Implementing Arcade by Günter Steinke in Csound

Daria Cheikh-Sarraf, Marijana Janevska, Shadi Kassaee, and Philipp Henke ⋆

[1] Incontri - Institut for contemporary music at the HMTM Hannover
[2] FMSBW
incontri@hmtm-hannover.de

Abstract. This paper is about the process of implementing the live-electronics of the solo cello and electronics piece "Arcade" by Günter Steinke. We will discuss problems that occurred during the process of implementation and how we approached the transfer of the electronic procedures that were originally on big hardware machines to the Csound programming environment. The main focus of this paper also discusses the possibilities of the Csound FrontEnd *CsoundQt*, that we mainly used for the performance with its GUI capabilities.

Keywords: CsoundQt, Live-electronic, Instrument, Hannover, Incontri, FMSBW, Günter Steinke, Arcade

1 Introduction

Back in the time of 1992, the German composer Günter Steinke begun work on a piece which was to become his cello and live-electronics piece. He couldn't have known that much of the equipment he was using in the Freiburger Experimentalstudio would soon become obsolete. As time passed, the computer became increasingly accessible, convenient, and powerful. A piece which would have required truckloads of equipment before could now be realized in a small machine, namely the *notebook.* Powerful programming languages like Csound made it possible to realize the needs of a piece like Arcade and to create a sophisticated version which is purely software-based. In this paper, we will describe the process it took to realize a complex piece like Arcade in Csound and how we dealt with other implementations of the piece in other music-programming languages like Pure Data and Max/MSP.

2 Speakers and Microfon

In Günter Steinke's Arcade, the cello, played live, should be amplified in addition to the electronics. To achieve and maintain a good balance in the overall volume, we used two microphones for the performance at the Sprengel Museum

⋆ Without the help of Joachim Heintz this would not have been possible.

in Hannover. Firstly, the Shure microphone: the advantages of this microphone is its consistent cardioid characteristic, which should reduce feedback, and an optimum transmission range for drums, percussion and instruments, which is good for the pizzicati in the cello part. However, due to the weight of the coil, the Shure microphone sometimes sounds sluggish, especially the high frequency response, which may sound a bit covered. And secondly, the DPA microphone: the DPA microphone is well suited for instruments. It is clear, clean, and has a high-resolution. In this piece, the cello has a wide dynamic range. For example, there are very quiet passages (see min. 09:00) of sul ponticello or pianississimo, but also loud, dominant pizzicato parts (see min. 05:02), which sound very percussive and present a strong contrast. It was often necessary to emphasize the above-mentioned passages manually by amplifying the input of the cello, i.e. how much came into the microphone or the two microphones. For the pizzicato parts, we had to amplify especially the Shure microphone precisely for the percussive parts. However, this could have been automated in order to avoid supposedly minor errors.

3 Electronics and Problems of Implementation

Steinke's Arcade uses a wide array of different electronic procedures. He uses different modules like, pitch-shifting, Halafon, delay-lines, noch weitere hinzufügen!!! Since 1992, Arcade has been translated for the computer. The first computer realization was made with the programming language Max/MSP in 2000. Another significant implementation has been made for the Pure Data programming environment by Orm Finnendahl, which was realistically, a translation of the Max/MSP implementation. Analyzing all the implementations from the past, there is no denial that each of the implementations had to deal with problems of translation from hardware to software. One of the first tasks was to analyze the possibilities of the Csound programming language, in order to avoid creating a PD version in Csound, but rather a native Csound implementation using the power of the Csound programming language.

3.1 Analysis

One of the first pitfalls to avoid when confronting an implementation is to resist direct translation e.g. between PD-Objects and their Csound equivalents. Oftentimes, one finds a similar opcode of the same name in Csound. However, it should be noted that one has to first look at the specific functionalities of the opcode, like the quality of the filter, and the order of the bandpass filter used. While analyzing the Max/MSP and PD versions, one notices that both patches do not actually use a filterbank like in the original realisation of *Arcade*, they used stations of spectral masks done with fft, to create a similar sound to the original filterbaks. However, this would contradict our approach and goal of a Csound native implementation which is true to the orginal realisation, proposed in Steinke's score. Consequently, we sought solutions dealing with the wide array

