Extending Networked Mapping Research Middleware Into the Browser Sandbox

Matthew Peachey
Graphics and Experiential Media Lab
Dalhousie Univeristy
Halifax, Canada
peacheym@dal.ca

Kyle Smith
Graphics and Experiential Media Lab
Dalhousie Univeristy
Halifax, Canada
kylesmith@dal.ca

Joseph Malloch
Graphics and Experiential Media Lab
Dalhousie Univeristy
Halifax, Canada
jmalloch@dal.ca

***Abstract*—Web-based technologies have seen rapid technological improvements over the past several decades. This is especially true as it relates to audio applications, with the specification of the WebAudio API enabling users to deploy highly performant audio projects to the web. Existing literature describes how mappings are a critical component of end-to-end audio projects, including Digital Musical Instruments, Internet of Sounds devices, and more. Due to this, years of research efforts have produced mapping middleware for the facilitation of establishing mappings between sources and destinations. This paper discusses how the libmapper [?] ecosystem is extended to support mapping to and from a sandboxed browser environment. Establishing libmapper connectivity on the web is achieved through websockets and a backend daemon. In this paper, we discuss the implementation details of binding libmapper to the web as well discuss potential use-cases via user-story driven scenarios.**

***Index Terms*—Mappings, Internet of Sounds, WebAudio.**

I. Introduction

Web technologies have seen massive technical improvements and community adoption in the past several years. The audio domain is one such example of this, as browser-based technologies have rapidly become more capable and performant for handling audio tasks. For instance, the WebAudio API [17] has enabled developers to build robust audio applications for the web, including Synthesizers, Audio Processors, and more. This technology also provides a platform for fields such as the Internet of Sounds (IoS) as well as the design and development of browser-based Digital Musical Instruments (DMIs).

Mappings are an essential part of creating end-to-end multimedia experiences as they are the glue between various inputs and outputs as well as what defines the behavior between these endpoints. This is especially true in the context of audio focused project as observed in the context of Digital Musical Instruments (DMIs), modular synthesis and many other systems. Musicians and other creatives are often interested in exploring novel ways to interact or perform with these types of instruments that extend beyond the capabilities of traditional music control protocols such as MIDI, and therefore creating mappings is often a key part of their creative workflow. Due to the importance of mappings as it relates to DMIs and other end-to-end audio applications, a vast amount of academic research has been focused on how people think about and create mappings as well as the development of tools that are designed to support and facilitate the mapping process [3], [12], [24], [25].

This paper explores how to integrate libmapper, a popular tool for creating & managing mappings across a network at run-time, into modern web browsers and applications. While the browser is a sandboxed environment (meaning there is no option to communicate with external processes over TCP, which is typically a requirement of libmapper) this paper highlights a new WebSocket based approach for binding libmapper to the web. We present both a backend daemon, responsible for managing the libmapper network and it's devices, as well as a front-end module for JavaScript that allows users to quickly and easily associate HTML elements with libmapper signals. By bringing libmapper to the web, we will support both musicians working with web-based interfaces as well as provide additional options for defining behaviours across the rapidly growing Internet of Sounds research community.

Finally, we conclude the paper first with a discussion of potential use-case scenarios and how our solution addresses a user's needs. Each of the scenarios are described using a user-story and show how different users with unique requirements are able to interact with our software. Finally, we discuss future work for the project including both technical implementations and formal evaluations of the software.

II. Related Work

A. Audio Applications in the Browser

Recent advancements in web-audio technology as well as the flexibility of working with many different front-end frameworks has resulted in the browser becoming a suitable target for deploying audio applications. The WebAudio API in conjunction with WebAssembly has enabled the creation of many highly performant audio applications, including Synthesizers, Digital Audio Workstations (DAWs), and more. For instance, FAUST (a functional programming language for developing audio projects [19]) allows users to target the web directly [4]. Similarly, The Web Audio Modules project has resulted in professional-grade audio plugins for the web [5]. There are many other examples of audio projects deployed to the browser, including Tube Amp simulators [11], Musical Metaverse playgrounds [1], and more. Each of these

applications feature some aspect of user-adjustable parameters, and thus providing an option for mapping capabilities may be of interest to their respective communities.

