ABSTRACT

This paper lists and discusses some basic ideas for the design of software using realistic musical input. A three part system will be described: (1) feature extracting processes, (2) compositional processes, (3) performance processes. My main concern with this description, drawn from the user's point of view, is to set a bias for communication between scientists and musicians, and within the computer science community. I will emphasize computer science issues such as portability, efficiency and documentation and musical issues such as the computer-computer/instrumental relationship, musical data structure, and dedicated software for natural or improved music. Each of our examples will be demonstrated live on a prototype workbench built at CARL. They are the results of the combined efforts of a laboratory (CARL), and a music faculty (UCSD Music Department).

RESUME

Ce papier énumère quelques idées de base pour la conception de programmes utilisant des paramètres musicaux spécifiques au temps réel. Un système de trois parties sera décrit: (1) processus d'extraction des paramètres, (2) processus composentiaux, (3) processus d'expression. Avec cette description fait du point de vue de l'utilisateur, mon principal souci est d'offrir une base pour la communication entre scientifiques et musiciens et dans la communauté informatique musicale. J'insisterai sur des notions informatiques comme portabilité, efficacité et documentation et des notions musicales comme relations compositeur-composant-instrumentation, interprétation et données musicales, et implémenterai au projet prototype de la troisième étape. Chaque exemple sera démontré sur un prototype de la troisième étape et sera développé au CARL. Le but est d'évaluer ensemble des effets combinés et de laboratoires (CARL) et de musiciens (Département de Musique & UCSD).

1. INTRODUCTION

The purpose of this paper is to outline the concept instruments/instrumentation with musical examples and to emphasize the implied difficulties for the musician using a real-time environment. This paper makes a proposal for a model which can be used as an interface specific between existing or performing software modules. It gives data structures as a contribution to MDD software.

Interest in using parameters extracted from a live acoustic instrument as musical input has already been demonstrated (Vercoe 1984, Derrington 1984, Bourguet and Chabot 1984 et al, Georgiou Liszewski in concert). Features such as timbre, amplitude, timbres, cymbals, sones of different kinds as well as programmable digital synthesizers and sound processors are beginning to appear in the literature and on the market. The MIDI standard expands in normalizing the protocol between them and with their host computer. The use of these tools in live performance introduces new musical problems for which there are not yet answers or for which there are too many partial ones: (1) the real-time control is a capability, but also a constraint for the composer who has to program since such control has traditionally been the task of the instrumentalist. trained only to use his sophisticated sensory system as well as auditory, physical and visual cues. Let us recall that the topic is not even 15 years old, as long as it takes a good student to learn an instrument; (2) The ability of the computer to manipulate, process and reuse the musical data input by instrumentals is totally new. It materializes what were traditionally abstract musical thoughts. On the other hand, it is expected that the instrument will be given a new impulse.

My experience with the OS at BRUCAT and the work by CARL reinforces the idea that the psychological implications for the composer and the instrumentalist are far from being negligible. Training with so-called low-cost high-performance units has to be part of undergrond programs. It is very urgent to make efforts in developing software together with musical thoughts rather than work in the user-software.

This leads me to emphasize the importance of (1) collaboration between musician and software, and (2) communication and exchange within the computer community. The first point doesn't seem to be always accomplished in the laboratories I have valued and the second lacks an appropriate medium (for example the ISMC is dedicated to information rather than distribution).

2. INSTRUMENT vs. INSTRUMENTALIST

2.1 Introduction

Interesting features of realistic systems are synchronization to external source and feedback. Such a realistic system is that of the instrument/instrumentation; feedback comes from auditions and the mental feedback to the instrument; see Florence and Cadot (1984).

The relation instrument/instrumentalist also worked that way for years having exact synchronism as a fixed parameter. "Swanton" (Brown 1957 and 1964), and "caprice pour piano" (Bouton 1972, 1977 and 1975), are famous examples of a new concept of character music featuring variable coordination rather than exact synchronism. Even clearer in the following "Swanton for Piano" (Bouton 1977) started before 1960.

The relation sound/composer or sound/instrumental was fixed by the HX5 computer tradition till the last fifty: electronic music examples of variable sounds are Bodroz's "L'oiseau lunaire" and Stockhausen's "Klavierstück X".

These citations are to introduce a comment: the computer can be at the same time an instrument, a computer and a composer. I don't expect to represent these entities as well as many could do; it would be a semantic competition. My main concern is to explore the variability and the complexity of the relationship computer-composer-composition. Let us see now what it implies for the musician/performer to deal with a computer.

