CHAPTER 8:
ANALYSIS OF TECHNOLOGY REQUIREMENTS: CORT LIPPE'S *MUSIC FOR CLARINET AND ISPW* (1992)

In 1996 I presented a master's degree recital in computer music at the Peabody Conservatory in Baltimore, Maryland. Cort Lippe's *Music for Clarinet and ISPW* was a central work on that program, which consisted of four interactive electroacoustic works for clarinet. Lippe provided the ISPW hardware and software, and he set up and monitored the system during the performance. My recital commentary, submitted in partial fulfillment of requirements for the master of music degree in computer music performance and concert production, described in detail the process of preparing for a performance using the original equipment.[80] This commentary did not, however, attempt describe the complex inner workings of the interactive system, which is the purpose of this chapter.

8.1 HISTORICAL BACKGROUND

Cort Lippe composed *Music for Clarinet and ISPW* at the Institut de Recherche et Coordination Acoustique/Musique (IRCAM) in Paris and at the Center for Computer Music and Computer Technology, Kunitachi College of Music in Tokyo, Japan, where it was commissioned.[81] Clarinetist Esther Lamnek gave the first performance in March

[80]David Brooke Wetzel, "Music, Sound, Noise, and Silence: Commentary on a Computer Music Recital," master's degree commentary (Baltimore: Peabody Institute of the Johns Hopkins University, 1999).

[81] Cort Lippe, *Music for Clarinet and ISPW,* manuscript score, 1992.

1992, and has since recorded it for the Centaur/CDCM label.[82]

The original version, for IRCAM Signal Processing Workstation (ISPW), required a NeXT computer (now obsolete) and special sound processing hardware (the ISPW sound card). A version of Miller Puckette's Max software for the NeXT operating system controlled the synthesis and signal processing functions of the ISPW.[83] At the time of its composition, the live audio signal processing featured in *Music for Clarinet and ISPW* was beyond the capacity of common desktop (Macintosh or Windows) computer systems. No commercially available effects processor offered the flexibility or computing power required for a realization of this piece.

In 1999 Lippe created a new implementation of the interactive electronics using Max/MSP for Macintosh G3 (or later) computers. In this version all real-time signal processing previously performed by the ISPW hardware is now handled by the main processor of a relatively inexpensive desktop or laptop computer. As of spring 2004 Max/MSP has been released for the Microsoft Windows computing platform. Therefore, Lippe's latest software version is at least theoretically cross-platform compatible with either Windows or Macintosh computers.

However, cross-platform compatibility of the software is not the same as universal access. Users of competing real-time audio processing systems such as SuperCollider or Kyma are still left out. Furthermore, the future prospects for this work

---

[82] James Dashow et al., "The Composer in the Computer Age—VII" CDCM Computer Music Series, Volume 24, Compact Disc, (Centaur Records, CRC 2310, 1999).

[83] Cort Lippe and Miller Puckette, "Musical performance using the IRCAM Workstation," *Proceedings of the International Computer Music Conference* (International Computer Music Association, 1991), 533

still depend on regular updates of the performance software. While the original version was limited to performers with access to an ISPW/NeXT system (and often the composer's direct participation), the latest version is now limited to users of Max/MSP. While this undoubtedly improves the prospects for diffusion of this work among performers, it does not guarantee its existence beyond the current realization. The following technical analysis of the performance system is intended to guide new implementations of *Music for Clarinet and ISPW* using alternate technology.

8.2 MUSICAL ROLE OF TECHNOLOGY IN LIPPE'S *MUSIC FOR CLARINET AND ISPW*

Lippe describes the relationship between the live clarinet and the interactive computer music system:

> All the sounds used in the electronic part come from the composed clarinet part, and are transformed by the computer in real time during the piece. Thus, the musical and sound material for the instrumental and electronic parts are one and the same. The instrument/machine relationship is neither a dialog nor a duo. Musically, the computer part is not separate from the clarinet part, but serves rather to "amplify" the clarinet in a multitude of dimensions and directions.[84]

Music for Clarinet and ISPW is an eighteen-minute *tour de force* for computer-enhanced clarinet. The sounds created by the computer system are strikingly unlike the sound of the acoustic clarinet, though they are generated from it. These effects are achieved by digital signal processing (DSP) routines that alter the source sound in complex ways, often bending the normal rules of acoustics by directly modifying the

[84] Cort Lippe, *Music for Clarinet and ISPW,* program notes accompanying the score, 1992

sound's harmonic components. The combined effect of Lippe's energetic (sometimes aggressive) clarinet score with the computer-generated sound is a breathtaking experience not normally achieved in solo literature for the clarinet.

While *Music for Clarinet and ISPW* is primarily a solo work for electronically "extended" clarinet, Lippe advocates an important performance role for a technical assistant. The technical assistant is required to monitor the system during performance, advance through system changes manually if necessary (as described in section 8.3.3 below), and most importantly, manage the sound system and the balance between the clarinet and the computer-generated sound. Lippe compares the role of the sound technician "to the work a conductor does for balance, color, etc."[85]

8.3 ANALYSIS OF TECHNOLOGY COMPONENTS

The computer music system required for *Music for Clarinet and ISPW* can be separated into two basic parts: a general-purpose interactive synthesis and signal processing instrument, and a set of sound sources and system instructions that are specific to the composition. Therefore, the same synthesis and DSP system could (theoretically) be adapted for use in other musical works. In fact, Lippe's *Music for Piano and Computer* is based on the same system, with a few modifications. While the computer part does indeed function as a sonic extension of the clarinet, its structure can also be seen as following the familiar model of "instrument" and "score."

[85] Cort Lippe, email to the author, June 24, 2004

Lippe's interactive synthesis and DSP system combines live sound and prerecorded audio samples to generate complex audio effects. Within this software "instrument," a series of discrete modules for sound processing or synthesis may be combined in any order for layered effects. This system is controlled by multiple parameter variables that can be changed in real time (i.e. while the system is still running, without the need to re-compile software code or restart the system). In performance, changes to these system parameters occur at numbered "event" points, and the complete sequence of parameter changes for each event is stored in a database, or event list, which functions as an electronic "score" for the computer instrument.

