An Intelligent audio workstation in the browser

Nicholas Jillings
Digital Media Technology Lab,
Birmingham City University
Birmingham, B4 7XG
nicholas.jillings@bcu.ac.uk

Ryan Stables
Digital Media Technology Lab,
Birmingham City University
Birmingham, B4 7XG
ryan.stables@bcu.ac.uk

ABSTRACT

Music production is a complex process requiring skill and time to undertake. The industry has undergone a digital revolution, but unlike other industries the process has not changed. However, intelligent systems, using the semantic web and signal processing, can reduce this complexity by making certain decisions for the user with minimal interaction, saving both time and investment on the engineers' part. This paper will outline an intelligent Digital Audio Workstation (DAW) designed for use in the browser. It outlines the architecture of the DAW with its audio engine (built on the Web Audio API), using AngularJS for the user interface and a relational database.

1. INTRODUCTION

Audio production is predominantly performed inside a Digital Audio Workstation (DAW), running on a computer. Historically, this was limited to high-end recording studios, but over time have been integrated into the home studio. Despite the digital revolution, where music production has moved away from bulky hardware onto software-defined mixing environments, the actual process of mixing has not changed. In intelligent production systems, audio parameters and decisions are controlled by autonomous processes, rather than directly by the mix engineer. This should reduce the burden of production into a computer, lowering the knowledge required by an end user. To achieve this, intelligent systems require a knowledge-base to drive the decision making process [4]. These can be stored in a knowledge based data store, such as an ontology or RDF. Ontologies exist for describing the music production environment [6, 14] allowing a system to understand a device or process.

This paper will outline the structure of an intelligent browser-based DAW, along with the audio, database, and user interface interactions. The paper will then outline the unique aspects and opportunities a browser-based solution can provide for audio research and analysis. The paper concludes with new results from a mixing study [8] using the software.

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

2. ARCHITECTURE

A Digital Audio Workstation is built around three core concepts: the audio engine, the user interface (UI) and the data-store. The audio engine expands the web audio API functionality into logical DAW blocks, such as tracks and audio regions. The UI interprets the audio engine and presents this information to the user. The interface is built using AngularJS[1] to supply a modular data-view of the audio engine model. Behind all of this is a relational database to store the mixing session data, user interactions and audio file data. A DAW can be represented as a Model-View-Presenter workflow. The model is the audio engine and relational database. The database only communicates with the engine to provide control information and receive interaction data. The user interface is the view, controlled by AngularJS. As with all web products, the Presenter is the browser, which may behave differently between brands, models and devices.

2.1 Audio Engine

The audio engine is the core of any audio software, where each N-sample block of audio is passed through a predefined network of processors. An audio engine for a DAW must create several discrete modules to link together, such as the track, audio buses, master bus, plugins, sends and audio regions. The Web audio API facilitates the development of a modular audio engine. Figure 1 shows this in a track object, with send points, JSAP SubFactory [9] and output stages. The Web Audio API gain node can be used extensively as a computationally inexpensive summation point. The API sums all inputs together on arrival into a node and all the browsers optimize out any gain nodes with the gain value at unity or zero.[2].

2.2 User Interface

Having a well-defined model simplifies the view creation process. The user interface is built using AnuglarJS, which facilitates the building of dynamic HTML pages. Each view element is bound onto the model to provide an interaction and feedback. For instance the solo and mute buttons provide user feedback on the current state whilst also giving a simple interaction to toggle the state. A DAW traditionally has two views, the timeline and mixer. To reduce the potential training / adjustment time for participants the presented browser DAW retains these views. Both views interact with the same underlying model but are rendered with different controls. The timeline view, shown in Figure 2, places audio

