COMPOSITES 1: An Exploration into Real-Time Animated Notation in the Web Browser

Daniel McKemie

Independent Author
daniel.mckemie@gmail.com

Abstract. *COMPOSITES 1 for Modular Synthesizer Soloist and Four Accompanists* is a real-time, graphically notated work for modular synthesizer soloist and four accompaniment parts that utilizes the power of Node.js, WebSockets, Web Audio, and CSS to realize an OS-agnostic and web-deliverable electroacoustic composition that can be accessed on any device with a web browser. This paper details the technology stack used to write and perform the work, including examples of how it is used compositionally and in performance. Recent developments in web browser technology, including the Web Audio API and Document Object Model (DOM) manipulation techniques in vanilla JavaScript, have improved the possibilities for the synchronization of audio and visuals using only the browser itself. This paper also seeks to introduce the reader to the aforementioned technologies, and what benefits might exist in the realization of creative works using this stack, specifically regarding the construction of real-time compositions with interactive graphic notations.

Keywords: JavaScript; Node.js; WebSockets; CSS; Animated Notation; Web Browser; Web Audio API; Mobile Device Music

1 Introduction

During my studies with John Bischoff and Chris Brown [1], I developed my aesthetic of network music that is largely influenced by West Coast Experimentalism. However, it is only recently that have I made serious developments in my own work through the use of web-based technologies, and in particular, the construction and realization of network-based pieces and real-time compositions for live performance. The work of Georg Hajdu, most notably for this context his Quintet.net [2, 3], has served as a great example of contemporary approaches to network music; bringing the traditions of tethered, machine-based network music [1] into the age of mobile devices and wireless capabilities [4, 5].

2 Full Stack Web Technology

JavaScript is the language of the web, and over the last decade it has expanded greatly into server-side tools and technology. The Document Object Model (DOM) is used to create dynamic changes to, and interactions between, HTML elements on the webpage, and can be linked to a number of processes afforded to work in the browser. For the realization of *COMPSOITES 1*, there is a great degree of nested communication in Node.js using WebSockets, with the Web Audio API treating audio data on the front-end, all linked through JavaScript as the primary mode of construction. This architecture is very similar to that implemented in the *Soundworks* framework built by Sébastian Robaszkiewicz and Norbert Schnell at IRCAM [6]. In this section, I will briefly give an overview of these primary components used to write *COMPOSITES 1 for Modular Synthesizer Soloist and Four Accompanists*.

2.1 Web Audio API

The Web Audio API (Application Programming Interface) is a high-level JavaScript API that enables audio synthesis and digital signal processing (DSP) in the browser. The dynamic nature of JavaScript allows the API to be used in conjunction with an array of libraries and tools available in web development, including those on the server side with Node.js. The structure of the API, specifically the modular nature of audio node routing, is similar to other audio software environments in that it provides the user with a large array of options for the synthesis and processing of audio.

2.2 Node.js

Node.js is an open source server environment that allows JavaScript to run on the server and return content to the client. This is used to power data flow from back end databases all the way to the browser page using a single language. Global modules, variables, and functionalities can be spread across multiple pages in the server and can be used with a multitude of frameworks to maximize the efficient construction of an API. In this case, the popular framework Express was used as it is lightweight and wide support.

2.3 WebSockets/Socket.IO

WebSocket technology allows for real-time data transfer to and from the server, without the need to refresh the webpage. This enables the manipulation of back end data through client activity, the broadcasting of unique front-end HTML/CSS stylings to multiple devices or individuals, and the projection of real-time manipulations of said stylings to multiple URLs.

Socket.IO is a client and server-side library that uses WebSockets, with some added features that can open thousands of connections vs other methods [7]. Socket.IO is optimized to implement real-time binary streaming, and for this reason was chosen as the best option for the needs of this project.

