Soundworks
A Framework for Networked Music Systems on the Web

State of Affairs and New Developments

Benjamin Matuszewski
CICM/musidance EA1572, Université Paris 8
STMS Ircam-CNRS-Sorbonne Université
Paris, France
benjamin.matuszewski@ircam.fr

ABSTRACT

This paper presents a novel major version of `soundworks`, a framework dedicated at developing distributed multimedia applications on the web and entirely written in `javascript`. Since its first release in 2015, the framework has served as a basis for numerous artistic and research projects such as concerts, installations, workshops, teaching or experimental setups. These diverse use cases and situations permitted to validate numerous aspects of the framework but also showed some limitations—particularly in terms of inclusion of non-expert developers such as artists and researchers—leading to the novel version presented here.

The paper first presents some applications developed in the last year and show that, despite their idiosyncrasies, recurring problems have emerged during their elaboration and development (e.g. state-management). Second, we present new design and implementation aspects of the framework developed to overcome these issues. Finally we describe a simple testbed application—designed to summarize a number of recurring features and constraints encountered in Network Music Systems—and some elements of its implementation within `soundworks`.

We believe that this novel version will provide solid foundations for the design and implementation of higher-level tools dedicated to non-expert developers, and thereby, foster new artistic, technological and epistemic areas. The `soundworks` framework is open-source and released under BSD-3-Clause license.

1. INTRODUCTION

The recent specification and development of the WebAudio API—together with other APIs such as WebSockets [17] and the developments of ubiquitous computing [16] with smartphones and nano-computers—enabled novel possibilities in the area of Networked Music Systems. These novel tools, alongside with the possibilities offered by a full-featured and interactivity-centered language such as `javascript` permit to envision these technologies from several points of view. First, they can be considered as a new development and a natural extension in the long history of multi-source electro-acoustic music. [13] Second, they can provide a novel platform for composition and performance. [3] Third, they enable a wide range of possibilities in the creation of new interfaces for musical expression. [8] In all cases, these trends tends to show that these technologies—with their simplicity, ubiquity and inherent networked nature—can play a central role in the evolution of Networked Music Systems. [2, 15]

In this context, the development of a dedicated framework, adapted to and designed for the specificities of the web platform is essential. In the last years, a number of such frameworks have been proposed by the community. [1, 7] For now, however, these solutions are far from the maturity of environments such as *Max/MSP* or *PureData*. While these environment have developed a number of key concepts along the years, [9, 10] we think however that adapting these concepts directly to the web [1, 2] tends to neglect the main specificity of the platform (and the novel possibilities it unfolds): the *network*.

`soundworks`[3,4] is a framework dedicated to the development of distributed multimedia applications on the web. The initial version of the framework, released in 2015, has been written by S. Robaszkiewicz and N. Schnell. [11] Since then, the framework has known two major revisions (in 2016 and 2017) and has served as a basis for numerous artistic and research projects (e.g. concerts, installations, workshops, pedagogical or experimental setups). Thereby, `soundworks` has permitted to explore Network Music Systems in many directions such as: participative performances, use of smartphones as speaker array or as new instruments, measures of movements in collective settings. While these achievements tends to prove the efficacy of the framework considered as an experimental platform, [12] they also permitted to highlight some inherent and recurring difficulties. `soundworks#v3` aims to address some of these difficulties as well as to provide solid foundations upon which environments that facilitate the inclusion and agency of non-expert developers can be built.

In Section 2, we review three recent `soundworks` applications implemented in close collaboration with multiple stakeholders, and present of some of their similarities despite their different aims and goals. In Section 3, we present the main conceptual and technical aspects of `soundworks#v3`, designed to provide a better support to the recurring needs described in 2. Finally, in Section 4, we describe a simple, yet not trivial, application that we consider representative of recurring aspects of Networked Music Systems, and shortly present elements of its implementation using `soundworks`.

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

[1] https://github.com/petervdn/webaudiotool
[2] https://github.com/Fr0stbyteR/webaudio-patcher
[3] https://github.com/collective-soundworks/soundworks
[4] https://www.npmjs.com/org/soundworks

2. ACHIEVEMENTS AND RATIONALES

In this section we shortly present three applications that has been designed and used in several contexts (e.g. workshops, performances, installations, scientific experiments) in the last year. These descriptions focus on the features and strategies implemented to support the needs, agency and workflows of non-expert developer users (e.g. artist, researcher, performers) in working situations. We conclude with a formalization of these recurring patterns, leading to the developments presented in section 3.

