WebAudioXML: Proposing a new standard for structuring web audio

Hans Lindetorp
Royal College of Music in Stockholm
KTH Royal Institution of Technology
hans.lindetorp@kmh.se

Kjetil Falkenberg
KTH Royal Institution of Technology
kjetil@kth.se

ABSTRACT

We present WebAudioXML as a suggested candidate for establishing a standard for describing Web Audio configurations. The aim is to lower the barrier for artistic creators for working within web audio applications as well as providing a modular system that can integrate into larger applications. WebAudioXML provides means for making interactive music without having to learn a programming language like JavaScript and consists of an XML syntax specification and a parser. The framework has been developed with and tested by audio experts and lecturers from music production and Sound and Music Computing. Workshop participants report that WebAudioXML has potential in keeping focus on the creative process instead of web development. We argue that an XML standard for Web Audio configurations would be beneficial for modular and collaborative development and therefore recommend a wider discussion on the topic. With the discussion we aim to promote the artistic in the making of interactive audio applications.

1. INTRODUCTION

There are several platforms available for composers and producers who want to make music for digital interactive environments. In addition to native formats, open web standards have recently become an increasingly interesting platform for this purpose, supporting a wide range of clients including ordinary smartphones. Composers and producers that have interest in producing music within this context may experience that web technologies and mapping user interactions to audio can be too big of an obstacle, and this project therefore seeks to bridge this gap with a framework that does not require any programming. We are proposing and evaluating such a framework for making web-based audio applications using XML only. Compared to other related frameworks, it has a simpler syntax and introduces a hierarchical structure that makes it easy to map user actions to audio parameters. This hopefully leads to shifting focus from the technological to the artistic in the making of interactive audio applications.

1.1 HTML5 and Web Audio API

With HTML5, video and audio became a native part of the standard web technology. Initially only supporting the simple `<Audio>` element for buffering and playing audio files, the Web Audio API (application programming interface) was later added to support dynamic and interactive implementation of audio. Since 2018, Web Audio API has been a "W3C Candidate Recommendation" [1], and web pages can now host advanced audio applications without any need for compilations or installations. Web Audio API offers advanced audio processing in all major browsers on desktop as well as mobile devices and lets the developer control deep aspects of audio signal processing using ordinary JavaScript.

The introduction of Web Audio API has resulted in academic activities, [1] many online demo examples, [2] commercial tools built for learning, [3] and production of music. [4] While there apparently exist great examples, and although the technology and standard is in active development, we argue that there still is a great potential for new use cases to be explored and new groups of audio designers to discover the possibilities and benefits of Web Audio.

1.2 Challenges and aims

Through experience of teaching interactive music production at the Royal College of Music (KMH) in Stockholm, we have identified a population of students who have a deep understanding of music and audio production but little or no programming experience. For this group, programming the audio signal routing and mapping variables to audio parameters have a relatively steep learning curve which often have lead to less interest and students dropping artistic ambitions in their interactive music projects [2]. We have also noticed that this student group generally finds HTML syntax easier to understand than JavaScript. This led us to develop a framework that uses a declarative approach and to use XML rather than JSON to represent Web Audio nodes. We have no reason to believe that the findings about our particular student group are unique, and we even assume that the approach can aid web audio application development in general for as well pedagogical, artistic as commercial purposes.

We therefore propose a new standard called WebAudioXML that aims at lowering the barrier for

[1] e.g., the Web Audio Conference, https://webaudioconf.com
[2] e.g., https://github.com/mdn/webaudio-examples
[3] e.g. Music First, https://www.musicfirst.com
[4] e.g. Soundtrap, https://www.soundtrap.com

artistic creators to enter into web audio application development as well as offering a modular approach to integrate audio objects into larger applications. WebAudioXML also contributes to the Sound and Music Computing and NIME communities with tools for rapid prototyping, online demos and teaching audio and interaction. We argue that an XML standard for describing Web Audio configurations would be beneficial and contribute to the Web Audio Conference community with WebAudioXML being one candidate for such a standard. With this paper, we also wish to initiate a wider discussion about the topic.

