Rapid Prototyping of Distributed Musical Things using Web Technologies

Benjamin Matuszewski, Aliénor Golvet

Rapid Prototyping of Distributed Musical Things using Web Technologies

Matuszewski Benjamin
STMS Ircam-CNRS-Sorbonne Université
Paris, France
benjamin.matuszewski@ircam.fr

Golvet Aliénor
STMS Ircam-CNRS-Sorbonne Université
Paris, France
alienor.golvet@ircam.fr

***Abstract*—This short paper reports on recent advances in the development of an open-source prototyping platform for the creation of distributed and embedded musical systems based on Web technologies. The proposed architecture is based on user-grade hardware and open-source software and aims at fostering rapid-prototyping and experimentation in the context of colocated distributed systems for musical research, creation and performance. After a short review of related works, the paper describes the general design and the different building blocks of the system. Then, it exposes a first characterization of the proposed design, and concludes with the description of a prototype that highlights different features of the platform.**

***Index Terms*—Sound and music computing, Distributed applications, Web standards, Prototyping platform**

I. Introduction

Recent development of Web technologies–in particular the Web Audio API specification [1]–and of Internet of Things (IoT) technologies have enabled new avenues for researchers, designers and artists relying on distributed multimedia environments and collaborative interaction. Indeed, by democratizing the access of both software and hardware, the combination of these technologies can unfold novel possibilities in several areas such as multisource electro-acoustic music [2] or new interfaces for musical expression [3].

However, the inherent complexity of such distributed systems, particularly regarding the real-time, synchronized and multimodal interactions required for music and performing arts, opens novel questions on designing interaction modalities that enable practitioners to use and to creatively appropriate these technologies. Another challenge stands in the difficulty of designing and implementing a distributed platform that is both stable in terms of hardware and software, while being also simple to manipulate and open to modification in order to be adapted to specific experimental projects [4].

In this paper, we present recent advances in the development of an open-source prototyping platform for the creation of distributed and embedded musical systems based on Web technologies [5], [6]. The platform aims at supporting the development of a wide range of musical applications, such as audio installations, live performances or dedicated tools for composers and performers. Within the broader field of the Internet of Sound (IoS) [7], our approach focuses on fostering rapid-prototyping and experimental practices in the context of colocated distributed systems for musical research, creation and performance.

After a short and non-exhaustive review of similar approaches in Section II, we describe the general design of our system in Section III. Then, we report on some characterization of the proposed design regarding latencies in Section IV. Finally, we conclude with the description of a prototype highlighting several features of our platform in Section V.

II. Related Work

Several environments dedicated to music and artistic creation, and targeted at embedded devices and single-board computers have been proposed over the years. These environments can be broadly separated into two categories: the ones that propose a combination of software running on top of a dedicated hardware and the ones that only rely on software that can be executed on generic consumer grade hardware.

In the first category, probably most famous example is the *Bela* platform [8], which is an environment composed of an extension board for the BeagleBone Black focused on processing audio and data signals at very low latency, and of a dedicated Linux distribution with a real-time kernel. For the end-user, the platform can be programmed using C++ or PureData. Focused on the creation of new instruments, the Bela has been used in numerous artistic and research projects [9].

In the other category, the *Satellite CCRMA* [10] proposed a dedicated Linux distribution specifically designed for the Raspberry Pi. The distribution featured a real-time kernel specifically configured for interactive multimedia applications. However, even if some projects using the *Satellite CCRMA* has been proposed recently [11], the last published version of the *Satellite CCRMA Distribution* only targets the Raspberry Pi 2 and has not been updated since then[1]. This project exemplifies the difficulty of maintaining such a project in the long term due to the rapid evolution of the technologies and of the highly particular knowledge and skills that maintaining a distribution requires.

More recently, Elk OS [12], [13] proposed an interesting alternative in the form of a very efficient and dedicated operating

[1]Note that the first version of the Raspberry Pi 3 was proposed in February 2016

system. Compared to the *Satellite CCRMA*, *Elk OS* is oriented toward the use of de facto audio industry standards such as the VST plugin format to ease the development of new instruments and dedicated hardware [14]. However, we consider this focus on such a limited number of technologies and formats comes with a drawback regarding more experimental and exploratory practices with other technologies.