[1] https://angularjs.org/
[2] https://padenot.github.io/web-audio-perf/

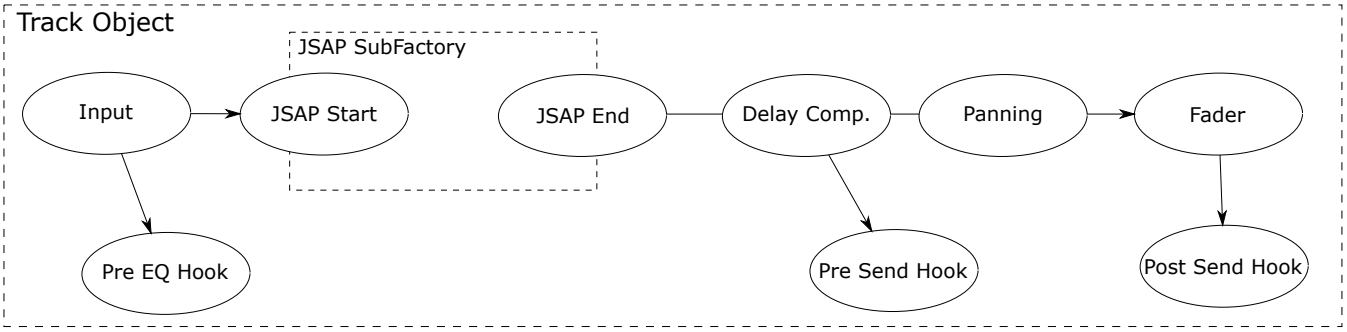

Figure 1: Structure of a track object in the DAW built using the Web Audio API.

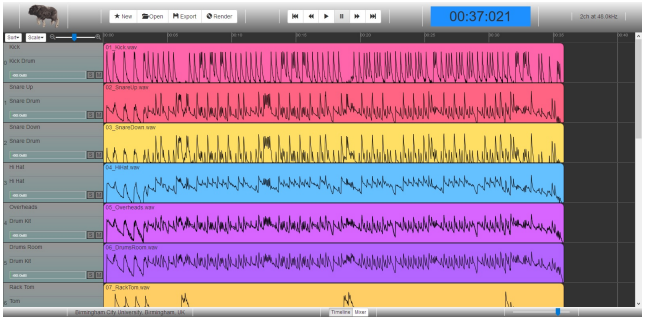

Figure 2: The DAW presented in Firefox 51, showing the Timeline view

regions onto a movable background, delineated with timestamps indicating when regions will play. However this view removes the channel-based controls such as volume and panning. The mixer view gives all of the controls but the user looses the timeline view of regions.

An important mechanism in music production is UI feedback, through updating interface elements. This includes track meters, session clocks, and moving play-heads. These elements are all triggered by using the `requestAnimationFrame` from the HTML Living standard[3]. This adds the passed function into an animation queue which is executed at the next rendered animation frame. Executing a function and updating on an arbitrary time-frame will either update too frequently, wasting resources, or too sparse causing jitter and missed-frames. By processing these interactions on animation requests, the interface is kept smooth and dynamically scales the amount of track data.

3. TIMING AND EVENTS

Timing for Web Audio has been studied before in JavaScript [5, 12]. However, in a DAW environment, there is an increased amount of timing requirements than just audio scheduling. The user interface needs to be updated frequently to redraw the session clock, play-head bar and track meters. These tasks are vital to both the process and the user experience.

The DAW needs to schedule several events including audio playback and automation. However it also needs to handle partial playback of audio regions as well as future playback scheduling. Starting and stopping the play-head is a major event in a DAW, requiring canceling playback of buffers, resetting the play-head to the start, signaling plugins to clean up any temporary information and rescheduling any automations. Pausing and resuming of playback when mixing can be handled by suspending and resuming the audio context. For rendering, audio tails can be preserved. Scheduling of playback is also simplified as the positioning of buffers can be evaluated before they arrive, removing the need for lookahead approaches.

The Web Audio API is powerful for these events, with both scheduling and cancel agents on parameters and buffers, along with linear and exponential ramping. However outside the API, it is difficult to interact with. For example, there is no clear indication of an upper limit of the distance to schedule, nor is there any way to collect the interim data which is needed for user interaction. To update the user interface with indication of the current automation state, the interface itself needs to calculate the automation at a given time, such as fader animation on automation reading Even if all automation techniques are just linear points, this is still a significant programming overhead that could be avoided by the web audio API.

Timing is also critical in parallel worker scripts. Some events, such as the track volume meters, require analysis of the audio frame. To keep the main JavaScript thread clear, and to improve the user experience, the audio processing for meters in the DAW is sent to Web Worker[4] threads, running the JS-Xtract [7] feature extraction library. This allows us to return the requested audio features such as the RMS Amplitude and the spectral centroid by default. These workers operate asynchronously and call a passed callback function on returning data. To not block the drawing of meters, the meter is only updated when new data is returned. However, as more tracks are created in the DAW, more pressure is imposed on the web workers. The impact of running multiple instances of active workers has an undefined impact on browsers [13], with some browser / platform combinations performing significantly better than others.

