Using Simple Controls to Manipulate Complex Objects: Application to the Drum-Boy Interactive Percussion System

by

Fumiaki Matsumoto

B.A., Mathematics and Computer Science
Cornell University, 1989

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Media Arts and Sciences

at the Massachusetts Institute of Technology
September, 1993

© Massachusetts Institute of Technology 1993. All rights reserved

Author______________________________

Fumiaki Matsumoto
'rogram in Media Arts and Sciences
August 6, 1993

Certified by______________________________

Tod Machover, M.M.
Associate Professor of Music and Media
Information and Entertainment Group, MIT Media Laboratory
Thesis Supervisor

Accepted by______________________________

Stephen A. Benton, Chairman
Departmental Committee on Graduate Students

Rotch
MASSACHUSETTS INSTITUTE
OF TECHNOLOGY
OCT 18 1993

Using Simple Controls to Manipulate Complex Objects: Application to the Drum-Boy Interactive Percussion System

by

Fumiaki Matsumoto

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
on August 6, 1993 in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Media Arts and Sciences

Abstract

Design issues in developing an interactive performance system are discussed. The development of Drum-Boy, an interactive percussion system for non-experts, is documented, along with discussions of features that make such a real-time creative environment effective. Concrete examples of problems encountered during the design and development phase of Drum-Boy are investigated. Specific issues include the user interface, modes of interaction, and the methods to characterize and transform drum patterns. An "agent" approach is used to construct descriptive transformers and their complement, analyzers. These tools allow for the modification of rhythmic material by employing adjectives, giving power to non-expert users who can recognize and describe (but not build from scratch) the music they like.

Thesis Supervisor: Tod Machover
Associate Professor of Music and Media
Information and Entertainment Group, MIT Media Laboratory

This work was supported in part by the Yamaha Corporation.

Using Simple Controls to Manipulate Complex Objects: Application to the Drum-Boy Interactive Percussion System

by

Fumiaki Matsumoto

Thesis Readers

Thesis Reader [illegible]

Pattie Maes, Ph. D.

Assistant Professor of Media Arts and Sciences

Learning and Common Sense Group, MIT Media Laboratory

Thesis Reader [illegible]

Rosalind W. Picard, Sc. D.

Assistant Professor of Media Technology

NEC Career Development Professor of Computers and Communications

Perceptual Computing Group, MIT Media Laboratory

Acknowledgments

Drum-Boy was a group project which was started in 1991 and lasted exactly two years during my stay at MIT. Demos in Japan made the deadlines unyielding. There were at least as many all-nighters as there were deadlines, but the presence of others working on the project made those moments memorable. Many at the lab, particularly the music group, made my stay here fun and educational. In particular, I would like to thank the following:

Joe Chung, for getting me started in the Hyperlisp environment and supporting me throughout the project, coming up with lots of good ideas and implementing them.

Cas Wierzynski, for all the sick jokes and sick code.

Alex Rigopulos, for sharing an office and a house with me without driving me crazy. (And creating the high level transformers and helping me with demos.)

Mike Wu and Eric Metois for asking me questions that prompted me to think further about my thesis.

Suzanne McDermott, for always being on top of things.

Andy Hong, for showing me parts of MIT that I didn't know existed.

Kako, for moral and culinary support.

Yamaha Corporation and their researchers for maintaining the enthusiasm in their support of the project.

My readers Pattie Maes and Roz Picard for their comments.

And above all, I am grateful to my advisor Tod Machover, not only for all the valuable insights that helped shape the thesis into its final form, but for introducing me to a different world. There was much to be learned from his insistence on excellence.

Contents

1 Introduction

The goal of the Hyperinstrument Research Group at the MIT Media Laboratory has been to create real-time music performance systems that give performers enhanced expressivity through the use of technology. Since 1987, the group has worked on several projects and created different hyperinstruments. Most hyperinstruments developed so far take an existing instrument as the controller. A computer carefully monitors a performer's gestures by some combination of the following:

- looking at the MIDI stream produced by the instrument
- using unobtrusive sensors such as a bow position sensor on a cello
- performing digital signal processing on acoustic sounds produced by the instrument

These hyperinstruments were developed for expert performers, whose every small gesture during a performance is deliberate. Much effort was put into the monitoring process to capture those subtle expressions. An important aspect of hyperinstruments is that just as with traditional instruments, a performer can improve by practicing. Hyperinstruments are tools for expression, not tools for automation.

From 1987 through 1991, most hyperinstruments were programmed to accommodate specific musical compositions, during which the relationship between the gesture and computer output changes. These mode changes were either triggered by the performer (the triggers are usually explicit in the score) or the computer operator (who would also be following the score). Storing the score in the computer meant that the performer could control the vast amount of musical material, as if to conduct an orchestra.

Storing the score also meant that the system could not be used for performing another piece without reprogramming the software. Customizing a hyperinstrument for another piece involves the following: loading sound samples, or sound fragments, into computer memory; specifying triggers for the mode changes; redefining mode-specific parameters (which are not accessible to performers) to algorithms that establish the computer interaction; and programming sequences, or musical phrases, which are generated by the computer. Making all these modifications to the system requires work by computer experts. The lack of generality of the environment in which the system can be used and the skills required by the performer severely limit the applications of the hyperinstruments.

The goal of our most recent research project is to develop a working model of a general purpose instrument that addresses a larger audience. The actual test-bed for the new model is a project which was started in 1991: Drum-Boy, a percussion hyperinstrument. It is intended for professional musicians, as well as amateur musicians who lack the training necessary to physically master an instrument but who can both recognize and describe the material they like. The interface for such an instrument should be different from that of traditional instruments. In traditional instruments, each gesture shapes a note (or chord in the case of polyphonic instruments). The precise ability to control the envelope and timbre of sounds allows skilled musicians to use subtle nuances to communicate. However, for a novice musician, the concentration required to produce the right notes often reduces the attention that should be given to controlling more musical aspects of the performance.

The software portion of Drum-Boy has undergone several revisions, and the prototyping project is now considered complete after two years of work. While several possibilities to enhance the system will be discussed in a later chapter, we believe that most fundamental issues have been addressed.

The Drum-Boy system consists of a MIDI keyboard input device, a Macintosh Quadra 950 computer running Drum-Boy software in the Hyperlisp environment, and a drum tone generator. The participants in this group project are Tod Machover (originated the hyperinstrument project, and is the project leader of Drum-Boy), Joe Chung (created the Hyperlisp environment, and worked on the "player model" portion of the instrument and implemented many other features), Casimir Wierzynski (helped write the feature-space analysis code), and Alex Rigopulos (helped construct the intelligent pattern transformer instances).

My involvement with the project concentrates in the following areas:

- developing the methods for characterizing drum patterns (See section 2.5.3)
- designing the user interface and setting up the query system for the database search (See section 2.1 for general discussion of the user interface, and Appendix A, Drum-Boy User's Manual)
- implementing a system for using adjectives to manipulate patterns (See section 2.6.2)

This thesis will discuss and analyze each of these problems in detail. In characterizing drum patterns, I chose to define a feature-space that is suitable for describing rhythms. In implementing the query system, I applied the feature-space characterization and used "navigation" controls in the feature-space to find the desired database entry. In developing the descriptive manipulation system, I explored two methods, the interpolation method and the more general, transformational, method.

The issues that were encountered during the development phase can be extended and applied to other systems that involve the manipulation of complex objects. While specific techniques may differ, I believe that a system similar to Drum-Boy could be very effective in other domains. In the case of a musical instrument, the techniques could be applied to develop a "hyper-bass" or "hyper-piano." By extending the system to accommodate images, even a presentation or video story-boarding system could take on a similar architecture.

1.1 Interactive Music Systems

Robert Rowe defines interactive computer music systems as "those whose behavior changes in response to musical input" [Rowe92]. Not explicit in this definition is the notion of mutuality; the definition suggests a unidirectional force of influence from the performer to the system. To qualify as an *interactive* computer music system, the system must affect how the other agent (presumably a person) behaves as well.

Rowe suggests a three-dimensional categorization of such systems:

- Score-driven vs. Performance-driven
- Transformative vs. Generative vs. Sequenced
- Instrument paradigm vs. Player paradigm

These parameters lie on a continuous spectrum as opposed to a discrete, quantized scale. Furthermore, the parameters can take on a range of values, and the value may change over time. For example, an interactive musical system that is primarily score-driven at the beginning of a piece may evolve into a less score-oriented, more performance-driven system.

A score-driven system has preprogrammed knowledge of the score, or musical text, of the piece. It may or may not have musical fragments that deterministically get played at specific parts of the piece. Such a system is characterized by a score-following or score-orientation mechanism, and typically has mode changes that are sequential.

A performance-driven system has no knowledge of the piece; therefore, it does not require specialization for each musical piece. The sequential mode changes are either replaced with random access mode changes (mode changes can be triggered in any order) or the idea of mode change itself is disposed of. Traditional musical instruments operate in this domain.

The second parameter describes how musical output is produced. A *transformation* takes musical material and modifies it using some algorithm (which can either be static or dynamic). *Generation* means that the material is not recorded or stored, but is algorithmically produced. *Sequencing* employs no algorithms; the material is preprogrammed, and the output is identical to the original input.

The third parameter defines the role of the system. On one end of the spectrum is the instrument paradigm, in which the system assumes a passive role. In this scenario, the interactivity of the system is somewhat diminished since the extent to which the system influences the human player is small, since in an ideal instrument, the player should be able to anticipate the output of an instrument. On the other end of the spectrum is the player paradigm, where a virtual persona is created within the system. The computational musician changes its behavior depending on what the human player does.

I would like to add another axis to Rowe's categorization scheme. A system can either behave deterministically or non-deterministically. This factor can have artistic implications as well as more practical ones (for example, how does one replicate a performance using a non-deterministic system?). This design decision of choosing one over the other is a choice between control and serendipity.

The third category (instrument versus player) illustrates the most characteristic feature of all four. Systems lying far apart on this axis seem very different. While it is easy to modify a system to change the characteristics on the other axes, this property seems to be intrinsic in the basic design. A player system can not be modified into an instrument model very easily.

The following are some examples of existing interactive music systems, laid out on the third axis:

On one extreme, George Lewis has created a system using the player paradigm. His goal in the project has been to build a musical personality into the program. Even though the output is influenced by external forces, it can not be explicitly controlled once the program starts running. The creation of the system itself is a work of art, as much as is the performance of any of its particular musical instantiations.

David Wessel's work in the improvisational setting moves one step towards the instrument paradigm [Wessel91]. His system has a listening assistant component, whose task is to record and analyze the accompanying acoustic performance. Composing assistants intelligently transform the analyzed material into different material. For example, chords can be generated from melodic phrases. Performance assistants monitor the performer's gestures and translate them into output. Exactly where this system would be placed on the Instrument/Player scale depends on the sophistication of the performance assistants and the composing assistants used. The more sophisticated the performance assistants are, the closer the system is towards the instrument paradigm; the more sophisticated the composing assistants are, the closer the system is to the player paradigm.

Tod Machover's hyperinstruments [Machover92] have been closer to traditional musical instruments. The hyperinstruments are not designed to mimic the human perception or intelligence used in generating or performing musical material. His emphasis has been to create an environment where human performers can give superb performances. However, his systems differ from traditional instruments because a traditional instrument will fit in Rowe's three-dimensional space as a performance-driven system in the instrument paradigm. (The second coordinate will not apply in the case of traditional instruments because the feature is realizable only by having the resources to compute and store information.) Hyperinstruments to date have all been tailored for specific pieces; they are score-driven.

In developing Drum-Boy, the goal was to create a tool in which an improvisational performance as well as a compositional process would be aided. It is modeled after the instrument paradigm. To have it work in a variety of musical contexts, it is designed to be performance-driven. Our goal is not to equip the system with all the arbitrary tools that one can conceive; it is not to set up an environment and give the user/performer the ability to program every aspect of the system's behavior -- we believe this is the task of the "instrument" designer. Playing an instrument should not involve manipulation of symbolic objects or syntactic constructs. In developing our system, each time a technical problem was solved it became another milestone for our bigger problem: the *design* of tools for manipulating complex material using simple controls in a creative environment.

1.1.1 Overcoming Difficulty in Mastering Traditional Instruments

Jeff Pressing characterizes traditional musical instrument by its versatility, identity, and degree of development of the instrument [Pressing92].

> A highly developed instrument acquires highly developed special playing techniques, along with a dependable relationship between characteristic performance gestures and the sound produced... Many such instruments have a depth or power of expression that makes them capable of successful extended solo performance, like the violin or electric guitar. [Pressing92]

The cost of the versatility and precise control of the instrument is the difficulty in mastering it. Often, in the initial stages of learning to play an instrument, much effort is needed to produce the correct pitch. Such heavy mental load leaves no room for expressiveness for beginners.

Max Mathews tried to tackle this problem by introducing a new relationship between the performer, the musical score and the instrument. He noticed that in the traditional composer-performer relationship, the performer has no freedom in picking which notes to play (although there is some freedom in moving the pitch up and down within the boundaries of the note).

The performer does, however, have the freedom to interpret *when* the note is played, and to control the dynamics of the note. If the performer has no freedom of choosing the note, why give him the freedom at all? In Mathews' *Sequential Drum*, the score of a piece is stored in computer memory, and is played back one event at a time in predetermined order, as controlled by the performer. The performer has no control over what notes to play except that he will play the next note[Mathews80]. Without the added effort to control the unnecessary degree of freedom (i.e., playing the right notes), the performer can pay closer attention to control the shape and the timing of the notes.

1.1.2 The Instrument as a Mapping Device

Traditional instruments are constructed of several functional blocks. In addition to supplying the physical system that actually produces audible sounds, an instrument must act as a controller that translates a performer's gestures into an input to the physical system. In other words, an instrument *maps* the performer's action to a parameter that gets fed into a physical system.

The electronic circuit's ability to produce sounds evolved into several newer instruments, such as the electric piano and the theremin. In a computer system, the parameter extracted from a human gesture can be an input not only into a physical system, but into any system that is computable.

In a Hyperinstrument system, a computer collects performance data from the performer and analyzes it. Parameters that get extracted during the analysis are then mapped to some controllable parameters in the output. Since the output is controlled by the computer, there is no physical constraint to the mapping; for instance, the vertical position of the cello bow, which is used to control the timbre for a traditional cello, can be mapped to control the panning to create a stereo effect.

Computer analysis of the input can also recognize higher level performance gestures. For example, a performer can play a phrase that can then be projected onto a one-dimensional space with legato on one end of the scale and staccato on the other. The position on the scale can then be used as a parameter to control the timbre of the accompanying part.

Most of the Hyperinstrument systems developed to date have used a traditional instrument interface to control a computer process.[1] These systems required performers already capable of playing the established instruments well. They were designed for *experts*. In developing our percussion hyperinstrument, the use of an established instrument interface was not a constraint because a high level of skill or dexterity in the controls was not presumed, and in fact was not desirable.

We chose a MIDI keyboard as the input device because of the following reasons:

- a MIDI keyboard is ubiquitous
- many musicians have technical proficiency on the keyboard
- it supports the wide bandwidth that the system was thought to require (lots of buttons, can add pedals, sensitive to velocity, etc.)
- it is easy to play for non-musicians, i.e., no special technique is required (unlike a cello, in which generating a single note requires much practice)

1.2 MIDI

MIDI (Musical Instrument Digital Interface) is a protocol that permits us to interconnect musical controllers and tone generators from different manufacturers freely. The physical interface and the messages are standardized.

In the Drum-Boy system, the core component is a Macintosh program that interprets an input MIDI stream produced by the user on a MIDI keyboard, and then generates an output MIDI stream which is sent to a drum tone generator. By working at the MIDI level, a layer of abstraction is automatically made. MIDI works with note events. A MIDI note is composed of note-on events that signal which note was hit and how hard it was hit. Its complement, the note-off event, signals when the note was released.

Although the description of the pitch and velocity information is standardized in MIDI, one can not describe timbre using MIDI (except by using System Exclusive commands, which

[1]The exception is the DHM glove, used by the conductor in Machover's piece, *Bug-Mudra.*

are specific to each manufacturer of the instruments). There is no standardization for the mapping between timbres and notes. Moreover, in case of the drum, there is no standardized mapping between the notes and instrument. Therefore, the program has to keep track of the mapping between the MIDI note numbers and the drums to which they correspond. Drum-Boy keeps the mapping separately in a module, and adding another set of mapping is easily accomplished

As a result, Drum-Boy can be used with any keyboard that supports the MIDI standard (it must support some optional functionality such as velocity-sensitive keys, two foot pedals, two modulation wheel controllers, channel aftertouch[2], etc.). Any MIDI tone generator capable of producing drum sounds can be used, provided that the mapping is defined in the software.

1.3 Real-time control of drum machines

Typically, a drum machine has been used in a performance context in either one of the following two extreme modes:

1) <u>As a playback device for the "taped" rhythm section</u>. In this mode, the performer hits the play button on the drum machine at the beginning of the piece. The behavior of the drum machine during the performance has been deterministically pre-programmed, so the performer has no control over the material being played by the drum machine (except general controls such as changing the overall volume or stopping the playback).

---

[2] A key on a MIDI keyboard typically has one or two sensors:

- a sensor for *initial velocity* (or simply velocity), which detects how fast the key goes down (and optionally, how fast the key goes up - this is known as the release velocity). This control usually determines how loud the note gets played.
- optionally, an aftertouch sensor for detecting pressure exerted on the key after the key has been pressed down. This control is often used to shape the note, for example, by controlling the vibrato or brightness.

A MIDI keyboard with *channel* aftertouch capability has one aftertouch sensor for the whole keyboard. In effect, its output will correspond to the maximum pressure exerted to the keys.

2) As a tone generator. In this mode, the performer is usually equipped with a MIDI controller drum kit, that sends out MIDI messages when corresponding drum pads are hit. The performer controls each note explicitly. In this mode, the sequencing capabilities of the drum machine are ignored.

The obvious need for the capability of more sophisticated real-time control spawned new functionalities in commercial products. For example, the Roland TR808 has had a strong following among dance music performers for its capability of real-time programming. The performer can program the next measure of its playback by using the sixteen buttons, each corresponding to a sixteenth-note in the next measure. The performer decides which drum to program, and enters the pattern by pressing on the buttons whose corresponding sixteenth-note time slots will contain a hit. This crude interface is not ideal for complex interaction. The idea behind Drum-Boy has been to extend this concept of real-time control of drum machines.

2 Drum-Boy: Overview

Drum-Boy is a computer percussion instrument that maps incoming MIDI data (as performed by a musician) to outgoing MIDI data that forms complex rhythms on the drum set. Functionally, it is a superset of a drum machine; Drum-Boy addresses several additional problems that a user comes across when using a drum-machine:

- Drum machines are typically packaged with a small set of pre-stored patterns.
- Entering new materials into the pattern database requires the user to specify each of the drum hits explicitly.
- Typically, real-time control of the drum machine has been limited.

The first problem is harder to solve than it seems because loading the database with more patterns creates another problem. If the database is small, the user can simply listen to all the patterns and select one. However, given a large database, the user needs an inquiry system to find the pattern. Constructing such a system is difficult because unlike indexing words in a dictionary (which can be sorted in one dimension using a well-defined ordering scheme), indexing drum patterns involves defining a highly dimensional space based on perceptually salient features. Such a feature-space for drum patterns has not yet been established.

Organization of drum patterns into a multidimensional space has been attempted by at least one commercial system, the Roland R70 [RolandCorp92], but because unquantifiable and arbitrary features such as "feel" and "variation" have been used, finding a desired pattern is not always easy in the R70. In this system, the user specifies three parameters: Genre, Feel, and Variation. The system generates a set of drum patterns based on the values of these parameters. However, since the parameters (except for genre) do not correspond to perceptual features, continuity is not preserved when moving from one pattern to the next.

