Illinois, 1987

HMSL: Overview (Version 3.1) and Notes on Intelligent Instrument Design

Larry Polansky, David Rosenboom, and Phil Burk
Center for Contemporary Music, Mills College, Oakland, CA 94613
david@mills.berkeley.edu

Abstract: 1) HMSL (Hierarchical Music Programming Language) is defined and described, in terms of its programming environment, features, and especially, data structures. 2) Some theoretical examples of the use of the language by composers (Rosenboom, Polansky) are given, with particular attention to the language's capability for the creation of intelligent "instruments". This section describes several important morphological transformation functions that have been used in HMSL as algorithmic composition and performance ideas.

1. Overview of HMSL

1.1 Introduction

1.1.1 Purpose of HMSL. *HMSL* (Hierarchical Music Specification Language) is a programming language for use by composers, researchers, and performers. It is intended to be a flexible and highly general environment, whose data structures and high-level programming tools provide the user with the resources to realize works and experiments in a wide variety of styles. One of the primary intents of the HMSL data structure design is to provide a "non-stylistically based" language for musical form.

1.1.2 Programming Environment. HMSL is written in an object oriented programming environment called *ODE* (Object Development Environment), written by Phil Burk. ODE is written in FORTH, and is based on the SmallTalk object oriented programming model, which includes class inheritance, use of methods, message passing, and hidden data for objects. All three levels of programming (FORTH, ODE, HMSL) are available to the user at all times. Many of HMSL's libraries (like the MIDI drivers) do not have to be used within the powerful hierarchical scheduling environment of HMSL itself, but may be called directly from FORTH or ODE. Conversely, any high level routine written in HMSL may use simple FORTH or ODE tools.

1.1.3 Real-Time Aspect of HMSL. HMSL is both a real-time and non-real-time environment. The language can be used for computations which are not immediately heard, but in general, HMSL is designed as a "WYSIWYH" environment -- "what you see is what you hear". Events and forms at any hierarchical level can be edited while they are being computed and heard. In addition, the generalized stimulus-response environment (see *Perform* below) can interact with any other part of the system in real-time, so that events and information generated externally to HMSL may be easily integrated into the environment. All computational processes may be executed in real-time, simultaneous with editing and performance processes, but of course, time deformations may occur if a routine is too consumptive of CPU time. HMSL incorporates several scheduling techniques for dealing with these deformations.

1.1.4 Morphological Experimentation. One of the main theoretical and musical intents of HMSL's design is to provide an environment for experiments in musical *morphology*. By a *morphology*, we mean any set of ordered points (not necessarily in time), where points might refer to musical data ranging from the simple (points in a waveform, parameters of melodic events) to more complex (hierarchical information about other morphologies, or arguments to morphological transformation routines). The data structures and tools of HMSL are optimized for use in the creation and transformation of morphologies.

In HMSL, morphologies ("*morphs*") may be created, transformed, and rearranged at any hierarchical level. A morph's "points" are most often a list of pointers to other morphs. Trees of any depth may be created, and of any complexity. Most morphs may contain most other morphs (in HMSL, called "parent" and "child"). The design of each class of HMSL morph has a different musical motivation, so that even though most can be nested interchangeably, the ways that different morphs *execute* are quite different.

1.2 HMSL Organization. HMSL has three main modes of operation: *Create*, *Perform*, and *Execute*.

1.2.1 Create. *Create* is the real-time editing mode of HMSL, organized into various screens. At present, there are three such screens: the *Shape-Editor*, the *Action-Screen*, and the *Custom User-Screen*. One more screen, the *Hierarchy Editor*, is part of the HMSL design, but because of its experimental features, is not yet included in the current release of HMSL.