Lippe's piece is extremely complex, both theoretically and technically. The Max/MSP environment is a graphical programming language that makes interconnections between its parts explicit, and is therefore a tremendous aid in understanding the operation of the program. However, Lippe's software was written for performance functionality, and not as a theoretical model for analysis. Therefore it presents some formidable challenges to the researcher trying to describe what is going on "under the hood." The following is an attempt to thoroughly classify and explain the functions and control parameters of each component in Lippe's interactive system.

8.3.1 Sound System and Necessary Hardware

For all the complexity of the interactive software system, *Music for Clarinet and ISPW* requires a relatively simple hardware setup. A minimal performance system will include a microphone connected to the audio interface (analog to digital converter, or

ADC) of a computer powerful enough to execute the necessary real-time digital signal processing routines. Output from the computer's sound card (digital to analog converter or DAC) is routed to a stereo pair of loudspeakers. A simple control device, such as a MIDI foot pedal or the computer keyboard is needed to advance the system through a series of pre-programmed events. Lippe recommends additional microphones used solely for sound reinforcement (i.e. not routed to the computer for processing) and the application of a mild artificial reverb to the live clarinet in order to balance its sound with that of the computer system. This basic setup is shown in figure 8.1.

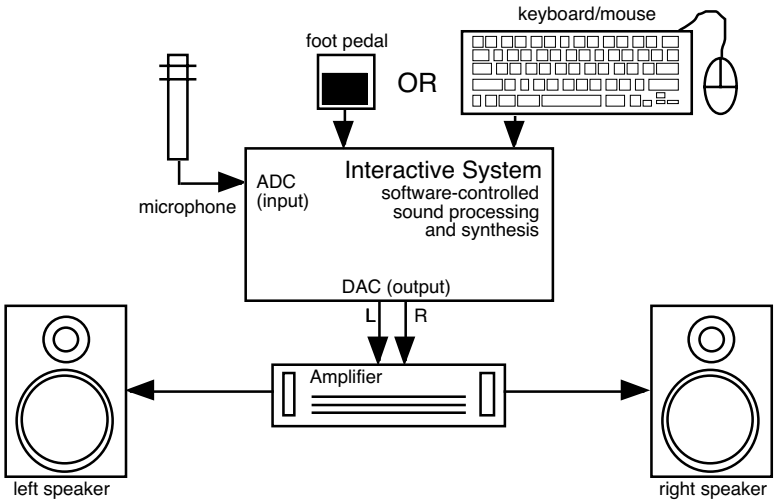

Figure 8.1 – Minimal sound system and control hardware

8.3.2 Sound Sources

Music for Clarinet and ISPW is based on the transformation of the clarinet sound. A series of pre-recorded clarinet samples, drawn from the score, provide material for complex audio effects and synthesis techniques. Live sound picked up by the

microphone is transformed in several ways, and also provides a source of control information for the interactive system.

Microphone input. The live clarinet sound is picked up by a microphone and processed for two distinct purposes. First, it is used as an audio source for signal processing modules that transform the live sound of the clarinet in various ways. Secondly, information about the clarinet sound (i.e., pitch and amplitude measurements attained through real-time analysis) is used to control various parameters of synthesis algorithms. The choice of microphone is left to the performer's discretion, taking into consideration the need for an acceptable signal for both pitch tracking and effects processing while avoiding picking up too much ambient sound.

Pre-recorded Samples. In addition to the live sound from the microphone, eight ten-second audio samples are used, each of which is a pre-recorded clarinet excerpt from the score. Lippe provides the necessary sound files with the current software realization. A clarinetist could also re-record them using score excerpts, if so inclined. The score excerpts for each of the pre-recorded samples are shown in figures 8.2 – 8.9.

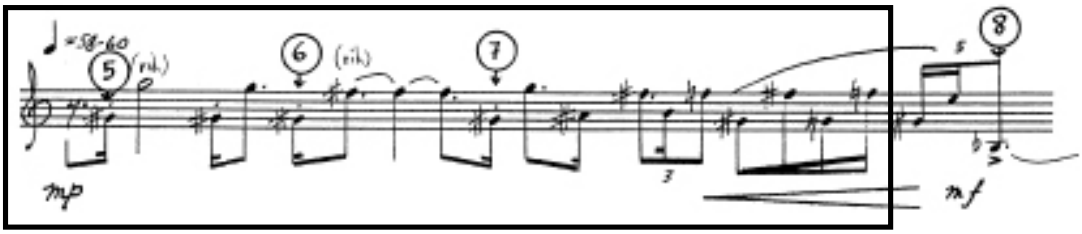

Figure 8.2. Score example: sample 1 (section I, events 5 – 7)

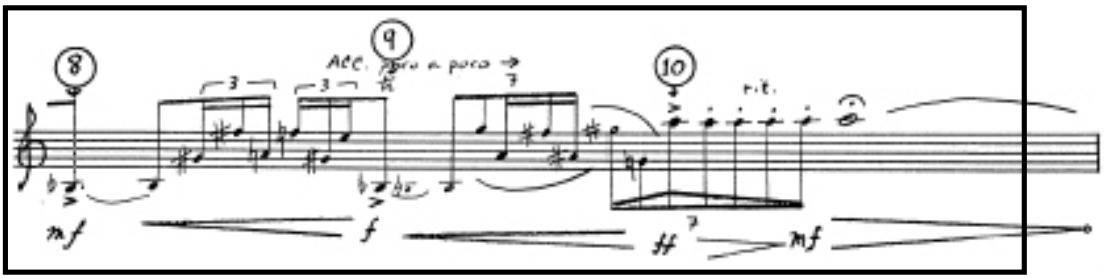

Figure 8.3. Score example: sample 2 (section I, events 8 – 10)

