Dsp.rack: Laptop-based modular, programmable digital signal processing and mixing for live performance

William Kleinsasser
Towson University
Department of Music
8000 York Road
Baltimore, MD 21252 USA
wkleinsasser@towson.edu

ABSTRACT

This is a demo of software for live signal processing using a model of signal modules, recording buffers, and file playback systems interconnected through a programmable matrix within the Max/MSP environment.

KEYWORDS

Digital signal processing, Max/MSP, computer music performance, matrix routing, programmable live processing.

1. INTRODUCTION

Dsp.rack is a suite of tools that runs in Max/MSP on a Macintosh Powerbook or iBook with 500 mHz or faster cpu. Dsp.rack was designed to model the familiar paradigm of mixer, patch bay, and signal processor rig that became a popular approach to integrating electronic music with live performance during the 1980s and 1990s. Dsp.rack has been developed to take advantage of the familiarity of this paradigm, the decades of performance practice related to it, and the processing speeds of the latest computers. With the relative ease of expandability offered by a software-based system, Dsp.rack integrates the functions of programmable mixing, routing, and processing of audio along with the ability to play overlaid, pre-recorded sound files. Dsp.rack was created with the intention of offering an entry point to composers, performers, students, and teachers as a set of tools that are relatively familiar, flexible, and open-ended.

