

A Web-Based Framework for Distributed Music System Research and Creation

BENJAMIN MATUSZEWSKI

(benjamin.matuszewski@ircam.fr)

CICM/musidance EA1572, Université Paris 8, STMS Ircam-CNRS-Sorbonne Université Paris, France

This paper presents `soundworks`, a framework dedicated to prototyping and developing distributed multimedia applications using Web technologies. Since its first release in 2015, the framework has been used in numerous artistic and research projects such as concerts, installations, workshops, teaching, and experimental setups. We first present how this diversity of contexts and objectives permitted us to identify a set of patterns able to support recurring needs of expert users in exploratory tasks. We then detail new developments that have been achieved to provide better support to these patterns. More particularly, we describe the novel distributed state management system dedicated to simplifying the implementation of remote control and monitoring interfaces and the plug-in system implemented to improve the extensibility of the framework and foster composition of dedicated functionalities. We believe that these new developments can provide a solid ground for further research and artistic practices in the area of distributed music systems. The `soundworks` framework is open source and released under BSD-3-Clause license.

0 INTRODUCTION

The specification and development of the WebAudio Application Programming Interface (API) [1, 2]—alongside APIs such as WebSockets [3] or WebGL [4] and the possibilities offered by a full-featured scripting language such as [5]—has permitted to envision the Web platform [6] as a viable technical platform for artistic creation and more precisely for computer music practices [7]. Furthermore, the recent developments of ubiquitous and pervasive computing [8], with the democratization of smartphones and large spread of nanocomputers, led us to consider Web technologies as a possible solution for recurring integration and interoperability issues [9]. These two complementary aspects therefore authorize us to consider the Web as an interesting environment in the development of Networked Music Systems [10–12]. Moreover, this novel approach could unfold novel possibilities in related areas such as multisource electro-acoustic music [13, 14] or interfaces for musical expression [15, 16]. In this context, the development of a dedicated framework, designed to support both the specificities of the web platform and of computer music research and practices seems essential.

Indeed, computer music is a field that spans across multiple disciplines—from scientific to artistic through social sciences and humanities—and thus gathers a great diversity of goals, skills, and methodologies (e.g., experimental

studies, practices-based research). It appears that a common ground for the support of this diversity can be found in the concept of experimental systems—as systems composed of epistemic things and technical objects in constant evolution and reconfiguration—developed by Rheinberger [17, 18] and pursued by Schwab in the context of artistic research [19]. We postulate that such epistemological ground can lead to the implementation of particular patterns [20] in order to support this diversity of research practices effectively.

`soundworks` [21]—initiated by S. Robaszkiewicz and N. Schnell [22] in 2015—is a framework dedicated to the development of distributed multimedia applications on the web. It has known two major revisions (in 2016 and 2017) and has been used in numerous artistic and research projects (e.g., concerts, installations, workshops, pedagogical or experimental setups) [23–25]. While these achievements tended to validate the efficacy of the framework considered as an experimental platform, they also permitted to highlight some inherent and recurring difficulties. The third version of `soundworks`—initiated in 2019 [26]—presented in this paper aims to address some of these difficulties, as well as to provide solid foundations upon which environments facilitating the inclusion and agency of nonexpert developers users can be built.

After a short review of the related works (cf. Sec. 1), we describe in Sec. 2 different contexts in which our framework

has been successfully used in the last two years and which informs us about recurring and important needs our framework must support. In Sec. 3, we present an overview of the framework architecture, philosophy, and basic functionalities. Finally, in Secs. 4 and 5, we detail new features dedicated to supporting patterns—namely *remote monitoring and control*, *composability*, and *extensibility*—that we consider of primary importance to provide an effective experimental platform supporting distributed music systems research and creation.

1 RELATED WORKS

Max/MSP or *Pure Data* [27, 28] are well-established environments used for many years by artists and researchers in a wide range of contexts. The success of these visual programming environments lies in part in their successful implementation of certain patterns that permitted users to create and compose their own application in a very interactive fashion [25]. However, the environments also come with their drawbacks in our context. First, they are not primarily oriented toward distributed applications and are difficult to operate in large and dynamic networks of computers. Second, they necessitate the installation of software, making applications difficult to distribute and thus to deploy in large collective settings, precluding new forms of public and collective participation.

On the Web platform, attempts have been made to implement equivalent environments [29]. However, while interesting, these tools are far from being as mature as the original ones. Also, they tend to neglect one of the most interesting aspects of using the Web platform (namely, the network). Finally, some frameworks dedicated to network music systems, such as *Rhizome* [30] or *Nexus* [31], have been proposed. While similar to *soundworks* in their scope, these tools do not seem to be maintained or in active development.

2 CONTEXTS

In this section, we review different contexts in which we deployed web-based distributed systems implemented using our framework. Each of these contexts will be illustrated with a particular project that has been developed in the last years. Note that while these 3 applications have been designed and developed with the previous version of *soundworks*, two of them (i.e., *Playground* and *CoMo*) have already been ported to the novel version. These rewrites permitted the simplification of the code base and, moreover, enabled new artistic and research possibilities, assessing thus the concepts and design decisions presented in this paper.

2.1 Concerts and Performances

Playground is an application that allows a composer/performer to remotely distribute and control audio materials rendered on the smartphones of the audience.

