

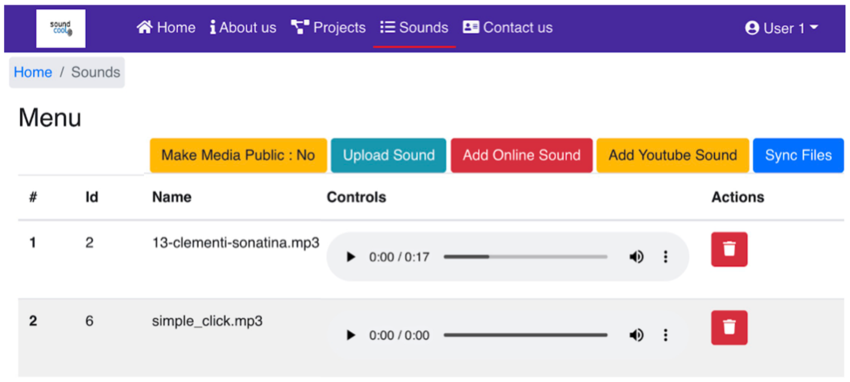
3.3 Backend and Database

Projects can be saved on the server in an SQL database using a JSON representation of modules, connections and project meta information. Users can also download and upload projects to and from local files using the same JSON format.

Besides saving and sharing projects, the interface offers a dashboard for quick access to a user's personal projects and to projects shared by others.

Although Soundcool projects often incorporate audio samples and loops, we are concerned about storing audio content, which could create a considerable demand for storage and bandwidth. Although we currently store samples on our server, we consider it a "feature" that users can obtain sounds from other online sources such as Freesound (8). Figure 3 illustrates our server page where users click "Add Online Sound" to add a sound to their library. Any Soundcool module that uses a sound, e.g. Player, has a button to select a sound from the user's library.

Figure 3: User sound inventory page.



#	Id	Name	Controls	Actions
1	2	13-clementi-sonatina.mp3	0:00 / 0:17	
2	6	simple_click.mp3	0:00 / 0:00	

3.4 Alpha Release and Deployment

Soundcool Online is implemented in Javascript under the `Node.js` environment. In the front-end graphical user interface design, the React framework utilizes `jsx` syntax, a combination of Javascript and HTML, to create more intuitive React components. In the development, we utilized `npm`, a package manager for Node JavaScript, to keep track of necessary libraries and hundreds of dependencies. In the locally executable version, these dependencies are compiled and linked with the `pkg` library to create a self-contained executable with no external dependencies on the `Node.js` environment.

Soundcool Online can be deployed in a classroom setting as a stand-alone application available as a single-binary executable file. This is created by the `pkg` library, which creates a self-contained executable with no external dependencies on the `Node.js` environment. We compile executables for three operating systems: Windows, Mac and Linux.

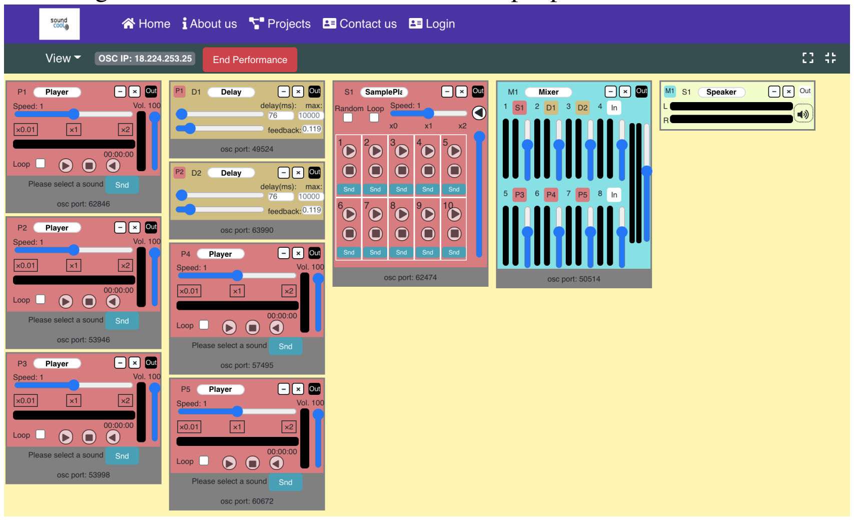
Alternatively, we have deployed *Soundcool* in the cloud, providing a shared server for many users, as long as they have Internet access. We serve the project with Amazon AWS ElasticBeanstalk, a service for deploying and scaling web applications. Under this service, we run the application on an EC2 server, and keep SQLite as database manager for objects in S3 storage. In order to achieve collaboration and sharing, we also implemented login services in the back end and store user accounts in the database.

