# SYMBOLIC COMPOSER - a description and evaluation

Nigel Morgan - IMPAC Consultants

This paper addresses those teaching composition whose work is not exclusively concerned with adventures in music technology or 'sonic' exploration but is rooted in the traditions, conventions and practices of art, popular and world musics. Symbolic Composer mediates between the two. It is a thinking environment for composition with the traditional elements of pitch, rhythm, and harmony; it is MIDI production tool.

## Introduction

The text describes and evaluates a recent development in computer software for composers called Symbolic Composer. This appears to provide an acceptable link and a stepping stone between the MIDI sequencer / scorewriter and interaction with a high-level programming language, LISP. Acceptable because it takes the form of an expert-system incorporating a rich vocabulary of extendible musical functions. Furthermore, the software can be integrated with current MIDI sequencer / scorewriters on existing Macintosh and Atari computer platforms of 4 megabytes and upwards.

The evaluation begins from the author's premise that for those undertaking serious work or study in composition the MIDI sequencer may fall far short as an intellectually acceptable environment for sustained creative partnership. The sequencer provides a way of organising, auditioning and editing material usually generated from on-line musical performance rather than as a result of reflective and off-line musical thought. As a result a composer's material collects all the baggage of performance as data and the composer will devote large tracts of creative energy to editing rather than imagining and reflective listening.

This premise further recognises the issue of mediating between the language of compositional elements, techniques and considerations appropriate to music for performers to play and that plethora of tongues found in software synthesis and signal processing; the property of machine performance. Music education, now and for the immediate future, continues to value the acquisition of a western traditional symbolic language based on staff notation and the supremacy of tempered pitch and divisive rhythm, whilst encouraging, from the primary classroom onwards, a sensitivity towards timbre and looser alternative (and often non-western) musical modes of expression and performance.

The establishment of MIDI in the mid 1980's as a computer protocol dispensed with the need for structured 'programming' in composition with computers and substituted digital recording and editing. It effectively replaced composition with composing. It all but grounded the introduction into education of the many developments concerned with defining an acceptable language for expressing music in the digital domain. Those languages that have developed have usually been vehicles for research work with artificial intelligence and have not been available or accessible to the practising composer.

Programming for the expression and development of musical composition (other than electroacoustic music), music cognition or analysis has for these reasons failed to find a foothold in education.

As a result composers using MIDI sequencers seldom employ or understand the software features involving elements of programming, structured or logic-based thinking. An example here might be in the list processing of sequences or in boolean editing of note data. MIDI programming has become synonymous with learning the C

language and writing bespoke applications or accessing the 'score' file protocol of CSOUND. Programming to create a structured environment through which a non-electroacoustic composition might evolve appears not be recognised as having aesthetic potential or educational value.

MIDI has been described as being ' to some extent capable of expressing what we might call conventional western music. MIDI treats all concepts outside this realm as aberrations - including, among other things, many characteristic features of .... serious music written since 1900 (Weston 1991)'. It is a protocol for data transmission and not a structured language. The essential concepts of MIDI required for music composition can, in fact, be demonstrated in five short examples (Morgan 1990).

Algorithmic tools for composing (the real-time activity of setting predefined musical elements into playful juxtaposition) and the midIfile protocol are able to extend the MIDI sequencer's role as a composition tool. '*M', Realtime, Fingers, Tunesmith, Jam Factory, Tango, Improviser* and most recently (and significantly) *MAX* and Subotnik's *Interactor* provide on-line composing environments whose time-based processes can, in some cases, be captured and replayed as data recordings. Fractal generation programs have appeared that offer another, though limited, field of possibilities; limited because these programs usually express only one fractal algorithm.

Symbolic Composer can be seen as an environment both for structured programming and the realisation of generative algorithms in a format that integrates fully with the MIDI sequencer / scorewriter. It offers a medium for creative thinking and exploration able to use any information structure as a source for musical composition.

The following descriptions explain how this integration is achieved and explore the potential SC affords for creative thinking, analysis, music cognition and composition itself. But first, a rationale for LISP programming for music composition.

Programming and Music Composition

Most musicians currently leading the study and practice of composition within education began writing music before having access to computers. They have experienced the essential chemistry that results from reflective thinking and contextual feedback. Exposure to the techniques of serial composition whether quantitative or qualitative has encouraged a desire for unity, equilibrium and connection between the elements of musical expression within a composition, whatever its style.