The application exposes several dynamic control interfaces, optimized for touch interfaces such as tablets (see Fig. 1), that can be jointly used:

- The first one allows for triggering sound files on a given smartphone, represented on the screen as a colored square.
- The second one allows for controlling granular synthesis among subsets of the audience’s smartphones.
- The third one is dedicated to controlling the spatial rendering of audio files synchronized among all smartphones.
- Finally, the fourth one is dedicated to managing all presets and configuration variables as well as to assigning particular sound banks to the other control interfaces.

In this application, a number of strategies are implemented to provide the composer and performer a dynamic environment in which they can test sonic material and configure many aspects of the synthesis (e.g., dynamic update of sound files, creation of presets) in the studio but also have useful feedback on the state of audiences’ smartphones (e.g., loading states, position in concert hall) during the performance.

Playground has been designed together with the composer Garth Paine and implemented for the creation of *Future Perfect*, an immersive 3D audio visual performance¹. Since then, the application has been used for the creation of several pieces—by the composer himself or other composers—as well as in workshops and pedagogical situations.

2.2 Installations

The context of an installation comes with different constraints and requirements than the ones of performances. In such contexts, the usage of so-called nanocomputers is interesting for several reasons [25], the most important one being the simplicity they offer in term of orchestration and task automation compared to smartphones.

For example, *Biotope* [32], composed by Jean-Luc Hervé², is a generative and interactive installation that features 27 Raspberry Pi nanocomputers running *Node.js* *soundworks* clients. The audio synthesis is achieved using a *Node.js* wrapper on top of the *libpd* library [33, 34].

In this system, a number of strategies have been implemented to provide a dynamic and testable environment to the composer and to the computer music designer. Among them, we have implemented a centralized controller dedicated to controlling and monitoring the state and parameters of each agent in real time. For example, each square in the right half of Fig. 2 represents a musical agent in its relative

¹ *Future Perfect* has been composed and realized during a research/creation residency that took place in 2018 between IRCAM and ZKM.

² *Biotope* has been realized at IRCAM and created at the Centre Georges Pompidou, Paris, in the context of the exhibition “La fabrique du vivant.”

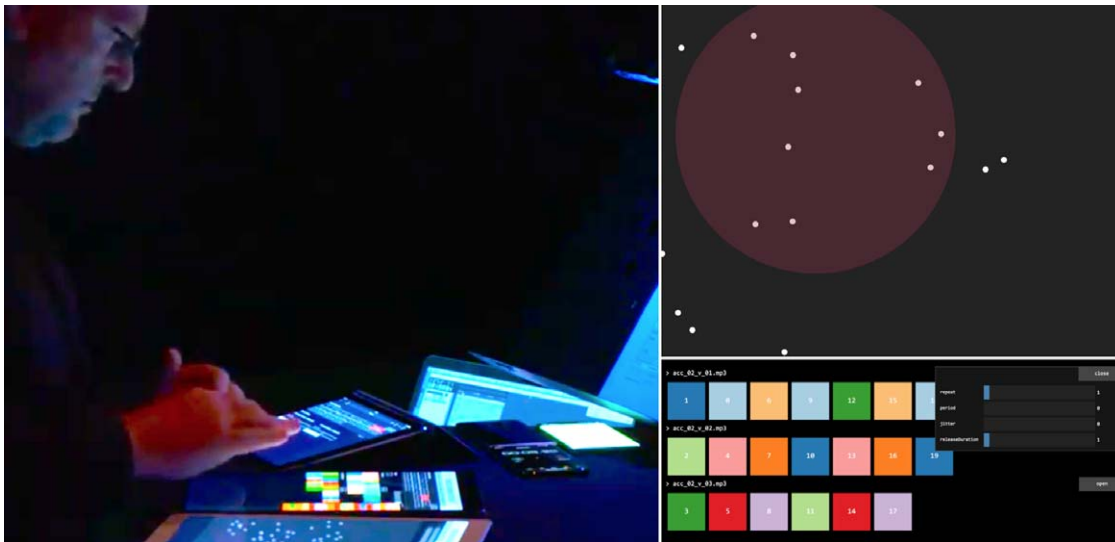


Fig. 1. On the left, Garth Paine performing *Future Perfect*. On the right, screenshots of two of the four control interfaces of the *Playground* application.

position in the exhibition space, the different colors giving an overview of their state in real time.

2.3 Scientific Settings and Measurements

A third important context of computer music research relates to scientific experimental research. In this context, some the characteristics of our framework, such as clock synchronization [35], enabled novel possibilities in scientific experimentations.

For example, the project *EmoDemos* [36] included an experiment dedicated to measuring precision and synchronization of the movement in groups of children practicing music (see Fig. 2, right). This experimental setup has been developed on top of *CoMo*—an application dedicated to creating movement-based distributed Interactive Machine Learning scenarios—[16], and allowed to record the motion sensors of smartphones tagged with synchronized timestamps. The portability and simplicity of deployment of the

system permitted us to measure almost 200 children, divided into groups of 10 to 15.

Once again, the system exposes a dedicated client to control and monitor the state of the application, allowing the experimenters to prototype and refine the protocol as well as to ensure smooth measurements in a very constrained timeline and environment. *CoMo* has also been used in different settings such as music, design and dance researches, artworks [37], and workshops.