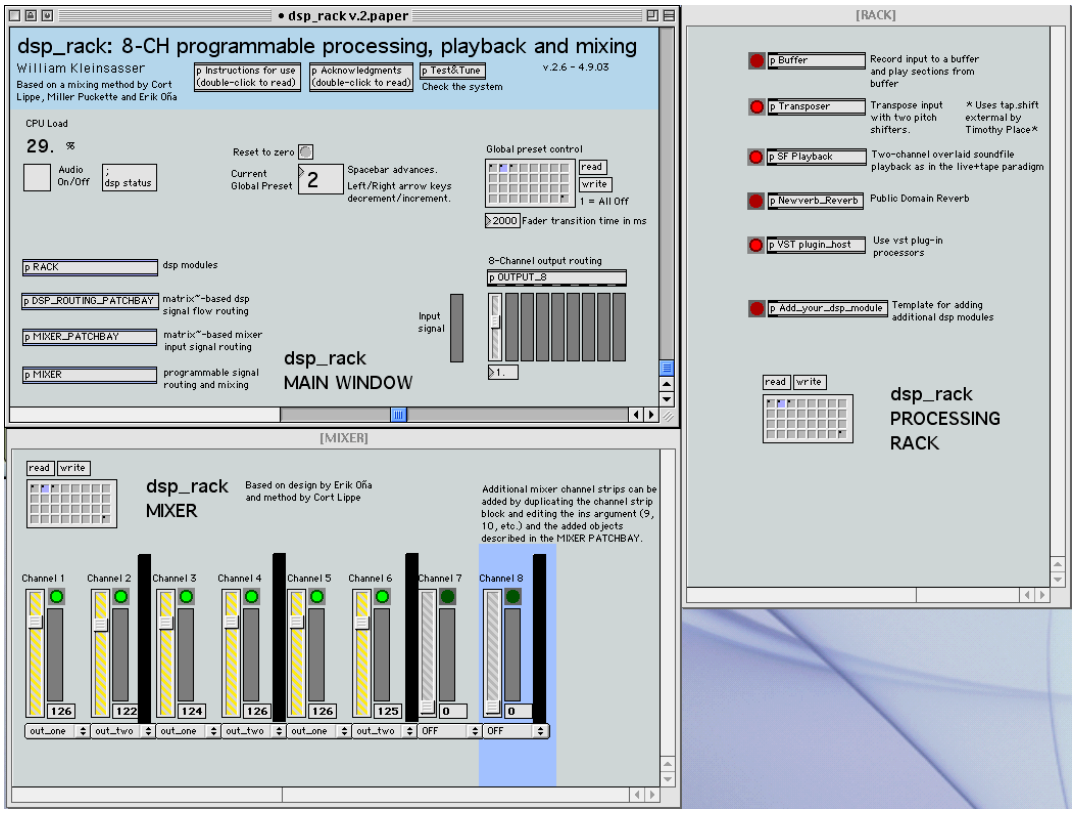


Figure 1. dsp.rack screens

2. THE MODEL

To date, Dsp.rack has undergone two stages of development resulting in two versions which offer beginning and more advanced environments for live signal processing. Dsp.rack version 1 uses a menu-driven crossbar method for routing signals. This version offers a flexible and simple approach to integrating signal input, routing, processing, mixing, and output. Version 2 uses the signal routing matrix objects in Max/MSP which allow for more complex and flexible combinations of signal routings but at the cost of processor load, visual feedback, and ease of learning curve. Having been developed in Max/MSP, Dsp.rack benefits from the open sharing of resources that is part of that environment.

In the spirit of technological ecology, Dsp.rack offers many of the functions of hardware processing and mixing devices but integrates these within a software environment that can be extended and updated without the need to purchase new equipment and is open to the creative additions of each user. It uses commonly-understood models which the intention of easing the learning burden for new users and those new to the field.

A basic set of processing modules is included with the distribution of Dsp.rack and a mini-tutorial on integrating user-designed additional modules is provided. The mixer and patch bay are fully extendable only limited by screen saturation and processing speed of the computer.

Running on a Powerbook with a multi-channel adc/dac i/o converter like the RME Hammerfall, Dsp.rack can support eight or more independent input and output channels. This makes Dsp.rack capable of instrumental ensemble processing with multi-channel output.

