Self Portrait (2002)

For Seven Musicians

Nigel Morgan

Self-Portrait (2002) – for Seven Musicians

One of the unique aspects of using a language-based computer environment for composing is the trace left by the code. This trace – here the SCOM scorefile - demonstrates how material is generated, processed and edited to produce the complete pitch, rhythm, dynamic and even orchestration elements of the music. The score file, in practice, rarely shows much of the actual decision-making that prefigures the written code; the false starts, discarded attempts, trial compilations are not included here. Nevertheless, there is much more evidence about how the piece was made than one can usually elicit from a staff-notated score.

Why should one bother presenting and annotating this information about a work's origination? There are several reasons: one, the composer finds it invaluable to have a concise record of how a piece was written in case a device or process needs to be reused in the future. There are some good examples of this in some of Nigel Morgan's scores to date. The code of the *Toccata* for piano was as the basis for 2 movements for double bass and piano in *Eurus*. In the song cycle *Stone and Flower* the code developed for the piano solo *Rising, Falling, Hovering . . .* is employed to create the keyboard material for central song in the cycle *London Revisited* albeit with a completely different tonality in operation. The *Six Concertos* (2003-5) all use the same code structure, but with radically different end results. The second reason concerns the potential such composing data has for making new and different versions of the same piece. This is not just the making of a new arrangement for different instruments from an original source, but the whole-scale reordering and reprocessing of the musical material to create a truly different score, still carrying the essence of the work, but containing new elements and ordering.

There are some very practical difficulties for composer and performer with progressing such an ambition. In making new and different versions of a piece accurate performing material does need to be produced afresh. The technology of the MIDI-based scorewriter and the ubiquitous MIDIfile do provide the foundation for achieving this. Symbolic Composer can produce wholly accurate midi data (analogous to 'clicking in' data on a scorewriter), which can be read immediately by a scorewriter. The proviso is that this data should not include articulations such as *staccato*, which cannot yet be interpreted consistently by all scorewriters.

At the present time an idea being pursued is to enable the performer to 'play' with possible changes to the structure and ordering of a work as part of a web application: to audition possible versions and then save the result as a portable MIDIfile capable of being read accurately in a scorewriter. *Self-Portrait (2002)* is the first score of this kind to offer to its performers such a web-based feature. At this stage the feature simply enables the 'blocks' to be arranged in any order (including repeats of blocks) and auditioned seamlessly.

At a later date it is hoped to be able to present more complex possibilities with a web interface that might talk directly to the original SCOM code, code that does produce all aspects and detail of the musical material required for a performance.

In essence *Self-Portrait* is about the composer's fascination with the potential of the whole-tone scale. Its ambiguity continues to intrigue and offer new possibilities, directions and connections. It is a tonality that provides the best foundation to explore ideas that surround the concept of 'open form', which *Self-Portrait* celebrates. With its two potential tonality 'poles', morphing the whole tone scale opens up an extraordinary world of original tonalities. *Self-Portrait* offers a rich gamut of scale forms, combinatory devices, and tonality progressions all generated, processed and organised algorithmically.

Block 1 explores gradually transforming one position of the whole-tone scale into the other, articulating this change with rhythm and orchestration.

Block 2 brings the two positions of the scale directly into 'play' with each other.

Block 3 returns to the idea of building new tonalities through transformation between the two scale positions, but also incorporates the two scales unaltered – as in Block 2. It also introduces a radical approach to timesheet control no longer governed by the single resolution: each beat/space on the timesheet can now have its own resolution, in effect a different time-signature.

Block 0 is all about creating new chordal harmonies by making all permutations of the whole-tone scales available through the MRAC function power-set. It also uses for the first time a method of orchestration of timbre based on generating timesheet strings for each instrumental part, a technique now at the heart of the *Six Concertos*.

```
;; Self Portrait - block 1 (shaping a phrase) - 7 part ensemble

; changing the length value of the first beat of a repeated sequence
; creating a transformation from one position of a whole-tone scale to
; another

;; functions

(defun length-expand (l-value l-list)
"expands number of lengths in a list according to a division by l-value"
(cond ((is-flat l-list)
(flatten (mapcar  (function (lambda (x)  (build-list l-value x)))
  (change-length :divide l-value l-list))))
(t (mapcar  (function (lambda (x)  (build-list l-value x)))
  (change-length :divide l-value (mapcar 'make-zone l-list))))))

; (length-expand '1/16 '(1/4 1/8 1/8 1/8))
; > (1/16 1/16 1/16 1/16 1/16 1/16 1/16 1/16 1/16 1/16)

(defun select-length-to-string (length lis-of-lengths)
"creates a string pattern based on selected length value"
   (let ((out ""))
     (dolist (x lis-of-lengths)
       (if (equal x length)
         (setq out (str-cat out "-"))
         (setq out (str-cat out " "))))
     out))

; (select-length-to-string '3/8 '(3/8 3/8 3/8 5/16 1/2 9/16 5/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 1/2 3/8 7/8))
; (---    --------- - )



(setq mat (p-select (list-a-scale 0 7 :scr 2 ) (g-symbol 'a 'm))
; (a c e g i k m)
      matx. (p-select (list-a-scale 1 6 :scr 2) (g-symbol 'a 'm))
; (b d f h j l)
      pat (cf-noise-white 15 1.0 .37 mat)
; (e m a c g e a k c k a g k c a)
      patx (cf-noise-white 15 1.0 .37 matx))
; (f l b d h f b j d j b h j d b)
```

Here the two positions of the whole-tone scale have been generated as symbols-lists in a chromatic tonality

The symbol-lists are now varied and extended using a white-noise generator

```
(setq paty
(symbol-transform
   from pat
   to patx
   order (symbol-shuffle (g-integer 0 14) 0.1)
   changes '(3)
   repeats '(1)
))

; (e m b c g e a k d k a g k d a f m b d g e a j d k a . . . .

(setq patz
(symbol-transform
   from pat
   to patx
   order  (symbol-shuffle (g-integer 0 14) 0.12)
   changes '(3)
   repeats '(1)
))

; (e m a d g e a k c k b h k c a e m a d g e a j c k b . . .

(setq control-value '24)

(setq len
(gen-collect 0.59 control-value :list
'(append (list (pick-random '(1/8 1/8 1/8 1/8 1/8 1/4 1/4 1/4. 1/16)))
        (pick-random (append (gen-repeat 12 '((1/8 1/8)))
                '((1/8 1/8 1/8)) '((1/8 1/8 1/8 1/8))
                                '((1/8 1/16 1/8 1/8 1/8)))))))

; > ((1/8 1/8 1/8) (1/8 1/8 1/8) (1/8 1/8 1/8) (1/16 1/8 1/8) . . . .


(setq dur
(gen-collect 0.59 control-value :list
'(append (list (pick-random '(1/8 1/8 1/8 1/8 1/8 1/4 1/4 1/4. 1/16)))
        (pick-random (append (gen-repeat 12 '((1/16 1/16)))
                '((1/16 1/16 1/16)) '((1/16 1/16 1/16 1/16))
                '((1/16 1/32 1/16 1/16 1/16)))))))

(setq ls (mapcar 'length len))
; > (3 3 3 3 3 5 3 3 3 3 3 3 3 3 3 3 3 5 3 3 3 3 6)
```

symbol-transform is a powerful SCOM primitive that enables precise control over the transformation of one symbol-list into another. Here the white-noise varied list of one whole-tone scale is morphed into the other with three changes happening per transform cycle.

