Subject: Re: How Object-oriented?
Posted by davidf on Thu, 22 May 1997 07:00:00 GMT
View Forum Message <> Reply to Message

David Ritscher <david.ritscher@zibmt.uni-ulm.de> writes:

> Is anyone ready yet to comment on how object- oriented IDL 5.0 is?

I am *not* an expert on OOP. My only qualification here is that reticence is not usually associated with my name. Please keep that in mind.

And my remarks are going to be colored by the fact that I was up until the wee hours of the morning trying to do the very simplest thing in IDL 5.0: display a surface with some axes on it. I have spent the better part of three days on this, so far without success, so I am feeling peevish about OOP just this moment. So please keep that in mind.

And finally, to give RSI every benefit of the doubt, I am still working with the beta version of the software, which everyone expects to be buggy. So *please* keep that in mind. :-)

> OO has become a big buzz word, so it has become important to at least
> give lip service to this concept. Can anyone comment on IDL 5.0?
> Is it more on the lip-service level, or are the changes significant
> enough that one can write real OO software?

I believe that RSI's commitment to OOP is more than lip service. The pointer implementation is just outstanding and I like very much the way they have implemented objects. I think the object structures are straightforward. I like the way they can be automatically initialized. I especially appreciate the almost complete flexibility you have in writing methods for your objects. And the elegance and simplicity of the Data Miner object implementation may be the best thing RSI has ever done with respect to building easy-to-use software.

I think, in short, that yes, the changes are significant enough that you will be able to begin to write real OO software. You only have to work with objects for a few minutes to start getting all kinds of ideas for powerful programs that could be written with them. I think, eventually, that objects will have as big an impact on IDL programs as widgets did when they were first introduced.

But I can remember when widgets were first introduced and how frustrated I got at times. The documentation was sketchy

at best, often wrong or misleading. The software was buggy. Things changed from one version of IDL to another, usually without warning, etc. (To give RSI credit, they usually changed for the better. I just sometimes didn't want to hear it after I had spent many days modifying my programs to work around some bug.)

I am in that frustrated mode right now after trying to write what I consider to be the very simplest graphics displays with the new object graphics routines. How in the world can something touted to be the very latest in programming innovation be so infernally hard to program!

I want to do to two simple things: (1) Display a surface with axes, and (2) Change the size of the axes labeling to something other than the grotesquely large default size. (I was actually embarrassed for RSI when I first saw the plots that came up in the Insight demo on my Windows NT machine.)

I expected to have to do more low-level programming with the object graphics system, especially with the first release of the software. But I did not expect to have to do so much low-level programming without adequate documentation. Last night I was reduced to the kind of programming I hate: making almost random changes in my programs in the futile hope that I might see a pattern in the results that gave me a clue as to what in the world was going on.

Just to give an example. I cannot figure out how to put axes on my surface so that it looks like a normal surface plot. (Yes, I know there are examples. I've run them. They work fine when the data is DIST(40), but they don't work at all if I try to display *real* data!) There is a LOCATION keyword in the Axis object that would seems to be what I need. Here is the explanation of that keyword:

> "Set this keyword to a two- or three-element vector of the form [x, y] or [x, y, z] to specify the coordinate through which the axis should pass. The default is [0, 0, 0]."

"The coordinate through which the axis should pass!?" What can this possibly mean? I can imagine an X axis having to pass through a point in the YZ plane, but beyond this I am stumped. And in any case, two hours worth of plugging all kinds of values into this keyword led to no insights. (A pun.)

And although I can now change the size of an axis title,

I have still discovered no way to change the size of the
axes annotation, beyond specifying each individual
annotation separately, which cannot possibly be what
RSI has in mind.

So, bottom line? I think RSI is moving in the right direction
with their OOP ideas. I think eventually it will revolutionize
the way IDL programs are written. But the current implementation
of object graphics (in my beta version, please remember) is
not ready for prime time.

