Using PD for Live Interactions in Sound. An Exploratory Approach

Agostino DI SCIPIO
School of Electronic Music
Conservatory of Naples, Italy
discipio@tin.it

Abstract

The author illustrates the compositional concept and the main technical details of his *Modes of Interference*, a work conceived as an audio feedback system, performed with trumpet and electronics.

A Pure Data signal patch was specially prepared, only leaning on very basic features of PD as a programming language for real-time digital audio. However, it has a not-so-basic goal to fulfill, and that is to implement a nonlinear coupling between a microphone and a pair of loudspeakers, such that a self-regulating feedback dynamics is established, providing the trumpet player with control over the artifacts of the audio feedback loop (Larsen tones). The patch operates a few transformations, too, of the material circulating in the feedback loop, with granular sampling methods. The transformed material itself enters the feedback loop, affecting its behaviour.

The musical potential of the approach depends on innumerable acoustical and technical circumstances of the live performance, yet the sounding results consistently reflect the composed network of sonic interactions and all subtle nuances in the trumpet player's action.

Keywords

Audio Feedback Control. Live-electronics. Real-time Generation and Processing of Control Signals. Pure Data.

1. INTRODUCTION

The phrase *live interactions in sound*, in the title of the present paper, suggests that the technological infrastructure in a performance of live-electronics music can be seen as a network of mutually influencing sound-generating and -transforming processes or devices (including not only signal processing software and musical instruments, but all transducers and electroacoustic devices involved in the performance set-up). In abstract terms, the real-time coupling function of any two nodes in the network is defined as a recursive function:

$N1 \Leftarrow f(N2)$
$N2 \Leftarrow f(N1)$.

In a more realistic example, we should consider many more nodes, each coupled with each other.

That becomes especially relevant, in my opinion, when all coupling functions - i.e. all communication among network components - takes place primarily if not solely *in sound*. That requires of us that we deal with sound not as just a raw material to forge and deliver to listeners, but as the source of dynamical behaviour within the network, i.e. the vehicle of information transferred from any one node to another. Sound then becomes the *interface*, the medium itself for control and interaction among agencies participating in a performance - either human or machine agencies. In this perspective, a composer composes *not* the music itself, as a definite object existing regardless of given and accepted conditions, but rather a veritable dynamical system and the real-time interactions among the system components. The emergent sonic behaviour we may call *music* (Di Scipio, 2003) (Anderson, 2005).

1.1 Background

My efforts in this line of experimentation recently resulted in sound installations (e.g. *Ökosystemische Klanginstallation in einem kline halligen Raum*, Berlin, 2005) and in a series of live-electronics solo works for concert performance (*Audible Ecosystemics*, 2002-05). Most of these works are based on audio feedback between microphones and loudspeakers, and have some signal processing software handle the feedback gain (thus actually mediating between microphones and speakers) and transform the audible byproducts born of audio feedback. There is *no sound input* to the technical set-up of such pieces: the only source of energy they chew on is background noise, and that is more than enough to mantain their self-sustaining process (Di Scipio, 2005).

1.2 Current Work: *Modes of Interference*

In a similar vein, I also started experimenting with the idea that the audio feedback between speakers and microphones could be controlled not only by signal processing software, but by one or more human performers too. Someone playing an instrument intrudes into the autonomous self-standing feedback loop, and interferes in and with it, in ways peculiar to his/her instrument.[1]

[1] Other composers have experimetned similar approaches, notably including Alvin Lucier (see, e.g., Lucier 1995) and, more recently, the Italian composer Michelangelo Lupone in his remarkable 1999 composition *Gran Cassa*, for big bass drum and feedback system (Lupone and Seno, 2006). Recent