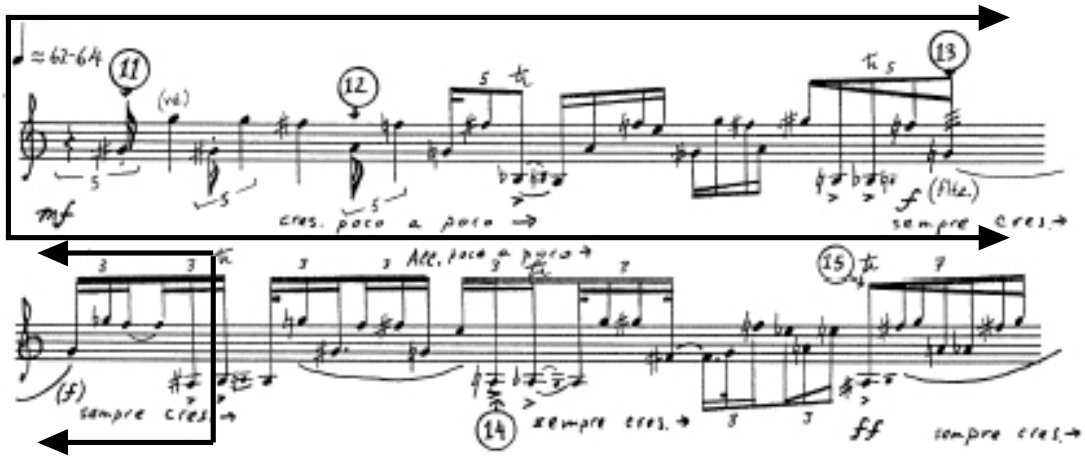

Figure 8.4. Score example: sample 3 (section I, events 11 – 13)

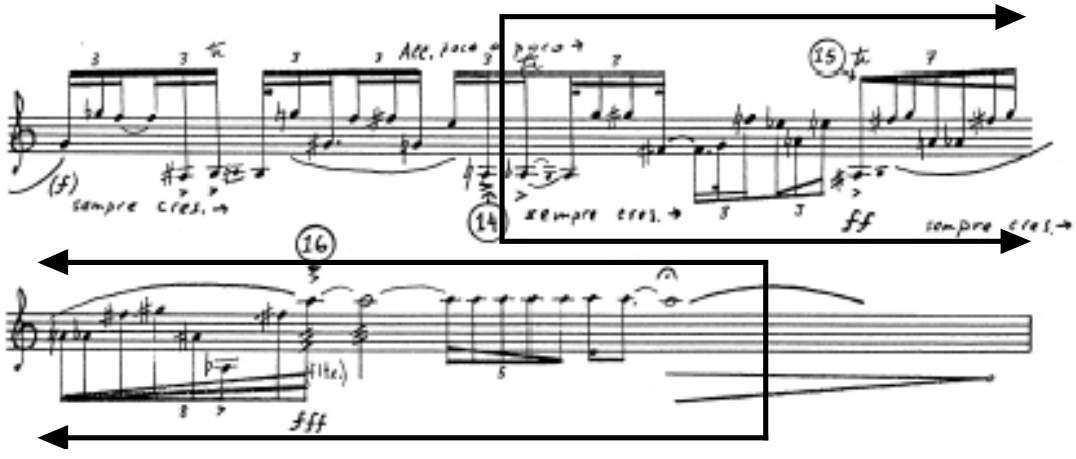

Figure 8.5. Score example: sample 4 (section I, events 14 – 16)

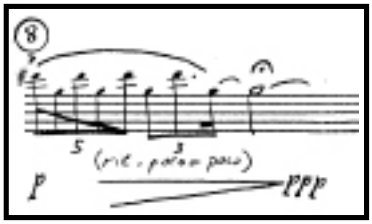

Figure 8.6. Score example: sample 5 (section II, event 8)

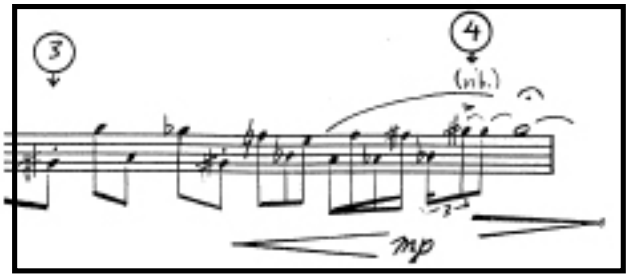

Figure 8.7. Score example: sample 6 (section I, events 3 – 4)

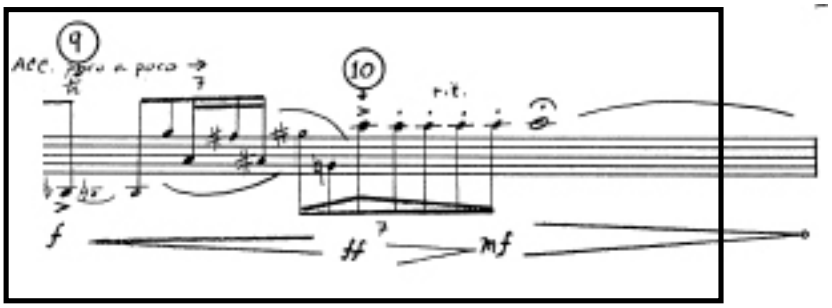

Figure 8.8. Score example: sample 7 (section I, events 9 – 10)

