Vivace: a collaborative live coding language and platform

John[1]

[1]Universal University

Abstract

Live coding is a performance and creative technique based on improvised and interactive coding. Many recent endeavors have focused in livecoding both because of aesthetics and as a way to alleviate performance drawbacks when the musical instrument is a computer. This paper describes the principles and the design of Vivace, a live coding language and environment built with Web technologies to be executed on web browsers. The approach is striking by 1) allowing many performers to code simultaneously; 2) the synthesis of audio and video; 3) a very simple syntax; 4) being a multiplatform software. We also strive to contextualize Vivace by means of historical and usage summaries including a live coding sub-genre.

Introduction

Live coding is an artistic performance and creative technique based upon writing software code in a live and improvised manner [**?**]. It can be used to generate e.g. sound, images, video, lights and poetry although it is prevalent in computer music [**?**]. Mot often, the code is continually changed and projected in a large surface as a way to make it visible to the audience [**?**]. The usage of live coding in non-performative contexts is also reported, such as in sound design and art installations [**?**]. In this paper, we describe Vivace, a live coding language and platform. It emerged from pragmatic and aesthetic needs, as described in the following sections. The software is cross-platform because it is based on web technologies, such as HTML5 and Web Audio API, and is oriented towards video and music rendering. The language is very simple, allowing for a primary goal of Vivace to be achieved: the simultaneous writing of the code by many performers and the audience.

In summary, this paper describes how a number of artistic presentations motivated the language and platform, describes the Vivace language and platform and how this endeavor lead to the creation of a live coding sub-genre called "freak coding" (e.g. by its manifesto [9] and related artists).

A historical outline

In November of 2011, a live coding trio called *FooBarBaz* [1] unleashed its first presentation for a wide audience. Its performers used two instances of ChucK [2] and a dedicated mixing instance composed by Puredata and an analog mixer [1]. Live coding had been gaining world wide popularity [3, 4, 5, 6], while remaining quite untouched in Brazil. To the best of our knowledge, the presentation was the first live coding performance in our country with such a wide audience – almost 5,000 attendees were in the gathering where code was used on-the-fly to create the music they were listening. At the same time, two live coding desktop work-spaces were projected on large screens to the public, following the principles from the TOPLAP manifesto [7].

During the performance, the trio used ChucK in an unconventional way. Instead of writing loops and conditionals, one of the live coders manipulated parameters of audio files by editing lists of numerical values together with mnemonic operations like retrograde and transposition. The other live coder focused on more fluid lines with large sounds with evolving characteristics; this contributed for coherent musical arcs. Audio mixing with Puredata was carried out by the third performer literally using handwaving gestures tracked by a camera and custom color detection algorithms designed by us. Live coders used code templates for quick insertion in the text editors (Vi and Emacs). Other visual resources the performers focused on: Unix "cowsay" gener-

[1]Pictures of the presentation available at `RemovedforanonymizationasrequestedbySBCM`.

ated phrases and animated bouncing balls – stimulating the audience to imitate Rapid Eyes Movements (REM) – on both terminals, i.e. on both screens projected to the audience. The performance was reported as interesting by technicians, artists and the general audience. Nonetheless, it was altogether complex, not to say messy.

Based on the aforementioned elements, and the need for greater simplicity and interactivity with the public, the Vivace was designed as a new live coding language and platform [2] [8]. To avoid software configuration, and to make it easy to share the session and the system, the Web was chosen as the running environment for Vivace. On every new session performed using Vivace, new principles were added into the language and, at the same time, into our artistic approach.

Additional motivation & inspiration: arrange the room, the code is dirty

Vivace is inspired by various live coding languages. The syntax of Vivace, as shown in Figure 1, borrows elements from ixi lang [10] such as the use of sequences to control audio parameters in real time. ABT [11] and FIGGUS [12] were tightly relevant to the development of Vivace as well and we are planning to rewrite some of their components – originally in Python – inside Vivace. ChucK was an influence from the beginning, and Vivace resembles the simplified interfaces that we constructed in ChucK which were often minimized into lists in few lines. Fluxus [13] was also inspirational for the Vivace environment where the code is shown on top of the video frames.