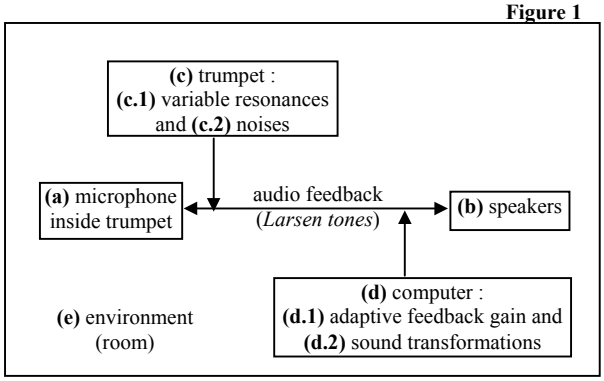
One outcome is *Modes of Interference*, a work to perform with trumpet and some electronics (ZKM Karlsruhe, Winter 2005-06). The electronics include some digital signal processing, currently having the form of a signal patch designed with Pure Data. I relied on basic features of PD as a programming language for real-time audio (Puckette, 2003), quite useful for rapid prototyping and testing of signal processing methods in live-electronics contexts.[2]

In the following, I will first overview the system infrastructure of *Modes of Interference* and then describe in some detail the PD signal patch itself. In passing, I will eventually illustrate the peculiar role assigned to the trumpet in relationship to the signal processing patch and the overall technical set-up.

2. SYSTEM OVERVIEW

The system infrastructure to *Modes of Inteference* can be illustrated as depicted in figure 1.

Figure 1



2.1 Audio Feedback Loop

Fundamental is a feedback loop between **(a)** a miniature microphone placed *inside* the trumpet (figure 2) and **(b)** two or more loudspeakers. A high gain is requested, so the feedback loop builds up until audible oscillations result (sometimes called *Larsen tones*). In the managment of sound systems, the latter phenomenon usually represents a technical problem to be avoided at all times. In this work, instead, it becomes the main source of music.

2.2 Mediation between the Loop Ends

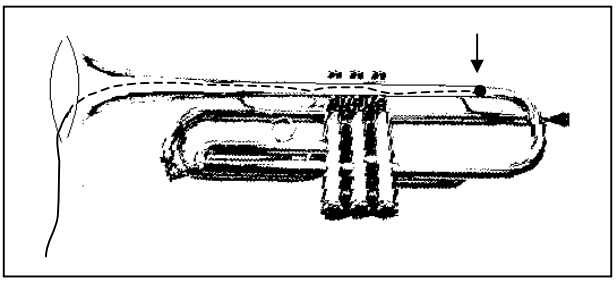
In between the two ends of the audio loop are **(c)** the trumpet tube and **(d)** a signal-processing computer. Both are to control and play with the audio feedback artifacts, and to turn a problem into an opportunity.

2.2.1 Trumpet Mediations

We should observe, that the room surrounding the miniature microphone, inside the trumpet, is a very small niche (diameter ≈ 1 cm), a passageway narrow enough to act as a strong resonator. That causes a strong reinforcement of sounds captured by the microphone, and add to them a peculiar spectral coloration (the geometry of the cylinder tube creates a waveguide with its own resonant frequencies). That helps make the gain in the feedback loop larger, so we may lean not exclusively or primarily on pre-amp stages (circuitry on the mixing desk, or else) in order to deliberately cause Larsen tones.

Actions to perform on the trumpet are meant to **(c.1)** change the resonances in the tube. That is made by operating on the piston valves (no blowing requested). The instrument becomes a kind of variable filter internal to the feedback loop. Other actions on the trumpet are to **(c.2)** introduce noise into the system (e.g. percussive effects, *air* sounds, etc.). Only occasionally has the trumpet player to blow (softly) into the mouthpiece, and anyway never has s/he to use the lips as a double-reed, like in normal playing.

Figure 2



Piston valve action and the introduction of tiny noise events into the system have the effect to change the current state in the feedback loop, and result in changes of pitch, amplitude and onset time of Larsen tones.

Furthermore, the trumpet performance becomes the source for variations in the signal processing. Especially features like the amplitude as well as the density (= amount of onset transients in time) of events circulating in the trumpet tube, are used to drive some crucial variables in the signal patch - as we'll see later in this paper. In essence, the trumpet here is both a means to create sounds and a means to exert control over signal processing operations (tha latter transform the sounds created by the trumpet).

2.2.2 Computer Mediations

The signal patch has two main roles to play. In the first place **(d.1)** it dynamically adjusts the feedback gain, trying to keep the system in equilibrium at all times. As we'll see later, this is implemented as an adaptive process, depending on the total sound

---

research has been also carried out - with PD - to implement some John Cage's work with feedback systems (Burns, 2004).