2.4 Common Requirements and Patterns

These different examples show the large diversity of contexts a framework dedicated to computer music research and creation must support. Furthermore, they all implied intertwined periods of research, development, composition, and tests in the laboratory or the studio (possibly with musicians and performers) that deepen further the diversity of spaces and temporalities involved.

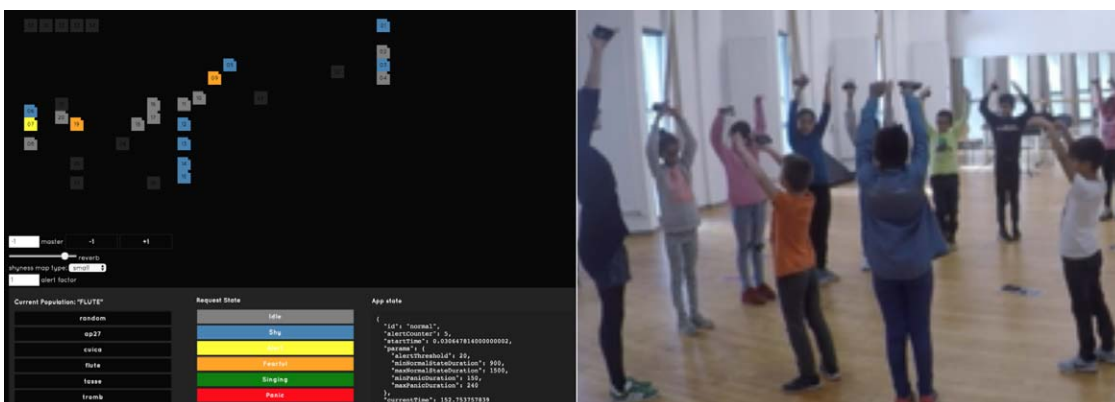


Fig. 2. On the left, a screenshot of the centralized controller developed for the installation *Biotope* composed by Jean-Luc Hervé. On the right, a session of measurement during the *EmoDemos* research project.

To adapt to these different contexts and their inherent constraints, the technological system must thus be easily developed, modified, or extended. This leads us to consider that two design aspects are of primary importance in the development of our framework.

First, the importance for *remote monitoring and control* that allows a single user in working situation (e.g., composer, researcher) to operate the distributed system—possibly composed of hundreds of devices—as a “single coherent system” [38]. We will describe in Sec. 5 how our framework proposes to support and facilitate the implementation of such functionality.

Second, the importance of being able to easily reuse existing functionalities but also to extend the framework with novel and dedicated components, to support exploratory workflows. Such problems can be addressed by introducing and supporting *composability* and *extensibility* in the system. We will describe in Sec. 5 how we propose to promote such aspects in `soundworks`.

3 ARCHITECTURE OVERVIEW

In this section, we present some high-level and general aspects of the `soundworks` framework. We present first the general architecture and scope of the framework and second a formalization of its most basic functionalities.

3.1 General Principles

Since its inception, `soundworks` has been dedicated to simplifying the development of web-based and distributed real-time musical systems. Applications created using `soundworks` follow a star network topology centered around a server written using `Node.js` (see Fig. 3). In these applications, clients can have multiple responsibilities (e.g., audio rendering, visual rendering, control) and be of different kinds (e.g., mobile, desktop, nanocomputers).

In previous versions, the framework was mainly focused on mobile applications and therefore privileged certain characteristics of these platforms (e.g., graphical user interface, usability). However, to support more diverse applications and use-cases, it 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 Internet of Things [IoT] elements). In this objective, the scope of the framework has been refined and narrowed down to focus only on four key aspects: initialization, communications, distributed state management, and plug-in host for external and dedicated functionalities. As a consequence, a number of functionalities (e.g., templating, graphical and audio rendering) have been removed from the core of the framework and delegated to external and specialized libraries. These developments also permitted us to reduce the API surface area of the framework, the number of dependencies, and, finally, improved its maintainability and learnability.

3.2 Initialization and Communications

The most basic functionality exposed by the framework is to easily bootstrap an application by taking care of initializing processes and communications. Fig. 4 summarizes the initialization process common to all `soundworks` clients:

- The `init` step consists in connecting two WebSockets to the server, one dedicated to JavaScript Object Notation (JSON)-compliant string data and a second to binary data. The API of both sockets is similar and exposes a simple `publish/subscribe` interface.
- Once sockets are connected, the plug-ins initialization can start. To support dependencies between plug-ins, `soundworks` can create a dependency graph start each plug-in accordingly.
- Finally, when all plug-ins are in a `ready` state, the application specific code (called `Experience` in `soundworks` terminology) can start.

Fig. 4 also illustrates a novel feature of the framework that enables the seamless implementation of `soundworks` clients in the two main environments: browsers and `Node.js`. Indeed, while this approach has already been tested and deployed in a production setting (cf. Sec. 3.2), the novel version the framework properly integrates it by making most of the code compatible to both platforms. This novel feature should foster IoT approaches [39, 25] by simplifying the creation of applications composed of diverse type of clients (e.g., smartphones, nanocomputers).

4 DISTRIBUTED STATE MANAGEMENT

An important novel feature of `soundworks` is the integration of a distributed state management system. This component is dedicated to supporting and simplifying the implementation of remote control and monitoring functionalities.

