LIVE CODING: AN OVERVIEW

IOhannes m zmölnig *Gerhard Eckel*
University of Music and Dramatic Arts, Graz
Institute of Electronic Music and Acoustics

ABSTRACT

Within the last few years a new performance practice has established itself in the field of intermedia art including computer music: Live Coding. By this a media performance is understood, where the performers create and modify their software-based instruments during the performance. One promise of this technique has been to provide performers with a way of improvising within the realm of algorithms rather than notes, while at the same time offering the audience a primary perception of the the algorithms used by projecting the source code, as opposed to the secondary perception of algorithms in traditional computer music by means of music alone. Another aspect is the development and demonstration of a technical mastery of the instrument "software".

In this paper we try to give an overview on the evolution of Live Coding within the last decade.

1. INTRODUCTION

Programs should be written for people to read,
and only incidentally for machines to execute.[3]

Historically, computer music and tape music are closely related. This is not only true for tape music created with the help of computers, but also for many so-called real-time computer music performances, at least from the audience's point of view. Often, there is not much difference between a laptop-performer who starts the playback of a multi-track recording and then awaits the end of a piece, and a laptop-performer who starts dsp-engines, modifies effects and parameterises algorithms controlling the former in real-time.

While it is often not so interesting for the audience to watch pale faces illuminated by computer screens, it is at the same times often not very interesting for the performers to play their (however complex) systems and algorithms with the limited interface of keyboard, mouse and fader-boxes. [1]

The performers are almost exclusively the same people who have designed and written the software instruments in countless hours. The traditional separation into composer, instrumentalist and instrument maker is not valid for them anymore. And since these people spend most of their time at the design of their instruments (which, due to the power of general purpose machines, are not "just" instruments but can also hold scores and algorithms, which will eventually form the "composition"), it is only logical that this is the field where they gain the greatest skill and virtuosity: the design of algorithms and their implementation in source code.

It was only a matter of time until these people started to utilise their specific skills and explore them in live performances, in club concerts and at experimental music festivals.

Writing code in front of the audience ideally gives the performers an intellectually challenging way to play and improvise with their instrument. From the audience's point of view, the performers are also physically involved into the making of music, at least they are struggling with the computer in a perceivable way. In order to perceive not only the fact *that* the performers are doing something but also *what* they are doing, it helps to provide the audience with some secondary information about the code, for instance by projecting it onto a video wall.

2. A SHORT HISTORY

Emerging from the club scene at the end of the 1990s, a number of prominent Live Coding performers soon organised themselves into an international consortium "for the proliferation of live audio programming": *TOPLAP* [5, 2].

However, traces of Live Coding can be found far earlier.

2.1. The Early Days

Being an art form heavily depending on general purpose computers, not much Live Coding can be found before the 1950s. Some Authors [2] argue that the tournament on cubic equations between the two Italian mathematicians Nicolo Fontana Tartaglia and Antonio Maria Fior about 1539 might be considered an early Live Coding performance (albeit it lasted for several weeks and is thus not directly comparable to today's short-lived performances).

With the advent of the microcomputer in the 1970s, computers finally became small enough to fit on stage, where they could be used as performance instrument. One of the first known Live Coding sessions is attributed to Ron Kuivila, performed at STEIM, Amsterdam, in 1985[4].

"The Hub", one of the early experimental network computer bands in the 1980s [6], allowed the audience to per-

[1] This does not deny all the efforts being undertaken in the field of New Interfaces for Musical Expression. And doubtless there are performers who are happy with limited interfaces that make it quite impossible to break the entire setup during performance.

ceive their decision making by giving them read access to their monitors:

> "The Hub's composers [...] allowed us to walk around, observe their computer screen messages, and assuage our curiosity."[11]

The languages of choice at that time were *Lisp* dialects (the Hub) and *Forth* (the Hub and Kuivila).

2.2. The Dark Ages

In the 1990s, the energies of the computer pioneers to do Live Coding seem to have ebbed away. The hype about the Internet and (related to it) Open Source/Free Software seems to have bound most of the creative potential within the hands of software experts. However, both the FLOSS movement and the Internet have contributed much to the development and the public perception of software art[26].

