SONICBARD – STORYTELLING WITH REAL-TIME SOUND CONTROL, SYNTHESIS AND PROCESSING USING EMERGING BROWSER-BASED TECHNOLOGIES

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

Pallav Shinghal
School of Computing
National University of Singapore
pallav@nus.edu.sg

ABSTRACT

sonicBard is a system providing real-time sound interaction for traditional oral storytelling performers. The system is designed to give storytellers an extended capability to accompany themselves with music, sound effects, and vocal voice transformations while minimally disrupting their established performative techniques. Since a given story may potentially incorporate dozens of sounds, each with their own interaction demands, a serious challenge is posed. The constraints stemming from the "hands and eyes busy" nature of storytelling leaves little room for the attentional demands of typical computer-controlled instruments. This paper presents the interface design and implementation, and describes newly emerging browser platform capabilities upon which we have chosen to develop sonicBard.

1. INTRODUCTION

sonicBard is designed as an instrument to be used by traditional oral storytellers to accompany themselves with sound. Some storytelling traditions are very theatrical or visual, however we focus here on those that are primarily oral and aural – an art of speech and sound.

Of course, all live storytelling involves a performative and visual element, and storytellers in many different traditions use a variety of simple props and instruments to support their oral craft. Props can be anything, from stones knocked on a surface to signal a change of scene[1], to fans, scarves, and staffs. The instruments storytellers often for accompaniment range from those in the lute and lyre families[2] such as the Pipa[3] or guitar to percussion instruments such as tambourines, rattles, or drums. sonicBard is distinguished from traditional instruments because the range of sounds it can generate is limited only by what sound designers can create, and because it can process sounds (such as the storyteller's voice) as well as generate them.

Our design goals are based on how stories are told and shared, and on the way storytellers typically use musical instrument[4]. One goal for the instrument was to construct a system that is easy to hold, restricts the teller as little as possible from moving about and using their arms and hands to gesture when not using the instrument, and to some extent even when the instrument is being used. We also did not want the instrument to require visual attention. These criteria determined our choice of a hand-held mobile device as the instrument.

The fact that they system can control any sound, and any number of sounds, creates a challenge for interface design. Sounds must be selected and controlled in a manageable way for a storyteller whose primary goal is the story, not the instrument.

Another design goal for sonicBard is to maintain the fluent way that stories are shared among storytellers in oral traditions. This is a very difficult criterion for computer-based technology to meet, since typically physical media or installation on a device would be required. A story on the other hand is essentially shared once it has been told and heard. This design goal drove the entire project to be developed on the web platform. With sonicBard, "having" the sonically-enabled story after hearing it is as simple as pointing a mobile device browser at a URL.

2. SYSTEM DESIGN

The architecture for sonicBard consists of:

1) a collection of "sound models",
2) a scene navigation and interface controller, and
3) a system for organizing collections of sounds into "scenes" and providing mappings between the controller and the sounds.

2.1. Sound Models

The collection of sound models are built using the formative W3C Web Audio API[5] (discussed below). In addition, we have constructed a library called jsaSound which provides utilities (opcode-like generators, unit converters, etc), templates (code for frequently used model structures), and of primary importance for the sonicBard project, a simple and consistent interface for all sounds.

The jsaSound interface makes a clear distinction between *sound developers* and *sound users*. Sound developers use JavaScript, the Web Audio API, and the jsaSound library tools to build sound models, whereas sound users such as game, music application, or web developers simply load, play, and set up relationships between application or user actions and interactive sound behavior. The API covering the key functionality of jsaSound models is shown in Table 1.

The jsaSound API makes it straightforward to, for example, create a graphical user interface that works for all sound models (xxx.html to see it in action). The API also makes it easy to create mappings between controllers and sound behavior which is an essential part of authoring story rigs in sonicBard.

play()	Start the sound playing
stop()	Stop the sound playing
release()	Stop allowing any release segment of a sound
setParam([name, number], val)	Set a parameter value using its natural units
setParamNorm([name, number], val)	Set a parameter value using a normalized range [0,1]
getParam ([name, number], ["name", "type", "val", "normval", "min","max"])	Get information about a parameter (e.g. its name, type, value, or range in natural units)
getNumParams()	Get the number of parameters a model exposes
getAboutText()	Get textual information about the sound model

Table 1. The single API exposed by all jsaSound models built with the Web Audio API and JavaScript.

