Using Web Audio Modules for Immersive Audio Collaboration in the Musical Metaverse

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

Dorian Girard
Université Côte d'Azur,
Sophia-Antipolis, France
dorian.girard@etu.unice.fr

Ayoub Hofr
Université Côte d'Azur, I3S, CNRS, INRIA, EMSI
Sophia-Antipolis, France
ayoub.hofr@inria.fr

***Abstract*— W3C Web standards now enable online applications that were unthinkable just a few years ago. In 2020, the Web Audio Modules V2 (WAM) initiative, known as "VST for the web", revolutionized music on the web. WAMs facilitate host applications using reusable web components such as note generators, virtual instruments and audio effects. Today, the WAM ecosystem includes over 100 plugins, tutorials and host applications such as Sequencer Party, a real-time collaborative platform built entirely with WAMs, that allows multiple participants to build music installations by patching plugins together. This article adapts the concepts of Sequencer Party to the musical metaverse, reusing existing WAMs without modification and generating new 3D interfaces for VR/XR use. BabylonJS was used for 3D rendering with WebXR support, and a "Conflict-Free Replicated Data Type » (CRDT) approach was chosen for client synchronization. The software is open-source and hosted online.**

Keywords*— *Musical Metaverse, WebXR WebAudio, WebMIDI, Web Audio Plugins, Web standards, Web Audio Modules, CRDT, Collaborative Experiences

I. Introduction

The Musical Metaverse (MM) [1] is a concept that refers to an immersive virtual space dedicated to musical activities. It is an extension of the "Metaverse" concept, which defines a set of interconnected virtual worlds where users can interact with each other and with digital objects. The applications of the musical metaverse are varied: virtual concerts, educational applications, shared and/or collaborative musical performances, etc. Transposing user experiences into conventional 2D applications generally does not work in an immersive environment, and new ergonomic and sensory approaches need to be employed. In the scientific literature, however, there are still very few publications on this subject, as stated by Boem and Turchet and in [2], while the challenges are many, concerning application security, user experience, interactions for shared musical applications, constraints, benefits and innovation potential of web browser applications on VR/XR devices, and so on. This article does not address all these points, but focuses on a persistent, real-time multi-participant immersive world for shared music installation creation, exploiting recent W3C web standards such as Web Audio, Web MIDI, WebXR, WebGL, WebGPU, WebAssembly, WebRTC now implemented in the web browsers of the most common VR/XR headsets available on the market. One of the authors of this paper is the W3C Advisory Committee Representative from his university[1]. He has been participating since 2015 in the W3C Web Audio Working Group in charge of Web Audio and Web MIDI standards and is actively involved in the emergence of the open-source Web Audio plug-in standard (aka "VSTs for the web") called "Web Audio Modules" (or "WAM"). The work presented here, WAM Jam Party, has been designed as a "WAM host", based mainly on the use of WAM plugins (note generators, virtual instruments, audio effects) in a multi-participant universe.

Section II presents the audio technologies used, section III details how existing concepts for collaborative music creation have been ported into an immersive application, section IV details the UX design used, section V is about users' 3D representations, section VI presents usability tests conducted, while sections VII and VIII present and compare the related works and deal with limit and possible enhancements.

II. Audio Technologies in WAM Jam Party

The Web Audio API

The W3C Web Audio API, as of 2021, is a frozen standard [2] providing a set of "audio nodes" for sound processing or generation. These nodes are used in JavaScript by instantiating API classes and can be connected to form an "audio graph." Sound moves through this graph at the sampling rate and undergoes various transformations. Nodes include wave generators, sound sources like microphone inputs or loaded sound files, and sound transformers. The API comes with a limited set of "standard" nodes for common operations such as gain control, frequency filtering, delay, reverb, dynamics processing, 2D and 3D sound spatialization, and so on. Real-time Audio libraries/APIs typically divide tasks between a control thread and a rendering thread ([4], [5], [6]). The Web Audio API follows this model, with a high priority "audio thread" dedicated to processing sound and delivering audio samples to the operating system under strict real-time constraints. Failure to deliver samples on time results in audible anomalies. The control thread, also called "the main thread", is of lower priority and executes the user interface's JavaScript code, makes calls to the Web Audio API, and manages modifications to the audio graph, such as connecting/disconnecting nodes and adjusting their parameters. All nodes except one, in the Web Audio API, are "pre-made building blocks". They are written in C++ (or Rust in Firefox) for high performance and their DSP algorithms are

[1] An "AC Rep W3C" is a person who represents their organization on the Advisory Committee of the W3C, contributing to the development of web standards.

[2] A "recommendation", the highest level of standardization according to the W3C, see https://www.w3.org/press-releases/2021/webaudio/.

hard coded in the browser source code and cannot be modified, only their predefined parameters can be adjusted. The notable exception is the AudioWorklet node, introduced in 2018, which allows for low-level custom audio processing within the audio thread [7]. With the Web Audio API, developers can assemble nodes into a graph to create complex audio effects and instruments, such as delay effects, auto-Wah, chorus, distortion, synthesizers, and samplers. Over the years, many high-level audio effects and instruments have been developed using this approach [8].

However, it is often necessary to chain effects and audio instruments, as in a guitarist's pedalboard or when composing / producing music in DAWs, where multiple effects and instruments are used to process tracks or regions. In these cases, the Web Audio API is too low-level, requiring a higher-level unit like a native audio plugin [9]. Until 2015, there was no high-level standard for "inter-operable web audio plugins" and "host" applications on the Web platform [10]. Several initiatives have since been launched, one of which has become a "community standard", described in detail in the following section.

Figure 1: a drum sampler, and two distortion effect WAMs.

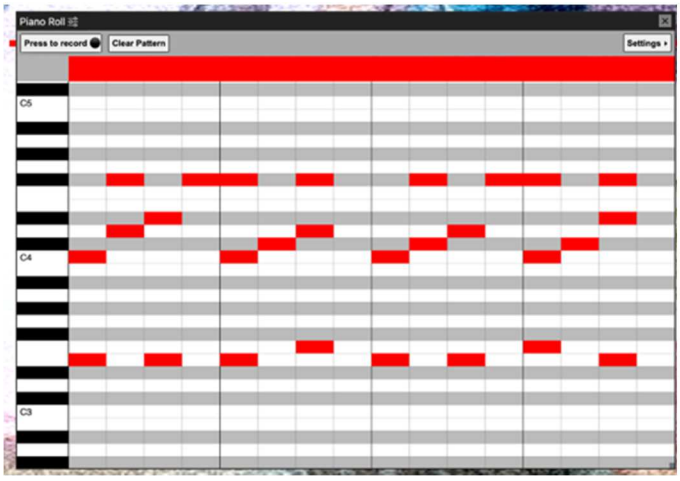

Figure 2: Piano roll WAM.

