Evolution of the Web Audio Modules Ecosystem

Michel Buffa, Shihong Ren, Tom Burns, Antoine Vidal-Mazuy, Stéphane Letz

Evolution of the Web Audio Modules Ecosystem

Michel Buffa
Université Côte d'Azur
I3S, CNRS, INRIA
buffa@univ-cotedazur.fr

Shihong Ren
Shanghai Conservatory of Music, Université de Saint-Etienne
shihong.ren
@univ-st-etienne.fr

Tom Burns
sequencer.party
tom@burns.ca

Antoine Vidal-Mazuy
antoine.vidalma@gmail.com
Université Côte d'Azur
CNRS, INRIA

Stéphane Letz
Univ Lyon, GRAME-CNCM, INSA Lyon, Inria, CITI, EA3720, 69621
Villeurbanne, France
letz@grame.fr

ABSTRACT

The Web Audio Modules (WAM) is a mature proposal for Web Audio Plugins. Started in 2015 [3, 6, 7, 8], this open source proposal evolved and reached version 2.0 in 2021 [17]. This resource paper provides an overview of the evolution of the ecosystem surrounding the WAM standard. Since V2.0 publication WAM has been the subject of numerous publications, demonstrations, and presentations at various conferences such as the Audio Developer Conference, Sound and Music Computing, Internet of Sounds, and The Web Conference, to name a few. As a direct consequence, there is now a large set of WAM-format plugins and hosts, along with new online tutorials and tools that facilitate plugin development (online IDEs, etc.), the vast majority being open source. An initiative called Wam Community has also emerged to simplify the sharing of effects and instruments in the WAM format, accompanied by an API and online tools. New WAM hosts have been introduced, including two Digital Audio Workstations (DAWs), collaborative WAM hosts applications for music creation, and compatible visual programming languages. Furthermore, the WAM format supports extensions and this article presents some of the available ones for handling video, 3D, WebGL shaders, and parameter modulation between WAMs, among other functionalities.

CCS CONCEPTS

• **Software and its engineering → Software organization and properties → Software system structures →** Abstraction, modeling and modularity

KEYWORDS

Web Audio, Audio Effects and Instruments, plugin Architecture, Web Standards

1 INTRODUCTION

The official Web Audio Module (WAM) distribution proposes two complementary SDKs and an abstract API for developing Web Audio plugins [22]. The SDKs are available on the official GitHub repository[1] along with some examples of WAM hosts and plugins written using different approaches[2] (in plain JavaScript, using build systems, written in TypeScript, using some popular frameworks such as React, or cross compiled in Web Assembly from C++ or from DSL like Faust or CSound). The SDKs are also available as npm modules[3] and also hosted directly online on a CDN for a direct inclusion in plain HTML/JS code[4].

Since their publications, these resources only had minor bug fixes and enhancements, highlighting the maturity and stability of their design and implementation.

However, the feedback that has been collected through the web audio slack channel, the audio developer discord server (that has a channel dedicated to web development), and github issues and comments, showed that there was a need for more tutorials, for simpler examples and simpler ways to develop WAM plugins and hosts (aka : better tools, such as code generators, online IDEs, etc.). Furthermore, in its official release the WAM standard focused on the essential features required by plugin and host development, but it has been designed to support extensions. Rapidly developers started to propose WAM extensions for interacting with real time video (webcam), for 3D rendering, WebGL shader animation (WAM for visualization) and for adding some extra functionalities to (parameter modulation between WAMs, etc.).

Section 2 of this paper will present webaudiomodules.com, the new web site that will act as a hub for the WAM ecosystem, and focus on Wam-Community, a new initiative for publishing, sharing and reusing WAMs. Section 3 will present some tools that ease the development of WAM hosts and plugins (code generators, online tutorials, online IDEs). Section 4 will present some remarkable WAM effects and instruments that have been developed, as well as some hosts (DAWs etc.). Section 5 will present the WAM extensions available, as well as how you can learn how to use them in your own software. Section 6 will present the perspectives about the future of WAMs and conclude.

2 WEBAUDIOMODULES.COM, WAM COMMUNITY

For a long time, the webaudiomodules.org website was the reference center for WAM-related content. Unfortunately, this site is no longer maintained, and the people currently involved in the WAM standard do not own the domain. In the meantime, a new website, webaudiomodules.com, has been created, bringing together the most relevant WAM resources. This new domain also hosts online versions of the SDKs[5], an API for querying WAM plugins published by the WAM community[6], and a WAM gallery based on this API, which will facilitate the reuse of existing WAMs.

WAM plugins have been designed so that they can be used in host software simply by using a JavaScript import on their URI. Include the SDK, import the URI and you can then connect the plugin to a Web Audio graph (any instance of WAM is seen as an AudioNode) and display its graphical interface (seen as a Web Component aka a simple HTML div with encapsulated code). This makes it very easy to reuse Web-hosted WAMs. The Wam Community initiative

[1] https://github.com/webaudiomodules, MIT Licence
[2] https://github.com/webaudiomodules/wam-examples
[3] https://www.npmjs.com/settings/webaudiomodules/packages
[4] https://codepen.io/w3devcampus/pen/LYaGzMQ?editors=1010
[5] https://codepen.io/w3devcampus/pen/xxBOvVJ?editors=1010
[6] https://www.webaudiomodules.com/community/plugins.json

was created to facilitate the publication and sharing of WAMs, and comprises several tools: 1) a GitHub repository (https://github.com/boourns/wam-community) with pre-built, ready-to-use WAMs . Developers can add their own WAMs by making simple Pull Requests. 2) The webaudiomodules.com server hosts the plugins published on the previous repository and offers a REST API to obtain the list of available WAMs and expose their URIs. 3) An online gallery (Figure 1) lets you browse WAMs, with a search engine and filters by category, author, etc. (https://mainline.i3s.unice.fr/wamGallery/public). It's also possible to test WAMs online and copy/paste the source code of a mini WAM host, with several possible options (Figure 2).

