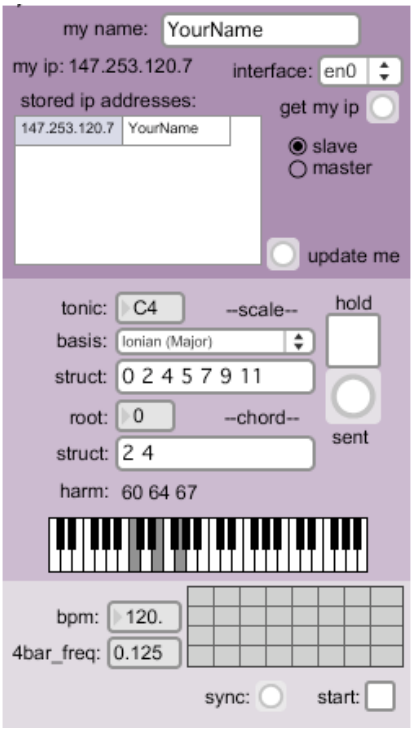
The MPG Carepackage: coordinating collective improvisation in Max/MSP
Extended Abstract - Submission for SEAMUS 2010
Nathan Wolek, Ph.D. - Stetson University - DeLand, FL

Mobile Performance Group is a collaborative, multimedia project involving faculty and students from Stetson University's Digital Arts program. The group's primary mission is to find new ways of presenting art outside of traditional venues. Since 2004, MPG has fulfilled this objective by presenting a number of site-specific performances at festivals and conferences throughout the United States. These performances are built on the practice of collective improvisation and use audio and video materials that are gathered throughout the host city in the days leading up the event. In order to ensure that our music is coordinated along such common parameters as tonic, pitch set and rhythmic timing, the author has developed a series of patches in Max/MSP that allow a single set of data to be easily shared over a network. Members of the group are able to build instruments that react to these messages and adapt the sonic output accordingly. In this way, multiple laptop musicians are able to perform a collective improvisation that is unified in its musical character while they design instruments that allow them to control such parameters as timbral quality, density and texture with individual freedom. This presentation will review the major components of the MPG Carepackage system, demonstrate the system in use on multiple machines networked via WiFi and advise interested parties on how to adapt the system for their own use.



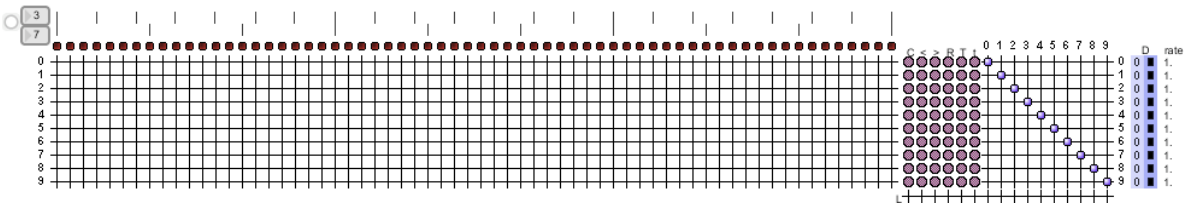
The MPG Carepackage consists of two primary patches: *musiclinks* and *riddumbank*. The *musiclinks* patch (pictured on the right) is used to share musical controls over the network between multiple computers during a performance. The functions of the patch are divided into 3 parts by color. The first section is shaded deep purple and handles matters of network connectivity. Users must first type a unique name into a text field at the very top in order be identified within the system. The user's name and IP address are broadcast using the max.nethole java class to all other computers listening with the same *musiclinks* patch and is stored by every one locally. At a basic level, this facilitates messaging between machines. Once *musiclinks* is embedded within another patch, Max messages can be prepended with one of these unique names and sent into the proper inlet to be directed at a corresponding machine. A list of stored IP addresses and names is displayed for easy reference by the user and can be used as a means of verifying that the automatic connections between the machines have indeed been completed successfully. The IP addresses are also used to direct the CNMAT Open Sound Control objects in their task of passing all other messages to those registered within the network. Using the OSC objects for the more critical musical data ensures

better timing than what is provided via the networking features of Max's java implementation.

Each member must choose between the slave and master modes of operation. When in master mode, the musical information contained in other sections of the patch is broadcast over the network for others to receive. Whenever users are testing a patch without being networked to others, they should place *musiclinks* in master mode to ensure that changes are broadcast to their own patch. Logically, the slave mode allows a user's patch to receive these messages and react willingly. Users must coordinate amongst themselves who will act as the master and who will operate in slave mode. There is nothing currently in the system to prevent multiple users from setting their interface to master mode. The musical effects of this "multiple master" situation can be interesting, but are usually undesirable. If nothing else, this potential conflict should encourage the musicians to discuss and cooperate on who will take the lead during each segment of a performance.

The medium purple section of *musiclinks* handles information related to pitch via five interconnected musical parameters: tonic, scale_basis, scale_struct, chord_root, and chord_struct. The tonic is expressed as a pitch class with octave, where C4 corresponds to middle C on a piano, and is used as the basis for all other pitch controls. The scale_basis provides a list of 35 preformed scales that have been adapted from David Cope's book, *Techniques of the Contemporary Composer* (page 27). Once a selection has been made from this menu, the list of pitches is loaded into the scale_struct text field where each scale member is displayed as half-steps above the tonic. Here further modifications can be made, such as deleting pitches, adding pitches or converting pitches from integer to floating-point numbers. Next, the chord_root defines which index within the scale structure should be used as the basis for the prevailing chord for harmonic emphasis. Finally, the chord_struct defines which indices above or below the root should be used to define the prevailing chord. A keyboard slider is used to provide a visual display of the defined chord once the necessary information has been entered.

