THE *TimeStruct* TOOLKIT: IMPOSING TEMPORAL STRUCTURE IN Max

Aron Glennon
New York University
Music and Audio Research Lab (MARL)
apg250@nyu.edu

Nick Didkovsky
Algomusic
nick@didkovky.com

ABSTRACT

Standard tools for making high-level decisions about temporal structure in music do not exist for computer music composers using Max. We describe a toolkit for temporal structure creation and manipulation in Max that we developed to address this. We discuss the toolkit's objects in the context of a proposed general 'time structure' representation scheme. A composer makes high-level compositional decisions with these objects by specifying certain general properties about the form of a desired time structure while leaving details about its contents to chance. The user may influence how the computer randomly chooses values for these details by specifying a desired probability mass/density function. A time structure's contents may be unfolded over a specified length of time (using objects provided in the toolkit) in order to appropriately interface with other Max objects.

1. INTRODUCTION

Music is an art form composed of sonic 'events'. A major focus for any music composer is the organization of these events in time. While many contemporary music composers prefer to notate the exact timing of every event in a piece, others prefer to work with the event timing and general temporal structure of their material from a higher level [4].

For computer music composers, this is typically not possible using sequencer software because individual events must be manually placed on a timeline. Therefore, those interested in making high-level decisions about event timing must turn either to less restricted software packages like Max [5], Chuck, or SuperCollider or to full-scale programming languages like Java or C++. Historically, Max has been the most popular choice for several reasons: (1) Its learning curve is not as steep as that of Java or C++; (2) it has a larger user community than any other dedicated music programming environment; and (3) its graphical interface is less daunting for non-programmers than a terminal window.

If composers are more apt to turn to Max when typical sequencer software is limiting, then it is important that they are afforded the appropriate tools within Max to realize their vision. While Max has a number of useful objects to creatively generate and manipulate audio and MIDI data, its object set for temporal structure generation and manipulation is quite limited. Max's *seq* and *timeline* objects have many of the limitations that typical sequencer software does and are, therefore, not ideal for this task. The limitations exhibited by these objects were the impetus for creating the *TimeStruct* toolkit. It should be noted that externals do exist that could allow the same functionality that this toolkit brings to Max (e.g. *maxlisp*, *csound~*, *cmix~*, *chuck~*, *js*, *live.object*), but these externals all require proficiency in another language or software environment and therefore are not sufficient for a composer familiar and comfortable only in the Max environment.

2. THE *TimeStruct* TOOLKIT

We designed the objects inside the *TimeStruct* toolkit [1] - in C for Max running on Mac OSX - to provide users with the facility to use only high-level decisions when determining how to order their sonic material in time.

A few of the many possible applications of this toolkit follow:

- Automatically segment a piece into different movements
- Further segment movements into sections, sentences, phrases, etc.
- Specify note densities as opposed to specific note onset locations
- Produce time stamps for tempo variations within a performance
- Provide microstructure for granular synthesis attributes [3] or parameter trajectories for time-evolving audio effects

The *TimeStruct* toolkit is also applicable to many tasks beyond those related to audio. Additional applications range from those in video art (analogous to those listed above for audio, using Max's Jitter Library) to general list creation and manipulation.

2.1. Representation of Events

In the context of this paper, an 'event' corresponds to *any* musical change (e.g. note onset, timbral shift). A piece of music can therefore be segmented into a number of events, each corresponding to change or evolution at some point in time along some musical dimension. This segmentation can exist on only one level (e.g. an event can be associated with any instance of change) or on multiple levels (e.g. a piece could be segmented into 'section' events, which could then be segmented into 'note events', which could then be further segmented into events marking how each note evolves timbrally or dynamically). Therefore, different kinds of events may exist on different time scales. However, no matter the representation, each event will have a beginning ('event marker') and, with every beginning, an associated time stamp.

Being time-scale neutral, the *TimeStruct* toolkit represents a series of events as a list of float values increasing from 0.0 to 1.0, where each float-valued time stamp represents its corresponding event's placement on a timeline relative to the entire length of that timeline (i.e. duration of the entire series of events). Thus, the toolkit works with lists of unit-less timestamps that we refer to as '(generalized) time structures.' This representation was designed to provide all objects in the *TimeStruct* toolkit with a common fundamental unit with which to work and also to allow for easy scalability to any duration. When the user has finished manipulating a time structure with the toolkit and is ready to use it to control events in Max, they can utilize the *time_scaler* object (see section 2.5) to translate its unit-less representation into millisecond or second values spanning any desired length of time.