2.3. A New Dawn

At the end of the 1990s, software art (and its sub-category code art) [9] evolved from *net.art*. Exploring the beauty of algorithms, this movement dealt with code as a new form of expression[12].

With the ever-growing power of computers, interpreted languages (or rather: their interpreters) had by now become fast enough to be used to generate audio and even video in real-time.

In 2000, the duo *SLUB* (Alex McLean and Adrian Ward) did their first Live Coding performance (including projection of source code), utilising a self-written environment "hacked together" in languages like Perl and REALbasic.

At about the same time, Julian Rohrhuber did first experiments with Live Coding in SuperCollider.

Since then, an increasing number of people have expressed themselves by the means of source code on stage, which eventually lead to the founding of an organisation dedicated to Live Coding, *TOPLAP*[2].

While at the beginning Live Coding was confined to the *ab*use of general purpose languages (Perl) and the extension of existing computer music frameworks (SuperCollider), soon the development of integrated environments dedicated solely to the Live Coding of sound and multimedia (ChucK, impromptu; see Section 4) started.

2.4. Parallel Evolutions

Naturally, Live Coding cannot be viewed isolated from other, parallel developments: for instance, pioneers like Tetsuo Kogawa have started to do "Hardware Hacking" performances, where physical circuits are soldered in real time in front of an audience[15].

A less "artistic" development can be observed in the computer sciences: agile programming techniques such as *eXtreme Programming* introduced the principle of *Pair Programming*. Here two programmers write software together at one workstation, thus exposing the algorithms directly to an audience and allowing the audience to take part in the evolution of the algorithms.

3. PARADIGMS AND AESTHETICS

In the Lübeck04 manifesto[25], the members of TOPLAP state what they believe is important for Live Coding. The two main issues (apart from the obvious fact that software has to be written in real-time in order to be called "Live Coding") are:

- Live Coding is about algorithms rather than tools
- "Obscurantism is dangerous. Show us your screens."

Adhering to the "Show us your screens" demand, in a usual Live Coding performance, the source code is usually projected - a somewhat simplistic approach that doesn't necessarily guarantee the kind of "open aesthetics" advocated by Wang and Cook[30]. Ensembles like *Powerbooks Unplugged*[1] try to overcome this by letting the performers sit down among the audience and presenting the code not via big video projections but in the privacy of their laptop screens [10].

At a first glance, Live Coding seems to celebrate the performer as a virtuoso, who is in total control of algorithms, source code and the keyboard.

In contradiction to this, several prominent live-coders are following an (anti-)postmodernist aesthetics, trying to overcome the traditional ideas of genius: While Amy Alexander idealises a "goofy" anti-aesthetics[4] as often found with "geeks" within the current software culture[16], Tom Hall and Julian Rohrhuber take a more serious approach by proposing a *slow code* movement, which tries to eliminate the virtuosity of speed typing from Live Coding performances: "The slow code movement is to music what the slow food movement is to cooking."[14]

4. ENVIRONMENTS

In theory, every programming environment with the ability to produce sound and a reasonably fast implementation-execution cycle can be used for Live Coding.

While theoretically it is possible to use compiled languages for Live Coding, in practice the slow edit-compile-run cycle allows too little direct interaction for most improvisers.

Since Live Coding is still at its infancy, there are not many Live Coding systems available yet. Therefore Live Coders either have to write their own Live Coding environment from scratch, or extend existing real-time systems.

Alex McLean and Adrian Ward are well known for using *REALbasic*[7], *Perl*[19] and the Unix command line interpreter *bash*[18] as a Live Coding environment.

Compared to multi-purpose programming languages like *Perl*, (real-time) computer music languages ease the task of creating sound to a large degree.

4.1. SuperCollider & JITLib

Most likely the first environment based on a computer-music system and dedicated to Live Coding (or *Just-In-Time Programming* as it is called here) has been the *Parcel* extension to SuperCollider[17] by Julian Rohrhuber, which was later developed further into *JITLib* [22].