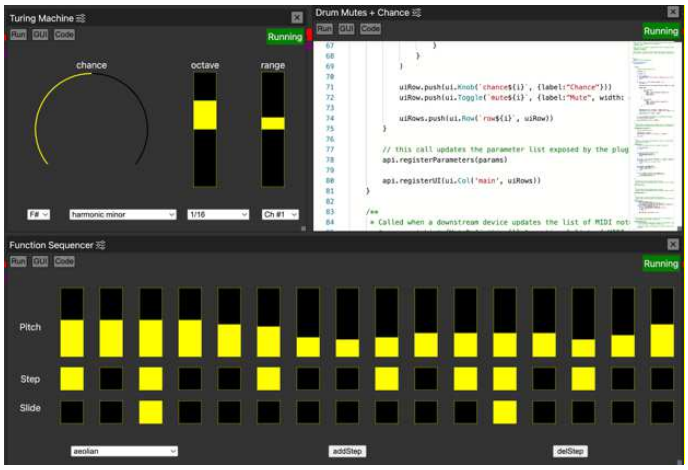

Figure 3: A programmable Step Sequencer WAM.

Web Audio Modules

In 2015, Jari Kleimola and Olivier Larkin proposed the "Web Audio Modules" (WAM) standard for Web Audio plugins [10] and in 2021, Web Audio Modules version 2.0 was released, establishing a community standard (API, SDK) that offers developers more freedom (support for build tools, TypeScript, frameworks like React, etc.), improved performance, simplified access to plugin parameters, and easier integration into online DAWs [11]. Dozens of WAM plugins are available through the WAM Community Endpoint and can be easily integrated into host applications using their URI[3]. We are talking here about elements comparable in quality and complexity to those existing in the world of native applications (some instruments and effects are in fact Web Assembly ports of the source code of true native plugins written in C++). The WAM ecosystem is evolving rapidly, for example, as an extension system is now supported [12], allowing WAMs to manipulate video input and outputs, to do 3D rendering or WebGL shader animations, to support parameter modulation or to load/save assets from the host storage (i.e. useful for samplers). Available WAMs include *note generators* such as: a piano roll (Figure 2), a programmable step sequencer (Figure 3, Figure 4), random note generators, chord generators, *virtual instruments*: synthesizers (Figure 4), samplers (Figure 1), physical modeling of instruments (flute, clarinet, brass, djembe), and *audio effects* (more than one hundred available, including all classics: flanger, chorus, reverb -multiple types-, distortion, fuzz, overdrive, envelope followers, etc.).

Sequencer Party (Figure 6, 4, 5) is a real-time collaborative audio/visual platform built entirely out of WAMs that allows building and adjusting music installations. Users work together in real-time sessions and can share WAM presets and projects publicly on the website. It supports a large collection of open-source WAMs, including all the ones published by the WAM Community initiative[4]. An integrated menu exposes a categorized list of available plugins (Figure 5). A user acts as "master of ceremonies", creates a session and then invites other participants by sending a link. Each "screen" is a "track" (Figure 6), and sessions can have many

[3] The WAM Community plugins and API is available from the https://www.webaudiomodules.com/ web site.

[4] https://www.webaudiomodules.com/docs/community/

tracks. A mixer for adjusting the volume of each track is provided.

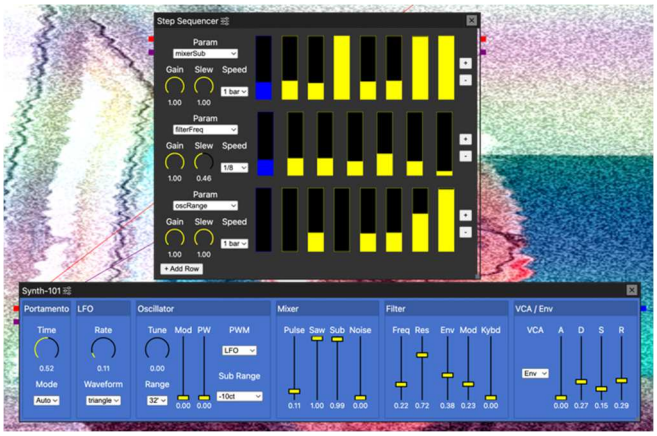

Figure 4: The same component (programmed differently) from the previous picture, used to modulate parameters of a WAM synthesizer.

Examples of multi-participant realtime WAM hosts

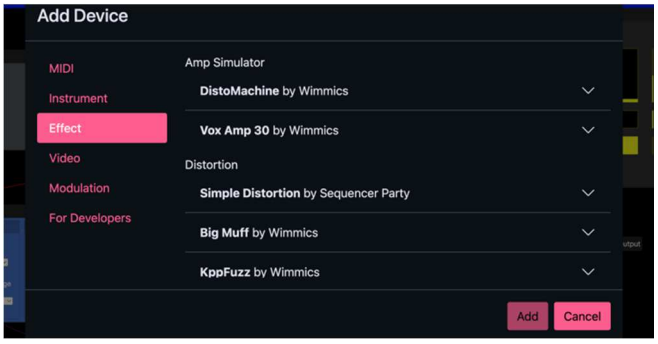

Figure 5: Sequencer Party menu for adding WAMs into the playground.

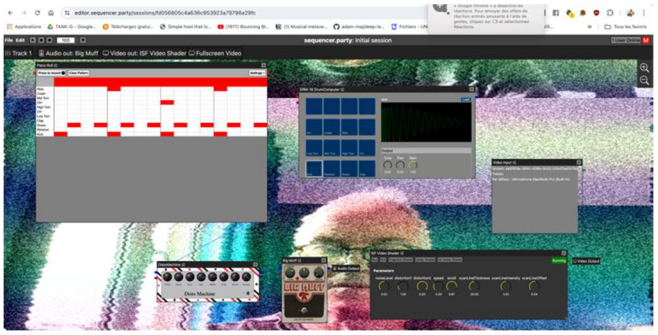

Figure 6: a typical track in Sequencer Party, it's a screen!

Participants can intervene at any time, and every change in their session state (adding, deleting, moving, connecting, disconnecting a WAM or modifying a parameter) is propagated to the server and other clients. Global host parameters, such as tempo, tracks or global mix values are also shared. *Rhythm synchronization is done locally*, so if one adds a drum machine, a step sequencer and a bass generator, all these WAMs will be synchronized locally; if the tempo changes, it will also change on all clients, and synchronization will be preserved. Sequencer Party has been used on numerous occasions to broadcast "parties" on Twitch and has succeeded in gathering a community of "online electronic musicians". The sessions are all persistent and can be restored any time.

