NRCI: SOFTWARE TOOLS FOR LAPTOP ENSEMBLE

Christopher Burns and *Greg Surges*
University of Wisconsin-Milwaukee
Department of Music
{cburns, gssurges}@uwm.edu

ABSTRACT

NRCI is a suite of Pd tools designed to facilitate laptop ensemble performance, and to foster a culture of group music-making which emphasizes custom software design and improvisational practice. NRCI provides a large library of abstractions with a consistent interface; these modules ease the challenge of instrument design for novices, and aid in rapid development and live coding for advanced users. NRCI also implements a network protocol (using OSC) which provides for the wireless exchange of control data and messaging in both improvisational and pre-structured situations. NRCI is designed for and tested by MiLO (the Milwaukee Laptop Orchestra), and is freely available online.

1. INTRODUCTION: THE LAPTOP ENSEMBLE

The laptop orchestra is an idea of the moment in the computer music community. Ensembles have recently appeared at Carnegie Mellon, Cincinnati College of Music, the Moscow Conservatory, and Stanford, among other institutions; PLOrk is perhaps the best-documented instance of the phenomenon right now [1]. Laptop ensembles are appealing as a pedagogical opportunity, creating a focused learning community and a "studio culture" at a time when electroacoustic music is increasingly decentralized. They also offer extraordinary creative opportunities; as Dan Trueman notes, "simply imagining how [the laptop ensemble] might work, what kind of music might be composed for it, and what it would be like to play in is inspiring" [2].

Other recent trends in electroacoustic music-making intersect neatly with the model of the laptop ensemble. Improvisation and live coding practices fit naturally into multiple-laptop performance, with each performer pursuing independent activity [3, 4]. Interdisciplinary practices also integrate well the laptop orchestra; technology-oriented examples include motion capture for dance, and computer-assisted visual performance and projection. (Low-tech solutions like spoken word and instrumental performance are equally effective). And with ubiquitous wireless networking, network music feels "native" to the laptop ensemble [5, 6, 7].

Our ensemble, the Milwaukee Laptop Orchestra (MiLO), represents a particular perspective on these phenomena. MiLO's fundamental practice is free improvisation. Each performer brings their own approach (including field recording, a variety of synthesis aesthetics, and live processing of audio) and their own tools (from cepstral.com and ElectroPlankton through DJay and Digital Performer to SuperCollider and Pd). Instrumental performance is welcome; one member plays piano and saxophones exclusively, and several double on wind instruments. Interdisciplinary approaches are commonplace, with several members of the group self-identifying as visual artists or filmmakers, and video projections (including visuals made with GEM, Isadora, Processing, and vvvv) are frequent in performance. The group is non-hierarchical and consensus-driven, combining students, faculty, and non-university-affiliated members. This principle means that MiLO has to be particularly mindful about creating and maintaining a positive culture within the group.

MiLO is the initial target audience and testing group for our new software, NRCI (Networked Resources for Collaborative Improvisation). NRCI is designed to support the group and its creative trajectory in several ways, serving as a platform for experiment, as a learning tool, and as a form of acculturation into the ensemble.

2. NRCI DESIGN OBJECTIVES

NRCI is intended to serve at least three distinct groups within MiLO: novice members, advanced users, and the NRCI developers (the authors of this paper). For new members, NRCI is designed to facilitate rapid integration and socialization into the ensemble. NRCI provides both a friendly welcome to custom software design for novice users, and near-instant gratification which motivates more advanced learning and design. Our intention is to maximize NRCI's creative potential while emphasizing simplicity and ease-of-use.

For more experienced users, the NRCI library facilitates rapid application development, providing modular, reusable, and modifiable tools for a variety of common tasks. These tools offer built-in networking capabilities, so that participation in the NRCI network protocol is "free" with the use of the library. For the developers, NRCI serves as an experimental testbed; it represents our initial forays into both live coding and network music. NRCI's ease of use makes it a highly hackable modular environment for on-the-fly coding, while the networking functionality allows us to explore the creative possibilities of network music in improvisational and compositional contexts.