There is an emerging generation of composers who, with access to MIDI sequencers from an early age, may never encounter that particular experience of composition where the chemistry of thought overides the presence of musical clock time and establishes a virtual or imagined space in which musical ideas form and develop.

Programming may well have an important role in redressing this situation. To do so the composer needs to perceive the potential in the creative outcome and for their intellectual development in acquiring both a language and the logic skills to use and understand it. At the present time even MIDI is often considered with a phrase-book mentality; its limited protocol rarely understood. There is more than enough evidence to demonstrate that programming need not be just codes and commands but an adjunct to a natural way of learning about the nature of thought and communication. Programming languages such as BASIC and C are not recognised by either educationalists or artists as having the potential to fulfil this requirement. LOGO, the only computer language to achieve recognition within our National Curriculum, and LISP certainly are.

Both these languages encourage an approach to programming that is much closer to creative pattern-making and recognition that forms the basis of most musical composition. To program in LISP we seek to represent what we know about our creative objective and the way we describe what we know about it. In the process of doing this the object or objective actually reveals itself. In music composition utilising a symbolic language this makes possible an integration of and between the core material of different musical elements, pitch and rhythm, harmony and structure, dynamics and register. Indeed, any parameter can interact, be part of the definition of, or transform any other.

Until Symbolic Composer it has not been easy to make the musical connections with LISP in a way that did not require an extensive prior knowledge of the language. LISP, as it stands, knows nothing about music. But, because its data format is symbolic and music can be described and worked with at a symbolic level, it can acquire a knowledge-base which contains all the knowledge about music it needs and the ability to work with that knowledge in a formal way.

Symbolic Composer makes LISP an expert on matters musical. Together they form an 'expert-system' where structures with which knowledge is represented can be manipulated. These systems use programs that are descriptive ( I would like to see all the possibilities of this pitch series by a forward rotation of one step at a time - I don't care how it's done ) rather than prescriptive (I shall have to write a program telling my computer exactly how to do this operation).

To the composer all this means that one can access a very powerful high-level computer language without having to engage the language in any other activity than music. In fact, the amount of specific LISP needed to work at an advanced level with Symbolic Composer can be written in a few lines of instructions, its formal properties covered in an hour, like the rules of chess.

