Quick Live Coding Collaboration In The Web Browser

Chad McKinney
University of Sussex, UK
C.Mckinney@sussex.ac.uk

ABSTRACT

With the growing adoption of internet connectivity across the world, online collaboration is still a difficult and slow endeavor. Many amazing languages and tools such as SuperCollider, ChucK, and Max/MSP all facilitate networking and collaboration, however these languages and tools were not created explicitly to make group performances simple and intuitive. New web standards such as Web Audio and Web GL introduce the capability for web browsers to duplicate many of the features in computer music tools. This paper introduces Lich.js, an effort to bring musicians together over the internet with minimal effort by leveraging web technologies.

Keywords

Live Coding, Network Music, Web Audio, Web GL

1. INTRODUCTION

There are a myriad of computer music languages and environments with built in networking libraries. Common examples are SuperCollider, Max/MSP, and ChucK, which all make use of the Open Sound Control (OSC) [25] protocol to share information between computers and programs. While powerful, these libraries require a large amount of effort to create some framework for groups to use. This has led to a proliferation of higher level network libraries and frameworks such as OSCGroups [1], Benoit Lib [16], the Co-Audicle [22], and the Republic [10] which all serve to simplify collaboration by building on top of existing technologies. These tools are rich and powerful but are difficult to use, requiring expertise and with varying levels of stability and operating system support.

Browser based technologies are a good choice for collaborative frameworks because web browsers are common and their development is widely supported with heavily funded development. The recent advent of web standards such as WebGL [11] and more recently Web Audio [19] has led to many early efforts for web based graphics and audio applications. WebGL and Web Audio are important because they give developers access to powerful functionality such as OpenGL Shader Language (GLSL) support and real-time audio synthesis that was previously unavailable. Among these online graphics and audio applications are several live coding environments. Live Coding is a natural fit for web development, not just because of WebGL and Web Audio, but also because of the built in support for text editing in HTML documents and the ability to use JavaScript as a target language for code generation.

Before browser based live coding, networked live coding has been an established technique because the information that is shared between users is lightweight and highly abstracted. Unlike audio and video streaming approaches, the sharing of code between users requires very little bandwidth and adverse effects from dropped packets can be easily corrected or even ignored depending on the intent of the system design. PowerBooks Unplugged were early pioneers in merging networking techniques with live coding performances [20]. Eschewing the stage to sit among the audience, members use only the sound produced by their laptop speakers, although sometimes augmented by an amplified signal for mid and low frequencies, using OSC messages to share their code among members. Live coding performances have utilized many different languages and approaches often with an emphasis on honesty and communication with audiences [23]. Because of the difficulty of programming live, several live coding specific languages have subsequently been created such as Tidal [14] and IxiLang [12]. These languages provide very high level syntaxes and semantics for the fast creation and manipulation of audio sequences and synths graphs.

Most browser based live coding systems don't use the same code sharing approach as a group such as PowerBooks Unplugged because they are designed for a single user. Before the creation of WebGL and Web Audio, sites like Jsaxus [2] and Flaxus [9] used JavaScript and Adobe Flash to allow users to program graphics based applications in real time. The standardization of the Web GL specification allowed for 3D accelerated programs to be written entirely inside a web browser. Sites like livecoding.io [8], Livecodelab [6], Livecoder [15], WebGL Playground [21], and GLSL Sandbox [4] have harnessed this API to create live programming environments that can be easily accessed from any computer using a web browser. More recently the Web Audio specification has been developed, but the project is younger than WebGL and is still undergoing active development and lacks a complete cross-browser implementation. CoffeeCollider [26] is a newer framework that looks to bring SuperCollider like functionality to the browser. It leverages CoffeeScript for its more elegant syntax while providing basic functionality for synthesis and sequencing, but does not support graphics or collaborative programming. Gibber is an early adopter and provides a thorough audio synthesis environment for live music and graphics programming [17]. The author has proposed adding collaborative sessions and clock synchronization to the project however currently these features are still in development [18]. Because none of these

NIME'14, June 30 – July 03, 2014, Goldsmiths, University of London, UK.

technologies are complete or offer the features required to bring ease to online computer music collaboration, the author has endeavored to create a new language designed from the beginning with quick collaboration in mind. In this paper Lich.js is presented with an explanation for the language design, audio and graphics capabilities, networking, and summed up with a look towards the future.