3 COMPOSITES 1

In the musical work *COMPOSITES 1 for Modular Synthesizer Soloist and Four Accompanists*[1], there is one host point (the soloist) who goes to the homepage/server site, hooks a modular synthesizer in to be the performance instrument, and assists in score generation. Each of the four accompanying parts can be assigned to any pitched instruments and can be augmented with electronics if desired. The soloist is instructed to send out a simple waveform from the synthesizer, with sine, triangle, and sawtooth working best in that order, into the computer (via an audio interface) which will then send the signal to the web browser to be pitch-tracked. The resultant data of the pitch will be used to generate the notational material for the accompanists. While this waveform is uninterrupted in its signal path directly to the server site, the soloist is to construct a performance patch around that patch point before it reaches the server. The performer may modulate that waveform's frequency but is advised not

[1] Demonstration video: https://www.danielmckemie.com/composites1

modulate too heavily: a "cleaner" tone, such as a sine wave, will lead to a more legible performance score for the accompanists.
The frequency is tracked, and certain frequency thresholds lead to the generation of visuals and/or "decisions" to change the existing visuals. The visuals include color changes, projected pitches on the staff, shapes, shade contours, and so on. These visuals are broadcast to individual URLs, and each of the four accompanists can then access on their own device. For the sake of brevity, I will only discuss examples of one accompanying part, and the technology used to create the server and client pages.

3.1 File Structure

The file directory is set up with client-side scripts in the public folder that are sent to the browser screen which is nested inside the working directory of the Node.js and Socket scripts. Beginning with the client-side calls, there is an HTML and JavaScript file for the host and each accompanist respectively. The host file holds the connections to the server, sockets, and analysis code for the incoming audio stream.

3.2 Client-Side Input and Analysis

An audio input stream is created using the Web Audio DAW (Wad) library [8]. Based on the same concept as a guitar tuner, the signal's frequency is tracked, and because the signal is a simple waveform from an electronic source, the stability of tracking is far better than that of an acoustic instrument (Fig. 1). A major benefit of the Web Audio API is that the hardware synchronization is all done globally. The audio configurations in the computer's system preferences are automatically picked up by the browser, which frees the need for any external drivers or OS dependencies.

The code in Fig. 1 calls for the input and runs the signal through the logPitch() function which is a product of the Wad library. This function calls to analyze the frequency of the incoming signal, and also return its corresponding note name as it coincides on the staff.

```
// host.js

let input = new Wad({ source: 'mic' });
let tuner = new Wad.Poly();

// Sets the library to begin tracking input signal info
tuner.setVolume(0);
tuner.add(input);
input.play();
tuner.updatePitch();

// Logs the signal to frequency number and note name
let inputFreq = null;
let inputNote = null;
let logPitch = function() {
  requestAnimationFrame(logPitch);
  inputFreq = tuner.pitch;
  inputNote = tuner.noteName; };
```

Fig. 1. Code to declare input and track frequency. The frequency and note name captured by the logPitch() function is assigned to the global inputFreq and inputNote, to allow for transfer and broadcast to the clients.

4 Client to Server Connection

4.1 Client-Side

After the input, analyses, and assignments on the client-side have been rendered, the WebSockets must connect this information to the server in order for it to be broadcast to the client's webpages. The 'broadcast' button calls for the host data to be emitted via the socket, and back to the server, and this occurs automatically every 20 milliseconds in order to simulate the transmission of real-time data flow. A number of sockets can be implemented to send data, and each are identified with unique IDs created by the user. In the following case 'frequency' is the unique ID associated with the analysis of the incoming audio stream (Fig. 2).

```
// host.js
// A trigger that automates a button click every 20ms
const buttonBroadcast =
document.getElementById('broadcaster');
setInterval(function() {
  buttonBroadcast.click()}, 20);

// Clicking the button sends the pitch data to the server
buttonBroadcast.addEventListener('click', function(e) {
  e.preventDefault();
  socket.emit('frequency', {
buttonBroadcast.addEventListener('click', function(e) {
  e.preventDefault();
  socket.emit('frequency', {
    pitch: inputFreq,
    note: inputNote })
});
```

Fig. 2. The frequency assignment sends the pitch and note keys with the values of the global variables, inputFreq and inputNote, which were declared prior. This sends/emits this data to the server environment via Socket.IO