2. A MINIMAL WORKING EXAMPLE

To give an understanding of what WebAudioXML offers, we present a minimal working example of an interactive audio application built with three different environments: WebAudioXML, JavaScript, and, to compare with a standard environment for audio programming, also with Pure Data (http://pure-data.info). The application consists of an oscillator with a sawtooth waveform playing through a low-pass filter connected via a gain to the audio output. MouseDown events play the sound and MouseUp events mute the sound. Moving the cursor along the X-axis controls the frequency of the oscillator and moving it along the Y-axis controls the cut-off frequency of the low-pass filter (both directly mapped to cursor pixel-position).

2.1 WebAudioXML code

The audio nodes are declared using XML elements inside a `<Chain>` element which makes them automatically connected in a chain. In Web Audio, the low-pass filter is included in a biquad filter function. The mapping between mouse interactions and a particular parameter is done using the 'follow' attribute.

```
<?xml version="1.0" encoding="UTF-8"?>
<Audio version="1.0">
  <Chain>

    <OscillatorNode type="sawtooth">
      <frequency follow="clientX"></frequency>
    </OscillatorNode>

    <BiquadFilterNode>
      <frequency follow="clientY"></frequency>
    </BiquadFilterNode>

    <GainNode>
      <gain follow="mousedown"></gain>
    </GainNode>

  </Chain>
</Audio>
```

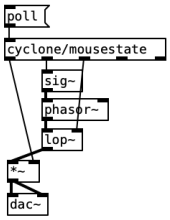
2.2 JavaScript code

The JavaScript code is the conventional way of building an application with Web Audio. It includes initializing an AudioContext, creating the audio nodes, connecting them to each other, to add eventListeners to the mousemove event, and finally to map the clientX and clientY variables to the desired audio parameters.

```
// setup the audio nodes
var ctx = new AudioContext();
var oscNode = ctx.createOscillator();
oscNode.type = "sawtooth";
var gainNode = ctx.createGain();
gainNode.gain.value = 0;
var filterNode = ctx.createBiquadFilter();

oscNode.connect(gainNode);
gainNode.connect(filterNode);
filterNode.connect(ctx.destination);
oscNode.start();


// map mouse events to audio parameters
document.addEventListener("mouseup", e => {
  gainNode.gain
   .setTargetAtTime(0, ctx.currentTime, 0);
});

document.addEventListener("mousemove", e => {
  oscNode.frequency
  .setTargetAtTime(e.clientX, ctx.currentTime, 0);

  filterNode.frequency
  .setTargetAtTime(e.clientY, ctx.currentTime, 0);
});
```

2.3 Pure Data code

We include a Pure Data (Pd) example for comparison. Pd patches are programmed using graphical elements:



Readable code is generated from the patches:

```
#N canvas 0 23 450 300 10;
#X msg 121 90 poll;
#X obj 121 111 cyclone/mousestate;
#X obj 147 133 sig~;
#X obj 147 155 phasor~;
#X obj 147 177 lop~;
#X obj 121 201 *~;
#X obj 121 223 dac~;
#X connect 0 0 1 0;
#X connect 1 0 5 0;
#X connect 1 1 2 0;
#X connect 1 2 4 1;
#X connect 2 0 3 0;
#X connect 3 0 4 0;
#X connect 4 0 5 1;
#X connect 5 0 6 0;
#X connect 5 0 6 1;
```

Mouse events cannot be captured without a so-called external library, and in the example we use the cyclone library. Unlike the WebAudioXML and JavaScript examples, this code cannot be run in a browser.

3. RELATED WORK

There are several technologies and projects aiming to solve similar challenges and cater to similar needs as we do with WebAudioXML. In the following section we describe some of these and make comparisons with WebAudioXML.

