A COMPOSER'S AMANUENSIS FOR THE WEB

Matthew Malsky
Clark University
Department of Visual and Performing Arts
950 Main St.
Worcester, MA 01610-1477
mmalsky@clarku.edu

Abstract:
This paper describes MIDI composition software that facilitates a transformative algorithmic method. After a brief discussion of the compositional model and objectives, these tools are examined in detail and their operations explicated by way of an extended musical example. The paper ends with a discussion of the Internet as a means to store the software's output, and to foster exchange among composers using the software.

1. Introductory Notes

This paper begins with a description of a collection of MIDI software tools that facilitate a generative algorithmic compositional method based on principles of musical variation or transformation (Rowe 1993, Winkler 1998). After a brief discussion of a compositional model and objectives, the two principle of these tools will be examined in detail: a data structure for an abstract representation of a theme or 'musical object' and a template for composing variations (or transformations) upon musical input. These tools are written in the graphical programming language Max, and are presented as an application within the frame of a Max 'patcher.' In the second section of this paper, the musical utility and operations of this method will be examined through a short musical example. Seven transformations (variations) on a musical object show the design of transformative functions and their musical results. This section will conclude with a discussion of means provided for the combination of these resulting transformations. In conjunction with a standard web browser, musical objects and transformations may be stored independent of the original Max 'patch,' as part of a database on an Internet-accessible server. The third section of the paper will describe the use of the Internet as a storage medium.

2. The compositional method

These tools will allow a composer to build new, consanguineous musical "objects" from existing ones. These objects, the building blocks from which a composition may be built, are the progeny of Pierre Schaeffer's *objets musicaux*, Varesian 'sound masses' and Ralph Shapey's 'graven images.' An object, as a self-identical form, has the potential to be altered (varied, embellished, dissected, juxtaposed, overlaid, enlarged, and fragmented) while still retaining its essential, ontological musical identity. These material building blocks are created through a cyclical process of statement and modification/variation, and are then combined into a larger form. Within the scope of operations of this algorithmic environment, musical objects are statements to be shaped and combined to meet the requirements of this larger context, the compositional design, which makes demands upon its component parts. Good fit, a composer's judgment in light of a "programmatic clarity" of a composition, is a necessary property. Form, or a composition's logical internal constitution, results from clarity in using a compositional language (Alexander 1964).

3. The data structure and transformation engine

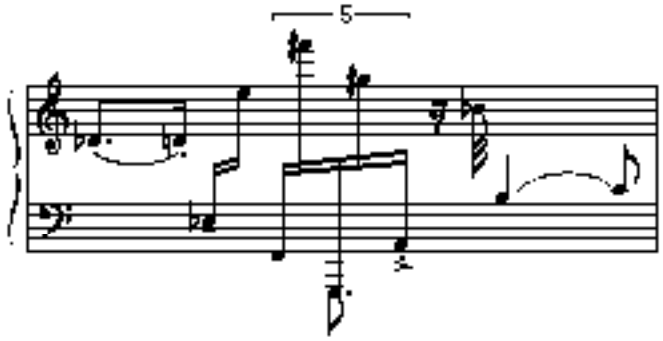
At the core of this composing method are two principle software tools. A data structure is an object which abstracts a MIDI representation of a musical object into a matrix of parameters for processing, collectively called transformation data. Second, a tranformation engine can be used to design and compute alterations to transformation data, and return a new object.

Through the data structure object, MIDI input (either in the form of a Standard MIDI file or a live performance) becomes array of events and relationships, the grist for the transformational mill. Transformation data describes the events in an object in terms of two classes of musical parameters, STATIC and DYNAMIC. There are five STATIC classes: PITCH, VELOCITY, GATETIME, RELEASE and POSITION. There are four DYNAMIC classes: ΔPITCH, ΔVELOCITY, ΔGATETIME, and ΔRELEASE. The STATIC parameters are concerned with the measurable musical characteristic of each given event of an object while DYNAMIC parameters measure aspects of the movement between successive events. Pitch and velocity are both notated in CSound's octave point pitch-class form (Vercoe 1993). Timing information is notated as beats plus parts of beats. Additionally, editable object constants include STEP SIZE, the resolution of the quarter note (default is 480), MIDI CHANNEL (which is initially set according to the original input), and TEMPO. INITIAL PITCH and INITIAL VELOCITY are used as starting points for DYNAMIC calculations.

