Vivace: A Collaborative Live Coding Language

Vilson Vieira, Guilherme Lunhani, Geraldo Magela de Castro Rocha Junior, Caleb Mascarenhas Luporini, Daniel Penalva, Ricardo Fabbri, Renato Fabbri
Instituto de Física de São Carlos
Universidade de São Paulo
Avenida Trabalhador São-carlense, 400
Parque Arnold Schimidt
São Carlos, São Paulo 13566-590 Brazil
vilson@void.cc, gcravista@gmail.com, gera.sp@gmail.com, calebml@gmail.com, dkajah@gmail.com, rfabbri@gmail.com, renato.fabbri@gmail.com

« AUTHOR TELEPHONE (not for publication): +55 16 8108 7007 »

Abstract

This paper describes the principles and the design of Vivace, a live coding language and environment built with Web technologies to be executed, ideally, in any ordinary browser. It starts by reviewing what motivated and inspired the creation of the language, in the context of actual performances. That leads to specifications of the language and how it is parsed and then executed using the recently created real-time Web Audio API. A brief discussion is presented on why the Web is an environment of interest to collaborative live coding and how it affects the performances. This work concludes by describing how Vivace has motivated the creation of "freak coding", a live coding sub-genre.

In November of 2011, a live coding trio called *FooBarBaz* (Renato Fabbri 2011) unleashed its first presentation for a wide audience. Its performers were using two instances of ChucK (Wang et al. 2003) and a dedicated mixing Puredata + Analog Mixer instance [1]. Live coding has been gaining world wide popularity (Nilson 2007; Collins et al. 2003; Brown and Sorensen 2007; Collins 2011), while remaining quite untouched in Brazil. To the best of our knowledge, that presentation was the first live coding performance in our country – almost 5,000 attendees were in the audience 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 big screens to the public, following the principles from the TOPLAP manifesto (Ward et al. 2004).

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 having evolving characteristics; this contributed with larger musical arcs. Audio mixing with Puredata was carried out by the third performer literally using handwaving gestures tracked by a camera and our custom-designed color detection algorithms. Live coders used code templates quick-inserted by programmer's text editors (Vi and Emacs). Other visual resources the performers focused on were: Unix "cowsay" generated phrases on individual terminals and animated bouncing balls – stimulating the audience to imitate Rapid Eyes Movements (REM). These artifacts – code, "cowsay" phrases, moving REM-like points of reference, all projected in big screens to the public – were incorporated as good practices during the live coding performance.

[1] Pictures of the presentation available at http://www.flickr.com/photos/festivalcontato/6436260557

Based on the aforementioned elements, a new language was designed: Vivace [2] (Vieira and Fabbri 2012). To avoid software configuration and to make it easy to share the session – and the system itself – with everyone, 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 behavior.

In summary, this paper has a circular structure: it describes how actual performances motivated the language and how the language itself influenced the creation of a live coding sub-genre that is being developed by many hands, called "freak coding" by its manifesto (Lunhani et al. 2012), other written resources, and in spoken instances.

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 (Magnusson 2011b) such as the use of sequences to control audio parameters in real time. ABT (Fabbri 2008) and FIGGUS (Fabbri 2012) 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 suggested chained unit generators specified by the $=>$ operator, mimicking the object connections in visual programming languages like Puredata. Fluxus (Griffiths 2013) was also inspirational for the Vivace environment where the code is shown on top of the video frames.

After the creation of Audio Data API (Humphrey et al. 2010) and the most recent Web Audio API (Rogers 2012) – enabling real time audio processing inside Web browsers – a

[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.