JITLib provides a proxy-system for diverse processes (like synthesis) to be added, modified and deleted at will, while providing a unified way to switch between various processes by means of crossfading[23].

Recent additions to this library also allow several performers connected via a physical network to share and collaboratively manipulate these processes[10].

4.2. ChucK

ChucK[27] is probably the first computer-music language dedicated to and designed for Live Coding (or *On-the-fly Programming* as the authors refer to it). *ChucK* provides language constructs to control, modify and replace *shreds* (tightly synced processes running in parallel at different speeds) programmatically[30]. In addition to these formal constructs, *ChucK* also provides an integrated environment called *Audicle* designed to handle them efficiently and to visualise the structure and system-interaction of the resulting software apart from simply showing the source-code[28] - an important aspect for an audience that is not necessarily code literate.

4.3. Environments for Multimedia

While *ChucK* provides several graphical representations of the live-coded software, it is not meant for creating visual output (yet[29]).

In the meantime, several other Live Coding environments have been developed with a focus on graphics, video and multimedia. Since many of these environments are targeted at VJs producing "visuals", they have inherent multimedia capabilities like basic analysis of incoming sound, in order to tightly couple audio and video.

Other (more direct) multimedia approaches include the exchange of control data between various system-nodes dedicated to different media via a higher level protocol, or the creation of several stimuli from within a true multimedia environment[8].

One common problem of these environments is that both the primary artistic output (the images) and the secondary one (the code that creates them) are visual impressions and thus overlap in the presentation (at least, if it is important that the code is shown to the audience).

One solution to this is to present code and imagery on different screens, eventually with different sizes in order to focus the audience's attention on one of the two representations. Another solution is to integrate the code into the imagery, at the cost of making the source code less readable.

The Thingee and its underlying language *ThingeeLanguage* are based on Macromedia's Director and its scripting language *Lingo*. Contrary to most Live Coding environments where the focus is on the expressivity of the language, *The Thingee* aims at the (eventually software-illiterate) audience that wants to understand what the programmer does and how this translates into an artistic outcome [4].

A more traditional approach (with a focus on the language) is represented by *fluxus*, a *scheme/Lisp* based 3d rendering engine with a special editor for Live Coding[13].

Another *scheme* based environment that is dedicated to both audio and video creation is *impromptu*. Unlike most other environments described here, *impromptu* also provides ways for collaborative Live Coding, where several programmers interact on the code level[24].

4.4. Graphical Environments

Graphical computer music languages, such as Max/MSP or Pure data[21] have the advantage of offering a representation of the source code that is easily accesible by the audience. While it is arguable that graphical programs can be quite complicated to read and hard to understand[20], they offer a certain familiarity to people who are probably not willing to read text-based source code.

Many of these systems provide mature multimedia extensions (Jitter for Max/MSP; GEM, pdp and GridFlow for Pure data) for integrated Live Coding of both audio and video. But none of them offer any specific constructs for handling discontinuities when switching between processes.

5. CONCLUSION

Live Coding has established itself as an alternative to traditional laptop performances. Offering a form of improvisation at an algorithmic level, it provides the audience with an insight into the used algorithms by making the source code visible, while at the same time focusing on the (physical) presence of the performers.

Currently many of the Live Coding performances take place either informally in clubs or more formally in experimental festivals, like the *LOSS Livecode Festival* or the *Linux Audio Conference*. The main focus is still on the joyful exploration of this new technique.

Once this performance practice has grown out of its infancy, it might well be that one day live-coding "software musicians" will be part of traditional ensembles.

6. REFERENCES

[1] Powerbooks unplugged. `http://pbup.goto10.org/`, 2003-. powerbooks2003unplugged.

[2] Toplap homepage. `http://toplap.org`, 2004.

[3] H. Abelson, G. J. Sussman, and J. Sussman. *Structure and Interpretation of Computer Programs*. The

MIT Press, Cambridge, Massachusetts, 2 edition, 1996.

[4] W. Adrian, R. Julian, O. Fredrik, M. Alex, G. Dave, C. Nick, and A. Alexander. *Live Algorithm Programming and a Temporary Organisation for its Promotion*, pages 243–261. README, 2004.