In a single transformation, there are three levels of nested function definitions which are enacted sequentially. A first order transformation definition returns a chosen transformation data parameter altered by an arithmetic expression applied to this specified parameter. Additionally, at this level of transformation, the values for one parameter may be mapped onto another. For example, an object's ΔPITCH values be converted into VELOCITY information to create dynamic and timbral changes which are analogous to the motions of the object's interval progression. Second order functions are expressions which use the parameter POSITION as an index. The effect of second order functions increases (or decreases) as a function of the sequential position of events in an object. Finally, third order functions are expressions which can utilize multiple parameters simultaneously. By nesting the first and second order functions within this final superior level, a complex expression may be created. The syntax for third order functions are identical to first order expressions except that whenever there is a first or second order expression modifying a parameter defined in the third order expression, the result of the lower order function are evaluated first and the resultant number becomes the input for the higher order function. At each of these levels, a limiting BIAS may be applied to either the index or to the data itself.

Several subsidiary objects support the two discussed above, and round out the functions of the application. Objects are included to convert the data structure back to MIDI (and audible output), support graphical viewing and editing of the variations, create a 'score' to control timing for combining individual variations, and direct input from either a live performance or from a sequence stored as a Standard MIDI file.

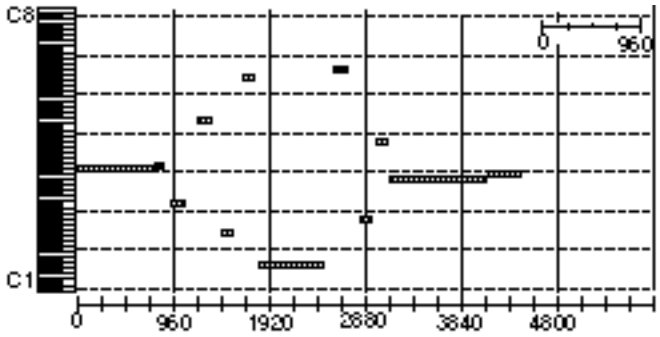
4. Operation by way of musical example



This object, taken from my composition, The *Ballad of the Strange Strand,* will serve here as the basis for creating a short musical example which will demonstrate this process of transformations. The musical events notated yield the following transformation data:

pos	pitch	vel	gate	rel	Δpitch	Δvel	Δgate	Δrel	RT
1	5.01	5.04	0.360						0/0
2	5.02	6.03	0.059	0	0.01	1.04	-0.301	0	0.360
3	4.03	5.04	0.107	0.060	0.01	-1.01	0.048	0.060	0.119
4	6.04	5.04	0.096	0.012	0.01	0.01	-0.011	-0.048	0.119
5	3.05	5.04	0.082	0.023	0.01	0	-0.014	0.011	0.119
6	7.06	5.04	0.082	0.014	0.01	0	0	-0.009	0.096
7	2.07	5.04	0.289	0.014	0.01	0	0.207	0	0.096
8	6.08	7.06	0.82	-0.194	0.01	2.0	-0.207	-0.208	0.096
9	4.09	5.04	0.047	0.014	0.01	-2.02	-0.035	0.208	0.096
10	5.1	5.04	0.051	0.169	0.01	-0.02	0.004	0.155	0.216
11	4.11	5.04	1.0	0.068	0.01	0	-1.051	-0.101	0.060
12	5.00	4.02	0.191	0	0.01	-1.0	-1.191	-0.068	1.0

The process of building this musical example began with experimentation. This object was used as the basis for creating, through trial and error, a large collection of new but associated objects. They share a common set of transformation data. Each variation is defined by its transformation definitions.