2.1 Elements

Elements[5] is an application that has been specifically designed to conceive and prototype movement-based distributed Interactive Machine Learning scenarios. [5] The application has been iteratively tested and developed in several contexts such as workshops, artworks and performances[6] or scientific experiments.[7]

The key aspects for the appropriation of the application by non-expert developer users stand in two complementary elements:

- A JSON file dedicated at configuring the different clients in terms of interface, type of synthesis or mapping.
- A controller (see Figure 1, left) that allows for both remote monitoring (e.g. plot sensors, decoding of the ML algorithm) and remote control of each client (e.g. mute, volume).

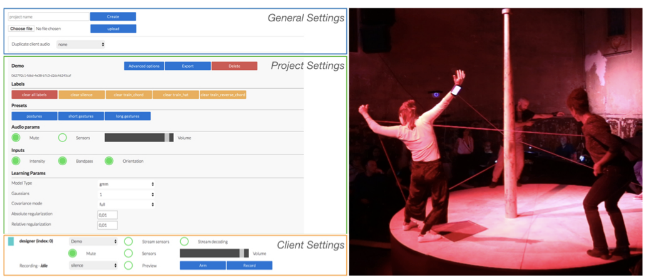

Figure 1: On the left, *Elements*' controller interface highlighting the different possibilities of monitoring and control. On the right, performance of *Cordas* composed by Michelle Agnes Magalhaes.

2.2 Future Perfect

Future Perfect is an immersive 3D audio visual performance and installation work developed by Garth Paine during a residency that took place in 2018 between Ircam and ZKM.[8] The application allows the composer to perform on the audience smartphone's speakers using several dedicated interfaces. Figure 2 shows the composer in performance situation with three iPads as well as screenshots of the different interfaces designed and used for composing and performing.

In this application, many strategies have been implemented to provide the composer a dynamic environment in which he could test sonic material (i.e. dynamic update of sound files, creation of presets), simply configure many aspect of the synthesis (e.g. granular synthesis parameters, fade times), but also have useful feedback on the state of audience's smartphones (e.g. loading states, position in concert hall). Again, the key elements here were the remote monitoring and control interfaces that enabled a rapid feedback loop in both composition and performance situations.

[5] https://como.ircam.fr/apps/elements
[6] https://www.youtube.com/watch?v=c6Flruf_Udc
[7] https://www.unige.ch/cisa/emodemos/
[8] https://www.ircam.fr/person/garth-paine/

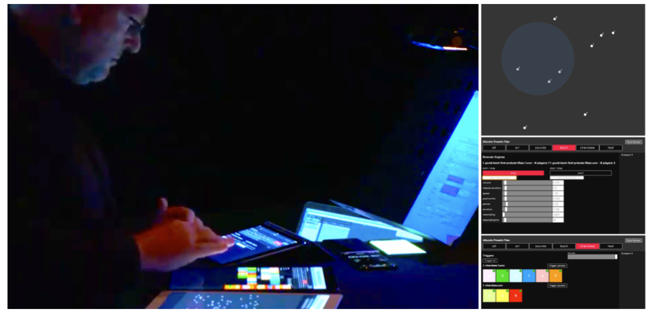

Figure 2: On the left, Garth Paine performing *Future Perfect*. On the right, screenshots of the different interfaces used during composition and performance.

2.3 Biotope

Biotope is an generative and interactive installation composed by Jean-Luc Hervé, realized at Ircam and exposed at the Centre Georges Pompidou, Paris in the context of the exhibition "La fabrique du vivant".[9,10] The installation is composed of 27 *Raspberry Pi* nano-computers (see Figure 3, left) running `soundworks` clients written using `NodeJs`. [4] The audio synthesis is achieved using a `NodeJs` wrapper on top of `libpd`.[11]

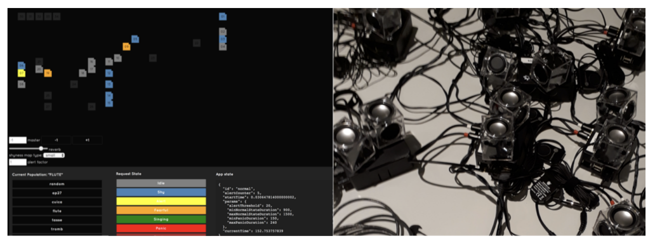

Figure 3: On the left, screenshot of the centralized controller. On the right, photography of the musical agents, running on Raspberry Pi, created for the *Biotope* installation composed by Jean-Luc Hervé.

