Poly-temporality Towards an ecology of time-oriented live coding

Alejandro Franco Briones
McMaster University
francoba@mcmaster.ca

Diego Villaseñor
Independent Researcher
diego.vid.eco@gmail.com

David Ogborn
McMaster University
ogbornd@mcmaster.ca

Abstract

The current paper traces the development of three platforms for poly-temporal live-coding: Canon-Generator, FluentCan and TimeNot. The platforms rely on concepts and ideas developed by the Mexican-American composer Conlon Nancarrow and are based on an ongoing collaboration around sonic experimentation and time. The paper describes how a process of tensions and resistances have become a productive context for research and knowledge production.

Introduction

The current research has at its core the path opened by the composer Conlon Nancarrow and his life-work on time and music, more specifically poly-temporal composition strategies. This research is an on-going critical reflection that has pushed the authors to think differently about certain problems regarding time, sound, live coding and poly-temporality. Firstly, it was necessary to produce a conception of Nancarrow's work that differs from the main North-American / European narrative, even resist the tendency to assess Nancarrow as an expatriate American composer in order to reframe him as a Mexican-American socialist artist, thus conveying a constellation of implications that go beyond the conventional understanding of Nancarrow's work. Secondly, as access to the artifacts and scores regarding his oeuvre are restricted (particularly for people that are not supported by academic infrastructures), it was necessary to refer mainly to the ideas and notions that Nancarrow produced and that were interpreted and analysed by people like Gann, Murcot, Collins, Thomas, Sandoval, Estrada, among many others. We appropriated these loose sets of ideas and used them as a creative and imaginative starting point. And finally, the conversations kept by the authors of this text rely on the idea of resistance (Franco and Villaseñor, 2018). Resisting each others impulses to dominate the conversation and push forward what emerged from the tension between the arguments.

The output of this research took its first form as the platform

Nanc-In-A-Can/Canon-Generator, a series of SuperCollider classes and functions capable of producing poly-temporality based on the concepts, ideas and strategies of Conlon Nancarrow enmeshed with live coding's different conceptions of rhythm and time. From this point, two diverging parallel paths were enabled. Namely, FluentCan, a SuperCollider extension and notation that offers new possibilities to produce poly-temporal structures in a way that fits the purposes and idiosyncrasies of live coding, and the computational notation TimeNot. The present paper describes the latter in detail, both its notation and its multi-contextual approach which breaks from some core ideas of Nanc-In-A-Can in order to expand upon the expressive capabilities for time-oriented live coding.

The three aforementioned projects critique the infrastructure and time conceptions of live coding communities, seeking to widen the possibilities of their practices. The poly-temporal structures enabled by this notation form a space of resistance that might allow listeners and performers to experience time beyond the scope of accelerated and linear neoliberal subjectivity. The present project is an extensive attempt to produce a mode of performance that emerges from an experience and conception of time as slow, constant, in resistance, multiple, simultaneous, non-linear, digital/analogue, and rhythmic.

Nanc-In-A-Can/Canon-Generator is a SuperCollider library designed to produce temporal canons, like the ones proposed by Conlon Nancarrow (Franco and Villaseñor, 2018). The ideas of Nancarrow are explored in order to create new temporal conceptions within the field of live coding. Notationally Canon-Generator offers an API that has a compositional focus. It requires the performer or composer to define a canon with all the traditional parameters one would expect for it: a sequence for durations and melody, a convergence point, a list of tempos, and also other less conventional options. This means that, on the one hand, the musician has to define every one of these parameters from the very beginning. On the other hand, the musician, upon reading the code, can have a fairly clear idea of what is going to happen.

In contrast, the FluentCan extension for SuperCollider is an API wrapper for Canon-Generator that responds to the necessities of live coding performances. It offers a highly expressive syntax, various strategies for providing default values, and powerful tools for expressing novel musical ideas in the field of poly-temporality..