2.2. Time Structure Creation

The *lookup* object in the *TimeStruct* toolkit creates a list of event markers (and therefore, implicitly, a series of events). The user specifies the number of unique events to produce and the *lookup* object outputs a list of values from 0.0 to 1.0 of this size (see Figure 1).

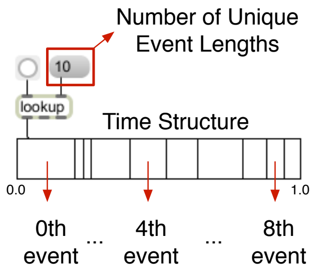

Figure 1. The *lookup* object.

Since the output list ranges from 0.0 to 1.0, it can be used as a time structure by itself, but its intended use is as a lookup table from which to choose events when building a separate time structure. The latter application allows for the creation of a time structure containing multiple copies of the same event.

There are two objects in the toolkit used for indexing into the lookup object's output to create time structures: *index_structure* and *create_structure*.

The *index_structure* object creates a time structure by indexing into the *lookup* object's output using a specified list of indices (integers). For example, if the list of indices sent to the *index_structure* object is (0 4 3), the *index_structure* object will output a time structure containing the events at the 0th, 4th, and 3rd positions in the lookup table. The output list is normalized to range from 0.0 to 1.0 so that it has the appropriate form of a time structure and will therefore be recognized as such by the other objects in the toolkit. This normalization changes the absolute length of each event, but retains the relative length of each with respect to every other. Thus, the character of the structure (i.e. its shape) is maintained (see Figure 2a). Recall that the absolute length of each event is unit-less and only an indication of its size with respect to the entire time structure, so no important information is lost during normalization.

The *create_structure* object randomly selects a specified number of events (with reselection possible) from the *lookup* object's output and, like the *index_structure* object, normalizes the resultant list to range from 0.0 to 1.0 (see Figure 2b).

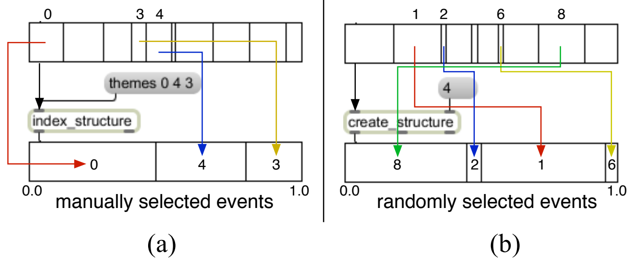

Figure 2. (a) The *index_structure* object. **(b)** The *create_structure* object.

2.3. Themes and Patterns

As discussed above, the *index_structure* and *create_structure* objects use the output of a *lookup* object as a lookup table for creating time structures. When the *lookup* object is used in this way, the location (i.e. index) of each event in the lookup table can be viewed as its corresponding 'theme'. In this context, a theme is simply a unique identifier for an event, so that if multiple copies of an event are used in a time structure, it is easy to determine where each exists. Therefore, whenever an object from the toolkit outputs a time structure, it also outputs a list of theme values associated with the different events in the time structure.

If a time structure is created to specify section boundaries within a piece, then the theme values provide a useful representation for keeping track of which sections repeat and where. In such a situation it may be desirable to strip the original themes away from their associated sections (i.e. events) and replace them with different theme assignments (e.g. when the repetition of a verse in a pop song is longer than its first occurrence or when two different sections in a piece are of the same length, but contain different content).

We developed two objects in the *TimeStruct* toolbox to provide the user with high-level control over theme assignments: *random_themes* and *pattern*.

random_themes simply generates a random list (of specified length) of theme numbers (i.e. integers) from zero to some specified maximum theme value.

pattern takes a list of themes and either inserts a specified number of theme patterns into the list or replaces portions of the list with these patterns. For example, if the theme pattern is (1 1 2) and a list of (3 0 2 5 1 2 1) is input to the pattern object, then inserting two repetitions of the pattern might return the following: (3 0 (1 1 2) 2 5 (1 1 2) 1 2 1). Alternatively, if 'replacement' is selected, then the output will look something like: ((1 1 2) 5 (1 1 2)) (see Figure 3). This allows the user to place repetitive segments into a time structure.