In this system, many strategies have been implemented to provide a dynamic and testable environment to the composer and to the computer music designer. Among them, the more important ones are: a mean to easily update audio content, and a centralized controller dedicated at both monitoring the state of the application (for example, each square in Figure 3 right, represents a musical agent in its relative position in the exhibition space, the different colors give an overview of their state in real-time) and at controlling the state and parameters of each client in real-time.

2.4 Recurring Patterns

We can see that these different examples—which span across a wide range of applications (from experimental system to performance or installation)—share common strategies. First, they all provide a dedicated client that allows to monitor and take control over every client of the system in a simple way. This point stands to be of primary importance to maintain the agency of the user working in a complex setup composed of many devices. Second, they all allow—at different levels of maturity and usability—to update content, mappings and synthesis parameters dynamically or from configuration files. Indeed, these applications implement a similar architectural pattern where the state of each client is synchronized

[9] https://www.ircam.fr/agenda/biotope/detail/
[10] https://youtu.be/RmSujqdT6L0
[11] https://github.com/ircam-jstools/node-libpd

in some way with the server, allowing to update every part of the distributed application from a centralized point.

This pattern—that appeared very effective for implementing versatile and adaptable tools fostering creativity—provide insights on the functionalities our framework must facilitate. More precisely, it shows the necessity of a robust and versatile distributed state management system, aimed at simplifying *remote monitoring and control* in an environment composed of many devices.

While the presented examples showed the feasibility of creating such systems using the current version of `soundworks`, the experience showed that these aspects were not properly supported by the framework, leading to overly complicated and redundant architectures. This is these drawbacks that the novel version of `soundworks` presented in the next section propose to overcome.

3. DESIGN AND IMPLEMENTATION

In this section, we present some design and implementation aspects of the third version of the `soundworks` framework. First, we present the scope and high-level aspects of the framework. Second, we describe in more depth the novel state management component that have been introduced to support recurring patterns described in Section 2. We briefly conclude by presenting the motivations and expected benefits of the novel packaging and distribution strategy.

3.1 Architecture Overview

Since its inception, `soundworks` is dedicated at simplifying the development of web-based and distributed real-time musical systems. Figure 4 presents a bird's-eye view of a typical `soundworks` application. Applications created using `soundworks` follow a star network architecture centered around a `NodeJs` server. [6] Clients can have multiple responsibilities (e.g. audio rendering, visual rendering, control) and be of different kinds (e.g. mobile, desktop, nano-computers).

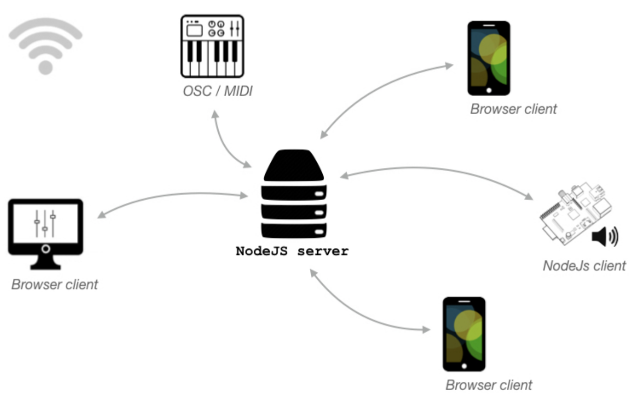

Figure 4: High-level view of the architecture of a typical soundworks application: clients of various types (e.g. mobile and desktop browsers, embedded hardware running a `NodeJs` client, external software communicating through OSC or MIDI) connected to a central `NodeJs` server.

Until now, the framework has mainly focused on mobile applications and has therefore privileged certain caracterictics of these platforms (e.g. graphical user interface, usability). While these aspects remain important, it appears now that—to preserve its efficiency as an experimental platform and to support more and more complex applications and use-cases—the framework must evolve toward more modularity and extensibility, considering both software (e.g. integration of third party components and libraries) and hardware (e.g. integration of IoT elements).

In this objective, the scope of the framework has been refined and narrowed down to focus on three key aspects, namely: *communications*, *service management* and *state management*. As a consequence, some features such as templating, have been removed from the framework and are now delegated to external and specialized libraries. These developments also permitted to reduce the API surface area of the framework.

