and Steinhardt 1986):

$$\omega_{pq} = \frac{2\pi}{1+\frac{1}{\phi^2}}[p+\frac{q}{\phi}]$$

and with a different intensity for each component. If we take $X = 2\pi q - \omega_{pq}/\phi$ then the intensity is proportional to

$$\frac{4sin^2\frac{X}{2}}{X^2}$$

A pitch defined by p and q is more intense if $\phi q - p$ is small or p/q close to ϕ , that is when (p,q) are successive Fibonacci integers (F_n, F_{n-1}). Outside this sequence the intensities decrease strongly. If we distribute the pitches more intense according with the intensity the following harmonic fields are obtained ($p,q = 1,2,...20$, and the frecuencies are scaled by a factor ten):

$$pp : \{D\flat_5\}; p : \{A_3, C\sharp_4, G_5, C\sharp_6\}; mp : \{F_4, B\flat_4, F_5\}; mf : \{G_2, D_3, B\flat_5\};$$
$$f : \{F_3, G_4, A\flat_4, E\flat_5; E_5, A_5, E\flat_6\}; ff : \{D_2, B_3, E\flat_4, B_4, D_5, F\sharp_5, A\flat_5, C_6, D_6\}$$

4.Conclusion.

The Fibonacci sequence is an example of the great variety of temporal structures we can get by means of aperiodic systems in 1D. We can think in the pairs intensity-pitch as the Fourier spectrum of aperiodic rithms generated automatically. These sequences have another interesting propierty: they are selfsimilar. There exists a transformation in which each interval is subdivided into pieces that can rejoin to form a new sequence with all intervals scaled down by a factor ϕ. This hierarchy can be used in order to articulate the global musical form.

Models based on aperiodic systems have been used by the author in works like *Moradas* for organ, *Nocturno* for sopran and ensemble, *Imágenes* for two pianos and others.

References.

Levine, D. and Steinhardt P. (1986) Quasicrystals.I.Definition and structure. *Physical Review B.* **34**,596.

Stockhausen K.(1963) ...wie die Zeit vergeht... *Texte zur elektronischen und instrumentalen Musik. Bd.1.* Dumont Buchverlag, Köln.

THE NECESSITY OF COMPOSING WITH LIVE-ELECTRONICS
A short account of the piece "Gegensaetze (gegenseitig)" and of the hardware (AUDIACSYSTEM) used to produce the real-time processes on it.

JAVIER ALEJANDRO GARAVAGLIA
Hendrik-Wittestr. 11
45128- Essen.
Deutschland. (Germany)
Tel: (0049-201-229831)
E-Mail Address: gara@folkwang.uni-essen.de
ICEM- FOLKWANG-HOCHSCHULE ESSEN
Klemesborn 39 / 45239- ESSEN. Germany

ABSTRACT

The aim of this paper is to speak about my piece -Gegensaetze (gegenseitig) [Contraries (reciprocally)] for alto flute, 4 Channel-tape and live electronics (1994)- making an account of how and why the work was concieved. The hardware-and-the-software environments which are responsable for the real-time processes (AUDIACSYSTEM, a project carried on by the ICEM (Institut for Computer music and electronic Media) at the Folkwang Hochschule-Essen and by the company Micro-Control GmbH & Co KG, both in Germany) will be described. Finally some examples and passages of the piece will be explained.

"CONTRARIES "

"Gegensaetze (gegenseitig)" was the result of an idea that I have had for a long time: to compose a piece in which contraries should be shown not only against each other (in a negative way), but also that they could be able to build some kind of unity by creating something completely new, constructive and positive.

My first problem was how to put this into music without using a text about the subject. At the beginning I simply wanted to make a contrast between a normal instrument and a prerecorded tape, but it didn't seem like being the solution to the problem because it could actually show only the contraries themselves but not the reciprocal action of both elements. The instrument should make with the electronic something really new and this should happen in real time and not with recorded material. That was the reason why I first began to work on the tape itself, making sounds with two Yamaha synthesizers (TX 802 - TG77) that shouldn't have any relation with normal instruments. I composed then a previous piece for stereo-tape alone, from which I took the materials for the definitive version of the work. When the tape materials were selected, I knew already that the instrument should have to be a very soft one, an the election was that of an alto flute. How should then the "reciprocal action " look like? I was now pretty sure that it should be performed with live-electronics. This decision conduced me to the next problem: what type of live-electronics did I really want and much further, which kind of system should I use?. There are basically two ways of working with live-electronics: on one side, those whose aim is to create a new conception of how the live instruments could be projected into a particular space or room, normally using only echoes and delay lines; on the other side, the more complicated ones, in which the sound will be actually processed in real-time (through FM, AM, filters, envelope generators, envelope-followers, tranpositions, etc) up to the point in which the instrument itself could be no longer recognizable. At the ICEM of the Folkwang Hochschule in Essen (Germany), there's no IRCAM board, but there's a completly different project, which has been carried on since eight years at the ICEM by a group of german composers and ingeneers. This project is the AUDIACSYSTEM, about which I shall speak later in this lecture.

