Collaborative textual improvisation in a laptop ensemble

Collaborative Textual Improvisation in a Laptop Ensemble

Jason Freeman* **and Akito Van Troyer†**
*School of Music
Georgia Institute of Technology
840 McMillan Street
Atlanta, Georgia 30332-0456 USA
jason.freeman@gatech.edu
†MIT Media Lab
Opera of the Future
75 Ames Street, E14-374D
Cambridge, Massachusetts 02142 USA
akito@media.mit.edu

For us, text-based performance interfaces, such as those used in live coding systems (Collins et al. 2003), are a fascinating fusion of composition and improvisation. Textual performance interfaces can offer a precise and concise means to define, manipulate, and transform motives, gestures, and processes in real time and at multiple hierarchical layers. They can also render musical thinking visible to the audience by projecting the text as it is written.

We are particularly interested in textual performance interfaces in laptop ensemble contexts. We believe that the dynamics of ensemble performance can lead laptop musicians in new creative directions, pushing them towards more real-time creativity and combining the diverse skills, ideas, and schemas of the ensemble's members to create unexpected, novel music in performance. But when laptop performance interfaces move beyond simple one-to-one mappings, they present unique ensemble challenges, particularly in terms of the synchronization and sharing of musical material. Specialized improvisation environments for specific performances (e.g., Trueman 2008) or ensembles (e.g., Rebelo and Renaud 2006) can help groups to negotiate these challenges and structure their collaboration: The tools can create powerful new channels of networked communication among ensemble members to supplement aural and visual interaction. Textual performance environments are uniquely positioned in this regard: They potentially offer greater efficiency and flexibility in performance, and because text is already a dominant networked communication medium, they can draw from an abundance of interaction models in areas such as instant messaging, collaborative document editing, and the real-time Web.

Although many text-based performance environments do support collaboration, most current systems are challenging to use with ensembles of more than a few musicians. As a result, we created a new textual performance environment for laptop ensemble named LOLC. (LOLC was initially an acronym for Laptop Orchestra Live Coding, though we now consider the applicability of the term "live coding" to be debatable.) From its inception, we designed LOLC to focus on ensemble-based collaboration. Though inspired by live coding systems, LOLC is not itself a programming language: It is neither Turing-complete nor does it allow its users to define new computational processes. Instead, its design, in which musicians create rhythmic patterns based on sound files and share and transform those patterns over a local network, facilitates an interaction paradigm inspired by other forms of ensemble improvisation, particularly in jazz and avant-garde music.

In this article, we discuss the related work upon which LOLC builds; we outline the main goals of the LOLC environment; we describe the environment's design, its technical implementation, and the motivations behind some of our key decisions; and we evaluate its success through discussion of a recent performance.

Related Work

The design and implementation of LOLC was influenced by existing models for collaborative text-based laptop performance, by approaches to collaborative improvisation in other types of ensembles, and by collaborative composition systems.

Computer Music Journal, 35:2, pp. 8–21, Summer 2011

Existing Models for Collaborative Text-Based Performance

In principle, laptop-based musical ensembles can use any software environment to perform together; they need not be connected over a data network, and they can coordinate their musical activities solely via aural and visual cues. But although such a strategy works well with an instrumental ensemble, it can be more problematic in a laptop ensemble. Without a shared clock or live beat tracking, time synchronization is difficult. Further, the borrowing and transformation of musical motives across members of the group—an interaction paradigm that is common to many forms of improvisation—can be difficult in a laptop ensemble, where each member would need to manually recreate the musical content or underlying algorithm in his or her own environment.

To more effectively share information about timing and musical content, several live-coding environments implement collaboration features over a local-area network. Rohrhuber's JITLib (Collins et al. 2003) and Sorensen's Impromptu (Brown and Sorensen 2009) both take a similar approach: They enable clients to share and manipulate dynamic objects or variables over a network. Recent versions of Impromptu utilize tuple space to implement this type of synchronized sharing more robustly (Sorensen 2010), whereas JITLib uses proxy structures as placeholders for dynamically created and shared material (Rohrhuber, de Campo, and Wieser 2005). In small group settings, this approach can lead to deep and meaningful collaborations, as with Sorenson and Brown's aa-cell (Sorensen and Brown 2007). In larger ensembles, the shared control over objects can cause a shift from an ensemble of individual voices into a general "data climate" of software reacting to the state of the object space, as in the experiments Julian Rohrhuber has conducted with his classes (Collins et al. 2003). Although this can be an effective performance paradigm for large ensembles, it stands apart from most traditional ensemble models; at its extreme, each member loses control over his or her own sound output and, by extension, his or her identity as an individual voice within the group.

Instead of sharing dynamic objects or variables, some live coders have shared actual code fragments among members of the ensemble. A design document for the CoAudicle (Wang et al. 2005) supports such exchanges with a client-server or a peer-to-peer model, but the specifications remain vague and the system was never implemented. Andrew Sorenson plans to implement a collaborative text editor in Impromptu (Sorensen and Brown 2007), similar to SubEthaEdit (CodingMonkeys 2010) or Google Docs (Google 2010). JITLib (Collins et al. 2003) implements text chat functionality and enables musicians to interpret code directly on each other's machines (Rohrhuber 2010); the laptop ensemble Powerbooks Unplugged uses this environment to share code as an "open letter to the others, who may (or may not) read it, copy it, and modify it further" (Rohrhuber et al. 2007). A similar approach was followed in a laptop orchestra performance titled *TBA* (Smallwood et al. 2008). We also find these approaches appealing, but share others' concerns about the daunting code synchronization challenges involved (Wang et al. 2005).

Finally, it is important to note that many ensembles do not use standardized tools among all performers. Instead, they simply define a shared protocol for communication. Slub (Collins et al. 2003) follows this paradigm, with the shared protocol solely facilitating time synchronization. And The Hub (Brown and Bischoff 2002) followed a similar approach; each composition they performed defined a new protocol for the ensemble, but each musician used their own software.

