Live Writing: Asynchronous Playback of Live Coding and Writing

Sang Won Lee
University of Michigan, Ann Arbor
snaglee@umich.edu

Georg Essl
University of Michigan, Ann Arbor
gessl@umich.edu

ABSTRACT

We introduce Live Writing, asynchronous playback of a live coding performance or, more generally, writing. The concept of Live Writing is realized in a web-based application which logs every keystroke that a writer makes and let the writing later playable by the audience or the readers in real-time. The goal of Live Writing is twofold. One, it aims to support collaboration between musicians by reproducing a live coding performance based on keystroke data logged in the platform. This offers a new way for a live coder to archive live coding music and to communicate with others in asynchronous fashion. Two, it aims to transform written communication into a real-time experience so that a writer can display a writing to readers in a real-time manner as if it is being typed in front of the readers. We explain the design and the implementation of the system and demonstrate two different use cases of the system: live coding and writing.

1. Introduction

Live coding music is a type of written communication between human and machine expressed in programming language. At the same time, the algorithmic representation of music is a window into a composer's thoughts. Therefore, in the case of collaborative live coding, code sharing is an important part of communication, understanding each other's thoughts, and exchanging ideas. Most of the times, the collaboration in this live coding practice occurs in real time, due to its live nature and asynchronous communication being limited. However, being able to archive and replay live performances can further expand and enable important aspects of live coding performance culture. Benefits include the ability to (1) collaborate asynchronously, (2) incorporate last live performances into reinterpretation, reappropriation, and the creation of a persistent performance corpus. Static code is an core outcome of expressing compositional ideas that arise throughout the preparation process of a live coding performance. Nonetheless, program code itself does not represent the entire progress of the music performance. Static code does not contains temporal information of code execution nor captures a performer's virtuosity, such as skills and techniques to utilize existing code quickly enough to make changes in the music in a timely manner. In addition, some code written on the fly may not even exist in the final code that remains after the performance. While the screen recording of a live coder's play may capture these kinds of real-time gestures, it is fixed media in a non-symbolic format. Hence, it is impossible to get textual data from the recording so as to copy and transform, for example, someone else's code. This calls for representations that that capture the dynamic and evolving character of live code production in a format that is readily modifiable. To this end, we developed the Live Writing platform. The Live Writing platform logs all keystrokes made by a live coder with timestamp information in order to reproduce the musician's play later. The system can be used to archive any live coding performance ranging from individual practice to actual performances in public. The term "asynchronous" collaboration in "live" coding sounds contradictory. However, it seems potentially exciting to envision a live coding system such as one that will facilitate asynchronous collaboration. In other words, how do we utilize technologies to help a musician communicate with other musicians without needing to be present at the same time? As an extreme example, how could a live coder reproduce a piece that has been written by a live coder who has since passed away. Naturally, this poses two questions: how to notate a live coding piece and how to archive a live coder's playing.

Our second goal with Live Writing is to transform, as the name suggests, general written communication into a real time experience. Writing is rich form of communication and we live in an age when we produce large volumes of writing through digital platforms such as the World Wide Web, and mobile devices. The Live Writing platform lets writers (not necessarily just live coders and artists but anyone) present their writing to readers in the same way that it was typed at the time of creation. The real-time rendering of writing gives much more expressive power to the writer and gives readers the intimacy of peeking at someone's computer screen in private. The core idea of Live Writing is to reveal the incomplete stages of writing and of the textual dynamic as part of the written expression. For a quick demo of Live Writing, visit http://www.echobin.com/?aid=aboutechobin. A screenshot of Live Writing platform is shown in Figure 1.

