Heterogeneous data orchestration
Interactive fantasy under SuperCollider

Sébastien Clara
CIEREC UJM – PhD student
35 rue du 11 novembre
Saint-Étienne, France, 42000
sebastien.clara@univ-st-etienne.fr

Abstract

For *L'Imaginaire* music ensemble, I composed a piece of interactive music involving acoustic instruments and surround electronic music. The interaction between live musicians and electronics is based on data collected in real time from acoustic instruments. This data is further used to adjust timbre and synchronize electronics with the rest of the music.

Keywords

SuperCollider, interactivity, music mixed

1 Introduction

For *L'Imaginaire*[1] music ensemble, I composed a piece of interactive music mixed. Mixed music is a term used in musicological literature to refer to a musical genre. It is defined by the alloy of instrumental music and electronic music. For this paper, I wish to focus on an interactive property of my piece.

Historically, the electronic part of mixed music is composed in a studio and fixed on a magnetic tape[2]. This technique makes impossible the interaction between the interpreters and the electronic sound. Feedback helps to improve the compositional process of the tape. The other technique is to perform audio processing of the acoustic instruments in real time[3]. In this case, we are talking about a device that increases the sonic possibilities of the acoustic instrument. Feedback is used to regulate electronic sound by a human or an automaton.

I use these two techniques of electronic sound accompaniment in my work, but the increase in computing power and the versatility of the tools have allowed a median way. In the next chapter, I present my problematic. Thereafter, I will show my issue with an example which could be extrapolated to other devices.

2 Problematic

According to Robert Rowe, "*Interactive computer music systems are what are the changes in response to musical input. Such responsiveness allows these systems to participate in live performances, of both notated and improvised music*" [1]. How can a system listen to a musician and make an appropriate decision to generate a sound response? In his book, Rowe analyzes systems that use the *MIDI* standard to communicate between instruments and computers. But how can we use traditional instruments?

The audio descriptors[4] correspond to parameters that describe an analyzed sound. A set of descriptors is used to construct a data set and create spaces for representing the sound. The parameters that are extracted can be described in different ways, depending on what is expected from the information conveyed by the parameter.

The sound acquisition sensor for the processing unit is a microphone. Therefore, the use of audio descriptors in interactive computer music systems allows the use of standard acoustic instruments.

[1]Musical ensemble composed by Keiko Murakami (flutes), Philippe Koerper (saxophones) and Maxime Springer (piano). http://www.limaginaire.org/.

[2]The first pieces of music using this technique: *Orphée 51* and *Orphée 53* by Pierre Schaeffer and Pierre Henry (1951 and 1953) and *Musica su due dimensioni* by Bruno Maderna (1952 and 1958).

[3]The first pieces of music using this technique: *Mixtur* (1964) and *Mikrophonie I* (1964) by Karlheinz Stockhausen.

[4]With *SuperCollider*, it is necessary to install its extension package to benefit from a wide choice of descriptors : https://github.com/supercollider/sc3-plugins.

To control electronic sound with the sound of an acoustic instrument, many descriptors can be used. To do this, it is necessary to match a sound characteristic with the values of the parameters that describe it. The preparation of this report will allow us to establish a particular threshold and once it is crossed, the system can trigger a response.

However, the interval between the extreme values of a parameter depends on its nature and on the analyzed sound source. To use this technique of interaction between an instrumentalist and electronic sound, a first difficulty is to negotiate with the heterogeneity of the data produced by the different audio descriptors.

For example, I want to use the nuance and pitch of a sound to build a particular accompaniment. The amplitude descriptor returns a number that can range from 0 to 1 and the pitch descriptor returns a frequency. The range of extreme values returned by the pitch descriptor depends on the ambit of the analyzed sound source. Determining a trigger threshold to activate a response of a system involves crossing values of different magnitudes sometimes from sound sources of different natures.

