MODULAR PROGRESSION PROGRAMMING IN OPENMUSIC

Matthew Lane
Université de Montréal
matthew.lane@umontreal.ca

ABSTRACT

This paper presents the author's modular approach to designing and using OpenMusic patches to produce multi-dimensional progressions in instrumental music. Musical passages (in one or several instruments) are passed from process to process with tailored parameters for each process and instrument, much like an assembly line working on a single base material. Individual progressions gradually change the passage in each instrument, altering combinations of pitch, time, structure, and stochastic elements, before passing the result to the next progression. The organizing structure is presented, as is the justification for this system and the guidelines for moving between these patches and composing by hand. Examples are presented in two compositions by the author (*Sliding Apart* and *Melodious Viscosity)* that made use of these patches.

1. INTRODUCTION

Computer-assisted composition (CAC) and specifically the use of OpenMusic,[1] is especially prevalent in my music for the development of progressions. This applies both to continuous progressions (one long line that continues to change), and sequences (progressions where the same idea is repeated while evolving continuously). Sequences are essentially changing repetitions, and have long been in use, but have until the 20th been mostly confined to diatonic (and sometime chromatic) modulations of the base element (model). In the 20th Century, especially with Messiaen, rhythm begins to change in a formal way as part of progressions.[2] Eventually, ever element of the music became a possible parameter for a progression, including pitch, rhythm, articulation, timbre, speed and all their subcategories.

My compositional process general falls into the following steps: creation of an original idea (seed), creation of related ideas, triage of ideas, development of remaining ideas, reinterpretation of developed ideas, formal organization, and linking. I have used CAC in nearly each of these processes, but the present focus will be on development. Developing different parts of a piece using the same patches, with the distinct traces and colours they leave on the music, helps to create unity between sections, and shapes the aesthetic of the piece.

The document will refer throughout to two pieces: *Melodious Viscosity*, a wind quintet written for the Brevà ensemble; and *Sliding Apart,* a quintet for flute, clarinet, piano, violin, and cello written for a reading by the Meitar Ensemble.

2. THEORY, PROGRAMMING, AND INTERPRETATION

2.1. Programming structure

Because progressions and sequences invariably play a role in every one of my pieces, to remain useful and portable, patches have to be flexible and rearrangeable, with sufficient parameters for control and variety, but with a simplicity to render them efficient. For this reason, modular patches are used, each one for a different treatment, and applied in sequence. In some cases, order of application made no difference, such as with a patch that progressively changes the pitch of all the notes and another that removes half the notes, but in other cases, order is crucial, such as with a patch that stretches time in places and another that randomizes the attacks based on temporal location.

Figure 1 shows the modular framework of this system. The music is passed from an original motive or note at the top to the complete altered progression at the bottom. At the beginning, a *Multi-Seq* class (A) contains an imported motive created in XML or MIDI. The *createbasepolypattern* patch (B) then multiplies this motive, creating and joining any number of copies of it, to later be modified and create a sequence. From here, the list of *chord-seq* classes is passed to *applytoseveralscoreobjects* (C), which can be attached to any lambda function (a function that can be passed as an

[1]For information on the background of OpenMusic, see "Computer-Assisted Composition at IRCAM: From PatchWork to OpenMusic" in the Computer Music Journal by Assayag et al. [1]

[2]When referring to rhythm changing, it is in a gradual and clearly directed way. It is true that rhythms in Bach, for example, occasionally change during sequences, but almost never in a consistent, directional, and quantifiable way. See *The Technique of My Musical Language* (Chapters III-IV by Olivier Messaien for more on Messaien's rythmic processes. [6]

argument to another function).[3] With the variant *applydifferentlytoscoreobejcts* patch (D), the parameters can be attached as a list of lists to the calling function, allowing differentiated values for each instrument or line. Any number of these functions can be applied to the lines. Finally, several closing functions are used, although none are needed. Two of the most relevant functions are:

mergegroups (E) - This function combines certain lines together as part of one staff. For example, in some cases, one instrument (like a piano) may actually be playing several lines at once – this allows them to be merged together into one staff.

assignchannels (F) – This function assigns each line its own MIDI channel, imperative for auditioning the result through a sequencer.