[2] I tried to keep the patch design as tidy and simple as possible, like a template providing enough general information for possible implementions with other programming tools.

level at any time in the performance. The goal is to make sure feedback gain is high enough to result into Larsen tones and other artifacts, but not too high to cause clipping of the digital signal or saturation of the analog components in the technical set-up.

Secondly **(d.2)**, the patch also extends and transforms all sounds circulating in the loop process, including wanted and unwanted sounds (handling the trumpet may itself create tiny noises). Because the products of the signal processing patch are heard over the speakers, they of course enter the feedback loop and interfere with its audible byproducts, e.g, affecting the pitch or ampltude of Larsen tones.

2.2.3 Performance Process

The instructions and score to *Modes of Interference* describe a number of ways for the trumpet player to interfere with the audio feedback and explore its sonic potential.[3] What s/he actually does is to learn and explore the system dynamics of a network made of interconnected sound-generating or -transforming components (as overviewed figure 1). To be precise, we should consider the trumpet player him/herself as part of the feedback system, not as an external entity: his/her own playing should, or at least may, vary depending on the sounding results brought forth at any time by the audio loop. All his/her actions affect the feedback loop, but the sound born of the latter make him/her decide on actions to be done next. A chain of causes and effects is established, a process where not only every component instantaneously affects every other, but where every event of sound has consequences on events to come, both in the short and the long run.

Clearly, such as system is also susceptible of being affected by perturbations in the environment - as already pointed out in figure 1 **(e)**. That is the case to the extent that the material body of the trumpet (I mean the cavities of the tube, bell, and valve mechanisms) resonate to and from external foreign events (background noise, accidental events in handling the trumpet, accidental events in the room, etc). In other words, the external world does enter the feedback loop, affecting its behaviour, but only through the mediation of (= only filtered out by) the trumpet as a sounding object with its material properties.The instrument becomes a proper *sound interface* between inside and outside.

3. The PD audio signal patch

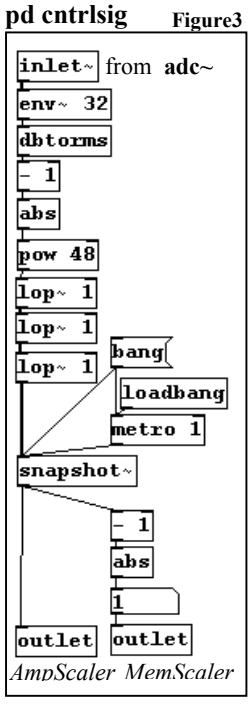
The digital signal patch for the computer was prepared with PD 0.39.2.[4] In preparing it, I wanted it to be as computationally inexpensive as possible, and with a user interface easy and straightforward enough to be operated by some other person - possibly the trumpet player himself (the GUI, however, will not be illustrated in this paper). Therefore, I confined myself to the roughest and simplest implementation possible, trying to only get the best acceptable-yet-minimal results that would be consistent with both my compositional concept and the practicalities of the performance moment.

The patch was prototyped and first tested on laptop computers under XP (either Centrino or PentiumIV, 500 Mhz or higher), and then tested again on Linux (PentiumIII/dual 600 Mhz) and OSX desktop computers (acceptable performances only with a G5/dual 2 GHz).

The difference in latency time (minimal under Linux, maximal under XP) raises an interesting problem, because it adds into the audio feedback process a different delay upon each different computer. The different delay alone (all other circumstances being equal) may result into a slightly different system dynamics. The solution here can only be on the practical side: one may either adjust a few time-related parameters in the signal patch, or try a slightly higher feedback gain than otherwise necessary. The problem is deemed solved only based on a qualitative (musical) criterion, namely that enough control is allowed to the trumpet player to flexibly deal with the audio feedback loop.

3.1 Generation of Control Signals

We want to create one or more feasible control signals out of the input sound (by *input sound* I mean either Larsen tones or anything captured by the microphone inside the trumpet). To do so, we extract some information out of the signal, and process that information turning it into a control signal.



3.1.1 Feature-extraction

Many feature-extraction methods are known. Here we use is one of the most basic, *envelope following* (the **env~** object). With the subpatch **pd cntrlsig** (figure 3) we take the average value of the digital sample values every next 32 msec (that's one of the values that possibly need adjustments,

[3] See http://xoomer.virgilio.it/adiscipi/internal_modes.htm
[4] Puckette's version, http://puredata.info/, on 15.10.2005

as mentioned in the previous subsection).

