Towards Large-Scale Artistic Practice with Web Technologies

Luis Arandas
Faculdade de Engenharia e Universidade do Porto
Braga Media Arts
luis.arandas@gmail.com

José Alberto Gomes
Portuguese Catholic University
CITAR School of Arts
Braga Media Arts
jagomes@porto.ucp.pt

Rui Penha
Escola Superior de Música e Artes do Espectáculo
INESC-TEC Porto
ruipenha@esmae.ipp.pt

ABSTRACT

In this article, we present a software architecture that explores the technological potential of the web as a programmable interface and as an interpersonal connection point in the artistic practice. This structure exposes the recently proposed Akson audio-visual (AV) environment, also raising a technical evaluation of the technologies and design used to allow the development of both the platform and the network.

1. INTRODUCTION

The main objective of this article is to expose in detail the architecture of a web-based AV environment for artistic collaboration. We propose a centralized way to explore different modes of networked interaction that can be used in distributed artistic practice. This practice is fundamentally focused on computer music and contemporary media art through digital interfaces and is designed to explore the cloud as a medium for communication. This environment provides the artist with various forms of expression in a pre-established network, explores the constraints that exist within it and applies them to digital artistic performance.

The proposed system is programmed to use the capabilities of a Chromium browser. It is allocated on the Heroku[1] [20] infrastructure and uses node.js[2] [26], a JavaScript (JS) runtime built on Chrome's V8 JS engine [15]. We establish two-way communication structures using the library socket.io[3] [19, 34], generate audio signals using the WebAudio API[4] [5] and graphics with the WebGL API[5] [13]. All the connected devices are linked by default, and it is possible to dynamically change between the implemented modes of interaction (see section 6). To interact with the system the user can click on the generated graphics or use the integrated WebMIDI platform.

[1]Heroku https://www.heroku.com/
[2]Node.js https://nodejs.org/en/
[3]Socket.IO library website https://socket.io
[4]WebAudio API https://www.w3.org/TR/webaudio/
[5]WebGL API https://www.khronos.org/webgl/

Web Audio Conference WAC-2019, December 4–6, 2019, Trondheim, Norway.

All software and interface design was developed specifically for this research and in the context of Braga Media Arts[6]. This outcome is also product of a review into collaborative interfaces [4] for AV systems and networked interaction [23, 29].

2. THE SYSTEM

Nowadays, the cloud offers enormous potential as an artistic production space [1, 30]. We reflect not only on the freedom offered by the web as an extension of the human gesture, but also as a multimodal shared musical and visual space.

In this project[7] we contract some issues related to human embodiment and communication efficiency [18, 23] that are present in the development of communication systems. Some of those problems are addressed by much contemporary research in this area and one example of that is the Distributed Immersive Performance (DIP) project [9, 33, 38] working on latency issues for data streaming. We address this issue by adopting some techniques often found in the *live-coding* [25] practice. We base the system on socket communications with multiple events throughout the code, and use them to control the generators on each instance of the system. In this way small pieces of information are sent quickly, using the server as a mediator.

But this is just one of the multiple problems addressed. Other example might be the type of system (i.e., centralized; decentralized) and the necessary characteristics (i.e. dependencies of external libraries; hardware). In the particular case of this investigation we establish a single static server to provide the files, include all dependencies in the source code and instantiate all generators/connections in a predefined way.

When a successful link is established to a device capable of reproducing the platform (i.e. WebGL 2.0 [12]) and having included all dependencies and modules without the need for external links, the platform is structured. These include NPM[8] and the Express[9] [22] project, both used as frameworks for node. Regarding node modules, the source code includes: *portfinder*[10] - a port scanner on the current machine; *osc-js*[11] package - to use the Open Sound Control

[6]http://www.bragamediaarts.com/pt/
[7]https://github.com/luisArandas/akson
[8]NPM https://www.npmjs.com/
[9]Express https://expressjs.com/
[10]*portfinder* https://www.npmjs.com/package/portfinder
[11]*osc-js* https://www.npmjs.com/package/osc-js

[37] (OSC) protocol and to establish UDP connections with software like Pure-Data [27] or SuperCollider [35]; and the *winston*[12] package - a universal logging library that can help deal with runtime problems.

