Live Coding The Mobile Music Instrument

Sang Won Lee
Computer Science & Engineering Division
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@eecs.umich.edu

ABSTRACT
We introduce a form of networked music performance where a performer plays a mobile music instrument while it is being implemented on the fly by a live coder. This setup poses a set of challenges in performing a musical instrument which changes over time and we suggest design guidelines such as making a smooth transition, varying adoption of change, and sharing information between the pair of two performers. A proof-of-concept instrument is implemented on a mobile device using *UrMus*, applying the suggested guidelines. We wish that this model would expand the scope of live coding to the distributed interactive system, drawing existing performance ideas of NIMEs.

Keywords
live coding, network music, on-the-fly instrument, mobile music

1. INTRODUCTION
While live coding have blurred the borders among an instrument builder, a composer and a performer, we are in favor of transplanting the outcome of live coding from speakers and screen to an instrument performer. In this paper, an extended form of live coding performance is introduced, where a performer plays a musical instrument on mobile device while the instrument is being built on-the fly by a live coder over the network (Figure 1). We suggest that decoupling the notion of musical instrument from live coding will expand the expressivity of live coding music.

The distributed music performance combines existing forms of computer music; live coding, networked ensemble, and on-the-fly mapping musical instruments. Bringing these models together, we take advantage of the flexibility of live coding to reinforce digital music instruments. We utilize *UrMus* [11], a programming environment to support interactive music performance for mobile phone, which is readily available for live coding over wireless network.

This paper describes the background and context in which the model is developed; explores new opportunities exhibited by the dual model of an instrument builder and an instrument player; addresses implications and design challenges on playability of the instrument; proposes an example implementation of solutions in response to the specified challenges; and discuss future works planned with these extensions.

NIME'13, May 27-30, 2013, KAIST, Daejeon, Korea.

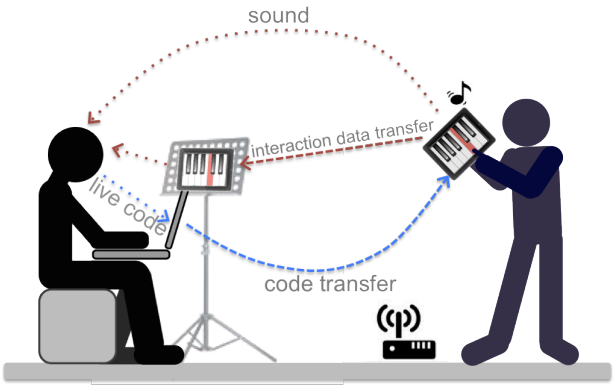

Figure 1 Performance Concept: a live coder(left) building a mobile musical instrument and a performer(right) playing the mobile instrument.

2. RELATED WORKS
Live coding [6] has yielded a new practice in electronic music performance. It is a music making practice where a programmer/musician codes, runs and modifies a program live while music (and/or visuals) is generated. Many programming languages has been developed (or repurposed) to facilitate live coding in a musical performance, such as Supercollider [27], ChucK [39], Impromptu [37] and many more [5, 8, 26, 34, 36, 38]. A number of works have looked at exploring the hands-on knowledge in live coding practice [2, 30]. In live coding, the programming language is seen as the musical instrument [1, 40]. While the traditional notion of a musical instrument does not fit well in the live coding model, it brings a unique intellectual/aesthetic challenge to musicians, where one has to convert composition ideas into working code and organize sounds in expressive ways under the time constraint.

The distributed music making environment of this work is influenced by creative works in the field of computer music where multiple users perform one instrument interdependently [20]. Although there's no technology involved, John Cage's *Imaginary Landscape No. 4* [4] is one of the earliest examples where two people have different roles in playing one instrument, which is radio in this case. The goal of multi-user instruments is often to facilitate collaborative creativity, such as *DaisyPhone* [3] or *FMOL* [18]. In contrast, there has been a different type of networked musical instrument where the instrument mediates distributed musical expression by multiple players. *Squeezebles* by Weinberg exemplifies this approach of each player influencing a collective improvisation by controlling different mapping parameters (e.g. level, timbre, pitch etc.) [43]

The author (Lee) continuously attempted to propose various formats of distributed music ensembles. In a recent extension of *LOLC* [15], music notation became a medium to integrate acoustic instrument players into the environment in which laptop musicians type, run shell-script-like language and generate music score for collaborative improvisation [23]. In

another previous work, a networked mobile instrument enabled the audience to participate as performers and to play the musical instruments while a musician on stage controls chord progression of audience's play over the network [22].

