Pure Data: another integrated computer music environment

Miller S. Puckette

Department of Music, UCSD, La Jolla, Ca. 92039-0326 msp@ucsd.edu

 Reprinted from Proceedings, Second Intercollege Computer Music Concerts, Tachikawa, pp. 37-41.

May 7, 1997

Abstract

A new software system, called Pure Data, is in the early stages of development. Its design attempts to remedy some of the deficiencies of the Max program while preserving its strengths. The most important weakness of Max is the difficulty of maintaining compound data structures of the type that might arise when analyzing and resynthesizing sounds or when recording and modifying sequences of events of many different types. Also, it has proved hard to integrate non-audio signals (video, for instance, and also audio spectra) into Max's rigid "tilde object" system. Finally, the whole issue of maintaining two separate copies of all data structures (one to edit and one to access in real time) has caused much confusion and difficulty. Pd's working prototype attempts to simplify the data structures in Max to make them more readily combined into novel user-defined data structures. Also, the relationship between the graphical process and the real-time one (which is handled in one way on the Macintosh and another way on the ISPW) is replaced by yet a third solution.

1 Introduction

The design of real-time computer music systems has been a subject of active research since the RTSKED program [1]. By 1986 several authors were proposing formal or semi-formal real-time protocols, sometimes in the guise of complete systems for doing real-time computer music [2], [3]. The question of making software systems which were really usable by non-computer scientists was addressed by the Max program [4]. Max was an attempt to make a screen-based patching language that could imitate the modalities of a patchable analog synthesizer. Many other graphical "patching languages" had been proposed that did not sufficiently address the real-time control aspect; and many other researchers had by then proposed much more sophisticated real-time control strategies without presenting a clear and fun-to-use graphical interface; Max was in essence a compromise that got part way toward both goals.

As soon as Philippe Manoury's *Pluton* was realized using Max (thus proving Max to be an interesting environment for real-time computer music), a stream of criticisms of Max started to appear. Max wasn't originally intended as a programming language; yet many users treated it as one. As such Max had obvious shortcomings, some of which are reported in [5]. Also, the original version of Max, being written for a Mac Plus computer, didn't address the question of computer-generated audio, remaining instead in the realm of MIDI. This was addressed in [6], but only for special hardware.

The question of how to use Max to amass and use *data* arose in IRCAM during the design phase of the ISPW. A new software idea, called Animal, was proposed and implemented [7]. Many ideas in Pd owe their origin to the Animal program.

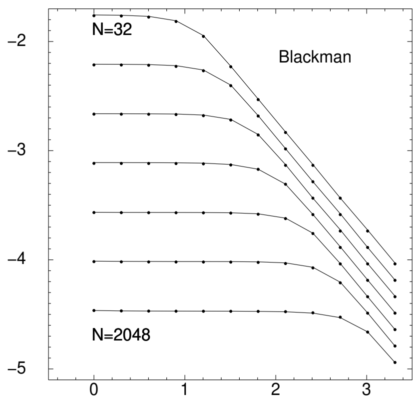

Figure 1: Sample Pd-generated graph.

2 Design

Pd's first application has been to prepare the figures for an upcoming signal processing paper by Puckette and Brown. Commercially available software for graphing data proved unsuitable for this application. (The same can often be said of music software!) The first exercise for Pd has been to render graphs such as in Figure 1. There is as yet no editor for these graphs. The source for the one shown follows:

```
#N canvas noise-sim-bla;
  #X font -*-helvetica-medium-r-*-*-*-300-*;
  #X text 13 3 Blackman;
  #X graph noise-sim-bla.plot;
    #X xticks 2 0.2 5;
    #X xlabel -5.2 0 1 2 3 4;
    #X yticks -5 .1 10;
    #X ylabel -0.85 -2 -3 -4 -5 -6;
    #X style 7 0 point;
      ... [6 more "style" lines omitted]...
    #X text .2 -1.9 N=32;
    #X text .3 -4.7 N=2048;
  #X pop;
#X pop;
```

The format is very much like the Max file format, which features embedded descriptions of patchers. As with Max, the message system is used in the same way to restore saved documents as in message passing for real-time computer music control and synthesis.