Once I had already got the three Instrumental groups (alto flute, 4-channel tape and the 4-channel live-electronics), I wanted to prosecute composing each parameter (from the micro-up to the macro-structures) with the same concepts of THESIS-ANTITHESIS working together to create something new, so that at any point of the piece the main idea could be shown. For this purpose, I've chosen very empirically two principles which are opposite to each other: a "single-principle" and a "totality-principle". Both principles should have to be the

main generators of every event throughout the work and are mainly represented everywhere in the piece by two objects: "glissando-object" representing the "totality-principle" and "a single-note-object", representing the "single-principle".

For the whole structure of the work, I have chosen a numerical-row, whose first four components were explicit selected by myself, but from the 5th component on, they should always be the addition of the last three numbers (that means that the next figure in the row, will be constituted with the reciprocal action of the former three). It comes as result a bigger new value that stands as a contrary to the first, for example, the row begins with (1 1 3 5), which are the numbers that I arbitrary selected; the next value will be 9 (1+3+5), the next 17 (9+5+3) and so on. Each single element contributes to make a partial new totality. This row plays an extremely important role in the composition of the pitches, rhythms, metronomic values, form, and the stage-production, as well.

The form of the piece consists of 5 parts, each one showing the principles already mentioned:

1- Solo alto flute ("single-principle")
2- Alto flute + Tape (as opposites)
3- Only Tape ("single-principle")
4- Alto flute + Tape + live-electronics ("totality-principle"- reciprocally action of all three Instrumentals)
5- Only live-electronics ("single-principle" as result of the reciprocally action of all three Instrumentals)

The rhythms have been composed with the numerical row too. There is a unit value which is the sixteenth, which will be multiplied or divided with the numbers 1, 3, 5, 9, in all possible combinations within these 4 numbers (for example, ratio 9:5 means that 9 equal durations should be instead of 5 sixteenths; ratio1:3 results in a dotted eight,etc).

The stage-production is also supposed to work with contraries. The stage should only be iluminated when the flautist has to play (parts 1,2,4 and 5). In part 3, where only the 4-channel-tape is present, the whole stage and the whole hall (if possible) should be dark.

The material for the pitches has been derived from a chromatic scale beginning with the pitch g3 (the deepest note for the alto flute in G), representing a whole or totality object, a metasymbol of the "glissando-object ". This object plays one of the most important roles throughout all parameters in the piece, not only for the flute-part, but also and mainly for the tape and the live-electronics. The process of genereting the whole pitches for the flute part are produced by an algorithm that eliminates some notes in such a way that at the end, there's only one pitch left. The result is a process going from the whole (all 12 tones) up to ONE SINGLE element, generating a tension between the two main principles mentioned above. The pitches which were eliminated, will be used later in part 4, in the form of 3 improvisations, in which only the rhythms are totally free. These improvisations make a counterpoint to the live-electronics and even modulate them, as it actually happens in the third one.

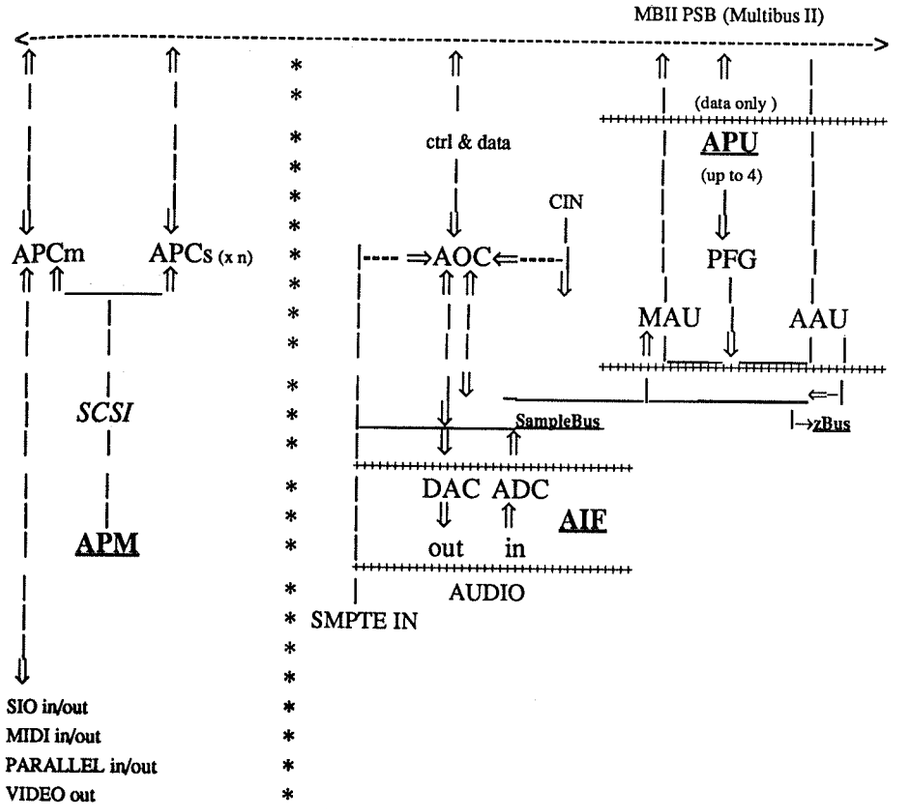
The whole 4-channel-tape part has been worked up with many different methods, for example CommonMusic, transpositions and filters (mostly with Sound Designer II), echoes and even with the AUDIACsystem itself. This twenty minutes long 4-channel tape makes at its begining a counterpoint to the alto flute, than develops itself alone and at last must fade out very slowly as the live-electronics start. I think that at this point, the time has come to make a short description of the AUDIACsystem and its application for the live-electronics in the piece.