These lists control the articulation of the note-length lists

```
(setq len-m
(do-section :all
 '(list (car x) (add-up (cdr x))) len))
; > ((1/8 1/4) (1/8 1/4) (1/8 1/4) (1/16 1/4) . . .
```

This adds a further layer of note-length values

```
; either - picks randomly from 'pat'

(setq sym
      (gen-process '(pick-rnd nil :content x y) ls pat :list))

; ((a k c) (g k e) (c e c) (c a c) . . .

; or - selects sequentially from 'pat'

(setq sym-1
      (gen-process '(pick-random (symbol-divide x nil nil y)) ls pat :list)
      sym-2
      (symbol-divide ls nil nil patz)
      sym-3
      (symbol-divide ls nil nil paty))

; ((e m a) (e m a) (k c a) (k a g)
```

The symbol-patterns are now being broken up to fit the note-length lists. Note the two different approaches here.

```
(setq b-sym-1 (do-section (gen-template 0.97 4 1(length sym-1))
                          '(find-beat x) sym-1)
      b-sym-2 (do-section (gen-template 0.971 4 1(length sym-2))
                          '(find-beat x) sym-2)
      m-sym-1 (do-section :all ' (strip-singles x)
                          (do-section :all '(e-insert '= 0 x)
                          (do-section :all '(g-chord nil 2 3 0 0 x) sym)))
      m-sym-2 (do-section :all ' (strip-singles x)
                          (do-section :all '(e-insert '= 0 x)
                          (do-section :all '(g-chord nil 2 3 0 0 x) sym-3)))
)

(setq rh-sym (append (mapcar 'find-anacrusis sym) (mapcar 'find-anacrusis sym-3))
      lh-sym (append sym-1 sym-2)
)
```

Bass and chordal material is generated here to produce this kind of output:

((= ac) (= e e) (= kc) (= akc) (= ck g) (= kc ac)

```
(setq dyn
(gen-collect 0.59 control-value :list
'(append (list (pick-random '(80 80 80 80 80 96 110 120)))
       (pick-random (append (gen-repeat 12 '((55 45)))
                '((55 45 45)) '((45 45 45 45))
'((45 65 45 45 45)))))))

(setq dyn2
(gen-collect 0.59 control-value :list
'(append (list (pick-random
                  (change-length :sub 15 '(80 80 80 80 80 96 110 120))))
       (pick-random (append (gen-repeat 12 '((65 55)))
                '((65 55 55)) '((55 55 55 55))
'((55 75 55 55 55)))))))

(setq dyn1
(gen-collect 0.59 control-value :list
'(append (list (pick-random
                (change-length :sub 35 '(80 80 80 80 80 96 110 120))))
      (pick-random (append (gen-repeat 12 '((45 50)))
                '((40 45 50 55 60 65)) '((40 45 50 55 60 65 70 75))
'((45 45 65 45 50 55 60 65 70 75)))))))


(setq zones
'(3/8 3/8 3/8 5/16 1/2 9/16 5/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 1/2 3/8 7/8 1/2 3/8
3/8 5/16 11/16 3/8 3/8 3/8 5/16 1/2 9/16 5/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 1/2
3/8 7/8 1/2 3/8 3/8 5/16 11/16))

(setq wd/br-s (select-length-to-string '3/8 zones)
      pc/st-s (invert-string wd/br-s))

; either 'unedited'

;wd "---    --------- -  --  ---    --------- -  --  "
;br "---    --------- -  --  ---    --------- -  --  "
;pc "   ----         - --  --   ----         - --  --"
;st "   ----         - --  --   ----         - --  --"


(setq wd/br-z (zone-convertor zones wd/br-s)
      pc/st-z (zone-convertor zones pc/st-s))
; (3/8 3/8 3/8 -5/16 -1/2 -9/16 . . .
```

This section of the code shows how a timesheet graphic of the movement has been arrived at. The unique feature here is that the timesheet's resolution is constantly changing in line with the sequence of zones. The function zone-convertor provides the essential 'swallowing' mechanism to enable parts to 'line-up' correctly when spaces occur in the beat/space string.

```
; (-3/8 -3/8 -3/8 5/16 1/2 9/16  . . .

; or 'edited'

;wd "---    --------- -  -------    --------- -  ----"
;br "---    --------- -  -------    --------- -  ----"
;pc "   ----         ----  --   ----         - ------"
;st "   ----         ----  --   ----         - ------"

(setq pc/st-ze (zone-convertor zones   "   ----         ----  --   ----         - ------" )
      wd/br-ze (zone-convertor zones   "---    --------- -  -------    --------- -  ----"))

(setq pc/st-anacrusis
       (find-beat (string-to-symbol 'x "   ----         ----  --   ----         - ------" )))

;; score

(def-tonality
wd (activate-tonality (chromatic e 5))
br (activate-tonality (chromatic e 3))
pc (activate-tonality (chromatic e 5))
st (activate-tonality (chromatic e 3))
ml (activate-tonality (chromatic e 5))
rh (activate-tonality (chromatic e 5))
lh (activate-tonality (chromatic e 4))
bs (activate-tonality (chromatic e 2))
)


(def-symbol
wd (do-section :all '(p-replace nil 'first '= x) rh-sym)
br (do-section :all '(p-replace nil 'first '= x) lh-sym)
pc (do-section pc/st-anacrusis '(p-replace nil 'first '= x)
               (do-section :all '(gen-variants-tfc nil 4 4 x) rh-sym))
st (do-section pc/st-anacrusis '(p-replace nil 'first '= x)
               (do-section :all '(gen-variants-tfc nil 4 4 x) lh-sym))
ml (append m-sym-1 m-sym-2)
rh rh-sym
lh lh-sym
bs (append b-sym-1 b-sym-2)
)
```

```
(def-length
wd (append len len)
br (append len len)
pc (length-expand '1/16 len)
st (length-expand '1/16 len)
ml (append len-m len-m)
default (append len len)
)

(def-duration
pc (length-expand '1/16 len)
st '(1/32)
ml (append len-m len-m)
default (append dur dur)
)

(def-zone
wd wd/br-ze
br wd/br-ze
pc pc/st-ze
st pc/st-ze
default (make-zone-list (symbol-of rh) (length-of default))
)

(def-velocity
pc (append dyn1 dyn2)
st (append dyn1 dyn2)
default (append dyn dyn)
)

(def-channel
wd 4
br 5
pc 6
st 7
ml 2
rh 1
lh 1
bs 3
)

(def-program gm-sound-set
wd soprano-sax
br trombone
```

