A CommonLisp interface for dynamic patching with the IRCAM Signal Processing Workstation

Thomas Hummel
IRCAM, 75004 Paris, France
hummel@ircam.fr

Abstract

The usual architecture of a patch in the IRCAM Signal Processing Workstation (ISPW) [Puckette, 1991] is a cross-bar network consisting of basic controls and synthesis modules.

An alternative architecture based on a dynamic patching system is presented. The basic modules can be dynamically connected, disconnected and are switched on or off. It is shown that about 30 basic audio modules per 660 processor can be freely interconnected, replacing a cross-bar system with more than 2000 nodes. A CommonLisp interface is presented which creates glites which assemble and disassemble temporal patches.

A piece for string quartet and ISPW demonstrates also the power of this technique.

I Introduction

The introduction of audio modules into MAX is a basic improvement of the IRCAM Signal Processing Workstation (ISPW) compared with MAX/Opode running on the Macintosh [Puckette 1991]. The change from a pure control environment towards a mixed control/synthesis environment in the ISPW implies also a significant change in the patch architecture. This is especially true if the ISPW is used in real time situations as will be assumed in this document.

A typical strategy of the composer is the development of different "situations" - in this case patches - which are sequenced in a piece. Control objects in MAX allow this technique perfectly because they consume only RAM but, unless they are triggered, no CPU-time.

Audio modules, however, accumulate CPU-time since they are continuously updated. In the normal case, the restriction in the number of audio modules is taken into account through two different architectures. The one uses only one or a few, perhaps complex, synthesis modules and controls them in various ways.

The other architecture frequently used imitates old analogue synthesizers like the EMS and its cross-bar system. The basic modules are used in differing contexts or, better, patches through an audio network where all audio inputs are connected to all audio outputs through faders:

The other architecture frequently used imitates old analogue synthesizers like the EMS and its cross-bar system. The basic modules are used in differing contexts or, better, patches through an audio network where all audio inputs are connected to all audio outputs through faders:

![Diagram of audio network with ins and outs](image)

Since each cross-bar element contains an audio multiplexing and a summation, the cross-bar system stresses the processors in proportion to the product of ins and outs, thus rapidly exceeding the processor capacity. In fact, the message list of all fader positions now represents a sort of temporal patch losing the intuitivity of the original graphic MAX representation.
2 Dynamic patching

We will see that dynamic patching on the ISPW is a promising alternative to the cross-bar method due to its much more economic and flexible behaviour. Dynamic patches have already been tested on earlier systems, such as the Sun-Mercury system (Dierk and Rodet, 1988).

2.1 Basic concepts

MAX on the ISPW offers 3 objects which allow dynamic patching (for complete documentation refer to [Battino, 1993]): throw-, catch- and switch-. The throw-object may get a message set address so that the audio signal is transferred to a catch-object named address. This allows the redirection of an audio connection. The switch-object removes a part of the patch from, or introduces it into, the calculation sequence so that only those subpatches are processed which are necessary for a temporal patch. The throw-catch-pair replaces the large cross-bar system.

![Diagram of throw, catch, and switch objects](https://via.placeholder.com/150)

**Figure 1:** Example of an atom with indication of a dynamic connection to another atom.

The architecture for a dynamic patching system is a set of isolated audio subpatches named atoms, each having catch-objects as inputs, throw-objects as outputs and switch-objects (Figure 1). Frequently, functionally identical copies of the atoms are used (e.g. line-controls). A temporal patch is installed by a list of switch-on-messages and set-messages, which build the connections. They could be called patch-on-messages and patch-off-messages, in analogy to note-on/note-off.

Certain help-atoms are needed if atoms are connected to several others in a dynamic patch, since throw-- does not allow multiple addressing:

![Diagram of throw, catch, and help atoms](https://via.placeholder.com/150)

2.2 Patch sequence

When sequencing different dynamic patches, some side-effects have to be treated during patch-on and patch-off processes. Dynamic line modules can be labelled to be amplitude-sensitive. When a patch-off-message is encountered, these line modules fade to zero and the patch is switched off, with a certain delay to avoid clicks. After a patch-on-message outputs of delay-lines are set to zero for the desired delay time because the corresponding tables remain the last content when switched on.

Signal scale modules can be introduced at amplitude-sensitive positions in the patch. They receive their value during the patch-on-message. This method controls the overall amplitude of a patch similarly to the velocity information of a MIDI-note-on.