The Internet of Sounds is an emerging field of research focused on exploring how networks of *sound things* communicate sonic information [23] with each other. As this field continues to mature, we expect adoption of browser-based instruments to also grow. With the previously discussed capabilities of the WebAudio API, the browser presents itself as an excellent candidate for the deployment of *sound things* that involve some degree of human interaction.

B. Digital Musical Instruments

Digital Musical Instruments (DMIs) have been an active area of research for over twenty-five years. DMIs are defined by Miranda & Wanderley as an instrument that is comprised of a gestural controller and a sound production unit, with some specified mapping strategy defining the relationship between them [16]. There are many kinds of DMIs that have emerged over the years, many of which with their own unique form factors and use cases. Smart Musical Instruments (SMIs) [21] are one type of DMI that places significant weight on connectivity, usage of sensors, and on-board intelligence and are one of the most common *things* found within the IoS. Furthermore, Peachey & Malloch illustrate in their 2023 work how effective mappings are key to achieving the design goals of SMIs [20]. With these items in mind, there is a natural connection to be made between the Browser, DMIs, Mappings and the IoS and the common tools and processes these technologies may choose to rely on.

C. Mappings

Mappings are what define the behaviour between an input controller and an sonic output. Mappings can either be *explicit* (where a programmer or musician defines the precise relationship between each input and output) or *implicit* (where the mapping is learned by examples, such as in the context of a Machine Learning algorithm). In either case, methodically considering mappings is critical when designing DMIs for both users and their perspective audiences.

There are four commonly found topologies used when creating mappings between sources and destinations as described by Hunt [10] and depicted in figure 1. One to one mappings, or *direct* mappings are the simplest of mapping topologies, which makes it easier for both a performer and audience alike to understand the relationship between input and output, but may not provide as engaging of an experience to the user as a non one to one mapping [9]. One to many or *divergent* mappings are a single mapping source may be mapped to several mapping destinations. Similarly, many to one or *convergent* mappings are when several mapping sources are influencing a single mapping destination. Finally, many to many or *complex* mappings are when several mapping sources are mapped to several mapping destinations using any combination of the former three techniques.

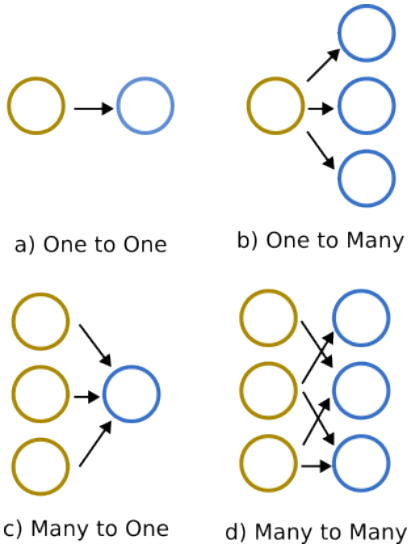

Fig. 1. Each mapping topology serves a specific type of situations and certain mapping middleware can enable users to decide which is best to use in order to support their own personal use case.

Mapping Tools: As we have seen, literature makes it clear that mappings are an essential part of Digital Musical Instruments as well as many other multimedia systems and performances. Due to the importance of mappings as well as the many different paradigms for creating & managing mappings, a plethora of research has gone into the design and development of tools to facilitate the usage of mappings.

One such project is *libmapper*, which supports the collaborative creation and management of run-time mappings between a distributed graph of software and hardware objects. Software developers and instruments designers can independently declare "signals" that are either *outputs* (producers of real-time data) or *inputs* (consumers of real-time data). Producers can range from software GUIs to sensor data from connected control surfaces, computer peripherals, DMIs, or IoT devices. Similarly, consumers can range from a synthesizer parameter to position of an object in a game engine. Signals are encapsulated into collections named "devices" based on their own individual requirements and workflows; some devices contain both input and output signals. Finally, users are also able to associate metadata to signals and devices that may be exposed to the entire mapping network to assist with the mapping creation process. Once users have defined their libmapper devices and signals, they are able to establish maps between them at run-time using any of the previously discussed mapping topologies. Each map has an associated user-editable *expression* as part of its metadata that defines the exact relationship between connected signal; the expression syntax supports FIR and IIR filtering, arbitrary combination functions for convergent maps, and map-reduce across source signals, vector elements, historical samples, and signal instances [13].

