Subject: Re: Object Widgets
Posted by J.D. Smith on Wed, 17 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Struan Gray wrote:
>
> J.D. Smith, jdsmith@astro.cornell.edu writes:
>
>> http://www.astro.cornell.edu/staff/jdsmith/objmsg/objmsg.htm l
>>
>> Anyway, comments are welcome.
>
> Nice stuff. Thanks for the sneak peek.
>
> I have something similar (though I got lazy and subclassed from
> IDL_Container rather than write my own list manager), and am still
> muddling over whether to make the actual messages objects as well.
>
> At some point one has to start deciding how the message should be
> handled by the receiver. It looks to me as if you do that by
> modifying the ObjMsg::Message method, which implies that all the
> entities participating in the message passing have to be objects and
> have to have a Messsage method.
>
> For older widgets which I can't be bothered to objectise (fewer
> and fewer with time) I created a Gossip object which exists solely to
> turn a call like
>
> sendlist[i]->Message, msg
>
> in ObjMsg::MsgSend into a suitable widget event and put it in the
> event queue. The old-style widget then processes it with a
> conventional old-style event loop.
>
> This would obviously work in your scheme too, but I extended (or,
> more accurately, am in the process of extending) the gossip object
> idea so that even objects and object widgets generate gossip objects
> to handle messaging (they don't *have* to of course). This means an
> object can effectively have different Message methods for
> communicating with different objects. I would need to see how you
> create and hendle your msg structures, but I get the feeling I
> seperate the behaviour of the list manager and list items more
> explicitly than you do.
>
> As an example: I have a generic IDLgrModel viewer which generates
> sub-widgets in their own TLBs to control settings for the trackball,
> the viewing position, colours, etc. Instead of a single Message
> method which looks at fields (or properties) of the message to decide

> where it's coming from and what to do, I have several gossip objects
> which already know which sub-widget they are dealing with.
>
> As is often the case in OOP, all I'm really doing is avoiding a
> lot of IF or CASE statements in a single, big routine (in this case, a
> general message handler). It's a style issue, but I find this easier
> to get my head round, a little easier to adapt to specific cases, and
> a tiny bit faster as it cuts down on the need for Message methods to
> run detailed checks on every possible event type.

The Message methods of all object derived from ObjMsg are called *only* for those messages for which they have signed up (where the details of "signing up", including what information you sign up, are left to inheriting classes). This is handled by the sender object in the method MsgSendWhich. Objects are only sent the messages they want, which is typically quite few, so nothing like the long if-then-else-case jungles seen in most *event* handlers are required.

This is the key distinction between the object message paradigm and the standard widget event paradigm. It's sort of like David's general method of separate event handlers for different sources of events, but can be reconfigured dynamically during run-time, on both the sending an receiving ends, and associations aren't determined by widget hierarchy, but by the programmer, using whatever structuring he finds convenient. Granted the skeletal ObjMsg doesn't fully demonstrate that functionality, which is, as you guessed, developed by properly extending ObjMsg methods. The doc header suggest what to override/extend and how.

I guess the basic difference is one of organization. You seem to favor many specialty methods for handling "messages" (though you can't really call them that anymore when there's no standard for delivery). I typically make objects that service just some small part of an application, and receive only a few messages necessary for that functionality. I divide by making separate objects, as opposed to single objects with separate methods. This makes possible the use of objects which, thanks to having a standard shared set of interface elements, can be accessed with the minimum of fuss. One big idea using this stuff which I think is within reach is an image viewing application that supports easy to write, easy to exchange, easy to use "plug-ins", so that you can custom tailor an environment that works for you. Need statistics? Plug in a stats module. Need Convolution? Convolve module? Aperture Photometry? etc... That was the original thinking, though it hasn't played out fully yet. It's still not quite easy enough to write them (though using is pretty simple). Here is an excerpt from an end-user program which puts together some of the plug-in tools for use:

```
;; A slicer object, green
slicer=obj_new('tvSlice',tvD,COLOR=stretcher->GetColor('Green'), $
            /EXCLUSIVE)

;; a zoom in tool, green
```

```
zoomer=obj_new('tvZoom',tvD,/EXCLUSIVE,  $
            COLOR=stretcher->GetColor('Green'), _EXTRA=e)

;; a histogram tool, red, with stretcher as its color object.
hist=obj_new('tvHist',tvD,COLOR=stretcher->GetColor('Red'),/EXCLUSIVE, $
          /CORNERS,COLOBJ=stretcher,_EXTRA=e)

;; a base for our selector buttons
sbase=widget_base(base,/ROW,/FRAME,SPACE=1)

;; a box statistics tool, yellow
stats=obj_new('tvStats',base,tvD,COLOR=stretcher->GetColor('Yellow'), $
           /EXCLUSIVE,/CORNERS,/HIDE,HANDLE=2,_EXTRA=e)
```

That's basically all thats required. No need to setup message flow, etc., since that's handled inately in the ObjMsg objects themselves.

An interesting idea about a reverse compatibility object. I think a Gossip object would fit nicely into this framework.

What I really think could use some work in my stuff is the way in which messages, and the functionality they represent, can be advertised and subscribed to. Currently, this is not extensible enough. I wanted to be free to do this in different ways for different projects, which is why ObjMsg doesn't contain specifics, but now I'm realizing the utility of some reuseable paradigm.

Anyway, food for thought.

JD

--
J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-6263
304 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|