Moreover, this technique uses sound capture and this information is variable depending on its environment. Consequently, elaborate settings in a studio with a particular electro-acoustic chain would be less effective in another location with a different electro-acoustic chain. So, how can we treat the heterogeneity of the data produced by different audio descriptors, different sound sources, different electro-acoustic chains and different concert locations, to achieve homogeneous electronic music accompaniment for each interpretation of the same piece?

3 Creating a repository

The first step is to establish a reference. This will serve as a standard for a single piece, a specific electro-acoustic chain and a particular place. We will be obliged to renew it each time one of these three parameters changes. Furthermore, this repository will allow us to characterize our data and to determine thresholds for performing a particular electronic sound accompaniment.

In the following example, we use the amplitude descriptor (*Amplitude.kr*) and pitch (*Pitch.kr*). The values are transmitted by the *OSC* protocol (*SendReply.kr*) to the *SuperCollider* clients at the */dataTrigger* address. The data is sent whenever an onset is detected (*Onsets.kr*). When instantiating the synthesizer, two arguments are available. The first determines the number of inputs of the signal to be analyzed on our audio interface (*SoundIn.ar*). The second determines the onset detection threshold.

```
(
SynthDef(\dataTrigger, { arg in = 0, onsetsThres = 0.5;
    var signal = SoundIn.ar(in);

    var chain = FFT(LocalBuf(1024), signal);

    var trig = Onsets.kr(chain, onsetsThres);

    SendReply.kr(trig, '/dataTrigger',
        [Amplitude.kr(signal), Pitch.kr(signal)[0]]);
}).add;
)
```

Figure 1

Execution of figure 1 will only give the definition of your synthesizer to *SuperCollider* audio server[5]. Synthesizer is not instantiated, so it does not work and it does not ask for resources to your hardware. We will run[6] it only when we want to receive data (figure 2).

We now have to build a data collector to constitute our repository. To do this, we use the *OSCFunc* object. It is fast responder for incoming *OSC* messages. We configure it with the previously defined *OSC* address. When a new message arrives from the analyzer, it executes a function. In this case, this function saves the amplitude and pitch of the signal in array global variables.

```
(
var onsetsThres = -9.dbamp;

~dataTrigger = Dictionary.new;

// Run analytical synthesizer
~analyzerTrig = Synth(\dataTrigger,[\onsetsThres, onsetsThres]);


// Saving data from the current analysis
~oscTrigger = OSCFunc({ arg msg;
    ~dataTrigger[\amp] = ~dataTrigger[\amp].add(msg[3]);
    ~dataTrigger[\pitch] = ~dataTrigger[\pitch].add(msg[4]);
}, '/dataTrigger');
)


// At the end, free objects
~analyzerTrig.free; ~oscTrigger.free;
```

Figure 2

We set to -9 dB the onsets detection threshold. We can change this parameter for change the density of the data reception. The analyzer listens to the first input of our audio interface (default setting). In figure 2, running the first block starts the acquisition of the data. Running the last line kills the synthesizer and responder instances, frees memory and processor usage. The collected

[5] http://doc.sccode.org/Classes/SynthDef.html.

[6] http://doc.sccode.org/Classes/Synth.html.

information is stored in variable arrays. We can plot data in graph or histogram (figure 3).

```
~dataTrigger[\pitch].plot;
~dataTrigger[\pitch].plotHisto;
```

Figure 3

The graph shows the evolution of the parameter over time and allows us to make a correspondence between a sound characteristic and some values. The histogram provides a representation of the distribution of the values of the parsed parameter. This observation allows us to characterize the distribution produced by a descriptor.

4 Using the repository

In our system, we determine the value of a threshold to trigger a response to accomplish dynamic electronic music accompaniment. This choice may be arbitrary or be determined in response to a specificity of the analyzed sound source. Above a certain value, our program triggers a response for example. We can also choose this value according to its frequency of appearance in the distribution and according to the sound result produced by this choice, we can increase or decrease the density of our electronic accompaniment by modifying the value of our threshold. However, how do we handle values of different magnitudes?

We manipulate repository by the requested percentile. To use this method, it is necessary to install an additional library. For that, you can use the *SuperCollider* package manager[7] to install *MathLib*. This one gives us access to additional statistics methods for arrays.

