‘Ambiguous Live’ – Exploring Collaborative, Dynamic Control of MIDI Sequencers

David Eyers

Computer Laboratory
University of Cambridge
David.Eyers@cl.cam.ac.uk

Abstract. Computer sequencers generally control other digital musical instruments. This paper discusses our particular interest in playing the sequencers themselves as digital musical instruments. We present our ensemble’s approach to live sequencer-based performance, supporting multiple musicians coordinating activities through one central control system. We then review a number of promising software technologies currently available for the purpose of such live sequencer-based performances, and finally discuss our consequent future performance goals.

1 Introduction

People have gathered together to enjoy musical performances for thousands of years. Many of these performances have included groups of musicians who, through improvisation or interpretation, cooperate to resonate their ideas and musical intuition. Modern day orchestras and bands combine the talents of their constituent musicians to form a dynamic and complex higher-order performance entity.

How does this compare with the music flourishing throughout the night-clubs of the world today, however? Many have no respect for this ‘dance’ music, due to its prefabricated nature. However, we feel that proliferation of digital musical technology is an opportunity rather than an affront to music performance (or it is at least both!).

Whilst hardly a house-hold name, ‘Ambiguous Live’, was a duo formed in part by the author for the sake of investigating the ease with which we could design software, supported by a hardware rig of synthesisers, to enable live, largely improvised electronic music performance. Our approach involved the high-level control of a computer sequencer, centrally coordinating the actions of multiple musicians. In the group’s brief lifetime, we performed in a number of university-oriented Sydney nightclubs, and were interviewed regarding our live operating environment in the street magazine ‘3DWorld’ under its ‘localz’ section in March 2001 [1]. Unfortunately this group had to suspend activities when the author left Australia to pursue postgraduate education in the UK.

This paper is organised as follows. We provide some background discussion relating to electronic music performance in section 2. We discuss our performance approach in section 3. A number of technologies which support our goals to a greater or lesser extent are reviewed in section 4. Section 5 presents our plans and desires for future performance environments. Finally we conclude the paper in section 7.

U.K. Wiil (Ed.): CMMR 2003, LNCS 2771, pp. 54–63, 2004.

2 Background

Disc Jockeys (DJs) originally had the job of setting up the media to be played in sequence for radio stations. They may have provided voice interludes between the songs, and indeed may have had a greater or lesser degree of control in the actual sequence of music played. Although possibly very musically knowledgeable, they were not musicians in any performance sense.

Nightclubs also need someone to keep music playing when there are no live bands, so the role name was transfered to this environment, although the job being performed varies greatly. Some DJs merely choose an order of tracks to play, reading the mood of the crowd but nothing more than that in a performance sense. These DJs are sometimes bestowed undue credit; the audience members are actually enjoying the work of the composers rather than the DJs, but the DJs provide a face to which to direct appreciation. However, most club DJs interact more directly with their performance machinery, controlling the speed and looping of a number of sound sources – they simultaneously perform audio mixing and compositing. Compact Disc (CD) players (or possibly computer-based multi-track MP3 players) may be used by comparatively unskilled DJs, although most serious DJs use traditional phonograph turntables. Undoubtedly good turntable control is a skilled role, and there is ample space for artistic creativity. However, the musical building blocks with which such DJs perform are still very coarse in musical granularity (at the level of whole tracks or phrases).

The most sophisticated DJs in fact render their own musical material, and indeed are the artists responsible for making the material other DJs mix and play back to their audiences. However, even these DJs have only limited abilities to create live music performance. Some have taken to including drum machines, or other simple sequencers into their performance rigs – we feel that is an initial step to acknowledging the relevance of sequencer performance discussed in this paper.

3 Ambiguous Live

Our performance goal was to be able to create synthesised music via live collaboration between performers, with a heavy emphasis on improvisation. Limitations in our software sequencer meant that we did need to prerecord a number of musical patterns, however for most monophonic parts we divorced the rhythmic control from the tonal control. Essentially we programmed the synthesiser to convolve separate rhythmic and melodic patterns into MIDI events, in effect creating a simple Control Voltage (a MIDI precursor) to MIDI converter.

Our equipment rig grew in between each of our performances. In our most recent performance, our inventory included three MIDI-controller keyboards (in the piano sense), two general purpose MIDI controller knob banks, two samplers, two analog modelling synthesisers (one monophonic, and one 16-channel multitimbral unit), a multitimbral digital synthesiser, a drum machine, four effects processors, two mixers, a Sony VAIO laptop, and a desk fan.

