thinking in the view of Jean-Jacques Nattiez \[8\] who points out that, by contrast, post-modernism is more concerned with the passing of time.

In fact, Xenakis also defines a third category, the temporal which might be puzzling until one realizes that although sound parameters have an Abelian (commutative) group structure for physical time, being irreversible, does not have a group structure (i.e. there are no inverses). The temporal category undermines this peculiarity of the experience of time that sets it apart from the other aspects of a musical composition. Now, although an asymmetric succession of arbitrary durations does not form a group, nonretraversable rhythms and iterations such as ostinato or traditional, constant steady meter, do. They create a cyclical time associated with the “eternal return” identified by Mircea Eliade as characteristic to all rituals \[3\]; they compel us to participate in an exemplary action that is repeated at regular intervals and supersede our perception of the mundane flowing of “ordinary” time.

In DISSCO, indeterminate processes that distribute events in-time complement or even upset these unhistorical, cyclical returns created by sieves templates. Since a variant of the manifold is generated in one seamless run, it owes its distinct integrity, its own “personality”, to a particular “history” provided by a sequence of random numbers. However, a random number generator actually produces pseudo random numbers and the same seed engenders the same exact output. An interesting situation is then created: the events in the piece depend on indeterminacy but the sequence of random numbers, the source of randomness, is itself a deterministic chain. One more layer in the play between structure and chance, causality and indeterminacy.

Sieves that generate rhythmic palindrome ostinatos also provide a gateway for virtual incursions in time and collections of relative, equal, but mutually exclusive aleatory-type alternatives. By combining the use of group structures with indeterminate procedures in the selection of some but not all possible time events, two worlds are made to compete with each other in complex and realistic ways. When the artistic product is also a manifold composition with its arbitrary number of actual and potential variates, having the same outside-time structure, the relativity and the ephemeralness of all in-time artifacts is underscored.

5. REFERENCES


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.

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 audio server whenever it detects 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 as a light on the controller application so that a user can add this party to its party list, and also it will add the party's interface to a back-end written in 'C' speeding things up, but still required user installation. In 2008, Adobe Flash added real-time synthesis on its pervasive platform. Despite its popularity, Flash was still a browser plug-in, and the resulting audio latency performance was still unacceptable.

In the late 1990's, Netscape introduced 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-scale software projects nor for demanding computational tasks such as signal processing and synthesis (c.f. Crockford[8] for a discussion of the strengths and weaknesses of the language). Further hindering serious real-time interactive computer music and sound development in browsers is the security procedures necessary for protecting users running code from networked sites such as "sandboxing" 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 into implementations from major browser platform providers.

### Table 1

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>play()</td>
<td>Start the sound playing</td>
</tr>
<tr>
<td>stop()</td>
<td>Stop the sound playing</td>
</tr>
<tr>
<td>release()</td>
<td>Stop allowing any release</td>
</tr>
<tr>
<td>setParam(name, value)</td>
<td>Set a parameter value using text</td>
</tr>
<tr>
<td>getParam(name)</td>
<td>Get information about a parameter</td>
</tr>
<tr>
<td>getNumParams()</td>
<td>Get the number of parameters</td>
</tr>
<tr>
<td>getAboutText()</td>
<td>Get textual information</td>
</tr>
<tr>
<td>setParamNorm(name, val)</td>
<td>Set a parameter value using norm</td>
</tr>
<tr>
<td>setParamNorm(name, val, range)</td>
<td>Set a parameter value using normalized range [0,1]</td>
</tr>
<tr>
<td>getNumParam()</td>
<td>Get the number of parameters</td>
</tr>
</tbody>
</table>

### 2.2. Communication

The control interface and the synthesis are two separate and independent web pages that do not 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 sound devices 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 audio server whenever it detects 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 as a light on the controller application so that a user can add this party to its party list, and also it will add the party's interface to a back-end written in 'C' speeding things up, but still required user installation. In 2008, Adobe Flash added real-time synthesis on its pervasive platform.