3.2 Communications and Services

While similar in their principles, the *communication* and *service management* components have evolved toward more simplicity and efficiency. Figure 5 summarizes the initialization process common to all `soundworks` clients:

- The `init` step consists in connecting two WebSockets to the server, one dedicated to string (JSON compliant) data and another dedicated to binary data. The API of both sockets is similar and expose a simple `publish / subscribe` interface.
- Once both sockets are connected, `soundworks` can start the services initialization. As services can depend on each others (for example, the clock synchronization process can rely on a resumed audio context), `soundworks` takes care of the services' dependency graph and start each service accordingly.
- Finally, when all services are in `ready` state the application specific code (called `Experience` in `soundworks`' terminology) can start.

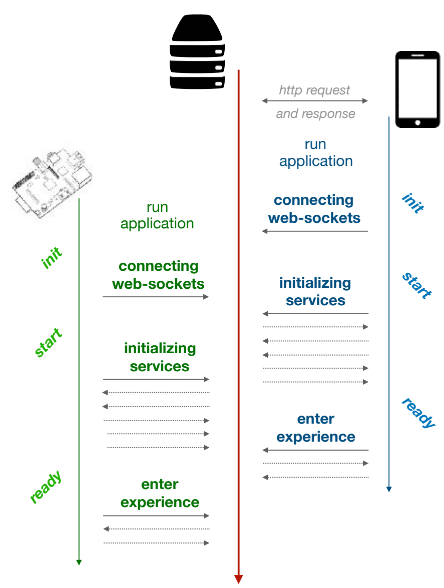

Figure 5: Initialization process of a `soundworks` clients, here a mobile browser and a `NodeJs` client running in embedded hardware.

Figure 5 also illustrates a novel feature of the framework that enables the seamless implementation of `soundworks` clients in any type of `javascript` environment (i.e. browsers or `NodeJs`). Indeed, while this approach has already been tested and deployed in a production setting (cf. 2.3), the complete rewriting of the framework permitted to properly integrate it by making most of the code compatible to both platforms. This novel feature should simplify the creation of applications composed of multiple kind of clients (e.g. smartphones and nano-computers), and thus allow to generalize and democratize the concept of *web of audio things* described in [4] (note that similar ideas has been proposed in [14]).

3.3 State Management

An important novel feature of `soundworks` is the integration of a state management system.

Indeed, since the introduction of the *Flux* pattern proposed by Facebook,[12] a number of state management libraries[13] have been proposed. The usage of this pattern is nowadays widespread and considered a good practice among the `javascript` community. However, existing libraries are not firstly designed for distributed applications and are difficult to adapt to our context for two main reasons. First, they do not formalize nor integrate the notion of discrete and volatile events very common in our applications (e.g. triggering a sound). Second, they do not provide *out-of-the-box* a simple way of synchronizing states across several nodes in the network.[14]

To tackle these issues, we created a novel protocol and implemented a novel component, inspired by the *Flux* pattern and adapted to the particular requirements of our applications.

Concepts and Requirements

In the context of real-time, audio-centered and distributed applications, the application of such circular pattern presents certain particularities schematized in Figure 6.

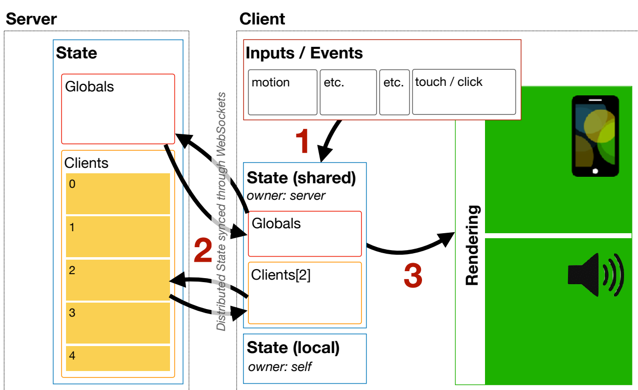

Figure 6: Conceptual overview of a circular and distributed state management system inspired by the *Flux* pattern. The main addition stands in the necessity to keep states synchronized with the server.