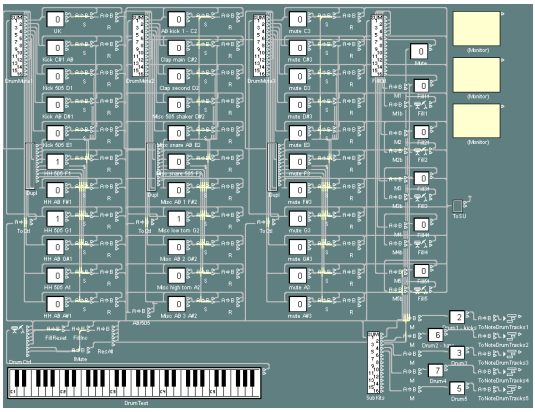

Fig. 1. The ‘Ambiguous Live’ rhythm control environment screen-shot

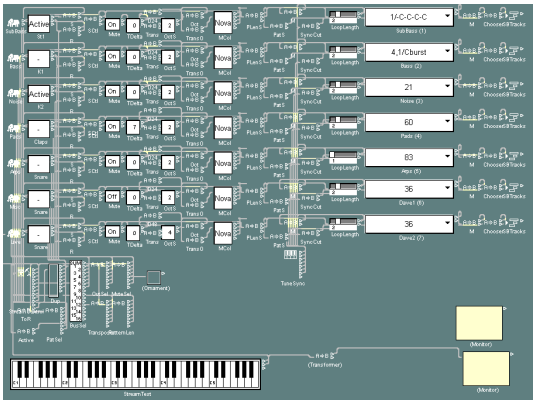

Fig. 2. The ‘Ambiguous Live’ stream control environment screen-shot

3.1 Music Control Structure

Our top-level control architecture consisted of two main regions. One whole control area in our environment was devoted to rhythmic parts (drums and playback of prerecorded samples); a screen-shot is shown in Figure 1.

The other main control area related to numerous ‘streams’. We used this term so as not to confuse streams with MIDI channels; any stream could output to any number of MIDI channels on any of our devices. Essentially each stream was responsible for a section of our music. A screen-shot of the stream environment is shown in Figure 2. We generally used the following streams:

Sub-bass. Used for very low bass used predominantly to keep sub-woofer speakers occupied, generally for the sake of shaking the performance venue rather than making particularly pitched contributions.

Bass. Provide audible bass-lines providing low-pitch texture and rhythm.

Noise. This stream was employed to play signature samples or noises to identify each particular part of a musical set.

Pads. Used for fairly soft-timbred chords providing a harmonic foundation.

Arps. Arpeggios are often used to produce high-pitch texture and rhythm.

Lead. Synthesiser lines which stand out well in front of the rest of the music. Usually monophonic.

Note that while we were using a sequencer which provided a piano-roll style view, the actual music output did not come directly from the time-line. Some control parts from the time-line were used to influence pattern playback however. For example patterns of different length aligned themselves on specific bar numbers, ensuring that changing from an 8 bar to a 1 bar and back to an 8 bar pattern would not cause the second 8 bar pattern to be played out of alignment with other 8 bar patterns.

3.2 Music Control Inputs

Another experimental aspect of our performances was that the centralised sequencer permitted both performers to collaborate, interacting across all the controls. There were several ways in which we could exert control over our music sequencing environment:

Via laptop. The mouse and keyboard of the laptop running our sequencer could be used to adjust parameters and override controls. We were only using one physical display screen, so needed to use computer keyboard hot-keys to change which virtual screen of data within the sequencer we were examining or manipulating.

Via direct synthesiser controls. All of our audio synthesis equipment was present at each performance, so we were able to directly manipulate each device's controls.

MIDI dial boxes. We used two banks of assignable MIDI dial boxes to permit us to control parameters on the equipment. We programmed the sequencer to provide a number of default pages of knob assignments, but included the capacity for knobs to be assigned by the laptop during performance.

MIDI keyboards. In the relative dark, we found it easiest to use three MIDI piano keyboards between us for both melodic performance, and for remote control of the sequencer. On the melodic keyboards, upper octaves played notes in various lead streams. The lowest octave was used for real-time transposition of streams currently set to have non-fixed root notes. Thus if the sequencer was playing some particular bass-line, and an introduced chord progression required its transposition to fit harmonically, it was sufficient for a performer to tap an appropriate lower octave key once to set a new root note. The middle octaves of the keyboard were reserved for muting, routing and assignment control. The mute controls allowed stream output to be switched on and off easily. The routing controls allowed each stream to be moved dynamically to other synthesisers, and finally the assignment controls allowed any of our prerecorded patterns to be assigned to any of the streams.

