Models and Opportunities for Networked Live Coding

Sang Won Lee
Computer Science and Engineering
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121
snaglee@umich.edu

Georg Essl
Electrical Engineering & Computer Science and Music
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121
gessl@umich.edu

ABSTRACT

Since the inception of live coding, networked live coding has been present and is increasingly being used in practice. In this paper, we review existing works on networked live coding, associate such works with the dimensions of network music, and point out new opportunities in the field.

Keywords

live coding, network music, collaborative music making

1. INTRODUCTION

Since its inception, the presence of live coding ensemble has, in practice, posed challenges pertaining to facilitating collaboration and coordinating shared programming practices. In response to such challenges, researchers and practitioners have developed a number of live coding environments that contain networking capability. The goal of this paper is to look at existing works of networked live coding, to review responses to the aforementioned challenges and to find new opportunities that build upon the tradition of network music.

2. WHAT IS SHARED IN NETWORKED LIVE CODING?

Modern computers all but invite the use of networks in collaborative music-making contexts. The same can be said for collaborative live coding. Networked communication and data sharing can facilitate collaboration among live coders. While collaborative live coding does not necessarily mean that computers need to be connected over a network, the potential of networked live coding has been present from live coding's inception of live coding [8]. Realizing networked live coding requires detailed consideration of the networked system. There are a multitude of design choices that pertain to what kind of data is shared and how the system maintains that shared data across different network topologies and how it will facilitate collaboration among musicians over the network. Our purpose here is to review the types of shared data that are used in existing works while also advancing the needs induced by the data-sharing process in live coding practice. In particular we will review time sharing (synchronization), code sharing (text/program representation), program state sharing (run-time states, variables, objects, memory), access control, and communication facilitation (chat). We hope this classification provides an overview for live coding researchers creating a collaborative live coding environment and that it sheds light on collaborative aspects supported by the system.

Live Coding and Collaboration Symposium '14, Sep. 25 – July 26, 2014, University of Birmingham, UK.

Time sharing: In any form of multi-performer music, what is important is playing together, that is to say being synchronized. This is certainly also true for live coding performance. Hence, in a networked live coding setting, it is important to consider how to facilitate synchronization. One crucial precondition to this is having one synchronized clock between machines. Many networked live coding environments enable clock synchronization or implement ways to synchronize timing of musical events [6, 8, 10, 14, 26, 31, 36]. The method of clock synchronization varies depending on the architecture of the network and the distributed nature of the ensemble and well-known methods can be found in [9, 26]

Code sharing: Some live coders share actual code fragments among members of ensemble during the performance. Live coding environments on a web browser such as *Gibber* [27] or *Sketchpad* [1] offer a Google doc-like shared editor. In *LOLC*, commands are typed into an instant messaging-style interface that shows both commands and chat messages so that you can see your code, others' codes and chat messages [16]. Sharing code text is essential not only in facilitating collaboration but also in improving the sense of collaboration. In a previous work of ours [14], we had a shared program state but no shared code text. Through an unofficial self assessment, we discovered that programmers frequently look over a colleague's shoulder and read the code of others, not to understand the code but to see where he or she is currently working on. For the same reason that live coding is projected on screen to communicate with an audience [23], participants had a more engaging experience monitoring the progress of collaborators. This lead us to add a shared text editor using Google Realtime API [11] to our environment.

Program-state-sharing: Instead of sharing code text, some live coding environments enable sharing dynamic objects or variables in the program state. The ways in which sharing objects is implemented are diverse: re-rendering objects by evaluating code fragments in both the local machine and the remote machines [6, 30, 20], transmitting sound objects as serializable data [10], synchronizing the value of variables using tuple space [31], or being shared inherently due to one centralized program state for multiple live coders [14]. One may think that sharing code text is good enough as one has access to the code that can reproduce the same objects in a local machine. For a live coder, however, reading, interpreting, and evaluating the code fragment are additional cognitive loads. Furthermore, the code text in the editor is not a complete representation of the program state and there almost always exists a discrepancy between the code text (*State of Code*) and the program state (*State of World*) [32]. In other words, the code associated with a certain sound (or any outcome from the live coding) at the moment may not even exist in the text editor any more because it may have been modified. Conversely, not all the code in the editor is available in the program state until it is evaluated.