The *Shape-Editor* provides the capability of editing any dimension of any *shape* in real-time. Shapes are the "lowest-level" morph in HMSL (see the section on HMSL Data Structures below), and the meaning of a shape's values are arbitrarily assignable by the user. Thus, the Shape-Editor provides real-time access to almost any level of the system (for example, a shape's data might be the mean values of a given parameter, like duration, in a list of higher-level morphs). The shape-editor has most standard editing features (cut, copy, paste, insert, replace, zoom and pan) as well as some features specifically related to morphological transformation (inversion, retrograde, transposition, and windowed randomization).

The *Action-Screen* allows the user to edit the configuration of the HMSL *Perform* environment (see below). In this screen, users may turn on or off stimulus-response *actions*, alter the priority of execution of these actions, enable or disable the PERFORM environment itself, and turn on and off the *MIDI parser* (an HMSL utility for interpreting MIDI input).

The *Custom User-Screen* provides the user with access to the HMSL graphics library for creation of their own screens.

The *Hierarchy Editor* will provide the user with the capability to create and edit morph hierarchies, execute and stop execution of a hierarchy, and store specific hierarchies to disk. It will be included in the next release of the system.

1.2.2 Perform. *Perform* is the HMSL generalized stimulus-response module. It makes use of objects called *Actions* which allow users to define arbitrary stimulus-response events. Actions may interact with HMSL in any way. They may affect other morphs, turn other actions on or off, or directly access any part of the hardware or software. Actions have several features specifically designed for the Perform environment: local and global counters, procedures for their initialization and termination, prioritized execution, and simple protocols for the definition of stimuli and responses. Actions are executed by an HMSL morph called the *action-table*, which is itself scheduled and executed by the HMSL *polymorphous executive* (see *Execute* below).

1.2.3 Execute. *Execute* is the part of HMSL responsible for executing morph hierarchies in real-time. This part of the software is referred to as the *polymorphous executive (PE)*, in that it is capable of reliably scheduling, executing, and keeping track of any number of morph hierarchies, each nested to any level. It is the most complex aspect of the system, and fully use the object-oriented programming techniques of message passing in communicating up and down morph trees.

The PE only deals with "time" at the very lowest level of the HMSL morph hierarchy -- the two data structures called *Players* and *Jobs* (see Data Structures below). The elements of these latter two morphs have specific durations, and the PE is responsible for seeing that they are accurately scheduled and executed.

Users may significantly alter the concept of time itself in HMSL in several ways: by directly affecting the system clock, or by choosing between the two types of scheduling available (independently selectable for a given morph): *epochal* and *durational*. In epochal scheduling (absolute time), events are executed at specific absolute times, and if a time deformation occurs, the system will "catch up" to that time. The user may specifiy a "too-late window" for events, which tells the PE to ignore events that are too late (by the user specified time value). In durational scheduling (relative time), the time of occurrence of a given event is computed strictly from the time of occurrence of a previous event. These two scheduling types may be used concurrently in the system -- one morph might be durational and another might be epochal. In addition, the type of scheduling used by a morph may be software selected by any other part of the system (for example, actions).

Since most morphs have their own "execution intelligence" (see Data Structures below), the PE is mainly concerned with keeping track of the morph trees, and executing the "next" morph when the "previous" morph is done. Often, decisions about the sequence of morph execution can be quite complex, but that decision making process is internal to the morphs themselves.

1.3 Virtual Device Interface (VDI). The HMSL VDI is designed to separate the intelligence of the system from the specific form of its output. The output of HMSL may be used to control MIDI, analog sound generation, local sound of its host computer (as is the case with the Macintosh and Amiga implementations), graphics and video, special purpose hardware, to generate numeric output for conversion to traditional music notation, or for anything else the user might define.

The VDI is comprised of user and system defined *instruments*. HMSL provides users with a set of tools for designing these instruments, as well as some standard instruments themselves (for MIDI and "local" sound on the Amiga and Macintosh). Instruments have their own intelligence, including the ability to *translate* HMSL generated morphological data to the specific form needed by devices. In this way, HMSL may be thought of as a "compositional data engine", whose output is completely customizable by users through the VDI.