TimeNot, is a computational notation that is capable of producing, in an expressive way, complex rhythmic ideas embedded in poly-temporal structures. Many relevant aspects of the notation draw its main features, particularly poly-temporal strategies for music creation, from Canon-Generator. The notation encourages performers to project musical ideas further from the present than in the conventional interaction model of live coding.

TimeNot allows the production of tempo canons as in the platforms FluentCan and Canon-Generator but, in addition, it explores new and expressive ways of representing them by complementing the production of canons with strategies to describe other forms of temporal organisation such as global durations and a specific form of rhythm. The notation of TimeNot has been implemented in two different complementary ways with distinct advantages: It is embedded in Estuary (Ogborn et al. 2017) as a mini-language which allows it to engage in ensemble dynamics and be integrated in a rich ecosystem of languages that encourage diverse ways of thinking time and music. It is also an extension of the platform SuperCollider (Wilson, 2011) using its IDE and server taking full advantage of its sound-synthesis power and allowing it to be easily distributed.

Context

The poly-temporality that Nancarrow proposed can be regarded as highly algorithmic; in it, temporal and pitch mapping functions are the basic principle. This means that the melodic material of any given music work (which consists mainly of pitch and duration series) can by transposed into any tempo or any pitch register. Nancarrow developed the concept of a convergence point (CP) as a way of organizing this intricate musical material. The CP is a point in time in which the formal and the chronological temporalities of a musical idea are identical. According to Thomas (1999), this strategy allows us to listen different timelines moving towards the same point in chronological and formal time. Given that a poly-temporal canon can be generated very easily by algorithms, it is a musical strategy that can be implemented in computational settings with relative ease. Collins (2003, pp 1) describes a compositional system capable of producing various kinds of tempo canons, a precedent for this notation. More recently we designed the software Canon-Generator (Authors, 2018). Canon-Generator provides a programming notation that allows musicians, artists, programmers and other creative users of code to create multiple and simultaneous sound timelines based on the work of Conlon Nancarrow. This software was developed in SuperCollider (McCartney, 2002) because of its powerful computational and synthesis capabilities and its extended use among live coding practitioners in Mexico City. Listing 1 allows the user to create a major scale in three different tempos that converge at the fifth event (G, midinote 67).

```
Can.init;
(
// convergence canon;
~conv = Can.converge(
    melody: Can.melody(
    [8,8,8,8,8,8,8,8].reciprocal, // 1/8 rhythmic
        figures
    [60,62,64,65,67,69,71,72]),
    cp: 5,
    voices: Can.convoices([50,72.5,75],[-12,0,12])
);
~conv.visualize(s)
);
```

Listing 1: Canon-Generator

What we find more interesting are the "human-to-human" communication aspects of this notation, including but not limited to the cognitive dimensions of computational notation (Green, et al. 2001). This piece of code is not only role-expressive (in Green's terms) but also brings to the foreground a series of cultural references that locate the user of the program within a cultural and socio-political framework: the work of Conlon Nancarrow and its most salient temporal strategy to establish poly-temporal musical structures: Tempo Canon. The word 'can' also makes reference to the rigid steel or tinplate container normally used to preserve food which often is related with vulgar or industrialised products and, in this case, cheap nourishment for survival. This notation attempts to re-contextualise and re-appropriate musical strategies often related to a privileged music elite and academic activity, that claimed the ideas of Nancarrow as part of their tradition even though the Mexican-American socialist artist required an untraditional context to flourish (Franco and Villaseñor, 2018). The canon generator, by distributing it in communities away from the people that often control the canon of the music academy, allows new meanings for the work of Nancarrow.