Compared to these projects, our approach is to build on top of very common and accessible technologies. This choice aims at minimizing the risk of obsolescence by using generic hardware and software at the low-level, and to accept their potential drawbacks (e.g. in terms of efficiency). On the contrary, this choice enables the creation of a versatile environment within which each individual component is rather low-cost, easily replaceable and benefits from a large community of users and resources (e.g. softwares, tutorials, documentation, etc.). Additionally, as our focus is on building distributed systems composed of numerous devices, some tradeoffs that are not acceptable when creating a Digital Musical Instruments (DMI), e.g. according to latencies [15], can be reconsidered according to this specific context. For example, in a generative system composed of tens or hundreds of devices, the question is more about their synchronization than on their individual latency.

III. General Design

As introduced in the previous section, our goal is to build an environment that is both easily adaptable and resilient to the rapid evolution of software and hardware. To that end the core of our system is built around Web technologies, standardized by the World Wide Web Consortium (W3C) and on the Node.js JavaScript runtime, which while not per se a standard, is backed by the Open JS Fundation[2] (see Fig 1). Such an approach, which indeed comes with some drawbacks, introduces a layer of abstraction between the software and the hardware which we think should help to maintain our environment while fostering rapid prototyping and versatility of the system, which we consider crucial in experimental artistic practices.

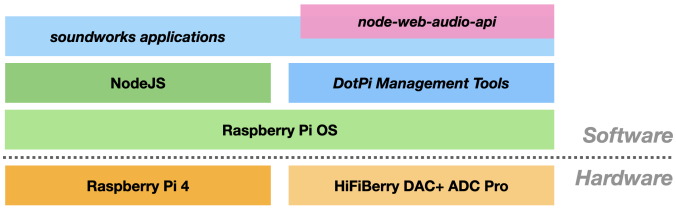

Fig. 1. Hardware and software stack

A. Hardware

Following these main design principles, we therefore chose to build our environment on top of the widespread Raspberry Pi platform [16]. In our view, using this platform has two main benefits: first, it is rather low cost which is of primary importance when building a fleet of devices composed of numerous devices, second, it benefits from a large community of users and of important documentation and support. In its version 4, the Raspberry Pi offers the following features: a quad core Cortex-A72 (ARM v8) 64-bit processor at 1.8GHz; up to 8GB of LPDDR4-3200 SDRAM; two USB 2.0 and two USB 3.0 ports; 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0 and BLE Gigabit Ethernet for networking and communications, and a 40 pin GPIO header to interface with external electronic devices.

Additionally, we chose to extend the platform with a HiFiBerry DAC+ ADC Pro sound card [3] which features stereo input and output, support up to 192kHz sample rate and communicate with the Raspberry Pi through the I2S bus.

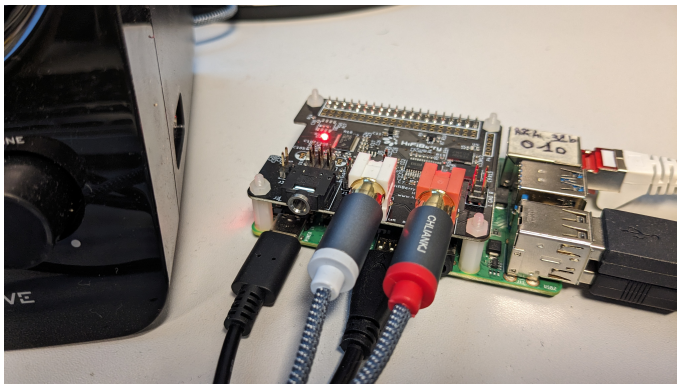

Fig. 2. Illustration of the "standard" hardware used in our environment: a Raspberry Pi 4 with a HiFiBerry DAC+ ADC Pro

Depending on the project, this "standard" minimal setup can then be easily extended or adapted. For example, the Raspberry Pi can easily work together with microcontrollers such as the Arduino platform for the prototyping of more traditional DMI. The whole setup can run on batteries, which may simplify the deployment of a system in unconventional spaces (e.g. gardens) for performances. Furthermore, for more constrained or less demanding projects, the environment can run with only few configuration modifications on other boards, such as the Raspberry Pi Zero W [4] or with different sound cards[5], e.g. for sound installations that would only require audio output.