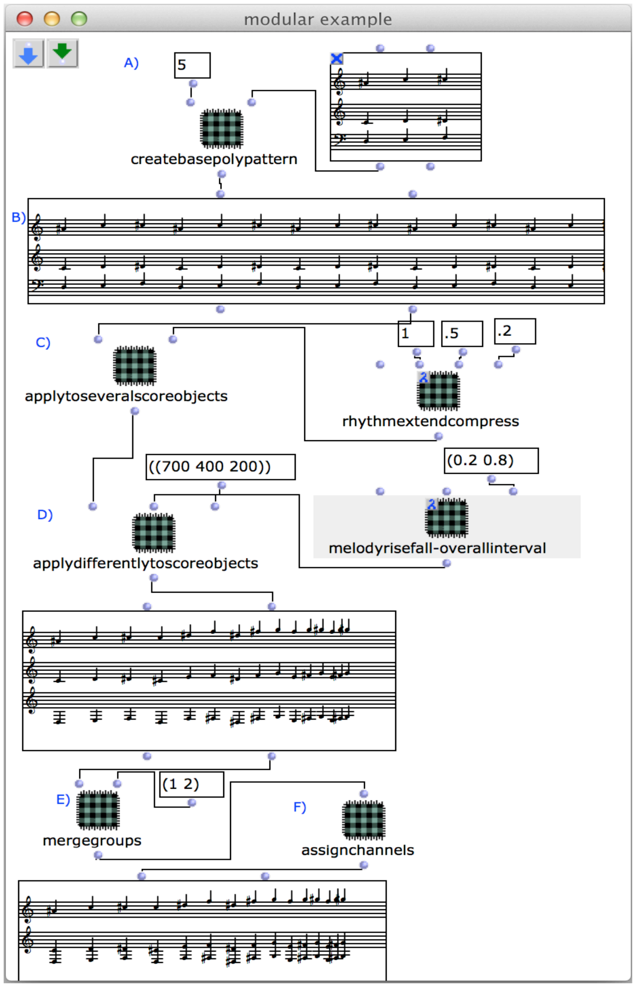

Figure 1: Example modular patch

2.2. Types of progressions

The types of progressions used can be grouped into several categories: pitch functions, time functions, and stochastic functions, although with a significant degree of crossover. Functions such as the gradual raising/lowering of pitch or the gradual expansion or compression of a melody's range are pitch functions. Time functions might include the gradual slowing down/speeding up of a passage, the application of a curve or mathematical function to the speed of a passage, or the gradual alteration of the length of a note. Stochastic functions, usually used in conjunction with pitch or time, would include a gradual increase/decrease of randomness in time or pitch, but might also include processes like the gradual but random removal, repetition, or freezing of notes.

2.3. Advantages of progression programming

There has proven to be several advantages to conceiving and programming progressions in this manner, especially with regards to formal unity throughout a piece. The primary advantage is the amount of material easily generated, and the possibility to hear quick approximate results through the changing of various parameters. For example, imagine a patch that gradually stretches the time (1), then gradually raises the pitch as a function of time (2), then a function that removes notes as a function of time (3), and finally a function that increases the range of the motive as a function of how many notes into the passage it is (4). While the result of this process is easily musically imaginable, the creation of such a passage, lasting perhaps thirty seconds at a quick tempo, could take an afternoon. Now imagine that the end result was not quite satisfactory, and one wanted the time to stretch slightly more over the passage (1). This process affects each of processes 2, 3, and 4 in sequence, and rewriting this by hand could take another afternoon, whereas verifying the workability of this alternative would take seconds in OpenMusic.

Stemming from this advantage of variability are several formal advantages for composing, including the possibility of efficiently creating variants of a passage. These can either be parametrically different passages, which may require only a slight shift in parameters, or stochastic variations requiring only a new evaluation of the same patches. Because of the nested nature of the patches, it is also possible to create progressions of progressions (to any number of layers), retaining a formal direction in each level of the nesting.[4]

A variant of this is the possibility to easily create progressions that go elsewhere from the original instance.

