WebAudio plugins in DAWs and for live performance

Michel Buffa
Université Côte d'Azur
CNRS, INRIA
buffa@i3s.unice.fr

Jerome Lebrun
Université Côte d'Azur
CNRS
lebrun@i3s.unice.fr

Guillaume Pellerin
IRCAM
pellerin@ircam.fr

OUTLINE

As part of the ANR WASABI french research project, by collaborating with researchers and developers from other groups or companies, we recently introduced a WebAudio plugin standard (aka VSTs in a browser).

In this demo, we propose to showcase some of the large set of WebAudio plugins we've already been developing. These are running in different host environments such as Amped Studio, an online digital audio workstation or directly in a virtual pedal board application more suited for live performances. The plugins are of different kinds: audio effects, virtual instruments (synthesizers), guitar amp simulations. These plugins can be connected together via audio or midi routes, and written using multiple approaches/programming languages such as JavaScript, DSL languages like FAUST or C/C++.

Our plugin standard is unique today, quite well suited to the Web platforms (each plugin has a URI and can be downloaded and run on the fly by host applications), and relies on recent W3C standards such as WebComponents, WebAudio and WebMidi.

SOME DETAILS

The WebAudio Plugin standard (WAP) is a proposal for unifying different existing approaches used by WebAudio developers. It has been introduced last year by us and other researchers and developers to pave the way to initially make interoperable browser-based audio components [1,2] based on the WebAudio API (https://www.w3.org/TR/webaudio/). This framework is addressed to developers, sound designers and composers who need to share their work and re-use it quickly in a OS independent environment, that is, the context of the Web.

We did the proof of concept that many plugins could be easily included from multiple origins [3]:

1. written in pure JavaScript / Web Audio - we made a dozen of such plugins, including guitar tube amp simulators [4, 5], and ported some others created by other developers (such as a general midi synth, etc.),
2. WebAudioModules [6] (VST/JUCE native plugins ported to WebAssembly/AudioWorklet),
3. written in FAUST [7], a popular DSL for writing DSP code, here again ported to WebAssembly/AudioWorklet. Other sources and importers are on the way (eg. MAX DSP/Pure Data).

The WebAudio plugins are identified by their unique URI and located on local or remote repositories. They can be controllable using midi controllers and connected together inside a host application, i.e. a digital audio workstation, exchanging audio and midi signals. The host application, scans these repositories and then gets the plugin metadata.

Many examples and tutorials are available in the WAP GitHub repo (https://github.com/micbuffa/WebAudioPlugins), and can be tried online (https://wasabi.i3s.unice.fr/dynamicPedalboard/# and in video tutorials (https://youtu.be/pe8zq8O-BFs).

Our current plugin set contains:

- The most common audio effects in the form of "FX pedals" (as they have been designed initially for guitar players): chorus, flanger, phasers, delays, multiple reverbs, phasers, dual pitch shifter, stereo frequency shifter, noise gate, looper, compressor, etc;
- Three tube guitar amp simulations that have been very positively evaluated, showing low latency and robustness, up to the standard of the best native commercial competitors [3, 4, 5], including an accurate Marshall JCM800 emulation and an acoustic guitar simulation. As a complement we also designed a guitar tuner plugin;
- Virtual instruments : multiple synthesizers (DX7 and Oberheim OB-Xd VST ports compiled to WebAssembly, a Korg Minilogue port, a General Midi synthesizer and some original synthesizers)
- Utility plugins for assembling the plugin graph in a more versatile way: mixer, route switcher, graphic equalizer;

- Utility plugins for MIDI: virtual piano keyboards, midi event loggers, midi routing plugins.

We propose to demonstrate how easy this scheme works[1], how it can be included in a usual composition or sound design workflow and how low latency can be achieved, for example, for live performance.