2.2. Communication

The control interface and the synthesis are two separate systems each implemented as web pages that communicate with each other. The control interface is accessed through a browser on a smart phone (where access to accelerometers is also available through a JavaScript API), and the synthesis system runs on a PC (which may also function as the web server for both systems).

The "bridge" between these two systems is a server that also acts as a "message router". The broad-level functionality of this server is based on the concept of "parties". A "party" is a logical collection of controllers and synths that share messages with each other, and is indexed by a 9-character code shared between party members. The concept is very similar to chat rooms.

When the controller web page is accessed, it automatically generates the 9-character code that will serve to identify a new "party". The controller then sends a "register" message to the server along with its party code. After this, the controller sends messages to the server whenever it receives a user-generated event such as a button-press or a spatial motion signal (coming from the accelerometers on the device). The controller also displays its party code on the screen, so that a user may use it to add a "synth" to the same party, as described below.

The collection of sound models resides on a separate web page. A machine running a browser that supports the Web Audio API (discussed below) (Macs or PCs running Chrome, Canary or Safari) points the browser to the URL for this page. The "messaging" phase starts when the user enters the 9-digit code that the synthesizer page registers with the server to let it know it wants to receive all messages from parties with the same ID.

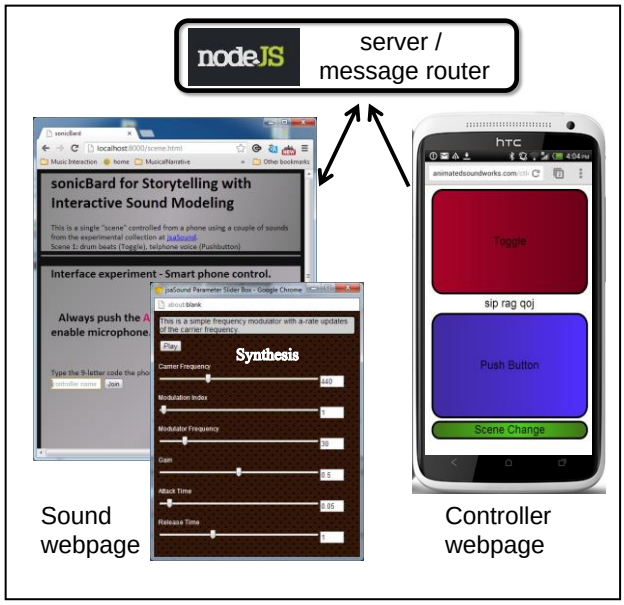

Figure 1. The communication structure for the controller and the synthesis systems. One machine navigates to the synthesizer webpage, the other to the controller webpage. A 9-letter code is used to identify groups for message sharing.

A story rig is a collection of scenes, where a "scene" consists of two main elements: the list of all models that the synthesizer needs to load from the jsaSound library and have available at one particular time, and a list of "handlers" that represent the mappings from the range of possible controller messages to the sound model behaviors. Controller messages are mapped to states or parameter settings across any number of sound models.

The desire to make story rigs sharable drove us to consider the web as a platform for the entire system. However, we have been stretching the limits of what is possible on this platform just as host of new capabilities are becoming available. The next section discusses these new developments and the feasibility of the web for large-scale interactive computer music and synthesis systems.

3. THE WEB PLATFORM FOR CONTROL COMMUNICATION AND SYNTHESIS

The browser has never been the platform of choice for serious real-time interactive computer music and sound synthesis. Up until recently, real-time sound synthesis could only be accomplished in a specially written plug-in such as the pure-Java ASound[6] where latencies below 100ms were impossible to achieve on common operating systems, or JSyn[7] which was originally built on a back-end written in 'C' speeding things up, but still

required user installation. In 2008, Adobe Flash added real-time synthesis on their pervasive platform. Despite its popularity, Flash was still a browser plug-in, and the real-time synthesis latency performance was abysmal.

In the late 1990's, Netscape introduce JavaScript as part of the browser platform. It became a standard way to add interactivity to client-side browsers, but it was an interpreted language and designed neither for large software projects nor for demanding computational tasks such as signal processing and synthesis (c.f. Crockford[8] for a discussion of the strengths and weakness of the language). Further hindering serious sound application development in browsers is the security procedures necessary for protecting users running code from networked sites such as "sand-boxing" that restricts access to the local operating system, file system, and media such as the microphone and video camera.