2. LICH.JS

Lich.js was created to achieve several goals including quick collaboration, graphics and audio live coding, a terse and expressive language, and a set of powerful but safe language features. Collaboration is the main priority and it is for this reason that the language is web based. The goal of Lich.js is to make collaboration fast and painless for everyone, from the most experienced network musicians to the casual interested party. Having the language running on the web allows for members to join a group and start making art without installing a single piece of software. The end goal is for a group of people to sit down at any computer and start making music together, as if the computer were like any other instrument, lacking the requirement of some special software pre-installed on the device.

The consequence of choosing a web based design is that JavaScript now becomes the target language. I decided early on that the syntax provided by JavaScript is too verbose to allow for easy live coding. By creating a domain specific language, unique syntaxes can be created to make expressions terse and clear with respect to music and graphics. Lich.js is a pure functional language with syntax similar to Haskell, but unlike Haskell it is dynamically typed and compiles to JavaScript. Haskell-like syntax was chosen because it provides both an existing language as an established reference point and because the syntax is terse and declarative. It also provides some other useful syntax elements such as pattern matching, which allows function arguments and case statements to easily select and decompose objects by type. Lich.js also implements partial application, which is similar to currying in Haskell, and allows for functions to be passed a number of arguments less than the expected total. The return value is now a new function with that many fewer arguments, and with the curried arguments predefined in a new closure. Lich.js also widely uses a streaming operator >> for data flow, which takes inspiration from the |> forward pipe operator in F#. This operator fits well in a music language because it composes chains of computation that are expressed similarly to a signal path through a series of effects pedals or synth modules. For example "saw 440 >> lowpass 900 1 >> gain 0.1" expresses a saw oscillator feeding into a low pass filter and finally being lowered in volume.

Lich.js's lexer and parser are written in Jison [5], which compiles input strings of Lich.js into an abstract syntax tree. This syntax tree is then recursively traversed generating substrings of code which are concatenated together to produce the final target JavaScript code string. This JavaScript string is evaluated against a pre-built runtime that facilitates audio and graphics sequencing, and networking capabilities. The entire framework is hosted on a server and users simply visit a website to immediately begin collaborating. Immediacy is an important goal for the Lich.js design because it makes iteration and code dissemination incredibly simple for the ensemble. Juggling code bases can be difficult even for experienced ensembles, especially those using mixed operating systems. Lich.js is supported on Windows, Mac OSX, Linux, and can even be run on mobile devices such as tablets and phones.

A functional paradigm was chosen because it allows for highly modular and terse programs, two features that are useful when writing and using code in a real-time performance. Like Haskell, Lich.js enforces immutability of objects. This means that variables can not be rebound or mutated, including containers such as lists and dictionaries. By disallowing object mutation it becomes easier to reason about the behavior of a program, allowing for chains of pure functions to build up complex, but more predictable programs than imperative designs. This becomes especially important in networked coding contexts where collisions between code bases can easily cause unforeseen consequences.

Being a live coding language, it would be rather useless if there wasn't some way to allow for side effects in the system and Lich.js provides a few well defined ways to allow for these effects. The easiest is to use the interactive mode, which is the default mode when visiting a Lich.js website. In interactive mode, commands can be executed and global scope variables can be rebound by hand using a "let myVar = myValue" syntax similar to GHCI for Haskell. Functions or any other language construct still have no ability to change objects and so any mutation is directly requested by the user and only at global scope. For live coding this is very useful because changes to synth definitions and patterns are defined in the global scope. The user can now rely on the vast majority of their programs being created with pure functions. The other mode for writing code is the library mode which is similar to a basic Haskell file. Global variables don't require use of the "let" syntax but can only be defined once and never changed as well as anything else. Predefined user code can be compiled ahead of time with useful functions such as chord progressions, section changes, useful data structures, and so on, with the guarantees of immutability. In interactive mode, library files can be imported and called like any other system code.