Since the introduction of the *Flux* pattern proposed by *Facebook* [40], usage of libraries that enforce unidirectional and circular data flow in the application is considered a good practice among the community. In our case, using such a pattern, which considers rendering as a pure function of the state, could therefore be very interesting, as the state of any node could be modified from a remote control interface in a transparent way for the node itself. However, existing libraries are not firstly designed for distributed applications and are difficult to adapt to our specific 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 a simple way to synchronize states across several nodes in the network. To tackle these issues, we designed a novel component implementing such unidirectional and circular data flow approach and adapted to the particular requirements of our applications.

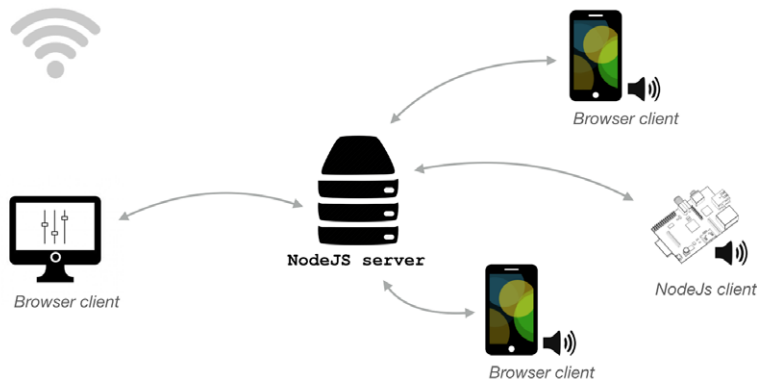


Fig. 3. Overview of the architecture of a typical soundworks application.

4.1 Concepts and Requirements

In our contexts, the application of such a unidirectional and circular pattern presents certain particularities, illustrated in Fig. 5.

First, the state of every client has to be kept synchronized server-side. The rationale for this design strategy stands in the importance of being able 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) in which the final applica-

tion cannot be specified beforehand and emerges from an iterative process..

Second, Fig. 5 highlights the need of a certain granularity in the definition and synchronization of the states. More precisely, while some variables and parameters (named `globals` in Fig. 5) needs to be accessible to every client (e.g., master volume, mute), the particular state a client (`clients [2]` in Fig. 5) 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/or performance situations.

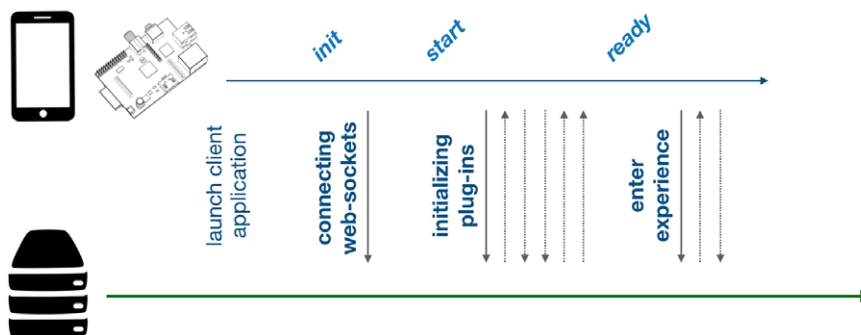


Fig. 4. Initialization steps of a soundworks client, mobile browser or *Node.js* process running on embedded hardware.

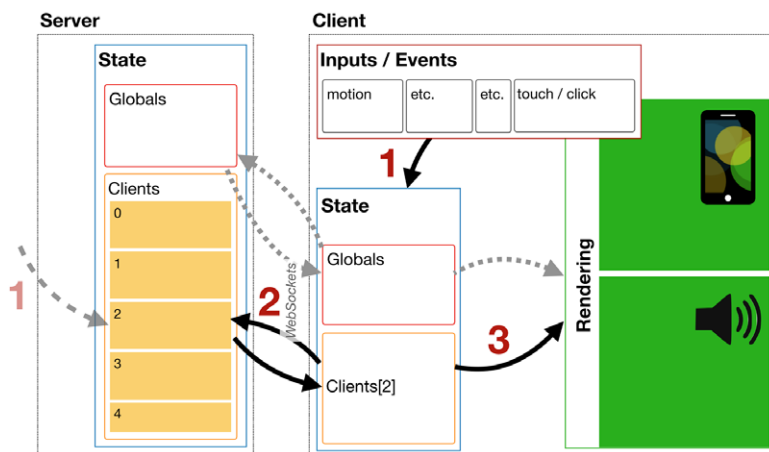


Fig. 5. Conceptual overview of a state management system enforcing unidirectional and circular data flow in a distributed context.