Finally, the key idea of live coding musical instruments on a mobile phones is directly inspired by earlier works to create on-the-fly musical instruments. In principle, many electronic musical instruments (e.g. synthesizer, MPC) let a performer change configurations (e.g. timbre, level, voice, effect, etc.) with some interface (pedal, knob, slider, buttons or external interface) in live performance. However, on-the-fly programmable musical instruments go beyond re-configurability. Both live patching environments [21, 31, 32] and live coding [6] deal with the fundamental concept of constructing musical instruments (i.e. sound synthesis or control input). Particularly, *ChucK* enables techniques for programmable interface of a musical controller (e.g. MIDI devices, acoustic instruments) so that one can dynamically change mapping of the controller [42]. Using its mobility and interactivity, a mobile phone became a generic platform to implement a musical instrument and invited real-time sound synthesis and on-the-fly mapping. *SpeedDial* is a mobile musical instrument that allows users to build mapping between sensors and sound synthesis on the fly [10]. *SenSynth* is analogous to the concept of *SpeedDial* with more focus on sonification of various mobile sensor data [28]. *massMobile* built a remote controller framework on a mobile phone particularly for audience participation where mobile GUI configuration can be changed by a preprogrammed sequence or manually modifying while its being used [44]. The closest work to this project is the recent extension of *Control* [35]. In this work, widgets (e.g. sensors, sliders, buttons) can be generated dynamically on a mobile phone by sending OSC messages and users can send OSC messages back to a computer to change sound control parameter in live coding environment. This work concentrated more on implementing the front end (control/interface) of a musical instrument using preset GUI components. We combine these efforts from prior works (See Table 1) to fully explore instrument design in crafting an on-the-fly musical instrument.

Table 1 Prior Works of On-The-Fly Musical Instruments

	sensor	user interface	mapping	sound synthesis
ChucK			X	X
SpeedDial/SenSynth	X		X	X
massMobile		X		
Control	X	X		

3. MOTIVATION

In the era of NIMEs, the development of musical instruments, composition, and performance are often concurrent/out of sequence. For instance, Cook suggest that composing a piece first is a good principle to develop a NIME [7] while Murray Brown et al. argues that concurrent process of composition in instrument building will help convey the music to audiences [29]. Embracing the unclear order of today's computer music making, we believe deferring the creation of a musical instrument until the time of performance can push the level of the liveness of a musical performance. As one of the mainstays in music aesthetic is to violate expectation [16], we can bring the tension from the level of music notation and performer's play down to that of instrument design and instrument builder's action. The role of the instrument builder can be constructive so that the musical variability of the instrument builds up over time, whereas and vice versa, one can collapse the space of musical expression such as an destructive example in which a musician plays a piano while it is being burnt down [46].

Another motivation of the on-the-fly musical instrument building is the fluidity of the concept. Magnusson defines "composing an instrument" as a process of designing constraints for a musical space [25]. Therefore, the act of live coding an instrument would be analogous to improvisation in the space, which will facilitates impromptu creativity given the changes of constraints. The motive of a dynamic affordance/constraint of expressivity can vary. For example, it can be a compositional decision of an instrument builder in collaborative improvisation, while, in a different scenario, it can be adaptation (or confrontation) in response to a particular performer's play style (e.g. Jazz instrument player vs. Live looping player). In another case, as already explored in [33, 44], the instrument can be utilized as a device for audience participation where the instrument provides progressive expansion of expressive space based on the learning curve of audience members.

