Sparkler: An Audio-Driven Interactive Live Computer Performance for Symphony Orchestra

Tristan Jehan, Tod Machover, Mike Fabio
MIT Media Laboratory
email: {tristan, tod, revrev}@media.mit.edu

Abstract

This paper describes the design, implementation, and application of an audio-driven computer-based system for live and interactive performance with an orchestra. We begin by describing the compositional and musical goals, emphasizing the electronics aspect. We then discuss the challenges of working with an orchestra and integrating the electronics in a concert hall. We describe the hardware setup and then detail the software implementation, from audio analysis and "perceptual parameter" estimation to the generative algorithm.

1 Introduction

The orchestra is a musical institution that has the potential to take the lead in experimenting with the integration technology and new music. However, it has not been at the forefront of the computer-music community. We have therefore endeavored to develop a computer-based system that interacts with a full symphony orchestra in the context of a live performance.

This system was developed for Tod Machover's piece *Sparkler* for orchestra and interactive electronics (Machover 2001). *Sparkler* was premiered on October 14, 2001 by the American Composers Orchestra, conducted by Paul Lustig Dunkel in Carnegie Hall, New York City. It was commissioned by the American Composers Orchestra for its *Orchestra Tech* program, and was designed to be the opening work of a larger project called *Toy Symphony* that was premiered in Berlin on February 24, 2002 by the Deutsches Symphonie-Orchester, conducted by Kent Nagano.

2 Musical Goals

Sparkler was written to explore many different relationships between acoustic orchestral sound and electronic sound, sometimes contrasting the two, sometimes complementing them, and at other times blending the two into a new whole. Most previous work involved specially designed electronic instruments to complement the orchestra (Madden, Smith, Wright, and Wessel 2001), or musical events synchronized to a score follower (Puckette 1992) or solo instruments that were enhanced in some way (e.g. amplified or electronically processed). Our piece uses microphones to capture the audio of the entire orchestra which is then analyzed in real time and formulated into perceptual parameters through software. These instrumental sound masses — which are performed with a certain freedom by players and conductor — generate and control complex electronic extensions, turning the whole ensemble into a kind of "hyperorchestra" (Whiting 2002).

The musicians play their traditional acoustic instruments without any modifications or additions. There are only a few carefully placed microphones used to capture large sections of the orchestra sound. We generate the electronic sounds through flexible algorithms that take in streams of analyzed features from the audio and create complex sound textures that evolve over the course of the piece. At the climactic section, the orchestra shapes a kind of "texture blob" by bringing out different instrumental timbres and creating dramatic accents. To both the players and audience it is quite clear that the orchestra is directly controlling the electronics and is dramatically shaping this expressive enhancement of its own playing.

3 Challenges

There are practical, logistical, and technical challenges that come up when working with orchestras. Concert halls are not always set up with amplification and microphones, and it can be difficult to incorporate even the simplest piece of equipment on stage. Amplified synthetic sounds are not easy to mix with an orchestra. The acoustics of a concert hall and the dynamic range of an orchestra does not necessarily fit well with an electronic setup. It is definitely not easy to mike an orchestra and accomplish accurate instrumental group differentiation with the amount of reverberation present. Fortunately, we were able to experiment with several types of microphones and placements with the local MIT Symphony Orchestra in the early stages of our work. Rehearsal time was extremely limited, so reliability and flexibility was key. There were several parameters and ranges that could only be set during a

live rehearsal. Proper microphone calibration had to be very simple and could only be done with a fully silent house. In a concert situation, this could only be done with a mouse click during the fraction of a second that precedes the onset of the piece when both orchestra and audience are very quiet. Since the conductor needs to start anywhere in the piece in rehearsal, the software had to be flexible enough to jump to any section instantly. As we were dealing with dozens of musicians — often playing different notes — on each audio channel (three total), it was unrealistic to extract and use pitch. Spectral energy gave a better source of information. The real challenge was to measure the activity of the entire orchestra as an entity, not as a group of soloists.

4 Implementation

4.1 Hardware

We carefully placed six microphones on stage (see Figure 1) and mixed them down by pairs to only three audio channels in a Yamaha O3D digital mixing board. We paired a cardioid EV microphone and a small shot gun Shure microphone for each audio channel. We attempted to separate the left section (violins), middle section (woodwinds, percussions, and brass), and right section (violas, cellos, and basses) as well as possible.