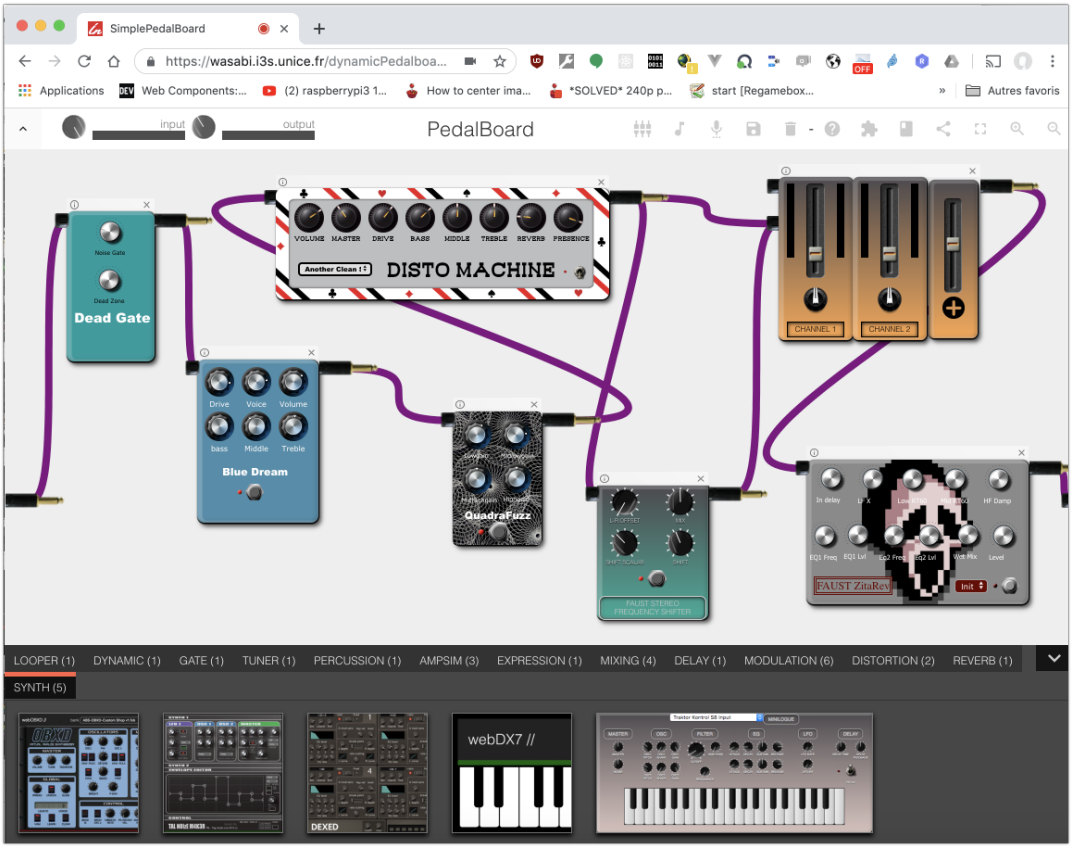

Figure 1: some plugins loaded and patched in the PedalBoard.

[1]Some videos are available such as : https://youtu.be/pe8zg8O-BFs (showcase of some plugin assembled together for real time guitar),, https://www.youtube.com/watch?v=r2SQYh6DiYQ&t=1s (midi plugins mixed with audio plugins) https://youtu.be/PiOD7n3g-Qs (hi-gain web-based guitar amplification, this is the guitar amp simulation designer used to generate the guitar amp sim plugins).