4.2 Server Side

The server connection to the localhost and the incoming sockets are all housed within the primary application file in Node.js. Following the thread from the client, the socket connections must be written in such a way that the host data can pass to anywhere on the server via Socket.IO. In this case, the data is broadcast to all clients on the server; and while there are options to broadcast to select clients instead of broadcasting all data in one stream (frequency) while avoiding others, it is beyond the scope of this paper, and not necessary for the success of the piece (Fig. 3).

```
// app.js
io.on('connection', function(socket) {
  socket.on('frequency', function(data) {
    io.sockets.emit('frequency', data) });
```

Fig. 3. This code enables global data sharing and accessibility among all the clients but must be called later by specific clients for their own use.

4.3 Return to Client

The data received from the host and brought to the server, can now be pulled back through to any of the four remaining clients (accompanists) through their respective script.js files. In order to see unique transmissions and treatments of this data, a route must be set up in our app.js file so that a connector can access for their own broadcast (Fig. 4).

The JavaScript calls the data emitted by Socket.IO and can be treated in any number of ways. In this example, the frequency number is run through a switch statement, and as certain conditionals are met, the background color of our <body> is changed accordingly (Fig. 5). This data can be sent to clients in many fashions, from private messages to select clients, and so on, but for the sake of brevity we will emit data to all clients equally.

```
// app.js
// Piping the pitch data back up to the client-side page
app.get('/player1', function(req, res) {
  res.sendFile(__dirname + '/public/player1.html');
});

// script1.js
// As the data is read, a connection is made between the
// background color of the web page and the pitch
// material. A switch statement changes the colors
// according to frequency range

let backColor = null;
socket.on('frequency', function(data) {
  switch (true) {
    case (data.pitch < '200'):
      backColor = 'red';
      break;
    case (data.pitch < '300'):
      backColor = 'orange';
      break;
   };
  document.body.style.backgroundColor = backColor });
```

Fig. 4. The app.js file sends our player1.html file to the browser, which is linked to our script1.js file, which can treat the data sent out to the server through WebSockets.

5 Use as a Compositional Tool

The browser houses a healthy number of ways to compose, manipulate, and animate visual elements, and CSS and DOM manipulation techniques alone include a large number of options to create scores that convey musical information to performers. As outlined in previous sections, these elements can be synchronized through the server and delivered to individually unique locations. The following will outline a number of examples as to how *COMPOSITES 1* uses these techniques to create a real-time score for the accompanying parts.

The elements of the score include the background color, shapes, words and staff notation. Changes to these elements are based on the host input's frequency and/or note name values. The goal of the work was to not inundate the performers with instructions and elements, but rather let them choose pathways in which to realize musical material. This was to lessen the problems that can arise when constructing notation in real-time. [9]

5.1 Colors

The use of color as a notational element has seen more traction in recent years, but no consistent practice has been established. Lindsay Vickery's research into the topic suggests the need for further exploration into the field, especially in regard to multimedia works [10]. As opposed to leaving the interpretation of colors open or to be designated for assignment, the decision was made to give the performer overall qualities to enhance the stylistic qualities of their playing. A classic and notable example of color usage is that of John Cage's Aria (1958), in which he uses color to denote different singing styles for each line [11]. I wanted to take a similar approach in *COMPOSITES 1*, to denote a type of executable action in response to color as opposed to a specific executable action. The colors of the spectrum are at play, and call for the following:

- Red: Poignant
- Orange: Fleeting
- Yellow: Bright
- Green: Stable
- Blue: Metastatic
- Indigo: Dark
- Violet: Razor-like

```
"E4"
"E4"
336 "E4"
```

Fig. 5. Example of the frequency to color relationship as seen through the host's console (right) and Player 1's yellow background (left).

