Interactive Audio Web Development Workflow

Lonce Wyse
Communications and New Media
National University of Singapore
lonce.wyse@nus.edu.sg

ABSTRACT

New low-level sound synthesis capabilities have recently become available in Web browsers. However, there is a considerable gap between the enabling technology for interactive audio and its wide-spread adoption in Web media content. We identify several areas where technologies are necessary to support the various stages of development and deployment, describe systems we have developed to address those needs, and show how they work together within a specific Web content development scenario.

Categories and Subject Descriptors

H.5.5 [**Information Systems**]: Information Interfaces and Presentation (HCI) – *sound and music computing.*

General Terms

Design; Human Factors

Keywords

Interactive Audio; W3C Web Audio API; Development tools; Web development workflow.

1. INTRODUCTION

Interactive sound has recently been gaining attention in large part due to new standards established by the World Wide Web Consortium (W3C) and implemented on most major browsers running on personal computers and mobile devices. The Web Audio API is comprised of a set of low-level audio "nodes" that can be connected to form signal flow audio graphs for real-time synthesis and processing. The API also defines parameters for influencing the behavior of the algorithms as they are executing so that synthesis can be responsive in real-time to data generated from sources such as graphical or user behavior.

In this paper, we describe three key components of a software ecosystem that we view as critical to the workflow of developers in order for the new interactive audio capabilities to work their way in to the everyday experience of Web media for the masses. The first is a library that supports the construction of audio graphs with rich and complex sonic behaviors which is designed for a specialist developer community that have a professional level of both audio and programming skills. The second is a standard API for interactive sound "models" that provides a simple and powerful means for controlling sounds from applications designed to be used by Web developers who are not audio specialists. The third is a way of organizing and accessing sound models in a database served from "the cloud" which minimizes complexity for Web developers and provides broad and efficient distribution for sound model developers. We describe strategies and implementations for each of these three components, and then demonstrate their utility in the context of a typical Web development workflow.

MM'14, November 3–7, 2014, Orlando, Florida, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3063-3/14/11…$15.00.
http://dx.doi.org/10.1145/2647868.2655064

2. SOUND MODELS

We use the term "sound model" (or simply "model") to refer to parameterized algorithms for generating a class of sounds that run in real time. They can be as trivial as a single oscillator or as rich as an algorithmic piece of music, but typically they represent objects (doors, car engines, bells, birds, musical instruments), events (rolling, scraping, bouncing), musical processes (beat patterns, melody generators), or abstract sound-generating algorithms and equations (chaotic dynamical systems)[13]. In popular graphical programming languages for interactive sound such as Max/MSP [14] or PD[9], sound models are known as "patches" in reference to the pre-digital technique of constructing sounds by patching analogue sound generating and processing modules together with cables. We use the term "sound" to refer to a specific instance of an acoustic wave generated naturally or synthetically that might be recorded and stored as a digital audio signal file. Thus a sound model generates a class of sounds that change depending on the parameter values that influence the synthesis algorithm.

The recent Web Audio API[1] breaks new ground for sound synthesis in W3C standards-compliant browsers. Previous to the implementation of the new standard, audio elements (e.g. recorded samples stored in files) could only be triggered, or else could be synthesized in real-time only with browser plug-ins presenting installation obstacles to users, security risks, and non-native performance.

With the Web Audio API, developers use JavaScript to connect audioNodes together into signal flow graphs that are managed by an audioContext to generate sound in real time. There are very few audioNodes available compared to the number of objects in Max/MSP [14] which has in the neighborhood of one thousand objects without counting those available from the community of users. Nonetheless, some rich synthesis models can be constructed, controlled with sample accuracy, and manipulated in real time with the limited set of audioNodes already supported in the Web Audio API. There are also audioNodes that perform convolution and compute power spectra. Since all of the audioNodes are built in to the browser, they are precompiled and run at native speeds. They are also given their own thread, separate from the browser UI thread, so that audio can reliably deliver its stream of audio samples without glitches due to blocking from user and graphical activity in the browser (See [11] for more details about the capabilities and limitations of the current Web Audio API).