1.4 HMSL Data Structures (Morphs)

1.4.1 Shapes. *Shapes* are n-dimensional sets of points, whose values have no inherent meaning inside the shape itself. Shapes are raw data used by other morphs. Many of the *methods* defined for shapes are concerned with morphological transformation, and most can be used interactively in the shape-editor. Shapes may not be executed, but must be put in *players*, which interpret their data to the VDI.

1.4.2 Collections. *Collections* are the basic hierarchical morph, from which all other morphs are defined. Collections may contain as "children" most other morphs, including *jobs, players, productions, structures, tstructures*, and other collections. Several methods are defined for collections which are inherited by other morphs. The two most important of these concern the use of a *repeat-count* (specifying how many times to execute the collection) and a *nodal weight* (a value associated with the collection often used for determining likelihood of execution by a "parent" morph).

There are two types of collections: *parallel* and *sequential*. *Parallel collections* execute their children simultaneously. *Sequential collections* execute their children one after another. Parallel and sequential collections may be nested inside each other in any way, to form any possible type of tree.

1.4.3 Structures and Tstructures. *Structures* are collections with an added capability for execution intelligence, called a *behavior*. Behaviors are user defined routines which determine the sequence of execution of a structure's children. Any morph that can be put in a collection may be put in a structure (including other structures). Several default behaviors are defined by the system, and user defined behaviors may be of great complexity.

Tstructures are structures with an added *nodal tendency grid*, which may be used by the tstructure's behavior to determine likelihood of execution of one child after another (like a simple first order transition probability matrix). However, this grid may be used by the tstructure (or any other part of HMSL) in any way the user wishes. There are methods defined for editing that grid as well.

1.4.4 Productions. *Productions* are collections which do not contain other morphs, but which contain user defined routines as their "children". These routines are executed when the production is executed, and take control of the system until they are finished. Productions are typically used to create or transform shapes in real-time, but they may have a wide variety of other uses. In general, they are the utility by which users may imbed their own software inside a hierarchy. Productions may be put in collections or structures to be executed hierarchically.

1.4.5 Jobs. *Jobs* are productions which have a *duration* associated with their function. A job thus has a concept of time, and is scheduled repeatedly by the PE until it is turned off (by itself, or some other part of the system). Jobs may be put in collections or structures, but, like productions, may not contain other morphs. They may however, communicate directly with the VDI by calling instruments (see *players*, below).

1.4.6 Players. *Players* are jobs with an associated instrument, and associated shapes. The player is the morph used most typically for interpreting shape data in time, and sending that data to the VDI (or more specifically, the instrument in use by that player). Players, like jobs, may not contain other morphs, but may be contained in collections or structures.

1.4.7 Actions. *Actions* are the basic morph of the stimulus-response environment (see Perform above). They are not typically used inside any other morphs.

1.5 Current and Future Implementations. HMSL Version 3.1 is currently implemented on the Commodore Amiga and Apple Macintosh Plus computers. Plans exist to implement HMSL on more powerful workstations (like those of Sun Microsystems, the Apple Macintosh II, or the NeXT machine).

Significant additions to the standard system in the future will include implementation of morphological metrics (distance functions between morphs), "concept spaces" (Rosenboom's term for the organization of high level morphological transformation trajectories, described in Section 2), more robust editing (including the aforementioned hierarchy editor, and a tstructure editor), and support of emerging standards and protocols for signal processing, sampling, and software synthesis files.

2. Notes on HMSL as an Intelligent Instrument

2.1. Forms for representing musical knowledge. HMSL encourages the user to treat musical entities as morphologies such as shapes, units that represent parametric contours. The HMSL data structure design described above does not attempt to adapt itself to a priori notational or stylistic conventions. This facilitates an open-ended experimental environment. The design of HMSL presupposes that for experimental purposes, the use which may be made of a representation is more important than the representation's form or notation.