Finally, NRCI is in no way intended to be "the" MiLO software. Each member of the group can decide for themselves if, when, and how to use NRCI (though some structured improvisational designs may require the networking component). Accordingly, NRCI must *invite* use to be a success in the context of the group. Because Pd is the most widely used software in MiLO, it was an obvious choice of platform for NRCI, and its open-source nature fits nicely with our objectives [8]. NRCI is

intended to be as aesthetically open-ended as possible, with a minimum of musical preconceptions.

3. THE NRCI LIBRARY

NRCI provides a library of interoperable abstractions, all designed to facilitate the construction of software instruments. There are five categories of abstractions: interface elements, timing generators, control data generators, audio generators, and audio processors. In general, these abstractions are arranged in sequence - timing generators drive control generators, which provide parameter data for audio generators, which provide input for audio processors, with interface elements entering the chain at a variety of points (Figure 1). A large part of the NRCI workspace (the top-level Pd patch provided with the code) is an empty canvas for pre-performance or on-the-fly assembly of these abstractions into custom instrument designs.

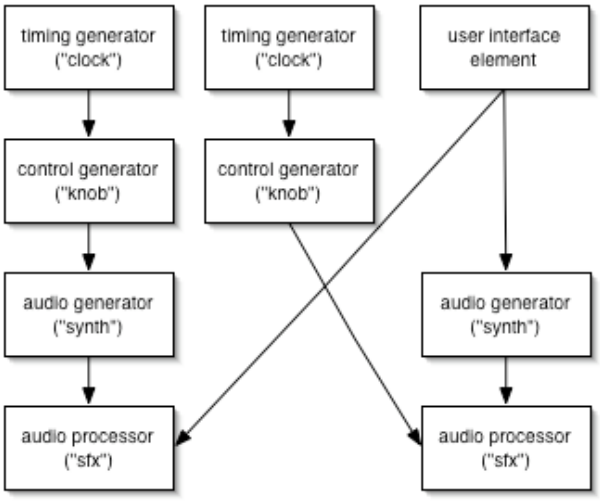

Figure 1. Possible I/O configurations of the abstraction types.

3.1. Library elements

User interface elements include keyboard, mouse, and networking controls. There are two abstractions facilitating keyboard input, *ui-fretboard-number* and *ui-fretboard-pitch*. Inspired by the *Lattice* improvisation instrument and the SMELT toolkit, these abstractions treat the QWERTY keyboard as ordered rows or columns of keys [9, 10]. The *ui-fretboard-number* abstraction maps keyboard input to numeric output from 2.5 to 100; the *ui-fretboard-pitch* abstraction maps keys to MIDI pitch values. A creation argument provides for transpositions, and the *fretboard* abstractions can easily be modified for other types of scales and mappings.

Where the keyboard abstractions each provide a single data stream based on keypresses, the *ui-mouse* abstraction provides three simultaneous data streams. Two of these map the mouse pointer x- and y-axes to data values between 0 and 100; a third outlet outputs a bang message when the first mouse button is clicked.

The third category of user interface abstractions provides access to the NRCI network interface. *Net-request-handler* facilitates the request of control data from other users on the network, and *net-request-out* and *net-command-receive* import the resulting streams of network data into the user's instrument. A fourth abstraction, *net-ui-chat*, allows a user to send text messages to other performers participating in a local-area NRCI network. These objects are described in greater depth elsewhere [11].

A variety of timing generators ("clocks") are provided in the NRCI library. Deterministic strategies include periodic interonset values, linearly or geometrically increasing values (when an upper threshold is crossed, the next time is reset to a lower threshold), linearly/geometrically decreasing values, and values oscillating around a specified center, with a user-specified frequency and waveform (sinusoidal or triangular). Stochastic strategies include random timing values (from a variety of distributions), periodic values with random shifts of interonset time, and randomly shifting subdivisions of a periodic base tempo.

