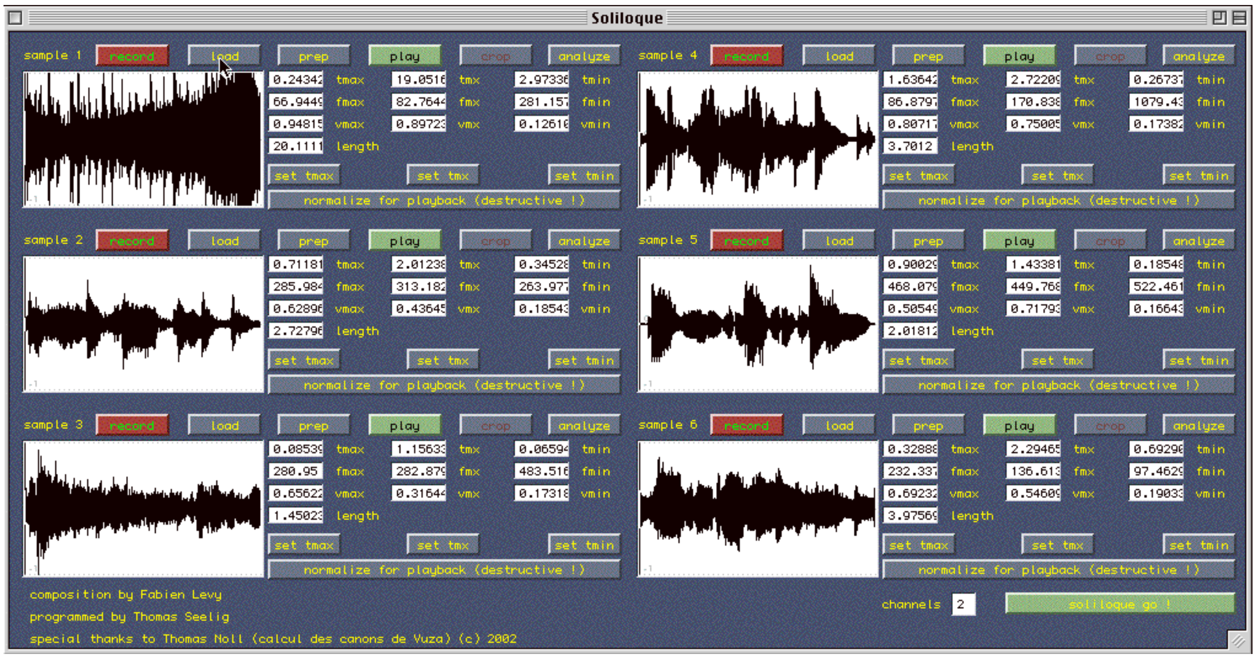
SOLILOQUE SUR [X, X, X ET X], MUSICAL COMMENTARIES FROM A COMPUTER ABOUT A CONCERT MISUNDERSTOOD BY IT

Fabien Lévy
Composer, Hanns-Eisler Conservatory, Berlin
fabien.levy@gmx.net

Thomas Seelig
Programmer, Electronic Studio TU-Berlin
seelig@snafu.de



```
sample 1
0.24342 tmax   19.0516 tmx   2.97336 tmin
66.9449 fmax   82.7644 fmx   281.157 fmin
0.94815 vmax   0.89723 vmx   0.12616 vmin
20.1111 length

sample 2
0.71181 tmax   2.01238 tmx   0.34528 tmin
285.984 fmax   313.182 fmx   263.977 fmin
0.62896 vmax   0.43645 vmx   0.18543 vmin
2.72796 length

sample 3
0.08539 tmax   1.15633 tmx   0.06594 tmin
280.95  fmax   282.879 fmx   483.516 fmin
0.65622 vmax   0.31644 vmx   0.17318 vmin
1.45023 length

sample 4
1.63642 tmax   2.72209 tmx   0.26737 tmin
86.8797 fmax   170.838 fmx   1079.43 fmin
0.80717 vmax   0.75005 vmx   0.17382 vmin
3.7012  length

sample 5
0.90029 tmax   1.43381 tmx   0.18548 tmin
468.079 fmax   449.768 fmx   522.461 fmin
0.50549 vmax   0.71793 vmx   0.16643 vmin
2.01812 length

sample 6
0.32888 tmax   2.29465 tmx   0.69296 tmin
232.337 fmax   136.613 fmx   97.4629 fmin
0.69232 vmax   0.54609 vmx   0.19033 vmin
3.97569 length

composition by Fabien Levy
programmed by Thomas Seelig
special thanks to Thomas Noll (calcul des canons de Vuza) (c) 2002

channels 2
```