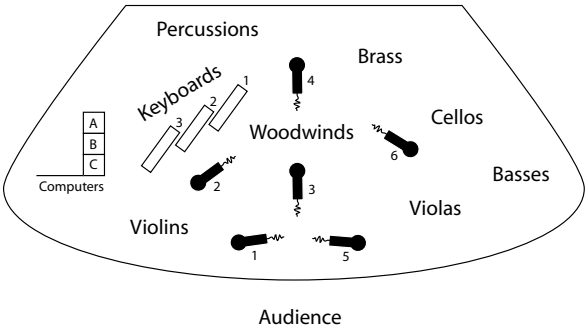

Figure 1: Microphone, keyboard and computer set up on stage. Microphone channels were mixed down to three audio input channels as follows: left = 1 and 2, middle = 3 and 4, right = 5 and 6.

In addition to real-time analysis and algorithmically generated music, the piece involved three MIDI keyboards (Yamaha P-80s). Two of them played commercially available MIDI synthesizers (two Yamaha FS1Rs and two Kurzweil K2500s extended with orchestra ROM). The voices often changed during the course of the piece thanks to the software. The third keyboard simultaneously sent instructions to the computer system (voice or channel numbers, modes and notes available to the generative algorithm) and played precomposed samples.

The whole piece ran on three locally networked Macintosh G4 computers (500 MHz with 512 Mb of RAM.) Although we could have used only two machines, we chose to separate computational tasks; the extra machine was being reused for another piece in the context of *Toy Symphony*. Each computer had its own role: computer 'A' was the *analysis machine* and exclusively dealt with sound analysis; computer 'B' was the *keyboard machine* which routed all the MIDI data and also played the precomposed samples of MIDI keyboard 3; and computer 'C' was the *synthesis machine* with the generative algorithm and software synthesis program. All computers used MOTU 2408 sound I/O hardware with the option of doing multichannel sound output. We decided to use the stereo house sound system since Carnegie Hall — like most concert halls — is not equipped with a surround-sound system. We routed all the audio channels directly to the house mixing board (a Yamaha PM1-D).

4.2 Software

Overview

The software was implemented in the Max/MSP environment (Zicarelli 1998), which allows for fast prototyping and experimenting with MIDI and audio. Each machine ran its own patch and communicated with one another through a local Ethernet network, using the freely available Max objects `otudp` and `OpenSoundControl` (OSC) (Wright and Freed 1997). For instance, the *analysis computer* sent a continuous stream of analyzed features (see Audio analysis) to the *synthesis computer*. Keyboard 3 both triggered mode changes sent from the *keyboard computer* to the *synthesis computer*, and sound samples. We used several samplers loaded with a total of 125 Mb of precomposed audio sequences in the software synthesis program Reason (Propellerhead 2000). Instructions also included changing voices on the commercial synthesizers and rerouting MIDI notes and sustain to different channels. The Max object `coll` was used extensively to store lists of program changes as well as octave transpositions or channel numbers. One keyboard could be playing up to four voices at a time.

Audio analysis

Three audio channels representing left, center, and right sections of the orchestra were simultaneously analyzed throughout the whole piece. At the heart of the analysis was the newly written MSP external `analyzer~`, previously described in (Jehan and Schoner 2001). `analyzer~` can estimate the following series of perceptual features: pitch, loudness, brightness, noisiness, onsets, and Bark scale decomposition (see Figure 2). Given the complex audio signal to be analyzed, we have not used the pitch extractor or the onset detector in this application. Each object calculates a real-FFT of approximately 24 ms of audio and

outputs an updated list of features every 12 ms. The object is capable of delaying the occurrence of its FFT so that when used in parallel, occurrences for each object can be unsynchronized to avoid overloading the CPU. We exploited this feature to compute the three simultaneous FFTs at 3 ms intervals (128 samples at 44.1 KHz.) The analysis application was measured to use about 15% of CPU load on a 500 MHz Macintosh G4.