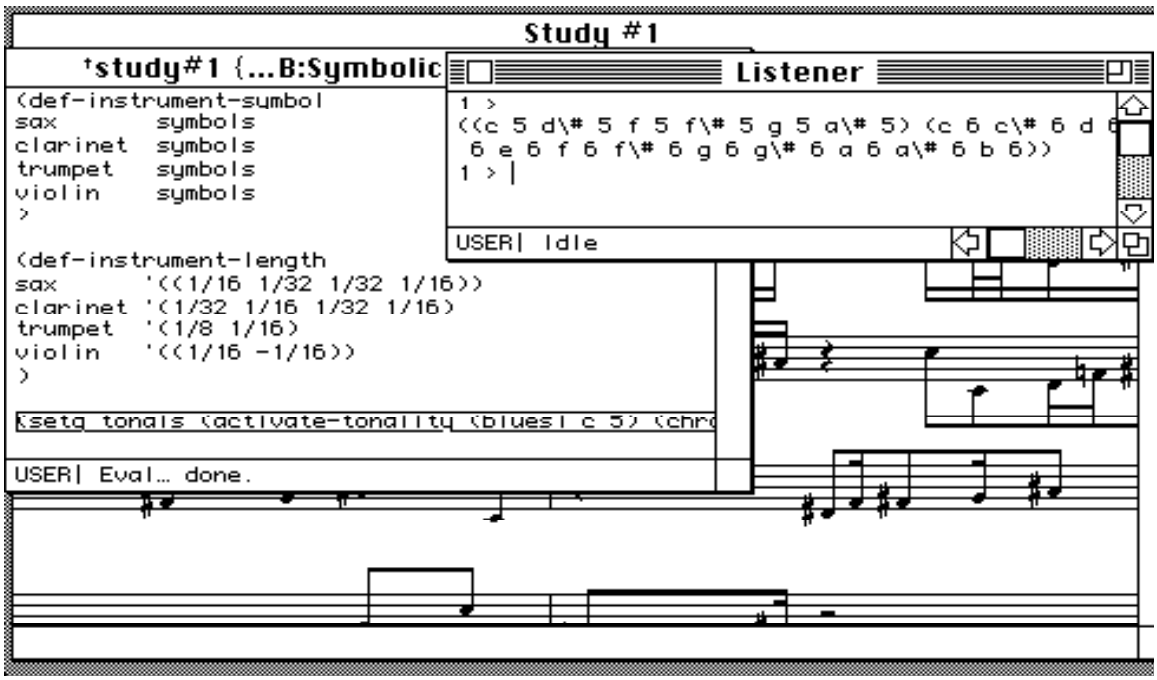



*Figure.1* The Symbolic Composer screen with the MCL interpreter, editor and *Concertware* Music Writer displaying the material for study#1 from '*An Introduction to Symbolic Composer*".

A Description of Symbolic Composer

This covers the very basic features of Symbolic Composer that might be encountered in a preliminary tutorial session. The text and examples come from the Hypercard stack '*An Introduction to Symbolic Composer*' that accompanies the Macintosh version of the software.

**SYMBOLIC COMPOSER** was created by Peter Stone of Tonality Systems between 1987 and 1992. It has been described as.. *a powerful modelling and development system suitable for all kinds of music.*

- It allows the mapping of structures made up of

  SYMBOLS (a b c d) or NUMBERS [1 2 3 4] directly onto
  elements of MUSIC r x e q
- Musical data can be realised as notes, rhythms, velocities, durations and instruments and saved to a standard 1.0 midifile.
- Performance, library or improvised material can be analysed from a midifile and generate symbolic or numerical information.

**A Guide for Composers**

Symbolic Composer has a detailed and authorative manual written and edited by composers Janet Owen Thomas and Nigel Morgan. It is designed for day to day use with the software and includes tutorial material.

In the Tutorial's 'Guide for Composers' there are ten example programs. These evolve from a one bar quartet into an extended composition for instrumental octet. In the process all the standard elements of the Common Music Language (CML) are presented in a working musical context. Following the examples a composer should be able to acquire the essential programming techniques and language concepts needed to begin composition.

There are two supplements for **Rhythm** and **Pitch** with accompanying program templates that allow a composer to see how particular functions work with their own material without having to construct a complete composition.

**Technical Introduction**

ATARI - SC runs under Metacomco LISP and requires at least 4 megabytes of RAM; a hard disk is essential. A MIDI sequencer or scorewriter that reads midifiles is necessary. The software can operate as a module within Dr.T's Multi-Programming Environment.

MACINTOSH - SC runs under Macintosh Common LISP, one of the most powerful LISP implementations in the world. With 4 megabytes MCL can handle multi-tasking to work tom proceed within other programs while compiling CML files. A MIDI sequencer or scorewriter that reads midifiles is required.

**An Example from A Guide for Composers**

STUDY #1- is the culmination of the first stage of exercises.

Here is the **Timesheet** of Study #1.

```
(compile-song "hd 40mb:sc:output:studies:" 1/4 separate

     ; BARS                  |---|---|---|---|
     scale         tonals  ".....       ..."
     sax           scale   "-  - -- - --- --"
     clarinet      scale   " -  -- -- -- ---"
     trumpet       scale   "  -   -- -- -- -"
     violin        scale   "   -   --  -----" )
```

The Timesheet shows the moment of time when each instrument plays and is muted, and the points of tonality change.

The line ( - ) in a resolution of 1/4 indicates that each column is of a 1/4 length. The notes inside that column can be of arbitary length. The dots (. . .) in the *tonals* line indicate the point where the tonality changes - from blues to chromatic, chromatic to blues.

The word *scale* describes the two tonalities inside the word *tonals* and the time points at which these tonalities change. Look at the next card for a description of *tonals*.