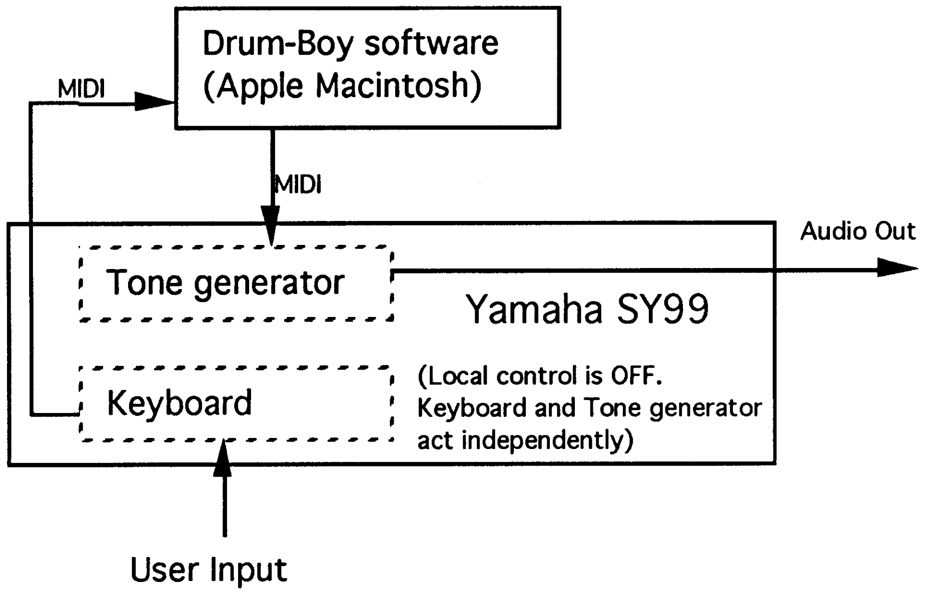
The first problem we addressed in designing Drum-Boy was this: to define a mathematical model of rhythm perception by using a multidimensional space and to set up a "navigation" system as the interface to the database search problem. A set of features that define the axes of the mathematical space was determined and the features' calculation methods were developed. By applying the set of calculation procedures to a pattern, a vector could be obtained, which would provide the coordinates of the point in the feature-space. Each database entry was indexed with its coordinates, so that a pattern corresponding to the point closest to another (calculated by the Euclidean distance in the space) could be retrieved. The navigation solution, however, evolved into a more general transformation system that does not require a large database. This system is discussed in detail in later sections.

The second problem, the need to explicitly specify all notes, is a limitation to those amateur users that we addressed as the main audience of the system. A solution for such a problem must free the user from specifying details that are often of little interest. In developing a solution, an important issue to address was: what level of controls should the user have? An excessively simplified control with much automation would lighten the user's work load, but at the same time would take away the sense of control. A small set of useful tools at various levels should be developed. (As an extreme example, imagine a computer painting program in which a user had to specify all the RGB values of each pixel! In such a setting, users benefit greatly by having higher level tools that allow them to manipulate larger objects. The users would also benefit by having effect filters that transform the picture.)

Our solution to this problem consists of several tools that work together within the environment: phrase-level editing and triggering of material, and high-level, descriptive transformations. The phrase-level editing features deal with changing certain aspects of a structure, for example, re-accentuating the hi-hat track. The phrase-level triggering functionality gives the user the ability to call up pre-stored rhythm fragments. High-level transformations allow the user to descriptively transform a pattern -- for example, transforming an existing pattern into something "calmer" (but retaining many of the characteristics in the original pattern).

The third problem is related to the second. A real time interface that requires the user to specify all notes explicitly would call for a drum kit and a skilled drummer. A novel interface in which the user is freed from specifying the details is suggested in order to capture the amateur musicians as an audience. The note-level and phrase-level triggering functionalities described above were refined to the level such that Drum-Boy qualifies as a performance tool.

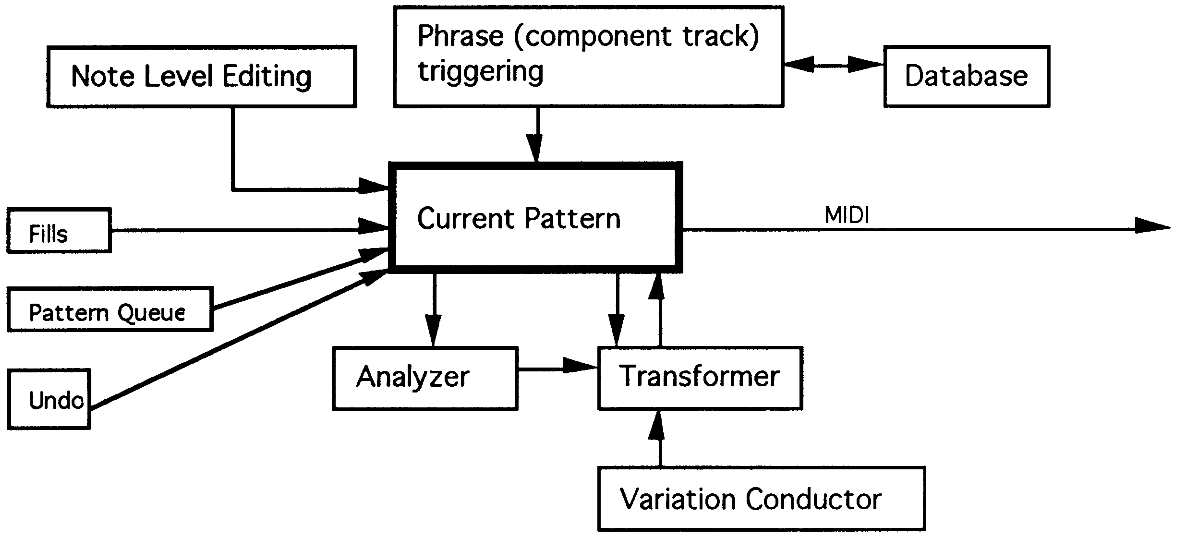
The hardware configuration of the developmental Drum-Boy system is shown in the following diagram.



Moving to a different environment with a different hardware configuration is easily accomplished. It involves remapping of the MIDI note numbers to different drum sounds. The remapping is necessary because there is no standard mapping between note numbers and the drum sounds in the MIDI specification. For example, for the SY99, a snare drum sound is triggered when it receives the note C#2; other drum machines will respond differently, or not respond at all.

Such a standard mapping exists in the General MIDI standard. The General MIDI mapping is prepared as an option.

The functional block diagram of the software is shown here:



While Drum-Boy is running, a *current pattern* is repeated (possibly with automatic variation) until a user tells the system to modify it. The user interface allows the user to interact with the system on one of three layers:

- Note-level event triggering and editing
- Phrase-level event control
- Pattern-level transformations

The goal was to create an instrument that has a natural instrument-like feel when it is being played. Some issues brought up during the design were:

- Physical interface
- Feedback
- Navigation: generative vs. transformational vs. pre-stored data search

2.1 Design Rationale

According to Carroll [Carroll et. al91], a design rationale is a detailed description of the history and meaning of an artifact. It can enumerate the design issues that were considered during a design process, along with dependencies on and implications for other issues, and the final decision that was taken. Such techniques have been used in understanding human-computer interaction technology. This section discusses such issues encountered during the design and development of Drum-Boy. Its purpose is to provide a case study, a source of information for those interested in pursuing future work in this field.

2.1.1 Selecting the Audience

In designing a system, it is essential for the designer to have an understanding of the representative user. Since previous Hyperinstruments were designed to take advantage of the skills offered by expert performers, one motivation behind the inception of Drum-Boy was to expand the territory. Some questions we had to address were:

- What additional requirements would we have to satisfy if we were to address novices instead of (or in addition to) the experts we are used to working with?
- Is it possible to design a system fit for both an expert and a novice? Or are their requirements so different that a separate system has to be developed for each?

What does it mean to be a beginner?

The electronic American Heritage Dictionary defines the word "novice" as "a person new to a field or activity; beginner." A novice can be identified or characterized by the following properties:

- lack of dexterity to perform a task
- lack of knowledge of relevant vocabulary and concepts specific to a field
- lack of the ability to adapt ideas in related but different situations

An expert, on the other hand, is defined as "A person with a high degree of skill in or knowledge of a certain subject." Experts can be characterized by the following:

- development of special physical coordination
- development of knowledge-base that is multiply indexable using several different key-words or key-concepts

Depending on the nature of the task, some results can not be evaluated objectively. One can evaluate how "good" a typist is by analyzing how fast and accurately he or she types. But how does one evaluate exactly how "good" a painter or composer is? Or for that matter, how good a painting or a musical composition is? When objective criteria do not exist, learning an "evaluation function" is part of the training process for becoming an expert. While the "evaluation function" itself is not taught explicitly, exposure to great works of art has been an important facet of art education.

In Drum-Boy, the "novices" we target are a specific type of novices. These beginners lack the physical dexterity to play the drums well. Furthermore, they lack the understanding of the instrument required to construct rhythm patterns that satisfy what's needed in the user's context (the music material that is to be played with the rhythm patterns). However, these novices can, to some degree, evaluate a rhythmic material's suitability in a context. The last characteristic is not one by choice; the present ubiquity of music has familiarized many with rock and Latin rhythms. Even the most "uneducated" can tap out a basic rock drum pattern.

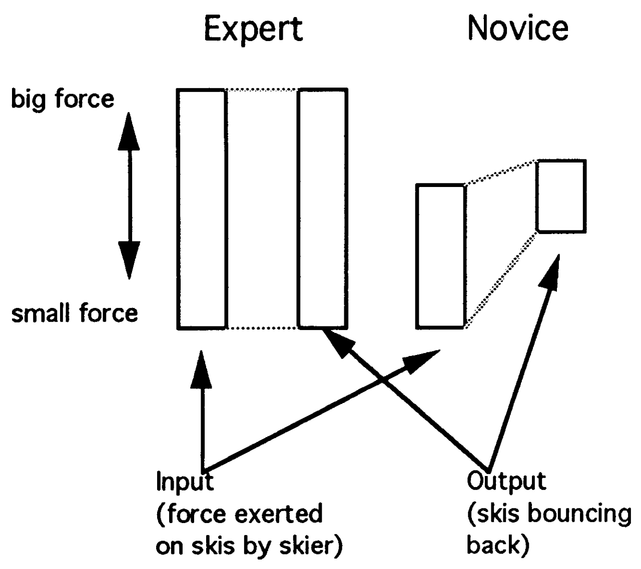
Tools for the Novice and the Expert

How are tools for the expert different from the tools for the novice? By looking at how existing tools differ between the models intended for beginners and those intended for experts, I attempt to extract the features that attract each group.

The following is a simplified explanation of the differences between skis for beginners and skis for experts. Typically, a beginner's skis have the following properties:

- inexpensive -> quality is compromised
- "soft" -> the dynamic range of the force in which the skis bounce back is compressed
- shorter and lighter -> less physical force is required to maneuver the skis; however, stability is compromised

1The following diagram illustrates the relationship between the skier's input (how much force the skier exerts on the skis) and the output (how much the skis bounce back).



The diagram suggests two general comments:

- the expert's tool must be able to handle a big range of input
- since the beginner's input is often imprecise to begin with, the information can be "compressed" or downsampled without loss of accuracy

In case of a painter's tool, a big difference between what experts use and what beginners buy is in the way the tools are packaged. A beginner typically buys a pre-assembled set of brushes or paint. While this arrangement effectively limits a degree of freedom, beginners do not feel that their expressivity is reduced because their attention and interest are usually elsewhere. It is more important for them to make the drawing look right. Only after they have acquired those drawing and painting skills will beginners need or want finer granularity in their selection of colors. Furthermore, the beginner lacks the experience to be able to deliberately select a color or tool with precision. *A deliberate choice can be a burden for a beginner.*

When a person performs a task, it could be an act of *using* a tool, or of *selecting* something (for example one can choose which brush to use). The two types of actions are illustrated in the above examples. The difference between them is that in using the tool, dexterity introduces error; whereas, in selecting something, the error (or lack of precision, or directedness) is attributed to inexperience or inadequate knowledge, since dexterity is irrelevant.

2.1.2 Design as an Iterative Process

Unlike other types of problem solving in which there are clear, correct solutions, design problems are often open-ended and have no best solution. The virtues and shortcomings of a proposed design solution are sometimes not visible until extensive user testing is performed. Design is an iterative process because analysis alone can not predict all possible scenarios that develop within the context for which the design solution is proposed. For years, human-computer interaction (HCI) researchers have tried to formalize the design process, but essentially all agree that the iteration process is necessary. Many recent papers in the field do not present any formalizations but indicate areas in HCI where formalizations can and can not occur, and why that is the case.

Discussing the design of Drum-Boy poses a similar problem. Because of the nature of any design process, much of the derivation is necessarily ad hoc or a result of a brainstorming session. This reality reflects the notion that design is not a science. However, I believe that by documenting the motivations for making each decision and employing post-design analysis, this thesis can be used as a valuable case study for the problem of designing an interactive system whose goal is to aid creativity.

Landauer discusses in his position paper [Landauer91] why results of most HCI research can not be directly applied to real-life design problems. He lists several "rules of thumb" that can be applied, but stresses that they have only limited impact.[3]

[3] Some of the simple rules he cites are:

- Fitts's law, or make sure to locate conveniently and make the size of the buttons big enough on the screen so that they are easy to actuate
- Hick's law, or hierarchical menus should be as flat as possible

Existing Principles

Norman discusses in [Norman88] that there are virtues in the "design of everyday things" that make them easy to use. Good design evolves over time. By analyzing what makes good design and what makes bad design in physical objects, we can apply some of these principles to design an interactive system.

The following are some principles that he defined. I will explain each principle by applying it in the context of the Drum-Boy interface.

Object Symbol

Some controls have the property that they themselves offer visual feedback. For example, when a physical lever is moved, the physical appearance of the device is changed at the same time the system state is changed. Norman named such a situation the "object symbol." Such a mode of operation was intrinsic in simple physical systems; however, with modern electronic systems, the controls and indicators no longer have intrinsic spatial or physical relationship. The useful source of information (which came for free in these simple physical systems) no longer can be taken for granted. The designer of the system must design an effective way of communicating the states.

The situation where a possible benefit of an object symbol disappears is when a momentary switch is used as a switch to toggle states. If a momentary switch such as the keys on a MIDI keyboard or on a computer keyboard is used as a toggle switch, confusion may occur since the key itself cannot convey the state information. A non-locking "caps lock" key without an indicator is a classic example.

In the Drum-Boy interface, most of the controls used are momentary switches (that is, the keys spring back to the original state as soon as they are released). The confusion possible with software toggle keys (like the non-locking caps lock key) is avoided by making most internal modes correspond directly to the state of the keys. As soon as all the keys are released, the internal state springs back to "normal" as well.

The only violations to this principle are 1) the "now-playing" versus "now-stopped" state, and 2) the auto-variation mode. The user can very easily determine whether the system is playing or stopped (since if the system is playing, rhythmic material can be heard, along with visible highlighting on the graphical feedback display). Auto-variation is a little problematic because the auditory cue of this mode is not as obvious, and the only indication of the state is in the graphical display. However, since the likelihood of the system being left in this mode for a long time is high, and the effort required to keep a key or pedal pressed is costly, the object symbol approach was abandoned for this parameter.

Naturalness of a mapping

Norman also claims that the "naturalness" of a mapping is related to the directness of the mapping, where directness can be measured by the complexity of the relationship between representation and value, measured by the length of the description of the mapping. When a mapping is not natural, experts derive new terms to simplify the mapping, increasing the "naturalness" at the expense of added computation in expanding the newly derived terms. In normal behavior, this degradation will not be noticeable, but under heavy workload, the added computation will degrade the performance of the expert.

In Drum-Boy, most mapping was intended to be "natural," using Norman's definition. For example, the harder the user hits a key, the more complex the rhythmical material will be. The position of the modulation wheel (a continuous controller) directly reflects the tempo of rhythm playback.

Forcing Functions

"Forcing functions are the design properties that use the intrinsic properties of a representation to force a specific behavior upon the person." [Norman88]. For example, by installing a plate as a handle for a door, one is forced to push the door, and an embarrassing situation of trying to pull on a door that is supposed to be pushed, can be avoided. Some cabinets are designed so that only one of its several drawers can be opened at the same time. This prevents the whole cabinet from falling over.

In Drum-Boy, the concept of forcing functions is used in limiting the possibility of getting something unfavorable as a result of a mistake. Human error is to be expected, and a designer's job is to prevent it, or to make the error non-fatal. For example, in Drum-Boy, the user can easily access a previously recorded drum pattern by pressing a single key. As it has the capability of dramatically changing the present state, it is favorable to design the system so that a pattern is not launched by mistake. Therefore, the keys to trigger patterns are spatially located far away from the keys that trigger drum events, the keys that are likely to be played constantly.

The use of an off-the-shelf component as the physical interface (MIDI keyboard) meant that we had to accept the interface *as is*. While this additional constraint made the design problem easier (every constraint is a step closer to a solution), it also meant that some shortcomings could not be fixed (for example, the situation of different controls having identical shapes is not ideal - but it is in the nature of a MIDI keyboard, and we can not fix it unless we use a different controller).

2.1.3 Tool for Reflective and Experiential Modes

[Norman93] classifies all human activities into one of two categories: the reflective mode and the experiential mode. The *experiential* mode is the mode of human activity where the subject passively or actively experiences something without deep thought. TV watching, for example, can be classified in this category. Note, however, that the term *experiential* does not imply that the activity be passive. Driving a car, for instance, is classified as an activity in the experiential mode since it does not require deep thinking.

The *reflective* mode, on the other hand, requires a different level of concentration by the subject. Some examples of activities in this categories include: solving math problems, pondering the significance of each character in a literary work, and playing chess. A major difference between the two modes can be that the reflective mode requires an internal representation in the mind of the subject, on which manipulations can be applied, while the experiential mode does not.

Although this may be an oversimplification, the classification is relevant to the development of Drum-Boy. The question that came to my mind is, what mode of activity is the process of composition? Since composition is a process of creation and refinement of a representation (the music), it seems to qualify as a reflective process. However, when an expert musician improvises, it can either be a reflective process (as with Bach and his improvisational *Inventions*) or an experiential one that does not require much deep thought (as with certain rock guitarists). In the latter case, the musician seems to be experiencing music, and communicating his emotional state to the outside world using the instrument. This observation adds a footnote to Norman's classification of activities: *that expertise can change a reflective activity into an experiential one.*

It naturally follows then, that a tool that addresses both beginners and experts must be able to handle both types of activities.

Methods for Composition

Many rock musicians who do not read music notation often compose directly on their instruments. They try different chord progressions and melodies on the instrument to help "visualize" their creation. The repetitious nature of most rock music burdens the composer with little cognitive load of remembering the structure and details present in the piece. The internal representation (the music) can often be kept within the composer's mind while more material and embellishments are added.

Experienced composers who do not need an artifact to "visualize" an idea ("I know what Dm7 sounds like, so I don't have to play it.") often work without an instrument, but directly in their mind. The attention to detail and the non-repetitiveness of "serious" music calls for a different type of artifact, ones that free the composer from having to keep the whole representation in his mind. Often it is the printed form of the score, since it is the preferred method of communicating the music to the performers.