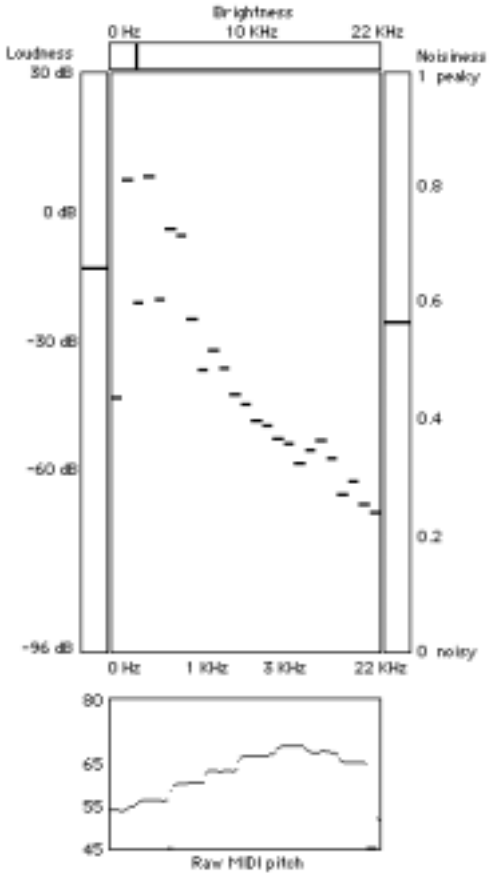

Figure 2: The `analyzer~` object in action. The top graphic shows a snapshot of instantaneous loudness, brightness, noisiness, and Bark spectral decomposition. The bottom window depicts the pitch curve of a singing voice.

More features were then derived from the raw analysis data. "Activity" was estimated from the loudness dynamics over a large period of time, i.e. the range between minimum and maximum loudness value over the last second. "Panning loudness" and "panning activity" — a sort of localization of center loudness and activity — were estimated by calculating the *centroid* (i.e. center of gravity) three loudness values and the three activity values respectively. The centroid was calculated by the equation

$$\text{centroid} = \frac{\sum_{k=1}^{N} k \cdot a_k}{\sum_{k=1}^{N} a_k} \qquad (1)$$

where a_k is the amplitude of the value of index k and N is the maximum number of values. In our case $N = 3$.

Since this equation returns a float value, it had to be rescaled to a MIDI compatible 7-bit integer (range from 0 to 127) just like all the rest of the analysis data. The raw data could at any moment in the performance be slightly rescaled through sliders so that it always ranged nicely from 0 to 127.

A specially written Max object called `smoother` was frequently used to smooth noisy data, making it perceptually more realistic. `smoother` features a median filter and a first-order low-pass filter and can process Max lists of integers and floats. The low-pass filter output is given by

$$Y_n = C \cdot X_n + (1 - C) \cdot Y_{n-1} \qquad (2)$$

where X_n is the current input, Y_{n-1} the previous output, and C the filter coefficient, with $0 < C < 1$.

Generative algorithm

During an important section of the piece, the conductor has more freedom in controlling certain aspects of timbre, by purposely emphasizing particular sections of the orchestra or groups of musicians. The tempo at this point in the piece becomes very free, and a single measure can last for a minute or so. The generative algorithm (Rowe 1992) does not rely on a particular beat or sequence of notes but is very flexible. It is designed to sound like a "cloud" of sounds or a "texture blob" that reacts to timbre changes and sound activity in an organic manner. Since the musicians were constrained by the available notes and rhythms that they could improvise with at any time, the "clouds" had to be constrained around those notes as well as pushed and pulled out of those limits when a particularly dramatic sound event occurred.

The texture blob section is composed of five different textures, each one being started and stopped from instructions sent by keyboard 3. Each texture is built out of six similar "core algorithms," each one generating notes on a given MIDI channel. The core algorithms take four arguments: the minimum and maximum length of notes (in ms) and the minimum and maximum speed rate of notes (in ms). It has 17 inputs, 12 being directly controllable by the result of the analysis with normalized values ranging from 0 to 127, i.e. volume, rhythmic speed, global pitch range, center note, probability, local pitch range, velocity range, velocity center, length range, length center, voice timbre, voice effect. The five remaining inputs are on/off, list of available notes, timbre controller number, effect controller number, and channel number. Inside the core algorithm, there exists several sub-algorithms that stochastically control the generation of pitches and rhythms (Winkler 1998), the note velocities and lengths, the timbre changes, and some additional effects.