Access-control: Regarding shared code text and state synchronization, it should be noted that the nature of collaboration will vary depending on the level of permission (e.g., read/write/execute) given each live coder to the shared data. One can design a system to allow all types of permission for all participants, which enables open collaboration. This would be like the early networked music piece *The Hub's Borrowing and Stealing* [7]. In [10], the environment supports the sharing of musical patterns with read/execute permission so that shared objects are not mutable by other live coders. This is intended to encourage borrowing to play the pattern and to then create a new pattern based on the borrowed one. Or a live coding environment can select a more conservative strategy where a live coder can choose to share a certain set of variables in the selective manner [14] and choose to evaluate a certain code fragment remotely to obtain a dislocated sound [30]. Giving permission for all types of interaction to all live coders imposes possible risk of code/state corruption while collaborating. For example, in [20], the author addressed the fact that a shared text editor can present problems such as unexpected code deletions and collisions and this reflects the risk on his design decision to have read-only permission for collaborators so as to safeguard the code against corruption.

Communication-facilitation: Enabling chat is an effective strategy to facilitate communication between live coders as well as to engage an audience in the communication loop when projected on screen [1, 10, 14, 20, 24, 27]. Lastly, we want to point out that sharing actual low-level outcome (such as raw audio) has been less explored. Later in this paper, we suggest a few cases where sharing raw data between computers enables new types of collaboration.

Potential expansions of shared data: Numerous expansions of the above paradigms are conceivable. In particular, contextual and situated information could be incorporated into performance. This information could come from the performer's computers as well as from devices the audience posseses. Some of these ideas have been realized in the context of locative music performances [33].

3. CENTRALIZED AND DECENTRALIZED MODEL

In order to understand the differences in characteristics between networked performances, it is helpful to investigate taxonomies that clarify the respective properties of different network configurations. One important taxonomy for this purpose was proposed by Weinberg [35], who described a number of topologies of interconnected music ensembles depending on the architecture of the network among musicians (*centralized*/*decentralized*) and the nature of interconnectivity (*synchronous*/*sequential*), depicted in Figure 1. In this section, we borrow the concept of *centralized* / *decentralized* model to understand existing works of networked collaboration in live coding and to associate various design choices of live coding systems with two models.

In the *centralized* approach, one central machine handles all computations. As a consequence only one machine generates music regardless of the number of connected musicians. In the live coding context, we can say only one program state exists. In the meantime, the *decentralized* approach implies that there are multiple machines that generate sound. Similarly, multiple program states are available. This number of states could be as many as the number of live coders in the ensemble. Similar ideas were also introduced by Wang in the context of live coding. Rather than using a centralized/decentralized dichotomy, he used the more network-centric server-client / peer-to-peer distinction [34]. As a platform for networked co-live coding he also realized a collaborative audio programming space, *Co-Audicle.*

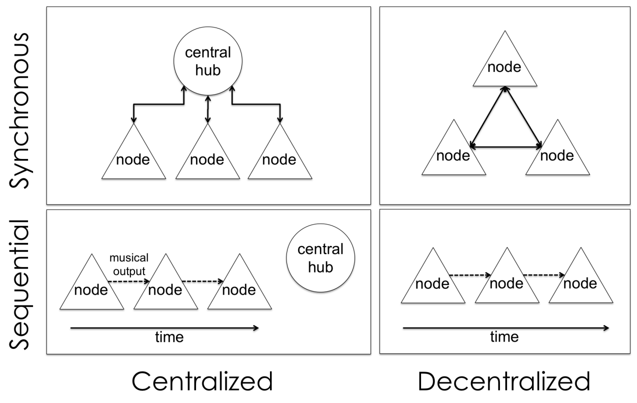

Figure 1: Various Topologies of Network Music Ensemble. Adapted after [35].

There are advantages and disadvantages in both the centralized and decentralized models, especially with respect to sharing dynamic objects and clock synchronization mentioned in the previous section.