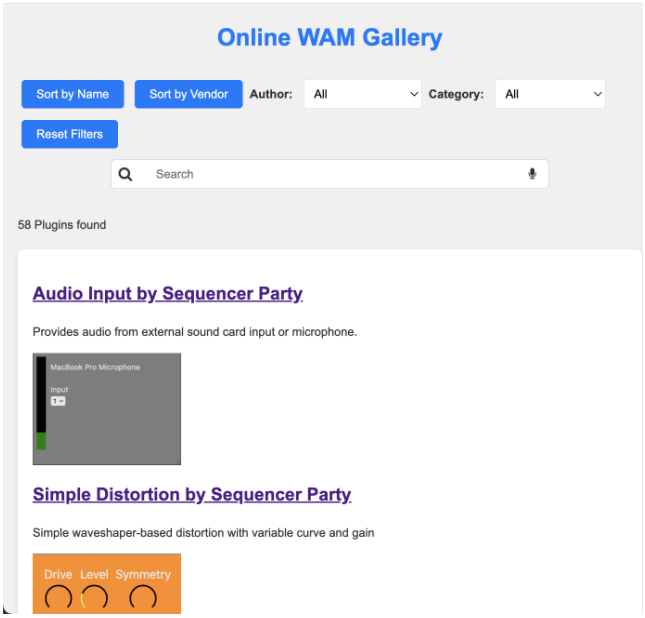

Figure 1: online WAM gallery web application.

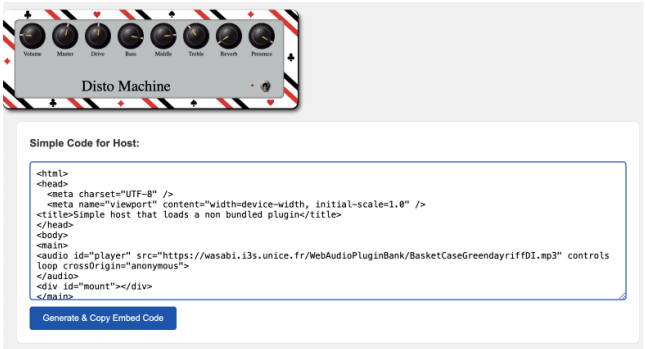

Figure 2: The WAM Gallery can run WAMs in demo pages and generate the source code for a simple host.

3 WAM TOOLS AND TUTORIALS

3.1 The Faust IDE, generating WAM tutorials…

Faust is a popular functional programming language for DSP programming; thousands of source code for audio effects, instruments, and more generally DSP algorithms, filters etc. are available in the open source community, or included in the distribution and in the online IDE [4, 14]. The Faust compiler supports exporting to a variety of platforms and standards, including Web Audio Modules. Since 2014, Faust DSPs can be compiled to JavaScript-compatible binary code and dynamically run the DSP within the browser [5]. In 2022, a new version of the Faust WebAssembly compiler named `faustwasm`, which provides TypeScript and JavaScript wrappers for Faust DSPs, was released [15]. It allows to generate static self-contained html pages or JavaScript modules (including the Faust code as a WebAssembly module and various additional resources), or even to integrate the libfaust compiler in applications which need to dynamically compile and deploy Faust DSP programs. The library can be used either in Node.js based projects or in web browsers and is published on NPM. Furthermore, an official online Faust IDE has been developed since 2019 using modern web technologies such as WebAssembly and AudioWorklet, offering various testing, debugging and audio visualization features, allowing connecting to different kinds of audio/MIDI inputs and outputs, making easier the development of wasm WAM plugins, with a standard CSS based GUI (auto-generated, see Figure 3) or with a custom GUI that can be designed with an embedded GUI Builder (Figure 4) [10, 16]. A step by step tutorial about how to build WAM plugins is available online[7]. Starting from an existing Faust code, it takes a few seconds to execute in the IDE, verify, export (download as a zip file), publish online, and run a WAM plugin (Figure 5).

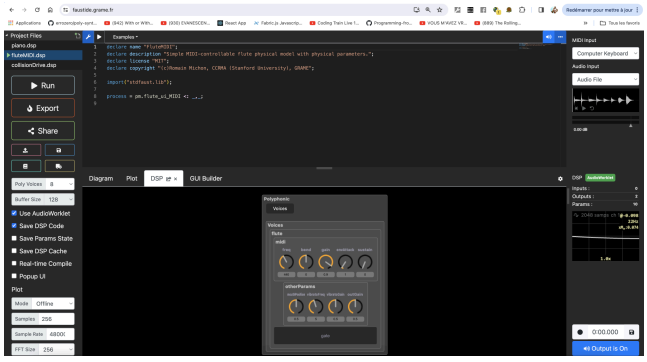

Figure 3: a physical modeled flute in the FAUST IDE.

Based on the `faustwasm` module, `faust2wam`[8] has been recently developed: a JavaScript tool that can generate self-contained Faust WAMs within the Node.js environment,

[7] http://tinyurl.com/yckdyax4
[8] https://github.com/Fr0stbyteR/faust2wam

or dynamically within the browsers. In addition, support has been added for polyphonic instruments and Faust-based spectral processors. These new generation targets (`web/wam2-ts`, `wam2-poly-ts`, and `wam2-fft-ts`) are now available in the Faust IDE in the Export window.

