and Steinhardt 1986): \[
\omega_{pq} = \frac{2\pi}{1 + \frac{q}{p}} \cdot \left| p + \frac{q}{\phi} \right|
\]

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

A pitch defined by \( p \) and \( q \) is more intense if \( \phi q - p \) is small or \( p/q \) close to \( \phi \), 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 \((p,q = 1,2,\ldots,20,\) and the frequencies are scaled by a factor ten):

\[
pp: \{D_b\}; \, p: \{A_2, C_5, E_3, G_4\}; \, mp: \{F_4, B_b, F_3\}; \, mf: \{G_5, D_2, B_b\};
\]

\[
f: \{F_5, G_4, A_b, E_b, E_3, G_5, A_3, E_5\}; \, ff: \{D_5, B_5, E_b, B_b, B_2, F_5, A_b, C_6, D_6\}
\]

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 rhythms generated automatically. These sequences have another interesting property: they are self-similar. 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 \( \phi \). 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 soprano and ensemble, Imágenes for two pianos and others.

References.


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 "Gegensatz (gegenesitig)" and of the hardware (AUDIACSYSTEM) used to produce the real-time processes on it.

JAVIER ALEJANDRO GARAVAGLIA
Hendrik-Wusuher, 11
45128- Essen.
Deutschland, (Germany)
Tel: (0049-201-229813)
E-Mail Address: gar@folkwang.uni-essen.de
ICEM-FOLKwang-Hochschule ESSEN
Klemensborn 99 / 45239- ESSEN, Germany

ABSTRACT

The aim of this paper is to speak about my piece -Gegensatz (gegenesitig) [Contraries (reciprocally)] for alto flute, 4 Channel-tape and live electronics (1994) - making an account of how and why the work was conceived. The hardware-and-the-software environments which are responsible for the real-time processes (AUDIACSYSTEM, a project carried on by the ICEM (Institut für 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"

"Gegensatz (gegenesitig)" 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 a piece with two Yamaha synthesizers (TX 502 - 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, as the attack 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 conducted 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, transpositions, 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 completely different project, which has been carried on since eight years at the ICEM by a group of german composers and engineers. 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
main generators of every event throughout the work and are mainly represented everywhere in the piece by two objects: "glimsando-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 explicitly 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 1 arbitrary selected; the next value will be 9 (1+3+5), the next 17 (9+3+5) 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; ratio 3:5 results in a dotted eighth, etc.).

The stage-production is also supposed to work with contraries. The stage should only be illuminated 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 e3 (the deepest note for the alto flute in G), representing a whole or totality-object, a metonymy of the "glimsando-object". This object plays one of the most important roles throughout all parameters in the piece, not only for the flute-part, but also for the tape and the live-electronics. The process of generating 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), echo and even with the AUDIACsystem itself. This twenty minutes long 4-channel tape makes at its beginning 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 Kimmel, Prof. Dirk Reith and the composers Markus Lepper and Thomas Neumann. 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, and which shall result in a chain of Pentiums or most probable P6s, acquiring 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 as the AUDIACsystem is shown on the following schematic representation:

- MIII PSB (Multibus II)
- ctrl & data
- APU (up to 4)
- MAU
- AUI
- SampleBus
- DAC ADC
- AIE
- AUDIO
- SMpte IN
- SIO in/out
- MIDI in/out
- PARALLEL in/out
- VIDEO out

The hardware architecture of the AUDIAC has been conceived 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 huge 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 perceives 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 SCI.

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:
composition, which is actually the case in my piece. The resulting Scores can be defined anew in two different ways: staticaly, 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, dynamic filters, etc). This data will be then translated, resulting in a row of orders to be interpreted and fulfilled.

