AIMC 2023

Liveness and machine listening in musical live coding: A conceptual framework for designing agent-based systems

Georgios Diapoulis

URL: https://aimc2023.pubpub.org/pub/dk76kh2j

Abstract

Music-making with live coding is a challenging endeavour during a performance. Contrary to traditional music performances, a live coder can be uncertain about how the next code evaluation will sound. Interactive artificial intelligence (AI) offers numerous techniques for generating future outcomes. These can be implemented on both the level of the liveness of the code and also on the generated musical sounds. I first examine the structural characteristics of various live coding systems that use agent-based technologies and present a high-level diagrammatic representation. I sketch simple block diagrams that enable me to construct a conceptual framework for designing agent-based systems. My aim is to provide a practical framework to be used by practitioners. This study has two parts: i) a high-level diagrammatic representation informed by previous studies, where I analyze patterns of interaction in eight live coding systems, and ii) a conceptual framework for designing agent-based performance systems by combining both liveness and machine listening. I identify diverse patterns of interactivities between the written code and the generated music, and I draw attention to future perspectives. One code snippet for SuperCollider is provided and mapped to the conceptual framework. The vision of the study is to raise awareness on interactive AI systems within the community and potentially help newcomers navigating in the vast potential of live coding.

Introduction

Live coding is a performance practice where the performers share their screens with the audience and apply modifications to the running program[1]. It offers a rich technique for generative music[2] and a novel computing platform for exploring autonomy and interactivity with AI systems. Here, musical AI systems may range from rule-based systems to evolutionary computations, and from swallow learning to deep learning approaches. Liveness is here used as the “ability to modify a running program”[3] and is an inherent quality during a live coding performance[4]. The term liveness is not used similarly to its meaning in musical aesthetics[5] but rather denotes a degree when something is live[6]. I will mainly focus on a technical understanding of liveness as introduced by Tanimoto, which is mostly concerned about the live feedback to the programmer[7]. Machine listening is used during live performances for online sensing of musical percepts[8]. There is a long history of agent-based systems without support of real-time audio listening[9][10], but in the last decade, there has been immense progress in audio applications. Such developments have resulted in machine learning and machine listening ecosystems, like FluCoMa[11] and Sema[12]. Here, I will not focus on ecosystem designs but on performance systems developed by live coding practitioners. These are typically developed for a specific performance and are usually experimental designs.

Many practitioners can be unclear on implementing AI architectures into their designs. Whereas this may look unjustified due to decades of experience with interactive AI technologies[13], creative AI practices have a slow diffusion into interactive music systems. This slow diffusion is partly reflected in modern commercial software applications, which lean towards seamless interactions. The same issues apply to live coding systems to a certain extent. Indicatively, there are recent efforts to implement machine listening in Chuck[1], and popular languages like Sonic Pi[2] and TidalCycles[3] do not offer built-in implementation for machine listening and machine learning. Thus, my goals are to raise awareness within the community and offer an entry-level for those unclear on how to incorporate interactive AI in their performance systems.

Here, I begin with a high-level diagrammatic representation of live coding and present a conceptual framework of agent-based systems in live coding. Musical agents are here seen as both human and software agents[14]. The framework is informed by examining various examples of live coding systems developed by practitioners and reflecting on my experience as a live coder. It addresses some affordances and temporal constraints to consider when designing interactions between code and music. It is an observational study where I reflect on my practice when necessary and provide one code example in SuperCollider. The example aims to help navigate the proposed conceptual framework when designing agent-based systems.

I will start with the related work on liveness and machine listening in live coding. Then I continue and present the methodology and the observation material. I examine the basic structural components of agent-based live coding systems and discuss interactivity patterns on the observed material. The last section presents a conceptual framework for designing agent-based systems in live coding.

Theoretical background

Tanimoto's hierarchy of liveness was initially presented with four levels[7]. The levels are informative (L1), informative and significant (L2), informative, significant and responsive (L3), informative, significant, responsive and live (L4). Later on, Tanimoto introduced two more levels, called tactically predictive (L5), and strategically predictive (L6). Liveness is an inherent property of live coding systems, and a system used in the performance is L4 liveness[15]. A promising area of investigation has been started looking at predictive models, which are moving towards L5 liveness. Recently, a few live coding systems have exhibited characteristics of tactical predictions[16][17][18]. Essentially, a tactically predictive system can inform the users of their programming behaviours, and a trivial case is the auto-completion mode of modern text editors. A strategically predictive system can perform intelligent predictions and examine the liveness concerning agency. Some sort of predictive modelling is typically used when we aim to advance from an L4 system to an L5 system, albeit not necessary. Here, it is important to notice that no system today claims to fulfil the requirements for making tactical predictions. There is also a critical view in the literature[19] on whether advanced levels of liveness, such as code previewing, can be useful to the performer, as the cognitive resources during a performance are scarce, and more information can be no more than a distraction[20].

Besides the technical characteristics, liveness in programming environments depends on the notation of the language and the environment itself[4]. In live coding, notation usually consists of the functional parts of code. However, secondary notation, like comments, indentation and syntax highlighting, can also play an important role in the dramatization of a performance[21]. In musical live coding, liveness can have more qualities as the musical outcome and the humans involved are necessary parts. Thus the environment extends further than that of a typical programming environment and includes the notation (code), the musical outcome (music), and the musical agents involved (agents). Typically the performers, but some authors would argue for the importance of the audiences[22], and for a relational sense of ‘otherness’ that artificial agents can induce during a performance[23].

One functional characteristic of the human agent in a live coding session is that it can hear the musical outcome. As already mentioned, when the running program is rendered to musical outcome within a performance context, then the live coding system is necessarily within the L4 liveness[15]. The code generates musical outcomes sensed by the coder, who modifies the running program. Thus, there is a feedback loop between code and musical sounds as mediated by our auditory perception. The code is rendered to another modality (sound/musical outcome) and consequently to a continuous auditory stream (listeners' perception) that the coder can monitor.