While live coding offer uncharted space of expressivity and virtuosity with its flexibility and computational superiority, there exist particular styles of music that can be "efficiently [19]" played with live coding: gradually evolving, repetitive rhythmic, synchronized beats and multiple voice layered music. In contrast, as mentioned in [30], it is difficult to achieve immediacy with live coding as if one would play a traditional musical instrument with "*one gesture to one acoustic event*" [45]. Therein lies one of our motivations in this work: to decouple an instrument player from live coding to add instrumental virtuosity and expressivity. In this performance model, the live coder takes role of a composer, instrument builder and meta-performer, whereas an instrumental player performs the processed and progressing instrument. We believe this distributed model will benefit the aesthetic framework of instrumental music from the fluidity of live coding. In addition, as live coding has focused heavily on audiovisuals, we wish to make an expansion of the field to user interaction setting. As already anticipated in [24], this will bring a set of research questions of on-the-fly instruments, such as playability for performer, which we will explore later in this paper, or communication for audience engagement.

4. DESIGN CHALLENGES

While there is no limitation in the form of instrument built, one way to build a musical instrument without physical interference in playing is to transfer source code to a device over a network and remotely run the digital musical instrument. That requires a device that already has a set of sensors, sound synthesis module and programmable platform. We chose mobile phone as the platform, which is well established as a platform for musical performance [14, 41]. Although the result of live coding will be restricted by computational power, available sensor and limited size of mobile phone, it is good enough for the programmer to inject code wirelessly as well as to access critical variables of the musical instrument; control input (sensor/interface), sound generation (synthesis) and mapping problems.

Playing a dynamically transforming musical instrument will be challenging. The specific challenges from the performer perspective include:

- Which sort of change happens?
- When exactly does it happen?
- How does the change affect the ongoing acoustic event?

The answer to these questions will vary depending on musical context that the instrument is being used (e.g. whether the performance is improvisation or composition, whether it is

collaborative, competitive or destructive). Even though the liveness of musical instrument will likely be part of the aesthetic, the instrumental player will want to know how much space he/she has for musical expression, at least, at the moment. When a transition is made, one also need to choose how the change will affect current ongoing acoustic events. We suggest four guidelines for these particular design challenges as follows:

- Visual feedback on both ends (the performer and the live coder)
- Crossfade of changes (both visual and auditory)
- Distinguish continuous data flow from note-on/note-off type event.
- Enable transitions to be pushed by the live coder and pulled by the performer.

We provide a set of techniques for following these four design challenges later in this paper.

5. URMUS FOR LIVE CODING

UrMus [11] is a meta-environment, which allows flexible design of mobile musical instruments. Already with its predecessor *SpeedDial* [10] and continuing with *UrMus* [11, 13], support for on-the-fly mentality was part of the scope of providing an environment for designing and developing interactive mobile instruments. However the environment is readily available for live coding in this networked scenario due to its already existing support for transport and remote execution of code over the network. *UrMus* has an audio engine of flexible data-flow pipelines between sensors and sound synthesis algorithm to allow interactive live patching on mobile phone [12]. The audio engine follows the concept of graphical patching language and lets a programmer build a patch by connecting flowboxes (sensor/unit generator/sink) either in textual or graphical environment. In lower layer, *UrMus* API in *Lua* [17] provides a way to build interactive graphics quickly and to connect GUI events with flowboxes in a sound synthesis patch. Originally Lua was meant to provide a layer of implementing different kinds of representation of code and performance. However in this project we utilize Lua itself as a vehicle for live-coding in *urMus*, by-passing the question of programming language representation.

There are a few advantages of *Lua* that makes well suited for live coding. In *Lua*, data structures and functions can easily be extended or modified without refactoring. Another strength of how *UrMus* employs *Lua* is that it preserves the address space created by previous code when a new code is interpreted, instead of executing program from scratch. Any code will be executed on top of a running program allowing new code access the memory space of previous code. For example, if a user declared variables (or functions) in running code, one can submit code which uses them without declaration or assignment like other live coding languages. In addition, *Lua* is also a very compact and efficient interpreter language so that code can be transferred to the mobile device over the network. The editing environment of *UrMus* is implemented as a web service on the mobile phone so that a user can code on any web browser (usually running on a laptop) and transfer code to be interpreted on the device over a local wireless network. This allows remote development without any physical interference on the device. One modification made for live coding purpose to *UrMus* was adding a new web page editor that enables multiple tabs of text editor (Figure 2) so that a user can organize codes into modules and run them selectively. In addition, a coder can execute a section of code by highlighting lines and triggering a keyboard shortcut (Ctrl+R or Cmd+R). This follows the capability of the live coding language to support selective execution of code.