4. INTELLIGENCE

The DAW is intended to be used as a deployment platform for intelligent tools such as automatic mixing, cross-adaptive audio effects [10] and semantic audio processing [11]. However, we also use the DAW to gather large datasets of music production data. As more participants use the DAW for

[3]https://html.spec.whatwg.org/multipage/webappapis.html

[4]https://html.spec.whatwg.org/multipage/#toc-workers

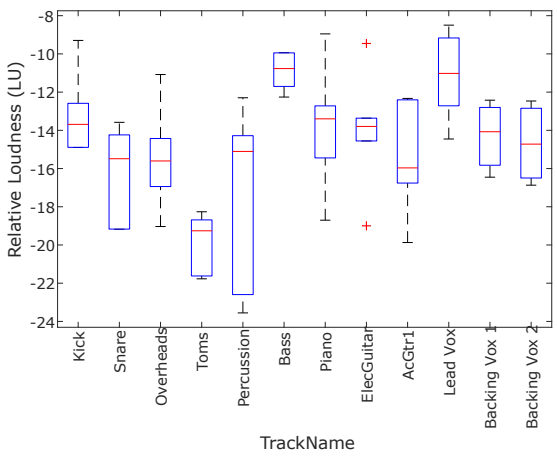

Figure 3: Loudness of the track relative to the final mix

their own projects, we are able to create production rules that can inform the next generations of intelligent music production. An on-line DAW is unique iteration of a traditional systems, as it is permanently connected to a relational database. This enables continuous, passive data collection of all user interactions over the web[1]. Auditory experiments on the web have been used before and can be as accurate as laboratory conditions so long as adequate selection of participants takes place [2].

In an early experiment, we were able to use the DAW to discover trends in users producing balance mixes [8]. The results showed that performing a mix on an excerpt of a song takes 100-150 actions over 8-16 minutes of work (an interaction every 4-10 seconds). Figure 3 shows that for some tracks there is a high degree of agreement in the loudness that they should have in the mix (Bass, Electric guitar and Overheads). However, for other tracks there is a very large variance, indicating these tracks may have more influence on mixing taste, rather than a fundamental track (Percussion, Snare and Piano). Overall, vocals were generally quite high in the mix, although not as high as previous studies indicate [3].

4.1 User interactions and grouping

Table 1 presents additional analysis of the data presented in [8]. Here, we show the amount of user actions on a specific parameter of tracks in the DAW.

The highest proportion of user interactions with the UI was for on auditioning the audio. 43.8% of all parameter movements modified the volume and pan positions. This indicates the vast majority of user interaction is from active mixing. Interestingly, 11.465% of interactions involved the use of solo button (enabling and disabling solo's), but only 2.559% were mute actions. Similarly, only a very small number of interactions involved changing the track name from the default "*Track N*" label.

Figures 4 and 5 show the results of applying hierarchical clustering to the track groupings, to identify similarities in the way instruments are combined in the mix. Figure 4 operates over the track names for the song Queen's Light. There were 13 entries in total for this song, and three prominent groups of tracks appear which can be attributed to the Drums (cyan), percussive (red) and synthesisers (green) of the mix. The bass guitar is mostly independent and has a high distance from the other instruments in the mix. Figure 5 operates over all of the sessions and looks at the instrument name, not track name, where there are 4 main clusters. Again the drums (cyan), then guitars / sustaining instruments (red). The green is split into two groups, the upper set are vocal or vocal-like instruments and the lower is percussion-based instruments.

Action	Number	%
Play/Pause/Stop	1,318	33.066%
Set Volume	1,247	31.284%
Set Pan position	500	12.543%
Toggle Solo	457	11.465%
Set Track Group	221	5.544%
Toggle Mute	102	2.559%
Add Bus	67	1.681%
Add Send to track	66	1.655%
Alter Track Name	8	0.201%

Table 1: Count of all action types when creating a balance mix.

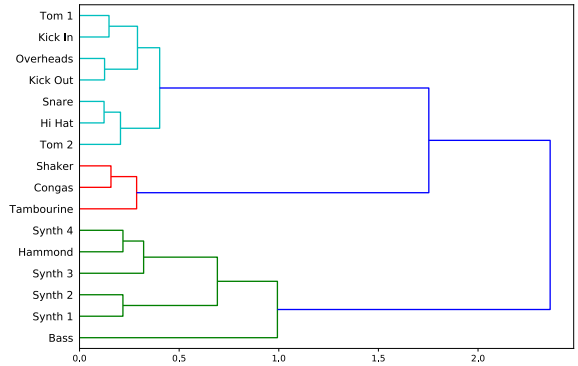

Figure 4: Hierarchical clustering of track groupings in *Queen's Light* by track name.

5. CONCLUSION

In this paper we have presented our intelligent Digital Audio Workstation developed in the browser using the Web Audio API. We have shown how using the audio engine as a model facilitates the representation the audio structure through multiple client views. We have also shown where shortfalls of the current Web Audio API may cause limitations in such a user interaction intensive environment.

