Polymorphic transformations in Kyma

Carla Scaletti
The Symbolic Sound Corporation / P.O. Box 2530 / Champaign IL 61825-2530 USA
Telephone: 217-555-6273 E-mail: carla_scaletti-symsound@nova.novanet.org

ABSTRACT

In the new generation of computer music languages, there is no longer a strict distinction between signal level and event level processing, between "orchestra and "score" (for example, Scaletti 1987, Dannenberg et al 1991, Burk 1991, Dessain and Honing 1991, Lippe and Puckette 1991). In such languages, a general transformation function can sometimes be inefficient or yield unexpected results.

One solution would be to write several versions of each transformation, each one appropriate to the signal that is to be transformed. The solution taken in Kyma is to embody the information about how to transform the signal within the signal object itself. The Kyma language provides a means for defining such encapsulations as well as a Parameter/Transformer Sound with flexible options for applying transformations to hierarchical structures. These tools make it possible to define a polymorphic transformation, that is a single transformation that can be applied to many different kinds of objects.

2. PROBLEM STATEMENT

In order to be applicable to any arbitrary input signal, a polymorphic transformation must assume nothing about the signal to be transformed. One way to do this would be to treat every signal as a stream of samples having no particular meaning and to extract and modify parameters using standard signal processing operations. Not only is this an inefficient way to transform the parameters of many signals, but some parameters are virtually impossible to extract using signal processing techniques.

For example, a general frequency-shifting transformation might perform a spectral analysis, scale each component of the spectrum in the frequency domain, and then resynthesize the same spectrum at the new fundamental frequency. If, for example, the signal to be frequency shifted were being generated by a fixed-waveform oscillator, it would be more efficient to symbolically rewrite the frequency parameter of the oscillator before it generates any samples.

A signal processing algorithm that could extract and modify the seed used by a random number generator to set the frequencies of a sequence of grains would be extremely complex; however, it would be trivial to symbolically rewrite the seed of the algorithm before the samples were generated.

The more that is known about the signal generation algorithm and its parameters, the more efficiently it can be transformed. If this knowledge is embodied in the transformation, a special transformation must be constructed for each kind of signal. If the knowledge is instead encapsulated with the signal itself, then a single transformation can be applied to any number of signals, and both signals and transformations can be combined or composed. This relieves the composer of the burden of checking for all data types and situations each time the transformation is applied to a new signal.

In a language like Kyma—where operations like concatenation, summation, multiplication, difference and others can be applied in both signal-level and event-level contexts—polymorphic transformations can sometimes have unexpected results; this is not because the transformations are mathematically incorrect, but because the composer has conceptually grouped two or more discrete objects as one functional unit. The composer expects the objects to be transformed as a unit, but the transformation blindly transforms each object individually with sometimes unexpected results.

Consider, for example, the concatenation of a short burst of noise followed by a silence and fed into a resonator. The result would sound like the classic plucked string sound of John Chaffin and Smith (1983). If the duration of this signal were to be scaled to twenty times the original duration, the composer might expect to hear a pluck with a slow decay time; however, there is nothing in the signal specification to indicate that the noise burst is a transient excitation and that the silence is a decay time. Thus, the transformation affects the noise burst as well as the silence, yielding a result that no longer sounds anything like a plucked string.
While unexpected results are often pathways of discovery, the composer should also have the option of grouping or encapsulating the constituent objects into a single functional unit and to specify how that unit’s parameters should behave under transformation.

3. BACKGROUND ON THE KYMA LANGUAGE

Kyra is a fully functioning and supported language used by a large number of composers and researchers (see, for example, papers by Belet, Haken, Holloway, Pomer, and Walker in these proceedings) and has been commercially available since 1990. Kyra is a sound specification language based on a unit called a Sound. A Sound can represent any point in the continuum from the lowest level signal specification to higher level compositional algorithms. In Kyra, there is no enforced separation of signal level and event level processing; between the concepts of orchestral and score; these models are suppressed by arbitrary hierarchical structures constructed by the composer out of uniform Sound objects (Scalietti 1990).