5.2 Shapes

Shapes have been a centerpiece in the evolution of western music notation and maintain an important role in certain brands of music education and vocal music [12]. While I did not seek to morph and modulate the current staff notation system, I did want to use basic shapes to convey a musical response, but not have any shapes that were simply a one-dimensional linear contour. This decision stems from not wanting to encourage a correlation of pitch with vertical placement [10], though if players wish to interpret the two-dimensional shapes this way, that is quite acceptable. With the circle, triangle, and square, the instruction to the performer is to assign a very focused action that best reflects the given shape and execute it regardless of surrounding context. Simple shapes were chosen as they can be easily recognizable compared to the others, and should result in a more focused performance [13]. This achieves two desired goals, the first being a consistent execution of sonic events that will occur throughout the piece; and the second, to allow the performer to have more unrestricted decision making in the performance process. The shapes are called by a change in the <*img*> tag source path and are set to appear when specific parameters are met.

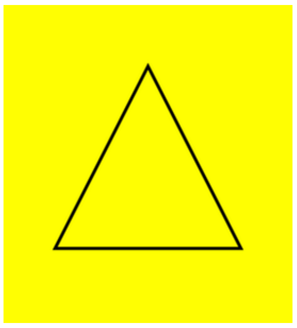

Fig. 6. The triangle shape used in COMPOSITES 1 (one of three shapes to appear), with the yellow background.

5.3 Words

The use of written words to enhance musical expression and performance has been around for centuries. In the case of *COMPOSITES 1*, instead of indicating musical results that could be achieved by using traditional markings (ie. *ff* instead of 'very loud' or a crescendo instead of 'get louder'), I sought to explore the use of words to allow for variations upon the already present situation. To expand, when 'pointed' is displayed, this can result in a number of different outcomes based on both decisions made by the performer, and the context in which it appears (Fig. 7).

The words are generated independently from the host's incoming audio signal by a random number generator and a switch statement assigned to the DOM. Inserting one independently-timed element that is separate from the host's actions gives the accompanying part its own pace. This decision grew out of a desire to eliminate any sort of perceptible rigidity in the performing group as a whole, without having completely asynchronous events.

Additionally, the use of simple, non-musical phrases enhances the space that the performer can work in, and at the same time not be overbearing. The words and shapes are the only elements of the score that appear and are then removed as opposed to remaining stationary and changing over time.

5.4 Pitch/Staff Notation

The images of the staff are the most straightforward of all the elements in the score. As the note changes, the performer uses it as a pitch reference to construct their material, and they also have the option to ignore any number of other elements and continue holding that pitch. The note indicated on the staff is directly correlated to the note name of the incoming waveform.

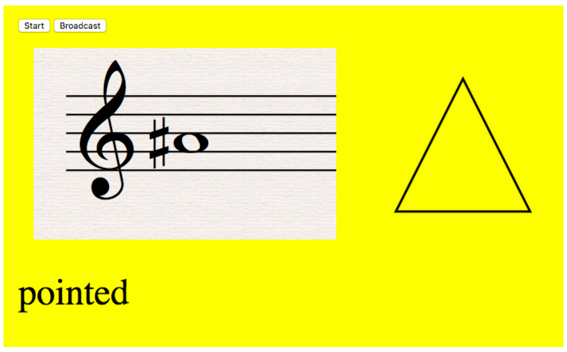

Fig. 7. An example of all elements displayed on one page as seen by an accompanying performer.

5.5 Goals and Challenges

As stated earlier, the resulting notation structures are designed to produce varying interpretations by the accompanying performers. The instructions provided encourage the performer to choose pathways and focus on or ignore elements as they choose. Granting the performers this flexibility helps to relieve any desire, whether purposeful or accidental, to simply follow the soloist. The use of colors, shapes, words, and traditional staff notation provides varying degrees of openness to closedness, and players can find their own level of comfort in the notational structures provided.

The chaotic nature of analyzing audio signals for pitch detection becomes potentially problematic for the stability of the display of accompaniment scores. For example, an input signal that accidentally becomes frequency-modulated would in turn send the accompanists' notation screens into a tail spin, with elements rapidly changing well beyond any realistic readability. Instead of leaving this open to cause an otherwise well executed performance to be ruined, this has been dealt with in two ways: firstly, the soloist is instructed to extensively practice with their synthesizer and this setup and to take note of reactions based on their actions; and second, if the notation system were to fall into chaos, accompanying performers are informed to do the same. Instead of treating this as a hinderance, the inherent fragility in the technology becomes an interesting musical element when executed properly.