For more complicated uses of state, Lich implements the State Monad as found in Haskell. The State Monad is a kind of wrapper that allows for state mutation to be emulated using a chain of pure computations. Regarding the type system, Lich.js is not a strict clone of Haskell, and eschews a strong type system . For this reason some semantics are more similar to Erlang or Scheme because of the ability to utilize mixed lists or other structures. This does introduce many more opportunities for confusion and unpredictability, offsetting some of the efforts of the pure functional design. Dynamic typing was chosen simply because it gives the user less to worry about while coding in the moment, which is useful for live coding, as evidenced by most of the other live coding languages using a similar type system. Additionally, enforcing type semantics would add another layer of computation during compilation which could be potentially introduce more audio glitches during performance.

3. SYNTHS AND PATTERNS

Web Audio is a new API that is still being implemented in major browsers. Chrome and Safari provide full support and Firefox has recently enabled it in their browser, although it is a less mature implementation. The API provides a relatively low-level interface for creating real-time synthesis in pure JavaScript. The metaphor of a mixing console with effects channels is present throughout, lending itself well to use in web applications and games. Unfortunately the API is unwieldy for live coding because it is verbose and lacks many higher level features that you can find in other audio programming languages. For these reasons Lich.js compiles user created synth definitions into

pure JavaScript code, generating the necessary node connections and managing unit generator life times. Currently the WebAudio API defines several oscillators (sine, triangle, saw, and square wave), filters (lowpass, highpass, bandpass, notch, and several shelf variants), a waveshaper, a limiter, buffer playback, convolution, mixing nodes, and FFT analysis. This is a good basis for any audio application, but pales in comparison to many other audio engines and as such requires augmentation. Lich.js currently supports all of the supplied audio types except for FFT analysis. Furthermore it includes a wide array of custom unit generators including noise generators, filters, distortion effects, reverb, bit crushing, decimation, buffer manipulation, envelopes, and a frequency shifter.

Lich.js also implements a robust synth definition syntax that allows for language constructs, such as pattern matching or conditional branching, to be used directly. This is possible because these synth definitions compile down to a graph of audio node connections in JavaScript and reference values in the same language. This contrasts with a language such as SuperCollider where synth definitions are walled off from many language semantics because the target language for audio processing is different. It should be noted that higher level language semantics in Lich.js such as conditionals won't run at audio rate in a synth definition, but instead will define the particular connections that are created inside the node graph upon compilation. One unique feature of the synth definition is the use of the >> operator. This operator simply takes the left operand and applies it to the right operand. The syntax is simple but would not be easily implemented in other languages that don't utilize partial application. For this reason Lich.js synth definitions have a signal path like flow that declaratively describe transformations.

```
-- Synth definitions use the => operator
let b => tri 80 >> lowpass 320 1 >> perc 0 1 0.3
let s => white 1 >> bandpass 900 3 >> perc 0 1 0.2
-- Impulse patterns use a layout based syntax
drums +> b s [b b] s
let leadSyn f => sin f >> delay 1 0.1 >> perc 1 0.1 1
-- Solo patterns sequence over argument values
lead ~> leadSyn 440 [660 990] 330 _ 220 _ 110 55
```

Figure 1: Some expressions for synth and pattern generation.

By using the *play* function, running synths can be generated from synth definitions, but using the generative pattern sequencers in Lich.js allows for more powerful control. Currently there are two different syntaxes for generating patterns in Lich.js. The first is impulse based patterns, such as a drum sequence, and is invoked using the +> operator. Impulse patterns define nested rhythms using variables and lists without the need for commas and with rests marked by the Haskell wildcard _ symbol. Rhythm duration modifiers can be added to the pattern to generate more complex expressions with a small amount of code. The other kind of pattern generator is the solo pattern invoked using the ~> operator. Unlike impulse patterns, solo patterns only take a single synth variable and afterwards are supplied a sequence of values to call that synth with. This is useful for sequencing bass lines, melodies, or any function that requires an argument. Like impulse patterns, modifiers can be added to the sequencer, but here they modify the values passed to synth instead of durations.