Collins[24] presented a cookbook for machine listening in live coding. He presented two main categories of systems. For the first possibility, also known as “live coding control of machine listening”, there are two design decisions: (i) a feature-adaptive design which employs some feedback processes and (ii) an event-based design. For the feature-adaptive design, Collins implements a code with a pitch extraction algorithm that operates at 10ms. The output of the pitch extractor is within a feedback loop that controls the same sound generator. For the event-based design, an onset detector senses the environment with a microphone and triggers sound events. Regarding the second possibility, also known as “machine listening control of live coding”[24], Collins presented a timbral instruction set approach to let the machine listening component do the programming on this (imaginary) computer architecture. This system has a clock-based operation and follows a bottom-up approach to live coding as the levels of abstraction are progressively built on-the-fly. This system exemplifies how we may program a computer using another sensory modality. In this case, the machine listening component is a model of human hearing. The results of this analysis are applying progressive modifications to the instruction set, which can successively perform on-the-fly computations.

Introducing the observational material

Xambó[25] reviewed agent-based systems for musical live coding practices, providing the main material of my study. I focus on the systems that do afford both social interactivity and learnability. These are the two dimensions that Xambó introduced and denote whether the musical agents can cooperate (‘social interactivity’) and whether the system affords either on-the-fly or pre-trained learning (‘learnability’). I call these interactive AI systems, which can either learn from the environment or sense the environment. Thus, systems like Cibo[26] are excluded from my study.

List of video material

The methodology is formulated from complete observations, meaning that the observer did not interact with or influence the observed cases[27], and abductive inference[28]. I used as modes of investigation clues, metaphors, patterns and explanations as have been formed from personal practice as a live coder. I set certain criteria for selecting the observation material, a method known as criterion sampling[29]. The criteria I set for the observations are: i) there is an online video of the system, either performance, or a demo, along with a corresponding article, ii) the system is using the 'standard paradigm' for live coding, that is typing on a keyboard in a textual programming language, iii) the system affords learning or sensing, and iv) all examples used in Xambó's review[25] that fulfill the criteria (i), (ii) and (iii).

The selection of the use cases presented here is a combination of a search on Google scholar using the keywords "live coding" AND "machine listening", "live coding" AND "software agents", "live coding" AND "musical agents". The retrieved material was constrained to the first 100 entries for each query. Also, all entries from the International Conference on Live Coding (ICLC) repository on Zenodo[4] were retrieved, and regular expressions were used to search relevant articles. A total of 85 articles were retrieved from Zenodo. Some entries were excluded when the article was not written in English. Also, several systems were excluded because of mixed designs on the user interaction, such as modifying the system using both code and interactive graphical interfaces (e.g., interactive sonification, interfaces for education). The retrieved studies were last time updated on 2023-01-30.

The first selection criterion for a video demonstration implies a requirement that a practitioner has developed the system. The second criterion constrains the observation material to the arguably most common approach to live coding, that is, typing on a keyboard. The third criterion sets the requirement for interactive AI systems, and the fourth criterion is using bibliographical information from state-of-the-art live coding practices and agent-based systems. This resulted in a corpus of 8 videos, as shown in Table 1. I will briefly introduce all systems in the following bullet list, and in the next section, I will examine the systems in relation to one another.

1	Attanayake et al.[16]	https://vimeo.com/447733242
2	Autopia[30]	https://vimeo.com/349044280
3	Cacharpo[31]	https://vimeo.com/227332172
4	Flock[32]	https://vimeo.com/145109691
5	Mégra[33]	https://vimeo.com/321099751
6	MIRLCa[34]	https://youtu.be/ZRqNfgg1HU0

7	Ruler[35]	https://youtu.be/zjTL0DOCNBo
8	Wilson et al.[17]	https://youtu.be/2F1D8Harnkc

Table 1: List with the observational material.

1. Attanayake[16] and colleagues present a recommender system to suggest novel musical patterns to the user. The system uses Markov chains on the melodic patterns to suggest novel musical sequences. The system has three user modes: (i) continue writing code, (ii) execute the recommendation, or (iii) request a new pattern. An interview study was conducted where the participants indicated that the 'disruptive' mode was more enjoyable than the others. The system does not use machine listening.
2. Autopia[30] is a collaborative live coding system that uses audience voting and evolutionary computations. It uses genetic programming to generate prescriptive notation, in this case, code chunks that generate music which is neither ambiguous nor imprecise[36] and run the code independently. The system can perform on its own (without live coders) and also can be controlled by the coders.
3. Cacharpo[31] is a system that offers a co-performer for collaborative live coding sessions. The autonomous agent listens to the coder and extracts low-level acoustical features, which are progressively linked to higher-level features. Then, semantic descriptions are informed by machine listening, and a pre-trained neural network model maps acoustical characteristics to musical code patterns. The system has two modes; the coder either writes the code and the agent awaits, or the agent writes the code, and the user awaits.
4. Flock[32] is a collaborative live coding system that uses machine listening and a voting algorithm. The network dynamics of the system use evolutionary computations and incorporate a preference function informed by machine listening. Such preferences are visualized using descriptive notation[37], a representation of what is heard as the preference algorithm controls the final audio mix.
5. Mégra[33] is a system that uses probabilistic Markov models to generate new patterns on-the-fly. The system does not use machine listening. Instead, it is similar to Attayake's online training procedure, focusing on small datasets. Contrary to Attanayake, the system does not offer a code preview.
6. MIRCLa[34] is a music information retrieval (MIR) for querying audio samples using semantic tags from the cloud. The system is trained on a database of sounds and computes similarity measures and retrieves audio samples from Freesound ready to use in performance. It has three main functions: i) audio repurposing, ii) audio rewiring, and iii) audio remixing.
7. Ruler[38] is a rule-based system that explores generative spaces. The system was developed as an approach to on-the-fly programming a sound synthesizer. There is no machine listening implemented, the evaluation was conducted with human participants and similarity measures on the rule-based generator.
8. Wilson et al. [17] developed an agent that suggests code patterns in TidalCycles to the user. The model was trained on a large code database. Essentially, the system affords code previewing, similar one of the user modes used in Attanayake and colleagues.

