The Context Modular Sequencer

Liam Goodacre
F1 802 Success Towers, Panchavati
Pashan, Pune 411008, India
liam.goodacre@gmail.com

Abstract

This paper presents Context, a powerful sequencer built in PD, and explores some new prospects for music composition made possible by the software. Through a series of case studies, it is shown how Context can represent musical structure and sequence many different sorts of patterns. By looking at Context's use of random number generation and internal communication, the paper explains how the composer can use Context to create generative music. The paper proposes the Context network—a collection of interconnected Contexts—as a medium of musical composition, and highlights the various possibilities and design choices that emerge during the development of a Context network.

Keywords

Context, sequencer, DAW, generative, composition

1 Introduction

Context is a modular sequencer built in PD. It aims to give the user more creative control over musical composition and to be versatile enough to fit into any PD patch, from beginner projects to sophisticated performances. Context is built with the familiar timeline paradigm at its core, but presents the timeline as a local object rather than a global environment. As Context is modular, it does not demand to be used in any particular way, inviting the user instead to create her own sequencing environment, according to her personal taste and style of composition.

I designed Context because I wanted a way of writing complex pieces of music in PD, as an alternative to Digital Audio Workstation sequencing. But building and testing the software has taken me in many directions that a conventional DAW could not, and has led me to question the very nature of musical composition. This paper presents a justification for the Context sequencer, along with some examples and musings on the kinds of music that it could create. It is not intended as a user manual: the software's functions are not listed in full and are described only insofar as they are necessary for the argument[1]. By the end of the paper, I hope that readers will appreciate what makes the Context sequencer special and have some ideas of their own about how they might use it.

[1] Full documentation is forthcoming at www.contextsequencer.wordpress.com

2 What is a sequencer?

A sequencer is a program or hardware unit that schedules events in time, for the purposes of making music. The events that it schedules are discrete in nature, consisting of simple messages (ones and zeroes, musical notes, etc), played back at the right time to create patterns and loops. Sequencers are arguably the greatest differentiating factor between electronic and acoustic music, because they relieve the musician from the obligation of playing each individual note. Computer music is *programmed* by the user/musician, not *played*. As such, sequencers are there to do what computers do best: to take care of routine operations, so that the user has time to focus on other, higher level decisions.

Sequencers are not the only attempt that has been made to program music. Musical scores are similar to sequencers in that they store music as information, to be replayed at a later date. As with sequencers, musical scores save the musician from the responsibility of memorizing or improvising. When she becomes literate, the musician has her repertoire expanded beyond the bounds of her memory, and the length and complexity of written music can be made correspondingly greater. In this way, sequencers can be seen not just as an invention of the digital age, but as the continuation of the classical attempt to formalize music. The difference is of course that a sequencer achieves a more complete degree of automation, containing both the score and the means to play it.

Sequencers occupy a small corner of electronic music theory, but the paradigm of scheduling events goes down to its very core. Waveforms are stored on computers as discrete data; their playback involves recalling the numbers at the right time and in the right order. Likewise, the timeline of a Digital Audio Workstation (DAW), populated with various recorded tracks and modulation envelopes, is also a way of organizing musical events in space so that they will occur at the right time. Oscillators, sequencers and DAW environments are all inherently concerned with determining what happens when, differing from each other only in scale, oscillators being the micro-, and DAW timelines the macro- approach to representing music as information.

2.1 Software sequencing environments

Most DAWs—indeed most pieces of software—provide an environment in which the user must function in a predefined way. Context does not take this approach. Instead, Context presents a small, simple building block which can be replicated and interconnected in many different ways. When more than one instances of Context interact with each other, Context networks are formed. It is these networks which define the musical composition.

Other software has adopted the network paradigm for representing composition. Nodal[2], one of the most successful generative sequencers, presents a circuit-diagram like grid where "nodes" represent musical events and random decisions. Koan and Noatikl[3] provide similarly unstructured environments where the user makes music by connecting various modular units. Many other programs have developed their own novel ways of representing composition spatially[4].

While such products provided some early inspiration, Context differs in that it is more completely modular. Context is not a suite; it is a tool. It is a PD abstraction which can be placed on any PD canvas, anywhere, and made to interact with other objects and existing patches. So rather than defining an environment to which the user must conform, Context lets users create their own sequencing networks within the PD environment. As such, a Context network stands closer to the paradigm of modular synthesizers—it even lets users "hack" an instance of Context beyond its original design[5]. But instead of shaping voltage or signal, Context determines events in time.