Note that to compress so many different functions into limited numbers of keys, chording was required (and we add works far better for keyboard-players on musical keyboards than on computer keyboards!). So, for example, three stream select notes could be played simultaneously with the mute key to affect them all together.

Our sequencer environment was created within Emagic's Logic Audio [4] software sequencer. The most remarkable feature of this program is its graphical MIDI programming environment. The next section discusses why we believe graphical environments are so well suited to sequencer programming and our type of application development.

3.3 Graphical Programming

Unlike traditional procedural programming in which one considers the computational steps sequentially, music performance, and thus sequencer programming, tends to be much more event driven. In effect, the performance environment can be thought of as a web of interconnected nodes, each of which responds to some sort of message or set of messages sent to it on its inputs, by itself generating some number of modified events on its outputs. Usually there is only very simple state held within each of these nodes. For MIDI sequencer programming, the environment is even more predictable, since the structure of the MIDI messages themselves are very well defined.

Emagic's Logic Audio package provides for the programming of MIDI 'environments', which consist of a number of types of nodes connected together visually by 'wires' which indicate where events will flow. The language is reasonably expressive; transformer nodes can themselves be modified by 'meta-events'; events intended only for the internal control of the sequencer.

Crucial to our programming were the Logic 'touch-tracks' node type, which allowed us to trigger the hundreds of simultaneous sequenced parts we needed as event sources under the control of the environment itself (visible on the right hand extreme of each stream path in Figure 2). In fact, the conventional sequencer time-line merely contained some muted emergency parts, and some parts providing control triggers into the environment.

Apart from some of the limits we reached in using Logic (discussed in section 5), we generally feel that graphical programming is particularly suited to live sequencer design and operation. By using two-dimensional spatial layout, a great deal of information can be compressed into each screen. Also, there is little distinction between design time and control-time. Graphical components can be controlled during performances in the same manner in which they are built up during sequencer environment programming.

4 Alternative Tools for Developing Live Performance Environments

We feel our performances merely scratched the surface of this fascinating potential area of live music. Due to availability and time limitations, we used the existing Logic sequencer as discussed above. There are a number of other promising current technologies which might be employed to perform music in a similar manner, some of the ones we experimented with before choosing Logic are described in this section.

4.1 Collaborative Music Environments

One of the most wide-spread collaborative music environments offered recently has been the Rocket Network [10]. Unfortunately the Rocket Network service providers

closed down their service in early 2003. The RocketControl™ software hooks into commercial sequencers to allow composers to conveniently exchange musical data, both in MIDI and audio forms. Some sequencer versions have been released which cater directly to this function, such as Logic Rocket, and Cubase InWired. The Rocket Network also provided a centralised storage facility for media.

There are numerous other musical collaboration sites on the Internet, but they generally offer a loose coupling, basically implementing some form of discussion board or Internet data storage site through which files can be passed back and forth between artists. However web browsers provide a very general interface to such services and suffer from not being able to cater specifically for the demands of music media.

Note that all of the above forms of musical collaboration environment were fundamentally unsuitable for our performances because they introduce a delay in the exchange of data and control. Whilst they provide great opportunities for globally distributed artists to work together with relative ease, they do little to help two performers standing next to each other to lever their mutual control of musical equipment.

4.2 Software Studios

Due to the recent surges in computer multimedia processing speed, it is now practical for a computer to do sufficient software synthesis to have prompted a number of companies to develop virtual studio packages.

Reason. Propellerhead Software's 'Reason' [9] is an excellent example of a virtual studio package. It provides the user an ability to create as many virtual synthesisers as any given computer is capable of producing faster than its audio streaming speed. Synthesisers can be linked together via a virtual patching system implemented within the software (and again, using a graphical view of signal flow).

For our purposes this software did not provide enough flexibility in the MIDI programming domain however. It might provide an excellent low-cost substitute for many types of MIDI synthesiser, but would really just perform as a target instrument in the rigs we have built.

Orion. Synapse Audio Software's 'Orion' [11] also deserves a mention. It provides the same excellent core of software synthesis and effects limited only by the speed of the computer on which it runs. However, the ability to experiment with custom MIDI and audio control and routing is more limited than that of Reason. So again, although the sound quality produced was truly impressive, we could not use this a package for any purpose other than as a very low-cost target MIDI instrument.

Cubase VST. Before designing our final performance environment in Emagic's Logic sequencer as we described above, we also experimented with Steinberg's Cubase VST (Virtual Studio Technology) software [12].