Here are the instructions for **pitch 'mapping'** in Study#1

```
(setq tonals (activate-tonality (blues1 c 5) (chromatic c 6)))
```

* Here is a LISP list! It is both data and a program in itself.
* Unless told otherwise by a single ' quote LISP sees the first symbol following a left parenthesis as being the name of the function to be evaluated, and the rest of the contents of the list as the arguments to be passed to the function. LISP then returns a result, beginning its evaluation from the innermost set of parenthesis.

```
IN: (setq tonals (activate-tonality (blues1 c 5)(chromatic c 6)))

OUT:((c 5 d# 5 f 5 g 5 a# 5 )
    (c 6 c# 6 d 6 d# 6 e 6 f 6 f# 6 g 6 g# 6 a 6 a# 6 b 6))
```

*activate-tonality* is a Symbolic Composer function that activates the tonality zones (blues & chromatic) and returns the notes associated with the names.

*tonals* is an invented name given to describe the two 'tonality zones', blues and chromatic. The symbol melody (a b c d e f g) - see the final card - used in the composition is mapped onto *tonals*. Look at the timesheet to see how it is used.

*setq* is a LISP function that sets a value or structure to the word it precedes. It creates a variable.

Here are the instructions for **note length** in Study#1

```
     (def-instrument-length
     sax           '((1/16 1/32 1/32 1/16))
     clarinet      '(1/32 1/16 1/32 1/16)
     trumpet       '(1/8 1/16)
     violin        '((1/16 -1/16)))
```

The quote sign ' returns its own argument i.e. the sax part plays in the rhythm of the note lengths between the brackets. The ' quote stops the list of note lengths from being evaluated. LISP would want to know what to do with the list of note lengths!

The sax and violin have 'nested' lists (( )) which restart at the beginning of each tonality zone. The clarinet and trumpet have single lists ( ) which allow the symbols to freely overlap tonality boundaries.

Here is how the **symbols** are defined and set to instruments. The symbol pattern will produce scale-like figures in all parts. It's followed by the complete program for Study#1.

```
(setq  symbols '(a b c d e f g)

(def-instrument-symbol
     sax        symbols
     clarinet   symbols
     trumpet    symbols
     violin     symbols)

(def-instrument-length
     sax           '((1/16 1/32 1/32 1/16))
     clarinet      '(1/32 1/16 1/32 1/16)
     trumpet       '(1/8 1/16)
     violin        '(( 1/16 -1/16)))

(setq tonals (activate-tonality (blues1 c 5) (chromatic c 6)))

(compile-song "hd 40mb:sc:output:studies:" 1/4 separate

     ; BARS                 |---|---|---|---|
     scale         tonals  "......       ..."
     sax           scale   "-  - -- - --- --"
     clarinet      scale   " -  -- -- -- ---"
     trumpet       scale   "  -   -- -- -- -"
     violin        scale   "   -   --  -----"
   )
```

## Why Symbols?

Serialism in both quantitive and qualitative forms has encouraged composers to devise ever more ingenious and complex structures and relationships of pitches, rhythms and timbres.

The idea of pitch-class generation and transformation is, in nature, symbolic as the 'series' is often transmuted to different levels and functions. Many of the disciplines of serialism have become part of the working practice of composers who build their music on ideas of 'process'.

By using a symbolic language and LISP processing, composers can explore relationships between musical elements impossible to achieve 'on paper' or via the MIDI sequencer.

Here is a short example to illustrate:

I have a progression of 3 chords: Caug7, F#aug9, Eminmaj7
I want to construct a melody and rhythm from each chord.

SC's `gen-sin-chord` function is used to calculate the sum of the frequencies of each chord.

```
IN:  (setq c+7mel
            (vector-to-symbol a  l
                  (gen-sin-chord
                  '( c 3 e 5 g# 5 a# 5) 30)))
```

The frequencies are calculated with `gen-sin-chord` and converted to 30 symbols lying between a and l with `vector-to-symbol` . The result is a list of symbols which can be mapped onto any scale, note-row or pitch region.

```
OUT:(g d h e a a f f d g h i e c a f b b c d e h i l l a f)
```

Next, to create a rhythm, use the same idea but with these alterations:

```
IN: (setq c+7rhy
           (vector-round 24 96
                  (gen-sin-chord
                  '( c 3 e 5 g# 5 a# 5) 20))))
```

The frequencies are calculated with `gen-sin-chord`, scaled to lie between 24 and 96 (MIDI tick values for semi-quaver and crotchet at a resolution of 384 to the whole note) with `vector-round`.

```
OUT:[24 35 76 89 67 25 45 57 89 23 57 36 87 26 93 57 24 67 74 56 89]
```

For exact rhythmic values from vector processing the function `vector-quantize` would allow only these values to be produced:

24 36 48 60 72 84 96

```
(setq c+7rhyq (vector-quantize 7 20 c+7rhy))
```

Note that there are 7 'levels' between 24 and 96 and 20 'samples' taken.