4 Performance

In return for some additional latency, the cloud-based server solution offers an interesting opportunity for collaborative performance over the web. Instead of sharing a screen and low-quality audio over a conferencing system such as Zoom, we can run "clones" of a Soundcool project locally for every performer and for audiences too. Local copies of the project can compute and deliver high-quality audio. Collaborative control is achieved by sending copies of all control changes to every instance of the Soundcool project. Due to variations in network timing, not every instance will compute exactly the same sound, but given the overall response time, small timing variations are not critical.

To enable this type of distributed performance control and synthesis, our server supports named performances which are projects with a set of reserved OSC port numbers. Players that want to collaborate open copies of the performance by name and then share all control changes. A reservation system ensures that other projects can use other port numbers for OSC control without interference. For example, we can have one student in China operating player module 1 and one student in Spain operating player module 2, and they hear approximately the same audio from their local browsers.

Figure 4: Performance Panel of the example performance Mascletà



4.1 Performance Design

In the following points, we provide a specification for the life cycle of utilizing the performance feature.

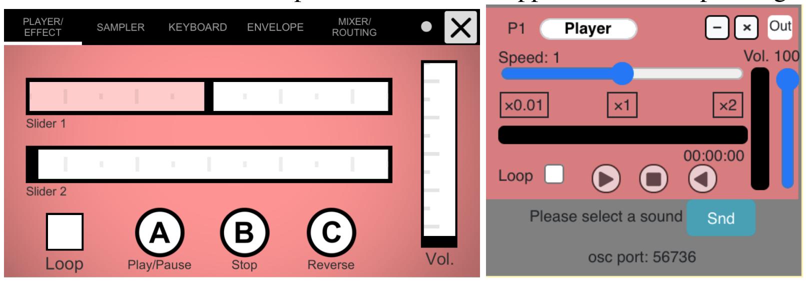
- **Initialization** A performance will be opened in a window similar to the project editor, but with a small interface difference (yellow background color, removal of add-module panel), indicating we are in "performance mode" instead of "project editing mode".
- **Playing** Since the goal is to perform instead of edit (which creates problems: how can edits be performed on all copies, and should edits be saved?), the patching of a performance is NOT allowed to be changed in the performing mode. The module parameters can be changed via OSC. After a performance, no changes will be saved (next time it is still going to be opened in the initial, default setting).
- **Sharing** After a performance is initialized, others are able to join it and receive simultaneous OSC updates from the server. The user is prompted for an existing performance name to join. Note that the sharing of performance is *global*, meaning that users from any account or even guest users are able to join a performance given the performance's name.
- **Ending**: In the performance panel shown in Figure 4, there is an "End Performance" button that will delete the copy of the performance and stop allowing others to join. However, we

also set a limitation of 4 weeks in case no one attempts to end it. Notice that, although performances can be joined globally, only the account that initialized this performance can end it.

