CACHARPO:
CO-PERFORMING CUMBIA SONIDERA WITH DEEP ABSTRACTIONS

Luis Navarro Del Angel
McMaster University
navarrol@mcmaster.ca

David Ogborn
McMaster University
ogbornd@mcmaster.ca

Abstract

This paper describes Cacharpo, an autonomous agent that serves as an assistant within live coding performance. Cacharpo uses techniques of Machine Listening and Music Information Retrieval to "hear" musical parameters made by a human live coder and respond to them by typing code and making music. The development of Cacharpo addresses the challenge that typing as a physical act can take excessive time undermining the theatricality of the performance. Cacharpo's main focus is music influenced by cumbia sonidera, a genre developed in Mexico and derived from Colombian cumbia. Cacharpo consists of four subsystems: a low-level feature extractor, an Artificial Neural Network to identify cumbia roles and other musical parameters, an algorithm to generate SuperCollider code, and SuperCollider classes providing notations for economical live coding of cumbia sonidera elements.

1 Introduction

Live coding is the practice of making and unmaking software in order to generate expressive material, such as music and image. Although programming without typing is possible, live coding practice often relies on typing interfaces, such as the keyboard (TOPLAP 2010). A common challenge in live coding performance is that typing as a physical act takes time, especially when starting from a blank screen (Brown and Sorensen 2007). This is not only a problem at the beginning of a musical performance — melodic, harmonic and/or rhythmic changes also take time, which can result in monotony. Because the live coder is constantly occupied, typing as fast as they can to produce music, the theatricality of the performance can be undermined (Brown and Sorensen 2009). To address this challenge, live coders employ strategies such as collaborative coding and the use of abstractions that let them start quickly and make quick changes.

Following these traditions, Cacharpo is an autonomous electronic music improviser and live coder that functions as a co-performer making timely musical changes. Cacharpo generates music and code through high-level abstractions wrapping SuperCollider's pattern system and the use of multiple layers of semantic mapping of low-level audio inputs. Cacharpo's main focus is to make music influenced by the cumbia sonidera genre developed in the central part of Mexico and derived from Colombian cumbia. This paper discusses the motivation, fundamental concepts, and architecture of Cacharpo. The paper concludes with a discussion of directions for ongoing and future development of the system.

2 Motivation and Fundamental Concepts

2.1 RGGTRN and cumbia sonidera

RGGTRN is a collective whose interests include live coding improvisation influenced by Latin American dance music and its recontextualization into the realms of electronic dance music. It was

founded in 2012 and it first debuted as a duo named ~ON at the *VIVO* International Symposium in Mexico City. The members of the collective are distributed across Mexico and Canada and include Jessica Rodríguez, Marianne Teixido, Emilio Ocelotl and the first author of this paper. RGGTRN explores music genres such as Reggaeton, Tribal, and Cumbia sonidera.

Cumbia sonidera is derived from Colombian cumbia and was mostly developed during the 1980s in the central part of Mexico. Its orchestration consists of acoustic instruments (e.g. drum set, timbales, congas, guiro, brass instruments, etc.), electronic instruments (e.g. electric guitars, electric bass, and synthesizers), and sound systems characteristic of DJ culture. Precedents for Cumbia sonidera can be traced from the 1960s onwards with early genres including "Cumbia tropical/tropical Cumbia" and "Cumbia mexicana/Mexican Cumbia". Many covers of traditional Colombian cumbia were recorded by Mexican artists and international record labels during the 1970s reaching the central and northern part of Mexico as well as the south of the United States. Nevertheless, many of the original composers were not credited in many of these recordings to avoid paying royalties, perpetuating a false idea that these compositions were made by Mexican composers (Blanco Arboleda 2012). Despite the latter, it is important to highlight that cumbia sonidera is a cultural manifestation ingrained in working-class neighborhoods of Mexico and it is considered a cultural heritage by the communities formed around this music (Ramírez Cornejo 2012).