III. Transposing the Sequencer Party paradigm to immersive worlds

Soon after the first publications on the musical metaverse and its eponymous research project, the idea of transposing Sequencer Party into a shared immersive 3D universe was born. This also raised several challenges: the Web-based 2D elements had to be transposed into a universe created with WebXR technology, Sequencer Party being close source, a solution had to be designed and implemented to synchronize clients while retaining the idea of a persistent world, and above all, how to use WAM plugins via a VR interface? How to design 3D GUIs for existing WAMs? And finally, how to make it all usable in a simple way?

WebXR technologies used

WebXR [5] is the W3C standard for Web applications running in VR/AR headsets. Although this standard is still evolving[6], implementations are available in the web browsers of the most common headsets on the market, such as the Oculus series marketed by Meta (Oculus 2, 3 and Pro). The WebXR API provides key low-level functionalities such as stereoscopic rendering for immersive 3D perception, precise tracking of head and controller movements and position tracking for complete immersion. It is also compatible with higher-level 3D frameworks such as BabylonJS[7], Three.js[8] and A-Frame[9]. WebGL high-level 3D libraries such as ThreeJS and BabylonJS offer high-level functionality and greatly simplify the process of creating interactive 3D scenes by providing abstractions for manipulating 3D objects, cameras, lights and materials, and come with several physics engines. They are used for the development of data visualizations, interactive animations, 3D games and, more recently, WebXR applications. Their functionalities are very similar, but some advantages can be found on certain points in one or the other: BabylonJS integrates, for example, the Havok physics engine used by the main AAA games, whereas ThreeJS has served as the basis for additional engine layers such as the Needle Engine[10], which simplifies the integration with Unity, and for A-Frame and its networked A-Frame extension[11], which facilitates the creation of multi-participant WebXR applications using a declarative approach. Note that Turchet and Boem successfully used networked A-Frame for their work on the "Musical Metaverse Playgrounds" [2].

BabylonJS: for the presented application, BabylonJS was chosen for several reasons: it supports spatialized sound via components exploiting the Web Audio API, it is possible to access the audio context of BabylonJS scenes and thus connect WAM plugins such as note generators, instruments, audio effects to the web audio graph, we had a long experience with this library, used in the 3D programming courses we give to our master students, and finally, total control over the network / synchronization layer was essential, as the approach

[5] https://www.w3.org/TR/webxr/
[6] It is currently at "*Candidate Recommendation Draft*" status according to the W3C - this means that the specification is considered stable and has passed several stages of revision and testing.
[7] https://babylonjs.com/
[8] https://threejs.org/
[9] https://aframe.io/
[10] https://needle.tools/
[11] https://github.com/networked-aframe

we were considering (state synchronization like Google Docs) was not the one proposed by networked A-Frame, for example.

State synchronization between clients

WAM states: the Web Audio Modules API specification[12] specifies that each WAM must implement the methods of the `WamNode` interface which includes the `getState()` and `setState(…)` methods for managing its internal state. Most of the time, the state of a plugin is made up of the value of all the plugin's parameters, and this is implemented by default by the WAM SDK classes that most developers use. However it is possible to redefine the `getState()` and `setState(…)` methods to manage custom states.

Sync states instead of sending events for each change: the systematic presence of these methods makes it possible to globally synchronize the states of all WAM plugins present in a virtual immersive universe. To achieve this, each client performs state serialization/deserialization cycles several times a second. To the internal state of a WAM, we add its position and orientation in 3D space and its connections in the WAM graph. Special states are also created: one for the global host application parameters such as the tempo and the client id, and one for the 3D avatar of the user (position and orientation). In parallel, we receive state updates several times a second. In this case, we perform the opposite operations, calling the WAMs' `setState(…)` methods and updating their positions and orientations, connections and we restore also global parameters and other avatar positions and orientations. In complex graphs, composed of Web Audio Modules that may include several dozen parameters (this is the case of some WAM instruments), the state of a WAM may be a large object, and the sum of all these states of an ongoing session composed of dozens of WAMs can reach some kilobytes.

States as text documents: given the nature of the states we need to synchronize, the basic idea used in Sequencer Party, which we adopted here, was to *consider the global state of a session as a set of text documents*. Based on this assumption, approaches used by collaborative text editors such as Google Docs, Word online and Etherpad, can be used for synchronization: *Operational Transformation* (OT) and *Conflict-free Replicated Data Types* (CRDT). OT is very popular (used by Google Docs) and handles conflicts by transforming concurrent operations so that they can be applied safely, even if they are executed in a different order on different clients. CRDT for collaborative editing are an alternative approach to OT. A very simple differentiation between the two approaches is that OT attempts to transform index positions to ensure convergence (all clients end up with the same content), while CRDTs use mathematical models that usually do not involve index transformations, like linked lists. CRDTs are better suited for distributed systems, provide additional guarantees that the document can be synced with remote clients, and do not require a central source of truth [14].

Rely on a CRDT implementation for syncing states as text documents: we choose the CRDT implementation provided by the Yjs JavaScript library used by several open-source collaborative text applications[13]. Yjs implements a modified version of the algorithm described in [13], which adds some optimizations to the original CRDT approach. It has also been shown that Yjs is suitable for more general applications that require state synchronization [14] as it exchanges only the differences when syncing two clients. Yjs supports both P2P (WebRTC-based) and server centric design (WebSocket-based). WAM Jam Party uses in its current version the P2P design.

Turning WAM state objects into Yjs documents: an issue that had to be addressed is that the WAM states are plain old JavaScript object (POJOs), and to synchronize these state objects they need to be YJS documents. Tom Burns, designer of sequencer party, wrote a library especially for synchronizing Yjs documents to/from POJOs, called y-pojo[15], but for the moment POJO to Yjs documents transformations are handled manually.

Optimization: Every WAM state is stored in its own YJS document (instead of storing all states in a "global" document), and the host and 3D avatar states too. Yjs focuses on real-time collaboration and convergence but does not inherently maintain a full change history in each document, however it has been noticed that documents tend to grow over time. When updating YJS documents this operation will get slower and slower when more edit events have occurred. By keeping state objects segmented into separate YJS documents it delays that inevitable growth. Fortunately, even during long sessions (about 40mns long) the updating time did not impact the user experience.