The libmapper ecosystem is constantly growing as research and community adoption of the project continues. Currently there is a large collection of programming languages (C,

C++, C#, Python, Java, Rust), environments (Max, Pure Data, Processing, Touch Designer, Ableton Live [2]) and hardware (Arduino/ESP32, the T-Stick DMI [14], [18], Probatio [6]) that is already supported.

While there are certainly other mapping tools and research software available for connecting aspects of DMIs (such as [7], [22], [8], [15]), however in the context of this paper, we will focus on the continued development of the libmapper ecosystem.

III. Implementation Details

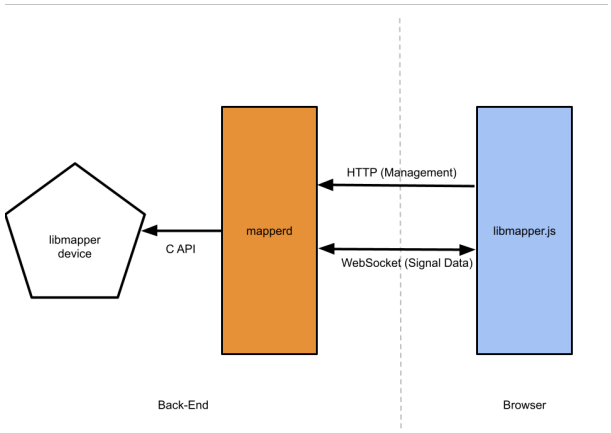

Fig. 2. A system diagram visualization of how web-browsers can interact with libmapper despite the sandboxed nature of modern browsers. A backend daemon leverages libmapper's C API to create devices and signals, establish maps, and facilitate other libmapper tasks. A browser side JavaScript module communicates with that daemon via both HTTP (for admin tasks) and websockets (for both session initialization and real-time data transfer). The entire communication process is depicted fully in Figure 3.

Bringing libmapper to the browser is not a simple task, as browsers have significant restrictions compared to most other platforms. Most significantly, libmapper relies on raw TCP and UDP streams to communicate with peer devices. These features are not supported from within sandboxed browsers and therefore require an alternative approach. As a workaround to these restrictions, we created a WebSocket and HTTP based server to run in the background, providing access to the libmapper graph through unrestricted channels.

Another challenge was creating appropriate front-end JavaScript library to allow users to interact with libmapper in a natural way. Simply providing the equivalent of the C API wouldn't feel idiomatic in JavaScript, and so a higher-level wrapper was created, along with more specialized modules allowing users to simply include a script for some use cases.

A. Backend WebSocket Server

Given the locked-down enviroment we have to work with inside a web browser, a back-end bridge was needed to allow sandboxed Javascript to interact with other networked libmapper devices. The server, referred to as "mapperd", exposes most of the commonly-used API surface from the libmapper C library with little logic of its own, save for automatic polling of devices. Most actions are taken via an HTTP REST API, however clients are required to keep a persistent WebSocket connection active to facilitate real-time data updates for signal values.

Figure 3 is an illustration of the communication flow between browser and server. The first step is for the browser to initiate a WebSocket connection and request a session token. This model was chosen to allow multiple browsers or sites to reuse the same server, while keeping their devices and signals isolated from each other. The WebSocket connection also serves as a stateful connection to the server, so in the event of unexpected disconnection it can destroy all associated devices appropriately.

The source code and additional information for the `mapperd` backend component is available on GitHub via https://github.com/libmapper/mapperd.

B. Frontend JavaScript Library

JavaScript is a multi-paradigm programming language with a huge amount of preexisting software and a massive number of daily users. Therefore, writing a new module for interacting with libmapper in JavaScript that is both robust as well as usable by a large audience is critical. Our JavaScript module, libmapper.js, is comprised of two main classes, namely `LibmapperDevice` and `LibmapperSignal` each of which encapsulates functionality similar to that of their respective libmapper C API counterparts but with communication with the larger libmapper network being handled through `mapperd` as depicted in Figure 3.