After the creation of the Audio Data API [14] and the most recent Web Audio API [15] – easing real time audio processing inside Web browsers – a collection of audio Web applications started to emerge. The same holds for live coding languages and systems. Thus, in addition to the *desktop languages* above, Vivace was also inspired by recent Web live coding languages and is part of this *family* of Web applications together with Gibber [16], livecoder [17], livecoding.io [18] and livecodelab [19], to mention just a few.

A remarkable difference between Vivace and other languages and environments in the same family is the element of collaborativity. Vivace was built to enable writing code by many hands at the same time, as with the now popular "e-pads" or collaborative real-time text editors [3], a feature which is naturally implementable on the Web. Another difference is the unconcern to be a Turing-complete language. This made the design of Vivace more flexible and closer to musical thinking as opposed to a computing process (a characteristic perceived in ixi lang as well). Vivace is designed to associate the precision of code and the flexibility of artistic expression while maintaining simplicity.

The language (specification) we all speak

Vivace, as a language [4], is a collaborative live coding language with use of extremely simple syntax, mnemonic operations, easy audio mixing, template editing and audio parameters automation. The use of shared code, sounds and images leads to a more complex scenario, thus increasing the possibility of inconsistency of compiled code as well as artistic results.

Vivace is not an imperative language. Instead of routines and procedures to control audio attributes, it uses definitions related with musical scores and the *track paradigm* common on music production software [**?**]. It is natural to musicians (and, as we experienced during performances, also to non-musicians) to understand a sequence of notes, or audio parameters, repeating over and over again, than for-loops and if-chains. In this way, Vivace is a declarative, domain specific language, based on the following principles:

- Names are literals like foo, bar, baz and are defined as the user wants.

[2] Live demonstrations of Vivace are on-line at `http://void.cc/freakcoding` and `http://void.cc/cranio`, ready to be used by everyone using Google Chrome or Apple Safari.

[3] Etherpad on: `http://etherpad.org`.

[4] The complete specification can be found at `https://github.com/automata/vivace/wiki/Language-spec`.

Figure 1: The Vivace environment.

- Music is constituted by voices (instruments).
- Voices have name, timbre and parameters changing along time.
- The language should be simple. One only defines some properties with a set of values (i.e. arrays, dictionaries) making it possible to generate sequences.
- Mnemonic musical operations (reverse, inverse, transpose) on properties by use of syntax sugar: few chars, powerful changes.
- Timbre are signals made by chains of audio generators and filters or video files as described below.
- Parameters are musical notes, amplitudes, oscillator frequencies, delay time and so on.
- Parameters change their values at specific times and for certain durations.

Here is a "Hello, World!" Vivace code [5]:

```
# foo is a simple audio sample,
# oscillator or video file
foo.src = youtube('YOUTUBE_URL')
# defining the video positions
# (in seconds) to be played
foo.pos = [10, 20, 35]
# the durations (as time ratios of a pulse)
# to be played at each position
foo.cdur = [1/2, 1/4, 1/8, 1/16, 1]
```

A voice is defined as *foo* and its parameters are specified using the *dot* operator. Every parameter changes over time as the values written in numerical sequences, surrounded by brackets. A special sequence exists to every parameter. This is essentially all of the Vivace syntax.

There are extra semantics to operate on the sequences. Every sequence accepts operators: mnemonic commands used to reverse, transpose and even replace elements of the sequence based on list comprehensions. Those operations are common in music composition [**?**] and having them as mnemonics makes typing fast and handy for live coding. The next listing presents the standard operators:

```
# one can use operators
foo.pos = [1, 2, 3] reverse
# result is [3, 2, 1]
foo.pos = [1, 2, 3] inverse
# result is [1, 0, -1]
foo.pos = [1, 2, 3] transpose +2
# result is [3, 4, 5]

# list comprehension
foo.pos = [1/i+1 for i in [1, 2, 3]]
# result is [2, 3, 4]
```

[5] YOUTUBE_URL stands for any youtube video url such as `http://www.youtube.com/watch?v=XXX`

```
# or combine both
foo.pos=[5/i for i in [1,2,3]] reverse
# result is [4, 3, 2] as expected
```

Vivace is written in JavaScript to take advantage of Web technologies. To parse Vivace, Jison [20] comes handy, a JavaScript library that clones Flex and Bison functionality as lexer and parser. This flexibility to parse and execute new languages as JavaScript inside every browser opens a remarkable opportunity to experiment with new syntax and semantics for live coding. To make the Vivace editor collaborative we used ShareJS [21] which makes Web applications content live concurrent. In this way it is possible to share Vivace code with any user accessing a common URL. Furthermore, considering the tradition of UI design and development on the Web thanks to HTML and CSS, one can experiment those new languages with fast prototyped UI – a requisite already addressed by live coding languages [22, 23] like Texture [24], Al-Jazari and Betablocker. Along these advantages, it is important to note: every live coding language built on the Web runs everywhere a browser is installed. No firewall chain to bypass for OSC, no software installation and configuration, no dependencies, people just need to type an URL.