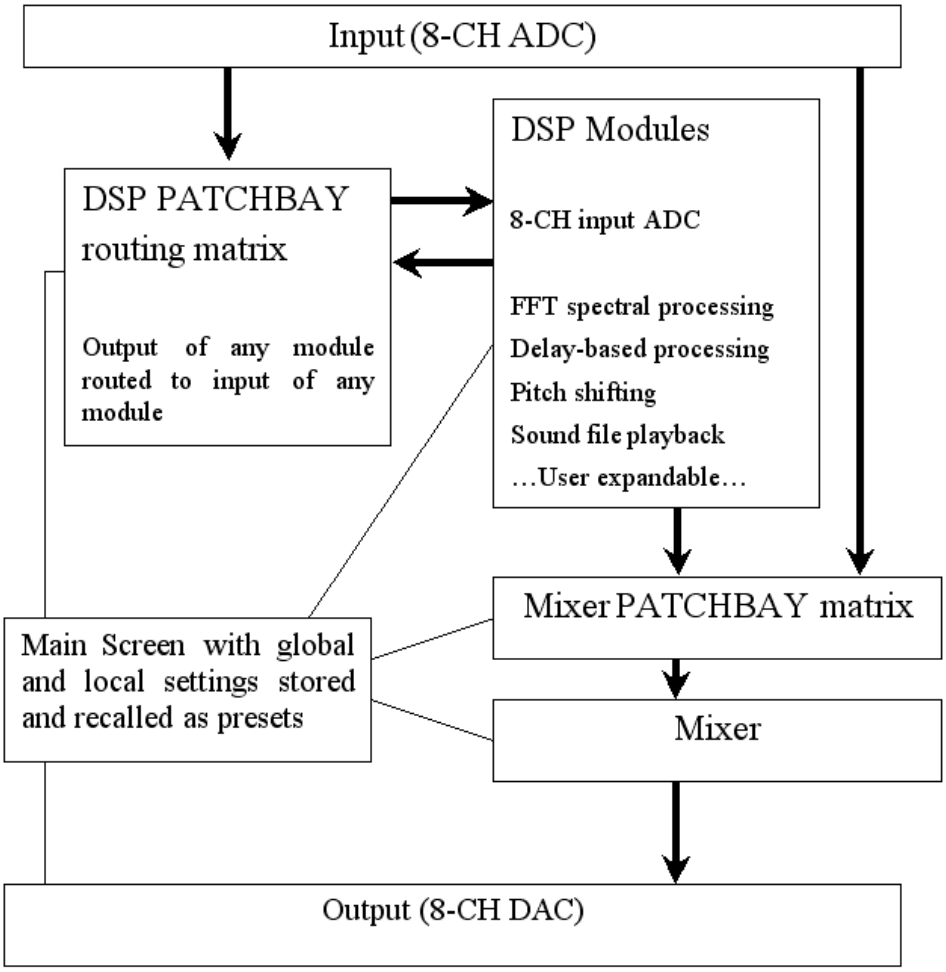


Figure 2. dsp.rack block diagram

3. INS AND OUTS

Dsp.rack supports eight channel input and output. The i/o can be easily modified for fewer than eight channels and with the required i/o hardware, the i/o can be extended to support up to 24 input and output channels. There is a module for playing a pre-recorded sound files as the input source so that Dsp.rack can be used as a studio processor to process recorded files as well as live input.

4. RACK: THE PROCESSING MODULE WINDOW

The RACK window contains the signal processor modules as well as a buffer module that records input to a memory buffer that can be played with access to segment locations, direction, and speed. Version 2 adds a module for VST plug-ins which uses an automatically loaded menu for listing of all VST plug-ins that are in the VstPlugIns folder inside the Max/MSP folder. Additional processing modules can be created and added by the user. Included in the processor rack window is a tutorial module to show how to add user-designed dsp processes to the rack.

The processing rack window can be filled with the processors needed for a given work and all of the settings for each module, mixer and routing definition can be stored as presets and recalled in coordination from a master preset change object in the main window.