Diagrammatic representation of the systems

From the above descriptions and the video observations, I identify four necessary parts of the systems: i) Human agent(s), ii) Software agent(s), iii) Code, and iv) Music. I abbreviate as Human (*H*), Software (*S*), Code (*C*), and Music (*M*) when necessary.

To facilitate understanding, I use block diagrams to represent the systems above using a rather simplistic representation of musical live coding (Figure 1). A typical live coding session can be schematically represented in the below diagram (*H*, *C*, *M*). The *H* writes text rendered to *C*, and *C* generates sound rendered to *M*. This is an incomplete representation, as the human also listens to the musical outcome during a performance (see transition labels on Figure 2).

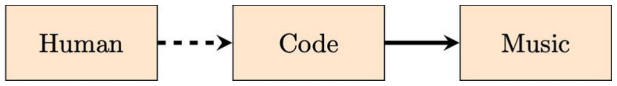

Figure 1: Simplistic schematic representation of a musical live coding session.

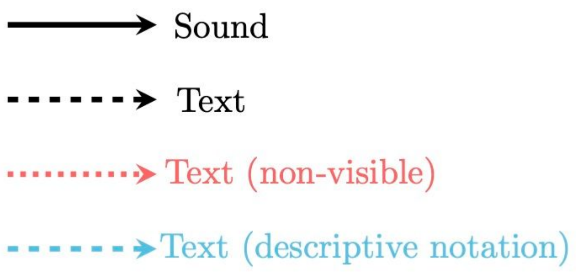

Figure 2: Descriptions of the transitions between the blocks.

Thus, a more accurate schematic representation of an agent-based system would include auditory feedback to the human (Figure 3). Here, the schematic representation does not show how the system informs the software agent. None of the systems in Table 1 can be represented in this manner, as they all have software agents aware of either the written code or the musical outcome.

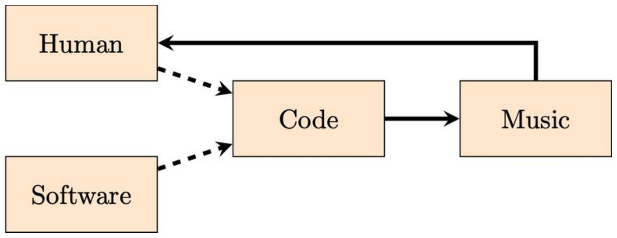

Figure 3: A simple schematic representation of an agent-based system for musical live coding, not corresponding to any of the observed cases.

More specifically, the systems by Attanayake et al.[16], Wilson et al.[17], and Autopia[30] are monitoring the written code and applying code modifications to prescriptive code chunks[37] (Figure 4). Some of the systems require the coder's permission to apply the changes, essentially a code preview feature that is not shown on the diagram below.

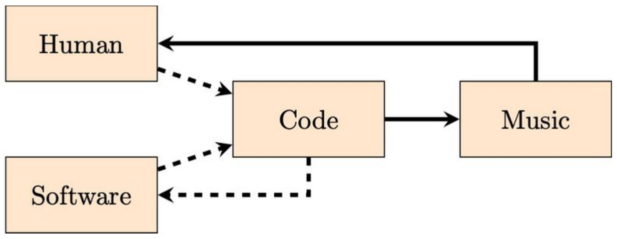

Figure 4: Agent-based systems informed by textual data.

Contrary to the systems above, Mégra and Ruler do not apply visible modifications to the code, as indicated by the red dotted arrow (Figure 5). Instead, the systems apply probabilistic and rule-based AI algorithms, respectively, to enrich the musical outcome as a seamless process. This functionality can be useful when fast musical variations are required, typically due to musical style requirements (e.g., dance music). Furthermore, Mégra has the potential for interactive visualizations, which can be seen as a descriptive notation. This feature is discussed by Reppel[33] but is not demonstrated in the accompanied video demo, and not visualized in the diagram below.

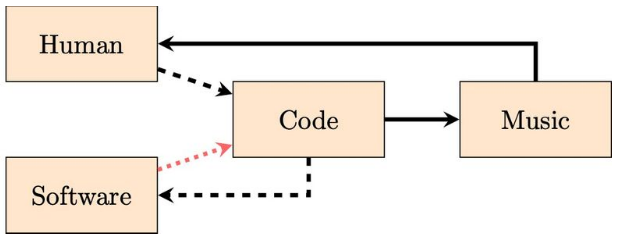

Figure 5: Agent-based systems informed by textual data without modifications on the prescriptive notation.

On the other hand, MIRCLa by Xambó[39], while agnostic about the content of the written code, the system is semantically aware of the musical outcome. The system has access to the acoustical features database on Freesound, and can retrieve similar context to the played sounds. This is denoted with the dashed arrow from *M* to *S*, indicating that the software agent is receiving textual data (Figure 6). Furthermore, while *S* is modifying *C* (red dotted arrow), this has no visible consequences for the user on the prescriptive notation. The user can only monitor the output of the interpreter to get informed about the applied code modifications. In practice, the system is much more complex, but I represent its main interactivity during a performance in the simple schematic.

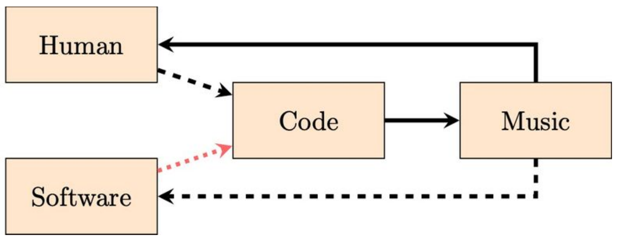

Figure 6: Agent-based systems informed by textual data related to the musical outcome, without modifications on the prescriptive notation.