Cumbia sonidera can be performed both by DJs alone (e.g. Sonido La Changa and Sonido Rolas) and by additional musical performers (e.g. Los Ángeles Azules, Grupo Soñador, and Celso Piña). In both cases, groups of instruments with defined musical roles can be heard. The percussion section are in charge of the rhythm, bass, keyboard, electric guitar, and brass manage the harmony, and the lead voice or melody can be provided by a singer or any other melodic instrument. DJs can take part in any of these musical roles, but their main role is to speak on top of the music, mentioning the name of the band, sending greetings to the people, making jokes, or even mocking corrupt politicians (Blanco Arboleda 2012).

Because the members of RGGTRN live in two different countries, each of them often performs solo. Typing and developing cumbia sonidera roles is difficult under these circumstances as much typing is required to generate constant changes with no guarantee that they will be timely and dynamic. The focus of this research project, then, is to introduce an autonomous agent capable of performing, with one or more members of RGGTRN, the distinct musical roles and elements of the cumbia sonidera genre. This project aims to increase the theatricality of RGGTRN's performances and the enjoyment of its audience.

2.2 Theatricality in live coding

Live coding is about recognizing the conflicts between the human, the machine and the source code as inherent to the process of creation with automatic systems. For the live coding audience, entertainment comes not only from the music but also from observing this conflict and resolution. It is the struggle of the coder when trying to control the machine, and to bend the algorithm, that the audience perceives as dramaturgical. The live coder, too, feeds on multiple sources of information (i.e. audience, co-performers or the code itself) which could change the final course of the piece. Live coding performance, nevertheless, happens in a place of relative safety where unpredictability is calculated and contained. The amount of liveness in performance, and therefore, the number of theatrical possibilities, both depend on how secure or unpredictable this place is (Norman 2015).

Moreover, the interplay between the coder(s) in-scene, the machine and the audience generates a unique moment where exact repetition is avoided (Attali 1985; Auslander 2002). Liveness in music — more variation and less repetition — is difficult to achieve. A live coder needs to type fast, to think forward to devise the musical ideas and to be able to describe them as code in order to generate quick but opportune changes. Liveness in music performance depends not only on the amount of security or unpredictability it has but also on how dynamic and timely are the musical changes that happen within it.

Typing is constrained by the human body — our fingers have a typing speed limit — and the discreteness of written language (Ludwig 1983). To provide a meaningful instruction, the coder needs to write a complete error-free sequence of letters and symbols in order to be properly understood by the machine. Depending on the length of the instruction and the characters per minute the coder is capable of, typing can take excessive time. This leads to losing valuable time, depriving the audience of enjoying melodic, harmonic and/or rhythmic changes in the music and increasing the stability and monotony of the performance.

2.3 Strategies to address typing constraints

One way to address these constraints is reducing the amount of typing. Strategies to do this include collaborative and networked coding (Brown and Sorensen 2007; Freeman and Van Troyer 2011; Lee and Essl 2014a; Lee and Essl 2014b; McKinney 2014; Ogborn 2014), where each member of the group focuses on fewer tasks, typing less but developing musical layers more deeply. The use of abstractions is another strategy to reduce the amount of typing. If well conceived and implemented, they enable the live coder to produce rich musical content with few lines of code. Abstractions made during performance are often smaller, anonymous, and ephemeral. Abstractions prepared ahead of performance are potentially longer, detailed, and deeper. On the other hand, abstractions are generally complex, opaque, and difficult to understand for both the audience and the performer as developing them involves hiding data.

Autonomous agents (Cope 1987; Eacott 2007; Lewis 2000; Polansky 1975; Roads 1985; Wamser and Wamser 31996) combine the idea of collaboration and the use of abstractions, therefore becoming another strategy to reducing typing load. Moreover, through Machine Listening they can mimic cognitive processes (Collins 2006) helping them interact with a human live coder in ways analogous to those of human co-performers (McLean and Sicchio 2014; Stowell 2010; Yee-King 2011).

3 Implementation

Cacharpo is an autonomous live coder developed to address these research concerns. Cacharpo “hears” the music made by its co-performer(s) and responds by generating code and music in SuperCollider. Machine Listening and Music Information Retrieval provide Cacharpo enough flexibility to adapt to structures, harmonies, and synthesizers that could be used by the human co-performer (Figure 1).