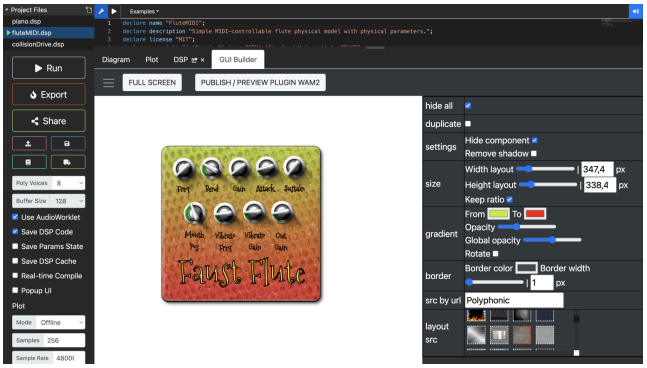

Figure 4: The same flute with a custom GUI made with the embedded GUI Builder.

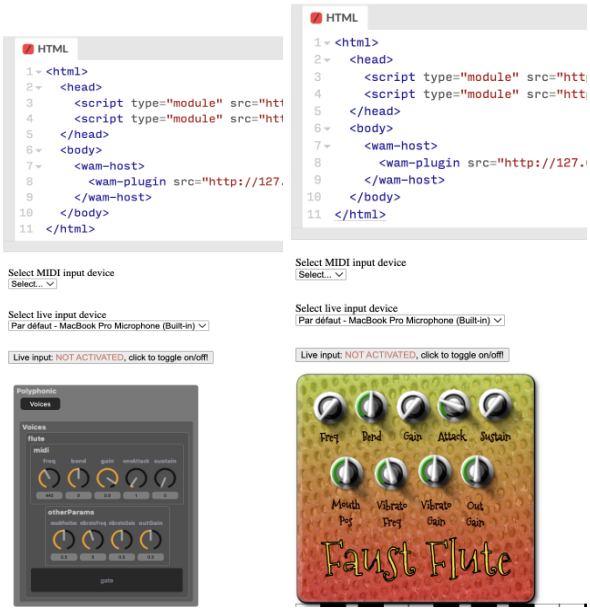

Figure 5: the exported WAM can be used by an external host, here in codePen[9]. Left with standard CSS-based generated UI, right with a custom GUI.

3.2 Online tutorials

A set of seven tutorials has been published[10]. They show how to load from JavaScript a WAM plugin, attach it to a web audio graph, how to chain it with other audio modules, how to perform parameter automation at the frequency rate, how to send MIDI events to a WAM instrument (using a WAM virtual piano keyboard, or a piano roll WAM, see Figure 6). It also shows how to write an audio player as an audio worklet using DSP code written in C++ and cross compiled to web assembly.

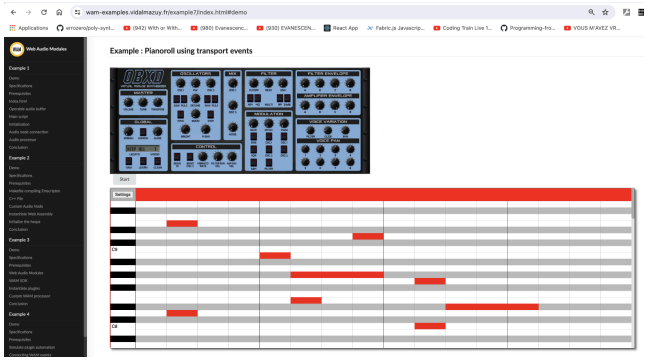

Figure 6: one of the online tutorials showing how to use transport events with a piano roll WAM sending events to a WAM instrument (Oberheim OB-Xd emulation).

3.3 `<wam-host>` and `<wam-plugin>` Web Components

These two Web Components, have been developed to facilitate the embedding of WAM plugins in a Web Page, for demo purposes or just for testing if a WAM runs correctly.

```
<wam-host>
    <wam-plugin src="URI_OF_WAM1">
    <wam-plugin src="URI_OF_WAM2">
    <wam-plugin src="URI_OF_WAM3">
</wam-host>
```

The previous code will display a chain of three plugins, and depending on the type of the plugins and their location in the chain, different elements will be also generated in the demo page: a virtual MIDI keyboard with a MIDI input device selection menu (for WAM instruments), an audio player with a button for activating live input, along with an audio device selection menu, etc. as shown in the examples of Figure 5 (see the codepen for source code).

4 Available WAM plugins and hosts overview

Figure 7: the wam-bank pedalboard WAM acts as a complete pedalboard of audio effects.

[9] https://codepen.io/w3devcampus/pen/YzgwxMV?editors=1010

[10] Online version: https://wam-examples.vidalmazuy.fr/, also on GitHub repo: https://github.com/Brotherta/wam-examples

The Wam-bank pedalboard (Figure 7) is a WAM that acts as a host for managing chains of effect plugins. It comes with a preset manager, with about 30 different plugins (mainly from the wam-community initiative presented in section 2), it can filter WAM plugins by category and the set of plugins is easily extensible. It comes with an online standalone demo[11], exposes its URI for making it embeddable in external hosts[12] (like in the Wam-Studio DAW, see next section).

Figure 8: tube guitar amp simulators.

Tube Guitar Amp Simulations: a Vox AC30 and a Marshall JCM800 (Disto Machine) [1, 2] are already shipped with the WAM2 distribution and available in the previous pedalboard along with many classic effect pedals for guitarists. Other WAM1 amp simulations exist for Metal, Classic Rock/Blues and Clean sounds [18], with a complete simulation of all the tube guitar amp stages. Authors are porting them to WAM2 (Figure 8), see demos on video[13].