4.2 Shared Control via OSC

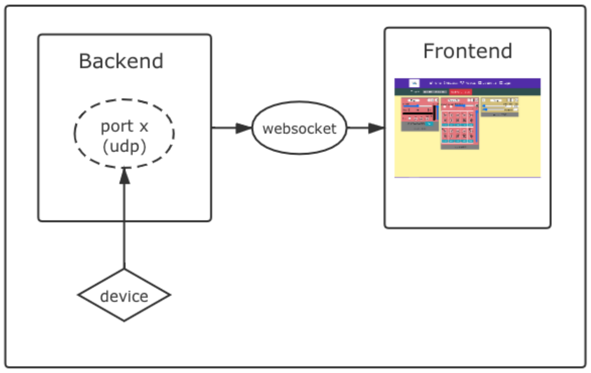
Open Sound Control (OSC) is a protocol for networking sound synthesizers, computers, and other multimedia devices for purposes such as musical performance or show control. OSC messages are transported across the internet and within local subnets using UDP/IP and Ethernet. For the *Soundcool* Project, there is a mobile phone app that emits OSC messages for Soundcool module control (9). See Figure 5 for a comparison between mobile OSC interface and module interface. Currently, we have implemented OSC control for Player, Sample Player, Delay and Mixer. In an ideal performance, each performer is going to use their mobile device to control one of the modules.

Figure 5: Interface of the mobile phone OSC control app and it's corresponding module



The biggest feature of performance mode is that it allows multiple front-end clients to receive simultaneous updates of the same performance, and this is achieved by the server directing OSC messages to multiple frontends via websockets, as shown in Figure 6. To be more specific, the pipeline for OSC control flow is as follows:

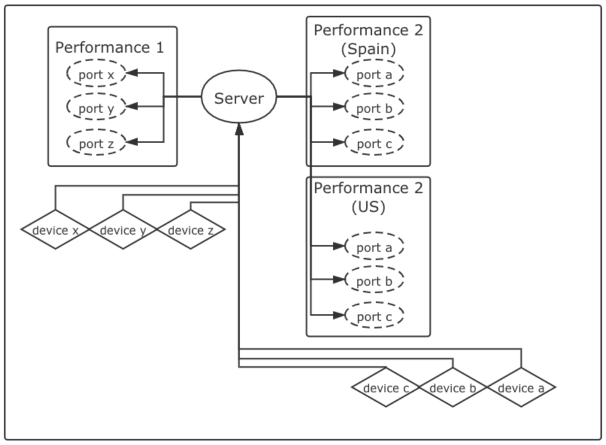
Figure 6: Flowchart of OSC pipeline



1. Backend opens up UDP ports when a performance is initialized.
2. Mobile OSC devices send to the server IP and module port.
3. Backend port receives control messages sent from mobile devices.
4. Backend translates and redirects the message to all front-end clients using Socket.IO (websocket).

5. Each frontend receives the translated message and handles the operation.

Figure 7: Shared Control Diagram



Multiple Performances It is possible that different performances can be opened simultaneously on the same server, and by assigning them different OSC ports, we can avoid them interfering with each other.
Port Assignment As shown in Figure 4, each module with OSC control is assigned a port number randomly. Port assignment is done when a performance gets initialized, and stays the same throughout the life cycle of a performance. Details of random port generation and collision handling are in Section 4.3.1.
Storage The server keeps track of active performances by storing a temporary JSON file with all the modules, ports information and so on. When a performance ends, the server needs to close all the associated ports and remove the database entry.

4.3 Collisions

4.3.1 Collision of Ports

As mentioned in 4.2, the shared control of a performance is done by directing the OSC messages to the same server and then distributing websocket messages to the front-ends. In order to prevent unintended interferences, instead of allowing users to propose their own ports, we ask the server to assign unused ports when a performance is initialized.