Flock, by Knotts[32], is a system that applies a voting algorithm on collaborative live coding sessions. The analysis is performed on the musical outcome of the system, and the software agents modify descriptive notation, indicated with the blue dashed arrow, along with controlling the final audio mix (Figure 7). Again, the system is more complex as it is a multi-user performance.

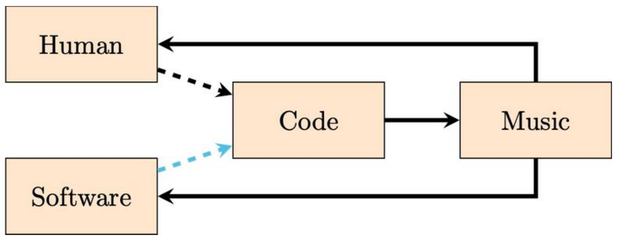

Figure 7: Agent-based systems informed by sound with modifications on the descriptive notation.

Cacharpo[31] is a system that simulates a co-performer and has a turn-taking design. The software agent awaits the user to provide permission to start typing novel code chunks. The system is performing an analysis of the acoustical features of the musical outcome and is not aware of the written code (Figure 8). Thus, Cacharpo generates prescriptive notation and listens to the musical outcome.

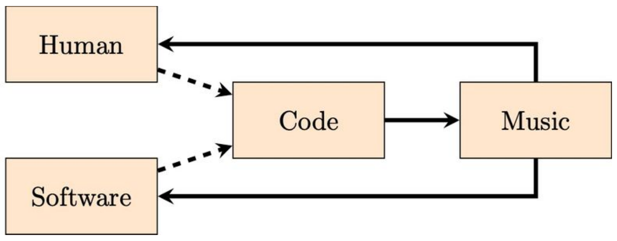

Figure 8: Agent-based systems informed by sound and applying modifications on the prescriptive code.

In all above cases the relations between the *H*, *C*, and *M* are always the same. The coder writes an encoded text which is decoded and rendered to music, and consequently, the coder listens to the sound to modify the code. From the observations, it becomes obvious that none of the systems is informed by both the music and the code

(Figure 9). More possibilities can be explored with this simple diagrammatic representation, and its complexity can increase when adding more components or transitions.

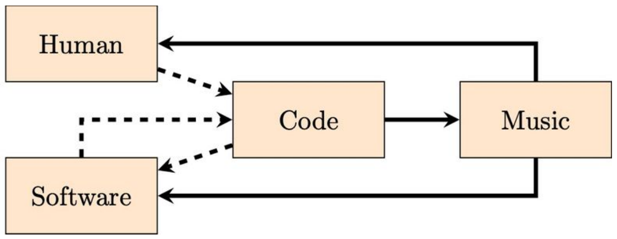

Figure 9: A future system informed by both the written code and the musical outcome.

Relations between the systems: Interactions of musical agents in live coding

Some systems incorporate software agents as assistants[31] or recommender systems[16][39]. There are several reasons for doing so, either because of intrinsic temporal constraints (e.g., typing speed) or because of extrinsic temporal constraints (e.g., browsing a large sample bank). For instance, Navarro and Ogborn[31] developed a software assistant (Cacharpo), which can be used for collaborative live coding performance between humans and software agents. The system uses machine listening and neural networks to generate novel code chunks in the autonomous agent workspace. This type of system can be seen as having intrinsic temporal constraints, as one performer cannot type the amount of code that two live coders can do. An example of extrinsic temporal constraints is MIRLCa by Xambó[39]. This system retrieves sound samples from the Freesound cloud database based on semantic queries (e.g., rain, noise). In this case, the live coder does not have the temporal capacity to search Freesound's webpage and select an audio sample that sounds like rain or noise. Consequently, MIRLCa system would do a better job given the constraints imposed on a live coder during a music performance, as it will calculate similarity measures between the semantic tag and the acoustical features of the sound sample.

On the other hand, Ruler, Mégra, and Autopia are agnostic of the acoustic characteristics of the musical outcome. These systems do not use machine listening. Instead, they apply AI algorithms in the domain of code. Whereas Autopia applies visible modifications on prescriptive code, Ruler and Mégra do not afford visible code modifications. The AI algorithms run on background processes, and, in the case of Ruler, the user is informed by the printouts from the interpreter. None of the three systems uses any descriptive notation, as Flock does, and none uses machine listening on the audio other than Flock and Cacharpo. To clarify, MIRLCa is applying machine listening-related technologies that use semantic information from the cloud. Thus, no real-time audio processing is performed, instead the system is informed from offline acoustical characteristics stored on the cloud.

All the abovementioned systems (Flock, Cacharpo, MIRLCa, Ruler, Mégra, Autopia) have L4 liveness. Below I continue with more cases which may be seen as advanced levels of liveness (L5 and L6). So far, two systems following the ‘standard paradigm’ have been developed towards making tactical predictions during a live coding session. These are the showcases by Wilson et al.[17] and Attanayake et al.[16]. The systems vary considerably in design decisions and available features. In Wilson et al.[17], the system is based on TidalCycles using a deep learning architecture pre-trained on a large corpus of code examples. In Attanayake et al., the system's predictive algorithm is a Markov model capable of online predictions of musical patterns. The two abovementioned cases make predictions on code segments, implementing a code preview functionality, with no involvement of machine listening. Their main difference is that Attanayake's et al. system affords online learning, whereas Wilson's et al. system affords offline learning.