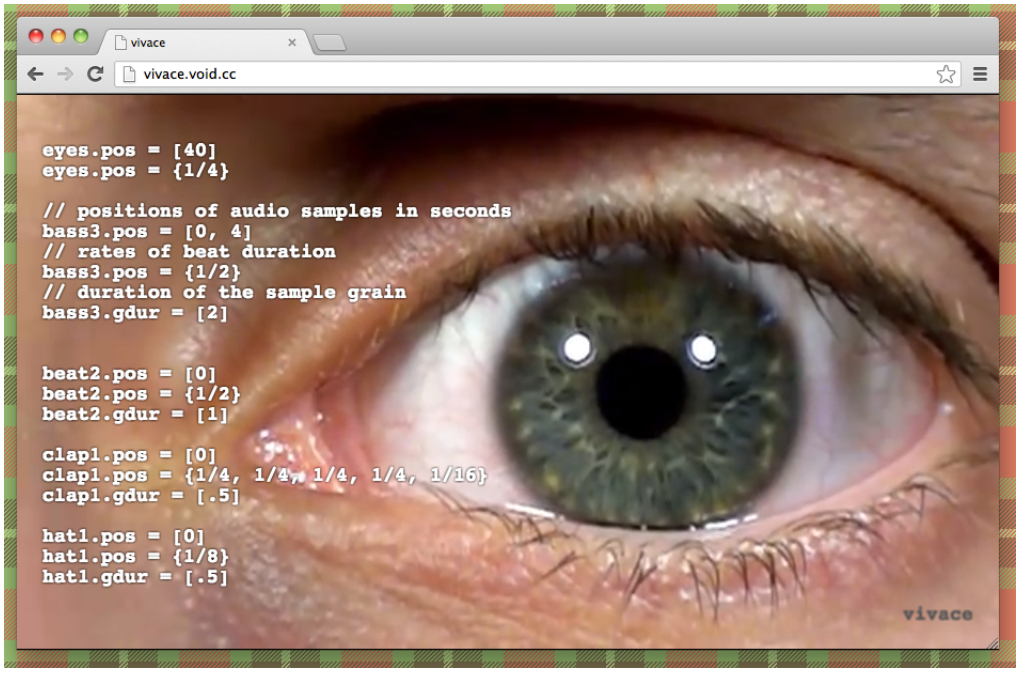

Figure 1. The Vivace environment.

collection of audio Web applications started to emerge. The same holds for live coding languages and systems. Thus, in addition to the *desktop languages* pointed above, Vivace was also inspired by recent Web live coding languages and is part of this *family* of Web applications together with Gibber (Roberts 2012), livecoder (Obermeyer 2012), livecoding.io (Florit 2012) and livecodelab (Casa 2012), to note some of them.

A remarkable difference between Vivace and other languages or environments in the same family is the element of collaborativity. Vivace was built to enable writing code with 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 the musical process rather than the computing process (a characteristic perceived in ixi lang as well). Vivace tries to place itself between the precision of code and the flexibility of artistic expression.

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

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 actions for artistic result. This enhances the fit of specific language syntax.

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 a for-loop 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.
- Music is made by voices (instruments).
- Voices have name, timbre and parameters changing along time.
- The language should be simple. "Freak coders", with a dedicated section below, only define some properties with a set of values (i.e. arrays, dictionaries) making it possible to generate sequences, even using list comprehension.
- Mnemonic musical operations (reverse, inverse, transpose) on properties by use of syntax sugar: few chars, big results.

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

- 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:

```
# foo is a simple audio sample, oscillator or video file
foo.src = youtube('http://www.youtube.com/watch?v=XXX')
# defining the video positions (in seconds) to be played
foo.pos = [10, 20, 35]
# the durations (as time ratios) to play 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 as 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:

```
# we 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]

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

# or combine both
foo.pos = [1/i+1 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 (Carter 2010) 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 (Gentle 2011) which make Web applications content live concurrent. In this way it was 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 (McLean et al. 2010; Magnusson 2011a) like Texture (McLean and Wiggins 2011), 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 [5]. 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 to 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 spacialization 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.

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

[5] 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.

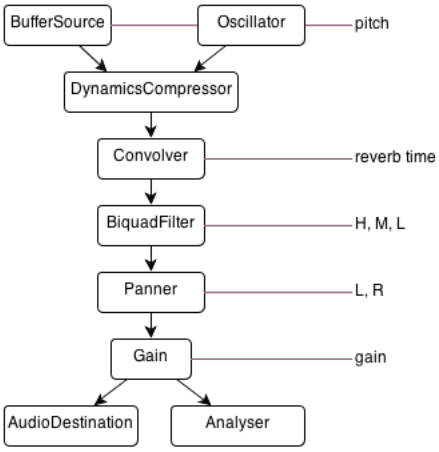

Figure 2. Standard audio unit chains as Web Audio API objects for each Vivace voice.