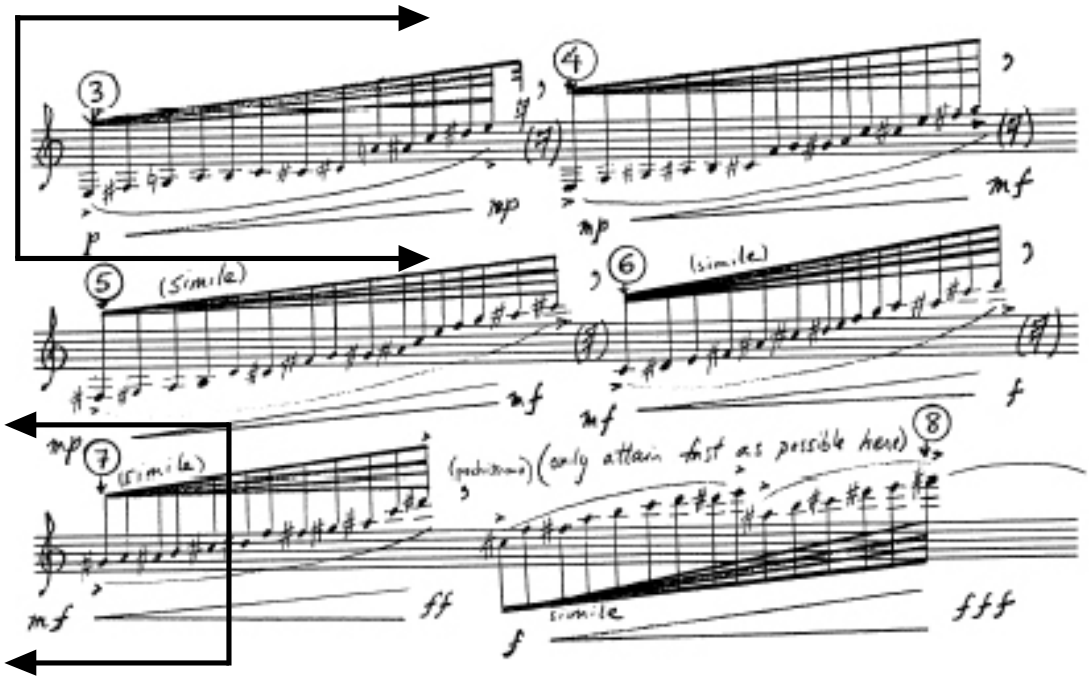

Figure 8.9. Score example: sample 8 (section V, events 3 – 7)

Each sample file must be precisely 10 seconds (10000 milliseconds) in duration, regardless of the actual length of the notated excerpt. In most cases the sample must be truncated to conform to the 10-second limit. Sample 5, which may be considerably shorter than ten seconds, should be filled out to 10 seconds with recorded silence.

8.3.3 Control Sources

The interactive system is controlled from several sources simultaneously, in the form of system parameters stored in score-related data files, real-time analysis of the clarinet pitch and amplitude, embedded algorithmic processing modules, the graphical user interface, and (optional) score-following algorithms.

Event list. The system is controlled by multiple variables, each of which can be changed individually during performance. Numbered cue points in the score refer to system events in which multiple system parameters may be changed according to a sequential list. Each line of text in the list serves as a command to the software system assigning a new value to a particular variable. Variable values in the event list may also trigger complex automated processes embedded in subprograms within the system software.

The event list is stored in an external database, and is loaded by the software into an event cue. Variables are set as a block unless preceded by an integer, which initiates a time delay in milliseconds for execution of the next event or block of events. As shown in figure 8.10, lines 1 – 7 are executed immediately. Lines 8 and 9 are executed after a 1 second (1000 millisecond) delay. Lines 10 – 12 are executed after a further one-second delay, and lines 13 – 15 one second after that. Each line contains one named variable followed by its new value or array of values.

```
ptof 0;
spatinc 4;
spaton 1;
ttoh 127;
htoh 107;
hdel 60;
ptoh 127;
1000 which2_table 3;
play-rand-pit2 7600;
1000 which2_table 2;
play-rand-pit2 4500, 8300 1000;
play-rand-pit2 8000;
1000 which2_table 4;
play-rand-pit2 8300, 5000 1000;
spatinc 55;
```

Figure 8.10. Event list excerpt: section III, event 14

Music for Clarinet and ISPW is divided into five sections, with approximately 10 to 30 events per section totaling some 2,225 lines of code in the event list. Therefore, I will not reproduce the entire event list here. Variable names used in the event list and in the system software are shorthand terms created by the composer specifically for this work. They are not in any way standard signal processing terms, and therefore, a key is required to understand the meaning of each variable and its intended effect on the relevant synthesis and signal processing routines. Each variable and its range of values will be explained below with the processing module in which it is used. A summary of all variables used in the piece is included in appendix D, and variable names given throughout this text will appear in italics (e.g. *hfreq*).

Many of the control variables found in the event list use value ranges based on the MIDI specification. Therefore, a number of pitch or amplitude parameters are expressed as MIDI note or velocity values between 0 and 127. These variable values are often converted within synthesis/DSP modules to frequency or amplitude values. In the case of MIDI pitch values, Lippe extends the standard MIDI note range to incorporate microtonal variations. These pitch variations are expressed in cents, appended to the MIDI note number by the operation: MIDI note number * 100 + microtonal variation in cents (0 – 99). For example, a pitch that is a quartertone above middle C (MIDI note 60) would be expressed as 6050. For the remainder of this paper, this system of pitch notation will be referred to as "MIDI+."

Pitch Tracking. The incoming signal from the clarinet microphone is continuously analyzed for pitch content. Two pitch variables are generated: *pitch-track-out* (as MIDI+) and *continuous_pitch* (as a frequency in Hertz). Lippe currently uses the Max/MSP *fiddle~* object (based on Fourier analysis) to track clarinet pitch. The goal is to reliably identify the pitches being played by the clarinetist for purposes of score following, and to provide continuous monitoring of the clarinet pitch for use as a control value in several signal-processing routines. Alternate strategies for pitch tracking would in no way alter the essential functionality of the computer system, as long as the requirements outlined above are met.

Envelope following. The envelope follower tracks the volume contour of the incoming clarinet signal from the microphone. The amplitude *envelope* of an audio signal is a chart of its signal strength (volume) from its initial attack through its sustain and eventual release. In one particular synthesis/processing effect (frequency/amplitude modulation) the continuous amplitude envelope of the clarinet signal is used to directly control signal gain of multiple sound-generating oscillators.

Automated processes. Several score events trigger manipulation of various sound-processing modules according to complex algorithmic processes. These control algorithms will be described in more detail in the contexts of the synthesis/DSP modules upon which they primarily act. Synthesis/DSP modules that are affected by algorithmic processing include the sampler, frequency shifter, harmonizer, and spatializer.

Graphical User Interface (GUI). The front panel of the current software implementation of *Music for Clarinet and ISPW* includes direct access to all of the variable parameters for the DSP modules and the audio signal routing network. Most of these controls are provided as a convenience for testing or for direct intervention during performance if necessary.

Most importantly for performance, the graphical user interface (GUI) allows an assistant to advance through the event list manually, by mouse or keypad actions. Additional inputs may be added for performer control of the system, using a foot pedal or similar triggering device.