The Figure particularly highlights two important requirements and implications for the implementation of this pattern. First, it shows that the state of every client has to be kept synchronized server-side. The rationale for this design strategy (see Section 2 for details) stands in the need to remotely monitor and control any client of the system from a centralized point. Indeed the possibility to dynamically interact with any node of the network, and the rapid feedback loop it enables, is of primary importance in working situations. Furthermore, it appears to be crucial in exploratory contexts (such as artistic and research activities) where the final application cannot be specified beforehand and emerges from a iterative process.

Second, the Figure highlights the need of a certain granularity in the definition and synchronization of the states. Indeed, while some variables and parameters (named `globals` in the Figure) needs to be accessible to every client (e.g. master volume, mute), the particular state a client (`clients[2]` in the Figure) should not be shared with all its peers. It only needs to be monitored or controlled by particular types of clients dedicated to authoring and performance situations.

Protocol and API

To fulfill these requirement while preserving the idea of circular flow between actions, data and rendering proposed by the *Flux* pattern, we designed a simple protocol and implemented a new library.[15] The main principles of the protocol we propose are:

- Allow any node to *create* a new state from a declared *schema*.
- Allow to keep the state *synchronized* with the server.
- Allow any node to *observe* new states created on the network.
- Allow any node to *attach* to a state created by another node.

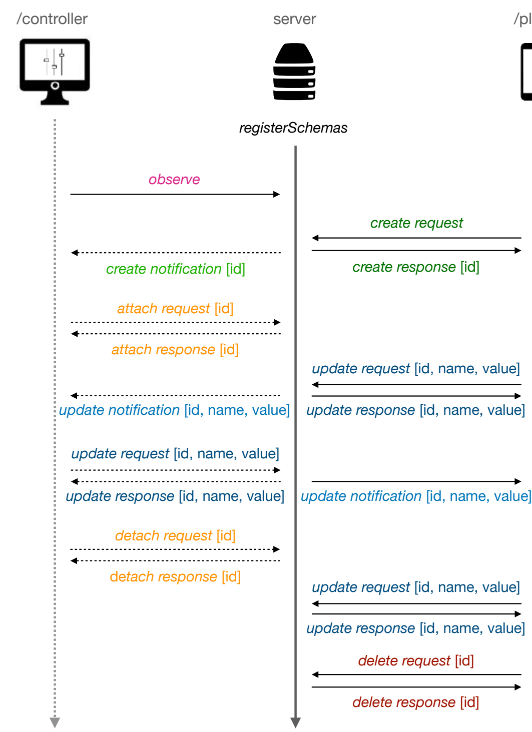

Figure 7: Example of the protocol implemented by the `StateManager`.

Figure 7 illustrates a generic scenario enabled by this protocol. A client (we name `controller`) observes the server and attach to the state created by another client (here, called `player`). When attached, the `controller` receives a notification each time the state is updated by its creator (or any other attached node), enabling *remote monitoring*. The `controller` can also update values of the attached state, enabling *remote control*. At any moment, the `controller` can detach from the state and stop to receive update notifications.

The protocol is abstracted behind a small and simple API illustrated in the pseudo-code example of Listing 8. This simple example also highlights two interesting aspects of the component:

- The complete abstraction of network communications, allowing users to focus on the application logic rather than routing of network messages.

[12] https://facebook.github.io/flux/

[13] For example: https://redux.js.org/ or https://vuex.vuejs.org/

[14] The `dop.js` (https://distributedobjectprotocol.org/) library propose an interesting approach, however it aims at synchronizing a single state across every node which is not optimal (particularly in terms of bandwidth) in our context.

[15] While the component is for now integrated as a first class citizen in `soundworks`, it will be abstracted and released as a standalone library in a near future.

```
// server-side
const synthSchema = {
  volume: { type: 'float', min: -80, max: 6 },
  triggerSynth: { type: 'any', event: true },
};
const manager = new StateManager(server);
manager.registerSchema('synth', synthSchema);

// client-side
const manager = new StateManager(client);
const playerState = await manager.create('synth');
playerState.subscribe(updates => {
  for (let [key, val] of Object.entries(updates)) {
    switch (key) {
      case 'volume':
        mixer.volume = val;
        break;
      case 'triggerSynth':
        synth.trigger();
        break;
    }
  }
});
// later (or from any other attached node)
playerState.set({ volume: -6 });
```

Figure 8: Pseudo-code example of the main aspects of `soundworks` state manager API.

- The possibility to reflect on the schemas' declarations to generate controls and monitoring interfaces, simplifying the implementation of dynamic and complex interfaces.