Before starting to build the interface [7], at the same time that the generators and controllers are instantiated, a model of communication between users is established. The established (bidirectional) sockets are defined throughout the project, accompanied by unique ID's, which in turn correspond to functions on the front end. By default, the *socket.broadcast.emit* method is set to send to all connected machines except the sending agent.

This homogeneous connection topology will later serve as a means of defining modes of interaction. Applying patterns and/or condition matrices to the code that is running on the devices, is an approach that allows for a slight and rapid state change on the various bindings. It is then possible to define communication structures in the created network, which can objectively control the scope of the user's action. How the data navigates the network is represented in figure 1 as a diagram.

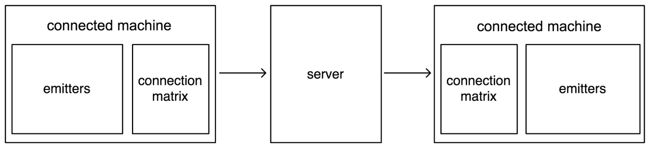

Figure 1: Communication structure between two machines connected to the server. The presented diagram shows that no data emission is performed without first passing through the established interaction matrix and the server.

3. THE INTERFACE

After establishing the connections, the type of server and the dependencies we can start thinking about the development of an interface as a mediator of the experience [32]. Of course, there are several options to be taken when it comes to developing interfaces (graphical or not), and we have decided to expose its existence by encapsulating the technical conditions of this system. Respecting the characteristics of this project as an AV environment we decided to develop two types of layers. A graphical user interface (GUI) composed by widgets, sliders and panels and also a structure that allows interacting directly in the graphical scene (see section 4/5).

This includes some JS dependencies such as: jQuery[13] - for the use of its functions for document processing; The NexusUI[14] project [2] - an open-source collection of HTML5 interfaces and JS helper functions to assist with building web audio instruments in the browser; the WUI[15] project - an easy to use and lightweight collection of widgets for the web browser using vanilla JS with no dependencies; And some adapted buttons from the Bootstrap[16] collection. A responsive and popular component library for quick prototyping.

[12]*winston* https://www.npmjs.com/package/winston
[13]jQuery https://jquery.com
[14]Nexus http://nexus-js.github.io/ui/
[15]WUI https://github.com/grz0zrg/wui
[16]Bootstrap https://getbootstrap.com

We have also included a way to communicate with external peripherals via the MIDI protocol. We have developed a platform for data reception and processing using Web-MIDI API[17] [5] instances and a system for data emission via OSC protocol. As a web protocol that allows controllers and general electronics to communicate and synchronize through MIDI, there are some libraries that present dynamic learning systems, often found in digital audio workstations (DAW). The *SimpleMidiInput.js*[18] library does an abstraction over the input *navigator.requestMIDIAccess* and we integrated it alongside *WebMidi.js*[19], for data processing.

4. AUDIO ENGINE

As previously mentioned, we built the system by offering communications with small portions of information, somewhat similar to the *a.bel* system [10] (i.e., method properties, or interface changes). This is a structure that was thought to be able to control large numbers of distributed interfaces without the need of an enormous computational power also inferred by latency.

Together with native WebAudio, we also explore the *tone.js*[20] [21] library. It has a set of abstractions that allow the creation of generation instances, scheduling and the use of processing effects. We have linked the Nexus widgets with various methods of audio generators by assigning the corresponding IDs to back-end sockets, also with an instance of the MediaStream Recording API for direct registration on each connected instance, making it easy to document the collective performance.