Typically we worked on mapping the normalized result of the analysis to the different inputs of each core algorithm until we were satisfied with the musical effect and the reactivity to sound input. For example, brightness was intuitively mapped to center note. The higher the brightness value, the higher the pitch. Other intuitive mappings include the control of volume and rhythmic speed by analyzed loudness and activity respectively. We also mapped the activity parameter to the probability input. If the activity of the orchestra increases, the range of available notes used for synthesis widens. We sometimes associated one analysis channel to a pair of core algorithms, resulting in a specific part of the orchestra having a particular sound color.

Synthesis

All the MIDI data produced by the generative algorithm was sent locally to Reason through several IAC buses. The sounds were either created in a sampler (NN-19) or an analog synthesizer (Subtractor) for a total of 30 virtual devices. The sounds were diverse and contrasted, and ranged from percussive pitched instruments such as marimba or glockenspiel to much more synthetic and metallic analog types of sounds. Many sounds were created using the synthesis program Metasynth. Auxiliary effects (i.e. reverb, delay, chorus, phaser, equalizer) and individual volumes were carefully adjusted on each track. The precomposed sequences were made using several synthesizers, synthesis programs, and original cello, voice, or violin recordings.

5 Conclusions

We have described, conceptually and technically, the different electronic aspects of the piece including hardware setup, sound analysis, generative algorithm, and synthesis. In our final assessment, there are both successes and failures in our design.

Keyboard 3 used as a controller was a success since it allowed us to easily send lists of notes (i.e. chords or scales) to the generative algorithm without relying on pitch extraction and score following. It was important to be able to rescale the data in real time since the analysis for the most part relied on the acoustics of the performance space, the microphone setup, and the dynamics of the orchestra. The technical production was very well organized: setting up the hardware was very efficient. Neither the microphones nor the presence of three computers and racks on stage seemed to disturb the musicians. The collaboration between musically knowledgeable engineers and composer was a successful iterative process where the sound result was refined until both sides were musically and technically satisfied. The generative algorithm was very flexible, but additional techniques for creating smoother sound textures such as additive synthesis could have nicely complemented our system. Even though both conductor and orchestra could hear the electronics through the main house amplification and several additional directional amplified speakers on stage, they were not necessarily exploring all the possibilities offered by the system. Longer rehearsal time and formal tests would have probably helped.

Working with a full orchestra that controls a live computer-based performance system was a very special experience. It is our hope that orchestras in the future will become more involved with this kind of interactive technology.

Acknowledgements

Special thanks to Laird Nolan and Peter Colao for the production of the piece and to Ariane Martins for the management. Also thanks to the MIT Symphony Orchestra for letting us test with microphone placements. Thanks to Cati Vaucelle and Mary Farbood for proofreading this article. The worldwide project *Toy Symphony* was made possible by the Sega/CSK Corporation.

References

Jehan, T. and B. Schoner (2001). An audio-driven perceptually meaningful timbre synthesizer. In *Proceedings International Computer Music Conference*, La Habana, Cuba, pp. 381–388.

Machover, T. (2001). *Sparkler - musical score*. New York: Boosey and Hawkes.

Madden, T., R. B. Smith, M. Wright, and D. Wessel (2001). Preparation for interactive live computer performance in collaboration with a symphony orchestra. In *Proceedings International Computer Music Conference*, La Habana, Cuba, pp. 310–313.

Propellerhead (2000). `http://www.propellerheads.se/`.

Puckette, M. (1992). Score following in practice. In *Proceedings International Computer Music Conference*, San Francisco, pp. 182–185.

Rowe, R. (1992). *Interactive Music Systems*. MIT Press.

Whiting, M. (2002). Getting to know you: can orchestras and technology have a happy marriage? Tod Machover says yes. *Symphony Magazine (January-February)*, 15–22.

Winkler, T. (1998). *Composing Interactive Music: Techniques and Ideas Using Max*. MIT press.

Wright, M. and A. Freed (1997). OpenSound control: A new protocol for communicating with sound synthesizers. In *Proceedings International Computer Music Conference*, Thessaloniki, Greece, pp. 101–104.

Zicarelli, D. (1998). An extensible real-time signal processing environment for Max. In *Proceedings International Computer Music Conference*, Ann Arbor, Michigan, pp. 463–466.