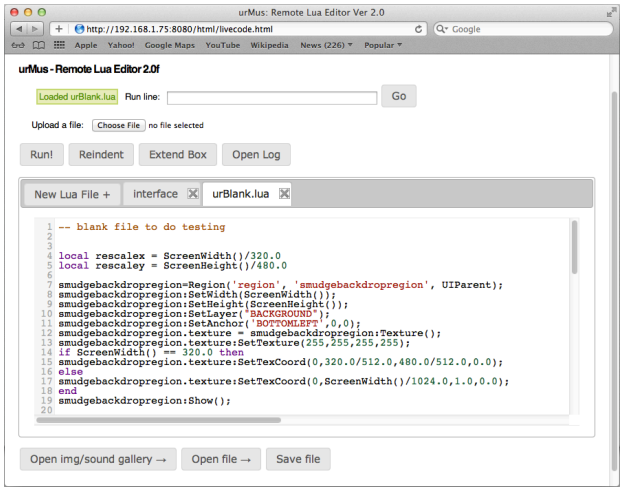

Figure 2 New multiple tab editing environment for modular execution.

6. IMPLEMENTATION

As mentioned in section 4, a set of programming/patching techniques will help achieve playability with respect to acquiring the current state of the instrument being built. These techniques are not canonical by any means but they provide useful references for relevant live coding projects and for developing the library of functions required to support the performance model. A simple proof-of-concept instrument has been implemented on a tablet device to help understand the concept in practice.

The idea of the instrument is a simple tone-matrix interface where a performer sets which note (y axis) to play at specific time (x axis) in a looped melody (see Figure 3). While the performer will define the content (looped melody) in the tone matrix interface, the design of the instrument can be replaced or modified on the fly by the live coder. For example, one can change not only GUI elements such as the number of rows (pitch register) and the number of column (meter) of the matrix, but also underlying musical parameters such as base note, scale, tempo (play bar speed) or voice (synthesis algorithm). The Figure 3 (left) shows the case where the tone matrix is in pentatonic scale with the base note of C and a six beat loop and this is later modified to E-minor scale in a eight beat loop (Figure 3, right).

In addition, we introduce another type of instrument in contrast to the tone matrix (see Figure 4, left). On the rightmost column, a slide instrument is added, which has the same pitch register with the tone matrix but requires a real-time control with touch (like slide-theremin). The finger movement will allow expressive pitch control like vibrato or slide in guitar. At the same time, accelerometer data (y axis) is fed into the level parameter of the slide instrument to control dynamics. The performer can switch back and forth between two instruments, modifying the loop and improvising on the slide instrument. The live coder can modify the instrument in response to how the performers play, for example, they can i) reinforce the slide instrument with more pitch register and timbre control ii) evolve the looped melody (like you would do in traditional live coding) while letting the performer focus on the slide instrument or iii) gradually reduce the range of expressivity to end the piece.

As mentioned earlier, visual feedback is needed so that the instrument player knows the current state of the instrument. The instrument exploits graphical user interface design to display visual feedback, such as color-coded pitch information and note names in the leftmost column.

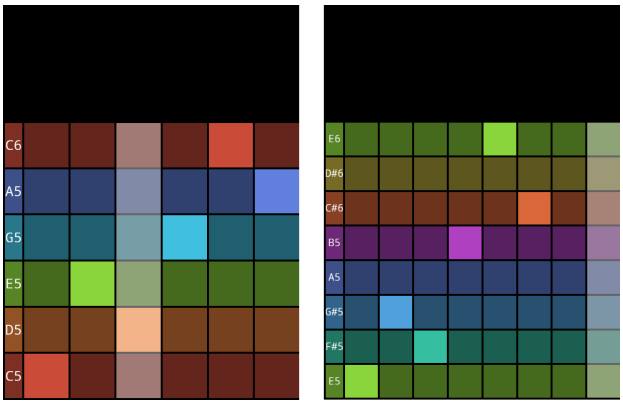

Figure 3 The tone matrix interface. (left) The tone matrix is in pentatonic scale with the base note of middle C and six beat loop. (right) The tone matrix is modified to E minor scale with eight beat loop with existing notes transposed.