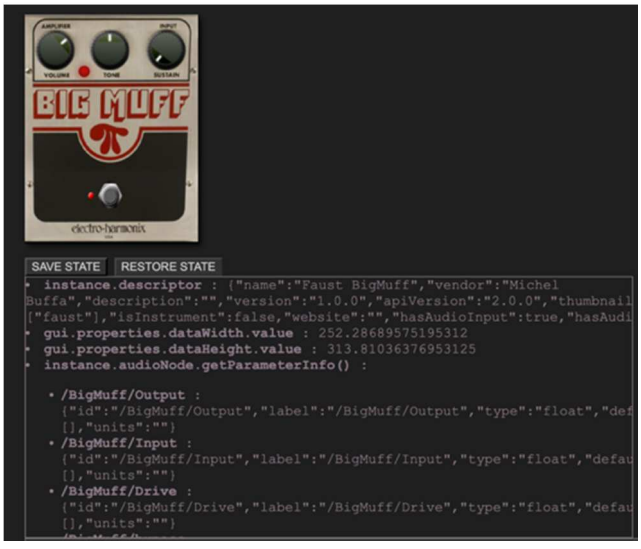

Figure 7: The Big Muff WAM (written in FAUST, compiled to WebAssembly) and its parameter description returned by a call to the `getParameterInfo()` method.

Session persistence: during live sessions, a special client, that is always online, is called the Session Manager. Its task consists in managing the persistence of the sessions. Like all regular clients it receives state updates, and it saves them in a single file every second. Each WAM Yjs document *is not saved*, only *the JSON representation of all objects is made persistent and does not include the additional information*

[12] https://github.com/webaudiomodules/api/
[13] https://github.com/yjs/yjs
[14] https://blog.kevinjahns.de/are-crdts-suitable-for-shared-editing/ gives more insight about the performance characteristics in Yjs.
[15] https://github.com/boourns/y-pojo.

stored in the Yjs documents for the sake of the CRDT algorithm. So, for example, it is possible to close a session (all clients disconnect, except the Session Manager) one day and continue working on it another day. When a saved session is opened by the Session Manager, new Yjs documents are created from the saved states and synchronization is performed.

Web Audio Modules in 3D immersive environments

Generate a 3D GUI using introspection, without modifying existing WAMs: assuming that dozens of Web Audio Modules have been developed, using many different languages, frameworks, building tools, etc. we chose an approach consisting in loading a WAM plugin from its URI, load only its audio processing Node (the WAM specification forces separating the GUI and the audio Node parts), and call the `getParameterInfo()` method that returns a JavaScript object describing the WAM parameters. From this description, then we generate an interactive 3D model.

```
{
  "name": "Big Muff",
  "url": "https://www.webaudiomodules.com/community/plugins/wimmics/BigMuff/index.js",
  "root": "/BigMuff/",
  "customParameters": [
    {
      "name": "Drive",
      "used": true,
      "color": "#00ff00"
    },
    {
      "name": "Input",
      "used": true
    },
    {
      "name": "Output",
      "used": false
    },
    {
      "name": "Tone",
      "used": true,
      "color": "#0000ff"
    },
    {
      "name": "bypass",
      "used": true,
      "type": "button"
    }
  ],
  "defaultParameter": {
    "type": "cylinder",
    "color": "#ff0000"
  }
}
```

Figure 8: Example of a WAM configuration JSON file

Optionally, use a JSON file for the mapping of these parameters to different kind of 3D widgets, indicate the shape, the color, the label values, etc. We chose a "*convention over configuration approach*"[16] so that by default an existing mapping exists (knobs will be generated as draggable vertical 3D cylinders, switches as push buttons, colors will be randomized for the cylinders, labels will be the ones from original parameters, etc.). However, if one wants to customize a given parameter or if a virtual instrument has too many parameter/GUI elements, these can be hidden and will not be in the 3D GUI, then adjust the configuration file. For example, a synthesizer can only display a preset menu if we only need to reuse existing sounds, instead of its numerous knobs, sliders, switches from its 2D GUI.

Figure 7 shows an example with the WAM recreation of the famous "Big Muff" fuzz pedal and Figure 8 shows its configuration file. It contains the plugin name and its URI for retrieving the WAM. The parameter configuration section allows for granular control over the visual representation of each parameter. Also notice the default parameter configuration section, enabling users to predefine the default color and type for each parameter. This file is generated automatically with default values, for each WAM supported, starting from a list of URIs collected from the WAM community endpoint[17].

The default 3D representation of a WAM consists of a box that serves as the primary container for its parameters. This box dynamically adapts its size to accommodate the number of parameters associated with the WAM. On the box, each parameter is represented by a visual element that corresponds to its type. For instance, HTML buttons are represented by 3D buttons and HTML knobs or sliders are represented by 3D cylinders (Figure 9). On each side of the box, we can see a green sphere and a red sphere used for connecting the WAM to other WAMs or to specific elements in the 3D music installation.

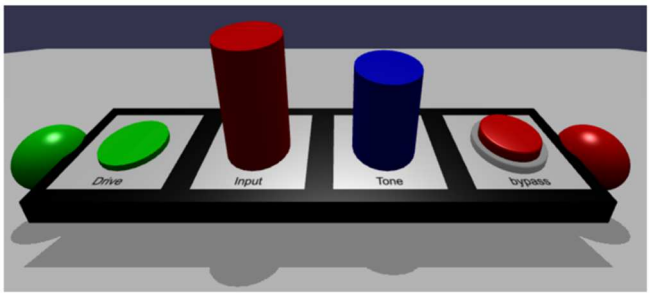

Figure 9: 3D representation of the Big Muff WAM from Figure 7.

Allow for custom 3D models and interactions: the configuration files can also delegate the 3D representation of a WAM to custom JavaScript code. For some WAM with complex GUIs, such as a step sequencer or instruments with specific parameters. This is the case of the step sequencer / four channel synthesizer we used in the current version of our application, shown in Figure 10.

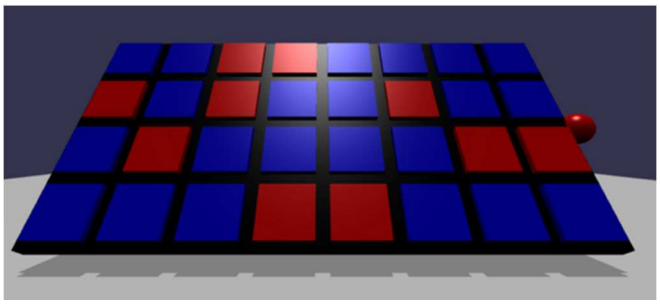

Figure 10: a 3D step sequencer WAM with a custom GUI. Built using ToneJS's step sequencer and synthesizer.

IV. User Interaction

In this section, we present the main actions that can be taken to build and modify music installations in the proposed immersive world. Interactions play a crucial role in an application like the one presented in this article. During the design and development phases, we were our first testers, making numerous modifications to the UX design to make it

[16] https://en.wikipedia.org/wiki/Convention_over_configuration

[17] https://www.webaudiomodules.com/community/plugins.json

usable. After the completion of first functional prototypes, we then carried out user tests, which are described in section VI.