The key aspects of our module is the ability for the user to quickly instantiate a LibmapperDevice object which establishes a connection to the `mapperd` service automatically using a series of asynchronous functions. One a device is fully initialized, users can bind signals in a number of ways. First, we allow users to bind signals through an `addSignal()` function in JavaScript that takes an HTML element (typically as found by ID) as a parameter. The function then automatically registers that signal with the daemon and establishes a websocket connection. The second method is to attach an HTML attribute to a given element. For instance, if a user adds the attribute `mpr-signal-name` to an input element, the module will again automatically register that as a libmapper signal and establish a websocket communication with the daemon.

The source code and additional information for the `libmapper.js` frontend component is available on GitHub via https://github.com/libmapper/libmapper.js.

IV. Use Case Scenarios

Allowing users to integrate libmapper into their browser-first workflows will enable many different creative projects. In order to better understand the flexibility and usefulness of our software, we provide the following imagined scenarios. Each scenario provides a brief look at the experience & requirements of a potential user with their high level goals presented in the form of a "user story". Each scenario also includes a

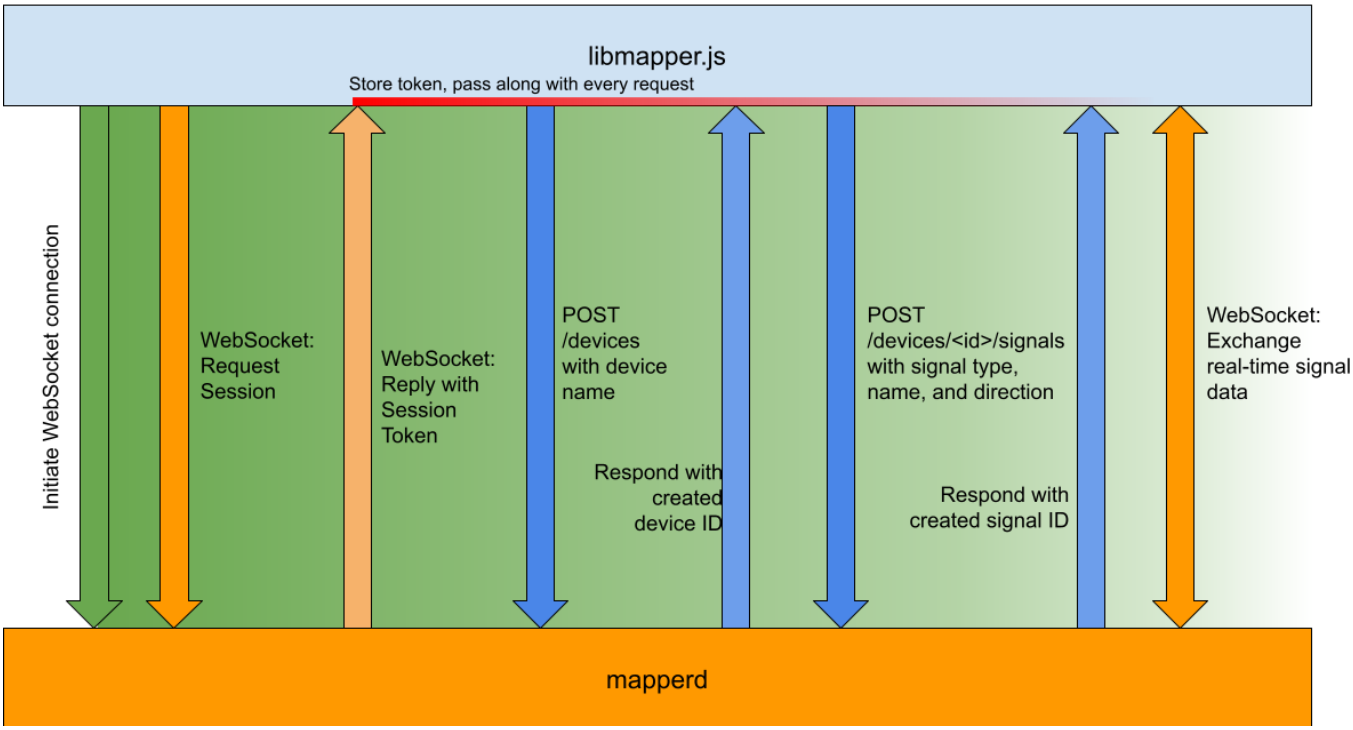

Fig. 3. A visualization of the communication flow between libmapper.js (front-end) and mapperd (back-end) throughout the entire life-cycle of a session. Note that a WebSocket connection is initiated at the beginning and persists for as long as libmapper.js is running on the front-end. Each of the other HTTP & WebSocket communication channels are shown, and together these form the basis of our browser bindings for libmapper.

discussion about how bringing libmapper to the browser helps a user achieve their goals and connect to a wider community of users.