The notation style of Canon-Generator is heavily embedded in SuperCollider's inherent notation and it has a limited scope. There are some aspects that do not facilitate live coding performance, for instance: the nested parentheses very particular to SuperCollider. SuperCollider language is not an expressive notation for rhythm as time, in this context, is mostly represented through inter-offset durations, namely wait patterns as understood in SuperCollider's Pattern Library (Harkins, 2009). This action produces a wait pattern, which is expressed as duration, obfuscating its meaning and compromising role-expressiveness. A more transparent use of the duration could refer to the total length of a musical idea (which cannot be expressed easily in the SuperCollider Pattern Library) or the length of a sound event (which is expressed as legato). Rhythm is more intuitively described in terms of offsets and onsets over an underlying grid, which can hardly be inferred by the duration value described in SuperCollider Pattern library. It also requires a binary distribution to be installed, which is an impediment in some settings. From this starting point a need for a new notational system that might be capable of representing the novel and compelling temporal forms already made possible by Canon-Generator was identified. At this point the endeavours of the two principal authors of this platform diverged: Diego Villaseñor implemented FluentCan as a wrapper and extension of Canon-Generator while Alejandro Franco developed the notation of TimeNot.

Diverging Paths

FluentCan

FluentCan responds to one of the original premises of Nanc-in-a-Can/Canon-Generator: to design a live coding interface and notation that can be self-contained in SuperCollider. This has allowed its integration to a broad ecosystem of software development and practices; one with strong communities worldwide and which has been particularly active in Mexico City (Nancarrow's creative home).

Conceptually FluentCan is a direct response to the particular necessities of live coding practices, where a flexible, composable and expressive notation is often desired. Flexibility means that the order of parameters in the definition of a canon, or the fact that some of them may be missing, should not be tremendously important: behind the scenes, the parameters should be put in the correct order and default values should be provided. Composability suggests that small ideas should be able to gradually grow and transform through the course of the performance. And expressivity means that it should be easy to generate abstractions, be they of canons or of transformations of canons (for example, allowing the musician to easily use one canon as a prototype for another).

FluentCan takes its name from the so called "fluent interface" technique of object oriented programming. Using this technique it is possible to wrap the data driven and immutable Canon-Generator API into a notation that can make use of method chaining to build musical ideas. This simple wrapping offers not just the ability to write canons in a more efficient way (by providing defaults), but also allows for inheritance (canonB may inherit it configuration from canonA) and offers a whole range of methods to transform inherited and non-inherited data, the most radical of which is .apply (described below).

From a conceptual and processual standpoint, it has allowed Canon-Generator to enter into a resistant dialogue with TimeNot: because TimeNot and FluentCan are, so to speak, livecoding native notations, they have allowed their authors to exchange ideas in a fruitful dialogue of shared interests.

Syntactic simplicity

Because the new interface provided by FluentCan does not require the user to provide all the parameters that Canon-Generator needs, it allows them to take a simpler approach for making music. This means the user can now use FluentCan for making non-canonic structures, that are effective both syntactically and musically.

```
// Isorhythmic single voice sequence, 7 notes within
    2 seconds. So called 'color' comes from notes, '
    talea' from durs

Can.init;
c = FluentCan(\can1).notes([60, 62, 67]).durs([2, 3])
    .period(2).len(7).play;
```

Listing 2: Code Example 2

This idea can then be easily extended into a temporal canon like so:

Listing 3: Code Example 3

```
// Converts into a 2 voice canon with lower voice a
    perfect fifth below.
c.transps([0, -7]).tempos([2, 3]).play;
```

In this sense, now it is idiomatic to express non-canonic ideas while at the same time providing the means for the music and musicians to project these ideas into a poly-temporal dimension.

3.1.2 The .apply case Throughout the TimeNot development process, a notation for expressing rhythm called xo notation was developed; a notation that is similar to ixilang (Magnusson, Thor, 2011) and Gibber (Roberts, Charlie & Kuchera-Morin, JoAnn, 2012) in which graphic notation takes an important role. At its core, it consists of a string of x's and o's (i.e. xooxooxo) which serves to determine whether an event should sound (x) or not (o). The concept of the notation itself is conducive to different interpretations and variations. This has been the case not only between TimeNot and FluentCan, but also within FluentCan itself. Triggered by the effervescent dialogue with the TimeNot philosophy, for FluentCan it was chosen to allow any implementation of xo notation to be possible and easy to use. This was the main driving force behind the development of the .apply method.

