January 1996

Applying ImprovisationBuilder to Interactive Composition with MIDI Piano

William Walker

Brian Belet
San Jose State University, brian.belet@sjsu.edu

Recommended Citation

William Walker and Brian Belet. "Applying ImprovisationBuilder to Interactive Composition with MIDI Piano" *Proceedings of the 1996 International Computer Music Conference* (1996): 386-389.

Applying ImprovisationBuilder to Interactive Composition with MIDI Piano

William Walker

Research Scientist
Advanced Technology Group
Apple Computer, Inc.
walker@apple.com

Brian Belet

Assistant Professor
School of Music
San Jose State University
bbelet@sjsuvm1.sjsu.edu

Abstract

In this presentation we will show how the ImprovisationBuilder framework was adapted to our compositions for pianists and computers. Software design issues include: capturing, transforming, and realizing MIDI piano data, and an easy-to-use graphical interface for altering parameters during performance. Compositional issues include: the causal responsorial aspect of the ImprovisationBuilder output, and varying the small-scale event and gestural levels while preserving the large-scale structure among multiple performances. The presentation will discuss the compositional and performance importance of these issues and show how each of these issues was handled within the ImprovisationBuilder framework.

Work on this project suggests the advantages of addressing compositional and software design issues simultaneously and cooperatively. Both composer and software designer can benefit from this collaboration.

1. Introduction

We first collaborated on ImprovisationBuilder (IB) during a computer music workshop in 1993, a collaboration we resumed in 1994. Our initial efforts were concentrated into an intense full-time two week period. Belet had composed some material for a pianist and wanted to design an IB configuration that transform the material in interesting and appropriate ways. This material formed the basis of Belet's *(Disturbed) Radiance* [Belet, 1994].

We worked together to create some new software components, such as one for creating trills from sustained single notes and another for adding short ornamentation notes to the attack of single notes. As we wrote and debugged these algorithms together, Belet gained a greater understanding of IB's musical representation, while Walker came to understand how Belet intended these musical transformations to relate to the music he had composed for the pianist.

This foundation helped us to communicate when we resumed work in 1994, culminating in our premiere of *Cross-Town Traffic*, a piece for two improvising pianists and IB in November of 1995. Each pianist's improvisation is transformed by a copy of IB. The transformed material is realized on the other pianists' piano while he is playing. While each of us used a distinct IB configuration, our mutual understanding of the implementation and musical consequences of each other's IB configuration greatly aided our improvisation.

2. ImprovisationBuilder

ImprovisationBuilder[Walker, 1994; Walker, Hebel, Martirano, & Scaletti, 1992] is a framework for building computer improvisors. It is written in ParcPlace ObjectWorks/Smalltalk-80 on the Apple Macintosh. Smalltalk-80 primitives written in C connect IB to the Yamaha Disklavier and other MIDI devices. MIDI connections are handled by the Apple MIDI Manager, which sends data through the Macintosh serial port to a MIDI interface.

The IB framework provides Smalltalk-80 classes corresponding to the tasks performed by improvising systems. *Listeners* process the incoming music, parsing it into phrases and focusing the system's attention. *Transformers* and *Composers* create new phrases, either by transforming phrases captured by the listener or by some compositional algorithm. *Realizers* express musical ideas appropriately, both through real-time presentation and by controlling *Timbres* that represent sound generating hardware. *Improvisors* monitor the information gathered by *Listeners* and use it to track progress through a map of the shared musical structure.

By providing a class hierarchy of components for improvising systems, IB facilitates rapid testing and prototyping through the combination of existing components [Johnson & Foote, 1988]. Users can extend the framework by adding their own components to it. These new components reuse code provided by their superclasses and only implement the unique aspects of their behavior. IB also provides a standard representation of musical events for use by all components.

ImprovisorComponents are combined in a linear chain, with a *Listener* at one end and a *Timbre* at the other. This chain of components is contained within an *Improvisor*, along with all the information the computer uses to improvise. The following sections describe the components of the framework.

2.1. Listener

IB's input components parse a stream of low-level *MusicMessages* into a stream of musically useful *Phrases*. This process involves two steps. First, a *ChordStream* processes the *MusicMessages* from the *MIDIInputChannel* into *Chords*. The *ChordStream* creates a new Chord for each new *MusicMessage*. Since the *ChordStream* performs a straightforward conversion between representations, it is reused in all IB configurations. Second, a *Listener* groups the *Chords* from the *ChordStream* into *Phrases* based on the *Chords'* pitches and durations.

