HMSL
(Hierarchical Music Specification Language)
A Real-Time Environment For Formal, Perceptual And Compositional Experimentation

Larry Polansky and David Rosenboom
Center for Contemporary Music
Music Department
Mills College
Oakland, California
USA 94613

ABSTRACT

HMSL is a real-time, non-stylistically based music composition and experimentation language for the creation and execution of hierarchical morphologies. Integral to the definition of the language are certain fundamental concepts drawn from perception and cognition theory, and from the authors' previous work in musical artificial intelligence. Among these are: the notion of "distance" between shapes, or **morphological metrics;** "tendential and nodal" apsects of movement between successive morphologies and within morphologies themselves; the hierarchical nature of morphological perceptual and compositional processes; a "Virtual Device Interface", which assumes a generalized data structure and links the compositional intelligence to sound producing hardware of various types; strict definition of data structure types, including those that are more appropriately called **productions** which themselves produce morphologies when executed.

HMSL consists primarily of three modes, **CREATE, PERFORM,** and **EXECUTE. CREATE** is a graphics based, primarily mouse oriented environment for defining data, behaviors, structures, and hardware states. **PERFORM** is a highly general stimulus-response system which allows the user to define actions, or events, in a predefined logically structured stimulus-response syntax, or in the native language of the machine. The user may also utilize pre-defined events (like system clocks, enabling or disabling actions, or reading data, from and sending it to hardware devices) to control the execution of morphologies or to link the system to performance oriented input structures (like a motion sensor, or pitch follower). **EXECUTE** is a "polyphonic" morphology executive, which determines the order and nature by which data structures are "sent" to the Virtual Device Interface, which in turn determines the eventual sound producing hardware destination. **EXECUTE** contains the intelligence for determining the nodal and tendential behavior of a given morphology, or given point in a morphology, and may or may not use **PERFORM** to determine the stimulus-response aspect of events. All three modes function simultaneously, so that the data structures, hardware states, and stimulus-response environment may be edited during execution. In this paper, a typical student session with HMSL is described.

HMSL (Version I) currently resides on an S-100, MC68000 based system with full graphics capability, generalized I/O (including high resolution DAC and ADC), and a variety of digital and analog sound producing peripherals, including a six-voice digital waveshaping oscillator system. The entire system is part of the Mills College Center for Contemporary Music Hybrid Studio, and facilities exist for the system's communication with other devices there, such as the **TOUCHE** keyboard based digital synthesizer, a variety of input devices, other small computer and hybrid synthesis systems, a fully equipped 8-track recording studio, and a VAX 11/780 Berkeley UNIX BSD 4.2 system in the Mills Academic Computer Center. **HMSL** (version 1) is nearing completion of its implementation, and has already been used in compositions by staff, graduate composition students, and visiting artists. It also serves as an invaluable pedagogical and research tool at the CCM.

1. THEORETICAL, HISTORICAL AND MUSICAL MOTIVATIONS

1.1 THEORETICAL AND MUSICAL

1.1.1 Hierarchical Notions of Musical Structure The design of **HMSL** is based in large part on the authors' and others' long-standing interest and research in hierarchical models of musical structure (see Tenney 1961, 1977; Polansky, 1980, 1983, Rosenboom, 1976, 1978; Tenney with Polansky, 1980). Essentially, we are concerned with models of musical form in which the individual segregative and cohesive components and procedures are generalizable to various temporal and perceptual levels. Much evidence exists that this model of for the perception of structure strongly reflects the way that our cognitive processes make formal distinctions, both statistically (that is, regardless of the ordering of the input stream), and morphologically, in which the order of the input stream is considered along with the values of the data. The approach taken here depends on the assumption that many of the perceptual processes that distinguish musical entities on one level (like the grouping of a series of short events into a "phrase") are archetypically the same as those which group larger units (like "phrases" into sections). Tenney has explored these notions in detail (Tenney with Polansky, 1980), but mostly in terms of the statistical grouping of musical events into a formal representation. In **HMSL**, one of the main concerns is with processing and creation of complex morphological data.

1.1.2 Morphological Transformations In **HMSL**, we have defined a **morphology** as a set of values with an associated ordering. The transformation of one morphology into another, is a process that we consider to be fundamental to musical perception and creation. Much of the language deals with investigating the nature of these morphological transformations. In addition, morphologies in **HMSL** are hierarchical in nature: one "shape" might really be an ordering (by some predefined order operation) of lower level such shapes. Implicit in this latter concept is the notion of a **morphological metric**, or a single-valued distance function for any two metrics. With these operations, morphologies themselves may be considered as single points in a metric space, one that is definable by the composer or experimenter. In the design of **HMSL**, much attention has been given to the way in which the composer may design her own morphological metric spaces, and thus define the transformational space in which musical processes may occur. David Rosenboom has referred to these large scale orderings of morphologies as "concept spaces" and has used them extensively in the recent work, Zones of Influence (Rosenboom 1984-5; see also Rosenboom 1984b). In this multi-movement work for computer and percussion, Rosenboom has used correlation functions to map the distances between successive transformations of a source morphology to an origin and a target, and used the computer to plot "trajectories" through the "concept space".

