

ImprovisationBuilder: improvisation as conversation

William Walker, Kurt Hebel, Salvatore Martirano, Carla Scaletti

CERL Sound Group & School of Music / University of Illinois
252 Engineering Research Laboratory / 103 S. Mathews / Urbana IL 61801-2977 / USA
Telephone: (217) 333-0766 Email: walker@cs.uiuc.edu

ABSTRACT

To participate in musical improvisations, an interacting system must both generate appropriate musical materials and express those materials appropriately in collaboration with other performers. We have used results from the study of conversation and discourse to identify the major components for a theory of improvisation—listening, composing, and realizing. Our goal is a framework based on a model of musical improvisation as conversation that incorporates signal level as well as event level control of sound.

2. BACKGROUND

2.1. Sal-Mar Construction and SAL

The Sal-Mar Construction (Franco 1974, Martirano 1971) was an interdisciplinary project involving Salvatore Martirano, computer science graduate student Sergio Franco, and ILLIAC III designers Rich Borovec and James Divilbiss. It was based on the idea of “zoomable” control—being able to apply the same controls at any level, from the micro-structure of individual timbres to the macro-structure of an entire musical composition. Weighing in at 1500 pounds, the Sal-Mar Construction provided digital control over analog synthesis modules through a unique touch panel consisting of banks of switches assignable to any level of control.

The YahaSALmaMAC orchestra features MIDI synthesizers under control of the Sound and Logic (SAL) program, which was implemented in LeLisp on the Apple Macintosh by Salvatore Martirano and David Tscheng. Sound and Logic participates in performances by transforming gestures played by the human performers into new gestures. Using the Macintosh keyboard, the human performer can cause the computer to perform looping or change orchestration, intervening in otherwise automated processes (as with the Sal-Mar Construction). A major impetus for the current project was the desire to include timbral control in an improvisation system—a combination of the best features of SAL and the Sal-Mar Construction.

2.2. Kyma System

Kyma is a sound specification language that does not distinguish between signal level and event level processing or between the concepts of orchestra and score; these models are supplanted by arbitrary hierarchical structures constructed by the composer out of uniform Sound objects (Scaletti 1987, 1991). Kyma Sounds are generated in real-time by the Capybara, a digital signal multiprocessor. A Macintosh driver for controlling the Capybara enables any program to play a Kyma Sound and to control its parameters in real time.

2.3. Improvisation as Conversation

While computers that converse are still years away, systems that interact in musical performances exist today (see, for example, Rowe 1991). Many have used linguistic techniques in the representation of musical knowledge; the logical next step is to consider musical interaction as being analogous to language interaction, or conversation. Hartmann describes one manifestation of this analogy—“trading fours”—in jazz performance (Hartmann 1991):

“A four measure phrase leaves the player enough room (say, three to six seconds) to develop one idea, to make one statement; yet there is no mistaking the dialogue within which each statement takes its place, and often the musicians answer each other directly. The resemblance to conversation is uncanny.”

Conversation and musical improvisation are at once similar and different. For example, they seem to differ in their degree of simultaneity; improvisation entails that the participants “talk” at the same time, while most models of conversation assume strictly alternating speakers. However, overlap often occurs in real conversations. Much of the simultaneous activity during improvisation is akin to conversational overlap; a pianist accompanying a soloist is like a listener acknowledging and

encouraging a speaker. Conversation seems to lack the strict temporal structure of harmonic rhythm that often supports group jazz improvisation. However, the timing of conversational overlap is crucial; a conversational participant who never (or always) interrupts other speakers will not fully participate. In addition to timing issues, realizing speech materials in a conversation requires control over timbre as well as timing. For example, a sarcastic tone can turn the phrase “Yeah, yeah” from an affirmation into a rejection. The improviser relies on timbral control and variation as much as rhythmic and melodic development.

By analogy with conversation, an improvising system should have four properties. First, it should listen to the musical environment in which it finds itself. Second, it should generate musical material which relates to that environment. Third, it should realize these musical materials with timing that displays awareness of the other performers. Fourth, it should employ timbral control as a coherent part of realizing musical materials.

3. THE IMPROVISATIONBUILDER PROGRAM

3.1. Plan

One constraint in this project is that at every stage of development, ImprovisationBuilder must remain usable by a composer in actual performance and concerts. In order to achieve this, we have devised a four step plan:

- (i) Translate SAL into Smalltalk-80 and add to it some compositional features of the Sal-Mar Construction
- (ii) Expand the Smalltalk-80 version to include control of a real-time signal processor
- (iii) Generalize and expand upon the specifics of SAL, culminating in a full conversation-based framework for improvisation
- (iv) Expand support for other input and output devices

3.2. Progress Report

As of this writing we are largely finished with (i) and making considerable progress on (ii). The framework supports four basic kinds of activity, corresponding to the four requirements for improvising systems set forth above. *Listeners* process the incoming music, parsing it into phrases and focusing the system’s attention. *Players* create new phrases, either by transforming phrases supplied by the listener or via some compositional algorithm. *Players* could also supply previously composed phrases. *Realizers* attempt to express these phrases appropriately, both through timely presentation and timbral control.