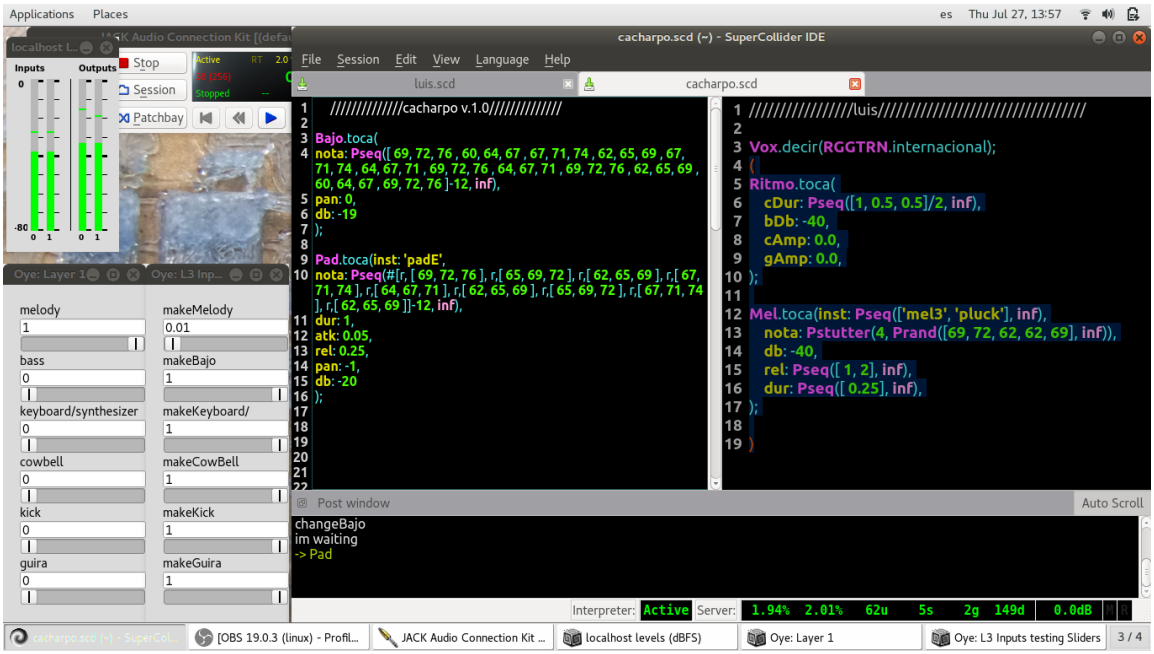

Figure 1: Co-performing with Cacharpo

Cacharpo's system consists of four connected subsystems developed specifically for this project: a low-level audio feature extractor, an Artificial Neural Network (ANN) to identify cumbia roles and other musical parameters, an algorithm to generate SuperCollider code, and SuperCollider classes providing notations for economical live coding of cumbia sonidera elements.

The connections between these subsystems were informed by the concept of three-layer mapping proposed by Hunt and Wanderley (2002). This paradigm was helpful to this project as it encourages the organization and analysis of the data before its use and highlights that perceptual analysis should validate the correctness of the mapping. In Cacharpo's system, a first layer of mapping generates high-level, "semantic" music features from low-level audio features. In the third layer, high-level "semantic" descriptions of musical intentions and constraints are mapped on to lower-level behaviours and variables of a code generation system. The second layer connects first layer outputs and third layer inputs (Figure 2).