Context does resemble other sequencers built in PD, especially Xeq[6]. The main differences here seem to be the Graphical User Interface (GUI) and the fact that Context is a patch, not an external. While this comes at a cost to the CPU, it has the advantage that it is more versatile and accessible to the user.

3 Description of the Context sequencer

When Context is started (double clicked), the cursor moves across the canvas and plays the given pattern.

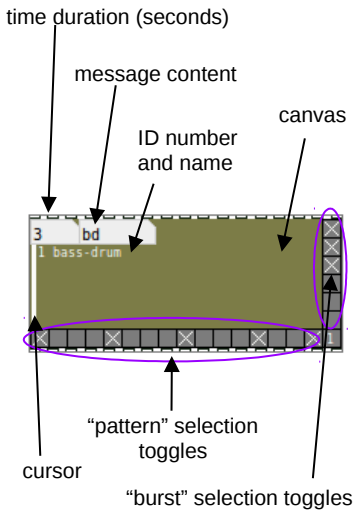

Figure 1: Anatomy of Context

- **Time duration** determines the length of time that Context takes to complete one sequence (ie. how long the cursor takes to move across the canvas).
- **"Pattern" toggles** determine the pattern or rhythm that is played.
- **Message** determines the message(s) that are sent by the pattern.
- **"Burst" toggles** determine what happens after the pattern is finished.
- **Inlets** are all identical. When an inlet receives a message, it will start the Context cycle.
- **Outlets** are synched with the "pattern" toggles, so an outlet will send a message when the cursor moves over a selected toggle.

The distinction between the "pattern" and the "burst" is central to Context's operation. The pattern interprets its selection temporally (a *then* b *then* c), resulting in a fixed rhythm or melody played out through time. The burst interprets its selection logically (a *and/or* b *and/or* c), making a decision about what happens once the Context cycle is complete. As the pattern and the burst are always present, each Context contains a temporal and a logical component, allowing the user to define a musical phrase and consider its consequences at the same time.

Both axes are click-draggable, so the pattern and burst arrays can come in any length, and Context can occupy any amount of space on the canvas. Unfortunately, PD only allows for outlets on the bottom of an object and not on the side. To access outlets from the "burst" toggles, the user can flip the x- and y- axis; the pattern will then

[2] CEMA, Monash Universtiy, 2007-2013: http://www.csse.monash.edu.au/~cema/nodal

[3] Intermorphic, 2007-2016: https://intermorphic.com/noatikl

[4] Notably Blokdust: Twyman, Philips & Silverton 2016: www.blokdust.com.

[5] See "overlay hacking", Context documentation (forthcoming)

[6] Czaja, "Time and Structure in Xeq", 2004: http://puredata.info/community/conventions/convention04/lectures/tk_czaja

play from top to bottom, and the outlets will fire simultaneously at the end of the Context cycle.[7]

4 Events in Context

It has been said that Context sequences events in time, but specifically what events? Context does not discriminate between different types of events it schedules. It sends PD messages[8] which can be interpreted in any way. This point sounds trivial, but is important for what follows. Some of the events that Context could schedule are:

1. **A musical note** (ie. "C#) or rhythmic beat in another PD patch
2. **A note or beat in another program**, through MIDI or OSC
3. **A modulation event** (ie. "filter to 1000 Hz")
4. **An arbitrary instruction** (ie. "launch the rocket")
5. **An instruction to start another Context, or restart itself**
6. **An instruction to change another Context in any way** (ie. "open toggle #1" ; "change the delay time to 2 seconds")

Numbers 5 and 6 are of special significance for the next sections of this paper.

4 Sequencing musical structure

I define musical structure as the pattern or alteration of musical or rhythmic phrases through time. So for instance a simple rhythmic structure might be:

```
      A A A B
where A = 1 x 1 x
and   B = 1 1 1 x
```

Figure 2: In this notation, 1 = one quarter note and x = one quarter note rest.