Centralized (server-client) approach: Each live coder uses a computer (client) as a "dumb terminal" that holds code text to be sent to a central server. Since there is only one program state, there is no need for state/clock synchronization. However, each individual may face conflicts, collisions, and interruptions of some kind. This is because only one state space exists, leaving open the risk one inadvertently modifies the state space that might have been created by someone else. In [34], any code evaluation is encapsulated in a *shred* so the conflict cannot occur while the system does not support inter-shred state synchronization. In [14], our solution to the issue is to give each individual live coder his/her own *namespace* and to create a *shared namespace* separately that everyone has access to. In addition, the environment provides a summarized view of each namespace based on data polled from the server so that live coders can monitor live values of variables, functions and expressions in their own namespace as well as others at a certain time.

Decentralized approach: This approach is more frequently used in networked live coding. In this approach, one can make many design choices about which content (clock, state, code text, etc.) to share, as mentioned in Section 2. The simplest way to share data on the decentralized network is to broadcast (or push) the code fragment to (either all or a part of) connected nodes whenever it is evaluated. The use of an algorithm as transmitted data empowers a machine to re-render dynamic objects and a live coder to alter the algorithm, although it is based on the premise that all the members of the ensemble use the same live coding environment. *Powerbooks Unplugged* [29] is a typical decentralized ensemble where each machine has its own state, code is shared over a local area network whenever evaluated and a live coder, by evaluating the code remotely, can generate sound dislocated from the local machine.

A more sophisticated way to share data in the decentralized network is to synchronize the program states and timing of certain events. Ogborn recently introduced *EspGrid*, which supports the decentralized structure for synchronization of beats and related temporal parameters [24]. The system requires no central server and is tolerant to inhomogeneity of audio programming languages [26].

Decentralized programming with central timing: Decentralized live coding are often combined with the centralized approach. *Impromptu Spaces* exemplifies the state synchronization in a distributed manner with centralized time synchronization [31]. With *Spaces*, variables can be read and modified with full access (read/write) from multiple machines, utilizing tuple space. In addition, the system implements the server-client structure in which one controls timing (e.g., metronome) while the other processes use the primitives locally to synchronize a series of events. *LOLC* also utilizes the server-client structure and the server is responsible for the clock synchronization and sharing musical patterns [10]. Notice that the two examples above require one dedicated server machine but still belong to the *decentralized* model since each client holds a program state that generates music.

4. DISPARATE COLLABORATION IN NETWORKED LIVE CODING

According to Weinberg's classification, most examples we have reviewed fall into the *synchronous* collaboration. In contrast, the *sequential* relationship indicates non-real time interaction where collaboration occurs in a certain order between participants. For instance, in the report of Dagstuhl Seminar [5], McLean introduced *Mexican Roulette* where live coders take turns and write music at a computer.

Differing from the notion of the *sequential* relationship originally suggested, it can be expanded to a real-time situation. Imagine a laptop performer processing the acoustic sounds of a musical instrument; here the laptop performer forms a *sequential* relationship with the instrumental performer and mediates, in real time, the acoustic outcome. The similar mode of collaboration can be applied in a live coding context, enabling a novel musical aesthetic. For example, one of the live coders only works on a low-level sound synthesis algorithm and, at the same time, the other live coder uses the algorithm to generate musical events. Alternatively, raw audio data generated by a live coder can be transmitted to another so that the streamed audio becomes the basic material for the latter to transform and process the sound thereafter. *Pea Stew* by BEER is a good example where live coders sent raw audio signal to each other and created networked audio feedback [36]. The authors recognized that this interconnected relationship "presents challenges somewhat different" from those in traditional live coding performances and describe the piece as "several performers simultaneously playing a single instrument."

The *sequential* (and real-time) relationship provides a new aesthetic by connecting live coding musicians with instrumental performers. Live coding musicians can mediate the sonic outcome of an acoustic instrument, either by providing a musical medium for instrument performers to perform with or by processing an acoustic outcome of musical instruments with algorithms (note that the order of sequence is reversed). We have been part of the development of musical systems that support this type of divergent collaboration. In a recent extension of *LOLC*, laptop musicians generated real-time music notation on the fly by typing textual commands and instrumental musicians sight-read the generated notation for collaborative improvisation [16]. In [13], an instrument performer played a mobile music instrument while the mobile music instrument application was being live coded on the fly by on-stage programmers. In both works, note that the outcome of live coding is not generative music but media that instrumental performers play with. In addition, the sonic result of the performance can be drastically different from typical live coding music where one can demonstrate the immediate expressivity. Although it was a solo performance, a similar concept of a live-coded musical instrument was present where a performer live-coded sound synthesis and changed the mapping of a digital music instrument on the fly [2].