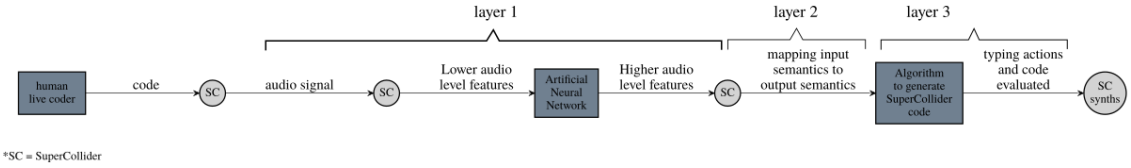

Figure 2: A flowchart of the system

3.1 Low-level audio feature extractor

This subsystem uses both Machine Listening and Music Information Retrieval (MIR) algorithms implemented as native functions in SuperCollider. In this subsystem, stereo sound resulting from the human live coder's activity is routed to a unit generator (or "Ugen", i.e. an algorithm that consumes and/or produces audio signals) which performs a Fast Fourier Analysis to decompose it into a range of

frequencies. This spectral information, then, is routed to MIR UGens which calculate the amount of "brightness" (spectral centroid), the amount of "noisiness" (spectral flatness), when a note begins (onset detection), and the distribution of the energy in different bands of frequencies (spectral percentile, high, low and pass-band filters). Additionally, how often the signal intercepts the zero axis (zero-crossing rate) is calculated from the monophonic audio stream.

Twenty four audio features are continuously extracted at a rate of twenty times per second. Once this basic analysis is done, this data is sent through Open Sound Control (OSC) messages to an application that parses and organizes the data to route it to an Artificial Neural Network (ANN) in real-time.

3.2 Artificial Neural Network (ANN) to "hear" musical parameters

To answer Cacharpo's questions "is a bass playing?", "is there a melody?", "which pitches should I use", and so forth, multiple ANNs were used. Twenty times per second high-level musical features calculated by the ANNs are sent into the second layer of mapping where they are then remapped onto high-level parameters of the third, output layer of the system. These features enable Cacharpo to identify the presence of cumbia sonidera roles (i.e. rhythm, harmony, and melody), instruments (i.e. percussions, bass, keyboard, synthesizers), pitch sets, and the amount of "energy" during an ongoing performance.

The ANNs for this system were developed with Haskell, a pure functional programming language that has been used in projects connected to algorithmic composition and sound synthesis (Drape 2015; Hudak 2015; Hudak and Wadler 2007; Quick and Hudak 2013) as well as live coding (McLean 2014; McLean and Wiggins 2010; Murphy 2016a; Murphy 2016b; Ogborn et al. 2015). ANNs are possible to develop using SuperCollider as demonstrated by the SCMIR library (Collins 2011). Haskell, however, has valuable characteristics — leading to robust, consistent, and reusable software — such as the possibility to prove program properties, step-by-step evaluations and calculations, symbolic testing, and an exhaustive debugging feedback system (Thompson 1999). In addition to this, intuition suggested that the training time, the usage of RAM, and the management of big sets of data has been more efficient using this programming language.

To train the ANNs, an offline training (i.e. using a static dataset) was performed with data collected from sound recordings of Cumbia sonidera SuperCollider performances, featuring both sampled and synthesized sounds. Each recording lasted between three to six seconds. For the ANNs to be able to include information from the past two seconds in their calculations the input to the network consisted of the current low-level audio features as well as forty delayed versions of the same low-level features. Audio features extracted at twenty times per second were stored in lists including information from forty frames behind (Table 1).

Table 1. ANNs used by Cacharpo, their audio features and, input values.

Network Name	**Network task**	**N° of audio features used**	**Audio features used**	**N° of input training sets used**
Melody	Identify the role of melody	9	Spectral centroid, spectral flatness, zero-crossing rate, onset detection, spectral percentile, high, low, and	1885

			band-pass filters	
Teclado	Identify the role harmony	9	Spectral centroid, spectral flatness, zero-crossing rate, onset detection, spectral percentile, high, low, and band-pass filters	1400
Bajo	Identify the role of the bass	9	Spectral centroid, spectral flatness, zero-crossing rate, onset detection, spectral percentile, high, low, and band-pass filters	1407
Kick	Identify the role of the rhythmic section	9	Spectral centroid, spectral flatness, zero-crossing rate, onset detection, spectral percentile, high, low, and band-pass filters	1362
Guira	Identify the role of the rhythmic section	9	Spectral centroid, spectral flatness, zero-crossing rate, onset detection, spectral percentile, high, low, and band-pass filters	1882
Cowbell	Identify the role of the rhythmic section	9	Spectral centroid, spectral flatness, zero-crossing rate, onset detection, spectral percentile, high, low, and band-pass filters	1997
Pitch recognition	Identify pitches	12	Band-pass filters	2280
Energy	Identify the amount of energy of the ongoing performance	3	Loudness, onset detection, and high-pass filter	435

