

GrIPD: A Graphical Interface Editing Tool and Run-time Environment for Pure Data

Joseph A. Sarlo

Department of Music, University of California, San Diego
email: jsarlo@ucsd.edu

Abstract

We describe here a new interface tool for Pure Data (Pd). GrIPD (Graphical Interface for Pure Data) is a cross-platform software package that allows one to design custom graphical user interfaces for Pd patches. GrIPD is not a replacement for the native Pd interface, but rather, is intended to allow one to create a “performance-time” front end for a Pd patch. GrIPD extends the usability of Pd through various features including a multi-process design structure and TCP/IP network communication system that natively allow for various multiple-computer implementations.

1 Introduction

In the realm of real-time computer interaction, knowledge of the current state of the system is crucial. This is especially the case for real-time interactive computer music performances, in which the cost of system interpretation error is high. Indeed, the very concept of interactivity necessitates some degree of system awareness. Frequently in computer music performances, this awareness is derived from auditory cues, as the performer responds to the sound generated by the system. However, visual information is often equally important. The user must also be able to alter the state of the system. As Norman defines (1986), there is a Gulf of Evaluation and a Gulf of Execution between the user and the system that must be bridged. It is the role of the user interface to bridge these gaps.

Many real-time computer music systems exist. One such system is Pure Data (Pd) (Puckette 1996). Essentially, a real-time system is implemented in Pd by defining a set of interconnected objects, the graph of which is referred to as a patch. The same interface is used for both designing the system and interacting with it.

GrIPD (Graphical Interface for Pure Data) has been developed as a “performance-time” user interface system for Pd. GrIPD is not intended as a replacement for the native Pd interface, which is used at “design-time” for constructing Pd patches. Instead, the purpose of GrIPD is to allow for the creation and usage of

custom “performance-time” graphical user interfaces (GUIs) for individual Pd patches. Like Pd itself, GrIPD is both cross-platform and open-source. GrIPD is currently known to run on x86 Linux/GTK+ systems and all Microsoft Windows operating systems from Windows 95 to Windows XP.

In this paper we will discuss the architecture and design of GrIPD, its usage and noteworthy features, and plans for future improvements.

2 Architecture and Design

The GrIPD software package consists of two main components: the GrIPD Pd object and the GUI editor and run-time environment application. These two components operate as completely separate processes and communicate via TCP/IP. Messages are passed between the Pd object and the GUI application using a simple ASCII protocol.

2.1 The GrIPD Pd Object

The GrIPD Pd object is essentially an external library that is instantiated inside a Pd patch. The object itself is written in the C programming language. It has three main functions. Firstly, it acts as the TCP/IP server for the GrIPD communication system, for which the GUI editor and run-time environment application is the client. Also, it includes the internal messaging system that allows a patch to seamlessly communicate with the GrIPD system without having to explicitly patch to the GrIPD Pd object. Lastly, the Pd object has functions to control aspects of the GUI application such as opening, closing, and locking.

The TCP/IP server aspect of the GrIPD Pd object functions in much the same way that most servers do. It opens a socket using an available port number or a port number specified explicitly as an argument to the object at instantiation time. The port is polled until a connection is accepted from the client GUI application. The GrIPD Pd object has an outlet to report the current connection status. After the connection is established, the socket is continually polled for incoming messages from the client. Similarly, messages that are to be sent from the server to the client are queued into a buffer that

is periodically sent and flushed. The polling interval for both sending and receiving are user adjustable.

A Pd patch communicates with the GrIPD system through the Pd messaging system of “send” and “receive” objects. This is similar to the way the native Pd GUI objects function, and thus, the GrIPD system can easily be incorporated into existing Pd patches. When the GrIPD Pd object receives a message reporting the value of a control (e.g. slider, button, etc.) from the GUI client application, the GrIPD Pd object sends that value to the appropriate “receive” objects of the Pd patch. Similarly, the GrIPD Pd object contains multiple “receive” objects corresponding to the GUI controls of the client GUI application. The GrIPD Pd object can thus listen for appropriate messages from the “send” objects of the Pd patch and relay them to the GUI client application controls.