2010, there are two main problems which, due to the ATOS kernel (a real-time operating system specifically developed for musical applications) has got many functions at their disposal, which are needed for the multitasking operations. The ATOS configuracions are created on the AP on APOS and will be later called by the APOS, generating or working up sound units. The APOS and the Signal processor run asynchronously. The heart of the APOS is the APOS (Audio Processor Unit), the real Audio Processor. Beside it, there are a number of auxiliary units such as the AOC (a unit capable of transferring data and time code between the AP units, also from one to another two simultaneously, and which could be programmed separately); the CMU (a control interface with a 16 times multiplexer A-D converter, through which up to 16 control voltage units could be brought in); the AIP (the A-D and D-A converters). The APOS consists of one Memory Unit (MAU-MAU) and an arithmetic unit (AAU, a multiplier). It is possible to put up to 4 APOS plus one AOC together, connected through a connected bus. The data can be read and written on the Multibus II. The two memories of the APOS (XMY and YMY) can be addressed alone or parallely. The in- and output ports work with the FIFO principle and connect the AP with the output 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 subtractions), 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 utilization for a way of being programmed. The synthesis or working up of sounds result from micro-programs specially developed for this APOS.

The Parameter-Functions-Generator (PFG), which is a computing unit in itself works within the APOS. 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 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 APOS. 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 row, for example: back to the first value, mode switch, segment switch, interrupt and hold function (fermatum). Interrupts are possible through the APOS.

MICROCODES

The biggest time unit to take account of is that of the Sample-rate. The time between two samples will be called "MICINCYCLE". There are multiple "cycle calculations" within such a "MICINCYCLE" which are coordinated to different process channels (PROC). One cycle calculation can be divided into a given number of microcycles, which correspond to that of the machine rate, which normally is fixed at 12 MHz. All calculations necessary for the generation of a sample must occur within a single "MICINCYCLE". 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 "MICINCYCLE" 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 precompiled material could be later imported to the system. Both possibilities don't exclude each other, but could be mixed throughout a
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 NeusStep operating system. The system currently uses the Mathews/Boies 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 system 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 baton at a given moment. The message uses a channel pressure MIDI message that encodes the required request as a pressure value. The message is sent to the MIDI port at a position message that encodes the required request. The position message is sent as a pressure value to enable the computer to retrieve the position. The computer can also ask for the current value of the four positioner.

- **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 contains 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 MIDI library routines include:

- **Detection of hits based on direction reversal**. The current implementation uses a threshold level based detection scheme which can reliably detect very fast hits 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 would also include a system message to change the period of the transmission so as to enable the controlling software to throttle down the message rate.

Better internal linearization routes for the three axes of control.

**2.2 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 NextStep's Interface Builder and the program runs on any system that supports the NextStep operating system. The PadMaster is connected through MIDI to the NextStep controller and to external synthesizers. The program uses the coordinates of the incoming Trigger and Release messages and an internal calibration map to map the surface of the drum into up to 30 virtual pads. Each pad is independently programmable to re-map to the position of the hit and to the position information. The program can be switched 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, notes generating algorithms or sound files. The graphical representation of the pads on the screen gives instant visual feedback to the performer. Pads change color and status 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 later 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.

**5. 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 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 other) with an Editable parameter.
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 cello). The piece is an environment for improvisation in which the PadMaster and celloto performers exchange ideas and play with predetermined materials. The piece is composed in three PadMaster Scenes, each with several groups of related determined 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 instantaneous 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 minimizing MIDI usage in a more precise way to avoid sending too much information, but at the same time to avoid control loss if the sampling frequency falls 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.

Future developments

PadMaster is currently undergoing a complete rewrite to implement new and improved functionality. Pads will be resizeable 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. Resizeable pads would also allow for linking performance behavior to the place where the pad is hit. The NextStep operating system includes a remarkably easy to use environment to communicate with remote objects (objects that live in other computer[s] but are directly linked to the execution of a local program). This 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 open interface 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 to or instead of the Radio Drum (percussion controllers, normal keyboards, MIDI pedals, etc).

References:

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