The developers who will be able to make use of the full potential of the Web Audio API are skilled programmers with audio knowledge, not typical Web developers. Even for audio specialists, the Web Audio API is fairly "low level." For this reason, we have developed the jsaSound library. jsaSound is a JavaScript library designed to support common sound model-building operations such as generating rhythmic patterns and event loops, encapsulating models so that they behave like audioNodes to be combined in more complex graphs[10], and for hiding complexity when managing resources such as microphone input or audio files. WAAX[4] is another example of a library designed for simplifying sound model development with the Web Audio API.

A key feature of the jsaSound library is support for creating and exposing a consistent interface for users of sound models.

3. A SOUND MODEL API

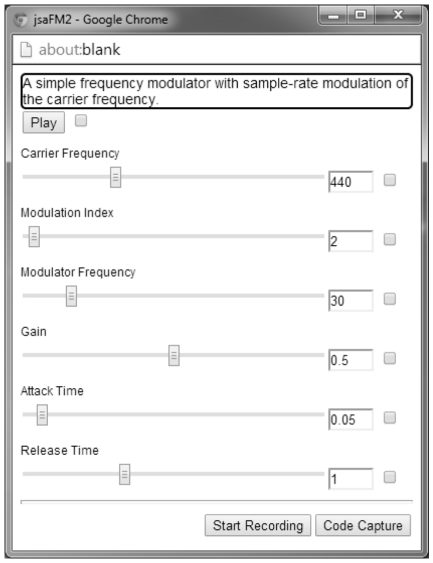
Sound model parameters can vary greatly in terms of how they affect a model, their natural units, and their names which ideally reflect their function. For example, a “car engine” might expose parameters for RPM, torque, and horsepower, while a “footsteps” model might expose parameters for kilometers per hour, and gait (running or walking)[5]. Similar to the Java plug-in synthesis system ASound[12], jsaSound standardizes the parameter interface by using the parameter name or index as an argument and providing just two “setters”, setParam(id, value) and setParamNorm(id, value). The former uses the natural units of the parameter, and the latter takes values in [0,1] and maps them to range defined for the natural units.. setParam (id, value) is also used for string parameters such as file names. Thus the interface for controlling interactive sounds is simply comprised of:

- play()
- release()
- setParam()
- setParamNorm()

Three additional methods are available for getting information about the sound model:

- getParam(id, ["name", "type", "val", "normval", "min", "max"])
- getNumParams() – the number of parameters a sound exposes,
- getAboutText() –developer-supplied text descriptions

The simplicity of the scheme makes real-time interactive control of sounds only incrementally more involved than playing static sound files. The interface allows a Web developer to control sounds “generically” (without needing specific knowledge about individual sound models). It also enables for example, the automatically-generated slider-box graphical interface "player" provided with jsaSound (Figure 1). It facilitates sound model reuse and sharing otherwise impractical with raw heterogeneous Web-Audio API audio graphs stored as JavaScript objects or files.



A simple frequency modulator with sample-rate modulation of the carrier frequency.

Carrier Frequency 440
Modulation Index 2
Modulator Frequency 30
Gain 0.5
Attack Time 0.05
Release Time 1

Figure 1. Automatically-generated user interface using the jsaSound model API. The Recording button is used to save user-generated sounds; the Code Capture button generates the JavaScript code for using the model interactively within Web applications (see section 4. A Workflow Example)

4. ORGANIZING MODELS FOR ACCESS

Collections of sounds and music are typically licensed and accessed from databases using textual descriptions for indexing. Sometimes they are distributed as specialized collections, but more often they are searched and “auditioned” on-line and then downloaded individually. There is no such standard way of distributing algorithmic sound models, because there has been no standard system for creating and incorporating them in media content. MIDI-controlled soundbanks and their variants are file systems for instruments and sounds, but the sound sets all use a single “general purpose” synthesizer. There are also several aggregate collections of Max/MSP objects on the Web[1], but most of the sharing of algorithmic audio objects is done in an ad hoc way directly between designers and users of common systems. The standardization of the Web Audio API is likely to generate a demand for a process of distribution for interactive sound models analogous to those currently in place for static sound files.

