Subject: Re: Destroying objects
Posted by MKatz843 on Mon, 16 Feb 2004 20:10:31 GMT
View Forum Message <> Reply to Message

David Fanning <david@dfanning.com> wrote in message news:
> ...
> Talk about the yin and yang. But I think objects
> tend to be like this. When they are good they are very,
> very good; but when they are bad, they are horrid.
>
> ... Maybe it has to do with having
> to hold too much information in short-term memory. It isn't
> just *this* program you have to worry about, but its
> superclass, and the superclass above that.
>
> ...You can make the objects do what you want them to do, eventually,
> but by the time you are finished you feel like you have
> cobbled together a Rube-Goldberg contraption. You know,
> in your heart of hearts, that it just shouldn't be like
> this.

Rube would be proud of me lately. I've been working with objects in IDL for nigh on 4 years and I still consider myself a beginner. Maybe because there is no good reference I could have read to learn from. Yet I have learned some important issues along the way. I'll offer some of my thoughts.

I generally think that anything you can do with objects could be done without them, and the number of lines of code might end up being nearly the same, but with objects you can really push an "idea" forward and make things that work more elegantly than without. Once you've written a good object, you'll use it over and over without thinking about it. I also like having the ability to go in an add new methods and fields that do not break the backward compatability, but add new features to object you've been using for a while.

The hardest part is the initial debug. Probably, this is always true, but when an object crashes, the cause is often more mysterious than with straight up code. Some routine will die because it's trying to execute an object that isn't defined. Good luck tracking that down. There goes an evening. With regular code, even event-driven widgets, it's much easier to trace the steps than with object code, I find.

Due to inheritance, one object's methods can override another's of the same name. This is a truly great and powerful feature, but it can strain the short term memory, as David says. I would hate to debug someone else's object code more than their non-object code for this reason.

Lately, I've been designing objects where their fields contain other objects. Not like a container, but more like a plug-in. A specific object needs to "calibrate" and "uncalibrate" scalar values before communicating them to the outside world. Here was a great chance to design a calibration object class. Actually several: linear calibration, interpolation-based, function-based, null, etc. So the object that holds the information doesn't actually know what kind of calibration will to be performed--it just knows what other object will be doing the calibration. I think that might be hard to do in the non-object world. You would probably need pointers to functions, which (I believe) IDL does not have, but other languages do.

One main source of heartburn in objects comes from passing parameters through _Extra and _Ref_Extra. I use them all the time and am still confused by them. Since each of those calibration objects might need different fields, I let arbitrary keywords pass on through the main object in its definition, where they are then covertly passed on to the object that needs them. Now *that* can get confusing, especially if your Init routine tries to ascertain whether or not the keyword has or has not been set and comes up with a default behavior if it hasn't. You might never know that your keyword arguments didn't make it all the way to your object Init. Thank goodness for "printf debugging" as they say.

The last thing I'll share is that objects are helping me to replace poor programming in my old routines. Another case where pointers to functions might have been handy are in places where I used to use an Execute command. Since you can taylor the string, it makes it easy to be a poor programmer. Since I'm now of the school that says Execute should be avoided, I'm trying to use objects instead. Objects are created with a string argument containing their class name. Et voila. Now you can cobble together string arguments and call objects in a similar manner as you used Execute in the past. It's not as fully flexible, but what do you really need? If you're building a runtime application (which I'm working towards) you'll still have to make sure every possible object class is compiled before you go sending arbitrary strings to obj_new().

All this talk of objects and pointers is making me think of the IDL vs. Matlab threads that were going on 2 months back. For all the pros and cons on the syntax and speed, how great is it that IDL allows us to use object and pointers! to dream these great abstractions (and spend our weekends debugging.)

M. Katz