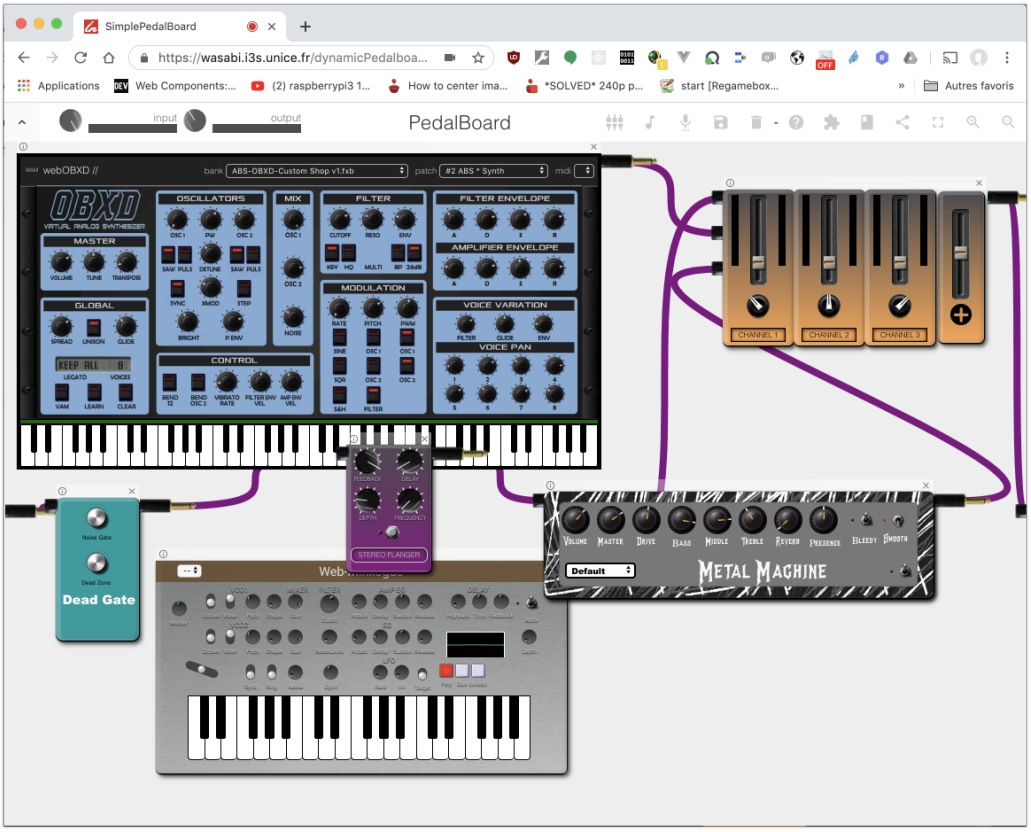

Figure 2: some plugins loaded and patched in the PedalBoard including two WebAudio based synthesizers.

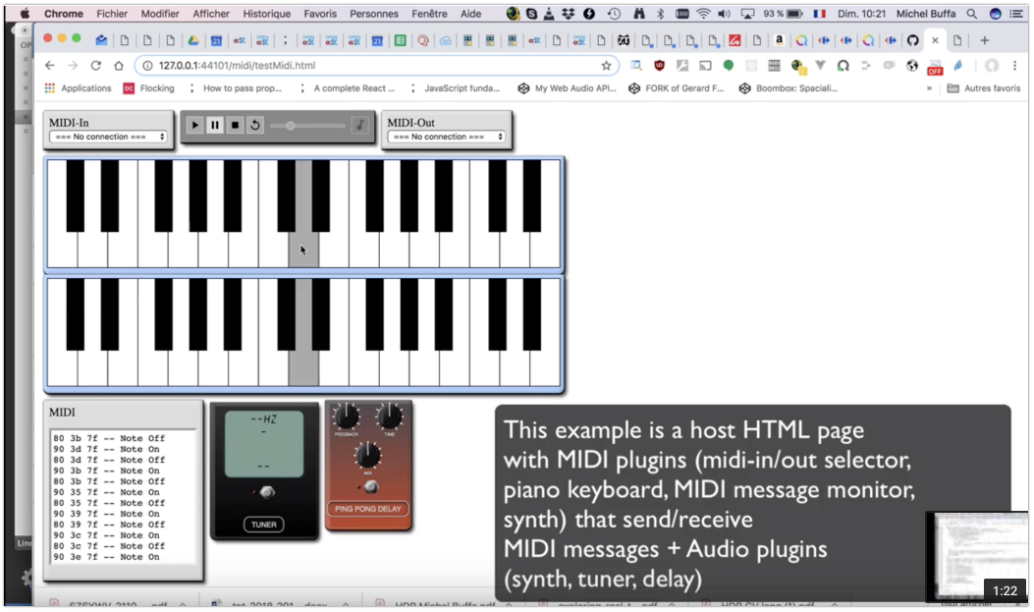

Figure 3: some midi plugins. From top to bottom, left to right: a Midi in selector, a midi player, a midi out selector, a virtual keyboard, a virtual General Midi synthesizer with a keyboard, a midi event logger, a guitar tuner and a delay. Midi events come from the midi in device selected by the first plugin then propagates as all midi plugins are chained (input to output). In the end, the GM synthesizer outputs its signal to two pure audio plugins: the guitar tuner and the delay.