3.1.2 Mapping (processing of extracted data)

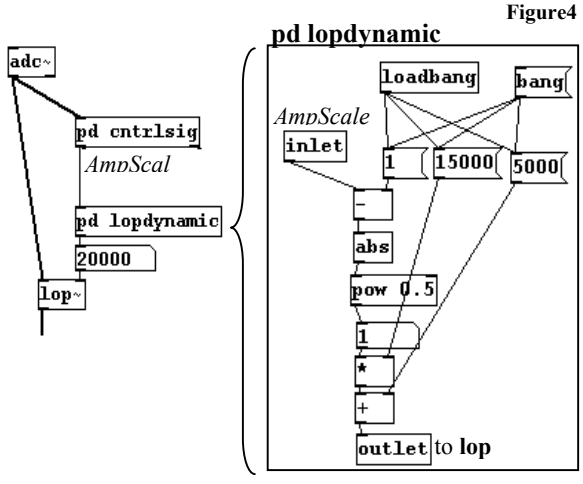
We take the absolute value of the **env~** output signal, and shape it by raising it to the power of 48 (different exponents could also do). We then use three cascaded one-pole low-pass filters (**lop~**) with a cut-off frequency of 1 hz, to smooth-out ripples and small amp fluctuations in the signal. The result (let's call it *AmpScaler*) is sent out for use in other subpatches. It is a low-frequency unipolar signal, in the full scale range [0, 1], following changes in input amplitude, but in inverse relation to it: when the input sound gets louder, *AmpScaler* gets smaller, when input gets softer, *AmpScaler* gets larger. The complement (*MemScaler* = 1 - *AmpScaler*) is also calculated and sent out for use in other subpatches. It stands in direct (but exponential) relation to input amplitude.

3.2 Feedback (Self-)Control

3.2.1 Automated Bandwidth Control

The input signal is routed to a low-pass filter (**lop~**) whose cutoff-frequency is driven by the *AmpScaler* control signal. The mapping from scale values to Hz values is [0, 1] ⇒ [20000, 5000], and follows a logarithmic curve in that interval. This takes place in the **pd lopdynamic** subpatch (figure 4). As a result, when the input amplitude is softer, the low-pass bandwidth is larger, and when it is louder, the bandwidth is narrower.

Figure4



3.2.2 Automated Gain Control

The output of **lop~** is then multipled by *AmpScaler* (figure 5). Remember that *AmpScaler* is in inverse relation to the input amplitude. Therefore, as the input gets louder (and the bandwidth narrower), the filtered signal is scaled down.

Next, the signal amp is boosted-up again by a user-defined value (included in the GUI), called the **inputGain** (set to an arbitrary value of 8, in figure 5). The boosted signal is sent out for digital to analog conversion, but it is also used as input to another subpatch (**pd gr8**, described below).

3.2.3 Effect on the Audio Feedback Loop

The latter process, in coordination with bandwidth control, has decisive effects. The **inputGain** value should be large enough to cause audio feedback when output is routed to the speakers - however, *AmpScaler* reduces the output amplitude as the input amplitude increases, so the audio feedback loop will never really grow to the point of clipping or distorting. It will rather adjust itself, in a continuing adaptive balancing between input and output. When there is little input, the feedback gain is larger to make way to Larsen tones and other byproducts; as Larsen tones and other sound materials get louder, however, the gain decreases enough to preventing too strong howling distorted sounds. When the a perfect balance is found, the process gives way to the emergence of Larsen tones, keeps them at a rather constant amplitude, and eventually lets them fade out slowly.

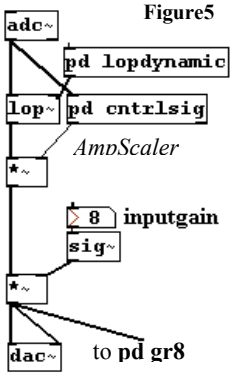
Because of the automated bandwidth control, the chance that high-pitch Larsen tones emerge from the feedback loop is greater when the input is soft, and smaller when the input is loud. That makes sure (1) that there is enough energy in the system across a large frequency range, so Larsen tones of any frequency can in principle come out, and (2) that very high frequencies are dampened before they come out too strong (and painful for the ears).