The .apply method is a generalization of the necessity to allow for user generated functions to be used at runtime. The function itself does nothing more than applying the built up canonic data structure (inside the FluentCan instance) to a function that returns a new FluentCan instance (in Haskell style type notation it could be easily described as FluentCan -¿ FluentCan). This method can be called multiple times so that the effects of these calls are composed (see example below).

The class IsoFluent is one that provides static methods (as pure functions) which fulfill precisely this interface. Because Canon-Generator can take functions for transposition[1], the method IsoFluent.xo creates a transposition function that iterates over it's melody, for each voice of the canon (a MidiNote array). Using modulo arithmetic it returns a new melody with rests in place of a o's and the input melody notes in place of the x's, or when given a number (that corresponds to the voice index) it returns the input note if the index voice corresponds to the given number or a rest if not.

Through .apply FluentCan effectively extends Canon-Generator in a possibly infinite number of ways. It gives the performer the ability to create and modify their own functions (at compile time or on the fly). Even more so, it opens Canon-Generator and the exploration of time to the communal imagination. Now libraries that follow this interface can be independently developed, each of which may explore different ideas that nevertheless compose with any other ideas coming from the community.

```
Can.init; Can.defaultServerConfig;s.boot;

// Models
m = FluentCan().period(1.5).len(5); // model
n = FluentCan().period(2).len(4); // model 2
o = FluentCan().transps([0, -7]).tempos([1, 1/2]).
    period(2); // use this to create canon, tempos
    1/2 and 2 work well

//custom function for .apply
(
~aksakish = {|...nums| // input any sequence of
    numbers
         var aksakXos = nums.collect({|n| // make a
             list of x's and o's
         n.collect({|i| if(i == 0, {"x"}, {"o"})});
     }).flatten;
         // return a function that will receive the
```

[1]For transposing Canon-Generator takes an array of values on the key transp. Each value corresponds to the transposition of the voice that has the same index as the value. transp takes either an array of numbers ([Number]) or an array of functions ([[Number] -> [Number]]) or a combination of numbers and functions. When provided a Number, it simply maps the melodic sequence array over a function that adds the given number; if the value is a function [[Number] -> [Number]], it maps the melodic sequence over it.

```
            FluentCan instance
        {|fluentCan|
                // do whatever with the instance
                fluentCan
                .len(aksakXos.size)

                                .apply(IsoFluent.xo(
                                    aksakXos));
        }
}
);

// it is possible to switch from the different models
     (m, n, o)
a = o.def(\1).notes([65, 62]).apply(IsoFluent.xo("
    xo01o")).cp(2).len(7).play;
(
b = o.def(\2)
.notes([70, 77])
// ‘applyable’ functions can be composed:
.apply(~aksakish.(7, 5, 3))
.apply(IsoFluent.xo("10")) // voice 0 and 1 alternate
.cp(3)
.play;
)

(
c = o.def(\3)
.cp(4)
.period(6.5)
.notes([65, 62, 67, 75])
.len(15)
.apply(IsoFluent.xo("o1ox0xo"))
.play;
)

a.pause;
b.pause;
c.pause;
```

Listing 4: Code Example 4

TimeNot

TimeNot was created in a specific context: at McMaster University, and as a major research project (MRP) by Alejandro Franco in the MA program in Communication and New Media. Moreover, its development has been influenced by the activity in the Networked Imagination Laboratory (NIL), a space in which the concept of network is explored broadly and often in relationship with live coding practices. One notion that networked imagination conveys is the understanding of artistic creation and research as an ecology of — among many other things — natural and computational languages, artistic styles, techniques, technologies, strategies, etc. In other words, in the NIL a heterogeneous set of media is articulated, hacked, developed and explored in order to produce aesthetic and intellectual works. The premise that creation depends on an ecological approach to knowledge resonates with Mexico City's live coding scene (where Alejandro Franco developed many of his artistic practices), where collective creation and assemblages of heterogeneous artistic practices are often prominent. The main idea targeted by this notation is the tempo canon. However, what is particular about this exploration in the context of the poly-temporal ecology here explored is the many additional temporal and rhythmic strategies and techniques that can be notated easily.