The GUI also provides a convenient method for setting up the system and resetting it immediately prior to a performance. The details of the various initialization and safety functions incorporated into the GUI and control system are specific to the current software platform and implementation, and therefore are not essential to an understanding of the musical use of interactive electronics in this work.

Automated Score Following. The NeXT/ISPW version of *Music for Clarinet and ISPW* relied on score-following algorithms developed by Miller Puckette to advance through the event list in synchronization with the clarinetist's performance. The score following system was intended to avoid the need for system control by means of foot pedals or computer keypad control, since the computer would follow the performer and execute events in the list automatically at the appropriate points in the score. The computer system accomplished this task by comparing incoming note data from the pitch-tracking

module to a simplified electronic version of the score. Puckette and Lippe thoroughly describe the algorithms used for score following in a paper given at the 1992 International Computer Music Conference.[86]

Lippe strongly recommends against using the score-following feature in the current Max/MSP implementation, though it is fully functional. Instead, the composer advocates manual advancement of events during performance by a technical assistant. This arrangement adds an element of ensemble performance to what would otherwise be a completely solo work. The role of the technician in this regard is minimal, but it is critical to the success of the work since events must be advanced in time with the clarinet performance.

Whether the original ISPW score-following algorithm is used or a newly invented strategy is followed, the same basic principle should apply: the clarinetist's performance is compared to a stored version of the score, and events are cued according to the player's progress through the piece.

8.3.4 Synthesis and Signal Processing

Lippe's signal processing instrument is highly complex and generates a number of unusual audio effects. The algorithms that implement these effects are based on simple techniques and standard principles of digital signal processing, such as amplitude or frequency modulation, filtering, and delay lines. However, Lippe tends to layer and

---

[86] Miller Puckette and Cort Lippe, "Score Following in Practice" *Proceedings of the International Computer Music Conference* (San Jose, CA: International Computer Music Association, 1992) 182 – 185.

interconnect these techniques in ways that defy easy description. Therefore, full technical specifications for each of the required signal processing components are given in the form of block diagrams, found in appendix C. The following is a brief description of each component in terms of its basic audio processing technique, its general effect on an incoming audio signal, and the functions and parameters of its variable controls.

Sampler. The sampler plays back prerecorded audio from a set of stored sound files. It can play up to sixteen voices simultaneously (eight per channel) and allows for dynamic control of playback speed (affecting pitch) and direction, glissando (through accelerating or decelerating playback speed), amplitude envelope, and onset time within the file. The following table shows the variable parameters for controlling sample playback. Because multiple sampler "voices" can play at once, there are two sets of named variables for each parameter per output channel (four named variables for pitch, four for velocity, etc.; actual variable names are given in appendix C). Separately named variables can be changed individually, i.e., variables "pit1" and "pit2" would store pitch values for two separate notes that may sound simultaneously.

Table 8.1. Variable parameters for sampler playback

Parameter	Description
Pitch	*Transposition*. Middle C (60) = normal pitch. Transposition is expressed as MIDI note + cents (e.g. 6050 = quarter tone above middle C). Playback is initiated when a pitch value is received.
Velocity	*Amplitude scaling*. Sent as MIDI values (0-127). Playback volume = velocity * .01
Onset	*Start time*. Playback is offset from the start of the file in milliseconds. Also controls forward/backward playback
Attack	*Envelope attack*. Sets time in milliseconds for volume ramp from 0 to amplitude set by velocity.
Decay	*Envelope decay*. Sets time in milliseconds for volume ramp from playback amplitude to 0.
Gliss	*Pitch glissando*. Value sets glissando effect. Playback rate accelerated or decelerated to create pitch shift.
Gliss time	*Glissando time*. Time in milliseconds for gliss to last
Sample	*Sample table*. Number (1-8) of the sample to play

Granular Sampling. Three sub-routines control sample playback according to algorithmic processes, using a technique known as "granular sampling." The basic concept of granular sampling is described by the composer as:

> … the production of a multitude of short sounds (grains) consisting of a waveform, to which an amplitude envelope has been applied, at a specified frequency and amplitude. These grains of sound, produced at a high rate of speed, are usually overlapped with neighboring grains in order to produce a certain density and continuity of sound.[87]

Compositional variables used by Lippe in his granular sampling algorithms include the waveform (the specific recorded sound from which to sample), amplitude envelope, peak amplitude, pitch (controlled by playback speed of each individual grain),

[87] Cort Lippe, "Real-time Granular Sampling Using the IRCAM Signal Processing Workstation" *Contemporary Music Review* 10, no. 2 (1994), 149.

onset into the stored sample, grain duration, grain density, and grain overlap.[88] Sound file onset time for each grain is critically important to the sound produced, and grains may be played in sequential order corresponding to the normal linear progression of the sample's timeline, or they may be chosen in non-linear or even random sequences. Furthermore, onset times that follow the normal timeline need not correspond to the normal playback rate. Lippe uses the term "precession" rate to describe the sequence of onset times for each grain relative to the normal timeline of a sample. The precession rate therefore describes the rate of progress of granular playback through a stored sound file, which may be faster, slower, or the same rate as normal playback. It may also remain stationary, continuously sampling from the same point in the file. This technique allows both pitch and playback speed to be treated completely separately.

Granular sampling can be used for "time-stretching" a recorded sound in the following manner: 50 millisecond bursts of sound (grains) are played back from a stored sound file every 40 milliseconds (resulting in a 10 millisecond overlap between grains). If the onset time of each grain is progressively 10 milliseconds (the precession rate) further into the file from the start point, it will take four times longer to progress through the file than normal playback speed. Pitch is unaffected because each individual grain is played at the normal speed, but playback speed is radically altered.

Lippe controls granular sampling with three separate subroutines: *Trevor* (named for the British composer Trevor Wishart), *PLAY-RAND*, and *Trevor_back*. *Trevor* is a straightforward implementation of Lippe's granular sampling algorithm for time

[88] Ibid, 4

stretching. Playback may be initiated at any one of five points (0, 2, 4, 6, or 8 seconds into the sample table) by setting a trigger message for one of the *variables t-start/t-start1, t-start2, t-start3, t-start4, or t-start5*. Pitch transposition of the grains is set by *t-transpose* (MIDI + cents) and precession rate is set by *t-precess* (0 – 238, 127 = approximately normal playback speed). Grain duration is fixed at 50 milliseconds, and velocity (determining peak amplitude) is fixed at 75 (on a scale from 0 – 127). Grain onset (and therefore grain overlap as well) is not directly controlled by a system variable, but is instead controlled by a repeating trigger with its timer randomized from 10 to 19 milliseconds. Values generated by the granular synthesis engine are used to play short segments of audio from the stored sample tables by sending out values for the sampler variables described in Table 8.1 (above).