[3]For a better understanding of Lambda functions, see the online OpenMusic manual's chapter on higher-order functions. [2]

[4]This nesting (and possible recursive nesting in OpenMusic) can be used to generate many coherent musical structures, including and especially fractal structures. For more on this, see “Iterated Functions Systems Music” by M. Gogins. [4]

For example, the same progression could occur many times, with each transposing to a different tonal area, possibly over a different period of time.

2.4. A fork in the road: Further programming or interpretation of data into music

The output data usually provides a clear-cut progression with an obvious direction, but like some ideas developed using CAC, it sounds overly mechanical played raw. Here, the composer is left with a choice: interpret the data and begin to compose with it by hand, or develop further processes in OpenMusic to render data closer to the musical intention. It is crucial to understand where and why the data falls short of being effective music, whether for the purpose of improving the programming process or simply for the sake of efficiently pinpointing what needs fixing by hand. While this discussion could be the work of a small book, below are a few main areas of interest that have come up in these patches.

One of the principal difficulties relates to randomness. What we perceive as randomness and seek occasionally in our music is rarely true randomness, but rather even distribution with a degree of randomness. Four voices stacked on top of one another are occasionally bound to line up, creating a synchronicity that our ears perceive as ordered. In the same way, in a sequence of random note values, there will occasionally be several of the same in a row, creating an ordered impression where there is technically no order.[5] In many cases in OpenMusic, several evaluations are required before obtaining a result that sounds truly random. One possibility to explore in future patches is using constraints to control the amount of repetition of an element in a random sequence or to control/restrict the output of several elements that follow a pattern (for example, four random numbers that result in 2-4-6-8).[6] Another possible approach to this is avoiding randomness altogether, and instead creating sequences with a random sounding distribution, and apply Markov chains to produce variants, therefore lowering the probability of undesirable patterns. Currently, I have dealt with this issue in two steps. First, the patch is evaluated several times to find a result that sounds appealing. Secondly, the result is reworked by hand, altering rhythms or sequences by ear or by searching through the preceding material for patterns/notes/note-values that have not yet appeared.

A second issue is that progressions produced in OpenMusic generally imply an explicit direction and stick to it. Once a progression becomes too predictable, however, it loses some of its dramatic value. Even randomness, which adds a degree of instability, can itself become predictable. One approach to dealing with this is through interruptions in the progression. As of yet, these are most often added after leaving OpenMusic. Recently, however, I have begun exploring the possibility of incorporating them in patches. Randomly (or partially controlled random) placed freezes, pauses, or other processes, such as flashbacks to previous sections, occasional scrubbing backwards and forwards in the sequence, or elements as simple as sudden octave transpositions are all possible to do in OpenMusic. The formal layout for this approach will be a function that that, using several coefficients or parameters, determines segments or locations throughout the passage for these ideas to be applied, and then a separate *lambda* function that actually applies them.

The current process for composing these interruptions primarily involves listening. As soon as I clearly recognize the intention of a progression, it means there isn't long to change it before it loses its musical interest. From there, the question is whether the progression should be interrupted and continued again, interrupted and restarted, or outright stopped. When a progression is returning for the 2nd or 3rd time, the third option is often best. Next, the nature of the interruption comes into question. It may be a slightly altered continuation of the progression already in progress (for example, an upward scale that momentarily doubles in speed or becomes an upwards *glissando*). It may also be a sort of stop or break, such as a freeze or a silence. It may also refer to completely different material, although usually material that has already been introduced.[7] Finally, only once the first interruption is written, is it possible to move forward and determine where and if another interruption is needed. To be musically useful, this process *must* take into account the material and time that precedes it.

A third shortcoming is that the processes' directions are primarily linear, or at the very best, exponential. Both are easily predictable by the ear, and asides from interruptions, which tend to create a stop or a deviation from the progression, other approaches should be explored. The most practical, especially in the context of OpenMusic, would be to give the functions BPFs[8] instead of two-point line or curve segments to interpret. This would allow more freedom, and also the possibility of a more organic, hand-composed element before the final data output. This will be incorporated in future patches.