Vivace audio and video engine

Before the Web Audio API, the only way to create sound in web pages was using plug-ins. Recently, the Web Audio API enabled real time audio processing on Web browsers [6]. Every functionality is implemented as native code (in C++ and Assembly when appropriate) to guarantee maximum performance. The API is based on a convenient and familiar paradigm: audio unit graphs. Web Audio specifies a collection of nodes (*AudioNode* objects) and routines to connect and disconnect them. While manipulating those nodes we can create a large number of audio applications: synthesizers, filters, analyzers, mixers and even real time audio engines for live coding. This motivated basic explorations of multichannel expansion, filtering and audio effects, controlling an integrated Web audio system.

Every voice in Vivace is represented as a default audio chain such as the one shown in Figure 2. All audio unit parameters within this chain (e.g. pitch, reverb time, high, medium and low channel levels, panner values and gain) can be manipulated editing the code or by sliders on a GUI (Figure 3). This kind of interface is more familiar to musicians, resembling a real mixer, and enables an adequate treatment of voice timbre and spatialization of the sound sources by means of parameters like level of stereophonic channels L and R, quality, central frequency and gain of a 3-band equalization filter, and reverb time control.

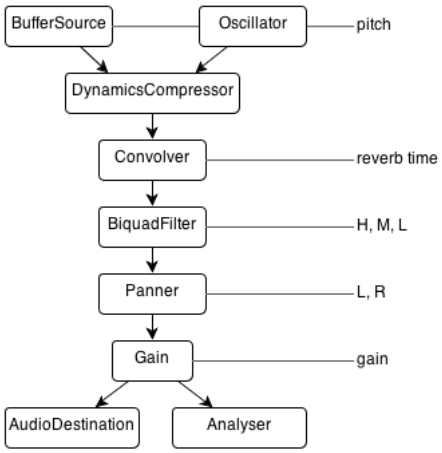

Figure 2: Standard audio unit chains as Web Audio API objects for each Vivace voice.

Vivace supports every audio unit implemented by Web Audio API. It is possible to load audio files or synthesize in real time using wave-table oscillators. The default audio chain of each voice can be modified at any time while it is running. It is interesting to note the presence of an "Analyzer" inside the default chain. It uses FFT (natively implemented) to expose energies and frequencies, enabling the use of those values to animate videos and render graphical forms inside Vivace.

Along with audio, Vivace supports video files. It is possible to upload files or use YouTube URLs. Videos are treated the same way as buffer sources or oscillators, i.e. as voices, and can be

[6] At the time this paper was written, only Google Chrome and Apple Safari supported the API. Mozilla is working to have it running on Firefox as well.