ImprovisationBuilder is written in ParcPlace Smalltalk-80 on the Apple Macintosh. Smalltalk primitives connect ImprovisationBuilder with input and output devices. Connections to MIDI devices are handled by the Apple MIDI Manager, while connections to the Symbolic Sound Capybara are handled by a driver that controls its operation (see Figure 1). A *MusicScheduler* dispatches all MIDI and Capybara events, decoupling the *Players* and *Listeners* from playback timing.

Music encompasses many different time scales. In the short term, onset times must be governed to within a few milliseconds, while longer term structures like blues choruses may last tens of seconds. These processes require radically different amounts of computation, all to be satisfied simultaneously. Careful use of round-robin and priority scheduling allows the graphical user interface, ImprovisationBuilder, and Smalltalk-80’s garbage collector to share the same computer.

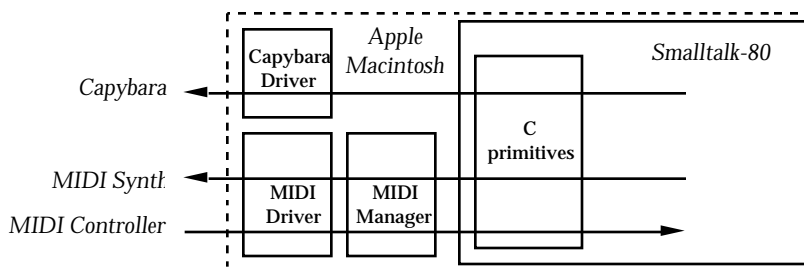


Figure 1: connections between Smalltalk-80 and music hardware

We achieved our first goal of translating SAL by building Smalltalk-80 classes that models each component. A *Listener* filters incoming MIDI events and maintains a buffer containing the most recent notes played by the human performer. A *FancyPlayer* periodically copies this buffer and selects two to four excerpts from it. These excerpts are subjected to a set of transformations and then sent to the *MusicScheduler* for playback. At present, the set of available transformations includes transposition, contour inversion, retrograde, and decimation. More sophisticated transformations include a chord voicer that provides a continuum of voicings from closed to open and a Markov chain that generates new material based on the distribution of existing transitions.

Transformed excerpts from the *FancyPlayer* are also placed in an output queue, where they are consumed by a *SimplePlayer* that further transforms them before playing them. Both Players perform playback via their own *Orchestration*, which chooses a new, random subset of *Timbres* every few notes. A *MusicControlPanel* allows the human performer to exercise control over which transformations are used and what their parameters are. By connecting these objects (see Figure 2) we can either simulate the existing Sound and Logic architecture or explore new configurations.

Object-oriented programming techniques have simplified development of these components. *FancyPlayer* and *SimplePlayer* are subclasses of *Player*, an abstract class that captures similarities between the two. Only the differences between them (the different sets of transformations, for example) need be programmed. The transformations are all subclasses of *Transformation*, and observe the same protocol, so they can be used interchangeably. New transformations are easily developed and tested by substituting them for the default ones.

We have made significant progress on our second goal of control of timbral micro-structure through control of real-time software synthesis. There is a direct, high speed connection between the host computer and a digital signal processor, the Capybara. The composer designs timbres using the Kyma System. ImprovisationBuilder can control any parameter of a Kyma Sound during performance.

Our starting point for connecting Kyma to ImprovisationBuilder was a phase modulation primitive written by K. Hebel (Hebel 1991). This primitive allows the Capybara to synthesize timbres similar to the ones Martirano used with the YahaSALmaMAC orchestra. Having done this, we can proceed to design new kinds of timbres for improvisation. In addition to facilitating timbral control, Kyma can also process other instruments under ImprovisationBuilder control through use of the Capybara's analog-to-digital converter.

The *MusicScheduler* has been expanded to handle data intended for both MIDI synthesizers and the Capybara. Since *MIDIMessage* and *KymaMessage* are both subclasses of *MusicMessage*, the two data streams can be merged together. The *Players* compile phrases into linked lists of messages. These lists are merged into the scheduler's pending events queue in chronological order. Since the *Players* are working several seconds into the future, merging a newly created linked list is often simply a matter of appending it to the end of the pending events queue, a significant computational savings.

3.3. Future

Work is proceeding on the latter half of the four step plan outlined in Section 4.1. Step (iii) is to turn ImprovisationBuilder into a full, conversation-based framework for improvisation. At present, the SAL implementation in Smalltalk-80 is a largely one-sided conversation. Musicality emerges when the human performer learns to adapt to an improvisation partner who interrupts rather often and sometimes focuses on insignificant parts of the human's performance. An important step will be the construction of a *Realizer* that avoids interrupting other performers. We will also investigate controlling timbral parameters and how they can be incorporated into improvisation.

