AUDIO PARAMETER MAPPING MADE EXPLICIT USING WEBAUDIOXML

Hans LINDETORP (hans.lindetorp@kmh.se) [1,2] and **Kjetil FALKENBERG** (kjetil@kth.se) [2]

[1] *Department of Music Production*, **KMH Royal College of Music**, Stockholm, Sweden
[2] *School of Electrical Engineering and Computer Science*, **KTH Royal Institute of Technology**, Stockholm, Sweden

ABSTRACT

Sonification using audio parameter mapping involves both aesthetic and technical challenges and requires interdisciplinary skills on a high level to produce a successful result. With the aim to lower the barrier for students to enter the field of sonification, we developed and presented WebAudioXML at SMC2020. Since then, more than 40 student projects has successfully proven that the technology is highly beneficial for non-programmers to learn how to create interactive web audio applications. With this study, we present new feature for WebAudioXML that also makes advanced audio parameter mapping, data interpolation and value conversion more accessible and easy to assess. Three student projects act as base for the syntax definition and by using an annotated portfolio and video recorded interviews with experts from the sound and music computing community, we present important insights from the project. The participants contributed with critical feedback and questions that helped us to better understand the strengths and weaknesses with the proposed syntax. We conclude that the technology is robust and useful and present new ideas that emerged from this study.

1. INTRODUCTION

Audio parameter mapping is a commonly used approach in a wide range of applications spanning from data sonification projects to physical, digital instruments with knobs and sliders. Common for these applications is that variables, e.g., statistical data or the value of a knob on a synthesizer, are used to shape or control the playback of a sound.

Sonification using audio parameter mapping involves both aesthetic and technical challenges and requires interdisciplinary skills on a high level to produce a successful result. The process includes data preparation, sound synthesis, mapping parameters and finally listening and tuning the settings [1] to produce a meaningful result. It also requires an understanding of auditory perception [2], sound design, and musical composition. Studies show that the relationship between the original data and the auditory domain is far from being a simple linear link [3] and even if there have been attempts at trying to formalize ways of describing those relationships [4] there is still a great potential for further research. Various programming languages have been developed aiming at meeting those needs, including CSound [5] Max/MSP [6], Pure Data [7], SuperCollider [8] and ChucK [9]. There are also tools available for creating sonifications like xSonify [10], Toolkit for Sonification [11], SonifYer [12] and SoniPy [13] but most available tools arguably requires both programming and audio synthesis skills.

At The Royal College of Music (KMH) and the KTH Royal Institute of Technology in Stockholm we teach sonification, sound design and sound synthesis to both music producers and engineers, which has led to the development of the new coding environments WebAudioXML [14] and iMusic [15], as well as an online sonification toolkit [16]. Our research projects are often tightly connected with the pedagogical activities of ongoing courses [17] and point out that there is a great potential for audio tools that are easy to use in order for the students to express their creativity rather than stumbling on technical challenges [18].

Many of our courses are aimed at building bridges between art and technology as we want knowledge and practices to be shared between the disciplines. While most of the students at KMH have no prior knowledge of programming, we have been challenged to find ways for them to get into coding as easy as possible. Most of our students have a basic understanding of HTML and as the web has become an increasingly important platform for sharing not only static content but also interactive online audio applications, we have decided to explore Web Audio API [19] as the platform for our experiments. We particularly appreciate that web technology is open source and that the applications we build are cross platform, online accessible and require no installations for the end user.

During our sonification classes, we have also discovered the potential for a standardized way of describing the various parameter mappings used in an application to encourage shared knowledge, flexible program architectures, open plugin structures with common libraries of sounding objects, mappings and data manipulations. We aim for solutions that make configurations and mappings explicit to encourage readability and easy assessment to avoid the "black box"-phenomenon that often can be said about sonification applications.

This study contributes to the community of sound and music computing with a proposal for a descriptive XML syntax for parameter mappings in audio applications in general and for web audio applications in particular. We have developed a working example and aim at a better un-

derstanding of the strengths and weaknesses of such a syntax.