THE AUDIACSYSTEM

The AUDIACsystem is a project developed at the Folkwang-Hochschule in Essen (Germany) by the ICEM (Institut for Computer music and Electronic Media) and the Micro-Control GmbH & Co KG. The people involved in its whole design are: Dr. Helmut Zander, Dipl. Ing. Gerhard Kümmel, Prof. Dirk Reith and the composers Markus Lepper and Thomas Neuhaus. The whole began in 1987 and attaches not only the hardware architecture, whose specially designed Audio Processor Unit has got still today the power of 2,5 Pentiums - naturally regarding only the audio processing capacities- but also the software itself, which was exclusively created for this particular environment. The hardware configuration employed in my piece should be contemplated today as an already finished stage of its own development, because almost the whole is going to be actualized, replacing the current design with a new one, which shall result in a chain of Pentiums or most probable P6s, adquiring a RISC- processor configuration and making the whole a bit smaller than today's one cubic meter , possibly making it also compatible with a Power-PC.

HARDWARE CONFIGURATION

The hardware configuration of the AUDIACSYSTEM is shown on the following schematic representation:



The hardware architecture of the AUDIAC has been concieved with the principle of the specialized subsystems. It has not only been made to generate organized forms for the musical production, but also incorporates the generation and working up of sounds in real-time. The whole implies a hughe measure of different demands in relation of its computing potential, which can only be solved with the above mentioned subsystems and their communication capacities.

The whole system could be described as the cooperation of a **"von-Neumann"** unit on the one side and a **Signal-processing unit** on the other. The former percieves configurations (devices), control and driving functions, which steer the processes of generating and working up of sounds from the latter. The communication is guaranteed with the help of the Multibus II. The "von-Neumann" part consists of a Manager (APM) and one or more control units, the APCs. Both do communicate via SCSI.

The APM (Audio Processor Manager) is a 486 Computer with 66 Mhz clock-rate, where the software specially designed for the AUDIAC is implemented. This software is the language APOS which means Audio Processing Operating System and which was specially created by the german composer Markus Lepper for this purpose. APOS pursues three goals which are:

1- a monolithic system architecture, in terms that every hard-and-software levels could be described with the same language, from an individual bit of the hardware up to very complicated abstract compositional models;

2- an enlargable antrophomorphic surface, in the sense that each composer can use not only algorithms that are already defined but also can implement his own language for a particular use as well;

3- an abstraction from the technical necessities, meaning that composing should be allowed on a symbolic level, without caring about technical details.

APOS is an object-orientated language that works with two levels of interpreter : an outer interpreter, which recieves the information in ASCII code, and an inner interpreter, which reads a row of object-references, which are references about objects that already exist and could be recognized as such. The software runs in protected-mode because of memory management reasons, and makes possible that some kind of tasks -which are necessary for the actual configuration of the system- can be percieved.

Regarding the APCs (Audio Processor Controller), the system can afford from only one up to four units. These are all 186 computers which, due to the ATOS kernel (a real-time operating system kernel specially developed for musical applications) has got many functions at their disposal, which are needed for the multitasking operations. The ATOS configurations are created on the APM in APOS and will be later called by the APC, generating or working up sounds. The APC and the Signal processor run asynchronously. The heart of the APC is the APU (Audio Processor Unit), the real Audio processor. Beside it, there are a number of auxiliary units, such as the AOC (a unit capable of transfering data and time code between the APUs, also from one to other two simultaneously, and which could be programmed separately; the CIN (a low control interface with a 16 times multiplex A-D converter, through which up to 16 control voltage units could be brought in); the AIF (the A-D and D-A converters). The APU consists of one Memory Unit (MAU=Memory Address Unit) and an arithmetic unit (AAU, a multiplier). It is possible to put up to 4 APU plus one AOC together, connected through a z-bus. The data could be read and written on the Multibus II. The two memories of the APU (XMY and YMY) can be addressed alone or parallel. The in-and-out sample ports work with the Fifo principle and connect the APU with the out world through the A-D and D-A converters. The interface has 2 inputs and 4 outputs, which could be enlarged up to 32 and 64 respectively. The computing processes run parallel, that means that it could make up to two additions (or substractions), one multiplication, twice read and write from and to the D-RAM (or four times from the S-RAM) at once. The flexible handling of the signal processing unit is guaranteed due to its totally free way of being programmed. The synthesis or working up of sounds result from micro-programms specially developed for this APU.