3.4 Distribution: Core and Services

A final aspect that we want to present is the novel approach for the packaging and the distribution of `soundworks`. The framework is now distributed behind its own `npm` organization namespace: `@soundworks`[16]. Furthermore the `core` of the framework and the different `services` have been decoupled.[17] As such, services are now imported in the application as plugins that must be registered in the `ServiceManager`.

We think this modular approach will facilitate future evolutions of the codebase, as well as maintenance of existing applications. Furthermore, this strategy should help to simplify the design and development of new components as well as their testing and documentation.

4. A *TODO(NOISE)* APPLICATION FOR DISTRIBUTED AUDIO FRAMEWORKS

In this section we describe a simple application, inspired by the *TodoMVC* project[18], that aims at providing a common basis to test and compare frameworks dedicated at building distributed audio applications. We first present the motivations and features of the application and, second, describe elements of its implementation within `soundworks`.

4.1 User Story

The proposed application purposely privileges the point of view of a user in a working situation (i.e. developer, designer, composer or performer) rather than the point of view of the end user (e.g. participant, audience). Indeed, while the later tends to be very application or artwork specific, we have shown in Section 2 that the former embodies common properties—the need for remote monitoring and control of the distributed state of the application—that can be reduced to simple features.

To illustrate these features, we have designed a basic application composed of two different clients. The first client, we call *player*, can be envisioned as the client dedicated to the end users. The application can accept any number of *players*. Each `player` has access to the following fonctionalities:

- can trigger a sound
- can start and stop a synthesizer
- can update a parameter (i.e. volume)

The second client, we call *controller*, is dedicated to the user in working situation (e.g. design, composition, research, performance). The application can accept any number of `controller`. A `controller` can:

- control global parameters of the application (i.e. mute, master volume)
- take control over each `player` (i.e. volume, trigger and state of the synthesizer)

Globals parameters of the application (i.e. *mute* and *master*) must stay synchronized across every clients of the application (i.e. *player* and *controller*).

We think this minimal set of functionalities provides a good reduction of important and recurring aspects of distributed audio applications. We also believe that it could, after eventual refinements, provide a good basis for testing, demonstrating and compare different frameworks and approaches.

4.2 Elements of Implementation

We implemented this application using `soundworks` (see Figure 9).[19] The experience showed possible to implement all the specified features relying on the state management system described in 3.3, allowing to focus on application logic rather than on network communications and routing. This single fact tends to validate the addition of this component to the core of `soundworks`.

Figure 9: Interface of the (a.) `controller` and (b.) `player` clients of the *Todo(Noise)* application. Here, the `controller` duplicates the interface of the `player` with `id 32`, allowing for remote monitoring and control of this particular client.

The application is composed of only two schemas:

- The `globals` schema contains the list of connected `player ids`, the `id` of the remote controlled user (if any), the values of the `mute` and `master` volume parameters.
- The `player` schema contains the current value of the `player`'s local `volume`, the state of the synth (*started* or *stopped*) and a volatile event dedicated at triggering a sound.

[16] https://www.npmjs.com/org/soundworks
[17] https://github.com/collective-soundworks/soundworks
[18] http://todomvc.com/
[19] https://github.com/collective-soundworks/soundworks-todo-noise

The main logic of the application is implemented in the `controller` client. Indeed this client (cf. Listing 10), in its subscription to the `globals` state, observes the value of the `remoteControlled` parameter and `attach` to the state of the corresponding `player`. When attached to the `player` state, the `controller` simply instantiate the `player`'s GUI to locally create a remote and synchronized monitor and control interface.

```
// src/client/controller/ControllerExperience.js
this.globals.subscribe(async (updates) => {
  for (let [key, val] in Object.entries(updates)) {
    if (key === 'remoteControlled') {
      const playerId = val;
      // detach from previous player
      if (this.playerState) {
        await this.playerState.detach();
        this.playerState = null;
      }
      // attach to new remote player
      if (playerId !== null) {
        this.playerState = await stateManager.attach
             ('player', playerId);
        // keep GUI synced with player state
        this.playerState.subscribe(this.render);
        // handle disconnection
        this.playerState.onDetach(() =>
             this.playerState = null);
      }
    }
  }
  this.render();
});
```

Figure 10: Main pseudo-code logic written in the `controller` to remotely monitor and control any `player` of the application.