In the late 1990's, Netscape introduced 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-scale software projects nor for demanding computational tasks such as signal processing and synthesis (c.f. Crockford[8] for a discussion of the strengths and weaknesses of the language).

Further hindering serious real-time interactive computer music and sound development in browsers is the security procedures necessary for protecting users running code from networked sites such as "sandboxing" 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 into implementations from 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 W3C Web Audio API built by Google and released in 2008. The Chrome and Safari browsers run on the V8 engine, as do the communication and server technologies (Node.js and Express.js) discussed below. Using a combination of "compilation" instead of interpretation, efficient memory management, and smart garbage collection, V8 allows JavaScript to achieve real-time performance improvement over previous JavaScript engines.

The most directly relevant enabler for sound and music synthesis on browsers is the emerging Web Audio API[5] standard. It provides an audioContext object which runs graphs of nodes (such as oscillators, filters, effectors, wavefolders, 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), 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 necessary data for a given application is typically generated on the server where the node graphs are executed in a dedicated thread to guarantee thread isolation 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 way that can be used in constructing the audio graphs with the other system-defined nodes. This is obviously a critical capability for the future.

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 support, so 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 capabilities from a different working group developing WebRTC[10] is used. WebRTC is an open source project dedicated to real time communication and supports peer-to-peer media exchanges among other things.

The getMedia() 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. Lateness to enable 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 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 feedback 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 measurable 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 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 and the server provider (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 managing 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, simpler to use through the usage of JavaScript, a very "consistent" developer environment across both servers and clients, making the code maintainable and extensible.

Express.js is a lightweight and flexible Node.js framework that uses the Node.js to serve the web pages (technically, the "static" content) for both the controller
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 host mappings between controllers and sound behavior which is an essential part of authoring story rigs in soniBard.

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

Figure 1. The communication structure for the controller and the synthesis systems. One machine navigates to the web page, the other to the controller page. 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 not 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, SYNTHESIS, AND COMMUNICATION

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 use it in a specifically 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 resulting audio latency performance was simply unacceptable.

In the late 1990’s, Netscape introduced 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 weaknesses 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 into 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 coming together to create a new opportunity for music and sound on the web, many of which sonicBard builds upon.

Underlying all of the technological developments above is the built-in 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) 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, wavefomers, 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), this is conceptually similar to graphical programming languages such Max/MSP and PD. Defined nodes also perform the computationally demanding convolution and FFT operations. Of course, the notation that is defined in this emerging standard are implemented by browser providers in native code. Graphs are executed in a dedicated thread to guarantee thread safety of the signal flow, and the result is a process stream and process them through the node graphs.

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, wavefomers, 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), this is conceptually similar to graphical programming languages such Max/MSP and PD. Defined nodes also perform the computationally demanding convolution and FFT operations. Of course, the notation that is defined in this emerging standard are implemented by browser providers in native code. Graphs are executed in a dedicated thread to guarantee thread safety of the signal flow, and the result is a process stream and process them through the node graphs.

For real-time voice transformation in sonicBard, new browser capabilities 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 get(/**/Media)**) 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- sensitive 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 applications on the same platform using the default system audio driver. The latency was measured by placing a microphone in front of the speaker, triggering the volume levels to be near the feedback threshold, delivering an impulse response and recording signal to measure the feed back 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 and the server-side browser (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 applications, including 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 lightweight framework that works with Node.js to serve the web pages (technically, the “static” content) for both the controller.