The author opted to create a new, independent, software project based on the ideas of Canon-Generator so the notation could be adapted to many contexts. TimeNot is now available in two forms: within the Estuary platform, and as a SuperCollider extension. Estuary (Ogborn et al., 2017) is an experimental software that can be defined as a platform for learning and creating, and using live coding as a main strategy that favours a multilingual approach, and which makes live coding languages available on a zero-installation basis, like

Gibber (Roberts & Kuchera-Morin, 2012), LiveCodeLab, and Hydra. SuperCollider (McCartney, 2002) has a powerful audio synthesis engine as well as a well established community of users around the world. Both platforms respond in different ways to the author's cultural and social context; these are the best ways to give access both to a broad, general userbase as well as to key developers and more specialised audiences. There are other positive aspects of this decision; the separation of TimeNot and Canon-Generator give room for Diego Villaseñor to develop the notational style embedded in SuperCollider that would become FluentCan.

```
|. 4s .| xxxxxxxx
ra: 4:5:6 tr: 0|12|24 cp: last
synths: saw sqr tri pitch: 60 62 64 65 67 69 71 72
```

Listing 5: Code Example 5

Listing 5 reproduces the musical scale of the previous Canon-Generator example and presents a program that exemplifies all the main possibilities that TimeNot allows: a sequence of global durations and a rhythm arrangement in line one, a canonic configuration in line two, and a configuration of instruments and pitch in line three. The interplay among these four components produces rich poly-temporal sonic textures. When this example is executed it produces a series of musical events extending from the moment of evaluation into the future (a C major scale repeated in three octaves with the temporal proportions of 4:5:6 over a total duration of 4 seconds).

Strata

The architecture of the TimeNot notation is organised into four main strata or sub-notations. Three of these allow the user to organise sound events in time. The last stratum aids the organisation of sound parameters.

The three time-oriented sub-notations reflect three main temporal strategies that imply a heterogeneous understanding of temporal relationships. The first one allows the user to embed the sound output in a specific overall duration; this allows the user a very intuitive degree of control over the music material. The second sub-notation provides an expressive and comprehensive way to create rhythmic structures; this aspect of the notation was identified as the one that diverges the most from the possibilities endemic to SuperCollider's notational style. The third sub-notation is the stratum in which rhythmic ideas are transformed into a canon. This is the part of the notation in which the tempo canon ideas developed by Nancarrow are put into practice. The last sub-notation provides a simple syntax to invoke instruments and organise its parameters into different kinds of patterns.

Duration Notation

Having control over the global duration of the canonic/rhythmic structure helps to achieve an overall view of the result. This stratum represents time as a succession of events that do not favour detailed categorisation or differentiation among its internal components. Time here corresponds to the concept of durèe (Bergson, 2002) allowing users to produce simple sequences of events producing an ever-going sense of becoming.

The duration of the rhythmic/canonic structure is determined by a number followed by an s that represents seconds. However, the number could represent beats in a given tempo by adding a t or indicate a certain amount of cycles (cps) by adding a c. These multiple ways of expressing duration respond to the multi-contextual nature of this notation.

To determine a duration the number should be embedded in a special list that uses symbols and separators. The lists that uses the |: :| as a delimiter and the | as an internal separator generate an infinitely looping musical idea. An unlooped event is delimited by |. .|. A finite number of repetitions can be expressed with the symbol % followed by whole number that determines the number of repetitions. If the duration of the structure is omitted, the default is a 2 second event.