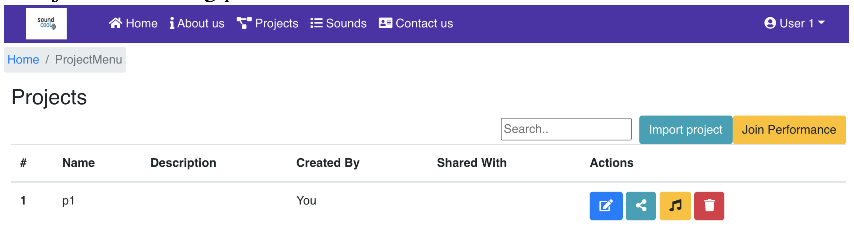
According to (10), range 49152 to 65535 contains dynamic or private ports that cannot be registered with IANA. The port numbers we assign are chosen from this range. A list of opened port numbers is maintained in the backend, and random choices are generated from the unused numbers. There are approximately 16K ports in this range, meaning that we can support more than 1K performances happening simultaneously, assuming each performance has 16 modules to control. Given the current scale of applications and users, we consider this to be sufficient, at least until we are wildly successful. Before we run out of ports, we would probably redirect users to servers in multiple regions, and each server would have a different IP address with an additional 16K ports to allocate.

Short Port Numbering Given the alpha release state of our application, we are adopting a short port numbering system: The port numbers are allocated in blocks of 100: 50000, 50100, etc. This will make it easier when users configure ports in the App. We can revisit this decision if the application develops to a larger scale.

4.3.2 Collision of Names

As described in Section 4.1, the sharing of performances is global, meaning that a performance is not restricted to a specific account. Everyone can join with the name. Thus the performance name must be unique. In order to prevent the collision of names, users are prompted to enter a different name when we have a performance with the same name.

Figure 8: Initializing performance: the yellow buttons indicates entries to create a performance from a project or to join an existing performance



#	Name	Description	Created By	Shared With	Actions
1	p1		You		

4.4 Limitations

- In the current implementation, there is no actual restriction on editing performances, and any edits are local only and not seen by others, allowing performances to behave differently even under shared OSC control.
- To achieve shared editing, the original project must be edited and a new performance created, which reassigns port numbers and requires every performer to update their OSC device.
- This implementation will limit the creation of new performances when ports run out, even if most ports are not actually in use. A future system might allow performances to have conflicting ports when all but one are inactive. This will mean that when a performance starts, ports may have to be reassigned and users will have to be notified to update their mobile device settings.
- After a performance is started and controls are changed, and a new user joins the performance, they will essentially open up a new copy of the project, without immediately syncing up with the other performers. Their client end will only be synced up when new OSC messages come in.

5 Implementation

The implementation of *Soundcool Online* alpha release now contains 168 files with over 30,000 lines of code. The majority of the code is written in React Javascript, and a breakdown by language is show in Table 1. The compiled binary executables are roughly 70 megabytes.

5.1 Performance Implementation

The performance implementation is based on a performance module that is added as a React component. Approximately 3,000 lines were added, and over 5,000 lines were modified, including modifications of frontend components, backend RESTful API, as well as the database structure.

Table 1: Implementation code breakdown by language.

Language	Files	Code
JavaScript React	58	10,427
XML	4	8,905
JavaScript	52	7,078
CSS	6	4,774
SQL	1	101
HTML	1	17

The most important change is the server-client interaction that is implemented via Websockets. As the binary is running as a single React Application on the server, there is no need to open a separate process for each front-end client. Instead, `Node.js` keeps a list of client sockets to which it can send Websocket messages. Similarly, OSC ports were also kept in a list. With the help of `Dgram` library, a UDP server is bound to each port, and messages are forwarded to all client sockets, as shown in Figure 7. Of course, each front-end will filter the messages from sockets and only respond to the one that corresponds to the ports on their modules. In our prototype implementation, we send every incoming parameter update to every client and let the client decide if there is the module with corresponding port, but the system would be much more scalable if the server forwarded updates only to clients participating in the performance.

6 Experiments and Feedback

6.1 Latency

With Open Sound Control, we can set up communication between mobile devices and the server. Latency is critical, and we performed the following latency test:

- Create a project, and connect a mobile phone Soundcool App to a Player module in the project.
- Load a clear click sound into the Player module.
- Tap the play button on the phone to start the sound from the Player module.
- From an audio recording, measure the time between audibly tapping the control and the audio output of the click sound, giving an end-to-end estimation of system latency using OSC control.

