

# META-SEQUENCING: CONTROLLING SEQUENCE VOICES AND POLYPHONY USING THE POLYMAP, SIEVE, VALVE AND MASKXOR OBJECTS IN PURE DATA

Dr Edward Kelly  
London College of  
Communication  
University of the Arts London  
London SE1 6SB  
*morph\_2016@yahoo.co.uk*

## ABSTRACT

I describe a system of polyphony for Pure Data (Puckette, 1996) that is applicable to situations where multiple sequencers are routed to multiple voices, so that the degree of polyphony may be controlled for each sequencer, and polyphonic voices are assigned automatically. Three new objects are crucial for this purpose: **maskxor**, a bitmask object, **sieve**, an array-based router, **polymap**, a polyphony-controlled matrix map and **valve**, a source identifier object. These objects are configured to dynamically adjust the allocation of sequencer events and playback voices so that for example in the *Speechcutter-poly* application developed in association with the Centre for Creative Research in Sound Art Practise<sup>1</sup> at the London College of Communication, eight sequencers independent of one other are routed to 32 shared voices, and relationships and linkages between the sequencers may be altered on-the-fly.

## 1. INTRODUCTION

Traditional approaches to sequencing have a top-down system of polyphony control, where the polyphony is managed as a layer of functions between the event data and the playback engine, or by the playback engine itself as a separate module (figure 1). A clock references an arbitrary sequence of events that are triggered at a prescribed time (e.g. MIDI note on and off messages) and this is independent of the type of events to be triggered (e.g. sample lengths). In contrast to this, **event chain sequencing** happens when the timing of events is dictated by the nature of the events themselves.

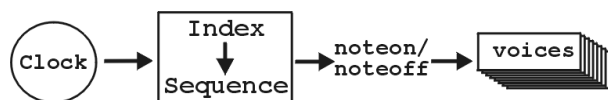


Figure 1. Traditional clock-based sequencing triggers events regardless of the nature of events to be triggered.

In relation to the *Speechcutter* application, the form of polyphonic management system departs from the traditional methods in favour of a layer of polyphony management that interacts with the voices themselves, so that events determine the sequence timings. Playback polyphony is handled dynamically by the system, and

sequencer polyphony is determined by the user but implemented using the new objects. Furthermore, the possibility of interleaving and bifurcating chain event sequences is presented, and so a mechanism has been developed to ensure that this process is robust in real time performance.

## 2. EVENT CHAIN SEQUENCING

Previous applications<sup>2</sup> developed by the author had used a form of sequencing hereby known as **event-chain sequencing** (figure 2), where each event is triggered by each previous event's termination. More recently, the *Speechcutter* was developed as a tool for creating analysis-based sequences of speech fragments based on research into taxonomic systems for classification of speech timbres (Kelly, 2006). Originally this consisted of an analysis tool for a single sound file and a set of sequencing tools. Each method of sequencing (e.g. brassage, fragment time-order) had its own monophonic voice, so that there was one voice per-method and no linkage between each method (sequencer). Since event-chain sequencing can be used to reassemble a sequence of segmented audio with virtually no clicks or glitches between each segment<sup>3</sup>.

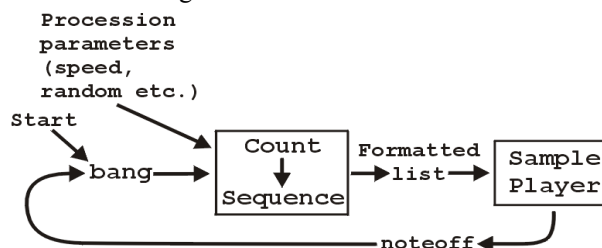


Figure 2. The basic architecture of a simple monophonic chain-event sequencing system. The First box contains

<sup>2</sup> For example, *Troubled Waters* for Max/MSP at <http://www.sharktracks.co.uk/software/msp/troubledwaters.zip>

<sup>3</sup> Although this is only true when the material is played back at pitch, using **tabplay~** in PD for example, since **tabplay~** emits a bang that is accurate to the end of the table rather than an external measurement of time. Externally measuring the amount of time taken to play a section of time is necessary when using **tabread4~** - so that the pitch can be changed by varying the messages sent to **line~**. Then, very slight glitches are heard in the output, but these are unnoticeable unless it is the *original* phrase that is reassembled.

<sup>1</sup> [Http://www.crisap.org](http://www.crisap.org)

the sequence information, and a counter that steps through the sequence. Procession parameters such as speed of procession, randomization and so on, are used to control the sequence as it unfolds.