A Sound is either atomic or a function of one or more Sounds. The composer constructs arbitrary hierarchical structures by specifying complex Sounds as functions of simple Sounds. There are more than one hundred built-in classes of Sounds in the current release of Kyra, including generators, modifiers, and cominers. In Kyra, all Sounds are objects: the transformation of a Sound is itself an object.

Since Kyra is not based on the modes of music notation and performers, it does not always have the advantages (or the restrictions) of commonly understood meanings. For example, in Kyra, there are no built-in objects representing such things as “trills”, “glissandi”, “mordents”, but there are tools that could be used for constructing these objects as well as other, new kinds of sonic objects having no historical precedent.

The Sound object provides a relatively general, uniform structure. Meaning is assigned by the composer through context, structure, naming, and encapsulation. As a simple example, the two concatenations shown below are acoustically equivalent, but the composer has given the sequence more structure by breaking it up into meaningful subsequences. (In this example, each Sound is represented by an icon with any argument Sounds displayed to its right.)

To abstract from one particular Sound to an entire class of Sounds, the composer sets some parameters to variables or expressions involving variables (Scalietti 1991). This effect-every defines the way in which the object interacts with the outside world: defining its set of externally accessible parameters and setting up relationships between its internal parameters. When a Sound of a user-defined class is played, it is first expanded into its constituent Sounds and then those Sounds are played.

4. THE PARAMETER TRANSFORMER SOUND

The Kyra ParameterTransformer is a Sound object that performs structural transformations on its argument Sound. In Kyra, all Sounds are objects; thus the transformation itself is also a Sound object. The kinds of transformations it performs are symbolic, and they are applied prior to compilation and samples generation. Kyra also provides ways to vary parameters during samples generation, but these are discussed elsewhere (Scalietti 1990); the ParameterTransformer discussed here carries out pre-performance manipulations and transformations. The expansion of a ParameterTransformer is its subSound with appropriately transformed parameters.

Among its other arguments, the ParameterTransformer has a subSound and a transformation that is to be applied to that subSound. The transformation is written in Smalltalk-80 supplemented by some additional messages and conventions particular to the ParameterTransformer. The Sound that is to be transformed is referenced as self, and any parameter of the Sound and its subSounds can be accessed by name. For example, the statement $nduration: 1 second$ would set all durations, wherever they

250
occur in the structure, to one second, and snd duration: snd duration \* 0.5 would scale each duration parameter to half of its original value. Transformations can also depend on the untransformed time and the untransformed total duration of the ParameterTransformer's sub sounds, referenced respectively as time and totalDuration. For example, and duration: snd duration / (1.0 + time / totalDuration) would effect a linear change over the course of the Sound from the original durations to durations half as long.

There can be any number of parameters named in each line, and there can be any number of lines. Smalltalk expressions can be used to compute parameters in terms of their original values or in terms of other parameters; for example,

\[ \text{snd duration: (1.0 / snd frequency)} \] seconds.

\[ \text{snd retrograde: snd retrograde not.} \]

Each Sound applies the transformation block first to itself and then recursively to its subsets. If a Sound in the structure does not contain one of the referenced parameters, the transformation of that Sound is abandoned and the transformation is recursively applied to its subsets.

In general, a conceptually atomic Sound (that is, one that should act as a functional unit) is transformed before it is expanded, whereas a composite Sound is expanded before it is transformed. This is in keeping with the expectation that an atomic Sound will behave as an integrated unit and that transforming its subsets separately could have unexpected results.

Because the Sound structures have no inherent meaning, the composer is provided with a means for overriding the default assumptions of the ParameterTransformer. There are expressions that force a Sound to expand before the transformation is applied and others that can be used to block transformation of a Sound or group of Sounds. (Scalaletti 1990)

5. EXAMPLES

The plucked-string example given in the problem statement illustrates the potential problem in transforming a Sound that is conceptually a single entity but is in actuality made up of several Sounds. One way to leave the noise burst duration the same but scale all other durations by 20, would be to construct a conditional transformation, for example, (snd isKindOf: Noise) ifTrue: [snd duration: snd duration \* 20]. Even this conditional transformation would not be quite correct since the total duration of the Sound would be twenty times the original duration of the silence plus the duration of the noise burst. While the conditional transformation accomplishes something close to the desired result for this particular case, it has a disadvantage in that it works correctly only on this particular case. This approach requires that a special transformation be written for each kind of Sound.