2.1.4 Modes of Interaction

Using the terminology of the standard Human-Computer Interaction literature, there are basically five types of dialogue styles:

- Menu
- Form filling
- Command languages
- Natural language
- Direct manipulation

Downton analyzes these established modes of interaction in detail. They are summarized as follows [Downton91]:

Menus are currently widespread in personal computers and workstations, used primarily for issuing commands. Some characteristics of the menu system are:

- minimal typing is required
- low memory load
- well-defined structure
- good for decision-making tasks, but not for data entry
- suitable for casual, intermittent, and novice users

The *form filling* interface is ideal for entering data. An advantage of this paradigm is that most people are familiar with the concept of filling out forms. The user has to adjust to the syntactic rules when a different system is used. Such rules may include: display field constraints, optional fields, default entries, help, field termination, and navigation. Like the menu system, only a low memory load is required, but some familiarity with the syntax is necessary.

In the *command language* paradigm, the user types in a key combination or key word as commands to the computer. This method requires training. It burdens the user with high memory load. Several guidelines have been suggested to minimize memory load and typing errors. These include:

- choose short, memorable, non-confusable command word, and keep the format consistent
- provide on-line help
- place optional or least used items at the end of the command list, and use default parameters

In using *natural language*, no special syntax is need, and there is a possibility of the system being very flexible and powerful. However, when commands are ambiguous or imprecise, the resolution may be problematic.

Direct manipulation encompasses a wide variety of interfaces. An example is the WIMP (window, icon, menu, pointer) interface, where the user can drag the file icons into a folder. Direct manipulation interface encourages exploration, and reduces the time required for the user to become familiar with the system.

Drum-Boy uses several of these schemes where appropriate. The primary mode of interaction is based on the combination of direct manipulation and command language. By keeping the mapping between the user's actions and the triggered material simple and "natural," the user is presented with the illusion of manipulating these virtual objects (the rhythm patterns) directly. Just as a piano would produce a louder note when a key is hit hard, Drum-Boy would respond to a key being hit hard with more "complex" material. Also when a pattern is stored in a temporary save location, there is always a simple one-to-one mapping. At the expense of space to hold more patterns, there is a direct relationship between the position on the keyboard and the stored data - no shift key or page key to flip between "pages" of stored patterns.

The command language is also kept simple. Each command has the equivalent function key assigned to a key on the MIDI keyboard controller. Once again, there is a direct one-key-to-one-function relationship. The commands are grouped by function into sets of adjacent keys. An issue in developing the command language was to come up with a small number of useful commands. This was an iterative process - a process of adding commands that the system lacked, and removing commands that were not very useful.

While most real time commands were implemented using the command language and direct manipulation methods, some of the preparatory interactions were delegated to the more conventional menu-driven interface. For example, the selection of the hardware set-up is specified using a hierarchical menu. Loading pre-stored patterns from a disk file involves selecting a command from the menu and specifying the file name. While these modes of interaction subtract from the illusion of direct manipulation of the virtual instrument, they were seen as exceptions since the two types of interaction typically occur only once during a session, that is, in the preparatory phase before beginning the session.

2.1.5 Modeless Interface

Most physical tools, or artifacts, are modeless. When tools are combined into one physical unit, they often become modal. Take, for example, the Swiss army knife - every time a user pulls out a new part of the knife (and pushes back in the unnecessary tool), a different functionality is assumed. The user has to switch between the screwdriver, the knife, and an emery board. The switching between modes is acceptable if the user wishes to use the tool (the mode) for an extended period of time; however, if the user has to switch between modes frequently, the user is much better off, for example, if the army knife disassembled into several tools that are accessible to the user instantly. If switching between modes (tools) requires effort, the user may end up using a tool that is not intended for that purpose. For example, the user may use a bigger knife to cut out something where precision is necessary even though a smaller knife that offers precision is available, just because it is faster by not switching to the new tool.

A computer system complicates the situation even more. In a modal physical system, the mode is usually clearly visible because the physical controls often act as indicators of the mode. (see the section on object symbol) The user does not need a mental duplication of the internal state of the system because of the continual reminder of the state. For example, the user of the Swiss army knife does not have to remember that it is now functioning as a screwdriver because of its visibility.

Drum-Boy is designed to be modeless. The commands at different interface levels are not organized by hierarchy, but according to location on the keyboard. No precious time is lost in switching between modes. The only internal modes are: play/stop, bank number, and on/off state of auto-variation mode. These states are readily visible to the user.

2.1.6 Help

Even though the interface is designed to be as intuitive as possible, the user would benefit greatly by a help feature. When a user who is not familiar with the system approaches the system, he or she may pick up the interface just by trying out a few things. However, for those who are more cautious, the help key presents the user with messages, prompting for more interaction to obtain more specific help messages. Once again, the help facility is also non-modal. If the user decides that the help message is not necessary, releasing the Help key eliminates the help screen.

2.1.7 Input Device

Most of the interaction between the Drum-Boy system and the user takes place using the keys on the MIDI keyboard. The method for actuating the keys is self-evident, and the association with traditional keyboard instruments suggests that the system behaves differently depending on how hard the keys are pressed. Velocity and after touch are two dimensions in addition to the selection of the key. An added dimension (for example, some prototype keyboards are sensitive to the vertical location of the finger on the key) would eliminate the need to store the bank number internal state variable.

In addition to the keys, controllers such as the modulation wheel and foot pedals are used. The modulation wheel acts as a controller capable of specifying a value from a range of numbers. In Drum-Boy, it is mapped to the tempo control. The lack of dexterity of the feet forces the foot pedals to be mapped to simple functions. On the other hand, they can act independently from the hands; controls that can not be assigned to keys (because more than two hands will be required) can be assigned to the foot pedal. The final version of Drum-Boy uses two pedals, one functionally similar to the sustain pedal in a piano (Delay Release pedal), and one for altering the behavior of the drum keys (Event/Phrase pedal).

While the MIDI keyboard fulfilled our needs in the project, we do not view the interface as being in its ideal form. For example, many would probably prefer tapping in rhythms through the traditional "drum sticks and drum pad" physical interface. The one dimensional layout of the keys do not help the user form a mental grouping of keys (although the octave layout and graphical key labels do help to some extent). The interface for specifying transformations can take on a more "natural" interface, such as those resembling steering wheels[4].

2.1.8 Feedback

Drum-Boy offers feedback in three of the human senses: tactile, visual, and auditory.

[4]The steering wheel interface was suggested by Tod Machover.

Tactile feedback

The feedback offered using the tactile sense is not an active one; it is intrinsic in the physical controller used in Drum-Boy. The piano layout keyboard has keys arranged in such a way that orienting the hand onto and within each octave is easily done without the aid of vision. Blind blues and jazz pianists are proof of that. Familiarity with Drum-Boy will remove the need to constantly keep the eyes on the keyboard.

The degree of aftertouch (pressure exerted on the key after the key is down all the way) is felt by the performer using the tactile sense. While the state of the foot pedal can be verified by other means, the tactile channel gives the user continuous feedback on its state.

Visual feedback

Since Drum-Boy was originally conceived to follow the traditional instrument paradigm, less emphasis was given to visual cues. After all, expert performers of traditional instruments rely little on visual feedback from the instrument. In Drum-Boy, some of the cues come from the controller itself, since in typical use the MIDI keyboard is labeled with a strip of icons (small pictures representing the functionality of each key). Additional cues come from the explicitly designed graphical feedback shown on a computer display.

One of the design goals of Drum-Boy was to necessitate as little visual feedback as possible using auditory feedback as the primary source of feedback. However, Drum-Boy has *memory*, an element that traditional instruments typically don't have. Since memorizing is a task at which people perform poorly, the graphical feedback helps the user to remember information; for example, the visual feedback tells the user which of the many slots (corresponding to Pattern keys) already hold patterns.

Other information conveyed through visual feedback is mostly redundant, including:

- indication of error
- visual representation of the current pattern, what notes are being played now
- mapping between keys and timbre
- tempo
- command that the user is about to execute

All of this information is available through other means, but *conveying the same information through different modes reinforces the user's sense of being in control.*

Very little textual information is displayed in order to lessen the amount of semantic processing. Graphical icons such as those shown in Appendix A are primarily used in conveying information.

Auditory feedback

The audio output of the system is its final musical product, which is the object of constant manipulation. It is important for the user to be able to perceive this object without distraction. Therefore, sounds that can convey additional information were purposely omitted. For example, a simple beep or click is often used as an indicator for the actuation of a switch or button; a different beep can be used to denote an error. Such techniques to convey feedback information had to be adapted into the other modes, such as the visual channel using the graphical display. For the same reason, the metronome functionality (or click track) was deleted in the final version.

2.2 Computer as (Musical) Instrument

Up until this point, I have used the terminology and concepts developed in the field of human-computer interaction to describe the design of an interactive music system. Perhaps this is the time to generalize my analysis and apply it back to the problem of interface design.

Brenda Laurel develops the idea of using the theater as an analogy in designing an interactive computer system [Laurel91]. Her notion is that the degree of interactivity can be measured by the degree to which "you feel yourself to be participating in the ongoing action of the representation." This definition of the measure of interactivity calls for an immersive environment. What better immersive environment is there than the theater?

Such thinking advocates the idea of studying interactive theater and novels as a model in designing a computer system. Often such interactive media employ *agents*[5] as the intermediary between the virtual world and the audience. Techniques from artificial-intelligence, theater, psychology, and human-computer interaction converge into this interdisciplinary area of study.

While this seemingly bold analogy between theater and computers holds for certain types of human-computer activities, I assert that it is not suited for others. Laurel classified human-computer activity into two broad categories, experiential[6] and productive [Laurel86]. For experiential activities, such as playing computer games, the theater analogy is perfect. But what would it mean to hold a *productive* session with a theater? Do you really want to delegate all your activities to computer "agents"? What if you just want to be alone, sit down with a pen and paper and write to a friend - do you want a theater in front of you?

I further characterize the experiential activities by a parameter that specifies the degree of interaction. An experiential activity can be a passive one or an active one. Watching TV, as often cited, is a perfect example of a passive experiential activity. Exploring a "virtual" island using force-feedback controls and stereo vision is an active experiential activity. Active experiential activity often has the characteristic that the participants' interactions are pleasant in themselves. Furthermore, the interaction does not need to involve an agent; a crossword puzzle, a pinball machine, and most notably a musical instrument all have the potential to award the participant with a pleasing, actively experiential activity.

Musical instruments are very special things. Using Laurel's classification, playing music is both an experiential and a productive activity. Music is fun to play, and one can produce a recording or compose a piece with an instrument. Using Norman's classification, it is an experiential activity for an expert, and a reflective activity for a novice (as a novice often has to think about how to produce the right note).

---

[5]"Agent" in this context means *one who initates and performs actions*[Laurel91].

[6]Note that the term *experiential* as used by Norman describes the cognitive aspect of an activity. When appropriate responses are generated without effort or delay, he calls the mode of cognition *experiential.* He uses the word experiential to emphasize the subjective aspect. An equally valid name for this mode of cognition, he states, is "reactive" [Norman93].

Here are some characteristics of a typical musical instrument from the interface point of view:

- low cognitive load to play - no syntactic processing, no load on memory while playing
- simple controls, though some controls are arbitrary and have to be learned (such as fingering on a sax)
- the effect of an action is immediately perceivable
- physically tightly coupled with the player[7]

An expert musician usually prefers that a recording be kept whole. If a recording of a piece were constructed by assembling parts from several recordings, the continuity is lost. When a musician performs a piece, the internal states (often emotional) within the performer's mind change continuously over time. Since the evolution of these parameters are different every time a performer plays a piece, the cut and paste approach would leave discontinuities everywhere.

Present day word processors and similar tools disrupt the thoughts of the writer all the time. In most word processors for the Macintosh, to italicize a word, the user either 1) types the whole sentence, selects the word to italicize, and then tells the computer to italicize it using an often hierarchical menu; or 2) the user can stop typing mid-way through the sentence right before the word to italicize, issue a menu command, pause again after the word to issue a plain text command. Either way, the mental flow is unfortunately terminated.

In programs that support graphic design (where, incidentally, modal dialogues, a major contributor as a flow stopper, are rampant), the user often types in a number value to adjust the shading. (Let's try 50%... no that's a little too dark, how about 40%? etc.) Why work discretely in a continuous domain?

[7] We see street musicians playing five instruments simultaneously as being humorous, not because of the number of instruments, but because 1) the "gadgets" are attached to him, and 2) there is an unclear notion of who/what is controlling whom/what. We don't see musicians playing traditional instruments as funny because we are used to seeing people playing these instruments. For example, how would one feel at the first sight of someone with a huge metal object on his shoulder, sticking something in his mouth and blowing it with his cheeks bloated?

In the first example of italicizing words mid-sentence, an "instrumentalist" solution may be to equip the system with foot pedals that correspond to the different effects one can apply to letters that are being typed. In the second example, one solution may be to have the user blow into a tube with a pressure sensor, whose output determines the shading of the objects in the drawing programs (the shading has to be updated on the screen constantly for feedback, much as an instrument would change the volume of the tone as the breathing is varied).

Maybe these examples are unrealistic solutions. And these modes of interaction require practice. But as Laurel points out, "Interface metaphors[8] have limited usefulness. What you gain now you may have to pay for later. [Laurel91]"

The essence of a musical instrument is that it captures the evolution of emotion continuously over time. The ability to do so is essential in a tool for expression, a tool for creativity.

2.3 Artifacts in a Creative Environment

In designing a tool whose purpose is to aid a creative process, it is worthwhile to look at other processes of creation and what kinds of tools are used.

Take, for example, writing. An indispensable tool for a present day writer is the word processor. A typical word processor gives the user the ability to realize the following:

- the user can view the document as it will be printed (WYSIWYG - "what you see is what you get")
- the user can add sections in the document in any order, and move around parts of the document
- the user can "find and replace" words or phrases

From a functional point of view, the purpose of a word processor is the same as that of a typewriter: to give as the output a sheet of paper printed with what the writer wanted to

[8] An example of the interface metaphor is the Macintosh "desktop."

say. However, by working on a representation of a piece of paper (and not on paper itself), manipulation, such as moving paragraphs around, becomes a problem not of how to execute it, but of how to effectively communicate the intention of doing something. Modifying data is easy; modifying a physical object is not.

Other tools for writers exist, and many of these have been translated into the electronic domain. Many of these tools have existed as reference books, such as a dictionary, thesaurus, and Elements of Style by Strunk and White. Electronic versions of these tools can be a direct translation or applied. An electronic dictionary, for example, can be equal to its physical counterpart in its functionality, or it can offer more; commercial products exist in which if one types "yellow and flower" the computer comes up with words such as chrysanthemum and dandelion. One can have an on-line reference to Strunk and White, or have a grammar corrector that can point out dangling modifiers. There even exist electronic dictionaries that pronounce the words for you!

Reference materials can aid creativity in two ways. *Browsing* gives artists ideas and inspirations. I conjecture that browsed material works as random indices to the associative memory, often triggering the retrieval of multiple, unrelated objects or concepts. These objects or concepts, in turn, are processed and combined to form "novel" ideas.

Directed search can be used to verify something (such as checking a dictionary for spelling) or as an index to related entries that can in turn be browsed. A thesaurus listing related words can help a novice writer find a word suitable in a particular context.

2.3.1 Properties of Instruments that Stimulate Creativity

Having instant access to tools at different levels is very important in a creative environment. Very often, such an environment either supplies only tools that work exclusively at a certain level, or forces users to switch between modes or navigate through a hierarchy of menus or commands. An important feature of systems intended to stimulate creativity is that operations must be easily realizable. When good ideas that flash by the user's mind are not instantly realizable, the momentum is often lost.

Equally important are features that allow the user to easily undo changes. By keeping the cost of making mistakes low, the user can try different operations on the material "just for the fun of it." The undo feature also helps alleviate the anxiety that may restrict the user's imagination.

Serendipity also classifies as one of the features useful in a creativity tool. The computer can either introduce randomness into the system, or use a process that is very complicated such that the user will not be able to predict the outcome. When it is part of a system, the user's ability to select the good from the bad becomes important.

In Drum-Boy, there are two aspects to this issue. In the player model, it is favorable for the user to have as much control over the process as possible. All the mapping is deterministic and simple, and no random factors are involved. The transformers, on the other hand, are a way for the system to suggest material to the user. Non-determinism (or processes complex enough that the user would not be able to predict the outcome) here is welcome.

2.4 Drum-Boy Interface

2.4.1 Multi-level Interface

The system succeeds in addressing a wide audience by allowing the user to interact with it at various levels. While drum patterns can be meticulously crafted by using the note-level editing tools, users who want to sketch out a song can benefit from the phrase level controls which will free them from the task of specifying each drum hit without taking away the sense of control. Users who do not have the understanding of a drum set but "have the ears" to recognize and describe in musical terms what they want can also benefit from the system, because they can navigate through the space composed of drum patterns by giving directional commands.

2.4.2 Global parameter control

There are two global parameters that can be controlled: tempo and shuffle, both of which can vary over a continuous spectrum. In our system, the tempo parameter is controlled by a modulation wheel (which is a continuous controller often used for changing the timbre of notes when a MIDI keyboard is used to control a sound generator in the traditional manner, i.e. one key mapped to one note).

Shuffle, however, is a parameter whose relationship between the numerical value and the actual effect is less direct. In Drum-Boy, shuffle determines the placement of the upbeats of sixteenth-notes. A straight feel (shuffle = 0%) means that the onset timing values are equal for the downbeats and the upbeats. A 100% shuffle feel denotes that the sixteenth-note upbeats are actually only half as "long" as the downbeats. The perceived change in duration is achieved by changing the onset timing. In particular, the upbeats are moved forward in time so that the downbeat comes sooner after the upbeat.

A straight beat will have the following onset time values (the values are offsets from the beginning of the measure):

(0 120 240 360 480...)

A 100% shuffled beat will be:

(0 160 240 400 480 ...)

A 50% shuffled beat is:

(0 140 240 380 480 ...)

It is not easy to guess exactly how a 50% shuffle sounds. However, its monotonic characteristic is easily perceivable. When two patterns with shuffle degree of 30% and 50% are compared, one immediately notices that the latter has more shuffle. Our implementation of the control for shuffle reflect this characteristic. Rather than have an absolute scale controller like the modulation wheel, we have control over the change. The user can get more shuffle, or subtract some of the shuffle feel depending on how hard the shuffle key is hit.

2.4.3 Note-level Event Control

At the note level control, the user can perform the following:

- Insert and delete a drum hit
- Change the accent of an existing drum hit
- Quantize drum hits (i.e., correct the timing)
- Play a drum (without changing the current pattern)

These are simple controls; yet, for a trained musician who knows exactly what drums to play to achieve certain effects, these tools are sufficient to build the desired drum patterns quickly. The user can communicate any rhythm pattern by the use of these tools.

The user can make incremental changes to a pattern in its natural time-based domain (as opposed to static graphical representation) and get immediate feedback on the changes that were made. The pattern is continuously "refreshed" by looping.

Traditional "cut and paste" editing functionality was not incorporated because, due to the size of the material (one measure rhythm patterns), the cost of duplicating a rhythm is not too expensive. Also, since pattern fragments have to be manually duplicated, duplicated fragments will be always slightly different. Exact duplicates take away the richness that layering can provide.

For a detailed explanation of each function, see Appendix A.

2.4.4 Phrase-level Event Control

A *component* is comprised of instruments logically grouped together - for example, an open hi-hat and closed hi-hat comprise a hi-hat component; a kick drum and a snare drum comprise a kick-snare component. There are presently four types of components:

- kick-snare component
- hi-hat component
- monophonic percussion component
- polyphonic percussion component

Our database consists of entries for each type of component, allowing different components with the same type to share the database (for example, the conga component, comprised of high-conga and low-conga, share the same database as a woodblock component, comprised of high-woodblock and low-woodblock).

By breaking down patterns into components, the number of effective patterns in the system becomes large due to the combinatoric "magic." Our initial fear of the need to find a perceptual compatibility function[9] disappeared after discovering that most rock or Latin rhythm fragments are compatible with one another, as long as the rhythms are played on appropriate instruments.