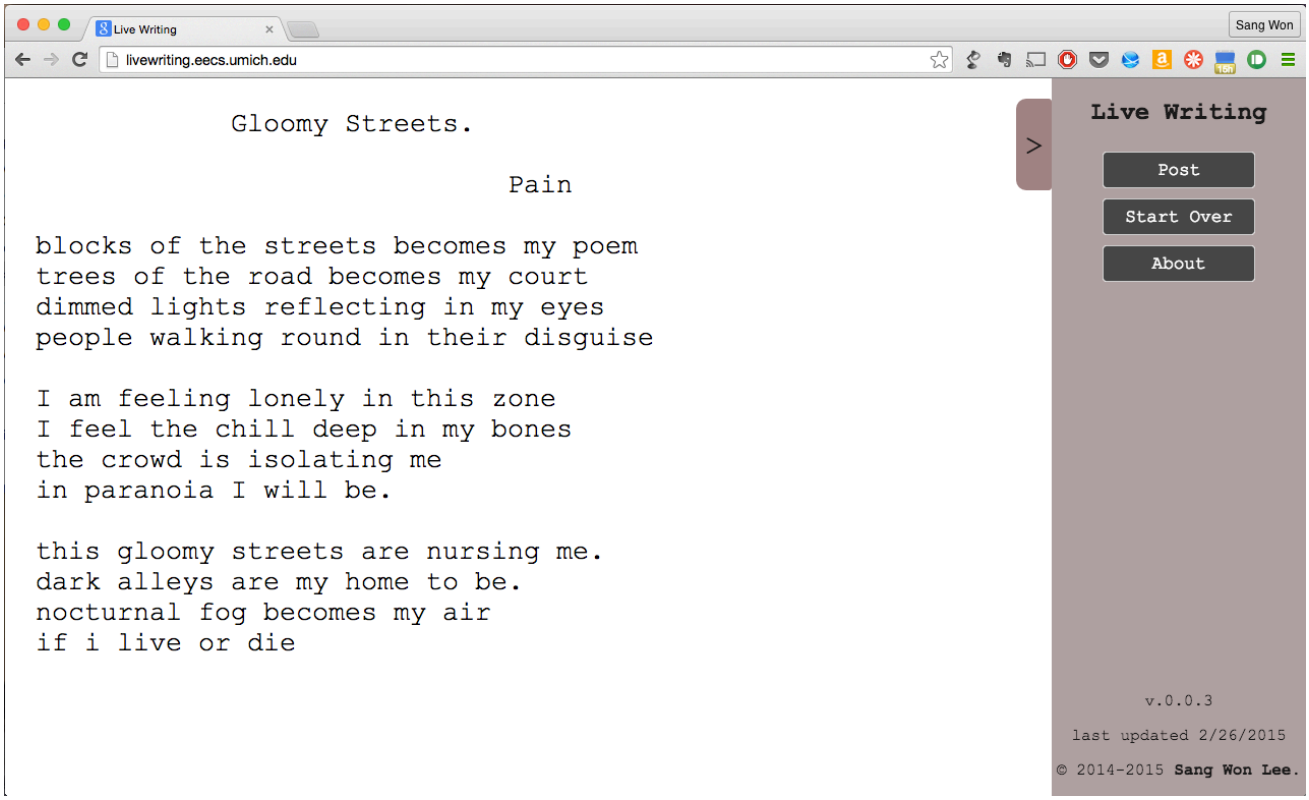

Figure 1: *a screenshot of Live Writing example. The poem written by Pain.*

In this paper, we introduce the background upon which the idea is built. We further motivate the core ideas, and describe the current implementation. Demonstrated on the web browser are examples in the context of live coding as well as writing. Finally, we conclude with future research opportunities offered by the system.

2. Asynchronous Collaboration in Live Coding

2.1. Live Coding as Network Music

Networked communication and data sharing can facilitate collaboration among live coders. The potential of networked collaboration in live coding has been present from the inception of live coding (Collins et al. 2003). *Power-Books_UnPlugged* exemplifies an interconnected network ensemble by sharing code over a wireless network among live coders and enabling sound dislocated from the code (Rohrhuber et al. 2007). Similarly, *aa-cell* added network capability to Impromptu to facilitate collaboration over the local network for sharing and executing code remotely (Brown and Sorensen 2007). The scale of collaboration in live coding has reached that of an ensemble (Wilson et al. 2014) and a laptop orchestra (Ogborn 2014c). Recently, the authors made an extension to *urMus* (Essl 2010), a programming environment for mobile music, to support multiple live coders, addressing issues that emerged for collaborative live coding in general (Lee and Essl 2014a).

In our previous work, we applied existing frameworks coming from the tradition of network music to the previous works of networked collaboration in live coding (Lee and Essl 2014b). For example, we used Barbosa's framework to classify computer-supported collaborative music based on synchronism (synchronous / asynchronous) and tele-presence (local / remote), as shown in the Figure 2 (Barbosa 2003). This is used to learn opportunities in live coding and to identify relatively underdeveloped areas in the classification. Most networked live coding performances fall into the local/synchronous category, where live coders are co-located and play at the same time. Recently, researchers explored remote/synchronous collaboration in live coding music: networked live coding at Daghstuhl Seminar (Swift, Gardner, and Sorensen 2014), *extramous*, language-neutral shared-buffer networked live coding system (Ogborn 2014d), and *Gibber*, a live coding environment on a web browser, which supports remote code edit and run (similar to Google Doc) (Roberts and Kuchera-Morin 2012). However, most previous live coding ensembles have focused on collaboration in synchronous fashion (the lower half of the chart in Figure 2).