1.1 Background

Since the presentation of WebAudioXML [14], we have supervised more than 40 student projects using the technology with promising results. WebAudioXML is a framework that uses XML to abstract an audio configuration for Web Audio API. The declarative syntax has proven to make Web Audio application development accessible for creators and opens up possibilities for students with little or no prior programming experience to turn interactive, audio application ideas into reality. We have also developed WebAudioXML Sonification Toolkit [16] – an online tool for exploring mappings between statistical data and audio parameters – that has been tested by students with no prior experience in either sonification, programming or audio synthesis. The evaluation points towards the need for the creators to access an audio configuration on a meta-level rather than having to understand and control all low-level parameters in a complex audio object.

The initial focus for WebAudioXML was to offer an XML-syntax that described audio connections and configurations. The first version supported mapping external variables to audio parameter but the solution was limited in many aspects. The mappings were defined individually for all audio parameters, only one external variable could control an audio parameter and the mapping was restricted to one pair of in-values and out-values with only one setting for interpolations. While the old version has proven to inspire creativity, it has also indicated a great potential for new features like the ones presented in this study.

1.2 Design Process

This study builds upon earlier experiences from the development and evaluation of WebAudioXML. It covers a further development of parameter mapping in the syntax and introduces a new variable object that offers a standardized way of specifying parameter mappings including scaling, quantization and conversion of values. We use three cases springing from student projects as a driving factor for the design and document the process of developing the syntax. The code is finally tested and published with online examples [1] and discussed with three technically experienced music artists.

1.3 Design Goals

WebAudioXML offers a simple syntax with a hierarchical structure that aims at shifting focus from the technological to the artistic in the making of interactive audio applications. The intention with the current study is to stay true to the same design goals while introducing more complex and flexible parameter mapping solutions. It shares the logical approach to parameter mappings with graphical environments like Pure Data, it is readable like a text based language as SuperCollider, and it uses a spreadsheet-like approach for data bindings to make mappings more explicit. There are different approaches to the use of XML: One is to use elements for encapsulating all data and another is to use elements for the hierarchical structure but store the data using attributes. In this study we want to stay true to the path set out in WebAudioXML where the audio nodes are represented by elements and their parameter values are specified using attributes. This arguably makes the code more compact and easy to write even if attributes are less flexible than elements. We are, for the continued development process, interested in testing how snippets of code and logic for parameter mapping can blend with the current hierarchical structure for audio routing in WebAudioXML. Before building a prototype we set out a few design rules to steer the development:

Platform:
The syntax shall integrate with and become a part of WebAudioXML

Language:
The syntax shall use XML to declare variables and mappings using elements and attributes

Flexibility:
The syntax shall allow for flexible mapping and conversion of values

Readability:
The syntax shall be readable and comprehensible with focus on hierarchical structures and dependencies

1.4 The spreadsheet approach

An important influence for the current project is the approach used in spreadsheet applications like Excel. We value the affordances of spreadsheet applications for non-programmers to create logic and visual representation of data. The syntax proposed in this study borrows from that approach by offering XML elements and attributes to define variables and formulas in a similar way to data and formulas in a spreadsheet application. The difference is that the result of the calculation causes an audio parameter to change rather than a visual graph to update, see Fig. 1 and Listing. 1.

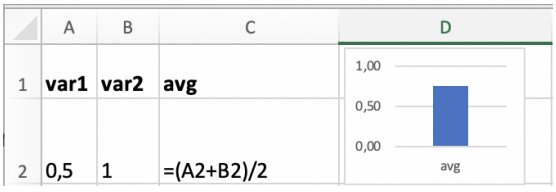

Figure 1. Spreadsheet formula and graph.

