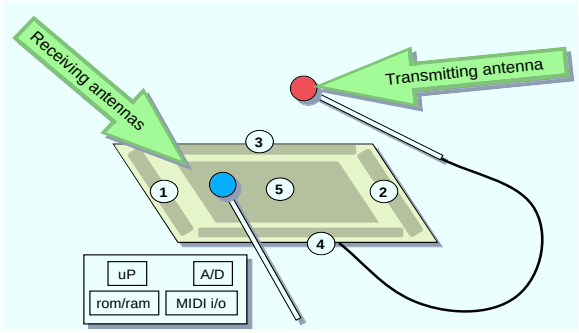
PadMaster: an improvisation environment for real time performance

Fernando Lopez-Lezcano
CCRMA (Center for Computer Research in Music and Acoustics), Stanford University
(nando@ccrma.stanford.edu)

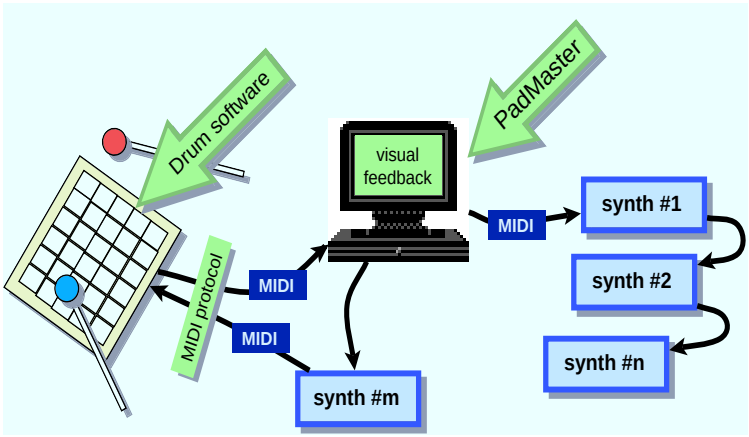
ABSTRACT: This paper will describe the design and implementation of PadMaster, a real-time improvisation environment running under the NextStep operating system. The system currently uses the Mathews/Boie Radio Drum as a three dimensional controller for interaction with the performer. PadMaster splits the surface of the drum into virtual programmable pads which can be grouped into scenes so that the behavior of the surface can be subtly or drastically altered during the performance.

1.0 The Radio Drum and the MIDI communication protocol

The current implementation of the Stanford Radio Drum was developed at CCRMA by Max Mathews as a simpler alternative to Boie's previous design. The two batons act as radio transmitting antennas. The signals are received by five antennas located underneath the surface of the drum. A multiplexed A/D converter translates the received signal strength from each antenna into numbers which are used by the on-board microprocessor to calculate the absolute position of each baton in space. The microprocessor uses this information to track the movement of the batons and to detect hits on the surface. Information about each hit includes both the x-y coordinates and the hit velocity. In addition to the two batons, the Radio Drum hardware includes two switches and four potentiometers. It has a MIDI interface that it can use to communicate with computers or synthesizers.



The behavior of the Drum is defined by the program stored in its EPROM. The drum software includes several functionally different programs that can be externally activated through MIDI System Exclusive messages. Through them, the Radio Drum can act as a stand alone conductor of a score or MIDI file, can improvise with several different options that map baton movement to MIDI commands or can act as a general purpose MIDI controller. This last program and the underlying protocol built on top of MIDI were originally designed by David Jaffe and Andy Schloss. This existing general purpose controller program was completely redesigned by the author. A more efficient and faster protocol was created that uses just one MIDI channel and is more bandwidth efficient in the use of MIDI resources. The protocol was also expanded to allow the controlling computer to upload / download calibration data from / to the Drum. Once the program is activated through a sysex message, the Radio Drum behaves as a three dimensional controller with six degrees of freedom.



Following is a short description of most of the control protocol:

- **System exclusive configuration messages**: can be used to turn

ON or OFF the communication program, set the MIDI channel used by the rest of the protocol, dump and load the internal calibration tables, set the trigger and release heights for both batons, request raw A/D measurements (useful for testing), etc.

- **Trigger / Release messages:** sent by the drum when a baton hits / leaves the surface. Each hit or release is represented by three MIDI controller messages, using continuous controllers 26 through 31. The messages are used to send the x and y positions and velocity of the hit or release.
- **Switches:** a switch message is sent by the drum when one of the two hardware switches changes state. The information is sent through controllers 5E to 5F.
- **Poll request:** sent by the computer to request the position in space of the batons at a given moment. The message uses a channel pressure MIDI message that encodes the required request as a pressure value. The controlling program can thus request the position of one or both batons and can also ask for the current value of the four potentiometers.
- **Poll answer:** sent by the drum in response to a poll request message. The requested information is sent through a string of channel pressure messages. As opposed to the Trigger / Release messages, the Poll Answer message contain no state information, which means that a state machine in the receiver program has to track the incoming messages. While this opens the possibility of garbled information due to lost MIDI bytes it was deemed more important to reduce the bandwidth used by the protocol as this is a frequently used message and the information gathered through it is refreshed periodically.