Rhythmic Notation

The rhythmic aspects of the notation can be used independently from the canonic ones. Nevertheless a minimal rhythmic idea has to be written for the notation to produce sound. An example of a comprehensive use of this rhythmic notation might be the idea in the following code example, in which an opening idea is presented in 5/8 manually introducing rhythmic onsets and offsets, then it is concatenated to an Euclidean pattern representing a tresillo (ie. the Euclidean rhythm that distributes 3 attacks over 8 slots), over a two attack onset pattern that is repeated two times. Finally, another 5/8 idea is manually expressed to close the musical idea.

```
|. 8s .|
xoxxo !3:e:8 p: xx #2 xooox  samples: hibongo
```

Listing 6: Code Example 6

Onset Patterns

Notating minimal rhythmic ideas as onset patterns has two main advantages; it provides a declarative materiality to the notation, and it frees the user from being limited to algorithmic structures (such as Euclidean rhythms), allowing arbitrary organisations that do not respond to explicit logical or computational patterns.

```
xxoooooo xxoooooo samples: hibongo
xooxooxo ooxoxooo samples: cabasa
```

Listing 7: Code Example 7

Patterns are notated with the characters x and o, where x denotes an attack and o denotes a rest. In the following example, the first pattern produces two attacks followed by 6 rests and the second pattern produces the Cuban clave rhythm. The following expressions should be evaluated one at the time, to evaluate both add a ; at the end of the first expression.

Repeat Patterns

Repetition allows the user to produce meta-metric cycles in which the same idea is presented until a variation marks the end of a period. This can be achieved because the onset patterns and the repeat patterns can be composed together to express a single musical idea. With the symbols ! and # we can indicate how to repeat a pattern. It is possible to create nested ideas within this language, such that a repeat pattern can contain an onset, repeat or Euclidean pattern. After the ! the onset, repeat or euclidean pattern to be repeated should be written and after the # a value that represents the number of repetitions. The first line in Listing 8 can be simplified as the example shown in the last section.

```
!xx!o#6#2  samples: hibongo
```

Listing 8: Code Example 8

3.2.1.2.3 Euclidean Patterns TimeNot's Euclidean Pattern subnotation is a very expressive and complete tool for rhythm which can integrate embedded onset, repetition and other euclidean patterns. The non-optional values to be given are n and k values as specified by Godfried Toussaint (2003). The Euclidean algorithm produces patterns of distribution of integer numbers as evenly as possible. Given n time intervals and k impulses, this algorithm provides a simple way to distribute the k impulses over the n time intervals. These patterns are found in many forms of music, particularly "in sub-Saharan African music, and world music in general" (Toussaint, 2003, pp. 1). The syntax proposed in this notation is k:e:n, a number representing impulses and another representing intervals. The first line of the following code example generates a Cuban tresillo and the second generates a clave that is the combination of a tresillo concatenated to a 4/4 measure with onsets only in the second and third beats.

```
3:e:8 samples: hibongo
3:e:8 oo xo xo oo samples: hibongo
5:e:8:r:4 p: xx  samples: bd
```

Listing 9: Code Example 9

The k and n values are chained by the operator :e:;There are two other ways to manipulate the structure produced by this sub-notation: :r: which creates a rotation value, wherein the pattern produced will maintain its same number of impulses and same intervals but will appear starting from a different point of the structure. The operator p: produces a structure that can be a repeat, an onset or another euclidean. This pattern becomes the k value and the non-k values of the time intervals, a series of offsets occupying the same length as the pattern are produced, if k is xx the non-k will be oo, as in the example above

Canonic Notation

A fundamental difference between the tempo canons produced by the earlier Canon-Generator project and those produced by TimeNot concerns the minimal components of a tempo canon. Conlon Nancarrow was restrained by the media at hand: the player piano. The player piano can not play two or more notes that are the same, in the same register, at the same time. To perceive distinct tempos it was necessary to differentiate the voices of the canon somehow. This was achieved by pitch transposition. In TimeNot, where an alternative mechanism to differentiate voices is provided, pitch transposition is not as fundamental. In TimeNot pitch transposition is not a mandatory parameter for canonic transformation. Differentiation of voices is acquired by providing a straightforward mechanism to select instruments. Different voices can be identified by different timbral qualities instead of, or in addition to, pitch and register. Canons can be thought of as based on samples. In TimeNot, pitch transposition is optional in the case of pitched instruments and, in the case of unpitched (sample-based) instruments, it is incompatible. In this way, less typing is needed to produce a "canonic transformation" and the idea of a tempo canon becomes even more focused on temporal aspects of the musical ideas.