The percentile rank corresponds to the proportion of the values of a distribution less than or equal to a determined value. We manipulate our data with float values from 0 to 1. For example, if we want to know the value equal to 90% of our data, we use 0.9.

```
~dataTrigger[\pitch].percentile(0.9);
```

Figure 4

During the adjustment phase of our system, we can tune several parameters of different magnitudes transparently with a single scale.

5 System Response

In order for our system to respond to certain stimuli, we must attribute to it a means of sound production. To do this, we define an arbitrary synthesizer (Figure 5).

```
(
SynthDef(\fmGrain, {arg out = 0, amp = 0.75, density = 20, carfreq
440, modfreq = 200, modIndex = 1, pos=0, dur = 3;

    var env = EnvGen.kr(Env.perc, levelScale: amp, timeScale:dur,
doneAction: 2);

    var signal = FMGrain.ar(Impulse.ar(density), 0.05, carfreq,
modfreq, env*modIndex, env);

    Out.ar(out, Pan2.ar(signal, Line.kr(pos, pos.neg, dur)))
}).add;
)
```

Figure 5

To implement a concrete example, we assume that our device listens to two types of percussion. One of the percussions emits sounds high-pitched than the other. We decide that our system will respond only to the percussion which emits the most high-pitched sounds and to the most loud sounds. With onsets detection threshold, our condition for triggering a response depends on two others parameters:

```
pitchThres = ~dataTrigger[\pitch].percentile(0.7);
ampThres = ~dataTrigger[\amp].percentile(0.5);
```

Figure 6

If the frequency of the answers does not suit us, we can return to the choice of the values of these variables to modify the sensitivity of our system.

```
(
var pitchThres, ampThres, onsetsThres;

// Interaction control
pitchThres = ~dataTrigger[\pitch].percentile(0.7);
ampThres = ~dataTrigger[\amp].percentile(0.5);
onsetsThres = -9.dbamp;


// Run analytical synthesizer
~analyzerTrig = Synth(\dataTrigger,[\onsetsThres, onsetsThres]);


~oscReply = OSCFunc({ arg msg;

    // Condition for reply density
    if((msg[4] > pitchThres).and(msg[3] > ampThres), {

        // Mapping control
        Synth(\fmGrain,[
            \density, msg[3].linlin(ampThres,1, 3,20),

            \carfreq, msg[4].linlin(
                pitchThres,
                ~dataTrigger[\pitch].maxItem,
                ~dataTrigger[\pitch].percentile(0.05),
                ~dataTrigger[\pitch].percentile(0.95)),

            \modfreq, msg[4] * msg[3].linlin(ampThres,1, 0.5,3),

            \modIndex, msg[3].explin(ampThres,1, 1,20),

            \dur, msg[3].linlin(ampThres,1, 2,10),

            \pos, 1.0.rand2
        ]);
    });
}, '/dataTrigger');
)
```

Figure 7

[7] http://doc.sccode.org/Classes/Quarks.html.

The implementation of a response for our system follows the same structure as Figure 2. A responder wait for incoming *OSC* messages from the analyzer. When a new message arrives, if it meets the previously formulated conditions (high-pitched and loud), a response is issued.

This response is customized according to the analysis data. This connection is made by a sonification process, "*technique of rendering sound in response to data and interactions*" [2]. I do not deal with the mapping technique in this paper, but its mastery is a source of variation and expressiveness for electronic music. To make this relation, we map the synthesizer parameter from an input range to an output range. We can set the input range with the repository information and adjust the output according to the desired sound quality. Figure 7 is the implementation of the response of our system.

At the end, we must free objects for frees memory and processor usage.

```
~analyzerTrig.free; ~oscReply.free;
```

Figure 8

6 For further

For this paper, we concentrated our system to its simplest expression. In this chapter, we wish to develop it design. Some of these ideas were conceived during the development of our piece and others afterwards.