Figure 1. User Interface of *Soliloque sur...* (with the visualisation of the six samples and some related acoustic data)

ABSTRACT

To compose a composer... *Soliloque sur [X, X, X et X]*[1] is not just a musical work, but more a meta-score, generated by the computer in real-time from other works of the concert. It means that for every concert, the generated work is different, in its sounds as well as in its organization. This computer application also requires a particular real-time technology, because it involves changing a signal processing network "on the fly" without sound interruption.

1. TO COMPOSE A MINI-COMPOSER

1.1. When the computer improvises

Soliloque sur [X, X, X et X][2], *Commentaries from a computer about a concert misunderstood by it* is not really a musical work, but more a meta-work, that the computer generates in real-time from the analysis and the sounds extracted from the samples of other works of the same concert. It means that for every concert, the generated work is different. Different, on the one hand, because the material which constitutes this mosaic is made from samples of the other works, as if the computer was remembering a few sonorities heard before in the concert. And different, on the other hand, because the organization, that is, the "score" of the work, is

[1] The application *Soliloque sur[X, X, X]* is freely downloadable with install explanations in the web-address: http://membres.lycos.fr/fabienlevy/SOLILOQUE.html or in: http://homepage.mac.com/thomas.seelig/TheSite/SuperCollider/Soliloque.html
Soliloque works on a PowerMac G4, System Mac OS 9.2, 867 MHz or more, with the freeware *SuperCollider* 2.2.x

[2] To obtain the exact name of the current work, the [X] will be replaced with the first name of the other composers played at the same concert. The first version, *Soliloque sur Francesco, Emanuele, Agostino et Nicola* has been premiered on July 5th, 2002 during the Festival *Inventionen* in Berlin. Further versions of *Soliloque* have been presented in Italy and Germany.

numerically elaborated in relation to the acoustic data of the samples. In order to calculate these acoustic values, the computer makes an acoustic analysis of each sample beforehand.

Aesthetically, the resulting work is less a combination of audible quotations of the other pieces of the concert, than a kind of deconstructed musical commentary by a computer of this concert. Thus, the samples, which consist generally of a chord here, a short motif there, do not exceed twenty seconds, and are never quoted longer than two seconds. They are also deeply transformed and overlapping with the others. Also, even if the work is different every time, the musical language remains about the same, faithful to the author's processes of composition as well as to his musical preoccupations.

1.2. Techniques and constraints of composition

Let us first note that the generating algorithm for *Soliloque* is entirely deterministic, without randomness: with the same samples and the same positions of analysis, the generated work would remain the same. The general algorithm consists of several overlapped processes of varying lengths, varying starting time, and varying progression, according to the values of the parameters calculated by the acoustic analysis of the samples.

Most of the processes come from Lévy's instrumental composition techniques. This permitted, on the one hand, to create a dense "sonority" of agglomerated small cells, like tiles in a large mosaic, familiar in his music. This principle would also avoid the long flows of sound and morphologies present in too many recent concrete music works. For instance, we transposed the principle of *transparametrical inflexion* to the world of the concrete sounds. This technique consists in repeating a sound at a periodicity in connection with the frequency of the fundamental of this sound, and then to combine different pulsed sounds in order to weave dense mosaics (figure 2).