Correlations are one example of a morphological metric which is non-order preserving, where a rearrangement of the set will not give different results for the function. Another example of that class of metrics might be a measure of similarity, suggested by Tenney and others, defined as the union of two sets divided by their interesection. Order preserving metrics are more difficult to define and less common. One example is a metric which might be called a "transition matrix" metric, in which the distance between two morphologies is a measure of the intersection of two matrices in which the individual cells are -1, 0, or 1 depending on the direction of change between all values in the morphologies. This metric reduces magnitude information completely (unlike the correlation), but preserves an important measure of comparison between the "pure" shape (up or down) of the two morphologies. A further extension of this would be to consider the first order difference functions of the two morphologies in like manner. In addition, any of these measures might be combined, with appropriate weights to reflect the desired perceptual significance, to produce a meaningful, highly useful, and experimental notion of musical transformation. Classically, these processes have played a fundamental role in composition, from early fugal forms to the limited transformation processes of serial technique. **HMSL** attempts to afford the composer with ability to study the different effects of distance functions upon compositional ideas.

1.2 HISTORICAL MOTIVATIONS

Several other factors, not all strictly theoretical, have influenced the language design. One such is the attempt to create a composition environment which is, as much as possible, non-stylistically based, or at least not fundamentally motivated by a desire to imitate certain historical compositional procedures. We are interested that the language should afford the composer the facility to recreate a given sytle or historic method, should (s)he wish to do so, by simply building upon the axiomatic constructs of the language itself. Thus, the nature of those axioms should not preclude the possibility of embedding other notions of musical form, including the more traditional ideas of "notes", equal octave divisions, and so on. **HMSL** contains no reference to these types of ideas, nor does it have any predefined relationship to conventional input structures (like the keyboard), but it does allow the user to define these in a rather straighforward manner. It would be possible, for example to define a "front-end" for the language in which the data structures were represented in conventional notation, and subjected to the morphological and stimulus/response procedures in the same way as more abstract data. As we will explain below, "attaching" arbitrary input structures to the language is rather simple as well, and several different ideas have already been tried.

It is of great importance to us that the language reflect as little as possible musical styles and procedures that have already been implemented — like conventional musical notation — and that it begin from the standpoint of rather primitive notions of form derived from cognitive and perceptual experimentation and evidence. We hope that from these ideas and their implementation, a set of new musical styles might evolve, along with their accompanying theoretical and linguistic implications.

2. HMSL DESIGN AND IMPLEMENTATION

2.1 INTRODUCTION

HMSL consists primarily of three related program modules, **CREATE**, **PERFORM**, and **EXECUTE**, each of which reflects a different type of user interaction and program operation (see Figure 1). All of these run simultaneously, but each may be disabled or enabled by the mouse. This enables the system to run in various "configurations" depending on the user's immediate need . For example, if no predefined data structures are needed, and the user does not need to do any editing of hardware states or the stimulus/response configuration, both **CREATE** and **EXECUTE** can be switched off so that **PERFORM**, the stimulus/response event processor, will run at top speed. Since **HMSL** is imbedded in a large scan loop, it has the advantage of being able to function in any or all of its capabilities at any time. It has the disadvantage, at present, of being slowed down at times by complex editing actions or stimulus processing. However, the significant use of parallel