3.1 MusicXML

MusicXML was released 2004 and serves as a standard file format for sheet music [3]. It provides a way of sharing music scores between different notation software. Like MusicXML, WebAudioXML provides a standard format but it differs in that it does not at all describe musical structure. Instead, it defines how the audio signals are routed. The two formats can practically complement each other and make it possible to build a web application rendering the notes specified in a MusicXML file through an audio setup specified in a WebAudioXML-file, which includes for instance virtual instruments, mixers and effects.

3.2 Tone.js

Tone.js [4] is a JavaScript frameworks aimed to simplify the development of audio applications. It adds high level functions and objects to create a Web Audio configuration and comes with a library of effects like chorus, reverb and visualizers. WebAudioXML, on the contrary, does not contain any effects but rather encourages active community participation to build objects that easily interconnect using a standardized XML syntax.

3.3 Web Audio Modules (WAM)

Web Audio Modules (WAM) is an API that enables virtual instruments and effects processors (similar to those in Digital Audio Workstations) to integrate into web pages using Web Audio API [5]. WAM uses a single Web Audio node to integrate audio algorithm development made with JavaScript or cross-compiled C/C++. One aim is to standardize the integration of high performance audio plug-ins, while WebAudioXML only builds on top of Web Audio API and rather standardizes the configuration of the audio nodes using XML. WebAudioXML can therefore potentially describe a configuration that includes a WAM plug-in.

3.4 Web Audio Plug-in (WAP)

Web Audio Plug-in (WAP) is a proposal for an open Web Audio plug-in standard [6]. It aims at standardizing the integration of audio effects and instruments into web-based host applications. It supports WAM and FAUST [7] plug-ins as well as JavaScript plug-ins made of multiple AudioNodes using Web Audio API. WebAudioXML aims at solving similar challenges as WAP but uses XML syntax rather than JavaScript. Potentially, WebAudioXML can be used both as a way of building JavaScript-based plug-ins for WAP and as a format for describing configurations that include WAP objects.

3.5 WebAudio frameworks not maintained

There are several WebAudio related frameworks that are no longer maintained. One is WAAX [8] which adds JavaScript objects in a similar way to tone.js. Another one is Audio Tags [9] - a Mozilla Hacks experiment from 2014 offering reusable audio blocks built with HTML Custom Elements to give developers an easy way to experiment with Web Audio. Audio Tags aimed at solving similar challenges as WebAudioXML faces, but solved them from a graphical interface perspective while WebAudioXML focuses on the audio configuration.

3.6 Other systems

Some of the most common programming environments within the SMC field are CSound, Max/MSP, Pure Data and Supercollider. None of these were initially intended to be used for web applications as they were developed long before web browsers were capable of music programming [10]. In recent years there have been made several attempts to integrate these environments in browsers, such as Flocking [11], WebPd [12], Gibberish/Interface [13], and PNaCl [14].

Serving as an example, Pure Data was developed over 25 years ago to allow realtime collaboration using network protocols and on different platforms [15]. While it is relatively straightforward to use Pd patches in mobile applications using the libpd library [16], including external objects like mousestate complicates development, and using Pure Data from web pages is still a work in progress (mainly through WebPd, (https://github.com/sebpiq/WebPd/issues/113). WebPd uses AudioWorklet to integrate into the Web Audio environment and it could potentially be possible to use WebAudioXML to define an audio node tree where one node is holding a Pd patch.

4. IMPLEMENTATION

This section describes the process of designing and evaluating WebAudioXML. It also contains basic code examples to present the syntax and to illustrate how to use WebAudioXML. There are more examples using complex configurations available at https://github.com/hanslindetorp/WebAudioXML.