[5] R. Andrews. Real djs code live. *Wired: Technology News*, 2006.

[6] C. Brown and J. Bischoff. Indigenous to the net: Early network music bands in the san francisco bay area. `http://crossfade.walkerart.org/brownbischoff/IndigenoustotheNetPrint.html`, 2002.

[7] N. Collins, A. McLean, J. Rohrhuber, and A. Ward. Live coding techniques for laptop performance. *Organised Sound*, 8(3):321–330, 2003.

[8] N. Collins and F. Olofsson. klipp av: Live algorithmic splicing and audiovisual event capture. *Computer Music Journal*, 30(2):8–18, 2006.

[9] F. Cramer. *Zehn Thesen zur Softwarekunst*, chapter 1, pages 6–13. Künstlerhaus Bethanien, Berlin, 2003.

[10] A. de Campo, A. Vacca, H. Hölzl, E. Ho, J. Rohrhuber, and R. Wieser. Code as performance interface - a case study. In *Proc. of NIME*, New York, to be published.

[11] K. Gann. The hub musica telephonica. *The Village Voice*, (6-23-87), 1987.

[12] G. Gohlke, editor. *Software Art - Eine Reportage über den Code*. Künstlerhaus Bethanien, Berlin, 2003.

[13] D. Griffiths. Live coding of graphics. `http://www.toplap.org/index.php/Live_coding_of_graphics`, 2004.

[14] T. Hall and J. Rohrhuber. Slow code. `http://www.ludions.com/slowcode/`. accessed 2007-04-30.

[15] T. Kogawa. Tetsuo kogawa cooks a fm transmitter. `http://anarchy.translocal.jp/streaming/19911104tkcookstx.ram`, 1991.

[16] L. Konzack. Geek culture: The 3rd counter-culture. In *Proc. of FNG2006*, Preston, England, 2006.

[17] J. McCartney. Rethinking the computer music language: Supercollider. *Computer Music Journal*, 26(4):61–68, 2002.

[18] A. McLean. Angry - /usr/bin/bash as a performance tool. In S. Albert, editor, *Cream*, volume 12. Twenteenth Century, 2003.

[19] A. McLean. Hacking perl in nightclubs. `http://www.perl.com/pub/a/2004/08/31/livecode.html`, 2004.

[20] M. Petre. Why looking isn't always seeing: Readership skipps and graphical programming. *Communications of the ACM*, 38(6):33–44, 1995.

[21] M. S. Puckette. Pure data. In *Proceedings of the International Computer Music Conference*, pages 224–227. International Computer Music Association, 1997.

[22] J. Rohrhuber and A. de Campo. Uncertainty and waiting in computer music networks. In *Proceedings of the International Computer Music Conference*, 2004.

[23] J. Rohrhuber, A. de Campo, and R. Wieser. Algorithms today - notes on language design for just in time programming. In *Proceedings of the International Computer Music Conference*, Barcelona, 2005.

[24] A. Sorensen. Impromptu: an interactive programming environment for composition and performance. In *Proceedings of the Australasian Computer Music Conference*, pages 149–153, 2005.

[25] toplap. Toplap manifesto. `http://toplap.org/index.php/ManifestoDraft`, 2004.

[26] G. Trogemann and J. Viehoff. *CodeArt - Eine elementare Einführung in die Programmierung als künstlerische Praxis*. Springer, Wien, 2005.

[27] G. Wang and P. R. Cook. Chuck: a concurrent, on-the-fly audio programming language. In *Proceedings of the International Computer Music Conference*, 2003.

[28] G. Wang and P. R. Cook. The audicle: a context-sensitive, on-the-fly audio programming environ/mentality. In *Proceedings of the International Computer Music Conference*, 2004.

[29] G. Wang and P. R. Cook. Chuck: A programming language for on-the-fly, real-time audio synthesis and multimedia. In *ACM Multimedia*, 2004.

[30] G. Wang and P. R. Cook. On-the-fly programming: using code as an expressive musical instrument. In *New Interfaces for Musical Expression (NIME)*, Hamamatsu, Japan, 2004.