3.1 Scheduling

The scheduler in Lich.js has been greatly improved upon since the original implementation. JavaScript timers can have a variation of tens to hundreds of milliseconds which is not reliable enough for an audio sequencer, and for this reason the original scheduling in Lich.js was uneven and inconsistent. In a massive revision the scheduler was rewritten to use a technique found in several other web audio applications [24]. The most important part of the technique is to recognize that Web Audio implementations are run in a different thread from the main user view and the Web Audio scheduler is highly accurate. The problem is finding how to leverage the accuracy of the Web Audio timer without having the standard JavaScript timers mar the fidelity. The solution is to write an event scheduler with a look ahead time. Because Web Audio Objects have an explicit start and stop time, if you can schedule events far enough ahead of time (100ms or so should suffice) then even if your scheduler has variance, the actual web audio event will still occur accurately. Furthermore the scheduler is more robust and can withstand higher variance in CPU activity from other parts of the program.

4. GRAPHICS

Most of the graphics functionality in Lich.js is built upon the THREE.js [3] library which is a WebGL based JavaScript library for 3D graphics. Although WebGL has been supported longer than Web Audio, the API is very low level, requiring tremendous effort to make something usable, let alone live coding. Pragmatism was the deciding factor for using THREE.js. Writing a new language, audio system, and client/server architecture were all tremendous tasks, but necessary to materialize the unique features required in Lich.js. WebGL on the other hand is older than Web Audio and THREE.js has well established itself as a stable library that implements most features that would be needed in a graphics based application. While THREE.js abstracts away most of the difficulties of OpenGL, it still requires more code than is feasible to write in real time. For this reason most of the graphics work in Lich.js has been to take the implementation of THREE.js and build on top of it a collection of higher level functions and algorithms for live coding visuals.

Lich has several functions for 3D mesh generation from basic shapes such as spheres and boxes to complex generative algorithms. The functions take position and color data in addition to any other values they require, creating the mesh and directly drawing it in the scene. All meshes can have their position, rotation, angular and linear momentum, color, and scale manipulated in real-time. Furthermore, because the pattern sequencers described in the previous section support more than just audio functions, you can compose a definition that manipulates a mesh instead of audio. Beyond mesh manipulation, Lich.js has some preliminary code for shader generation using a mini-language called Splice. Splice is a very basic language that takes a string and translates every character into a GLSL function, then wraps up the result with some boiler plate code to create a new GLSL shader. The result is often glitchy with some surprising behavior. Passing native GLSL code directly is supported, but writing pure GLSL in real-time is cumbersome. At the moment there is no 2D Graphics API or support for textures. These are among several features that will be added in upcoming versions of the language.

5. NETWORKING

Networking was the greatest motivating factor for creating Lich.js with a web based implementation. The networking features currently supported are simple, but the focus has been to make the project stable and usable first, before

adding more features. Networking in Lich.js uses a simple client and server architecture with the server written in Node.js [7]. There are three main networking features in the language: the networked IDE, chat based communication, and shared code execution. The Lich.js IDE is based on the ACE HTML5 code editor which supports syntax highlighting, matching bracket highlighting, and customizable keyboard bindings. The work on the IDE so far has served to create a completely automated user management experience. When a user visits a Lich.js site they are greeted with a request for a user name. The server receives the name and stores a cookie on the client computer so they don't have to reenter this information in the future. There is currently no password system so this is the extent of logging on. Once into the main Lich.js page they are presented with a code editor and a post view on the bottom. If another user joins the same site the code editing view is automatically split in half using a smooth transition animation, and now both users can see other's code. Originally the code editor was a single document, but this caused significant issues with unexpected code deletions and collisions. The current method affords many of the same features of the shared editor, namely the visibility of the code and efforts of the other users, but the networked code editors are read-only for everyone but the owner.

A chat feature was added to the system because even though the post window could be used as a chat using print functions, it was easy to lose messages to system print out and was too small for an audience to read. The new chat feature prints chat messages very large and right justified over the entire window. The messages fade in and out and only last a few seconds. They are legible and invite attention without being too distracting. Finally, code is executed across the network. Whenever any user executes a function that function sent is across the network and executed for each client. While currently not implemented, it would be useful to have a more sophisticated implementation that uses time stamps with pre-calculated latency to execute changes synchronously across the system. Another concern is that while it is very simple to start using Lich, if an interested user wanted to create their own server, the procedure is more complicated. It requires the installation of Node.js and using some command line tools. A more elegant method needs to be developed to make independent servers easier to create for newer users.

