# Intelligent audio plugin framework for the Web Audio API

Nicholas Jillings, Yonghao Wang, Ryan Stables
Digital Media Technology Lab,
Birmingham City University
{nicholas.jillings, yonghao.wang, ryan.stables}@bcu.ac.uk

Joshua D. Reiss
Centre for Digital Music,
Queen Mary University of London
joshua.reiss@eecs.qmul.ac.uk

## ABSTRACT

The Web Audio API introduced native audio processing into web browsers. Audio plugin standards have been created for developers to create audio-rich processors and deploy them into media rich websites. It is critical these standards support flexible designs with clear host-plugin interaction to ease integration and avoid non-standard plugins. Intelligent features should be embedded into standards to help develop next-generation interfaces and designs. This paper presents a discussion on audio plugins in the web audio API, how they should behave and leverage web technologies with an overview of current standards.

## 1. INTRODUCTION

The Web Audio API[1] defines several low-level processing blocks, such as gain and filter nodes. These can be linked together to create highly complex processing graphs for real-time rendering with native language performance. Intelligent audio production systems enable computers to make creative decisions based upon semantic cues and the production environment. These have been in development since 1975 with automatic microphone mixing for conferences [4]. More recent work covers automated [7, 6] and semantic [13, 14] audio processors, known as plugins. Plugins can recommend parameters based on descriptions [14] or user profiles [1]. Intelligent effects are aided by web ontologies [5] to understand the relationships of the information. The web does not have a standard to facilitate intelligent processing of audio streams. This paper analyses existing audio processor frameworks and presents a flexible web-based audio plugin format.

## 2. BACKGROUND

Native audio plugins, such as RTAS or VST, cannot exist in the DAW as browsers will not execute user binary code for security. Therefore several frameworks have been developed to build audio processing chains in the web. Tuna[2] expands the number of base processing nodes by creating Web Audio-like objects. These have limited functionality but behave as traditional web audio nodes. Web Audio API Extension (WAAX) [3] and Web Audio Modules (WAM) [11] both define methods for builidng graphical user interfaces. However neither define the host implementations, limiting the design scope to traditional audio effects. The Web Audio Modules build javascript process units rather than leveraging native nodes, making them less efficient[2]. JavaScript Audio Plugin (JSAP) [9] defines the host interface and provides a wrapper for deploying prototype chains with a parameter interface similar to the Web Audio API.

Cross-adaptive effects are one method for building automatic systems [10, 12], see fig. 1, taking audio from other channels to compute parameters based on the extracted audio features.

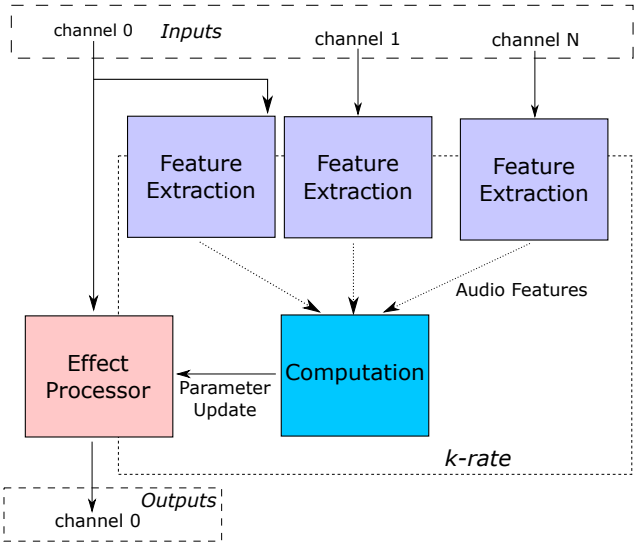


**Figure 1: Structure of a Cross-Adaptive Effect.**

## 3. FRAMEWORK

Audio plugins are self-contained environments processing discrete audio frames with a host interface. The host serves audio frames and handles the lower-rate communications for parameter controls, playback events and user interfacing, see fig. 3. A plugin in the browser should behave the same way and therefore is important for the host to be defined in the plugin standard.

Plugin instances must investigate their environment, such

[1]https://www.w3.org/TR/webaudio/
[2]https://github.com/Theodeus/tuna



*Web Audio Conference WAC-2017, August 21–23, 2017, London, UK.*

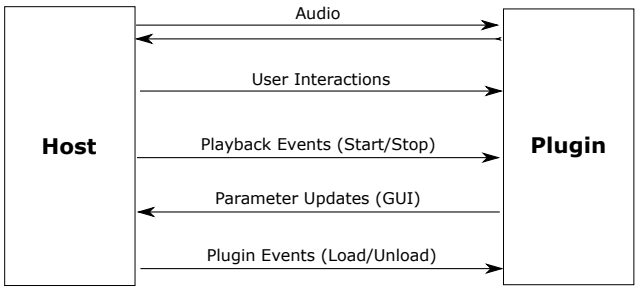


**Figure 2: Layout of Host and Plugin interactions**

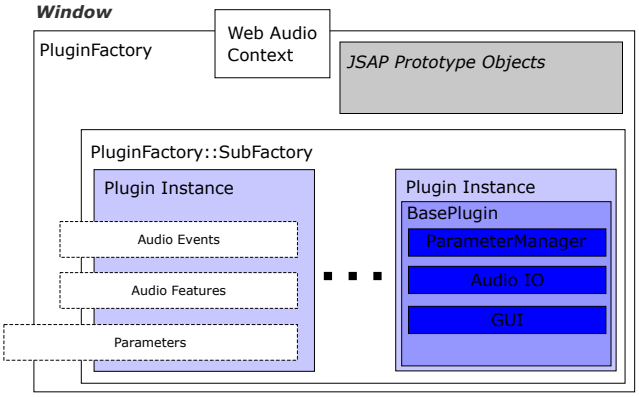


**Figure 3: Structure of JSAP, with the host (`PluginFactory` and `SubFactory`) to the plugin instances, with interfaces and inherited objects.**