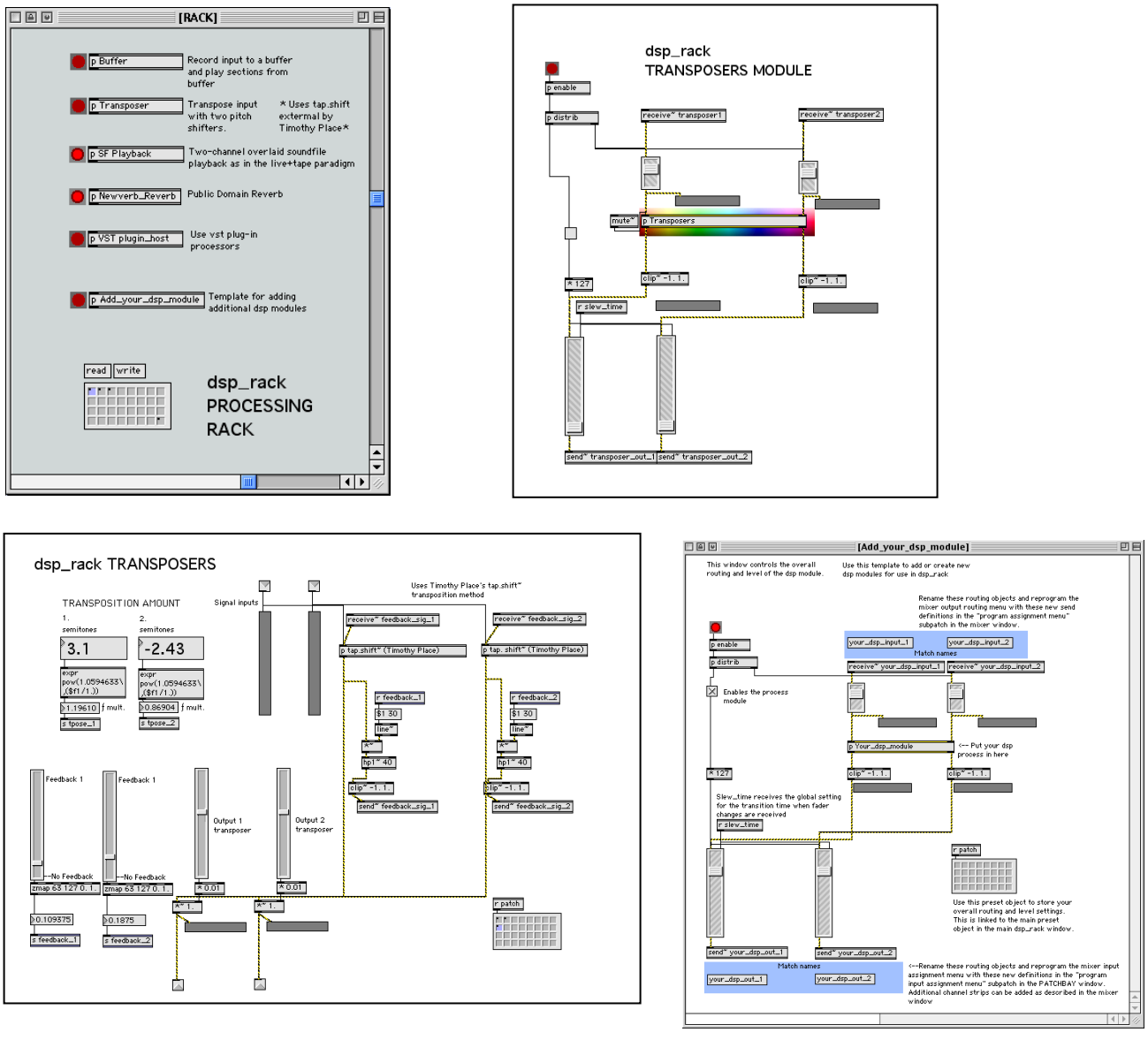


Figure 3. RACK of processors, details of the transposition module, and the user-expansion tutorial module

5. DSP PATCHBAY: ASSIGNMENT OF THE PROCESSING MODULES SIGNAL FLOW

The DSP PATCHBAY window allows for programmable assignments of the inputs to the mixer channel strips. In Version 1, pop-down menus are used to assign the input source for each mixer channel. The number of assignment modules can be extended by duplicating the modules as needed. Additional mixer channel strips can be added to the mixer to accommodate more than 16 input assignments.

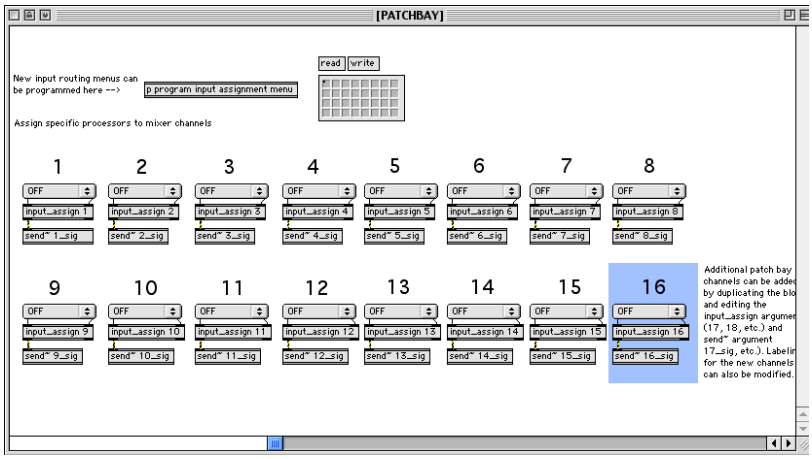


Figure 4. Version 1 DSP PATCHBAY with menu-driven routing

The menus that control the input routing for the mixer channels are easily modified and extended. Using a message box that feeds all of the mixer input routing menus, the list of input assignments can be updated with a click of the mouse.

In version 2, the menu-driven signal routing is replaced with Max/MSP's matrix~ object supporting multiple overlapping signal flow possibilities. DSP modules can now send to and receive from any combination of the other modules simultaneously through the nodes of the matrix~ and matrixctrl objects in the patch bay windows.