Robert Rowe divides his interactive computer music system (*Cypher*) into two sections [1]. The listener analyzes the data produced by a musician and the player delivers a musical response. The structure of *SuperCollider* source code implies this organization. Our analysis synthesizer is the listener and the function of the responder object for incoming *OSC* messages is the player. We keep these terms to locate the following points.

Our listener can also have an implicit function of time master. Indeed, we transmit the data of the analysis every time an onset is detected. But we can transmit them at a given frequency. The responses delivered by the system would then be have a beat.

We can use our listener to produce an automation (with *Env* and *EnvGen.kr* objects). An automation allows to control and to automate the variation of a parameter over a given time. In this way, we determine the evolution of any threshold or parameter.

We can parallelize other listeners who analyze other sound qualities (brightness, noise, dissonance, etc.) to achieve other triggers threshold and make complex electronic music accompaniment executed by our system.

For our player, we can define a maximum number of synthesizers executed in parallel in order to preserve the resources of the system and / or to control the acoustic density so as not to saturate our perception.

In addition, we can perform a certain musical process[8] or that feeds on our repository instead of running a simple synthesizer. The interactive system developed by Jean-Claude Risset for his duets for one pianist [3] is very interesting for this way. He uses *MIDI* data (pitch, velocity and duration) which he transforms according to traditional compositional operations: transposition, reversal, canon, etc.

7 Implementation

I control my audio processor by a graphical interface (Figure 9) and *MIDI* controller. The different audio tracks allow me to adjust the intensity of the electronic sound layers. The flute, sax and piano tracks deal with interactive electronic sound accompaniment. The synth track manages my non-real time composite electronic music. Finally, the live track amplifies the acoustic instruments.

The creation of the repository is realized directly in the interface and makes this action transparent. Graphical interface allows a sound engineer to play my work and this interface makes rehearsals and concerts easier.

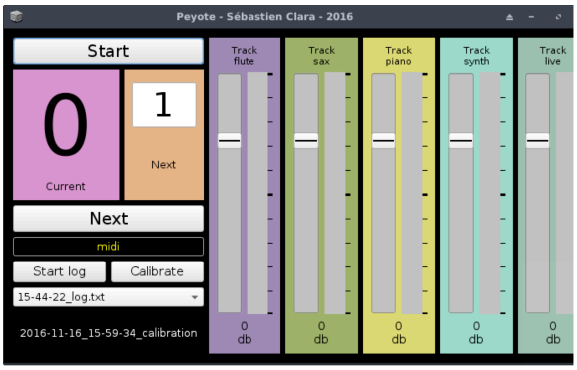

Figure 9

A *MIDI* pedal assigned to a performer manages the overall setting. Fourteen key moments

[8] http://doc.sccode.org/Tutorials/A-Practical-Guide/PG_01_Introduction.html.

articulate the electronics for a duration of fifteen minutes.

To create my electronic accompaniment, I use the descriptors of amplitude, pitch, centroid and noise. The operation of coupling between the data of the analysis and the parameters of the synthesizers [2] makes it possible to control the form of electronic music accompaniment by scaling, transposition or reversal. This relation can be fixed or variable.

Generally, the sound source of my electronic music accompaniment does not come from sound synthesis, but acoustic instruments. When the density of the responses of the system is not limited, the responses are superimposed to continuously transform the timbre of the electronic sound. When the density of the responses of the system is limited, the responses can arrive in successive waves and produce a dynamic accompaniment.

8 Epilogue

To design our system, our initial motivation was to simplify the use of different descriptors, to simplify our system settings, to customize the responses of the system according to the sensitivity of the interpreter and to produce a homogeneous electronic accompaniment to each interpretation of the same notated music under different conditions.

But in addition, we obtain an open interactive system that can be adapted from a specific model to the intuition of a musician. We identified four steps to explore in order to implement our practical solution and develop an interactive scenario [4].

The first step focuses on the sound of the instrumentalist. What particularities of sound do we want to relate to our system? What filter do we want to use to trigger an answer? In the example developed for this paper, we make our filter with the parameters of onset detection, pitch and amplitude. The thresholds established to constitute this filter allow us to play on the sensitivity or the particularity of the answers delivered by our system.