Let's just say I'm glad I'm not an RSI technical support
engineer right now!

Cheers,

David

David Fanning, Ph.D.
Fanning Software Consulting
Customizable IDL Programming Courses
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com

Subject: Re: How Object-oriented?
Posted by davidf on Fri, 23 May 1997 07:00:00 GMT
View Forum Message <> Reply to Message

Stein Vidar Hagfors Haugan <s.v.h.haugan@astro.uio.no> writes
in a thoughtful article:

> Unlike David Fanning, however, I'm not completely happy with the
> pointer implementation - as far as I have understood it, you
> have no "address of" operator. I.e., you cannot *mix*
> normal and pointer variables in the sense that you cannot
> make a pointer variable point to a normal variable like you
> can in e.g., C, and thus change (or read) the contents of that
> variable by using the pointer.

A little experimenting convinces me that Stein is right
about this, but I am nevertheless still pleased with the
implementation. What if RSI *had* made it possible to
point to normal variables? I can imagine all hell breaking
loose within IDL as variables were willy-nilly redefined
and pointers were now pointing everywhere or nowhere.
I presume the cost of the software, as high as it is, would

skyrocket, just to pay for the additional technical
support engineers RSI would have to hire to sort it out.

I think they absolutely did the best, sensible thing to
make pointers that point at heap variables, which are
global in scope and with can be reached with easily copied
and passed-around variables.

> This increases the amount of work that needs to be done to
> make existing programs benefit from pointers in conjunction
> with *new* programs. Let's say you would like to make a huge
> dataset used in an existing program available to a new
> routine (or preferably, an object). It would be nice to be
> able to pass the object a *pointer* only to this data, to
> avoid copying the data, and allowing the object to keep the
> pointer for future reference. This seems to be impossible
> (though I may have misunderstood the documentation..).

Well, I'm no great fan of the documenation, but in this case
I think you read it correctly.

> Instead, you'd have to rewrite (parts of) the existing program
> to use pointers instead of normal variables - though I
> presume routines being passed a variable by reference wouldn't
> care if the call was e.g.,
>
> my_routine,*var_ptr instead of my_routine,var

I think in some instances you will probably have to re-write
your programs to use pointer dereferences, but you are right
about the syntax above. It doesn't matter.

> I guess that the reason for the lack of an address operator
> is that "normal" variables are allocated on the stack,
> whereas pointer variables are allocated from "heap" memory...
> I.e., some nontrivial part of the information on a "normal"
> variable is kept on the stack - not just the addres of where
> that information is.. This would of course make it dangerous
> to make a pointer to a stack variable, since e.g., the pointer
> could be stored in a common block and then accessed after the
> stack variable had been deallocated (and the space possibly
> allocated to something completely else!). Or maybe it's just
> the lack of reference counting that does it - local variables
> are automatically deallocated (irrespective of where the actual
> variable is stored) on returns....

I think you hit the nail on the head. It would be exceedingly
dangerous to give real address operators out. IDL would have

to be completely redesigned to accommodate it. (And you
and I will be retired before that version is released anyway.)

> Actually, one may write object oriented programs in almost
> any language - certainly you can write OO programs in IDL 4,
> compound widgets being the obvious example of object orientation,
> though with handles etc other possibilities are clearly present.
> Over the last 2-3 years (much to my surprise and amusement) I've
> been "rediscovering" OO programming in IDL after ceasing to use
> Simula several years earlier.

I wa surprised to find out--once I looked into what
object-oriented programming was all about--that I had been
teaching many of the fundamental concepts in my widget
programming courses for several years. "Whoa", I thought
to myself, "This is easy enough that even *I* can figure
it out." :-)

Alas, I'm still bogging down on surfaces with axes, but
now I have a real version to play with. :-)

> David's (temporary, I'm sure!) frustration about the
> object-oriented graphics comes as no surprise at all - just
> think about the time we've all spent getting "up to speed"
> with all of the "direct" graphics stuf, and writing our
> own favourite procedures to do this and that exactly the way
> we want. Without having any hands-on experience of OO graphics
> in IDL 5.0, I imagine it's almost like throwing away most of that
> experience and all those neat solutions all at once, and having
> to get up to speed once more in an unfamiliar, if not hostile
> terrain!