The example in Listing 1 shows how the new syntax proposed in this study uses XML elements and attributes to emulate a behaviour similar to the example from Excel in

[1] https://github.com/hanslindetorp/WebAudioXML/wiki/Parameter-Mapping

Fig. 1. The syntax is explained to a greater detail in Section 2.

```
<var name="var1" value="0.5"></var>
<var name="var2" value="1.0"></var>
<var name="avg" value="($var1+$var2)/2"></var>

<OscillatorNode>
  <frequency
    value="$avg"
    convert="Math.pow(10,x*3)">
  </frequency>
</OscillatorNode>
```

Listing 1. Code example for the Spreadsheet approach.

2. DEVELOPMENT PROCESS

Our aim with this study, in addition to the software development, is to gain more knowledge about strengths and weaknesses of the proposed syntax. We defined three cases as a starting point for the new specification and used an annotated portfolio [20] to collect insights from the design and development process. We also invited three experts from the field of sound and music computing to discuss the concept during the development process. Two participants are PhD students and one is a Postdoc, and all have extensive experience of using audio programming languages like Pure Data and SuperCollider in their artistic professions, and also in teaching the platforms at universities. They were interviewed individually and gave their consent for us to video record, analyse and use the results for research purposes. The interviews were between 30 and 60 minutes long where the participants contributed with both personal reflections and answers to prepared questions regarding clarity, strengths, weaknesses, potential, and challenges for the proposed syntax.

The three cases used as a starting point for this study spring from artistic sonification ideas formulated in our current student projects and act as a requirement specification for the syntax development. Below is a description of the three cases including the proposed syntax to solve them.

2.1 Case 1 – The Meta Knob: One-to-Many

In the first case, we identified a need for a complex audio configuration to be controlled on a meta level in a similar way as synthesizers can have a single knob for controlling multiple parameters at the same time. This makes it straightforward to build instrument plugins that can be used in sonification applications by developers without knowledge of how the instrument operates on a low-level. The first case requests variable objects anywhere in the XML-structure that can be read by multiple audio parameters on a lower level, see Fig. 2.

The <var>-element is similar to a cell in a spreadsheet-application. It can contain anything from simple data to complex formulas and mapping definitions for calculating and converting its value according to external variables (e.g. user interaction data) or other <var>-elements.

In Listing 2, the <var>-element named “param” continuously follows “$relX” which is a global variable referring

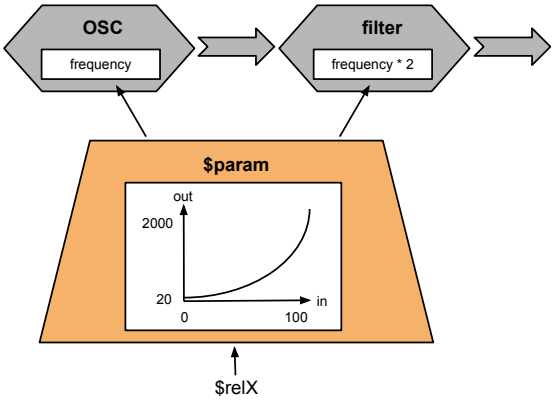

Figure 2. Systematic sketch for Case 1 with one-to-many mapping.

to the current horizontal position of the pointer. “$relX” is mapped exponentially from a value between 0 and 100 to a value between 20 and 2000 before it is stored in the variable object “$param”. Finally, the Oscillator and BiquadFilter nodes continuously update their frequency using the value of “$param” where the filter follows the same frequency value as the oscillator but one octave above.

```
<Chain>
  <var name="param"
    value="$relX"
    mapIn="0, 100"
    mapOut="20, 2000"
    curve="exp">
  </var>

  <OscillatorNode frequency="$param">
  </OscillatorNode>

  <BiquadFilterNode frequency="$param*2">
  </BiquadFilterNode>
</Chain>
```

Listing 2. Code example for Case 1 (see Fig. 2).

2.2 Case 2 – The Flexible Scale: Many-to-One

A common strategy for mapping values to frequencies is to quantize them to a musical scale, which is demonstrated in the second case. In traditional music though, the scale is rarely used without variation, and alterations typically occur depending on the direction of a phrase or the current harmony in the arrangement. The second case thus requests a solution where the state of multiple variables affect the frequency of an oscillator.