HMSL's data structures tend to be representative of a cognitive model of music along the lines of Anderson's (1983) "tri-code" model for the ACT programs. *Temporal strings*, which encode a set's linear order, are reflected in shapes, collections, structures, and the other non-"intelligent" morphs. *Spatial images* are implemented in the various hierarchical configurations, like collections of collections, or shapes whose multi-dimensional parametric data represents higher level musical forms. *Abstract propositions* in HMSL can be easily defined by productions, actions (which may interact), behaviors, or jobs, or in more complex definitions of relatedness between morphs (like *morphological metrics*, or the *concept spaces* described below).

The cognitive modelling focus is based on the idea that powerful musical software tools should support individual composers' creation of abstract musical models (see Rosenboom, 1987c). We hope that this approach to software will promote individuality in computer music. One way that this is achieved in HMSL is in the design of HMSL instruments which mirror or, in fact, animate the cognitive models of music on which the works are based.

2.2. Real-time editing, shapes as high-level data streams. The HMSL *shape-editor* can be a powerful tool for live performance, especially when data is used for directing high-level processes. Users may directly sculpt values which are used as arguments by productions, behaviors, or actions; or these values may serve as deterministic or probabilistic outlines of compositional macro-structure. Even at lower levels, HMSL's editing screen can be used in performance for direct manipulation of musical patterns in effective ways. Rosenboom has made extensive use of this capability in the work, *Systems of Judgment* (Rosenboom, 1987), in which data was used to direct processes in a sampling instrument, (Emulator II+HD), and Polansky (1987) has devised multi-dimensional "instruments" in this screen to simultaneously and precisely control nine parameters of a commercial signal processor (Roland DEP-5) in a live performance work entitled *Cocks Crow, Dogs Bark...*

2.3 Intelligent and heuristic systems of actions. HMSL's Perform environment allows for the creation of a wide variety of stimulus-response mappings. Direct, fixed stimulus-response mappings are easy to implement, as is the less obvious notion of an *action network*, where actions can activate each other's stimuli in patterns which fan out in tree-like arrangements so that simple actions can cascade into complex responses. Other actions can activate and deactivate various nodes along this network facilitating or inhibiting the cascading process. Polansky's work entitled *Simple Actions* uses the perform environment in a way inspired by Minsky's "society of mind" — a multitude of simple but interactive intelligences (actions) produce complex larger results. Further ideas being explored for improvising instruments involve heuristic learning networks. In this case, actions are combined with tstructures and behaviors, whose tendencies and weights influence how actions call each other. Other actions provide reinforcing or inhibiting feedback to alter the tendencies and weights, directing the action network towards more and more learned responses.

2.4. Dynamic generation of musical hierarchies. Rosenboom's work in progress, *On Being Invisible II*, uses this feature of HMSL. In this piece, significant changes in a musical output stream are analyzed for there ability to evoke event related potentials (ERPs) from the human brain. The results are used to segment the musical output and create real-time instantiations of a structural hierarchy using HMSL's collections and structures. The statistical trends seen in the ERPs are extracted and used to provide feedback to the heuristic action network described above.

2.5 Productions, interpolators, melisma and ornament generators. This process emphasizes the use of *productions* and *behaviors*. Productions may be defined which interpolate between elements in a shape by inserting predefined contours, also defined as shapes, between those elements. These can be prototypes intended to add ornaments or melismas from a limited, precomposed set to an otherwise unadorned sequence. This process requires that the end points of the inserted shapes be scaled so as to connect elements in the object smoothly together. Behaviors can be used to choose melismas or ornament shapes according to probabilities or other algorithms. Finally the whole process can be interactively turned on and off in performance by actions.

2.6. Parametric Shape Transformations. Three recent compositions by Rosenboom *Champ Vital (Life Field)*, for violin, piano, and percussion, (1987), *Zones of Influence*, five movements for percussion and computer music system, (1984-6), and *Systems of Judgment*, a large dance score for electronic and acoustic sources, (1987), all involve systems of parametric shape transformation with HMSL procedures. Some of these processes are also implemented with structures designed for improvisation with computerized instruments. Three examples of these types of processes are described below.

