stateX5.5
concept - composition - technical realisation

Georg Holzmann

Nov. 2004, Graz

stateX5.5 is a composition for baritone sax, cello, double bass, percussion, laptop, a conductor, 12 speakers spread over the room, live-electronic and an audience. I made stateX5.5 in summer 2004 within the project *Morgensterne* of the austrian newspaper *Kleine Zeitung*.
This document describes the concept and realisation of the composition.

Contents

1 concept of stateX5.5

1.1 speaker arrangement and the audience

stateX5.5 is something between a sound-installation and a composition.
The speakers are positioned directly in the audience on the floor, so that the listeners can walk through this array of speakers and can hear the composition from different points during the performance (see figure 1).
So the location of the listener is important: if you stand in front of a loudspeaker, a lot of

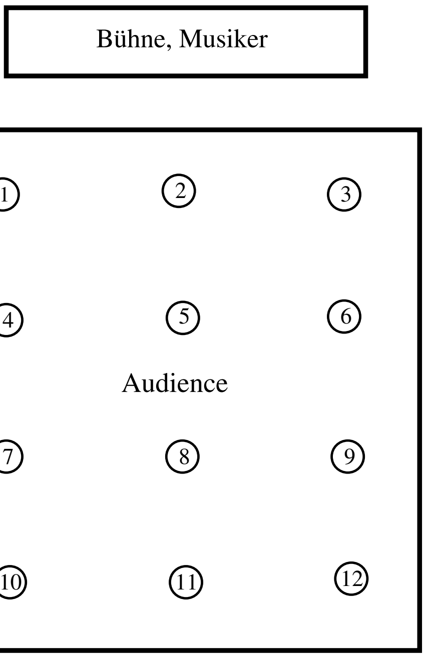


Figure 1: speaker arrangement

other sounds may be masked.
Because of this there sounds a different composition on every location.

1.2 states and sounds

1.2.1 a sound - a state

In general one sound, from an instrument or from the computer, sounds on one speaker in the room. This is a state - the sound on a specific speaker.
One sound changes its state after a certain time. This means it goes to an other (a next) speaker and changes its parameters a little bit, but the new sound is still similar to the previous one.
Such states can also "propagate", so that out of one previous state can become two or more different ones, which are all similar, but have different positions in the room (speakers).

1.3 the 5 parts (”big-states”)

The whole composition consists of 5 parts. Every ”big-state” lasts between 4 and 6 minutes and between those there is silence for 40 to 60 seconds.
If one part changes to the next, the overall musical parameters for the next part are also changed a little bit (similar to the concept of the sound-states).
In every part the sound-states have different prefered motions how they change the speakers, so you can perceive different movements in the whole room.
Sometimes you can also hear sound-states from the past, or you can hear one sound-state on more speakers at the same time.

1.4 instruments

In stateX5.5 there are 5 instruments (baritone sax, cello, double bass, percussion, laptop) and a conductor on the stage. The laptop and the ”natural” instruments are treated as equivalent sound-producers. The conductor ”only” shows cues.

1.4.1 categorization of the sounds

In general there exists 3 main categories of sounds (with a number of sub-categories): very low sounds, noisy sounds and very high sounds.
So I had to choose instruments, which can produce sounds from all categories.
Also the instruments are prepared (see figure 2) and played with non-classical playing techniques, so that maybe you can hear sometimes the sounds without identifying the instrument at once.

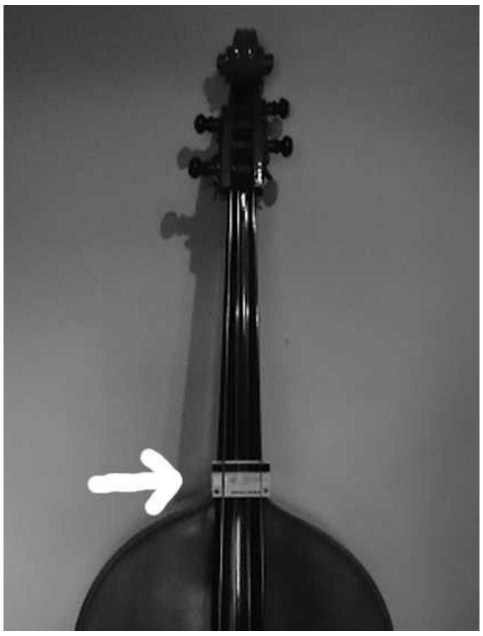


Figure 2: preparation of the double bass with a credit card

1.4.2 realisation of the sound-states

For every sound-state an instrument has to play a specific playing technique with a specific rhythmic model and in a constant dynamic (because the sounds are recorded and processed with the computer).
The borders of this rhythmic model are notated, inbetween these borders the instruments can improvise (see figure 3).