An online platform like the one presented here is useful for both the deployment of intelligent systems and for the collection of data, both actively and passively. The online DAW captures session information in far greater detail, enabling behavior analysis and extracting the context around decision making processes by the engineer. This is shown in our experiments, where we can identify trends in grouping and mixing data.

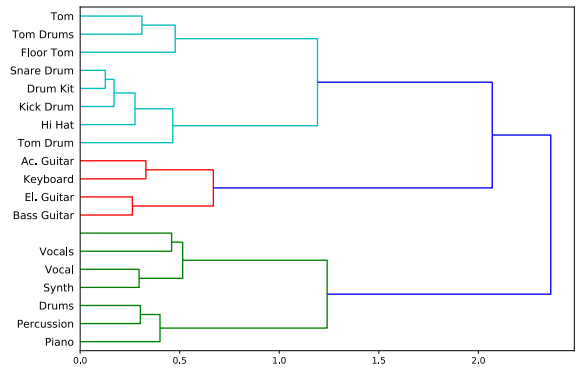

Figure 5: Hierarchical clustering of all sessions by the instrument name.

5.1 Future Work

The online DAW can be operated as a traditional mixing interface or can be modified for targeted experiments. The balance experiments in [8] use a restricted environment where JSAP plugins were not available, nor any editing of the audio regions. Likewise other restrictions may include removing track names, applying restrictions to creating/deleting tracks or buses, banning solo/mute, disabling volume/pan controls, amongst others. This allows the DAW to be used in more bespoke ways for specific data collection tasks. Equally, more subverted tests may include changing the information of parameters from RMS to other audio features, or not displaying a waveform but some other waveguide in the audio regions, such as spectral centroid.

The DAW can also gather off-line information from audio files, including the audio features. Combined with semantic information from the session (such as genre and BPM) and the user-labeled track names and instruments, a database of known instrument features can be created. This can be fed into a machine learning algorithm to build a classifier which is user-corrected.

6. REFERENCES

[1] M. H. Birnbaum. Human research and data collection via the internet. *Annual Review of Psychology*, 55:803–832, October 2003.

[2] M. Cartwright, B. Pardo, G. J. Mysore, and M. Hoffman. Fast and easy crowdsourced perceptual audio evaluation. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 619–623, March 2016.

[3] B. De Man, B. Leonard, R. King, and J. D. Reiss. An analysis and evaluation of audio features for multitrack music mixtures. In *15th International Society for Music Information Retrieval Conference (ISMIR 2014)*, October 2014.

[4] B. De Man and J. D. Reiss. A semantic approach to autonomous mixing. *Journal of the Art of Record Production*, 8, December 2013.

[5] B. Dias, H. S. Pinto, and D. M. Matos. BPMtimeline: Javascript tempo functions and time mappings using an analytical solution. In *Proceedings of the 2nd Web Audio Conference*. Georgia Institute of Technology, April 2016.

[6] G. Fazekas and M. B. Sandler. The studio ontology framework. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011)*, pages 471–476, October 2011.

[7] N. Jillings, J. Bullock, and R. Stables. Js-xtract: A realtime audio feature extraction library for the web. In *International Society for Music Information Retrieval Conference*, 2016.

[8] N. Jillings and R. Stables. Investigating music production using a semantically powered digital audio workstation in the browser. In *Audio Engineering Society Conference: 2017 AES International Conference on Semantic Audio*, Erlangen, Germany, June 2017. Audio Engineering Society.

[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*. Audio Engineering Society, 2016.

[10] E. Perez-Gonzalez and J. Reiss. Automatic equalization of multichannel audio using cross-adaptive methods. In *Audio Engineering Society Convention 127*. Audio Engineering Society, 2009.

[11] R. Stables, S. Enderby, B. De Man, G. Fazekas, and J. Reiss. Safe: A system for the extraction and retrieval of semantic audio descriptors. In *15th International Society for Music Information Retrieval Conference (ISMIR 2014)*, 2014.

[12] J. Sullivan. Alternatives to lookahead audio scheduling. In *Proceedings of the 2nd Web Audio Conference*. Georgia Institute of Technology, April 2016.

[13] J. Verdú, J. J. Costa, and A. Pajuelo. Dynamic web worker pool management for highly parallel javascript web applications. *Concurrency and Computation: Practice and Experience*, 28(13):3525–3539, September 2015.

[14] T. Wilmering, G. Fazekas, and M. B. Sandler. The audio effects ontology. In *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR 2013)*, pages 215–220, November 2013.