Whilst Cubase VST is an excellent package, which we found easy to use and which features good audio and MIDI integration, it unfortunately lacked programmability to the degree we required for our live sequencer performance. It has some controls which play patterns rather than just notes when keys are pressed, but without the ability to re-program MIDI event routing and transformations from within the sequencer environment itself, we were unable to live control it sufficiently for our needs.

4.3 Off-Line Programming Environments

We found it particularly convenient to have little distinction between programming and performance contexts, since development was an on-going process. However, if the design of your desired sequencer performance machines are fairly static, a number of tools exist for easing the programming of such applications.

One notable programming environment is C-Sound [3]. Audio and MIDI based performance systems can be designed in its programming language, and then compiled to make operating machines. As with most programming, the compilation step potentially provides a significant performance boost, but at the cost of preventing dynamic adjustment of the application's software.

Naturally general-purpose programming languages could also be employed to develop performance machines. This was not a viable option for our performances; our music machine design was constantly being extended, and our development time was severely limited.

In terms of reducing the programming load (at a speed penalty), the Sun's Java language [6] provides library functions to communicate with MIDI devices. Thus a significant amount of the machine-specific programming can be avoided and the programmer left to focus on the application. It also has the advantage of portability to other machines. As a real-time environment, however, even if we had had the time required to develop such custom software, we were sceptical that Java's timing would have been fast or reliable enough for our performance needs. Even so, JavaBeans provide an interesting potential component framework in which to move the flexibility of Java code modules into a live, graphical programming environment. Since it is already embedded in an event-driven framework, it is described in the next section.

4.4 Live Programming Environments

We noted a number of other potential live, graphical music-oriented programming environments described on the Internet. Cycling'89's Max and Max/MSP [13] products look extremely promising, but we only had Windows platform computers, for which they have not yet released their software.

The Reaktor software package from Native Instruments [5] is another live, graphical musical programming environment, but tends to be more focused in the audio synthesis domain, rather than the manipulation of MIDI events. Whilst a very flexible and powerful transformation tool, Reaktor's use for us would have been complicated by its lack of the musical editing views provided by software sequencers such as Cubase or Logic.

We have been interested to see many large software vendors recently proposing component-based development architectures. Although most developers have not yet fully embraced such programming ideals, we feel that the domain of MIDI event processing and performance is particularly suitable to component-based design; indeed the environments in Logic employ its internal component-based toolkit.

As mentioned above, we are concerned that the real-time properties (for example its garbage collection strategy) of Java are not particularly desirable for performance. Such concerns aside, Sun's JavaBeans Component architecture for Java [7] may provide a useful framework in which to develop dynamic MIDI machines such as our performance environment. At the moment no MIDI-related components are listed in Sun's

JavaBeans on-line directory, but their Bean Builder application may indeed eventually surpass the functionality of the Logic environment through extensibility. We feel the most likely problems with JavaBeans design will be the Bean Builder interface being polluted by unnecessary component details, or possibly that the change in and out of the design mode will be too disruptive to allow the MIDI event handling to continue while the application is being extended or modified.

5 Extensions

This section describes a number of our future plans, based on where we were not able to to work around the restrictions imposed on us.

5.1 Live Recording Elements

The biggest limitation we faced was that we were not able to record passages for looping back during our set. For example, we might improvise an eight bar chord pattern via a sequence of piano key presses in an octave we programmed to set the transposition root note of certain streams. Ideally, the software environment would support elements with memory which would maintain and repeat this eight bar pattern until further input was received to override it. Our performances only had the ability to playback prerecorded patterns, although as mentioned above these patterns could span granularity from a simple rhythm up to a complete musical phrase.

The Logic environment would potentially permit the programming of a very simple step-wise live-recording sequencer, but the amount of extra environment design work required would be massive, and rather redundant given the touch-tracks node type is already offering a read-only version of the service we want, plus the code to handle recording is already implemented to run the conventional Logic sequencer time-line.

Note that such an element would not merely provide the ability to live-record and loop musical patterns; more importantly the control stream events which affect the sequencer environment itself would be able to be recorded. Uses might include recording a list of chord sequences which can be mapped and unmapped to any stream, as mentioned above, or indeed changing the timbre or other parameters of a particular synthesiser in a periodic manner.

5.2 Unified Design and Performance Contexts

Generally, when we had started a particular performance set, we would not make any changes to the design of our environment. Whilst one would not normally expect to make modifications, we found that certain changes within the Logic environment would unsettle other events passing through the system. Ideally, like many of the other live music or audio programming products discussed above, there would be no distinction between design and performance environments. Placing new objects within the sequencer programming should not affect the current live function of the sequencer at all.