It becomes obvious that prescriptive and visible notation is used by many systems (Autopia, Cacharpo, Attanayake et al., and Wilson et al.). The agents can adjust, or write from scratch, the prescriptive part of the notation, allowing the user to get involved with the generated code chunks. On the other hand, three out of eight systems (Mégra, MIRLCa and Ruler) do not use visible modifications on the prescriptive part of the notation or any descriptive notation. Only Flock provides the feature of modifications on the descriptive part of the notation. As for the modality of the music, it becomes evident that only Flock and Cacharpo use the acoustical characteristics for online sensing of the generated musical sounds, and MIRLCa applies offline sensing of acoustical characteristics using semantic information from the cloud. Similarly, some systems afford online training algorithms, and some afford pre-trained algorithms. I will further elaborate on these in the next section.

Conceptual framework for designing agent-based systems

As an extension of the previous section, where I examined the interactions of agent-based systems, I identified three different domains of interest when designing a system. The domains of i) ‘Code’, ii) ‘Music’ and iii) ‘Software agents’, will be referred to as the *coding domain*, the *musical domain*, and the *software agents*, respectively.

Below I introduce a conceptual framework (Figure 10) informed by technologies of liveness and machine listening. My focus is not on how to technically implement software agents in live coding but on what concepts can be useful when designing interactive AI systems. The framework is not meant to be exhaustive but rather a tool to aid in analysing and developing agent-based systems. Live coding is known to “resist or trouble any easy classification, categorization, or explanation”[40](p. 2). It is a non-linear framework that composes new knowledge from the literature and my musical practice. More specifically, Collins[24] recommendation on how machine listening and live coding may be combined is shown along with a part of a model for the timescales of auditory perception by Petri Toiviainen[41] on the musical domain. Some parts, like the temporal constraints and the affordances of the code, are formulated as a result of the literature study. The domain of software agents is constructed as part of the observation material. The thick semi-transparent line patterns indicate a ‘weak’ border between the three main domains of code, software agents, and music. Thus, the three domains span vertically (pattern coding), whereas the three categories of affordances, temporal constraints, and negative timescales span horizontally (colour coding). The outward arrows from the domain of *software agents* to the *musical domain* and the *coding domain* indicate that agents can act upon these modalities. As live coding also facilitates interactive visualizations, the framework constrains the creative aspects that can appear in a live coding session, as it does not consider the potential of visualization technologies. Finally, I provide a code snippet and map it on the framework to facilitate understanding of how to navigate the conceptual framework.

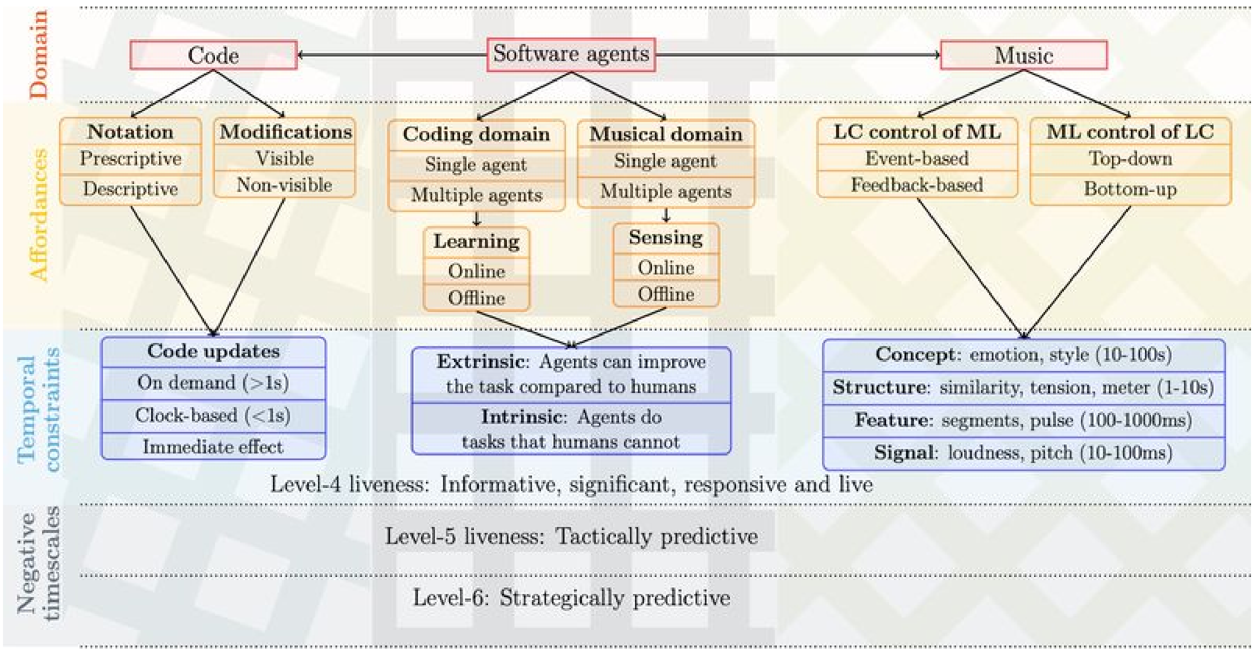

Figure 10: Conceptual framework for designing agent-based systems for musical live coding.

Domains of the conceptual framework

Coding domain

The leftmost column (Figure 10) shows the *coding domain*, along with some of the affordances of a system and the temporal constraints related to user interaction. The code can do an action (prescriptive notation) or can describe an action (descriptive notation). Secondary notation[22], such as code comments, is a descriptive notation. The visibility of the code is independent of whether the code is prescriptive or descriptive. Some processes can run in the background and be invisible to the user.

Moreover, the code updates demonstrate some inherent temporal constraints. The *coding domain* temporal constraints and affordances depend on actions that are either system actions, human agent actions, or software agent actions[42]. A question arises whether we consider the code part of the system, the coder’s reasoning and performative processes, or an autonomous entity. Thus, it is related to how we ascribe agency to the code, and this opens a wider discussion on aesthetic appreciation[43], which go beyond the scope of the article. For instance, Tidal has an inherent tempo clock which can be seen as a clock-based system action. Code updates can be performed by users or autonomous agents on demand or immediately (e.g., Attanayake et al.[16]).