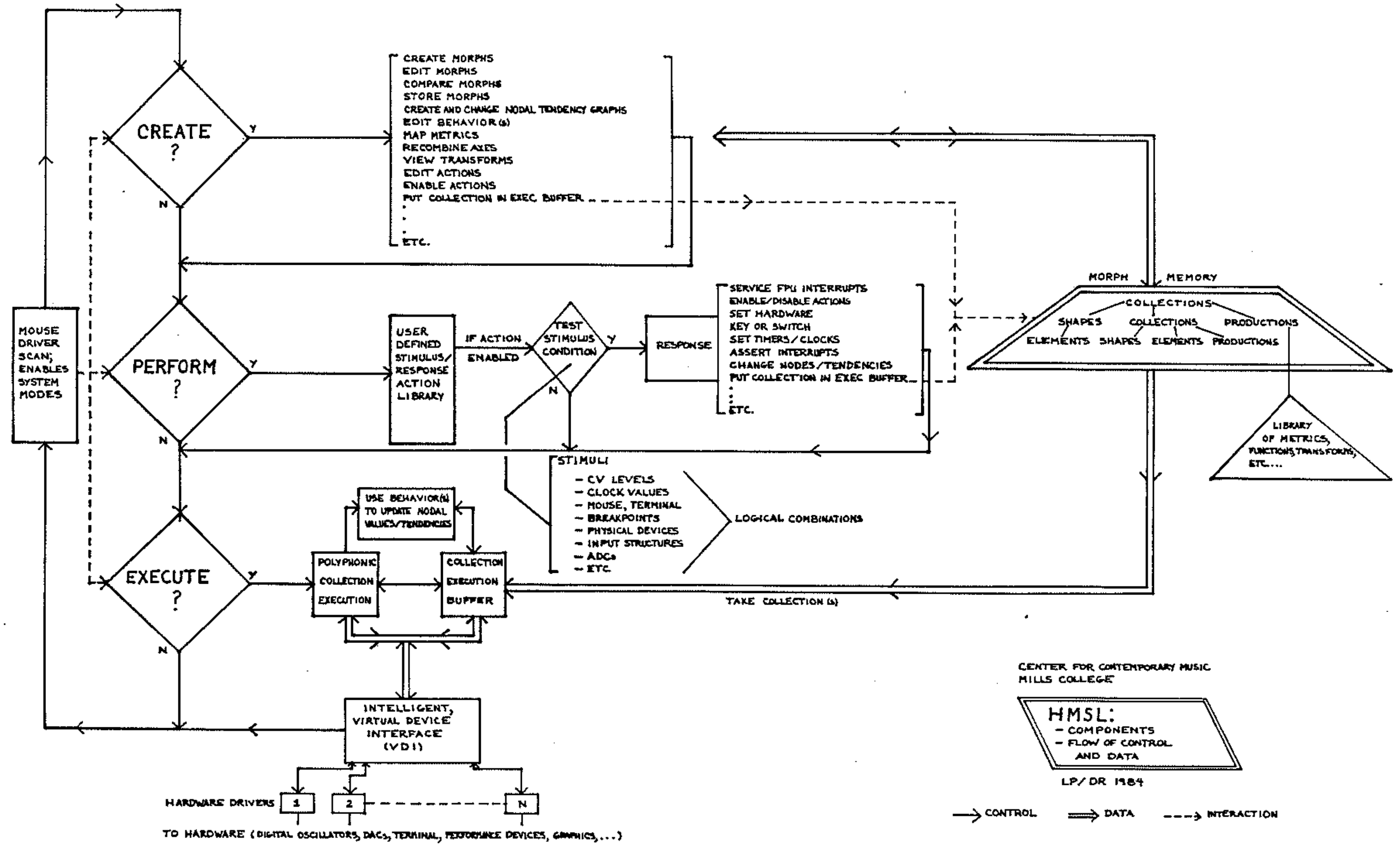

FIGURE 1: OVERVIEW OF HMSL

processers (like waveshaping digital oscillators and real time function processors for envelope generation) alleviate the need for the system to do much time-consuming and unnecessary computation. The future integration of interrupt controlled devices and parallel analog and digital signal processing equipment will also add greatly to its synthesis power and speed.

2.2 CREATE

CREATE is the mode of the system used for all screen oriented user interaction. It is primarily mouse driven, though terminal input is used sometimes for alphanumeric strings, such as when the user wishes to name a data structure. The mouse used at present has four switches, and a general protocol is used for switch functions, so that the user may quickly gain an intuitive sense of what to do even in unfamilar territory. For example, Switch #1 is generally used for performing an action or for selecting a menu option; Switch #2 is used to move **up** a level in the menu hierarchy; Switch #3 is used in general to initialize, redraw or clear a screen (like a waveform transformation table, or hardware state display); and Switch #4 is used to perform a "subsidiary" action to Switch #1 (like selecting a waveshaping table to be edited by Switch #1).