The Parameter-Functions-Generator (PFG), which is a computing unit in itself works within the APU. It is coupled on one side to the APU and can (due to its complexity) be seen as an independent unit. Its multiple possibilities of application could be resumed in the providing of control instruments for the manipulation of sound: envelopes, spectral control, sound intensity, etc. For each parameter to be controlled, there could be placed pro time-unit one "value-pair" plus a bit-control. Each sample of every four could take a new PFG value. There are altogether 128 PFG free for each APU. The PFG has basically two operating modes: one, in which a "value-pair" INC/FIN makes a linear interpolation, building an envelope which makes a continuous alteration of the y values through the time axis; the other, which interprets a "value-pair" y-*dt* , where y takes one value and *dt* represents the duration of it, building discrete values. The control-bits allow a flexible and interactive influence to the corresponding value rows, for example: back to the first value, mode switch, segment switch,interrupt and hold function (fermata). Interrupts are possible in the first operating mode over each FIN value; in the second mode, at the moment of any new y-value. Through the use or the interrupt features, new support values can be called, resulting in more support values for only one parameter function.

MICROCODES

The biggest time unit to take account of is that of the Sample-rate. The time between two samples will be called "MINICYCLE". There are multiple "cycle calculations" within such a "MINICYCLE" which are coordinated to different process channels (PROK). One cycle calculation can be divided into a given number of microcycles, which correspond to that of the machine rate, which normally is normally set at 10 MHz. All calculations necessary for the generation of a sample must occur within a single "MINICYCLE". The cycle will be finished with a reset signal, which guides to the next step, that is the D-A conversion. With a sample-rate of 48 kHz., the duration of a "MINICYCLE" comes up to around 20 micro-seconds.

WORKING WITH THE AUDIAC

The way in which the input data can be programmed, may be defined in two different forms: on one side it could be done algorithmically; on the other side however, a specially precomposed material could be later imported to the system. Both possibilities don't exclude each other, but could be mixed throughout a

composition, which is actually the case in my piece. The resulting Score can be defined anew in two different ways: statically, creating discrete values for the structure, or dynamically, in which the begin and end of each event is particularly significant, because any kind of process can be programmed between both extremes (for example, transpositions, dynamical filters, etc). This data will be then translated, resulting in a row of orders to be interpreted and fulfilled.

Coming back to my piece, the around 13 minutes long live-electronics part is divided into three different groups: "LA", "LB", "LC", "L" meaning in this case *"live"*. For the programming in APOS, I had got the unvaluable help of Markus Lepper, who I 've already mentioned as the creator of this language.

Regarding the first part,"LA"-with a tempo of quarter equal 50 and measure 3/4- the AUDIAC has to record three different types of single events played by the alto flute, namely: breath-out-noise, one multiphonic and a row of slap tones played separately. They will be given back with intervals of 9, 5 and 3 quarters (proportions taken from the numerical row, which I spoke about in the first part of this lecture), rotating from one channel to the other against the direction of the clock needles, in opposition to the Tape's channel distribution, which is the ordinary one (1. front-left, 2. front-right, 3. right-back, 4. left-back)

The recording of the different sounds is made through an object defined in APOS as "Recorder", which works in such a way as a normal recorder. It has a begin- and an end-buffer-time, an amplitude value, etc. The samples recorded will be played by another object, the "Player" which also has a begin- and an end-buffer-time, an amplitude, an input to vary its frequency ("FINC" transposing the sample) and an input to loop the sample from a given buffer-time-point. There are four players, each one corresponding to each one of the four channels. Each recorded sound becomes a different memory address, so that it could be called at any time. For the time allocation of these events, the computer was asked to find the best possible distribution through all four channels between space and time, in order to force the events to meet quite often at one same channel. When this actually happens, one event will multiply the other, modulating each other (Amplitude Modulation). When all events (once breath-out-noise, once a multiphonic, and five times different slap-tones) have been played and recorded, the computer begins to transpose the information of 3 of the 4 players with different ratios (which are taken from the numerical-row). This transposition, made through the input "FINC" of each "Player", takes place dynamically, that means that within its time limits given in the score, the frequency will be varied every fourth sample, making "glissando-structures".

For "LB", there are two moments to be recorded, both 12 seconds long. This part makes a formal "crossfade" with "LA", and is all about transpositons on all 4 channels of both recorded materials. These transpositions, however, are not dynamical, but discrete. The first of the two recorded materials of "LB", must be further stored, because it will be used in the next part "LC".