Software agents domain

The middle column shows the domain of *software agents*. I identify that the agents can act upon the coding domain and the musical domain and can be either single agent systems or multiple agent systems. When they act on the musical domain they afford sensing, and when they act on the coding domain, they afford learning. Both sensing and learning can be either online or offline, as discussed above. I examined the relations between the different systems and I discussed how agents could exhibit either intrinsic or extrinsic temporal constraints.

Musical domain

The rightmost column show the *musical domain*. The affordances of the musical outcome are discussed as presented by Collins[24], and the temporal constraints present the temporal characteristics of low-level (signal), mid-level (feature) and high-level (structure, concept) acoustical features. There are indicative durations for each feature family, which indicate an inherent delay time for computations when applying machine listening.

Mapping use cases on the framework

Figure 11 shows examples of mapping different systems on the conceptual framework. At least one attribute from each block (e.g., Notation, Modifications, Learning, Sensing) is necessary unless a system is not operating on a specific domain. For instance, many systems do not incorporate machine listening technologies, like Attanayake et al.[16] which is agnostic of the musical domain.

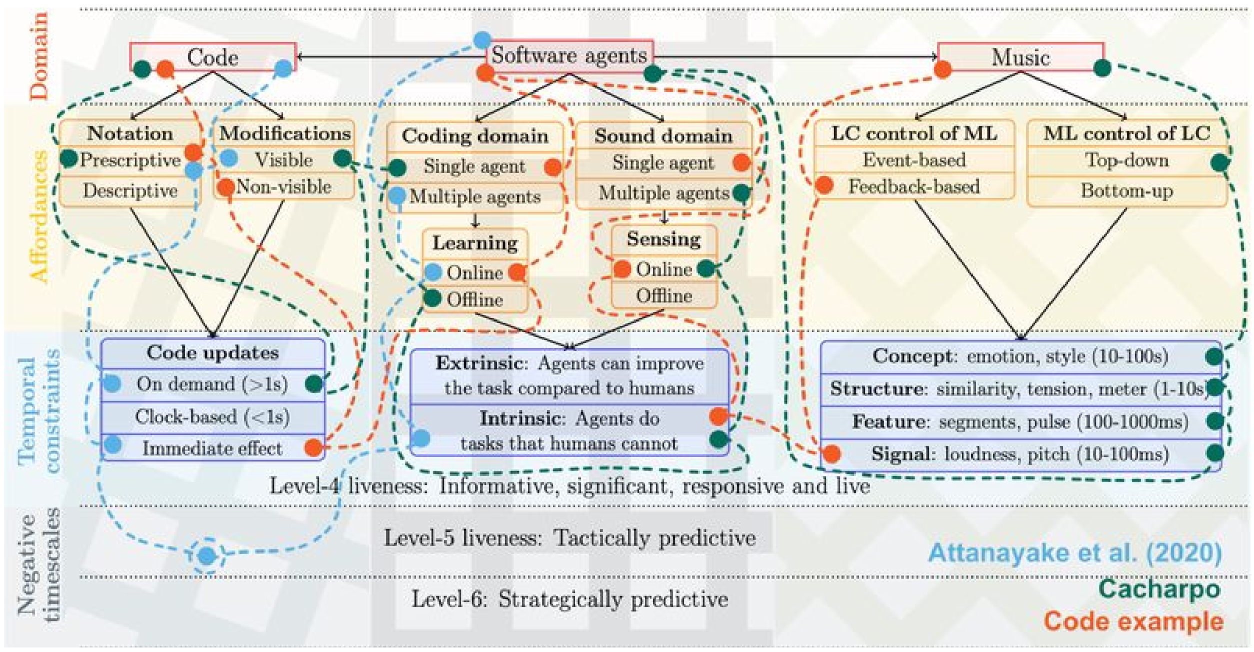

Figure 11: Mapping the code example and two of the systems on the conceptual framework.