Trevor_back is essentially identical to *Trevor,* except that its precession moves backwards through the sample file. Start points are given at 2, 4, 6, 8, and 9.7 seconds into the file. Its variables are functionally the same as those of *Trevor* and are named in a similar manner. The full list of *Trevor_back* variables will be included in the complete table of system variables found in Appendix D.

The *PLAY_RAND* module accomplishes granular sampling in the same manner as the *Trevor* modules, with the exception that stochastic (random) processes control the parameters for sound-grain production. Two granular sampling units are placed in parallel. For each unit, grain pitch fluctuation, onset, duration, and overlap/grain density are controlled by a series of pseudo-random number generators. Pitch transposition, grain amplitude, grain envelope shape, and sample table number are set by event-list

variables. The first granular sampling unit has an additional control for producing randomized pitch glissandi within each grain. Full specifications for this sub-program will be included with the block diagrams in Appendix C.

Additional algorithmic processes are used to control the sampler at specific events in the score. In Section I, events 12 – 16, note values from the pitch tracker are used to control the *Trevor* and *Trevor-back* granular sampling modules. The algorithm, triggered in the event list by the variable *ichgate* (1 = on, 0 = off) is shown in table 8.2, below.

```
IF the value of ichgate is 1, THEN
  SET the value of tt02, sto2, tto4, and sto4 to 127; (full volume)
  IF the value of pitch-track-out is between 50 and 52,
     THEN {
     TRIGGER spatXY spatialization algorithm;
     SET b-precess value to [pitch-track-out * b-precess-val];
     SET b-transpose value to [pitch-track-out – 3];
     TRIGGER granular sampling module Trevor-back with onset value
     set by brevor-onset;
     }
  IF the value of pitch-track-out is between 76 and 79,
     THEN {
     SET t-precess value to [pitch-track-out * t-precess-val];
     SET t-transpose value to pitch-track-out;
     TRIGGER granular sampling module Trevor with onset value set
     by trevor-onset;
     }
```

Figure 8.11. Algorithmic control of granular sampling

Similar sampler control algorithms are used in Section I events 1, 5 – 6, 5 – 10, and 9, Section II events 11 and 26, Section III event 23, Section IV event 3, and section V events 3 – 6. These 8 additional algorithms will be fully explained in Appendix C.

Harmonizer. Pitch is shifted a fixed amount using a specialized implementation of varying-time delay. Variable parameters include delay time (*hdel,* in milliseconds), pitch

(*hfreq,* 0 – 127), modulation depth (*hwind1,* 0 –127), direct signal output amplitude (*diramp,* 0 –127), harmonizer output amplitude (*hamp,* 0 –127), and final output feedback to the harmonizer input (*htoh,* 0 –127).

Two delay lines are used in parallel, and the delay time for each is incremented according to a linear function from its start point (*hdel*) to an offset (*hwind1*) at a given speed (*hfreq*).[89] The delay-time increment ramps are phase-offset from each other by 180 degrees, so that as the first ramp reaches the offset value while the other is at the halfway point. The output of the two delays are then cross- faded at the rate set by *hfreq* to produce a smooth alternation between the two delay lines. Alternation between the two delay lines creates the illusion of a constantly increasing (or decreasing) delay time and therefore a constant and stable interval of pitch shifting.

Pitch shifting in this manner is not dependent on the initial delay time. The critical element is the increase or decrease in delay time at a constant linear rate. The continuously changing delay time shifts the pitch by effectively speeding up or slowing down playback of the delayed signal. Pitch offset from the original signal is a function of both speed (*hfreq*) and offset (*hwind1*) of the delay modulation. Increasing the delay time lowers the pitch, while decreasing the delay time raises the pitch.

Amplitude of the input signal and the output of the harmonizer are controlled separately (by *diramp* and *hamp* respectively) and combined on output, allowing for control of the mix between the original signal and its harmonization. The output signal

[89] Note that this type of delay modulation is used in Thea Musgrave's *Narcissus*, with variable parameters for delay time and speed/depth of time modulation.

may be fed back into the harmonizer input, set by the system variable *htoh*, (0 – 127), to shift the sound a second time by the amount specified by *hwind1* and *hfreq*.

Reverb. The input signal is fed into a series of fixed short delays to create artificial reverberation. Variable feedback of the reverberation creates the illusion of a larger or smaller space, depending on the amount of feedback (the amplitude of the output signal fed back into the reverb input). Input values for *Rgate*, *Revfb*, and *Rout* control input level, feedback level, and output level respectively. Values for each variable are in the range of 0 – 127. *Rgate* and *Rout* scale these values between 0 and 1 exponentially (as described in the equation included with the Reverb diagram in Appendix C). Values for *Revfb* are scaled linearly from 0 to 0.5 (values from 0 to 127 are divided by 254). "Infinite reverb" effects are achieved by closing the input (*Rgate* = 0), and setting feedback to 1.0 (*Revfb* = 254), trapping sound in the reverb system and looping it indefinitely.

Noise modulation. The input signal is fed through a series of short delays, each of which is amplitude modulated by a set of eight variable speed noise wavetable oscillators.[90] The input variable *fnois* controls the frequencies of the noise oscillators. Values for *fnois* (0 – 127) are scaled slightly differently for each oscillator, but fall within an approximate

[90] A wavetable is a series of digital audio sample values describing a single cycle of a waveform. A wavetable oscillator generates sound by storing the waveform in a buffer and looping its playback at a specified frequency. In Max/MSP, wavetable oscillators are most often implemented using the cycle~ object, which loads a 512-sample wavetable and takes variable arguments for frequency and phase offset. The default waveform for the cycle~ object is a simple cosine wave.

range of 0.016 – 29.4 Hz. At the lowest oscillator speeds, the effect is a slight fluctuation of the signal volume (a sort of random tremolo). At higher speeds, the signal begins to distort, and additional frequency components are generated (due to the side-band effects of amplitude modulation). At the upper extreme, the signal resembles white noise, but retains it amplitude contour (i.e., rhythms are recognizable, but the tone quality is not).