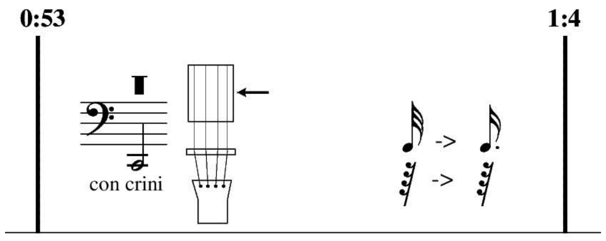


Figure 3: notation of one sound-state (cello)

1.5 live electronic

All sounds of the instruments are recorded and sent to the live electronic computer.
In that computer the sounds are recorded, processed etc.
Also the whole spatialisation is made with that computer, so that the sounds are now thrown into the audience.

2 technical realisation

2.1 generation of the score

The generation of the score can be splited in 3 parts:

1. all the data for the score is generated in PD (Pure Data - see [3])
2. a parser (written in C++) generates Lisp-code for CMN (Common Music Notation - see [4]) out of the PD-data
3. CMN generates the scores for all instruments out of the Lisp-code

2.1.1 generating the data in PD

The concept I described in section 1 is implemented in PD (see figure 4).
Therefore I wrote the external-library *PDContainer* (see [2]) for PD, which is an implementation of the C++ STL containers. This library should allow to make complex algorithms in PD in an easy way.