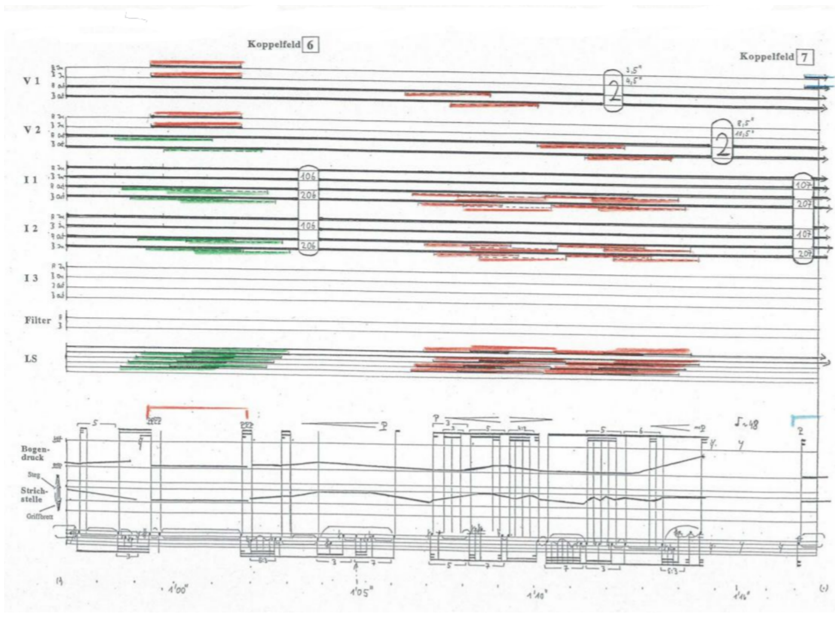


Fig. 1. Example of the notation in Steinke's *Arcade*

of filters that the Csound programming environment has to offer. Visually, the biggest difference one finds when working on an implementation is that Steinke used an analog matrix (*Koppelfeld* during the premiere of the piece. Because the matrix is so essential to the functionality of the piece, Max/MSP and PD[1] come with their respective matrix applications, whereas in Csound, a text-based programming environment, one has to build the matrix to work while also using the GUI possibilities of CsoundQt to make it more useable in the performance situation.

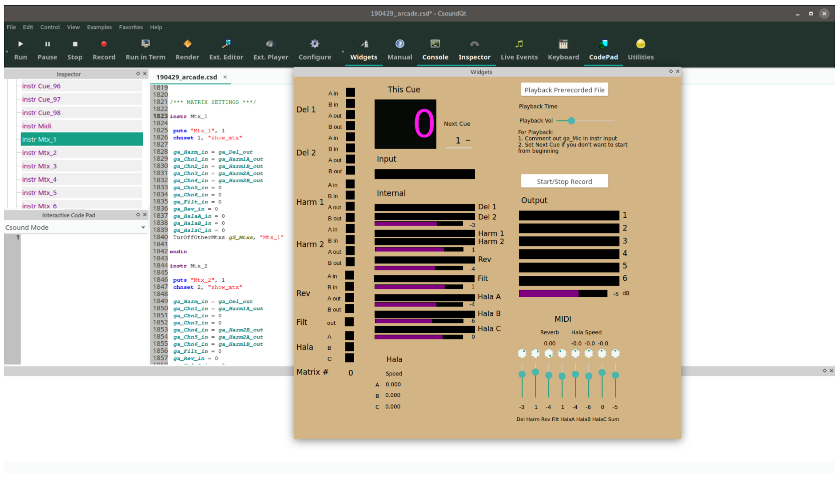
Example of the Matrix in Csound

```
/*** MATRIX SETTINGS ***/

instr Mtx_1

 puts "Mtx_1", 1
 chnset 1, "show_mtx"

 ga_Harm_in = ga_Del_out
 ga_Chn1_in = ga_Harm1A_out
 ga_Chn2_in = ga_Harm1B_out
 ga_Chn3_in = ga_Harm2A_out
 ga_Chn4_in = ga_Harm2B_out
 ga_Chn5_in = 0
 ga_Chn6_in = 0
 ga_Filt_in = 0
 ga_Rev_in = 0
 ga_HalaA_in = 0
 ga_HalaB_in = 0
 ga_HalaC_in = 0
 TurOffOtherMtxs gS_Mtxs, "Mtx_1"

endin

instr Mtx_2

 puts "Mtx_2", 1
 chnset 2, "show_mtx"

 ga_Harm_in = ga_Del_out
 ga_Chn1_in = ga_Harm1A_out
 ga_Chn2_in = 0
 ga_Chn3_in = 0
 ga_Chn4_in = ga_Harm2B_out
 ga_Chn5_in = ga_Harm2A_out
 ga_Chn6_in = ga_Harm1B_out
 ga_Filt_in = 0
 ga_Rev_in = 0
 ga_HalaA_in = 0
 ga_HalaB_in = 0
 ga_HalaC_in = 0
 TurOffOtherMtxs gS_Mtxs, "Mtx_2"

endin
```

Example from the Csound implementation of *Arcade* by Günter Steinke. In creating a hybrid gui application handling the matrix fucntion for us, we found a convinient way to solve the problems concerning the realisation of Steinke's analog *Koppelfeld.*