By default, the way to make music is by playing notes using the graphics system. By interacting with the objects on the page, notes are generated within a scale. In order to facilitate the gestures between instances, several scales have been created, encompassing three octaves. These scales can be found in the class entitled *ScalesPlaying*.

```
var scaleMatrix = new ScalePlaying();
var scale = scaleMatrix.cMajorPentatonic();
var note = Math.floor(Math.random() *
scale.length);
Tone.context.resume().then(() => {
currentSynthesizer.triggerAttackRelease(
    scale[note], "4n");});
```

This piece of code provides a way to instantiate the class and play a note within it. The *Math* object is used to perform a random operation within the array and is called a generator method. By default, all instances start in major pentatonic, and all class scales are arranged in C to facilitate phrase coordination between devices.

The group of scales that we can choose from is composed of: *cMajorPentatonic()*; *cMinorPentatonic()*; *cMajor()*; *cMinor()*; *cHarmonicMinor()*; *cMelodicMinor()*; *cAdonaiMalakh()*; *cHungarianMajor()*; *cHirajoshiJapan()*; *cIonian()* and *cLocrian()*. The way this is implemented is simply based on a dynamic way of writing to the currently used array. In the following example we instantiate a list return.

[17]WebMIDI https://webaudio.github.io/web-midi-api/
[18]SimpleMidiInput.js https://github.com/kchapelier/SimpleMidiInput.js
[19]WebMidi.js https://github.com/djipco/webmidi
[20]Tone.js https://tonejs.github.io

```
cHarmonicMinor() {
    return ['C2', 'D2', 'D#2', 'F2', 'G2',
        'G#2', 'B2', 'C3', 'D3', 'D#3', 'F3
        ', 'G3', 'G#3', 'B3', 'C4', 'D4', '
        D#4', 'F4', 'G4', 'G#4', 'B4', 'C5
        ', 'D5', 'D#5', 'F5', 'G5', 'G#5',
        'B5'];
}
```

As described above, these notes are called randomly within the scale. The software has no structure to help formalize musical structures (i.e., stochastic modelling or any other probability theory implementation [28]), often making randomness a problem. The way to choose notes by hand in a graphical interface is sometimes not enough when you want to create precise coordination between distant players. A popular and easy to implement example multiple times found in automatic music systems are the Markov chains. The *js-markov*[21] package is a recent example of that mathematical implementation, and also the *foswig.js*[22] library that works with text can be used for symbolic content generation.

5. GRAPHICS ENGINE

The graphic system is conceptually built with some similarities to the previous one, using WebGL. It also explores the generation of graphics in fixed or mobile devices using the browser, exchanging small amounts of information such as mouse clicks when interacting with the generated content.

It uses the *three.js*[23] [14] library and its custom rendering system alongside GLSL [31] shaders for post-production. We use it to pose the methods and characteristics of the renderer, instantiate the geometries and define the raycaster which is the main algorithm used for graphical interaction (instantaneously used by synthesizer).

This system is constructed to seek a synesthetic coherence between the graphical interaction and the sound generation, from the moment the artist's action occurs [8, 11]. It is possible to explore some kind of feedback between devices when they are interacting, such as color change, which is a result of a successful click. User geometries and actions are also assigned to individual IDs that will be matched on the server. The graphical interface is also populated by the visual system related methods such as camera movements, light control and instantiated geometry attributes.

6. INTERACTION MODELS

As a system focused on large-scale artistic practice, whether defined by a large number of interfaces or by large distances around the world, it is meant to explore the dynamic change of data flows in the same network.

Referencing the study area of human-computer interaction [16, 17, 24, 36] (HCI) on the development of digital collaborative interfaces, there are several options that can be taken regarding the interaction models offered to users. With the proposed architecture we can send *user data* (i.e. browser attributes), *sound data* (i.e. generator attributes; filters), *graphics data* (i.e. geometry properties; color changes) and *system info* (i.e. the machine being used; the position in space). All these atributes are set up in matrices and arranged in such a way that they can be easily manipulated.

The connection matrix will be a validation of conditions before the sockets are transmitted anywhere else, thus allowing for its dynamic change. We then establish relationships between instances [36].

Several network topologies existing in the history of contemporary artistic performance can be implemented, always maintaining a causal relationship between the type of network and the artist's freedom.