The Hub's work *Borrowing and Stealing* offers another intriguing model that served as a direct inspiration for LOLC. Instead of sharing code or variables, members share music. In the piece, a shared data store maintains symbolic representations of musical fragments created by each player. Musicians then retrieve the fragments created by other players, manipulate them, play them, and store the new versions in the database. We found this approach a compelling model for LOLC because it supports aural interaction by providing a technical foundation through which musicians can more easily recreate and transform the material of other ensemble members.

Collaborative Improvisation in Other Ensembles

In designing LOLC, we not only built upon the ideas of live coding languages; we also considered the interactions among improvising musicians in other types of ensembles.

Many composers have created structured environments for ensemble improvisation in their works. In *Cobra* (Zorn 1995), "tactical" gestures ask players to imitate or trade motives. In *meanwhile, back at the ranch . . .* (Walshe 2005), hand-drawn instructions ask players to copy the music of another performer using a variety of strategies: literal reproduction, transformation into a new color, or merging their own music with the color of another performer. And in *Virtual Concerto* (Lewis 2004), sections of the orchestra improvise together, following a matrix of instructions: Some boxes in the matrix instruct them to imitate the music improvised by another group, either simultaneously with them or immediately after they finish. Similar ideas pervade ensemble-based improvisation pedagogy; a "dominoes" exercise, for example, asks players to sit in a circle and play in sequence, closely imitating the gesture of the person preceding them (Allen 2002).

Many ethnomusicologists have characterized group interaction among improvising jazz musicians as a conversation. Monson, for example, notes in one instance how "the exchange of the [musical] idea not only established an abstract succession of sounds and rhythms but linked [the musicians] as musical personalities . . . at a particular moment in time" (Monson 1996, p. 80). Berliner describes how musicians respond to each other, particularly when trading short improvised phrases, noting how "musicians pursue a middle ground that satisfies their desire for both continuity and change by borrowing material from one another and transforming it" (Berliner 1994, p. 370).

Collaborative Composition

Because LOLC combines elements of composition and improvisation in its interface, we also wanted to consider the ways in which music has been collaboratively composed, out of real time, by groups. Members of Les Six collaboratively composed the music for the ballet *Les Mariés* by each writing one or two of the nine pieces that constitute the ballet's score (Gottlieb 2005). That collaborative process has been intensified as it has migrated to the Internet through projects such as the Wiki Collaborative Composition Project (Frankel 2008). Collaborative remix sites, such as ccMixter (Yew 2009), encourage users to create songs that incorporate tracks and samples from those of other community members.

Goals

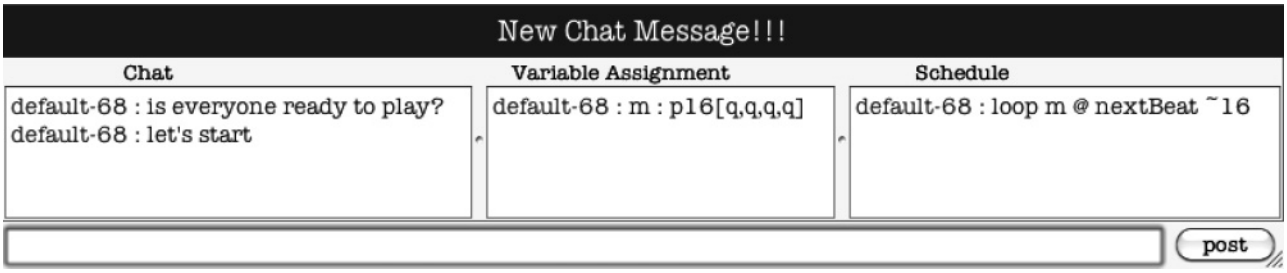
Our primary goal with LOLC was to create a text-based performance environment that scaled, both technically and musically, to large ensemble performance. In doing so, we wanted to facilitate collaborative modes common to traditional group improvisation and composition, particularly a conversational style of interaction in which musical material is continually shared and transformed.

A secondary goal was accessibility. Live-coding languages require a daunting level of technical virtuosity and practice to effectively program on the fly (Nilson 2007), and few computer musicians possess the requisite talent and experience to do so. By moving away from a complete language towards a focused textual interface, we wanted LOLC to be readily accessible to novice programmers and even non-programmers. We hoped this approach would bring some of the experiences that make live coding so exciting to a broader group of practitioners, including skilled musicians without previous experience in computer music, and that it would improve our ability to field a large ensemble of proficient LOLC musicians. In these respects, LOLC is inspired by ixi lang (Magnusson 2010) in both its goals and its design.

Further, we wanted LOLC to be accessible not only to performers, but also to audiences. We wanted audiences to understand what the musicians in the ensemble were doing and how they were interacting with each other.

Finally, we had some explicit non-goals in the development of LOLC. We did not want to

Figure 1. The chat interface in the LOLC client software.



create a full-featured language along the lines of ChucK (Wang and Cook 2003) or SuperCollider (McCartney 2002). In order to effectively facilitate ensemble interaction and to make the environment accessible to novice users, we knew that the scope and flexibility of the environment would be limited. LOLC does not, for example, support sound synthesis or signal processing.

We also did not want to dictate a particular structural model for performances with LOLC. Decisions such as the roles ensemble members play, the musical structure of the performance, and the use of a conductor are left to the musicians to decide.

Design Principles

In this section, we discuss some of the key design decisions we made in creating an environment to support our goals.

Conversational Interaction

With LOLC, we wanted to model the conversational interactions that many ethnomusicologists have identified in jazz improvisation. We wanted this conversation to unfold both aurally and over a computer network; the latter eliminates the need for musicians to recreate the material they borrow from others.

The dominant paradigm for real-time conversational interaction over computer networks is text-based chat or instant messaging (Nardi, Whittaker, and Bradner 2000). LOLC follows this paradigm. Its main chat window (see Figure 1) is similar to most standard chat clients, but it separates different message types into distinct columns. Code is shared through the chat interface as it is executed, and clients can also send messages to each other to coordinate and plan their performance.