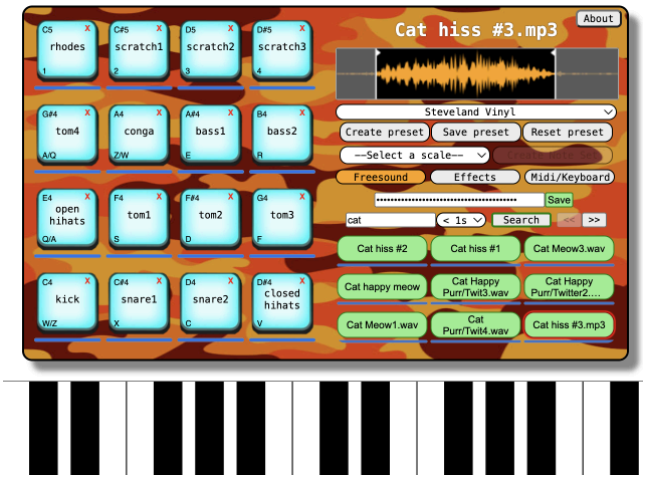

Figure 9: a freesound.org powered sampler.

WAM-sampler (Figure 9) is an open source creative sampler[14] that comes with a set of standard sounds (drums, piano, hip hop, etc.) triggered by MIDI events (it supports velocity). Its main originality is that it can also fetch sounds from the millions available on freesound.org, preview them, trim them, adjust their ADSR envelope, volume, panning, and even adjust audio effects before dragging and dropping them onto the sampler's control pads. Furthermore, a powerful note set generator can automatically assign the pads with a sound made up of different notes in a variety of scales. An online demo[15] and a video presentation is available[16].

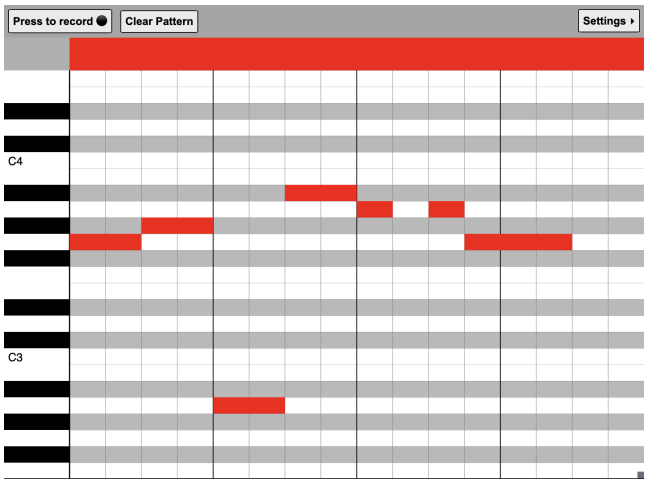

Figure 10: a piano roll MIDI sequencer.

Piano Roll (Figure 10) is an open source polyphonic MIDI piano roll sequence editor. In hosts that support the

[11] https://wam-bank.i3s.univ-cotedazur.fr/
[12] See this codepen example: https://codepen.io/w3devcampus/pen/BabzVvP
[13] https://mainline.i3s.unice.fr/jaes2023/userTestsAndLatency.html
[14] https://github.com/micbuffa/WAMSampler
[15] https://codepen.io/w3devcampus/pen/ZEPOgar?editors=1010
[16] https://www.youtube.com/watch?v=3jf2KCdwU1A

Patterns WAM Extension, a single Piano Roll WAM instance can hold multiple pattern clips of varying lengths. MIDI note events are generated with sample-accurate timing by sending the WAM note events from within the AudioWorklet.

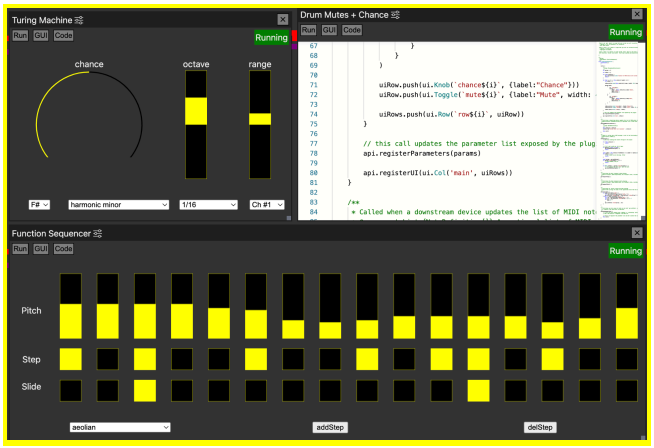

Figure 11: a live-coding MIDI sequencer.

The WAM Function Sequencer (Figure 11) is an open source live-coding MIDI sequencer environment developed to reduce the needed effort for Javascript developers to iterate on a MIDI sequencer idea. The patch developer uses Javascript to register a custom interface and code event handlers, without needing to worry about build processes or code hosting. In collaborative WAM hosts, the patch code can be edited simultaneously by multiple participants.

4.2 Remarquable WAM hosts

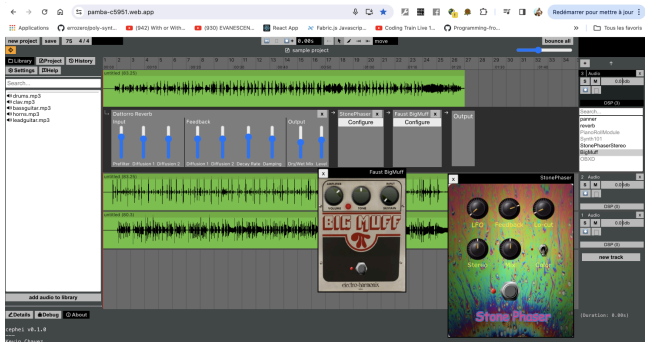

Figure 12: CEPHEI: a React-based DAW that supports WAM effects and instruments.