For Pitch & Rhythm Music:

The MetaScore Language of Symbolic Composer incorporates all the common conventions of music and allows them to be extended and juxtaposed in exciting and novel ways. Here is a selection.

**melody symbols** are one-letter symbols. 'a is always mapped to the first note of a tonality. (a b c ) in C major gives c d ; (-a -b -c) gives c b a.
**chord symbols** put the symbols together: (ace) in C major maps to c e g.
**transpose symbols** add a + or - number indicating a semitone transposition. (a b (-1c)) produces in C major c d eb.
**length symbols** are defined with the m/n formula. 1/4 is crotchet length. 1/4. a dotted crotchet. 1/4t is a crotchet triplet. 1/4-5 is a quintuplet.
**rest length** notation adds a minus before a length symbol. -1/4 is a crotchet rest.

For Electroacoustic Music:

Using MIDI controlled samplers and synthesisers SC lets you structure sounds and their performance characteristics at a symbolic level. Any reference to conventional music structures can be avoided.

ANY information structure can be used as a seed to control sonic or musical elements. Remember, the basic idea of the software is the freedom of conversion between symbolic and numeric information.

## Symbol and Vector Generators

These can become the basic source of melodies, note-lengths, rhythms, chords, dynamics, tonalities and musical structures.

SYMBOL GENERATORS produce symbol patterns using recursive definitions, recursive string rewriting L-system or Fibonacci strings.

VECTOR GENERATORS produce vector patterns - unlimited length vectors of real or integer numbers. Both symbol and vector patterns can be scaled, shifted, mixed, filtered and looked at graphically

Since LISP has a complete set of mathematical functions you could add symbol or vector generators based on any 'thinkable' method. Here are the predefined vector generators:

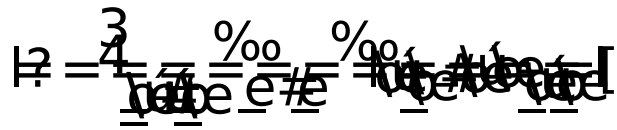
DIGITAL SYNTHESISER - has unlimited digital waveform generators, modulators, mixers and filters. It can be used in additive or FM synthesis.
HOPALONG GENERATOR - iterates the Hopalong algorithm and generates x y vector patterns. When the values are drawn on an x y plane the figure resembles cells and organic structures, or gothic floor designs.
BROWNIAN GENERATOR - generates brownian noise using a midpoint algorithm. 'Natural' phenomena fractals can be made using brownian motion.
HYPERCUBE GENERATOR - create n depth hypercubes in n dimensions. The points of each dimension are returned as a vector pattern, which can be used to control composition parameters.

**Symbol and Vector Processing**

Composers are familiar with the conventions of inversion, mirroring, transposing and rotation. Symbolic Composer allows all these operations (and many others) to take place on symbolic material. Since the operation is symbolic and the values are later mapped to actual tonalities, these operations may be diatonically accurate unlike the majority of MIDI operations which are solely chromatic. The following examples demonstrate this with one symbol melody and a variety of tonalities:

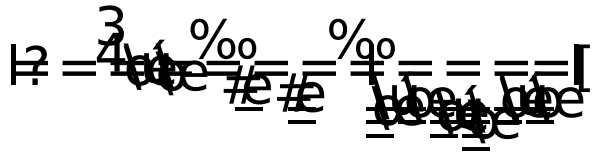
```
Melody        (setq melody '(a b = c d = f e g h e e))
Tonality      (setq tonal (activate-tonality (chromatic c 3)))
Length        (setq rhythm '(1/8))
```

ex1



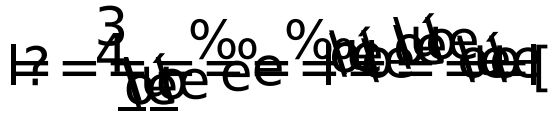
```
Melody        (setq melinv (symbol-mirror 'a  melody))
Tonality      (setq tonal  (activate-tonality (whole-tone g 3)))
Length        (setq rhythm '(1/8))
```