When **CREATE** mode is enabled, **HMSL** is menu driven, and all editing of formal and sonic parameters produce results in real-time. Some typical **CREATE** functions currently implemented are:

— draw or edit a waveform transformation table
— enable or disable system modes (**CREATE, PERFORM,** or **EXECUTE**)
— enable or disable a stimulus/response **action** (see **PERFORM**, below)
— change the values of any existing hardware function (pitch, spectra, timbral and frequency modulation rate and index, amplitude, stereo location, etc.)
— edit and create envelopes
— edit and create data structures, and (See Figure 2)
reconfigure nodal tendencies and weights (see **EXECUTE**, below)
— create and edit "concept spaces", through which morphological trajectories are plotted
— change the values of arbitrarily reassignable drawbars (called in the system "Analog Control") which may affect any large or small scale value in the system

Since the main synthesis device at present (but not the only one) is a six-voice digital waveshaping oscillator, the system provides several methods for the creation and editing of waveform transformation tables. These methods are: draw an arbitrary waveshaping table, deform a sinusoid (represented as a diagonal line for the sinusoidal transfer function), the "rubber band method"

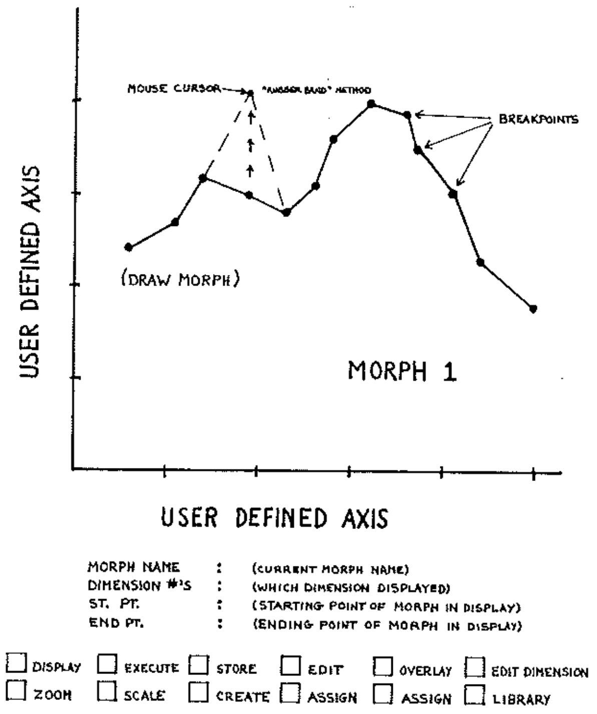

FIGURE 2: MORPH EDITING

(stretch the waveshaping table from two selected points to a third), and plot the spectra of the desired waveform. The latter method uses a Chebyshev transformation to compute the necessary waveshaping table for the desired spectral configuration, (the code for this routine was adapted from a program contributed by Martin Bartlett). The user can hear the effects of each of these processes simultaneously with their operation, and these routines serve as an invaluable pedagogical tool for the teaching of synthesis techniques, especially non-linear synthesis. In addition, the "knob-turning" character of this approach provides interesting and useful kinesthetic feedback when one is designing sounds, especially if other complex formal and timbral processes are occurring simultaneously. In this way, we have integrated much of the immediate control of more traditional analog synthesis systems with the power of a more intelligent and precise digital instrument. Of course, the contents of any of the six waveshaping tables may be changed easily under algorithmic or stimuli activated control. For example, natural sounds may be "sampled", processed, and the results stored in waveshaping tables in response to predefined stimuli. This often produces interesting (and still surprising) results. In addition, any of the six "voices" (FM and TM capable oscillators) may be assigned to any waveshaping table at any time by the mouse or other system control.

2.2.1 Other Editing Most other **CREATE** functions in the system are accomplished rather simply — by pointing the mouse and pressing a switch. Editing functions are a part of the system which are continually expanding and evolving to meet the needs and reflect the ideas of composers and users. Currently we are designing various ways to create envelopes and assign them to parameters. Envelopes may also be typed into the system in a simple way, as a two-dimensional array (value, time), and are arbitrary in length. In addition, a given envelope may be dynamically reassigned to any sonic parameter by the system either algorithmically or in response to stimuli (see **PERFORM**). Editing of data structures and Universal Behaviors will be discussed below.

The "Analog Control" capability allows the user to link any value in the system (pitch of an oscillator, temporal density of a sequence of events, probabilistic values, time values for an envelope, weighting of a distance function, rate of change for a piece, etc.) to any or all of eight arbitrarily assignable and mouse controlled graphic "drawbars". Thus, the user has the ability to "turn knobs" to control any value (s)he wishes, and the assignments or weighted effects of those "knobs" can change under system control or via the stimulus/response actions.

The **Action Table** (see **PERFORM**, below), may also be edited in a rather simple but useful fashion. The names of all defined actions (stimuli and responses) appear on the screen when the **Action Table** is selected. By pointing and switching the mouse, actions are enabled and disabled so as to have real time control over the event processing configuration of the system. For example, if the pitch of a group of oscillators is responding to algorithmic control as well as the input pitch of, say, a marimba, one might easily enable and disable the input from the marimba in real time via the **Action Table** to significantly alter the musical results.