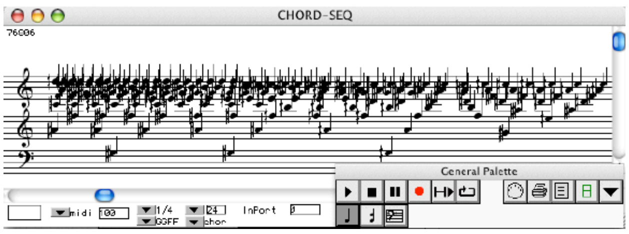


Figure 2. Principle of transparametric pulsation in instrumental music (primary drafts made on the software Open Music for the Lévy's orchestral work *Hérédo-Ribotes*)

In *Soliloque sur…*, the compositional techniques are transposed from traditional parameters of instrumental music (rhythm, tones, nuances) to acoustical parameters: the processes do not act on tones but on sound objects, not on rhythms but on positions of sound-reading and on duration (the pulsed object is for instance read in various positions around a precise point, that generates rhythmical sentences of various accents), not on intervals but on sound reading speeds, not on nuances but on intensities, filtering and space movement. The pulsed sound is then transformed with a half dozen of time-depending processes of sound transformation (filtering, ring-modulator, resonant filters, time transposition, forward-reading, complex spacing, granular synthesis, etc.). Aesthetically, this projection of instrumental musical processes applied to acoustic categories allowed one to compose an electro-acoustic work based on highly symbolical structures and form.

The techniques used in *Soliloque*, like for instance the *transparametrical pulsation*, thus outline short primary cells of various colours, built from the interlacing of different transformed and pulsed samples, which represents a kind of granular but deterministic synthesis. Yet a technical constraint related to the capacity of the computer has limited this coloured braiding: Each cell, of a maximum length of one second, is made of approximately three sounds extracted from three samples, read at various positions, at various speeds, modified by several combinations of transformations such as time-filtering, forward-reading, ring-modulation, time-speed-variation, or 3D-movements in space. It means that the rendering of a primary cell already requires numerous real-time data-processing calculations, which has then to be superimposed with other primary cells according to the musical processes. In the prior phase of research, the CPU of the computer sometimes saturated to more than 200 %, then requiring a technical and musical limitation in the superposition of cells.

1.3. An electro-acoustical application of the Vuza Canons

It was thus necessary to find some stratagem in order to avoid too much superposition of cells in the mosaic, while still giving musically the impression of a dense counterpoint. For instance, the final part of *Soliloque sur [X, X et X]* consisted formally of an explosive accumulation of numerous primary cells in the space. In order to respond to this technical constraint, we transposed in the electro-acoustical world the technique of the *Vuza canons*. The *Vuza Canons* are rhythmical canons, with a pattern of augmented identical rhythmical sentences, which sum together with neither empty space nor superposition[1].

[1] Those canons, whose manual elaboration is impossible, were calculated by the Rumanian mathematician Theodor Vuza, and then largely studied and implemented on computer by the Italian mathematician Moreno Andreatta (Andreatta, 1999) and, concerning the particular case of the augmented canons, by the German mathematician Thomas Noll.