Our assumption is that the overall latency measured above can be roughly broken down into four parts, as follows:

- **Touch screen latency**: Latency includes the response time of the mobile device, from tapping the screen until an OSC UDP packet is transmitted. We do not have network measurements, but GameBench reports around 90 ms response times for graphics response on both iPhones and Galaxy devices, as measured with a high-speed camera (blog.gamebench.net/touch-latency-benchmarks-iphone-xs-max-galaxy-note-10). These measurements include game graphics but not message transmission over Wi-Fi, so it is only an approximation for our application.
- **Network latency**: We used the command `ping` to measure the time it takes for network transmission to the Soundcool server, which is currently located in Ohio, US. We take the average round-trip time, and we report measurements from both China (over 11,000 km)

Table 2: the Latency Estimates (ms).

Location	Touch Screen	Network	Web Audio	**Total Measured**
China	90	260	52	**410**
Pittsburgh	90	33	45	**190**
Local	90	6	45	**150**

and Pittsburgh (about 200 km). The "Local" location refers to the Soundcool server running on the same laptop as the browser, and the Network time there is an estimate of OSC over Wi-Fi plus inter-process communication times.

- **Browser and Audio latency**: The time that Web Audio needs to respond to an event with audio output. This is mostly due to audio buffers, but includes front-end scheduling and processing time. To estimate this component, we simply measure the time it takes for a GUI button click to start the player, without going through OSC control. This is measured by recording the mouse click and sound output with a microphone and analyzing the result. Of course, this uses mouse event processing that is not strictly comparable to reacting to an incoming network message, but the major component, audio latency, is the same.
- **Server response latency**: In principle, we could subtract estimates for other time components from the total to estimate server processing time, but we suspect the server time is on the order of a few ms, which cannot be resolved with these measurements.

Although latencies in Table 2 seem high, it should be noted that human response time for critical listening and adjusting faders is on the order of hundreds of milliseconds (11). While it might not be possible to play Soundcool modules like conventional instruments, this delay is reasonable for collaborative performances that involve cuing ambient sounds—but not in rhythm—and adjusting filters, reverberation, gain, and pitch.

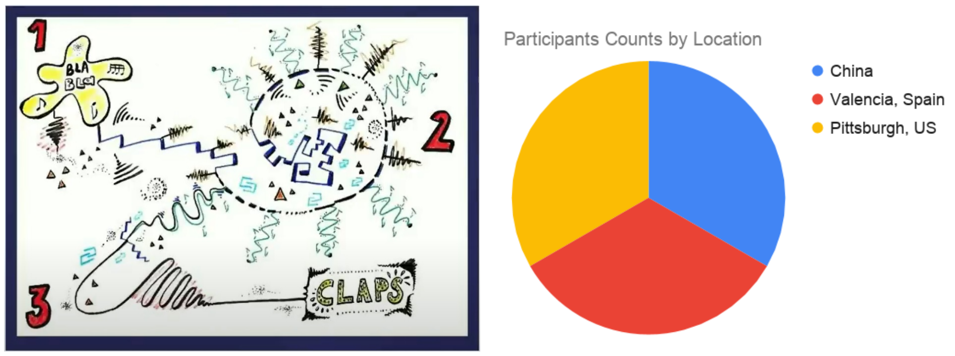


Figure 9: Left: *Mascletà* graphic score; Right: Participants counts by location

6.2 Subjective Evaluations

6.2.1 A Demo Performance

On Apr. 21st, 2021, a interactive performance called *Mascletà* happenend through *Soundcool Online*, with 9 participants physically located in US, Spain and China. The piece is a digital presentation of *Mascletà*, a type of pyrotechnic event that is unique to Valencia. The graphic score of the piece is shown in Figure 9. All participants were asked to log into the same copy of the performance on *Soundcool Online*, where they connect to one assigned module by text. A list of modules and sounds

Table 3: A list of modules and specifications from the Mascletà performance.

