Subject: Re: object toddling
Posted by davidf on Fri, 30 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Martin Schultz (mgs@io.harvard.edu) writes:

> inspired by my youngest daughter who is taking her first attempts to
> reach higher objects ;-) by climbing, I thought I should finally give
> objects a try as well (for purely educational purposes ... yet).

Just for the record, Martin, I would be perfectly
content if you used your copious amounts of free time
to fix that Explore program of yours and put the output
in draw widgets. But...since you bring objects up.

> Here is what I don't understand about the IDLgrAxis objects:
>
> * why do you have to specify X, Y, or Z for the COORD_CONV keyword? If
> it is really only one axis, say X, there shouldn't be any need to refer
> to Y or Z.

How else could you get that nice, fat 3D axis? Your problem
is that you are thinking in 2D space, when you want to be
thinking in 3D space. (I'll concede that it is more natural
to think of axes as 2D or even 1D objects, but you choose
the example to build, not me.) The IDL object graphics
system is a 3D system, for better or worse.

> That should be a matter of the Model class, shouldn't it?

It *can* be a property of the Model class. In fact, it often
is. But most of these graphics primitives that RSI provides
can also be scaled and translated individually. How you do
the scaling is completely up to you. You do the same thing
whether you do it to the model or to the primitive.

> * apparently it is not possible to specify coordinate transformations
> other than linear. In my daily life, I often deal with pressure
> coordinates or sometimes with Arrhenius plots (i.e. 1000/T). In my
> understanding of OOP, there should be a method IDLgrAxis::CoorConv which
> would take care of this, and when you build a plot, you would have to
> choose between e.g. a IDLgrLinearAxis, IDLgrLogAxis, IDLgrArrheniusAxis,
> IDLgrPressureAxis, etc. Does this make sense?

Sure. I presume this is why you are building your own axis, right?
If you want a pressure axis, you are free to build it and write your
own CoorConv method that scales the axis appropriately for your
intended use. That is exactly the point of objects.

> * if the last point is reasonable: how do you deal with on-the-fly
> object creation (example: a hypothetical widget driven plot program
> would create a plot with linear axis as default, but the user should be
> able to switch these to any other defined axis type per drop down list).

I would probably write that CoorConv method you want to
write above to accept an argument that indicates what
kind of axis you want to build: 0 means linear, 1 means log,
2 means pressure, etc.

Changing the axis on the fly only means calling the SetAxisType
method you are also going to write to change the scaling appropriately.

> * here comes the most subtle and severe problem: is there any way to
> dynamically change methods of a superclass? Say you have a
> grAxis::DrawLine method which would take care of drawing the basic axis
> line. Now someone wants to change the look of the axis line e.g. by
> giving it a color shading or a 3D effect (pretty gimmicky but it's just
> an example). Just defining another subclass which would take care of
> this wouldn't work, because then you "loose" all the derived axis like
> linear, log, etc.). On the other hand, if you have to rewrite the
> grAxis::DrawLine method, you can't go back (in a sense) and you wouldn't
> be able to do so if the grAxis object was only available as a SAV file.

Object methods are just regular IDL procedures and functions,
so creating a "new" superclass draw method would really only
involve compiling a "new" IDL program module with the correct
name. That would take care of SAVE files.

But I fail to see the need for this, and I think you
may be misunderstanding how the draw methods for these objects
will really work. I don't think of them, necessarily, as "layering"
one draw method on top of another. I think of the method as working
with a bunch of graphic primitives to build the object you want to
display.

> There must be some solution to this, because it somehow resembles the
> problems that you have with different devices. Maybe the trick is to
> "atomize" even further and have grAxis consist of grLine objects which
> could then be replaced by grFancyLine? But how do you do this? And how
> do you do this dynamically? And where do you stop atomizing?

I think this is the correct approach, although I have to admit
I feel a little unsure what question you are really asking here.
You stop "atomizing", as you put it, when you get to the basic
atomic primitives.

> Let me give
> another example where this could be a problem: data transformations.
> Say, we have this great program to plot any data any way we like (of
> course with the help of a nifty object hierarchy), but now you want to
> perform calculations with the data that you are plotting. Adding a
> predefined calculator object would probably not work, because you never
> know which calculations the user wants to perform. Hence, you should be
> able to call basically any (mathematical) function from within the
> object. And when you want to combine values from two or more objects,
> this problem becomes even messier. Perhaps you could get away with some
> EXECUTE statement in the obj::Calculate method, but wouldn't that defy
> the whole concept of OOP?

I fail to see why this would defy the whole concept of OOP.
Unless you think there is some stricture in the definition of
OOP that would prohibit someone from writing boneheaded programs.
If there is, I'm quite sure there aren't enough program police
to enforce the rule anyway. :-)

But, in any case, this is what error checking is all about and you
have to do it with objects just like you have to do it with
everything else.

> What I didn't like at all when I read it, was the sentence: "Objects are
> rendered in 3 dimensions ...

"Get used to disappointment," as Welsey says in the Princess Bride.

> Shouldn't RSInc have started with a
> 2D object model and add 3D functionality as sub classes?

In retrospect!? Maybe. But they didn't. And object graphics ain't.
And they *sure* weren't going to delay the release of IDL 5.0 for
another year and a half more while they backtracked.

> I would really like to
> see an "object-shell" around the familar direct graphics.

Ah, well. Then you definitely want to be first in line to get
my new book. There is a LOT of good that can be done with an object
shell around the familiar direct graphics. These include faster
drawing, *w-a-y* faster printing, and fast development time.
(And Dick and I would be willing to sell you something if you don't
want to build your own. :-)

> If you don't think my questions are
> outright silly, then maybe I can hop on board for this kind of
> development at some point.

Wouldn't it be fun to get 10-12 people together to write programs
for 3-4 days? With the right people we would have so many good
programs come out of it that the newsgroup would be using our
programs for years to come. :-)

> One pre-requisite though: I would make it a
> strong point that plots should be readily "scalable", i.e. you would
> define a page size and panel size (and plotwindow size), and when you
> change the panel size, everything in the plot would shrink, unlike in
> normal direct graphics mode where you are never saved from suprisingly
> different charsizes, label positions, etc.

You must be reading that book I sent you! :-)

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

[Note: This follow-up was e-mailed to the cited author.]
