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

Hi all,

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). After all, I want to be able to understand what my kids are talking about once they learn IDL ;-) So, I read a little, and quite early there were some things in the OOP guide which seemed strange to me. I had done one previous excursion into object land a couple of years ago (when Borland Pascal added that functionality), and as far as I remember, I got stuck at a similar stage. As a nice example, I thought of writing my own grAxis object -- virtue being that something like this already exists so one could peek into the problem solution once a while.

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

* 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?

* 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). Do you have to code in a case statement like:

```
if (type eq 0) then left_axis = obj_new( 'grLinearAxis', ...)
          1                    'grLogAxis',...
```

... Hey! I guess you can refer to the class via a string variable:

```
AxisClass = ClassName[WidgetInfo,listID,select=select]
left_axis = obj_new( AxisClass, common_options )
```

true?

* 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. 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? 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? After all, you don't know from within the object what this "classic" function would do to your data!

What I didn't like at all when I read it, was the sentence: "Objects are rendered in 3 dimensions ... As a result, the time needed to render a given object ... will often be longer than the time taken to draw the analogous image in Direct Graphics." Shouldn't RSInc have started with a 2D object model and add 3D functionality as sub classes? Or was this not feasible because of problems similar to the ones described above (or much trickier ones which I am too novice to see)? I would really like to see an "object-shell" around the familar direct graphics, I guess this was something David F. has in mind ? 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. 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.

David: are you dealing with these issues in your forthcoming book?

Thanks for any helpful replies,
Martin.

--

||||||||||||||||\\\\\\\\\\\\\\\----------////////////// //||||||||||||||||
Martin Schultz, DEAS, Harvard University, 29 Oxford St., Pierce 109,
Cambridge, MA 02138 phone (617) 496 8318 fax (617) 495 4551
e-mail mgs@io.harvard.edu web http://www-as/people/staff/mgs/