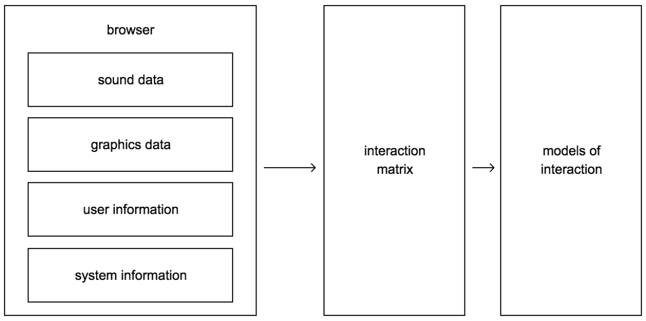

Figure 2: Data flow between browsers used to establish interaction models within the network.

7. SCALABILITY

Collaborative systems used for AV purposes in the cloud tend to require special development when all connected instances have complete freedom of action. Even if the system can (i.e. in terms of computing power) withstand hundreds or even thousands of interfaces (with or without human agents), it is necessary to create dedicated sub-systems when it comes to action management. That number of artists connected at the same time in a singe performance can be interesting as an artistic phenomenon, but it can also be a problem in many ways.

Assuming that the infrastructure is set (i.e. Heroku; Amazon AWS[24]; Microsoft Azure[25]) and using the architecture presented throughout this article, a dynamic count of users and their automatic grouping can be a good starting point.

It is possible to solve some of the problems raised by programming a unified group, internally separated by the various modes of interaction (in sub-groups) maintaining a constant count of entries on the server (log-ins). Offering the possibility to change model to each user (including being alone), with each x number of entries in the system we can dynamically create copies of that group and allocate users there as needed (or if chosen by them). Respecting the rules of communication present in each model, we can also introduce a dedicated console in the front-end showing all these updates in real time, so that it is possible to orchestrate gestures between people and have several performances taking place at the same time separately.

Still, multiplying the various instances automatically by keeping the previously presented rules does not solve interaction freedom. Having a transmutable system can help in optimizing the environment but it is necessary to stress that the various interaction models are finite. There may

[21]js-markov https://www.npmjs.com/package/js-markov
[22]forwig.js https://github.com/mrsharpoblunto/foswig.js
[23]Three.js https://threejs.org/

[24]Amazon AWS https://aws.amazon.com
[25]Microsoft Azure azure.microsoft.com

be some kind of collaborative behaviour model that doesn't exist, which can lead to an easy-to-use preset and back-end control system. By providing this solution, we can have some degree of control when large numbers of people want to use the system from the same domain but do not want to interact with each other.

A public experiment already conducted with the proposed environment took place at the xCoAx international conference (Figure 3) [3]. The stage was occupied by two performers, creating an instance of the proposed system. At the beginning of the concert, it was explained to the audience how they could participate/interact, and that this would be done in a dedicated network.

The agents on stage composed the piece exploring the uncontrolled interaction of the participants for half the duration of the concert, defining what all the connected users could play. In this way the AV content was modeled, without the external participants being able to divert the purpose of this event.

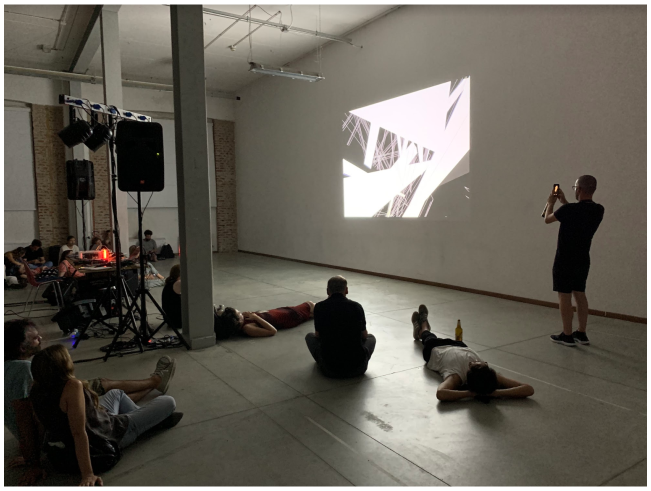

Figure 3: Public performance in the International Conference on Computation, Communication, Aesthetics and X - Milan, Italy.