Because it is cumbersome to send and view long textual fragments through a text-messaging interface, we designed LOLC to consist entirely of short, single-line statements.

Code Is Music

Although we wanted LOLC musicians to see each other's code—and we supported this via the chat-based interaction—we also wanted the interaction model to focus on sharing musical, not computational, content. LOLC's digital interaction primarily supports existing aural interaction, making it easier for musicians to re-use and transform the material they hear others playing.

Shared code statements in LOLC take the form of pattern definitions: symbolic representations of rhythmic, dynamic, and sound-source information. This approach follows closely from systems such as The Hub's for *Borrowing and Stealing*.

Patterns Are Immutable

We were concerned that a dynamic binding system would sacrifice too much individual agency, moving the collaboration model too far away from traditional group improvisation techniques. Instead, patterns in LOLC are always immutable once defined. A change or transformation is stored as a new pattern. Although this increases the size of the pattern name space, our visualization interface (see Figure 2) helps

users more easily sort through that space to find the material they wish to borrow.

Time Synchronization

As with most collaborative performance tools, we wanted LOLC to help musicians play in sync with each other. The LOLC server maintains a counter that tracks beat, measure, and hypermeasure. (A *hypermeasure* is a metrically marked measure that begins a group of measures. By default, LOLC makes every fourth measure a hypermeasure.) Meter and tempo are configured from the server's graphical user interface. Players synchronize by using scheduling operations that reference the counter. LOLC implements time synchronization among machines internally, using its own NTP-style protocol to calculate and compensate for time offsets between machines, following from the Carnegie Mellon Laptop Orchestra's strategy (Dannenberg et al. 2007). This frees synchronization from dependence on network latency or access to an external time server.

Ease of Use

Because we wanted LOLC to be accessible to novice users, the environment is simple and has an easy learning curve. All expressions are a single line in length, and there are only two expression types: pattern definitions and scheduling operations. We use musical terminology instead of numerical data when possible. Dynamics and durations, for instance, are represented in musical terms (see Table 1 in the next section). Similarly, users do not code the logic of algorithms directly, but instead draw from a library of pre-defined operations (see Table 2 in the next section).

In addition to making the environment simple, we wanted to make the user interface simple. Musicians code in LOLC using a tightly coupled integrated development environment (IDE), similar to ChucK's mini-Audicle (Salazar, Wang, and Cook 2006). The IDE (see Figure 2) includes code editors, chat windows, a console, configuration dialogs, and tutorials.

Finally, it was important to us that LOLC be easy to deploy in large ensemble settings, as network and application configuration can otherwise waste valuable rehearsal time. LOLC networking setup is trivial. Time synchronization happens automatically, and pre-defined patterns are stored in a server configuration file and automatically broadcast to clients as they connect.

Design of LOLC

LOLC focuses on the collaborative creation and manipulation of rhythmic patterns based on pre-recorded, percussive sound files. This section explains the details of LOLC and its implementation.

Sound Files

Pre-recorded sound files serve as the base musical element in LOLC. Although any sound file can theoretically be used, short and percussive sounds tend to work best.

Sound files are loaded into LOLC from a default folder by creating a single-element pattern:

```
mySound : "sound.aif"
```