The third, light purple section handles tempo information. The shared rhythmic cycle is controlled within the MSP domain. Through much trial and error, the author has found that this is the most reliable method to ensure uniform timing across machines in Max/MSP. The tempo is expressed in beats per minute and controls the frequency of a *[phasor~]*, which is calculated based on the assumption that the *[phasor~]* should complete one cycle for every four measures of common time. Although this assumption may not always match the application of the rhythmic cycle on all machines within the system, it provides a more intuitive control than setting the frequency of the *[phasor~]* directly would. Each machine in the system has it's own independent *[phasor~]*. These are kept phase-locked through a resync message that is broadcast once per cycle and more frequent updates of the tempo as it changes during musical gestures such as an accelerando.

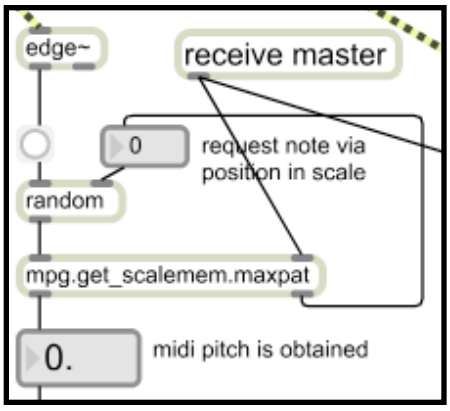


The control signal from the synchronized *[phasor~]* objects is used to drive the *riddumbank* patch (pictured at the top of page). When embedded in a host patch, the outlets from *riddumbank* produce a pulse signal output. Ten distinct rhythmic patterns can be entered on the rows of the main 64-step grid with green dots denoting an active cell. The current position within the rhythmic cycle is displayed by a series of red dots across the top that light up in succession to provide visual feedback of the cycle's progress. When the loop passes a pattern position that has been activated with a green dot, the MSP signal goes to 1.0; otherwise, the signal remains at 0.0. This signal can be easily used to control such objects as *[adsr~]* and provide enveloping for synthesis or sample triggering. The 10 rows do not correspond directly to the 10 outlets of *riddumbank*. Instead they can be patched in and out using the matrix on the right with purple dots denoting a made connection. This allows for the rhythmic patterns' destinations to be reconfigured enabling such actions as disabling the output of patterns selectively, directing a single row to be fed to multiple outlets or combining multiple patterns to be fed to a single outlet.

Several controls are added to these basic features of pattern entry and output patching. Across the top of the patch are a series of vertical ticks that allow the user to quickly see groupings that are of interest. The number of cells within a grouping can be changed using the number boxes in the upper-left hand corner of the patch. Between the pattern entry grid and the output patching martix, are a series of buttons that allow for quick permutations of the rhythmic patterns in a given row. The letter at the top of the column denotes which operation the button performs (notated in the table to the right). On the right edge of the patch are two final controls. The first is two numbers beneath the letter D that activate a "destablizing" algorithm which will cause the patch to dynamically choose random cells from the rhythmic pattern without playing them all, thereby increasing its variety. Lastly, the rate parameter allows the progress across each row of the grid to be controlled independently, providing the possibility of such effects as half-time (0.5), double time (2.0) and anything in between.

char	description
C	clears the pattern currently in the row, leaving it blank
R	enters a random pattern into the row
<	shift the current pattern one position to the left
>	shift the current pattern one position to the right
T	fill in all ticks visualized at the top of patch
t	fill in a random subset of the ticks from the top of patch

The pitch information from *musiclinks* can be easily accessed use two small helper abstractions: *get_scalemem* and *get_chordmem.* Both receive the shared musical parameters from the network and load the respective pitch sets into arrays for lookup functions. By sending integers to the first inlet, the object will look up the MIDI pitch stored at the corresponding index location and output this value stored. The abstraction also implements an element of circularity to extend the scale and chord pitch arrays up and down 3 octaves so that users can request pitches beyond the octave that the original set was confined to. In this way, the abstractions facilitate algorithmic compositional strategies where pseudo-random number generators and other functions can determine the selection of pitches for individual note events (as seen in the screenshot on the right). The second outlet of these abstractions provides the length of the scale or chord array so that pseudo-random number generators can be properly constrained.



The current set of patches are the result of development dating back to 2003 and will be made available for the first time via the internet before the conference. With this public release and demonstration, it is my hope that others will download the MPG Carepackage and adapt it to their own purposes and utilize it in performances. It should be of special interest to those that direct laptop-based performance ensembles as part of their teaching. The system allows students to rapidly develop patches in Max/MSP that can then be used in group performances. The pedagogical benefits of such deployment are that students become more easily engaged in live performance and the communal activities that they expect from music making. This translates into students that are more motivated to put in the hours of solitude often required to perfect their patch, because they have a clearer concept of what the eventual goal of their work is. The author looks forward to hearing feedback from the electronic music community as others work with and adapt the system to their own purposes.