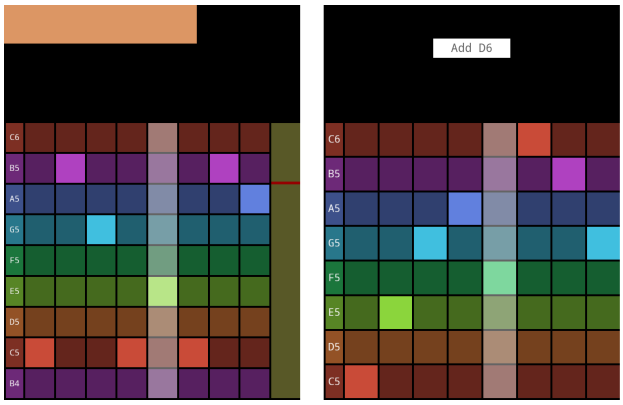

Figure 4 (left) The slide-instrument is shown at the rightmost column. The red line indicates the touch point and the horizontal bar at the top visualize the level controlled by tilting gesture. (right) A button is added for the performer to pull the change.

In addition, the live coder can directly notify the performer of an upcoming transition by sending a textual message. By the same token, the live coder needs to monitor the visual interface not only to see that the change had been made as intended but also to understand the play of the performers. One of the many ways to monitor the visual interface used in this example is to reproduce the same interface with the interaction that the performer made on an extra tablet (by transferring the same code to two devices). This feature is enabled by the performer's device sending OSC message to the other device whenever relevant events (e.g. OnTouchUp /OnAccelerate) occur. At the same time the device to be monitored would receive the events and call the relevant event handler (as presented in Figure 1). This approach is helpful particularly when the touch screen is the main interface, where a performer's action is not apparent just by looking at someone staring at and tapping on a tablet. On the other hand, it is important to present information effectively and keep the time of implementation. For example, it is not necessary that all aspects of the live coded interface shall be monitored. Some control (such as tilting the device) may not need to be presented back to the live coder as it is observable from the gestures of the performer.

New code can be transferred into the running program whenever the live coder finish a set of code and press the run button on the web page editor. The code would be executed immediately and applied from the next audio sample and the next screen update. As those abrupt changes would interfere with current interaction and audio output (e.g. clipping sound, button displacement), we took crossfading approach when making transition from existing code to new code. For sensor/sound synthesis code, a live coder can code a separate patch and gradually crossfade from the old patch to the new one by updating a mixing parameter over time. For visual GUI elements, animation of appearance /displacement/removal can be implemented for a smooth change.

On the other hand, crossfading is not always the best choice. For example, an fade-in animation would be fine for adding a button in the tone matrix. However, for a slide instrument, changing the scale into a different key would interfere with ongoing gesture, such as the case when the performer is playing a long sustained tone with vibrato. In this case, it is more natural to delay the transition until the current acoustic event ends and apply it from the next note-on event. This is due to the fundamental difference between execution levels of sound synthesis and that of gestural control. Most of musical instruments will have two definite states (note on/off) of gestural control. Hence it is useful to distinguish those kinds of interfaces from continuous data flow of sound synthesis and take different approaches to making transitions. For example, it will be useful to add a self-destructing feature for a patch to free flowboxes and its connection when the state is note-off and releasing acoustic events are over, instead of applying crossfader.

Another alternative, which may be combined with crossfading or not, is to let the performer pull new code. In collaborative mode of interaction, the live coder can provide code in a decomposed module, which will be triggered by a certain gesture. For example, the live coder can provide a simple button with textual description so that pressing button will trigger to run new code, which only will cost a few more lines of code. Figure 4 (right) shows an example where the performer can press a button to add one more row (pitch) in the tone matrix. After triggering, the button can either disappear or be reused to toggle the state back and forth.