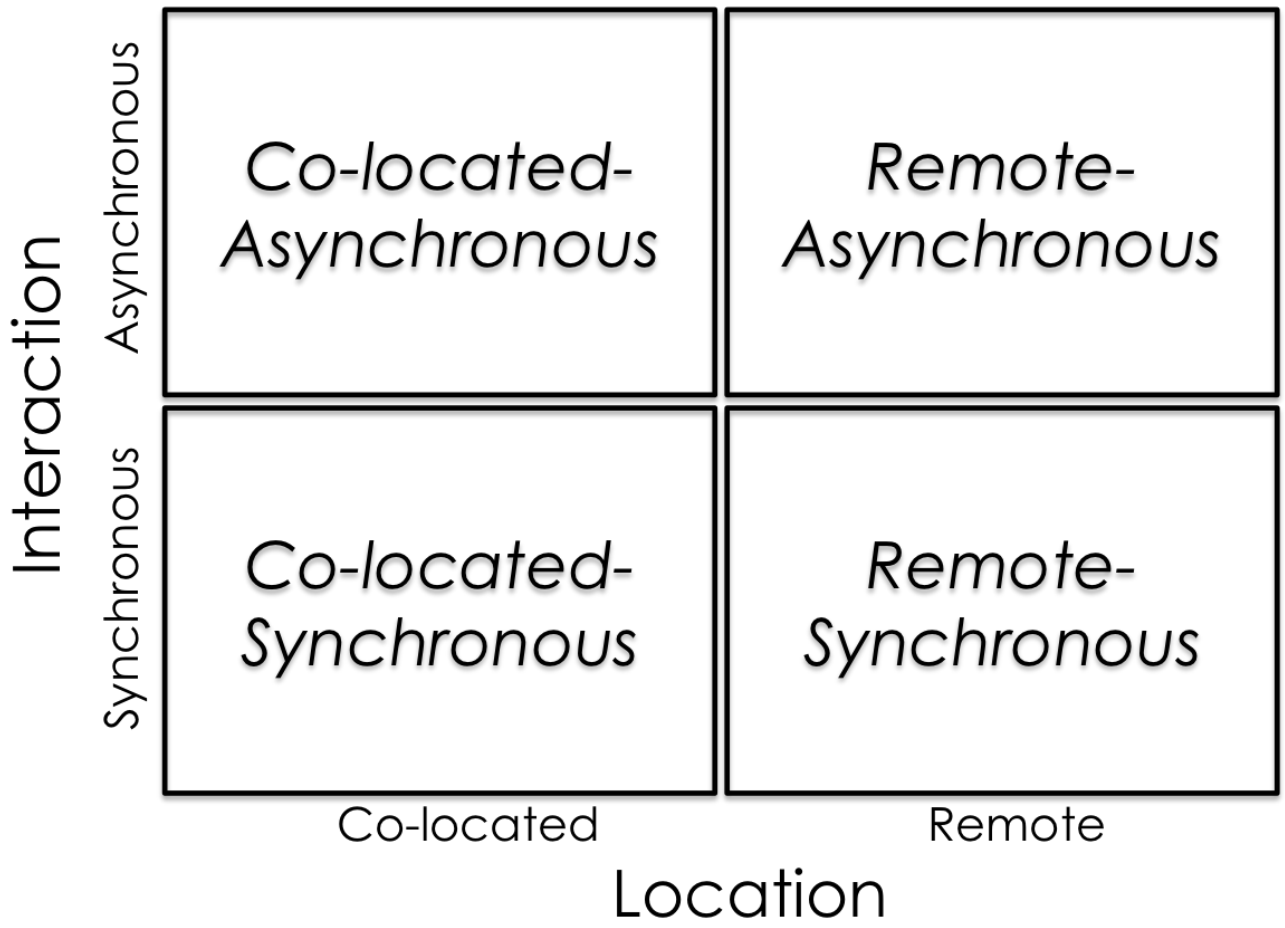

Figure 2: *Classification Space for Network Music. Adapted after (Barbosa 2003).*

2.2. Music Notation in Live Coding.

In the traditional sense of music notation, where the objective is for a composer to communicate with performers in the future, the music notation for live coding is not well-established for discussion. Traditional music notation is certainly not well fitted for notating live-coded music. This is because live coding is a highly improvisational and interactive form of art where composition occurs at the time of performance; in live coding there is no clear distinction between composer and performer. Rather, a live coder is "composer-performer."

Indeed, the discussion of notation in live coding has been focused on its real-time usage at the performance. In this context of real-time composition/performance, the code (either textual or graphical) serves as a music score and diverse scoring approaches as code representation have been attempted (Magnusson 2011). Graphical visualization of the code is effective for better audience communication (McLean et al. 2010). In addition, the real-time music notation can be a medium that connects live coding musicians and instrumental performers (Lee and Freeman 2013; Hall 2014).