By having the *AmpScaler* control signal both to drive the filter bandwith and the feedback gain, we actually establish a perceptually relevant link between spectral width (brilliance) and amplitude (level, strength). The coordination between the two mechanisms realizes a kind of self-gating process, an gating mechanism adapting to the continually changing input amplitude.

3.2.3 Systemic Relevance of Time Variables

Different from common *dynamic processing* methods (limiters, compressors, etc.), the strategy implemented here works with a short delay or latency determined by the *envelope following* mechanism (let's not consider now the 30-to-70 ms latency introduced by operating systems into the PD runtime performance). That is problematic if Larsen tones with fast or very fast onset transients occur, as in that case the signal may peak

Figure5



and get distorted for few instants before the self-gating mechanism is driven to scale the gain down enough to dampen the feedback oscillations.

However, using more common dynamic processing in place of our shortly-delayed self-gating mechanism, could make it problematic to get any audio feedback at all, and would anyaway make necessary to resort to much larger amplification levels. In a way, the latency of the amplitude following makes things easier, letting the Larsen phenomenon happen anyway and typically determining slower Larsen onset times than would be otherwise the case.

Furthermore, the latency artifact of amp following may even give way to a richer system behaviour: short peaks in the audio feedback loop may elicit particular room responses that would otherwise not be elicited, and thus it enlarges the array of sounding results. Finally, a short delay in the self-gating mechanism gives a little more room to the trumpet player in the attempt to gain and exert control over things in the process of happening. Provided s/he stands not too close to (and not too far from) the speakers, s/he can properly handle the occurence of peaking Larsen sounds by either operating the pistons, or producing some noise - or even just moving the instrument a little farther away or closer. With these security measures, s/he causes a shift to either weaker or stronger resonances: the former would stop the peaking Larsen tone directly: the latter would cause the self-gating mechanism to more quickly reach a smaller feedback gain value, thus dampening the Larsen indirectly.

For the same reasons, the latency imposed by the computer OS on PD is acceptable, at least within reasonable limits, and to the extent that it becomes an element contributing to the sytem dynamics. Some branches of the signal patch do include short delay lines (in the order of milliseconds). If the system dynamics seems to significantly suffer from latency or delay values, at least the delay lines and the window size in the envelope following should be adjusted, as already suggested above.

3.3. Cascaded Granular Resampling

As already mentioned, after a stage of amplification (**inputGain**), the signal is sent to not only the **dac~**, but to subpatch **pd gr8, for** further processing. Before entering the subpatch, it is first multiplyed by *AmpScaler* (figure 6): that way, it is made softer when the overall input is louder, and louder when the latter is softer. Thus, loud materials will never come out too loud from the processing itself, and will not add too much on top of the input.

In the **pd gr8** subpatch, the incoming samples are first written in a wavetable and then read off the wavetable, with modifications in the resampling values. The output signal is sent to the **dac~** and used, too, as input to subpatch **pd gr4**. Inside the latter, the signal samples are written in another wavetable and again resampled. The output is sent to the **dac~** (also via **pd shortDelay**) and to subpatch **pd gr2**. Inside the latter, the signal samples are written in still another wavetable and again resampled, and the output is sent to the **dac~** (also via **pd shortDelay**). The process, has a cascaded, iterative design : the output from the first granulation is the input to the second, and the output from the second is the input of the third. The signal of all three stages is part of the total output sound. The latter then consists not only of the audio feedback byproducts, but of three layers of resampled material too.

Subpatches called **pd shortDelay** represent single-tap delay lines, in the range 20 to 35 msec. They function is not discussed here, but should be clear from the way their signal is routed to the output.

3.3.1 Sound Granulation

The cascaded resampling method is primarily to *pulverize* or *granularize* the incoming sound. In figure 7 is the subpatch **pd gr8**. It follows closely from the design of the B11.sampler.pd patch (that is part of Puckette's library of PD audio examples). However, there are modifications and extensions in order to turn the process into a continuing wrap-around write/read process (new input samples replace old ones when the wavetable is filled), and to let important variables be driven by control signals.