The model that is implemented in TimeNot so far is what Nancarrow would have called a "convergence canon", that is to say a canon with only one CP and without tempo change per voice.

In TimeNot, the number of voices per canon is decided by the number of values given to the argument ratio. Ratio is a list of proportions which can have a corresponding list of transpositions that 'canonise' the rhythmic structure.

```
|. 35s .| xoox!5:e:8#5xxoxooox x
ra: 13:17:20 tr: 0|5|8 cp: last
synths: saw sqr tri
eu pitch: 60 67 65 68 72 75.5 67 55 56 59 60
```

Listing 10: Code Example 10

Sound Notation

Sound notation allows the performer to parametrise the sound aspects of the program. At the moment, there are 5 non-temporal aspects that can be arranged: instruments, pitch, amplitude, panning and out bus. If a list of instruments is provided, it will be distributed as one instrument per voice of the canon. The rest of the sound parameters are organised as sequences that will be replicated exactly in each voice of the canon. This can be arranged in two different ways: as an isorhythmic configuration, in which each value is assigned from left to right until all rhythmic events have been exhausted, or as a Euclidean distribution that distributes evenly k sound values over n rhythmic events.

Conclusions

With regards to FluentCan's .apply method, it is important to remark that it's development nevertheless is the direct result of the contrapuntal dialogue between the authors of both libraries. The original necessity for this method was to use it to implement a type of xo notation (as proposed by TimeNot). However its generality was able to exceed the requirements of this notation (as explained in 3.1.2). The result was a powerful opening up of both FluentCan and Canon-Generator to any number of possible extensions and new ideas coming from anyone interested from the SuperCollider and Live Coding communities. These results (conceptual, social, and technical) even if tangential in their surface relationship with TimeNot are strongly linked to it by the network of ideas underlying the conceptual space that has become Nanc-in-can. Conversely, TimeNot derived its duration sub-notation from the notion of period that is a product of the FluentCan development. In an interesting parallel with the properties of temporal canons, namely, their tendency to continuously diverge and converge (from which unforeseeable textures emerge); the process so far followed by the authors has spawned a critical and creative flux of independent-related ideas that can further find convergence points in the softwares so far developed or in new ones to come.

TimeNot pushes to its limits the notion of permeable autonomy. This means that it is part of multiple processes that consume it but not in its totality. Neither the possibilities enabled by Estuary, the algorithms inherited from Canon-Generator nor the dialogue that it maintains with FluentCan are sufficient to explain what TimeNot might actually be. Although TimeNot stands as an autonomous software, it is only possible for it to exist as the outcome of the multiple explicit and implicit conversations that happen around it. In this sense, its boundaries are unclear at the same time that its identity remains unambiguous. As this context can be identified as relevant to the development of this software a similar pattern can also be identified in the way its sub-notations are organised. Moreover, it also stands as a valid reflection on the way the multiple concepts of time interplay: Poly-temporality, rhythm and duration might be three ways of explaining time that presents three different perspectives of the same phenomenon. The ecology that surrounds the development of TimeNot is also the form that the notation takes and it is, as well, the way in which it allows artists to think about time: as an ecology.

The authors of the libraries here described, FluentCan and TimeNot, have paid particular attention to the way that researchers relate with knowledge. The concept of co-creation used by Donna Haraway (2015) seems to be relevant here. Co-creation here implies that knowledge is the product of conversations and exchange of ideas, but in a way that resistance and tension is not inhibited but foregrounded.

Forthcoming (Divergent) Convergences

Our exploration of the multiple temporal dimensions in music has begun to bring forth fundamental questions:

1. How is time and poly-temporality represented in other domains?
   a. What particular dimensions of time are exposed by each of these domains?