A. Mapping an External Input Controller to a Web Synth

A common use-case for mapping middleware is using it to connect input controllers to parameters of a synthesizer. This typically will enable users to achieve a high level of control over their instrument or performance. With the rapidly increasing capabilities of web-audio, synthesizers (as well as other digital musical instruments and even entire Digital Audio Workstations (DAWs)) are becoming common place on the web. Due to these innovations, there may be a desire to map existing input controllers to a browser-based audio project in a typical DMI architecture.

Scenario: Alice is a musician with several years of experience of using custom input controllers as part of their musical workflow. These controllers are typically used within the DMI paradigm and thus Alice has experience using the libmapper project as a middle ware solution for connecting her input controller to an external synthesizer. Toby is a friend of Alice's who recently showed her a new browser-based synthesizer they've been working on developing. During the demo, Alice expressed interest in mapping some of her libmapper-compatible input devices to parameters of Toby's new web-based synthesizer.

Alice's requirements in this scenario are reflected in the following user story:

> As a **musician exploring a new web-based synthesizer**,
>
> I want to **map my input controller to parameters of that synthesizer**,
>
> So that **I can explore new musical and performative ideas with this synthesizer**.

In this scenario, we see that a musician hopes to establish a mapping between their input controller of choice and a browser-based synthesizer. Together Alice and Toby are able to quickly integrate libmapper connectivity to Toby's browser-based synthesizer. To do this, they download the `libmapper.js` module and add it to their web-page's home directory. Toby shows Alice how there are already HTML input sliders that are bound to parameters of their synth. After initializing a libmapper device as shown in Figure 4 the duo is then able to simply grab these HTML elements by their ID and instantiate signals by passing that element to the `.addSignal()` function. Alice is then able to open a mapping session using webmapper and the pair are able to ideate on new musical ideas using familiar input controllers mapped to Toby's Synth. Therefore, thanks to the lightweight browser module for libmapper, Alice's user story is able to be resolved quickly and using very few lines of code.

B. Mapping a Web-UI to External Multimedia Systems

The rapidly growing web-technology ecosystem has become a primary way to develop GUIs for countless different use

```
import { LibmapperDevice, LibmapperSignal } from "./libmapper.js";

// Create a libmapper device to manage signals
let libmapperDevice = await LibmapperDevice.create("browserDev");

// Grab the input HTML element associated with pitch control by its ID
let pitchSlider = document.getElementById("pitchSlider");

// Bind a libmapper signal to the HTML element
libmapperDev.addSignal(pitchSlider, "INCOMING");
```

Fig. 4. Binding an outgoing libmapper signal is as simple as calling the `.addSignal()` function and passing the HTML element of interest as a parameter along with the signal's direction.

```
<div>
  <input type="range" mpr-signal-name="sliderOne"/>
  <input type="range" mpr-signal-name="sliderTwo"/>
  <input type="range" mpr-signal-name="sliderThree"/>
</div>

<script>
  let device = await LibmapperDevice.create("libmapperDev", autoDiscover=true);
</script>
```

Fig. 5. This JavaScript snippet automatically binds the value of several HTML UI elements to their own respective libmapper signals. Each HTML element with the `mpr-signal-name` attribute is bound to a signal with relevant metadata (such as min/max values) being passed along to the signal initialization process as well.

cases. This scenario is based on an experience web-developer who is interested in designing a new GUI to map to synthesizer parameters using web-technologies they're already familiar with.

Scenario: Aaron is a web-developer with over five years of experience working with HTML & JavaScript in a professional context. Recently Aaron has become interested in multi-media performances, and is looking for a way to connect several components he's been working on developing. Aaron would like to leverage his web-development skills to develop an input controller with knobs, sliders, and other input options.