4.1 Design process

The first ideas behind WebAudioXML was shaped alongside an interactive music project with students at KMH (such student contribution is an attested approach in our research [17]). The students created visual content with HTML and implemented audio loops with iMusic [18] to make them respond to user interactions. We noticed that the XML API in iMusic seemed to support their artistic visions better than the JavaScript API used in earlier projects. This encouraged us to implement an XML language for controlling the audio routing as well. The design objective was to specify an XML syntax that would make adding interactive audio components in a web page similar to adding visual content using HTML. After early tests in the student project, the code was separated from iMusic and reconstructed as a first prototype of WebAudioXML.

4.1.1 Feedback session 1

This prototype was tested in a workshop with seven experts from the Sound and Music Computing group at KTH Royal Institute of Technology. The workshop included a

presentation, exercises using WebAudioXML with CodePen (https://codepen.io/collection/DjaYkE), and a group discussion. The participants are experienced in audio processing using tools like Max/MSP, Pure Data, SuperCollider and various Digital Audio Workstations but none of them had earlier experience of working with Web Audio API.

The discussion was recorded and transcribed to collect the participants aspects of using the framework and suggestions for further development. The most important aspects from the discussion were that the system requires a clear and flexible routing system supporting object-based mixing, a modular way of structuring the audio node tree into multiple, reusable files, and the advantages of being able to evaluate any block of the code separately. The group also pointed out the great potential the system has for teaching audio signal processing, but also that the metaphor used requires an understanding of audio production concepts like mixers, channels and synthesizers.

4.1.2 Feedback session 2

After the workshop, some of the suggested features were implemented, and WebAudioXML was moved from prototype state to beta version. At this stage the framework was presented and discussed by three lecturers teaching music production and electroacoustic music at KMH. Their primary tools for making music are Logic ProX, Max/MSP and SuperCollider. The interviews were recorded and transcribed to collect emerging issues and perspectives.

The informants were overall very positive about the concept, the syntax and the possibilities Web Audio offers for mainly pedagogical use but also artistic work. They pointed out the benefits of a web-based system for distributing interactive learning objects needless of any software installations and the fact that everyone has got the physical hardware needed right in their pockets.

Compared to the syntax in SuperCollider the semantic and clear structure of XML was appreciated even if it was also imagined that the code could become large and difficult to overview for a complex configuration. The informants thought the modular structure with separate files was a good solution to the problem and pointed out the possibility to integrate patches from Pd and other tools using the AudioWorkletNode possibly being a key feature to reach the electroacoustic music community with the technology. All participants were a bit reluctant to view Web Audio as a professional alternative—partly because the music community is driven by its own standard tools that reinforce themselves but also because the lack of professional tools for debugging audio and for performance control. That said, they suggested WebAudioXML to be presented with powerful and complex interactive demos that triumphs in accessibility, compatibility and delivery aspects.

4.1.3 Feature requests

The lecturer teaching music production mainly focused on synthesizer-related questions asking for a syntax that conforms to synthesizer programming style. This would imply using milliseconds and percent as units for envelopes, and possibilities to connect MIDI velocity to control any aspects of a sound. Another feature request was modulated pulse width for the square wave oscillator to mimic the behaviour of many analogue synthesizers. A suggestion from one of the lecturers of electroacoustic music was support for multi-touch events that connects different fingers to different audio parameters.

All discussions also devoted time to reflect on the relationship between code and a visual interface. It appeared that the participants found it hard to code audio without a visual representation. WebAudioXML is not aimed at designing a graphical user interface, so controllers like sliders and knobs should arguably not be a part of the specification. But for developing purposes, the attribute 'controls' was added for the topmost `<Audio>` element. By setting it to 'true', standard `<input type="range">` HTML elements are generated for all parameters in the configuration. For building nice looking interfaces though, it's better to use frameworks like Web audio controls (https://github.com/g200kg/webaudio-controls) and connect the 'change'-events to control variables in WebAudioXML.