2.3 PERFORM

In **HMSL**, the stimulus/response environment of the system is defined as a set of **actions**, which are routines written in a combination of the resident computer language (in this case either high-level or assembler) and a predefined syntax which allows the user the ability to create simple actions easily without having much programming expertise.

The syntax for an action definition is:

> ACTION::=id <STIMULUS LIST> <LOGICAL OPERATORS> IF <RESPONSE LIST> ELSE <RESPONSE LIST> THEN ;

Thus, the user may use any of the predefined stimuli (like reading an ADC, the pitch or other parameter of an oscillator, a harware or software clock value, trigger, or mouse position), test the value, and execute a predefined response (send a value to a voice, to the DAC's, set a clock, enable or disable another action, fill a waveshaping table, execute a data structure, begin an envelope, etc.). The conditional tests are done in the high level or low level language of the machine. It is important to note that these actions may also be defined as complex machine procedures if the user so wishes and has the programming ability, and may be embedded in the **action table** simply with the necessary **id.**

When the **PERFORM** mode of the system is enabled, the system scans the **action table** from the first to the last currently defined action. If an action is enabled (either from software or by the user), its stimuli are tested and, if conditions are met, the responses are activated. Thus, complex actions whose responses depend on the main processor for execution will significantly alter the speed of the system as a whole. We have evolved and are evolving various techniques, both in software and hardware, for avoiding this type of inconvenience. Generation of envelopes does not tie up the system in this way and, so far, stimulus/response actions have not created speed problems for the composers using the system. The **Action Table** contains several of its own software counters and hardware clocks. Among the current utilities availible to the user is the ability to set, increment, and decrement the clocks, turn other actions on or off, and send envelopes. All common hardware devices (ADC's, DAC's, IO Ports, mouse, graphics drivers, oscillators, function processor) are "brought out" as simple stimuli and responses to the **Action Table**, and are easy for the composer to use. The **Action Table** is currently capable of storing 512 actions at one time, and each action may consist of program code of any length. The 512 limitation is due to the current memory configuration and graphics screen size, and is easy to upgrade should the hardware configuration enlarge.

The possibilities of **HMSL's** stimulus/response architecture are of great interest to us, and are the subject of a considerable degree of experimentation. At the Center for Contemporary Music, one area of research is the design of input structures for intelligent music systems. The **Action Table** affords an easy to use and powerful tool for this experimentation. For example, a keyboard might be tied to the language by simply naming it as an action, and defining the response to different keys (in fact, this has been done with several acoustic instruments). MIDI devices are currently being interfaced to the system as well, as are user utilities for pitch and envelope following. A further area of research currently planned involves the use of biofeedback techniques with this system (Rosenboom, 1975 and 1984a).

2.4 EXECUTE

EXECUTE, the polymorphous data structure handler, is the last module of the system to be coded in full. The projected implementation date is Fall, 1985. **EXECUTE** is that module of the system in which data structures are sent to various devices for sonic or other realization.

2.4.1 Data Structures **HMSL** recognizes five types of data structure: **elements, shapes, collections, productions, and structures,** all generically called **morphs** (short for "morphologies"). They are defined as follows:

> — an **ELEMENT** is an n-dimensional set of values (ELEMENT:: =id <n-dim> <set of values>;)
> — a **SHAPE** is an n-dimensional array of given length (SHAPE:: =id<length, dimension> <set of values>;)

— a **PRODUCTION** is a user-defined or library routine which produces data (see below for details)
— a **COLLECTION** is an n-dimensional array, of a given length with a hardware identifier (VDI Op-Code). **COLLECTIONS** differ from **SHAPES** and **ELEMENTS** in that they may contain as values pointers to other morphs: **ELEMENTS, SHAPES, PRODUCTIONS** and **COLLECTIONS.** (COLLECTION::=id<dimension> <length> $(ELEMENT|SHAPE| PRODUCTION| COLLECTION) <VDI OP-CODE>;)
— a **STRUCTURE** makes a **collection** which is executable by the system. Distinguishing it from a **collection** is a list of **link tendencies** from any component **collection** in the **structure** (node) to any or all of the others, and a **nodal weight** for each of its component **collections.** The default **structure** for a **collection** is a set of sequential links from each node to the next (STRUCTURE::=id<COLLECTION><NODAL LINK list>; NODE::=<MORPH pointer><Nodal weight> <Behavior code> <set of links to other nodes>; LINK::=<NODE pointer> <LINK Tendency>;)

Elements are the simplest morphs, roughly corresponding to "notes" in conventional terminology. They are distinguished from **shapes** only in their length restriction, and this distinction is made less for theoretical than for practical purposes. Neither an **element** nor a **shape** may be executed by the system.