CEPHEI[17] (Figure 12) is a React based DAW still in the early stage of implementation, by Kevin Chavez (@aykev). The author focused on GUI and ergonomy. At the time of writing, the DAW features audio playback, recording, bouncing, and a chain of audio FX associated with each track. The author is working on a WebGPU shader to render out the waveforms at different scales. All track elements are DOM based (no canvas, track regions are simple html divs, waveforms are CSS backgrounds, etc), an original and efficient approach.

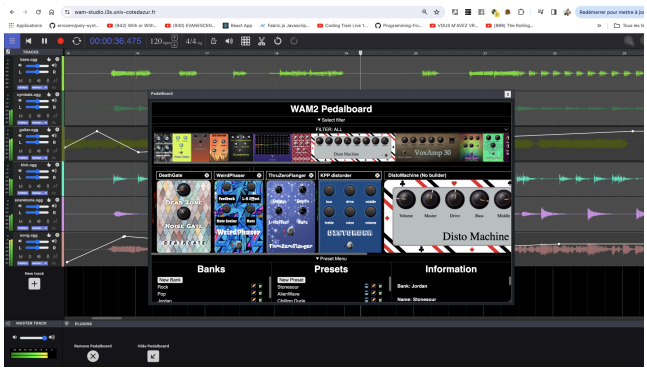

Figure 12: Wam-Studio, an open source DAW.

WAM Studio (Figure 12) is an open source, online Digital Audio Workstation (DAW) developed as a demonstrator of Web Audio Modules, as well as of recent W3C Web APIs, such as Web Assembly, Web Components, Web Midi, Media Devices etc. DAWs are feature-rich software and therefore particularly complex to develop in terms of design, implementation, performances and ergonomics. Very few commercial online DAWs exist today and the only open-source examples lack features (no support for inter-operable plugins, for example) and do not take advantage of the recent possibilities offered by modern W3C APIs (e.g. AudioWorklets/Web Assembly). See [11] for a survey of online Digital Audio Workstations (DAWs).

In Wam-Studio, each audio track is a WAM processor (i.e a subclass of an AudioWorkletProcessor) for playing or recording audio buffers. The main reason is that when both host and plugins are implemented this way (as Wam-Processor/AudioWorklet) then the WAM SDK provides under the hood optimized host/plugin communications that avoid crossing the audio thread. This way, parameter automation at the sampling rate, even with hundreds of parameters, can be done simply using Shared Array Buffers. Other events can also be sent this way without crossing the main/audio thread barrier, avoiding the need to schedule midi events in advance, as described in the famous “A tale of two clocks” Chris Wilson article[18].

In addition, Wam-Studio is also a good implementation example about how to record audio tracks in sync with other tracks playing at the same time. In [16] authors describe how they calibrate the input latency and perform latency compensation, as well as how they reused Paul Adenot’s ring buffer implementation[19] in a multi-thread architecture for a robust recording solution: i.e no glitches if the UI is resized or cpu stressed, while recording a waveform is redrawn regularly as the recording progresses, etc.[20]

Wam plugins can be associated with tracks too: a rich pedalboard handles audio FX chains, featuring a preset manager and filters for managing a large set of WAM

[17] https://pamba-c5951.web.app/

[18] https://web.dev/articles/audio-scheduling

[19] https://blog.paul.cx/post/a-wait-free-spsc-ringbuffer-for-the-web/

[20] Video: https://www.youtube.com/watch?v=0r6pox2eQH0

plugins (standard audio effects, original ones), and can be used standalone or embedded in other WAM hosts.
The DAW is online[21] and the GitHub repository[22] contains both front-end and back-end source code, as well as a Docker image configuration file, making the deployment of a new instance easy. Wam-studio project management source code (front and back-end) shows how a large set of .wav files, along with the state of the different components used by a project, can efficiently be saved or restored from a remote server. Wam-Studio can render/bounce the final mix or individual tracks using OfflineAudioContext, applying effects and automation.

Figure 13: Attune specific macro editor.

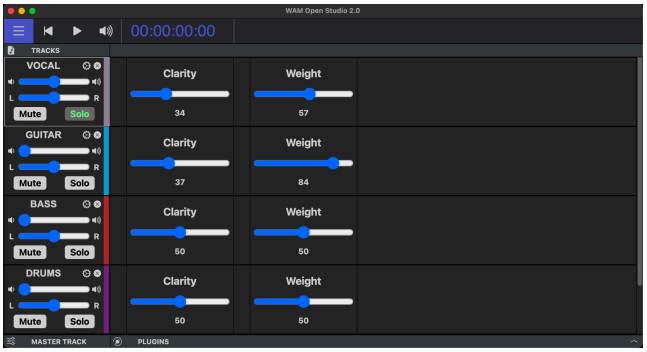

Figure 14: Simplified view of Attune allowing DHH users to adjust sound properties.

Attune is an open source Wam-Studio fork[23] that is being used by researchers from the REMI group at CCARMA/Stanford to Empower Cochlear Implant Users. With Attune, researchers can associate individual tracks with macros that control multiple WAM plugin parameters at once (Figure 14). While programming macro controls and customizing track parameters might have many applications in the music industry, they also present an opportunity to afford D/deaf or Heard-of-Hearing (DHH) users greater control over their music listening. In [12], authors present a case study illustrating how this tool could be used by Hard-of-Hearing users to modify individual musical elements in a multi-track listening context to create a more enjoyable listening experience. Figure 14 presents the simplified view for DHH users.