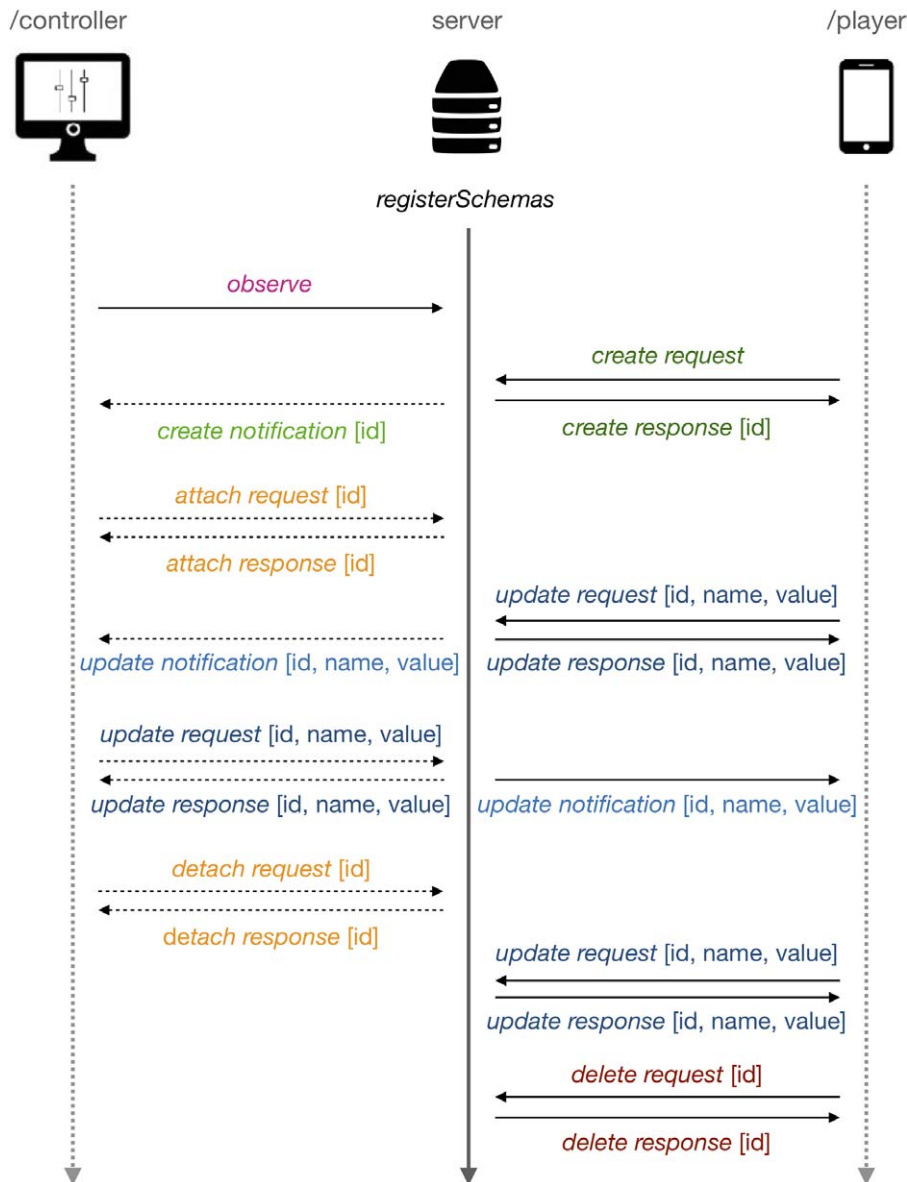


Fig. 6. Overview of the protocol designed for the soundworks state management system.

4.2 Protocol and API

To fulfill these requirements while preserving the idea of unidirectional and circular flow between actions, data, and rendering, we designed a simple protocol and implemented a new component. The main principles of the protocol we propose are as follows:

- Allow any node to *create* a new state from a declared schema.
- Allow any node 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.

Fig. 6 illustrates a generic scenario enabled by this protocol. A client (named "controller") *observes* the server and *attaches* to the state created by another client (named "player"). Once 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 receiving update notifications³.

The protocol is abstracted behind a reduced API illustrated in the pseudocode example of Fig. 7. This simple example also highlights two interesting aspects of the component:

³ Note that no particular guard has been implemented to prevent race conditions; therefore, the last event received wins.

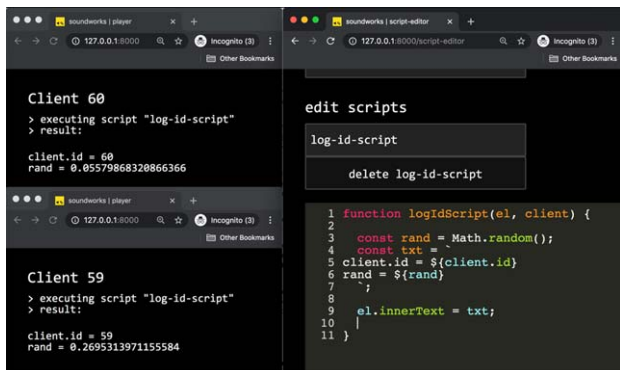


Fig. 7. *Pseudocode*—Main aspects of the soundworks state manager Application Programming Interface (API).

- The complete abstraction of network communications, allowing users to focus on the application logic rather than routing of network messages.
- The possibility to use schema declarations to generate controls and monitoring interfaces, simplifying fast prototyping and testing of ideas as well as implementation of dynamic and complex interfaces.

The simplicity of these synchronized data structures also enables more advanced uses of dynamic composition of states or distributed hierarchical state machines.

5 A HOST FOR PLUG-INS

Another important evolution of `soundworks` lies in its ability to act as a plug-in host for extending its basic functionalities. We believe this feature will also enhance modularity, allowing users to combine predefined components for a specific application but also to simplify maintenance and evolutions of both the framework and the applications. A number of plug-ins dedicated to synchronizing clocks, recording data, parsing, and watching the file system, to name a few, are already available.