Aaron's requirements for this scenario are reflected in the following user story:

> As a **web developer interested in multimedia systems**,
>
> I want to **build a libmapper compatible UI in the browser**,
>
> So that **I can map my browser UI to parameters of external multimedia systems**.

In this scenario, we observe an experienced web-developer attempting to build a browser-based interface to control parameter of other components of a larger system they are working on. As depicted in Figure 5 this is easy to accomplish using `libmapper.js`. Aaron is able to automatically bind existing input elements to libmapper signals by using the `mpr-signal-name` attribute. Then, when they initialize a new libmapper device using the create function, these signals will automatically be registered and a websocket connection to the daemon will be established.

C. Browser to Browser Mapping

The first two scenarios have utilized libmapper to facilitate mappings between a browser and non-browser in order to show how libmapper can now handle mappings across the border of a sandboxed browser. However, libmapper can also be used to facilitate mappings between browser tabs or even a single browser window if such a use case would be desired by a user.

Scenario: Sarah is a regular libmapper user that is interested in using the middleware as a communication protocol across browser tabs. After working with libmapper as part of several different DMI projects, Sarah is eager to explore how working with libmapper in the browser can allow her to develop new web-based projects using familiar tooling and mapping paradigms. While Sarah's project is exploratory in nature, basing her work on existing mental models is important to her creative workflow.

Sarah's requirements for this scenario are reflected in the following user story:

> As an **experienced libmapper user interested in browser-to-browser communication**,
>
> I want to **establish mappings between HTML elements across two web-pages**,
>
> So that **I can use a familiar toolkit to explore new creative goals on the web.**

In this scenario, we see that libmapper on the browser is not exclusively designed to be used between browser and non-browser mapping endpoints. Instead, it is a very straightforward process to realize the goals of Sarah's user story. Simply defining a set of libmapper signals (either incoming or outgoing as demonstrated in the previous scenarios) either within one-browser tab or throughout several windows allows for libmapper connectivity to function as expected. Sarah's can then turn to her session manager of choice and establish and ideate with several different mappings as she is used to doing when working within a libmapper-based workflow.

V. Conclusion & Future Work

Mappings are a crucial aspect of many sound and audio applications. As web technologies in the audio domain continue to progress, extending open-source interactivity middleware to include browser-based software enables musicians and other users to leverage the web for their own projects.

In this paper, we describe how the networked middleware libmapper is extended to work in the browser despite the limitations of that sandboxed environment. We explore several scenarios that encapsulate various use-cases and show how libmapper in the browser can address user requirements.

As the libmapper ecosystem continues to grow, there will be several opportunities to explore future work related to this project. Firstly, a formal evaluation of libmapper on

the browser will help us better understand how to represent libmapper in a front-end JavaScript API that is approachable to users. This evaluation shall include working with both musicians and web-developers alike to ensure that the tools fit their existing mental models and workflows. Evaluation may take the form of a traditional user-study or rely on a 'hackathon' style of evaluation where users actually build and perform with our tools. Furthermore, additional features that are identified as a result of these formal evaluations will be implemented. This work will benefit both musicians and the research community alike through the continued development of open-source mapping tools.

REFERENCES

[1] Alberto Boem and Luca Turchet. Musical metaverse playgrounds: exploring the design of shared virtual sonic experiences on web browsers. In *2023 4th International Symposium on the Internet of Sounds*, pages 1–9. IEEE, 2023.

[2] Brady Boettcher, Joseph Malloch, Johnty Wang, and Marcelo M Wanderley. Mapper4Live: Using control structures to embed complex mapping tools into Ableton Live. In *NIME 2022*. PubPub, 2021.

[3] Brady Boettcher, Eduardo A. L. Meneses, Christian Frisson, Marcelo M. Wanderley, and Joseph Malloch. Addressing barriers for entry and operation of a distributed signal mapping framework. In *New Interfaces for Musical Expression (NIME'23)*, Mexico City, Mexico, jun 2023.

[4] Myles Borins. From faust to web audio: Compiling faust to javascript using emscripten. In *Linux Audio Conference (LAC-14)*, 2014.