It was decided that this application be developed so that multiple sound files could be analysed and sequenced simultaneously. This developed into *Speechcutter-poly* where eight sequencers access eight *slots*, each of which contains a set of analyses (onsets, pitch, high frequency profile, loudness and voicing) that determine the fragmentation of a single sound file according to banded thresholds. The possibility to link sequences together so that for example the loudest fragments of one file are interleaved with the unvoiced fragments of another, became apparent. Therefore it was necessary to develop a flexible system of polyphony management, so that alterations to the system could be made in performance.

### 2.1. Meta-Sequencing

Taking the basic principle of event-chain sequencing it is fairly simple to devise a scheme whereby several sequencers can be interleaved. The count and sequence information is still stored in the sequencer module for each analysis of the voice, but these are triggered from a meta-sequence module that steps through each sequencer one by one, or plays them in a random order. Figure 3 shows the scheme for this system, and assumes still that each sequence has one voice. In fact, this system if triggered only once will only use one voice at a time, but further polyphony for each sequencer would be necessary if there were an arbitrary pulse applied instead of the chain-events such as a **metro** object in PD or Max so that clock-based and event-based sequencing could be used together in one application.

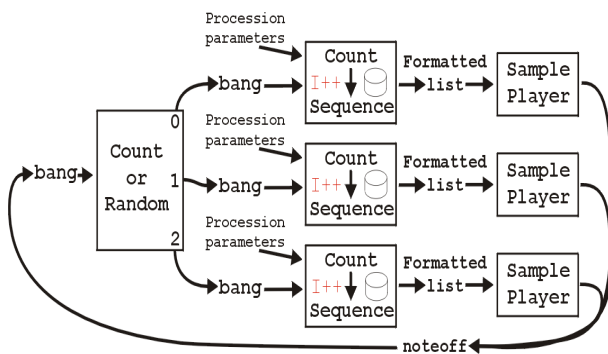


Figure 3. A more complex system involving three linked sequencers, each with a sample player, sequence and counter, but controlled by a central meta-sequence module (Count or Random, which steps through the sequence of sequences linearly or randomly).

### 2.2. Polysequencing – Realtime Complexity

So far this is fairly simple to achieve and uses few resources to create complex sequences of sound, where each component sequence of the meta-sequence proceeds position-wise independently of the others. However this is a fixed system of three, and the possibilities for greater flexibility in terms of live

performance become apparent by what this system cannot do:

1. Only one meta-sequence can run at a time with any coherent ordering of the segments
2. Transposing the segments down but keeping the time of procession the same results in note-stealing
3. Triggering the sequence from an external source (e.g. metro) results in note-stealing, and there is no way to set polyphony to overcome this.
4. Sequences may not be de-coupled (i.e. The architecture is fixed).

The *Speechcutter-poly* system contains eight independent sets of analyses of user-input sound recordings, and eight separate sequences. In order to develop this to include metronomic rather than chain-event sequencing and so that segments may overlap when transposed down, it would be possible to construct the system to include multiple voices per sequencer. But in order to minimize resources used, 64 voices of playback may restrict the software to only run on more powerful machines, and when a sequence of only four simultaneous segments is running there is no need for extra voices to be on. Instead a bank of 16 voices is used in a chain, where each sample player passes the note onto the next if it is busy. Another 16 voices are available if the user so desires, but this still complicates matters in terms of how many voices to allocate to each sequencer as defined by the following question: *If two voices are used for each sequencer, then clock-based sequencing can cause notes to be cut off (note stealing), but if the sample players are allocated dynamically then how can the system detect where each note came from originally (and hence which sequence to trigger next) when linked event-chain sequences are running?*

A system could be designed to switch between static and dynamic allocation, but this is further complicated when it is possible to interleaved and decouple sequencers dynamically, so that even if the sequence of origin is known it's relationship to others may have changed once a note has been played. This could potentially stall the system each time a change is made in the linkage of sequences to each other, and other sequences could be triggered by accident.

### 2.3. Dynamic Routing and Linked Sequences

Four objects have been created by the author in response to these problems, **sieve**, **maskxor**, **valve**, and **polymap**.

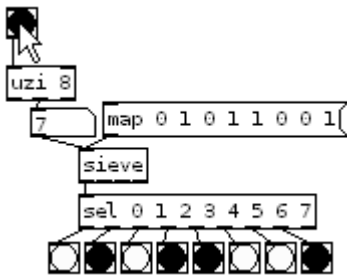


Figure 4. The floats into the inlet only pass through the sieve object if there is a corresponding non-zero value in the array at that index.