The wavetable in **pd gr8** is 8-second long. Dynamically controlled variables include **scan speed** (for the memory pointing mechanism), **grainsize** (length of sample chunks) and **freq** (resampling ratio). The output of the memory read is enveloped by a shifted positive cosine, is passed through a high-pass filter, and finally sent out. As already mentioned, it is routed to both the **dac~** and the **pd gr4** subpatch.

Subpatches **pd gr4** and **pd gr2** are identical with **pd gr8**, except that the wavetable length is respectively of 4 and 2 seconds.

3.3.2 Systemic Role of Granular Resampling

The output from the cascaded resampling process is scaled by *AmpScaler*. Therefore, a loud input will determine not only a dampening of the signal entering the process, but also a softening of the granulated sound material. This is releveant to the fundamental audio feedback loop. Indeed, were the input to the patch to be sparse or soft, the granular

material would be loud and probably spectrally rich enough as to interfere in the fundamantal feedback loop, and possibly changing its resonances, giving way to modifications in pitch of Larsen tones, etc. Were the input sound to be dense or loud, instead, the granular material would be scaled down, so it doesn't add too much on top of the input sound. A balancing strategy is thus created, similar to the one used for feedback control. That too is of some relevance in the attempt to keep the system in a state of dynamical equilibrium.

As illustrated in figure 8, the **scan speed** is driven by the *MemScaler* control signal (right outlet of **pd cntrlsig**, see also figure 3). The mapping from scale values to **scan speed** values is made in such a way that, when *MemScaler* is precisely 1, the pointer mechanism scans 4 times the wavetable through in 1 second (twice forwards and twice backwards, following the shape of a positive sine - see rightmost branch of the subpatch, in figure 8). When *MemScaler* is closer to 0, the pointer slows down. Eventually it freezes were *MemScale* to be 0 (pointing repeatedly to one and the same chunk of samples). Which is never really the case, as there is always background noise in the small microphone connected to the patch input, especially due to the gain necessary to create audio feedback).

pd gr8

Figure7

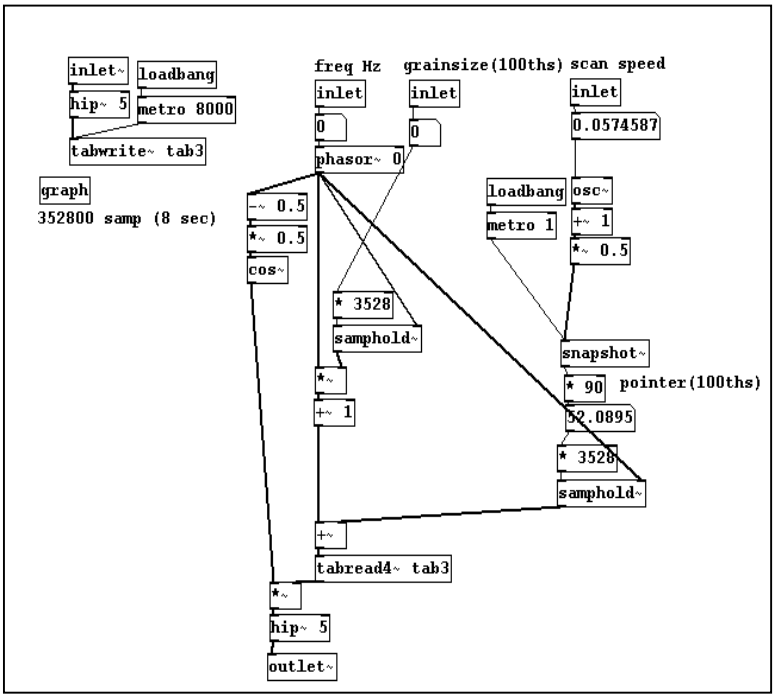
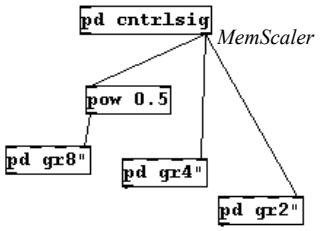


Figure8



In other words, when the overall input sound is louder, the pointer mechanism runs faster through the wavetable, and viceversa. The consequence is, feeble sound events will have the process focus only on few segments of the wavetable material, and hence result in prolonged granular textures, while louder events will determine a wider area of the wavetable be resampled and granulated, resulting in more articulated and gestural musical output.