Structure is extremely important in music, but curiously under resourced in most DAW environments[9]. A DAW timeline gives a unique timecode to every event in a composition, down to the minutest details, but this omniscience comes at a cost: what if the composer wishes to leave some event undefined or implied? Structure cannot be depicted in such an environment: if the user decides to make a structural change to the composition, she must carry out every alteration by hand, analogous to calculating 2+2+2+2+2+2+2+2 instead of 2*8. This problem is eased somewhat by "storage banks" programmed into most step sequencers—the user can schedule the events A and B to signal the switching from one pattern to another. But these storage banks have their limitations. For instance, what if the composer wishes to design a second order structure, such as:

```
        C C C D
where   C = E F F F
        D = F E E F
        E = 1 x 1 x
and     F = 1 1 1 x
```

Figure 3

To capture this structure, you would need a storage bank of storage banks. Some sequencers offer such a thing[10], but then the question arises, what if you want a third order structure? and so on. Clearly the limit of a DAW will soon be reached.

Since Context does not discriminate between the types of events that it schedules, it can make light work of complex, multi-layered structures such as these. Musical structure can be determined by the step sequencer with the same ease as a rhythmic pattern. The following diagram makes this clear:

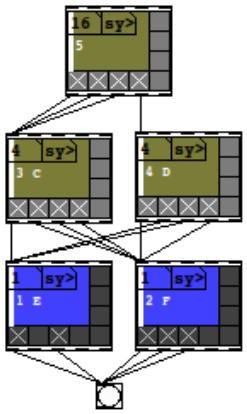

Figure 4: A context network representing the structure described in Figure 3. To see the structure, follow the connections: the first Context fires three times to C and once to D. C and D are

[7] This comes across more readily on screen than on paper, but recognizing the difference between a normal and a flipped Context will help the reader interpret the diagrams in this paper. A flipped Context has its cursor on the top, as in figure 6, rather than on the left, as in figure 1. The timeline is then read top to bottom, rather than left to right.

[8] Lists, symbols or floats

[9] For example, the Recording and Sequencing suite in Propellerheads Reason: https://www.propellerheads.se/reason/recording

[10] Ie. Reason's Matrix Pattern Sequencer: http://www.soundonsound.com/techniques/reasons-matrix-pattern-sequencer

similarly connected, to E and F, whose toggles represent quarter notes and quarter note rests.

Here we can see that the structure of the Context network reflects the structure of the music itself. The network is arranged in a hierarchy, with each level dictating what happens below. But hierarchy is not the only way of conceiving of musical structure. Consider the following structure:

```
          A B
where     A = 1 x 1 x
and       B = C or D or E
          C = 1 1 1 x
          D = 1 1 X 1
and       E = 1 X X X
```

Figure 5

What is required here is a way of depicting a set of parallel events. This is what Context's "burst" toggles are used for. The burst toggles all fire simultaneously at the end of the Context cycle, but specifically which toggles fire can vary. The following Context network satisfies the structure described in figure 5, with the "burst" toggles from A being used to select from a parallel series C,D,E.[11]

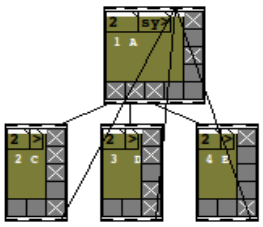

Figure 6: A Context network representing a parallel structure. Here, the "burst" toggles are on the x-axis and the "pattern" toggles are on the y-axis, so the timelines progress from top to bottom rather than left to right. The three open toggles in the "burst" section of Context 1 function as a selector, choosing first one then the next toggle once its cycle is complete.

With the techniques laid out in these two examples, Context can be used to create multi-layered musical structures with any degree of complexity. Hierarchies and parallel sets can be built layer upon layer and interconnected with each other in any way. Because Context can schedule structural events just as easily as musical events, the sequencer is capable of representing musical structure *per se*, where normal timeline environments leave it to be inferred.

[11] How the burst selects which toggles fire will be covered in the next section.

What is the point of representing music in structural terms? One practical reason is power and control. A tool that allows us to represent musical structure will save time and effort in the same way that a repeat sign or coda saves paper. This becomes most apparent when the composer decides to change something: a single alteration at a higher level can determine hundreds of other events, which would otherwise have had to be redefined individually[12]. Saving time is of course what computers do best, and so structural representation allows the user to focus on creative, rather than operational, decisions.