In this section, we first present a technical overview of the implementation and registration of a `soundworks` plug-in. Second, we illustrate this feature with two novel components dedicated to runtime distributed scripting and logging of arbitrary data.

5.1 Implementing and Registering Plug-Ins

Thanks to the dynamic nature of the language, the implementation of a new plug-in is relatively simple. Fig. 8 illustrates several important aspects of the implementation of a new plug-in. First, the module exports a factory function that itself returns the plug-in class definition. This simple pattern allows `soundworks` to dynamically pass the `AbstractPlugin` parent class to the plug-in factory function and thus avoid hard-coded and circular dependencies between the plug-in and the host. Second, it shows (cf. `start` method) the different states that the plug-in must report to the host. Indeed, reporting these steps are important to be able to deal with all the different asynchronous

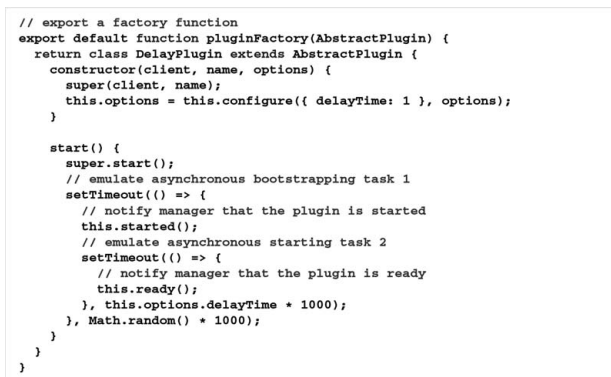


Fig. 8. *Pseudocode*—Main aspects of the implementation of a `soundworks` plug-in.

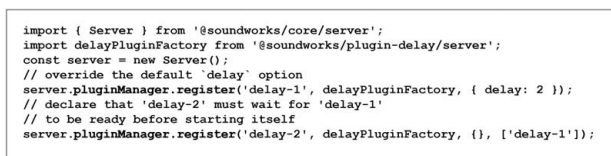


Fig. 9. *Pseudocode*—Server-side configuration and registration of a plug-in into `soundworks`.

tasks that have to be performed (e.g., network communication, particular graphical user interface [GUI] and user interactions) during the initialization of the application.

Fig. 9 illustrates how a plug-in is registered into `soundworks` (while Fig. 9 shows the process server-side, similar code would be written client-side) as well as two other possibilities. First, the possibility for a given plug-in factory to be used several times by registering it with a different identifier (e.g., `delay-1` and `delay-2`). For example, this capacity could be used to synchronize different clocks (e.g., audio clock and high-precision clock) on the same client. Second, it shows how dependencies between several plug-ins can be declared, enabling the possibility of implementing higher-order plug-ins on top of the functionalities offered by lower-level ones.

5.2 Examples

To illustrate the kind of functionalities the plug-in host system enable, we present two plug-ins we created for the novel version of the *CoMo* application (cf. Sec. 2.3). While designed and implemented with this specific use-case in mind, these two examples stand to be good examples of how this architecture facilitate the creation of modular and reusable components.

5.2.1 Runtime Distributed Scripting

The first plug-in we present, illustrated in Fig. 10, is dedicated to the scripting of focused parts of the application at runtime⁴. As such, the plug-in seeks to simplify the test

⁴<https://github.com/collective-soundworks/soundworks-plugin-scripting>

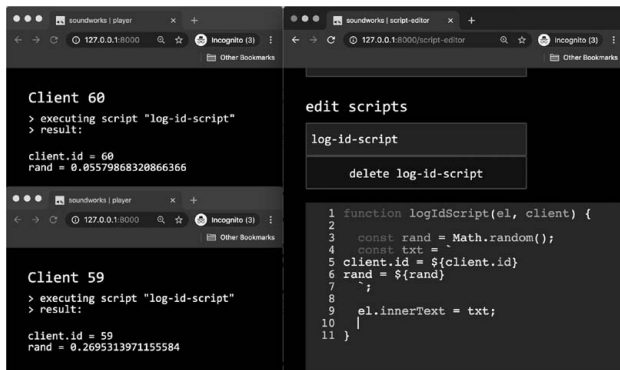


Fig. 10. Screenshot of the runtime distributed scripting interfaces. The function written on the editor (right) is dynamically executed on the two other clients (left) when updated.

of ideas and strategies (e.g., mappings, audio synthesis) in a very efficient manner: without having to reload the whole application—server and/or clients—nor having to implement each time a dedicated control interface.

Additionally, we think this plug-in can play an important pedagogical role by providing to users without expert programming knowledge (e.g., researchers, composers) a focused entry point where they can work within their own domain of expertise without having to understand the whole code base and architecture. We believe this functionality may turn out as an important addition to the tools our framework provide to support rapid prototyping, exploration, and testing of ideas.

5.2.2 Logging and Data Recording

The second component we present is dedicated to logging and storing on the server arbitrary data produced by any node of the network⁵. Indeed, simplifying access to such functionality to record and analyze data is obviously central to many scientific and research practices.

However, we believe that the simplicity of usage, illustrated in the Fig. 11, will also help to develop usages in other directions. For example, for auditing the system, testing components or benchmarking concurrent implementations in real-world situations or for recording and replaying examples of interactions (e.g., sensor data) to work on mappings and audio synthesis in the studio.