6. CONCLUSIONS AND FUTURE WORK

In this paper I have argued that the continued difficulty of networking has stifled the creation and productivity of network bands and ensembles; that there is a need for a simpler approach for users of all experience and skill levels. To accomplish this goal I have an introduced Lich.js, a new live coding language in the burgeoning field of web based languages. After surveying the current alternatives a justification was made for the need of yet another language; one based on collaboration from the beginning. Following the introduction, Lich.js was covered in detail, including the language syntax and semantics, synth definitions and pattern generation, graphics, and networking. Lich.js is open source and available on GitHub [13] but is still in development. The official release of Lich.js will align with NIME 2014. Leading up to the official release the project will see a concerted effort to minimize bugs and increase performance. New features are also being considered such as local area network clock synchronization, time stamped code execution, Open Sound Control responder implementation, and more graphics and audio functionality.

7. REFERENCES

[1] R. Bencina. OscGroups, 2010. http://www.audiomulch.com/~rossb/code/oscgroups/.

[2] J. Brodsky. Jsaxus, 2013. https://github.com/jonbro/jsaxus.

[3] R. Cabello. Three.js, 2010. http://threejs.org/.

[4] R. Cabello. Glsl sandbox, 2013. http://mrdoob.com/139/GLSL_Sandbox.

[5] Z. Carter. Jison, 2014. http://zaach.github.io/jison/docs/.

[6] D. D. Casa, S. McDonald, J. Stutters, and T. C. Ryan. Livecodlab, 2013. http://www.sketchpatch.net/livecodelab/index.html.

[7] R. L. Dahl. Node.js, 2009. http://nodejs.org/.

[8] G. Florit. livecoding.io, 2013. http://livecoding.io.

[9] I. Ivanoff and J. Jimenez. Flaxus, 2006. http://www.i2off.org/flaxus/screen.html.

[10] J. Rohruber, A. de Campo. The republic quark, 2011. https://github.com/supercollider-quarks/Republic.

[11] Khronos Group. Webgl - opengl es 2.0 for the web, 2013. http://www.khronos.org/webgl/.

[12] T. Magnusson. The ixi lang: A supercollider parasite for live coding. In *"Proceedings of the International Computer Music Conference"*, pages 198–200, 2011.

[13] C. McKinney. Lich.js, 2014. https://github.com/ChadMcKinney/Lich.js.

[14] A. McLean. *Artist-Programmers and Programming Languages for the Arts*. PhD thesis, Department of Computing, Goldsmiths, University of London, October 2011.

[15] F. Obermeyer. Livecoder, 2013. http://livecoder.net.

[16] Patrick Borgeat, Holger Ballweg, and Juan A. Romero. Benoitlib, 2012. https://github.com/cappelnord/BenoitLib.

[17] C. Roberts. The web browser as synthesizer and interface. 2013.

[18] C. Roberts. Gibber 2.0, 2014. http://charlie-roberts.com/gibber/info/.

[19] C. Rogers. Web audio api: W3c working draft, 2013. http://www.w3.org/TR/webaudio/.

[20] J. Rohrhuber, A. de Campo, R. Wieser, J.-K. van Kampen, E. Ho, and H. Hölzl. Purloined letters and distributed persons. In *Proceedings for the Music in the Global Village Conference*, 2007.

[21] K. Samp. Webgl Playground, 2013. http://webglplayground.net/.

[22] G. Wang, A. Misra, and P. R. Cook. Building Collaborative Graphical Interfaces in the Audicle. In *NIME '06: Proceedings of the 2006 conference on New interfaces for musical expression*, pages 49–52, Paris, France, France, 2006. IRCAM — Centre Pompidou.

[23] A. Ward, J. Rohrhuber, F. Olofsson, A. McLean, D. Griffiths, N. Collins, and A. Alexander. Live Algorithm Programming and a Temporary Organisation for its Promotion. In O. Goriunova and A. Shulgin, editors, *read_me — Software Art and Cultures*, 2004.

[24] C. Wilson. A tale of two clocks, 2013. http://www.html5rocks.com/en/tutorials/audio/scheduling/.

[25] M. Wright, 2002. Open sound control 1.0 specification. http://opensoundcontrol.org/spec-1_0.

[26] N. Yonamine. Coffeecollider, 2013. http://mohayonao.github.io/CoffeeCollider/.