This case requires a new feature where an attribute of a <var>-element can follow the value of another <var>-element or external variable. Listing 3 illustrates how the pointer direction on the X-axis (“$dirX”) is mapped to the <var>-element “$third” causing it to have a value of either 3 or 4. This value is used in the <var>-element “$pitch” and refers to the number of semitones above the root note specified by the attribute “steps”. The variable “$pitch” is here continuously following the pointers position on the X-axis (“$relX”) and is mapped from values between 0 and

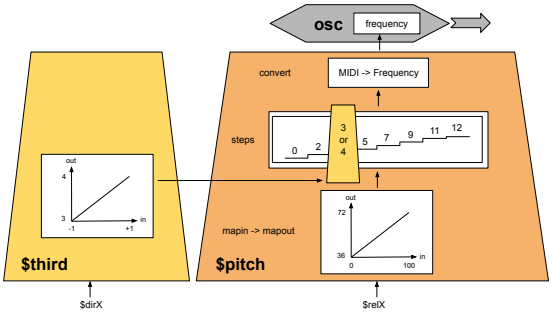

Figure 3. Systematic sketch for Case 2 with many-to-one mapping.

100 to a value between 36 to 72. After being quantized to a scale step (dynamically updated by "$dirX"), it is converted to a frequency in Hertz before the value is used to set the frequency of the OscillatorNode.

```
<var name="third" value="$dirX"
  mapin="-1, 1" mapout="3, 4">
</var>

<var name="pitch" value="$relX"
  mapin="0, 100" mapout="36, 72"
  steps="0, 2, $third, 5, 7, 8, 10, 12"
  convert="MIDI->frequency">
</var>

<OscillatorNode type="sine" frequency="$pitch">
</OscillatorNode>
```

Listing 3. Code example for Case 2 (see Fig. 3).

2.3 Case 3 – Complex Interpolation

In the third case we look for a non-limited way of defining interpolation curves for mapping an input value to a destination value. The metaphor we apply is the "automation"-feature typically found Digital Audio Workstations and graphical animation tools. Typically, this feature would map a time position to an audio or graphical parameter value using defined interpolation curves, but it could as well be used to describe any non-linear relationship between a variable and an audio parameter. To make this possible, the third case requires a syntax which supports multiple values for incoming values, outgoing values, curve shapes, steps and conversion functions.

While this case opens up for very complex mapping configurations, the addition to the WebAudioXML is fairly simple. The new feature supports multiple values for all attributes of a `<var>`-element or an audio parameter, see Listing 4 including "mapin", "mapout", "curve", "convert" and "steps". They can all be specified as one or several comma separated values with a few exceptions and restrictions: There has to be at least two "mapin"-values and typically the same number of "mapout"-values. More "mapout"-values will be ignored and fewer will be repeated. The number of "curve" and "convert"-values should be either a single or one less than the number of "mapin"-values to specify the interpolation between two values. If there are fewer "curve" or "convert"-value than

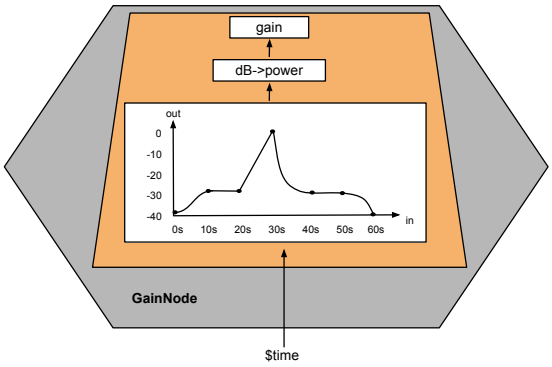

Figure 4. Systematic sketch for Case 3 with interpolation curves.

"mapin"-values, they will be repeated and used for multiple interpolation ranges.