The score for "LB" is programmed half algorithmically and half precomposed, as the following APOS example shows:

```
new plstarts "pls2"  200
*  open pls-kanal 0
*;                ANZAHL       ABSTAND
*;                EINSAETZE
*  put pls2       1    *   17
*  put pls2       3    *    9
*  put pls2       5    *    5
*  put pls2       9    *    3
*  put pls2      17    *    1
*
*  apl pls2 110 to 150 [ if [[_1 mod 21] ?eq (110 mod 21)] ['@ .p0 _0 ok] ]
*  apl pls2 111 to 150 [ if [[_1 mod 19] ?eq (111 mod 19)] ['@ .p1 _0 ok] ]
*  apl pls2 112 to 150 [ if [[_1 mod 15] ?eq (112 mod 15)] ['@ .p2 _0 ok] ]
*  apl pls2 113 to 150 [ if [[_1 mod  9] ?eq (113 mod  9)] ['@ .p3 _0 ok] ]
```

All these lines describe every starting point of every of the four players. The last four lines use an explicit indication (precomposed) of how the structure should finish; on the other side, the "put" lines use an automatic way of creating the starting points with an special syntax implemented for this purpose. This syntax will be implemented with the following APOS source text:

```
new latch "pls-kanal"
*   new latch "pls-Position"
*
*; 4 new methods are going to be defined for this purpose (dm)
*
```

```
*   dm [ put (any plstarts) @ (any integer) .p (any integer)
*>                            (any integer) * (any integer) ]
*>      [ apl 0 to [pred _6]
*>                 ['@ pls-Kanal     ['['_5 + __0] MOD 4] ;
*>                   @ pls-Position [' _3 + ['_8 * ['SUCC __0]]] ;
*>                   ~do _0_1 ]]
*;
*   dm [ ~do put (any plstarts) ]
*>      [ @ .p [pls-kanal] [@ _2 [pls-position]] OK ]
*;
*
*   dm [ open pls-kanal (any integer) ]
*>      [ @ _1_2 ; @ pls-position '(INTEGER 0) ]
*;
*   dm [ put (any plstarts) (any integer) * (any integer) ]
*>      [ _0_1 @ [pls-position] .p [[SUCC[pls-kanal]]MOD 4] _r2 ]
```

The evaluation of both texts results in a abstract-time-structure which could be edited either manually or automatically. In this latter case, it could be submitted to different processes of automatic transformation and interpretation, being the actual generation of sound only one of the multiple possible steps of such a chain.

The last part, "LC", begins with three eight-measure long statements of the alto-flute, which will be recorded and played by each speaker with an interval of 3, 5 and 1 quarters (the corresponding tempo is now quarter=90, the measure remains 3/4), making a canon that wanders all over the 4 speakers. Between each of these eight-measure statements, the alto flute plays three improvisations, whose durations are respectively of 9, 17 and 31 seconds. The third one modulates the frequency of the recorded signal of all three parts of the canon, and the result of this modulation will be anew recorded and again modulated from the alto flute. From this point on, the flute doesn't play any more, and the resulting modulation will be amplitude modulated -with the first element of "LB"- than it will be transposed dynamically. The transpositions will be gradually filtered with a *notch filter* , whose frequency is around the pitch f#5 and whose bandwidth will be dynamically narrowed up to this pitch. At the end there's only a filtered f#5 left, making an opposition to the first note of the piece, which was g3 (last and first note respectively of the chromatic row used as pitch-material).

"Gegensaetze (gegenseitig)" was the first piece of music using the AUDIACsystem in a real-time live performance. Up to its premiere, the system had only been used to steer another type of events (all within the electronic music production), but there were no pieces with live instruments composed specially for and with the system.

To bring this lecture to its end, I would like to clear up just one more point. The general conception of the work can be interpreted from several points of views, but my intention was to show the "Thesis-Antithesis" concept at the light of human, social and naturally also political relationships. I think that nowadays, a time in which Neonazi ideas and deeds wide out again (mostly in Europe and in the USA, but not only), the concept of a reciprocal action of opposite elements may be considered as the contrary to intolerance, racism and discrimination. That doesn't mean neither that my piece has got a secret program nor that it is a political work (which is mostly the case of Luigi Nono's music, to take only one example), but it may be able to recall these type of implications. The piece was first performed on June the 18th of 1994 in the city of Dortmund (Germany) by the german flautist Christianne Schulz.

Thank you very much.

June 1995
Javier Alejandro Garavaglia

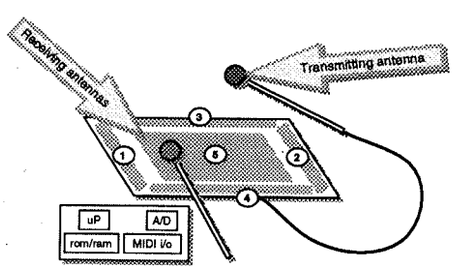
PadMaster: an improvisation environment for real time performance

Fernando Lopez-Lezcano
CCRMA (Center for Computer Research in Music and Acoustics), Stanford University
(nando@ccrma.stanford.edu)