Note the use of the bespoke function length-expand developed specifically for this movement.

```
pc marimba
st cello
ml vibraphone
default electric-piano-2
bs acoustic-bass
)

(def-controller gm-controllers
  (wd  panning  '((20)))
  (br   panning  '((50)))
  (pc   panning  '((80)))
  (st   panning  '((120)))
(ml panning  '((120)))
(default panning  '((64)))
(bs panning  '((10)))

)
(def-tempo 70)

(compile-instrument-p "ccl;output:" "Block1"
wd
br
pc
st
ml
rh
lh
bs
)



#|

; t-sigs

(setq zones
'(3/8 3/8 3/8 5/16 1/2 9/16 5/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 1/2 3/8 7/8 1/2 3/8
3/8 5/16 11/16 3/8 3/8 3/8 5/16 1/2 9/16 5/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 1/2
3/8 7/8 1/2 3/8 3/8 5/16 11/16))


|#
```

```
;; Self-Portrait - Block 2 - 7 part version

;; building a block from a rhythmic sketch. Using melodic phrases that
;;  move between 2 positions of the wholetone scale.


(setq mat (p-select (list-a-scale 0 7 :scr 2 ) (g-symbol 'a 'm))
; > (e m a c g e a k c k a g k c a)
      matx (p-select (list-a-scale 1 6 :scr 2) (g-symbol 'a 'm))
; > (f l b d h f b j d j b h j d b)
      pat (cf-noise-white 15 1.0 .37 mat)
      patx (cf-noise-white 15 1.0 .37 matx))


(setq pat-c (append (find-unique pat)(reverse (find-unique patx))))
; (e m a c g k j h d b l f) ; moving between 2 positions of wt scale

;;  2/4 3/4 7/8 2/8 5/8 3/4 5/8 3/4 5/4 3/2 3/2 3/2 15/16 - pre-composed outline
;; for use with any timesheet orchestration


(setq r-1 (qlength '(8-01 8-1111 8-1111 16-112)) ; keyboard rhythms
; contains anacrusis within previous section
      r-2 (qlength '(8-01 16-111111 16-112))
      r-3 (qlength '(8-011 8-11111 8-33 16-33 4-1))
      r-4 (qlength '(16-00011100 16-0011 8-113131))
      r-4i (qlength '(8-011111 8-113131))
      r-5 (qlength '(16-11001100110011001100))
      r-6 (qlength '(16-11001100110011000011))
      r-7 (qlength '(16-011111111111111)))

(setq r-b1 (qlength '(8-000002222222)) ; bass rhythms
      r-b2 (qlength '(8-111 8-11111 8-33 16-33 4-1))
      r-b3 (qlength '(8-0000000 16-0011 8-2221)) ; '(8-00000222221))
      r-b4 (qlength '(16-0 8-1111111)))

(setq r-m1 (qlength '(8-0000 16-011 16-111111 16-111111111)) ; mallet rhythms
      r-m2 (qlength '(8-011110))
      r-m3 (qlength '(16-111111112 8-00 16-11 8-33 16-33 4-1))
      r-m4 (qlength '(8-0000 16-011 16-111111 16-111111111110111))
)
```

See the image from the composer's note-book at the end of this text to view one of the rhythmic sketches from which the note-length code was developed.

```
(setq pat-cx (find-change (sort< pat))
      pat-cy (fill-rest pat-cx (symbol-compress -80 patx))
      pat-cz (symbol-mix pat-cx pat-cy))

(setq s-1 pat-c ; keyboard melodic phrases
      ; (e m a c g k j h d b l f)
      s-2 (reverse pat-c)
      ; (f l b d h j k g c a m e)
     s-3 (symbol-scroll -3 pat-c)
     ; (c g k j h d b l f e m a)
      s-4 (reverse s-3)
      s-5 (symbol-scroll -1 pat-c)
      s-6  (shuffle-pairs (symbol-scroll 1  pat-c) 0.147)
      ; (k j f e c g m a b l h d)
      s-7 (shuffle-pairs (symbol-scroll 1  pat-c) 0.14)
      ; (c g k j f e h d b l m a)
      s-8 (p-replace nil 'first '= pat-cz)
)


(setq s-b1 pat-c
      s-b2 s-3
      s-b3 s-4
      s-b4 s-5
      s-b5 s-6
      s-b6 s-7
      s-b7 (symbol-trim-r 7 (cdr pat-cz))
)

(setq s-m1 (find-change (shuffle-pairs (flatten
               (symbol-divide '(3 (-1)) nil nil pat-c)) 0.259))
      ; (e m h d j = f d b a = g k l = c g)
      s-m2 (p-remove '(1 3) (symbol-bundle 2 pat-c))
      s-m3 (append s-2 (symbol-trim-r 7 s-3))
      s-m4 (find-change (shuffle-pairs (flatten
             (symbol-divide '(3 (-1)) nil nil pat-c)) 0.6))

)

(setq zones '(3/2 7/8 19/8 2/1 3/2 3/2 3/2 15/16)
      zonex '(13/8 3/4 19/8 2/1 3/2 3/2 3/2 15/16))
```

The function shuffle-pairs was developed in the composition Eurus. See the annotations for that score to read the full code. Here its use results in a kind of appoggiatura effect.

```
(setq wd-ze    (zone-convertor zonex  "-- -- --")
      br-ze    (zone-convertor zones  "-- -- --")
      pc-ze    (zone-convertor zones  " --- ---")
      st-ze    (zone-convertor zones  "  -- ---"))

(setq s-6x (flatten (do-section '(= x) '(symbol-transpose (get-random 0 8) x)
                                       (mapcar 'list s-6)))
      s-6y (flatten (do-section '(= x) '(symbol-transpose (get-random 8 0) x)
                                       (mapcar 'list s-6)))
      s-5x (flatten (do-section '(= x) '(symbol-transpose (get-random -8 8) x)
                                       (mapcar 'list (symbol-scroll 1 s-5))))
)



;; score

(def-tonality
wd (activate-tonality (chromatic e 5))
br (activate-tonality (chromatic e 3))
pc (activate-tonality (chromatic e 5))
st (activate-tonality (chromatic e 4))
ml (activate-tonality (chromatic e 5))
rh (activate-tonality (chromatic e 5))
lh (activate-tonality (chromatic e 4))
bs (activate-tonality (chromatic e 2))
)


(def-symbol
wd (list s-m1 (symbol-demix 2 s-m2 :sort) s-m3 s-m4 s-5
         (symbol-scroll -1 s-6) (symbol-scroll -2 s-7) s-b7)
br (list s-1 s-2 s-3 s-b4
           s-5x s-5 (symbol-scroll 2 s-b6) s-b7)
pc (list s-1 s-2 s-m3  s-4 s-5 s-6y s-7 s-8)
st (list s-1 s-2 s-3 s-4 s-5 s-6x s-7
         (symbol-inversion 'g s-8))
ml (list s-m1 s-m2 s-m3 s-m4 s-5
         (symbol-scroll -1 s-6) (symbol-scroll -2 s-7) s-b7)
rh (list s-1 s-2 s-3 s-4 s-5 s-6 s-7 s-8)
lh (list s-1 s-2 s-3 s-4 s-5 s-6 s-7 (symbol-inversion 'g s-8))
bs (list s-b1 s-b2 s-b3 s-b4 s-5
           (symbol-scroll 1 s-b5) (symbol-scroll 2 s-b6) s-b7)
)
```