For the ANNs to synchronously identify high-level musical parameters, desired output parameters were also stored in the training file. These desired outputs described when a “melody”, “keyboard”, “guiro”, “cowbell”, “kick”, and/or “bass” were present in the music, as well as the individual pitches of the chromatic scale, and the energy of the ongoing performance. For each sound recording, these desired outputs were stored twenty times per second in arrays with no delay added (Table 2).

Table 2. Desired outputs used to train Cacharpo’s ANNs.

Network Name	**N° of desired outputs**	**Purpose of the desired outputs**
Melody	1	Identify three synthesizers used to make melodies.
Teclado	1	Identify two synthesizers used to make chords.
Bajo	1	Identify one synthesizer used to make a bass line.
Kick	1	Identify one sample used to make a kick.
Guira	1	Identify one synthesizer used to make a guira.

Cowbell	1	Identify one synthesizer used to make a cowbell.
Pitch recognition	12	Identify pitches of the chromatic scale in 4 octaves.
Energy	1	Identify high, Mid and low levels of energy.

The hnn Haskell library (Mestanogullari and Johnson 2009) was used to implement the ANNs. Based on the size of the lists generated, the structure of the ANNs used to identify roles and instruments resulted in three hundred sixty input neurons and one output neurons for each of them. The ANN used to recognize pitches resulted in four hundred eighty input neurons and twelve output neurons. The energy ANN resulted in one hundred twenty input neurons and one output neuron. One hidden layer of forty neurons was sufficient to convert low-level inputs into high-level outputs (Figure 3). The algorithm used for training was feedforward and the method of training was backpropagation, both frequently used to perform tasks such as the categorizations required for this project.

During training, the adaptability of the ANNs was measured through cross-validation techniques. This measurement was important because it calculates how well the ANNs will perform under circumstances different than the ones provided in the training samples. Cross-validation was performed with data collected from sound recordings of Cumbia sonidera SuperCollider performances featuring the same sampled and synthesized sounds of the training data (Table 3). Using the same sounds to cross-validate the training was necessary as the synthesizers used for this project are hand-made and idiosyncratic (Magnusson 2010).

Table 3. Cross-validation results of Cacharpo's ANNs.

Network Name	Quadratic error of training samples	Quadratic error of non-training samples
Melody	0.0184% per sample	0.5873% per sample
Teclado	0.0863% per sample	0.3756% per sample
Bajo	0.0072% per sample	0.3595% per sample
Kick	0.0070% per sample	0.4264% per sample
Guira	0.0954% per sample	0.3429% per sample
Cowbell	0.0538% per sample	0.2833% per sample
Pitch recognition	0.0023% per sample	0.016% per sample

3.3 An algorithm to generate SuperCollider code

Once Cacharpo has "heard" its first sound, a subsystem to generate SuperCollider code is activated. In this system, high-level "semantic" musical intentions and constraints are mapped on to lower-level behaviours and variables of a code generation system. Cacharpo, then, decides which cumbia roles and instruments it will play and in which key. For instance, if there is already a bass playing, then, it can choose to make a melody, or to play an instrument from the rhythmic section like a kick or a cowbell.

Cacharpo's generation of code is guided by a finite-state machine (Gill 2007). There are three states in the machine: "waiting", "deciding", and "typing" (Figure 3). "Waiting" happens when the system has been initialized and is waiting for input. It also happens after Cacharpo has finished typing, has evaluated the code in SuperCollider, and is waiting for the audience to hear its result.