ICMC '85 Proceedings
A computer program typically needs data to work on. By (some) analogy, a set of data is called a score. The computer programmer must split his idea, traditionally symbolized by a musical score, between program and data. I am not telling about the way he can specify his idea, e.g., about language, but how his idea is reproduced in the machine. This can be more complex if the computer contributes to produce and modify the score—data.

The strategy for using data as a function of time is in the synthesizer or control processor: organization of the degree of independence left to the machine, as its individuality. If the machine acts on its own concept of time, it is considered as an instrument, if it acts only upon external input, it is the instrument.

Feed back to the machine, depending on the theoretical model used and the parameter adjustments, allows the thorn to simulate behaviors between the two: external instrument and time-simulated. But, whatever the model's saliency, the reaction will be far from those of the actual instruments or real instrumentality. Playing this twofold sense is really fascinating but difficult for the performer.

Finally, here are my comments on those two oversown words: Feed back. I expect more from feed back to the instrumentarian than to the computer. I mean that an instrumentarian is likely more able to give an interesting response to a—perhaps unconscious—event coming from a tape, than a computer using even a sophisticated cognitive model.

Synthesizer: I am concerned about the recent forlorn for synthesizer, because it allows the weakness of the computer to hide behind the liveliness of the performer. I would like to be able to think that music has passed the stage of the XIXth century's overfertilized singer framed by a careful accompanist. The Akadue should be a parameter of the piece and NOT a capability of the system. Realtime can be boring if it is too real.

3.2 Data Loading Strategies
Several realizations of data loading strategies will be presented. For each, a simple schematic and comments will be given.

2.1. Simple Instrumentation: Transposition
A frequency shifter transposing the entire spectrum as well as a synthesizer send the input pitch as a content interval transposition. Depending on the content, it is expected to enhance the spectrum, the centre or the harmonic content of the input.

![Figure 1: Pitch driven frequency shifter.](image)

Transposition parameters will be changed on instrumental runs, as instrumental attack and size will have to be synchronized—good luck. All direct sound transformations are similar to this example. The problem for the performer is the same as when changing from one transposing instrument to another.

2.2. Complete Instrumentation: Transposition plus Delay
In all following examples, we will give examples in the case of a synthesizer driven from a live instrument.

Combination of delays and transposition are relatively simple to implement. The result can be quite complex. If the composer thinks of rhythm as additive pace in the measure, delays should be in parallel, if he thinks of rhythm in distance between events, delays will be in series.

The line allows rhythm transposition, the second rhythm pattern.

![Figure 2: Dips and Transposition](image)

On one note from the performing instrument, the system will produce a rhythmic arrangement. If the instrument plays a rhythmically syncopated line, whose pitch is related to the irregularly updated delay and transposition parameters, the result will be a complex material. Various coincidences and recurring effects can add to the interest of the material. I saw this setup as a very disturbing instrument for the instrumentarian to perform with. If delay parameters are adjusted with the performer's tempo, the thorn becomes instrumentarian itself. Results will be very interesting, but probably difficult to foresee.

2.3. Instrument + Instrumentation: Orchestration
The section 3 will explain how orchestration can be implemented in ad hoc. These non-exhaustive possibilities will be given: (1) orchestration taken from a material prepared beforehand, (2) orchestration taken from material input by the performer during a previous section of the piece, (3) orchestration is taken from musical input by other performers during previous section of the piece.

![Figure 3: Orchestration by another instrument](image)

In the first example, the machine acts clearly as instrument, but in the second one, it is more ambiguous. In the third one, possibly in an improvisation context, one instrument plays with the music of the other. The orchestration, depending on the way it is produced and placed, can be viewed as an independent line related to the performer's lines by the triggering criterion. This can be controlled by the dependence of the orchestration on the input's pitch and rhythm.

2.4. Record Player: These examples are a generalization of all previous examples. For instance, in an improvisation environment, must be clearly defined: trigger criteria, type of data processing (especially difficult if one wants to keep the incoherence) and the type of output.

3. THREE PART MODEL

3.1 Anatomy
Let us now introduce a model for performance (meaning with an relation to the composer in real time). The system includes: (1) feature retaining processors, (2) compositional processors and (3) performance processors (see below). The composer's task is to specify beforehand various based on feature patterns, detection of which will trigger and
3. Compositional Processes

Here are the processes involved and fed from the data produced by the feature extracting processes described above. The name used for each process must not replace the role; the compositional input is involved everywhere. They are dedicated to producing the final data to produce data. The following, let us assume that we want to listen to the test produced by as instrument in the same way that its timbral identity can be drowned by a sound processor. The sound processor can be monitored by acoustic parameters like permutation, composition, splitting and thus, or as suggested below, from another instrument, with the input from which we would build a new instrument around center some parts from the final instrument. These examples show that the word orchestration can be rethought. The compositional processes will be triggered by rules given by the feature extracting processes. Their input will indeed be data, either synthesized by a sound program or recorded from an instrument. Thus output will be data likely ready to be played.