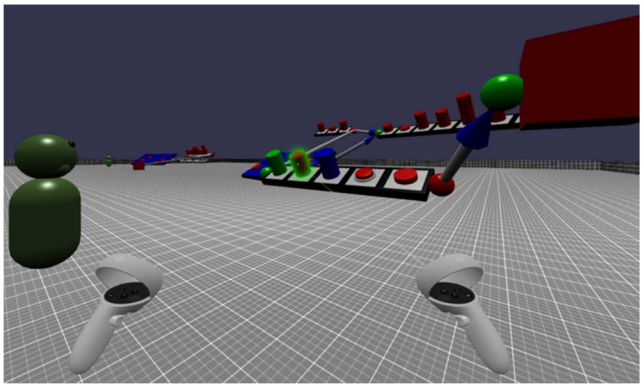

Figure 11: A music installation example just in front, another player's avatar on the left, and another player with the installation he built further away on the left.

The goal of the application is to allow users to build virtual music installations made of note generators, virtual instruments and audio effects (Figure 11**).** Each of these elements is considered as an audio node and connected to other nodes to form an oriented audio graph. Some nodes are specific and provided by the host, such as the "audio destination node" (Figure 12) that corresponds to the speakers. It is the only one that does not have a red sphere on the right end of its box for output connections.

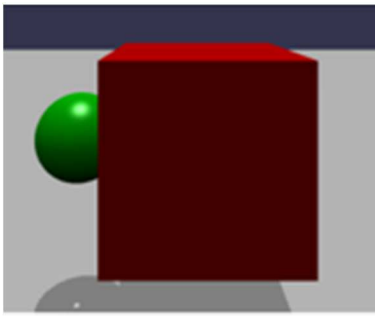

Figure 12: the special "audio destination / audio output node.

The minimal set of user interactions is:

- Users can walk in the 3D scene, in the XZ plan
- Users can create a session or join an existing one.
- Users can add a WAM, or a specific node (such as audio destination) in the 3D scene.
- Users can connect/disconnect WAMs.
- Users can adjust WAM parameter values.
- Users can change the position or orientation of a WAM, the connections should follow (3D representation of the arcs in the audio graph).
- Users can delete a WAM.
- Users can interact with the host (menu etc.).
- Users can move and see other participants.

Adding a WAM or a specific node: a JSON-based menu system was developed to facilitate the selection of the available components that can be added to the 3D scene, such as note generators, instruments, audio effects, as well as the audio output (Figure 13). The content of this menu, based on the Mixed Reality Toolkit (MRTK) for BabylonJS[18], is built from a JSON configuration file (Figure 14) that specifies the category names, the associated WAMs (including their names and their configuration file paths, described earlier), and the special nodes provided such as the audio destination node.

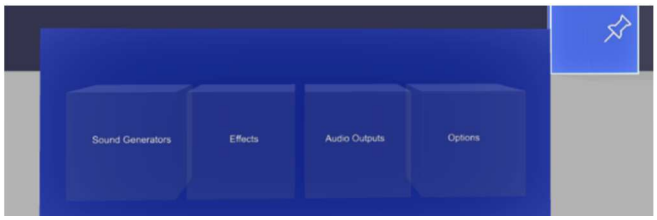

Figure 13 : main menu for adding elements in the 3D scene.

```
{
  "categories": [
    {
      "name": "Effects",
      "plugins": [
        {
          "name": "Big Muff",
          "configFile": "bigMuffConfig"
        },
        {
          "name": "Simple Delay",
          "configFile": "simpleDelayConfig"
        },
        {
          "name": "StonePhaser",
          "configFile": "stonePhaserConfig"
        },
        {
          "name": "Disto Machine",
          "configFile": "distoMachineConfig"
        }
      ]
```

Figure 14: Extract of the main menu configuration file.

Selecting and positioning an element: one key aspect of user interaction is the ability to manipulate the WAM plugins themselves. By holding the controller over the WAM box and dragging, users can freely reposition the entire WAM element within the virtual scene. This flexibility allows users to place WAMs in convenient locations for interaction and visual organization. We also added some "helpers" and "visual feedback" to this interaction: using the rays coming from the VR controllers, the "point then click the trigger to drag" operation is made easier as we are using behind the scenes an extra, invisible, larger box, making the selection of the WAM much easier. The invisible box appears with some transparency as visual feedback while the WAM is selected and moveable (Figure 15). In this state, the VR pointer movements move the selected WAM only in the X/Y plan relative to the user (up / down / left / right movements only), and to position a WAM closer or further to /from the user position along its local Z axis, the right joystick should be

[18] https://doc.babylonjs.com/features/featuresDeepDive/gui/mrtk

operated. In the current version, a selected WAM can only be rotated around its local X axis (to make it look up / down), as giving more freedom confused our users during the tests. When WAMs are not selected, users can move using standard camera movements provided by the BabylonJS library (Free Camera constrained to the XZ plan).

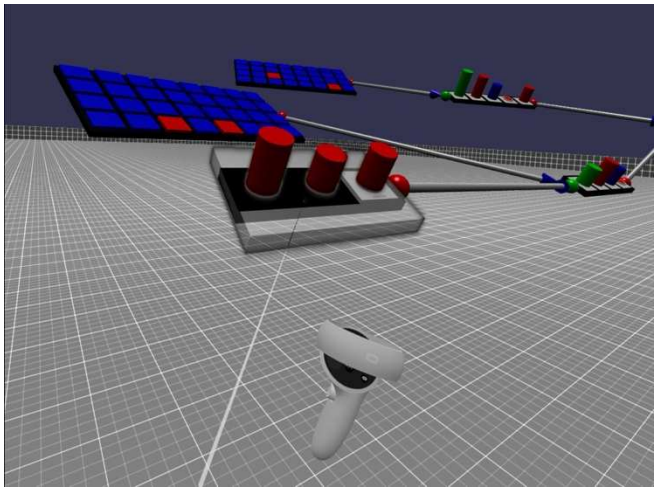

Figure 15: A WAM effect has been selected, its box is now transparent grey and larger, indicating that it can be dragged.

Connecting and disconnecting elements: to connect nodes and establish audio routing between them, users can utilize a simple and intuitive click and drag approach. Aim to the output sphere of an element, the sphere becomes highlighted. Then press the trigger of the controller and drag a wire with an arrow shape at its end to the input sphere of another element. When the output sphere is highlighted too, release the trigger to make the connection (Figure 16). This visual representation clearly indicates the audio stream, enabling users to easily manage their audio networks. To disconnect, do the same but aim first to the input sphere of an element, drag and release the connected wire extremity. If more than one wire is connected then, wires will be "separated" when clicked, select the one to disconnect. We also use larger, invisible spheres to enable selection of an input or output sphere even when the pointer is in proximity.