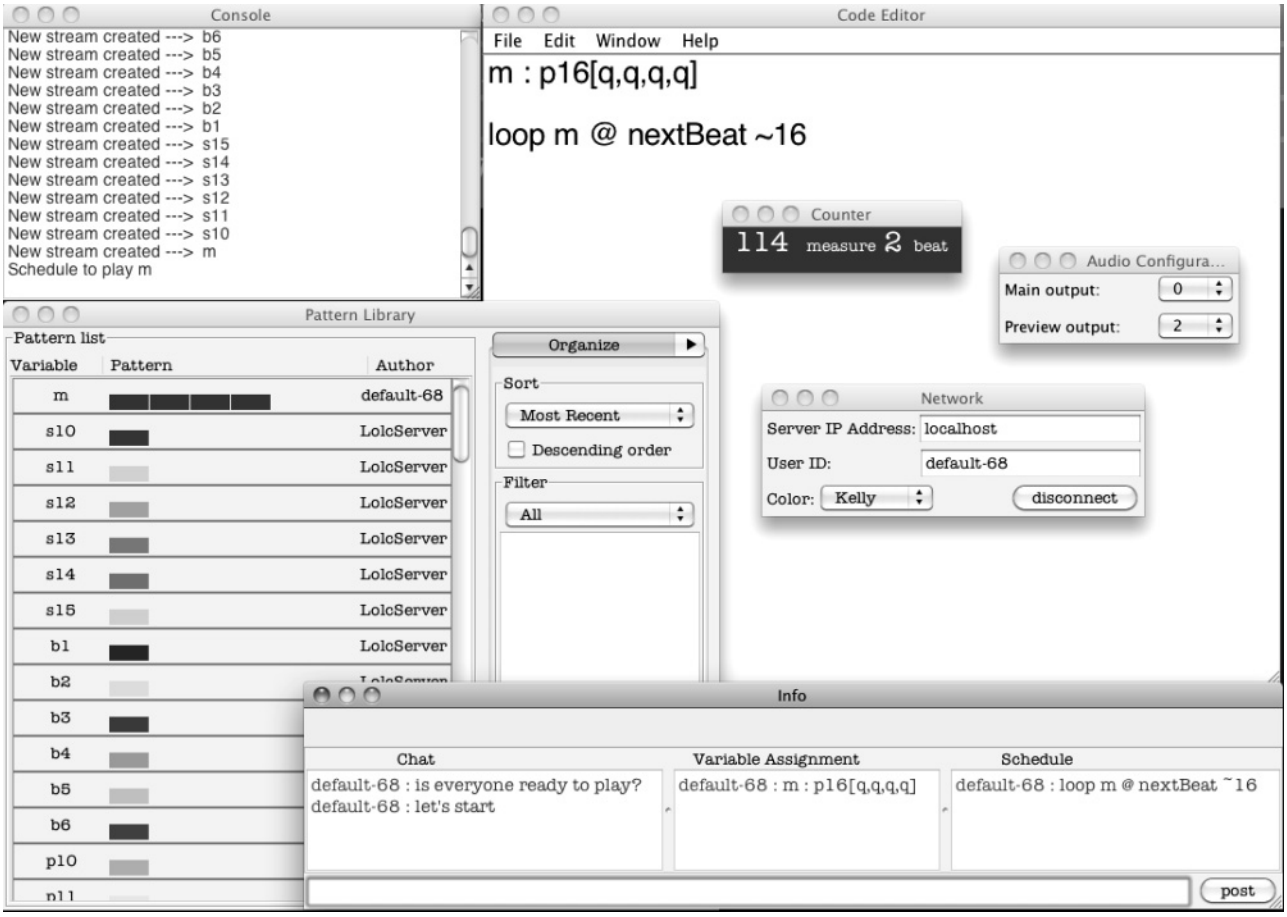
Pattern Creation

Rhythmic repetitions of sounds are created through a bracket syntax. For example, the following pattern definition plays the sound *mySound* as two quarter notes at *fortissimo*, an eighth-note rest, and three eighth notes at *pianissimo*:

```
myPattern : mySound[q.ff, q.ff, e.n,
                    e.pp, e.pp, e.pp]
```

Table 1 includes a complete list of available durations and dynamics. If dynamics are omitted, *mezzo-forte* is assigned to the note. Although these dynamics are limited to nine gradations, and have no envelope control, the use of

Figure 2. Screenshot of the complete LOLC client interface.



Western-style dynamic markings makes the environment more accessible to non-computer musicians. We also expect musicians to use these dynamics to mark occasional events rather than to manually set the dynamics for every event, and that dynamic envelopes, which would be tedious to define in performance, are intrinsic to the sound files themselves.

Patterns can also be nested. In this example, *myPattern* is played twice:

```
myNested : myPattern[w,h]
```

Each time the pattern is played, its note durations are stretched or compressed to match the target duration. In this example, the patterns remain unchanged for the whole note iteration and are halved for the half-note iteration.

All patterns are automatically shared with the other clients on the network when they are defined. Once a pattern is defined, it is final and immutable.

Like Vocables (McLean and Wiggins 2009), LOLC defines musical patterns as an ordered collection of items, but LOLC requires musicians to explicitly define rhythmic values, and it emphasizes rhythmic patterns that repeat single sounds. Ixi lang (Magnusson 2010) similarly emphasizes repetitions of single sounds and explicitly defined rhythms, but its grid-based approach to rhythm is both simpler and more constrained than LOLC's.

Table 1. Duration and Dynamic Values Available within LOLC

LOLC Expression	*Musical Meaning*
w	whole note
h	half note
q	quarter note
e	eighth note
s	sixteenth note
t	thirty-second note
x	sixty-fourth note
u	hundred-twenty-eighth note
n	*niente* (silent)
ppp	*pianississimo*
pp	*pianissimo*
p	*piano*
mp	*mezzo-piano*
mf	*mezzo-forte*
f	*forte*
ff	*fortissimo*
fff	*fortississimo*

Pattern Transformation

LOLC supports creating new patterns that transform or combine existing patterns. Table 2 lists the ten operations currently supported. These operations were chosen because of their importance in studies on improvisational interaction (Hodson 2007) and musical pattern manipulations (Spiegel 1981). Each operation's result maintains a degree of aural similarity to its source pattern(s), though if operations are chained or used in particular ways (e.g., dropping all but one item in a pattern), the connection to the original source(s) may be lost. Similarly, most any allowable pattern can be derived using a chain of operations and pattern nesting. The syntax for all transformations follows this example:

```
myPattern1 : sound1[w,h,h]
myPattern2 : sound2[q,q,q,q]
myConcat : concat(myPattern1,
  myPattern2)
myTrunc : trunc(myPattern2, 2)
```

The operation *concat* places two patterns in succession, so *myConcat* combines patterns based on two different audio-file sources. The operation *trunc* removes the final *n* items from a pattern, so *myTrunc* removes the final two quarter notes from *myPattern2*.

Scheduling

Patterns are not played immediately upon creation. Instead, musicians must use an LOLC scheduling expression to determine when and how they play. Typically, patterns are scheduled for playback at the next beat, measure, or hypermeasure:

```
play myPattern @ nextBeat
play myPattern @ nextMeasure
play myPattern @ nextHyperMeasure
```

LOLC's meter and tempo are configured in the server application. By default, the values are 4/4 and 120 beats per minute, respectively.

LOLC supports additional scheduling methods: `preview` will preview the pattern over headphones and `loop` will play it repeatedly:

```
loop myPattern @ nextMeasure ~16
```

This example loops the pattern sixteen times. (If the pattern length is not an integer multiple of the measure length, then some loop iterations will begin mid-measure.) Additional mechanisms support shorthand notations, scheduling at specific points of time in the future, and prematurely killing playback. Scheduling commands are shared with other clients via the text chat interface, but they are played solely on the local client machine.

Client Interface

In LOLC's client interface (see Figure 2), musicians type LOLC code in one or more code editor windows and press a hotkey combination to execute those statements (similar to SuperCollider). A console window shows error messages in parsing and execution. The Info window takes the form of a text chat interface, showing chat and code messages from other users. Other windows show status and configuration information.

Table 2. Pattern Transformation Operations Currently Supported in LOLC

LOLC Operation	*Result*
alternate(x,y)	Interweave the items in patterns x and y: [x1, y1, x2, y2, x3, y3, . . .].
cat(x,y,. . .)	Add the items in pattern y to the end of pattern x.
drop(x,n)	Remove the first n items from pattern x.
mirror(x)	Equivalent to cat(x, reverse(x)).
reverse(x)	Reverse the order of the items in pattern x.
rotate(x,n)	Remove n items from the beginning of pattern x and add them to the end of x (for positive n); remove n items from the end of pattern x and insert them at the beginning of x (for negative n).
scramble(x,y)	Equivalent to shuffle(cat(x,y)).
shuffle(x)	Randomly rearrange the items in pattern x.
trunc(x, n)	Remove the last n items from pattern x.
warp(x)	Randomly alter the durations and amplitudes of the items in x.