B. Software

In terms of software, we aim to keep the whole system architecture simple to facilitate its appropriation by users such as artists, makers and researchers. Hence, we chose for now to rely on a stock Raspberry Pi operating system, which therefore does not feature a real-time kernel. To facilitate the configuration and maintenance of fleets of similar devices (e.g. management of SSH keys, hostnames), and adapt the system to our software stack (e.g. to install Node.jsand the Jack audio server [17]), we only rely on a few configuration scripts and on

[2]https://openjsf.org/

[3]https://www.hifiberry.com/shop/boards/hifiberry-dac-adc-pro/

[4]https://www.raspberrypi.com/products/raspberry-pi-zero-w/

[5]e.g. https://www.hifiberry.com/shop/boards/hifiberry-dac-light/

a dedicated application, we call the *dotpi* environment[6]. Hence, this environment should be compatible with any Debian based operating system.

From this point, we solely rely on JavaScript and Web technologies, running our applications in the Node.js runtime which adds yet another layer of abstraction between our software stack and the underlying platform. Such approach both simplifies the prototyping of new applications which are not tightly tied to a particular platform and can therefore be developed on any machine, as well as maintenance and deployment as applications can seamlessly run on any available hardware that is able to run a Node.js runtime.

To develop our distributed applications, we rely on *soundworks* [6], [18], a framework dedicated to the rapid prototyping of applications written in JavaScript. The framework aims at fostering very rapid loops of trial and error, which is of primary importance in creative workflows, in systems composed of possibly tens of devices. To that end, it provides a set of tools dedicated to help developers to manage some of the complexities of distributed applications, such as distributed state management or synchronization.

Finally, the audio processing in embedded platforms is delegated to the `node-web-audio-api` library, which provides Node.js bindings on top an implementation of the Web Audio API written in the Rust programming language [19], [20][7]. This approach enables us to write audio engines and applications that can run seamlessly in Web browsers and embedded hardware in the Node.js runtime. As such, it fosters code reuse and opens new doors for experimenting novel forms of collective and distributed applications mixing human and synthetic agents.

IV. Characterization

In this section, we report on measurements conducted to characterize a baseline of what can be achieved within our environment considering both network and audio latencies. Please note these numbers are indicative and only reported here to give an order of magnitude of what can be obtained under particular circumstances.

A. Network latencies

To characterize network latencies, the following experimental setup has been implemented. A *soundworks* Node.js client running on a Raspberry Pi sends a packet of fixed size to the server which immediately sends it back to the client. Two variables have been tested: 1) the size of the packet: 10B, 100B, 1kB, 10kB, 100kB and 2) the type of network: an entry level WiFi Access Point (AP) (TP Link TL-MR3020[8]), a more professional grade AP (Unifi Nano HD[9]), and direct cabled connection between the Raspberry Pi and the server through RJ45. For each case 250 packets have been sent at a fixed interval of 50ms.

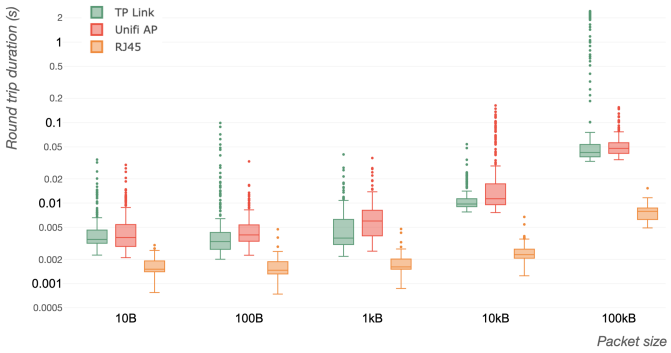

Fig. 3. Network round trip duration for different packet sizes measured on three different network configurations.

The box plots in Fig. 3 report the estimated round trip duration for each tested case. While not surprising, these results interestingly show that 1) close to negligible latency (≈2ms) with cabled setup can be achieved for very constrained situations such as experimental studies and that 2) the entry-grade AP, while surprisingly performant, rapidly degrades with the increase of packet size, when compared to the more professional one. Complementary measurements should be done to complete these results with more realistic setups composed of multiple devices concurrently sending messages.