Lastly, there have been attempts to reconcile live coding performance with musical instruments in a *synchronous* manner [5, 19, 25]. However, networking has not been incorporated in this context.

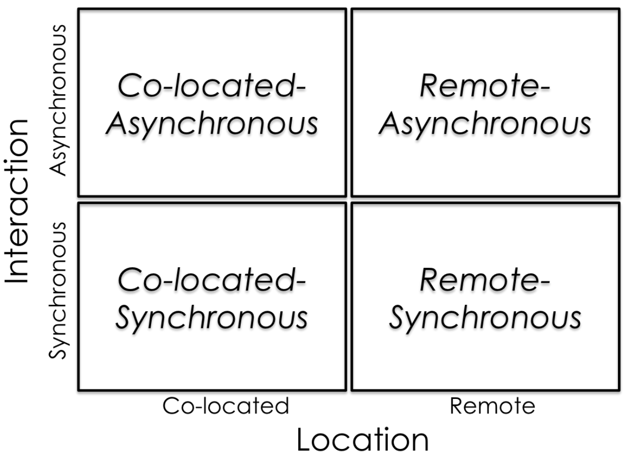

Figure 2: a Classification Space for Network Music. Adapted after [3].

5. REMOTE COLLABORATION IN LIVE CODING

One may think network music is a music performance by an ensemble at multiple geo-locations connected online in real time dealing with latency. However, other types of approaches exist. In [3], Barbosa suggested two dimensions to classify network music based on time (*synchronous*/*asynchronous*) and location (*local*/*remote*), depicted in Figure 2. Most networked live coding performances fall into *local/synchronous* setups where live coders are co-located and play in real time. *Remote/synchronous* collaboration has been attempted a few times in practice and seems relatively less explored compared to the ones in the co-located setup. For example, the launch report of Live Coding Research Network states that Sorensen performed live coding performance remotely [22]. Although the technical specification of the performance is unclear from the report, it is likely that the remote performer did not need, as does a typical telepresence performance, to transfer audio data from a site to the remote site. Rather code text can be transmitted online and be evaluated in a remote machine. In [17], the live coder collaborated remotely with other performers (non-live coders) in a bidirectional manner, used screen sharing to live-code on a remote laptop, and used a conference call to listen to the ensemble at the local site. *Gibber* [27] reported that the live coding environment on a web browser allows remote code execution with the remote collaborative code editing. We could not, however, find an example in which the environment had been used with the context. *OSCthulhu* is a data synchronization system possibly for remote live coding performance with the strategy of evaluating code remotely [21]. For all the examples above, the sound outcome from a local machine and remote machine may not be exactly the same for many reasons (e.g., packet loss, latency).

While it is beneficial to skip the audio streaming, to send only symbolic data (rather than audio streaming data) and to re-render sound remotely, there still remains the same concern about network latency to have audio feedback of the composite sonic result for the remote live coder. There can be algorithmic detours to minimize the divergence between the local and the remote outcome, such as i) quantization of the events in a minimum unit of beat, measure or hyper-measure, ii) synchronization of randomness within the distributed ensemble, and iii) adding latency in the local program state given a synchronized clock so that one can hear the realistic result by mixing the audio streamed from the remote site with the audio delayed on purpose from the local machine.

6. OPPORTUNITIES IN NETWORKED LIVE CODING

In reviewing networked live coding, we heavily leverage two classification methods of network music [3, 35] so as to find new opportunities in the field. We realize that there exist a few missing links between collaborative live coding and new trends in network music.

Asynchronicity and Notation: First, we have not discussed a system that will support *asynchronous* collaboration in live coding. However, the very notion of live coding breathes the potential to intentionally or accidentally go asynchronous, at least in traditional music performance. Hence it seems potentially exciting to envision such live coding systems. However, this leads to a need to bring back renewed notions of traditional forms of facilitation that keep music performers "together." A prime example of such a facilitation are scores and notation. In order to see asynchronous live coding be strengthened in the near future, we do believe it essential that there be a music notation system for the live coding piece is going to be essential. Live coding notation shall serve different purposes from on-stage visualization or static code text logs. We also doubt that the recording of a computer screen is the ideal form since it will contain performance-specific events (such as typos and bugs). Recently, Magnusson introduced *The Code Score* in his recent live coding environment, *Threnoscope* [18]. The graphical music score serves as a notation system for live coding composition, which lays out temporal patterns of code snippets in a piano-roll-like interface. The live coding community needs to start a discussion on this issue in order to disseminate live coding music to a wider community of artists. We may learn from the response of the laptop orchestra community to similar concerns in context of archiving a music piece [4].