The code example below demonstrates a simple agent-based system. If we follow the coding domain from top to bottom (orange dashed line on Figure 11), we can trace the path of interactivity starting from the musical domain (upper right on Figure 11). The code example applies feedback-based machine listening to extract a relatively low-level acoustical feature on-the-fly. The specific feature is the spectral centroid, a feature correlated to pitch and perceptual brightness. Thus, it affords online sensing using a single agent (musical domain). Following that, a conditional statement determines whether another single agent (coding domain) will generate sound or move into silence. This conditional statement (if statement in SynthDef ‘agent’) can be seen as a thermostat, which implements a negative-feedback design and can hardly be seen as a sufficient condition for learning by experience[44]. I will not attempt to discuss what can be seen as learning. Still, I would argue that this rather simple system exhibits some elementary properties that can be useful when designing systems that can learn to reach an equilibrium state. In complex systems, like a live coding performance, equilibria are not easy. Continuing in the coding domain, the code example, does not make any visible modifications on the notation and has immediate effect when reaching the threshold value.

```
// Code example in SuperCollider (SC3 v.3.13.0)

(
SynthDef(\ml, { arg audioBus = 0, controlBus = 2;
    var chain, feat;
    chain = FFT(LocalBuf(1024), InFeedback.ar(audioBus, 2));
    feat = SpecCentroid.kr(chain);
    Out.kr(controlBus,
           if(
                   RunningSum.kr(feat[0], 100) * 0.01 < 3000,
                   feat[0],
                   DC.kr(controlBus)
           ).poll(3)
    );
}).add;
arg freq = 330, ffreq = 2, pan = 0.0, amp = 1.0;
    var seq, pattern, trig, gate;
    pattern = [[1, 9/8, 2],     [3/2, 1, 0.5], [4/5, 3/2, 6/5]].mirror;
    gate =  Gate.kr(LFSaw.kr(ffreq).range(freq/2, freq),Impulse.kr(ffreq * 3));
    trig = Stepper.kr(Impulse.kr(ffreq), 0, 1, 9).fold(2, 7).fold(4, 5);
    seq = Demand.kr(Impulse.kr(trig), 0, Dseq(pattern * gate, inf));
    Out.ar(0, Pan2.ar(SinOsc.ar(seq).mean, pan, amp))
}).add;
SynthDef(\agent, {
    arg controlBus = 2, threshold = 3000;
    var seq = Stepper.kr(Impulse.kr(In.kr(controlBus)/100), 0, 1, 3);
    Out.ar(0,
        Pan2.ar(
            RLPF.ar(
                if(In.kr(controlBus) < threshold,
                    SinOsc.ar(In.kr(controlBus) * seq),
                        Silent.ar
                ), 3000, 0.5
            ), 0.0, 0.2
        ).tanh
    )
}).add;
)

// run line-by-line
~centroid = Synth(\ml);
~sine = Synth(\sine);
~agent = Synth(\agent);

~sine.set(\freq, 110);
~sine.set(\freq, 220);
~agent.set(\threshold, 100);  // deactivate
~agent.set(\threshold, 200);  // boundary
~sine.set(\freq, 1000);
~agent.set(\threshold, 1000);
~sine.set(\freq, 2000);
~agent.set(\threshold, 3000);
~sine.free;                   // free sine
~agent.set(\threshold, 100);  // autonomous
~agent.set(\threshold, 0);    // silence
~agent.free; ~centroid.free;
```

A similar logic can be applied to follow the traces of interactivities for Attanayake and Cacharpo, and in principle all use cases can be mapped to the framework. I would like to comment on the negative timescales, a term I adopted from Tanimoto's presentation[5] during the ICLC 2015. The dashed circle in Attanayake's system indicates that the system is not claimed to be an L5 system. I would claim that code previewing is L5 liveness, but it can be a complex issue whether code preview is L5 or not. Here, I support the authors' position (for both [16] and [17]) for not claiming to be L5 systems. As a last point on L5 systems, I would expect to see in the future liveness technologies that can compensate for machine listening inherent delays. Simply put, as the technical notion of liveness can 'see' into 'negative timescales' and machine listening has inherent delays due to digital signal processing constraints (e.g., sampling rate) and perceptual constraints, then I think that expectations are raised on these technologies[24].

Conclusions

In this study, I aimed to provide a practical framework for designing agent-based systems for live coding music performances. I reviewed studies focusing on the 'standard paradigm' to live coding, that is, typing on a keyboard, and I examined eight use cases with online video material. I presented a high-level diagrammatic representation of live coding systems and identified interaction patterns between code, music and musical agents. I identified from this analysis that there is little attention to machine listening. Many instead incorporate machine learning for text generation which renders visible and prescriptive code chunks, but in many other cases, the agents have seamless consequences for user interaction. Also, no system incorporates both machine listening and text processing algorithms by making informed decisions on both domains of the generated music and the written code. Following the eight use cases analysis, I constructed a conceptual framework that can be used when designing agent-based systems and provided a code snippet to facilitate understanding. The implications of this framework are that it can help practitioners to navigate into ecosystems for machine musicianship, such as FluCoMa and Sema, and can also provide practical insights when designing agent-based systems for live coding, maybe to be used in education. Although the study presents a simple code example and two use cases of agent-based implementations mapped to the present framework, further informed decisions can be made using interviews with the developers of the use cases in question.

Ethics Statement

I declare no conflict of interest because of funding or otherwise. The present study follows the free/open-source software ethos along with low consumption of computational resources. No human participants were recruited for this study and no sensitive data were collected. The main methodology is based on an observational study from online video material where I did not interact with the participants, and it is mainly a methodological study. The study has an educational orientation and aims to support the live coding research community. Principles of accessibility, environmental sustainability, inclusion and socio-economic fairness were considered.

Acknowledgements

I thank Sara Ljungblad for pointing me out to connect and draw the trajectories of the interconnected components of the framework.

Footnotes

1. https://chuck.stanford.edu/release/VERSIONS — Accessed 23-01-28 ↩
2. https://sonic-pi.net/ — Accessed 2023-03-14 ↩
3. https://tidalcycles.org/ — Accessed 2023-03-14 ↩
4. https://zenodo.org/oai2d?verb=ListRecords\&set=user-iclc\&metadataPrefix=oai_dc ↩
5. Tanimoto’s keynote presentation during the first International Conference on Live Coding (ICLC 2015), https://youtu.be/4cJANuMiq18 ↩