2.3. Archiving a live coding performance.

It seems that, for the time being, the music notation for live coding music with the traditional goal of documenting a piece may not need to reach a single conclusion. Instead, live coders should be able to archive live coding performances for the following benefits: i) an archive of a live coding performance can be used to exchange ideas and data with collaborators, ii) the collection of live coding pieces can be fruitful, from a long-term perspective, for the live coding community as a way to transfer knowledge, providing inspiration and developing the field for posterity.

One immediate solution to archive a live coding piece is to save, as electronic files, the program code that was used in the performance. However, the code alone is not enough for a live coder to document a music piece in a way that someone in the future (including oneself) would be able to repeat the performance. Static code itself cannot reproduce a live coding piece because it does not contain the performance aspects of algorithms execution. Furthermore, a certain amount of virtuosity in live coding is involved with real-time manipulation of code, which is not well exposed in the final text of the code. Practitioners use static code in combination with other tools such as audio recording, videotaping, screen recording, or open form score (e.g., textual or graphical notes) as part of the documentation in rehearsal, composition, and preparation steps of a performance.

Archiving a live coder's performance will have more potential in the scenario of a live coding ensemble to support asynchronous communication among the members. At the moment, the most obvious collaboration strategy that all live coding ensembles take is a combination of rehearsals and individual practice. However, it will be practically challenging to find the time (and the place) that every member of the ensemble can participate. The problem will worsen in scale, such as in the case of a live coding orchestra (Ogborn 2014c). Alternatively, each musician can do at-home practice to create new ideas for the ensemble and the individual. However, communication is delayed until the next gathering. To that end, a live coding ensemble will benefit from developing a new format of archiving a music performance, which will eventually support non real-time collaboration among members.

Lastly, archiving a live coding performance will enhance the composition process for a solo live coder. A live coder can log each composition session to document the progress; the archive will help the composer capture new ideas, which appear of a sudden and go away during the exploration and the improvisation.

2.4. Live Writing: Documenting and Replaying Textual Dynamics of Live Coding

The live coding piece performed on Live Writing platform will be logged in the keystroke levels with timestamp information so that the generative music can be reproduced by the typing simulation. Given it is sequenced data rather than fixed media, this differs from audio/video recording, and it is different from the music notation, as it can be used to capture a particular performance. It provides symbolic information, which a musician can easily copy, duplicate, and make changes to. The log file generated from Live Writing platform is analogous to a MIDI sequence file, which describes musical dynamics as an intermediate often input-centric representation. Live Writing serves a similar function in terms of capturing input-centric dynamics of text entry. Figure 3 shows the analogy between traditional music and live coding in terms of ways to archive a music performance.

The highlight of Live Writing platform is that it provides temporal information of the live coding piece that static code cannot. Replaying the code of a piece can inform the order of code written, the timing of execution, and the virtuosic moves of a live coder, elements that would have been impossible to know in the final code. The playback feature could help transfer practical knowledge of carrying out the piece and help people practice along with collaborator's simulated performance. The idea of Live Writing draws upon existing systems that enable asynchronous displays of code. In

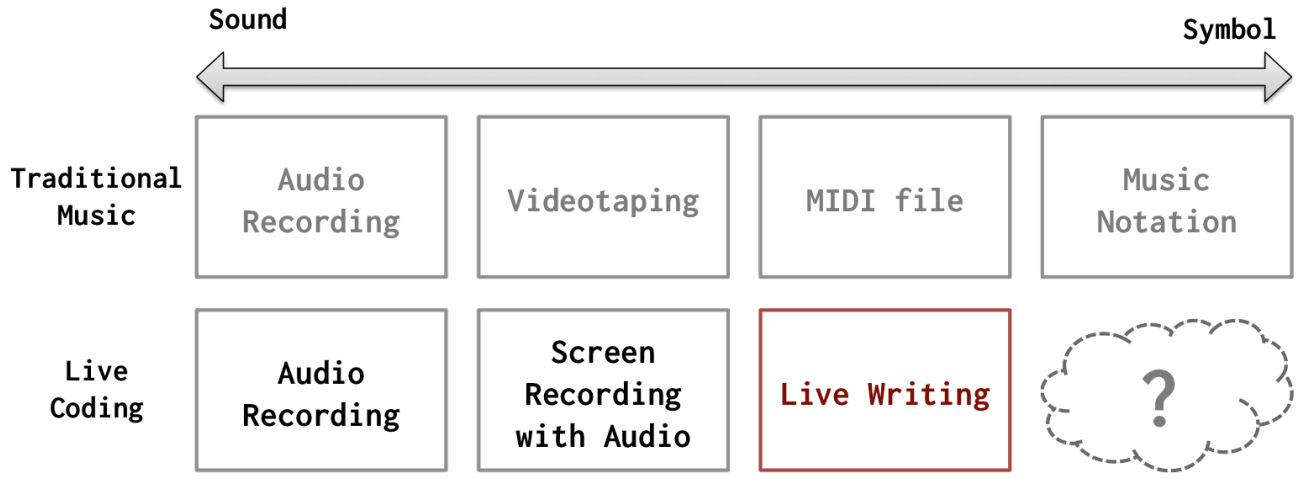

Figure 3: *The methods to archive music performances in traditional music and live coding.*