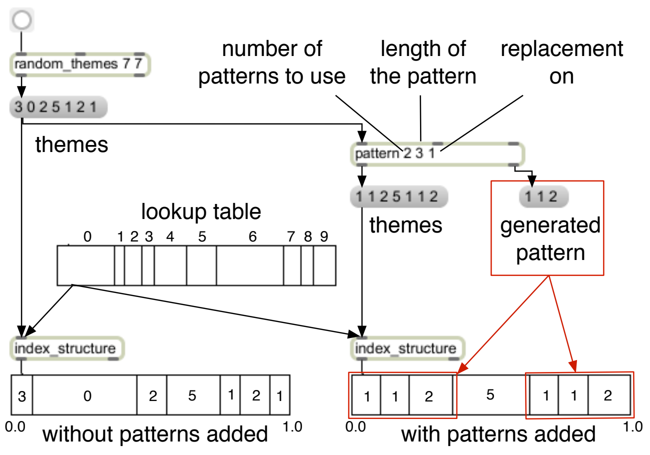

Figure 3. Adding event patterns to a time structure using the *pattern* object on the output of the *random_themes* object.

2.4. Incorporating Probability

The amount of high-level control given to a user would be severely limited if uniform random selection were the only mechanism used by *lookup* to choose event lengths, *create_structure* to select indices, and *random_themes* to generate theme numbers. A more useful set of tools would allow the user to provide a probability mass/density function [2] that specifies how to randomly choose values in each case mentioned above.

With the above consideration in mind, we designed the *lookup*, *create_structure*, and *random_theme* objects to allow the user to specify a list of probabilities used for selection. This list of probabilities is appropriately linearly interpolated for each situation, as will be discussed below.

The *lookup* object requires a continuous probability density function when selecting float-valued event lengths. To approximate the behavior of this density function, any list of probabilities sent to the *lookup* object is linearly interpolated to 1000 points, each representing a probability that a float value will be selected by *lookup* in a 1/1000th sized-interval between 0.0 and 1.0. When choosing event lengths, the corresponding distribution is sampled and an interval is returned. In order to generate a specific value inside that interval, it is uniformly sampled and the value chosen is returned as the event length.

For *create_structure*, the list of input probability values is interpolated to the length of the incoming lookup table, so that each event has an assigned probability of selection.

The *random_themes* probability mass function must have as many values as there are possible theme numbers, which is specified by the user as the maximum possible theme number to include in the list.

In each case, we designed the objects so that the list of values input into each, representing selection probabilities, do not have to sum to 1.0. Each object internally normalizes the probability mass function obtained after interpolation so that the user only has to decide upon the general shape of the mass function when inputting desired 'probabilities'. The probability distribution used for sampling is also normalized appropriately.

Realizing the applicability of this concept to a number of different paradigms in Max, we created the *distribution* object as a general-purpose mass function and distribution creation tool. The *distribution* object takes a list of values and a desired mass function list length and outputs a properly normalized probability mass function and distribution (each as a list of floats) of this length (using the same methods incorporated into the *lookup, create_structure,* and *random_themes* objects for interpolation and normalization) (see Figure 4).

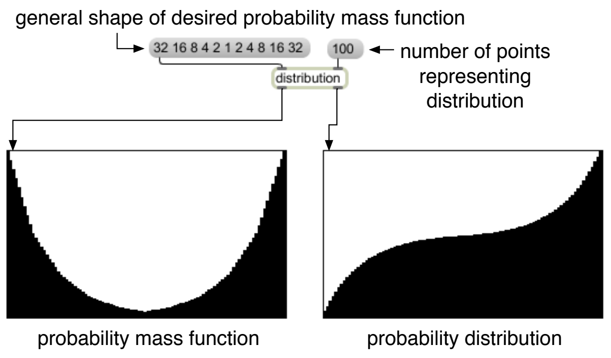
general shape of desired probability mass function
32 16 8 4 2 1 2 4 8 16 32
100
number of points representing distribution
distribution
probability mass function
probability distribution

Figure 4. The *distribution* object.

2.5. Time Structure Manipulation

A user may want to manipulate the locations of event markers in a time structure after it has been created. For example, if the time structure is used to specify when percussive hits occur in a pop song then the user may want to only allow hits on specific integer divisions of the beat (i.e. quantization). To accommodate such a need, we created two objects to manipulate event marker locations within a time structure: *quantizer* and *compressor*.