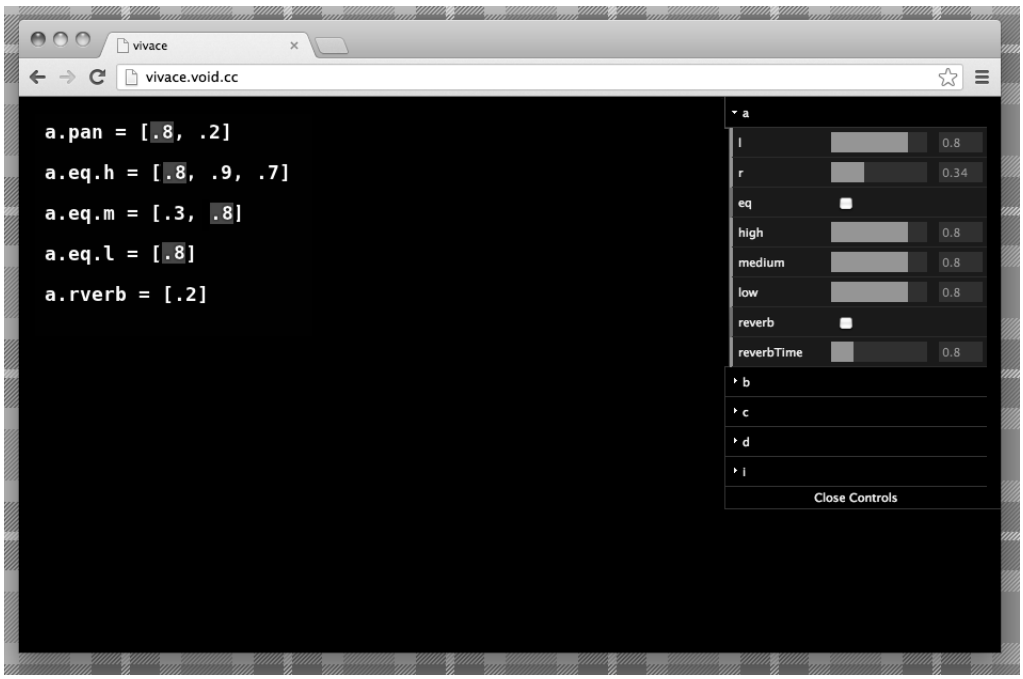

Figure 3: Every audio unit parameter can be manipulated by code or using the UI.

manipulated in real time, making Vivace a live cinema or a VJing tool.

Into the wild: the rise of freak coding makes it collaborative

Vivace as a tool enables interaction while everyone can use their own creativity. The interaction is not mediated by a common score, but by a mutual desire to create a composition in real time. In this context the *freak coder* was born (Figure 4): someone that adds his individuality with others, aiming to transform the computer into an instrument of artistic fruition, without restricting to himself the control of the machine but inviting everyone to join him in the activity. A freak coder decides what he is going to do and amplifies his own comprehension of the computer capacity as an instrument. By using simple rules, Vivace enables the emergence of the performance and makes it a kind of a collective game, where the rules, being visible to everyone through the code, eases audience and specialists alike to join in.

Live coding becomes a natural path to the type of use and technological development in which freak coders are involved, in confluence with the understanding that technology should never be

Figure 4: Freak coder: a live coder who, together with others, uses popular and freak media to gain audience attention to the shared code.

treated as a dogma or kept in secret. Live coding is seen as a behavioral de-alienation of a digital artist. The triad performer, code and audience characterizes the performance as live coding. This comprehension was possible after a presentation by labMacambira.sourceforge.net at the 9^{th} edition of AVAV (*ÁudioVisual Ao Vivo* or Live Audiovisual), an event where artists who are experimenting with audio and video in real time come together to show their works. In this presentation, the authors Caleb Luporini and Gera Rocha started without Renato Fabbri and Vilson Vieira, as they were on their way to the presentation, traveling from another city. Upon arrival, Fabbri and Vilson turned their laptops on and started taking part on the performance in such a way that no embarrassment or rupture was

brought into the event. It is important to state that no previous rehearsal had taken place between Mr. Luporini and Mr. Rocha.[7] In the 30 minutes-long performance, the audience started to take guidance from messages given on the performance large screen and actually edited Vivace code that was being played together with the starting four performers.

Another artifact noted on the presentation was the emergence, in a formal environment [8], of a collective euphoria fertilized by a human-machine interaction. It is the performer's posture that takes a spectator to an experience of a non-spectator and to take part on a highly technological activity as something playful and possible to be assimilated. During the entire presentation, all labMacambira.sourceforge.net members were cheerful and established a relation of lightness and brotherhood with the audience. Spectators were being constantly invited by the posture of labMacambira.sourceforge.net members to interact with what was being proposed. This interplay between the four elements therein present – performers, computer, Vivace and audience – created an environment of collaboration and liberty as generators of playfulness and technical knowledge unheard of, at least in Brazilian live coding, to our knowledge. This is the "facilitator" that emerged and received the name *freak coder*.

To attract the attention of the wider audience, we as freak coders used popular media as material. The code was displayed in front of video scenes sampled from popular Brazilian novels (as in Figure 5) and B-movies, which resulted in a "freak" style, with images of monsters and funny dialogues between novel actors. In other performances for heterogeneous attendees the effect was the same as the first presentation where we used these kinds of pop-art: the people was fascinated by the adherence between the code and the media they see every day on their TV sets. Since then, the use of popular and "freak" media has become a signature of "freak coding".