4.2 Description

The specification for WebAudioXML defines how to structure Audio objects in a hierarchical, modular way using XML. For integration in a web-based application, it requires WebAudioXML.js, a JavaScript library that parses the XML and creates and connects all Web Audio nodes into a tree-like structure, called an Audio Graph. The code is open source and freely available for download from GitHub (https://github.com/hanslindetorp/WebAudioXML). XML is normally case sensitive but for compatibility reasons WebAudioXML is not. The style used in the following examples conforms to the conventions used in the Web Audio API reference.

4.2.1 Adding WebAudioXML to a web page

WebAudioXML is added to a web page using one line of HTML-code added to the `<head>` element or at the end of the body. The 'data-source' attribute specifies the path to a WebAudioXML document.

```
<script
  src="WebAudioXML.js" data-source="audio.xml">
</script>
```

The 'data-source' can be a relative path or an address pointing to a remote file. It is also possible to point to an embedded XML element within the HTML-file using the 'id' attribute as an identifier.

4.2.2 XML structure

The following shows the simplest configuration using only one OscillatorNode connected to the Web Audio destination:

```
<?xml version="1.0" encoding="UTF-8"?>
<Audio version="1.0">
  <OscillatorNode></OscillatorNode>
</Audio>
```

The structure of the XML-data follows some basic rules. The root element is named `<Audio>` and the other elements can be either a valid Web Audio node, a Web Audio parameter or one of the following custom elements: `<Mixer>`, `<Chain>`, `<Synth>`, `<Voice>`, `<Send>` or `<Envelope>`. The following section describes the different elements in more detail.

4.2.3 Native Web Audio nodes and parameters

Any valid Web Audio node can potentially be specified using WebAudioXML. The name structure follows the Web Audio API specification. Currently, the following nodes are implemented and tested: AudioBufferSourceNode, MediaStreamAudioSourceNode, BiquadFilterNode, ConvolverNode, DelayNode, DynamicsCompressorNode, GainNode, OscillatorNode, StereoPannerNode, and WaveShaperNode. Any valid Web Audio parameters can be set using attributes. The following example shows an oscillator node with type set to 'sawtooth' and frequency set to 880 Hz.

```
<OscillatorNode
  type="sawtooth"
  frequency="880">
</OscillatorNode>
```

To extend the Oscillator type beyond the standard waveforms of sine, triangle, sawtooth and square, it can also be set to an external JSON file containing PeriodicWave data. Example files are available at https://github.com/GoogleChromeLabs/web-audio-samples/tree/gh-pages/samples/audio/wave-tables.

4.2.4 Events and variables

All audio parameters can dynamically be linked to any variable in the host application using an attribute called 'follow'. WebAudioXML has a number of touch, pointer and DeviceOrientation variables predefined and can respond to any variable through the `setVariable(key, value)` method. The variable can be mapped using a 'map' attribute. This attribute can specify how the source variable value shall be mapped to the target audio parameter using five values: "minIn, maxIn, minOut, maxOut, exponent". The exponent defaults to 1 and can be set to any value to create an exponential curve or any mathematical expression using x to represent the source value. A comprehensive description on mapping variables is found in the online documentation. [5] The following code creates a BiquadFilterNode with the cutoff frequency controlled by the horizontal mouse position between 0-1000px mapped exponentially to 20-20000 Hz.

```
<BiquadFilterNode>
  <frequency
    follow="clientX"
    map="0, 1000, 20, 20000, 2">
  </frequency>
</BiquadFilterNode>
```

There is also a possibility to connect an audio parameter to HTML DOM elements using eventListeners. This makes it possible to respond to any event in the host application by specifying the target using three comma separated values: CSS selectors [6], eventName, propertyOfTheEvent. The following example makes a slider element with id="slider1" control the frequency of an oscillator.

```
<!-- HTML-->
<input type="range"
  min="20" max="1000" value="440"
id="slider1" />


<!-- XML-->
<OscillatorNode>
  <frequency
    follow="#slider1, input, event.target.value">
  </frequency>
</OscillatorNode>
```

4.2.5 Custom elements

In addition to the native Web Audio nodes, WebAudioXML has some custom elements; `<Mixer>`, `<Chain>`, `<Synth>`, `<Voice>`, `<Send>` and `<Envelope>`. They serve as extensions to control the structure of events, audio signals and parameter values. The `<Mixer>` element is a container that mixes the audio of all child elements to its output. It shares this feature with the root element `<Audio>` but differs in the aspect that it can be used on any hierarchical level in the audio node tree.