[5] Michel Buffa, Antoine Vidal-Mazuy, and Quentin Plet. Web audio modules: Swiss knife for audio plugin developments on the web platform. In *Internet of Sounds 2023*, 2023.

[6] Filipe Calegario and Filipe Calegario. Evaluation of Probatio 0.2. *Designing Digital Musical Instruments Using Probatio: A Physical Prototyping Toolkit*, pages 81–134, 2019.

[7] Rebecca Fiebrink, Daniel Trueman, Perry R Cook, et al. A meta-instrument for interactive, on-the-fly machine learning. 2009.

[8] Angelo Fraietta, Oliver Bown, and Sam Ferguson. Transparent communication within multiplicities. In *2020 27th Conference of Open Innovations Association (FRUCT)*, pages 61–72. IEEE, 2020.

[9] Andy Hunt and Ross Kirk. Mapping strategies for musical performance. *Trends in gestural control of music*, 21(2000):231–258, 2000.

[10] Andy Hunt and Marcelo M Wanderley. Mapping performer parameters to synthesis engines. *Organised sound*, 7(2):97–108, 2002.

[11] Jerome Lebrun, Michel Buffa, and Jordan Sintes. From blues to rock to jazz: three different web audio tube guitar amplifier simulator plugins. In *WAC 2022-Web Audio Conference 2022*, 2022.

[12] Joseph Malloch, Stephen Sinclair, and Marcelo M Wanderley. Libmapper: (a library for connecting things). In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 3087–3090. 2013.

[13] Joseph Malloch, Stephen Sinclair, and Marcelo M. Wanderley. Generalized multi-instance control mapping for interactive media systems. *IEEE MultiMedia*, 25(1):39–50, 2018.

[14] Joseph Malloch and Marcelo M Wanderley. The T-Stick: From musical interface to musical instrument. In *Proceedings of the 7th international conference on New interfaces for musical expression*, pages 66–70, 2007.

[15] Benjamin Matuszewski. A web-based framework for distributed music system research and creation. *AES-Journal of the Audio Engineering Society Audio-Accoustics-Application*, 2020.

[16] Eduardo Reck Miranda and Marcelo M Wanderley. *New digital musical instruments: control and interaction beyond the keyboard*, volume 21. AR Editions, Inc., 2006.

[17] MozDevNet. Web audio api - web apis: Mdn.

[18] Alex Nieva, Johnty Wang, Joseph Malloch, and Marcelo M Wanderley. The T-Stick: Maintaining a 12 year-old digital musical instrument. In *NIME*, pages 198–199, 2018.

[19] Yann Orlarey, Dominique Fober, and Stéphane Letz. FAUST: an efficient functional approach to dsp programming. *New Computational Paradigms for Computer Music*, pages 65–96, 2009.

[20] Matthew Peachey and Joseph Malloch. FAUSTMapper: Facilitating complex mappings for smart musical instruments. In *2023 4th International Symposium on the Internet of Sounds*, pages 1–6. IEEE, 2023.

[21] Luca Turchet. Smart musical instruments: vision, design principles, and future directions. *IEEE Access*, 7:8944–8963, 2018.

[22] Luca Turchet and Francesco Antoniazzi. Semantic web of musical things: Achieving interoperability in the internet of musical things. *Journal of Web Semantics*, 75:100758, 2023.

[23] Luca Turchet, Mathieu Lagrange, Cristina Rottondi, György Fazekas, Nils Peters, Jan Østergaard, Frederic Font, Tom Bäckström, and Carlo Fischione. The internet of sounds: Convergent trends, insights, and future directions. *IEEE Internet of Things Journal*, 10(13):11264–11292, 2023.

[24] Johnty Wang, Joseph Malloch, Stephen Sinclair, Jonathan Wilansky, Aaron Krajeski, and Marcelo M Wanderley. Webmapper: A tool for visualizing and manipulating mappings in digital musical instruments. In *14th International Symposium on Computer Music Multidisciplinary Research*, page 823, 2019.

[25] Travis West, Baptiste Caramiaux, and Marcelo Wanderley. Making mappings: Examining the design process. In *NIME'20-New Interfaces for Musical Expression*, 2020.