The function symbol-demix enables a chordal part to be 'demixed' into individual voices.

```
(def-length
wd (list r-m1 r-m2 r-m3 r-m4 r-5 (symbol-scroll -1 r-5) r-6 r-b4)
br (list r-1 r-2 r-3 r-b3 r-5 r-5 r-6 r-b4)
pc (list r-1 r-2 r-m3 r-4 r-5 r-5 r-6 r-7)
st (list r-1 r-2 r-3 r-4i r-5 r-5 r-6 r-7)
ml (list r-m1 r-m2 r-m3 r-m4 r-5 (symbol-scroll -1 r-5) r-6 r-b4)
rh (list r-1 r-2 r-3 r-4 r-5 r-5 r-6 r-7)
lh (list r-1 r-2 r-3 r-4i r-5 r-5 r-6 r-7)
bs (list r-b1 r-b2 r-b3 r-5 (symbol-scroll 1 r-5) r-6 r-b4)
)

(def-zone
wd  wd-ze
br  br-ze
pc  pc-ze
st  st-ze
ml (make-zone-list (symbol-of ml) (length-of ml))
bs (make-zone-list (symbol-of bs) (length-of bs))
default (make-zone-list (symbol-of rh) (length-of rh))
; (13/8 3/4 19/8 2/1 3/2 3/2 3/2 15/16)
)

(def-velocity
default '(70)
)

(def-channel
wd 4
br 5
pc 6
st 7
ml 2
rh 1
lh 1
bs 3
)

(def-program gm-sound-set
wd soprano-sax
br trombone
pc marimba
st cello
ml vibraphone
rh electric-piano-2
```

```
lh  electric-piano-2
bs acoustic-bass
)

(def-tempo 70)

(compile-instrument-p "ccl;output:" "block2"
wd
br
pc
st
ml
rh
lh
bs
)
```

```
;; Block 3 - final (version for 7)

;; creating a block using a timesheet to control tonality change: whole-tone1, wholetone2,
;; gradual transformation from wt1 to wt2

;; functions

#|
(defun calculate-positions (lis)
"calculates positions of positive integers in a list of attack values"
(if
(cond ((symbolp (car lis))
       (e-position 'x lis))
      (t (e-position 'x
(string-to-symbol 'x (integer-to-string-1 lis)))))))

(defun p-replace-sections (lis sym-source sym-target)
"replaces sections of a target list with sections from a source list"
(p-replace nil (calculate-positions lis)
           (p-select (calculate-positions lis) sym-source) sym-target))


(calculate-positions '(= = = x x x x = = = x = = x x = x = x = = x))
; (3 4 5 6 10 13 14 16 18 20 23)

(calculate-positions '(0 0 0 4 4 3 3 0 0 0 2 0 0 3 3 0 3 0 3 0 2 0 0 3))
; (3 4 5 6 10 13 14 16 18 20 23)

(p-replace-sections '(x = x = x) '((a b) (c d) (e f) (g h)(i j)) '((m n)(o p)(q r) (s t) (u v)))
; ((a b) (o p) (e f) (s t) (i j))

|#

(setq output-2
 (gen-collect 0.5 40 :list
'(pick-random
  '((wt1)(wt2)(wt1)(wt2)(wt1)(wt2)(wt1)(wtm)(wtm)(wt1 wt2) (wt1 wt2)))))

(setq wt1-v (instrument-to-string 'wt1 output-2))

(setq wt2-v (instrument-to-string 'wt2 output-2))

(setq wtm-v (instrument-to-string 'wtm output-2))
```

The functions calculate-position and p-replace-section were developed for this movement and have become an essential device used in the composition of the Six Concertos (2003-5)

This is the process that you can find used at the beginning of each of the Six Concertos to derive an algorithmically-controlled play of timbres. The function instrument-to-string enables a timesheet string to be created showing when an instrument plays or is silent.

```
#|

(see above)

; template for 'a play of tonalities'
; wt1 / wt2 - 2 positions of wholetone scale
; wtm - transformation from wt1 to wt2

;; timesheet

wt1-v "  --- --  -- -- - - ----------   - - - "

wt2-v "- ----   -  -    -     -  -   -  - --  "

wtm-v " -      -      -   -           -- -   -"


wt1-a (0 0  5 4 4 0 3 3 0  0 2 3 0 3 3 0  3 0 3  0 2 3 3 3 4 3 3 5 3 3 3 0  0  0 5  0 5 0 2 0)

wt2-a (3 0  5 4 4 3 0 0 0  2 0 0 3 0 0 0  0 3 0  0 0 0 0 0 4 0 0 5 0 0 0 3  0  0 5  0 5 3 0 0)

wtm-a (0 13 0 0 0 0 0 0 13 0 0 0 0 0 0 13 0 0 0 12 0 0 0 0 0 0 0 0 0 0 0 0 0 12 13 0 13 0 0 0 13)

(see below)
 |#

(init-rnd 0.1)

(def-neuron attacks
(in 1 '((wt1))) (eval (pick-random '(3 3 3 2 2 4)))
(in 1 '((wt2))) (eval (pick-random '(3 3 3 2 2 4)))
(in 1 '((wtm)))(eval (get-random 5 15))
(in 1 '((wt1 wt2))) (eval (pick-random '(3 3 3 2 2 4 5)))
(otherwise (in 1 0)))

(setq attack-nos  (run-neuron 'attacks output-2))
; (3 13 5 4 4 3 3 3 13 2 2 3 3 3 3 13 3 3 3 12 2 3 3 3 4 3 3 5 3 3 3 3 12 13 5 13 5 3 2 13)

(setq wt1-a
      (e-substitute '(0) '(=)
                    (symbol-mask  attack-nos
                                  (string-to-symbol 'x wt1-v))))
; (0 0 5 4 4 0 3 3 0 0 2 3 0 3 3 0 3 0 3 0 2 3 3 3 4 3 3 5 3 3 3 0 0 0 5 0 5 0 2 0)
```

What is happening here is that each beat show on the timesheet graphic is being filled with a randomly-picked value creating the number of attacks present in that 'beat' (read zone of course - these numbers will eventually turn into time signatures).

```
(setq wt2-a
      (e-substitute '(0) '(=)
                    (symbol-mask  attack-nos
                                  (string-to-symbol 'x wt2-v))))
; (3 0 5 4 4 3 0 0 0 2 0 0 3 0 0 0 0 3 0 0 0 0 0 0 4 0 0 5 0 0 0 3 0 0 5 0 5 3 0 0)

(setq wtm-a
      (e-substitute '(0) '(=)
                    (symbol-mask  attack-nos
                                  (string-to-symbol 'x wtm-v))))
; (0 13 0 0 0 0 0 0 13 0 0 0 0 0 0 13 0 0 0 12 0 0 0 0 0 0 0 0 0 0 0 0 12 13 0 13 0 0 0 13)

(setq rhy-all (mapcar (function (lambda (x) (build-list '1/8 x))) attack-nos)
      rhy-all2 (mapcar (function (lambda (x) (build-list '1/16 x))) attack-nos))
; ((1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8) (1/8 1/8) . . .


(setq zones (add-up (p-replace-sections wtm-a rhy-all2 rhy-all)))
; (3/8 13/16 5/8 1/2 1/2 3/8 3/8 3/8 13/16 . . .


(setq wt1-z  (zone-convertor zones wt1-v)
; (-3/8 -13/16 5/8 1/2 1/2 -3/8 3/8 3/8 -13/16
      wt2-z  (zone-convertor zones wt2-v)
; (3/8 -13/16 5/8 1/2 1/2 3/8 -3/8 -3/8 -13/16  . . .
      wtm-z  (zone-convertor zones wtm-v)
; (-3/8 13/16 -5/8 -1/2 -1/2 -3/8 -3/8 -3/8 13/16 . . .
;
)

(def-neuron rhys-1
(in 1 '((wtm))) '((1/16))
(otherwise '((1/8))))

(setq rhythms-1 (run-neuron 'rhys-1 output-2)) ; lh, bs, ml

(def-neuron rhys-2
(in 1 '((wtm))) (list (gen-random nil 8 '(-1/8 1/8 1/8 1/8 1/4 1/16 1/16)))
(otherwise '((1/16))))

(setq rhythms-2 (run-neuron 'rhys-2 output-2)) ; rh part duo with ml
```