ex4



```
Melody        (setq meltr1 (symbol-transpose 1 melody))
Tonality      (setq tonal  (activate-tonality (major c 3)))
Length        (setq rhythm '(1/8))
```

ex7



Vector patterns can be mixed, amplified, scaled or the offset shifted. As this example from the Author's *Out From The Edge* demonstrates a sine wave can even be modulated by another sine wave to produce a list of 24 velocity values between 32 and 120.

```
(setq attack1
      (vector-round 32 120
            (gen-sin 1 0.5 24 90 (gen-sin 1 0.3 24))))
```

The function `gen-sin` returns a vector with a length determined by the number of samples (24) at a given frequency (1.0), amplitude (0.5) and phase angle (optional - 90).

**Recursion, fractals and the use of associative symbol structures**

*Recursion* is the ability of a piece of information or 'object' to recur by constantly going back on or 'calling' itself. If that material is defined in particular and simpler ways the recurrence does not end up being a straight repeat or loop. A pattern is generated which can explain a great deal about the nature and structure of our information or object. It can be used to solve a problem in terms of itself.

The best analogue for recursion is the fractal pattern. *Fractals* are those wonderful graphic images that appear to resemble natural phenomena. When applied to musical elements such as melody, rhythm and dynamics a similar 'natural' quality can be obtained.

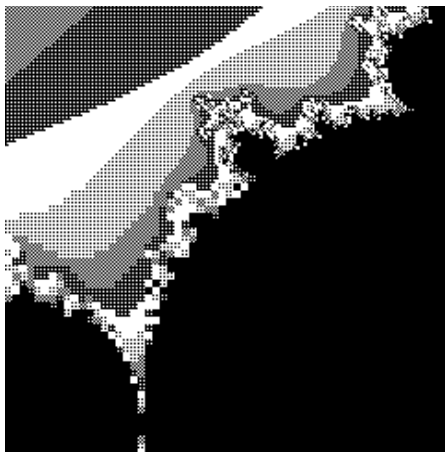


*Figure 2*. A Mandlebrot fractal

Symbolic Composer has a number of symbol generators that can create the most elegant fractal patterns with musical material. Although Symbolic Composer cannot produce true visual fractals itself it can represent symbol and vector patterns via draw-symbol and draw-vector functions. Here is a representation of the brownian noise fractal that creates the melodic, rhythm and tonality for the author's *Trio* for violin, sax and piano (example 3 in *Introduction to Symbolic Composer* ). It was drawn on the Atari version of Symbolic Composer with this procedure:

```
(draw-vector (gen-noise-brownian 9 0.4 0.4))
```

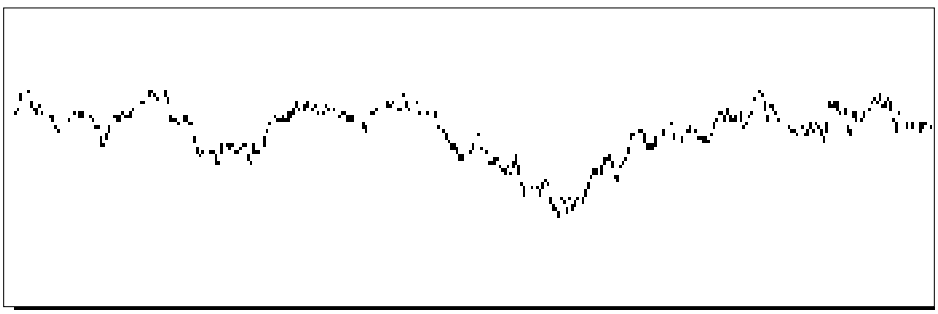


*Figure 3.* A Brownian noise fractal drawn by Symbolic Composer

The Symbolic Composer function `defsym` is the nucleus of three symbol generators defining a new IS-A relationship. It can be used to represent associative information; from a single seed one can generate associations recursively to any depth. A model of such a description can be seen in the next section as part of the material for the author's *Heartstone* for symphonic band.