4. The Performance Process

These processes include all interface to the sound processor or the instrument. Some controls are discrete such as ordered program loading, wave table or parameter loading, ordering to the hardware, others are continuous (e.g., functions. Data used is produced by the feature extracting processes from the present or previous sessions, or by any general-purpose sound program. The upper limit bandwidth of a particular system must be well-defined, known and reached by the musician in order to obtain different results. A software is in Virtue (1986) or hardware as in Moore (1980) this step takes place will be needed if the settings are event oriented. CIRCUIT/E (Matthews and Moore 1973) implements a function-oriented approach.

4. A DATA STRUCTURE

4.1. Introduction

The need for a data structure is obvious when using MIDI data, for example, and to write code for "compositional processes" for example to apply a permutation matrix to recorded data. The basic data follows below or above (1978). The implement will be the result of extensive talks with Gershon Kay and Andy VOllett at CIRCUIT. This data structure is closely related to the input data handler but independent code. Since we use the MIDI data from the Roland MPU as well as pitch, intensity and duration data from the violin pitch detector presented in VOllett (1983).

4.2. Principles

Here are some preliminary considerations: (1) Same data may be varied in several different contexts. Thus a compositional process will at least have to cope with the data before playing it. This may not be very efficient. (2) Each structure may evolve and data may be recorded and structured would be of very fast. Transitions provided to be needed. Our conclusion will be (1) Raw data must be kept in such it is output by the dataset and filtered by the method. A dynamic memory allo- cation will be needed. The data structure—dedicated for playing, editing or editing—will include variables data, relates to data from different points, data transformation rules in play before one. In a first stage, it will be event-oriented. We will distinguish between instances defini- tion data, note events and note modification data such as pitch bend or modulation (bend, aftertouch).

5. The clav

The clav is a very efficient UNIX scheme for dynamic memory allo- cation. It is based on a linked list called consisting of a changeful count and is open to the first and the last character in the buffer. Buffers are made of a linked list of blocks containing a bounded link and a character string—typically 256 w. 256. The important thing in that the blocks are aligned on 16-bit boundaries, depending on their size (128 or 256 — link plus character array).
At this stage of our research, the score is static and structured at the initialization time, but clearly several scores can be defined. A score is viewed as one track, even if it may actually use several synthesizer voices. Polyphony would be represented by several scores, but we haven’t studied yet the synchronization between scores. It would probably need a scheduler as in Flayer-LoP DMS.

Package includes (1) transformation functions such as allocate score, expand score or score, start loop and preset pointers to particular data, (2) resynthesizer outlined in that next previous or now memory with or without skipping a possible subscore, count scores, clear score and display, (3) record and play functions called by the device handler. Several strategies can be enabled by flag at the handler initialization.

Let us give examples to the case of MIDI data: let us recall first that the MIDI note event is made of four bytes: a byte tag, a channel number (in fact the instrument’s key number), and a velocity. We may want to use complete MIDI note event in a score or to refer independently to each of the four bytes in different buffers. For instance, let us see that we want to play a recorded pitch sequence with upbeat prominent or synthetic rhythm. Flags specify the way the score must be traversed; we may want to add or subtract omnigenous—represented by subvoices— from the main score. As it was suggested before, scores can have pointers to pitch bend or modulation data: if the pitch bend data appear before the note, an empty score is filled, with only the pointer to pitch bend data and a flag can indicate to the play function if the bend has to be played or not. Obviously, flags given to the record and the play function must be coherent, and it is convenient to set and reset them for both the same function.

4.5. Picture Matching

The data structure described above can be convenient at least for simple monochrome pattern matching. The pattern is specified in the form of a score to a point data. The pattern score is compared to the input data scores with pointers to incoming data updated at each new event. The FIFO scheme is part of the close package. The pattern score can be structured, which allows a simple specification of patterns such as tiles or a data for specified area through a regular expression. The literature on the subject is very prevalent and existing software is liable to be borrowed from the process, for example D. T. George and Blodgett (1983). The output of the pattern matching functions are monochrome.

4.6. Documentation

Although the project is in an early stage, a UNIX-like documentation is produced as soon as a function is written, and regularly updated. It can be a basis for specifying the interface between the modules to make software exchange possible.

5. SUMMARY

This paper presented some problems encountered by the musicians when using musical input to a real-time system. It has been tested out that a real-time system can act together as an instrument, an instrumentalist and can take compositional actions on top. A model in this paper has been proposed, which distinguishes culture extracting, compositional and performance processes. An example of implementation of a data structure has been given.

6. REFERENCES