6 CONCLUSION AND FUTURE WORKS

In this paper, we have presented the motivations, design, and implementation aspects of the novel version of *soundworks*, a framework dedicated to developing distributed multimedia applications on the web. First, we have presented the different contexts such a framework should support and illustrated these contexts with three projects we developed over the last few years. These different contexts allowed us to show that supporting particular patterns is im-

```
const pathname = `${date}-${uuid}/${username}.csv`;
const log = await loggerPlugin.create(pathname);

// later
log.write(`${time}; ${x}; ${y};`);
```

Fig. 11. Pseudocode—Creation of a "csv" log file and writing of arbitrary data using the logger plug-in.

portant for exploratory tasks. We then presented the general architecture and two novel features of our framework: 1) the distributed state management system, dedicated to simplifying the implementation of *remote control and monitoring* and 2) its capacity to host external plug-ins, to foster *composability* and *extensibility*.

While we think this novel version of *soundworks* provides solid foundations to further explore the possibilities of the web platform in the area of distributed music systems, it also opens new questions and large areas for new developments. An important aspect that needs to be reconsidered and solved is the interoperability between the framework and other tools, such as graphical or audio libraries. In this regard, we think that while the schema format used for the state management component could provide a good basis in that direction, it is for now insufficiently specified. Another important limitation and direction of improvement is the lack of support for collections in the state management system; such additions would facilitate the implementation of advanced features such as presets or sound banks. Finally, to further simplify and fasten the implementation of new applications, a Command Line Interface tool for scaffolding components, clients, and plug-ins would be an important addition. We believe that the addition of these features could foster further research and artistic practices and maybe provide a common ground for pluralistic approaches in the area of distributed music systems.

7 ACKNOWLEDGMENT

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 program* (H2020-ICT-2014-1, Project ID 644862) and the *EmoDemos* project supported by the *Philharmonie de Paris*. It has also been supported by the IRCAM (Institute for Research and Coordination in Acoustics/Music, Paris) project *BeCoMe*, which is featured in the *Constella(c)tions* residency of the Science, Technology & the Arts (STARTS) program of the European Commission.

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

8 REFERENCES

- [1] "WebAudio API Specification," <https://www.w3.org/TR/webaudio/>.