The `<Chain>` element is a container that connects all its child elements in a chain (like inserts on a channel on a mixing console). The output of the last child element is connected to the chain's output. A `<Synth>` element is a container for one or several voice elements that will be multiplied to match the number of voices set by the attribute 'voices' in the `<Synth>` element. It mixes the sound of all voices to its output similar to the `<Mixer>` element but adds the feature of mapping note events (from a graphical interface or via Web MIDI) to trig voices in a manner similar to synthesizers.

The `<Envelope>` element controls a parameter of an audio node. It can either be placed inside a parameter element or anywhere in the structure with its 'output' attribute set to target one or multiple parameters. There is an attribute called 'ADSR' that works as a typical attack–decay–sustain–release-envelope in a synthesizer. The time unit defaults to milliseconds and the range for sustain is in percent. To control the actual output value that matches the target parameter, there is an attribute called 'max'. The following example shows an envelope controlling gain level of a GainNode where the attack is set to 10 ms, the decay to 300 ms, the sustain level to 20% and the release set to 200 ms. The 'max' attribute is set to 1 to map the envelope to the max gain level.

```
<GainNode>
  <gain>
    <Envelope
      ADSR="10, 300, 20, 200"
      max="1">
    </Envelope>
  </gain>
</GainNode>
```

[5] https://github.com/hanslindetorp/WebAudioXML/wiki

[6] https://www.w3schools.com/cssref/css_selectors.asp

4.2.6 *Including separate XML-files*

WebAudioXML supports XML Inclusions [7] and the use of `<xi:include>` elements to split the code into separate XML files. This makes collaborative projects easier and encourages a modular approach using building blocks in WebAudioXML. The 'href' attribute specifies the external XML file. There is no limit for the hierarchical depth of inclusions but there is a restriction that stops infinite recursive loops from happening. External files should conform to the standard WebAudioXML structure.

4.2.7 *Signal routing*

There are ways to override the audio signal routing described above. Any element can be disconnected and routed to any other element(s) by setting its 'output' attribute. The syntax uses CSS selectors to specify the target element using class names or id. By using a selector the matches multiple elements, the signal will be connected to all of them. The `<Send>` element is a special element used for routing purposes. It is supposed to be placed inside a `<Chain>` element and acts like send bus in a mixing console with an 'output' and a 'gain' attribute. It will not break the standard signal flow in the `<Chain>` element. The following example shows a chain with an AudioBufferSourceNode and a send element. The send is connected to a `<ConvolverNode>` and another `<Chain>` element that creates a delay. The latter uses a `<Send>` element to create a feedback loop with delayTime set to 500ms. The `<Send>` element has the 'gain' set to -3dB, causing the volume for each feedback to decrease.

```
<Mixer timeUnit="ms">

  <Chain>
    <AudioBufferSourceNode src="loop.mp3">
    </AudioBufferSourceNode>
    <Send output="#reverb, #delay"></Send>
  </Chain>

  <ConvolverNode id="reverb"
    src="large-hall.mp3">
  </ConvolverNode>

  <Chain id="delay">
    <GainNode gain="-3dB"></GainNode>
    <DelayNode delayTime="500"></DelayNode>
    <Send output="#delay" gain="-3dB"></Send>
  </Chain>

</Mixer>
```

4.2.8 *Integration in larger applications*

In addition to the simple integration shown in the examples above, WebAudioXML can be a module in a larger application. The following example shows how WebAudioXML.js is required from another JavaScript file and a reference to the object is stored in a variable. By using the method `setVariable(key, value)`, any audio parameter following that variable inside the object will be updated. The example is simple but could scale to include any number of instances and configuration files for WebAudioXML.