In addition to presets such as "lin" and "exp", curves can also be specified as a mathematical expression in javascript like `Math.pow(x, 2)` where "x" is the mapped value scaled to a range of 0–1. There is also an extensive list with preset curves [2] with more intricate shapes including "easeIn", "easeInOut", "easeInOutQuad" etc. inherited from animation tools. These presets typically mimic behaviors in the physical world and can make a parameter interpolate from a linear variable in a way that e.g. accelerates in the lower part and retards in the upper. [3] While the curve-attribute affects the value before it is mapped to the mapout-value, the "convert"-attribute is the last step before the value affects the audio parameter. It can, similar to the curve-attribute take any javascript-expression to convert the value between different domains and also offers presets like "MIDI->frequency" and "dB->power".

The following code illustrates how the gain parameter can be controlled over time using different values and interpolation curves. The "$time"-variable refers to a user variable controlled from an external timer or slider using the javascript API `webAudioXML.setVariable("time", x)`.

```
<GainNode>
  <gain value="$time"
    mapin="0, 10, 20, 30, 40, 50, 60"
    mapout="-40, -30, 30, 0, -30, -30, -40"
    curve="easeInOut,lin,lin,easeOut,lin,exp"
    convert="dB->power">
  </gain>
</GainNode>
```

Listing 4. Code example for Case 3 (see Fig. 4).

3. RESULTS AND IMPLICATIONS

We present the result organized according to the design goals mentioned in Section 1.3. We also point out possible implications for further development related to each design goal.

[2] https://github.com/hanslindetorp/WebAudioXML/wiki/Parameter-Mapping

[3] https://easings.net/

3.1 Platform

The syntax shall integrate with and becoming a part of WebAudioXML.

The first proper tests of the new features was successful and proved the new syntax to be robust and straightforward to use with the WebAudioXML parser. The participants responded very positively to the potential on how Web Audio in general and WebAudioXML in particular can make audio applications easy accessible. One of them expressed that "it passes the grandma-test" when he realized that he could send someone a URL to a project and that it would run on any device without any installations. They also pointed out that the low entry style without the need for external classes makes it attractive for didactic and pedagogical purposes.

3.2 Language

The syntax shall use XML to declare variables and mappings using elements and attributes.

The first observation is that the new features merged into the previous syntax easily, while the implementation of them into the WebAudioXML-parser exposed some limitations and mistakes in the code design. The participants' response to the choice of XML for programming audio applications varied from being slightly surprised to heavily questioning. One participant objected to the use of XML for anything else but storing data and argued that the language loses its main purpose ("*the only thing XML is good for*") when the data is written using attributes with long strings rather than using the basic element tags. On the other hand, they agreed on that the hierarchical structure in XML is easy to understand as long as the configuration is limited to the constrains it provides.

The critique is interesting and raises the question about who the potential audience for a programming language is. WebAudioXML is primarily targeted towards web developers and should by design appeal more to that audience than to experts from the sound and music community. The critique regarding using XML for anything else than storing data is important and questions where the line should be drawn between storing data and logic. In this study we have explored the potential of implementing parts of the logic into the audio configuration model and to use the same language and file format for both. It promotes the building of reusable blocks in a similar way to that HTML has taken with custom elements but might be further discussed in terms of usability. The discussion also points towards a bigger question about what an application is and how it should be built. Similar to a spreadsheet application, WebAudioXML might not be the platform for a commercial products but rather offers new ways for prototyping ideas.

3.3 Flexibility

The syntax shall allow for flexible mapping and conversion of values.

All challenges from the three cases were solved and tested successfully. The new features also contributed to an update of old features like the possibility to write formulas directly into an attribute of an audio-element. The participants recognized several similarities with the proposed solution for parameter mapping to objects in other audio programming environments. UGens in SuperCollider and abstractions in PureData both offer similar functionality with inlets, outlets and a way of processing the value in-between. The experts perceived the platform as relatively limited in terms of possibilities and suggested a way of extending the format with a plugin-structure for adding any user-created audio object into the configuration.