B. Audio latency

To measure the round trip audio latency, we looped back the audio output to input with an audio cable and configured the Jack backend with the following setup (which has been manually found as the minimim configuration to avoid dropouts): *realtime*: on, *frame/periods*: 128, *period*: 2, *sample rate*: 48kHz, *priority*: 95. The calculation of the audio round trip is done using a small utility written in Rust[10] which allows it to be in the exact same audio toolchain as in our final software stack. The tool basically emits a dirac impulse to the output every second and computes the duration until a non zero value is found in the incoming signal.

The round trip latency reported by the tool is of ≈14.3ms. Compared to the result of 9.05ms reported by `jack_iodelay`, it shows that our software stack introduces an additional latency of 256 samples, which can be explained by the lack of duplex support of the audio backend we use[11]. All things considered, while important work still needs to be conducted to improve further these results (for example by applying a `PREEMPT RT` patch on the OS kernel [12]), we consider such result as low enough to be usable in many kinds of musical applications, except the most latency sensitive ones (e.g. augmented instruments).

[6]https://github.com/ircam-ismm/dotpi-manager (at time of writing, these tools are still under active development.)

[7]The Node.js bindings can be found in the https://github.com/ircam-ismm/node-web-audio-api/ repository, the underlying Rust implementation can be found at https://github.com/orottier/web-audio-api-rs/

[8]https://static.tp-link.com/res/down/doc/TL-MR3020_1.0_1_.pdf

[9]https://dl.ubnt.com/datasheets/unifi/UniFi_nanoHD_AP_DS.pdf

[10]See the `roundtrip_latency_test` example in the `web-audio-api-rs` repository

[11]https://github.com/RustAudio/cpal/issues/349

V. Example Use

To illustrate some of the possibilities of our environment we developed a prototype application that showcases the potential for fast prototyping, modification and deployment of audio effects on a fleet of devices. In this prototype, each device is equipped with a microphone and speakers, and the audio is processed locally on each device. Scripting, control and monitoring of the devices are done remotely in real-time.

Fig. 4. The application interface in a web browser. On the left, a text editor to create and edit scripts in real time. On the right, different controls and monitoring for three devices connected to the application.

The application's control interface shown in Fig. 4 is accessible within a Web browser. It is composed of two main parts:

- A text editor that allows you to create and to edit audio scripts in JavaScript. Using the Web Audio API, users can write their own audio graphs connecting the audio input to the output on the device. These user defined scripts are then shared over the network to be executed by the connected devices. Upon update of a script, any device associated with that script will instantly transition to the new version of the script with a crossfade.
- A control and monitoring zone for each device connected to the application that gives the possibility to remotely 1) set which script, i.e. audio effect, is used by said device, 2) change the input and output volumes and 3) monitor input (dry) signal and output (wet) signal.

Since the interface is built on top of web technologies, it can be accessed simultaneously on any device equipped with a web browser and connected to the local network of devices. Moreover, since audio scripts are written using Web Audio code, they can be interpreted on both web browsers with the Web Audio API or embedded devices with the `node-web-audio-api` library. This allows the use of an heterogeneous fleet of devices composed of, for example, mobile phones handled by human beings, Raspberry Pi with portable speakers spread over the performance space or laptop computers connected to larger speakers.

Since script updates are automatically distributed to all devices, this application fosters a trial-and-error workflow for prototyping experimental setups on fleets of devices. It also lowers the boundaries for appropriation by users with different skills and backgrounds, as they do not have to understand all the internals of the system. Finally, while the application has been tested only with a small number of concurrent devices, our previous works have shown that it should simply scale to dozens or hundreds of connected devices.

VI. Conclusion and Future Work

In this paper, we have reported on the ongoing development of an open-source prototyping platform for the creation of distributed and embedded applications based on Web technologies. We have first described the main design aspects of the system, presented some elements of characterization of the system, and concluded with the description of a prototype application that highlights several features of the proposed environment. While these results are encouraging, important work of design, research and development still needs to be achieved in order to provide an environment that can be appropriated by expert users (e.g. researchers, designers, artists) from different backgrounds and different programming skills. To that end, we will conduct in the near future research/creation residencies with musicians and composers to better understand the current possibilities and limitations of our platform in a more ecological context.

Acknowledgment

We would like to thank our colleagues at IRCAM for their precious contributions to the project. This project has received support from the DOTS research project funded by the French National Research Agency (ANR-22-CE33-0013-01).