Using the frequency of attack numbers as a basis, lists of note-lengths can now be generated.

```
(setq mat (p-select (list-a-scale 0 7 :scr 2 ) (g-symbol 'a 'm))
      matx (p-select (list-a-scale 1 6 :scr 2) (g-symbol 'a 'm))
      pat (cf-noise-white 15 1.0 .37 mat)
      patx (cf-noise-white 15 1.0 .37 matx))

(setq pat1 (cf-noise-white (add-up attack-nos) 1.0 .37 mat)
      patx1 (cf-noise-white (add-up attack-nos) 1.0 .37 matx)
)


(setq paty1 ; transforming whole-tone tonality 1 to whole-tone tonality 2
(symbol-transform
   from pat ; (e m a c g e a k c k a g k c a)
   to patx ; (f l b d h f b j d j b h j d b)
   order (symbol-shuffle (g-integer 1 14) 0.1)
   changes '(1)
   repeats '(1)
))


(setq mix-template (template-invert (string-to-symbol 'x (integer-to-string-1 wtm-a))))

(setq pat1-d (mapcar 'find-anacrusis ; wt1 material
                     (symbol-divide attack-nos nil nil pat1))
      patx1-d  (mapcar 'find-anacrusis ; wt2 material
                       (mapcar 'symbol-shuffle  (symbol-divide attack-nos nil nil patx1)))
      pat1-mix (do-section :all '(octavise x 6)
                           (do-section (template-invert mix-template) '(symbol-bundle 2 x)
                                       (do-section
                                        mix-template '(gen-variants nil 1 x)
                                        (p-replace-sections wt2-a patx1-d pat1-d))))

; look above to see how p-replace-sections works:

      pat1-mixY (p-replace-sections wt2-a patx1-d pat1-d)
      paty1-d (symbol-divide attack-nos nil nil paty1)
 ; ml part
      pat1-mixZ ; rh with ml material
      (do-section :all '(octavise x 6)
                  (do-section (template-invert mix-template)
                              '(symbol-bundle 2 x)
                              (p-replace-sections wtm-a paty1-d pat1-mix)))

)
```

Now it's time for the symbol-lists to be generated. Again, the main component here is the 2 whole-tone scales. By 'adding up' the attack number lists, long lists of white-noise generated symbols are created.

Because we have the attack number values for each zone (beat/space) the long list of symbols can be divided up into sub-lists to match the note-length lists.

```
(def-neuron dynamics
(in 1 '2) '((70 65))
(in 1 '3) '((80 70 75))
(in 1 '4) '((90 75 80 74))
(in 1 '5) '((95 75 80 74 70))
(in 1 '6) (list (gen-random nil 6 '(96 74 64 55 60 72 84)))
(otherwise '((64))))


(setq dyn (build-list '(64) (length pat1-d))
 dyn1 (mapcar (function (lambda (x) (symbol-trim x (gen-cresc 50 100 15)))) wtm-a)
 dyn2 (e-insert '((64)) (e-position 'nil dyn1) (delete 'nil dyn1))
 dyn3 (run-neuron 'dynamics wt1-a)
 dyn4 (run-neuron 'dynamics wt2-a)
 dyn5 (p-replace-sections wt2-a dyn4 dyn3)
 dyn6  (do-section mix-template '(change-length :sub 20 x)
                  (p-replace-sections wtm-a dyn2 dyn5))
)


;; score

(def-tonality
wd (activate-tonality (chromatic e 5))
br (activate-tonality (chromatic e 3))
pc (activate-tonality (chromatic e 5))
st (activate-tonality (chromatic e 4))
ml (activate-tonality (chromatic e 4))
rh (activate-tonality (chromatic e 5))
lh (activate-tonality (chromatic e 4))
bs (activate-tonality (chromatic e 3))
)


(def-symbol
wd patx1-d
br pat1-d
pc pat1-mixZ
st pat1-d
ml paty1-d
rh pat1-mixZ
lh pat1-d
bs patx1-d
)
```

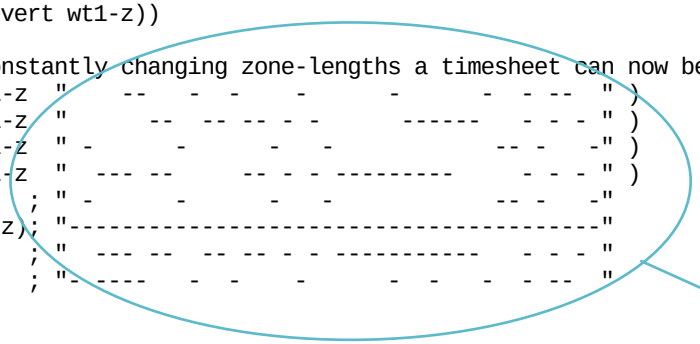
```
(def-length
wd rhythms-1
br rhythms-1
pc rhythms-2
st rhythms-1
ml rhythms-1
rh rhythms-2
lh rhythms-1
bs rhythms-1
)

(setq all-z (l-rest-revert wt1-z))

(def-zone ; despite constantly changing zone-lengths a timesheet can now be used for scoring !
wd (zone-convertor all-z  "    --   -  -    -      -      -  - --  " )
br (zone-convertor all-z  "      --  -- -- - -      ------   - - - " )
pc (zone-convertor all-z  " -      -      -   -           -- -   -" )
st (zone-convertor all-z  "  --- --     -- - - ---------     - - - " )
ml wtm-z                ; " -      -      -   -           -- -   -"
rh (l-rest-revert wt1-z); "----------------------------------------"
lh wt1-z                ; "  --- --  -- -- - - -----------   - - - "
bs wt2-z                ; "- ----   -  -    -      -  -   -  - --  "
)

(def-velocity
wd dyn4
br dyn3
pc dyn6
st dyn3
ml dyn2
rh dyn6
lh dyn3
bs dyn4
)

(def-channel
wd 4
br 5
pc 6
st 7
ml 2
rh 1
lh 1
bs 3
```