The pattern library window provides a visual interface for exploring ensemble activity, with the goal of facilitating closer collaboration and more pattern sharing among musicians. Patterns are displayed in the window as they are created, along with a visualization of their content: rhythmic data is indicated by horizontal bars and sound sources are color coded. Variables are highlighted when they are played. Musicians can also sort and filter the list to isolate patterns that are created by certain musicians, scheduled for certain points in the performance, or based on particular audio file sources.

Server Interface

The LOLC server software is intended to run unattended during performance, so its interface is simple: a console window, configuration dialog, and logging mechanisms. The server also includes a full-screen visualization for projection to the audience. The current visualization (see Figure 3) simply shows the measure counter and all code and chat messages in a stylized, large-print format.

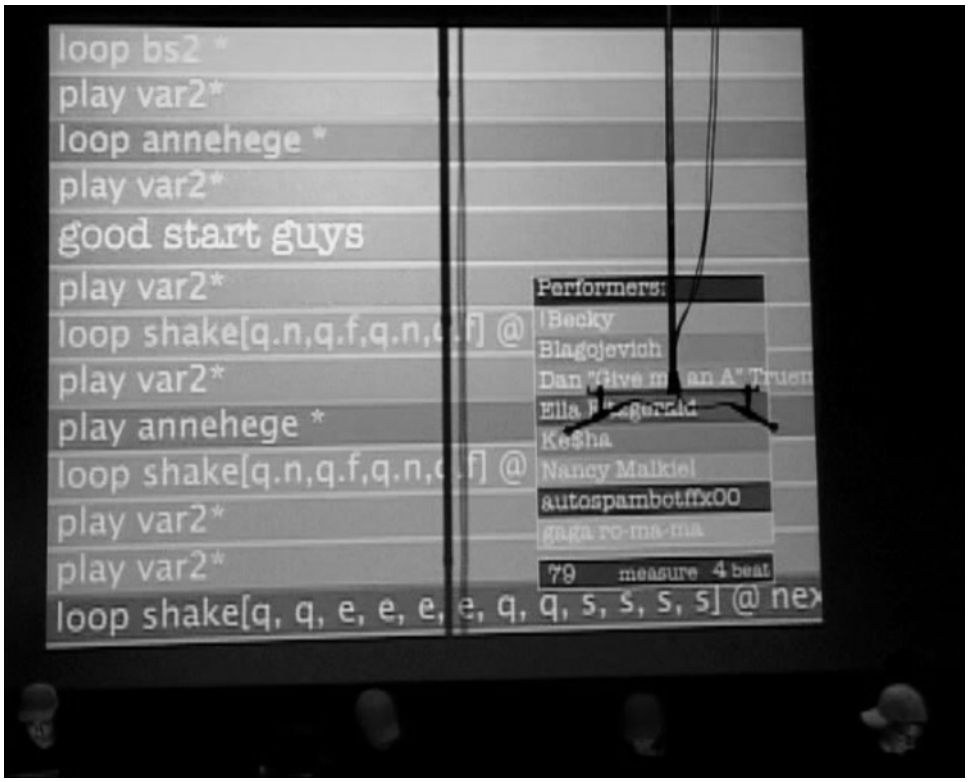
Technical Implementation

We implemented LOLC in pure Java, with the user interface in Processing and Swing. We originally implemented audio playback in SuperCollider (McCartney 2002) but switched to a pure Java implementation using Nick Didkovsky and Phil Burk's Java Music Specification Language (Didkovsky and Burk 2001) coupled with Burk's JSyn application programming interface (API) (Burk 1998). Although these environments lack some of the features, flexibility, and efficiency of SuperCollider, they easily handle the simple scheduling and audio file playback tasks of LOLC. We also found it easier to implement cross-platform support for a double-clickable application using this pure-Java implementation, and found that the application more easily relaunched in the event of a freeze or crash. Most importantly, the API's support for music notation also provided a foundation for future extensions to LOLC (see subsequent discussion).

For the LOLC interpreter, lexical analysis—the segmenting of LOLC code into tokens—is handled by JFLex (Klein 1999). Syntax analysis, in which the tokens are formed into an abstract syntax tree, is handled by CUP (Hudson, Flannery, and Ananian 2010). As compared to our initial implementation that did not use these industry-standard tools, our interpreter is now far more robust, more flexible to syntax changes and additions, and more informative in its error reporting.

LOLC employs a client–server network model (as opposed to a peer-to-peer model) to facilitate

Figure 3. Photo from the Princeton Laptop Orchestra performance of LOLC showing the visualization projected for the audience.



internal time synchronization, visualization rendering, central data logging, global configuration settings, and initial pattern definitions. The clients and server communicate via TCP/IP, not UDP: Because scheduling messages are always executed on the local machine, network messages are never extremely time critical. It is critical, however, that each message ultimately arrives at its destination. Because the server and client software are both written in Java, they share many code classes and send serialized Java objects across the network instead of using an independent network protocol.

Evaluation and Discussion

Between February and April 2010, eight members of the Princeton Laptop Orchestra learned LOLC, rehearsed intensively, and presented a 10-minute public performance using the environment. The performers were all undergraduate university students with majors in either computer science or in music.

Within this context, we used a variety of techniques to assess the degree to which LOLC succeeded at meeting its goals of facilitating (and encouraging) collaborative improvisation within an ensemble and being accessible to performers to learn and audiences to understand. We logged code and chat messages to disk and analyzed these logs quantitatively and qualitatively. We asked members of the ensemble questions about their experiences with LOLC. We spoke with musicians and audience members about their experiences. And we considered the musical output of the performance as well as our own experiences working with the musicians in rehearsals. These evaluation techniques served as a mechanism to obtain rapid feedback and to guide continuing development of LOLC; a more formal user study is forthcoming (see subsequent discussion).