particular, Live Writing platform is directly inspired by Ogborn (2014e)'s performance using *Daem0n.sc* system, which types the text of a file automatically—in sync with the default metronome in Supercollider. *Gibber* provides a playground in which people can publish and browse *Gibber* code so that the hub supports asynchronous collaboration in a broad sense within the community (Roberts et al. 2014). In *Threnoscope*, Magnusson (2014) introduced code snippets in a piano-roll-like notation where it is used as a secondary composition/performance interface, potentially a great archive format to document a performance on the system. In non-artistic cases, Google Wave, which is no longer available, contained the playback feature at the email transaction level so that one can understand the temporal order of email messages especially for those who got included in the conversation later. Lastly, thecodeplayer.com implements keystroke level playback of writing code but it is not currently accessible to the public to create contents("Thecodeplayer.Com").

2.5. Beyond Archive: New Media for Collaborative Live Coding

The symbolic representation of the archived performance offers novel opportunities for asynchronous collaboration among musicians. A live coder can access the code generated on the fly easily while a screencast or the final static code cannot offer such data, i.e., copy the code in the middle of the playback simulation and paste it on one's editor to use existing code to develop ideas, and to create collaborative musical patterns. It can be used throughout the rehearsal process for a live coding ensemble to sketch ideas and exchange code over emails, and to rehearse along with the ensemble members asynchronously.

Keystroke log is a valuable asset for monitoring/analyzing a live coder's performance. It is not only great for self-reflection on one's rehearsal/performance but also a useful material to analyze a live coder's performance. The performance log can be used to analyze live coding musicians' play and learn complex behaviors that are not translated easily from audio, static code and the artist's statement, for example, the virtuosity of a performer in programming and musical style. Such research has been done to create a performer's fingerprint (Swift et al. 2014) and would have been expedited and enriched with the corpus of logged data accumulated in the system, otherwise, such data should be manually annotated from the screencast.

Lastly, supplementing visualization/navigation techniques to be implemented in the future (the navigation bar, keystroke density visualization, temporal typography) will offer an interactivity in the middle of playback of live coding. For example, a reader can pause, modify, fork or remix a performance and create a unique live coding collaboration that can occur over a long time. Forthcoming features of the Live Writing system will be discussed in detail at the end of this paper.

3. Written Communication into a Real Time Performance

In this section, we switch the context of discussion from live coding music to writing in general. First, imagine a performer going on stage and starting to write a poem (whether it is typed on a screen or written on a piece of paper, shared with the audience). The live coding community would easily call this a live writing performance and the performer would need nothing more than sheets of paper and a pen or a text editor on a laptop projected on a screen. Live Writing platform

allows users to dynamically present a writing; readers can read the article as if it is being typed right in front of them. It does not necessarily need to be a synchronous performance in public but anything written in private (for example, poem, essay, instant messages, email, or tweets) can be presented as if readers were watching the writer typing in real time. The same platform that was used to archive live coding performances is repurposed for this new expressive art.

3.1. Keystroke Logging and Playback of Writing

The realization of Live Writing platform is to utilize keystroke logging on a computer. For a long time now, the idea of logging user input from a keyboard has been a simple but powerful research method in many different contexts. *Keystroke Level Model* (KLM) was an important aspect of research into modeling human performance in many human-computer interaction tasks (Card, Moran, and Newell 1980). It was also used to analyze email organization strategy (Bälter 2000). Keystroke dynamic has also been used for verifying identities. Habitual temporal patterns of key entries provide a behavioral biometric useful for identifying individuals (Joyce and Gupta 1990; Monrose and Rubin 2000).

In helping analyze real-time writing behavior, keystroke logging has become a common approach in the field of writing research. It provides a non-intrusive and inexpensive technique to monitor user inputs. Writing researchers have developed a number of keystroke logging applications—*Inputlog*(Leijten and Van Waes 2006) and *ScriptLog*(Strömqvist et al. 2006) are two examples of such programs. Most keystroke logging applications include real-time playback recorded keystrokes and it is an effective approach to help subjects account for their writing in retrospect, which is less intrusive than having them think aloud while writing (Latif 2008). These applications are mainly for the research purpose of real-time writing analysis rather than for writing in general public.