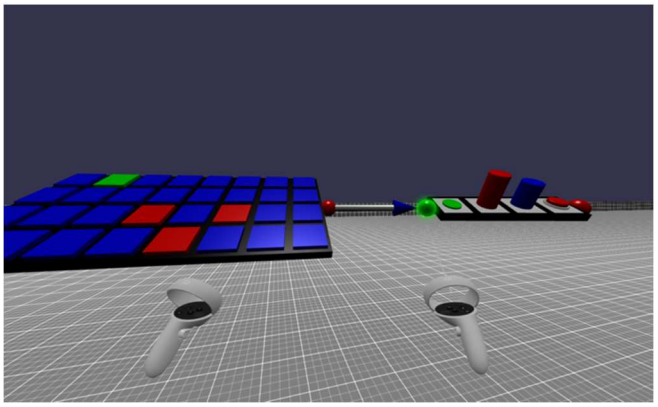

Figure 16: Drag arrows between speres to connect elements.

Adjusting parameters: moving beyond WAM placement, user interaction extends to manipulating individual audio parameters. Hovering the controller cursor over a parameter highlights it (Figure 17) providing visual feedback to indicate which parameter can be modified. Beyond highlighting, the interaction method is tailored to the specific parameter type. Button parameters can be modified by pointing them and clicking the trigger. Cylinder and Box parameters can be set by dragging the 3D element vertically with the controller. The adjusted value is displayed in real-time above the 3D element, providing clear visual feedback (Figure 18).

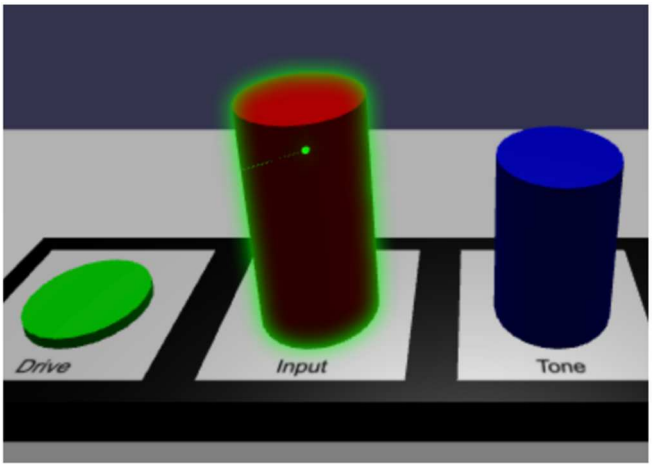

Figure 17: Example of highlighting a parameter.

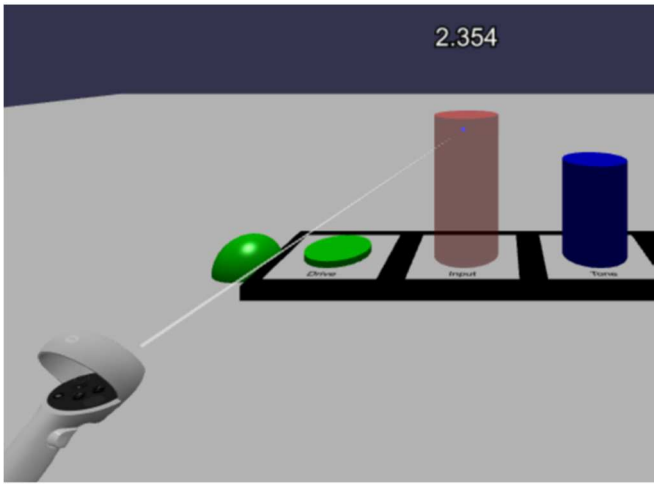

Figure 18: example of a parameter interaction.

Changes made to parameter values within the 3D representation are immediately communicated to the corresponding plugin via the WAM API. This ensures that the audio output of the WAM reflects the updated parameter values in real-time, enabling seamless sound design and manipulation.

Element deletion and advanced features: to further enhance the user interaction the project introduces a context menu that provides additional, less frequently used, control and manipulation options. The user can activate the context menu by clicking on a WAM with a designated controller button. One of the primary menu options is the ability to remove the selected WAM from the scene. Another key menu option allows for accurate 3D rotation of a WAM along its 3 axes.

V. Player Representation

To enhance the collaborative experience and provide a sense of presence within the virtual environment, the project implements a system for representing and synchronizing player avatars (Figure 19.)

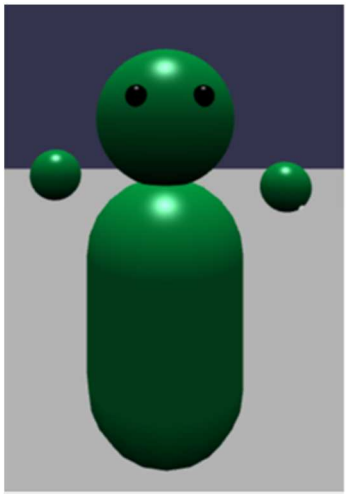

Figure 19: A player's avatar.

Each user in the virtual environment is represented by a simple avatar consisting of a body, head, and two hands. The avatar's head orientation is directly mapped to the user's VR headset orientation, ensuring that the avatar's gaze direction aligns with the user's actual view within the virtual environment. This visual representation enhances the sense of presence and allows participants to understand each other's perspectives. The avatar's hands are positioned and oriented to match the user's controller movements. This provides a clear indication of the user's current actions, such as selecting WAMs or manipulating parameters. The player avatar system utilizes the state representation and synchronization mechanism presented in section 0 to ensure that all participants have a consistent view of each other's positions and actions.

VI. Usability tests

Some preliminary user tests have been conducted even with the application still in its early stages of development. Six different users have participated, all are students from our research group (Master, PhD), none with experience in VR, using Oculus 2 and Oculus Pro headsets. We did two different sessions: one with P1, P2, P3 and P4 sharing the same experience, and one with just P5 and P6. All were in the same room. Our aim in this experiment was primarily to evaluate the UI/UX experience, rather than the robustness and performance of the application in remote set up experiments. Eleven different tasks had to be performed. The participants were briefed just before testing the application they discovered on this occasion, and were told what it consisted of, and which controller keys would be used (there are very few: A to display the main menu and trigger to select menu entries, 3D objects, move them, or connect them). Then we just told them orally what the next task was. We waited until each participant had completed the task in hand before moving on to the next. Notes were taken to record users' reactions during the experiment. They rated the tasks in real time by stating the score orally, before moving on to the next task.