Here is a graphic that has been laid out to represent a timesheet view of the movement. The resolution of this timesheet is a list of zones - a zone value for each beat/space on the timesheet. This represents a radical and important departure from the traditional use of the timesheet.

```
)

(def-program gm-sound-set
wd soprano-sax
br trombone
pc marimba
st cello
ml vibraphone
rh electric-piano-2
lh  electric-piano-2
bs acoustic-bass
)

(def-tempo 70)

(compile-instrument-p "ccl;output:" "block3"
wd
br
pc
st
ml
rh
lh
bs
)
#|

 . . time signatures / zones

(3/8 13/16 5/8 1/2 1/2 3/8 3/8 3/8 13/16 1/4 1/4 3/8 3/8 3/8 3/8 13/16 3/8 3/8 3/8 3/4
1/4 3/8 3/8 3/8 1/2 3/8 3/8 5/8 3/8 3/8 3/8 3/8 3/4 13/16 5/8 13/16 5/8 3/8 1/4 13/16)

;; sketch pad for deriving scoring

wt1-v "  --- --  -- -- - - -----------   - - - "

wt2-v "- ----   -  -    -      -  -   -  - --  "

wtm-v " -      -      -   -            -- -   -"

wd    "    --   -  -    -      -      -  - --  "
br    "  --- --  -- -- - -      ------   - - - "
pc    " -      -      -   -           -- -   -"
st    "  --- --  -- -- - - ---------     - - - "
```

```
|#

;; Block 0 -  material for opening section into Block 1

(setq mat (p-select (list-a-scale 0 7 :scr 2 ) (g-symbol 'a 'm))
; (a c e g i k m)
      matx (p-select (list-a-scale 1 6 :scr 2) (g-symbol 'a 'm)))
; (b d f h j l)

(init-rnd 0.31)

(setq matx-a (power-set matx)
; ((b d f h j) (b d f h) (b d f h l) (b d f j) (b d f) (b d f l) (b d f j l) . . . .
      matx-a2 (symbol-shuffle (collect-length '(2) matx-a)))
; ((j l) (d h) (h j) (b h) (f j) (f l) (d l) (b j) (b f) (d j) (f h) (d f) (b l) (b d) (h l))

(setq mat-a (power-set mat)
; ((a c e g i k) (a c e g i) (a c e g i m) (a c e g k) (a c e g) (a c e g m) . . .
      mat-a2 (symbol-shuffle (collect-length '(2) mat-a)))
;((e m) (i k) (c k) (e k) (a k) (e i) (i m) (g k) (g m) (a e) (c m) (c g) (a m) (g i) (a i) (c e) (c i) (e g) (a c) (k m) (a g))


(setq matx-b2 (symbol-shuffle (append matx-a2 mat-a2 ))
; ((b f) (g m) (e m) (a k) (g k) (a i) (i k) (a m) (b l) (c g) (c e) (k . . . .
      matx-b3 (symbol-shuffle (append mat-a2 matx-a2))
; ((a c) (a i) (e k) (i k) (b l) (a g) (c k) (c i) (i m) (b d) (f h) (e g) (d j) (h j) (e m)  . . .
)

(setq matx-c2 (symbol-trim (length (collect-length '(3 4) mat-a)) ; 70
                           (symbol-shuffle (append (collect-length '(3 4) mat-a) matx-a2 )))
; ((c g i k) (a c i m) (a i k m) (c g k m) (g i k) (e i k) (a g k m) (b d) (a c m) (g i m) (b l)  . . . .
      matx-d2 (symbol-trim (length (collect-length '(3 4) mat-a)) ; 70
                           (symbol-shuffle (append (collect-length '(3 4) matx-a) mat-a2 ))))
; ((c i) (c m) (i m) (e k) (c g) (b d j) (b f h) (k m) (b d h j) (g k) (f h j) (d h l) . . . . .


(setq matx-a2i (append matx-a2 matx-b2) ; 51
; ((j l) (dh) (h j) (b h) (f j) (f l) (dl) (bj) (b f) (dj) (fh) (d f) (b l). . . . (c k) (d f) (b d) (hl) (a e) (d l) (e k) (c m))
      matx-a3i (append (symbol-trim (length matx-a2) mat-a2) matx-b3)) ; 51
;((e m) (i k) (c k) (e k) (a k) (e i) (i m) (g k) (g m) (a e) (c m)  . . . (b f) (h l) (c g) (e i) (g k) (a e) (d f) (f j) (d l))

(setq chd-incidence-1 (gen-template 0.197 1 1 (length matx-a2i))
;;(= x = = = = = x x = = x x = = = = x = = = x = = = x = x x x = = = = = x x x x x = = = = = = x = = = =)
      chd-incidence-2 (gen-template 0.197 1 1 (length matx-c2)))
; (= x = = = = x x = x x = = = x = = x = = x = = = x = x x x = = = = = = x x x x = = = = = = x = = = = = x = = x x x = = x x x x = x x = =)
```

Block 0 was composed after Blocks 1-3. The basic generative device used at the outset is the MRAC function power-set. This function is able to generate not only permutations but all possible sub-permutations from a list of elements. All the permutations from both positions of the whole-tone scale are generated.

