Acousmatics, Sound Objects and Instruments of Music

Dr. Marcus Alessi Bittencourt, College of William and Mary (mabitt@wm.edu)

Abstract: Based on propositions by Pierre Schaeffer, this paper will demonstrate in what capacity the computer should be regarded not as a musical instrument in itself, but as a virtual arena where pseudo musical instruments are instantiated. A case study that addresses these concerns is provided.

1.1 Intention

The intention here is to gain a truer understanding of the role of Computers and their algorithmical minds in the composition of Electroacoustic Music and to be able to design and operate virtual musical instruments which reflect the phenomenological perceptual concerns posed by acousmatic listening, the Schaefferian Écoute Réduite [Schaeffer 1966].

1.2 Musical Instrument

In chapters 2,4-2,5 of his "Traité", Pierre Schaeffer [Schaeffer 1966] defined a musical instrument as being a sound-producing device endowed with three characteristics. First, the device has the property of endowing its sounds with a particular timbre, a "marque d'origine", that allows us to recognize all the sounds as coming from the same source. Second, the device possesses a gamut of possible physical manipulations that, when applied, produce the gamut of available sounds. Third, it possesses a collection of playing modes, of manners of playing, in other words, a playing style.

1.3 Pseudo Musical Instrument

Due to the supreme generality of their sound-producing capabilities, devices such as the computer, the tape recorder, the sampler, normally used to manufacture Electroacoustic Music, in themselves cannot constitute musical instruments. Nonetheless, this equipment can be used to produce pseudo-musical instruments, that is, virtual instruments that do not exist in the real world (they only exist in the form of electroacoustic simulations) but nonetheless possess the same three characteristics that Schaeffer isolated for a musical instrument: an origin mark (phenomenological timbre), a finite gamut of possible physical manipulations (and their resulting gamut of available sounds), and a style of playing.

Thus, it is important to realize that when one controls computer audio software through computer interfaces of any kind, one is not actually playing the computer itself as a musical instrument. Instead, it is the simulation the computer instantiated which is being played.

According to the extent that his simulations conform to this notion of Musical Instrument, the artist-musician will be dealing with phenomenological ideas of solos, duos, trios, and so on, all the way to orchestras of these virtual instruments.

With this pondered, in order to design such simulations of a Musical Instrument one should consider the perceptual unity of the Sound Objects [Schaeffer 1966] produced by the simulation, and how these are assembled into Musical Objects through the operation of a Musical System.

2.1 Acousmatics and Sound Objects

Originally, "acousmatic" was the name given to the disciples of Pythagoras who, for five years, had to listen to the lessons from behind a curtain, without seeing the master and in absolute silence. Resuscitated, this term is now used to define a sound that one listens without "seeing" (= caring for) the source where it comes from, a sound that disconnects from its source and becomes something else, a sound that disincarnates from its daily ordinary function of Source Index, thus entering the realm of Music.

It is this acousmatic way of listening that brought forth the Schaefferian term of "Sound Object": a perceptually cohesive sound event, listened with an acousmatic intention, in other words, perceived and appreciated for its own sound-value sake.

2.2 Musical Object

A Musical Object [Bittencourt 2003] is a collection of Sound Objects of any size, small or big, that encloses in itself a single and recognizable complete thought. Because it represents a complete thought, it has a definite beginning and an end: its boundaries can be assessed. Because it is recognizable, it can be repeated, varied, transmutated, combined with other objects, traced by our memory.

According to its "main course", to its foreground main ideas, a musical object can be said to gravitate between two poles: static, if the spotlight is focused on the constituent sound elements themselves, or dynamic, if the spotlight is focused on the internal evolution of the constituent sound elements.

In reality, all musical objects fluctuate somewhere between these two antipodes. The "static" nature refers