Lastly, we wanted to note that live coding the tone-matrix instrument, although the instrument can be modified in real time, heavily leverage a set of prepared helper functions and would be challenging to create from a scratch without a prolonged period of silence. It was more realistic for us to decompose a set of frequently used code in separate functions, for example, createButton(), pullChange(), notifyMessage(), insertRow() and so on. In fact, most of these functions can be used regardless of the type of instrument. Many of the techniques for implementing the design guidelines suggested can be automated by supporting the features in the API, though it is not clear that the API is the best place for such features. In addition, building GUI elements (e.g. drawing a button or animated graphics) would require more than several lines if built from scratch but it is a trivial task to be encapsulated in a few lines These features can be implemented in advance for a specific performance and loaded before the performance so that it would reduce time spent on mundane tasks and enable live coders to focus on more creative side of the performance. Most importantly, it does not impinge on the openness and flexibility of the live coding potential.

7. LIVE CODING PERFORMANCE

An improvisational piece with a live-coded mobile music instrument was presented during the final class concert of the Michigan Mobile Phone Ensemble in April 2013. The core goal of the performance was to build an instrument on a tablet from scratch (a blank screen and minimum helper functions) to convey the idea of live coding the instrument. In order to do so, we chose to have three performers on stage; two live coders implemented an x-y interface with matched synthesis algorithm

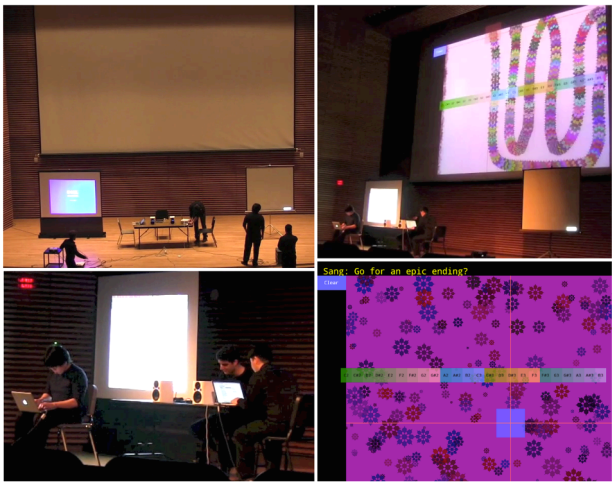

Figure 5 Live Coding Performance footage. Clockwise from topleft: 1) the stage configuration 2-3) snapshots of the performance 4) a screenshot of the live-coded instrument.

(see Figure 5) allowing for simple yet expressive continuous musical gestures. One live coder created the user interface. He started off, building a simple button on a 2-dimensional surface. The motion of the button in the plane was mapped to pitch and nonlinearity of the sound generated. The other live coder focused on sound synthesis algorithm, which utilized circle maps algorithm [9] to offer an expressive sound ranging from a pure pitched tone to the timbrally rich sound of a highly non-linear circle map. For the stage setup (see Figure 5), all three screens (two laptops and one tablet) were projected so that audiences could better understand our improvisation both on the laptops and the tablet.

The design guidelines suggested in the previous section were deployed for the performance. To give the performer feedback on the current state, a chat interface was added to the web editor so that live coders can inform what changes were made (e.g. "*x-axis covers full audible pitch range now.*"). The textual messages were displayed both on the tablet and the other live coder's chat interface. In addition, providing a button for the performer to pull the change (execute certain code) worked effectively, so that the performer knew some changes were ready to be made and chose when to inject them into the ongoing musical interaction.

Towards the end of the piece, live coders participated in the performance in more direct manner. The sound synthesis coder recorded a performer's play and looped the sound patterns in the background while the interface builder add visualization to the user interface based on the trajectory of the button, which changes the stationary button into a brush.

The performance succeeded in following the goal of the piece, but also by being able to handle technical glitches and coding bugs as the performance progressed. The audiences appreciated the nature of the piece by applause when the button functioning for the first time. Particularly, chat projected on a main screen was effective for musicians not only to reveal the process of collaborative coding but also to interact with audience members during the piece.

8. FUTURE WORKS

We introduced a new form of networked music performance where a programmer codes a mobile music instrument while it is being played by a performer. A simple instrument was implemented as a proof-of-concept showed how each design guideline is applied in practice and presented an improvised musical performance.

While we focused more on playability/usability of the performer's side in this paper, we wish to explore the same questions from the live coder's perspective. It would be beneficial to define challenges of live programming of interactive systems. Clearly, much work remains to be done on how a language and its development environment can support broad needs of networked live coding.