Another idea that came up as a result of discussing flexibility was to let small blocks of data (e.g., a series of numbers specifying a musical scale) to be stored in a separate file. This would arguably make the code easier to read and the data more reusable and might work extra well for Case 3 above where the amount of data potentially could grow large. One of the participants also mentioned support for the Open Sound Control protocol [21] to make the platform even more flexible.

3.4 Readability

The syntax shall be readable and comprehensible with focus on hierarchical structures and dependencies.

The participants were in general very appreciative regarding the accessibility and the descriptive language in WebAudioXML. They pointed out that the proposed naming conventions and coding strategies were clear and easy to understand. This being said, they were more skeptical regarding the use of long strings to express complex mappings. They argued that it shared similarities with old music programming languages like CSound and that the long strings used in attributes might be a bit scary for many developers. They also asked for contextual documentation as they would expect from a coding environment. While they expressed appreciation for the clear, hierarchical structure and relations between objects, parameters, and variables, they also pointed out the potential problem when audio signals and parameter mappings are set to break those structures and suggested a graphical interface to get a better overview.

We find the discussion about a graphical interface especially intriguing; it is easy to lose flexibility in favor for the usability a graphical interface can offer. One possible compromise that both stays true to the text-based concept of XML and offers contextual guidance in some XML editors is the use of a well-designed XSD-file. [4] One feature that was discussed by all participants was case 3 and the complex relation between multiple input and output values. Even if they thought the syntax was clear, all of them mentioned that the feature was hard to visualize without a graphical tool available.

[4] `https://www.w3.org/XML/Schema`

3.5 Other observations

The annotated portfolio from the development process also contributes to important insights. It was obvious that the addition of the new features to the parser challenged several design decisions made in the beginning of the WebAudioXML development. Some of them were updated during this process but there is still a need for a few new class definitions. The parser also contains some non-recommended solutions like `eval(expression)` that work well for a prototype but is a potential security risk if put into production. Furthermore, the "value"-attribute replaced the old "follow"-attribute. Its approach with any type of expression from a single target variable to a complex mathematical expression containing multiple variables proved to be a more flexible way to receive data.

One feature we struggled with during the implementation was the sequence of the mapping steps inside a variable object. Even if the attributes themselves can be defined in any order in the XML-element, we decided to make them always operate in the following order, and if any attribute is missing, that step will be bypassed.

```
value->mapin->curve->mapout->steps->convert
```

We also got new ideas for further development. One is to implement the structure used in case 2 for all attributes in a variable object, including mapout, curve and values in a convert expression. Another is to add built-in features for statistical evaluation in a variable and map the data according to the result. This could e.g. make it easy to send peak values to an audio parameter when the incoming data is near the confidence limits of a normally distributed dataset.

Finally, we think that it would be a great addition if the variable object could have built-in support for derivative and second order derivative which would be useful for applications aiming for physical interaction.

4. CONCLUSION AND FUTURE DIRECTION

The proposed syntax for audio parameter mapping has been designed, discussed and implemented into the latest version of WebAudioXML. The participating experts pointed at several promising features of the development. First, they confirm that the proposed solutions match the design goals that were set out. Second, they confirm the didactic and pedagogical value and potential of the WebAudioXML framework. Third, we noticed that attitudes towards XML would influence their immediate understanding, both positively and negatively. It is quite clear that the experts don't regard themselves as primary users of the framework, which is both expected and intentional. As such, the ambition of creating this system and its scope have been met.

We were particularly encouraged by the participants to be consequent in naming variables and to find ways for contextual help with syntax explanation and suggestions. The suggested need for and solution to case 3 was generally more difficult for them to grasp than cases 1 and 2, and would require a graphical interface to become useful. It was stated that syntax and concepts borrowed from other languages were easy to understand and that the concept of external files with top-level parameters exposed to the parent object was an expected feature relating to object oriented programming in general and "abstractions" in Pure Data in particular.

Forthcoming studies involving students in our courses will target parameter mappings in particular. Then, both the pedagogical potential and the syntax itself will be systematically evaluated with regards to what the students choose to create and how they will use WebAudioXML to accomplish their artistic goals.