[7] https://www.w3.org/TR/xinclude-11/

```
// JavaScript

var WebAudioXML = require('./WebAudioXML.js');
var ctx = new AudioContext();
var osc = new WebAudioXML("osc.xml", ctx);

// update the variable inside osc
osc.setVariable("freq", 100);


<!-- XML fragment -->
<OscillatorNode>
  <frequency
    follow="freq">
  </frequency>
</OscillatorNode>
```

5. DISCUSSION

From starting off as a simple audio signal routing system, WebAudioXML evolved into a high level modular XML layer on top of the Web Audio API interface. It proved to easily integrate into online tools like CodePen [8] and jsFiddle [9] and appealed to the lecturers as a useful framework for teaching audio signal processing and modular sound synthesis. The result indicates that WebAudioXML could be a potential platform for collective, modular development of interactive digital musical instruments where different teams develop separate audio objects that dynamically link into bigger projects.

The interviews touched upon the importance of a visual interface that presents the audio node tree and gives access to the parameter values via sliders, knobs and other objects. A useful feature to inspect the audio node structure created in Web Audio is the Web Audio inspector plug-in for Chrome but it is far from as powerful as the inspector for CSS and debugger for JavaScript in most browsers. Another feature that potentially could be a part of WebAudioXML is to specify how the output of an audio signal could be received by external code. That could e.g. include a simple way to connect a `<canvas>` element to the AnalyserNode, generating oscilloscopes and various visual representations of the FFT algorithm.

During the development, it appeared that it was not practical to rely on Web Audio nodes only but to allow for some custom elements like the `<Mixer>`, `<Chain>` and `<Envelope>` elements. While there are no standards yet, we took inspiration for some of them from the Audio Tags project and tried to make up as few new elements as possible. It was interesting to get input from different users and to discuss different possible solutions of a problem. To mention a few, it was important for the music production lecturer to use time and value units for envelopes that he recognised from music production applications. Also, gain levels was preferred to be set with decibels rather than to the signal power value. Another feedback was about the routing system relying on CSS selectors—which is common knowledge for web developers but unknown to the workshop participants of this study. This implies that the style of a new syntax can be perceived as familiar for some and unfamiliar for others and the matter of choice could have an impact on how well the technology is received in different communities.

[8] https://codepen.io
[9] https://jsfiddle.net

5.1 Future development

There are plenty of possible ways for further development of WebAudioXML. One step is to implement all Web Audio node types and to run proper tests on all major platforms. It would also be valuable to add easy integration for external frameworks like tone.js and to make WebAudioXML.js compatible with the Web Audio Plug-in format. Other valuable features would be an easy configurable implementation of Web MIDI events to control any parameter in the system and an API for custom elements to extend WebAudioXML.

A feature that is not yet fully implemented is the connection between variables in the host application and the audio parameters in WebAudioXML. To make the modular structure work properly it would require a variable handling where they are safely distributed throughout the hierarchical XML element structure. Another area for possible improvements is the support for multiple channel output. So far, the only implemented feature that relates to output channels is the StereoPannerNode. Web Audio API supports at least 32 channels and a way of routing sounds in a multi-channel setup with WebAudioXML is already a requested feature for future projects at KMH.

6. SUMMARY

The WebAudioXML framework proved to work as expected. The students with web coding background managed to use the framework making the audio more responsive to web page interactions while the audio experts identified a great potential for prototyping and teaching audio signal processing using the XML syntax. The composers and music production educators pointed out important areas to improve and saw possibilities for creative and as well as pedagogical projects for WebAudioXML. The code is open source and the development will continue to be available on GitHub for other developers and contributors (https://github.com/hanslindetorp/WebAudioXML).