The *quantizer* object has two modes: *relative* and *absolute*. The *relative* mode quantizes the lengths of the events (see Figure 5) so that only a specified number of integer multiples of a fundamental event length can exist. The *absolute* mode quantizes the locations of the event markers to fit on a grid created by dividing the entire time structure up into a specified number of evenly spaced time values.

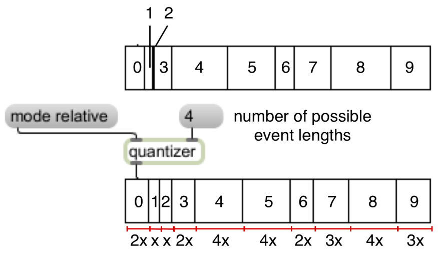

Figure 5. The *quantizer* object in relative mode using four possible event lengths.

The *compressor* object imposes a specified maximum ratio between event lengths in the time structure. For example, if a compression ratio of two is specified, the longest event is made twice as long as the shortest event. All other events are compressed/expanded proportional to their original lengths relative to the lengths of the longest and shortest events. This allows the user to either make long events longer and short ones shorter (in the case where the specified compression ratio is larger than the inherent ratio of the input list) or long events shorter and shorter ones longer (in the case where the specified compression ratio is smaller than the inherent ratio of the input list) (see Figure 6).

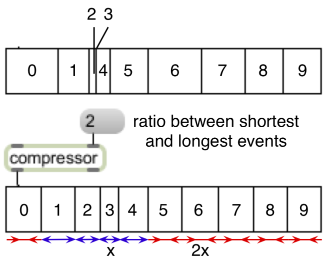

Figure 6. The *compressor* object using a compression ratio of two when the inherent input ratio is much larger than that.

2.6. Event Output over Time

Once a time structure is created, its event marker locations must be translated from relative time values into absolute time values so that it may be used within Max. This simply requires a scaling of the interval (0.0, 1.0) to a time scale that is understood by the object using the time structure's event information. We created the *time_scaler* object for this purpose.

The *time_scaler* object takes a list of numbers ranging from (0.0, 1.0) and scales them to range from (0.0 to x) where x is specified by the user. The result is a translation of the time structure from a unit-less quantity into the units of x.

We developed the *ramper* object to provide an intuitive and useful output of a list of event markers (in seconds) and their corresponding themes. The *ramper* object has one inlet and three outlets. The user first sends *ramper* a list of event markers concatenated with their associated themes. A bang, sent subsequently to the object, initiates output. When a bang is received at the *ramper*'s inlet, it outputs a linear ramp from (0.0, 1.0) over x seconds out of its first (leftmost) outlet to indicate the percentage of time elapsed in relation to x (the entire time structure length). Simultaneously, the second outlet sends out a ramp from (0.0, 1.0) for each event in the time structure over each event's duration. The third outlet outputs the theme associated with the event that the second outlet is ramping through. Therefore, sending a bang to the *ramper* object will 'play' its time structure, giving the user knowledge of the location of the play head through the entire time structure, the location of a play head set to run through the current event, and the theme associated with that event.

3. FURTHER WORK

The next logical step for further development of the *TimeStruct* toolkit is to start building structures using its objects in order to determine (1) if any useful functionality should be added to the existing objects or (2) whether supplemental objects should be developed. We also plan to develop Max for Live devices using these objects so that Ableton Live users are afforded similar mechanisms.

To download a beta version of the toolkit, visit:

http://homepages.nyu.edu/~apg250/MaxMSP_Externals.html

4. REFERENCES

[1] Fujinaga, I. *Max/MSP Externals Tutorial: Version 3.1*. Montreal, Canada: McGill University. 2004.

[2] Grimmett, G. R. and Stirzaker, D. R. *Probability and Random Processes*. Oxford, England: Oxford University Press. 1992.

[3] Roads, C. *Microsound*. Cambridge, Massachusetts, USA: MIT Press. 2001.

[4] Tenney, J. *Meta-Hodos and META Meta-Hodos*. Oakland, California, USA: Frog Peak Music. 1986,

[5] Zicarelli, D. and Taylor, G. *MAX Fundamentals*. San Francisco, California, USA: Cycling '74, 2006.