The LOLC software proved robust in rehearsal and performance. Additionally, LOLC was easy to

learn; ensemble members were able to comfortably perform with LOLC at the end of a two-hour introductory session. Although the members of the ensemble did have some previous programming experience, their programming background—about two years on average—was far less extensive than that of most live coders.

We did find that the musicians tended to play back patterns far more often than they created new ones; in our analysis of a dress rehearsal for their performance, we noted that the musicians executed 355 playback operations but just 38 creations or transformations of patterns. Although this disparity could well have been a musical or organizational decision, it may also reflect the relative ease of writing scheduling versus creation expressions in LOLC. Scheduling operations, unlike pattern definitions, can be typed once and executed many times. One musician noted to us that coding new patterns was always "slow" but that it would continue to get better with practice. Perhaps, then, the ratio of pattern definitions to scheduling operations corresponds to the musician's level of facility with LOLC. There was, in fact, significant variation within the ensemble in this regard: One musician scheduled 27 operations per pattern created, whereas at the other extreme, one musician scheduled only 3.9 operations per pattern created.

Audiences, for the most part, did not understand the code fragments as they were projected on the screen; the natural musical terminology used for durations and dynamics was of little help to them. There is, perhaps, an inherent design paradox here. Performing musicians tend to prefer a concise syntax that requires minimal typing, whereas audiences tend to have difficulty understanding text that is not sufficiently verbose. Faced with such situations, audiences "may become stuck on small details" of the code (McLean et al. 2010). A new visualization engine for LOLC, currently under development, seeks to address this challenge by displaying the typed code along with a graphical representation of its content and meaning.

The text chat messages among performers, however, provided unique insights into the collaborative process, and the audience members with whom we spoke really enjoyed seeing how the musicians were thinking about the music and interacting with each other. (Musicians wore colored baseball hats that matched the color of their onscreen text so that audiences could identify them.) At many points in the performance, audience members even laughed in response to the chat messages.

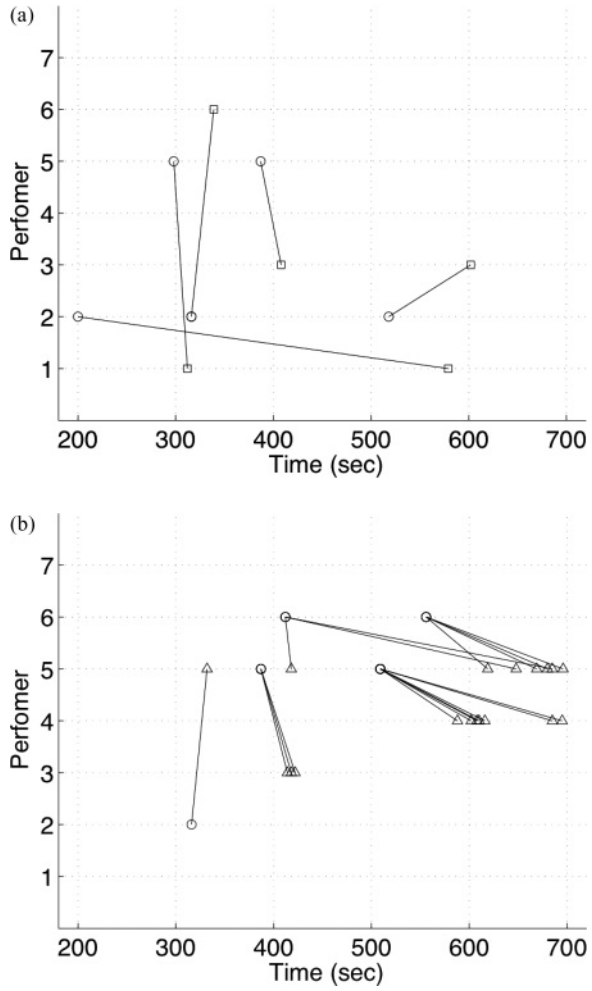
Our primary interest, of course, was in how the musicians collaborated to create each performance. One metric of collaboration is the degree to which patterns were shared among musicians. Although the quantity of shared patterns does not necessarily correlate to the quality of collaboration, it does suggest how much LOLC encouraged and facilitated such collaboration.

In our analysis of the group's dress rehearsal, we found that eight patterns were played by musicians other than their creators, and five patterns were transformed by musicians other than their creators. (A total of 38 patterns were created or transformed during the course of the performance.) Figure 4 shows the sharing of patterns graphically, and reveals an interesting detail: Most of the shared patterns were borrowed by just three of the players. This suggests that certain musicians were more interested in borrowing (or at least more adept at it). One musician, in fact, used only patterns borrowed from others, creating no new patterns from scratch, and two other musicians borrowed no material from others, using only the patterns they created themselves.

We were pleased that musicians did share material but wish they had done so more often. In our own observations and in comments from the musicians, the pattern visualization window emerged as a significant impediment to increased collaboration. Navigating its display and search features in performance was difficult, and it was hard to understand the sonic content of patterns from the color coding. (Musicians in this performance also chose not to use headphones for the preview functionality, so it was challenging to predict how a pattern would sound.) We believe that improvements to the visualization of patterns would encourage more sharing and transformation of patterns among ensemble members.