3.2. Live Writing : Writing as Real-Time Performance

The direct inspiration of the notion of live writing is live coding. Writing is an expressive process guided and adapted by thoughts that evolve over time. By revealing the real-time writing process to readers, Live Writing lets a writer show the thought process and the intermediate stages of the writing as it is typed, including typos, corrections, and deletions. This principle is well captured in the following statement of the *TOPLAP* manifesto: "Obscurantism is dangerous. Show us your screens." We expand the same idea to writing in general. Showing the entire process of writing as it emerges sheds light on the trajectory of thoughts and provides more expressive power than when reading final text. It is analogous to revealing the thought process of a composer as a form of algorithm in live coding music.

The real-time rendering of writing in the same way that it was typed is certainly more expressive than static writing. There are various kinds of writer's emotional states (such as contemplation, hesitation, confidence, or agitation) that can emerge during the process of typing based on temporal patterns, for instance, pause, bursts, or corrective steps. The fact that every single entry is shown in the final outcome transforms writing in general into a creative experience in real-time, similar to musical improvisation. Indeed, such an improvisational nature is prominent on the World Wide Web, for example, vlogging, podcasting, live game streaming, and video tutorials with screen recording and voice over.

It may be argued that calling this "Live" Writing is somewhat misleading as the Live Writing platform enables asynchronous (that is apparently not "live") uses. However, we hold that the very process of writing is the live act in this case, and this liveness of the writing process is captured, hence justifying the use of "live" in this context. Moreover, by explicitly capturing the the temporal dynamic of the writing process it suggests a changed mindset in the writer with respect to the meaning and performative function of the writing process itself. A writer can now ponder the arrangement of text over time and leverage the articulation of the temporal dynamics (like a composer organizing sound to make music). This can be premeditated as in compositions, or improvised. Further it suggests a deliberation of the delivery process even if the material is fixed, leading to an expressive layer of interpretation that transcends yet potentially interrelates with the text. One can see this analogous to the notion of a "live" recording. The production process is live, yet the reproduction method is not. This does by no means preclude the consideration of live writing in a fully real-time synchronous performance setting. In fact we are currently exploring synchronous live writing piece as a form of audiovisual performing arts, of which a visualization technique on typography is discussed elsewhere (Lee and Essl 2015).

4. Design and Implementation

Live Writing is implemented as a web application in order to be easily accessible from a variety of platforms and devices. The web browser is chosen to make the platform language neutral (as long as the live coding language is textual). In addition, the web browser is one of the most popular writing platforms today. Many live coding language environments

allows the editing part to be separated from the local machine and to reside on the web browser or web-based editors (such as Atom, Sublime Text, or Brackets I/O). This would work, of course, for live coding languages built into the web browser. We realize this system cannot support many live coders who use other popular editors (e.g. emacs). However, we chose the web browser given it is a good platform-independent application and the use of web browser is on the increase in live coding. Eventually, modern web-based editing APIs such as CodeMirror and ACE have been and will be evolving to support many functions like shortcuts and macro to replace such editors or the Live Writing feature can be easily implemented in across multiple editing environments if needed. It should be addressed that the system will be limited to textual live coding languages but the idea can be extended to graphical languages.

The API is designed to extend existing an html object <textarea> or code editing APIs (e.g. codemirror, ace) so that it can easily extend existing systems on the web browsers. In the following section, the implementation of the application for the writing purpose is described first. Demonstrated later in this section is an archiving and playback example built on top of *Gibber*.

4.1. Live Writing App for Written Communication

For the writing purpose of Live Writing, the application is publically available for anyone to use http://www.echobin.com/. The current implementation is crafted with minimal design; it has only one clean slate, an initial dialog with brief description, screen-sized <textarea> objects, and a few buttons hidden on the side (see Figure 1). Once a user presses the start button, all keystrokes and mouse cursor clicks made inside the <textarea> will be recorded on the local machine. And then, by pressing the post button on the right side, the user can post the piece of writing. Once the data is posted, the user is given the link that accesses the real-time playback of the writing. The link contains the article ID and anyone with the link can view the playback of the writing. Therefore, access control of the writing is up to the writer and whom he or she chooses to share the link with, an action that can be easily done via email, a facebook post, or instant messages.

The web application is realized in javascript/jQuery/AJAX and node.js. We released the code as an open-source API so that any html <textarea> can be extended to feature keystroke logging and playback by including one javascript file. The server runs node.js script which handles the static files and store/retrieve recorded keystrokes log in json format. All the other functions of keystroke logging and playback is implemented and run in the client machine which the writer uses. The logged data is not stored in the server until the user chooses to post the writing. Alternatively, a user can choose to download the log file instead of posting to the server, if the user wants to keep the logged data locally. Anyone who has the log file will be able, by uploading the file, to replay the writing. Providing raw data in the file will be useful for extracting high-level information such as keystroke dynamics or behavioral pattern when processed. We hope that this supports researchers wanting to use this as a keystroke-logging application.