Scalability: Secondly, it will pose many questions and offer new opportunities when networked live coding is scaled. Most cases of networked live coding we have reviewed here are on a small scale. Conventions and existing technologies in live coding may become unpractical when scaled. For example, it is not trivial to project screens if there are only five performers on stage [36]. In this sense, [26] is a noteworthy work where the author unfolds years of practice of live coding on the laptop orchestra scale. We are interested in scaling up even further. One can imagine that networked live coding could become a crowd-scale performance. How can collaborative coding be structured when one expects hundreds or more participants to operate jointly?

Diversification in Performance Practice: Lastly, the distributed nature will be more prominent. There will emerge new live coding performances that are hard to distinguish only by the *centralized/decentralized* model. A number of live coding environments enable disjunction between the client machines that hold code text and the machine that generates sound. For instance, the centralized model is implemented with the dislocation of code (n clients' machines) from the sound generation (one central server), which forms an n:*1* relationship. Possibly, a live coder can control a number of machines and orchestrate code distribution in aesthetically interesting ways (*1*:n). A truly distributed network of live coders, machines, performers, and audience (n:m:l:k) will create novel music performances such as a mixed live coding ensemble, mobile live coding, audience participation along with recent works in cloud computing [12], live coded instruments on mobile devices [13, 28] and code distribution techniques over a network [15].

7. CONCLUSIONS

In this paper we have reviewed networked live coding in the context of existing taxonomies of networked music performance, as well as discussed important aspects of the sharing process. Using the taxonomy, we suggested a few under-developed areas of networked live coding performances, urging a particular need to investigate issues of live coding notation systems, of concerns of scalability, as well as concerns regarding diversification of performance practices and their technical realization. In the end we may well see a combination of all of these factors, diverse, notated live coding performances on a very large scale, with many important research challenges to be met along the way.

8. REFERENCES

[1] Sketchpad. `http://sketchpad.cc/`. Accessed: 2014-07.

[2] M. Baalman. Gewording, 2014. Music Performance, the International Conference on New Interfaces for Musical Expression (NIME).

[3] Á. Barbosa. Displaced soundscapes: A survey of network systems for music and sonic art creation. *Leonardo Music Journal*, 13:53–59, 2003.

[4] S. D. Beck and C. Branton. Lela - laptop ensemble/library archive. In *Proceedings of the Symposium on Laptop Ensembles and Orchestras*, pages 27–30, 2012.

[5] A. Blackwell, A. McLean, J. Noble, and J. Rohrhuber. Collaboration and learning through live coding (Dagstuhl Seminar 13382). *Dagstuhl Reports*, 3(9):130–168, 2014.

[6] A. R. Brown and A. C. Sorensen. aa-cell in practice: An approach to musical live coding. In *Proceedings of the International Computer Music Conference*, pages 292–299. International Computer Music Association, 2007.

[7] C. Brown and J. Bischoff. Indigenous to the net: early network music bands in the san francisco bay area. *Available at crossfade. walkerart. org/brownbischoff*, 2002.

[8] N. Collins, A. McLean, J. Rohrhuber, and A. Ward. Live coding in laptop performance. *Organised Sound*, 8(03):321–330, 2003.

[9] R. B. Dannenberg, S. Cavaco, E. Ang, I. Avramovic, B. Aygun, J. Baek, E. Barndollar, D. Duterte, J. Grafton, R. Hunter, et al. The carnegie mellon laptop orchestra. 2007.

[10] J. Freeman and A. Troyer. Collaborative textual improvisation in a laptop ensemble. *Computer Music Journal*, 35(2):8–21, 2011.

[11] Google. Google drive realtime api. `https://developers.google.com/drive/realtime/`. Accessed: 2014-07.