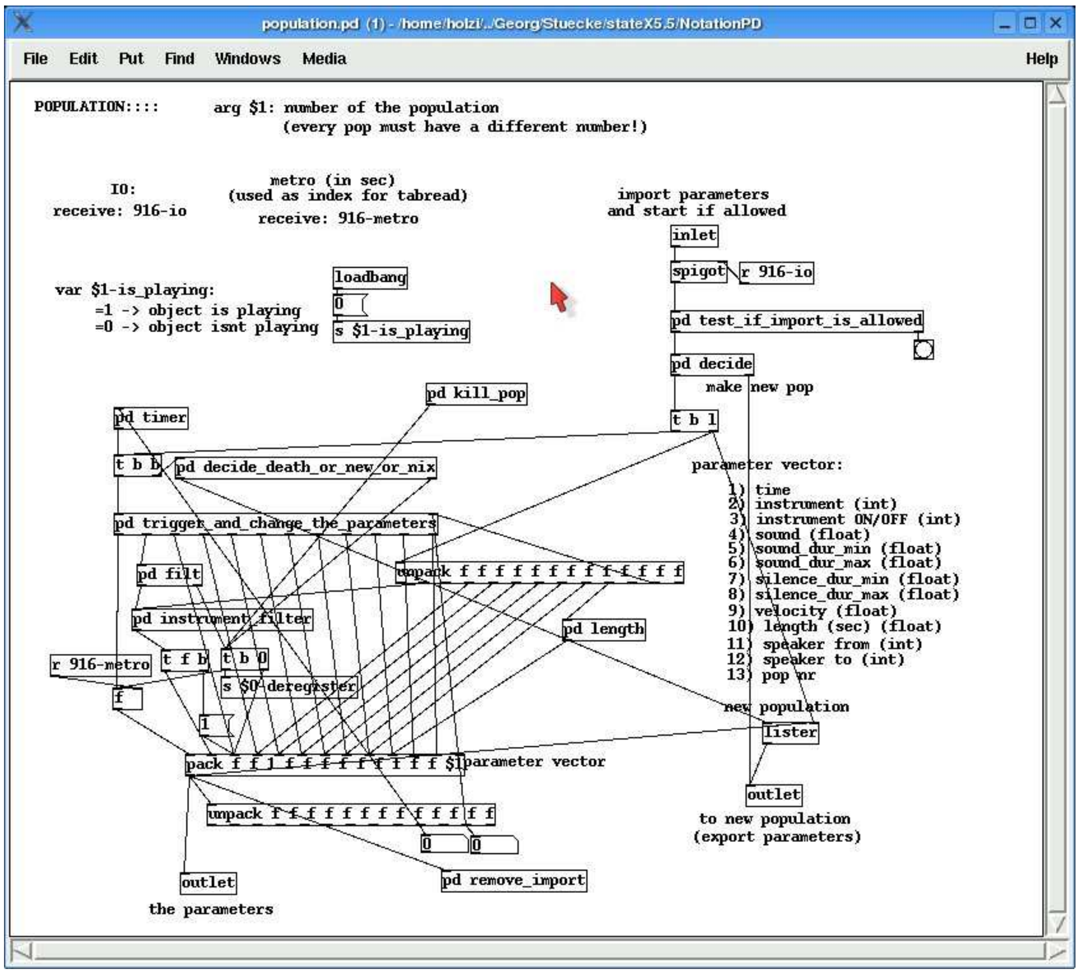


Figure 4: example of the PD implementation

53	2	1	0.689987	143.864	358.328	54.0597	59.9289	0.84633	10.8316	2	;
57	4	1	0.827693	439.807	607.406	324.172	163.567	0.744518	9.06207	4	;
64	2	0	1.63941	442.214	339.019	54.0597	59.9289	0.84633	10.8316	2	;
66	5	1	0.689987	197.91	164.868	223.343	187.884	0.720786	17.8331	3	;

Table 1: example parameter list

The ouput of PD is a list of all the needed data written to a text file (see table 1). Every line contains one sound event, which is equivalent to one sound-state.
The parameters in these events are:
start time - instrument nr - instrument ON/OFF - sound - sound duration min - sound duration max - silence duration min - silence duration max - volume - duration - speaker nr

2.1.2 parsing of the data

The parser reads the output file of PD and saves all the data in the C++ struct Event (see table 2).

```
struct Event
{
  int time;            // start time
  bool instrIO;        // instrument ON/OFF
  int instrument;      // instrument nr
  float sound;         // sound
  float sodur_min;     // sound duration min
  float sodur_max;     // sound duration max
  float sidur_min;     // silence duration min
  float sidur_max;     // silence duration max
  float volume;        // volume
};
```

Table 2: C++ struct of the event (parameter *speaker nr* is not necessary for the score)

Out of this array the Lisp-code is generated for all the instruments.

2.1.3 generating the score with CMN

Finally CMN makes out of the Lisp-code the score.
I used the clisp interpreter on a linux system.
Table 3 shows a CMN-code sample, which is generated out of the first event from table 1 and produces figure 3.

2.2 laptop, live-electronic

The laptop is a "sound-generator" like a normal instrument and is also on the stage. The sounds are produced with different samples and filters (see figure 5).

