Subject: Re: IDL objected oriented question
Posted by btt on Tue, 08 Apr 2003 16:53:35 GMT
View Forum Message <> Reply to Message

Sabir Pasha wrote:

> Basically
>
> define = { ClassA, $
>
>   ObjectB: Obj_New()}
>
> END
>
> the object gets defined in
> ObjectB = Obj_New("ClassB")
>
> And somewhere we define ObjectA
>
> ObjectA = Object_New("ClassA)
>
> and now in an event handler far far away
>
> Sinfo.objectA.objectB->member function
>
> doesnt' work because we cannot access Objects A's member variables
> only member functions.

Hi,

I think you have a couple of choices. The easiest and 'safest' is to provide access to the properties (member variables) of ObjA via the GetProperty method.

ObjA->GetProperty, ObjectB = ObjB

ObjB->DoMyMemberMethodThing

You would include, in this case, an OBJECTB = OBJECTB keyword in ObectA's GetProperty method.

PRO OBJECTA::GetProperty, OBJECTB = OBJECTB, etc=etc

 ;did the caller ask for objectB?
If ARG_PRESENT(ObjectB) then ObjectB = ObjectB

;other stuff

END ; GetProperty

This does make you go through two (or more programmatic steps) but it does the job nicely.

You know, if object A is behaving like a container object, then perhaps you could make OBJECTA inherit the properties and methods of IDL_CONTAINER? Then you would have easy access to all of the objectBs, objectCs, etc. Makes cleanup and transport issues a snap. And if you like that idea, try using Martin Schultz's flavor of IDL_CONTAINER (called MGS_CONTAINER) which has a handy method for finding contained objects by name (assuming each object contained has a name, of course.)

Cheers,
Ben

Subject: Re: IDL objected oriented question
Posted by David Fanning on Tue, 08 Apr 2003 17:11:19 GMT
View Forum Message <> Reply to Message

Sabir Pasha (pashas77@yahoo.com) writes:

> I'm a relative newbie to IDL. I'm working on with classes right now.
> I have a class which has objects as member variables. At runtime via
> the famous Info structure, I find that I need to use the objects
> member functions. But lo and behold, encapsulation is implemented in
> IDL 5.6(I don't believe that it was implemented in 5.5...correct me if
> I'm wrong).

You're wrong. :-)

> Basically
>
> define = { ClassA, $
>
> ObjectB: Obj_New()}
>
> END
>
> the object gets defined in
> ObjectB = Obj_New("ClassB")
>
> And somewhere we define ObjectA
>

> ObjectA = Object_New("ClassA)
>
> and now in an event handler far far away
>
> Sinfo.objectA.objectB->member function
>
> doesnt' work because we cannot access Objects A's member variables
> only member functions.

Exactly.

Perhaps you meant to INHERIT objectB, in which case
you could use all its methods and data directly in objectA.
But perhaps not. There are good reasons sometimes to simply
have objects as members of other objects.

Working with member objects in event handlers is tough,
because, of course, you have to have some way to *get*
the object you are interested in manipulating.

One way to do this is like this:

```
info.objectA -> GetProperty, ObjectB=objectB
```

Now you can call the methods on objectB directly:

```
objectB -> DoYourThing
```

This sort of defeats the purpose of object encapsulation,
but there you are. :-)

I would argue that ObjectA is the only one who is suppose to
know anything about ObjectB (since it is member data for
ObjectA), so anything that is done to it should be done
in an ObjectA method. This means you don't have to get
ObjectB, since it is already there:

```
PRO ObjectA::SomeMethod

   self.objectB -> DoYourThing

END
```

The problem you have is that you are not in objectA's methods,
but in an event handler. A bummer. :-)

Dave Burridge and I have solved this problem with our Catalyst
Object Library by wrapping all widgets up as objects. Then widget

events automatically get sent to event handler *methods* rather
than event handler procedures. This makes it possible to write
widget programs in the normal way, but you get to take advantage
of the many lovely properties of objects, too. It is the best
of both worlds, really.

Another huge advantage of our library is that it is based on
object containment hierarchies, which means objects get cleaned
up and destroyed almost magically. You almost never have to worry
about leaking memory, one of the most annoying problems with writing
large object programs. Objects can have many "parents", or objects
that care about them (three different views of a volumetric data object,
for example), but an object will only be destroyed when all the
parents have died. In our Catalyst world, children *always* outlive
their parents. :-)

> Is there a equivalent to the "public" keyword in C++.