3.3.3 Partially Feedback-independent Variables

The **freq** and **grainsize** variables in the resampling subpatches are not entirely dependent on live generated control signals. I use message boxes to arrange lists of values for these variables, and have an internal subpatch interpolate between the values, thus updating the variables. Basically, this is made in such a way that the range of frequency shift and grain duration widens as the peformance proceeds from beginning to half-way (a time span of around 4 minutes), and then gradually resumes the initial values as performance proceeds from half-way to the end (4 more minutes). That imposes some precise timing to the performance, although the trumpet player can enjoy a substantial felixibity in that aspect, across the two larger spans of time. Fermatas and cadenza-like passages included, the overall duration is around 10 minutes.

While they are automatically updated, these two variables are also driven by the *MemScaler* control signal, but only to add some small jitter deviations to the current **freq** and **grainsize** values.

4. Conclusions

In its main functionalities, the PD patch discussed above should be considered as the implementation of a nonlinear coupling function between a microphone and two loudspeakers. Its process is tightly intertwined with the overall performance process of *Modes of Interference*, and determines the boundaries of the system dynamics that is created live, in the very *here and now* of performance.

Taking a broader perspective, we should say that all elements are determinant of the performance process. Indeed all agencies in play - human performer, computer and audio equipment - participate in a process that is *productive* of sound and music, not *reproductive*. Even speakers - the most commonly given-for-granted technology in electroacoustic music - are less to deliver sounds than to create them. The overall technical set-up

and the room acoustics does not simply *host* the performance, but shapes it and contributes actively to it, while also setting precise material conditions and boundaries for it to happen. Different rooms will make for a different environment surrounding the trumpet - itself a smaller room. And that will make for a different system dynamics, with different sonic manifestations of the same overall musical structure.

Managing the nonlinear behaviour of the audio feedback system can represent quite a challange for a composer. It is certainly even more so for a performer, as complexity of real-time nonlinear behaviour in feedback sound can show peculiar and even idiosyncratic aspects to deal with. Trade-off is always in order. Must be. What is gained to the performance is tension that is tantamount to really have a *live* network system to manifest itself.

The phrase *complexity of behaviour* here refers not to some kind of computer modelling, as it is not captured in the particular signal processing methods - the latter are basically quite rough, as I tried to illustrate. It has little to do with the computational load necessary. It comes rather with the experience of the fact that all elements involved are connected and affect each other to some greater or smaller extent, with causal effects on the long run.

4.1 Acknowledgments

Thanks are due to the Institut für Musik und Akustik of ZKM for offering their residency/commission program. Special thanks to Götz Dipper for getting me started with PD, to Joachim Goßman for technical assistance in the concert in the Kubus (27.04.2006), and to trumpet player Marco Blaauw - to whom *Modes of Interference* is dedicated - for accepting the challanges put forth by this work upon him.

References

C.Anderson. 2006. Dynamic Networks of Sonic Interations. An interview with Agostino Di Scipio. *Computer Music Journal*, 29(3): 11-28.

C.Burns. 2004. Designing for Emergent Behavior: a John Cage realization. Proc. ICMC, 2004.

A.Di Scipio. 2003. Sound is the Interface. From *Interactive* to *Ecosystemic* Signal Processing. *Organized Sound*, 8(3): 266-277.

A.Di Scipio. 2005. Klangstaub (Die Notwindigkeit einer aesthetischen Orientierungslosigkeit). *Positionen*, 64: 45-48.

A.Lucier. 1995. *Reflexionen. Interviews, Notationen, Texte*. Edition MusikTexte, Köln.

M.Lupone, L.Seno. 2006. Gran Cassa and the Adaptive Instrument Feed-Drum. Text of a lecture delivered at the conference *Le Feedback dans la Creation Musicale*, GRAME, Lyon. From www.grame.fr/RMPD/RMPD2006/

M.Puckette. 2003. *Theory and Techniques of Electronic Music* (Draft). UCSD. From crca.ucsd.edu/~msp/techniques/latest/book-html/