To understand how sound models might usefully be organized, accessed, and used in media production, it is interesting to consider how large a useful collection of sound models might be. For example, the Freesound collection[6] contains over 160,000 user-contributed sounds, and the Web-based commercial SoundDogs collection numbers over 650,000. A sound model can generate an uncountable number of sounds because it is parameterized and furthermore, parameters can change over time. This might suggest that a database of sound models would be smaller than a corresponding sound file database of similar breadth. However, there are many different ways to synthesize a given sound, and the range of sounds that different models can make may intersect even as each model provides quite distinct behaviors under parametric variation. It is thus reasonable to think of the size of a collection of models as being on the same

[1] See for example, http://cnmat.berkeley.edu/downloads or maxobjects.com.

order as that of an analogous collection of sounds, and therefor that some kind of database scheme would be necessary for management.

There are some interesting opportunities and challenges that arise with databases of sound models due to the characteristics that differentiate them from fixed media elements such as sounds. One opportunity that distinguishes sound model databases arises because they are structured code rather than a "flat" string of data. Model structure can be an important aspect of what a database user is looking for, especially when designing a new model – it is much easier to find useful chunks of code than to write them from scratch. This has been explored extensively by Funkhauser and colleagues in the domain of 3D models. For example, associating high-level semantic feature descriptions with model structure facilitates novice model design creation[3]. Similarity metrics based on model structure can assist database navigation[8], and model building can be accomplished in part by example[7]. A second opportunity for model databases comes from the semantic "meta data" that comes with the names of the parameters that a sound model exposes. Possibilities include musical "pitch" for a bell, "rate" (for footsteps), or "roughness" for s scraping action. All could be usefully exploited for finding and associating sound models. The challenge in constructing and indexing databases of sound models is also in part due to the abstract nature of code. How does one index a model that represents many sounds which might be entirely unrelated perceptually, or that generates many of its useful sounds only under real-time manipulation of its parameters?

Indexing static sound files is more straightforward. The Freesound database of sounds uses two different methods of providing metadata for indexing. One is the sound name and additional descriptive text provided by the sound file contributor. The other is generated automatically by extracting audio features based on a suite of signal processing routines[2] designed for this purpose. Features such as spectral centroid, pitch, and roughness are saved with the sounds and can be used to discover relationships between sounds in the database that can be exploited in, for example, "sounds like" searches.

Our first-pass approach to organizing and searching collections of sound models is to do so indirectly through the sounds that they generate. Model developers use the GUI player (Figure 1) to generate as many different sounds as they feel is necessary to represent the range of possibilities for the model, and then upload the sound(s) to the Freesound database along with a text description used for searches. Each sound in the Freesound database has its own Web page with facilities for visualizing and playing the sound, and for examining the textual data provided by the contributor. Freesound supports HTML descriptions so that the sound model developer can create a graphical button that directly opens the player for the sound model responsible for generating the sound the user found in the database. With the sound model open in the player, users now have access not only to the specific sound they found in the database, but the entire class of sounds and behaviors that the model is capable of generating through interaction.

4.1 Serving from the Cloud

"Serving sound models from the cloud" means having them available on an Internet-connected server which can be accessed as an extension to a file system. For the developer, having code components stored at a location accessed the same way from wherever software is being developed or deployed is convenient and storage-efficient. Also, like other libraries that are often served this way (e.g. Google code libraries[2]), the developer is relieved of the responsibility for updates and bug fixes. One drawback is that Internet access is always necessary, though for Web applications this is generally assumed anyway. Another drawback is a dependency on others for keeping the servers available, as well as a loss of control over access (e.g. a malicious provider could unilaterally change their access policy).