[7]Videos were selected beforehand by Mr. Luporini alone without knowledge of the other performers.

[8]The four performers were in a light-less room, three of them facing the big screen and the other one facing the public.

All labMacambira.sourceforge.net members take part in the Brazilian free software movement. In a way, freak coding origins should be looked for inside this movement. It is inherent to the free software movement the continued transmission of what is known. The same happens on the demystification of technology and the festive and gregarious behavior. At the performer-computer relation is where this behavior becomes concrete. More than the materials used in the live coding sessions, the performer's stance in relation to the computer – as already expressed in the described presentation – is what really subverts not just the highly technical computer use but the relationship between the human and the machine. Namely a kind of a "rock and roll" stance. The freak coder breaks, by his own nature, the stigma of the computer as the source of a serious and professional posture. In the same way, breaks with the posture of the scholar performer, stern and closed in herself. The freak coding is "rock and roll". The freak coder becomes Jerry Lee of technology making "techno-pyrophagy". He codes and cheers at same time. The freak coder seduces through the computer screen and by the way he codes.

Conclusions and future work

Vivace was motivated by actual performances that took place in the recently emerging Brazilian live coding scene. The development of the language was guided by this direct contact of performers and the wider public. The language was designed and implemented after the identification of common patterns already used on presentations and the need for simplicity and interactivity. Following open source practice, Vivace is developed by many hands from computer scientists, musicians, activists and social scientists. At present, the language is certainly not perfect nor all-encompassing, but it does strike a useful balance between flexibility and rigor, making it an interesting language for artistic expression on collaborative sessions.

It is important to note the advantage of using the Web as the platform for experimentation on live coding and other computer music approaches. Recent APIs like Web Audio together

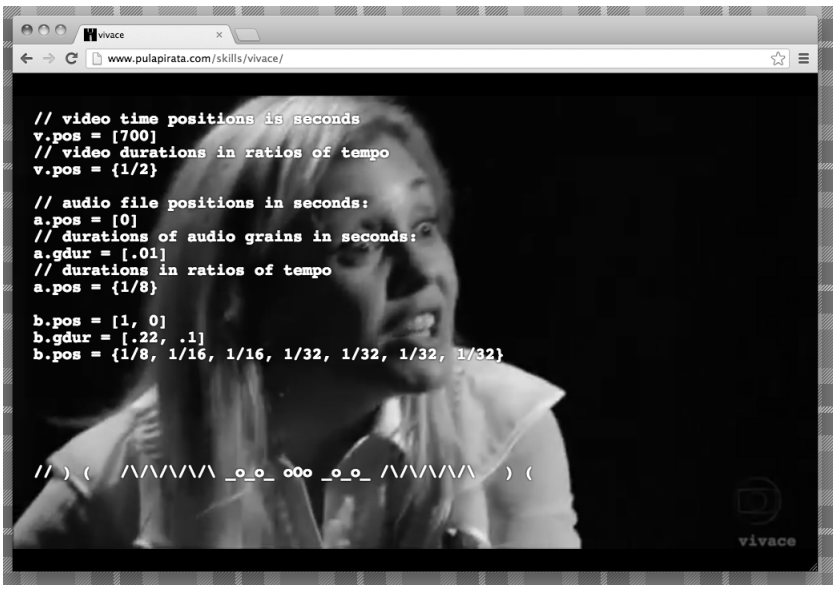

Figure 5: Videos sampled from popular Brazilian novels and B-movies were used as material: extreme pop-art as artifact to attract wider audience attention to the code.

with the rapid prototyping of multi-platform UI and language parsers creates a prolific scenario. Henceforth the most interesting characteristic is the collaboration proportioned by the Web. Using collaborative editors we can expose an entire music program to be edited by anyone, anywhere.

Vivace, although a "freak coding" language, is constrained in its music expression. Having a domain specific language as Vivace is interesting to express some musical ideas where it is hard or even impossible elsewhere. In this context we assume Vivace as one of the many tools and a contribution to create other languages and collaborative systems emerging from live coding practices. In this way, we can tell that the described performances and even Vivace are motivating the creation of other live coding tools. Carnaval [9] is one of these new realizations, it can be seen as a "personal TV channel". Each channel is related to a Vivace instance, making it possible for anyone to remix media and create their own composition. It is a social network of live coded remixes. Vivace, instead of an isolated piece of software is then used as a module, a part of Carnaval.