Planned enhancement to the protocol and underlying Radio Drum library routines include:

- **Detection of hits based on direction reversal**. The current implementation uses a height threshold based detection scheme which cannot reliably detect very fast rolls close to the surface of the drum.
- **Automatic position update**. To further decrease the MIDI bandwidth, the current polling scheme should be replaced with a timer based automatic transmission of the current position (that is, the Drum software should take care of sending periodic position messages). The new scheme will also include a sysex message to change the period of the transmission so as to enable the controlling software to throttle down the sampling rate of the position information when the MIDI stream becomes close to being saturated. This feature is currently implemented by changing the sampling rate of the position request polls.
- **Better internal linearization routines** for the three axes of control.

2.0 The PadMaster program

The PadMaster control code is written in Objective C, using the MusicKit as the foundation class hierarchy for MIDI event scheduling and control. The graphical interface was designed with NeXT's Interface Builder and the program runs on any workstation that supports the NextStep operating system (and has a MIDI driver available). PadMaster is connected through MIDI to the Radio Drum controller and to external synthesizers. The program uses the coordinates of the incoming Trigger and Release messages and an internal calibration map to split the surface of the drum into up to 30 virtual pads. Each pad is independently programmable to react in a specific way to the hit, and to the position information stream of one or more simultaneous axes of control. Pads can be grouped into Scenes, so that the behavior of the surface of the drum can be subtly or radically altered during the course of a performance. This is achieved by dynamically jumping to a different Scene, either through the use of a control pad programmed for that function or through another external controller. The screen of the computer continuously displays a representation of the virtual surface and gives visual feedback to the performer on the state of all the pads in the currently selected Scene.

The virtual pads can be split in two types depending on their function: **Performance** and **Control** pads.

2.1 Performance Pads

Performance Pads can be individually programmed to control the playback of MIDI sequences, note generating algorithms or soundfiles. The graphical representation of the pads on the screen gives instant visual feedback to the performer. Pads change color and status messages dynamically according to their state. A performance pad that is playing remains active even if the performer selects a different Scene, so that chains of events can be started from one Scene and will continue to run even though the performer latter switches to a

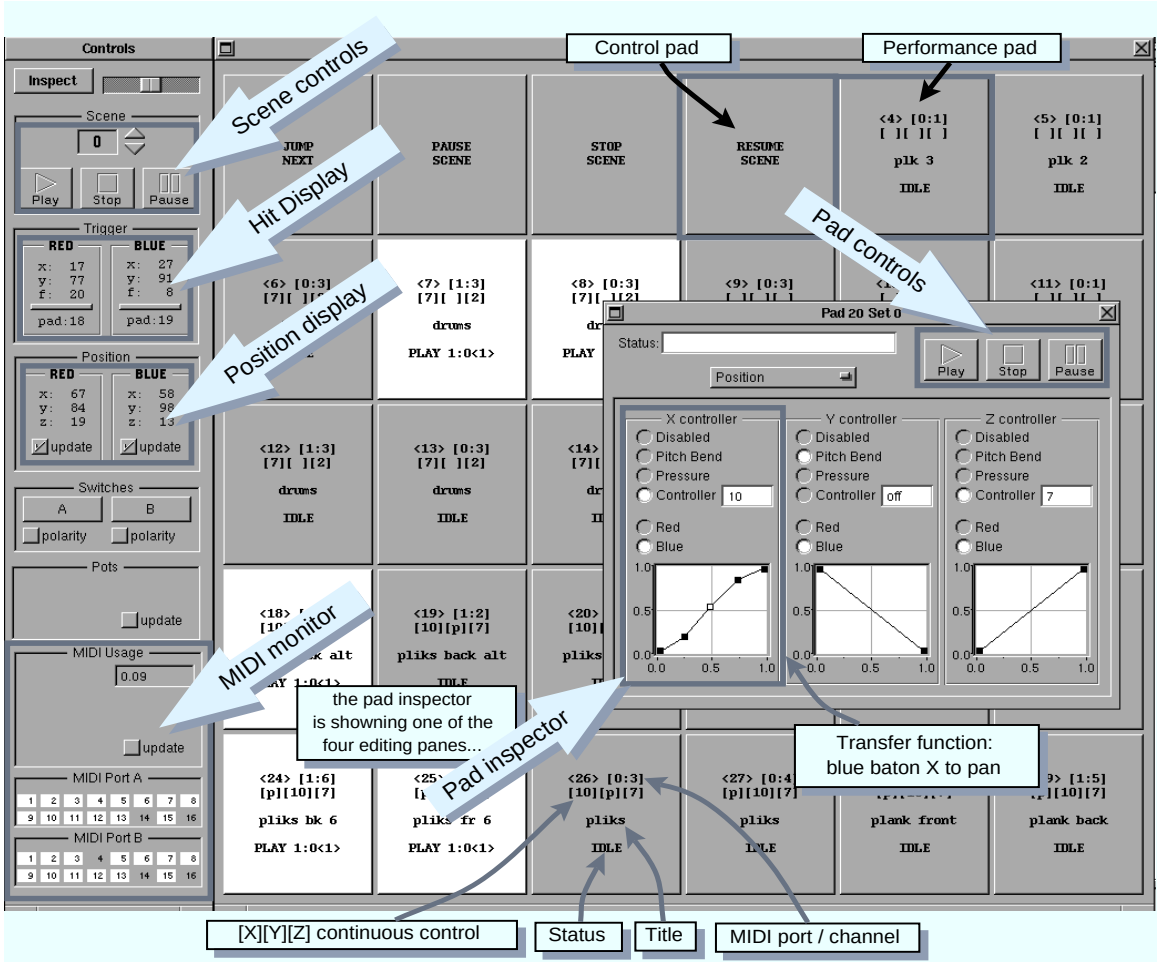
different Scene. The status is updated for all active pads but only those in the currently selected Scene show up on the graphical representation of the drum surface.