The manual describes the process:

*When expressing associative structures it is possible to take a snapshot of the inner geometries operating on the unconscious mind. Sometimes it happens, that when composing with SC the music that comes out of the symbol definitions corresponds very much to the inner feelings, as though the software can read the mind of the composer. This is natural, since the structures you intuitively choose are part of yourself.*

An Evaluation of Symbolic Composer

*Symbolic Composer will never limit your creative power - but in order to master it you must learn a new way to think....The power of this language is enormous. It is as universal as the DNA language that has coded every cell of all life forms. This comparison is not so far-fetched as it might sound at first....The Core System was programmed to operate on the same principle as the ribosome which converts RNA structures into life. And this Core - a kind of musical ribosome - became the heart of Symbolic Composer... Because of its roots and nature, it is for music as powerful a tool as RNA and Ribosome are for life.* (Stone1991)

You certainly do have to learn a new way to think. And for composers, particularly those with an established technique, this may not come easily. Whilst it was not difficult to utilise the functions of the language and 'get results' Symbolic Composer seemed to challenge one's relationship with the common elements and materials of pitch, rhythm, tonality and structure.

The co-ordination of rhythm was a particular stumbling block in that it appeared difficult to define that flexibility that melody demands. Riff-like structures seemed to be the only order of the day. The way out of this problem was to rigourously examine the different ways rhythm could be expressed; the production of a Rhythm Supplement devised twelve distinct methods. Rhythm can even be articulated within a melody symbol list.

`(a b c d e f g)` can include a rest symbol = `( = a b = c d = e f g)`

With the provision of a Neural Expert in the Macintosh version conversion from symbols to note lengths has become possible.

```
(def-neuron  sym-to-len
   (in 1 a) '(1/4)
 (in 1 b)'(1/8 1/8)
 (in 1 c) '(-1/8 1/8)
 (in 1 d) '(1/16 1/8 1/16)
 (in 1 e)'(1/16 1/16 1/16 1/16)
 (in 1 f)'(1/8 1/16 1/16)
)

(run-neuron 'sym-to-len
      '(a d e b b e f))
```

This would produce

q x e x x x x x \ e e e e x x x x e x x \

As my own work demonstrates, composition can be approached from many starting points and integrated with sequencer realisation at different stages within the process of composition. Here are some examples:

*Out From The Edge* - movement II (jazz-rock sextet). The melodic and rhythmic parts were derived from samples taken from the oscillating frequencies of a three chord sequence projected onto a sine wave (the program for this realisation can be seen earlier in the text) The harmonic part, a kind of continuo, was freely written on the sequencer.

*Out From  The Edge* - movement III (electroacoustic music for soloist and synthesisers). Whilst the melodic material comes from a progression of three intervals, the rhythm for the synthesisers is derived from a modulation of sine waves - and without quantisation is organic (and un-notatable). A series of velocity curves are generated from progressive modulation of a sine wave and provide a six part counterpoint of timbral change across the operators / partials of the synthesisers. A second section is based on the fibonacci generation of melodic symbols, rhythmic and velocity values.

```
(def-instrument-length
synperc (gen-fibonacci 5
          '(1/4 1/8)'(1/16 1/16 1/8))
synbass (gen-fibonacci 5
         '(1/8 1/8 -1/8) '(-1/16 1/16 -1/16 1/16))
sax (gen-fibonacci 5
         '(1/16 1/16 1/16 1/16 -1/16 1/16) '(-1/8 1/16 1/16)))

(def-instrument-symbol
synperc (gen-fibonacci 5
            '(q r)'(u s t))
synbass (gen-fibonacci-t 5
            '(a h =) '(= e = (-2 a)))
sax (gen-fibonacci-t 5
            '(a d c e = f) '(= g b)))

(def-instrument-velocity
synperc (gen-fibonacci 5
            '(90 70) '(85 70 75))
synbass (gen-fibonacci 5
            '(75 85 =) '(= 90 = 70))
sax (gen-fibonacci 5
            '(50 54 57 78 = 80) '(= 92 95)))
```

*Heartstone* - music for symphonic wind band - movements I & II. This takes a 'minimal' series of definitions and subjects each symbol to recursive definition at different depths of recursion.

```
(defsym a '(b c))
(defsym b '(b b a d))
(defsym c '(c c d a = ))
(defsym d '(a = = b c))
```