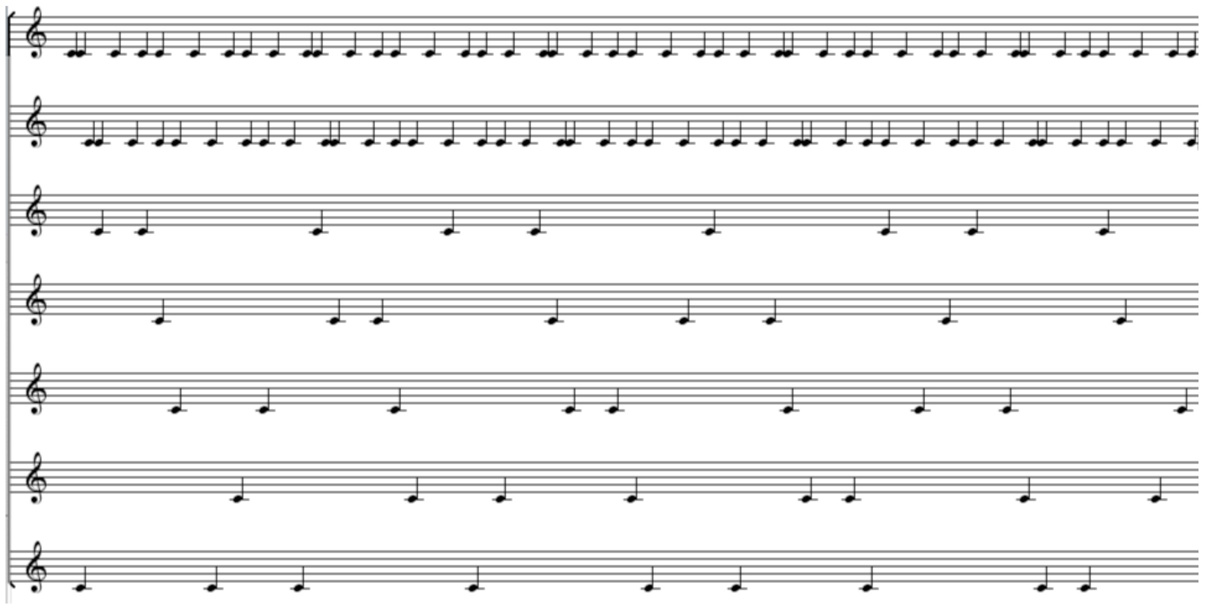


Figure 3. Musical representation of an augmented seven-voices Vuza Canon used in *Soliloque sur...*

2. COMPOSING BOTH HIGHLY SYMBOLIC STRUCTURES AND SIGNAL PROCESSING IN REAL TIME INVOLVES PARTICULAR TECHNOLOGIES

Soliloque sur [X, X, X] is not just a musical piece but also a software engineering project. In the beginning a decision had to be taken as to which development environment would be useful to realize the concepts given above. The most intuitive computer music language is probably – due to its graphical approach – *Max/MSP* and its cousins *jMax* and *PD*. Unfortunately this development system follows a quite static concept concerning signal processing networks. Before the audio processing starts, everything has to be well defined and in place and it is virtually impossible to change a signal processing graph "on the fly", instantly, without sound interruption. Efficiency is also an issue since the audio processing mechanisms under the hood of Max involve a lot of data shuffling. Max is easier to create a user interface for a program. A language like *SuperCollider* offers, on the contrary, the possibility to create and destroy "sounding objects" during runtime. Formulating algorithms with text instead of graphics makes construction of iterations and decisions also very easy. However, the design of a user interface is quite awkward but possible.

SuperCollider, in short, is a development system with a build-in editor, a runtime system, debugging features and a language syntax similar to Smalltalk and C++. In fact, it is a fully featured object oriented system. *SuperCollider* comes with a lot of unit generators for tweaking sound and a large library for generating streams and patterns which made it perfectly suited for what we tried to achieve. To quote the inventor, James McCartney: "*The user can write both the synthesis and compositional algorithms for their piece in the same high level language. This allows the creation of synthesis instruments with considerably more flexibility than allowed in lower level synthesis languages.*"

Although a different approach would be possible, we used the "classic" distinction between user interface, instrument definitions, and the score, as known from languages like *CMusic* and *CSound*. The difference with *CSound* is that the score itself is made up of a lot of so called "patterns", which generate the actual sounding events. *SuperCollider* has no linker mechanism as languages like C++ but a somehow similar mechanism in maintaining a precompiled library of functions. For this library one can define new classes which can be used in the main program. *Soliloque sur...* uses a class named "*SampleView*" which defines functions for the user interface and the sample handling before the actual program run.

```
SampleView {// CONSTRUCTION OF THE USER'S INTERFACE
    var <>tmax = 0,     // declaration of variables
        <>tmx = 0,  etc ...
//---------- new (constructor) ---------------
*new{arg                    aWindow,
        aXOffset = 0,
        aYOffset = 0,
        aLabel = "Sample x",
        aSoundFileName;
    var newInstance;
//--- create an instance of SampleView -----------
    newInstance = this.on(aWindow, aXOffset, aYOffset, aLabel,
aSoundFileName);
//--- create the interface elements --------------
    newInstance.createViews(aWindow, aLabel, aXOffset,
    aYOffset);
//--- Record (bind an interface element to an action) -
   newInstance.itsRecordButton.action_({newInstance.recordProc});
    etc ...
    }
    etc ...
}   // SampleView
```

Figure 4. Class definition for a sample handling user interface object (excerpt)

SampleView defines functions for the recording and playback of samples, a display, the definition of the "t", "f", and "v"- values used in the calculations of the patterns and much more. This class must be loaded and compiled before the actual Soliloque-Patch. From now on the class is known inside the system.