These two template patterns show how the incidence of chords occuring in the movement was generated. Compare these patterns against the notated score or the MIDIfile output to see/hear how the music follows these patterns.

```
(setq sym-lis (do-section chd-incidence-1 '(list (compress x)) matx-a2i) ; 51
; ((j l) (dh) (h j) (b h) (f j) (f l) (dl) (bj) (b f) (dj) (fh) (d f) (b l) . . .
     sym-lisx (do-section chd-incidence-1 '(list (compress x)) matx-a3i) ; material for keyboard lh  (51)
; ((e m) (ik) (c k) (e k) (a k) (e i) (im) (gk) (g m) (ae) (cm) (c g) (a m) (g i). . .
      sym-lis-1 (do-section chd-incidence-2 '(list (compress x)) matx-c2) ; 70
; ((c g i k) (acim) (a i k m) (c g k m) (g i k) (e i k) (agkm) (bd) (a c m) (gim) (bl) (a g k)  . .
      sym-lis1x (do-section chd-incidence-2 '(list (compress x)) matx-d2) ; material for keyboard lh (70)
; ((c i) (cm) (i m) (e k) (c g) (b d j) (bfh) (km) (b d h j) (gk) (fhj) . . .
)


;; devising note-lengths by processing symbol lists through the recognition of chords
; if a chord is encountered in a list longer values are triggered than for melodic symbols

(defun rhy-mkr (lis)
 ; see Elemental Dances
(do-quietly
(cond ((is-chord (car lis)) (pick1 nil '((2/8)(3/8) (4/8) (5/8) (6/8) (7/8))))
       (t (pick1 nil (list (build-list '1/8 (pick-random '(2 3 4 5)))
                           (build-list '1/16 (pick-random '(4 5 6 7)))))))))

(defun rhy-mkr2 (lis )
  ; see Elemental Dances
(do-quietly
(cond ((is-chord (car lis)) (pick1 nil '((2/8)(3/8) (4/8) (5/8))))
       (t (pick1 nil (list (build-list '1/8 (pick-random '(2 3)))
                           (build-list '1/16 (pick-random '(4 5)))))))))

(init-rnd 0.5656)

(setq rhy-lis (do-section :all '(rhy-mkr x) sym-lis)
; ((1/16 1/16 1/16 1/16 1/16 1/16 1/16) (7/8) (1/8 1/8) (1/8 1/8 1/8) (1/8 1/8) (1/8 1/8 1/8 1/8) (7/8)  . . .
     rhy-lis-1 (do-section :all '(rhy-mkr2 x ) sym-lis-1)
; ((1/16 1/16 1/16 1/16 1/16) (4/8) (1/8 1/8 1/8) (1/16 1/16 1/16 1/16) (1/16 1/16 1/16 1/16) (1/8 1/8) (5/8) (3/8) . . . .
)

;; here the symbol-lists are reprocessed to match the note-length lists and insert the occasional anacrusis

(setq sym-lis-x (do-section
                (template-invert (gen-template 0.197 4 1 (length matx-a2i)))
; (x = = = x = = = x = = x = = = = x = x = = x = = = x = = = = x x = x x = = = = = x = = x = x = = = = x)
                '(p-replace nil 'first '= x)
                (mapcar (function (lambda (x y)(symbol-trim x y)))
                        (mapcar 'length rhy-lis) sym-lis)))
```

```
; ((= l j l j l j) (dh) (h j) (b h b) (= j) (f l f l) (dl) (bj) (= f) (dj) (fh)  . . . .

(setq sym-lis-1x (do-section
                 (template-invert (gen-template 0.197 4 1 (length matx-c2 )))

; (x = = = x = = = x = = x = = = = x = x = = x = = = x = = = = x x = x x = = = = = x = = x = x = = = = x = = = = = = = = x x = = = = = = = x =)
                 '(p-replace nil 'first '= x)
                 (mapcar (function (lambda (x y)(symbol-trim x y)))
                         (mapcar 'length rhy-lis-1) sym-lis-1)))

; ((= g i k c) (acim) (a i k) (c g k m) (= i k g) (e i) (agkm) (bd) (= c m) (gim) (bl) (= g k). . . .

(setq sym-all (append sym-lis-x sym-lis-1x)
      sym-all-c (append sym-lisx sym-lis1x) ; material for keyboard lh
)

(defun chd-test (lis)
(cond ((is-chord (car lis)) 'x)
            (t '=)))

(setq chx (mapcar 'chd-test sym-all)
      cdd (mapcar 'chd-test sym-all-c))

; (= x = = = = x x = x x = = = x = = x = = x = = = x = x x x

(defun chd-test-x (lis)
(cond ((is-chord (car lis)) (list (car (symbol-melodize (car (list lis))))))
            (t '(=))))

(setq m-sym (do-section chx '(symbol-transpose 9 x) sym-all) ; only the chords in the list are transposed
; ((= l j l j l j) (mq) (h j) (b h b) (= j) (f l f l) (mu) (ks) (= f) (ms) (oq) . . .
      b-sym1  (do-section chx '(symbol-transpose -9 x) (mapcar 'chd-test-x sym-all-c))
; ((=) (-b) (=) (=) (=) (=) (-b) (-d) (=) (-j) (-h) (=) (=) (=) (-j) (=) (=) . . .
      b-sym2 (p-replace-sections  chx b-sym1 sym-all))
; ((= l j l j l j) (-b) (h j) (b h b) (= j) (f l f l) (-b) (-d) (= f) (-j) (-h) (= f d f) (b l b l) (b d b d) . . . .


;; dynamics - objective: to have crescendos on phrases with anacrusis, decrescendos otherwise

(setq a-dyn
      (p-replace-sections  (append (template-invert (gen-template 0.197 4 1 (length matx-a2i)))
                                   (template-invert (gen-template 0.197 4 1 (length matx-c2 ))))
                           (mapcar (function (lambda (x) (gen-cresc 40 70 (length x))))  sym-all)
                           (mapcar (function (lambda (x) (build-list '64 (length x))))  sym-all)))
```

```
; ((40 48 55 62 70) (64) (64 64 64 64) (64 64 64 64 64 64) (40 45 50 55 60 65 70) . . .

(setq a-dyn1 (p-replace-sections (append (gen-template 0.197 4 1 (length matx-a2i))
                                         (gen-template 0.197 4 1 (length matx-c2 )))
                                 (mapcar (function (lambda (x) (gen-cresc 70 40 (length x)))) sym-all)
                                 a-dyn))
; ((40 48 55 62 70) (70 40) (70 60 50 40) (70 64 58 52 46 40) (40 45 50 55 60 65 70) . . .


;; important technique here - how to replace specific values in a list

(setq r-lis (e-position '(70 40) a-dyn1))
; (1 5 6 7 9 10 14 17 20 24 26 27 28 35 36 37 . . .
(setq a-dyn2 (e-insert '((64)) r-lis (p-remove r-lis a-dyn1)))
; ((40 48 55 62 70) (64) (70 60 50 40) (70 64 58 52 46 40) (40 45 50 55 60 65 70) (64) (64) . . .


;; separating the content of sym-list-x and sym-lis-1x into two distinct lists
;; containing the distribution of the two whole-tone scales that form the material for the block

(def-neuron test
(in 1 '((0 0 0 0 0 0))) '(=)
(otherwise 'x))

(setq content-list (do-section :all '(e-count '(b d f h j l) x)
                                    (mapcar 'symbol-melodize (append sym-lis-x sym-lis-1x))))
; ((0 0 0 2 0 2) (0 1 0 0 0 1) (2 0 0 2 0 0) (3 3 0 0 0 0) . . . .  (0 1 1 0 0 0) (0 0 0 0 0 0) (0 0 0 0 0 0) (0 0 0 0 0 0))


(setq wt-2 (run-neuron 'test content-list)
; (x x x x x x x x x x x x x x x = = = x = = = x x = x x x = = = = x x = x . . . .
      wt-1 (template-invert wt-2))
; (= = = = = = = = = = = = = = = x x x = x x x = = x = = = x x x x = = x =  . . .


(symbol-to-string chx) ; the output is pasted below
(symbol-to-string wt-1)
(symbol-to-string wt-2)
```

```
;; these timesheets are used to produce zone lists (via the function zone-convertor) within def-zone (see below)

(setq   wt2 "----------------      -    -  ---- -  -- - --- -        -  -               - -        -- -   -       -       -  - -  ")
(setq   wt1 "               ------- ---- --    - --  - -   - --------- -- ---------------- - ---------  - --- ------- ------- -- - --")
(setq   chs " -    -- --   -  -  -   - ---     -----      -     -    -- --   -  -  -   - ---      -----      -     -  ----  ---- --  ")
(setq   bss "--   ------  -- -- --  ------     ------     ---   -------   --   -- -- - - -        -------    - -   - -  --- ----  --- ")
(setq   w+b "              - -  --  -   ----       - -  -  ---- ---   -  --- -  --   - -   -----      ------ ----- ----  -- ---  ---")
(setq   p+s "- ----  -  --- -   -   -        --     - --  -       -         -   --          -                           -   ")


;; bss handwritten to provide a complementary part to chs

(symbol-to-string (find-uncommon-1 (string-to-symbol 'a wt1) (string-to-symbol 'a chs))) ; output pasted above
(symbol-to-string  (find-uncommon-1 (string-to-symbol 'a wt2) (string-to-symbol 'a chs)))

; score

(def-tonality
 wd (activate-tonality (chromatic e 5))
 br (activate-tonality (chromatic e 3))
 pc (activate-tonality (chromatic e 5))
 st (activate-tonality (chromatic e 4))
 ml (activate-tonality (chromatic e 5))
 rh (activate-tonality (chromatic e 5))
 lh (activate-tonality (chromatic e 4))
 bs  (activate-tonality (chromatic e 3))

)

(def-symbol
wd sym-all
br sym-all
pc sym-all
st sym-all
ml m-sym
rh sym-all
lh sym-all-c
bs b-sym2
)

(def-length
default (append rhy-lis rhy-lis-1)
)
```

```
(def-zone
rh  (make-zone-list (symbol-of rh) (length-of default))
lh  (zone-convertor (zone-of rh) chs)
ml  (zone-convertor (zone-of rh) chs)
bs  (zone-convertor (zone-of rh) bss)
wd  (zone-convertor (zone-of rh) w+b)
br  (zone-convertor (zone-of rh) w+b)
pc  (zone-convertor (zone-of rh) p+s)
st  (zone-convertor (zone-of rh) p+s)
)

(def-velocity
default a-dyn2
)

(def-channel
 wd 4
 br 5
 pc 6
 st 7
 ml 2
 rh 1
 lh 1
 bs 3
)

(def-program gm-sound-set
wd soprano-sax
br trombone
pc marimba
st cello
ml  vibraphone
default electric-piano-2
bs acoustic-bass
)

(def-controller gm-controllers
  (wd  panning  '((20)))
  (br   panning  '((50)))
  (pc   panning  '((80)))
  (st   panning  '((120)))
(ml panning  '((120)))
(default panning  '((64)))
(bs panning  '((10)))
```