Acknowledgments

Thanks to the participants for contributing with highly valued insights and feedback.

5. REFERENCES

[1] B. J. Grond F., "Chapter 15: Parameter mapping sonification," in *The Sonification Handbook*, 2011, p. 366.

[2] M. A. Nees and B. N. Walker, "Listener, task, and auditory graph: Toward a conceptual model of auditory graph comprehension," in *International Conference on Auditory Display*. Georgia Institute of Technology, 2007.

[3] J. G. Neuhoff, J. Wayand, and G. Kramer, "Pitch and loudness interact in auditory displays: Can the data get lost in the map?" *Journal of Experimental Psychology: Applied*, vol. 8, no. 1, p. 17, 2002.

[4] J. Rohrhuber, "S–introducing sonification variables," in *In Proceedings of the Supercollider Symposium*, 2010.

[5] B. Vercoe, *Csound*, 1985 (accessed May 28, 2021). [Online]. Available: https://csound.com/

[6] M. Puckette, "Max at seventeen," *Computer Music Journal*, vol. 26, no. 4, pp. 31–43, 2002.

[7] M. S. Puckette, "Pure data," in *Proceedings: International Computer Music Conference 1997, Thessaloniki, Hellas, 25-30 september 1997*. The International Computer Music Association, 1997, pp. 224–227.

[8] J. McCartney, "Rethinking the computer music language: Supercollider," *Computer Music Journal*, vol. 26, no. 4, pp. 61–68, 2002.

[9] G. Wang, P. R. Cook *et al.*, "Chuck: A concurrent, on-the-fly, audio programming language," in *ICMC*, 2003.

[10] R. M. Candey, A. M. Schertenleib, and W. Diaz Merced, "Xsonify sonification tool for space physics," in *International Conference on Auditory Display*. Georgia Institute of Technology, 2006.

[11] S. Pauletto and A. Hunt, "A toolkit for interactive sonification," in *International Conference on Auditory Display*. Georgia Institute of Technology, 2004.

[12] A. Schoon and F. Dombois, "Sonification in music," in *International Conference on Auditory Display*. Georgia Institute of Technology, 2009.

[13] D. Worrall, M. Bylstra, S. Barrass, and R. Dean, "Sonipy: The design of an extendable software framework for sonification research and auditory display," in *Proc. ICAD*, 2007.

[14] H. Lindetorp and K. Falkenberg, "WebAudioXML: Proposing a new standard for structuring web audio," in *Sound and Music Computing Conference*. Zenodo, 2020 (accessed May 28, 2021), pp. 25–31, qC 20200722. [Online]. Available: https://zenodo.org/record/3898655#.X3HgbC0zLa4

[15] H. Lindetorp, *iMusic: JavaScript framework for interactive music*, 2016 (accessed May 28, 2021). [Online]. Available: https://github.com/hanslindetorp/imusic

[16] ——, *WebAudioXML Sonification Toolkit*, 2020 (accessed May 28, 2021). [Online]. Available: https://github.com/hanslindetorp/SonificationToolkit

[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 first Nordic SMC*. Zenodo, 2019, pp. 36–41.

[18] H. Lindetorp, "Immersive and interactive music for everyone," in *Proceedings of the Nordic Sound and Music Computing Conference 2019 (NSMC2019) and the Interactive Sonification Workshop 2019 (ISON2019) :*, 2019, pp. 16–20. [Online]. Available: http://smcsweden.se/proceedings/NordicSMC_ISon_2019_Proceedings.pdf

[19] P. Adenot and R. Toy, *Web Audio API: W3C Candidate Recommendation, 18 September 2018*, 2018 (accessed May 28, 2021). [Online]. Available: https://www.w3.org/TR/2018/CR-webaudio-20180918/

[20] B. Gaver and J. Bowers, "Annotated portfolios," *interactions*, vol. 19, no. 4, pp. 40–49, 2012.

[21] A. Freed, "Open sound control: A new protocol for communicating with sound synthesizers," in *International Computer Music Conference (ICMC)*, 1997.