Subject: Urgent object question
Posted by Ted Cary on Wed, 23 Jan 2002 03:48:47 GMT
View Forum Message <> Reply to Message

This is a long one, so you can skip to the last paragraph if you like.

I have several object widgets that all subclass the same object widget. Each of these "satellite" object widgets is realized in its own top level base, with its own user interface, and performs a specific type of analysis on the subclassed "planet" object widget. The analysis performed depends on the state of the subclassed widget and can in turn change that state, behavior that is easily achieved using object widgets.

The problem is that I would like to subclass all of these widgets at once, making one big multi-function object widget utility with a user interface to switch between each type of analysis. Several of the analysis windows could be open at once, and the inherited object widgets could share data if needed, all changing simultaneously in response to any calls to the common superclass's methods. That was in fact my plan when I wrote everything, when I had no experience in IDL. Now that I have a little experience, it does not seem so easy.

To extend the metaphor, all of my satellites crash into one another. Of course my multi-function utility object cannot inherit all of these objects at once, since they all subclass the same object and the structure fields would conflict. I could get the structure definition to be right using Struct_Assign, but then the methods of all the objects would not be inherited. I can't easily just use object references in the place of inherited self fields because the communication between these satellite object widgets and their planet object widget has to be two-way--as soon as the user changes the state of the common superclass widget, satellite object widget fields are updated, and vise versa. With object references available to them, the satellite widgets could call methods on the planet, but they would not automatically respond to methods called on the planet by another satellite or by the user. Pointers into the objects would violate encapsulation principles and confuse me. I could just cut and past and rename all the methods and make my big object structure using Struct_Assign, creating one big object in one big file, but this is ugly and would make the program hard to maintain and read, since I'll be wanting to add more analyses in the future.

What I want is something like an Obj_Assign procedure, some way to inherit all subclassed object methods while creating a self with only one unique copy of every field, even if two subclassed objects share the same field. I suppose the problem with this is that then the Init methods of all the inherited objects might each try to initialize the same self fields, but in

my case none of the self fields would ever be Init-ialized by more than one object Init method anyway. I was hoping a more experienced programmer could help me on this one.

Does anyone have experience with a problem like this? Or any uggestions? .Fanning?

Subject: Re: Urgent object question
Posted by Ted Cary on Thu, 24 Jan 2002 16:22:05 GMT
View Forum Message <> Reply to Message

Hi Davids,

I did not work out all the inheritance and keyword passing problems in detail, but I think (hope) my situation is simplified because each object in my container is of a different class. There's a one-to-one correlation between ISAs and HASAs in this case. I also (sort of) put the subclassed container object inside of the very container it subclasses, but I did this by passing an object reference to it as an argument to the Init method of every single object in the container. This means that the contained objects have to be written so that they know they are going to be contained in the subclassed container, but this was the only way I could figure out how to do the two-way communication I was speaking of before. If I'm doing something incredibly dense, please tell me--I created my first object ever only two weeks ago, and I need to finish this program soon... which is why I said "urgent" in the original message.

Thanks

Ted Cary
tedcary@yahoo.com

"David Burridge" <dave@clogic.f9.co.uk> wrote in message news:oxV38.1473$k91.79975@wards...
> Hi Ted,

> The way I see it, using the IDL_Container (or a subclass if you need some
> specific
> behaviour) has a ton of advantages. The only problem with it is 1) you need
> to be
> absolutely clear on the ISA vs HASA relationship and 2) you can have

get/set
> calls
> flying everywhere - causing infinite loops and making traceback
impossible!
>
> By only passing unresolved get/set keyword requests to the parent object,
> we've cut
> down on the tracking problem and eliminated infinite loops. This is
> relatively easy
> using the keyword inheritance methods. Secondly, we've buried all the
> mechanics
> in a single superclass (e.g. inheriting IDL_Container and passing
unresolved
> keywords to parent container objects) so that the hierarchy can be created
> almost
> transparently by simply inheriting our top-level object. Last of all, by
> accepting the
> parent class as a param to the init method, our object adds ITSELF to the
> container,
> adding to the transparency effect.
>

>
>