Regardless of how far the patches go, however, the data rarely amounts to *music.* My preferred solution to this is *reinterpretation*. The means are varied, but the principle is the same: become a musical performer and improvisor of one's own data. In some cases, this is playing through the passage on an instrument and recording or transcribing

[5]See reference [5]

[6]Several constraints programming systems exist for OpenMusic, notably OMClouds. For more on this approach, see C. Truchet et al. "OMClouds, a heuristic solver for musical constraints". [7]

[7]Note that these are the methods of dealing with CAC output data that I use the most in my aesthetic, but many others exist.

[8]BPFs are "breakpoint functions", a 2-dimensional graph with increasing X- (or time) values, and user-controlled Y-values. See the documentation for more information. [3]

what is played, especially the rhythm. This opens the possibility to move back and forth in time, add occasional notes, or alter harmonies where the ear suggests it. In cases with many lines undergoing processes simultaneously, I will take what strikes me as the leading line, reinterpret this, and then rebuild the other lines around this one.

Some computer-assisted tools for reinterpreting the passages are also useful, including scrubbing and transcribing the result in a notation program or sequencer, or simply tapping the tempo or chord changes and retranscribing the new rhythm. These are not, however, recipes to successful reinterpretation and require discretion to determine their utility.

Most importantly about the reinterpretation is the time and "space" required away from OpenMusic. Programming for CAC often involves laboriously going over the same passages many times for hours or days on end. Otherwise uncomfortable passages become familiar and comfortable sounding, and undesired rough edges become softened in our memory. This is why, for me, it's crucial to return to the output data several days after the evaluation, and ideally to return to it in a new context. The time allows me to rehear the musical ideas in a fresh way and to be surprised, a crucial element to my music. Either playing it on an instrument, in a sequencer, or in one's head can work, but returning to OpenMusic must be avoided. When using OpenMusic to review the material, the tendency is to resolve musical problems with OpenMusic and to think of the musical problems in terms of programming logistics, which while useful, do not necessarily represent how we hear music. If the progression as a whole does not work, returning to OpenMusic may be necessary, but many other musical problems arise out of attempting to compose to completion in OpenMusic, and thus the solution is often to relocate the musical data to a new medium. At least two or three of these reinterpretations are usually necessary, each separated by several days of memory-cleansing. For this reason, I tend more and more towards working on multiple parts of a piece at the same time, allowing me to always have something to which to dedicate my time (a principle of Toyota's *Lean production*).[9]

2.5. Conclusion

The issue surrounding CAC is consistently the question of how much *composing* the software does. At the present time, no data from any of these patches could be used as music in its raw form, but analyzing why this is helps us move closer to musical results, both in OpenMusic and by hand. More importantly, the more we can formalize solutions to the problems associated with musical data from OpenMusic, the better we are prepared to formalize solutions for issues in any of our music. It becomes irrelevant whether a composer creates a patch to rectify a problem passage or finally composes it by hand: once the patch is created, it means that the composer understands perfectly the cause and solution to the problem in small steps. Consequently, should a composer never even use a note of data taken from OpenMusic, the process of programming these progressions becomes inherently valuable. And while CAC programming may be pushed further to interpret our musical ideas, they remain *our* musical ideas, and not the computer's.

[9]See reference [8]

3. THE PATCHES AT WORK IN NEW COMPOSITIONS

3.1. Use and interpretation in *Sliding Apart*

CAC was implemented in nearly every section of this piece. Between sections of the piece, the main fast theme breaks down by dying and fading away in several dimensions. As soon as the motive begins to become obsessive, it breaks down rhythmically, slowing down at the same time, and putting the different lines out of sync. Certain notes go missing from the pattern, and the pitch slowly drops. Certain elements are fixed in this progression, while others have stochastic influences. The musical goal of this section is to create a sense of sudden falling apart in order to break away from a section that is otherwise extremely rigid and rhythmically precise.