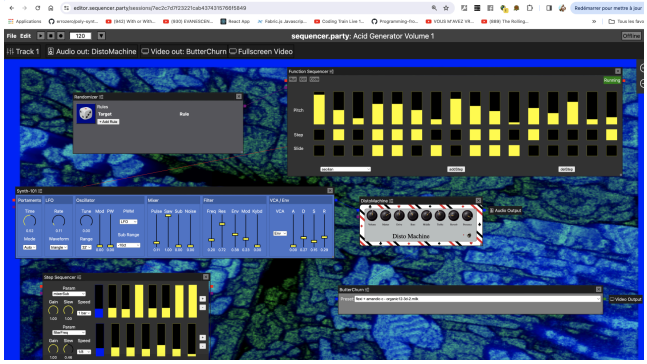

Figure 15: sequencer party, a realtime collaborative musical experience.

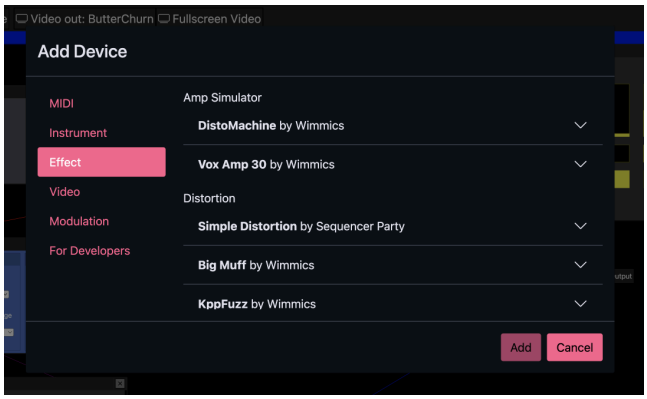

Figure 16: sequencer.party menu for adding a WAM plugin (sequencer, instrument, effect, display) to a track/screen.

Sequencer.Party[24] (Figure 15) is a realtime collaborative audio/visual platform built entirely out of WAMs. Users work together in real-time sessions, and can share WAM presets and projects publicly on the website. It comes with its own collection of open-source WAMs[25], and users may load remote WAMs by URI. An online integrated menu exposes a categorized list of plugins available,by querying the wam-community REST API (Figure 16).

[21] https://wam-studio.i3s.univ-cotedazur.fr/
[22] https://github.com/Brotherta/wam-studio
[23] https://github.com/Brotherta/wam-studio/tree/stanford-prototype
[24] https://sequencer.party
[25] https://github.com/boourns/burns-audio-wam

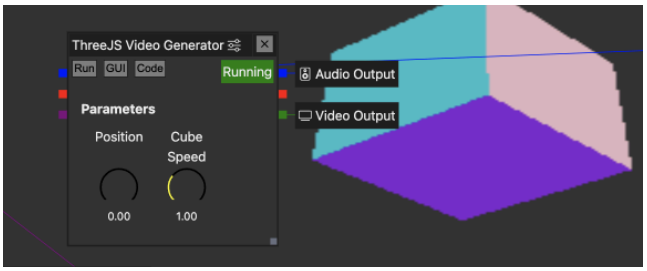

Figure 17: a WAM in Sequencer Party that can animate 3D scenes using the ThreeJS library. It includes a Visual Studio component for live coding the 3D rendering that can react to the input audio signal.

The WAM Extension system: Tom Burns, author of sequencer party, is a core member of the group who designed and published the WAM SDK. He designed the WAM extension system, and some of the open source plugins proposed in sequencer.party are good demonstrators of video extensions, WebGL extensions (Figure 17), modulation WAMs (WAMs that modulate other WAM parameters), etc. See section 5 about Wam-extensions.

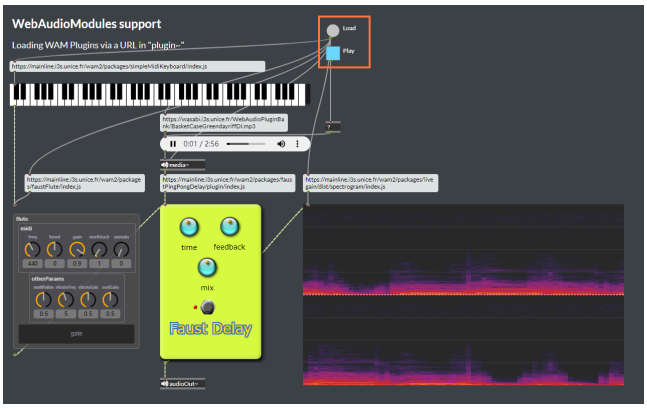

Figure 18 : WebAudioModules in a JSPatcher patch.

Figure 19: WAM plugins in the JSPatcher audio editor.

JSPatcher [9, 13] is an open source, online visual programming language[26] in the style of Max/PureData for interactive programming, audio processing, and realtime multimedia projects. It can be used as a graph editor for WebAudio nodes and WAMs. (Figure 18). It also comes with a powerful audio buffer editor that can use WAM plugins to apply effects to buffer regions (Figure 19).
Figure 20 shows an example in JSPatcher where a spectral denoiser WAM plugin is loaded from a URI. The plugin processes the microphone input signal and two spectroscopes display the result.

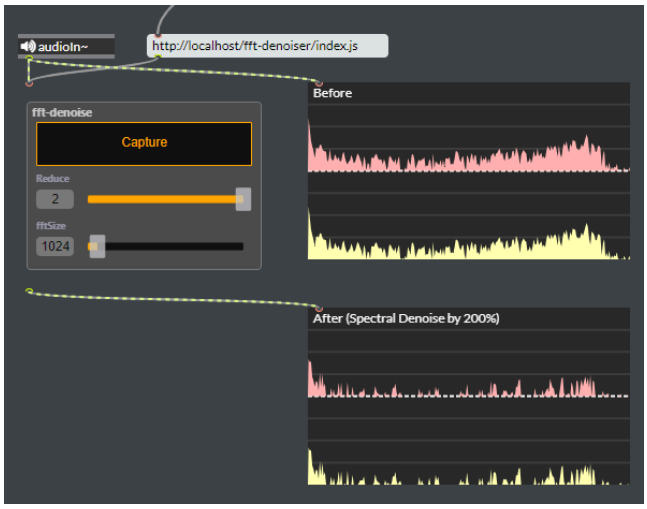

Figure 20: JSPatcher - a Max/MSP-like host - with some WAMs (a denoiser on the left and some audio visualizations on the right).