"Hostile terrain". I like that. ;-)

I appreciate the vote of confidence, Stein. Yes, I think I will
figure it out sooner or later. They say people pay you for your
experience. No better way to get a lot of experience, I guess,
than to be willing to make a lot of stupid mistakes. I'll
struggle on. But if you run across any information about axes,
I would love to hear it. :-)

Cheers!

David

David Fanning, Ph.D.
Fanning Software Consulting

Subject: Re: How Object-oriented?
Posted by Stein Vidar Hagfors H on Fri, 23 May 1997 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> David Ritscher <david.ritscher@zibmt.uni-ulm.de> writes:
>
>> Is anyone ready yet to comment on how object- oriented IDL 5.0 is?
>
[..disclaimers from David Fanning snipped..]
>
>> OO has become a big buzz word, so it has become important to at least
>> give lip service to this concept. Can anyone comment on IDL 5.0?
>> Is it more on the lip-service level, or are the changes significant
>> enough that one can write real OO software?
>

I'm not an expert on OOP either, but coming from a university
that uses Simula as the introductory language in Computer
Science, I feel qualified to comment even though I have only
looked at the *documentation* for the OOP extensions of IDL 5.0.

(BTW - "Simula does it with CLASS" - invented around 1967(!) and
definitely a *huge* influence on the design of almost any
existing OO language - it has almost all of the features of
most "modern" OO languages, except operator overloading and
multiple inheritance)

Reading the documentation on the OOP stuff itself in IDL 5.0
doesn't take long - it's just a very few pages describing the
very basic ideas of OOP and the syntax used to implement
those ideas.

To anwer your question in short: The changes are *very* significant,
all you need to write a completely object oriented program
with an object oriented syntax is there - it's significant enough
that you could call it a "different" language altogether (OODL ?:-)
if it werent't for the fact that IDL 4.0 programs can still be run.

David Fanning:
> The pointer implementation is just outstanding ...

Unlike David Fanning, however, I'm not completely happy with the pointer implementation - as far as I have understood it, you have no "address of" operator. I.e., you cannot *mix* normal and pointer variables in the sense that you cannot make a pointer variable point to a normal variable like you can in e.g., C, and thus change (or read) the contents of that variable by using the pointer.

This increases the amount of work that needs to be done to make existing programs benefit from pointers in conjunction with *new* programs. Let's say you would like to make a huge dataset used in an existing program available to a new routine (or preferably, an object). It would be nice to be able to pass the object a *pointer* only to this data, to avoid copying the data, and allowing the object to keep the pointer for future reference. This seems to be impossible (though I may have misunderstood the documentation..). Instead, you'd have to rewrite (parts of) the existing program to use pointers in stead of normal variables - though I presume routines being passed a variable by reference wouldn't care if the call was e.g.,

```
my_routine,*var_ptr    instead of    my_routine,var
```

I guess that the reason for the lack of an address operator is that "normal" variables are allocated on the stack, whereas pointer variables are allocated from "heap" memory... I.e., some nontrivial part of the information on a "normal" variable is kept on the stack - not just the addres of where that information is.. This would of course make it dangerous to make a pointer to a stack variable, since e.g., the pointer could be stored in a common block and then accessed after the stack variable had been deallocated (and the space possibly allocated to something completely else!). Or maybe it's just the lack of reference counting that does it - local variables are automatically deallocated (irrespective of where the actual variable is stored) on returns....

Anyway - I'll live with it...it is a big improvement in notation over handles.