5 Randomness

Thinking about music in structural terms is enlightening, but it can also become overly deterministic. The brilliance of music often comes across more in spontaneity than in structure, and this is a feature which is difficult, if not impossible, to program. Lacking the judgement of the musician, a computer must rely on pseudo random number generation (PNG) to simulate spontaneity. It is very easy to program a computer to play random notes at random time intervals, but the result is no more musical than a Geiger counter. To generate non-deterministic music, it is necessary to find some way of containing the output of PRG's to specific decisions, times and ranges, and for the user to be able to override or modify a random decision made by the computer if it is not to her liking.

Context is designed with generative music in mind, and it is capable of using PRG's in many different ways. Because the composition is represented in the form of a Context network, random decisions are localized. So, for instance, the user can determine that the third note of a particular melody should be random, with the same ease that she would determine that it should be C#. Bearing in mind that Context can represent structural as well as musical events, it follows that the user can incorporate randomness at any level of the composition.

There are three different pseudo-random

[12] At its mathematical limit, structural representation has an exponential effect: in a structural hierarchy such as the one described in figure 4, an alteration at the nth level can affect x^n other events (assuming that all Contexts have a pattern size of x).

generators built into Context, each with its own attributes and characters:

1. Random delay time
2. Random messages
3. Random burst

Random delay time affects the time cycle for an individual Context, and hence the speed with which its step sequencer plays. It is defined by a range, so that the allowed intervals will be no bigger than eg. 10 seconds, and by a resolution, so that the intervals will come in denominations of eg. half or whole seconds. The resolution can also be defined exponentially, so that the allowed intervals are 2^n up until the range limit, where n is the resolution. This has the advantage of filtering out non musical time intervals, so that a Context cycle could last a whole, half, quarter or eighth note, but nothing in between.

Random messages allow one or more random terms to be added to the main send message, useful for determining musical notes and other parameters. Context messages can be "solved" with more or less the same ability as a scientific calculator, so for instance a message "n (2+4)/3" will simplify to "n 2", and "m (?10 + 100)" would simplify to "m 109" if the random term ? 10 yields its maximum 9. All simple arithmetic can be integrated into random note generation, expanding the user's options for harnessing chaotic number streams.

Random burst determines which of the burst's toggles are selected on the completion of a Context sequence. This is useful for decision making and structural randomness. The burst uses Gaussian distribution to select a toggle, so that the user can determine the likelihood of a particular toggle firing. The random selection is then summed with a deterministic progression, so that 1 (or any number) is added to the selected toggle number successively. The same set of parameters is available for determining how many toggles are selected, as well as which ones. The result is a diverse palate of options for random decision making, ranging from linear sequences (A then B then C), to high & low probability sets (probably A but maybe B), to completely random scenarios (one, some or all).

In summary, the prospects for generative music are very rich in Context. Random timings can be made in congruence with standard musical intervals, random notes can be manipulated with arithmetic, and random decisions can be taken according to a probability distribution. To recall the previous section, it is easy to see how randomness can be incorporated into musical structure. Events at any level of structure can be decided by the burst, and a Context network can randomly choose between an A and a B section just as easily as it could randomly choose to play the note C#.

But the lines between rhythm and structure are not always clear. Figure 7 depicts a simple pattern in Context 6, but is it a rhythm or a structure? Instead of being played directly, the beats are passed on to the blue Context which acts as a logic gate, choosing between three further options: an eighth note, two sixteenth notes, and a rest (notice the third open toggle at the end). The pattern depicted by the first Context then is a sort of a "meta-rhythm", an outline of a rhythm which is then interpreted and played by other Contexts in the network. This is the sort of non-deterministic music that becomes possible with Context's application of pseudo random generators.

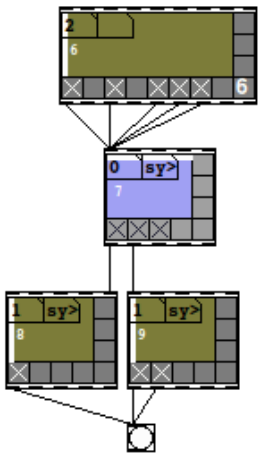

Figure 7: a structure involving random elements.