This is all changing very quickly now due to new browser standards that are being defined in several of the W3C working groups, and experimentally working its way on to implementations from most major browser platform providers. The confluence of related efforts in JavaScript engines, server architectures, and web standards for synthesis and security management are creating significant new opportunities for music and sound on the web, many of which sonicBard builds upon.

Underlying all of the technological developments above is the open-source V8 JavaScript engine built by Google and released in 2008. The Chrome and Safari browsers run on the V8 engine, as do the communications and server technologies (Node.js and Express resp.) discussed below. Using a combination of "pre-compilation" instead of interpretation, efficient memory management, and smart garbage collection, V8 and other modern engines represent a major performance improvement over previous JavaScript engines.

The most directly relevant enabler for sound and music synthesis with browsers is the emerging Web Audio API[5] standard. It provides an audioContext object which runs graphs of nodes (such as oscillators, filters, reverberators, waveshapers, mixers, and gain) that sound developers construct by connecting the outputs of one node to the inputs of another. Although graph construction using the API is text based (JavaScript code), this is conceptually similar to graphical programming languages such Max/MSP and PD. Predefined nodes also perform the computationally demanding convolution and FFT operations . Of course, the nodes that are defined in this emerging standard are implemented by browser providers in native code. Graphs are executed in a dedicated thread to guarantee timely processing of the signal flow, and the result is processing speeds and latencies that approach those of native applications such as Max/MSP.

A limitation of the Web Audio API as it stands at the time of this writing makes it difficult to incorporate use-defined signal generating and processing algorithms. Web Audio defines a ScriptProcessorNode which is intended to be used for wrapping JavaScript code in a that can be used in constructing the audio graphs with the other system-defined nodes. This is obviously a critical capability for the computer music community. Unfortunately, the behavior and capabilities of the ScriptProcessor node are quite different from other nodes. They do not have the same access to time information that other nodes have, do not support the sample-accurate starting and stopping that other signal source nodes do, do not support the AudioParam nodes with their sample-accurate automation. Subramanian[9] has provided details of these and related issues, as well as code for reasonable workarounds. The potential of the browser platform for serious computer music work will pivot on the capability of this component of the Web Audio API.

For real-time voice transformation in sonicBard, new browser capability from a different working group developing WebRTC[10] is used. WebRTC is an open source project devoted to real time communication and supports peer-to-peer media exchanges among other things. The getUserMedia() method permits access to the client video and audio inputs (with explicit user permission). The Web Audio API can then grab these streams and process them through the node graphs discussed above. Latency is the key to usable signal processing from external sources. On a 2.4GHz Windows 7 machine, we have measured a 60ms throughput interval from microphone to speaker which is as fast as any other native applications on the same platform using the default system audio driver. The latency was measured by placing a microphone in front of the speaker, parameterizing the volume levels to be near the feedback threshold, delivering an impulse response and recording signal to measure the feeback delay times. On the Mac platform, the microphone-to-speaker throughput is considerably shorter than what we measured on Windows - too short, in fact, to be measured using the method described above. The delay time on Windows is noticeable to the storyteller using the system for voice, but does not significantly interfere with the ability to speak. On the Mac platform, the short latency makes this browser-based system feel as natural as using a PA system.

Node.js[11] is a server-side technology from Joyent built on the Google V8 engine. It provides the capability for connecting and exchanging messages between the browser running the parametric controller interface (typically on Android or iOS devices running Chrome) with the browser pointed to the URL running the synthesizer. Canning[2] described a similar messaging passing system for musical notation. Among other things, Node.js provides non-blocking event-driven i/o, easy handling of JSON, access to a wide variety of libraries that, for example, simplify socket usage, and (through the usage of JavaScript), a very "consistent" developer environment across both servers and clients, making the code maintainable and extensible.

Express is a light-weight web application framework that works with Node.js to serve the web pages (technically, the "static" content) for both the controller

and the synthesizer. The advantages of using this server are primarily pragmatic. First, it is much easier to use than the weightier Apache on networked Linux servers or setting up the "IIS" manager on Windows. The inter-browser communication can be run over a wide-area network, but for storytelling and other musical applications, the typical latency and inconsistent message packet delivery times are disruptive. Second, using a single server for both the message routing, as well as for serving static files, allows us to data access issues created by the Same Origin Policy designed for security in networked environments. Node and Express can be run to serve pages from a local machine without rewriting any code.