In order to determine the selection of certain atoms it has to be known whether the patches are ordered sequentially or exist simultaneously on the same processor. Only if two patches have different resources are they able to exist in parallel. Resource allocation during the patch-on-messages is precalculated for a better real-time performance (see below).

Further, a real-time mechanism was developed to treat the problem that patch-on-messages may be triggered before previous patches which share the same resources have finished (a frequent occurrence in concert situations). In this case the last patch is switched off before the new one is installed.

ICIC Proceedings 1994
3 CommonLisp interface

A CommonLisp interface is introduced, facilitating the development of patch-on- and patch-off-metastase. It runs in Allegro Common-lisp on the NeXT.

3.1 Purpose

The CommonLisp interface writes MAX-loadable subpatches containing gliss and additional patch code which manages patch-on- and patch-off-messaging. Based on a given sequence of patches the interface calculates the resource allocation and the need for signal-duplicating help-atoms and then estimates the maximum necessary CPU-percentage.

3.2 Functionality

The description of a patch in CommonLisp is divided into:
1. the description of the modules
2. the description of the interconnections.

(setq myoscillator (nosc))

sets the user-defined variable myoscillator to the result of the key function nosc. nosc is a Lisp function which allocates a new oscillator module and returns its address.

(setq mymultiplication (nnmul))
(con mymultiplication myoscillator)

allocates an (audio)multiplication, and the function con establishes a temporal connection between the two modules.

A temporal patch is represented by a function containing all atoms and interconnections, e.g.:

(defun mypatch ()
  (setq myosc (nosc 0))
  (setq mymultiplication (nnmul))
  (setq myenvelope (naenve 39))
  (setq mydac (ndac 0))
  (con myenvelope (dev mymultiplication 1))
  (con myosc mymultiplication)
  (con mymultiplication mydac)
)

The following patch illustrates the result without showing cutsets, throws and the corresponding gliss:

The different patch functions are invoked within a time and duration context, e.g.:

(time 0)
(dur 2 4)
(mypatch)
(time 1)
(dur 3 4)
(mypatch1)

etc.

The duration is given as a fraction and gives the difference between the patch-on and the patch-off times in the gliss. The time-function serves only to define the simultaneity or non-simultaneity of different patches, i.e whether, or not, the patches are allowed to have the same resources. Finally, a function called evalproc writes the subpatches for all temporal patches.

3.3 Temporal interconnections

Dynamic patching allows not only patch-sequenceing but also the temporal (audio) composition of patches (just as, for instance, a patch which produces a control signal can be installed temporarily to send its control signal to the frequency input of the frequency shifter of another patch). In the CommonLisp interface it is thus possible to return the address of an atom invoked in a patch function, e.g.:

(defun control-patch ()
  (setq myoscillator (nosc))
  .
  myoscillator
)
and

(defun shifter-patch ()
  (mapcar (function myshifter) (kthk)))

myshifter
)

The function \texttt{con}, used outside the patches, establishes a temporal communication during the period defined by \texttt{dur}.
(con myoscillator myshifter)

4 Musical application

The dynamic patching system and the \texttt{CommonLisp} interface were used by the author in a work for string quartet and ISPW, named \texttt{cococon} (1993). The ISPW is employed exclusively for live-electronics. The four instruments have individual microphones, are transformed by the ISPW and projected over four channel PA. The patch turns on one card at 32 kHz and uses, on each processor (the following numbers are, therefore, to be doubled); 2 variable delay/reverbs; 3 small filterbanks; 5 frequency shifters; 5 oscillators (as signal controls, or for ring modulation); 8 envelope generators; 3 signal generators; 9 multipliers; and 15 scalings. The atoms are connected to around 100 structurally different temporal patches. Realizing the same with the cross-bar technique would need a 48 by 57 matrix (= 2756 nods!) on each processor. The advantage of the dynamic patching method is evident.

5 Summary

The dynamic patching method is revealed to be a powerful method for economizing on CPU-time and, at same time, showing a combinatorial flexibility, unachievable with other ISPW-patch architectures. As a further development it would be worth building a \texttt{CommonLisp} interface to transform an existing fixed patch into a dynamic patch, rather than building the patch in a, less intuitive, text format. Also, it would allow the user to test the patch in a graphical form.

References