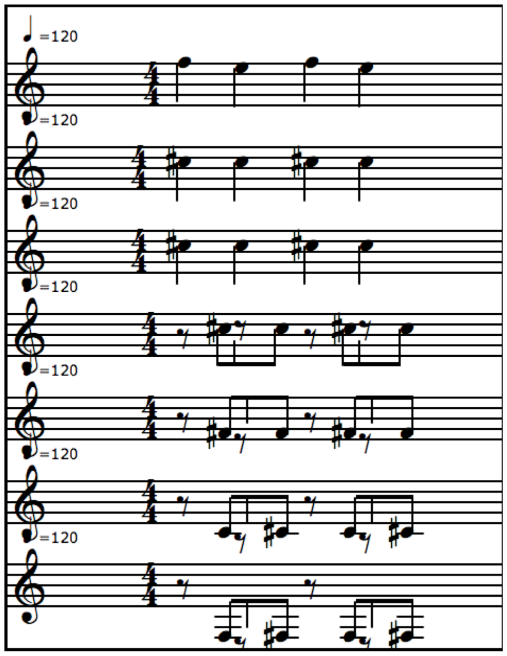

Figure 2: Base idea in OpenMusic

The progression takes place within one large patch.[10] There are a couple of things to note about the general structure of the patch before looking at the processes in detail. First, because the progressions are only functional on single-note lines (and not chords or other forms of harmony), the violin and cello double stops are broken up into two lines each.[11] Also, while every instrument is passed through every process, not every process affects every instrument: certain processes are simply told to have no effect on some of the instruments.

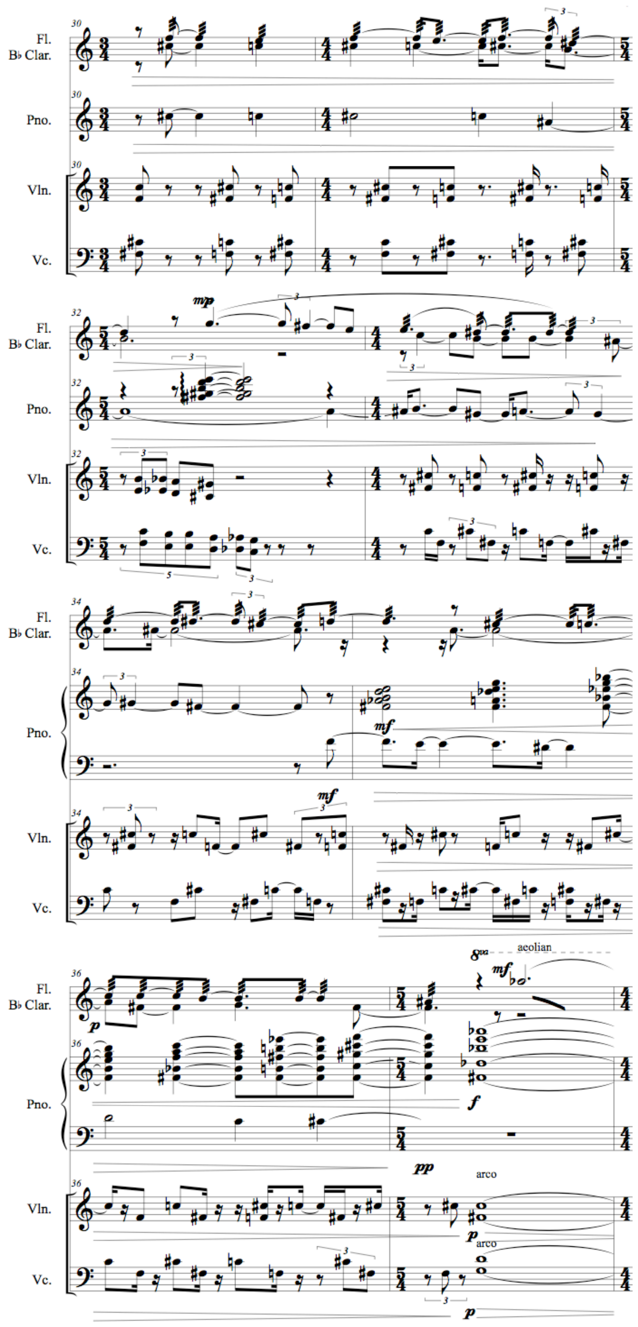

Figure 3: *Sliding Apart* score segment, showing the gradual deceleration and downward motion of the flute and clarinet lines, along with the increased randomness in both the violin and cello lines.