The musical material in six parts is expanded by tonality counterpoint shown graphically below. One tonality definition is shown - the others are identical except for their starting octave. A section from the timesheet shows exactly where each tonality changes over 32 bars. Note that the timesheet can be in any resolution. Here each - in the 'bars' line equals a whole-note.

```
(setq tonal1
(activate-tonality (chromatic f 6)(phrygian f 6)
 (diminished1 f 6)(whole-tone f 6)(pentatonic f 6)(pentamajor f 6)))

(compile-song "hd 40mb:sc:output:studies:" 2\/2 separate
% BARS            |---|---|---|---|---|---|---|---|
scale1 tonal1    "    .   .   .   .     .          "
scale2 tonal2    "      .   .   .   .       .      "
scale3 tonal3    "   .   .   .   .     .           "
scale4 tonal4    "  .      .   .      .        .   "
scale5 tonal5    "           .   .   .    .     . "
scale6 tonal6    "     .   .   .      .     .     "
```

*Trio* - violin, saxophone and piano. In addition to the brownian noise fractal (figure 3) generating melody, note-length, velocity and tonality change zones three summation series scales are created as the tonalities for the piece. The previous examples have all used tonalities predefined within Symbolic Composer. The summation scales were created specifically using a 'half-tone' notation. They have been adjusted to the compass of an octave using a method outlined by Joseph Schillinger (Schillinger 1947).

```
(create-tonality sumscale1 '(1 2 4 7 12 8 9 6 4 11))
; c db eb f# b g ab f eb bb

(create-tonality sumscale2 '(1 2 5 9 4 3 9 2 1))
; c db e ab eb d ab db c

(create-tonality sumscale3 '(1 2 6 11 8 10 9 10))
; c db f bb g a g# a
```

The changes from scale to scale are handled via a `symbol-to-tonality` function which imposes a symbol template (a fractal produced by brownian noise called here *ptch3*) as a tonality scheme. The scales are transposed diatonically according to the symbol. *Ptch3* is then itself mapped onto the tonalities.

```
IN: (setq tonal
      (symbols-to-tonality
          symbol: ptch3
        transpose: '((0 1 2 3 4 5 6 7 8 9)(0 1 2 3 4 5 6 7 8)
        (0 1 2 3 4 5 6 7))
        mapping:(activate-tonality (sumscale1 c 3) (sumscale2 c 3)
        (sumscale3 c 3))))


OUT: (setq ptch3 '(c c d b b c b b b a b a b b c d d d e e d d c c c
b d d d d e d c))

OUT: (setq tonal
c           '((d# 3 f# 3 b 3 g 3 g# 3 f 3 d# 3 a# 3 c 4 c# 4)
c           (e 3 g# 3 d# 3 d 3 g# 3 c# 3 c 3 c 4 c# 4)
d           (a# 3 g 3 a 3 g# 3 a 3 c 4 c# 4 f 4)
b           (c# 3 d# 3 f# 3 b 3 g 3 g# 3 f 3 d# 3 a# 3 c 4)
.      etc........)))
```


Summary

The author believes Symbolic Composer to have considerable potential as a supplement to composition work with computers in music education, where it could also be used to develop thinking skills and even for work in cognition and analysis. As a personal tool for composers it integrates well with existing sequencers and scorewriters and will run happily on platforms such as a 4/40 Macintosh *Classic* or *Mega 4* Atari. Compilation time is quite acceptable, even on the Atari which doesn't multi-task - the example Study#1 takes about 20 seconds to create four separate Midifiles.


REFERENCES


Morgan.N (1990) *The IMP Educational Sequencer : teachers resources.* (IMPAC Resources)
Schillinger.J (1947) *The Schillinger System of Musical Composition* (Da Capo)
Stone.P (1991) *Symbolic Composer : release 1.0 for the Atari* (Algorithmic Research)
Tolonen.P (1990) *Auditive Monitoring of Scientific Data* (Step-90)
Tolonen.P (1988) *Lisp-poh jainen saveltajan tvyokaluoh jelmisto Symbolic Composer* (Step-88)
Weston. C (1991) *Notewriter: A unique approach to music printing software* (Musicus vol2/i & ii)