The code of the above WebApp looks is shown in Figure 4.

```
// second parameter is URL
var midiin = new JazzSoftMidiIn(ctx, 'MidiIn');
var midiout = new JazzSoftMidiOut(ctx, 'MidiOut');
var player = new JazzSoftMidiPlayer(ctx, 'MidiPlayer');
var piano1 = new JazzSoftMidiKeyboard(ctx, 'MidiKeyboard');
var piano2 = new JazzSoftMidiKeyboard(ctx, 'MidiKeyboard');
var monitor = new JazzSoftMidiMonitor(ctx, 'MidiMonitor');
var tuner = new WasabiTunerMachine(ctx, '../plugins/PureJS/GuitarTuner');
var delay = new WasabiPingPongDelay(ctx, '../plugins/PureJS/PingPongDelay');

Promise.all([midiin.load(), midiout.load(), player.load(), piano1.load(),
             piano2.load(), monitor.load(), tuner.load(), delay.load()]).then((nnn) => {
  const [midiin_, midiout_, player_, piano1_, piano2_, monitor_, tuner_, delay_] = nnn;
  midiin.loadGui().then((elem) => { document.getElementById('midiin').appendChild(elem); });
  midiout.loadGui().then((elem) => { document.getElementById('midiout').appendChild(elem); });
  piano1.loadGui().then((elem) => {
    document.getElementById('piano2').appendChild(elem);
    piano2.loadGui().then((elem) => { document.getElementById('piano2').appendChild(elem); });
  });
  player.loadGui().then((elem) => {
    document.getElementById('player').appendChild(elem);
  });
  monitor.loadGui().then((elem) => { document.getElementById('monitor').appendChild(elem); });
  tuner.loadGui().then((elem) => { document.getElementById('tuner').appendChild(elem); });
  delay.loadGui().then((elem) => { document.getElementById('delay').appendChild(elem); });

  // MIDI connections
  midiin_.connectMidi(piano1_);
  player_.connectMidi(piano1_);
  piano1_.connectMidi(piano2_);
  piano2_.connectMidi(monitor_);
  monitor_.connectMidi(midiout_);

  // Audio connections
  piano2_.connect(tuner_);
  tuner_.connect(delay_);
  delay_.connect(ctx.destination);
});
```

Figure 4: Part of the code of a WebAudio plugin setup showing how components are loaded from their own URLs and then connected together.

Notice the use of URIs to identify the plugins and the fact that the SDK provides a factory for loading asynchronously and separately the DSP part of the plugin and its GUI (use of the load and loadGui methods. Furthermore, the plugins can be loaded without even knowing their class name, this is demonstrated in “plugin tester webapps” provided with the SDK and shown in Figure 5, in which developers can enter a plugin URI and pass unit tests on the fly. Even remote plugin repositories can be tested on the fly (https://wasabi.i3s.unice.fr/WebAudioPluginBank/testers/explorandtest.html).