Collections are the basic mechanism of heirarchy in **HMSL**, in that they may include other **collections** (or, in fact, themselves, in a recursive manner) which may in turn include others. At some point, however, nodes in a **collection** must eventually be data, either in the form of **shapes** or **elements.** Each **collection** is assigned upon its creation to a given existent hardware device through the Virtual Device Interface Op-Code. This is software which interprets basic information and routes it to the appropriate hardware. This facility of the system allows, we believe, for a high degree of portability, since that part of the language is the only place where the specific requirements (both software and hardware) of various devices and their drivers are known. The VDI requires such data as length, dimensionality, time references, and stimuli to execute a **collection.**

Productions are user definable algorithmic routines which generate data. **Productions** have the following generalized syntax:

> PRODUCTION::=id<input data/parameter passing> <pattern/condition matching> <procedure>; (N.B. This procedure may include compilation style creation of other data structures as well.)

The notions of metrics described above are included in the definitions of these morphs in that the parameter passing input may take the form of a set of **shapes,** and the "procedure" may be a metric ordering function, creating a new morph which is the "concept space" of the input morphs under the metric. **Productions** are written in the "native" language of the machine implementation (either high or low level code), and are treated as morphs by **collections** in that they have nodal weights, and are routed by the executive to an appropriate device.

Structures are the highest level morph, and are the data structure in which the concept of "directed execution" is embedded in the system. Each component **collection** of a structure has two things associated with it: a **nodal weight** and a list of **link tendencies** pointing to other **collections** in the structure. It is these values which determine in what order **collections** in a **structure** are executed, and in turn other morphs. No other morph but **structures** may be passed to the executive. Note that device information is held more locally, in the **collections,** to allow for greater output flexibility.

These weights and tendencies constitute a modelling facility for the specification of dynamic musical behaviors and perceptual strategies. The links and nodes are all under algorithmic control, so the activation of given stimuli or **production** may affect them, and dynamically alter the large scale musical form being heard in complex ways.

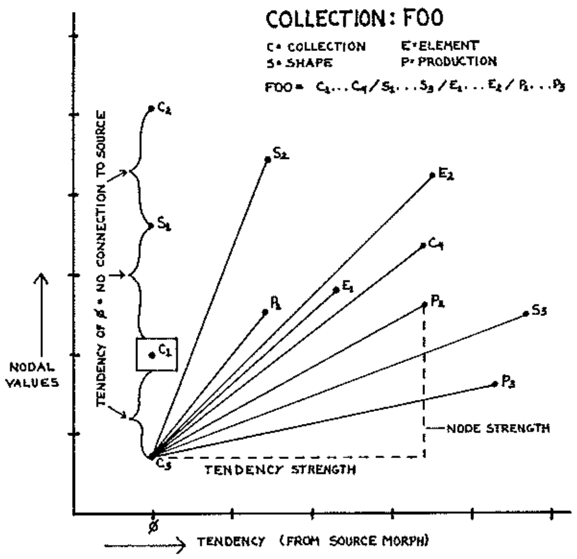

FIGURE 3:

NODAL WEIGHT and LINK TENDENCY EDITING

Graphic utilities in **CREATE** for the editing and defining of **structures** are currently being implemented. Figure 3 shows one model for formulating such an idea. In this figure, the x-axis represents the **link tendency** from a given morph to the morph under consideration (in this case, a **collection** named FOO). The Y-axis represents the more "absolute" **nodal weight** of the morphs in the graph, irrespective of the "source" morph FOO. Note that these morphs' relations to the Y-axis would be identical for all source morphs within this **structure.** This graph shows one way a user might quickly define the large scale form of a piece, a section of a piece, or some sonic environment, by simpling moving morphs around in the graphic space. A solid line between the source morph and another indicates a **link** (whose magnitude is its X-value). A morph which is directly vertical to the source, but not connected by a line (like C1, S1, and C2 in this example) is a morph whose **link tendency** to the source is 0 (which means that it is impossible for that morph to sequentially follow the source morph). A morph's Y-value is its **weight.** One rather unusual ramification of this graphic approach is that morphs which are graphically most distant from the source are actually those whose **link tendency** implies that they are nearest to it in execution (for their tendency value is the greatest!). Experimentation with this facility may suggest inverting the X-axis display to make it more intuitively representation of our concept of structural distance between morphs.