In addition, we wish to extend the distributed model of dual performers to a model of an interconnected ensemble. As we already experienced the mode of collaborative coding at the performance, we aim for *m:n* relationships where *m* people works on building musical instruments for *n* number of people. What sort of features would be desirable in a development environment of real-time collaborative coding setup? How do we deal with conflicts and version control when multiple people work on one application live ($m>n$)? How and when do we compose to distribute an instrument to each individual member of the ensemble ($m<n$)?

Clearly, live coding the musical instrument is an idea that still begs multitudes of explorations. In this paper, we discussed early steps of its potential and we are excited to see it realized.

9. ACKNOWLEDGEMENT

We would like to thank Bruno Yoshioka and Cameron Hejazi for performing the piece at the concert.

10. REFERENCES

[1] Blackwell, A. and Collins, N. The programming language as a musical instrument. In *Proceedings of Psychology of Programming Interest Group(PPIG)*. 2005.

[2] Brown, A.R. and Sorensen, A.C. aa-cell in practice: An approach to musical live coding. In *Proceedings of the International Computer Music Conference*. 2007. Copenhagen, Denmark.

[3] Bryan-Kinns, N. Daisyphone: the design and impact of a novel environment for remote group music improvisation. In *Proceedings of the Conference on Designing Interactive Systems (DIS)*. 2004. Cambridge USA: ACM.

[4] Cage, J. *Imaginary Landscape No. 4*. Composition. 1961.

[5] Collins, N. Live Coding of Consequence. *Leonardo*, 2011. 44(3): p. 207-211.

[6] Collins, N., McLean, A., Rohrhuber, J., and Ward, A. Live coding in laptop performance. *Organised Sound*, 2003. 8(3): p. 321-330.

[7] Cook, P. Principles for designing computer music controllers. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 2001. Seattle, WA, USA.

[8] Downie, M. Field - a new environment for making digital art. *Computers in Entertainment (CIE)*, 2008. 6(4): p. 54.

[9] Essl, G. Circle maps as a simple oscillators for complex behavior: I. basics. In *Proceedings of the International Computer Music Conference (ICMC)*. 2006. New Orleans, USA.

[10] Essl, G. Speeddial: Rapid and on-the-fly mapping of mobile phone instruments. In *Proceedings of New Interfaces for Musical Expression(NIME)*. 2009.

[11] Essl, G. UrMus - an environment for mobile instrument design and performance. In *Proceedings of the International Computer Music Conference*. 2010. New York.

[12] Essl, G. UrSound, live patching of audio and multimedia using a multi-rate normed single-stream data-flow engine. In *Proceedings of the International Computer Music Conference*. 2010. New York, USA.

[13] Essl, G. and Müller, A. Designing Mobile Musical Instruments and Environments with urMus. In