2.6.1. "Origin" to "Target" evolution by stochastic trajectories in a *concept space*. This process involves the evolution of a shape from an *origin* to a *target*. A combination of sliding scale factors and stochastic methods are used to produce a series of mutated or bent shapes which sound less and less like the origin and more and more like the target.

First, two input shapes are entered, either via the shape-editor, typing, or by reading MIDI input using the HMSL *midi-parser* , from a keyboard or other gesture capturing device, (like the recently introduced "Air Drums"). These shapes become the origin and target. Specially defined HMSL actions instantiate shapes in repsonse to stimuli, and call productions to fill the shapes by a mutation algorithm. The shapes are placed in a collection, and are output via an HMSL player to an HMSL instrument. The scale factors are then updated and the process of mutation and output repeated.

Rosenboom's mutation algorithm is based on the following equation:

$$A_i = | O_i S_o + T_i S_t + G_r D_f S_r |$$

Where,

A_i = the ith element of the mutated shape

O_i = the ith element of the origin shape

T_i = the ith element of the target shape

G_r = a Gaussian distributed random variable

S_o = origin scale factor

T_o = target scale factor

S_r = randomness or mutation scale factor

and D_f is a difference function computed thus,

$$D_f = (\ln | A_i - A_{i-1} + 1 |) D_s$$

where D_s is a user suplied difference function scale factor.

The difference function is included so that the amount of the random variable included in the computation is sensitive to the relative disjunctness of the shape in a particular region. Relatively smooth regions will tend to stay smooth, and relatively disjunct regions will be subject to more unpredictable variations.

Various mutations can be created by plugging in values for the various scale factors. In order to create a sequence of such mutations a cycle of Gaussian weighted scale factor changes is used. The Gaussian function,

$$G_f = ((1/\sqrt{2\pi})e^{(-.5g^2)}) \, 2.506633 \, Gs$$

normalized for a maximum amplitude of 1, is evaluated for values of g as follows: S_o, the origin scale factor is computed for g ranging from 0 to -3.2, a decreasing function; S_t, the target scale factor is computed for g ranging from 3.2 to 0, an increasing function; and S_r, the randomness scale factor is computed for g ranging from -3.2 to 3.2, an increasing then decreasing function. *Gs* and D_s are input by the user at run time according to taste. Other HMSL shapes could be used to store functions representing scale factor changes as well and then may be edited in real-time.

The resulting mutated shapes are then compared to the origin and target shapes by means of the standard linear correlation function,

$$C = \frac{N\Sigma xy - \Sigma x \Sigma y}{\sqrt{[N\Sigma x^2 - (\Sigma x)^2]}\sqrt{[N\Sigma y^2 - (\Sigma y)^2]}}$$

where x and y represent the corresponding elements of two shapes of length N.

This correlation is sensitive to the "up and down" qualities of the shapes' contours, though high correlation values may not always accurately reflect the notion of two shapes "sounding alike". An interesting example relating to musical perception involves the way this correlation measures inversion. Two shapes which are mirror inversions of each other produce a correlation coefficient of -1. Perceptually, one could argue that two shapes of this kind are more closely related than two shapes with a positive C but less obvious correspondence in contour. The absolute value of C may be used if one considers musical inversion as a perceptual invariant. Polansky's (1987) work on *morphological metrics* explores some other measures of shape transformation, and many of these have been experimentally implemented in the HMSL environment.

After the correlation coefficients of a series of pairs are computed, they are plotted in a *concept space*. This plots a "trajectory" of transformations moving from origin to target. The coordinates of the space measure correlation to the orgin and target shapes and the distances between points provide an approximate representation of similarity. The perceptual qualities of a given trajectory are then considered to be those of a high-level shape, which may be then edited in HMSL as a separate musical entity. The concept space in HMSL is more specifically a collection whose order is determined by the above correlation function. Many such collections can be created, all with different qualities, and stored in another collection (at a "higher" hierarchical level) for performance. Actions may activate the mutation process, select or modify scale factors, cause scale factors to be selected from other HMSL shapes, output shapes, or read in new shapes.