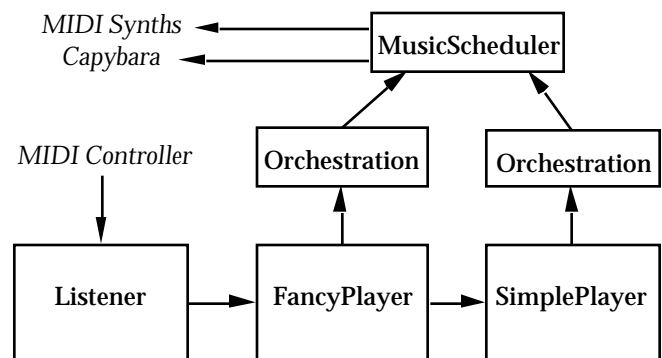


Figure 2: one possible ImprovisationBuilder configuration

Step (iv) is to expand support for other input and output devices. A benefit of proper object-oriented programming is that our framework can manipulate any kind of input or output event that observes a fairly small set of protocols. Thus, other kinds of devices can be incorporated into the performance environment and used within an improvisation. This would include, for example, videotape players, lighting equipment, and pressure sensitive controllers like the Continuum Music Keyboard (Haken et al, 1992).

4. CONCLUSION

Our goal is to create a conversation-based framework for musical improvisation. We have translated the Sound and Logic program into Smalltalk-80 (see Figure 3) and added some of the compositional features of the Sal-Mar construction. As work continues, we will explore further the analogy between conversation and improvisation and its ramifications for ImprovisationBuilder.

5. ACKNOWLEDGMENTS

This research is supported in part by a grant from the University of Illinois Research Board.

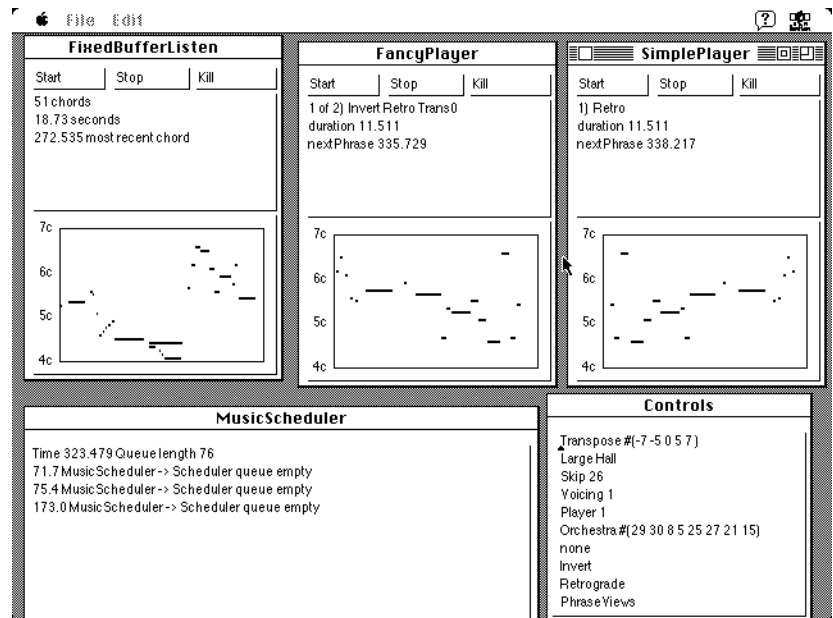


Figure 3: ImprovisationBuilder in action

6. REFERENCES

- S. Franco. *Hardware design of a real-time musical system*. Ph.D dissertation, University of Illinois, 1974.
- L. Haken, R. Abdullah, M. Smart. "The Continuum: a continuous music keyboard." in *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association, 1992.
- C. O. Hartmann. *Jazz Text: Voice and Improvisation in Poetry, Jazz and Song*. Princeton University Press, 1991.
- K. Hebel. "A Framework for Developing Signal Processing and Synthesis Algorithms for the Motorola 56001." in *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association, 1991.
- S. Martirano. *An Electronic Music Instrument which combines the Composing Process with Performance in Real-Time*. Unpublished. 1971.
- R. Rowe. "Machine Listening and Composing with Cypher." *Computer Music Journal*, Vol. 16, No. 1, 1992.
- C. Scaletti. "Kyma: An Object-oriented Language for Music Composition." in *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association, 1987.
- C. Scaletti. "The Kyma/Platypus Computer Music Workstation" in *The Well-tempered Object: Musical Applications of Object-oriented Programming*, Stephen Pope editor, MIT Press, 1991.