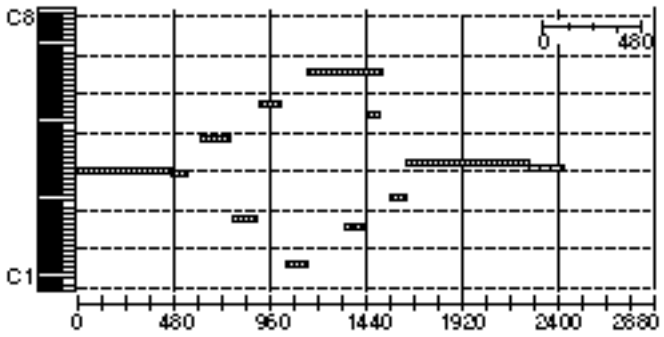


GT = 2 * GT

REL = $\sqrt{\text{REL} + 0.005}$

Object Two

Object$_2$ is a rhythmic augmentation of the original. Two first order functions alter the GATETIME, doubling the original GATETIME, and scaling the articulative space between notes (RELEASE) exponentially. The result is an object which is slightly longer than the original, more expansive with a little more space between notes. RUNNING TIME is recalculated so that the changes in GATETIME are reflected as melodic rather than harmonic alterations.



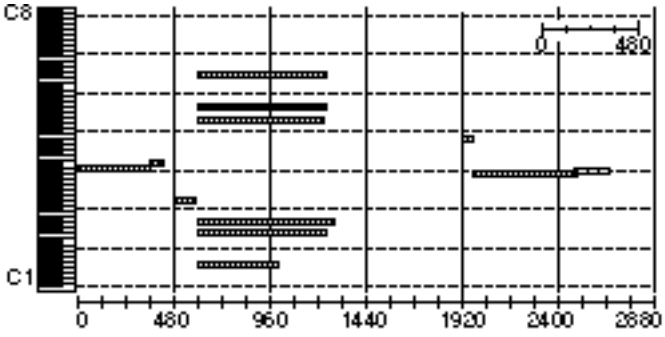
ΔPIT = ΔPIT * (-0.1)

GT = GT * (1.3)

REL = REL * (1.1)

Object Three

In Object$_3$, all interval directions are inverted (ΔPITCH) The object now consists of a descending chromatic scale beginning, again, on C#4. The octave displacements are preserved. In addition, GATETIME and RELEASE are augmented slightly, the first by 30%, the second by 10%. RUNNING TIME is recalculated.



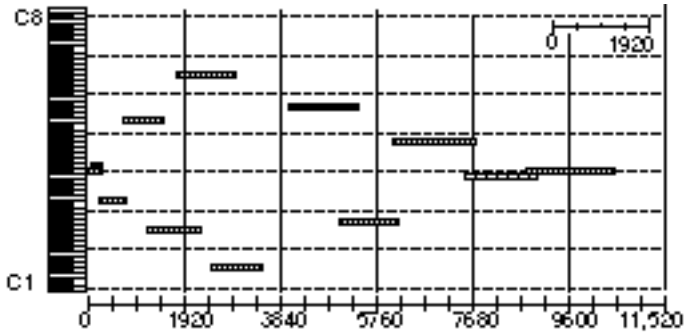
ΔGT * (-1)

ΔREL <-- (ΔGT * (-1))