Frequency shifter. Individual harmonic components of the input signal are shifted up or down by a specified frequency interval. The frequency shifter takes variable arguments for frequency (*fsh01,* MIDI+ converted to frequency in Hertz) and amplitude (*famp01,* 0 – 127). Positive values for *fsh01* shift frequency components up, negative values shift them down. The second variable, *fsamp01*, controls the final output amplitude of the pitch shift module. Values from 0 to 127 are scaled from 0 – 1.

Frequency shifting in this manner radically changes the timbre of the sound since the shifted frequencies no longer bear the same harmonic relationship to the one another. For example, a signal with frequency components of 200, 400, and 600 Hz, shifted up 100 Hz, will create a tone with frequency components at 300, 500, and 700 Hz. In this example, the input signal consists of a tone and its first two harmonic overtones (defined as integer multiples of the fundamental) but the shifted signal contains frequencies that lie outside the harmonic series. The result is a radical "detuning" of the sound in which all its spectral components are shifted out of alignment. This effect is achieved using a form of amplitude modulation (AM) in which only one sideband component is used, and

the original signal is filtered out.[91] Complete details of the frequency shifter algorithm are found in the block diagram in Appendix C.

Frequency shift variable values are normally set directly by commands in the event list. However, in section I event 18 and section II event 1, frequency shift (and harmonizer) values are controlled algorithmically using live input from the pitch tracker. The event-list variable *fsgate18* turns the control algorithm on when its value is set to 1. An *fsgate18* value of 0 turns it off. The processing algorithm is as follows:

```
IF the value of fsgate18 is 1, THEN:
  IF pitch-track-out value is 77,
     THEN {
     SET fsh01 value to a random number from –2000 to –6499.
     }
  IF the value of pitch-track-out is between 50 and 59,
     THEN {
     SET hfreq value to a random number from 40 to 63;
     SET ptoh value to 122; (clarinet signal vol. to harmonizer)
     }
  ELSE, SET ptoh value to 0.
  IF pitch-track-out value is between 71 and fstend18 value,
     THEN {
     SET ptof value to 127; (clar. signal vol. to freq. shifter)
  ELSE, SET ptof value to 0.
```

Figure 8.12. Algorithmic control of harmonizer and frequency shifter

The musical effect of this algorithm is illustrated in figure 11 below. Clarinet pitches below C#" are harmonized, while notes played above G"' are frequency shifted.

---

[91] Sean Costello "Signal Modifiers: Standard Filters: hilbert" 1999. <http://www.esm.rochester.edu/onlinedocs/csound_docs/sigmod/hilbert.htm> accessed 3/13/04. For a more complete explanation of Amplitude Modulation and AM synthesis, see Barry Truax, ed. "Amplitude Modulation," in *Handbook for Acoustic Ecology*, 2nd edition, Vancouver: Cambridge Street Publishing, 1999. <http://www2.sfu.ca/sonic-studio/handbook/Amplitude_Modulation.html> accessed March 23, 2004.

Furthermore, each time the g above the staff is played, a new frequency shifter value is chosen at random from –2000 to -6499.

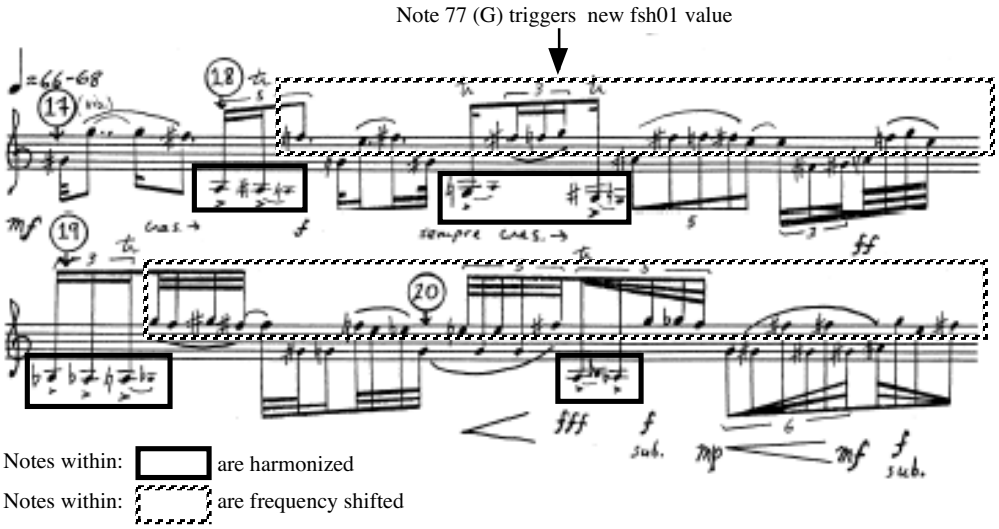

Figure 8.13. Score example: algorithmic control of frequency shifter and harmonizer

Flange. A flange (or flanger) is a standard signal processing effect most commonly associated with the electric guitar in Rock and Roll applications. The effect is more properly described as "phasing" and is based on a variable delay with a very short delay time (usually under 10 milliseconds). The effect is similar to a comb filter, in that the short delay produces phase cancellations at certain frequencies. When the delay time is modulated by LFO, the cancelled frequency bands move up and down the audio spectrum, producing a kind of "swishing" sound.[92]

[92] Barry Truax, "Phasing" *Handbook for Acoustic Ecology*, 2nd edition, Vancouver: Cambridge Street Publishing, 1999. <http://www2.sfu.ca/sonic-studio/handbook/Amplitude_Modulation.html> accessed March 23, 2004.

A standard flange effect found on most commercially produced effects processors usually includes controls for input level, LFO speed, depth, feedback, and output level. Lippe's flange module assigns the variables (each with a range from 0 –127) *flange-index, flange-speed*, *flange-del*, *flange-loop*, *flange-amp*, and *flange-master* to dynamically control these same parameters respectively. Full specifications for Lippe's flange module, including scaling factors for input values, will be included in the block diagrams in appendix C.