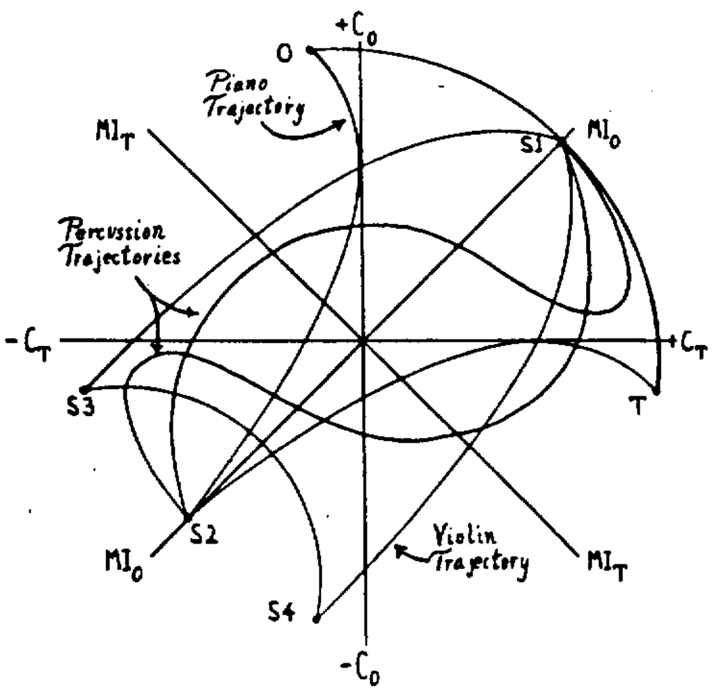

Figure 1: Visualization of a 4-dimensional *concept space* used in the composition of David Rosenboom's *Champ Vital (Life Field)*, for violin, piano, and percussion (1987). A counterpoint system is generated as transformed *shapes* are produced which lie along the trajectory lines shown for each instrument. The figure is repositioned in the space to outline trajectories which occur in various sections of the work. C_o= correlation with *origin shape*, C_t = correlation with target shape, MI_o = mutual information with origin shape, MI_t = mutual information with target shape, O = origin shape, T = target shape, and S1, S2, S3 and S4 = other anchor shapes for various trajectories. Note that negative correlation values result from interval direction inversion while mutual information values are always positive.

2.6.2. Non-linear transformations as generators of contrapuntal forms. The second method of transformation involves the use of functions implemented with HMSL *translators*, user-definable functions which accept a value and return another value according to a function or a lookup table. Transformation functions can be developed which generate various forms of contrapuntal variation or ornamentation. In *Champ Vital (Life Field)*, (Rosenboom, 1987), twenty or so translators are devised which produce melodic variations like interval augmentation, interval inversion, various types of contrary and warped contrary and similar motion, interval size complementing, range splitting, scale quantization, and ornamentation. These melodic transformations are combined with the mutation processes described above. The results are then maped into a four-dimensional concept space in which two new additional axes represent a value referred to as *mutual information*, a concept from algorithmic information theory. The mutual information content of two patterns, A and B, is defined to represent the extent to which knowing A helps to calculate B, (Chaitin, 1979). The mutual information content of a new pattern, generated by means of a translator, is plotted with respect to the origin and target shapes as described above. To simplify things, the mutual information of two sequences, MI(A:B), is in this case produced by rank ordering the transformation functions used to generate one pattern from the other according to their complexity, ie. MI(A:B) = $I(A) - I({}_AT_B)$, where T is the translator which transforms A into B.