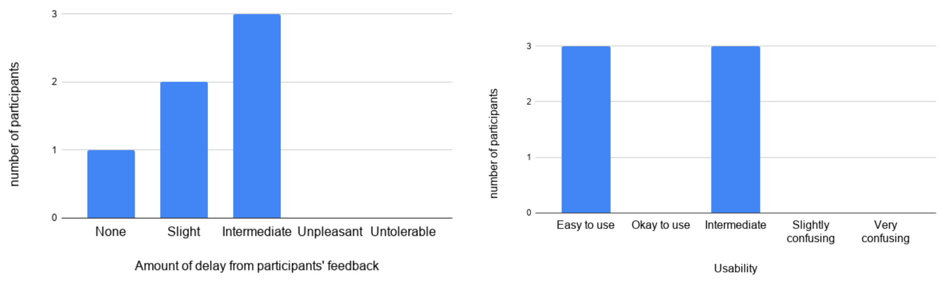
Module	Sound	Connected To
Speaker	N/A	N/A
Mixer	N/A	Speaker
Sample Player	1 - Castell 3	Mixer
	2 - Castell 4	
	3 - Traca 3	
	4 - Traca 4	
	5 - Masclet 7	
	6 - Masclet 8	
	7 - Silbador 4	
	8 - Senyor	
Player 1	Castell 2	Mixer
Player 2	Masclet 3	Mixer
Player 3	Ambient Sound	Mixer
Player 4	Applause	Mixer
Player 5	Silbador 3	Mixer

loaded is listed in Table 3, and the corresponding performance panel is shown in Figure 4. During the performance, the conductor gives performance instructions over a Zoom call for participants to send OSC controls to the server. As each participant loads their own copy of the performance in the browser, there is no need to send audio over Zoom. Zoom audio is muted and participants hear audio directly from their copy of the performance running locally in the browser.

6.2.2 User Feedback

Among the 9 participants, we were able to collect 6 voluntary responses that indicate experience with the performance. Among them we have experienced Soundcool users as well as new users. Their physical location distributions are shown in Figure 9, and Figure 10 showed their feedback on the amount of delay as well as the usability. The statistics show that most users experienced a tolerable delay, which confirms our experiemnts in the previous section. Also, the feedback shows that there is space for improvement in terms of the interface and design of our application.

Figure 10: Left: User feedback on the amount of delay; Right: User feedback on the usability of *Soundcool Online* performance design



Among the text feedback, the most frequent request that we receive is to get rid of the process of user loading sound manually. Sound should be stored as part of the performance and loaded automatically, but this aspect of saving and restoring has not been implemented yet. Also, given the large amount of control messages processed by the server during performances, we have the suggestions to minimize delay by stopping OSC changes from sitting in a queue. The server should limit updates to some rate, and the front-end code should empty the websocket queue to get the latest values and empty the queue before updating any module instead of processing one message at a time. Another suggestion, which is also discussed in the Future Work section, is that mouse-driven controls should be shared too.

7 Future Work

So far, we have implemented the basic functionalities for collaborative performance through server-directed OSC messages. We want to extend performance with more collaborative functionalities. Currently, there are still a lot of aspects of a performance that are not collaborative: Although the same performance rendering in different places can receive the same OSC control update, other parameters like sound loading and audio sharing are still separated. Thus, one future work direction is to achieve total shared control of a performance: When one performer takes control of their module on their client frontend (not through OSC), their changes are also reflected on the other client frontends of the same performance.

8 Summary

In this thesis, we present a new feature for *Soundcool Online*: Performance. *Soundcool Online* is a Web Audio re-implementation of the original Max/MSP implementation of Soundcool, and an overview of motivation and design is provided in Section 3. For the new performance feature, we explained its design and implementation in Sections 4 and 5. Measurements and evaluations are given in Section 6. The performance prototype provides the basic functionality of collaborative performance achieved through Open Sound Control and Websockets, and it is a crucial step that we took towards building an interactive music education system via *Soundcool Online*.