<u>**2.3.2 Polymorphous Execution**</u> The performance executive action of **HMSL** is rather simple. The user instructs the system to "execute" a **structure** (whether via the terminal, graphically, or under software or stimulus control), and that morph is placed in a circular stack and its values sent to the appropriate hardware. The execution stack is theoretically infinite in depth, but memory and speed limit it significantly. When a **structure** has been completely executed it is removed from the stack. The polymorphous executive keeps track of several things:

— which **structure** is being executed
— the order of **structures** in the stack
— its current position within each **structure,** requiring a significant amount of nesting, since morphs are inherently hierarchical
— the values of relevant system clocks and hardware values (updated in **PERFORM**) to control execution of values
— the next **collection** in a structure that must be executed.

This last is calculated as an orthogonal sum of the **nodal weights** and **link tendencies** from the current **collection.**

With this method, it is clear that most, if not all **structures** will continue to execute until they are removed from the execution stack by some stimulus. That is, there will always be a "next place to go", as defined by that **collection** in the **structure** with the highest combination of **nodal weight** and **link tendency.** However, one last characteristic of the system, called **universal behavior,** needs to be discussed. **Universal behaviors** are a set of routines which affect the **tendencies** and **weights** of **collections** in a pre-defined, system-wide manner. Initially, the design of **HMSL** allows for just a few of these as system routines, but plans for user definable behaviors exist.

Two illustrative examples of **universal behaviors** would include what we have been calling "resonant" and "decadent" behavior. In the former, each time a **collection** is called, its **weights** and **tendencies** increase proportionally to the other **collections** in its **structure** so that eventually no other morph may be executed — much in the way that feedback gradually "takes over" an audio path if encouraged to do so. "Decadent" behavior would be one in which the likelihood of a given **collection** being executed decreases each time it is called. When there is no possiblity of moving to another **collection** in a **structure,** that morph is removed from the stack. Another, possibly obvious, example would be stochastic procedures wherein the path taken through **nodes** in a **collection** may be the results of certain probabilities calculated within the system in relation to **weights** and **tendencies.**

Initially, the user will be able to choose between 16 such behaviors during the system's real-time operation, effecting the musical form in that way. In addition, **productions** and **actions** may affect the rate of change of a given behavior (how resonant or decadent), or may switch from one to another. These parameters may also be placed under "drawbar" control if desired. Later on, a syntax for user definition of her own **universal behaviors** will be established and implemented.

3. CURRENT HARDWARE, SOFTWARE, and STUDIO ENVIRONMENT

<u>3.1 HARDWARE</u>

HMSL currently runs on an S-100 based ERG MC68000 microprocessor, running at 8 MHZ. The S-100 bus also includes:

— 320 kb. memory
— interface card for two 8" floppy disks
— 512 x 480 pixel display, monochromatic graphics card, with on-board Z-80 MPU
— two I/O cards, with various highly programmable parallel and serial ports and real-time clocks
— 32 channels of high resolution ADC, and 4 channels of high resolution DAC
— interface card for digital oscillator system

Also on the system are a terminal, graphics monitor, dot matrix printer, two floppy disk drives, and optical mouse. Current sound output devices include a six-voice, pipelined, multiplexed digital oscillator which implements an advanced, non-linear waveshaping synthesis algorithm (designed by Buchla and Associates). This algorithm allows efficient, relatively inexpensive generation of an extremely rich timbre domain for sound creation requiring relatively little host processor overhead to keep it running.

<u>3.2 SOFTWARE</u>

The MC68000 CPU card contains a bootstrap version of FORTH-79 in ROM, and this serves as the underlying operating environment. The CCM expanded operating system contains a fully implemented assembler, extensive programmer utilities and math functions, primitive level and user accessible drivers for all system hardware, and a graphics language. **HMSL** is written in 68k Assembly language and FORTH, and as development proceeds, more and more of the FORTH code is being "translated" to assembler to acheive greater execution speed.

3.3 STUDIO ENVIRONMENT

The **HMSL** system currently resides in the Center for Contemporary Music Hybrid Studio, which contains a wide variety of analog and digital devices, as well as teaching facilities. At present, four other computer systems exist in this studio which are are useful in the **HMSL** environment:

— a **TOUCHE** keyboard based digital instrument, which contains similar synthesis equipment to **HMSL.** The **TOUCHE** has a considerable software base, including **FOIL-83** (Far Out Instrument Language), and MFOIL (Meta-FOIL), designed by David Rosenboom and, plans exist for interfacing this instrument to **HMSL**

— **PATCH-IV**, a hybrid control language, which uses the same function processing unit that exists in the **TOUCHE** and is similar to that used with **HMSL**

— a VAX/UNIX link, which we plan to use to provide data for **HMSL**, in very large applications.