6 Internal Messaging

All Context parameters, including time, toggle selection, and dimension, can be set manually through the GUI, or automatically through Context's internal messaging system. For example, to open toggle number 3 on the x axis, the user can either click on it, or send a message to that Context " :x 3 ". Since Context does not discriminate between the types of events it schedules, it can also send such a message, giving Context the ability to alter its own settings. For instance, the message " :x 3 " could be sent from one Context to another, or even to itself, instructing it to open its third outlet. Opening and closing outlet toggles is the most obvious application, but every Context parameter can be accessed in the same way. I call this internal messaging.

The consequence of internal messaging is that a Context network can evolve over time. This is more potent than random decision making—consider the difference between a junction that offers you a set of alternative paths, versus a junction that offers you the tools to pave new road. The possibilities for self-evolving compositions include:

- A melody or rhythm which changes at a set point, ie. C# becomes E.
- The modification of a structural event, ie. “go to section A” becomes “go to section B”.
- The alteration of the Context cycle, ie. the duration doubles.
- Conditional changes, such as “the first time that a C# is played, this outlet will open” or “every 5th time that this note plays, there is a 20% chance that it will be randomly changed to another note”

The way in which a network evolves is also open to definition. Internal messages can be sent by a dedicated Context timer which sends one ie. every 30 seconds, or they can be woven into the Context network together with notes and other events. Remembering that Context messages can also incorporate random numbers, we also have the possibility for Context networks to evolve in random ways. Randomness can be directed towards a specific parameter (ie. “a random toggle on this Context will open”) or towards the Context ID numbers (ie. “a random Context will have its third toggle opened”).

A Context network that evolves at random has some practical limitations: the law of Entropy dictates that over time the system will tend from an ordered to a disordered state. Intervention is necessary if the composer wants to achieve something purposeful and avoid musical degeneration. Since internal messages are interpreted the same as user defined events, the process of vetting can be carried out in the GUI in the same way as normal Context design. There is also a separate “undo” abstraction which keeps a log of all changes made to the network and reverses them on demand. So the user still has control over the Context network, even if it is moving in a chaotic way.

7 Recording

Context can record melodies and rhythms as well as playing them back. The record function saves notes or data to the message box, and a pattern to the pattern toggles, according to the timings in which they are received. Thus, what Context records is information, not sound.[13]

The recorded information can come from the object’s inlets, or from a send-receive channel. This allows one Context to record from others in the network, as defined by their ID tags. The duration of the recording is predefined by the Context cycle time. Because the Context has only a finite, usually small, number of outlet toggles, a kind of resolution is imposed, whereby the Context can record no more events than there are outlets.

[13] A future release of Context will feature sound recording as well.

There are two immediate implications for recording. First, a melody or rhythm can be played into a Context from an external source, ie. a midi keyboard. This is a useful way of interfacing with Context and speeding up composition. The second is that Context can sample itself. A pattern or melody that is generated from elsewhere in the network, perhaps randomly, can be recorded and folded back into the composition. The record command can either be executed by the user, or automatically by the computer through internal messages.

This has further implications for the way that Context networks evolve. Through internal messaging, they are likely to evolve slowly and uniformly, but recording has the potential of making the process more discrete. One Context might house a melody or rhythm which functions as a base for a larger section of the network, as in Figure 7. The melody evolves somewhat but never deviates far from the base, until the command is sent to re-record it. Then a new base is formed, and the process repeats.

7 Using the Context Canvas

The Context canvas is the coloured area that lies between the message boxes and the pattern toggles. It is not wasted space: the canvas functions as an embeddable timeline for linear playback. “Content”, a modified PD array object, allows the user to place arrays on the timeline, turning Context into a sample player. The Context sampler has most of the features that one would expect from a basic sampler: the user can loop, slice, speed up and reverse the sample through simple GUI gestures (or internal messages). The sampler can also relate to the sequence messaging system: with a tilde (~) character, a Context message will take a snapshot of the sample, making Content ideal for custom modulations.

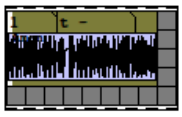

Figure 8: an embedded Content sample

It is also possible to embed one Context within the canvas of another; while the parent cursor hovers over it, the embedded Context is “on” and cycles continuously. Any number of objects can be embedded within one Context, and embedded objects can easily and accurately be moved around the canvas with a special drag-and-drop tool.