8. FUTURE WORK

Based on the scalability of this project and the proposed solutions to implement, we've listed some future work. We argue that this project can take different valuable paths both as a networked environment and as an AV platform of free access all over the world.

- As explored in section 7, we believe it is possible to respond to various future needs of this system by programming automation rules that also respect the freedom needed by the agents who are using it.

- Transpose this system into a decentralized architecture (i.e. using blockchain) by creating low-level instances on each device.

- Generation of AV content between machines using stochastic modeling (i.e. VMM's[6]). Allowing the creation of musical phrases between people without deterministic induction.

- A dynamic learning system that allows to create presets on back-end socket matrices to enable new interaction models

- The study of latency focused on data streaming in adaptive systems is always a valuable type of research to increase its robustness.

9. CONCLUSIONS

In this article we have presented a system made with web technologies that emerged from an investigation in computer music, media art and collaborative interfaces. We explore its existence as a digital structure, the technologies it uses, its properties as an AV platform in the cloud, and some steps that can be taken as future development.

Throughout this text, some paradigms inherent to contemporary AV performance are mentioned. The way artists interact is an issue that is mirrored in this project, and the attempt to respond to these needs is what led to use technologies such as node.js and socket.io.

The proposed system allows AV interaction from multiple devices distributed over potentially large distances across the globe, establishing communication matrices between machines.

10. ACKNOWLEDGMENTS

This article reflects and exposes part of an investigation conducted under Braga Media Arts on collaborative AV environments and web technologies. All related contributions are in it's context, part of UNESCO's[26] Creative Cities Network (UCCN)[27].

11. REFERENCES

[1] D. Akoumianakis, G. Ktistakis, G. Vlachakis, P. Zervas, and C. Alexandraki. Collaborative music making as remediated practice. In *2013 18th International Conference on Digital Signal Processing (DSP)*, pages 1–8. IEEE, 2013.

[2] J. T. Allison, Y. Oh, and B. Taylor. Nexus: Collaborative performance for the masses, handling instrument interface distribution through the web. In *NIME*, pages 1–6, 2013.

[3] L. Arandas, J. A. Gomes, R. Penha, and G. Bernardes. Never the less: A performance on networked art. *Proceedings of the xCoAx - Conference on Computation, Communication, Aesthetics X, Milan, Italy - In preparation*, 2019.

[4] Á. Barbosa. Displaced soundscapes: A survey of network systems for music and sonic art creation. *Leonardo Music Journal*, pages 53–59, 2003.

[5] J. Beggs and D. Thede. *Designing web audio.* " O'Reilly Media, Inc.", 2001.

[6] R. Begleiter, R. El-Yaniv, and G. Yona. On prediction using variable order markov models. *Journal of Artificial Intelligence Research*, 22:385–421, 2004.

[7] B. Bongers. Physical interfaces in the electronic arts. *Trends in gestural control of music*, pages 41–70, 2000.

[8] R. Carvalho. From clavilux to ufabulum. *Journal of Science and Technology of the Arts*, 5(1):43–52, 2013.

[26]UNESCO's European Website https://en.unesco.org
[27]UCCN https://en.unesco.org/creative-cities/home

[9] E. Chew, R. Zimmermann, A. A. Sawchuk, C. Kyriakakis, C. Papadopoulos, A. François, G. Kim, A. Rizzo, and A. Volk. Musical interaction at a distance: Distributed immersive performance. In *Proceedings of the MusicNetwork Fourth Open Workshop on Integration of Music in Multimedia Applications*, pages 15–16, 2004.

[10] A. Clément, F. Ribeiro, R. Rodrigues, and R. Penha. Bridging the gap between performers and the audience using networked smartphones: the a.bel system. In *Proceedings of the International Conference on Live Interfaces*, 2016.

