

ImpureData

Mathieu Bouchard <matju@artengine.ca>

September 24th, 2004 (1st draft)

Abstract

This paper covers the design principles used in the construction of ImpureData and the new features comparatively to PureData 0.37. It is targeted at people who are acquainted with PureData's source code, although the live demo can reach every user.

1 Separation of User-Oriented Concerns

1.1 The Observer/Observable relationship

Its goal is to separate the modification of a GUI object from the modification of its appearance. This way, objects don't have to care about how often they are updated and it is possible to arrange it so that many less messages are sent over to the client side. The **pd_changed()** function is the one to be called by an object to indicate that it should eventually be redrawn. Many calls to that function may be made very quickly but it is up to **pd_changed()** and related functions to figure out when updates should be done. If the GUI is being clogged by messages, some updates may be slightly delayed or just skipped, which is better than getting slowdowns and dropouts.

However this has not been fully implemented.

1.2 The Model/View pattern

Its goal is to separate a GUI object into a non-GUI part and a GUI part. There are many different variants on this pattern in use in many different GUI toolkits. Tk does not support it, but it is still possible to infiltrate the pattern somewhere. Anyway, in the context where Model/View is used, Tk is merely used as a vector graphics toolkit, and the Model/View split affects the structure of the C code (**g_*.c**) by separating the calls to **sys_vgui()** from the main part of the code.

1.3 Client/Server RPC

sys_vgui() calls can be further abstracted by making them disappear completely. First, the View code is rewritten in Tcl to be on the client side, and second, the calls to **sys_vgui()** can be replaced by calls to **pd_upload()**,

which copies the server-side object to the client-side (a strange but easy way to implement Model/View that can only occur in a client/server setting) and `sys_mgui()`, which calls a method on the view object. `pd_upload()` is implemented in terms of `savefn` which is used to save a patch to a file.

1.4 The Tcl language

The Tcl[1] language has many shortcomings when it comes to day-to-day programming. I mean compared to Perl5, Python, Ruby, etc. Mainly, it is lacking pointers, and its dictionaries (hashtables) are an extension of local/global variable dictionaries, and not a separate concept, which means that if you want a dictionary that lasts a long time, you make it global.

Nevertheless I got my way around the problems, and made the syntax to be roughly like the way object-oriented programming is done in Perl5, using a hack that may shock many programmers, yet is quite clean as it uses a forbidden style in a very sharp way that solves precisely the problem.

The way it works is that there is a global variable called “_”; yes, only one character, and it’s underscore; if I had known beforehand that it’s possible to use the blank string as a variable name, I would have used that, as it makes the syntax even lighter. (Outrageous, isn’t it?) Well, after that, you can read an object field using something like `$_($self:fieldname)`, which is quite similar in length to Perl5’s `$self->{fieldname}`.

Actually I have not really made an object system for Tcl; it’s just that I wanted to have the minimal tools I needed to pretend I have one, so that the rewriting effort gets much thinner when comes the day I want a real object system. To improve the syntax further, I will be able to use a macro preprocessor inside the **proc** blocks, even though there is none in Tcl, simply because the **proc** proc can be redefined, or else I can create one called **proc2** or **method** that behaves almost the same as the **proc** proc and gets called in the same way.

I know that there is a language called incrTcl which is some sort of Tcl/C++ hybrid, but given that Tcl is close enough to LISP, and that CommonLISP has its object system implemented as a *library*, why shouldn’t Tcl have the same? Anyhow, incrTcl, despite its age, is not popular enough to warrant its use. It’s usually not bundled with Tcl and usually not even on the same CD as the Tcl package in a typical OS distribution, *if* present at all.

1.5 User Interface Metaprogramming

Creating dialog boxes, element by element, is tiresome. It would be much better if dialog boxes could be created from a description of a data structure. Well, ImpureData already implements that and uses it in GUI object property dialogs and in the .pdrc editor.

2 The DRY (OnceAndOnlyOnce) Principle

OnceAndOnlyOnce a term that is used in the Smalltalk and Extreme Programming communities. DRY is a term that is used in the Ruby and Pragmatic Programmers communities. They're essentially the same: avoid code duplication. Use bottom-up thinking to unify parts that are the same. Modularise program parts in a way that separate the specific things from the generic things, so that the generic things may be merged together, and so that the specific things read more clearly. A related concept would be Data-Driven Programming, which is also quite documented.

This is the principle that motivates the reduction of the two slider classes, horizontal and vertical, to only one. I've been much more aggressively compacting code than just that, such that it now looks like a new, independent implementation of [bng], [tgl], [hsl], etc. And eventually all of the `g_*.c` subsystem.

3 A Tour of Impure Data

(This part is only in the talk, not the paper)

- Class Browser
- Name Completion
- Tool Tips
- Configurable Look and Feel
- Console
- Status Bar
- Button Bar
- Tcl Listener
- Drag-and-Drop (by Carmen Rocco)
- .pdrc Editor

References

- [1] *Tcl* by John Ousterhout, 1987; *Tcl8*, 1997.
- [2] *Tk*, a GUI toolkit for Tcl.