"Deciding" happens when the system has been initialized and has already been waiting for five seconds. If nothing has been heard still, then it decides to wait again. Conversely, if something has been heard, it presents itself, picks a cumbia role, and types and plays it. After the system has typed its first role and a period of time has passed for the audience to hear it, Cacharpo can decide to pick another role or to make changes to the one(s) already written. If the density and the loudness of the music are decreasing, Cacharpo decides to gently start stopping the music and deleting the code.
"Typing" happens after any decision ("wait" not included) has been taken. Decisions taken are translated to typing actions (i.e. move the cursor, insert, replace, and/or delete a character) in order to visualize them in SuperCollider. These actions are sorted and queued by the algorithm for Cacharpo to know what, when and how to type.

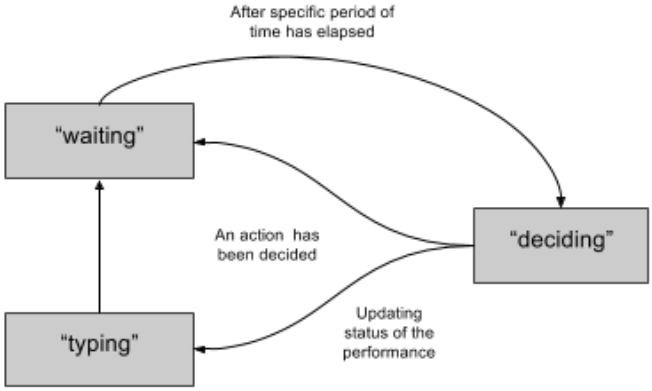

Figure 3. State transition diagram.

3.4 Notations for economical live coding of cumbia elements

Melodía, Teclado, Bajo, and Ritmo, are a set of miniature SuperCollider classes that function as an interface for quick generation of live coded music and are both used by the human live coder and the code generating agent. Each one of them can produce a basic layer of music, such as melody, harmony, or rhythm. Moreover, they are intended to represent music roles within the genre of Cumbia sonidera.

They were developed so that the audience would have to wait less in order to hear an initial cumbia sound. These classes include methods that wrap calls to underlying Pbindef methods where arguments such as instrument, pitch, amplitude, attack, release, duration, panning, and strum can be accessed and modified. An example of calling methods from the Ritmo and Bajo classes is provided below (Figure 4).

```
//A kick drum, a cowbell, and a guiro playing a basic cumbia pattern.
Ritmo.toca (bomboDb:0.75, campanaDb:0.5, guiroDb: 0.25);

//A bass pattern playing C as a quarter note and E and G as eighth notes.
Bajo.toca (nota:Pseq([60, 64, 67], inf), dur: Pseq ([1, 0.5, 0.5], inf));
```

Figure 4. The Ritmo and Bajo SuperCollider classes.

4 Discussion and Future Work

Cacharpo reduces the number of tasks and the amount of typing the human coder needs to do by taking musical roles, typing code, and making music. It also potentially increases the attentiveness of the audience by making consistent and timely changes to the music. The system does not require direct input from the human coder, such as chat messages or spoken directions, and its structure is simple and easy to maintain. Implementing an "ecology" of ANNs has enabled them to perform more specific tasks (i.e. cumbia roles, pitch recognition, and "energy" detection), therefore improving their accuracy. Long-term future development includes increasing the adaptability of the system to sounds, structures, and harmonies of music genres such as Reggaeton and Tribal. This will be done by training more ANNs, developing more SuperCollider music notations, and expanding the decisions the generative typing system can make.

Cacharpo has a limited way of sensing what is happening during the ongoing performance. Co-performing with it has required the author of this paper to be attentive to its decisions in order to make cohesive music. This has resulted in performances in which following Cacharpo has been necessary rather than evenly co-performing with it. One way to address this will be to increase the communication between the human live coder and Cacharpo by incorporating additional meaningful feature extraction for the system to sense the direction of the performance (e.g. "mood", rhythmic and melodic density).