2. How can the calculations involving time become as flexible, free and liberating, as their results?

   - In purely technical means, how can poly-temporal expressions evolve beyond the current strategy of precalculating each event in advance, into a completely live interaction that would allow us to interact with time and its effects in full real time?

The first set of questions beg for platforms other than SuperCollider and the TimeNot language. These platforms need to be powerful enough to deal with disparate mediums such as audio, graphics, web, poetry, etc; and should also be capable enough to deal with the larger scale architecture that dealing with multiple media requires. At their core, such questions requires a focus on pure time and an architecture that can route time events into different media effectuations.

The second question demands a disassembling of our current notions for constructing time oriented structures. The diverse elements that conform these structures must become independent of each other. Each one must be able to change without needing to stop or restart the temporal flow of events.

This new perspective and aspiration takes its cue from the experience in divergent development. What has been experienced so far is a fertile inter-fluence of independent elaborations over a shared conceptual environment, a truly evolving ecosystem made up of experiences and ideas about time. The first degree of divergence exposed here, two programs diverging and exchanging ideas, is to be interiorized into multiple divergences within a single convergent ecosystem-platform: now the mediums and the parameters of pure time must be allowed to diverge as well. The future platform should be simultaneously divergent, which means: no single point of focus, no single dominant idea or way of doing things, etc; and convergent, that is to say, composable ideas feeding into each other, multiple mediums expressing events organized by singular and multiple polytemporal instances and free routes for interpretations that expose the nature of time.

References

A.F. Blackwell, C. Britton, A. Cox, T.R.G. Green, C. Gurr,G. Kadoda, M.S. Kutar, M. Loomes, C.L. Nehaniv, M. Petre, C. Roast, C. Roe, A. Wong, R.M. Young (2001). Cognitive Dimensions of Notations: Design Tools for Cognitive Technology . Proceedings of the 4th International Conference on Cognitive Technology. Coventry, UK.

Bergson, Henri (2002). Concerning the nature of Time in Henri Bergson: Key Writings. Edited by
Ansell Pearson and John Mullarkey, London: Continuum.

Collins, Nick (2003). Microtonal Tempo Canons After Nancarrow/Jaffe. Proceedings of the International Computer Music Conference, Singapore.

Franco, B and Villaseñor, D. (2018). Nanc-in-a-Can Canon Generator. SuperCollider library for generating and visualising temporal canons critically and algorithmically. Proceedings of the International Conference of Live Coding, Madrid.

Haraway, Donna; Kenny, Martha (2015). Anthropocene, Capitalocene, Chthulhu. Art in the Anthropocene. Davis, Heather and Turpin Etienne. Encounters Among Aesthetics, Politics, Environments and Epistemologies. London, UK: Open Humanities Press

Harkins, James (2009). A Practical Guide to Patterns.
http://distractionandnonsense.com/sc/A_Practical_Guide_to_Patterns.pdf
Last accessed: 12th August, 2019.

Magnusson, Thor (2011). The IXI Lang: A SuperCollider Parasite for Live Coding.

McCartney, James (2002). "Rethinking the Computer Music Language: SuperCollider." Computer Music Journal 26 (4). MIT Press: 61–68.

Ogborn, David; Beverley, Jamie; Navarro Del Ángel. Luis; Tsabary, Eldad; McLean, Alex. Betancur, Esteban (2017). Estuary: Browser-based Collaborative Projectional Live Coding of Musical Patterns. International Conference on Live Coding (ICLC) 2017.

Roberts, Charlie & Kuchera-Morin, JoAnn (2012). Gibber: Live coding audio in the browser. 64-69.

Thomas, Margaret (1999). Nancarrow´s Temporal Dissonance. Intégral 13.

Toussaint, Godfried (2003). The Euclidean Algorithm Generates Traditional Musical Rhythms. School of Computer Science, McGill University Montréal, Quebec, Canada.

Wilson Scott; Cottle, David and Collins, Nick (2011). The Supercollider Book. The MIT Press.