The GrIPD Pd object also has functions to control various aspects of the GUI application itself. For this, the GrIPD Pd object has one inlet that accepts the following messages:

- **connect**: begin polling the socket for an incoming TCP/IP connection
- **disconnect**: close the TCP/IP connection
- **open** <optional filename>: launch the GrIPD GUI client and optionally open a GrIPD GUI definition file
- **open_locked** <optional filename>: launch the GrIPD GUI client in locked mode, disallowing any editing of the GUI, and optionally open a GrIPD GUI definition file
- **lock**: place the GrIPD GUI application in locked mode, disallowing any editing of the GUI
- **unlock**: place the GrIPD GUI application in unlocked (default) mode, allowing editing of the GUI
- **hide**: hide the window of the GrIPD GUI application, but do not close or disconnect from the TCP/IP connection
- **show**: show the window of the GrIPD GUI application that has been hidden, either by the GrIPD Pd object or the GrIPD GUI application itself
- **poll_send** <integer polling interval>: set the polling interval for sending and flushing the outgoing message queue
- **poll_receive** <integer polling interval>: set the polling interval for receiving messages
- **set_path** <filepath>: set the path to the location of the GrIPD GUI application (for use with open)

2.2 The GrIPD Editor and Run-time Environment

The GrIPD editor and run-time environment is a stand-alone application. It is in this application that the GUI for a Pd patch using the GrIPD system is both edited and used. It communicates with Pd through the GrIPD Pd object by acting as the TCP/IP client of the GrIPD communication system. Like Pd itself, it has two main modes of operation: edit mode and performance mode.

In edit mode, a GUI for a Pd patch can be designed and edited. This is done by creating GUI controls (e.g. sliders, buttons, etc.) and editing their properties as desired. The GrIPD GUI application implements a WYSIWYG (what-you-see-is-what-you-get) style of editing. When a control is created, it appears on the GUI application window and can be moved and resized by the usual WYSIWYG means of mouse dragging. Other properties, such as color and font, can be edited through a properties sub-window. Several global properties can also be set, including window title, background color, and polling intervals.

While in performance mode, the GUI controls provide their normal control functions. When acted upon, they report their status to the GrIPD Pd object via the TCP/IP communication system. For example, when a button is pressed, it reports both a “bang” message and its symbol name to the GrIPD Pd object, which then sends a “bang” to all “receive” objects in the Pd patch sharing that symbol name. Controls can also receive messages from the GrIPD Pd object to change their status by the reverse process. In addition to the status of the GUI controls, the GrIPD GUI application can also send keyboard, mouse, and joystick events to the GrIPD Pd object.

The GUI application itself was written using the Python programming language (Van Rossum and Drake 2002) and the wxWindows GUI library (Smart, *et al.* 2003) via the wxPython binding library. These tools offer several advantages. Firstly, both the Python language itself and the wxWindows library are cross platform. This allows the GUI application to be supported by multiple platforms without the need for multiple versions of source code. Secondly, the wxWindows GUI library takes advantage of native GUI controls of the various platforms it runs under. It accomplishes this by simply implementing its GUI components as wrappers around the system’s native GUI components. This has several advantages discussed below.

3 Features and Usage

The GrIPD system has several features that provide extended function and configuration to Pd. The most noteworthy of these are due to two main aspects of the GrIPD system: the graphical and control functions of the

GUI application and the multi-process design architecture.

GrIPD extends the interface possibilities of Pd by offering GUI controls not available in Pd itself. These include multi-size, color, and font dynamic-text, textbox style data entry, and dynamic “clickable” images. In addition, GrIPD can also capture input data from a joystick for use in controlling parameters of a Pd patch. Currently GrIPD offers the following types of GUI controls: push button, toggle button, spin button, radio button, vertical and horizontal slider, vertical and horizontal gauge, checkbox, textbox, mouse capture area, dynamic text rectangle frame, dynamic display text, and dynamic clickable image.