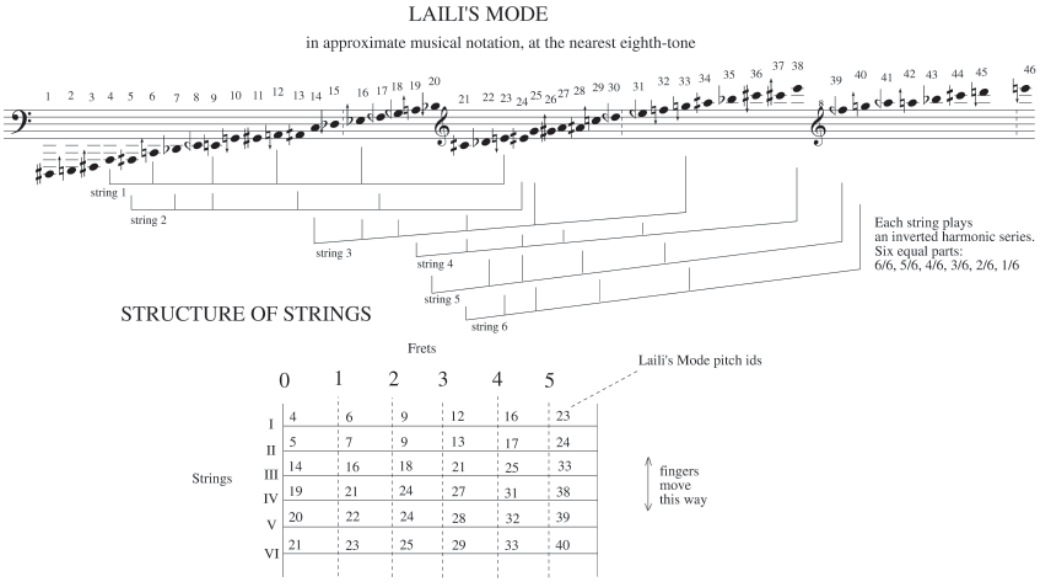

Fig. 1. Structure of Laili's Mode, and Tusk Harp strings

to operations in Musical Space, the "dynamic" nature, to operations in Musical Time [Bittencourt 2003].

2.3 Musical System

A Musical System is any set of rules that directly restricts the choices of sound possibilities. In other words, it is a set of constraints. To invent a Musical System is to create a set of rules that limit the use of the continuum of the characteristics of sound and that specify the universe of manipulations possible.

3.1 Case Study of Pseudo Musical Instrument and Musical System

I will here describe a reasonably-successful collection of algorithms I created for the seventh scene of my radiophonic opera *KA*, based on a story by Vielimir Khlebnikov. The interest here is that these algorithms materialize at the same time a pseudo-musical instrument and a Musical System with a precise collection of possible notes and timbres distributed in space, strict rules for manipulating these possibilities, and a complex rhythmical system.

3.2 General Description

Scene seven is supposed to contain an instrument made of an elephant tusk with five strings (later on, six) attached to the tusk by pegs of years. The five years on top show the times when the East invaded the West, and the five in the bottom, when the West invaded the East. It is also mentioned in Khlebnikov's story that each string is divided in six parts. Trying to conceive an image of this fantastic instrument, I thought of a C++ class that would "speak" through the RTcmix STRUM instrument.

First, I studied what happens when a string is divided in six equal parts. With frets positioned at those six points, your string will be set to play an inverted harmonic series: if 1x string length produces C3, for example, (5/6)x gives Eb3, (4/6)x gives G3, (3/6)x gives C4, (2/6)x gives G4, (1/6)x gives G5, everything in the natural tuning of the harmonic series, obviously.

Because this tusk harp was supposed to accompany the character Laili singing, I wanted it to use the "Laili mode", a microtonal scale I developed using another piece of software of mine, the ModeGenerator (see Figure 1). Thus, I searched for possibilities of finding six collections of six notes in this mode that could conform to that "minor chord" formation described above, with a maximum margin of error of a quarter tone, as if the strange tuning generated from the use of the Laili mode was derived from the frets being positioned slightly off from the equal string subdivisions.

To prevent the instrument from playing only arpeggios, the fingers were thought to move across and not along the strings. I was supposed to imagine five virtual fingers moving across the fretboards according to strict rules of fingering.

Each string of the tusk harp has its own fixed stereophonic positioning and a unique set of STRUM parameters so that each string possesses a different particular timbre. Also, a string has to be prepared to never play two notes at the same time. Unless it is played again, the string has to continue vibrating till the extinction of the sound, but if a string is still vibrating when a new pluck order is given, the previous note has to be stopped accordingly.

3.3 Implementation

To code such an instrument in C++, I designed a system of 3 classes.

A Strumline class is used to hold an RTcmix STRUM command and to keep track of its current state, if it is

still alive (vibrating, that is), or not. There are four basic methods: one to set the STRUM command line, one to recall it, one that verifies if the previous note is still vibrating, and one that adjusts the length of the previous note (i.e. turns it off).

Next, we have a Tusk_String class that contains one Strumline object and is used to control all the operations necessary for a string to play. Here we have five methods: one to set the output printing stream, one to set the pitches (in Hz) for each of the six positions along the string (five frets plus the open string), one to set the STRUM timbral parameters and the stereophonic positioning for the string, one to receive and realize playing commands, and finally, one method to flush the last Strumline class buffer.