A further related area of interest is in optimisation. Because we only had limited internal control over the MIDI message patterns being played by the touch-tracks elements, we sometimes needed to carry unnecessary information into the sequencer. For

example, one of our touch-tracks elements might play a particular stream in all 16 MIDI channels, when in fact all but one of these channels would be filtered out before reaching the sequencer's MIDI output device. An interesting challenge would be to design the sequencer so that it could 'push-back' such filter events as early as possible within the sequencer programming, thus reducing unnecessary MIDI message handling. Generally due to the cheapness of CPU cycles, this area has not been explored. In computer networks, however, packet handling can have very high costs. We are currently working with a research group in publish/subscribe middleware (see [8]) with a view to possibly employing such technologies as composite event detection and subscription spanning trees for live, collaborative performance arts.

5.3 Integration of Other Performance Media

Our future plans include performances with greater integration of music, lighting, and video media (where present). Most DJs have their music performances enhanced by lighting and/or video displays. In the same manner as for our discussion of DJs above, however, there is often only a token connection between these different performance areas.

Instead of having three people controlling the music, the lighting and video projections separately, we are interested in programming performance environments where, in the manner of our collaborative performances, these three separate people together control one sequencer, which then itself splits back out to control separate synthesiser, lighting and video equipment. This integration is intended to lift the lighting operator, say, from the level of control required to get lighting effects basically in time with the music, to deciding at a much higher, and more artistic level, how different sections of music will relate to the activity of the lighting fixtures. Naturally override controls would still be accessible at a device level for emergencies. Normally, however, the timing aspects would be left to the centralised event sequencer, since it already needs to understand and process the temporal context with respect to notes, beats, bars and phrases for the music output.

6 Our Live Sequencer-Based Collaboration Feature Checklist

Given that we were not able to satisfy all our needs with the software available, we felt we should indicate what our requirements would be:

Live Performance Speed. Many packages induce delays when incorporating collaborative input. We require that delays are imperceivable to live performers.

Unified Design and Performance Control. It is important that the musical collaboration environment make few distinctions between the performance and design phases. A performer should be able to modify the environment while another performer is playing it with little or no disruption.

Graphical Event Flow Views. We found it highly intuitive to program our performance machines in a graphical environment where events could be traced through an interconnected network of processing nodes.

Music Specific Interfaces. Many component programming environments do not yet provide convenient support for management of musical patterns (or score), audio, and the consequent temporal synchronisation and arrangement of these sequence blocks. Such interfaces are necessary to facilitate rapid composition.

Interface Design and Modularisation. It is highly useful to support encapsulation of processing networks into higher-level modules. These modules need to have user-programmable parameter interfaces.

7 Conclusion

This paper discussed modern popular electronic music performance, and suggested that developing live control environments around music sequencers is a promising new area of ensemble-based musicianship. We presented an overview of the performance environment used by the small Sydney group 'Ambiguous Live' formed in part by the author. A number of alternative current technologies potentially supporting the development of such environments are discussed. Finally, we propose a number of future directions in which we want to develop both our performances, and research into the music technology that will power them.

References

1. 3DWorld. Blurred beats - ambiguous live. `http://www.threedworld.com.au/`, March 2001.
2. MIDI Manufacturers Association. *The Complete MIDI 1.0 Detailed Specification*. MIDI Manufacturers Association, 1996.
3. Richard Charles Boulanger, editor. *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing,and Programming*. MIT Press, 2000.
4. Emagic. Logic audio. `http://www.emagic.de/products/ls/ls/index.php`.
5. Native Instruments. Reaktor. `http://www.nativeinstruments.de/index.php?reaktor_us`.
6. Bill Joy et al., editors. *Java*™ *Language Specification*. Addison-Wesley, second edition, June 2000.
7. Sun Microsystems. JavaBeans™ API specification. `http://java.sun.com/products/javabeans/docs/beans.101.pdf`.
8. Peter R. Pietzuch and Jean M. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. Proceedings of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02), July 2002.
9. Propellerhead Software. Reason. `http://www.propellerheads.se/products/reason/`.
10. Rocket Network Inc. The RocketControl™ online audio collaboration tool. `http://www.rocketnetwork.com/`.
11. Synapse Audio Software. Orion. `http://www.synapse-audio.com/products.php`.
12. Steinberg. Cubase. `http://cubase.net/`.
13. Todd Winkler. *Composing Interactive Music: Techniques and Ideas Using Max*. MIT Press, 2001.