[12] A. Hindle. Cloudorch: A portable soundcard in the cloud. In *Proceedings of New Interfaces for Musical Expression (NIME)*, London, United Kingdom, 2014.

[13] S. W. Lee and G. Essl. Live coding the mobile music instrument. In *Proceedings of New Interfaces for Musical Expression (NIME)*, Daejeon, South Korea, 2013.

[14] S. W. Lee and G. Essl. Communication, control, and state sharing in collaborative live coding. In *Proceedings of New Interfaces for Musical Expression (NIME)*, London, United Kingdom, 2014.

[15] S. W. Lee, G. Essl, and Z. M. Mao. Distributing mobile music applications for audience participation using mobile ad-hoc network (manet). In *Proceedings of New Interfaces for Musical Expression (NIME)*, London, United Kingdom, 2014.

[16] S. W. Lee and J. Freeman. Real-time music notation in mixed laptop–acoustic ensembles. *Computer Music Journal*, 37(4):24–36, 2013.

[17] lemuriformes's blog. `http://www.steim.org/projectblog/2011/04/15/residency-lemuriformes/`. Accessed: 2014-07.

[18] T. Magnusson. Improvising with the threnoscope: Integrating code, hardware, gui, network, and graphic scores.

[19] T. Magnusson and A. Sa. Fermata, 2014. Music Performance, the International Conference on New Interfaces for Musical Expression (NIME).

[20] C. McKinney. Quick live coding collaboration in the web browser. In *Proceedings of New Interfaces for Musical Expression (NIME)*, London, United Kingdom, 2014.

[21] C. McKinney and C. McKinney. *Oscthulhu: Applying video game state-based synchronization to network computer music*. Ann Arbor, MI: MPublishing, University of Michigan Library, 2012.

[22] A. Mclean. Live coding research network launch report. `http://www.livecodenetwork.org/launch-report/`. Accessed: 2014-07.

[23] A. McLean, D. Griffiths, N. Collins, and G. Wiggins. Visualisation of live code. *Proceedings of Electronic Visualisation and the Arts 2010*, 2010.

[24] D. Ogborn. Espgrid: A protocol for participatory electronic ensemble performance. In *Audio Engineering Society Convention 133*. Audio Engineering Society, 2012.

[25] D. Ogborn, 2014. Music Performance, Live Coding and the Body Symposium.

[26] D. Ogborn. Live coding in a scalable, participatory laptop orchestra. *Computer Music Journal*, 38(1):17–30, 2014.

[27] C. Roberts and J. Kuchera-Morin. Gibber: Live coding audio in the browser. In *Proceedings of the International Computer Music Conference (ICMC)*, Ljubljana, Slovenia, 2012.

[28] C. Roberts, M. Wright, J. Kuchera-Morin, and T. Höllerer. Rapid creation and publication of digital musical instruments. In *Proceedings of New Interfaces for Musical Expression (NIME)*, London, United Kingdom, 2014.

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

[30] J. Rohruber and A. d. Campo. The republic quark. `https://github.com/supercollider-quarks/Republic`, 2011.

[31] A. C. Sorensen. A distributed memory for networked livecoding performance. In *Proceedings of the International Computer Music Conference*, pages 530–533, 2010.

[32] B. Swift, A. C. Sorensen, H. Gardner, and J. Hosking. Visual code annotations for cyberphysical programming. In *1st International Workshop on Live Programming (LIVE)*. IEEE, 2013.

[33] A. Tanaka and P. Gemeinboeck. A framework for spatial interaction in locative media. In *NIME '06: Proceedings of the 2006 conference on New Interfaces for Musical Expression*, pages 26–30, June 2006.

[34] G. Wang, A. Misra, P. Davidson, and P. R. Cook. Coaudicle: A collaborative audio programming space. In *In Proceedings of the International Computer Music Conference*. Citeseer, 2005.

[35] G. Weinberg. Interconnected musical networks: Toward a theoretical framework. *Computer Music Journal*, 29(2):23–39, 2005.

[36] S. Wilson, N. Lorway, R. Coull, K. Vasilakos, and T. Moyers. Free as in beer: Some explorations into structured improvisation using networked live-coding systems. *Computer Music Journal*, 38(1):54–64, 2014.