In this example, the new "canvas" object is the window, which names itself `#X`. The third command to `#X` is to create a graph, which then receives its own messages before the first "pop" message resets `#X` to point to the original canvas. Text can be added either to the graph (in the graph's own coordinates, like "N=32") or to the canvas (in centimeters, like the string "Blackman"). The data to be graphed are taken from a file in this case, but support is also included for embedding the data in the Pd file.

Canvasses, which currently can hold only text and graph objects, will soon support the 0.26-style Max boxes and interconnections, and also a framework for collections, which will generalize Max's `explode` feature. Pd will therefore incorporate three window types of Max (patch, table, explode) into a single new window type.

3 Templates

In Pd, the notion of a "patch" as in Max, and the notion of a dialog (for searching, for instance, or for setting the range of a slider) are unified. (There is one exception: the file selection panels will be normal dialogs, not patches.) If, for instance, we "open" a Max-style number box, a template window appears as in Figure 2.

Another example of a template would correspond to a point of the graph of Figure 1; its two entries would be for the "x" and "y" values of the point. In this case, it would be permissible to add a field to the template, which would add the corresponding field to all the points belonging to that template. This could either be the points of one of the curves of the graph, all the points of the graph, or all the points of several graphs.

Templates differ from the "abstractions" of Max in that the template is not re-instantiated for each invocation; a single template acts as a non-modal dialog window for all instances of the data structure it controls. If an anonymous data structure (a one-off member of a heterogenous sequence, in effect a Max message as in a Max qlist) is opened, a made-to-order

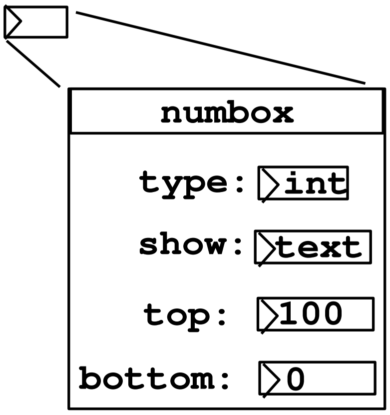

Figure 2: The template for a number box.

template is created to show it.

To support templates, a new "field" object is introduced. Fields differ from numbers in having "field names" as in Figure 2. Max-style patchable objects can query a template patch for the onset of a field of a given name, so patches can traverse data structures through their templates. Also, any data structure (which could be called a "Pure Datum") can become a Max message handled in the usual way.

4 DSP

Max/FTS, which was the predecessor of Pd, had a severely limited notion of audio signals, which has proven too restrictive for working with frequency-domain or non-audio signals. In Pd, users may create DSP "blocks" in which sample rate and vector size vary. This is done using two new "tilde" objects, `clock~` and `reclock~`. The first of these allows the user to attach a symbolic name to a specific combination of sample rate and vector size. This notion replaces the `switch~` mechanism of Max/FTS; any `clock~` can be turned on and off. The `reclock~` object simply converts a signal to any desired clock. For example, preparing a signal for overlap-4 FFT analysis is simply re-clocking to a clock with the necessary properties; the overlap-add step for converting back into the time domain is accomplished by re-clocking to the "usual" clock.

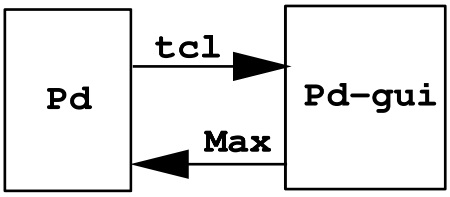

Figure 3: Pd implementation.

Pure Data also will get rid of one major annoyance in Max, which is the necessity of specifying `signal~` objects to provide constant inputs; Pd will allow tilde objects to query their connections; if no signals are connected to a signal input but a floating-point outlet is connected, a scalar-input version of the tilde object will be called if available; otherwise, a `signal~` object will be provided automatically.