Acknowledgments

Chris Rogers for inventing Web Audio API, The Sound and Music Computing group at KTH Royal Institute of Technology in Stockholm, Peter Schyborger, Mattias Sköld and Mattias Petersson at the Royal College of Music in Stockholm for participation in workshop and interviews.

7. REFERENCES

[1] P. Adenot and R. Toy, *Web Audio API: W3C Candidate Recommendation, 18 September 2018*, 2018 (accessed February 18, 2020). [Online]. Available: https://www.w3.org/TR/2018/CR-webaudio-20180918/

[2] H. Lindetorp, "Immersive and interactive music for everyone," in *Proceedings of the Nordic Sound and Music Computing Conference*, 2019, pp. 16–20. [Online]. Available: http://smcsweden.se/proceedings/NordicSMC_ISon_2019_Proceedings.pdf

[3] M. Good *et al.*, "MusicXML: An internet-friendly format for sheet music," in *Xml conference and expo*, 2001, pp. 03–04.

[4] Y. Mann, "Interactive music with tone.js," in *Proceedings of the 1st annual Web Audio Conference*, 2015.

[5] J. Kleimola and O. Larkin, "Web audio modules," in *Proc. 12th Sound and Music Computing Conference*, 2015.

[6] M. Buffa, J. Lebrun, J. Kleimola, O. Larkin, and S. Letz, "Towards an open Web Audio plugin standard," in *Companion Proceedings of the The Web Conference 2018*, 2018, pp. 759–766.

[7] S. Letz, Y. Orlarey, and D. Fober, "Compiling Faust audio DSP code to WebAssembly," 2017.

[8] H. Choi and J. Berger, "WAAX: Web Audio API eXtension," in *NIME*, 2013, pp. 499–502.

[9] S. Penadés and A. Fabbro, *Audio Tags: Web Components + Web Audio = <3*, 2014 (accessed February 18, 2020). [Online]. Available: https://hacks.mozilla.org/2014/03/audio-tags-web-components-web-audio

[10] L. Wyse and S. Subramanian, "The viability of the web browser as a computer music platform," *Computer Music Journal*, vol. 37, no. 4, pp. 10–23, dec 2013.

[11] C. B. Clark and A. Tindale, "Flocking: a framework for declarative music-making on the web," in *Sound and Music Computing Conference*, 2014, pp. 1550–1557.

[12] C. McCormick and S. Piquemal, *WebPd*, 2013 (accessed February 18, 2020. [Online]. Available: github.com/sebpiq/WebPd

[13] C. Roberts, G. Wakefield, M. Wright, and J. Kuchera-Morin, "Designing musical instruments for the browser," *Computer Music Journal*, vol. 39, no. 1, pp. 27–40, mar 2015.

[14] V. Lazzarini, E. Costello, S. Yi, and J. Fitch, "Csound on the web," in *Proceedings of the Linux Audio Conference*, 2014, pp. 77–84.

[15] J. Kreidler, *Loadbang*. Wolke Verlagsges. Mbh, 2013.

[16] P. Brinkmann, P. Kirn, R. Lawler, C. McCormick, M. Roth, and H.-C. Steiner, "Embedding pure data with libpd," in *Proceedings of the Pure Data Convention*, vol. 291, 2011.

[17] K. F. Hansen, R. Bresin, A. Holzapfel, S. Pauletto, T. Gulz, H. Lindetorp, O. Misgeld, and M. Sköld, "Student involvement in sound and music research: Current practices at KTH and KMH," in *Proceedings of the Nordic Sound and Music Computing Conference*, 2019, pp. 36–41.

[18] H. Lindetorp, *iMusic: JavaScript framework for interactive music*, 2016 (accessed February 18, 2020). [Online]. Available: https://github.com/hanslindetorp/imusic