Control data generators ("knobs") are generally similar to the "clocks". Deterministic strategies include constant output, linear and geometric output (as with the clocks), and sinusoidal and triangular oscillators. Stochastic strategies include randomly chosen values (again from a variety of distributions), "heap" (randomly choosing one of a small number of randomly generated values, with occasional substitutions of new values into the collection), and "cycle" (cycling through a short sequence of randomly generated values, with occasional substitutions of new values into the sequence). Control messages are generated "on demand" according to timing messages; the control data output stream maintains the inter-event timing of the "clock" input.

Audio generators ("synths") implemented to date include sine and FM oscillators, Karplus-Strong plucked string synthesis, pulse-train and pulsed-noise generators, granular synthesis, a flute model, and sample playback.

The last category encompasses audio processing abstractions ("sfx"). Amplitude modifiers include gating and enveloping; modulation types include FM, ring modulation, and feedback FM; filter categories include all basic types as well as comb filtering with and without random feedback variations; "effects" include reverb, delay (with and without random feedback variations), and waveshaping distortion. There are two more idiosyncratic audio processing abstractions: *sfx-scrub~* sweeps non-linearly through a delayed input signal, and *sfx-feedback~* implements a simple feedback network [12]. Finally, there are two output routings: *sfx-output~* streams data to the computer audio interface, while *sfx-record~* streams audio to a hard disk recorder.

3.2. Abstraction interfaces

The relationships between library elements are tightly controlled, to ensure ease-of-use especially for novices and in live coding situations. All clocks output timed sequences of bangs. All knobs receive a clock input, and output one control data stream, with values between 0 and 100. (All UI elements except *ui-fretboard-pitch* also output values between 0 and 100). This control data is interpretable as pitch (in the floating-point microtonal MIDI representation native to Pd), as a percentage (as in feedback coefficients), or as arbitrary parameter data; all

control data inputs across the library (synths and sfx) are set up to parse a 0-100 value. All synths take one control input and provide one audio output; control data is usually (though not always) interpreted as a frequency parameter. And all sfx take one audio input and one control input, and produce one audio output. (The "output" subclass is exceptional, in that they receive audio streams but don't output them). Because there are a wide variety of signal processing algorithms implemented in the sfx abstractions, input control data is interpreted on a per-module basis.

While the interfaces between modules imply one-to-one connections (for instance, from clock output to knob input), many-to-one connections are also possible. Multiple timing streams may be interleaved by connecting them to a common (timing-aware) inlet. This strategy allows for extremely complex rhythms, while still maintaining the simplicity of the individual abstractions. The same approach can be used with control data and audio streams (audio streams are mixed, rather than interleaved). Streams can also be manipulated using standard Pd objects and techniques; as always, the complete functionality of Pd is available.

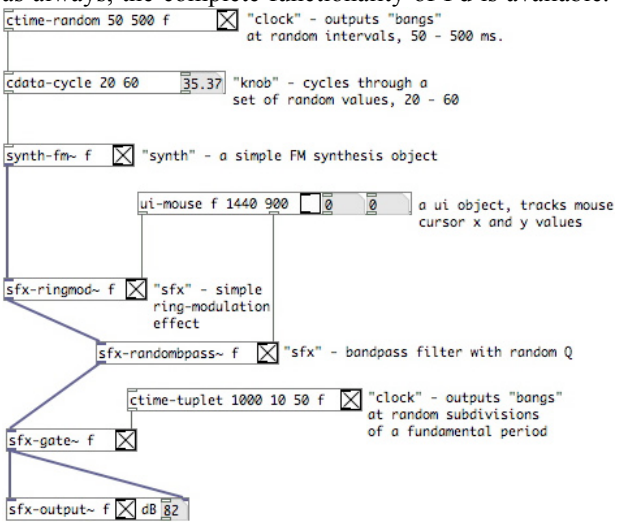

Figure 2. Example configuration of library elements.