5 Implementation

Pd is in two parts, as shown in Figure 3. The "real" Pd, shown at left, does real-time computations using a Max/FTS-like message interpreter and scheduler. All Pd documents reside in the address space of Pd. The other process, named Pd-gui, talks to the computer's window system through the "tk" toolkit [8]. This should allow portability to the X, MS, and Macintosh window systems, although the current prototype runs only on SGI hardware.

The real-time/non-real-time divide works very differently from that of Max/FTS; the "editor" resides in the real-time Pd layer, not in Pd-gui (in Max/FTS the four different editors all reside in the GUI layer, Max.) This choice reflects an important change that has occurred in computer hardware in the last ten years: multiprocessors like the 4X and ISPW are now giving way to uniprocessors (at least in computer music applications.) This year we can expect to see a uniprocessing computer announced which will equal the six-processor ISPW in speed, at less than half the cost. For the ISPW, it was necessary to make the document (the "patch") reside in the graphical layer, because otherwise there would be six documents. This caused nightmarish problems in keeping the two data sets (Max's and FTS's) coherent. In Pd, we sidestep this problem by putting all calculations, both editing moves and real-time audio rendering, in a single program. Pd is therefore badly suited to multiprocessing but more in line with current hardware developments than Max/FTS.

Pd should provide near patch-level compatibility with Max 0.26 from IRCAM, and near source compatibility with 0.26 externs. Most of the differences will be minor design changes (argument types of library functions, for example) which can easily be aliased in an "include" file. At the patch level, certain objects such as `FFT~` will be replaced, and the nefarious downsampling argument to `line~` and `sig~` will probably be replaced with the symbolic name of a `clock~` object. Pd will probably read Max patches but only save files in the Pd format. The C source code will be made freely available to the public.

6 Conclusion

It is too early to say whether or not Pd will replace Max as a real-time computer music environment. The success of Max reflects a lucky confluence of many events: the rise of the Macintosh, the arrival at IRCAM of di Giugno's 4X machine (which provided the problem that Max had to solve), David Zicarelli's brilliant work in getting Max published (not to mention his almost rewriting the entire program, much to its improvement), and the contributions of Zack Settel, Cort Lippe, Philippe Manoury, Chris Dobrian, David Wessel, and literally dozens of others to the design, documentation, and creative abuse of the program. The success or failure of Pd will ride as much on its finding a similar community as on its design specifics.

References

[1] Mathews, M. and J. Pasquale, 1981. "RTSKED, a Scheduled Performance Language for the Crumar General Development System." Proceedings of the 1981 International Computer Music Conference. San Francisco: Computer Music Association, p. 286.

[2] Anderson, D. and R. Kuivila, 1986. "A Model of Real-Time Computation for Computer Music." Proceedings of the 1986 International Computer Music Conference. San Francisco: Computer Music Association, pp. 35-41.

[3] Boynton, L. *et al.*, 1986. "MIDI-LISP: A LISP-Based Music Programming Environment for the MacIntosh." Proceedings of the 1986 International Computer Music Conference. San Francisco: Computer Music Association, pp. 183-186.

[4] Puckette, M., 1988. "The Patcher." Proceedings of the 1986 International Computer Music Conference. San Francisco: Computer Music Association, pp. 420-429.

[5] Lewis, G., 1992. ref-lewis "A Max Forum", *Array* 13/1, pp. 19-20.

[6] Puckette, M., 1991. "FTS: A Real-time Monitor for Multiprocessor Music Synthesis." Computer Music Journal 15(3): 58-67.

[7] Lindemann, E., 1991. "ANIMAL – a Rapid Prototyping Environment for Computer Music Systems." Computer Music Journal 15(3): pp. 78-100.

[8] Osterhout, J. K., 1994. *Tcl and the Tk toolkit.* Reading, Massachusetts: Addison-Wesley.