[9]Carnaval is being conceived as free software and a collaborative art piece since its beginning. The first sketches are on-line at `http://automata.cc/carnaval`

In our experiences as performers and developers of live coding languages we can assert this style of music realization as inspirational and flexible. Nevertheless, we continue to search for improvements on Vivace – and others derived tools – to increase an already consolidated objective of live coding as a musical practice: make computer music performance more human, more interactive with the wider audience [**?**].

Future improvements are planned on Vivace: the possibility to explicitly define large musical arcs as nested sequences related to audio units, the use of 3D graphics APIs to render forms – and relate them to running audio parameters – and text messages to the audience, and improved UI to make the code editing more flexible and reactive [25]. Along with the language and system itself, this paper is a live initiative. Freak coding as an artistic style (a sub-genre of live coding) will be explored more deeply on future studies – regarding its aesthetics – and in already planned performances.

We want to end by underlining the importance of social aspects regarding live coding. The authors, although working on the same collective, were not working close to each other since the creation of Vivace and the rise of freak coding. The performances were provided by the union of those artists. Their union created their own tools

and influenced their own attitude. And the circle starts again.

References

[1] Ricardo Fabbri Renato Fabbri, Vilson Vieira. Foobarbaz: Metasyntactic variables. http://wiki.nosdigitais.teia.org.br/FooBarBaz, 2011.

[2] G. Wang, P.R. Cook, et al. Chuck: A concurrent, on-the-fly audio programming language. In *Proceedings of the International Computer Music Conference*, pages 219–226. Singapore: International Computer Music Association (ICMA), 2003.

[3] C. Nilson. Live coding practice. *Proceedings of New Interfaces for Musical Expression (NIME)*, 2007.

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

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

[6] N. Collins. Live coding of consequence. *Leonardo*, 44(3):207–211, 2011.

[7] A. Ward, J. Rohrhuber, F. Olofsson, A. McLean, D. Griffiths, N. Collins, and A. Alexander. Live algorithm programming and a temporary organisation for its promotion. In *Proceedings of the README Software Art Conference*, 2004.

[8] Vilson Vieira and Renato Fabbri. Vivace. http://automata.github.com/vivace, 2012.

[9] Guilherme Lunhani, Geraldo Magela, Vilson Vieira, Caleb Luporini, and Renato Fabbri. Freak coding manifesto. http://pontaopad.me/freakcoding, 2012.

[10] T. Magnusson. ixi lang: a supercollider parasite for live coding. In *Proceedings of the International Computer Music Conference*. University of Huddersfield, 2011.

[11] Renato Fabbri. A beat tracker. http://trac.assembla.com/audioexperiments/browser/ABeatDetector, 2008.

[12] Renato Fabbri. Figgus: Finite groups in granular and unit synthesis. http://wiki.nosdigitais.teia.org.br/FIGGUS, 2012.

[13] David Griffiths. (fluxus). http://www.pawfal.org/fluxus/, 2013.

[14] David Humphrey, Corban Brook, Al MacDonald, Yury Delendik, Ricard Marxer, and Charles Cliffe. Audio data api. https://wiki.mozilla.org/Audio_Data_API, 2010.

[15] Chris Rogers. Web audio api: W3c working draft. http://www.w3.org/TR/webaudio/, 2012.

[16] Charlie Roberts. Gibber. http://www.charlie-roberts.com/gibber/, 2012.

[17] Fritz Obermeyer. Livecoder. http://livecoder.net, 2012.

[18] Gabriel Florit. livecoding.io. http://livecoding.io, 2012.

[19] Davide Della Casa. Livecodelab. http://www.sketchpatch.net/livecodelab, 2012.

[20] Zach Carter. Jison. http://jison.org, 2010.

[21] Joseph Gentle. Sharejs. http://www.sharejs.org, 2011.

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

[23] T. Magnusson. Algorithms as scores: Coding live music. *Leonardo Music Journal*, pages 19–23, 2011.

[24] A. McLean and G. Wiggins. Texture: Visual notation for live coding of pattern. In *Proceedings of the International Computer Music Conference*, 2011.

[25] Bret Victor. Learnable programming: Designing a programming system for understanding programs. http://worrydream.com/LearnableProgramming/, 2012.