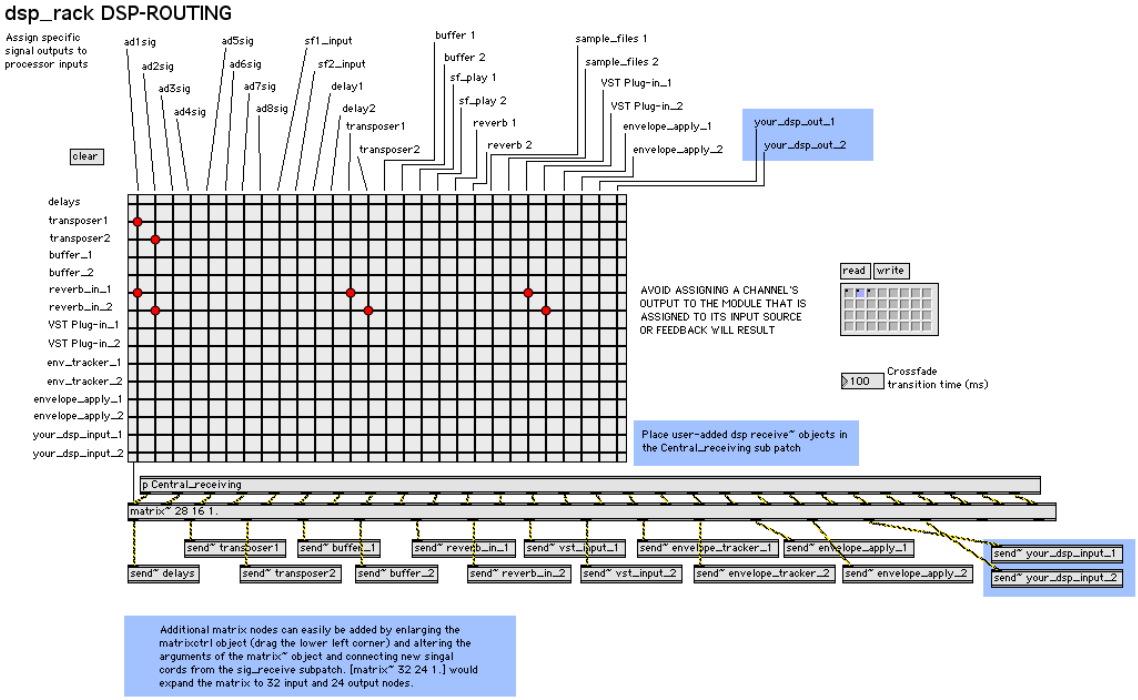


Figure 5. Version 2 matrix-driven routing

6. MIXER: PROGRAMMABLE MIXING AND OUTPUT ROUTING

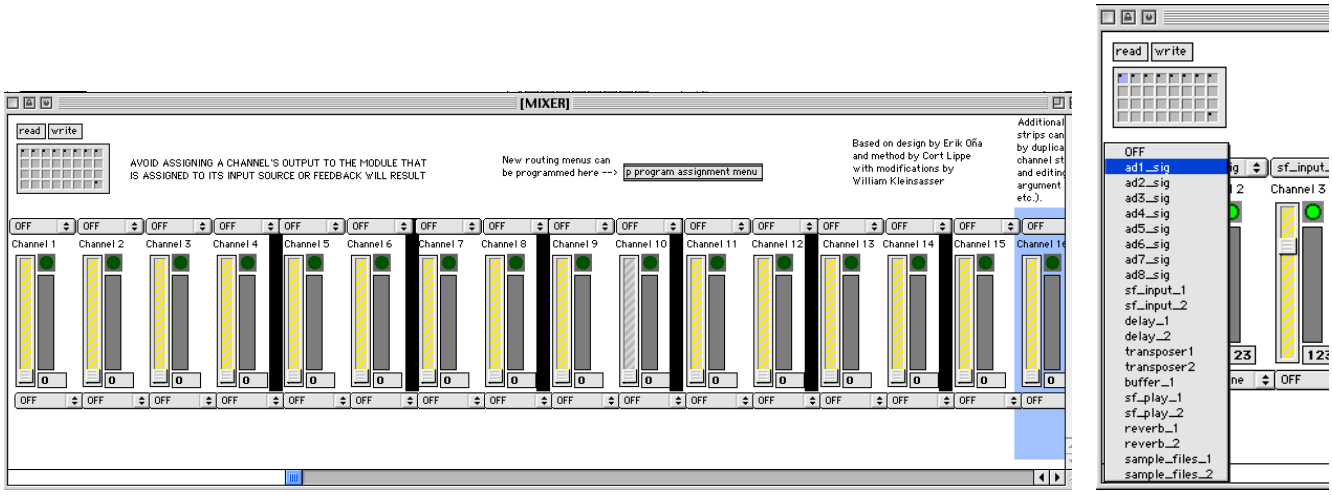


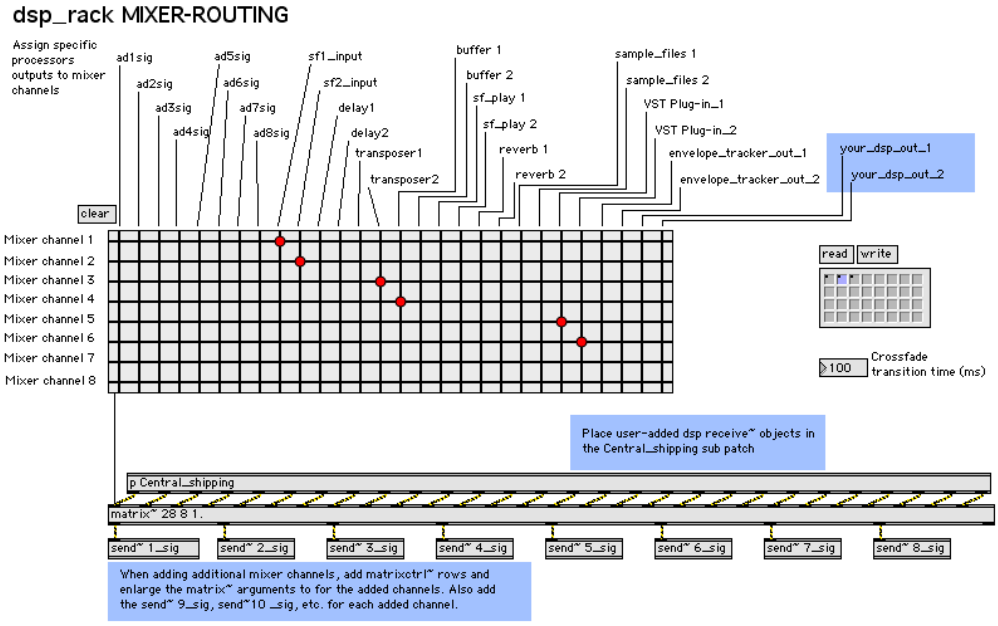
Figure 6. Version 1 MIXER with menu-driven channel inputs and output assignments

The output routing for each channel strip is handled with programmable assignments using a pop-down menus for target inputs. This design allows for easy reconfiguration of the entire routing matrix on a preset-by-preset basis. Dsp.rack has a mixer that can be extended by simply duplicating the mixer channel strip objects as described in the mixer window.

A programmable fader transition time avoids sudden level changes associated with programmed preset changes. Using the slew setting for the audio faders, the gain settings for faders can change gradually over a user-defined time interval.

The inputs for the mixer channels are handled using the same approach. Each of the matrix node configurations can be stored as a preset and recalled in sync with all of the other presets from the main window. Additional matrix nodes can be added by the user.

In version 2 the mixer strips are routed to the eight output channels for the adc~ object using menus. This means that mix level of the signal output for each dsp module must be set in the programmable fader settings within each dsp module and not in the mixer. In version 2 the mixer can be is the final level control for the output of each dsp chain.



Figures 7. matrix routing for inputs and processing modules

7. INTEGRATED SIGNAL PROCESSING AND PRE-RECORDED SOUND FILE PLAYBACK

The performer+tape paradigm that flourished from 1960-1990 offered a model of musical expression that expanded the capabilities of acoustic music into the electronic studio environment. Composers produced a body of works that presented acoustic performance in the context of technologically transformed music on tape.