The world of computer music did not need yet another language for sound synthesis and control. However, the browser platform brings with it a host of capabilities, opening up new creative possibilities. Thinking of the browser as an operating system (which it is in effect becoming), having sound synthesis and processing capabilities built in to the operating system makes it easy to integrate sound with the multitude of applications written for that platform. Tremendous networking and graphics capabilities are also in libraries within the same system. Finally, since browsers are supported on every major operating system and hardware platform, the holy grail of platform independent development may soon be upon us (although admittedly, this promise has been dashed as often as it has been made in the past).

4. AUTHORING

As described above, a story scene is defined by a set of sounds that are "prepared" with mappings from the control interface to specific behaviors controlled through the jsaSound interface. There may be many scenes in a story, each rigged with different sounds and mappings. In order to provide a system that puts minimal attentional demands on the storyteller, our initial design consists of a single physical interface for all scenes where all sounds are controlled by the exact same small set of physical gestures.

The current interface consists of a toggle-button that can turn a sound (or collection of sounds) on or off, and a push-button that switches between two states of a sound or set of sounds depending on whether the button is being held down or not. Finally, the pitch and the roll of the hand-held device are two independent controls that can be mapped to a specific range of sound parameters. An example of a mapping would be a drum set rhythm turned on/off with the toggle button, and a microphone that is always on, but switches between "dry" (no processing), and "wet" settings for some effect such as pitch shifting, filtering, or reverberation. For additional consistency, we have been using the pitch-angle of the device to control the volume of all sounds in all scenes, leaving roll to control some other scene-specific sound parameter - for the purposes of this example, the tempo of the rhythm set. A scene also defines the default settings that a scene opens with, and how sounds close at the end of the scene (either stopping immediately, or trailing off in a sound-specific "release" mode).

Currently, the set of scenes that comprise a rig are defined textually in a JSON data structure, but the goal is to allow storytellers to define their own rigs using a browser-based graphical tool and a database of sound models. This is to support the mutability of shared stories, but is also critical for allowing the creation of a customized interface sounds that each individual storyteller is most comfortable with.

5. STORYTELLING FEEDBACK AND THE REAL WORLD

We put sonicBard into the hands of a professional storyteller with a scene design informed by one of her stories, and asked for feedback on all aspects of the system from general set-up to the specific sound behaviour. Most of the comments concerned the general nature of the system.

Our assumption that we might be able to create a system that would not interfere with the gestures a storyteller would naturally make was brought in to question. Although the device fits comfortably in one hand and does not require visual attention, it does impose some limits on the kinds of gestures that the storyteller might otherwise make. A two-handed clawing gesture is one simple and common enough example. Possibly more important than the restrictions on certain gestures, is the fact that the presence of the device would be obvious to the audience. The teller commented that if the audience understands that there is a device, and that it is being used as an integral part of the performance, then pretending it is not there would be more distracting that unabashedly using it as a instrument/prop.

The possibility of treating the device as a prop with some kind of understanding shared with the audience about how it is used, opens up other possibilities for the form factor. For example, a tablet would certainly free up additional real estate for interaction, and at the same time lend itself to the possibility of having a presence in the same way that props or other instruments do. This approach would however not relieve the design constraint that the sounds be playable with minimal cognitive or visual attentional demands.

Other practical consideration that came to light from our user test was that if storytellers were going to be expected to use their own devices for interaction, then they would need to be "prepared" with more than just the authored story rig. Real-world issues such as the proximity of the intended interaction area on the screen to hardware or menu buttons that change the device behavior, the automatic landscape/portrait reorientation that many phones offer by default, and of course, the possibility of receiving phone all have to be managed and represent a threat to seamless storytelling.

Storytellers are already confronted with a variety of "risks" when they walk in to a venue to deal with variable audience sizes, lighting conditions, microphone/speaker set ups, and technical support

people they may have never met. Minimizing these dangers for a successful storytelling event is of paramount concern, so anything that presents a potential technical difficulty, or that could disrupt the timing or rapport with the audience is something to be avoided.

One thing that becomes clear in evaluating technology for creative use is that artists do not have problems in search of solutions in the same way that users of technologies do in more goal directed environments such as computer-supported collaborative work (CSCW). In the latter case, the objectives are clearer, and the effect of a technology introduced in to that context can be measured objectively, for example, in terms of time to achieve an objective. In creative contexts, the goals are much harder to identify (e.g. execute a satisfying musical transition), or at least harder to measure.