For the sound model supplier, cloud serving supports a variety of licensing and distribution models, as well as the ability to understand how and when sound models are being used by both developers and by end users. This usage data can be valuable in the same way it is for any other cloud-based service provider that monitors and takes advantage of knowledge gleaned from usage patterns to improve services. An overview of how the various components work together can be seen in Figure 2.

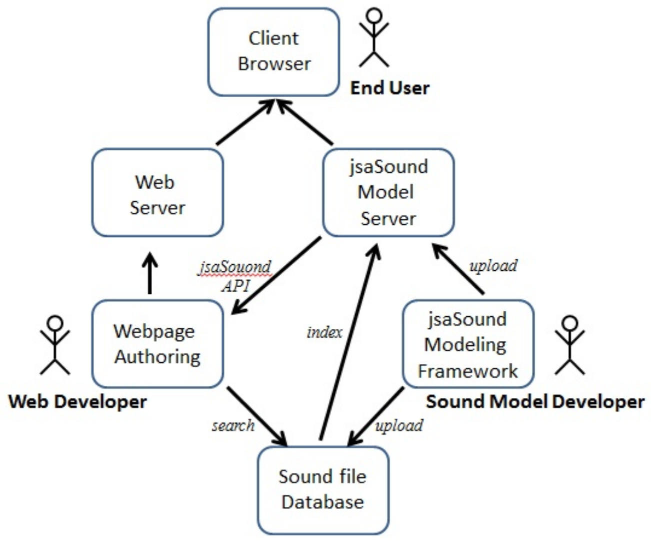


Figure 2. System Overview

jsaSounds are served this way using the light-weight JavaScript server stack based on node.js, express, and socket.io technologies. The require.js library provides functionality that C-programmers are familiar with as "#include." The Web developer-oriented simplicity of interactive sound API can be seen in Figure 3.

5. A WORKFLOW EXAMPLE

The value of the complete ecosystem described above can be appreciated by considering a workflow example.

A Web developer wants to animate an arrow shooting across the screen with a speed dependent on where a user clicks. She decides that the arrow needs a "swish" sound to accompany its flight that must change according to how fast the arrow moves. That is, the sound needs to be interactive with behavior tied to the motion of the graphics.

She navigates her browser to the Freesound.org database, types "swish" as a search term and auditions several of the 248 swish sounds her search returned and chooses the one that has the sonic

[2] https://developers.google.com/speed/libraries/

characteristics she was looking for. Of course, the recorded sound is not flexible in duration nor in the way it changes over time. However, she notices the presence of a button announcing the availability of the synthesis model that generated the swish, and clicks it to bring up the model in the slider-box user interface. She explores the model by moving the swish at various rates. She could record any one of her swish gestures if she needed a static sound different from the one the database search turned up, but for her arrow application, she needs to use the model "live" in her code to maintain its interactivity. She clicks the "generate code" button (Figure 1) which opens a window with the JavaScript text she needs to copy in to her program to load the sound from the server, and to set the default values to those she chose in the slider box. Her one remaining task is to start and stop the sound, and insert a setParameter call wherever she updates the moving arrow graphic, supplying that method with the arrow position. Now, for both her during development, and for the end users who visit her Web page, the sound will be retrieved from the server when the Web page is loaded, and the sonic evolution of the swish will be bound to the position and speed of the arrow.

```
require.config({
   paths: {"jsaSound": "http://animatedsoundworks.com"}
});
require(
  ["jsaSound/jsaModels/jsaFM"], // classic FM
  function (sndFactory) {

    var snd = sndFactory();

    window.onmousedown=function(e){
            snd.play();
    };

    window.onmouseup=function(e){
            snd.release();
    };

    window.onmousemove=function(e){
       //[0,1] normalized x/y mouse positions
            var normX = e.clientX/window.innerWidth;
            var normY = e.clientY/window.innerHeight;

       // setting "Modulation Index" by parameter name
            snd.setParamNorm("Modulation Index", normY);
       // setting "Carrier Frequency" by parameter index
            snd.setParamNorm(1, normX );
    };
  });
```

Figure 3. A complete JavaScript browser application that obtains a sound model from a server and provides real-time user-interaction. Two different forms of the jsaSound parameter setting API are also shown.