These transformation functions are used in four ways for melodic sequences, 1) pitch height of A determines index into the transformation function with that function value determining the pitch height of B, 2) pitch difference (interval size) of A determines index with the function value determining the pitch height of B, 3) pitch difference of A determines index and function value determines the pitch difference (interval) of B, and 4) pitch height of A determines index and function value determines the pitch difference of B.

Once the four-dimensional concept space is determined, a plot like that shown in Figure 1 is created. This plot shows some aspects of the overall form of *Chant Vital (Life Field)*, (Rosenboom, 1987).

A very interesting variation on the above process involves imbedding the translators inside a feedback loop where the transformed shapes are fed back as input shapes to the same translators. The results of these processes tend to group themselves into several catagories. One kind of feedback process explored in the above mentioned trio involves relatively simple transformation functions with small "bumps". These bumps tend to replicate themselves internally in smaller and smaller structures over several iterations of the process, producing a kind of melodic ornamentation. Simple curves tend to become sharper (increase in curvature). A second type produces a kind of formal "flip-flop" (reciprocal process), often producing alternate inversions of interval direction with sharpening linear curvature. A third type of feedback process, using relatively complex translation functions, produces intricate inner growth of self-similar patterns while preserving the outer macro-form of the transformation function. All three of these feedback systems can ususally be initiated by simple seed functions.

An important performance consideration lies in the fact that the transformation functions used by the translators can be edited in real-time in HMSL. Again, actions, productions, and behaviors may select shapes and translators and direct output.

References

Anderson, J.R.,*The Architecture of Cognition*, Harvard University Press, Cambridge, 1983

Burk, P., "HMSL - An Object Oriented Programming Language", *Robocity News*, Volume III, Issue 4, May 1987

Chaitin, G., "Godel's Theorem and Information", *International Journal of Theoretical Physics*, 22, pp. 941-954, 1982

Cox, Brad J., *Object Oriented Programming: An Evolutionary Approach*, Addison Wesley Publishing Co., 1986

Polansky, L., "Morphological Metrics: Introduction to a theory of formal distances", paper given at the International Computer Music Conference, 1987, Urbana, Illinois

Polansky, L., "Distance Musics I-VI", (contains *Simple Actions*) in *Perspectives of New Music*, Special James Tenney Packet (Polansky and Rosenboom, guest editors), Volume 25 I/II, Spring, 1987

Polansky, L., "The HMSL Experimental Intonation Environment", in *1/1, Jrnl. of the Just Intonation Network*, Vol. 3 #1, Winter 1987

Polansky, L., Burk, P., Marsanyi, R., and Hays, D., *HMSL Version 3.1 Programmer's Manual*, available with HMSL software, Frog Peak Music, Oakland, 1987

Polansky, L., *Cocks crow, dogs bark ...*, (for three performers, HMSL, and signal processor), Frog Peak Music, Oakland, 1987

Polansky, L., Rosenboon, D., "HMSL (Hierarchical Music Specification Language), A Real-Time Environment for Formal, Perceptual, and Compositional Experimentation", in the *Proceedings of the International Computer Music Conference*, Simon Fraser University, Vancouver, 1985, published by the Computer Music Association

Rosenboom, D.: "On Being Invisible", *Musicworks* 28, , Toronto, Summer, 1984

Rosenboom, D., *Zones of Influence*, five movement work for percussion, computer music system and auxiliary melody instrument parts, David Rosenboom Pub., Piedmont, CA, 1984-86

Rosenboom, D., *Systems of Judgment*, 65-minute work for a variety of electronic, acoustic, and computer instrument sources, made for the dance and sculpture work of the same name, David Rosenboom Pub., Piedmont, CA, 1987a

Rosenboom, D., *Chant Vital (Life Field)*, violin, piano, and percussion, David Rosenboom Pub., Piedmont, CA, 1987b

Rosenboom, D., "A Program for the Development of Performance Oriented Electronic Music Instrumentation in the Coming Decade: 'What You Conceive is What You Get'", in *Perspectives of New Music*, Vol. 25 I/II, 1987c

June, 1987