6 Future Work

The ever-expanding collection of libraries and development tools can be seamlessly integrated into and inspire works like *COMPOSITES 1*. More involved data visualizations can be built using libraries like D3 [14], and the sharing of data over common networks to control and manipulate individual Web Audio streams could lead to very interesting results, as it did in similar settings that came before it [1].

The back-end server is what supports the delivery of the parts to individual screens for each player and allows for a greater depth of concentration and execution of each player's part for a more effective performance. Taking inspiration from Kelly Michael Fox's *Accretion* [15], which uses individual monitors as opposed to a projected screen in which all players read from, not only maximizes the efficiency of the space in which to render elements, but also injects a sense of mystery for the audience: what are they all looking at?

The fact that the projected parts can be accessed via a multitude of different devices, so long as they support modern web browsers, is key in broadening accessibility and ease of performance of works such as *COMPOSITES 1* and those that choose to employ the same architecture. As stated earlier, the Web Audio API uses the global settings of the computer, which allows for a wider range of device types to be used, and the option to deploy the entire piece as a cloud-based app [16], removes the need to deliver code in any form, requiring nothing more than a simple URL.

7 References

1. Bischoff, J., Brown C.: Indigenous to the Net: Early Network Music Bands in the San Francisco Bay Area, http://crossfade.walkerart.org/brownbischoff/IndigenoustotheNetPrint.html (2002)
2. Hajdu, G.: Quintet.net: An Environment for Composing and Performing Music on the Internet. In: Leonardo, vol. 38, num. 1, pp. 23-30 (2005)
3. Hajdu, G.: Real-time Composition and Notation in Network Music Environments. In: Proceedings of the International Computer Music Conference. Belfast, N. Ireland (2008)
4. Hajdu, G.: Composing for Networks. In: Proceedings of the Symposium for Laptop Ensembles and Orchestras, pp. 98-102. Baton Rouge, Louisiana, USA (2012)
5. Carey, B.: SpectraScore VR: Networkable virtual reality software tools for real-time composition and performance. In: International Conference on New Interfaces for Musical Expression, pp. 3-4. Brisbane, Australia (2016)

6. Robaszkiewicz S., Schnell N.: Soundworks – A playground for artists and developers to create collaborative mobile web performances. In: 1st Web Audio Conference. Paris, France (2015)

7. Carey, B., Hajdu, G.: Netscore: An Image Server/Client Package for Transmitting Notated Music to Browser and Virtual Reality Interfaces. In: The International Conference on Technologies for Music Notation and Representation, pp. 151-156. Cambridge, UK (2016)

8. Serota, R.: Web Audio Library, https://github.com/rserota/wad

9. Freeman, J.: Extreme Sight-Reading, Mediated Expression, and Audience Participation: Real-Time Music Notation in Live Performance. Computer Music Journal, vol. 32 no. 3, pp. 25-41 (2008)

10. Vickery, L.: Some Approaches to Representing Sound with Colour and Shape. In: The International Conference on Technologies for Music Notation and Representation, pp. 165-173. Montreal, Canada (2018)

11. Poast, M.: Visual Color Notation for Musical Expression. Leonardo, vol. 33, no. 3, pp. 215-221 (2000)

12. Johnson, D.C.: Tradition with Kodály Applications. Kodály Envoy, vol. 35, no. 1, pp. 11-15 (2008)

13. Smith, R.R.: An Atomic Approach to Animated Music Notation. In: The International Conference on Technologies for Music Notation and Representation, pp. 40-48. Paris, France (2015)

14. Data-Driven Documents, https://d3js.org

15. Fox, M. K.: Flexible, Networked Animated Music Notation for Orchestra with the Raspberry Pi. In: TENOR: The International Conference on Technologies for Music Notation and Representation, 104-109. Paris, France (2015)

16. Deploying Node.js Apps on Heroku, https://devcenter.heroku.com/articles/deploying-nodejs