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>play()</td>
<td>Start the sound playing</td>
</tr>
<tr>
<td>stop()</td>
<td>Stop the sound playing</td>
</tr>
<tr>
<td>release()</td>
<td>Stop allowing any release</td>
</tr>
<tr>
<td>setParam([name, number])</td>
<td>Set a parameter value using its natural units</td>
</tr>
<tr>
<td>setParamNorm([name, number])</td>
<td>Set a parameter value using a normalized range [0,1]</td>
</tr>
<tr>
<td>getParam([name, number])</td>
<td>Get information about a parameter (e.g. its name, type, value, or range in natural units)</td>
</tr>
<tr>
<td>getNumParam()</td>
<td>Get the number of parameters a model exposes</td>
</tr>
<tr>
<td>getAboutTest()</td>
<td>Get textual information about the sound model</td>
</tr>
</tbody>
</table>
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 W3C “XMLHttpRequests. The interbrowser communication can be run over a wide-area network, but for storytelling and other musical applications, the typical latency and inconsistent message packet rate 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 sound design the multimedia applications written for that platform. Tremendous networking and graphics capabilities are also in libraries within the same system. Finally, since browsers are supported, 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 simple physical interface for all scenes where all sounds are controlled by the exact same small set of physical gestures.

The interface consists of a button-toggle that can turn a sound (or collection of sounds) on or off, and a push-button that switches between 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 used to toggle a 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. Additional consistency, we have been using the pitch -of the hand -held device are two independent controls that can turn a sound (or collection of sounds) on or off, and the synthesizer. The advantages of using this server are primarily pragmatic.

5. STORYTELLING FEEDBACK AND THE REAL WORLD

We put sonicBard into the hands of a professional storyteller with a scene design informed by her 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 require visual attention, it does impose some limits on the kinds of gestures that the storyteller might otherwise make. A two-handled clawing gesture is one simple and common enough example. Possible more is 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 unashamedly using it as an instrument/prop.

The possibility of using 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 controls for interaction, and it does 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 considerations 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 hand-held device to the screen, as well as the portability of the device that change the behavior, the automatic landscape/portrait orientation that many phones offer by default these days, the possibility of receiving SMS messages 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 this 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 solving the right kind of problems 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 clear, 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 hard to measure.

Artists experiment, 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 same trap of the “new technology solutions to problems that do not exist?”

One way forward to address this challenge for evaluating technology in a creative environment is to focus on whether the new technology offers in a particular artistic context, and then provide a more accessible to the new capability, and test whether or not it is used. In this case, the test is not for how well (or how 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 is, the intimacy with which storytellers can coordinate sound behavior with improvised aspects of the storytelling. For example, this system offers the teller the ability to time sounds and voice transformations 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 that might not be possible 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 to use in accompanying their own stories is being designed in collaboration with professional storytellers to use in accompanying their own stories is being designed in collaboration with professional storytellers to use in accompanying their own stories is being designed in collaboration with professional storytellers to use in accompanying their own stories is being designed in collaboration with professional storytellers to use in accompanying their own stories is being designed in collaboration with professional storytellers.

7. ACKNOWLEDGEMENTS

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 prototypes. Thanks to Srikumar Subramanian for his JavaScript wisdom and support in generating solutions 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

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 an “Apache” environment. The inter-broker communication can be run over a wide-area network, but for storytelling and other musical applications, the typical latency and inconsistent message pacing 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.

In order to provide a system that puts minimal additional real estate for interaction, and at the same time lends itself to the possibility of having a presence in the audience without being held by the 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 possible. More important is the restriction of the visual context. In the current setup, the presence of the device would be obvious to the audience. The teller remarked 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 than unobtrusively using it as an 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 does not lend itself to the possibility of having a presence in the same way that props or other instruments do. This approach would however not preserve 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 teller’s hands to the screen, other equipment, or hardware or menu buttons that change the device behavior, the automatic landscape/portrait orientation that many phones offer by default, etc. If so, the possibility of receiving a callback will 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 these variables, 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 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 perfect control over the space in which they produce work, and 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 failing in the context? Is it not for what (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 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. ACKNOWLEDGEMENTS

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 prototypes. Thanks to Srikanth Subramanian for his JavaScript wisdom and for his contributions to jsAudio and sonicBard. This work was supported by Singapore MOE grant FY2011-FRC3-003, “Folk Media: Interactive sonic rigs for traditional storytelling.”

8. REFERENCES