BIAS limits this function to events 4 - 9

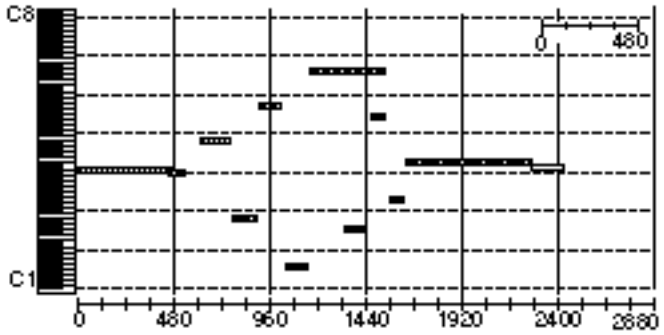
Object Four

A chord is created in the middle of Object$_4$ by inverting ΔGATETIME and mapping those returned values to ΔRELEASE for events 4 through 9. The range of application of these functions is determined through a BIAS applied to the ΔGATETIME and ΔRELEASE, limiting the effect of the transformation to only the middle of the object. When RUNNING TIME is recalculated using the new ΔGATETIME and old RUNNING TIME values for the ninth event, a slight gap, a silence is created immediately following the chord.



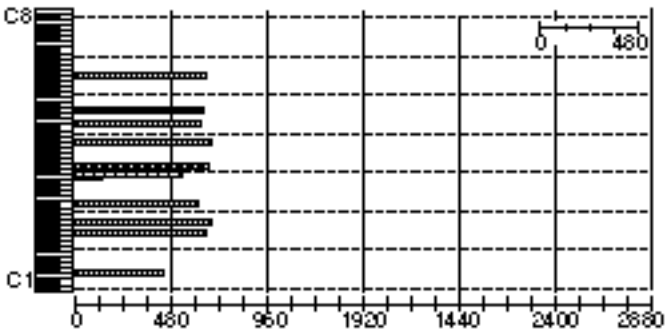
FIRST ORDER: ΔGT = ΔGT * (-1)
REL = ((-4) * REL) - 0.5
SECOND ORDER: ΔGT = ΔGT + (0.05 * POS)
Object Five

In Object$_5$, as in Object$_2$, the result of the transformations is a rhythmic augmentation of the original object. ΔGATETIME is altered in two steps. First, all of the ΔGATETIME values are inverted using a first order function; increases in GATETIME between contiguous events become decreases of equivalent size and vice versa. The rhythmic value of the first event (C#4) serves as the starting point for this transformation but remains unchanged. Additionally, ΔGATETIME is scaled using a second order function which, operating on the results of the first order transformation, arithmetically increasing the GATETIME of each successive event by multiples of 24 pulses. Working against the inherent rhythm of the original object, this increases rhythmic values over the course of the object. RELEASE is also scaled and inverted using a first order function. The result of switching the sign of each RELEASE value, the method of producing inversion, is to create overlaps between adjacent pitches but with no more than a dyad sounding at any given time.



FIRST ORDER: ΔPIT = ΔPIT * (-0.1)
GT = GT * (1.3) / REL = REL * (1.1)
SECOND ORDER: ΔVEL = ΔVEL + (-1 * POS)
Object Six

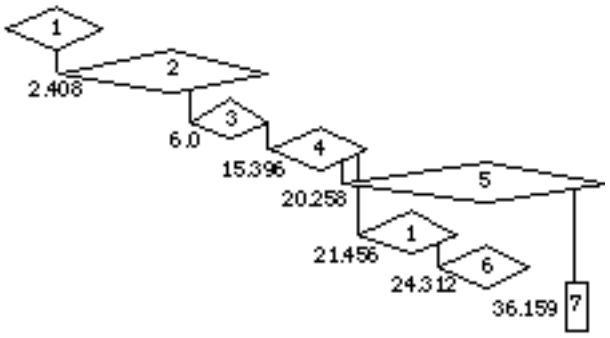
Object$_6$ is a variant similar to Object$_3$. In addition to the changes in the previous example, VELOCITY is transformed here. A second order function on ΔVELOCITY increases the change to VELOCITY between adjacent events over the course of the object, creating a crescendo effect.



ΔGT * (-1)
ΔREL <-- (ΔGT * (-1))
NO BIAS
Object Seven

Finally, Object$_7$ is a variant of Object$_4$. The only difference lies in the BIAS applied to the returned values of ΔGATETIME and ΔRELEASE. There is no BIAS and so the effect is to alter the RUNNING TIME of every event, creating a twelve note chord beginning at RUNNING TIME 0 whose voicing is determined by the octave displacements of the original object and whose length is set by the rhythmic value of its constituent notes.