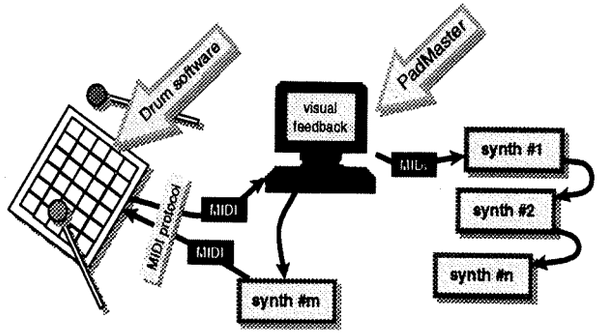
ABSTRACT: This paper will describe the design and implementation of PadMaster, a real-time improvisation environment running under the NextStep operating system. The system currently uses the Mathews/Boie Radio Drum as a three dimensional controller for interaction with the performer. PadMaster splits the surface of the drum into virtual programmable pads which can be grouped into scenes so that the behavior of the surface can be subtly or drastically altered during the performance.

1.0 The Radio Drum and the MIDI communication protocol

The current implementation of the Stanford Radio Drum was developed at CCRMA by Max Mathews as a simpler alternative to Boie's previous design. The two batons act as radio transmitting antennas. The signals are received by five antennas located underneath the surface of the drum. A multiplexed A/D converter translates the received signal strength from each antenna into numbers which are used by the on-board microprocessor to calculate the absolute position of each baton in space. The microprocessor uses this information to track the movement of the batons and to detect hits on the surface. Information about each hit includes both the x-y coordinates and the hit velocity. In addition to the two batons, the Radio Drum hardware includes two switches and four potentiometers. It has a MIDI interface that it can use to communicate with computers or synthesizers.



The behavior of the Drum is defined by the program stored in its EPROM. The drum software includes several functionally different programs that can be externally activated through MIDI System Exclusive messages. Through them, the Radio Drum can act as a stand alone conductor of a score or MIDI file, can improvise with several different options that map baton movement to MIDI commands or can act as a general purpose MIDI controller. This last program and the underlying protocol built on top of MIDI were originally designed by David Jaffe and Andy Schloss. This existing general purpose controller program was completely redesigned by the author. A more efficient and faster protocol was created that uses just one MIDI channel and is more bandwidth efficient in the use of MIDI resources. The protocol was also expanded to allow the controlling computer to upload / download calibration data from / to the Drum. Once the program is activated through a sysex message, the Radio Drum behaves as a three dimensional controller with six degrees of freedom.



Following is a short description of most of the control protocol:

- **System exclusive configuration messages**: can be used to turn

ON or OFF the communication program, set the MIDI channel used by the rest of the protocol, dump and load the internal calibration tables, set the trigger and release heights for both batons, request raw A/D measurements (useful for testing), etc.

- **Trigger / Release messages:** sent by the drum when a baton hits / leaves the surface. Each hit or release is represented by three MIDI controller messages, using continuous controllers 26 through 31. The messages are used to send the x and y positions and velocity of the hit or release.
- **Switches:** a switch message is sent by the drum when one of the two hardware switches changes state. The information is sent through controllers 5E to 5F.
- **Poll request:** sent by the computer to request the position in space of the batons at a given moment. The message uses a channel pressure MIDI message that encodes the required request as a pressure value. The controlling program can thus request the position of one or both batons and can also ask for the current value of the four potentiometers.
- **Poll answer:** sent by the drum in response to a poll request message. The requested information is sent through a string of channel pressure messages. As opposed to the Trigger / Release messages, the Poll Answer message contain no state information, which means that a state machine in the receiver program has to track the incoming messages. While this opens the possibility of garbled information due to lost MIDI bytes it was deemed more important to reduce the bandwidth used by the protocol as this is a frequently used message and the information gathered through it is refreshed periodically.

Planned enhancement to the protocol and underlying Radio Drum library routines include:

- **Detection of hits based on direction reversal**. The current implementation uses a height threshold based detection scheme which cannot reliably detect very fast rolls close to the surface of the drum.
- **Automatic position update**. To further decrease the MIDI bandwidth, the current polling scheme should be replaced with a timer based automatic transmission of the current position (that is, the Drum software should take care of sending periodic position messages). The new scheme will also include a sysex message to change the period of the transmission so as to enable the controlling software to throttle down the sampling rate of the position information when the MIDI stream becomes close to being saturated. This feature is currently implemented by changing the sampling rate of the position request polls.
- **Better internal linearization routines** for the three axes of control.

2.0 The PadMaster program

The PadMaster control code is written in Objective C, using the MusicKit as the foundation class hierarchy for MIDI event scheduling and control. The graphical interface was designed with NeXT's Interface Builder and the program runs on any workstation that supports the NextStep operating system (and has a MIDI driver available). PadMaster is connected through MIDI to the Radio Drum controller and to external synthesizers. The program uses the coordinates of the incoming Trigger and Release messages and an internal calibration map to split the surface of the drum into up to 30 virtual pads. Each pad is independently programmable to react in a specific way to the hit, and to the position information stream of one or more simultaneous axes of control. Pads can be grouped into Scenes, so that the behavior of the surface of the drum can be subtly or radically altered during the course of a performance. This is achieved by dynamically jumping to a different Scene, either through the use of a control pad programmed for that function or through another external controller. The screen of the computer continuously displays a representation of the virtual surface and gives visual feedback to the performer on the state of all the pads in the currently selected Scene.