At the phrase level, the user can trigger the system to play groups of notes. A phrase in Drum-Boy is a one-measure long segment of a rhythm pattern, either with limited instrumentation (using instruments from a single component) or the whole drum pattern.

[9] For example, "If a certain rhythm on the hi-hat is overlaid on a certain pattern played on the conga, is the result still pleasing?"

The component track phrases are all pre-stored in the database, and the system's job is to allow the user to call up desired entries in real time. An important design issue addressed here is the mapping between the user's input and the phrase picked by the system. In the current implementation, the user has three degrees of freedom. The first dimension, the drum component, is specified by the user according to the key that is played. The second dimension, how "convoluted" the phrase is, is specified by the player using the bank selection keys, that change the internal state of the system. At any time, the user can tell the computer to switch to a certain bank that corresponds to the type of phrase(s) that he wants to call up (the three banks right now correspond to: basic, funky, and "convoluted" - the organization of the phrases into these categories was done by hand). The last dimension, how complex/dense the phrase sounds, is specified by the velocity at which the key is hit.

The two dimensional space (in this case the convoluted-ness and the density/complexity of the phrase, selected by the bank and velocity) can be arbitrarily selected by the designer by defining two feature calculation functions. Since the two dimensional space is a projection of a space with higher dimensionality[10], it is preferable that the two features selected account for a large variance. Each feature calculation routine returns a scalar value which the system uses to automatically calculate the coordinates of the projection and put the phrases in bins that are addressed by the bank/velocity combination.

Using the key velocity to address one coordinate poses the problem of not being able to specify the exact pattern of choice. Since a MIDI controller is used in the present implementation, the velocity can take 127 discrete values. For the player to enjoy the maximum number of phrases addressable by velocity (i.e., map a different phrase for each velocity value), making an exact choice would be virtually impossible. By using a much smaller set of phrases for each bank, an exact choice is easier to make. However, this option severely limits the range of action the player can take. The author's belief is that if 1) the phrases are mapped onto a scale that is perceptually continuous, and 2) correcting a phrase that is mistakenly selected is easy, then the capability of exact addressing is not necessary.

[10]We believe this is so, but no empirical data proves that rhythms are highly dimensional. The actual number of dimensions, and the meaning of each feature, has not been determined.

Whole patterns can be stored in the system so that they can be retrieved instantaneously. Patterns are stored and called up by the user by pressing the corresponding keys in the lower octaves. When the key is hit hard, the stored pattern is immediately called up; when the key is hit softly, the pattern is placed on a queue, waiting to be played when the current measure finishes playing.

2.4.5 Pattern-level Transformations

This level of interaction is primarily for those users who can recognize material that they like when they hear it, but do not know how to assemble such material from scratch. In using this function, the user begins by entering an initial pattern that resembles something he/she wants. The same MIDI keyboard interface is then used to communicate to the system, for example, "Make the pattern more energetic," or "Make the pattern more stuttering."

Internally, the new pattern is created by the use of objects we call transformers, which take a pattern and algorithmically modify it. An "agent" approach was used in developing the descriptive high-level transformers. That is, a transformer is either a primitive transformer itself, or a combination of other transformers.

Such a structure allows easy experimentation within the environment. Since the construction of transformers was seen to be an iterative design process, ease of prototyping was important. The modular construct gave the designer reusability, and an idea that got thrown out during experimentation was easily scavenged for other experiments.

The "functional block" approach also eased the separation between system design and incorporation of highly specialized musical knowledge. A musician without knowledge in the specific computer programming environment could easily become part of the design team. This notion is expanded in a later section.[11]

A primitive transformer performs very simple tasks, such as adding a specific new drum track; or looking for a specific pattern of hits that occurs in the pattern, and replacing it with

[11]See section 2.6.2 Transformers and Analyzers. Appendix B lists all the transformer and analyzer classes.

something else. Another primitive transformer changes the tempo. The premise is that these simple transformers can be strung together into bigger transformers that can perform yet more complex and musically meaningful transformations.

One must recognize, however, that a single transformation algorithm does not, in general, perform transformations that make musical sense in all contexts. For example, if the current pattern already has a strong shuffle feel, notes should be added in positions that preserve the shuffle feel unless a change in the feel is desired by the user. Thus, a logical complement to the transformers is a set of analyzers that affect the transformation process. In the above example, a composite transformer can be built from two transformers, one for shuffled patterns and another for a straight pattern. The composite transformer will consult the output of two analyzers, shuffle feel and straight feel, and invoke the correct transformer. In Drum-Boy, analyzers are implemented similarly to transformers; primitive analyzers are combined to make higher level analyzers that can in turn be combined into higher level transformers.

The analyzers, as presently implemented, are simple objects. Often they do no more than count the number of notes that lie within certain temporal regions. Because Drum-Boy works in a very constrained stylistic context, further development of analyzers was not considered necessary. The architecture supports more complex analyzers, and these will be needed as more styles are incorporated and complicated transformers are developed.

2.5 Rhythm Pattern as a Multidimensional Object

Multidimensional modeling of perception has been a commonly studied field, especially after the 1970s when techniques such as multidimensional scaling started gaining wide acceptance. The premise in this scheme is that many perceptual phenomena can be modeled in a mathematical space whose axes correspond to some features that the subject perceives. The features might be shape, brightness, temperature, frequency, or loudness. Once such a space is set up, conventional distance metrics, such as the Euclidean distance or the "city block" distance, can be used in approximating the perceptual similarity between two perceptual phenomena.

The goal of Drum-Boy's navigational system was to reconstruct the perceptual rhythm space computationally. When such a space is constructed, the user can "navigate" in this space while listening to rhythm patterns that correspond to the points along the path. By using perceptually salient features as the axes, moving in the direction of the axis will allow the user to fetch a pattern whose specified feature is more pronounced (such as, "I want a pattern that is a little more syncopated," if syncopation is one of the features, or axes).

2.5.1 Separable vs. Integral Dimensions

While the construction of such perceptual space seems enticing, its construction is not straightforward. The first problem that has to be solved is the selection of the axis, or features. If objectively measurable physical dimensions of the stimulus can be used directly as the axes, construction of such space would not require any additional work. However, the validity of using the physical dimensions must be proved. One must answer the question of how the stimulus dimensions interact during perceptual processing.

There are two main classes of stimulus dimensions: integral and separable. If tests show that the dimensions are processed independently, the dimensions are termed *separable*. On the other hand, if tests show that the dimensions are processed as a unitary whole, the dimensions are *integral* . Prototypical separable dimensions and integral dimensions are hue and shape; and hue and brightness, respectively. Separable dimensions, because they are conceptually orthogonal, can be used as axes in modeling the perceptual space. However, integral dimensions, because they are not processed independently in human perception, must be analyzed so that human perception can be modeled. Multidimensional scaling (MDS) is one technique to determine the perceptual dimensionality of the physical integral dimensions. Based on the dimensionality and the points in the derived perceptual space, the methods for obtaining the feature vector must be reverse-engineered.

Maddox suggests four operational tests of separability of dimensions [Maddox92]. One of the most popular such tests is *Speeded Classification.* In this test, a subject is asked to categorize an object along one component (for example, the shape of an object). In the control condition, there is no variation along the irrelevant dimension (for example, the color of the object). In the filtering condition, however, variation along the irrelevant dimension is introduced. If the response times of the two conditions are different, the irrelevant dimension *interferes* with the processing of the relevant dimension, and because of the assumption that selective attention to one of two integral dimensions is difficult, the two dimensions will be determined to be integral dimensions. While the test is very commonly used in determining the separability of dimensions, because of the temporal nature of rhythms this technique can not be applied to the domain.

In *Direct Dissimilarity Scaling* , the subject is presented with all possible pairs of stimuli, and is asked to rate the dissimilarity (i.e., the distance) between the pair. The ratings are processed by a Multidimensional Scaling procedure. If the Euclidean distance metric fits the data, the dimensions are assumed to be integral; if the city block distance metric fits the data better, the dimensions are assumed to be separable.[12]

2.5.2 Application of Multidimensional Perceptual Models in Music[13]

The use of space as a musical control structure has been investigated by Wessel in 1979 [Wessel79]. He created a two-dimensional space with axes, "bite" and "brightness," and mapped timbres onto the space. Such a control is implemented in a few commercial synthesizers as "vector synthesis," where the user can use a joystick to control synthesis parameters in a multidimensional space.

Experiments have been conducted to show the validity of the argument that rhythm patterns are multidimensional objects [Gabrielsson73a]. Gabrielsson, in 1973, performed a series

---

[12]An excellent reference in this subject is [Ashby92]. Techniques for multidimensional modeling of perception and cognition are discussed in detail.

[13]One might argue the validity of using MDS for constructing a model of the perceptual space. This technique works only if psychological distance can be modeled by a well defined distance metric, such as the city-block or Euclidean metric. More generally, distance axioms must be satisfied. They include:

- all perceived self-dissimilarities are equal, or $d(i,i) = d(j,j)$
- *minimality*, or perceived dissimilarity between two different stimuli is at least as great as the perceived dissimilarity of either stimulus to itself, or $d(i,j) >= d(i,i)$ and $d(i,j) >= d(j,j)$
- *symmetry*, or perceived dissimilarity between stimuli i and j equals the perceived dissimilarity between j and i, or $d(i,j) = d(j,i)$
- triangle inequality, or for any three stimuli i, j, and k, the triangle inequality holds, or $d(i,j) + d(j,k) >= d(i,k)$

The empirical validity of these assumptions has been questioned in detail by [Tversky77].

of psychological experiments and applied the multidimensional scaling (MDS) technique to analyze the dimensionality of rhythms. He also performed factor analysis on the adjective ratings of rhythm patterns. He conducted experiments with both monophonic (single drum) and polyphonic (drum set) rhythms. However he points out the difficulty in generalizing his results because of the fact that there is practically an unlimited number of rhythm patterns, and the MDS technique only accommodates a limited number of samples. In his discussion, he states, "Since the population of [rhythms] is practically unlimited and since only a limited number can be included in each experiment, the generality of the obtained dimensions is hard to evaluate."

In multidimensional scaling, if the subject is asked to evaluate the perceptual distance between points, the amount of computation and number of judgments required by the subject increases as $O(n^2)$ when n = number of samples of different patterns. (There are n(n-1)/2 distances between each pair of samples.) This method is difficult for the subject, as quantifying the perceptual distance is often difficult without an aid to reference what, for example, a distance of 5 means. An alternate approach of comparing two distances is also possible. The subject is given three stimuli, A, B, and C, and is asked to judge whether the distance between A and B is greater than the difference between A and C. This lightens the load on the subject in making the judgments; however, the number of judgments required increases as $O(n^4)$.

2.5.3 Feature-space in Drum-Boy

Due to the impractical complexity of the problem, an alternate method was used to obtain the set of features. An ideal set of features will be composed of musically meaningful features which are orthogonal with respect to each other. Our goal was not to compile a comprehensive list of features that are perceptually meaningful, but to develop a set of features small enough so that a search in the space can be done in real time. The derivation of the feature set comprised of a brainstorming session where features were listed unmethodically, after which the list was pruned to satisfy the above criteria. In our implementation, the set consisted of the following features:

- number of hits
- half note feel
- quarter note feel

- eighth note feel
- sixteenth note feel
- symmetry

In defining the feature-space, the K-means clustering algorithm was used to make sure that patterns belonging to the same cluster sounded similar. This verification was based on an informal evaluation by the author and a colleague listening to all the patterns[14] belonging to each cluster. Individual features were given weights to make the Euclidean distance in the space simulate the perceptual distance. The weighting coefficient for each feature was obtained by an iterative process.

In matrix notation, the definition of the distance metric can be expressed as:

$$D(\mathbf{x},\mathbf{y}) = [\ (\mathbf{x}-\mathbf{y})^T\ \mathbf{W}^{-1}\ (\mathbf{x}-\mathbf{y})\ ]^{1/2}$$

where **x** and **y** are feature vectors, and **W** is the weighting matrix.

To give the user more control in manipulating the drum pattern, the feature-space analysis dealt with each drum component separately[15]; the objects placed in the feature-space were not whole drum patterns, but component tracks. The definition of the features were the same for each drum component. (However, the actual database entries were different for each drum component.) Since the drum patterns retrievable from the database consist of combinations of the component tracks, our component approach is equivalent to using whole patterns except that features pertaining to instrumentation were ignored.

The navigation system consisted of a currently playing pattern (the internal state); a component to modify, the axis and direction of navigation (user input); and a modified pattern (output) that incorporated the changes to the component. Given the current pattern and the component specification, the coordinates in the feature-space for the component were calculated. The database was searched and the entry for the component with coordinates closest to the original coordinates in the direction specified was chosen to replace the component track.

[14]The size of the data set was approximately half of the size as described in the next section.

[15]See section 2.4.4 for a detailed explanation of *components*.

This approach to modifying the pattern was adequate to the extent that specifying "give the hi-hat more eighth note feel" gave an output with the proper characteristics. However the change was not always smooth. This abruptness stemmed from the fact that 1) the database wasn't large enough (so the closest point was still sufficiently far), and 2) navigation was totally feature-based and ignored the number of hits that are shared by the old and new component tracks. A drum hit is shared by the two component tracks if the same drum hit occurs in both component tracks simultaneously (within a tolerance time window).

Smoothness in the transition suggests that the change be minimal. A possible enhancement to the feature-space approach (which I did not implement) would be to incorporate this measure of similarity, hits being shared, in the distance metric to make the transition smoother.

2.6 Transformational Approach

While the feature-space structure is well suited for a database search, our constraints (a lack of large database[16], goal of navigation is not "more eighth-note feel" but "more Latin feel", and the need for navigation on patterns and not components) suggested a different approach, namely, the transformation approach.

Transformation has the effect of filling up the empty areas in the "rhythm pattern space." The small database covered the space sparsely; transformation can expand the effective coverage of the space by each point.

Two methods, the interpolation method and the more general transformation approach, are described in the following sections.

---

[16]The size of the database for each component type was:

- hi-hat component: 40
- kick-snare component: 110
- monophonic percussion component: 94
- polyphonic percussion component: 36

An alternative method suggested, but not used, was the generative approach. Using this approach, algorithms to generate drum patterns had to be invented, either by modeling human methods or by ad hoc means.[17] The transformational approach was adopted because many of the concepts developed in the multidimensional model could be utilized in a transformational system.

2.6.1 Interpolation

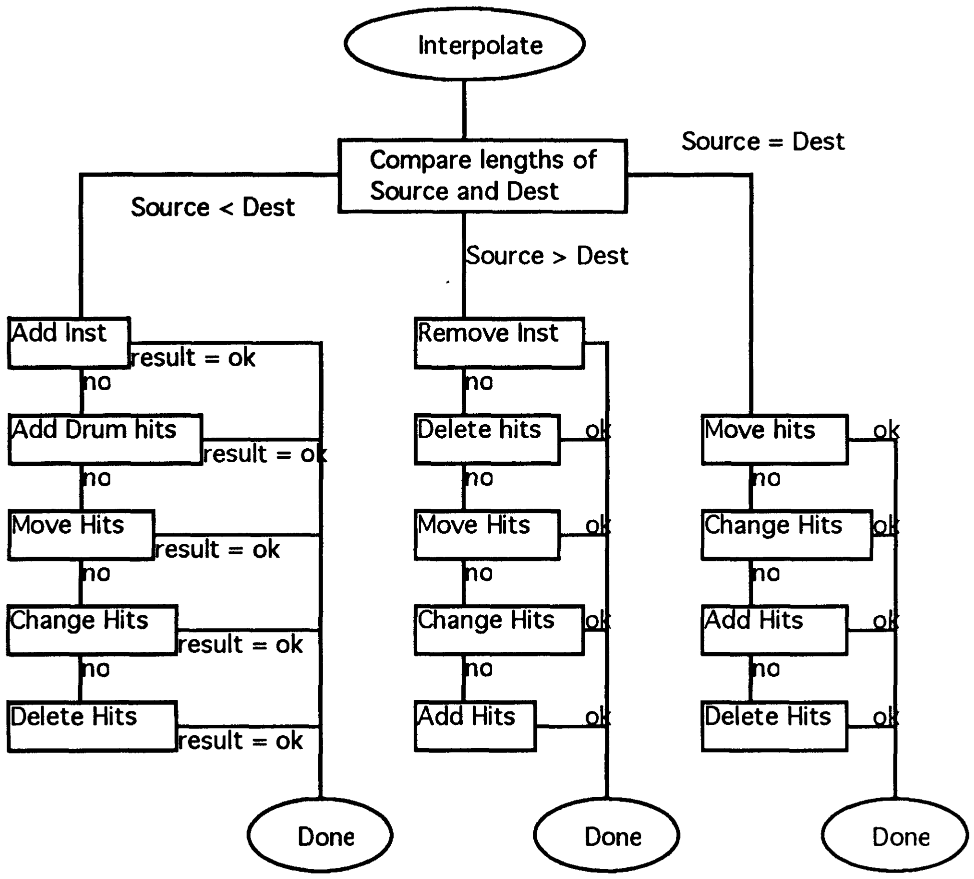
The first solution using the transformation approach was suggested in order to eliminate the ambiguity of the problem: What exactly does one mean by "more rock feel"? The phrase means different things to different musicians, or even to one person on different days! The solution suggested was that each transformation be an interpolation of the drum pattern towards a "reference pattern" prepared for each musical style (such as rock, Latin, or industrial-dance).

From the designer's point of view, the solution is that of placing additional constraints on an open-ended problem. From the user's point of view, having access to the reference patterns makes the interpolation more convincing, since any modification the system decides to apply to a pattern can be explained by the reference pattern.

The interpolation algorithm always makes modifications to the original pattern. This guarantees the perceived "smoothness" of the transition. The algorithm uses concepts such as adding an instrument, removing an instrument, adding/removing drum hits, and moving drum hits.

[17]Automatic methods to generate algorithms without relying on the traditional analysis/synthesis method have been explored by researchers [Koza92]. These methods, usually refered to as *genetic algorithms,* or *genetic programming,* use parameterized models to generate objects (or results). These objects are evaluated, either automatically by an evaluation function, or perceptually by human subjects. The parameter lists that generated objects that scored high are "cross-bred" with the parameter lists of other objects that scored high. The idea of selection and cross-breeding resembles the evolution of the genes, hence its name.

The following diagram describes the interpolation algorithm used.



The interpolation algorithm works at a notationally quantized level. Time slots are allocated, each with a time window whose size is equivalent to the resolution of the quantization used with the algorithm. In our experimentation, the quantization resolution was set at the sixteenth-note level.

The six primitive operators used in the interpolation algorithm are:

- add instrument
- remove instrument
- add hit
- remove hit
- move hit
- change hit

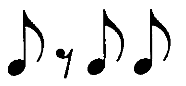
Add instrument adds a new component track to the pattern from the database. In order to keep the addition of the new instrument from changing the feel of the pattern very radically, the selection from the database was limited to simple patterns by limiting the range of the database entry used.

Remove instrument removes a major portion of the component track, by randomly deleting a certain percentage of all notes played by the component. 60% seemed to work the best.

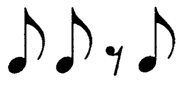
Add hit adds a hit that does not occur in the source pattern but does occur in the destination. Similarly, remove hit removes hits that occur in the source pattern but does not occur in the destination pattern.

In cases such as:

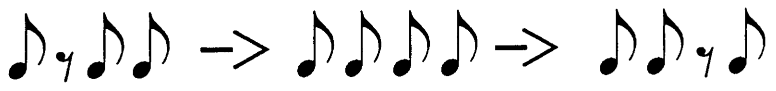
Source:



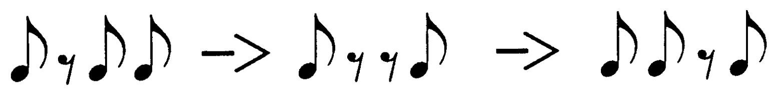
Destination:



It is preferable that the change from the source to the destination be a one-step process, and not a sequence of adding and deleting notes.



and



Move hit takes care of cases like the above, and moves the hits. Add hit and remove hit detect these situations and will not attempt to add or remove hits.