No, probably in IDL 6.1. :-)

(I don't know this, I only mention it for the amusement of
the IDL newsgroup regulars.)

> So I wanted to ask the IDL gurus out there, how you overcome these
> problems in very large IDL programs.

For very large programs, I use our Catalyst Library. I wouldn't
think of using anything else. For one thing, it reduces development
time by at least 25-50% by already providing a framework for building
large applications, not to mention the sizeable library of
building blocks that grow daily.

Cheers,

David
--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: IDL objected oriented question
Posted by Mark Hadfield on Tue, 08 Apr 2003 22:26:50 GMT
View Forum Message <> Reply to Message

"David Fanning" <david@dfanning.com> wrote in message

news:MPG.18fcb0a0da584bdf989b3a@news.frii.com...
> For very large programs, I use our Catalyst Library. I wouldn't
> think of using anything else. For one thing, it reduces development
> time by at least 25-50% by already providing a framework for building
> large applications, not to mention the sizeable library of
> building blocks that grow daily.

OK, you've convinced me. Where do I buy it?

--
Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

Subject: Re: IDL objected oriented question
Posted by David Fanning on Wed, 09 Apr 2003 03:19:51 GMT
View Forum Message <> Reply to Message

Mark Hadfield (m.hadfield@niwa.co.nz) writes:

> OK, you've convinced me. Where do I buy it?

Well, thank goodness. I thought *somebody* would be
interested in that nice piece of work! :-)

I'll be in contact, Mark.

Cheers,

David

P.S. If you think you would like some slick software, but
you don't want to write it yourself, now is also a VERY
good time to get a couple of very accomplished programmers
with excellent object-oriented programming skills on the cheap.
All the programs on the web page thrown in for free. :-)

--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: IDL objected oriented question

Posted by pashas77 on Wed, 09 Apr 2003 13:40:15 GMT
View Forum Message <> Reply to Message

David Fanning <david@dfanning.com> wrote in message
news:<MPG.18fcb0a0da584bdf989b3a@news.frii.com>...
> Sabir Pasha (pashas77@yahoo.com) writes:
>
>> I'm a relative newbie to IDL. I'm working on with classes right now.
>> I have a class which has objects as member variables. At runtime via
>> the famous Info structure, I find that I need to use the objects
>> member functions. But lo and behold, encapsulation is implemented in
>> IDL 5.6(I don't believe that it was implemented in 5.5...correct me if
>> I'm wrong).
>
> You're wrong. :-)
>
>> Basically
>>
>> define = { ClassA, $
>>
>> ObjectB: Obj_New()}
>>
>> END
>>
>> the object gets defined in
>> ObjectB = Obj_New("ClassB")
>>
>> And somewhere we define ObjectA
>>
>> ObjectA = Object_New("ClassA)
>>
>> and now in an event handler far far away
>>
>> Sinfo.objectA.objectB->member function
>>
>> doesnt' work because we cannot access Objects A's member variables
>> only member functions.
>
> Exactly.
>
> Perhaps you meant to INHERIT objectB, in which case
> you could use all its methods and data directly in objectA.
> But perhaps not. There are good reasons sometimes to simply
> have objects as members of other objects.
>
> Working with member objects in event handlers is tough,
> because, of course, you have to have some way to *get*
> the object you are interested in manipulating.
>