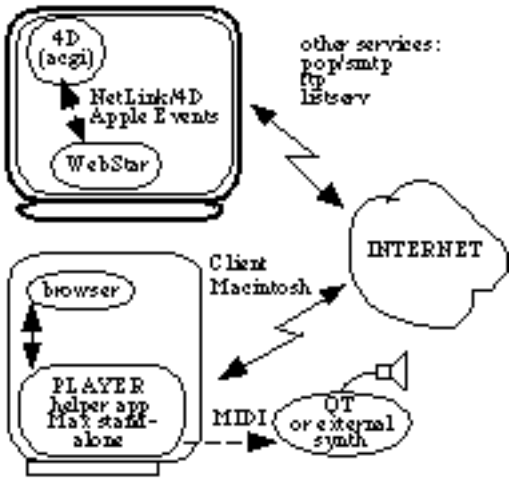
Now, to put it all together. The design of this musical example began once a larger vocabulary of objects had been constructed. The compositional intent was to strike a balance between two extremes: the repetitive statement of recognizable objects, both the initial and the related, derived objects, and the creation of a larger "ensemble" with a musical cohesiveness through a partial synthesis of these same objects. The following is a graphic representation of the examples discussed: the original object and the six variant forms arranged through time. Each object appears in this diagram as a diamond indicating, roughly, its registral shape and approximate length in relation to its fellow objects. This is how the components are organized and positioned.



Each transformation produces an editable sequence of its results. A score object provides a common clock for the playback of transformed objects. This musical example has been constructed principally with concern for the alignment of the entrances of each object. The original object begins the example. Object$_2$ begins with the attack of Object$_1$'s lowest note (G1) which has been designated as an anchor. Object$_3$ and Object$_2$ end together. Object$_4$ is concatenated with Object$_3$, following immediately after its last note. Object$_5$ and an exact repetition of Object$_1$ (the original) commence in rapid succession on the tenth and eleventh notes of Object$_4$ respectively. Dyads are created at these points of conjunction; Bb4-C#4 and B3-C#4. Object$_6$ begins on the tenth event of Object$_5$, creating a dyad Bb4 at the end of Object$_5$ with a C#4 which starts Object$_6$. Finally, the cluster of Object$_7$, accentuates the attack of the penultimate note of the musical example.

Although the entrances (and, in one case, the exit) of these objects are contrived to form a logical progression from one to the next, the musical surface of this example, as a continuous stream, is less exactingly formulated. When objects overlap, there is often a compound rhythm and harmonic content which was not specified in either of the component objects. These surface shapes can only be evaluated in terms of an original compositional notion of what this musical excerpt should be. The compositional combination of objects to form an interesting and coherent surface is integral to this compositional paradigm.

5. Internet application



{Internet server/client structure}

This project uses the Internet to distribute the client software, store the output of data structures and transformations, and foster interaction among composers using the software. In effect, the software described above serves as a helper application; the Internet site is a means for distribution and storage of its products.

Using any frame-compliant web browser, composers may (using the ftp functions of the web server and the client's web browser) upload data structures and transformations to this central storage facility. A relational database is the means for organizing these Max patches. Files created on individual composer's client computers may be cataloged by composer and composition (i.e. objects and transformations can be marked as part of a particular composition and belonging to a particular composer), and descriptive notes added. The database may be searched, and stored files downloaded for further work or study.

Citations:

Alexander, C. (1964). Notes on the synthesis of form. Cambridge, Harvard University Press.

Rowe, R. (1993). Interactive music systems: machine listening and composing. Cambridge, MIT Press.

Vercoe, B. (1993 [1986]). CSound: A Manual for the Audio Processing System and Supporting Programs. Cambridge, Media Lab, MIT.

Winkler, T. (1998). Composing Interactive Music. Cambridge, MIT Press.