NRCI abstractions are designed to appear and behave consistently according to their category (Figure 2). Timing generators have one inlet (an on/off switch), one outlet (a stream of bang messages representing timing events), and one graph-on-parent interface element (an on/off switch). Control data generators have one inlet (a stream of timing messages), one outlet (a stream of numeric values), and one interface element (display of the most recent output). Audio generators have two inlets (a stream of control values, and an on/off switch), one outlet (an audio stream), and one interface element (an on/off switch). Audio processors have two inlets (one for audio input, and one for a control data stream), one outlet (the processed audio stream), and one interface element (an on/off switch). The only exceptions are *synth-playback~*, *synth-grain~* and *sfx-record~*, which have a "choose soundfiles" button, and *sfx-output~*, which adds a volume control.

NRCI abstractions expose additional functionality through creation arguments. Most abstractions can be assigned to switch on and off via a user-specified hotkey. Other creation-time parameters are specific to the particular abstraction; for example, *sfx-combfilter~* exposes control of feedback coefficent. Each abstraction includes comments describing all available inlets, outlets, and creation arguments. While the number of abstractions makes memorization of creation arguments challenging, the documentation mitigates this issue.

Interface consistency extends to the NRCI network implementation. All clock objects report duration and onset data to the networking module for broadcast; similarly, all synth and sfx abstractions which interpret control input as pitch (oscillators, filter frequencies, etc.) report pitch data. The *sfx-output~* module passes audio output to an RMS calculation, which then passes amplitude data to the network module. Every NRCI user connected to a local NRCI network provides data to the network "request" protocol (described below) without any additional user action or coding.

4. THE NRCI NETWORK PROTOCOL

NRCI supports network music applications using a protocol implemented in Open Sound Control over UDP [13]. The network uses an all-client, broadcast design, with a central (802.11g wireless) router but no server.

The NRCI protocol provides four discrete sets of functionality. The first is a request-driven system for the exchange of control data. Performers can request streams of pitch, amplitude, and rhythmic data from other musicians on the network, and the requested data is automatically provided by NRCI, without any action on the part of the data provider. The interface is designed to encourage transformation of this data (rather than simple re-use), and this part of the protocol targets free improvisation and live coding.

The second system is similar to the first, but is controlled by the transmitter rather than the receiver. This "command" protocol benefits Hub-style network compositions in particular. Where the request protocol is limited to specific data types, the command protocol leverages the flexibility of OSC to allow any kind of data; however, message formats must be negotiated (in advance, or on the fly) between sender and receiver.

The third set of functionality is a ensemble-wide VU metering system; each laptop performer can visualize the RMS amplitude of all group members. This system helps the ensemble to manage the dissociation between physical gesture and sound characteristic of laptop music; the VU meters aid in identifying relationships between performers and sounds.

Finally, the NRCI protocol provides a chat system, for text communication, coordination, and introspection between NRCI users during performance.

To increase the utility of the network protocol, we have created an implementation in Processing; Jason Nanna has contributed an implementation for vvvv. The protocol is documented in greater detail in [11].

5. DOCUMENTATION

Documentation is a critical part of the NRCI project. Comments facilitate the joint development of the code;

clearly and uniformly commented code is also a source of assistance and "memory" while live-coding. Most importantly, documentation advances the pedagogical purposes of the project, demonstrating to new users how the abstractions interconnect, how Pd code is written, and how fundamental processes of digital audio work.

The NRCI user manual augments the documentation of the code, providing initial instructions for setting up Pd, running the code, and establishing a username for networking. It offers a brief introduction to the function of each abstraction in the library, and explains the connection paradigms between the types of abstractions. The manual serves to guide new members of MiLO, and to reinforce the culture of the group.

6. EVALUATION AND FUTURE WORK

MiLO has been working with beta versions of NRCI since November 2007, with four public performances to date. While assessment of the project is necessarily subjective, we feel that our key design goals were successfully realized. The fulcrum point of the project is the balance of simplicity and flexibility; while NRCI is not appropriate for every purpose, we feel that it invites participation and experimentation.