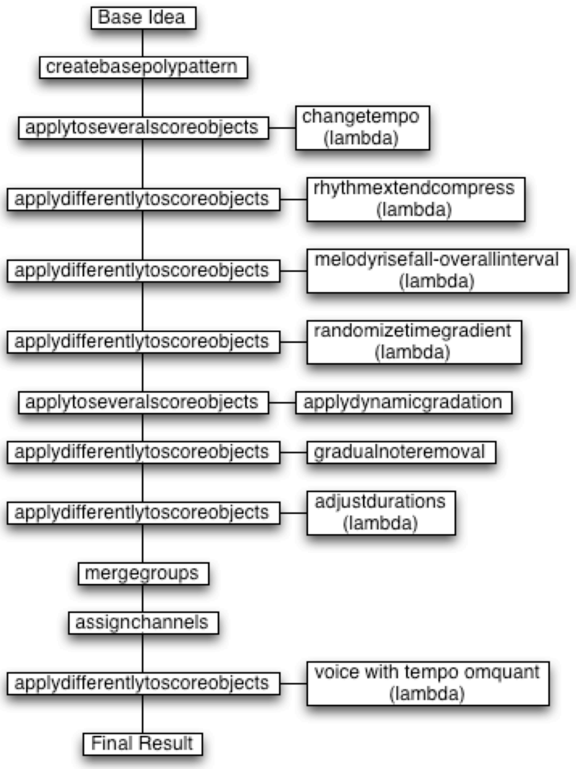
Base Idea
createbasepolypattern
applytoseveralscoreobjects
changetempo (lambda)
applydifferentlytoscoreobjects
rhythmextendcompress (lambda)
applydifferentlytoscoreobjects
melodyrisefall-overallinterval (lambda)
applydifferentlytoscoreobjects
randomizetimegradient (lambda)
applytoseveralscoreobjects
applydynamicgradation
applydifferentlytoscoreobjects
gradualnoteremoval
applydifferentlytoscoreobjects
adjustdurations (lambda)
mergegroups
assignchannels
applydifferentlytoscoreobjects
voice with tempo omquant (lambda)
Final Result

Figure 4: Processes as ordered in the original patch (top to bottom)

The first significant treatment, a rhythmic expansion (*rhythmextendcompress*), works in ratios: the initial ratio (1 for all the instruments) is attached to the lambda patch, while the destination ratios are attached to the *applydifferentlytoscoreobjects* patch. While this process doesn't apply to the strings at all (their ratio goes from 1 to 1), it does mean the flute becomes progressively slower, as do the clarinet and piano, though to lesser degrees. In conjunction with the next process (*melodyrisefall-overallinterval*), which lowers each line over a specific interval, the highest pitched instruments slow down and fall the most in pitch, contrary to our usual conception of inertia. Versions of this patch used in other parts of the piece, have an additional process after *rhythmextendcompress* to equalize the lengths of the different lines by padding them, so that time based processes could be applied similarly to all lines.

The stochastic processes are *randomizetimegradient* and *gradualnoteremoval*, and their interest lies not in being random, but in creating a gradient in randomness.

[10]A simplification of this patch is seen in Figure 4.

[11]The instrument order is flute, clarinet, piano, violin line 1, violin line 2, cello line 1, cello line 2. This is visible in Figure 2.

Randomizetimegradient takes four arguments: the amount of randomness in possible millisecond deviation at the beginning, the amount at the end, and the start and end points for the process. Here, for example, the process begins 10% of the way in and finishes half way through. Time in this function, does, however remain linear, meaning notes and their endings still retain the same order. The beginning and end times of the passage as a whole are also unchanged, ensuring that other processes can be used for the duration after this process is complete (the final 50%) without worrying that the notes have been altered. *Gradualnoteremoval* also contains a random element, removing more and more notes, but not necessarily the same ones on each evaluation, meaning different evaluations render different material from which to choose. Both these processes contribute to the sense of falling apart in the lines. Note that *randomizetimegradient* applies much more strongly to the strings (the final numbers), while *gradualnoteremoval* is not at all applied to the strings.