Change hit changes the instrumentation within the component class. For example, if the source pattern has a closed hi-hat where there is an open hi-hat in the destination, the closed hi-hat is changed into the open hi-hat.

The effectiveness of the general interpolation algorithm as a transformation tool is more pronounced near the source and the destination patterns. This observation is not surprising since the patterns near the endpoints resemble a "real" pattern, whereas the patterns in the middle often do not, and there is no mechanism for checking the overall coherence of the pattern ("Does this pattern make sense as a drum pattern?").

An effective interpolation scheme calls for a high level representation of the material. For example, in a successful image interpolation program, Xspace [Librande92], a *learning module* analyzes example images and computes a mathematical function which describes not only the examples but the result of interpolating between examples. This was made possible by having the bitmaps converted to sets of vector drawings. Furthermore, the features were matched among the images, and ordered according to the features used for interpolation.

This approach of human preprocessing to tag the data with additional knowledge could not be used in Drum-Boy because the source material for interpolation did not come from the database. High-level information about the material had to be extracted from the data itself.

In addition to the suggested need for an overall coherency test, in trying out the algorithm for different musical styles, a specialization of the algorithm seemed to be necessary. Depending on the style of the patterns used, both the kind of drum and the timing of the hit took on a different meaning.

2.6.2 Transformers and Analyzers

In order to fix the shortcomings of the interpolation system, some form of analysis had to be done on the drum pattern. Several research projects including Rosenthal's *Machine Rhythm* [Rosenthal92] and Rowe's *Cypher* [Rowe91] involve analysis of musical material using a "web" of simple agents. The agents themselves have simple structure and have only the ability to extract simple features from the material, yet when the agents are interconnected, the system as a whole shows a seemingly intelligent behavior.

In Cypher, both of the two major functional blocks, the listener and the player, are hierarchical structures, which provide the system with layers of abstraction. The higher levels need not deal with raw data because the lower levels will process it into more meaningful information. The lowest level examines individual events while the next higher level analyzes a group of events. This means that higher levels operate on structures that span longer duration of time.

In Drum-Boy, a similar hierarchical architecture is used in implementing the transformers and the analyzers. However, the higher level structures exist only as the means to organize and combine the lower level structures. The higher level structures do not exist independently, awaiting information to be brought by the lower level agents.

Choosing the transformation adjectives

Before constructing the transformation algorithms, the set of transformations available to the user had to be determined. Care was taken so as not to make the selection of the set arbitrary. An ideal set will be small and orthogonal (or uncorrelated). We chose to use adjectives as opposed to more specific, musical terms such as "more syncopated" or "more eighth-note feel." This was a deliberate decision since it was thought that these musical terms would be too specific for beginners, and that experts would most likely choose to perform the modifications explicitly (using the note-level tools).

A list of approximately 100 adjectives which are suitable for describing rhythms were suggested by Gabrielsson [Gabrielsson73a] in his study of factor analysis of rhythms. The result of his factor analysis on monophonic rhythm showed that three factors accounted for 91% of the total variance. From the table of adjectives with the most positive and most negative factor scores, five adjectives (mechanical, energetic, calm, graceful, and stuttering) were selected. (Since the table listed positive and negative scores, and there were three factors, six adjectives could be used, but the sixth adjective, floating, was substituted to make the bipolarity of the adjective set more eminent.)

Two additional adjectives (hard and soft) were selected from his experiment involving polyphonic rhythms (rhythms involving more than one instrument).

While the four pairs of adjectives seem to satisfy the orthogonality constraint, a fifth pair (simple/complex) was added to the list to accommodate the fact that the adjective pair is very often used in describing rhythms. In the context of factor analysis, the adjective pair is a composite feature that has many factors combined. It was used because the feature seems perceptually quantifiable.

The five following pairs of adjectives were used:

- complex vs. simple
- hard vs. soft
- mechanical vs. graceful
- stuttering vs. floating
- energetic vs. calm

The structure of the transformer/analyzer system

The transformation system consists of two parts, the analyzers and the transformers. Both are complex objects constructed from simpler analyzers and transformers respectively.

There are basically four kinds of primitive transformers:

- Instrumentation change
- Note position (timing) change
- Global parameter change (such as tempo)
- Velocity/accent change

As was the case with designing the note-level tools, one of the goals was to keep the set of tools small so that selection among them was easy. The above classes of tools, when combined, have the capability of transforming any rhythm into any other rhythm.

These primitive transformers are combined in one of several ways to form a higher level transformer:

- A list of transformers are used sequentially
- A list of transformers are used in random order
- One transformer is selected from a list depending on analyzer output

This architecture provides the designer of the transformation/analysis system with a language in which to describe complex algorithms. The data structure and the actual manipulation of the rhythmic data is hidden away in a layer of abstraction. Given a set of primitive transformers and analyzers, and a set of algorithms to combine them, the designer's task is to specialize the primitive objects when necessary, and to combine them. This task seems particularly well suited to a graphical interface environment.

In the current implementation of Drum-Boy, scripting an algorithm in this "language" still requires that it be written using Lisp syntax.

Collaboration Model

This architecture of transformation algorithm implementation allows the designer to think in terms of building blocks that can be combined freely. This shifts the focus from thinking about algorithms and parameters to thinking about functional blocks. It allows for an easy collaboration with non-programmer musicians to give advice on creating higher level transformers.

In the project, Alex Rigopulos, a drummer (but not a Lisp programmer), was involved with the actual development of the transformer instances. The layer of abstraction proved extremely effective in setting up the collaborative environment. By allowing direct access to the scripting of algorithms an expert can transfer his or her knowledge to a system without risking the information being diluted or unintentionally changed by an intermediary (i.e., the system programmer).

His approach to the problem was top-down at first. Since we had already selected the adjectives, he broke down the adjectives into characteristics and modifications required on the characteristics to make the appropriate transformations. For example, from "harder," he derived "instrumentation: soft ones to harder ones" and "over all volume: make louder." Another example, "graceful," required a modification "flutter." The "flutter" transformer ended up being a complicated structure that took in an analyzer output, and chose between the many primitive transformers of which the "flutter" transformer was composed.

Writing computer code to make the actual modifications was the programmer's job; the musician could "script" in an easier language when and how the code gets executed without writing a real computer program.

While this scripting feature is not intended for the end-user (although it could be), an architecture that makes accumulation of additional knowledge into a system easy leaves it room to grow. (But at the same time, such a system is never finished since it can always be made better!)

3 Implementation

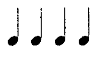
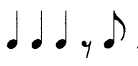
3.1 Hyperlisp

Drum-Boy runs under the Hyperlisp environment, created by Joe Chung, at the MIT Media Laboratory [Chung91]. The environment is an extension of Macintosh Common Lisp, with a scheduler capability and MIDI input and output handling. My code uses the Common Lisp Object System, the language in which the whole Drum-Boy application is written. In the development system, a Yamaha SY99 is used both as the input device (MIDI keyboard) and as the output device (MIDI tone generator).

The Hyperlisp environment supplies several layers of abstraction in which messages are passed. When a drum-boy object is initialized, it is set up as the handler to MIDI input. The drum-boy object, upon receiving the MIDI event, either changes its current mode (active only while a function key is pressed), or passes the MIDI event to its current mode object. The behavior of the drum-boy object in response to the MIDI input is defined in a table which is indexed by the MIDI opcode (note-in, note-off, etc.) and the operand (note number). The key assignments can be easily modified by altering this table. The mode object is in charge of making modifications to the current pattern as specified by the user.

3.2 Data Representation

It is well established that only onset times matter in our conception of rhythm, and not the duration of each note[18]. The data structure of rhythm patterns in Drum-Boy reflects this view and stores only the instrument, the onset times from the beginning of the measure, and the velocity (relative strength or loudness of the note). Being a prototype system,

[18] Actually, the onset times can trick one into thinking that the duration is different from what is actually played. For example, after counting 1,2,3,4, when one hears someone clapping  , all the four claps will be perceived to have the same duration. However, when he hears  , the last clap will be perceived as having a shorter duration[Johnson-Laird91a].

efficiency of calculation of transformations was not taken into account for designing the data structure; therefore, it is a simple sorted list of all events, an event being defined by the above three parameters. Given that the correct reader/accessor abstraction is used, the exact data representation should not matter to the rest of the implementation. More efficient data representation such as a hash table can be implemented instead; however, the optimization of the software is outside the scope of this thesis.

3.3 Object Definitions for Representative Classes

Drum-Boy is entirely written in the Common Lisp Object System (CLOS) environment. Some of the representative object classes are shown as follows:

- drums
- components (consists of groups of drum objects; linked to corresponding database)
- drum-machines (consists of groups of component objects)
- modes (the internal "handlers" to MIDI events. Modes are eventually responsible for deciding what to do with incoming MIDI.)
- transformers / analyzers (transforms patterns as specified by adjectives)
- database (linked to component types)
- drum-boy (keeps track of current pattern, global parameters; passes MIDI messages to the appropriate mode objects)

4 Related work

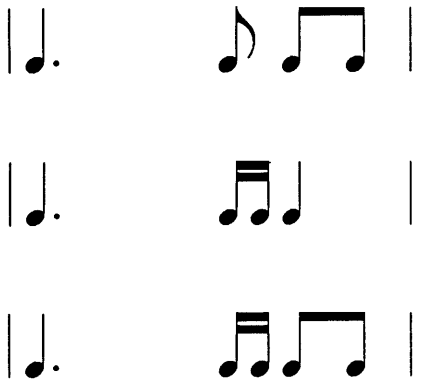
4.1 Theories of Rhythm Analysis

4.1.1 Classifying rhythms by their prototype

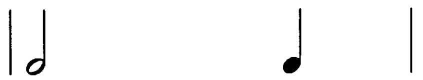
While not empirically confirmed, Johnson-Laird suggests a theory in which he states that rhythms can be classified into prototypes [Johnson-Laird91]. In his theory, there are two assumptions:

- A family of rhythm depends only on *onsets*, which can occur either on a beat or else between two beats. Onsets can occur as a syncopation or as an ordinary note.
- These onsets differ in their significance, and the family is defined by the most significant. The rank order of the significance is: syncopation, note on the beat, and all others.

His definition of family of rhythms asserts that the following three rhythms in 3/4 belong in the same family.

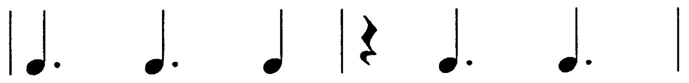


The family is based on the following prototype.



or I note other note I

Similarly, a Bossa Nova rhythm,



can be represented as I Note Sync Other Note I Other Note Sync Other I.

Johnson-Laird's hypotheses suggest that one can construct a metric to predict the perceptual distance between two rhythmic patterns. He does not explicitly state how his hypotheses apply to polyphonic rhythms, and he does not attempt to define a quantifiable metric. Therefore, applying his theory to Drum-Boy is not a strightforward process.

4.2 Automatic Parsing of Rhythm

Rosenthal succeeded in rhythmically parsing a stream of MIDI performance data [Rosenthal92]. His program, Machine Rhythm, interprets the MIDI data and constructs "an account of the rhythmic structure of the performance, which is isomorphic to the description implicit in a musical score." The MIDI data first goes through the *preprocessing* stage, where the data is grouped into chords and voices. Then the *startup* module takes over, constructing rhythmic hypotheses, structures that describes the rhythmic organization of the piece -- the time-signature and the position of the bar-lines. The *extender* module constructs hypotheses for the later events of the performance. Since there is often more than one method to parse a rhythm, the startup and extender modules suggest multiple hypotheses at each step. The exponential growth of the branching is kept under control by pruning the tree at each stage so that the total number of leaf nodes does not exceed a limit.

4.3 Control of Musical Fragments in Multidimensional Space

In Bounce [Machover92b], a piece by Tod Machover for hyperkeyboards (Yamaha Disklavier Grand and SY99 synthesizer), one of the sections uses a mode of interaction between the computer and the performer in which the performer controls the musical texture created by the Disklavier[19].

The performer uses the keys on the synthesizer to control the musical parameters: harmonicity of the pitch set, rhythmical complexity, overall loudness, and note range. Each octave corresponds to one parameter, or axis, and the twelve keys in each octave specify the parameter value between one and eleven. Note that the overall loudness parameter is controlled by the velocity with which the loudness control key is hit. The mapping here is more intuitive, and the performer has a finer control over the parameter (key-on velocity ranges over a value between 1 and 127, as opposed to the eleven discrete values one can choose by pressing a key in an octave).

[19]A Disklavier is a piano equipped with computer-controlled solenoids to play the notes automatically, and sensors on each key for monitoring when and how the keys were played. It uses MIDI for input and output.

A table with twelve entries was prepared for the pitch set as well as the rhythmic template. When a new parameter is specified, the new pitch set or rhythmical template is replaced in the currently playing pattern. The pattern is then processed to accommodate the other parameters - the overall loudness and the note range.

This set-up offered the performer power in two distinct ways. First it made it possible for the piano to produce sounds that a human performer is not capable of producing. The dense textures were populated both in time and space; a human performer can only play a certain number of notes at once, and only strike so many keys in a given time interval. The physical structure of the Disklavier and the control from the computer removed these limitations. Secondly, the performer was able to control the sound producing process at a higher level.

This removal of the constraint to produce the right notes left some performers uncomfortable, since concert pianists, for example, usually always produce each note themselves, and the cognitive process (of playing the notes) is tightly associated with the act of playing the piano. However, for a novice pianist (or someone who does not play the piano) this removal of the constraint and the cognitive load is a welcome change.

Interesting observations were made during the rehearsals at concerts with two different pianists (pianist A and pianist B). First, it seemed that concert pianist A was not used to giving up total control of the instrument, down to the individual note level. Pianist A took a long time to get used to the unconventional instrument interface, and spent a lot of time trying to figure out the exact outcome of a manipulation; he tried to form an exact mental picture of the "mapping" between the input and output of the system. Pianist B, on the other hand, tried to explore the space of the music that the system was capable of producing, and "composed" during the section successfully. It is interesting to note that Pianist A was superior to Pianist B in terms of how accurately he played the piano (in other sections where the piano was actually played).

When beginners used the system (informally, just for fun), they did not approach the system in the same way. Since precisely controlling the input device was hard for them anyway, their main focus was not in figuring out the system, but simply in getting something out that sounded good. The interaction with the system itself was a pleasurable experience because the system gave them the power to create musical segments that they could not play by other, more conventional, means. The musical output was not always intended, but nevertheless, beginners were just as good, or, in some cases, even more successful than the concert pianists in making "interesting, pleasurable" music.

5 Conclusion

The primary goal of the research prototype, Drum-Boy, was to create a system where a user can use high level controls to manipulate musical material. The system specializes in rock and Latin drumming styles, although by expanding the database and supplying more knowledge into the transformation system, additional styles can be incorporated.

In this thesis, I proposed several design principles (and analyzed some that others have proposed) useful in developing tools that aid a creative process. Becoming a good designer means that the knowledge-base for what makes good design is expanded. I agree with the constructionist point of view -- learning by doing, or learning by designing . It was through the design of Drum-Boy that I discovered these principles.

Perhaps a more thorough list of these design principles can be obtained by designing other sophisticated systems. Gargarian suggests that a designer develops a *design environment*, "the conceptual interface between the internal world of the designer and the external world in which the emerging artifact resides [Gargarian93]." When the rules or patterns that form the design environment (i.e., the designer's skills or knowledge) are published, designers can "learn without designing" (although this may not be the most effective way to learn).

5.1 Applications in Other Domains

I claimed earlier that a structure similar to that of Drum-Boy can be used in building interactive systems in other domains. I have already mentioned a video story-boarding system. This hypothetical story-boarding system works with cartoon characters, since the problem of interpolating between example drawings has been addressed by other researchers and has proved successful [Librande91]. The physical interface would have to be different, since working with images requires a spatial canvas, as opposed to a temporal one, and the piano keyboard is not suited for communicating spatial information in more than one dimension. The user would coordinate the characters in the story and would be able to control the way the characters are visually represented, using both quick, random-access methods and the transformative (interpolation) methods.

A more challenging instrument is that which will aid people in giving presentations.[20] The present day hype regarding "multimedia" presentation systems gives rise to many scripting environments that aid the creation of these presentations. LCD displays for projecting computer screens have been developed so that a marketing representative can bring his computer to his client and show pretty images and video, all synchronized to clever sound effects.

The essence of a presentation is that it is live, much like a musical performance. By being live, the presenter can look at the audience to see how much they are following or understanding the material. The presenter can change dynamically what slides to show and how much time to spend on each item. And it is this versatility and interactivity that makes a live presentation more effective than a videotape.

While lighting control and sound effects at business presentations seem distracting, I conjecture that they will become more widely used in the future, for it is not the presence of such effects, but their novelty, that distracts the experience. The reason these effects are not used widely is that current technology makes it difficult for one person to control all the variables. (And bringing a whole technical crew to a presentation is overkill in most cases.)[21] The temporal and "performance" nature of a presentation leads me to believe that techniques discussed in this thesis are relevant to the development of such systems.

5.2 Enhancing the Current System

5.2.1 Providing a Higher Quality Database

The present system defines only four component classes[22], and several instrument components share the same database. The sharing of the database reduced the number of total entries, thereby expediting the preparation phase of the project. In the final system,

[20]The idea of a "presentation hyperinstrument" is not an original one. Personal communication with Tod Machover made me think about the subject in more detail.

[21] Where does this discussion lead to? Presentations as theater!

[22]See section 2.4.4 for a discussion on components.

perhaps the number of component classes should be increased, reflecting the fact that the different instruments have their own functions. For example, in the present system, the crash cymbal shares the monophonic instrument database with a ride cymbal. The crash cymbal and the ride cymbal are functionally very different, so separate databases will likely enhance the quality of the musical output of the system.

In addition, the present database was obtained from a combination of two sources:

- processing data from novices tapping on the MIDI keyboard
- professional percussionist playing on a MIDI drum pad

The data-set from the professional percussionist was not large enough to stand on its own; therefore, the data from the novice was kept in the system to maintain the size of the database. The data from the novice was quantized to eliminate the deviation from the notated rhythm, forcing the rhythms to sound mechanical. Using a high quality database will undoubtedly lead to better performance by the system.

5.2.2 Empirically-based Modeling of Behavior

[Landauer91] suggests that one way in which HCI can help design an interactive system is by modeling representative users. For example, in designing a spelling corrector, where the industry norm seems to be to use ad hoc metrics of differences between strings, collecting data on how often typical users omit, substitute or transpose particular letters or particular position specific n-grams would allow the designer to test how well a correction scheme works by simulating its use.

Intelligent Quantizer

In Drum-Boy, empirically-based simulation can be used to fine-tune the quantizing function. Presently, in invoking the quantizer function, the user has to specify the grid onto which the notes will be automatically aligned. By collecting data on how users deviate from the precise temporal location, it may be possible for the system to "intelligently" guess where the user had intended the notes to go.

Other "intelligent" ways to extract a metrical score from unquantized data have been explored by a number of researchers. For a connectionist approach see [Desain and Honing91].

5.2.3 "Humanizing" Drum Patterns

Since Drum-Boy has no internal model of human drumming, the transformation process occurs at the notationally quantized level. However, it is well known that perfectly quantized rhythm is often perceived as mechanical and unnatural. Thus, "humanizing" becomes one of the useful features in this system.