Artists explore, create, and invent with what they have rather than viewing what they do not have as a problem. When we discussed the ability to create voice transformations with our technology, the response from the storyteller was that she already does that without technology. If she needs more sounds than she could possibly generate with a single instrument, she works with a musician. Does that mean we are falling in to the classic trip of creating technological solutions to problems that do not exist?

One way forward to address this challenge for evaluating technology in a creative environment is to first establish that there is something new the technology offers in a particular artistic context, and then provide situated access to the new capability, and test whether or not it is used. In this case, the test is not for how well (or not) a problem is solved, but rather to what extent a new technology is of value.

In the case of sonicBard, what differentiates the use of the system from working with a co-performing musician who has access to the same range of sounds sonicBard offers, is the intimacy with which storytellers can coordinate sound behavior with improvised aspects (unplanned in some aspect of content or timing) of the storytelling. For example, this system offers the teller the ability to time sounds and voice transformations with other story elements in a more intimate and nuanced way than would be possible be coordinating with a separate performer. Further testing along these lines is next on our agenda.

6. SUMMARY

A sound and voice transformation instrument for storytellers to use in accompanying their own stories is being designed in collaboration with professional storytellers. The system is pushing at the edges of the technological capabilities of browser-based platforms which we have found to be largely suitable with several caveats that we hope will be addressed as the standards and implementations evolve.

7. ACKNOWLDGEMENTS

We would like to thank storyteller Rosemary Somaiah for the valuable feedback she provided, as well as for her patience and willingness to experiment with our earliest prototype. Thanks to Srikumar Subramanian for his JavaScript wisdom and open source contributions to jsaSound and sonicBard. This work was supported by Singapore MOE grant FY2011-FRC3-003, “Folk Media: Interactive sonic rigs for traditional storytelling”.

8. REFERENCES

[1] V. Bordahl, F. Li, and H. Ying, Eds., *Four Masters of Chinese Storytelling: Full-length Repertoires of Yangzhou Storytelling on Video*. NIAS Press, 2004.

[2] T. Sheppard, “Musical Instruments for Traditional Storytelling,” *Musical Instruments for Traditional Storytelling*, 01-May-2004. [Online]. Available: http://www.timsheppard.co.uk/story/dir/traditions/instruments.html. [Accessed: 03-Mar-2013].

[3] H. Werle-Burger, “Interactios of the media Storytelling, Puppet Opera, Human Opera and Film,” in *The eternal Storyteller; Oral Literature in Modern China*, V. Bordahl, Ed. Curzon Press, 1999.

[4] L. Wyse and S. Subramanian, “Foundations of interactive sound design for traditional storytelling,” in *Proceedings of the International Computer Music Conference*, Ljubjana, Slovenia, 2012.

[5] C. Rogers, “Web Audio API,” *Web Audio API - W3C Editor’s Draft*, 2012. [Online]. Available: https://dvcs.w3.org/hg/audio/raw-file/tip/webaudio/specification.html. [Accessed: 03-Mar-2013].

[6] L. Wyse, “A Sound Modeling and Synthesis System Designed for Maximum Usability,” in *Proceedings of the 2003 International Computer Music Conference: 29th September-4th October 2003, Singapore*, 2003, p. 447.

[7] P. Burk, “JSyn–a real-time synthesis API for Java,” in *Proceedings of the 1998 International Computer Music Conference*, Ann Arbor, MI, pp. 252–255.

[8] D. Crockford, *JavaScript: the Good Parts*. O’Reilly Media / Yahoo Press, 2008.

[9] S. K. Subramanian, “Taming the ScriptProcessorNode - Codaholic.” [Online]. Available: http://sriku.org/blog/2013/01/30/taming-the-scriptprocessornode/. [Accessed: 03-Mar-2013].

[10] “WebRTC.” [Online]. Available: http://www.webrtc.org/. [Accessed: 03-Mar-2013].

[11] “node.js.” [Online]. Available: http://nodejs.org/. [Accessed: 03-Mar-2013].

[12] R. Canning, “Realtime Web Technologies in the networked Performance Environment,” in *Proceedings of the International Computer Music Conference*, Ljubljana, Slovenia, 2012, pp. 315–319.