The development and conceptual framework of Cacharpo addresses a need to humanize software that can create on its own. The latter is an ongoing exploration in the emerging field of Musical Metacreation (Eigenfeldt and Pasquier 2012). Ways of expanding the personalities of the system described in this paper will be explored further. For example, an ANN could perform an analysis of the human performer's input code, which could be useful for Cacharpo to recognize particular live coding styles and patterns from different live coders. This would enable Cacharpo to choose between using multiple modes of live coding styles (i.e. JITLib style, pattern style) and be aware of other useful metadata.

The further development and evolution of this system will potentially enable it to accompany and assist the live coding musician, perhaps even for decades, like the members of an ensemble. This has happened already with systems such as George Lewis' Voyager, which has been active since 1986 (Lewis 2000). To do this, continuous training of ANNs will be required in order to keep the system learning.

Although Computer Music and Artificial Intelligence has been used to emulate well-known styles of famous music composers with great results (Cope, 1987), this project aimed to emulate the behavior of the non-famous — almost anonymous — human live coder. Moreover, it was not intended to emulate any particular live coding composer or individual musician's style. Informal showcasing of videos performing accompanied by Cacharpo revealed that people are neither surprised nor disappointed by seeing code and hearing music produced by an autonomous agent. Further testing under dance floor conditions is required.

5 Links

A video of Cacharpo co-performing with a human live coder is available at the following URL: https://vimeo.com/227332172.

References

Attali, Jacques. 1985. Noise: The Political Economy of Music, translated by B. Massumi. Minnesota: University of Minnesota Press.

Auslander, Philip. 2002. Liveness: Performance in a Mediatized Culture. Routledge.

Blanco Arboleda, Darío. 2012. "Los Bailes Sonideros: Identidad y Resistencia de Los Grupos Populares Mexicanos Ante Los Embates de La Modernidad." In Sonideros En Las Aceras, Véngase La Gozadera, edited by Tumbona Ediciones, 53–73. Mexico: El Proyecto Sonidero.

Brown, Andrew R., and Andrew C. Sorensen. 2007. "Aa-Cell in Practice: An Approach to Musical Live Coding." In Proceedings of the International Computer Music Conference, ICMC 2007, 292–99.

———. 2009. "Interacting with Generative Music Through Live Coding." Contemporary Music Review 28 (1), 17-29.

Collins, Nick. 2006. "Towards Autonomous Agents for Live Computer Music: Realtime Machine Listening and Interactive Music Systems" (Doctoral Dissertation). University of Cambridge, UK. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.2661&rep=rep1&type=pdf.

———. 2011. "SCMIR: A Supercollider Music Information Retrieval Library." In Proceedings of the International Computer Music Conference, ICMC 2011.

Cope, David. 1987. "Experiments in Musical Intelligence." In Proceedings of the International Computer Music Conference, ICMC 1987.

Drape, Rohan. 2015. "The Hsc3 Package." https://hackage.haskell.org/package/hsc3.

Eacott, John F. 2007. "Contents May Vary: The Play and Behaviour of Generative Music Artefacts" (Doctoral Thesis). University of Westminster, UK. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=968ADEB891732CA15B4EDF2629F5F3A6?doi=10.1.1.101.8967&rep=rep1&type=pdf.