As noted before, *Soliloque sur…* makes use of mechanisms called "streams" and "patterns". A stream in *SuperCollider* is every object which responds to the messages "next" and "reset". It is possible to use math. operators to produce streams, and on top of this, "patterns" are a way to create multiple streams from a single specification. Examples for predefined patterns are:

```
Pseq(#[a, b, c,...], count); -> sequential repeat the elements
 of the array #[a, b, c,...]
Pser(#[a, b, c, ...], elementCount); -> output of “elemntCount”
elements
Prand(#[a, b, c,...], count); -> random choice
Pxrand -> same as above but like choice from an urn
Pshuf(#[a, b, c,...], count); -> shuffle, then output
Pseries(start, step, len); -> arithmetic series
Pgeom(start, faktor, len); -> geometric series
```

Like streams, pattern can be added, subtracted, etc.. Messages like *p.squared* (where the variable p contains a pattern) are understood. *Collect*, *select* and *reject* are filter operations on patterns: *collect* applies a function on all elements of a pattern, *select* uses boolean operations and *reject* is the negation of *select*. Interesting in the context of *Soliloque sur…* is that *Pfunc* gives a pattern of functions, e.g. different envelope generator functions can be send to an instrument as argument:

```
Pbind(\x, Pfunc({[{f1},{f2},{f3}].choose.value;})); -> This
line selects randomly one out of an array of functions and binds the
name x to it.
Ppar([pattern1, pattern2,...], repeatCount) -> plays patterns in
parallel.
This expression can be nested: Ppar([Ppar([p1, p2]), Ppar([p3,
p4])])
```

Finally a more complex example below shows a variation of a variable (here the variable *reading speed* of the sample sf1) as a complex trigonometric function of time, *cosFunc*. The actual time is taken from the number of the iteration. It was indeed necessary, for *Soliloque*, to build complex functions of time with a light algorithmic complexity (for instance a sine function of sine of time).

```
cosFunc = // CONSTRUCTION OF THE TRIGONOMETRICAL FUNCTION
                         cosFunc

    Pn(                  // repeat pattern
    Ppatmod(             // modify pattern
    Pseq(#[0]),          // Dummy for syntax reasons
    {arg pat, i; cos(0.125*i*2pi)},        // function, i: counts
                                                 iterations
    inf),                // forever
    inf);                // forever
```

```
p1 = Pbind(                // COMPOSITION OF THE PRIMARY CELL p1
    \soundFile, sf1,       // sf1 : name of the first sample in this
                           primary cell using a crossfade
    \soundFile1, sf3,      // sf3 : name of the second sample in this
                           primary cell using a crossfade
    \dur, Pseq([0.6],16), // duration of the repetition of the quote,
        here given in Pbind: 16 pulsations every 0.6 seconds
    \length, 0.5,          // length of the quotation : 0.5 seconds
                           (then 0.1 second of silence)
    \theCrossFreq, 0.5, // Parameter for the crossfade
    \volEnvIndex, 5,       // Volume
    \speed, cosFunc,       // time changing speed of the quotation,
                           using the complex function cosFunc
    \start, rampFunc,      // time changing position of the quotation,
                           using the complex function rampFunc
    \ugenFunc, instXFade // instrument used for this cell
    (instXfade = pulsed quotation of two samples with a crossfade)
);
```