Frequency/amplitude modulation. This module features an unconventional use of frequency modulation (FM) and amplitude modulation (AM) synthesis in that it transforms the live sound of the clarinet rather than simply generating a completely synthetic tone. Pitch and amplitude information derived from real-time analysis of the incoming clarinet signal is used to control FM and AM synthesis operators that in turn modulate the live clarinet sound. An envelope follower controls the amplitudes of several oscillators used for FM synthesis, based on the amplitude of the incoming clarinet signal from the microphone. Pitch data from the pitch-tracking module is used to set the carrier frequencies of two separate FM operators. Finally, the clarinet signal itself is used as a modulator signal.

This module contains two parallel FM synthesizers, *FM1* and *FM2*. *FM1* is a simple FM instrument. Its carrier is a 555 Hz cosine wave, modulated by a cosine wave with its frequency dynamically determined by the value of the system variable *continuous_pitch* (set by the pitch-tracker). The envelope follower controls amplitudes of

both the carrier and the final *FM1* output. *FM2* uses both FM and AM synthesis principles. The carrier frequency (also a cosine wave) of *FM2* is set by the *continuous_pitch* variable from the pitch-tracking module. The modulator is the direct clarinet signal from the microphone. The output amplitude for *FM2* is controlled by the envelope follower, and then modulated by a 223 Hz cosine wave. The output of this AM operation is then modulated a second time by a 50 Hz cosine wave. The output of *FM1* and *FM2* are summed before output. Final amplitude for the entire module is controlled by the *fmam-master* variable, which has a range of values from 0 – 127 and is scaled exponentially from 0 – 1 as linear amplitude.

The final output of the frequency/amplitude modulation module is mixed with the final output of the flange module. This combined signal is then mixed with the output of the frequency shifter. All three are therefore treated as a single module for purposes of internal signal routing and final output/spatialization, described below.

Signal Routing. The internal audio signal routing network of the signal processing and synthesis instrument is set up so that the signal input to each module is a customized mix of the outputs from all the other modules. For example, the reverb module can take as its input signal a combination of microphone signal, sampler outputs, noise, reverb, harmonizer, and frequency shifter, all scaled individually and controlled by variables set in the event list. Variables used to control final output levels for each module are *pto2*/*pto4* (clarinet signal), *sto2*/*sto4* (sampler 1), *tto2*/*tto4* (sampler 2), *nto2*/*nto4* (noise

modulation), *Rto2*/*Rto4* (reverb), *hto2*/*hto4* (harmonizer), and *fto2*/*fto4* (frequency shifter).

Spatializer. The output of each DSP unit is placed within the left-right stereo field by setting its signal level for each DAC output channel. Each sound source is controlled by a separate variable, allowing for precise control of each sound element's stereo placement. The Spatializer also controls the final amplitude of the entire DSP section output sent to the DAC according to the value of the global variable *tgp* (0 – 157, scaled exponentially). The DAC output is routed directly to the loudspeakers. Therefore, the values used to control the Spatializer determine the mix of computer-generated sound heard by the audience.

Input signals to the Spatializer are controlled by a set of variables, each with a range from 0-127 (based on MIDI controller values) scaled to control signal amplitude. These variables, in left/right pairs, are: *fto2*/*fto4* (frequency shifter); *hto2*/*hto4* (harmonizer); *Rto2*/*Rto4* (reverb); nto2/nto4 (noise); *tto2*/*tto4* (sampler 1); *sto2*/*sto4* (sampler 2); *pto2*/*pto4* (clarinet from the microphone).

At certain score events, the Spatializer is controlled algorithmically. Left/right speaker placement of the final output from the sampler, noise envelope generator, reverb, harmonizer, and frequency shifter are selectively put under algorithmic control. Only the modules selected for algorithmic control are affected by the following three automated panning routines.

In section II, event 19 through two seconds after the onset of event 20 (controlled by *spatXY-19*), sound output for all modules is set hard-left or hard-right at random intervals between one and six seconds, with both sides set to zero after a one-second delay. In section I, events 12- 16, hard-left or hard-right placement of the computer generated sound is chosen randomly each time a note between 50 and 62 is played (clarinet lowest E to the E one octave above) and is detected by the pitch tracker.

Throughout the event list, the variable *spaton,* with values from 0 – 4, is used to choose from among four automated cross-fade panning patterns. These cross-fade patterns are determined by a set of tables, each containing a sequence of 128 left-speaker and right-speaker amplitude values. A second variable, *spatinc,* is used to control the speed and direction through which the tables are read. Values from 0 – 63 cause the tables to be read backwards, while values from 65 – 127 cause the table to be read forward. At the extreme ends (*spatinc* values of 0 or 127) the table is read at a rate of one value every 20 milliseconds. Values approaching the mid-point of 64 become progressively slower. A value of 64 freezes the panning effect in place. A diagram of this effect algorithm, with a representation of the tables, will be included in Appendix C.

8.4 SUMMARY

Lippe's *Music for Clarinet and ISPW* presents a very complicated test case for the type of analysis I advocate for the preservation and reconstruction of older interactive

electroacoustic works. The software is simultaneously a composition and a working instrument, and was not designed for abstract analysis.

Despite the difficulties in comprehending not only Lippe's compositional and synthesis algorithms (not to mention his source code), careful analysis shows that all of the processes employed in this work are based on commonly accepted principles of audio signal processing and algorithmic composition. Each effect or technique used in *Music for Clarinet and ISPW* can be found, at least in its simplest form, in one or more standard computer music textbooks. Therefore the problem is one of classifying each effect, breaking it down to its component parts, identifying its variable parameters (and their values), and describing the controls needed to operate this system during performance.

One further application of this analysis might be to re-engineer the software in a way that completely separates the synthesis and signal-processing instrument from all of the score-related data and processes. This result would be a very sophisticated general-purpose virtual machine and a transparent format for preparing score-related data. This would provide a clearer window into Lippe's compositional technique, and may also provide a practical resource for composers interested in pursuing similar methods.