In rehearsal, the ensemble experimented with a variety of approaches to organizing their

Figure 4. Analysis of pattern sharing among ensemble members. Figure 4a shows when musicians schedule patterns, without modification, that have been created by other ensemble members; circles indicate the definition of the original pattern and squares indicate the scheduling of the borrowed pattern. Figure 4b shows when musicians transform patterns created by other ensemble members; circles again indicate the original pattern and triangles indicate the creation of the transformed pattern. In both figures, solid lines connect the shared patterns from the borrower back to the creator.



performance. They eventually chose a "conducted" model, in which one player sent cues via chat message to the other musicians to set up solos, musical textures, and structural arrival points. Other musicians responded as necessary with further thoughts, but most of the messages came from the conductor. The messages were not preset; rather, they were improvised by the conductor during the performance. In this excerpt from the text chat log, the conductor sends out a series of messages to coordinate an upcoming solo by a musician named Atrish:

> Alright, now let's switch over to p's at 185
> Alright, let's start an atrish solo at measure 220
> And everyone else say "atrish!"
> out loud when it happens
> Atrish, solo on the s sounds
> If you are reading this
> God, I hope you are.
> Everyone back in at 235
> With s's
> Atrish and Mary, switch to B's
> Everyone from clayton over, p's
> Me, Hannah, Gabe, P's.

Our own subjective assessment of the musical performance was largely positive. Overall, we found the music compelling and cohesive; it did not merely sync in time, but also seemed to mesh together to create composite textures and large-scale structural motion. We noticed two qualities, however, that suggest the musical and collaborative limitations of LOLC. First, the music was overwhelmingly loop-based. Musicians tended to create patterns and loop them for long periods of time; the net result was multi-layered, slowly evolving minimalist textures. Although there is nothing wrong with this musical aesthetic, we suspect that the nature of LOLC pushed the ensemble towards creating this type of music: The environment makes it easy to loop rhythmic patterns, and the performance situation encourages musicians to do so to avoid risking periods of uncomfortable silence. We suspect that with more rehearsal time and more performance experience, the ensemble might have branched out to try other approaches. And we suspect that additional features in LOLC—particularly a library of operations to algorithmically generate new patterns along with additional methods to transform existing ones—would also have encouraged this.

Along with the slowly evolving musical texture, we also noted a slow pace of collaboration that seems inherent to LOLC. In a jazz combo, musicians can respond to each other quickly: for example, by trading fours. In LOLC, such an interaction would require more planning and would likely unfold more slowly, because it requires advance creation of patterns and scheduling of events (and potentially coordination via text chat). We do not necessarily see this as a problem so much as an observation about the nature of collaborative text-based performance

as a medium situated between composition and improvisation. Live-coding scholars have noted the challenge of responding quickly in performance and have suggested some relevant practice and preparation techniques (Nilson 2007) and language and interface design considerations (Blackwell and Collins 2005; Magnusson 2007).

Future Work

LOLC has largely succeeded in creating an accessible environment for laptop ensembles to collaboratively improvise, and performances with the environment have been successful, but much work remains to be done. In addition to making revisions to the environment to address the feedback we have received to date, we wish to pursue several large-scale projects with LOLC.

First, we want to better understand how musicians—especially those with little or no background in computer programming and computer music—learn the environment and collaborate with it. To that end, we will be conducting a formal study with Sonic Generator, the professional ensemble-in-residence at Georgia Tech, as they rehearse and prepare a performance with the environment.

Second, we would like to explore the use of LOLC as an educational tool. Our experience with the Princeton Laptop Orchestra suggested to us its potential in this regard. Loosely following the Performamatics model (Ruthmann et al. 2010), we would like to formalize curricula for high-school and college-aged students that use LOLC to teach about improvisation and computer music to students who may have little background in either of these areas, and to determine what pedagogical advantages (and disadvantages) LOLC may have in comparison to live-coding or other languages.

Finally, we plan to extend LOLC to the realm of real-time notation, in which musicians sight-read conventional or graphical notation live, in performance, as it is rendered on a digital display (Freeman 2008). In this scenario, LOLC musicians can manipulate musical score fragments in addition to audio files, and those fragments are displayed in real time to sight-reading musicians (on traditional instruments) with whom each laptop musician is paired. To us, this extension gives LOLC greater power as a tool to facilitate large-ensemble collaborations between the acoustic and digital domains, and it also makes LOLC a useful platform for experimenting with real-time notation in general; there is currently a lack of such tools (Freeman and Colella 2010). A performance with this extended system is already planned with Sonic Generator, for an ensemble consisting of violin, cello, flute, clarinet, marimba, and five laptop musicians.

Acknowledgments

LOLC is supported by a grant from the National Science Foundation as part of a larger research project on musical improvisation in performance and education (NSF CreativeIT 0855758). In addition to the authors, Andrew Colella, Sang Won Lee, and Shannon Yao have all made substantial contributions to designing and developing LOLC. We also extend our thanks to Dan Trueman, Jeff Snyder, and the Princeton Laptop Orchestra. The LOLC software, a performance guide for using LOLC with your own ensemble, and videos of performances with LOLC are available at www.jasonfreeman.net/lolc.

References

Allen, S. 2002. "Teaching Large Ensemble Music Improvisation." *Radical Pedagogy* 4(1). Available on-line at radicalpedagogy.icaap.org/content/issue4_1/01_Allen.html. Accessed December 2010.

Berliner, P. 1994. *Thinking in Jazz: The Infinite Art of Improvisation*. Chicago: University of Chicago Press.

Blackwell, A., and N. Collins. 2005. "The Programming Language as a Musical Instrument." In *Proceedings of Psychology of Programming Interest Group*. Brighton, UK: University of Sussex, pp. 120–130.

Brown, A., and A. Sorensen. 2009. "Interacting with Generative Music through Live Coding." *Contemporary Music Review* 28(1):17–29.

Brown, C., and J. Bischoff. 2002. "Indigenous to the Net: Early Network Music Bands in the San Francisco Bay Area." Available on-line at crossfade.walkerart.org/brownbischoff. Accessed August 2010.