The interpretation of data for this piece involved multiple steps of reinterpretation. The major changes from the output data were the inclusion of an interruption and another process (also generated in OpenMusic) cross-fading with the ongoing progression. The interruption is a combination of two events: a freeze in the wind instruments and piano, while the strings continue the progression in a way by their downwards fall (Figure 5, m. 32). The cross-fading process, a rising, somewhat spectral piano chord progression, helps offset two problems in the first progression. It counteracts the loss of energy created by the falling apart progression, and it distracts from the increasing predictability of where the first progression is headed.

3.2. Use and interpretation in *Melodious Viscosity*

The wind quintet *Melodious Viscosity* also uses CAC in several places, primarily for progressions, but this repeating progression is the most salient example, and a relevant look at how many of the same processes from *Sliding Apart* can be used to different ends. This passage is building up towards a climax, and so many of the same processes as in *Sliding Apart* can be seen in reverse. There is also a process within a process, as the concatenated base segments of material also each undergo a process of de-randomization, hence the importance of the patch *CreateBasePatternWithMultipleEvaluations*, which allows the random element to work to its full potential, creating slightly varied instances before submitting them to the global chain of processes.

The most important process is *gradualNoteRemoval*, although this time in reverse, and again in conjunction with *adjustDurations* in order to fill out the space emptied by the removal of notes. Each instrument is given separate parameters for when to begin coming in (the bassoon undergoes no change in this process). There is also an acceleration treatment through the *rhythmextendcompress,* but in the end I deemed it simpler for the players to read an *accelerando* in the score, though the OpenMusic process helped me in gauging the efficiency of the process as a whole.

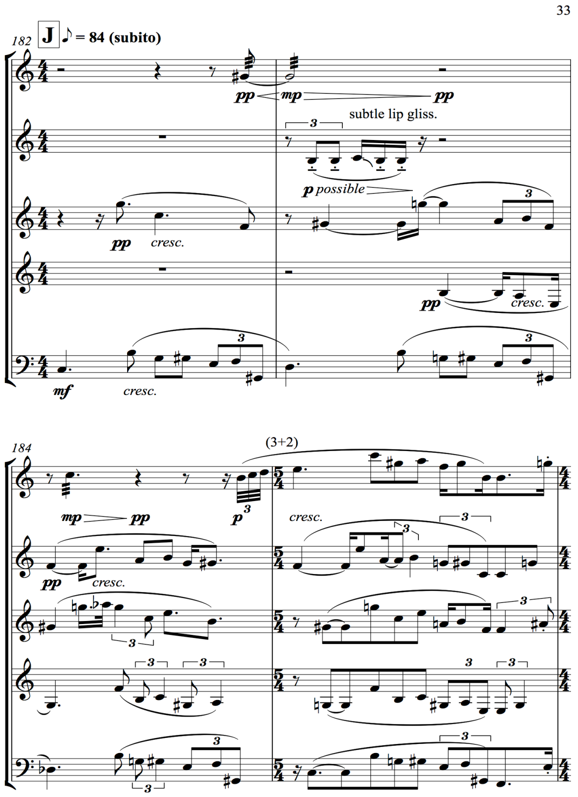

Figure 5: *Melodious Viscosity* (score segment)

A crucial part of the interpretation of results in this piece was many evaluations. After taking many evaluations, bars were chosen from different sets of results that filled certain auditory criteria, including a desire for a feeling of evenly distributed randomness, attack locations that didn't sound like errors, and interesting harmonic coincidences. Then a partial reinterpretation was done, primarily of what I perceived to be the leading lines (bassoon, but also often on the highest-pitch line present). The bassoon bass line's first notes of several measures were altered to make the passage feel less harmonically static. Finally, touches of colour were added: occasional harmonic contributions by instruments where the texture was too thin, and occasional performance techniques to momentarily draw attention away from the progression. This progression occurs in two section of the piece, and the CAC framework allowed the creation of two independent, but closely linked, progressions.