The data captured from human performance of rhythm patterns (see above) can also be used for "humanizing" rhythm patterns. Jeff Bilmes separated individual deviations (the difference between the performed rhythm and the rhythm suggested by its notated, or quantized, timing) and tempo changes in a performance[Bilmes93]. His experiments showed that quantized performance with tempo changes sounded mechanical, whereas an unquantized performance with the tempo changes removed did not. Bilmes defined a neural net learning algorithm to correlate between individual deviations and the rhythm pattern. A result of such modeling could prove to be effective in humanizing machine generated rhythms.

[Gabrielsson74] analyzes several examples of a pianist and a percussionist performing notated rhythms. While his analysis is by no means complete, he lists several observations which can be directly applied and implemented The following is a sampling of some of his observations[23]:

- in a sequence of an eighth-note followed by two sixteenth-notes within a beat, there is an obvious S-L (short-long) relation between the sixteenth-notes. i.e., the first sixteenth note is shorter than the second. However, there is a L-S (long-short) relation on the eighth-note level. i.e., the first eighth-note is longer than the duration of the two sixteenth-notes summed up.

[23]These observations were made from one of his experiments where a percussionist played 13 notated rhythms using drum sticks on a side drum, the scenario most relevant to the context of Drum-Boy. The four observations were also seen in another experiment where a percussionist played a bongo drum using his hands.

- when beats are divided into two eighth-notes, the deviations are unsystematic and too small to be regarded as a deviation from an EQ (equal duration) relation.
- punctuations (the dot of the dotted eighth) are "sharp." While the ratio implied by the notation between a dotted eighth-note and a sixteenth-note is 3:1, the performance showed the ratio to vary between 4.93:1 and 3.53:1
- the first sound event of the measure had a high peak amplitude.

Such positive correlation between the rhythmic material and the notes' temporal deviations from the norm suggests a strong possibility that a convincing "humaninzing" algorithm can be developed.

5.3 What I Learned

An open-ended problem, such as the design of Drum-Boy, is in fact a meta-problem: the designer does not know in advance what all the constraints and requirements will be. In the process of design, it is therefore often necessary to reformulate the constraints and to propose different solutions so as to better understand the problem itself. For this reason, several different versions of Drum-Boy exist, sharing little more in common than their rhythm playback engine. In fact, it is likely that the amount of computer code that was not used in the final version matches or exceeds that which was used.

My discussion on features that characterize good tools in a creative environment[24] applies to the environment in which I did my software development. Designing an interactive system and then implementing it are creative processes that certainly require specialized tools. For example, the ability to quickly test out ideas (and to revert back to the previous state should the idea turn out to be bad) proved to be crucial in our development environment.

This thesis poses several questions that remain unanswered. Much is still not fully understood about human or machine rhythm perception, and it has been difficult to extrapolate those results that do exist in the field of music cognition, since they were obtained in controlled environments (for example, monophonic rhythms and/or quantized

[24]See section 2.3.

rhythms). Our approach explored different techniques without giving ourselves the overwhelming task of unearthing the fundamentals.[25]

Multidimensional modeling of perception is very useful as a tool for communicating a representation without explicitly describing the instance itself. While it is difficult to visually represent a space that has more than three dimensions (or even two on a flat computer screen), understanding the concept of "navigating" in a mathematical space is not so difficult. As with any type of modeling, only the essence is captured, and the details are discarded. In fact, this feature of multidimensional modeling worked in our favor for the design of Drum-Boy, since our general goal was to hide such details from the user.

Beginners and experts have different requirements for their creativity tools. The difference is not just a quantitative one: since their goals and priorities are different, the tools must support different functionalities, at different levels of interaction. Experts need a tool to quickly and easily realize their ideas. On the other hand, beginners, since they can't always imagine what they want, would benefit from a tool that would help them find ideas.

We believe that in the design of Drum-Boy, we accomplished our goal of addressing different levels of expertise in our audience.

---

[25]This approach is analogous to the engineer's approach to solving problems.

> Often, laboratory results, and even well established theories simply can not be applied to real-life problems because of the complexity of the context for application. In such cases, models that explain or predict the outcome are constructed separately from the underlying basic scientific theories. For example, civil engineering has its own models of runoff and groundwater. In principle, such an understanding can be obtained from the results of studying the physics of fluid flow in porous materials. However, in the context of the application (in this case, for example, for building bridges), the civil engineering models are in the right form. [Brooks91]

Appendix A: Drum-Boy User's Manual

This section documents the directions for using Drum-Boy

A.1 Functional Overview

Drum-Boy is an interactive percussion system that allows the user to perform complex rhythmic material by using a MIDI keyboard. Its storage capabilities make it ideal as a rhythmic compositional tool as well. It addresses a wide audience by supporting multiple layers of user interface.

At the note level, the user can enter and edit each note in a measure. This level of interaction is ideal for users who have the understanding of a drum kit. The user can use these functions to program a pattern or play the instrument in real time.

At the phrase level, the user can call up component level phrases (Components are functional groups of instruments. For example, an high pitched conga and a low pitched conga form a component). The phrases include fills, in which a user can choose to apply to the whole pattern, or only for selected instruments. This level of interaction is useful in a performance context.

At the pattern level, the user can apply transformations described by adjectives, such as "calm," "complex," or "mechanical." This level of interaction is ideal for novice users who "have the ears" to judge material when they hear it but not the ability to construct the material by assembling it from scratch.

The system plays back a current pattern in a loop until the user engages in an interaction with the system, conducting a change in the pattern. The change can be specified in any of the three above mentioned levels. The user can either "lock in" the change, or can decide to throw away the change, as all changes are temporary by default. A multiple-step undo facility helps the user navigate back up to 20 steps.

In the auto-variation mode, the current pattern is algorithmically modified each time it is repeated. The algorithm is constructed so that the pattern does not drift away from the original pattern. A song like structure can be programmed in the auto-variation mode. For example, the variation conductor can specify a fill every four measures.

The patterns can be stored in memory and triggered by pressing the corresponding keys.

The graphical feedback shows the user what command has been selected. It also has a dynamic display of the current pattern. Dots corresponding to each drum note are highlighted when the note is played.

A.2 Hardware Requirements

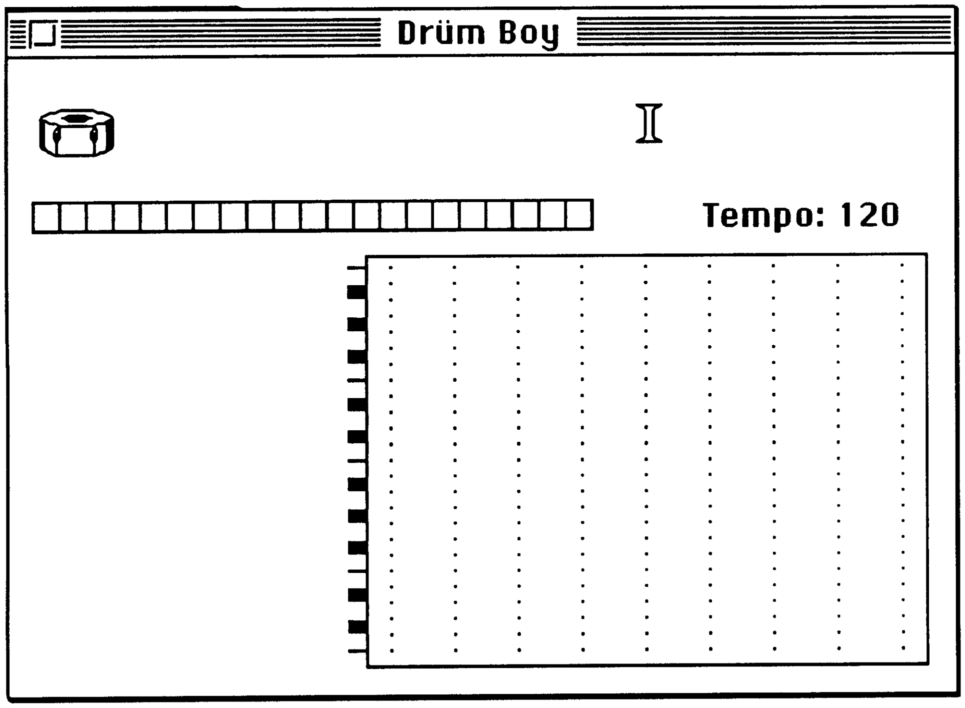
Drum-Boy runs on a Macintosh IIci or faster machine, with 8MB or more RAM. Macintosh Common Lisp 2.0 and MIDI Manager are required. A MIDI interface and cables are required. An SY99 or equivalent (MIDI keyboard controller with two pedals, modulation wheel, channel after touch, and 76 keys; and a drum tone generator are required. Currently supported tone generators include the HR16, and those that adhere to the General MIDI standard).

It is suggested that the keyboard label (file name: keyboard-label, file format: PICT) be printed out and attached to the MIDI keyboard controller.

A.3 Starting the System

The software can be started up in one of two ways. The Drum-Boy application can be double-clicked and launched directly from the Finder; or if the application does not exist, the file ">>drum-boy-load.lisp" can be dragged onto MCL 2.0, opening the loader file within the Lisp environment. In this case, Drum-Boy can be launched by evaluating the ">>drum-boy-load.lisp" buffer by selecting "Eval Buffer" from the Eval menu.

When the application is launched correctly, the following window appears:



A.4 Functionality

A.4.1 Getting a pattern running

From the blank state, a pattern can be entered into the current pattern buffer by using the following methods:

1) Pressing the Storage keys, if there are patterns stored in the corresponding key.

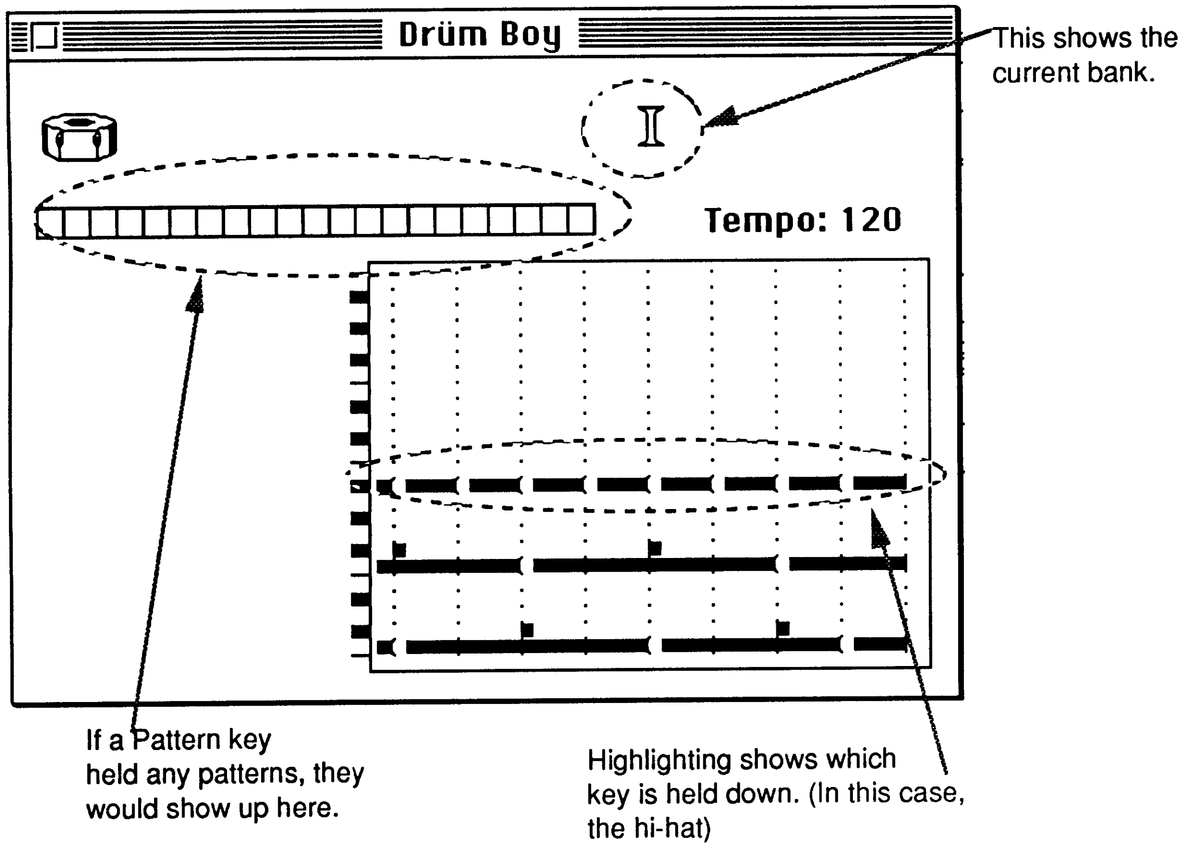
2) Pressing the Drum key. A database entry corresponding to the drum component, the current bank and the velocity at which the key is hit.

3) While holding the Insert Edit or the Replace Edit key, hit the Drum key in tempo. The first note of the measure has to be the last note entered. The tempo is automatically calculated.

The tempo can be changed by controlling the modulation wheel. The range of tempo supported by Drum-Boy is 40 to 167.

A pattern can always be stopped by pressing the Start / Stop key: ▶ ■

When there is a pattern running, the Drum-Boy window will look like this:


Drüm Boy
This shows the current bank.
Tempo: 120
If a Pattern key held any patterns, they would show up here.
Highlighting shows which key is held down. (In this case, the hi-hat)

A.4.2 Editing a Pattern

Drum-Boy requires no interaction using the mouse or the keyboard (except for performing commands such as changing the drum tone generator setting and loading in pattern data files). Editing a pattern requires the user to press a command key, and while the command key is pressed, to press an additional key that corresponds to the drum or the drum component to which the command is applied. Each editing command, in a sense takes on an "Apply xxx to the yyy drum (or drum component)" syntax.

Note that these commands are effective only while a pattern is playing. Otherwise the following icon is displayed on the screen, denoting an error:

Component Delete / Drum Delete

These commands delete notes from the current pattern as they are played by holding down the Delete Key while a corresponding Drum Key is pressed.

Quantize

The quantize function aligns the specified notes to "grids" as suggested by musical notation. The quantization grid has to be specified explicitly by the user. If the Quantize Key is hit above a certain velocity, the grid becomes finer; if the key is hit below a certain velocity, the grid becomes coarser. The grid value is shown in the graphical display.

Replace Edit / Insert Edit

These commands allow the user to record the drum hits to the current pattern in real time. Simply tap the Drum Keys while one of these command keys is held down. Replace Edit deletes any hits of the same drum already in the current pattern as it is played. Insert Edit keeps all drum hits, and layers the new notes onto the existing pattern.

A.4.3 Launching phrases

While no command key or pedal is pressed, the Drum Keys trigger a phrase to be played. The phrase replaces existing track played by the same drum component in the current pattern. Without any command key or pedal, the replacement is temporary - when the drum key is released, the track that existed before the replacement is restored. To make the change permanent, the Lock-In key has to be pressed while the replacement material is being played.

The Delay Release pedal acts similarly to the sustain pedal in a piano. The pedal locks in the new material temporarily by delaying the release of the drum key. While the Delay Release pedal is pressed, releasing the Drum key will not restore the original material.

While the Event/Phrase pedal is pressed, the Drum keys will not trigger phrases. The Drum keys will, however, trigger a note corresponding to the drum at the velocity at which the key is pressed. The existing track is not muted, and the note is not recorded anywhere. This feature is intended for real-time playing.

Selection of Phrases

The actual phrase triggered by the Drum key is a function of the drum component, the current bank, and the velocity at which the key is pressed. Each component is associated with a database. There are four kinds of databases, corresponding to the four component classes: monophonic percussion, polyphonic percussion, open/closed hi-hat, kick-snare. Each database entry is indexed using a two dimensional key. In the default configuration of Drum-Boy, the two axes correspond to the "convoluted-ness" and the complexity of the material. The convoluted-ness is specified by the current bank; there are three discrete levels of convoluted-ness. The current bank is shown in the graphical display. The complexity is mapped to the velocity.

Fills

Fills can be triggered in one of two ways. A general pattern fill can be triggered by using the Fill key. When this key is pressed, the fill pattern corresponding to the velocity at which the key is hit is inserted to the current pattern. The fill lasts until the end of the measure, so if the key is hit at the end of the measure, the effect may not be perceivable.

The alternate method is by using aftertouch. By pressing on the already pressed Drum key harder, MIDI aftertouch data is sent. The system detects a rapid rise in the aftertouch data, and depending on the peak after touch value, selects the fill pattern from a component fill database. Since channel after touch is used, the fill is triggered for all components whose keys are pressed (due to hardware limitation, individual key pressure can not be detected). The harder the aftertouch pressure, the more complex the fill selected will be.

A.4.4 Transforming a Pattern

Five pairs of adjectives are assigned to the Transformation keys. Pressing each of these keys calls up one of the two transformers assigned to the key. If the key is pressed with a velocity of more than 64, the first transformer is called; if the key is pressed with a velocity of less than 64, the second is called. The difference between the actual velocity and 64 is used as a parameter to the transformer specifying how much transformation is to be performed. (See appendix B for the actual transformation algorithms.)

A.4.5 Storing and Retrieving Patterns

A pattern can be temporarily stored in the Pattern keys by pressing the Store key, and the Pattern key. As with other command keys, the Store key has to be held down when the Pattern key is pressed. The graphical display will show which Pattern keys presently contain material.

The stored patterns can be retrieved by pressing the Pattern keys. If a Pattern key is hit with a velocity of less than 64 the pattern is queued up. The pattern will follow the last pattern on the queue, or if the queue is empty, the pattern will be played after the current measure finishes playing. If a Pattern key is hit with a velocity of more than 64, the pattern immediately replaces the current pattern.

A.4.6 Saving and Loading Stored Patterns

The patterns stored on the Pattern keys can be saved as and loaded from a disk file. Since this action involves specifying the file name and disk folder location, the standard Macintosh interface is used. The command is activated via the Drum-Boy menu.

Selecting "Save Patterns..." will prompt the user to enter a file name. A MIDI file is created with the same name. The mapping of the instruments to the note number corresponds to the SY99 tone generator mapping.

Similarly, "Load Patterns..." will prompt the user to select a MIDI file, which will be read into the system. The patterns are mapped to the Pattern keys.

If a different program is used to construct the MIDI file, the following guideline must be followed:

Use consecutive tracks, each track corresponding to an entry on the Pattern key. Only the first measure is used for each track. Do not leave a track empty. Only the first 18 tracks are registered (since there are only 18 Pattern keys). If there are less than 18 tracks, only the corresponding Pattern keys are filled.

Use the following mapping for specifying the drums.

Drum	Note	Drum	Note
Kick	A1	Closed Hat	A2
Snare	C#2	Open Hat	B2
Low Tom	C2	Clave	E6
Mid Tom	D2	Cowbell	D#6
Hi Tom	F1	Low Agogo	C6
Low Conga	D5	Hi Agogo	C#6
Hi Conga	D#5	Timbale	F5
Low Block	F6	Shaker	Bb5
Hi Block	G6	Ride	F#3
Tambourine	G#2	Crash	C#3

A.5 Selecting a Different Hardware Configuration

A different hardware setup (HR16, General MIDI) can be accommodated by selecting the corresponding set up from the "Mapping" hierarchical menu.

A.6 Key Function Mapping

Function	Key(s)	Function	Key(s)
Pattern keys	E0	Re-accent note	C#3
Store	B1	Re-accent track	D3
Start/Stop	C2	Delete drum	D#3
Undo	C#2	Delete component	E3
Transformation	D2 - F#2	Shuffle	F3
Variation	G2 - G#2	Fill	A3
Replace Edit	A#2	Lock in	B3
Insert Edit	B2	Drums	C4 -
Quantize	C3		

Appendix B: Transformation Algorithms

B.1 Primitive Transformers

B.1.1 Transformer Classes

`tempo-changer` *Slot variables: target-tempo, ceiling-p, floor-p*
Changes the tempo so that it approaches the target tempo. ceiling-p and floor-p parameters are used to specify whether the change is one-way only (for example, if tempo is already slower than the target tempo, you may not want to make the tempo any slower).

