Subject: Re: Object Blues
Posted by J.D. Smith on Tue, 20 Jan 1998 08:00:00 GMT
View Forum Message <> Reply to Message

mirko_vukovic@notes.mrc.sony.com wrote:
>
> In article <34BFBBD7.B8EC1D48@astrosun.tn.cornell.edu>,
> "J.D. Smith" <jdsmith@astrosun.tn.cornell.edu> wrote:
> stuff deleted ...
>> The second complaint is that IDL classes are not permitted to define
>> public data members. That is, you can't access "someObject.somedata"
>> anywhere outside that objec'ts own methods. Perhaps RSI saw the
>> exclusion of this feature as a simplifying decision, and one that
>> guarantees encapsulation. Indeed, limiting object access to its method
>> procedures and functions may, at first thought, seem an obvious way to
>> keep object code well-contained. However, it has other unfortunate
>> consequences, usually related to the need to acquire a small group of
>> parameters from an object for extensive and repeated use. The only
>> solution within the current context is a GetProperty method (which, as
>> you may recall, is hampered by the non-reference passing of inherited
>> keywords). This is the solution implemented by IDL's object graphics.
>> I maintain that an object method (such as most of the GetProperty's)
>> which simply returns some subset of the member data without any
>> pre-processing is wasteful and inelegant, and that the true power of
>> encapsulation is in the ability to select which parts of an object to
>> encapsulate.
> more stuff deleted ...
>
> jd,
>
> the way I work on this is to have a helper object defined, name 'obj'.
> This object has a method GetProperty. If I want to get a property of
> some object, I have it inherit 'obj'. These days as a matter of routine
> I have all my non-base objects inherit 'obj'.
>
> 'OBJ' could be expanded to include a field 'public'. This would be a
> string array that would specify an objects public members. GetProperty
> would check on that prior to returning the property.
>
> This does not look terribly inconvenient.

This solution is basically the one that RSI uses for its object graphics routines. Subclassing makes it slightly easier, but you'd still need to override the GetProperty function (meaning you can't use keyword inheritance -- which I solved by making a return structure with increasing number of fields as more properties get added). So, unfortunately, this is the same solution I am already stuck with. It requires each class contain a method (whether new or overriding) which

simply returns member data. While it may not look terribly inconvenient, it may be terribly inefficient, as shown by the example code:

*** As a main level routine:

```
a1=obj_new('thisObj')
a2={mem1:'this is a test',mem2:randomu(sd,2000,2000)}
  time=0.
  nt=1000
  for i=0,nt-1 do begin
    t0=systime(1)
;     b=a2.mem1    ;uncomment for case 1
;     b=(a1->GetProperty(/MEM1)).MEM1  ;uncomment for case 2
    t1=systime(1)
    time=time+(t1-t0)
  endfor
  time=time/(nt)
  print,'Average Time: ',time
  obj_destroy,a1
end
```

*** In a file thisobj__define.pro:

```
function thisObj::GetProperty, MEM1=mem1,MEM2=mem2
  if keyword_set(mem2) then begin
    st=create_struct('mem2',mem2)
  endif
  if keyword_set(mem1) then begin
    if is_struct(st) then st=create_struct(st,'mem1',mem1) else $
     st=create_struct('mem1',mem1)
  endif
  if is_struct(st) then return,st
  return,-1
end

function thisObj::Init
  self.mem1='this is a test'
  self.mem2=randomu(sd,2000,2000)
  return,1
end

pro thisobj__define
  t={thisobj, mem1:'',mem2:fltarr(2000,2000,/NOZERO)}
end
```

This first is for the direct structure access, the second by a

GetProperty method.

  CASE 1
IDL> .run timetest
% Compiled module: $MAIN$.
% Compiled module: THISOBJ__DEFINE.
Average Time: 8.0000043e-05

  CASE 2
IDL> .run timetest
% Compiled module: $MAIN$.
% Compiled module: IS_STRUCT.
Average Time: 0.0010100002

While these both may look small, the second is ~13 times longer. This means that calling a GetProperty method in this simple example is 13 times more costly than direct structure dereferencing. Now consider a much larger example, with more member data, and several superclasses to which to chain the GetProperty method. Such a real world example might increase the overhead tremendously, perhaps over 100. Now suppose that just you needed to access some data member of the "thisObj" class every time a motion event is generated... all of a sudden you must wait least 10 times as long *just to access a simple variable*. If the value never changes, of course it is simple (if inelegant) to make a copy of it once using GetProperty, and store it for use in the event routines, but what then if the value *is* changing (not to mention the breaking of the encapsulation philosphy by requiring a local copy be made). As you can see, this introduces a significant problem, which will only grow in magnitude with larger, more realistic applications. Note that the way I made my GetProperty method is not the most efficient (keyword passing might be faster), but one of the only ways to allow proper subclassing.

I recognize that adding public data members will also increase the overhead involved in object member dereferencing somewhat, but I severely doubt it will incur as substantial a penalty as the current option.

>
> on the other hand, can you expand on your xmanager problem? I recently
> wrote my first widget routine, and promptly dispensed with xmanager using
> widget_event instead. But I am still wondering how foolish I may have
> been.
>
> regards,
>
> mirko vukovic

As far as xmanager, for most ordinary widget applications, it serves its

purpose quite well. I recommend it to you for most small to mid-sized projects.

The problem I am addressing is how to create complex interrelationships among a large group of objects, without becoming overwhelmed by the quantity of code, and the complexity of the event-handling routine.

As an example, consider a draw widget with an image drawn, which dispenses several event types (motion, button, etc.). Suppose we requir a large number of algorithms to access those events to perform a multitude of actions (draw zoom boxes and zoom in, calculate statistics, adjust colormaps, etc., etc., etc.). The normal method (using up-the-tree event passing) would be for the draw widget's event-handler to catch all of these cases, and pass off the necessary events or event information to each routine as needed. This requires explicit coding in the event routine, if any changes are made.

My method allows any number of objects (which subclass from a prototype inter-object messaging superclass) to sign up for events from a suitably constructed draw object (constructed, in fact, by subclassing the same prototype). At runtime, more such objects can be added to the list of event recipients, their requested events and status can be changed, or they can be removed from the list entirely. Total flexibility. In fact, you are free to invent your own "messages" to do whatever you like, and have them handled in exactly the same fashion as bona-fide events.

Imagine I've worked two years on my regular event-driven image analysis tool. Now suppose I want to code a new algorithm that, say, applies a set of filters to the image in our draw widget. Rather than re-coding the event-handler for the draw widget, tracing down through all the muck to find where to explicitly plug in the new routines, and making sure I don't interfere with the event flow of the many and various other routines, I can simply subclass from my prototype, request the relevant events, and concentrate on perfecting the *algorithm*, not the bookkeeping. Simple.

Now suppose someone else wants to write little object module to work with my image draw object. I could give them my prototype specification, and their module would *magically* work with my draw widget object, without them ever having seen the code which disperses the events. Imagine 1000 such modules in an online library, each doing something different. You could download the ones you need, put them together in a small application (say ~1-2 lines for each object!) and have a custom made, high-power data analysis tool! Anyway, you get the point.

It takes a good deal of effort, but objects really do have a payoff.

JD

--
J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-4083
206 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|