References

- Attanayake, U., Swift, B., Gardner, H., & Sorensen, A. (2020). Disruption and creativity in live coding. *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 1–5. ↩
- Bernardo, F., Kiefer, C., & Magnusson, T. (2019). An AudioWorklet-based signal engine for a live coding language ecosystem. *Web Audio Conference (WAC 2019)*, 77–82. ↩
- Blackwell, A. F., & Collins, N. (2005). The Programming Language as a Musical Instrument. *PPIG*, 11. ↩
- Blackwell, A. F., Cocker, E., Cox, G., McLean, A., & Magnusson, T. (2022). *Live coding: a user's manual*. MIT Press. ↩
- Blackwell, A. F., Cocker, E., Cox, G., McLean, A., & Magnusson, T. (2022). Live Coding's Liveness(es). In *Live coding: a user's manual* (pp. 159–204). MIT Press. ↩
- Brown, A. R. (2016). Performing with the other: the relationship of musician and machine in live coding. *International Journal of Performance Arts and Digital Media*, *12*(2), 179–186. ↩
- Church, L., Nash, C., & Blackwell, A. F. (2010). Liveness in Notation Use: From Music to Programming. *PPIG*, 2. ↩
- Collins, N. (2003). Generative music and laptop performance. *Contemporary Music Review*, *22*(4), 67–79. ↩
- Collins, N. (2011). Live coding of consequence. *Leonardo*, *44*(3), 207–211. ↩
- Collins, N. (2011). SCMIR: A SuperCollider music information retrieval library. *ICMC*. ↩
- Collins, N. (2015). Live Coding and Machine Listening. *Proceedings of the International Conference on Live Coding*, 4–11. ↩
- Collins, N., McLean, A., Rohrhuber, J., & Ward, A. (2003). Live coding in laptop performance. *Organised Sound*, *8*(3), 321–330. ↩
- Croft, J. (2007). Theses on liveness. *Organised Sound*, *12*(1), 59–66. ↩
- Dahlstedt, P., & McBurney, P. (2006). Musical agents: toward computer-aided music composition using autonomous software agents. *Leonardo*, *39*(5), 469–470. ↩
- Diapoulis, G., Zannos, I., Tatar, K., & Dahlstedt, P. (2022). Bottom-up live coding: Analysis of continuous interactions towards predicting programming behaviours. *NIME 2022*. ↩
- Gifford, T., Knotts, S., McCormack, J., Kalonaris, S., Yee-King, M., & d'Inverno, M. (2018). Computational systems for music improvisation. *Digital Creativity*, *29*(1), 19–36. ↩
- Green, O., Tremblay, P. A., & Roma, G. (2019). Interdisciplinary Research as Musical Experimentation: A case study in musicianly approaches to sound corpora. *Electroacoustic Music Studies Network Conference: Electroacoustic Music: Is It Still a Form of Experimental Music?* ↩
- Knotts, S., & others. (2018). *Social Systems for Improvisation in Live Computer Music* [Phdthesis]. Durham University. ↩
- Lankoski, P., & Björk, S. (2015). Formal analysis of gameplay. In *Game research methods* (pp. 23–35). ↩
- Lewis, G. E. (2000). Too many notes: Computers, complexity and culture in" voyager". *Leonardo Music Journal*, 33–39. ↩

- Lorway, N., Powley, E., & Wilson, A. (2021). *Autopia: An AI collaborator for live networked computer music performance*. ↩
- Magnusson, T. (2019). *Sonic writing: technologies of material, symbolic, and signal inscriptions*. Bloomsbury Publishing USA. ↩
- Magnusson, T. (2019). *Sonic writing: technologies of material, symbolic, and signal inscriptions*. Bloomsbury Publishing USA. ↩
- McKechnie, L. E. F. (2008). Observational research. In L. M. Given (Ed.), *The SAGE Encyclopedia of QUALITATIVE RESEARCH METHODS* (pp. 573–576). SAGE. ↩
- McLean, A. (2014). Stress and Cognitive Load. In A. Blackwell, A. McLean, J. Noble, & J. Rohrhuber (Eds.), *Collaboration and learning through live coding* (pp. 145–146). ↩
- Nash, C., & Blackwell, A. F. (2012). Liveness and Flow in Notation Use. *NIME*. ↩
- Navarro, L., & Ogborn, D. (2017). Cacharpo: Co-performing Cumbia Sonidera with Deep Abstractions. *Proceedings of the International Conference on Live Coding*. ↩
- Olsson, C. M. (2015). *Fundamentals for writing research: a game-oriented perspective*. ↩
- Palys, T. (2008). Purposive sampling. In L. M. Given (Ed.), *The SAGE Encyclopedia of QUALITATIVE RESEARCH METHODS* (pp. 697–698). SAGE. ↩
- Paz Ortiz, A. I. (2022). *On-the-fly synthesizer programming with rule learning* [Phdthesis]. Universitat Politècnica de Catalunya. ↩
- Paz Ortiz, A. I. (2022). *On-the-fly synthesizer programming with rule learning*. ↩
- Reppel, N. (2020). The Mégra System-Small Data Music Composition and Live Coding Performance. *Proceedings of the 2020 International Conference on Live Coding*, 95–104. ↩
- Roberts, C., & Wakefield, G. (2018). *Tensions and Techniques in Live Coding Performance*. ↩
- Rowe, R. (1993). Chapter 5: Machine Listening. In *Interactive music systems: machine listening and composing*. MIT press. ↩
- Rowe, R. (2004). *Machine musicianship*. MIT press. ↩
- Stewart, J., & Lawson, S. (2019). Cibo: An autonomous tidalCyles performer. *Proceedings of the Fourth International Conference on Live Coding*, 353. ↩
- Tanimoto, S. L. (1990). VIVA: A visual language for image processing. *Journal of Visual Languages & Computing*, *1*(2), 127–139. ↩
- Tanimoto, S. L. (2013). A perspective on the evolution of live programming. *2013 1st International Workshop on Live Programming (LIVE)*, 31–34. ↩
- Toiviainen, P. (2015). *Lecture notes in Music Perception I*. University of Jyväskylä. ↩
- Wilson, E., Lawson, S., McLean, A., Stewart, J., & others. (2021). *Autonomous Creation of Musical Pattern from Types and Models in Live Coding*. ↩
- Wisdom, J. O. (1951). The hypothesis of cybernetics. *The British Journal for the Philosophy of Science*, *2*(5), 1–24. ↩
- Xambó, A. (2021). Virtual Agents in Live Coding: A Short Review. *arXiv Preprint arXiv:2106.14835*. ↩

- Xambó, A., Lerch, A., & Freeman, J. (2018). Music information retrieval in live coding: a theoretical framework. *Computer Music Journal*, *42*(4), 9–25. ↩
- Xambó, A., Roma, G., Roig, S., & Solaz, E. (2021). Live Coding with the Cloud and a Virtual Agent. *NIME 2021*. ↩