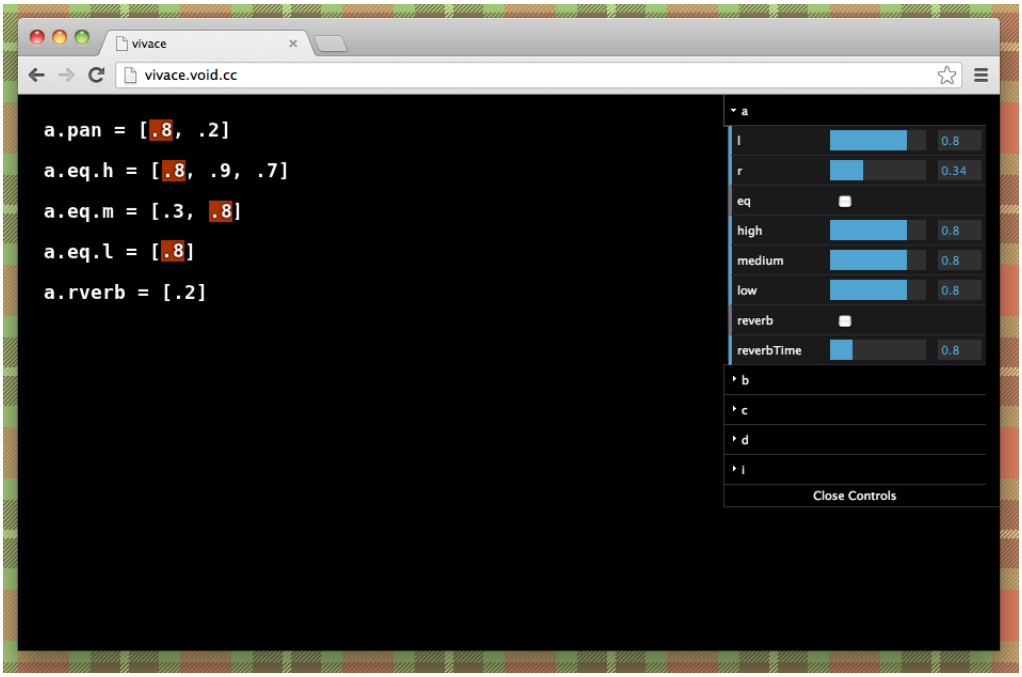

Figure 3. Every audio unit parameter can be manipulated by code or using the UI.

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 manipulated on 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 could 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 is born the *freak coder* (Figure 4): someone that adds his individuality up to 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 same 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, 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 treated as a dogma or kept in secret. Live coding is seen as a behavioral de-alienation of a digital artist. Not only the code is displayed and manipulated, but the computer screen and potentially any interaction between the performance and the computer. The triad performer, code and audience characterizes the performance as live

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

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 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, they both 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.[6] In the 30 minutes-long performance, the audience started to take guidance from messages given on the performance big 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 [7], of a collective euphoria fertilized by a man-machine interaction. It is the performer's posture that takes a spectator to an experience of a non-spectator and to take

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

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

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 – was able to create an environment of collaboration and liberty as generators of playfulness and technical knowledge unheard of, at least in Brazilian live coding terms. 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, what resulted in a freak style, with images of monsters and funny dialogs between novel actors. In other performances for heterogeneous attendees the effect was the same as the first presentation where we used those kinds of pop-art: the people was fascinated by the adherence between the code and the media they see every day on their TV set. Since then, the use of popular and freak media has become a signature of freak coding.

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 man and the machine. Namely a kind of a "rock and roll" stance. The freak coder

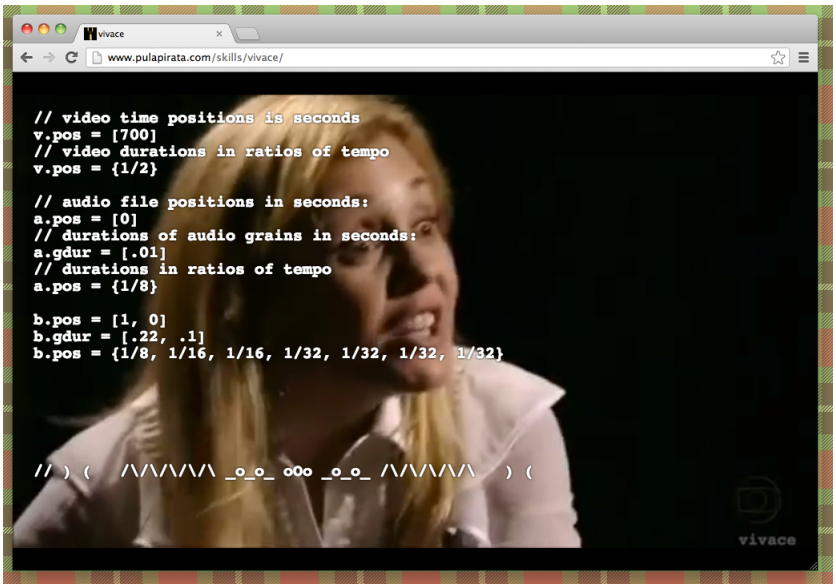

Figure 5. Videos sampled from popular Brazilian novels and B-movies were used as material: extreme pop-art as artifice to attract wider audience attention to the code.

