The Arugula Computer-Music Score Editor

Vance Maverick
Computer Science Division, UC Berkeley
maverick@cs.berkeley.edu

Abstract

Arugula is a general-purpose graphical score editor designed to create structures of notes, entities with time, duration, amplitude, and frequency. These structures may be translated into any textual computer-music language, such as cmusic [5] or the NeXT score-file format [2]. Arugula defines these structures through new graphical interfaces for expressing rhythm and tuning, applied to a hierarchical score structure.

1 The Note

In Arugula, each note carries properties called start, end, pitch, amplitude, and instrument. Each of these is mapped to the corresponding physical synthesis parameter by association with larger structures in the score.

Start refers the beginning of the note to a given timepoint (see below). Each timepoint is mapped to a single real time.

End places the end of the note at a fractional position between consecutive timepoints. The real time is computed in the same way as the start.

Pitch is simply a textual identifier, like "C#5". At every point in the score, there is a current tuning, which maps pitch to frequency.

Amplitude is a value in decibels, relative to a local default.

Instrument assignments for each note are textual identifiers, which Arugula translates into statements in the target language.

*Sponsored by the Defense Advanced Research Projects Agency (DARPA), monitored by Space and Naval Warfare Systems Command under Contract N00039-88-C-0292, and by California MICRO award 90-058. For a fuller description of the work, see [4].
2 The Stream

Intuitively, the Arugula notion of stream corresponds to the musical "voice". Operationally, it represents a group of notes which are subject to the same transformations of time and dynamics. Wherever two series of notes need to be modified differently, they belong in separate streams. The streams in a score form a hierarchy, enclosed by an infinite outermost stream. All other streams are bounded by timepoints (see below) defined in their ancestors.

To make this more concrete, consider an ensemble of instruments. In a conventional piece, they all "follow the same beat" but execute it in different ways. In Arugula this is expressed by establishing the beat as a series of timepoints in the parent stream, then subdividing the stream so that timing and dynamic variations may be added individually. Even parts for the same instrument may require separate streams, as when a pianist plays cross-rhythms between the two hands.

3 Tuning

The function of a tuning is to map the textual pitches associated with notes to frequencies. As in many systems, the tunings in Arugula are fixed. Such a scheme cannot cover all tuning needs [6]; but as we will see, Arugula provides unusual flexibility in the definition of a gamut. The mapping is at least conceptually a two-stage process; pitches are mapped to intermediate entities called tens, which in turn are mapped to frequencies. Most fixed realizations of the standard Western pitch system, for instance, define twelve tones per octave, but more than twice as many pitch names.

Tunings are constructed graphically by the composition of a number of operators. Each of these takes some number of "input" tones, and produces some number of "outputs", both possibly zero.

**Absolute** creates a tone at a given frequency.

**Ratio** creates a tone at a given frequency ratio to another.

**Division** creates equally-spaced tones to fill the interval between two tones.

**Stretch** scales a set of tones in log-frequency space, preserving the relative size of the intervals.

**Repeat** repeats a pattern of tones at a given interval.

**Mask** eliminates a tone used only for construction.

With these operators, users can implement most familiar fixed tunings, and an endless variety of new ones. Certain tunings, such as mean-tone, require some algebra and awkward construction; a more general model will require a full constraint-programming system.
4 Time

Time in Aragula is defined by timepoints, visible as vertical lines in the score display. Timepoints are created by divisors, which subdivide the interval between two timepoints (or two absolute times, in the case of “fundamental” divisors) by inserting more timepoints in between. This division may be repeated many times. These timepoints are generally equally spaced; the irregularities of actual ensemble timing are created in two ways. Either the timing may be modified by a time map [1], or it may be created irregular from the beginning. (Certain problems sometimes handled with time maps, for example “swing” eighth notes of jazz [7], may be more simply addressed with irregular division.) Time maps may be attached to divisors or to whole streams; both time maps and irregular divisors may be stored with names in global lists.

5 Dynamics

Each note carries a relative amplitude. In translating this to an actual amplitude, Aragula consults a series of amplitude functions, in a bottom-up traversal of the stream hierarchy. First, an amplitude pattern may be applied to the timepoints in a stream. This is created by a tree structure derived from the phonological work of Liberman and Prince [5]. (There are some modifications — the tree need not be binary, and the stress values define are continuous.) Then, the user may attach a time-amplitude function, such as a crescendo, to each stream on the path from the note to the root of the score. All these relative amplitudes are expressed in db; the final amplitude is simply their sum. These functions, both the continuous and discrete, may be stored in global lists.

6 Translation to the Target Language

For output, Aragula converts the score to the user’s target language using three types of text. Fixed prologue and epilogue passages begin and end the output. In between, each note is replaced by a block of text associated with its instrument identifier: in music, for instance, this would be a note command. Note-specific synthesis parameters are handled by macro replacement. For our local Lisp-based synthesis environment, an instrument “Bell” might be associated with the text

`(<bell :start $sec(start)) :dur $sec(end)-sec(start)) :freq $Hz(frequency)) :amp $linear(amplitude))`)

The macro strings $ and $ may be changed by the user if they conflict with the lexical structure of the target language. An Aragula score includes a global list of instrument identifiers, mapping each one to a code fragment like this.
7 Conclusion: Further Work

There are several obvious ways Arugula can be improved. First, it needs some form of real-time synthesis. Both MIDI and NeXT Sound Kit output are conceptually similar to the present language system, so their inclusion is only a matter of time. Further, the only unit of communication between Arugula and outside synthesis algorithms is the note. This excludes "effects" and other processing applied to the sound after the synthesis of notes. Also, the model of notes is too narrow: even commercial synthesizers provide parameters independent of the four simple properties Arugula manipulates now. Future versions will add user-definable note properties, as well as "processing events", inserted among the rhythmic dividers in the hierarchy of streams.

As research, though, the most interesting change for Arugula will be in its model of rhythm. In performance and perception, rhythm is manifested through interrelated effects of several physical parameters, principally timing and amplitude. The present model provides no structure which might coordinate these effects. Considerable work must be done in separate test programs before a model can be found which is both flexible and comprehensible enough for inclusion in a working scoring system.

References