The main *Soliloque* program starts with the declaration of global variables. Variables in SuperCollider are type-free, so they can contain anything from simple numbers to complex patterns. After the initialization, follows the definition of the instruments: *instSample*, *instFunc*, *instFiltr* (several variations), *instXFade*. The prefix *inst…* is just a private notation to keep the meaning of a variable clear. The following example shows the definition of the sample player *instSample*:

```
instSample ={ // CONSTRUCTION OF THE INSTRUMENT INSTSAMPLE
                         (sample player)
    arg soundFile, speed = 1, vol = 1, start = 0;  // arguments with
        default values (speed, intensity and position of reading)
    var theSound, itsLenght;                   // local variables
    start.clip(0.0, 1.0);
    theSound = soundFile.data.at(0);
    itsLenght = theSound.size - 2;             // size in Samples
    PlayBuf.ar(theSound, soundFile.sampleRate, speed,
                   start*itsLenght, 0, itsLength, vol);
                         // play it with audio rate (ar)
    };
```

SuperCollider denotes messages to objects with a dot, e.g. the variable start (which is of course an object which understands many messages) gets the message "clip" with the arguments (0.0, 1.0). After the definition of the instruments and the function, the “score” definition starts. To be clearer, we jump to the end of the program text:

```
Ptpar(                     // FINAL SCORE OF SOLILOQUE :
    [0.0,teil1.value,      // plays Part 1 of Soliloque at time [0.0]
    globalT1,teil2.value, // plays Part 2 at time [globalT1]
    globalT2,teil31.value, // plays Part 31 at time [globalT2]
    globalT31,teil32.value, // plays Part 32 at time [globalT31]
    globalT32,teil33.value, // etc...
    globalT33,teil4.value
    ], 1).play(channels: numChnls); // plays Soliloque on
                           [numChnls]channels (2 to 16 channels)
```

This is the highest level in the score's hierarchy. *Ptpar* means "start all of the subpatterns at a given time". "*teil1.value*" tells the function "teil1" (= "part1") to evaluate itself and *Ptpar* will start this pattern at time 0. The actual values of *globalT1…globalT33* are calculated from the results of the analysis of the six sample files.

The definition of the function *teil1* starts after the instrument and utility function definitions (*cosFunc*, *rampFunc* etc.) near the beginning of the program text. The function consists of a collection of pattern definitions, the primary cells, stored in variables systematically named like *t1c1p1, t1c1p2,* etc.. An example is given below (fig. 5).

3. A SYMBOLIC COMPOSITION OF CONCRETE SOUNDS

The project *Soliloque sur [X, X, et X]* had two main aesthetical aspirations:

On the one hand, we wanted to compose a work using complexes and non-harmonic concrete sounds, but at the same time achieving a high symbolic level of organization, such as which exists in the works written with a traditional instrumental score. Indeed, although digital techniques today offer more precise methods in order to compose with concrete sounds, composers are still split between two attitudes: Composers who continue to compose tones on the traditional score that offers a high level of thought of the writing but compel them to use traditional musical categories that are today limited (sound-complexes reduced to harmonic sounds and its fundamental, temporal evolution of a sound reduced to discrete duration and discrete nuances, etc...); And those composers who combine complex sounds with electro-acoustic software whose interfaces still remain intuitive and rudimentary. Indeed, except for a few really autonomous types of software, and except for those composers who decide to write directly in a computer language (Risset, Chowning,…), a high symbolic level of composition of concrete sounds remains difficult at present. For instance, with the actual software it remains difficult to move a sound in a 3D space along a precise and complex curve[1], to filter or transpose a sound according to precise temporal evolutions depending on the analytical evolution of other sounds, or to modify a variable in real-time along a complex time-changing process. Those transformations are however easily simulated on traditional music paper. *Soliloque sur [X, X, X]* tried to reach this symbolic level of composition while working on acoustical categories (reading-speed, curves of space-movement, time-filtering, etc..). Therefore, this work was written entirely in the computer-language *SuperCollider.*