6. CONCLUSION

The new Web Audio API enables interactive sound synthesis with native performance on any device with a browser. However, there are different developer communities with different needs that must still be addressed before the technology can be adopted on a massive scale. The jsaSound library provides tools supporting audio specialists in the development of sound models, as well as a standard API for all models that make them reusable, sharable, and straightforward to use for Web developers. Cloud service makes them easy to integrate with the community-supported Freesound.org database indexed by the sounds they are capable of generating, as well as convenient to deploy in Web applications. Remaining future work is just the development of hundreds of thousands of sound models.

7. ACKNOWLEDGMENTS

Thanks to the Freesound.org team at the Universitat Pompeu Fabra Music Technology Group, particularly Frederic Font and Dmitry Bogdanov, for their insights on the inner workings of Freesound. Thanks to Kumar Subramanian for his contributions to jsaSound and to the many other contributors to open source Web audio projects. Thanks to the Sonoport team for seeing the potential of this work in the commercial world. Thank you to the anonymous reviewers for helping to strengthen this presentation.

All software presented in this paper is open source and available at animatedsoundworks.com:8001.

8. REFERENCES

1. Adenot, P., Wilson, C., and Rogers, C. Web Audio API. 2013. http://www.w3.org/TR/webaudio/.
2. Bogdanov, D., Wack, N., Gómez, E., et al. Essentia: An audio analysis library for music information retrieval. *Proceedings of ISMIR*, (2013).
3. Chaudhuri, S., Kalogerakis, E., Giguere, S., and Funkhouser, T. Attribit: content creation with semantic attributes. *Proceedings of the 26th annual ACM symposium on User interface software and technology*, ACM (2013), 193–202.
4. Choi, H. and Berger, J. WAAX: Web Audio API eXtension. *Proceedings of New Interfaces for Musical Expression*, (2013).
5. Cook, P. Modeling Bill's gait: Analysis and parametric synthesis of walking sounds. Audio Engineering Society (2002).
6. Font, F., Roma, G., and Serra, X. Freesound technical demo. ACM (2013).
7. Funkhouser, T. Modeling by example. *ACM Transactions on Graphics 23*, 3 (2004).
8. Kim, V.G., Li, W., Mitra, N.J., Chaudhuri, S., DiVerdi, S., and Funkhouser, T. Learning part-based templates from large collections of 3D shapes. *ACM Transactions on Graphics 32*, 4 (2013), 70.
9. Puckette, M.S. Pure Data: another integrated computer music environment. *Proceedings of the International Computer Music Conference*, ICMA (1996).
10. Subramanian, S.K. Taming the ScriptProcessorNode - Codaholic. http://sriku.org/blog/2013/01/30/taming-the-scriptprocessornode/.
11. Wyse, L. and Subramanian, S.K. The Viability of the Web Browser as a Computer Music Platform. *Computer Music Journal 37*, 4 (2013), 10–23.
12. Wyse, L. A Sound Modeling and Synthesis System Designed for Maximum Usability. *Proceedings of the 2003 International Computer Music Conference: 29th September-4th October 2003, Singapore*, (2003), 447.
13. Wyse, L. Generative Sound Models. *Multimedia Modelling Conference, 2005. MMM 2005. Proceedings of the 11th International*, (2005), 370–377.
14. Zicarelli, D. Cycling '74. 2014. http://cycling74.com/.