To implement playback, the keystrokes logged are simulated by specifically reproducing what the keystroke would change in the content of <textarea>. In other words, a web browser does not allow, for security reasons, javascript to generate the exact same keystroke event. Instead, for example, to reproduce a user's pressing of the "s" key, it has to append the "s" character to where the cursor is in the <textarea> at the moment. Note that the <textarea> used in keystroke logging is again used for playback. Improvements to the website are in progress so that a user can customize the writing not only visually (font type, size etc.) but also through its playback setting (e.g.,, playback speed, navigation across time, etc.).

4.2. Playback of live coding : *Gibber* code on *codemirror*

To demonstrate the archiving and replaying features of Live Writing, this study has chosen *Gibber* (Roberts and Kuchera-Morin 2012), a live coding environment on a web browser. It could have been possible to create the same features with other live coding languages (e.g., Supercollider) that can be edited on a web browser and communicated with the live coding engine by sending textual data to the localhost. Because *Gibber* is built into the web browser, it makes the demo accessible to public without any configuration and installation. Lastly, for the code-editing environment, the study uses *Codemirror* (Haverbeke 2011). Codemirror comes with a set of features readily available for programming (e.g., syntax highlighting, keymap binding, autocomplete).

The user experience in the *Gibber* demo is similar to that of the writing web application introduced above. In terms of implementation, however, a few differences from the writing scenario should be noted. This uses high-level events (onchange, cursor activity) supported from codemirror instead of low-level events (e.g., keyUp, keyDown, keyPress) in <textarea>. In addition, the pressing of shortcut keys related to code execution (Ctrl-Enter, Ctrl-.) is separately stored as a message so that simulated typing will trigger a code run to get audiovisual outcome while playback. The working demo is available at http://www.echobin.com/gibber.html. For a simple playback demo without entering Gibber code, visit http://www.echobin.com/gibber.html?aid=OKDMWHgkDCdAmA which shows the demo captured in Figure 4.

Figure 4: *Screen captures of Gibber Demo Audiovisual outcome generated automatically by playback of Gibber code over time (from left to right). This demo is available at http://www.echobin.com/gibber.html?aid=OKDMWHgkDCdAmA*

5. Conclusion

In this paper, we have introduced Live Writing, a web-based application which enables keystroke logging for real-time playback of the writing. The motivation for this idea is to facilitate asynchronous collaboration among live coders and to archive a live coding performance in an informative way. Additionally, the Live Writing platform is repurposed for general written communication and has potentially turned a writing activity into a live coding like performance.

There are a number of directions that we can take for future research. We plan to integrate Live Writing editor with existing live coding environments and evaluate the usage of features in practice. It would be desirable to investigate the system usage in a collaboration scenario like live coding ensemble. On the other hand, with the features of playback, we would be able to build a live coding gallery in which users publish their piece and people can enjoy the live coding piece remotely and asynchronously. In addition, we want to make the playback feature not interfere with key entry so that a spectator can write text while keystrokes are replayed. This will create unique opportunities in a crowd-sourced music piece that grows over time by people's participation in a system that is similar to github.

Another ongoing effort is to explore the idea of live writing and to deliver a live-writing performance and a participatory audiovisual art on a web browser. Currently, temporal typography based on a web browser is in development to augment the text editor with animated fonts (Lee and Essl 2015). It will afford novel ways of visualizing text in combination with music and algorithms, associated with the content of writing. To explore the idea of live-writing poetry, an audiovisual piece is in preparation for a public concert. In addition, visualization of the text will be used to visualize the program state as well as music in the context of live coding.

Lastly, the Live Writing application will be further developed for general programming and evaluated in terms of human-computer interaction. We believe that the playback feature of the editor will attract programmers to use the editor for numerous reasons (self-evaluation, fine-grained version control, online tutorial, collaboration, programming challenge/interviews). Improvement of the editor in progress includes the navigation bar, typing density visualization, and automated summary. The collection of source code published to the server can be used to mine valuable information (i.e., temporal progress of good/bad coding style) in the context of software engineering and pedagogy in computer science

References

Bälter, Olle. 2000. "Keystroke Level Analysis of Email Message Organization." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 105–112. ACM.

Barbosa, Álvaro. 2003. "Displaced Soundscapes: a Survey of Network Systems for Music and Sonic Art Creation." *Leonardo Music Journal* 13: 53–59.

Brown, Andrew R, and Andrew C Sorensen. 2007. "Aa-Cell in Practice: an Approach to Musical Live Coding." In *Proceedings of the International Computer Music Conference*, 292–299. International Computer Music Association.

Card, Stuart K, Thomas P Moran, and Allen Newell. 1980. "The Keystroke-Level Model for User Performance Time with Interactive Systems." *Communications of the ACM* 23 (7): 396–410.