**Sieve** is a demultiplexer that stores an array of floats, and will only output a value if it indexes a value not equal to zero in the array as shown in figure 4. Floats are treated as integers (indexes to the map) in this instance, and the sieve object is used to route the correct note on and note off messages to the correct meta-sequencer (since each meta-sequencer addressed a number of sequences). This works well when the machines are running in a fixed configuration, but a problem arises when one sequencer is de-coupled from another whilst a note is playing. If the sequencer playing the note is one that has been de-coupled from the meta-sequencer driving the system, the note off message that tells the meta-sequencer to trigger another sequence element would not now pass through the sieve and so the sequence will stop.

The solution to this problem lies with the **maskxor** object (figure 5). This creates a bitmap that is the result of an exclusive or evaluation of each element of two arrays.

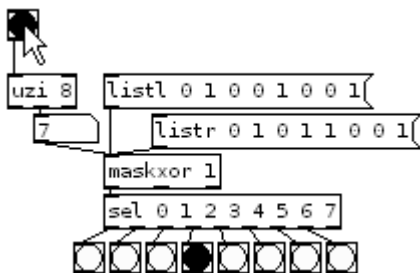


Figure 5. The difference between listr and listl is only one element, and the index of this element is the only number that will pass through the maskxor object.

The **listr** array is set to be the sieve mask prior to the decoupling, and the **listl** array is the sieve mask after the decoupling. So if a note off message comes from a sequence that has been decoupled then it may be used to trigger another element from the new meta-sequence. De-linking sequences in this instance results in only one meta-sequence playing, and sequences that were disconnected from the chain terminate. The potential to create **bifurcating meta-sequences** – meta-sequences that split into two independent meta-sequences requires that de-linked sequences are re-started as the sequences are de-linked from one another.

Each sequence has a corresponding **governor sequence**, one of eight master controllers that is associated with the first sequence in each meta-sequence or chain of

sequences. These could simply be switched off and then on again if the sequence were to bifurcate, but this would not be event-driven but arbitrary. In order for this to happen according to the timings of events it should be triggered by a note off message, and this presents another problem. Depending on what sequence's element of the meta-sequence is playing when the bifurcation occurs, the note off message will not necessarily correspond to the governor controlling the meta-sequence. Once the sequences are de-linked, a note off message previously corresponding to one governor would be sent to another, and the first sequence would terminate.

The **valve** object routes floats (note off messages) according to whether another float (the **governor id**) is present within the bitmask, and so this is used to identify which governor needs to be triggered as another is started in order to bifurcate the stream of events. This is shown schematically in figure 6.

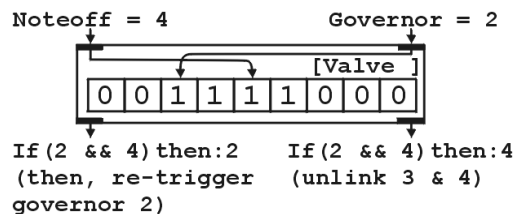


Figure 6. The **valve** object allows for sequences to bifurcate.

The **maskxor** object is cleared after this operation. Thus it is possible with these three objects to create a dynamic system of linkage between independent sequencers without the need to create exhaustive networks of all the possible routes a message might take in PD.

### 3.POLYPHONY CONTROL

The polyphony of each meta-sequence is controlled by the **polymap** object (figure 7, top). Each voice is given its own number, and sends a message to the polyphony control system on a note on or a note off. This consists of the structure [origin(sequence), destination(voice), on/off(1/0)] and this is unpacked and the origin is sent to the sieve object before it is repacked, ensuring that no elements of the 32 x 32 matrix within polymap are set that are not from the meta-sequence concerned. Elements within the matrix (bottom) are set, up to the maximum number allowed by the maxpoly parameter at the right inlet (or by the creation argument).

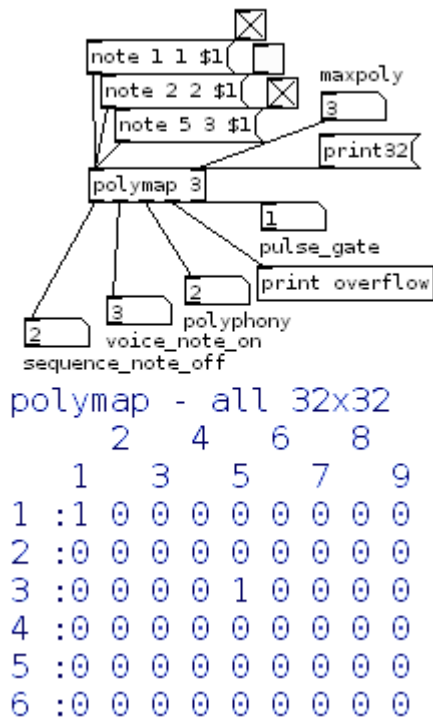


Figure 7. The **polymap** object (top) stores sequence/voice pairs in a matrix (bottom) up to the amount dictated by the maxpoly parameter. When a note on message is sent, a new instance of what playback voice was invoked is reported. When a corresponding note off message arrives, polymap reports that the sequencer has a voice free, and this can be used to trigger a new event

Playback voices invoked are reported at a note on event, and sequencer voices freed are reported at a note off event. This means that the polyphony can be managed in terms of how many voices a sequencer uses rather than how many playback voices are used, so events do not drop out of sequence in a phrase no matter how it is interleaved. This closes a gate when this polyphony is reached so that it can be used to control external metro objects, shutting them off so that they do not trigger unheard events, and the sequence is preserved<sup>1</sup>.