5 WAM EXTENSIONS

WAM Extensions[27] are optional additions to the WAM 2.0 API that add tighter integration between WAM plugin and host, better solving user interface problems and creating new use-cases for WAMs to solve.

5.1 EXAMPLE WAM EXTENSIONS

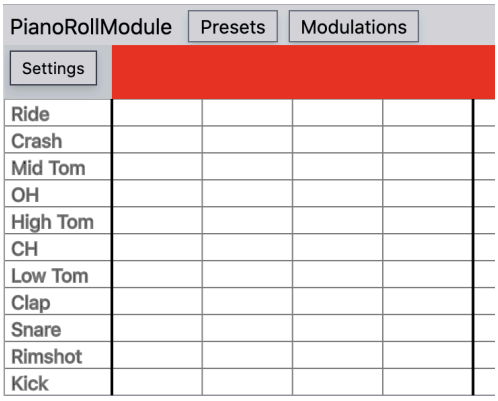

Figure 21: Piano Roll with note names, demonstrating the Notes WAM extension

[26] https://fr0stbyter.github.io/jspatcher/dist/

[27] https://github.com/boourns/wam-extensions

The Notes extension allows one WAM to publish a list of relevant notes with names, and other connected WAMs to receive the note list. This enables a better user interface when sequencing typical drum machines or samplers where only certain MIDI notes can be received (Figure 21).

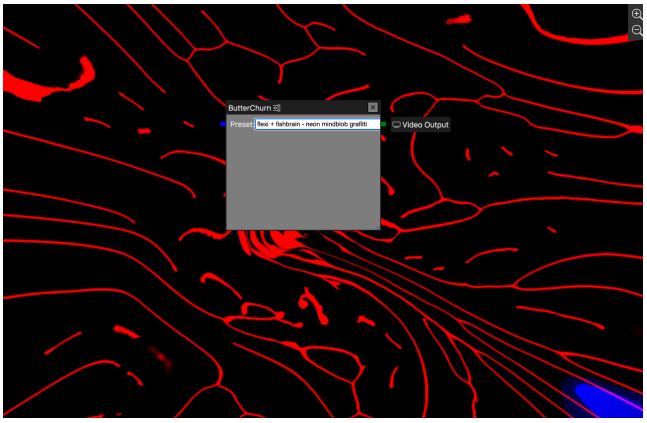

Figure 22: Butterchurn plugin demonstrating the Video WAM extension.

The Video extension adds video generation and processing capabilities to the WAM ecosystem. The WAM host manages a WebGL2 context, and plugins register render handlers to generate and process WebGL2 textures. Video WAMs can also process audio, allowing audio reactivity or simultaneous audio/video stream generation.

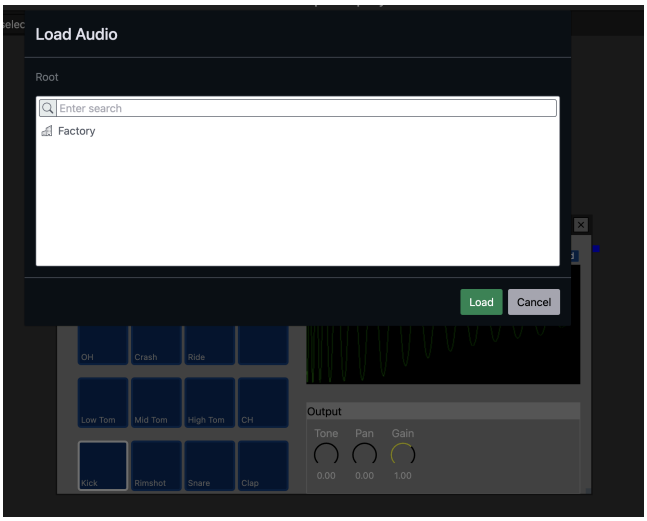

Figure 23: Drum Sampler WAM using the Asset Extension to allow the host to control asset load/save.

With the Asset extension, WAM hosts control asset loading and saving. Individual WAM plugins can rely on the host for cloud storage, and users can manage all files related to a musical project in the host without having each WAM plugin store files in separate cloud services or accounts (Figure 23).

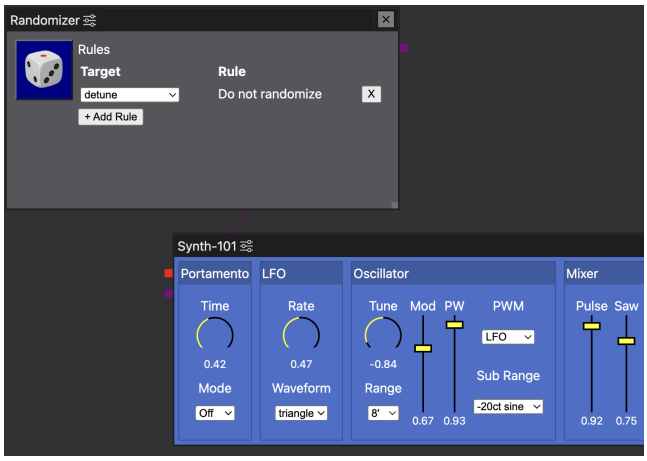

Figure 24: Randomizer plugin uses the Modulation Target extension to randomize the Synth-101 patch.