The virtual pads can be split in two types depending on their function: **Performance** and **Control** pads.

2.1 Performance Pads

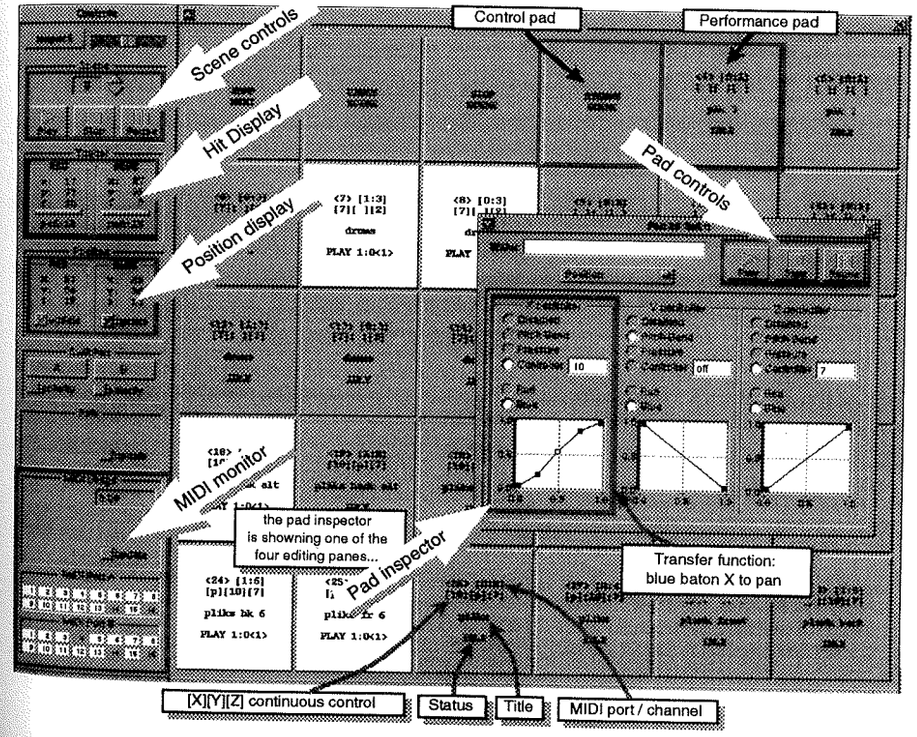
Performance Pads can be individually programmed to control the playback of MIDI sequences, note generating algorithms or soundfiles. The graphical representation of the pads on the screen gives instant visual feedback to the performer. Pads change color and status messages dynamically according to their state. A performance pad that is playing remains active even if the performer selects a different Scene, so that chains of events can be started from one Scene and will continue to run even though the performer latter switches to a different Scene. The status is updated for all active pads but only those in the currently selected Scene show up on the graphical representation of the drum surface.

2.2 Control Pads

Control Pads are used to trigger actions that globally affect the performance of a Scene. A pad can be programmed to change the current Scene when hit, jumping to the next or previous Scene, thus redefining the behavior of the whole surface of the drum. Control pads can also be used to pause, resume or stop all playing pads in the currently selected Scene.

3. Inside a pad

Editable parameters inside each pad can be changed through a standard NextStep inspector window with several editing panes. The first pane can be used to select the type of pad and, in the case of performance pads, the triggering baton and the action that is executed when the pad is hit. The possible actions include starting / pausing / resuming a sequence, starting a new overlapping sequence, or playing the next note of a list of notes. It also selects the MIDI port, channel and program number that will be used for MIDI transmission and allows editing of a graphical mapping of hit velocity to note velocity for the selected sequence. The second pane edits the tempo options for the pad. Tempo can be global, per pad or per sequence inside a pad (as there can be more than one instance of a sequence playing at the same time). There is a tempo envelope and it is also possible to control tempo with the hit velocity or with any of the six available axes of continuous control. The third pane lets you associate up to three continuous MIDI message streams (pitch bend, channel pressure or any con-



troller) with the position of up to three of the six axes of control. All these function mappings are created through graphical function editors. The fourth pane edits the sequence of notes that are played when the pad is hit. The sequence is expressed as a normal MusicKit text scorefile. The scorefile format has been enhanced with additional tags that represent all programmable parameters in a pad. It is then possible to externally generate a textual representation of a pad and then load it into PadMaster (for example, a set of pads for a performance might be algorithmically generated and then loaded into the program).