> One way to do this is like this:
>
> info.objectA -> GetProperty, ObjectB=objectB
>
> Now you can call the methods on objectB directly:
>
> objectB -> DoYourThing
>
> This sort of defeats the purpose of object encapsulation,
> but there you are. :-)
>
> I would argue that ObjectA is the only one who is suppose to
> know anything about ObjectB (since it is member data for
> ObjectA), so anything that is done to it should be done
> in an ObjectA method. This means you don't have to get
> ObjectB, since it is already there:
>
> PRO ObjectA::SomeMethod
>
> self.objectB -> DoYourThing
>
> END
>
> The problem you have is that you are not in objectA's methods,
> but in an event handler. A bummer. :-)
>
> Dave Burridge and I have solved this problem with our Catalyst
> Object Library by wrapping all widgets up as objects. Then widget
> events automatically get sent to event handler *methods* rather
> than event handler procedures. This makes it possible to write
> widget programs in the normal way, but you get to take advantage
> of the many lovely properties of objects, too. It is the best
> of both worlds, really.
>
> Another huge advantage of our library is that it is based on
> object containment hierarchies, which means objects get cleaned
> up and destroyed almost magically. You almost never have to worry
> about leaking memory, one of the most annoying problems with writing
> large object programs. Objects can have many "parents", or objects
> that care about them (three different views of a volumetric data object,
> for example), but an object will only be destroyed when all the
> parents have died. In our Catalyst world, children *always* outlive
> their parents. :-)
>
>> Is there a equivalent to the "public" keyword in C++.
>
> No, probably in IDL 6.1. :-)
>

> (I don't know this, I only mention it for the amusement of
> the IDL newsgroup regulars.)
>
>> So I wanted to ask the IDL gurus out there, how you overcome these
>> problems in very large IDL programs.
>
> For very large programs, I use our Catalyst Library. I wouldn't
> think of using anything else. For one thing, it reduces development
> time by at least 25-50% by already providing a framework for building
> large applications, not to mention the sizeable library of
> building blocks that grow daily.
>
> Cheers,
>
> David

Thanks all for the prompt reply. Yes, I thought about obtaining the objectB via a member function of Object A, but exactly as Mr. Fanning said, that would defeat the point of encapsulation. Inheritance, I think would be inappropriate in this case, because again, Object A does not need access to all of Objects B's member variables, thus breaking encapsulation again. Those event handlers are sometimes quite the monkey's wrench.

I think I'll end up using this method:

```
PRO ClassA::SomeMethod

    self.objectB -> DoYourThing

END
```

I guess i'll have to wait until the IDL includes the "public" keyword(don't hold my breath, I'm guessing??) Or as was subtly mentioned, get the Catalyst library....:)

Thanks again for the help, much obliged.

Sabir Pasha

Subject: Re: IDL objected oriented question
Posted by David Fanning on Wed, 09 Apr 2003 13:50:28 GMT
View Forum Message <> Reply to Message

Sabir Pasha (pashas77@yahoo.com) writes:

> Or as was subtly mentioned, get the Catalyst library....:)

Subtle!? I've never been any good at this marketing stuff. :-(

Cheers,

David
--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: IDL objected oriented question
Posted by Pavel Romashkin on Wed, 09 Apr 2003 16:33:37 GMT
View Forum Message <> Reply to Message

There is no question that IDL object implementation is crippled. Just as there is no question that even as it is, it is very useful.
I have been following the threads addressing heap cleanup and this one, and - what can we do? - yes, we have to take care of many things that other languages do on their own.
With regard to Catalyst, I suppose, you just right click on dfanning.com/catalyst.zip and choose "download to disk", and enjoy :-)
Jokes aside, I thought of doing a similar thing as Catalyst, but for a different reason - using small widget systems that don't even have to be visible to help objects do their business. So to say, fill the gaps in object implementation with widget events, just like in, for instance, VBA, where objects can simply listen to each other (in addition to true encapsulation and automatic cleanup :-)
Then of course, it all comes back to a global event sink, object self-awareness and transmogrification. Which ends up to be a Common variable or elusive orphaned pointer :-)
The reason I haven't done it is, I never really needed it. Shouldn't be very difficult and a fun, challenging project. Any takers?

Cheers,
Pavel

> Dave Burridge and I have solved this problem with our Catalyst
> Object Library by wrapping all widgets up as objects. Then widget
> events automatically get sent to event handler *methods* rather
> than event handler procedures. This makes it possible to write
> widget programs in the normal way, but you get to take advantage
> of the many lovely properties of objects, too. It is the best

> of both worlds, really.

Subject: Re: IDL objected oriented question
Posted by David Fanning on Wed, 09 Apr 2003 19:52:45 GMT
View Forum Message <> Reply to Message

Pavel Romashkin (pavel_romashkin@hotmail.com) writes:

> There is no question that IDL object implementation is crippled. Just as
> there is no question that even as it is, it is very useful.

I would say "limited", rather than crippled. But I agree
it is more useful than not having objects at all.