Freeman, Jason, and Akito Van Troyer. 2011. “Collaborative Textual Improvisation in a Laptop Ensemble.” Computer Music Journal 35 (2). The MIT Press: 8–21.
Gill, Arthur. 2007. Introduction to the Theory of Finite-State Machines. New York: McGraw-Hill.
Hudak, Paul, John Hughes, Simon Peyton Jones, and Philip Wadler. 2007. “A History of Haskell: Being Lazy with Class.” In Proceedings of the Third ACM Sigplan Conference on History of Programming Languages, ACM 2007.
Hudak, Paul. 2015. “The Haskell School of Music”. Yale University.
Hunt, Andy, and Marcelo M. Wanderley. 2002. “Mapping Performer Parameters to Synthesis Engines.” Organised Sound 7(2). Cambridge University Press: 97–108.
Lee, Sang Won, and Georg Essl. 2014a. “Communication, Control, and State Sharing in Networked Collaborative Live Coding.” In Proceedings of the International Conference on New Interfaces for Musical Expression, NIME 2014, 263–68.
———. 2014b. “Models and Opportunities for Networked Live Coding.” Live Coding and Collaboration Symposium ’14, University of Birmingham, UK.
Lewis, George. 2000. “Too Many Notes: Computers, Complexity and Culture in Voyager.” Leonardo Music Journal 10. Cambridge University Press: 33–39.
Ludwig, Otto. 1983. “Writing Systems and Written Language.” In Studies and Monographs 24, edited by Florian Coulmas and Konrad Ehlich, 31–44. New York: Mouton Publishers.
Magnusson, Thor. 2010. “Designing Constraints: Composing and Performing with Digital Musical Systems.” Computer Music Journal 34 (4): 62–73.
McKinney, Chad. 2014. “Quick Live Coding Collaboration in the Web Browser.” In Proceedings of the International Conference on New Interfaces for Musical Expression, NIME 2014, 379–82.
McLean, Alex. 2014. “Making Programming Languages to Dance to: Live Coding with Tidal.” In Proceedings of the 2nd ACM Sigplan International Workshop on Functional Art, Music, Modeling & Design, ACM 2014, 63–70.
McLean, Alex, and Kate Sicchio. 2014. “Sound Choreography <> Body Code.” In Proceedings of the 2nd Conference on Computation, Communication, Aesthetics and X, xCoAx 2014, 355–62.
McLean, Alex, and Geraint Wiggins. 2010. “Tidal–pattern Language for the Live Coding of Music.” In Proceedings of the 7th Sound and Music Computing Conference 2010.
Mestanogullari, Alp, and Gatlin Johnson. 2009. “The Hnn Package.” https://hackage.haskell.org/package/hnn.
Murphy, Tom. 2016a. “The Midair Package.” http://hackage.haskell.org/package/midair.
———. 2016b. “The Vivid Package.” https://hackage.haskell.org/package/vivid.
Norman, Sally-Jane. 2015. “Live Coding and Embodied Action in Performance Contexts.” Speech, International Conference on Live Coding 2015, Leeds, UK, July 2015. https://www.youtube.com/watch?v=2cHMbGHjXIA.

Ogborn, David. 2014. “Live Coding in a Scalable, Participatory Laptop Orchestra.” Computer Music Journal 38 (1). The MIT Press: 17–30.
Ogborn, David, . 2015. “Estuary.” https://github.com/d0kt0r0/estuary.
Polansky, Larry. 1975. “Four Voice Cannons.” http://eamusic.dartmouth.edu/~larry/fvc/.
Quick, Donya, and Paul Hudak. 2013. “Grammar-Based Automated Music Composition in Haskell.” In Proceedings of the First ACM Sigplan Workshop on Functional Art, Music, Modeling & Design, ACM 2013.
Ramírez Cornejo, M. 2012. “Entre Luces, Cables Y Bocinas: El Movimiento Sonidero.” In Sonideros En Las Aceras, Véngase La Gozadera, edited by Tumbona Ediciones, 99–122. Mexico: El Proyecto Sonidero.
Roads, Curtis. 1985. “Research in Music and Artificial Intelligence.” ACM Computing Surveys CSUR 17 (2). ACM:163–90.
Stowell, Dan. 2010. Making Music Through Real-Time Voice Timbre Analysis: Machine Learning and Timbral Control (Doctoral Dissertation). School of Electronic Engineering; Computer Science, Queen Mary University of London. http://www.mcld.co.uk/thesis/.
Thompson, Simon. 1999. Haskell: The Craft of Functional Programming (2nd Ed). Addison Wesley.
TOPLAP. 2010. “ManifestoDraft.” https://toplap.org/wiki/ManifestoDraft.
Wamser, Christian. A., and Carl C. Wamser. 1996. “Lejaren a. Hiller, Jr.: A Memorial Tribute to a Chemist-Composer.” Journal of Chemical Education 73 (7). The MIT Press: 601–7.
Yee-King, Mathew J. 2011. “Automatic Sound Synthesizer Programming: Techniques and Applications” (Doctoral Dissertation). University of Sussex, UK.