⁵<https://github.com/collective-soundworks/soundworks-plugin-logger>

- [2] H. Choi, "AudioWorklet: The Future of Web Audio," *Proc. International Computer Music Conference*, p. 7 (2018).
- [3] "The WebSocket Protocol," <https://tools.ietf.org/html/rfc6455>.
- [4] "WebGL Specification," <https://www.khronos.org/registry/webgl/specs/latest/>.
- [5] A. Wirfs-Brock and B. Eich, "JavaScript: The first 20 years," *Proc. ACM on Programming Languages*, vol. 4, pp. 77:1–77:189, <https://doi.org/10.1145/3386327>.
- [6] T. Berners Lee, R. Cailliau, J. Groff, and B. Pollermann, "World Wide Web: The Information Universe," *Internet Research*, vol. 20, no. 4, pp. 461–471, (2010 Aug.), <https://doi.org/10.1108/10662241011059471>.
- [7] L. Wyse and S. Subramanian, "The Viability of the Web Browser as a Computer Music Platform," *Computer Music Journal*, vol. 37, no. 4, pp. 10–23 (2013 Dec.), https://doi.org/10.1162/COMJ_a_00213.
- [8] M. Weiser, "The Computer for the 21st Century," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 3, no. 3, pp. 3–11 (2003 Dec.), <https://doi.org/10.1145/329124.329126>.
- [9] D. Guinard and V. Trifa, "Towards the Web of Things: Web Mashups for Embedded Devices," presented at the *In MEM 2009 in Proceedings of WWW 2009. ACM*, p. 8.
- [10] S. Gresham-Lancaster, "The Aesthetics and History of the Hub: The Effects of Changing Technology on Network Computer Music," *Leonardo Music Journal*, vol. 8, pp. 39–44 (1998 Jan.), <https://doi.org/10.2307/1513398>.
- [11] Á. Barbosa, "Displaced Soundscapes: A Survey of Network Systems for Music and Sonic Art Creation," *Leonardo Music Journal*, vol. 13, pp. 53–59 (2003 Dec.), <https://doi.org/10.1162/096112104322750791>.
- [12] G. Weinberg, "Interconnected Musical Networks: Toward a Theoretical Framework," *Computer Music Journal*, vol. 29, no. 2, pp. 23–39 (2005 Jun.), <https://doi.org/10.1162/0148926054094350>.
- [13] F. Bayle, "Space, and More," *Organised Sound*, vol. 12, no. 3, pp. 241–249 (2007 Dec.), <https://doi.org/10.1017/S1355771807001872>.
- [14] B. Taylor, "A History of the Audience as a Speaker Array," presented at the *Proceedings of the NIME17 Conference* (2017).
- [15] I. Poupyrev, M. J. Lyons, S. Fels, and T. Blaine (Bean), "New Interfaces for Musical Expression," presented at the *CHI '01 Extended Abstracts on Human Factors in Computing Systems*, pp. 491–492, <https://doi.org/10.1145/634067.634348>.
- [16] B. Matuszewski, J. Larralde, and F. Bevilacqua, "Designing Movement Driven Audio Applications Using a Web-Based Interactive Machine Learning Toolkit," *Proc. 4th Web Audio Conference* (2018).
- [17] H.-J. Reinberger, "Experimental Systems: Historicity, Narration, and Deconstruction," *Science in Context*, vol. 7, no. 1, pp. 65–81 (1994 Spring), doi :<https://doi.org/10.1017/S0269889700001599>.
- [18] H.-J. Rheinberger, "Consistency From the Perspective of an Experimental Systems Approach to the Sciences and Their Epistemic Objects," *Manuscripto*, vol. 34, no. 1, pp. 307–321 (2011 Jun.), <https://doi.org/10.1590/S0100-60452011000100014>.
- [19] M. Schwab (Ed.), *Experimental Systems: Future Knowledge in Artistic Research* (Leuven University Press, Leuven, Belgium, 2016).
- [20] C. Alexander, *Notes on the Synthesis of Form* (Harvard University Press, Cambridge, Massachusetts, 1964).
- [21] "Soundworks Repository," <https://github.com/collective-soundworks/soundworks>.
- [22] S. Robaszkiewicz and N. Schnell, "Soundworks—A Playground for Artists and Developers to Create Collaborative Mobile Web Performances," *Proc. of the 1st Web Audio Conference* (2015).
- [23] N. Schnell, B. Matuszewski, J.-P. Lambert, S. Robaszkiewicz, O. Mubarak, D. Cunin, S. Bianchini, X. Boissarie, and G. Cieslik, "Collective Loops: Multimodal Interactions Through Co-Located Mobile Devices and Synchronized Audiovisual Rendering Based on Web Standards," *Proc. of the Tenth International Conference on Tangible, Embedded, and Embodied Interaction*, pp. 217–224, <https://doi.org/10.1145/3024969.3024972>.
- [24] B. Matuszewski, N. Schnell, and F. Bevilacqua, "Interaction Topologies in Mobile-Based Situated Networked Music Systems," *Wireless Communications and Mobile Computing*, vol. 2019, pp. 1–9 (2019 Mar.), <https://doi.org/10.1155/2019/9142490>.
- [25] B. Matuszewski and F. Bevilacqua, "Toward a Web of Audio Things," *Proc. 15th Sound and Music Computing Conference* (2018).
- [26] B. Matuszewski, "Soundworks A Framework for Networked Music Systems on the Web," *Proc. 5th Web Audio Conference*, p. 6 (2019).
- [27] M. Puckette, "Combining Event and Signal Processing in the MAX Graphical Programming Environment," *Computer Music Journal*, vol. 15, no. 3, pp. 68–77 (1970 Feb.), <https://doi.org/10.2307/3680767>.
- [28] M. Puckette, "A Case Study in Software for Artists: Max/MSP and Pd," in *Art++*, David-Olivier Lartigaud (Ed.) (2016).
- [29] "WebAudio-Patcher," <https://github.com/Fr0stbyteR/webaudio-patcher>.
- [30] "Rhizome," S. Piquemal, <https://github.com/sebpiq/rhizome>.
- [31] J. Allison, Y. Oh, and B. Taylor, "NEXUS: Collaborative Performance for the Masses, Handling Instrument Interface Distribution Through the Web," *Proc. NIME13 Conference* (2013).
- [32] "Biotope Presentation," <https://youtu.be/RmSujqdT6L0>.
- [33] P. Brinkmann, C. McCormick, P. Kirn, M. Roth, R. Lawler, and H.-C. Steiner, "Embedding Pure Data with libpd," presented at the *Pure Data Convention Weimar 2011*.
- [34] "Node-libpd Repository," <https://github.com/ircam-jstools/node-libpd>.
- [35] J.-P. Lambert, S. Robaszkiewicz, and N. Schnell, "Synchronisation for Distributed Audio Rendering over

Heterogeneous Devices, in HTML5,” *Proc. 2nd Web Audio Conference* (2016).

[36] “Emodemos Website,” <https://www.unige.ch/cisa/emodemos/>.

[37] “Constella(c)tions - Residency,” <https://vertigo.starts.eu/calls/starts-residencies-call-3/residencies/constellactions/detail/>.

[38] M. van Steen and A. S. Tanenbaum, “A Brief Introduction to Distributed Systems,” *Computing*, vol.

98, no. 10, pp. 967–1009 (2016), <https://doi.org/10.1007/s00607-016-0508-7>.

[39] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthez, “Internet of Musical Things: Vision and Challenges,” *IEEE Access*, vol. 6, pp. 61994–62017 (2018), <https://doi.org/10.1109/ACCESS.2018.2872625>.

[40] “Flux Pattern,” <https://facebook.github.io/flux/>.

THE AUTHOR



Benjamin Matuszewski

Benjamin Matuszewski studied musicology and music theory before working for several years as web developer in the media industry. Since 2014, he has been a researcher and developer in the Sound Music Movement Interaction team at IRCAM (Institute for Research and Coordination

in Acoustics/Music), Paris, France. His research focuses on interactive and real-time distributed systems for music using Web technologies. Since 2015, he has also been a PhD student at the CICM (Centre de recherche Informatique et Creation Musicale, Paris 8 University).