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. Following open source practice, Vivace is being developed by many hands from computer scientists, musicians, activists and social scientists. At present, the language is not perfect nor all-encompassing, but it does strike a useful balance between flexibility

and rigour, 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 developments. Recent APIs like Web Audio together 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 while others become hard or even impossible to be expressed. In this context we assume Vivace as one of many tools and a first step to create other languages and collaborative systems, emerging from live coding practices. In this way, we can tell that those performances and even Vivace are motivating the creation of other live coding tools. Carnaval [8] is one of these new realizations, it can be seen as a "personal TV channel". Each channel has a Vivace instance running on it, making 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, part of Carnaval.

In our experiences as performances 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 the already consolidated objective of live coding as a musical practice: make computer

[8] Carnaval is being conceived as free software and a collaborative art piece since its beginning. The first sketches are on-line at http://pontaopad.me/vj

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 – passive to be changed by currently running audio parameters – and text messages to the audience, and improved UI to make the code editing more flexible and reactive (Victor 2012). Along with the language and system itself, this paper is a live initiative. Freak coding as an artistic style will be explored more deeply on future studies – regarding its own 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

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

Carter, Z. 2010. "Jison." http://jison.org.

Casa, D. D. 2012. "Livecodelab." http://www.sketchpatch.net/livecodelab.

Collins, N. 2011. "Live Coding of Consequence." *Leonardo* 44(3):207–211.

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

Fabbri, R. 2008. "A Beat Tracker." http://trac.assembla.com/audioexperiments/browser/ABeatDetector.

Fabbri, R. 2012. "FIGGUS: Finite Groups in Granular and Unit Synthesis." http://wiki.nosdigitais.teia.org.br/FIGGUS.

Florit, G. 2012. "livecoding.io." http://livecoding.io.

Gentle, J. 2011. "ShareJS." http://www.sharejs.org.

Griffiths, D. 2013. "(fluxus)." http://www.pawfal.org/fluxus/.

Humphrey, D., C. Brook, A. MacDonald, Y. Delendik, R. Marxer, and C. Cliffe. 2010. "Audio Data API." https://wiki.mozilla.org/Audio_Data_API.

Lunhani, G., G. Magela, V. Vieira, C. Luporini, and R. Fabbri. 2012. "Freak Coding Manifesto." http://pontaopad.me/freakcoding.

Magnusson, T. 2011a. "Algorithms as Scores: Coding Live Music." *Leonardo Music Journal* :19–23.

Magnusson, T. 2011b. "ixi lang: a SuperCollider parasite for live coding." In *Proceedings of the International Computer Music Conference*. University of Huddersfield.

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

McLean, A., and G. Wiggins. 2011. "Texture: Visual Notation for Live Coding of Pattern." In *Proceedings of the International Computer Music Conference*.

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

Obermeyer, F. 2012. "LiveCoder." http://livecoder.net.

Renato Fabbri, R. F., Vilson Vieira. 2011. “FooBarBaz: Metasyntactic Variables.” http://wiki.nosdigitais.teia.org.br/FooBarBaz.

Roberts, C. 2012. “Gibber.” http://www.charlie-roberts.com/gibber/.

Rogers, C. 2012. “Web Audio API: W3C Working Draft.” http://www.w3.org/TR/webaudio/.

Victor, B. 2012. “Learnable Programming: Designing a programming system for understanding programs.” http://worrydream.com/LearnableProgramming/.

Vieira, V., and R. Fabbri. 2012. “Vivace.” http://automata.github.com/vivace.

Wang, G., P. Cook, et al. 2003. “ChucK: A concurrent, on-the-fly audio programming language.” In *Proceedings of the International Computer Music Conference*. Singapore: International Computer Music Association (ICMA), pp. 219–226.

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