3.2 Filters

An important aspect of our implementation is that we did not use fft to recreate a sound emulation sounds of the premiere, but instead implemented Csound native filters to patch the piece. It has been the first realization since the premiere that uses true filter processing instead of spectral masks done in former implementations. In the process of programming the filters, we made a long process testing out the different filter opcodes in Csound. The filters are a crucial part of



```
/*** MATRIX SETTINGS ***/

instr Mtx_1

 puts "Mtx_1", 1
 chnset 1, "show_mtx"

 ga_Harm_in = ga_Del_out
 ga_Chn1_in = ga_Harm1A_out
 ga_Chn2_in = ga_Harm1B_out
 ga_Chn3_in = ga_Harm2A_out
 ga_Chn4_in = ga_Harm2B_out
 ga_Chn5_in = 0
 ga_Chn6_in = 0
 ga_Filt_in = 0
 ga_Rev_in = 0
 ga_HalaA_in = 0
 ga_HalaB_in = 0
 ga_HalaC_in = 0
 TurOffOtherMtxs gS_Mtxs, "Mtx_1"

endin

instr Mtx_2

 puts "Mtx_2", 1
 chnset 2, "show_mtx"

 ga_Harm_in = ga_Del_out
 ga_Chn1_in = ga_Harm1A_out
 ga_Chn2_in = 0
 ga_Chn3_in = 0
 ga_Chn4_in = ga_Harm2B_out
 ga_Chn5_in = ga_Harm2A_out
 ga_Chn6_in = ga_Harm1B_out
 ga_Filt_in = 0
 ga_Rev_in = 0
```

For Playback:
1. Comment out ga_Mic in instr Input
2. Set Next Cue if you don't want to start from beginning

Fig. 2. Example of the Widget view of our implementation (PNG).

the piece, in particular the way the piece sounds, for that we had to understand after what sound the composer is after. We decided to choose the *mode* filter opcode, because it could produce a very transparent and resonant sound combined with the cello. However there was an argument concerning the stability of the opcode and the advantages of using the *reson* filter over the *mode* filter.
Implementations of the original filter modules

```
/*** FILTER ***/

instr Filt_Seq_1

 kndx init 0
 kTime init 0
 kFiltSeq[] = gk_Filt_Seq_1
 iFirstProg = 1
 if kTime <= 0 then
  event "i", "ReadFiltProg", 0, 0, iFirstProg+kndx
  kTime = kFiltSeq[kndx]
  kndx += 1
  if kndx == lenarray(kFiltSeq) then
   printks "  Filt_Seq_1 turned off\n", 0
   turnoff
  endif
 endif
 kTime -= 1/kr
endin
```

```
instr Filt_A

 iBand = p4
 S_chnl sprintf "Filt_A_%d", iBand

 //midi pitch one tone below the first band
 iBasPch = 34
 iQ = 1

 iFreq mtof iBasPch + iBand*2
 kDb chnget S_chnl
 kDb port kDb, gi_Filt_FadeTim
 aFilt mode ga_Filt_in*ampdb(kDb), iFreq, iQ

 chnmix aFilt, "filt_A_collect"

endin
```

Example from the Csound implementation of *Arcade* by Günter Steinke.

4 Performenace Situation

In the case of Steinke's Arcade, not only the electronics and amplification played an important role. In the original score, the "programs" indicate which effects are triggered and which cello parts are recorded and edited with filters, delays etc. In our Csound version, the so-called "cues" always have sections that have been recorded through the microphone, which can be activated and stopped, effects being played on them and previously recorded patterns repeated. The difficulty was to activate the cues at the right moment. In certain places, for example where a delayline of the cello should be played back through the speakers which then would occur at the same time with the live cello, one has to be as precise as possible. No sounds or noises appearing too early or too late should be allowed into the triggered cue, since they could partly pull through the delays and through the whole piece, which would be a major disruptive factor. It is even more important not to leave everything to technology and to operate the cues and mixers by ourselves, as any performance of the live cello could vary in speed. It is a great help at particularly critical points in the play to agree with the player on assignments, so as to adapt the cues to the live cello as precisely as possible. Since this piece, and this is what makes it so special, even one wrongly timed cue can be heard as an error in the process of the piece. That is why one has to be precise with the triggering of the programs/cues.

4.1 Summary

Csound together with it's frontend CsoundQt provide sophisticated means to implement complex sounds and structurs into a simple and easy to use performance enviroment. As a text-based programming enviroment, csound is also easy on the CPU and can handle difficult calculation tasks, like multiple filter layers and harmonizer layers as well as complex spatialisation. However where Csound shines the most is it's tonal flexibilities and wide array of opcodes that help to shape the sound in many different ways. The csound frontend *CsoundQt* proved to be very useful in the performance situation, concerning the capability to use the widget to control the parameters of the electronic in realtime in a convenient way.

References

1. Heintz, J. et McCurdy, I.: Csound Floss Manual. Creative Commons Attribution 2.5 (2015)
2. Lazzarini, V. et al.: Csound: A Sound and Music Computing System. Springer (2016)
3. Steinke, G.: Arcade für Solo Cello und Live-Elektronik. Boosey and Hawkes (1992)
4. Csound Github site, `http://csound.github.io`