But back to the longer story on object orientation:

David F.:
> I think, in short, that yes, the changes are significant enough
> that you will be able to begin to write real OO software.
> You only have to work with objects for a few minutes to start
> getting all kinds of ideas for powerful programs that could

> be written with them. I think, eventually, that objects will have
> as big an impact on IDL programs as widgets did when they
> were first introduced.

Actually, one may write object oriented programs in almost
any language - certainly you can write OO programs in IDL 4,
compound widgets being the obvious example of object orientation,
though with handles etc other possibilities are clearly present.
Over the last 2-3 years (much to my surprise and amusement) I've
been "rediscovering" OO programming in IDL after ceasing to use
Simula several years earlier.

One good (textbook) example of a simple object that is not
just a compound widget is a queue. No - don't think of it as a
variable with elements - think of it (visualize it) as an
independent entity. Your program is over here on the left
side doing it's thing (like receiving and processing requests),
but requests are coming in too fast to handle. Well, you
create this "box" or "machine" or "object" or "thing"
or
whatever over there on the right side: that is a queue.
You can stuff "things" into it, and you can retrieve them.
That's all. So your "main" program could do something like

```
 requests = obj_new('queue')  ;; Make sure we do have a queue

 WHILE NOT finished DO BEGIN
   req = read_request()  ;; Read

   WHILE req NE no_request DO BEGIN ;;
     requests->insert,req  ;; Put into the queue
     req = read_request()  ;; Read in next request
   END

   ;; Process one request at a time before checking the
   ;; input again
   next = requests->next()  ;;
   IF next NE no_request THEN process_request(next)
 END
```

Now, object orientation is mostly a matter of how you choose
to organize your thoughts when dealing with a problem. You can
do this in IDL 4.0 with handles instead, with a slightly
different packaging.

```
 requests = mk_queue() ;; Make sure we do have a queue

 WHILE NOT finished DO BEGIN
```

```
    req = read_request()  ;; Read

    WHILE req NE no_request DO BEGIN ;;
      queue_insert,requests,req  ;; Put into the queue
      req = read_request()  ;; Read in next request
    END

    ;; Process one request at a time before checking the
    ;; input again
    next = queue_next(requests)  ;;
    IF next NE no_request THEN process_request(next)
  END
```

Of course, having an "object orientated language" makes the way ahead so much simpler. For example, each request could be objects (of varying type) with a "priority" function associated with them - the priority could depend on e.g., the time since the request was made (not necessarily a *linear* dependence!) etc. etc., and the queue object could be written to automatically take this into account each time a request->next() function was called etc..

Also, processing the requests may be done by a "processor" object/machine, which maintains it's own internal state, i.e., use

```
  machine = obj_new('processor')

  :
  :
     IF next NE no_request THEN machine->process,next
```

Now comes the fun of object orientation, with or without syntax geared specifically towards it:

Let's say you want to find out the difference of ignoring a certain type of events:

```
   machine1 = obj_new('processor','Machine 1')
   machine2 = obj_new('processor','Machine 2')

   :
   :
      IF next NE no_request THEN BEGIN
         machine1->process,next                    ;; Always
         IF next->good() then machine2->process,next ;; Sometimes
      END
```

Imagine the program output (let's say, printed out by
the machine->process procedure):

IDL> test_processes
Machine 2 Exploded - further messages stopped
Machine 1 Stopped - further messages stopped

On object oriented graphics:

David's (temporary, I'm sure!) frustration about the
object-oriented graphics comes as no surprise at all - just
think about the time we've all spent getting "up to speed"
with all of the "direct" graphics stuf, and writing our
own favourite procedures to do this and that exactly the way
we want. Without having any hands-on experience of OO graphics
in IDL 5.0, I imagine it's almost like throwing away most of that
experience and all those neat solutions all at once, and having
to get up to speed once more in an unfamiliar, if not hostile
terrain! It takes some time to reinvent the wheel, i.e., to get a
good understanding of how things work, and then reframing your
favourite tools as objects instead of procedures..

Regards,

Stein Vidar