Finally, we have the Tusk_Harp class, which contains six Tusk_String objects (one for each string, of course). The constructor method initializes the strings with the STRUM timbral parameters selected, their stereophonic positioning, and the pitches that the string frets are supposed to play. The Play() method, the one used inside another program to actually play the harp, receives only the parameters string, fret, point in time to start playing, and amplitude. It functions basically as a routing system, relaying the information to the appropriate string.

With all this, the very complex operations required to play the tusk harp and materialize its results into sound are hidden from the main user. Inside the actual algorithm that generates a musical piece, the user has access to the Harp simply by calling its Play() method.

The next step to play the Tusk Harp is to formalize the fingering rules. Remember that the fingers are thought to move across the fretboard and the strings, and that you can play with all five fingers.

When moving fret-wise (horizontally, if we imagine the strings running parallel to the ground), we can either keep in the same fret, move to its neighbors or to no fret (open string). From an open string, we can return to any fret. String-wise (vertically), you can move according to the availability of fingers. The fingers are numbered from 1 to 5, in reverse order than the piano fingering tradition. You can move to a new string if there is a finger available in that direction, remembering that two adjacent fingers do not have to necessarily move string by string, that is, jumps are allowed.

Chords up to six notes are possible and depend on the position of fingers at each moment. Since from a fret you can only reach its neighbors, only two adjacent fret regions (of different strings, obviously) can be stopped simultaneously. For a six note chord, at least one of the notes has to come from an open string.

All this has been programmed into two methods: one that performs the melodic changes of position, and one that creates chords.

Finally, to make the tusk harp play some musical fragment, we still have to add rhythmical procedures. The one I used here is based on a fixed row of a user-defined number of durations. These durations are chosen at the beginning of the algorithm, between 0.3 and 1.0 second, scaled by a "speed" proportion also defined by the user. The way to deploy this set of durations is a little bit intricate, but it generates very interesting "syncopations".

First, a certain number of successive notes to play is defined, chosen between 2 and 7. The durations are always used in the same order they were originally chosen, but the first action performed for a successive group of notes is to hold the duration on the top of the pile as a rest to be performed at the END of the group of notes. As an example, figure 2 shows the rhythmical result when we have a row of three durations, a, b and c, and we play three groups of notes with lengths of 4, 2, and 4 notes, respectively.

The main program will be controlled by the user. At the command line of the program, he has to define the name for the output soundfile, the total duration of the musical fragment, the speed (the multiplication factor for the row of durations), the forbidden string (because the instrument will sometimes have only five strings), the number of elements of the duration row, the sound output mode (real time or disk space), and a random seed. The output result of the program is identical every time you run it with the same seed.

4.1 The Function of Algorithmical Composition

Although the decisions of what to play in the Tusk Harp example are made randomly, these decisions revolve around a system of probabilities based on strict sets of constraints. As you would expect from a Musical System, these algorithms "taint" the sounds that come from it in a very recognizable way. For example, the horizontal profile

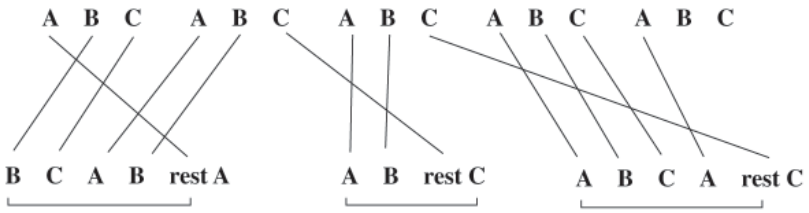

Fig. 2. Rhythmical procedure for the Tusk Harp

of the harp melodies and the structure of its chords are totally dependent on the fingering rules. An important point to notice is that these algorithms were created not to generate a musical passage, but to generate kindred musical materials. Here we have a finite set of possible sounds, carefully chosen so that they all seem to emanate from the same source (they bear the same "origin mark", the same phenomenological timbre), and we also have a playing style, generated from the coupling of the rhythmical system and the fingering rules. In other words, the C++ code materializes a pseudo musical instrument, in Schaefferian terms.

References

[Bittencourt 2003] Bittencourt, Marcus A. "Doctor Frankenstein, I Presume... or The Art of Vivisection". Doctoral dissertation at Columbia University, New York, 2003.

Khlebnikov, Vielimir. KA. inside The King of Time . Harvard University Press, Cambridge, Mass., 1985.

[Schaeffer 1966] Schaeffer, Pierre. Traité des Objets Musicaux. Éditions du Seuil, Paris, 1966.