The ModulationTarget extension enables plugin developers to create plugins whose purpose is to modulate, or control, the parameters of another WAM plugin (Figure 24). WAM Modulation plugins exist for parameter randomization, parameter sequencing, envelope following of an audio signal and LFO control. By using modulation WAMs, hosts enable deep parameter automation similar to modular synthesis.

6 CONCLUSION / DISCUSSION

Today, Web Audio Modules is the principal Web Audio plug-in system maintained and in constant evolution. The examples proposed when it was launched in 2021 showed great potential, but at the time the lack of tutorials, documentation, development tools and a reference website made the first steps difficult. Today this has changed, and this article has summarized some of the most remarkable contributions from which the Audio web developer community can benefit. A tool like Faust IDE lets you design, test and publish polyphonic WAM instruments or audio effects very quickly. The Wam-community initiative is also particularly important, as it facilitates the exchange and reuse of existing plugins, and we hope that other developers will contribute to its content. Several online DAWs are WAM-compatible, and at least one is open source and uses WAM in its core design (Wam-Studio). Examining its source code is a good way to understand how to design and implement basic mechanisms such as robust recording in sync with other audio tracks, latency compensation, project rendering/bouncing, load/save projects with large audio content, plugin states, parameter automation etc.

Last but not least, WAM's extension mechanism also opens up many new perspectives, notably by introducing the multimodal aspect (audio and image) that was missing from

the initial proposal. New types of applications will certainly emerge, such as the collaborative application sequencer.party, the first host application to take advantage of WAM extensions.

REFERENCES

[1] M. Buffa and J. Lebrun. Real time tube guitar amplifier simulation using WebAudio. *Web Audio Conference* (WAC 2017). London, UK.

[2] M. Buffa and J. Lebrun. Web Audio Guitar Tube Amplifier vs Native Simulations. *Web Audio Conference* (WAC 2017). London, UK.

[3] J. Kleimola and O. Larkin. Web audio modules. *12th Sound and Music Computing Conference* (SMC15). Maynooth, Ireland.

[4] Y. Orlarey, D. Fober, and S. Letz.. Syntactical and Semantical aspects of Faust. *Soft Computing* 8, 9 (2004). 623–632.

[5] S. Letz, Y. Orlarey, and D. Fober. Compiling Faust Audio DSP Code to WebAssembly.*Web Audio Conference* (WAC 2017). London, UK.

[6] M. Buffa, J. Lebrun, J. Kleimola, O.Larkin, and S. Letz. Towards an open Web Audio plugin standard.2018, April. *The Web Conference 2018,* Lyon, France (pp. 759-766).

[7] M. Buffa, J. Lebrun, J. Kleimola, O.Larkin, G. Pellerin, S. Letz.. WAP: Ideas for a Web Audio plug-in standard. *Web Audio Conference (WAC 2018),* Berlin, Germany.

[8] M. Buffa, J. Lebrun, S. Ren, S. Letz, Y. Orlarey, and al.. Emerging W3C APIs opened up commercial opportunities for computer music applications. *The Web Conference 2020 - DevTrack*, Apr 2020, Taipei.

[9] S. Ren, L. Pottier, M. Buffa. Build WebAudio and JavaScript Web Applications using JSPatcher: A Web-based Visual Programming Editor. *Web Audio Conference 2021*, Barcelona, Spain. ⟨hal-03519504⟩

[10] S. Ren, S.Letz, Y. Orlarey, R. Michon, D. Fober, et al. FAUST online IDE: dynamically compile and publish FAUST code as WebAudio Plugins. *5th Web Audio Conference, WAC 2019* Trondheim, Norway.

[11] M. Buffa, and A. Vidal-Mazuy. "WAM-studio, a Digital Audio Workstation (DAW) for the Web." *In Companion Proceedings of the ACM Web Conference 2023*, pp. 543-548. 2023.

[12] M. Buffa, A. Vidal-Mazuy, L. May & M. Winckler, (2023, August). WAM-Studio: A Web-Based Digital Audio Workstation to Empower Cochlear Implant Users. *In IFIP Conference on Human-Computer Interaction (pp. 101-110*). Cham: Springer Nature Switzerland.

[13] S. Ren, L. Pottier, M. Buffa, and Y. Yu, 2022. JSPatcher, a Visual Programming Environment for Building High-Performance Web Audio Applications. *Journal of the Audio Engineering Society,* 70(11), pp.938-950.

[14] Y. Orlarey, D. Fober, and S. Letz. FAUST : an Efficient Functional Approach to DSP Programming. In E. D. France, editor, *New Computational Paradigms for Computer Music*, pages 65–96. Paris,France, Jan. 2009.

[15] S. Ren, S. Letz, Y. Orlarey, D. Fober, R. Michon, M. Buffa, L. Pottier, and Y. Yu. Modernized Toolchains to Create JSPatcher Objects and Web Audio Modules from Faust Code. *In Proceedings of the International Web Audio Conference, Cannes, France*, July 2022. Universitéé Côte d'Azur.

[16] S. Ren, S. Letz, Y. Orlarey, R. Michon, D. Fober, M. Buffa, J. Lebrun. Using Faust DSL to develop custom, sample accurate DSP code and audio plugins for the Web browser. *Journal of the Audio Engineering Society*. 2020 Nov 30;68(10):703-16.

[17] M. Buffa, S. Ren, O. Campbell, T. Burns, S. Yi, J. Kleimola, O. Larkin. Web Audio Modules 2.0: An Open Web Audio Plugin Standard. InCompanion *Proceedings of the Web Conference* 2022 Apr 25 (pp. 364-369).

[18] M. Buffa, J. Lebrun. Rocking the Web With Browser-Based Simulations of Tube Guitar Amplifiers. *Journal of the Audio Engineering Society.* 2023 Nov 16;71(11):753-68.