RSI has a problem, though. Objects are real programming
tools. But because they are somewhat limiting (e.g., no
public data, no operator overloading, etc.) they don't
appeal, particularly, to the C++ programmer. And they appear
too complicated to the average scientist who is not necessarily
a computer programmer. Yet IDL is seeming sold to
people (it appears to me) with less and less programming
background. How in the world can we get people with
limited programming expertise to use the tools that will
benefit them the most in their programs?

I think RSI's answer to this is to build yet more
sophisticated object tools that can be used from the
IDL command line. All well and good if the tool does
what you want it to do. Which I'm sure it does, some
of the time. :-)

But what about when it doesn't? What if you don't care
to be pushed into really sophisticated object graphics?
What if you just wanted to build your own little tool,
using the direct graphics commands you already know and
understand, in a simple graphical interface, taking
advantage of the numerous benefits of objects? Then,
I'm afraid, RSI has not made life particularly easy for you.

Dave and I have struggled with these questions for over
a year now. We are still struggling a little bit with the
proper amount of "abstraction" to build into our library.
We wish to make the library easy to use, but not opaque
to the average user. We want scientists to be able to use
it easily. We realize it is a quixotic mission (and Dave
tells me that with my programming skills I'm a natural
for the role of Sancho), but we think it can be done.

We have already proven that these simple tools can be
scaled into large applications.

And because we have based our object framework around
the concept of a widget framework, we have shown that
even programmers with just basic widget programming
experience and no prior knowledge of objects can
easily learn to program in this system. In fact,
our library probably has a more consistent interface
for the user than even widgets themselves.

> With regard to Catalyst, I suppose, you just right click on
> dfanning.com/catalyst.zip and choose "download to disk", and enjoy :-)

Yes, something like that. You will have to throw a few
coins in the jar, too. :-)

Dave and I have spent about 8 man-months building this
system in our "spare" time. We plan to use it in our own
consulting work because it will give us an edge in building
applications faster than the competition. We haven't decided
exactly how we are going to sell it. (I've always belonged to
the oh-hell-give-it-away-and-be-famous school, but Dave has
a young family to think about.) But, of course, I can't
resist writing about it. We have solved a LOT of problems
associated with marrying objects and widgets. *Someone* needs
to know about that!

> Jokes aside, I thought of doing a similar thing as Catalyst, but for a
> different reason - using small widget systems that don't even have to be
> visible to help objects do their business.

Yes, exactly. A widget can be one of several physical
manifestations of an object. Another could be a VB control, etc.
Objects can have different physical manifestations at different
times, etc. Objects are infinitely flexible, it seems.
Most of our objects are built with the ability to call
a "Control Panel" that is a physical manifestation that allows
the user to change the properties of the object interactively.

> So to say, fill the gaps in
> object implementation with widget events, just like in, for instance,
> VBA, where objects can simply listen to each other (in addition to true
> encapsulation and automatic cleanup :-)

Yes, in addition to being able to pass widget events around
the object hierarchy, our objects also have a "messaging" system
built into them. If your object is interested in registering with

my object you can do so, specifying exactly what "message" you want to receive. When my object sends a particular type of message (they are published), then any object that has registered with the object will receive the message. So, for example, a slider object could broadcast its value to three or four different draw widget objects, so all could update their image contents, etc.

> Then of course, it all comes back to a global event sink, object
> self-awareness and transmogrification. Which ends up to be a Common
> variable or elusive orphaned pointer :-)

Uh, right. But we clean it up. (At least I *think* we do. Let me check with Dave.)

> The reason I haven't done it is, I never really needed it. Shouldn't be
> very difficult and a fun, challenging project. Any takers?