On the other hand, owing to the necessity of programming in code in order to reach this high symbolic level of composition of the complex sound, the second aesthetic aspiration was to not to choose precise numerical constants in order to define quantitatively the musical transformations. It was rather our intent to abstract them into parameters depending on the preliminary analysis of the samples. For instance, rather than choosing a fixed numerical value of duration and speed of pulsation, it was interesting to understand how this value, in general, would be chosen by the composer depending on the frequency of the sound in pulsation, and to abstract that rule in a process for every possible sample. These abstractions then motivated the open form of the work related to a context (in this case, of the other works heard in the concert). We hope to pursue with further projects this research of open-contextual works with a high level of symbolic composition (which means continuing the quite megalomaniac project to compose mini-composers !).

```
// -------COMPOSITION OF THE CELL T3C1---------
t3c1p1 = Pbind(\ugenFunc, instFiltr,\soundFile, soundFile1,
\start, tmin1 + 0.12*(l1/l3)*cosFunc, \speed, 5.01, \len,
0.047*(l1/l3),\freq, fmin5*(1.3+cosFunc), \q, 1.3,\frqEnvIndex,
2, \dur, Pseq([0.057*(l1/l3)], t3dur1/(0.057*(l1/l3))), \pan,
compFunc1, \vol, (1-vmin1)*(1-compFunc1) +(0.2*(1-
vmin1)*cosFunc), \volEnvIndex, 3);
t3c1p2 = Pbind(\ugenFunc, instFiltr,\soundFile, soundFile3,
\start, tmin3 + 0.1*(l2/l3)*cosFunc, \speed,
4.96*(fmin1/fmin3), \len, 0.046*(l2/l3),\freq,
fmin1*(1.4+cosFunc), \q, 1.2,\frqEnvIndex, 2, \dur,
Pseq([0.054*(l2/l3)], t3dur1/(0.054*(l2/l3))),\pan, compFunc1,
\vol, (1-vmin3)*(1-compFunc1) -(0.25*(1-vmin3)*cosFunc),
\volEnvIndex, 3);
t3c1p3 = Pbind(\ugenFunc, instFiltr,\soundFile, soundFile5,
\start, tmin5+ 0.16*(l4/l3)*cosFunc, \speed, 5*(fmin1/fmin5),
\len, 0.045*(l4/l3),\freq, fmin2*(1.2+cosFunc), \q,
1.5,\frqEnvIndex, 2, \dur, Pseq([0.048*(l4/l3)],
t3dur1/(0.048*(l4/l3))),\pan, compFunc1, \vol, (1-vmin5)*(1-
compFunc1)+(0.3*(1-vmin5)*cosFunc), \volEnvIndex, 3);
```

Figure 5. Example from *Soliloque sur...* how to write a cell (cell C1 of the part T3) in SuperCollider language. The cell T3C1 is composed with three pulsed primary cells t3c1p1, t3c1p2, et t3c1p3. Each primary cell uses a sample (soundfile1 for t3c1p1; soundfile3 for t3c1p2; etc.), which is filtered (instFiltr) according to a frequency related to the analysis of the fifth sample (fmin5) and of a temporal curve (fmin5*[1.3+cosFunc]). The sample is read in slightly changing time-positions in order to obtain rhythmical accents (\start, tmin1 + 0.12*(l1/l3)*cosFunc). This sample then is repeated with a duration related to the duration l1 or l3 of the samples 1 and 3 (\dur). Each sound of the cell has its own time-curve of nuances (\vol) and of movements in the space (\pan)

Digital technologies today offer new levels of aesthetical thought on the concept of musical work: it is possible to reach a high symbolic level of composition of complex sounds. Digital technologies also open up the possibility to compose complex open symbolic works reacting to the context or to various variables. In the future, these technologies will perhaps change the concept of score of concrete sound, and maybe somehow the concept of musical meta-work as well.

4. REFERENCES

[1] Andreatta, M. (1999), *La Théorie Mathématique de la musique de Guerino Mazzola et les canons rythmiques*, Mémoire de DEA, Université Paris IV, EHESS, Ircam, Paris.

[1] In the traditional software, the L-R curve is the only possible graphic interface, which allows to visually and precisely control the evolution of a space, but only in 2D. It means that it remains the only high level grammatological tool for space movement implemented in common software.