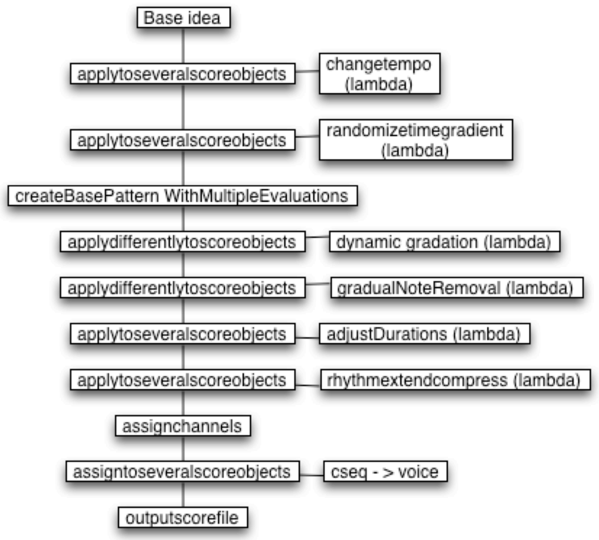
Base idea
applytoseveralscoreobjects
changetempo (lambda)
applytoseveralscoreobjects
randomizetimegradient (lambda)
createBasePattern WithMultipleEvaluations
applydifferentlytoscoreobjects
dynamic gradation (lambda)
applydifferentlytoscoreobjects
gradualNoteRemoval (lambda)
applytoseveralscoreobjects
adjustDurations (lambda)
applytoseveralscoreobjects
rhythmextendcompress (lambda)
assignchannels
assigntoseveralscoreobjects
cseq - > voice
outputscorefile

Figure 6: Patch layout in *Melodious Viscosity* (top to bottom)

3.3. Further Applications

While the complex applications for this modular system have not been exhaustively explored, some simple additional tools have drastically multiplied the contexts in which these patches are effective. These include, for example, a patch to constrain musical objects to a given mode, a patch to contract several instruments' parts to a common temporal length, and a patch to remove repeated notes. Other tools are still, however, being programmed for this library as new pieces require them, including pieces dealing with forms of melodic expansion, or with the gradual application of processes like frequency shifting. The primary goal for this library remains in the application of gradual processes.

Other resources in OpenMusic will hopefully allow an even more comprehensive use of layered processes like these. One such tool is a *BPF* (break-point-function), which allows not only a gradual application of a process, but a more intuitive, non-linear application – more closely able to respond to the composer's intuition. Another is the use of *Class-Arrays* paired with score objects to store information on extended techniques, making possible processes including extended techniques (ie. progressively playing more and more notes *pizzicato* rather than *arco*).

Although an infinite number of these progressions exists, the author's further research will most likely involve programming these progressions to allow their control in real-time – allowing the composer to guide the composition of the progression intuitively rather than through the continual alteration of typed variables. This will allow more impulsive gestures that will permit a true performance-composition of a piece, permitting the composer to maintain an uninterrupted musical thought process throughout the composition.

4. REFERENCES

[1] Assayag, G., Rueda, C., Laurson, M., Agon, C., and Delerue, O. «Computer-Assisted Composition at IRCAM: From PatchWork to OpenMusic», *Computer Music Journal*, MIT Press, vol. 23, n° 3, p. 59-72, 1999.

[2] Diatkine, C. «Higher-Order Programs and Functions», *OpenMusic Documentation*, Ircam – Centre Pompidou, Paris, 2011. «http://support.ircam.fr/docs/om/om6-manual/co/HighOrder.html»

[3] Diatkine, C. «2D Objects : BPF / BPC», *OpenMusic Documentation*, Ircam – Centre Pompidou, Paris, 2011. «http://support.ircam.fr/docs/om/om6-manual/co/BPF-BPC.html»

[4] Gogins, M. «Iterated Functions Systems Music», *Computer Music Journal*, MIT Press, vol. 15, n° 1, p. 40-48, 1991.

[5] Kahneman, D. «Misconceptions of Chance», *Thinking: Fast and Slow,* Doubleday, Canada, page 422, 2011.

[6] Messaien, O., *The Technique of My Musical Language.* J. Satterfield, translation. Alphonse Leduc, Paris, 1956.

[7] Truchet, C., Assayag, G., Codognet, P. "OMClouds, a heuristic solver for musical constraints" *MIC 2003: Metaheuristics International Conference*, Kyoto, Japan, 2003.

[8] Womack, J., Jones, D., and Roos, D.*The Machine That Changed the World: The Story of Lean Production - Toyota's Secret Weapon in the Global Car Wars That Is Now Revolutionizing World Industry,* Free Press, 2007.