Proceedings of the International Conference on New Interfaces for Musical Expression (NIME). 2010. Sydney, Austrailia.
[14] Essl, G. and Rohs, M. Interactivity for mobile music-making. *Organised Sound*, 2009. 14(02): p. 197-207.
[15] Freeman, J. and Van Troyer, A. Collaborative textual improvisation in a laptop ensemble. *Computer Music Journal*, 2011. 35(2): p. 8-21.
[16] Huron, D. *Sweet anticipation: Music and the psychology of expectation.* 2006 MIT press.
[17] Ierusalimschy, R. *Programming in lua*. 2006; Available from: Lua.org.
[18] Jordà, S. FMOL: Toward user-friendly, sophisticated new musical instruments. *Computer Music Journal*, 2002. 26(3): p. 23-39.
[19] Jordà, S. Digital Instruments and Players: Part I – Efficiency and Apprenticeship. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 2004. Hamamatsu, Japan.
[20] Jordà, S. Multi-user instruments: models, examples and promises. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 2005. Vancouver, Canada.
[21] Kaltenbrunner, M., Geiger, G., and Jordà, S. Dynamic patches for live musical performance. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 2004. National University of Singapore.
[22] Lee, S.W. and Freeman, J. Echobo : A Mobile Music Instrument Designed for Audience To Play. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 2013. Daejeon, Korea.
[23] Lee, S.W., Freeman, J., and Colella, A. Real-Time Music Notation, Collaborative Improvisation, and Laptop Ensembles. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 2012. Ann Arbor, MI, USA.
[24] Linson, A. Unnecessary constraints: a challenge to some assumptions of digital musical instrument design. In *Proceedings of the International Computer Music Conference*. 2011. Huddersfield, UK.
[25] Magnusson, T. Designing constraints: Composing and performing with digital musical systems. *Computer Music Journal*, 2010. 34(4): p. 62-73.
[26] Magnusson, T. ixi lang: a SuperCollider parasite for live coding. In *Proceedings of the International Computer Music Conference*. 2011. University of Huddersfield.
[27] McCartney, J. Rethinking the computer music language: SuperCollider. *Computer Music Journal*, 2002. 26(4): p. 61-68.
[28] McGee, R., Ashbrook, D., and White, S. SenSynth: a Mobile Application for Dynamic Sensor to Sound Mapping. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 2012. Ann Arbor, MI, USA.
[29] Murray-Browne, T., Mainstone, D., Bryan-Kinns, N., and Plumbley, M.D. The medium is the message: Composing instruments and performing mappings. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 2011. Oslo, Norway.
[30] Nilson, C. Live coding practice. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 2007. New York, NY, USA.
[31] Puckette, M. Combining event and signal processing in the MAX graphical programming environment. *Computer Music Journal*, 1991: p. 68-77.
[32] Puckette, M. Pure Data: another integrated computer music environment. In *Proceedings of the International Computer Music Conference*. 1996.
[33] Roberts, C. and Hollerer, T. Composition for conductor and audience: new uses for mobile devices in the concert hall. In *Proceedings of the 24th annual ACM symposium adjunct on User interface software and technology*. 2011. ACM.
[34] Roberts, C. and Kuchera-Morin, J.A. Gibber: Live Coding Audio in the Broswer. In *Proceedings of the International Computer Music Conference*. 2012. Ljubljana, Slovenia.
[35] Roberts, C., Wakefield, G., and Wright, M. Mobile Controls On-The-Fly: An Abstraction for Distributed NIMEs. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 2012. Ann Arbor, MI, USA.
[36] Ruthmann, A., Heines, J.M., Greher, G.R., Laidler, P., and Saulters II, C. Teaching computational thinking through musical live coding in scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education*. 2010. ACM.
[37] Sorensen, A. Impromptu: An interactive programming environment for composition and performance. In *Proceedings of the Australasian Computer Music Conference 2009*. 2005.
[38] Wakefield, G., Smith, W., and Roberts, C. LuaAV: Extensibility and Heterogeneity for Audiovisual Computing. In *Proceedings of the Linux Audio Conference*. 2010.
[39] Wang, G. and Cook, P.R. ChucK: A concurrent, on-the-fly audio programming language. In *Proceedings of the International Computer Music Conference*. 2003. Singapore: International Computer Music Association (ICMA).
[40] Wang, G. and Cook, P.R. On-the-fly programming: using code as an expressive musical instrument. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 2004. National University of Singapore.
[41] Wang, G., Essl, G., and Penttinen, H. Do mobile phones dream of electric orchestras. In *Proceedings of the International Computer Music Conference*. 2008. Belfast, UK.
[42] Wang, G., Misra, A., Kapur, A., and Cook, P.R. Yeah, ChucK it!, dynamic, controllable interface mapping. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 2005. National University of Singapore.
[43] Weinberg, G. and Gan, S.L. The squeezables: Toward an expressive and interdependent multi-player musical instrument. *Computer Music Journal*, 2001. 25(2): p. 37-45.
[44] Weitzner, N., Freeman, J., Garrett, S., and Chen, Y. massMobile – an Audience Participation Framework. In *Proceedings of the International Conferences on New Interfaces for Musical Expression (NIME)*. 2012. Ann Arbor, MI, USA.
[45] Wessel, D. and Wright, M. Problems and prospects for intimate musical control of computers. *Computer Music Journal*, 2002. 26(3): p. 11-22.
[46] Yamashita, Y. *Burning Piano*. video available at http://youtu.be/YpKT_eeCVNI. 2008.