— an HP-6400A logic development station, with 68000 emulation capabilities, C compiler, software performance analysis, and logic timing and analysis modules. This system is useful for **HMSL** hardware and software development, and will be as well in VAX/68000 communications

The studio is also well-equipped with analog digital signal processing, conditioning, and synthesis equipment, (more and more of which is MIDI compatible) including envelope followers and a Gentle Electric Pitch Follower (distributed by Serge Modular). High quality mixing, recording, and monitoring facilities exist as well (quad and stereo), and the studio is adjacent and linked by audio and control lines to the CCM multi-track recording studio (which also contains several outboard signal processing devices) for more extended recording projects.

3.4 LATER VERSIONS

Plans currently exist to write several other versions of **HMSL** for other, more common systems. The design of **HMSL** emphasizes hardware independence, via the VDI and the heavy reliance on data types as a determinant for the language. The next implementations will probably include one for the Apple Macintosh (much of the code is already in 68000 assembler), using the SUMAC development environment, a version in C for the VAX 11/780 at Mills (UNIX BSD 4.2)

3.5 CURRENT DEVELOPMENT

HMSL is running, as of June, 1985. However, certain sections, like a full version of EXECUTE and some of the more advanced editing ideas (nodes and tendencies, Universal Behaviors) are presently being coded. The system is fully usable in the studio environment, and has been a valuable educational resource as well. The operating system and CCM system utilities have been fully documented by Larry Polansky and John Levin of the CCM (with additional assistance from Richard Zvonar), and are on UNIX and availible to students. Documentation for **HMSL**, also on UNIX, is nearly complete.

A "working" version of the language has only been available to students and composers for a few months, but several works have so far been produced. Two graduate thesis concerts made significant use of it (including Jin Hi Kim's Su Wol Yong Yul for computer, harpsichord and 'cello), and several other students have composed and/or experimented with the system. CCM staff have also written and worked with the system and one invited composer (Ron Kuivila) made extensive use of the system (including contributing in interesting ways to the software development) in a concert at Mills in March with CCM staff member Larry Polansky.

HMSL has been used in all of the electronic music classes at Mills (both undergraduate and graduate), and has been the subject of several public demonstrations and seminars. We expect use of the system to increase exponentially in the current school year, since the language is far more developed and the documentation has progressed well. In addition, we hope and believe that students and guest composers will contribute to the conceptual and implementational development of the system in a variety of ways.

References

Clynes, M. (Ed.). (1982) **Music, Mind, and Brain.** Plenum Press, NY.

Deutsch, D. (Ed.). (1982) **The Psychology of Music.** Academic Press, NY.

Fu, K.S.. (1974) **Syntactic Methods in Pattern Recognition.** Academy Press.

Minsky, M. (1979) "Music, Mind and Meaning". Computer Music Journal, 3/4.

Polansky, L. (1980) "A Hierarchical Gestalt Analysis of Ruggles' 'Portals'". Proceedings of the 4th Intnl. Computer Music Conference. Northwestern University Press.

Polansky, L. "Morphological Metrics", seminar given on Mills Seminar in Formal Methods Series, 1980, and as Mills Faculty Colloquia, 1983.

Rosenboom, D. (Ed.). (1976) **Biofeedback and the Arts: Results of Early Experiments.** A.R.C Pub., Vancouver, B.C.

Rosenboom, D. (1976) "A Model for Detection and Analysis of Information Processing Modalities of the Nervous System through an Adaptive, Interactive, Electronic Music Instrument". Proceedings of the Music Computation Conference. Univ. of Ill. Press.

Rosenboom, D. (1984a) "On Being Invisible". Musicworks 28.

Rosenboom, D. (1984b) "Study for ZONES". Musicworks 28 (audio cassette).

Rothenberg, D. (1976) "A Non-Procedural Language for Musical Composition", Proceedings of the Music Computation Conference. Univ. of Ill. Press.

Shepard, R. (1962) "Attention and the Metric Structure of the Stimulus Space". Bell Telephone Labs Technical Memorandum. 10/19/62.

Shepard, R. (1962b) "The Analysis of Proximities: Multidimensional Scaling with an Unknown Distance Function". Psychometrika, 27, 125-140.

Tenney, J. (1961) **Meta + Hodos — A Phenomenology of 20th Century Music and an Approach to the Study of Form.** Privately circulated ms.

Tenney, J. (1977) "META Meta + Hodos". Journal of Experimental Aesthetics, 1. A.R.C Publications, Vancouver, B.C.

Tenney, J., with Polansky, L. (1980) "Hierarchical Temporal Gestalt Perception in Music: A Metric Space Model". Journal of Music Theory, 24/2.

Thompson, D. (1952) **On Growth and Form,** Cambridge University Press.