Thus, Context does not force the user to abandon the global-linear paradigm that has become so ubiquitous in music software, where each event occupies a point on a unified timeline. But rather than being conditions of the environment, globalism and linearity are choices that the user makes in designing a Context network. The user might reject altogether the localized and random approaches suggested in this paper and instead use one Context canvas as an environment to structure a whole composition, embedding and arranging samples and other PD instruments on the timeline. Such design choices are not restrictive: the user can easily create many such environments and have them interact as part of a larger network. In short, Context is capable of functioning as a universe as well as an atom. As an atom it can build more complicated structures, and as a universe it can host other objects for linear playback.

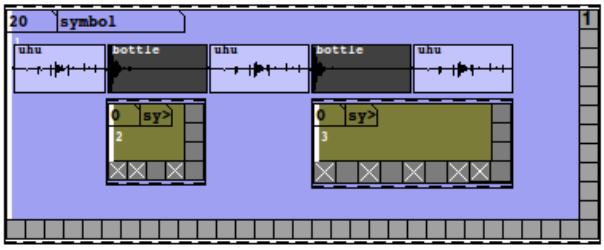

Figure 9: One Context acting as host to other embedded objects

8 Conclusion

What kind of music can be made on Context? As its designer, my main motivation is to see somebody else using Context to create music in a way that I hadn't even imagined. Context offers the user the opportunity of designing and shaping her own sequencing environment, rather than inheriting one in pre-packaged form. As such, Context can be whatever you want it to be, from simple to complex, deterministic to random, linear to non-linear. The limitations of the software remain to be found, as they surely will be, but for now the designing and exploring of Context networks promises to open many new horizons for generative and performance musicians.

A Context network can become a hive of activity, with events of all different sorts triggering each other and cascading through feedback loops like a giant, chaotic marble run. The resulting sounds might not resemble any conventional composition, and indeed it has been suggested that "composition" is not the right word for it at all. A Context network is like a musical score in that it has been designed by an artist to represent a piece of music, but this might guarantee nothing more than a starting point. The way in which the music progresses from there might be thought of as a sort of dialogue between performer and computer, with both parties free to make changes to the network. The computer takes some of these decisions at random, to which the user acts as a supervisor, accepting, canceling or modifying the computer's actions as she decides.

Context invites other forms of collaboration as well. The Context network is inherently de-centralized, with no part possessing absolute control over another. Because of this, it would be possible to design a Context network to be operated by two or more performers. This could be done in an organized way: one user having control over percussion while the other has control over melody. Or it could be done in a disorganized way, where both participants have access to arbitrary, undefined parts of the network, the resulting music being a surprise to them both. As Context networks can run themselves with a high degree of automation, the performer is also free to respond in other ways, for instance by playing a "real" instrument in conjunction with the Context composition.

To finish with a metaphor, I would hope that a Context network has something in common with a garden. The seeds originally sewn are not the same as the plants which grow to envelope their surroundings, and the gardener must constantly cultivate, prune and train her work, which is bursting with its own energy and will to return to nature. There is no final product, only the continual confrontation between an artist's intentions and forces that are beyond her control.

9 Acknowledgements

My thanks goes to the PDCon16~ organisation team, and to many people in the online PD community who have helped me with this project. I am also deeply indebted to Tristan Chambers for introducing me to the PD platform and inspiring me to make my own music software.

Context Version 2 is forthcoming at www.contextsequencer.wordpress.com, together with full documentation.

References

CEMA, Monash Universtiy, 2007-2013, Nodal:
http://www.csse.monash.edu.au/~cema/nodal

Intermorphic, 2007-2016, Noatikl:
https://intermorphic.com/noatikl

Twyman, Philips & Silverton, 2016, Blokdust:
www.blokdust.com

Czaja, “Time and Structure in Xeq”, 2004:
http://puredata.info/community/conventions/convention04/lectures/tk_czaja

Propellerheads, Reason, 1994-2016:
https://www.propellerheads.se/reason

Price, Sound on Sound, “Reason’s Matrix Pattern Sequencer”, 2007:
http://www.soundonsound.com/techniques/reasons-matrix-pattern-sequencer

Goodacre, 2016, Context:
www.contextsequencer.wordpress.com.