Collins, Nick, Alex. McLean, Julian. Rohrhuber, and Adrian. Ward. 2003. "Live Coding in Laptop Performance." *Organised Sound* 8 (03): 321–330.

Essl, G. 2010. "UrMus – An Environment for Mobile Instrument Design and Performance." In *Proceedings of the International Computer Music Conference (ICMC)*. Stony Brooks/New York.

Hall, Tom. 2014. “Live Digital Notations for Collaborative Music Performance.” In *Proceedings of the Live Coding and Collaboration Symposium 2014*. Birmingham, United Kingdom.

Haverbeke, M. 2011. “Codemirror.” http://codemirror.net/.

Joyce, Rick, and Gopal Gupta. 1990. “Identity Authentication Based on Keystroke Latencies.” *Communications of the ACM* 33 (2): 168–176.

Latif, Muhammad M Abdel. 2008. “A State-of-the-Art Review of the Real-Time Computer-Aided Study of the Writing Process.” *IJES, International Journal of English Studies* 8 (1): 29–50.

Lee, Sang Won, and Georg Essl. 2014a. “Communication, Control, and State Sharing in Collaborative Live Coding.” In *Proceedings of New Interfaces for Musical Expression (NIME)*. London, United Kingdom.

———. 2014b. “Models and Opportunities for Networked Live Coding.” In *Proceedings of the Live Coding and Collaboration Symposium 2014*. Birmingham, United Kingdom.

———. 2015. “Web-Based Temporal Typography for Musical Expression and Performance.” In *Proceedings of New Interfaces for Musical Expression (NIME)*. Baton Rouge, United States.

Lee, Sang Won, and Jason Freeman. 2013. “Real-Time Music Notation in Mixed Laptop–Acoustic Ensembles.” *Computer Music Journal* 37 (4): 24–36.

Leijten, Mariëlle, and Luuk Van Waes. 2006. “Inputlog: New Perspectives on the Logging of on-Line Writing Processes in a Windows Environment.” *Studies in Writing* 18: 73.

Magnusson, Thor. 2011. “Algorithms as Scores: Coding Live Music.” *Leonardo Music Journal* 21: 19–23.

———. 2014. “Improvising with the Threnoscope: Integrating Code, Hardware, GUI, Network, and Graphic Scores.” In NIME.

McLean, A., D. Griffiths, N. Collins, and G. Wiggins. 2010. “Visualisation of Live Code.” *Proceedings of Electronic Visualisation and the Arts 2010*.

Monrose, Fabian, and Aviel D Rubin. 2000. “Keystroke Dynamics as a Biometric for Authentication.” *Future Generation Computer Systems* 16 (4): 351–359.

Ogborn, David. 2014c. “Live Coding in a Scalable, Participatory Laptop Orchestra.” *Computer Music Journal* 38 (1): 17–30.

———. 2014d. “Extramuros.” https://github.com/d0kt0r0/extramuros.

———. 2014e. “Daem0n.” https://github.com/d0kt0r0/Daem0n.sc.

Roberts, C., and J.A. Kuchera-Morin. 2012. “Gibber: Live Coding Audio in the Browser.” In *Proceedings of the International Computer Music Conference (ICMC)*. Ljubljana, Slovenia.

Roberts, Charles, Matthew Wright, J Kuchera-Morin, and Tobias Höllerer. 2014. “Rapid Creation and Publication of Digital Musical Instruments.” In *Proceedings of New Interfaces for Musical Expression*.

Rohrhuber, Julian, Alberto de Campo, Renate Wieser, Jan-Kees van Kampen, Echo Ho, and Hannes Hölzl. 2007. “Purloined Letters and Distributed Persons.” In *Music in the Global Village Conference (Budapest)*.

Strömqvist, Sven, Kenneth Holmqvist, Victoria Johansson, Henrik Karlsson, and Åsa Wengelin. 2006. “What Keystroke-Logging Can Reveal About Writing.” *Computer Key-Stroke Logging and Writing: Methods and Applications (Studies in Writing)* 18: 45–72.

Swift, Ben, Henry Gardner, and Andrew Sorensen. 2014. “Networked Livecoding at VL/HCC 2013.” In *Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium on*, 221–222. IEEE.

Swift, Ben, Andrew Sorensen, Michael Martin, and Henry Gardner. 2014. “Coding Livecoding.” In *Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems*, 1021–1024. ACM.

“Thecodeplayer.Com.” http://thecodeplayer.com.

Wilson, Scott, Norah Lorway, Rosalyn Coull, Konstantinos Vasilakos, and Tim Moyers. 2014. “Free as in BEER: Some Explorations into Structured Improvisation Using Networked Live-Coding Systems.” *Computer Music Journal* 38 (1): 54–64.