2.2 Control Pads

Control Pads are used to trigger actions that globally affect the performance of a Scene. A pad can be programmed to change the current Scene when hit, jumping to the next or previous Scene, thus redefining the behavior of the whole surface of the drum. Control pads can also be used to pause, resume or stop all playing pads in the currently selected Scene.

3. Inside a pad

Editable parameters inside each pad can be changed through a standard NextStep inspector window with several editing panes. The first pane can be used to select the type of pad and, in the case of performance pads, the triggering baton and the action that is executed when the pad is hit. The possible actions include starting / pausing / resuming a sequence, starting a new overlapping sequence, or playing the next note of a list of notes. It also selects the MIDI port, channel and program number that will be used for MIDI transmission and allows editing of a graphical mapping of hit velocity to note velocity for the selected sequence. The second pane edits the tempo options for the pad. Tempo can be global, per pad or per sequence inside a pad (as there can be more than one instance of a sequence playing at the same time). There is a tempo envelope and it is also possible to control tempo with the hit velocity or with any of the six available axes of continuous control. The third pane lets you associate up to three continuous MIDI message streams (pitch bend, channel pressure or any con-



troller) with the position of up to three of the six axes of control. All these function mappings are created through graphical function editors. The fourth pane edits the sequence of notes that are played when the pad is hit. The sequence is expressed as a normal MusicKit text scorefile. The scorefile format has been enhanced with additional tags that represent all programmable parameters in a pad. It is then possible to externally generate a textual representation of a pad and then load it into PadMaster (for example, a set of pads for a performance might be algorithmically generated and then loaded into the program).

4. PadMaster in performance

PadMaster has been used to compose and perform "Espresso Machine", a piece for PadMaster and Radio Drum, two TG77's and processed electronic cello (Chris Chafe playing his celletto). The piece is an environment for improvisation in which the PadMaster and celletto performers exchange ideas and play with predetermined materials. The piece is composed in three PadMaster Scenes, each with several groups of related materials that are triggered during the performance. One baton is reserved for triggering pads and the other for continuous three dimensional control of the currently performing pads.

The performance of this piece on several occasions has raised several issues. The simultaneous mapping for several pads of baton movement to MIDI continuous controllers is one of the most interesting performance capabilities of the program, but also raises the possibility of serious MIDI bandwidth clogging. The current version of PadMaster dynamically adapts to the changing conditions and adjusts the position sampling frequency to try to reduce the bandwidth used when several pads are playing simultaneously. More work needs to be done in measuring MIDI usage in a more precise way to avoid sending too much information, but at the same time to avoid control lag if the sampling frequency fall to a very low value. There is also a measurable gap in playback when scenes are changed, during which MIDI activity is not updated as the MIDI and graphical routines share the same execution thread.

5. Future developments

PadMaster is currently undergoing a complete rewrite to implement new and improved functionality.

Pads will be resizable so that each scene can have different number and size of pads if necessary. We have found that sometimes it would be desirable to concentrate important or critical performance functions in a few big pads. Resizable pads would also allow for linking performance behavior to the place where the pad is hit, something that is not possible in the current version. Another enhancement to pads will be inheritance, so that multiple pads with related behavior (something we have found common and very useful) could be grouped together, with the common functionality being editable as a group.

The action types of performance pads will be enhanced to allow for inclusion of algorithms and soundfile playback. The algorithms will be able to use a well defined API to enable linking of baton movement to algorithm parameters.

The NextStep operating system includes a remarkably easy to use system to communicate with remote objects (objects that live in other computer[s] but are directly linked to the execution of a local program). That opens the possibility of using remote computers connected through Ethernet as servers for MIDI or soundfile playback. There are also several scheduled refinements for performance such as a completely separate thread for all MIDI interaction so that graphics may lag behind the performance but there will be no delays when switching between scenes.

Another important enhancement will be defining a way to use different MIDI controllers in addition or instead of the Radio Drum (percussion controllers, normal keyboards, MIDI pedals, etc).

References:

[1] Max Mathews, *The Stanford Radio Drum, 1990*

[2] David Jaffe, Julius Smith and others, *The MusicKit "http://ccrma-www/CCRMA/Software/MusicKit/MusicKit.html"*

[3] Carlos Cerana (composer) / Adrian Rodriguez (programmer), *MiniMax, a piece for Radio Drum*