`shuffler` *Slot variable: target-shuffle*
Shifts the sixteenth-note up beats. The current shuffle is stored so that when new phrases are added, the shuffle will be retained.

`comp-adder` *Slot variables: target-components, range*
Random component database entry are added to the pattern. The list of candidate components are specified by target-components, and range specifies which of the first n entries in the database might be added.

`comp-remover` *Slot variables: target-components*
Removes a random component (from the list).

`comp-track-replacer` *Slot variable: comp-track*
Replaces or adds a specific component track.

`replacer`
Slot variables: search-string, replace-string, gridded-search, event-velocity, vel-rand-range, max-replace-num
Searches for an occurrence of a certain pattern and replaces with another. gridded-search constrain the search of the patterns to those occurring at specific points in the measure.

`note-adder`
Slot variables: find-site, epsilon, add-drum-candidates, velocity, only-when-exit
Adds notes to a pattern. add-drum-candidates specifies the drum. only-when-exist can constrain the addition to sites where other hits exist.

`phaser` *Slot variables: phase-component, phase-degree*
Introduces phase-shift of a component.

`non-quant-replacer`
Slot variables: search-components, search-template, replace-template
Searches for a pattern and replaces with another. Can specify the drum in case of multi-instrument component. Velocity can be specified either in terms of the original hits or absolute velocity values.

`reorchestrater`
Slot variables: source-components, dest-components, dest-must-exist, dest-must-not-exist
Changes the instrumentation. Works at component level. Can constrain the components involved.

`drum-reorchestrater`
Slot variables: source-drums, dest-drums, dest-must-exist, dest-must-not-exist
Changes the instrumentation. Works at drum level. The drums involved can be constrained.

`velocity-randomizer` *Slot variables: delta-max, components*
Randomizes the velocity of the specified component(s) within the specified range.

`velocity-changer` *Slot variables: target-velocity, components*
Changes the velocity of specified component(s) toward a target value.

`velocity-compander`
Slot variables: mean-velocity, approach-factor, components
Compresses or expands the range of velocity of specified component(s).

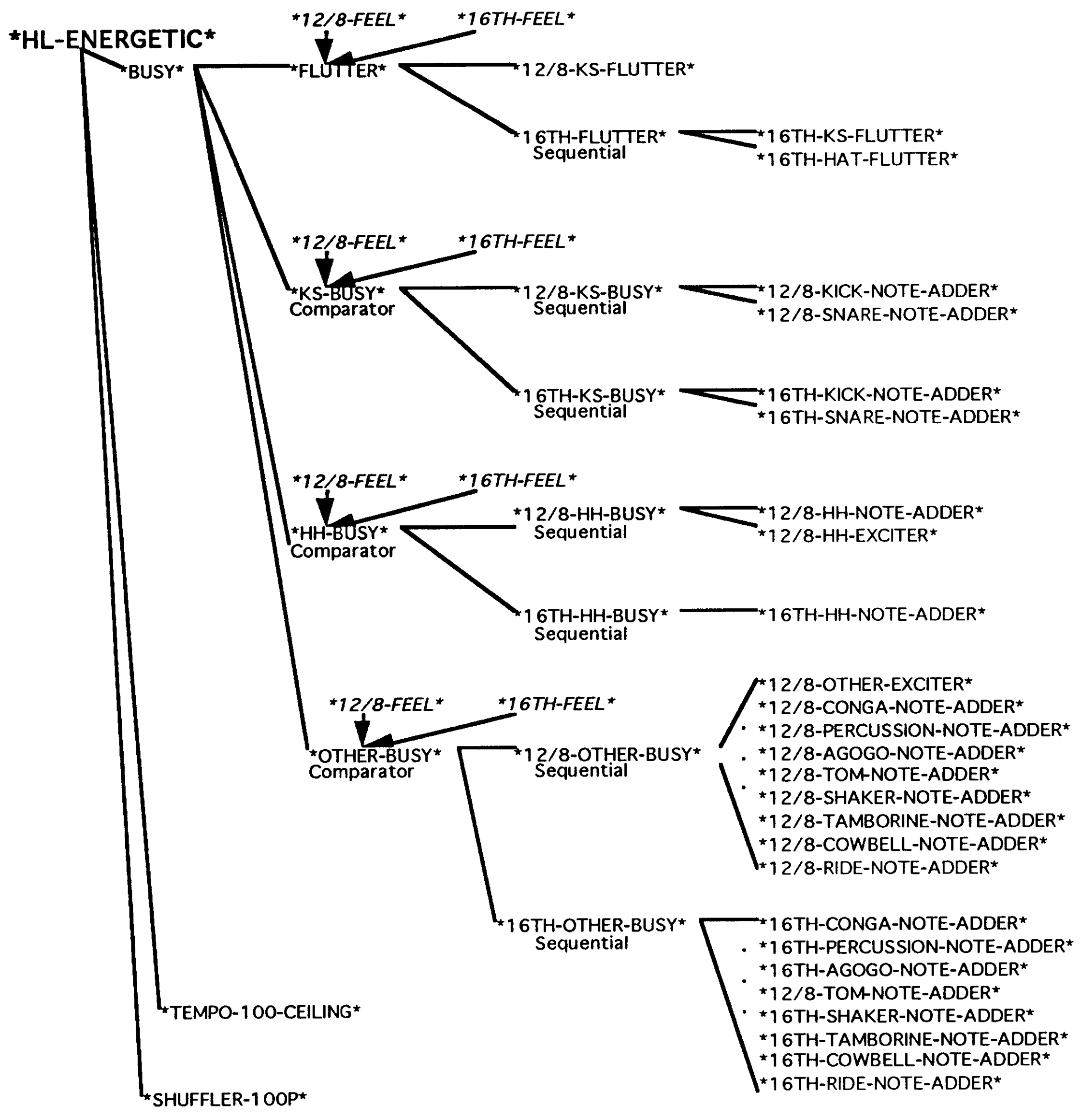
`quantizer` *Slot variables: note-epsilon-list, search-components*

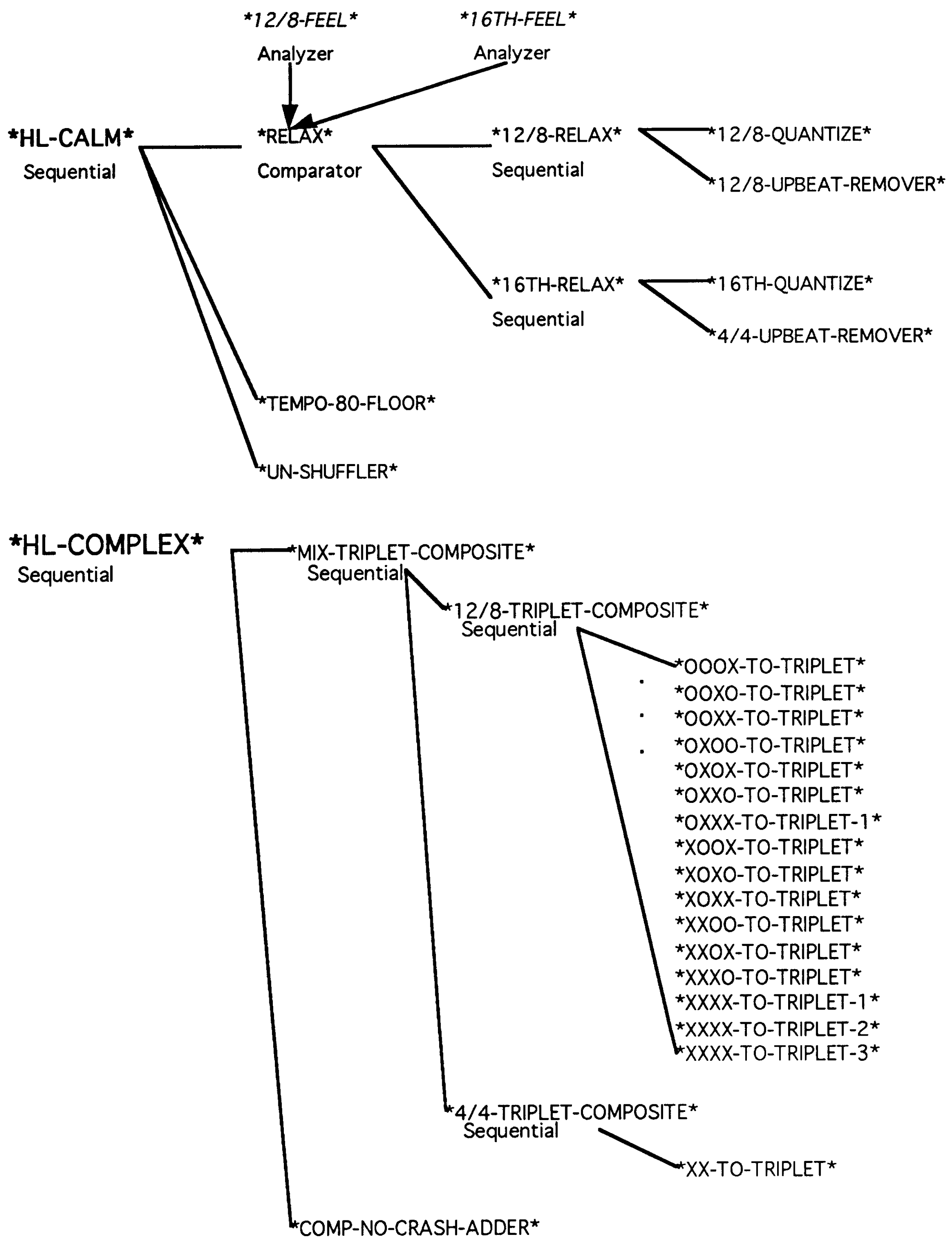
"Corrects" the timing of hits by aligning the timing values to a specified grid.

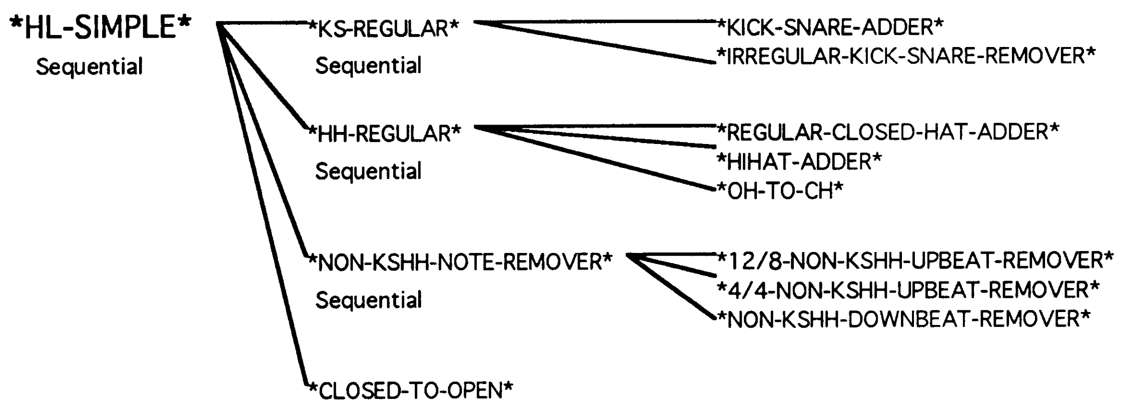
B.2 High Level Transformers:

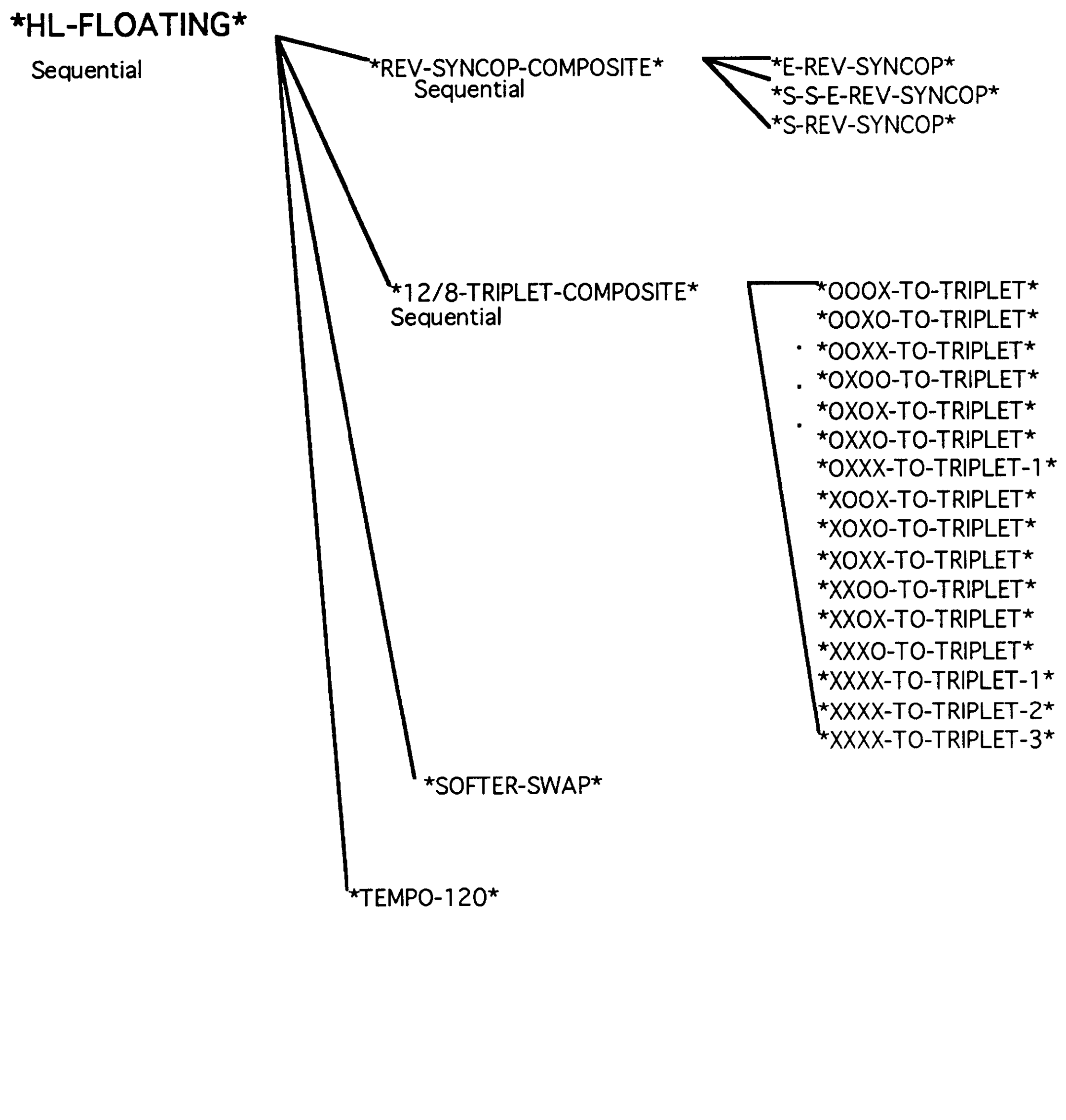
Rather than specify all the parameters for the primitive instances, only their names (hopefully meaningful enough) will be used in describing the higher level transformers.

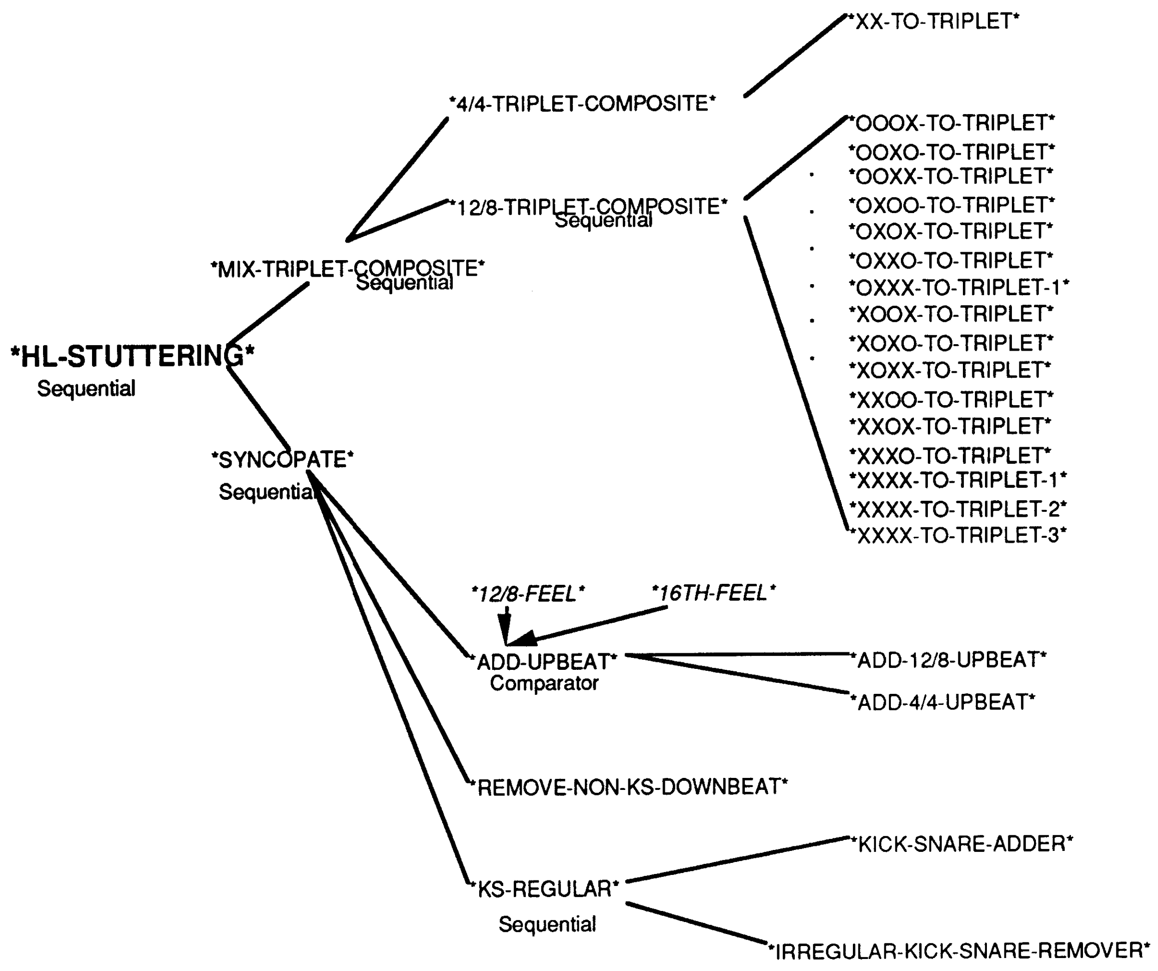
The transformers and analyzers have names enclosed in asterisks. Analyzers are italicized. The leaf nodes in the tree diagram are primitive transformers. Higher level transfomers have the type of combination (sequential, comparator, etc.) specified below it.