Scores between 1 (difficult to achieve) and 5 (easy to achieve) were given after the completion of each task and are shown in the table in Figure 20. Detailed results are online[19], we only present a summary below:

Task	Description	P1	P2	P3	P4	P5	P6	Mean	Deviation
1	Move and navigate in the 3D Universe	5	5	4	5	5	4	4,7	0,5
2	Show and hide the main menu	5	5	5	5	5	5	5,0	0,0
3	Create a note generator of type Step Sequencer	5	5	5	5	5	5	5,0	0,0
4	Position it in space by clicking and dragging	4	4	2	3	5	5	3,8	1,2
5	Add an effect into the installation and position it	3	3	2	4	5	4	3,5	1,0
6	Create an audio output element, position it	4	3	2	4	5	5	3,8	1,2
7	Connect elements to form a graph	4	5	4	4	5	5	4,5	0,5
8	Add notes to the Step Sequencer	5	5	5	5	5	5	5,0	0,0
9	Modify some audio effect parameters	5	5	5	4	5	5	4,8	0,4
10	Add another element	4	5	4	5	5	5	4,7	0,5
11	Move to another player's installation and modify it	5	5	3	5	5	5	4,7	0,8

Figure 20: Usability results (5 = easy, 0 = hard)

The test involved an extra client that was a computer with the "immersive web emulator" browser extension, that allowed us to monitor the session. Most tasks proved to be easy to perform, and even easier after users got used to the controls. Nearly all difficulties encountered were related to element positioning accurately: P2: "would have liked to be able to rotate objects around the Y axis". He created an element from another position and could not align it with the previous objects, after a while he found out that he can move in the virtual world to get a better position and positioning object at the right place is much easier. P3 found a bug related to the scrolling plane when moving objects. Instead of moving in the X/Y plane perpendicular to the Z axis of the camera and passing through the selected object, movements were unpredictable. He reloaded the page, found the session in the state it was before he left, and could continue. P2 and P6 found that collision detection between participants and 3D elements was sometimes complicated and made manipulating objects difficult, he suggested to redo some testing without collision detection enabled. P2 would have liked to be able to rescale some objects (like the step sequencer that occupied too much space). P3 would have liked some objects, like the step sequencer, to appear vertically, not horizontally; this would have made note selection easier. P3: "When objects are far away, it's sometimes difficult to select them."

[19] https://tinyurl.com/shbry583

Throughout the 40-minute experiment, no lag was noticed, and the experience went smoothly even when one of the participants found a bug and had to reload the WAM Jam Party page to rejoin the ongoing session. Most problems were related to the lack of an easy way to rotate objects, and two-hand gestures have been suggested as a solution. Collision detection also posed problems, and it was suggested to either try the experience without it or implement a more forgiving way of handling collisions. The first experiments involved four actors at the same time and the second one only two, and no slowdown was witnessed.

VII. Areas for improvements, limits, future prospects

As it has been said, the work presented is still a prototype and a lot of work still needs to be done. From the usability tests we conducted, new UX interactions (such as two hand gestures) for performing some tasks (like rotating objects) need to be evaluated, new WAMs would make the application more versatile and will be added, also, more visually appealing and realistic environments have been asked by our test users. In addition, the WAM standard supports many extensions, some being the ability to render 3D objects or WebGL shaders that respond to the sound being played. Such WAMs would create a more dynamic experience, further solidifying the connection between sound and visuals as this is one of the most appreciated features of the original Sequencer Party. We also need to make some performance measurements about network latency, maximum load, (how many WAMs with lots of parameters, how many parameters can the application handle before degradation), etc. Collaboration is another area for improvement: future iterations could introduce like proximity voice chat, enabling more natural and spontaneous interactions between collaborators [2]. This would reinforce the sense of presence in the virtual space and enhances the collaboration experience. The introduction of user account management for creating and sharing, multiple sessions would be a game-changer. These functions would facilitate collaboration by enabling users to work together on projects seamlessly and present their creations in the "musical multiverse".

Finally, WAMs are designed to be inter operable in the HTML world, this is not yet the case in the WebXR world, as the UI/interaction code is for the moment specific to BabylonJS. We would like to collaborate with other research teams who started to develop musical metaverse applications using other 3D frameworks and rethink the WAM design of “WAM adapters” for different targets, or maybe define a pivot code for describing abstract 3D GUIs.

VIII. Related work and conclusion

Current application status: all the basic building blocks needed to build an immersive, multi participant application for creating electronic music installations have been put in place. At the time of writing, only ten or so WAMs plugins are available, in the application but new ones are being added regularly. Compared to Sequencer Party, this first prototype still lacks some "comfort" features but is fully functional. The proposed musical universe is consistent between participants, each change is propagated and can be experienced without being perturbated by the latency, as audio synchronization is managed locally. The 3D world is still simple in this first prototype (a plan with a grid texture, with walls at the boundaries), but it is sufficient to give a good sensation of space and depth. The application can be tested online and is open source[20].

A number of related works have been carried out, based on native applications (as opposed to web applications), particularly in the context of multi-user VR instruments and performances, such as Hamilton's with Carillon [22], a music composition and interactive performance environment centered around a multiplayer collaboratively-controlled virtual instrument with gesture-based hand tracking, or Ciciliani who studied the way avatars can interact with the topology of the virtual worlds and the use of the 3D environment as both instrument and score [23]. Syukur and al. designed Gamelan Land [24] as a virtual world game for learning and playing orchestral music, in which the rules are based on the social presence of participants. Works by Bell with PatchXR [21] for multi-user exploration of large sound corpus [18] or by Bauer and al. on an AR compositional platform which allows to create interactive music experiences are also relevant [20].

However, there are not a lot of related works using WebXR technology for the Musical Metaverse. Dziwis and al. proposed Orchestra [14], an open-source toolbox that uses the Web Audio API to realize live performances with various performance practices for web-based metaverse environments: from live streaming of volumetric audio and video, live coding in multiple programming languages [16], to performing with generative algorithms or virtual instruments developed in PureData [19].