9 Contributions

Soundcool Online is a project with many contributors. The main components are the Web Audio sound processing code for each module, the graphical interface for each module, the overall front-end, the back-end and the new support for performances. Since joining the project, I have contributed:

1. The majority of front-end graphical user interface development through the React and Redux framework, with an emphsis in the core logic of the project editing page and individual module interfaces.
2. Final deployment and maintenance of the alpha release on Amazon AWS server.
3. For this thesis project, I have rewritten the original OSC implementation, database structure, and back-end RESTful API to enable the collaborative performance functionality.

In total, I have added about 3000 lines of code in this project and revised another 5000 lines. In addition, I coauthored the paper (2), which gives an overview of the motivation, design and implementation of *Soundcool Online* before the addition of the performance feature. I also performed latency measurements and experiments demonstrated in the paper.

10 Acknowledgment

I am very grateful to all the contributors in the *Soundcool Online* team, as well as those who helped in organizing my demo performance: (in alphabetical order of the given name) Aditi Kashi, Amit

Kumar Meena, Ankit Joshi, Amey Patel, Diego Villar Sánchez, Jorge Sastre, Manuel Alcañiz, Saúl Moncho and Valeria Herrero. My special thanks goes to Prof. Roger Dannenberg for his invaluable advice and support in this thesis project and for Riccardo Schulz for serving on my committee.

References

[1] S. Scarani, A. Muñoz, J. Serquera, J. Sastre, and R. B. Dannenberg, “Software for interactive and collaborative creation in the classroom and beyond: An overview of the soundcool software,” *Computer Music Journal*, vol. 43, pp. 12–24, Winter 2019.

[2] R. Dannenberg, H. Zhang, A. Meena, A. Joshi, A. K. Patel, and J. Sastre, “(to appear) collaborative music creation and performance with *Soundcool Online*,” in *Web Audio Conference*, 2021.

[3] J. B. Freeman, B. Magerko, D. Edwards, R. Moore, T. McKlin, and A. Xambo, “Earsketch: A steam approach to broadening participation in computer science principles,” in *2015 Research in Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT)*, pp. 1–2, 2015.

[4] R. Shapiro, A. Kelly, M. Ahrens, B. Johnson, H. Politi, and R. Fiebrink, “Tangible distributed computer music for youth,” *Computer Music Journal*, vol. 41, no. 2, pp. 52–68, 2017.

[5] K. Kritsis, M. Bouillon, D. Martín-Albo, C. Acosta, R. Piéchaud, and V. Katsouros, “imuscica: A web platform for science education through music activities,” in *Web Audio Conference WAC-2019* (A. Xambó, S. R. Martín, and G. Roma, eds.), WAC ’19, (Trondheim, Norway), NTNU, 2019.

[6] M. Puckette, “Pure data: another integrated computer music environment,” in *in Proceedings, International Computer Music Conference*, pp. 37–41, 1996.

[7] S. Rudnick, “Synth kitchen,” in *Proceedings of the International Web Audio Conference* (A. Xambó, S. R. Martín, and G. Roma, eds.), WAC ’19, (Trondheim, Norway), p. 143, NTNU, December 2019.

[8] F. Font, G. Roma, and X. Serra, “Freesound technical demo,” in *Proceedings of the 21st ACM International Conference on Multimedia*, MM ’13, (New York, NY, USA), p. 411–412, Association for Computing Machinery, 2013.

[9] J. Sastre and R. Dannenberg, “Soundcool: collaborative sound and visual creation,” in *Sonic Ideas (CMMAS)*, pp. 75–86, 2020.

[10] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire, “Internet assigned numbers authority (iana) procedures for the management of the service name and transport protocol port number registry,” 2011.

[11] A. Jain, R. Bansal, A. Kumar, and K. Singh, “A comparative study of visual and auditory reaction times on the basis of gender and physical activity levels of medical first year students,” *Int J Appl Basic Med Res*, vol. 5, pp. 124–127, May-Aug 2015.