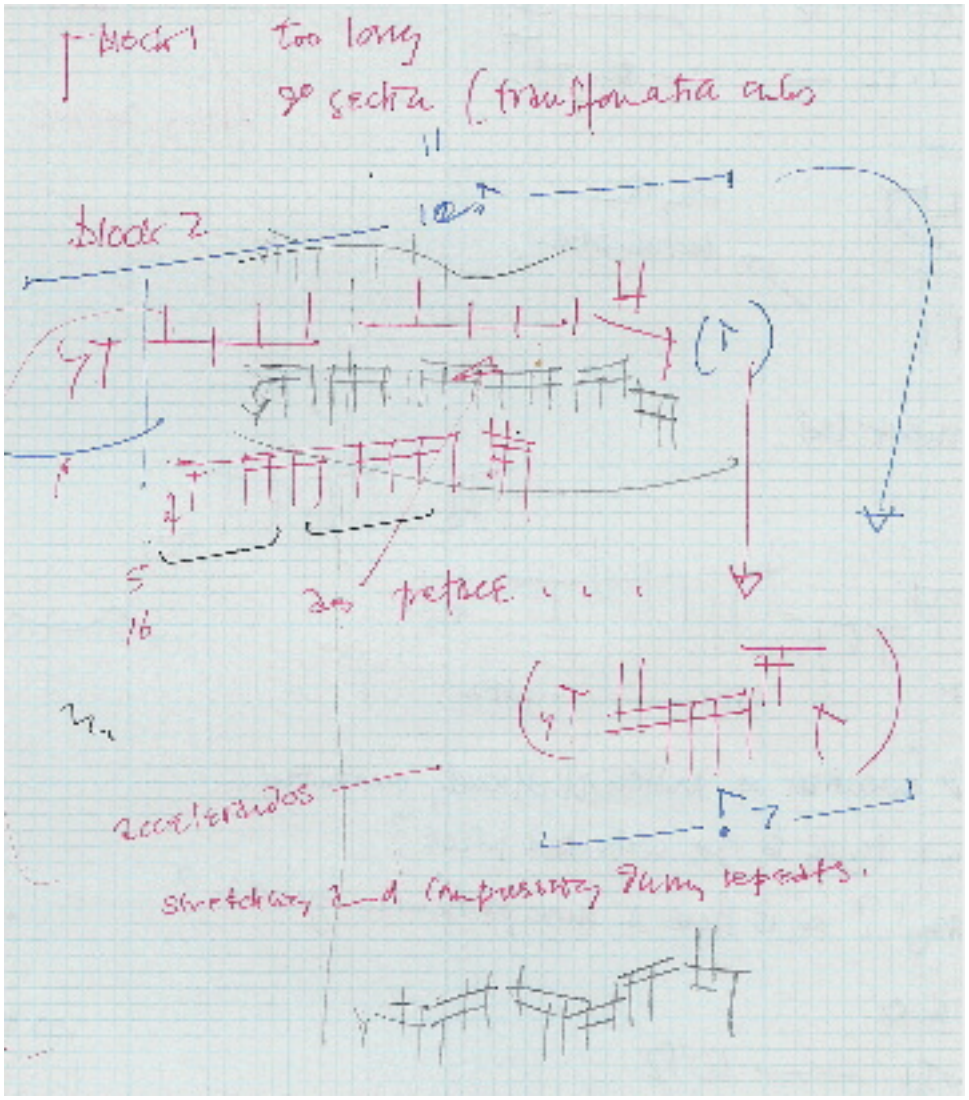
```
)
(def-tempo 70)

(compile-instrument-p "ccl;output:" "block0"
wd
br
pc
st
ml
rh
lh
bs
)

#|

time-signatures


(7/16 7/8 1/4 3/8 1/4 1/2 7/8 5/8 1/4 1/4 3/4 1/2 1/2 1/2 3/4 3/8 7/16 1/2
1/4 1/4 1/4 1/4 7/16 1/4 1/4 1/2 1/4 1/4 1/4 5/16 3/8 5/8 3/8 1/2 1/2 1/2
7/8 7/8 7/8 1/2 1/4 3/8 3/8 1/4 1/2 3/8 7/8 1/4 3/8 1/4 3/8 5/16 1/2 3/8
1/4 1/4 1/4 5/8 3/8 3/8 3/8 3/8 3/8 1/4 1/4 5/8 1/4 1/4 5/8 5/16 3/8 1/2
1/4 3/8 5/16 1/4 3/8 3/8 1/4 5/8 5/16 3/8 3/8 1/4 3/8 1/4 1/2 1/2 5/8 1/2
1/4 3/8 1/4 5/16 1/4 3/8 1/4 3/8 1/4 3/8 1/4 1/4 3/8 1/4 3/8 1/4 1/2 5/8
1/2 1/2 3/8 1/4 5/8 5/8 3/8 1/4 1/4 5/8 1/4 1/4 1/4)
|#
```



Here is an image from the composer's note-book of some of the rhythmic ideas and sketches that form the material for Block 2.

In the SCOM score-file this material is transcribed into a sequence of note-length lists using the MRAC qlength function, probably the fastest way of notating rhythmic sequences.