The closest WebXR-based work related to WAM Jam Party is “Musical Metaverse Playgrounds” by Boem and Turchet. However, WAM Jam Party differs from that of Boem and Turchet in several aspects: 1) The note generators, virtual instruments, and audio effects are all Web Audio Modules interoperable plugins. There are over a hundred ready-to-use Web Audio Modules available on the web that can be integrated rapidly using JavaScript dynamic imports with their URI. Creating new ones is relatively simple, especially using the FAUST IDE [21], which can compile WAM plugins in web assembly in and publish them on a server in seconds, making them usable to all WAM host applications. 2) The interactive 3D graphical interfaces of these Web Audio Modules are generated on the fly and do not attempt to resemble 2D knobs, sliders, or switches from existing VST plugins recreated in 3D, but rather interfaces adapted for VR/XR use. 3) The application can operate via WebRTC or WebSockets (only one line of code needs to be changed), and CRDT-type algorithms used are particularly well-suited for synchronized object state sharing, offering robustness in managing connections/disconnections (everything is logged) and are well-suited for building persistent worlds. 4) The application is entirely coded in TypeScript and uses numerous features of BabylonJS, including ported elements from the MRTK toolkit of the HoloLens. BabylonJS offers many VR/XR-specific 'behaviors' that allow customizable interactions with elements (e.g., dragging an element along a predefined plane or hemisphere, resizing objects, etc.). 5)

[20] https://musical-metaverse.onrender.com/, source code : https://github.com/Doori4N/musical-multiverse-vr

[21] PatchXR is a commercial product that serves as a sandbox where creators can build interactive virtual worlds using modular patches, similar to how you would patch together elements in modular synthesizers: https://patchxr.com/

WAM Jam Party allows adding 3D WAMs in the proposed Musical Metaverse virtual world, and these WAMs can be positioned, oriented, connected to form complex graphs that can be manipulated real time by all participants, added or removed. Proposing new WAMs consists in editing a configuration JSON file, and they will be available to all participants in the main menu.

In conclusion, WAM JAM Party is an extensible construction tool that provides a good basis for construction and experimentation in the musical metaverse. Its extensible design, based on WAM plugins and allowing the generation of 3D GUIs on the fly, opens a wide range of possibilities. The multi-participant creation of musical installations by patching high-level plugins sets it apart from other works.

ACKNOWLEDGMENT

This work has been supported by the French government, through the France 2030 investment plan managed by the “Agence Nationale de la Recherche”, as part of the "UCA DS4H" project, reference ANR-17-EURE-0004

REFERENCES

[1] L. Turchet. "Musical Metaverse: vision, opportunities, and challenges." Personal and Ubiquitous Computing 27: pp 811-1827. 2023.

[2] A. Boem, and L. Turchet. "Musical Metaverse Playgrounds: exploring the design of shared virtual sonic experiences on web browsers.", 4th International Symposium on the Internet of Sounds. IEEE, 2023.

[3] M. Buffa, J. Lebrun, J. Kleimola, O. Larkin & S. Letz. Towards an open Web Audio plugin standard. In Companion Proceedings of the The Web Conference 2018 (pp. 759-766), 2018.

[4] F. Déchelle, R. Borghesi, M. De Cecco, E., B., and N. Schnell. jMax : an environment for real-time musical applications. In Computer Music Journal 23, 3, pp 50–58. 1999.

[5] J. McCartney. Rethinking the computer music language: Super collider. In Computer Music Journal 26, 4, pp 61–68. 2002.

[6] M. Puckette. FTS : A real-time monitor for multiprocessor music synthesis. In Computer Music Journal 15, 3, pp 58–67. 2002.

[7] H. Choi. Audioworklet : the Future of Web Audio. In Proceedings of the International Computer Music Conference. Daegu, South Korea, pp 110–116. 2018

[8] M. Buffa, J. Lebrun, S. Ren, S. Letz, Y. Orlarey, R. Michon, and D. Fober. Emerging W3C APIs opened up commercial opportunities for computer music applications. In The Web Conference 2020 DevTrack. Taipei, Taiwan. 2020.

[9] M. Buffa, P. Kouyoumdjian, Q. Beauchet, Y. Forner, and M.Marynowic. Making a guitar rack plugin -WebAudio Modules 2.0. In proceedings of the Web Audio Conference 2022., Cannes, France, 2022. https://doi.org/10.5281/zenodo.6769098

[10] J. Kleimola and O. Larkin. Web Audio Modules. In Proceedings of the Sound and Music Computing Conference 2015.

[11] M. Buffa, S. Ren, O. Campbell, J. Kleimola, O. Larkin, and T. Burns. 2022. Web Audio Modules 2.0 : An Open Web Audio Plugin Standard. In WWW ’22 : The ACM Web Conference. 2022. ACM, France, pp 364–369. 2022 https ://doi.org/10.1145/3487553.3524225

[12] M. Buffa, S. Ren, T. Burns, A. Vidal-Mazuy & S. Letz. Evolution of the Web Audio Modules Ecosystem. In proceedings of the Web Audio Conference 2024. https://zenodo.org/doi/10.5281/zenodo.10825646

[13] P. Nicolaescu, K. Jahns, M. Derntl, & R. Klamma. Near real-time peer-to-peer shared editing on extensible data types. In Proceedings of the 2016 ACM International Conference on Supporting Group Work (pp. 39-49). 2016.

[14] D. Sun, C. Sun, A. Ng, & W. Cai. Real differences between OT and CRDT in correctness and complexity for consistency maintenance in co-editors. Proceedings of the ACM on Human-Computer Interaction, 4(CSCW1), 1-30. 2020.

[15] D. Dziwis, H. Von Coler and C. Pörschmann,. Orchestra: a toolbox for live music performances in a web-based metaverse. *Journal of the Audio Engineering Society*, *71*(11), pp.802-812. 2023.

[16] D. Dziwis, H. Von Coler and C. Porschmann, , October. Live Coding in the Metaverse. In *2023 4th International Symposium on the Internet of Sounds* (pp. 1-8). 2023.

[17] L. Turchet,. Smart musical instruments: vision, design principles, and future directions, IEEE Access 7 (2019) 8944–8963.

[18] J. Bell, 2023, October. Networked Music Performance in PatchXR and FluCoMa. In International Computer Music Conference (ICMC) 2023.

[19] D. Dziwis. PdXR-the Evolution of Pure Data into the Metaverse. In Proceeding of the International Computer Music Conference (ICMC) (pp. 1-8). 2023.

[20] V. Bauer. and T. Bouchara. First steps towards augmented reality interactive electronic music production. In 2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW) (p. 90-93). IEEE. 2021.

[21] Ren, S., Letz, S., Orlarey, Y., Michon, R., Fober, D., Buffa, M. and Lebrun, J.,. Using Faust DSL to develop custom, sample accurate DSP code and audio plugins for the Web browser. Journal of the Audio Engineering Society, 68(10), pp.703-716. 2020.

[22] R. Hamilton and C. Platz. Gesture-based collaborative virtual reality performance in carillon. In Proceedings of the 2016 international computer music conference (pp. 337-340). 2016.

[23] M. Ciciliani.. Virtual 3D environments as composition and performance spaces. Journal of New Music Research, 49(1), pp.104-113. 2020.

[24] A. Syukur, P.N. Andono and A.M Syarif. A Multiplayer Virtual Reality Game for Playing Gamelan Music in a Virtual Orchestra Mode. In 3rd International Conference on Intelligent Cybernetics Technology & Applications (ICICyTA) (pp. 375-379). 2023