Actually, it was more difficult than I thought it would be when we started it. But it really, really helps to have someone to help you with it. Solving all these problems on your own can be a real drag. :-)

Cheers,

David

P.S. This library is far from finished, but it is in good enough shape for early adaptors to play with a little bit. We are working on a User's Guide sort of thing now that explains the philosophy of what we are trying to do and how the framework works. That *will* be available on my web page when its finished. In the meantime, we are looking for someone with a good application (and maybe a little cash) who wants to see what this library can do for them. We will be more than happy to hold your hand through the entire process. :-)

--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: IDL objected oriented question
Posted by Randall Skelton on Thu, 10 Apr 2003 06:34:44 GMT
View Forum Message <> Reply to Message

I asked if any of my object programming 'feature requests' would make it

into IDL 6 and was told simply that there wasn't enough demand. Please submit your feature requests!!! Especially the object ones as the most certainly overlap with my requests ;)

My top 5 list I currently lists:

(1) operator overloading
(2) proper inheritance mechanisms (i.e. avoiding namespace clashes that occur in structures)
(3) public / private methods
(4) a C-API interface to the object/heap variables.
(5) Map objects

Cheers,

Randall

Subject: Re: IDL objected oriented question
Posted by tam on Thu, 10 Apr 2003 12:49:37 GMT
View Forum Message <> Reply to Message

David Fanning wrote:

.... much discussion about an object library ....
> P.S. This library is far from finished, but it is in good enough
> shape for early adaptors to play with a little bit.

Usually one talks about early adopters, but I daresay
you're right to call the first users of an object oriented library
early adapters! Just hightlights the flexibility of the interface.

Regards,
Tom McGlynn

Subject: Re: IDL objected oriented question
Posted by JD Smith on Sun, 13 Apr 2003 04:35:20 GMT
View Forum Message <> Reply to Message

On Wed, 09 Apr 2003 09:33:37 -0700, Pavel Romashkin wrote:

> There is no question that IDL object implementation is crippled. Just as
> there is no question that even as it is, it is very useful. I have been
> following the threads addressing heap cleanup and this one, and - what
> can we do? - yes, we have to take care of many things that other
> languages do on their own.

> With regard to Catalyst, I suppose, you just right click on
> dfanning.com/catalyst.zip and choose "download to disk", and enjoy :-)
> Jokes aside, I thought of doing a similar thing as Catalyst, but for a
> different reason - using small widget systems that don't even have to be
> visible to help objects do their business. So to say, fill the gaps in
> object implementation with widget events, just like in, for instance,
> VBA, where objects can simply listen to each other (in addition to true
> encapsulation and automatic cleanup :-) Then of course, it all comes
> back to a global event sink, object self-awareness and
> transmogrification. Which ends up to be a Common variable or elusive
> orphaned pointer :-) The reason I haven't done it is, I never really
> needed it. Shouldn't be very difficult and a fun, challenging project.
> Any takers?
>

I certainly haven't held my tongue when it comes to complaining about IDL's object system, but I think you may be overlooking some relevant points. There is not single, definitive object system or object programming paradigm. People spend countless years on newsgroups less reputable than this one arguing mundane and esoteric points such as the utility vs. harmfulness of, e.g., operator overloading. When you look at the major OOP languages out there, IDL, quite surprisingly, seems most comparable to SmallTalk. This language was developed in the late 70's, and was the first to use object orientation. Interestingly, the main architect, Alan Kay, was quoted, "I invented the term `Object-Oriented', and I can tell you I did not have C++ in mind."

IDL, like SmallTalk, practices total information hiding: instance variables (the "fields" of the class-struct) are only accessible within the class. Also like SmallTalk, methods are always fully public. The only places it seems to deviate is in permitting multiple inheritance (SmallTalk only permits single inheritance), and of course the fact the SmallTalk is "pure" OO -- everything is an object, whereas objects were grafted onto IDL somewhat inelegantly after 20 years without them. The fact that these policies seem limiting is more a statement of current OO languages of choice (C++,Java, etc.), which provide more facilities for access control and encapsulation, than of any consensus of best-practices. What's interesting is that most authorities, even of languages which allow it, consistently discredit the use of explicit instance variables outside of the class itself. That's not to say I enjoy all the GetProperty calls scattered about my code. The problem is, if I were allowed to access all of those fields directly, being the lazy person I am, my objects would turn into glorified structures with the single notational convenience of not needing to pass the struct into a procedure.

E.g

st_object->Print

vs.

Print,st_object

The real point of an object is to hide as many operational details as possible from it's user. This isn't always convenient. But it does pay off in the long run in terms of re-use and code isolation. There are plenty of places IDL's objects could use improvement, but strict comparison against the questionable feature-set of some popular object-based languages may not be the most productive avenue for discovering them.

JD