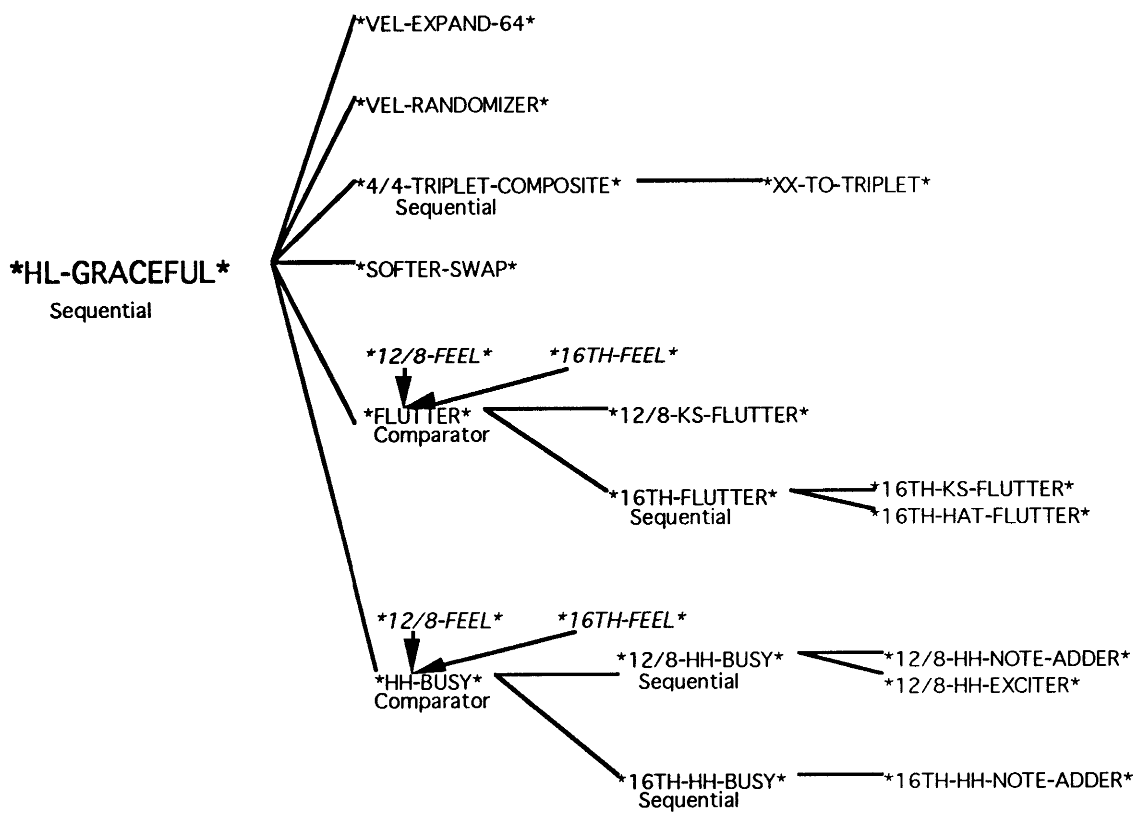

HL-ENERGETIC
BUSY
12/8-FEEL
16TH-FEEL
FLUTTER
12/8-KS-FLUTTER
16TH-FLUTTER
Sequential
16TH-KS-FLUTTER
16TH-HAT-FLUTTER
12/8-FEEL
16TH-FEEL
KS-BUSY
Comparator
12/8-KS-BUSY
Sequential
12/8-KICK-NOTE-ADDER
12/8-SNARE-NOTE-ADDER
16TH-KS-BUSY
Sequential
16TH-KICK-NOTE-ADDER
16TH-SNARE-NOTE-ADDER
12/8-FEEL
16TH-FEEL
HH-BUSY
Comparator
12/8-HH-BUSY
Sequential
12/8-HH-NOTE-ADDER
12/8-HH-EXCITER
16TH-HH-BUSY
Sequential
16TH-HH-NOTE-ADDER
12/8-FEEL
16TH-FEEL
OTHER-BUSY
Comparator
12/8-OTHER-BUSY
Sequential
12/8-OTHER-EXCITER
12/8-CONGA-NOTE-ADDER
12/8-PERCUSSION-NOTE-ADDER
12/8-AGOGO-NOTE-ADDER
12/8-TOM-NOTE-ADDER
12/8-SHAKER-NOTE-ADDER
12/8-TAMBORINE-NOTE-ADDER
12/8-COWBELL-NOTE-ADDER
12/8-RIDE-NOTE-ADDER
16TH-OTHER-BUSY
Sequential
16TH-CONGA-NOTE-ADDER
16TH-PERCUSSION-NOTE-ADDER
16TH-AGOGO-NOTE-ADDER
12/8-TOM-NOTE-ADDER
16TH-SHAKER-NOTE-ADDER
16TH-TAMBORINE-NOTE-ADDER
16TH-COWBELL-NOTE-ADDER
16TH-RIDE-NOTE-ADDER
TEMPO-100-CEILING
SHUFFLER-100P

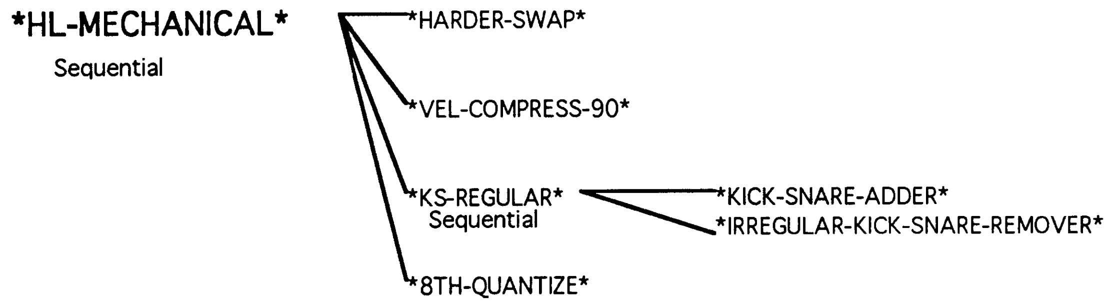

12/8-FEEL
Analyzer
16TH-FEEL
Analyzer
HL-CALM
Sequential
RELAX
Comparator
12/8-RELAX
Sequential
12/8-QUANTIZE
12/8-UPBEAT-REMOVER
16TH-RELAX
Sequential
16TH-QUANTIZE
4/4-UPBEAT-REMOVER
TEMPO-80-FLOOR
UN-SHUFFLER
HL-COMPLEX
Sequential
MIX-TRIPLET-COMPOSITE
Sequential
12/8-TRIPLET-COMPOSITE
Sequential
OOOX-TO-TRIPLET
OOXO-TO-TRIPLET
OOXX-TO-TRIPLET
OXOO-TO-TRIPLET
OXOX-TO-TRIPLET
OXXO-TO-TRIPLET
OXXX-TO-TRIPLET-1
XOOX-TO-TRIPLET
XOXO-TO-TRIPLET
XOXX-TO-TRIPLET
XXOO-TO-TRIPLET
XXOX-TO-TRIPLET
XXXO-TO-TRIPLET
XXXX-TO-TRIPLET-1
XXXX-TO-TRIPLET-2
XXXX-TO-TRIPLET-3
4/4-TRIPLET-COMPOSITE
Sequential
XX-TO-TRIPLET
COMP-NO-CRASH-ADDER

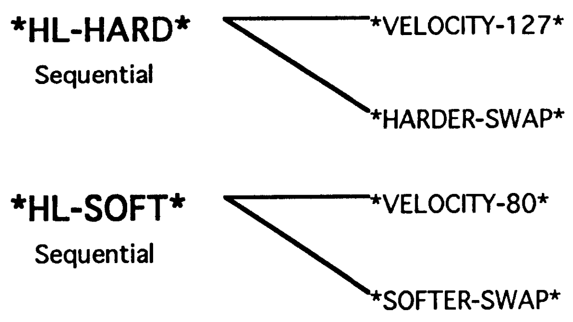

HL-SIMPLE
Sequential
KS-REGULAR
Sequential
KICK-SNARE-ADDER
IRREGULAR-KICK-SNARE-REMOVER
HH-REGULAR
Sequential
REGULAR-CLOSED-HAT-ADDER
HIHAT-ADDER
OH-TO-CH
NON-KSHH-NOTE-REMOVER
Sequential
12/8-NON-KSHH-UPBEAT-REMOVER
4/4-NON-KSHH-UPBEAT-REMOVER
NON-KSHH-DOWNBEAT-REMOVER
CLOSED-TO-OPEN


HL-FLOATING
Sequential
REV-SYNCOP-COMPOSITE
Sequential
E-REV-SYNCOP
S-S-E-REV-SYNCOP
S-REV-SYNCOP
12/8-TRIPLET-COMPOSITE
Sequential
OOOX-TO-TRIPLET
OOXO-TO-TRIPLET
OOXX-TO-TRIPLET
OXOO-TO-TRIPLET
OXOX-TO-TRIPLET
OXXO-TO-TRIPLET
OXXX-TO-TRIPLET-1
XOOX-TO-TRIPLET
XOXO-TO-TRIPLET
XOXX-TO-TRIPLET
XXOO-TO-TRIPLET
XXOX-TO-TRIPLET
XXXO-TO-TRIPLET
XXXX-TO-TRIPLET-1
XXXX-TO-TRIPLET-2
XXXX-TO-TRIPLET-3
SOFTER-SWAP
TEMPO-120


HL-STUTTERING
Sequential
MIX-TRIPLET-COMPOSITE
Sequential
4/4-TRIPLET-COMPOSITE
XX-TO-TRIPLET
12/8-TRIPLET-COMPOSITE
Sequential
OOOX-TO-TRIPLET
OOXO-TO-TRIPLET
OOXX-TO-TRIPLET
OXOO-TO-TRIPLET
OXOX-TO-TRIPLET
OXXO-TO-TRIPLET
OXXX-TO-TRIPLET-1
XOOX-TO-TRIPLET
XOXO-TO-TRIPLET
XOXX-TO-TRIPLET
XXOO-TO-TRIPLET
XXOX-TO-TRIPLET
XXXO-TO-TRIPLET
XXXX-TO-TRIPLET-1
XXXX-TO-TRIPLET-2
XXXX-TO-TRIPLET-3
SYNCOPATE
Sequential
12/8-FEEL
16TH-FEEL
ADD-UPBEAT
Comparator
ADD-12/8-UPBEAT
ADD-4/4-UPBEAT
REMOVE-NON-KS-DOWNBEAT
KS-REGULAR
Sequential
KICK-SNARE-ADDER
IRREGULAR-KICK-SNARE-REMOVER


HL-GRACEFUL
Sequential
VEL-EXPAND-64
VEL-RANDOMIZER
4/4-TRIPLET-COMPOSITE
Sequential
XX-TO-TRIPLET
SOFTER-SWAP
12/8-FEEL
16TH-FEEL
FLUTTER
Comparator
12/8-KS-FLUTTER
16TH-FLUTTER
Sequential
16TH-KS-FLUTTER
16TH-HAT-FLUTTER
12/8-FEEL
16TH-FEEL
HH-BUSY
Comparator
12/8-HH-BUSY
Sequential
12/8-HH-NOTE-ADDER
12/8-HH-EXCITER
16TH-HH-BUSY
Sequential
16TH-HH-NOTE-ADDER


HL-MECHANICAL
Sequential
HARDER-SWAP
VEL-COMPRESS-90
KS-REGULAR
Sequential
KICK-SNARE-ADDER
IRREGULAR-KICK-SNARE-REMOVER
8TH-QUANTIZE



B.3 Analyzers

There are presently two classes of analyzers.

`note-counter`
Slot variables: start-points, search-template, epsilon, search-components
Counts the number of notes occurring at each of the specified slots.

`note-matcher`
Slot variables: start-points, search-template, epsilon, search-components
Counts the number of times a specified sub-pattern occurs in a pattern.

B.3.1 Analyzer Instances

Six instances of analyzers exist in the current version:

12/8-feel
16th-feel
downbeat-feel
8th-upbeat-feel
16th-upbeat-feel
12/8-upbeat-feel

Bibliography

[Bilmes93]	Jeff Bilmes, *Timing is of the Essence: Perceptual and Computational Techniques for Representing, Learning, and Reproducing Expressive Timing in Percussive Rhythm*, M. S. thesis for MIT Media Laboratory, 1993.
[Brooks91]	Ruven Brooks, *Comparative Task Analysis: An Alternative Direction for Human-Computer Interaction Science,* John M. Carroll, ed., Cambridge Series on Human-Computer Interaction, 1991.
[Carroll91]	John M. Carroll, Introduction: *The Kittle House Manifesto*, Designing Interaction, John M. Carroll, ed., Cambridge Series on Human-Computer Interaction, 1991.
[Chung91]	Joseph Chung, *Hyperlisp Reference Manual*, available from the Media Laboratory, Massachusetts Institute of Technology, 1991.
[Desain and Honing91]	Desain, Peter, and Honing, Henkjan, *The Quantization of Musical Time: A Connectionist Approach*, 150 - 160, from Music and Connectionism, edited by Peter M Todd and D Gareth Loy, The MIT Press, Cambridge, Massachusetts, 1991.
[Downton91]	Andy Downton, *Dialogue styles: basic techniques and guidelines*, 65 - 118, Engineering the Human-Computer Interface, Andy Downton, ed., McGraw-Hill Book Company (UK) Limited, 1991.
[Downton et. al 91]	Andy Downton and Graham Leedham, *Human aspects of human-computer interaction,* 13-27, Engineering the Human-Computer Interface, edited by Andy Downton, McGraw-Hill Book Company (UK) Limited, 1991.
[Gabrielsson73a]	Alf Gabrielsson, *Adjective Ratings and Dimension Analyses of Auditory Rhythm Patterns*, Scandinavian Journal of Psychology, 1973, 14, pp 244-260.
[Gabrielsson73a]	Alf Gabrielsson, *Similarity Ratings and Dimension Analyses of Auditory Rhythm Patterns I*, Scandinavian Journal of Psychology, 1973, 14, pp 138-160.
[Gabrielsson74]	Alf Gabrielsson, Performance of Rhythm Patterns, Scandinavian Journal of Psychology, 1974, 15, pp 63-72.
[Gargarian93]	Gregory Gargarian, The Art of Design: Expressive Intelligence in Music, Ph.D. thesis, Media Laboratory, Massachusetts Institute of Technology, 1993.

[Greif91] Siegfried Greif, *The Role of German Work Psychology in the Design of Artifacts, Designing* Interaction, Cambridge Series on Human-Computer Interaction, 1991

[Hawley93] Michael Hawley, *Structure out of Sound*, Ph. D. thesis defense for the Media Laboratory, Massachusetts Institute of Technology, 1993.

[Hunt,Kirk&Orton90] Andy Hunt, Ross Kirk, and Richard Orton. *MIDIGRID, An Innovative Computer-Based Performance and Composition System.* International Computer Music Conference Proceedings, pp 392-394 , 1990.

[Johnson&Wichern88] Richard A. Johnson and Dean W. Wichern, *Applied Multivariate Statistical Analysis,* second edition. Englewood Cliffs, New Jersey, Prentice Hall, 1988.

[Johnson-Laird91a] Johnson-Laird, Philip N., Rhythm and Meter: A theory at the computational level, Psychomusicology, 1991, 10, 88-106

[Johnson-Laird91b] Johnson-Laird, Philip N., *Jazz improvisation: A theory at the computational level.* Representing musical structure, P. Howell, R. West, & I. Cross Eds., pp 291-325. London : Academic, 1991.

[Kim90] Scott Kim, *Interdisciplinary Collaboration,* The Art of Human-Computer Interface Design, pp 31-44, edited by Brenda Laurel, Addison-Wesley Publishing Company, Reading, Massachusetts, 1990.

[Koza92] John R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, the MIT Press, Cambridge, Massachusetts, 1992.

[Landauer91] Thomas K. Landauer, *Let's Get Real: A Position Paper on the Role of Cognitive Psychology in the Design of Humanly Useful and Usable Systems*, Designing Interaction, John M. Carroll ed., Cambridge Series on Human-Computer Interaction, 1991.

[Laurel91] Brenda Laurel, *Computers as Theater*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1991.

[Leedham91] Graham Leedham, *Input/output Hardware*, 187 - 219, Engineering the Human-Computer Interface, Andy Downton, ed., McGraw-Hill Book Company (UK) Limited, 1991.

[Librande92] Steve Librande, *Example-Based Character Drawing,* M. S. Thesis for MIT Media Laboratory, 1992.

[Machover92] Tod Machover, *Hyperinstruments: A Progress Report*, 1992.

[Machover92b] Tod Machover, *Bounce*, musical score, Ricordi, Paris/Milan, 1992.

[Maddox92] W. Todd Maddox, *Perceptual and Decisional Separability*, pp 147-180, Multidimensional Models of Perception and Cognition, F. Gregory Ashby, ed., Scientific Psychology Series, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1992.

[Mathews80] Max Mathews, *The Sequential Drum*, Rapports IRCAM, Paris, 1980.

[McFall93] Michael McFall, *On the Beat (The Electronic Musician Drum Machine Shootout)*, Electronic Musician July 1993.

[Mountford90] S. Joy Mountford, *Tools and Techniques for Creative Design,* The Art of Human-Computer Interface Design, edited by Brenda Laurel, Addison-Wesley Publishing Company, Reading, Massachusetts, 1990.

[Norman88] Donald Norman, *The Design of Everyday Things*, 1988.

[Norman91] Donald A. Norman, *Cognitive Artifacts*, Designing Interaction, John Carroll, ed., Cambridge Series on Human-Computer Interaction, 1991.

[Norman93] Donald A. Norman, *Things that Make Us Smart*, 1993.

[Povel84] D-J. Povel, *A theoretical framework for rhythm perception.* Psychological Research, 45,315-337, 1984.

[Pressing92] Jeff Pressing, *Synthesizer Performance and Real-Time Techniques.* Madison, Wisconsin. A-R Editions, Inc., 1992.

[Rosenthal92] David Rosenthal, *Machine Rhythm: Computer Emulation of Human Rhythm Perception,* Ph. D. thesis for MIT Media Laboratory, 1992.

[Rowe91] Robert Rowe, *Machine Listening and Composing Making Sense of Music with Cooperating Real-Time Agents*, Ph. D. thesis for MIT Media Laboratory 1991.

[Rowe92] Robert Rowe, *Interactive Music Systems.* Cambridge, Massachusetts. The MIT Press, 1993.

[Schloss90] W. Andrew Schloss, *Recent Advances In the Coupling of the Language Max with the Mathews/Boie Radio Drum* International Computer Music Conference Proceedings, pp 398-400, 1990.

[Swigart90] Rob Swigart, *A Writer's Desktop*, The Art of Human-Computer Interface Design, pp 135-142, edited by Brenda Laurel, Addison-Wesley Publishing Company, Reading, Massachusetts, 1990.

[Therrien89] Charles W. Therrien, *Decision, Estimation and Classification.* New York, John Wiley & Sons, 1989.

[Tversky77] A. Tversky, *Features of similarity*, Psychological Review, 84, 327 - 352, 1977.

[Wagner90] Annette Wagner, *Prototyping: A Day in the Life of an Interface Designer*, The Art of Human-Computer Interface Design, 79-84, edited by Brenda Laurel, Addison-Wesley Publishing Company, Reading, Massachusetts, 1990.

[Wessel91] David Wessel, *Improvisation with Highly Interactive Real-Time Performance Systems.* pp 344-347, International Computer Music Conference Proceedings, 1991.

[Wessel79] David Wessel, *Timbral Space as a Musical Control Structure*, Computer Music Journal 3 (2), pp 45-54, 1979.

[RolandCorp92] Operator's Manual for R-70, Roland Corp., 1992.

Software:

[Offenhartz] John Offenhartz, *UpBeat, The Smart Rhythm Programmer,* Intelligent Music.

[Zicarelli a] David Zicarelli, *M, The Interactive Composing and Performing System*, Intelligent Music.

[Zicarelli b] David Zicarelli, *JamFactory, The Improvisation and Live Performance Processor,* Intelligent Music.