Since many of the wxWindows GUI components are simply wrappers around the native system GUI components, GrIPD controls automatically take on the “standard” look and feel of the user’s system. This can greatly improve usability and provides a familiar and comfortable environment for the user. It also grants inherent theme ability to the GrIPD GUI application on those systems that natively support GUI themes, such as Microsoft Windows XP and Linux/GTK+. In this way, the user can customize the look and feel of the GUI by using means already provided by the system.

The multi-process architecture of GrIPD also offers several advantages. Firstly, it grants the ability for prioritizing the audio processing above GUI updating for operating systems that support process scheduling priority. For these systems we can set the scheduling priority of the Pd process higher than that of the GrIPD process. Thus, we needn’t worry about issues, such as garbage collection, that usually disallow interpreted languages, like Python, from being used with real-time audio processing. We should note that the Pd native interface takes advantage of this as well.

Also, since the Pd and GrIPD processes are separate and communicate via TCP/IP, we can run them on separate computers. Specifically, GrIPD allows this to be done easily and nearly transparently. The user simply sets the port for both the GrIPD GUI application and the GrIPD Pd object to be equal. This is done via a creation argument for the Pd object and the options menu for the GUI application. The user then sets the connection address for the GUI application to the IP or DNS address of the server running Pd via the options menu. Both components can then dynamically connect and disconnect.

Using this multiple computer feature, GrIPD offers several performance possibilities that can be easily implemented. For example, a large and powerful computer can run Pd and be used for audio processing while a smaller and more portable computer can run the GrIPD GUI application and be used as its controller.

The multi-computer feature is not limited to a single-client and single-server implementation. Multiple GrIPD GUI clients can connect to a single Pd patch by

simply instantiating multiple GrIPD Pd objects. Therefore, one could easily display information from one Pd patch to multiple performers using multiple computers. Also, multiple computers could be used to control a single Pd patch. This opens up a multitude of real-time performance collaboration possibilities.

Since GrIPD uses TCP/IP as its communication protocol, GrIPD users can take immediate advantage of such technologies as IEEE 802.11 wireless networking and the Internet itself. Thus, client and server computers can be linked wirelessly using commonly available and cost-effective standard consumer wireless networking equipment. They can also be seamlessly linked across great distances via the Internet. Also, since GrIPD is cross-platform, these two computers do not need to use the same operating system, which can be preferable since different operating systems offer different advantages.

4 Future Improvements

While the GrIPD system currently offers many advanced features, it is still in its early phase of development. Many improvements and features are planned for the future. In addition to the usual code improvements, GUI enhancements, and usability extensions, GrIPD is planned to be ported to all operating systems that Pd supports, primarily Mac OS X. It is hoped that this will further enhance multiple computer possibilities and extend the user base. Also, further control possibilities such as MIDI input from the GrIPD GUI application are planned. In a multiple computer implementation, this would allow a client computer running the GrIPD GUI application to accept MIDI input and transmit the MIDI data to the server computer running Pd. In conjunction with the previously mentioned multiple computer feature, this could greatly expand performance possibilities.

References

- Norman, D. A., and S. W. Draper, (eds). 1986. *User Centered System Design: New Perspectives on Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Puckette, M. 1996. “Pure Data.” *Proceedings of the International Computer Music Conference*. International Computer Music Association, pp. 269-272.
- Smart, J., R. Roebing, V. Zeitlin and R. Dunn. 2003. *wxWindows 2.4.0: A portable C++ and Python GUI toolkit*. Artificial Intelligence Applications Institute, University of Edinburgh. Available at <http://www.wxwindows.org>.
- Van Rossum, G., and F.L. Drake (eds). 2002. *Python Reference Manual*. Fredericksberg, Virginia: PythonLabs. Available at <http://www.python.org>.