References

[1] "Web Audio API Specification," 2021. [Online]. Available: https://www.w3.org/TR/webaudio/

[2] B. Taylor, "A History of the Audience as a Speaker Array," in *Proceedings of the NIME'17 Conference*, 2017. [Online]. Available: http://homes.create.aau.dk/dano/nime17/papers/0091/paper0091.pdf

[3] I. Poupyrev, M. J. Lyons, S. Fels, and T. Blaine (Bean), "New interfaces for musical expression," in *CHI '01 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '01. Seattle, Washington: Association for Computing Machinery, Mar. 2001, pp. 491–492.

[4] H.-J. Rheinberger, "Consistency from the perspective of an experimental systems approach to the sciences and their epistemic objects," *Manuscrito*, vol. 34, no. 1, pp. 307–321, Jun. 2011. [Online]. Available: http://dx.doi.org/10.1590/S0100-60452011000100014.

[5] B. Matuszewski and F. Bevilacqua, "Toward a Web of Audio Things," in *Proceedings of the 2018 Sound and Music Computing Conference*, Limassol, Cyprus, 2018.

[6] B. Matuszewski, "A Web-Based Framework for Distributed Music System Research and Creation," *Journal of Audio Engineering Society*, vol. 68, no. 10, pp. 717–726, Oct. 2020.

[7] L. Turchet, M. Lagrange, C. Rottondi, G. Fazekas, N. Peters, J. Østergaard, F. Font, T. Bäckström, and C. Fischione, "The Internet of Sounds: Convergent Trends, Insights, and Future Directions," *IEEE Internet of Things Journal*, vol. 10, no. 13, pp. 11 264–11 292, Jul. 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10061604/

[8] A. McPherson and V. Zappi, "An environment for submillisecond-latency audio and sensor processing on BeagleBone Black," in *Audio Engineering Society Convention 138*. Warsaw, Poland: Audio Engineering Society, 2015.

[9] V. Zappi and A. McPherson, "Hackable Instruments: Supporting Appropriation and Modification in Digital Musical Interaction," *Frontiers in ICT*, vol. 5, p. 26, Oct. 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fict.2018.00026/full

[10] E. Berdahl, S. Salazar, and M. Borins, in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Daejeon, Korea, 2013.

[11] J. Sullivan, J. Vanasse, M. M. Wanderley, and C. Guastavino, "Reinventing the Noisebox: Designing Embedded Instruments for Active Musicians," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Birmingham, United Kingdom, 2020.

[12] L. Turchet and C. Fischione, "Elk Audio OS: An Open Source Operating System for the Internet of Musical Things," *ACM Transactions on Internet of Things*, vol. 2, no. 2, pp. 1–18, May 2021. [Online]. Available: https://dl.acm.org/doi/10.1145/3446393

[13] L. Vignati, S. Zambon, and L. Turchet, "A Comparison of Real-Time Linux-Based Architectures for Embedded Musical Applications," *Journal of the Audio Engineering Society*, vol. 70, no. 1/2, pp. 83–93, Jan. 2021. [Online]. Available: https://www.aes.org/e-lib/browse.cfm?elib=21553

[14] D. Stefani and L. Turchet, "On the challenges of embedded real-time music information retrieval," in *Proceedings of the International Conference on Digital Audio Effects*, Vienna, Austria, 2022.

[15] A. P. McPherson, R. H. Jack, and G. Moro, "Action-Sound Latency: Are Our Tools Fast Enough?" in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2016.

[16] J. D. Brock, R. F. Bruce, and M. E. Cameron, "Changing the world with a Raspberry Pi," *Journal of Computing Sciences in Colleges*, vol. 29, no. 2, pp. 151–153, Dec. 2013.

[17] S. Letz, Y. Orlarey, and D. Fober, "Jack audio server for multi-processor machines," in *Proceedings of the International Computer Music Conference*, Barcelona, Spain, 2005.

[18] B. Matuszewski, "Soundworks - A Framework for Networked Music Systems on the Web," in *Proceedings of the 5th Web Audio Conference*, Throndheim, Norway, 2019.

[19] O. Rottier and B. Matuszewski, "A Rust Implementation of the Web Audio API," in *Proceedings of the 7th Web Audio Conference*, Cannes, France, 2022.

[20] B. Matuszewski and O. Rottier, "The Web Audio API as a standardized interface beyond Web browsers," *Journal of Audio Engineering Society*, 2023, in press.