Newcomers to MiLO, to Pd, and to software design are able to use NRCI with a minimum of coaching. The most vivid example of NRCI's approachability comes from an April 2008 workshop at Lawrence University; with less than an hour of demonstration and hands-on exploration, ten undergraduates (who had never used Pd before) were all joyfully creating NRCI instruments and improvising together.

More advanced users in MiLO have been eager to take advantage of the NRCI features. While MiLO will undoubtedly continue to use a variety of software, NRCI has seen substantial use in our latest performances, and there has been a surge of plans for new NRCI-based structured improvisations from within the group. The authors have worked extensively with NRCI live coding in rehearsal and performance. Our subjective experience is that the simplicity and consistency of the NRCI library (coupled with the full expressive range of Pd) maps well to facile and flexible live coding.

We have evaluated the network protocol in part by recreating and performing classic Hub compositions using NRCI. MiLO has performed *Vague Notions of Lost Textures* and *Stuck Note* (both by Scot Gresham-Lancaster), stressing and demonstrating the value of the chat and command protocols in particular. The request protocol has seen use primarily in free improvisation, where it facilitates types of pitch and rhythmic relation unusual to laptop improvisation. This system models a kind of hyper-detailed listening, augmenting the performer's mental model of the music in progress with precise data about other users' activity. Finally, chat has proven a fascinating augmentation of inter-ensemble communication (though not a replacement for purely musical means). In a long improvisation, chat can facilitate complex processes of formal negotiation. Several members of MiLO have found it worthwhile to run Pd and NRCI in performance alongside other applications, solely for access to the chat function!

The network protocol and modular design are now stable; we expect that future releases of NRCI will emphasize the addition of new abstractions to the library, and improvements to the documentation. Additional future work includes tools for the statistical analysis of network traffic, additional implementations of the network protocol, and new compositions and improvisational specifications based on the software.

NRCI source code and documentation are freely available from http://ccrma.stanford.edu/~cburns/NRCI.

7. REFERENCES

[1] Trueman, D., et. al. "PLOrk: the Princeton Laptop Orchestra, Year 1", *ICMC Proceedings*, New Orleans, USA, 2006.

[2] Trueman, D. "Why A Laptop Orchestra?" *Organised Sound* 12/2: 171-179.

[3] Collins, N. et. al. "Live Coding in Laptop Performance", *Organised Sound* 8/3: 321-330.

[4] Wang, G., and Cook, P. "On-the-fly Programming: Using Code as an Expressive Musical Instrument", *NIME Proceedings*, Hamamatsu, Japan, 2004.

[5] Brown, C., and Bischoff, J. "INDIGENOUS TO THE NET: Early Network Music Bands in the San Francisco Bay Area." http://crossfade.walkerart.org/brownbischoff/IndigenoustotheNetPrint.html

[6] Gresham-Lancaster, S. "The Aesthetics and History of the Hub: The Effects of Changing Technology on Network Music", *Leonardo Music Journal* 8: 39-44.

[7] Kane, B. "Aesthetic Problems of Net Music", *Spark Proceedings*, Minneapolis, USA, 2007.

[8] Puckette, M. "Pure Data", *ICMC Proceedings*, Hong Kong, China, 1996.

[9] Burns, C. "*Lattice*: Strategies for and against control in a laptop instrument", *ICMC Proceedings*, Barcelona, Spain, 2005.

[10] Fiebrink, R. et. al. "Don't Forget the Laptop: Using Native Input Capabilities for Expressive Musical Control", *NIME Proceedings*, New York, USA, 2007.

[11] Surges, G., and Burns, C. "Networking Infrastructure for Collaborative Laptop Improvisation", *Spark Festival Proceedings*, Minneapolis, USA, 2008.

[12] Burns, C. "Emergent Behavior from Idiosyncratic Feedback Networks", *ICMC Proceedings*, Singapore, 2003.

[13] Wright, M. "Open Sound Control, an Enabling Technology for Musical Networking", *Organised Sound* 10/3: 193-200.