4. PadMaster in performance

PadMaster has been used to compose and perform "Espresso Machine", a piece for PadMaster and Radio Drum, two TG77's and processed electronic cello (Chris Chafe playing his celletto). The piece is an environment for improvisation in which the PadMaster and celletto performers exchange ideas and play with predetermined materials. The piece is composed in three PadMaster Scenes, each with several groups of related materials that are triggered during the performance. One baton is reserved for triggering pads and the other for continuous three dimensional control of the currently performing pads.

The performance of this piece on several occasions has raised several issues. The simultaneous mapping for several pads of baton movement to MIDI continuous controllers is one of the most interesting performance capabilities of the program, but also raises the possibility of serious MIDI bandwidth clogging. The current version of PadMaster dynamically adapts to the changing conditions and adjusts the position sampling frequency to try to reduce the bandwidth used when several pads are playing simultaneously. More work needs to be done in measuring MIDI usage in a more precise way to avoid sending too much information, but at the same time to avoid control lag if the sampling frequency fall to a very low value. There is also a measurable gap in playback when scenes are changed, during which MIDI activity is not updated as the MIDI and graphical routines share the same execution thread.

5. Future developments

PadMaster is currently undergoing a complete rewrite to implement new and improved functionality.

Pads will be resizable so that each scene can have different number and size of pads if necessary. We have found that sometimes it would be desirable to concentrate important or critical performance functions in a few big pads. Resizable pads would also allow for linking performance behavior to the place where the pad is hit, something that is not possible in the current version. Another enhancement to pads will be inheritance, so that multiple pads with related behavior (something we have found common and very useful) could be grouped together, with the common functionality being editable as a group.

The action types of performance pads will be enhanced to allow for inclusion of algorithms and soundfile playback. The algorithms will be able to use a well defined API to enable linking of baton movement to algorithm parameters.

The NextStep operating system includes a remarkably easy to use system to communicate with remote objects (objects that live in other computer[s] but are directly linked to the execution of a local program). That opens the possibility of using remote computers connected through Ethernet as servers for MIDI or soundfile playback. There are also several scheduled refinements for performance such as a completely separate thread for all MIDI interaction so that graphics may lag behind the performance but there will be no delays when switching between scenes.

Another important enhancement will be defining a way to use different MIDI controllers in addition or instead of the Radio Drum (percussion controllers, normal keyboards, MIDI pedals, etc).

References:

[1] Max Mathews, *The Stanford Radio Drum, 1990*

[2] David Jaffe, Julius Smith and others, *The MusicKit "http://ccrma-www/CCRMA/Software/MusicKit/MusicKit.html"*

[3] Carlos Cerana (composer) / Adrian Rodriguez (programmer), *MiniMax, a piece for Radio Drum*

INTERACTIVE COMPOSITION USING MARKOV CHAIN AND BOUNDARY FUNCTIONS

JÔNATAS MANZOLLI * & ADOLFO MAIA JR. **
*Interdisciplinary Nucleus for Sound Studies (NICS)**
*Applied Mathematics Department (IMEEC)***
University of Campinas (UNICAMP)
13081-970 Campinas SP- BRAZIL
Jonatas@dsif.fee.unicamp.br

ABSTRACT

The research presented here reviews the use of stochastic processes on composition; Markov chains were studied as a tool for developing musical structures in the 70s. This paper presents a model based on Probability Vectors and Boundary Functions which are used on iterative processes to generate further sequences of vectors. The resultant numerical structures are mapped to different classes of musical parameters. The text presents this new approach - the musical ideas behind the method as well as the mathematical model. Finally, it describes a graphic-based compositional system for a MS-Windows Environment.

1. INTRODUCTION

One of the composer's main tasks is to make choice and it would be possible to find several moments within a compositional process in which his/her choices are based on chance. Adding to this the Boulez's concept (1986) that there is inside of each composer a *kernel of darkness* and the Cage's concept that *pure chance is ultimately irrational, by which I mean there can be no closed system of explanation that includes it, save for the Universe itself* (Cage, 1961), we have joined ideas which give us a good flavour and an insight for presenting an application of stochastic process to musical composition..

There are several historical references of composers using random methods in their compositions. Even musicians such as Mozart, Haydn and C.P.E. Bach worked with compositional techniques based on chance. As described by Loy (1988), this method was called Mozart's Dice Game (Würfelspiel); it was a music game used for composing minuets and other incidental works. If the strict musical aesthetic of the Classical Era provided a framework for such approach, it would not perhaps be so surprising to find Cage in the twentieth century using chance as a compositional tool. An interesting connection between the world of computer experimental music and the world of chances was established in the first Computer Music piece by Cage-Hiller called HPSCHD (1967-69) as described by Austin (1992). The genesis the compositional model used in HPSCHD was the work of Hiller & Isaacson (1959); their pioneering research was the seed for an model called *rules-driven-by-noise*. The development of this concept subsequently grew in many different directions and many different decision-techniques were related to stochastic process and Markov chains, as reported by Hiller & Isaacson (1959), Xenakis (1971) and Jones (1981).