#### 4.CONCLUSION

The resultant system shown in figure 6 allows for extremely precise control of polyphony, structure and linkage within a network of eight sequencers and 32 playback voices. It is configured in such a way, using the four new objects described, that multiple sequencers can interleave or bifurcate, and meta-sequences can grow or shrink in performance time. It's elegance lies in the way the system adapts to preserve the sequential ordering of each sequence instead of cutting off events halfway, or negating them altogether, although this is also controllable from within the *Speechcutter-poly* application.

<sup>1</sup> Although this can be overridden on the *speechcutter-poly*, so that the timing of the sequence has precedence over the sequence order.

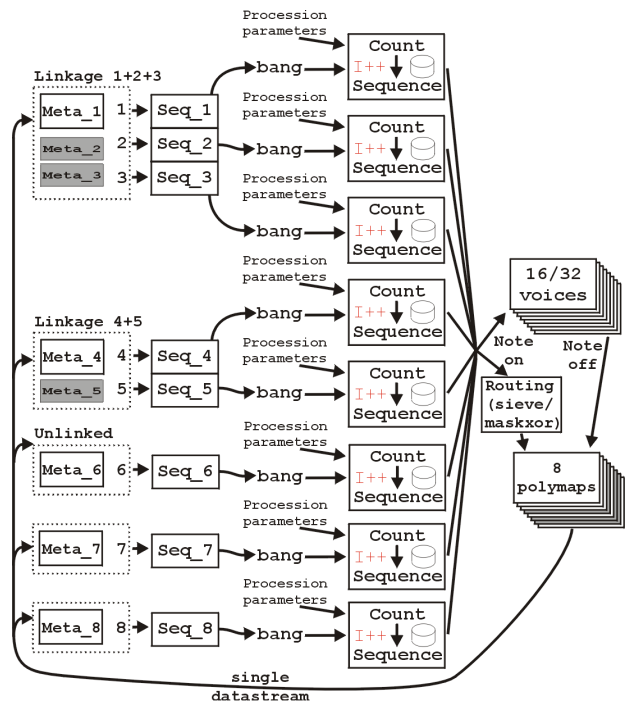


Figure 8. The architecture of *Speechcutter-poly* in some detail. The single datastream at the bottom of the figure is sorted within the individual meta objects. The meta objects are the triggers for sequence events, and whether or not the next event happens depends on whether a polymap object reports a freed *sequence* voice (rather than *playback* as in traditional polyphony management). Managing event data in this way is different from other polyphonic systems in that the polyphony management and data flow is integrated into the generation of events, and thus it can be made to create sequences of material that evolve according to the nature of the material rather than an arbitrary scheme. Furthermore, it is a system of sequencing that is interactive and live, and may point towards future developments in structured improvisation using live electronics.

#### 5.REFERENCES

- [1] Kernighan, B and Ritchie, D. 1978, 1988. "The C Programming Language", Prentice Hall, New Jersey, USA
- [2] Puckette, M. 1996. "Pure Data: another integrated computer music environment." *Proceedings, Second Intercollege Computer Music Concerts*, Tachikawa, Japan, pp. 37-41
- [3] Kelly, E. 2002. "Time in Music: Strategies for Engagement", PhD thesis, University of East Anglia Library, Norwich, UK.
- [4] Kelly, E. 2006. "Deconstructing Speech: New Tools for Speech Manipulation." *Organised Sound*, Volume 11, No. 1: Cambridge University Press, UK.

#### 6. Source Code:

1. Pure Data is available from Miller Puckette's website at <http://crca.ucsd.edu/~msp/software.html>
2. polymap, sieve, maskxor and valve are available via cvs from the Sourceforge website: <http://pure-data.sourceforge.net/old/developer.php>
3. These objects are also available as part of the Pure Data Extended release from Hans Christoph Steiner: <http://at.or.at/hans/pd/installers.html>
4. *Speechcutter* is available from the CriSAP website at [http://www.crisap.org/index.php?new\\_creative\\_tools](http://www.crisap.org/index.php?new_creative_tools)
5. The PhD thesis referred to above may be accessed at <http://www.sharktracks.co.uk/epk/thesis.html>