Two particular *Listeners* have seen the most use in our work with the Yamaha Disklavier. A *PauseListener* detects silences longer than two seconds and begins a new *Phrase* after each pause, placing each incoming *Chord* into exactly one *Phrase*. In contrast, a *FixedBufferListener* stores the incoming *Chords* in a fixed size buffer and converts the buffer's contents into a *Phrase* when the system demands a *Phrase* to operate on. If a *FixedBufferListener* receives too many *Chords* before producing a *Phrase*, it discards the oldest *Chords*.

2.2. Transformers and Composers

Having parsed input from the other performers, the system's next task is to generate its own musical contribution. IB generates music with *Composers*, *Transformers*, or a combination of the two. *Composers* contain a musical algorithm that will create new *Phrases*. *Transformers* transform the output of a *Listener*, *Composer*, or another *Transformer*. Varying the parameters of *Composers* and *Transformers* can tailor them to specific musical situations. The human performer sets these parameters during performance by means of computer keyboard commands or through a control panel on the computer screen (see Figure 1). *Composers* and *Transformers* can also derive their parameters from the *Improvisor's PolicyDictionary*, a repository for shared information about the improvisation.

Sal Martirano's Sound and Logic was the inspiration for the first *Transformers* in IB, including both

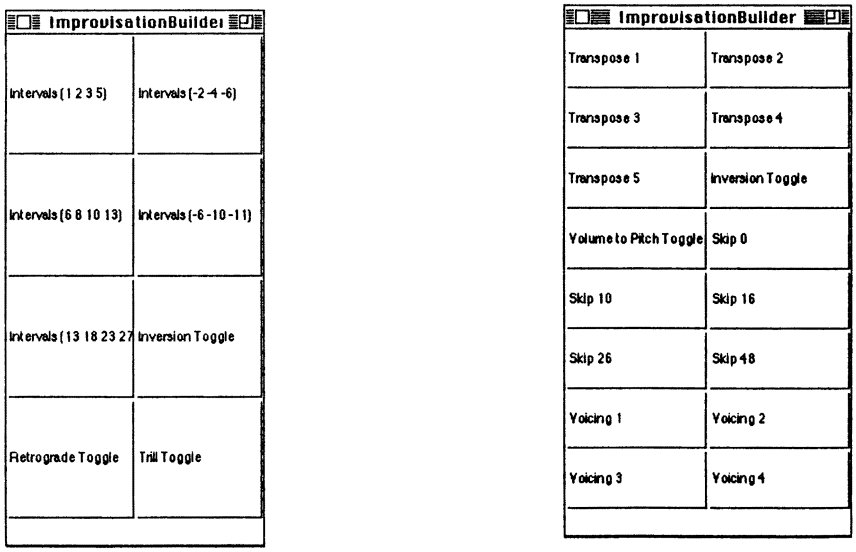

Figure 1. Belet's on-screen buttons (left) turn several processes on and off and control the pitches used for trills and attack harmonization, while Walker's buttons (right) govern the intervals of transposition, cyclic permutations for voicing, and percentage of note omission.

standard techniques, such as *Retrograde* and *Transpose*, and special-purpose techniques, such as *Excerpt*, a scheme for generating short excepts from an input buffer, and *Voicing*, his cyclical permutation scheme for revoicing chords [Martirano, 1971; Martirano, 1988; Martirano, 1995]. The second set of *Transformers*, inspired by Belet's compositional ideas, include *AttackHarmonize*, which adds transient notes to the attack of single notes, and *Trill*, which turns sustained notes into trills of various intervals.

Composers create new *Phrases* by means of compositional algorithms. The existing *Composer* classes implement a variety of algorithms. *HarmonyGenerator* uses a library of chord voicings and rhythmic templates to create accompaniment or solo parts for a given set of chord changes. It serves as the basis for participating in jazz improvisation. *RandomPhraseSource* generates completely random Phrases, which are used for testing purposes. *DrumPatternBuilder* uses the same rhythmic templates as *HarmonyGenerator* to produce drum patterns with random variations. *TransitionTable* uses Markov chains to create material.

3. Compositional Issues

The output data generated by IB is responsorial by design (as described above). At any given moment, the IB output is based on material the pianist has performed earlier. This delay creates a composition (and performance improvisation) parameter that is simultaneously a physical restriction and an aesthetic focus. This restriction is neither positive nor negative; rather, it is part of the overall context in which this music exists, and it contributes directly to the composition process and the performing attitude.