5. CONCLUSION AND FUTURE WORKS

In this paper, we have presented the motivations, design and implementation of a novel version of `soundworks`, a framework dedicated at developing distributed multimedia applications on the web. First, we have presented three applications implemented or refined in the last year and discussed some of the recurring difficulties—centered on the point of view of the user in working situation—that the current version of the framework failed to properly address. Second, we have presented the novel architecture as well as a new component dedicated to distributed state management and designed to address these recurring issues. Finally, we have described a simple application—designed on the model of the *TodoMVC* project—that summarizes recurring aspects of distributed audio applications, and some elements of its implementation within the new version of `soundworks`.

While we think this new version provides solid foundations to further explore the possibilities of the web platform for Network Music Systems, it also opens large areas of new developments. First, a cli tool for scaffolding applications would be an important addition. Second, the integration of `NodeJs` clients in the core of the framework should simplify testing and thus help to stabilize the framework. Third, and more important, it opens many paths for creating a more dynamic working environment, facilitating the inclusion of users with different backgrounds (e.g. artists, researchers) and transdisciplinary approaches.

6. AKNOWLEDGEMENTS

The presented work has been initiated in the *CoSiMa* research project funded by the french National Research Agency (ANR, ANR-13-CORD- 0010) and further developed in the framework of the *Rapid-Mix* Project from the European Union's*Horizon 2020 research and innovation programme* (H2020-ICT-2014-1, Project ID 644862). It has also been supported by the Ircam project *BeCoMe*, which is featured in the *Constella(c)tions* residency of the STARTS program of the European Commission.

We would like to thank our projects partners and our colleagues at IRCAM for their precious contributions to the project.

7. REFERENCES

[1] J. Allison, Y. Oh, and B. Taylor. NEXUS: Collaborative Performance for the Masses, Handling Instrument Interface Distribution through the Web. In *Proceedings of the NIME'13 Conference*, Daejeon, Seoul, Korea, 2013.

[2] A. Barbosa. Displaced Soundscapes: A Survey of Network Systems for Music and Sonic Art Creation. *Leonardo Music Journal*, 13, Dec. 2003.

[3] J. Bischoff, R. Gold, and J. Horton. Music for an Interactive Network of Microcomputers. *Computer Music Journal*, 2(3), 1978.

[4] B. Matuszewski and F. Bevilacqua. Toward a Web of Audio Things. In *Proceedings of the 15th Sound and Music Computing Conference*, Limassol, Cyprus, 2018.

[5] B. Matuszewski, J. Larralde, and F. Bevilacqua. Designing Movement Driven Audio Applications Using a Web-Based Interactive Machine Learning Toolkit. In *Proceedings of the 4th Web Audio Conference*, Berlin, Germany, 2018.

[6] B. Matuszewski, N. Schnell, and F. Bevilacqua. Interaction Topologies in Mobile-Based Situated Networked Music Systems. *Wireless Communications and Mobile Computing*, 2019, Mar. 2019.

[7] S. Piquemal. Rhizome. https://github.com/sebpiq/rhizome. Accessed: 2019-06-24.

[8] I. Poupyrev, M. J. Lyons, S. Fels, and T. Blaine. New Interfaces for Musical Expression. In *CHI '01 Extended Abstracts on Human Factors in Computing Systems*, 2001.

[9] M. Puckette. The Patcher. In *Proceedings of the International Computer Music Conference*, 1988.

[10] M. Puckette. Combining Event and Signal Processing in the MAX Graphical Programming Environment. *Computer Music Journal*, 15(3), 1991.

[11] S. Robaszkiewicz and N. Schnell. Soundworks – a playground for artists and developers to create collaborative mobile web performances. In *Proceedings of the 1rst Web Audio Conference*, 2015.

[12] M. Schwab. *Experimental Systems: Future Knowledge in Artistic Research*. Orpheus Institute series. Leuven University Press, 2013.

[13] B. Taylor. A History of the Audience as a Speaker Array. In *Proceedings of the NIME'17 Conference*, 2017.

[14] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthet. Internet of Musical Things: Vision and Challenges. *IEEE Access*, 6, 2018.

[15] G. Weinberg. Interconnected Musical Networks: Toward a Theoretical Framework. *Computer Music Journal*, 29(2), June 2005.

[16] M. Weiser. The Computer for the 21 st Century. *Scientific american*, 265(3), 1991.

[17] L. Wyse and S. Subramanian. The viability of the web browser as a computer music platform. *Computer Music Journal*, 37(4), 2013.