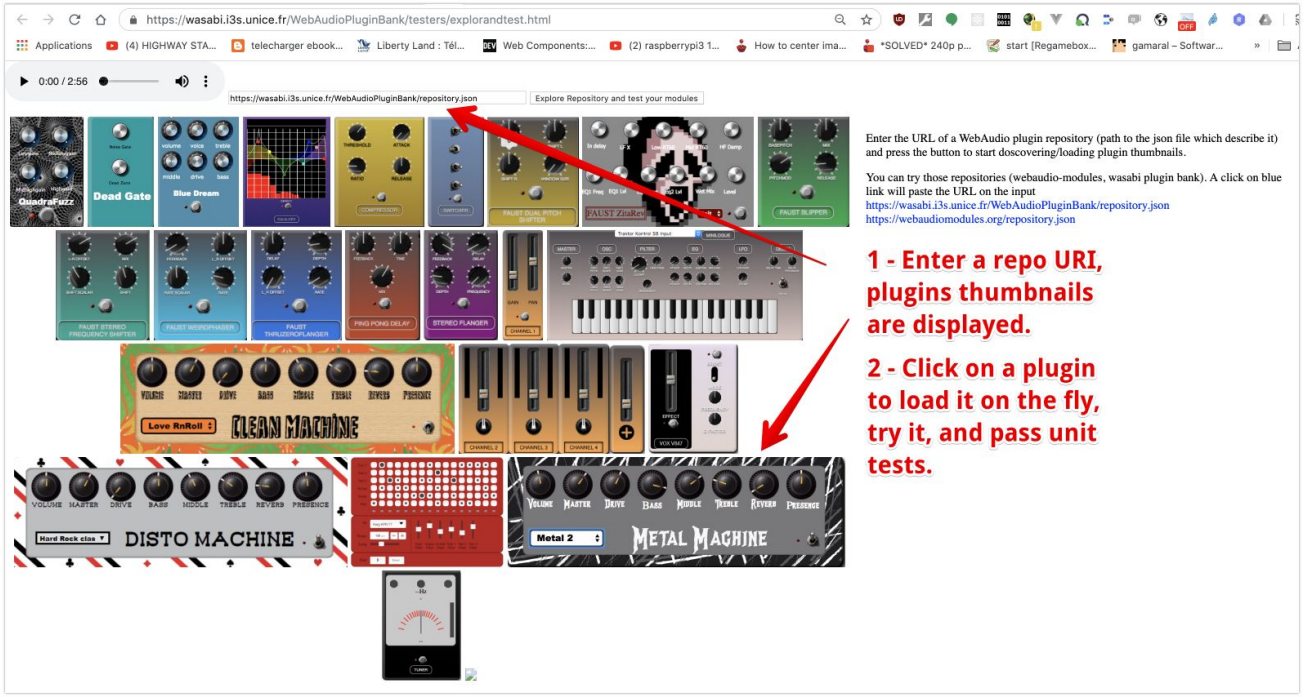

Figure 5: The WebAudio Plugin testing environment.

DEMO SETTINGS

For the demo we would need a big screen or TV, a couple of decent amplified speakers and a low latency sound card.

A typical demo is:

- setup a custom plugin in an online editor and test it (optional),
- load a bunch of plugins from the list or a URLs or a preset,
- link the plugins together in the patch environment,
- load an audio sample or an external live input,
- play,
- enjoy,
- save and share your work.

The demos will be presented through two OS: OSX and Linux.

ACKNOWLEDGMENTS

ElMahdi Korfed, Guillaume Etevenard and Jordan Sintes who helped developing these tools. French Research National Agency (ANR) and the WASABI project team (contract ANR-16-CE23-0017-01).

REFERENCES

[1] Michel Buffa, Jerome Lebrun, Jari Kleimola, Oliver Larkin, Guillaume Pellerin, et al.. WAP: Ideas for a Web Audio Plug-in Standard. Web Audio Conf, Sep 2018, Berlin, France. ⟨hal-01893660⟩

[2] Michel Buffa, Jerome Lebrun, Jari Kleimola, Oliver Larkin, Stephane Letz. Towards an open Web Audio plug-in standard. WWW2018 - TheWebConf 2018 : The Web Conference, 27th International World Wide Web Conference, Mar 2018, Lyon, France. ⟨hal-01721483⟩

[3] Michel Buffa, Jerome Lebrun. Guitarists will be happy: guitar tube amp simulators and FX pedals in a virtual pedal board, and more!. Web Audio Conf 2018, Sep 2018, Berlin, Germany. ⟨hal-01893681⟩

[4] Buffa, M. and Lebrun, J. 2017. Real time tube guitar amplifier simulation using WebAudio. In Proc. 3rd Web Audio Conference (WAC 2017). London, UK.

[5] Buffa, M. and Lebrun, J. 2017. Web Audio Guitar Tube Amplifier vs Native Simulations. In Proc. 3rd Web Audio Conference (WAC 2017). London, UK.

[6] Kleimola, J. and Larkin, O. 2015. Web audio modules. In Proc. 12th Sound and Music Computing Conference (SMC 2015), Maynooth, Ireland.

[7] Letz, S., Orlarey, Y. & Fober, D. 2017. Compiling Faust audio DSP code to WebAssembly. In Proc. 3rd Web Audio Conference (WAC 2017), London, United Kingdom.