"Interactive" electro-acoustic music has been the hot buzzword in our field for several years, and it therefore requires qualification for each specific use. For compositions generated using IB in this context "performance interaction" is defined as a commentary by the computer on the input piano music data. It is a compositional restriction created for the specific aesthetic needs of our work, and is not offered as the only approach for all compositional needs. The performance model of small ensemble jazz improvisation is relevant to this paradigm. As one layer of music is performed by one player, the remaining ensemble members listen to and then respond to this music in their subsequent responsorial improvisations. Here the IB buffer is the analog to listening to the first music, and the IB algorithmic processing and output is the analog to the improvised response. This process can continue as the performer in turn listens to the IB output and then responds to it with additional improvisation, and the process can continue from performer to machine to performer, closing the cycle of improvisational feedback. The process can be significantly expanded when two pianos are used with two computers each running IB. In our composition *Cross-town Traffic* the output of one performer's IB was used as the input to the other performer's piano, and vice versa. This creates a complex set of responsorial relationships as one performer is able to respond to the music generated directly by the other performer as well as to the music being generated on both pianos by cross-related IB outputs.

A MIDI grand piano is used as the live performing instrument so that MIDI information may be used as input data for processing while the regular acoustic piano sound is heard directly. Our work has addressed composing and performing using either one or two Yamaha Disklaviers. The Disklavier has been fertile ground for other composers interested in extending traditional piano performance practice and compositional models through computer interaction [Bolzinger, 1992; Risset & Duyne, 1996].

As this interactive, responsorial layer is governed by probabilities and the random element within the *Excerpt* operation, the resulting music is different for each performance on the event and phrase levels. The structural level remains constant and deterministically in agreement with the source acoustic piano music (this assumes that the performers are working with a set structure for successive performances of a given composition so that the input to IB essentially steers the output towards the intended structure). As a result, a great deal of event and phrase level diversity is achieved from performance to performance, while unity of structure and design is preserved.

4. Conclusions

Much of software development for interactive computer music follows one of two patterns. In the first pattern the programmer is also the composer. Software development is guided by the composer's own aesthetics, and software design feedback is largely introspective. Such systems are likely to fit very closely to the composer's conceptual model and working style, offering high productivity and usability. However, such systems rarely become sufficiently general to serve a larger audience.

In the second pattern, the composer builds a system from off-the-shelf software packages. These systems face a difficult trade-off: they either offer low-level, "aesthetically neutral" constructs and services (requiring considerable effort on the composer's part), or they contain musical assumptions (some of which may not coincide with the composer's vision).

IB succeeds largely because of an iterative, collaborative design process. Believing that contrived, sterile laboratory conditions offer no real context for testing the usability or utility of an artifact, our goal is to get working prototypes into the hands of real users as early and as frequently as possible. By observing how these prototypes succeed or fail, we learn valuable lessons that inform the next prototype.

Our intention as composers and performers is to explore and augment the ensemble improvisation process. Using IB has proved to be an aesthetically successful means of pursuing this goal.

References

[Belet, B., 1994] Integrating Real-time Interactive Software Synthesis, Pre-processed Resynthesis, MIDI data, and Acoustic Piano in Composition and Live Performance. In S. D. Beck (Ed.), *Proceedings of the Society for Electro-Acoustic Music in the United States*, (pp. 16–17). Middlebury, Vermont: SEAMUS.

[Bolzinger, S., 1992] DKompose: A Package for Interactive Composition in the Max Environment, Adapted to the Acoustic MIDI Disklavier Piano. In *Proceedings of the International Computer Music Conference*, (pp. 162–165). San Jose: International Computer Music Association.

[Johnson, R., & Foote, B., 1988] Designing Reusable Classes. *Journal of Object-Oriented Programming, 1*(2), 22—35.

[Martirano, S., 1971] An Electronic Music Instrument which combines the Composing Process with Performance in Real-Time. University of Illinois at Urbana-Champaign: unpublished.

[Martirano, S., 1988] Everything Goes When the Whistle Blows. Baton Rouge: Centaur Records.

[Martirano, S., 1995] A Salvatore Martirano Retrospective: 1962–1992. Baton Rouge: Centaur Records.

[Risset, J.-C., & Duyne, S. V., 1996] Real-Time Performance Interaction with a Computer-Controlled Acoustic Piano. *Computer Music Journal, 20*(1), 62–75.

[Walker, W., 1994] *A Conversation-based Framework for Musical Improvisation*. Doctoral Dissertation, University of Illinois at Urbana-Champaign.

[Walker, W., Hebel, K., Martirano, S., & Scaletti, C., 1992] ImprovisationBuilder: Improvisation as Conversation. In *Proceedings of the International Computer Music Conference*, . San Jose: International Computer Music Association.