But the synchronization issues of performer+tape music can be a major drawback in these works. Dsp.rack is designed to support live interactive signal processing but it also can support the repertoire of performer+tape compositions by retaining the ability to present pre-recorded, overlapping sound files supporting a method of presenting pre-recorded sound files as overlaid layers which allows for timing flexibility in performance. Dsp.rack offers a module for loading and playing sound files with up to four overlapping multi-channel players. These sound files can either be routed directly out to the sound system, or following the general crossbar approach, can be routed to the inputs of the other processing modules.

A patch is provided that shows how to dub 8-channel files in from TDIF, ADAT MTRs or similar external sources.

8. PERFORMANCE AND CPU LOAD

While relatively long latency presents a problem when using the Macintosh's Sound Manager for i/o, using a converter like the RME Hammerfall reduces latency. With several complex dsp modules, an 8-channel mixer, and 8-channel i/o,

Dsp.rack uses about 35% of the cpu on a 1G G4 Powerbook. The same setup uses about 75% of a 500 mHz G3 Powerbook. Dsp.rack uses the mute object for enabling and disabling each individual dsp module which is useful for processing-intensive modules but less so for lighter load modules.

Dsp.rack is distributed as freeware that runs within the Max/MSP environment. It provides a relatively inexpensive path to familiar, personally expandable tools for integrating computer music with live performance. Dsp.rack has been designed to offer the benefits of relatively easy start-up and open-ended potential for refinement and growth. It is hoped that this will prove attractive to composers, performers, students, and those who teach others to engage this field.

9. ACKNOWLEDGEMENTS

The audio processing contained in this distribution of Dsp.rack is based on standard-issue Max/MSP objects with the exception of the tap.shift pitch shifting object which is distributed with Dsp.rack by permission from its programmer, Timothy Place, and the Newverb~ object by Richard Dudas which is available as a public domain reverb object from the Cycling74 web page of shared objects. Tap.shift is distribued with dsp.rack with a free-use license for this application so long as no fee is collected for its use and so long as the ReadMe document associated with it is included in the distribution.

The following work by Max/MSP developers are acknowledged as having been of great use in the development of Dsp.rack: Erik Oña and Cort Lippe offered models for the main "crossbar" mixing method using menu-driven routing.

Chris Dobrian and Cort Lippe offered help on the buffer writing method and other audio handling. The sound file playback and delay methods were developed in order to help, and deriving help from, my students Brian Comotto, Daniel Hope, Scott Leake, Ljiljana Jovanovic, and Nicholas Schoeb. Scott Leake also helped a great deal with the troubleshooting phase and development of the VST plug-in module in version 2. Thanks to Miller Puckette and David Zicarelli for developing Max and Max/MSP. Thanks also to all of the Max/MSP developers who have shared their solutions and ideas

10. REFERENCES

[1] Dobrian, C. Programming New real-time DSP Possibilities with MSP, *Proceedings of the 2nd COST G-6 Workshop on Digital Audio Effects (DAFx99)*, NTNU, Trondheim, December 9-11, 1999

[2] Lippe, C. A Look at Performer/Machine Interaction Using Real-time Systems, *Proceedings of the International Computer Music Conference*, Hong Kong, 1996

[3] Lippe, C. A Composition for Clarinet and real-time Signal Processing: Using Max on the IRCAM Signal Processing Workstation, *Proceedings of the 10th Italian Colloquium on Computer Music*, Milan, 1993

[4] Lippe, C. Music for Piano and Computer: A Description, *Information Processing Society of Japan SIG Notes*, Volume 97, Number 122, 1997

[5] Place, T. Silicon Prairie Intermedia, http://www.sp-intermedia.com/downloads/index.html

[6] Puckette, M. New Public-Domain Realizations of Standard Pieces for Instruments and Live Electronics, *Proceedings, International Computer MusicConference*, 2001

[7] Puckette, M. Pure Data; Recent Progress, *Proceedings, Third Intercollege Computer Music Festival*, Tokyo, Japan, 1997

[8] Rowe, R. *Interactive Music Systems: Machine Listening and Composing*, The MIT Press, Cambridge, 1993

[9] Zicarelli, D. An Extensible Real-Time Signal Processing Envirnoment for Max, *Proceedings of the International Computer Music Conference*, Ann Arbor, 1998