Burk, P. 1998. "JSyn—A Real-time Synthesis API for Java." In *Proceedings of the International Computer Music Conference*, pp. 252–255.
CodingMonkeys. 2010. "SubEthaEdit." Available on-line at www.codingmonkeys.de/subethaedit. Accessed August 2010.
Collins, N., et al. 2003. "Live Coding in Laptop Performance." *Organised Sound* 8(3):321–330.
Dannenberg, R., et al. 2007. "The Carnegie Mellon Laptop Orchestra." In *Proceedings of the International Computer Music Conference*, pp. 340–343.
Didkovsky, N., and P. Burk. 2001. Java Music Specification Language, an Introduction and Overview. In *Proceedings of the International Computer Music Conference*, pp. 123–126.
Frankel, J. 2008. "The Wiki Collaborative Composition Project." Available on-line at wikicompose.pbworks.com/w/page/14280678/FrontPage. Accessed December 2010.
Freeman, J. 2008. "Extreme Sight-Reading, Mediated Expression, and Audience Participation: Real-time Music Notation in Live Performance." *Computer Music Journal* 32(3):25–41.
Freeman, J., and A. Colella. 2010. "Tools for Real-Time Notation." *Contemporary Music Review* 29(1):101–113.
Google. 2010. "Google Docs." Available on-line at docs.google.com. Accessed August 2010.
Gottlieb, L. 2005. "Images, Technology, and Music: The Ballets Suédois and Les mariés de la Tour Eiffel." *The Musical Quarterly* 88(4):523–555.
Hodson, R. 2007. *Interaction, Improvisation, and Interplay in Jazz*. New York: Routledge.
Hudson, S., F. Flannery, and C. Ananian. 2010. "CUP." Available on-line at www2.cs.tum.edu/projects/cup. Accessed August 2010.
Klein, G. 1999. "Jflex User's Manual." Available on-line at www.jflex.de. Accessed August 2010.
Lewis, G. 2004. "Virtual Concerto." Unpublished musical score.
Magnusson, T. 2007. "The ixiQuarks: Merging Code and GUI in One Creative Space." In *Proceedings of the International Computer Music Conference*, vol. 2, pp. 332–339.
Magnusson, T. 2010. "ixi lang: A SuperCollider Parasite for Live Coding." In *Proceedings of the 2010 SuperCollider Symposium*. Available on-line at www.ixi-software.net/thor/ixilang.pdf. Accessed December 2010.
McCartney, J. 2002. "Rethinking the Computer Music Language: SuperCollider." *Computer Music Journal* 26(4):61–68.
McLean, A., et al. 2010. "Visualisation of Live Code." In *Proceedings of Electronic Visualisation and the Arts 2010*. Available on-line at http://www.bcs.org/content/conWebDoc/36045. Accessed December 2010.
McLean, A., and G. Wiggins. 2009. "Words, Movement and Timbre." In *Proceedings of the 9th International Conference of New Interfaces for Musical Expression*. Available on-line at doc.gold.ac.uk/~ma503am/writing/nime09.pdf. Accessed December 2010.
Monson, I. 1996. *Saying Something: Jazz Improvisation and Interaction*. Chicago: University of Chicago Press.
Nardi, B., S. Whittaker, and E. Bradner. 2000. "Interaction and Outeraction: Instant Messaging in Action." In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*. New York: ACM, pp. 79–88.
Nilson, C. 2007. "Live Coding Practice." In *Proceedings of the 7th International Conference on New Interfaces for Musical Expression*, pp. 112–117.
Rebelo, P., and A. Renaud. 2006. "The Frequencyliator—Distributing Structures for Networked Laptop Improvisation." *Proceedings of the 2006 International Conference on New Interfaces for Musical Expression*. Paris: IRCAM–Centre Pompidou, pp. 53–56.
Rohrhuber, J. 2010. "Networked Programming." Available on-line at supercollider.svn.sourceforge.net/viewvc/supercollider/trunk/common/build/Help/Libraries/JITLib/tutorials/jitlib_networking.html?revision=10439. Accessed December 2010.
Rohrhuber, J., A. de Campo, and R. Wieser. 2005. "Algorithms Today: Notes on Language Design for Just In Time Programming." In *Proceedings of the International Computer Music Conference*. Available on-line at quod.lib.umich.edu/i/icmc. Accessed December 2010.
Rohrhuber, J., et al. 2007. "Purloined Letters and Distributed Persons." *Music in the Global Village*. Available on-line at globalvillagemusic.net/2007/wp-content/uploads/pbup_paper.pdf. Accessed December 2010.
Ruthmann, A., et al. 2010. "Teaching Computational Thinking through Musical Live Coding in Scratch." In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. New York: ACM, pp. 351–355.
Salazar, S., G. Wang, and P. Cook. 2006. "miniAudicle and ChucK Shell: New Interfaces for ChucK Development and Performance." In *Proceedings of the International Computer Music Conference*, pp. 63–66.

Smallwood, S., et al. 2008. "Composing for Laptop Orchestra." *Computer Music Journal* 32(1):9–25.
Sorensen, A. 2010. "A Distributed Memory for Networked Livecoding Performance." In *Proceedings of the International Computer Music Conference*, pp. 530–533.
Sorensen, A., and A. Brown. 2007. "aa-cell in practice: An approach to musical live coding." In *Proceedings of the International Computer Music Conference*, pp. 292–299.
Spiegel, L. 1981. "Manipulations of Musical Patterns." In *Proceedings of the Symposium on Small Computers and the Arts*. Los Angeles: IEEE Computer Society, pp. 19–22.
Trueman, D. 2008. "The Telephone Game: Oil/Water/Ether." Available on-line at www.turbulence.org/Works/plork. Accessed December 2010.
Walshe, J. 2005. *meanwhile, back at the ranch*. Dublin: Milker Corporation.
Wang, G., and P. Cook. 2003. "ChucK: a concurrent, on-the-fly audio programming language." In *Proceedings of the International Computer Music Conference*, pp. 219–226.
Wang, G., et al. 2005. "CoAudicle: A Collaborative Audio Programming Space." In *Proceedings of the International Computer Music Conference*, pp. 331–334.
Yew, J. 2009. "Social performances: Understanding the motivations for online participatory behavior." In *Proceedings of the ACM 2009 International Conference on Supporting Group Work*. New York: ACM, pp. 397–398.
Zorn, J. 1995. *John Zorn's Cobra: Live at the Knitting Factory*. New York: Knitting Factory Works KFW 124, compact disc.