The second question to implement our solution is to choose the type of response to trigger. Should the answer be monophonic, polyphonic, contrapuntal, etc.? In other words, what organizational model should we use to develop our response? In our example, we produce one item per answer. This element is strongly correlated with the sound analyzed and the operations applied to determine the sound characteristics of our response are conceived during the last step of this practical solution.

The third step in using our system is to determine which synthesizer we want to assign to our system. Controls can be implemented in synthesizers. This possibility can give us solutions to build a previously chosen model. For example, a synthesizer can perform a glissando. In the example developed for this paper, we use a simple granular FM synthesis.

The final step in implementing our solution is to determine the type of relationship between the analysis data and the parameters of our synthesizer. How to get expressive sounds with sonification process [2] ? Should our relationship be static or dynamic? How should the plan of connections between these various elements be established? For the example developed in this paper, we have established a one-to-many and many-to-one static connection plan. The amplitude determined by the analysis is correlated with the granular density of the synthesis, the modulation index and the duration of the response. The pitch determined by the analysis is correlated with the pitch of our response. A transposition is performed by a scaling operation. Finally, the amplitude and the pitch analyzed serve to determine the modulation frequency of the FM synthesis of the response delivered by our system.

9 Conclusion

For this paper, we have implemented our open interactive system under *SuperCollider* - platform for audio synthesis and algorithmic composition. We could have implemented this system on other software. Moreover, use free software increases the durability of our work. Laurent Pottier recalls the history of the precariousness of technologies in electronic music [5] and free software answers to this problem.

An example concerns the portage of *Pluton* by Philippe Manoury from the *4X*[9] to *Max* [6]. The piece did not really sound exactly the same way on both platforms. After a thorough study of the *4X*, the engineers discovered that a *4X* hardware limitation influenced the sound result. This limitation was implemented in the *Max* patch to find an electronic music equivalent [7].

[9] *4X* is real time effect processors designed by Giuseppe di Guigno at IRCAM in the 1970s.

Free software is an important factor of durability and reproducibility in the digital art. The ubiquity [8] of free software allows more flexibility to imagine original devices [9]. In the end, researchers have no lock to study and increase the common good.

10 Acknowledgements

Thanks to Laurent Pottier for his encouragement and his pertinent advices.

References

[1] R. Rowe. 1993. *Interactive Music Systems: Machine Listening And Composing*. MIT Press, Cambridge.

[2] T. Hermann, A. Hunt, J. G. Neuhoff. 2011. *The Sonification Handbook*, Logos Verlag, Berlin.

[3] J.-C. Risset, S. Van Duyne. 1996. Real-Time Performance Interaction with a Computer-Controlled Acoustic Piano. In *Computer Music Journal 20/1*, pp. 62–75, MIT Press, Cambridge.

[4] B. Laurel. 2014, Second Edition. *Computers as Theatre*. Addison-Wesley, Crawfordsville.

[5] L. Pottier. 2015. L'évolution des outils technologiques pour la musique électronique, en rapport avec la pérennité des œuvres. Constat, propositions. A. Saemmer, editor, *E-Formes 3, Les frontières de l'œuvre numérique*, pp. 245-261, PUSE, Saint-Étienne.

[6] M. Puckette. 2002. Max at Seventeen. In *Computer Music Journal 26/4*, pp. 31-43, MIT Press, Cambridge.

[7] J. Szpirglas. 2012. *Composer, même avec trois bouts de ficelle… Entretien avec Philippe Manoury*. http://etincelle.ircam.fr/1077.10.html.

[8] S. Letz, S. Denoux, Y. Orlarey. 2014. Audio Rendering/Processing and Control Ubiquity? a Solution Built Using the Faust Dynamic Compiler and JACK/NetJack. In *Proceedings ICMC|SMC*, Athens.

[9] M. Lallement. 2015. *L'âge du faire. Hacking, travail, anarchie*. Le Seuil, Paris.