as the track it is operating upon and the device being used, to be intelligent. If this information is not available through the host interface the plugin may access these in an undefined method, creating more work for site-managers to adapt for non-standard implementations.

Fig. 3 shows the JSAP project scope. The host is the `PluginFactory` and holds every plugin instance. Each plugin communicates through the factory to get any semantic information. The page gives this to the factory in a standardised way. Plugin parameters are exposed globally but the audio events and features are managed by the factory. These interactions are *k*-rate (see fig. 1) since they operate on blocks of audio not samples.

The SAFE plugins [14] utilise audio features and semantic context to derive parameter controls. These have been re-made into JSAP instances and available online[3]. Standard audio effects have been developed exploring the more complex effects, including feedback delays, equalisers and auto-adaptive effects.

## 4. INTELLIGENT PROCESSING

Cross-adaptive effects use a set of rules[12, 15] to map the features into meaningful parameter control signals. Ontologies store the semantic descriptors and session information to help define these rules. Plugins must therefore have access to the session semantic information and have feature extraction libraries to make correct and meaningful decisions. Without access to this information the plugin may have to infer this information from the audio or non-standard iteractions. At worst, plugins may not be intelligent and simply adaptive to its own audio stream. The rules can come from studying texts and experts or from listening tests.

JSAP provides feature extraction using JS-xtract [8] and the `PluginFactory` manages the routing of features between plugins, saving developer time and computational resources to build intelligent effects. Any plugin output can be the source for another plugins' cross-adaptive input.

[3]http://www.semantic.audio.co.uk/jsap

## 5. CONCLUSION

This paper presents the layout of an intelligent plugin framework. A suitable framework must provide clear control pathways between host and plugin instances to ease deployment and development. The plugins must have access to the session to be able to make creative mixing decisions intelligently. Plugins can perform sample-level processing but these are inefficient in JavaScript compared to native implementations which should be preferred. Plugins are also bound to the supported interactions of the web environment and strict communication protocols, whilst desktop programs can utilise a wider communication platform.

## 6. REFERENCES

[1] M. Cartwright, B. Pardo, and J. D. Reiss. Mixploration: Rethinking the audio mixer interface. In *19th Intl. Conf. on Intelligent User Interfaces*, pages 365–370. ACM, 2014.

[2] A. Charland and B. Leroux. Mobile application development: Web vs. native. *Commun. ACM*, 54(5):49–53, May 2011.

[3] H. Choi and J. Berger. WAAX: Web audio api extension. In *New Interfaces for Musical Expression*, pages 499–502, 2013.

[4] D. Dugan. Automatic microphone mixing. *J. Audio Eng. Soc*, 23(6):442–449, 1975.

[5] G. Fazekas and M. B. Sandler. The studio ontology framework. In *12th Intl. Soc. Music Inf. Retr.*, pages 471–476, Oct. 2011.

[6] D. Giannoulis, M. Massberg, and J. D. Reiss. Parameter automation in a dynamic range compressor. *J. Audio Eng. Soc*, 61(10):716–726, 2013.

[7] S. Hafezi and J. D. Reiss. Autonomous multitrack equalization based on masking reduction. *J. Audio Eng. Soc*, 63(5):312–323, 2015.

[8] N. Jillings, J. Bullock, and R. Stables. JS-xtract: A realtime audio feature extraction library for the web. In *17th Intl. Soc. Music Inf. Retr.*, 2016.

[9] N. Jillings, Y. Wang, J. D. Reiss, and R. Stables. JSAP: A plugin standard for the web audio api with intelligent functionality. In *Audio Engineering Society Convention 141*, 2016.

[10] I. Jordal, Ø. Brandtsegg, and G. Tufte. Evolving neural networks for cross-adaptive audio effects. In *Proceedings of the 2nd AES Workshop on Intelligent Music Production*, volume 13, 2016.

[11] J. Kleimola and O. Larkin. Web audio modules. *Sound and Music Computing*, 2015.

[12] J. D. Reiss. Intelligent systems for mixing multichannel audio. In *17th Intl. Conf. on Digital Signal Processing (DSP)*, pages 1–6, July 2011.

[13] P. Seetharaman and B. Pardo. Audealize: Crowdsourced audio production tools. *Journal of the Audio Engineering Society*, 64(9):683–695, 2016.

[14] R. Stables, S. Enderby, B. De Man, G. Fazekas, and J. D. Reiss. Safe: A system for the extraction and retrieval of semantic audio descriptors. In *15th Intl. Soc. Music Inf. Retr. (ISMIR)*, 2014.

[15] A. Wilson and B. Fazenda. Variation in multitrack mixes: analysis of low-level audio signal features. *Journal of the Audio Engineering Society*, 64(7/8):466–473, 2016.