[11] N. Correia et al. *Interactive audiovisual objects*. School of Arts, Design and Architecture, 2013.

[12] P. Cozzi. *WebGL insights*. AK Peters/CRC Press, 2015.

[13] B. Danchilla. *Beginning WebGL for HTML5*. Apress, 2012.

[14] J. Dirksen. *Learning Three.js: the JavaScript 3D library for WebGL*. Packt Publishing Ltd, 2013.

[15] J. Gray. Google chrome: the making of a cross-platform browser. *Linux Journal*, 2009(185):1, 2009.

[16] A. Henderson. Interaction design: beyond human-computer interaction. *Ubiquity*, 2002(March):6, 2002.

[17] S. Holland, T. Mudd, K. Wilkie-McKenna, A. McPherson, and M. M. Wanderley. *New Directions in Music and Human-Computer Interaction*. Springer, 2019.

[18] Z. Jin, R. Oda, A. Finkelstein, and R. Fiebrink. Mallo: A distributed, synchronized instrument for internet music performance. In *Proceedings of the international conference on new interfaces for musical expression (NIME)*, 2015.

[19] L. Kalita. Socket programming. *International Journal of Computer Science and Information Technologies*, 5(3):4802–4807, 2014.

[20] C. Kemp and B. Gyger. *Professional Heroku Programming*. John Wiley & Sons, 2013.

[21] Y. Mann. Interactive music with tone. js. In *Proceedings of the 1st annual Web Audio Conference*. Citeseer, 2015.

[22] A. Mardan. *Express.js Guide: The Comprehensive Book on Express.js*. CreateSpace Independent Publishing Platform, 2013.

[23] C. McKinney. *Collaboration and embodiment in networked music interfaces for live performance*. PhD thesis, University of Sussex, 2016.

[24] R. Mills. Telematics, art and the evolution of networked music performance. In *Tele-Improvisation: Intercultural Interaction in the Online Global Music Jam Session*, pages 21–57. Springer, 2019.

[25] C. Nilson. Live coding practice. In *Proceedings of the 7th international conference on New interfaces for musical expression*, pages 112–117. ACM, 2007.

[26] C. R. Pereira. *Aplicações web real-time com Node. js*. Editora Casa do Código, 2014.

[27] M. Puckette et al. Pure data: another integrated computer music environment. *Proceedings of the second intercollege computer music concerts*, pages 37–41, 1996.

[28] L. R. Rabiner and B.-H. Juang. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986.

[29] S. Rafaeli and F. Sudweeks. Networked interactivity. *Journal of computer-mediated communication*, 2(4):JCMC243, 1997.

[30] D. B. Ramsay and J. A. Paradiso. Grouploop: a collaborative, network-enabled audio feedback instrument. In *NIME*, pages 251–254, 2015.

[31] R. J. Rost, B. Licea-Kane, D. Ginsburg, J. Kessenich, B. Lichtenbelt, H. Malan, and M. Weiblen. *OpenGL shading language*. Pearson Education, 2009.

[32] C. Sá. *What is an Interface? The Entification and Identification of the Interface as a Mediation Complex*. PhD thesis, Catholic University of Porto, 2011.

[33] G. Weinberg. Interconnected musical networks: Toward a theoretical framework. *Computer Music Journal*, 29(2):23–39, 2005.

[34] M. Williams, C. Benfield, B. Warner, M. Zadka, D. Mitchell, K. Samuel, and P. Tardy. Push data to browsers and micro-services with websocket. In *Expert Twisted*, pages 285–304. Springer, 2019.

[35] S. Wilson, D. Cottle, and N. Collins. *The SuperCollider Book*. The MIT Press, 2011.

[36] T. Winkler. *Composing interactive music: techniques and ideas using Max*. MIT press, 1998.

[37] M. Wright. Open sound control: an enabling technology for musical networking. *Organised Sound*, 10(3):193–200, 2005.

[38] R. Zimmermann, E. Chew, S. A. Ay, and M. Pawar. Distributed musical performances: Architecture and stream management. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 4(2):14, 2008.