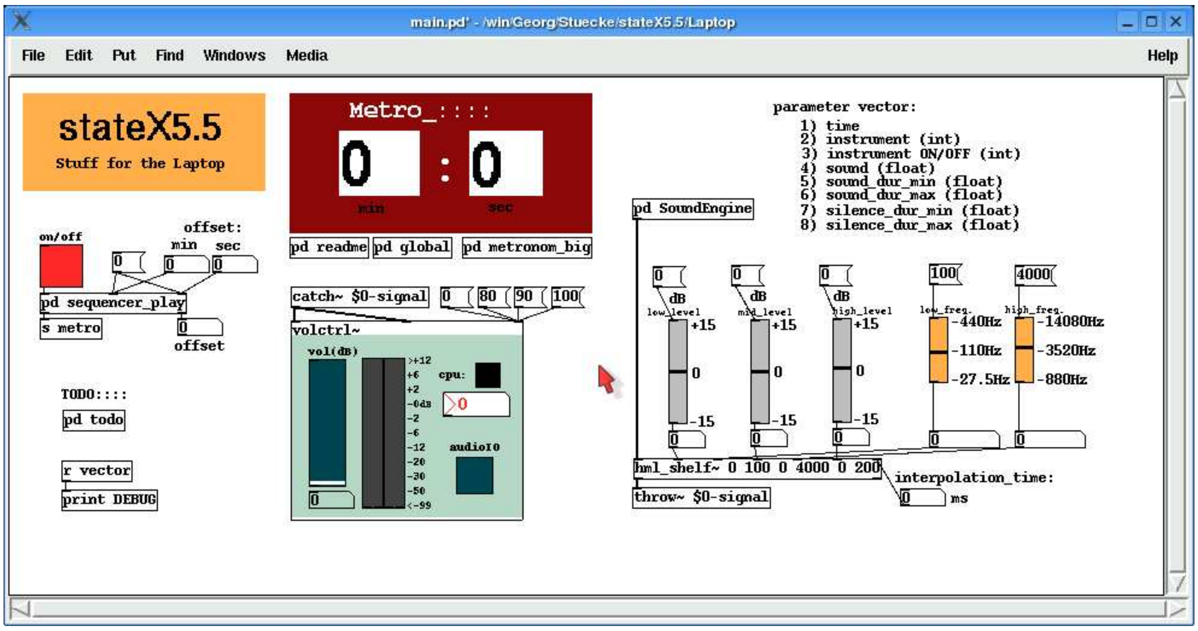
All the sounds are finally processed with the live-electronic computer (see section 1.5).
Software for the live-electronic and laptop is also implemented in PD on a linux system.
For more information regarding laptop/live-electronic you can download the PD-patches on [1].

2.3 other stuff

duration: ca. 30 min.
scores, record and PD-patch: `http://grh.mur.at/projects/statex55.html`

```
(bar (dy -5.5) (height 5.5)
(rehearsal-letter "0:53" (dx -0.5) (dy 5) (scale 0.7 0.7) ) )
(dashed-bar (dy -5.5) (height 11) )
(bar (dy -5.5) (height 5.5)
(rehearsal-letter "0:53" (dx -0.5) (dy 5) (scale 0.7 0.7) ) )
(dashed-bar (dy -5.5) (height 11) )
(c4 h (onset 73 ) no-stem (note-head-size 0)
(ff (dy -2.5) (dx 0.3) (scale 1.5 1.5) )
(x0 1.2))
(c4 h (onset 74) no-stem (note-head-size 0)
(graphics (file "SoundSymbols/CelloLow2.eps") (dy -5.5) (dx -1)
(scale .35 .35) ) (x0 8) )
(d4 32nd (onset 75) stem-up (text "->" (font-size 8) (dy 0)
(dx 0.9)) (dy -3) (x0 0.7))
(sixty-fourth-rest (onset 75) (text "->" (font-size 8) (dy -4.7)
(dx 0.9)) (dy -4.5))
(d4 s. (onset 76) stem-up (dy -3) (x0 0.5))
(sixty-fourth-rest (onset 76) (dy -4.5))
```

Table 3: CMN Lisp-code from the first event of table 1, which produces the score shown in figure 3



```
parameter vector:
  1) time
  2) instrument (int)
  3) instrument ON/OFF (int)
  4) sound (float)
  5) sound_dur_min (float)
  6) sound_dur_max (float)
  7) silence_dur_min (float)
  8) silence_dur_max (float)
```

Figure 5: PD interface for the laptop

3 performers

These are the performers, who also helped me a lot in figuring out all the different playing techniques:

- Bernadette Köbele, cello
- Franz Hofferer, percussion
- Tamas Schultz, double bass
- Florian Gessler, baritone sax
- Georg Holzmann, laptop
- Peter Plessas, live-electronic
- Edo Micic, conductor

References

[1] Georg Holzmann. my webpage. you can download the software, the score, records, etc. available online under http://grh.mur.at.

[2] Georg Holzmann. Pd external library for algorithms, 2004.
available online under http://grh.mur.at/software/pdcontainer.html.

[3] Miller Puckette et al. a graphical, open source programming language for audio/video processing.
available online under http://puredata.org, or http://www-crca.ucsd.edu/~msp/.

[4] Bill Schottstaedt. The ”common” family of sound and music software resides at the center for computer research in music and acoustics at stanford university (ccrma).
for more information see http://ccrma.stanford.edu/software/clm/.