In order to get the expected result, the duration of the noise burst, rather than being a function of the total pluck duration, should be a function of the frequency: a single period of the pluck frequency. The duration of the silence following the burst of noise is a function of the total desired duration minus the duration of the noise burst. Using the Kyna class editor, the composer can create a new Sound class based on this parameterized instance: the new class has frequency and duration parameters and correctly maps them to the durations of the noise burst, the period of the resonance, and the duration of the silence. This new object can be transformed without any special conditions, and it will respond to an atomic Sound would be expected to respond: the transient noise burst is always one period long, and the ringing is a function of the specified duration.

It is not always the case that a composite Sound is thought of as a single sigma: some Sounds are conceived of as collections of discrete events and should be transformed accordingly. The CellularAutomaton, for example, takes a collection of sub sounds and algorithmically assigns start times to them. Usually, when composers perform a time-dependent transformation of a CellularAutomaton they expect to transform it after it is expanded—that is, after the Sounds have been assigned their start times—and not before.
Most of the so-called classic transformation problems arise from the assumption that a composite object is in fact atomic; in other words, these objects lie in that interesting region of ambiguity between signal processing and event processing. Confusions arise when concepts like "vibrato," "trill," "glissando," and "mordent" are transplanted whole from traditional music notation into a functional sound description environment. A functional sound description does not, on its own, necessarily behave like a "vibrato," even if it happens to sound like vibrato. The "vibrato behavior" must be built into the definition; if the definition is correct, then it will behave as expected under transformation.

Consider a simple, fixed-wavelength oscillator. Since frequency is a parameter of the oscillator, the ParameterTransformer can simply rewrite the value of the frequency when the Sound expands. But what if the oscillator were being used to control the "vibrato" rate of another Sound? In this case, when the frequency transformation is applied, the frequency of the main oscillator and the rate of vibrato will both be altered. If the composer wants a fixed vibrato rate, no matter what the frequency of the main oscillator, then that should be built into the definition of the vibrato object. For example, the frequency that controls vibrato rate could be renamed to rate so that it will be transformed independently of frequency.

To avoid changing the rate of a vibrato when the duration of the trill is transformed, the composer could define a trill as a Repetition of a two-note pattern, possibly concatenated with a ending turn. When the trill is asked to change its duration, it would not change the durations of the objects in the two-note pattern; instead its duration would be translated into the nearest integer number of repetitions that could fit within that duration, with any slack taken up by adjusting the duration of the ending turn.

A mordent is almost identical to the pluck example given above: the speed of the mordent should remain invariant under duration transformations. It can be solved by renaming the duration parameter of the mordent and expressing the duration of the note following the ornament in terms of the ornament's duration.

While the solutions to these transformation problems from traditional music are interesting, even more interesting are the problems posed by practicing composers and the definition of "behaviors for sonic structures yet to be created. Kyma provides a sufficiently general representation to handle these "classic" problems grounded in the notion of scores and performers and yet also allow composers to expand the definition of music in terms of sound rather than written notation.

7. SUMMARY

Unexpected results of transformation in the new generation of computer music languages are usually the result of applying a transformation to an object that the composer thinks of as indivisible and but which is actually a composite of two or more objects. By fully specifying the transformable parameters of an object and the relationships between its parameters, the composer can encapsulate the composite object and its behavior under transformation as a single atomic object. The Kyma language provides a means for defining such encapsulations as well as flexible options for applying transformations to hierarchical structures. These tools make it possible to define a polyphonic transformations.

8. ACKNOWLEDGMENTS

Thanks to Kurt J. Hebel for the many, ongoing discussions of these issues during the development of Kyma.

9. REFERENCES


