Subject: Re: What about real polymorphism ??
Posted by David Fanning on Thu, 09 Dec 2004 15:00:54 GMT
View Forum Message <> Reply to Message

Antonio Santiago writes:

> my problem is with IDL's polymorphism, i think it's a half-polymorphims
> instead real polymorphism.
>
> Supose we have a class Person and two subclasses Man and Woman. Class
> Person has a class called GetInfo() that is overriden in both Man and
> Woman. The question is:
>
> How can i create an array of Person's (that is Man or Woman objects) and
> call the method GetInfo() in the way that depends on the subtype of
> every object it invoques the Man GetInfo() or the Woman GetInfo()??
>
> One possible solution is using OBJ_ISA funtion for every object in the
> array but i want to know if it is possible only with polymorfism in IDL
> like in Java or other OO languages.

I don't understand this question. (Probably I don't
understand what "polymorphism" means, in IDL or anywhere
else, but leave that for a moment.) If you have an array
of mixed man and woman objects, and you call a GetInfo()
method on *any* object, how is it possible NOT to call
the correct method, assuming both man and woman objects
have overwritten the person method? I just can't see
where the problem arises.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: What about real polymorphism ??
Posted by Antonio Santiago on Thu, 09 Dec 2004 16:34:51 GMT
View Forum Message <> Reply to Message

Sorry, i think my english is little poor :)

Here are class C1 with a method called "datos" to print object data. It
prints: name and num.

C2 is a subclass of C1 and overrides "datos" method, that prints: name, num and think.

In a language like java i can "cast" C2 object to a C1 object and invoque "datos". The result is the execution of method "datos" of the C2 object.

How can i do the same with IDL? It posible in IDL to cast to superclass?

Example:

```
IDL> o1 = obj_new('c1')
IDL> o2 = obj_new('c2')
IDL> o1->datos
clase1
       1

IDL> o2->datos
class2
       2
thing 2
```

I want to "cast" o2 from C2 to C1 class and invoque "datos". A real polymorphism "detects" that o2 really is an C2 object and that it had overriden "datos" method and invoque it.

Really i have a class called VOLUM that draws some kind of data. My data can be en cartesian or polar data, then my idea is to create two derived clases VOLUM_CART and VOLUM_POLAR that overrides some methods of VOLUM (for exmaple: "draw_data") and extend other news.

I want my application has some number of objects VOLUM, that can be VOLUM_POLAR or VOLUM_CART. From application point of view they are only VOLUM object. Then when executes the method "draw_data" depends of type of object VOLUM (VOLUM_CART or VOLUM_DATA) i want IDL executes VOLUM_CART::draw_data or VOLUM_POLAR::draw_data.

In Java, C++, ... it is easy but i think it is not possible in IDL.

Mmm... by other hand... while i write this message :) I supose that like you say i can create an object array, assign different type object and

invoque the xxx method on every object.

The problem is that IDL can't brings me the possibility of abstract the
concept of VOLUM_CART and VOLUM_POLAR to a more generic class VOLUM.

To finshing, i think i answer my self :)

Thanks.

```
PRO c1__define

   struct = { $
     c1, $
     name: '', $
     num: 0 $
     }
END

FUNCTION c1::init

   self.name = 'clase1'
   self.num = 1
   return, 1
END

pro c1::datos
   print, self.name
   print, self.num
END



PRO c2__define
```

```
    struct = { $
      c2, $
      INHERITS c1, $
      thing: '' $
      }
END

FUNCTION c2::init

    r=self->c1::init()

    self.name = 'class2'
    self.num = 2
    self.thing = 'thing 2'
    return, 1
END

pro c2::datos
    print, self.name
    print, self.num
    print, self.thing
END
```

File Attachments

1) c1__define.pro, downloaded 76 times
2) c2__define.pro, downloaded 73 times

Subject: Re: What about real polymorphism ??
Posted by David Fanning on Thu, 09 Dec 2004 16:57:39 GMT
View Forum Message <> Reply to Message

Antonio Santiago writes:

> Sorry, i think my english is little poor :)

It's good enough for this crowd. :-)

> In a language like java i can "cast" C2 object to a C1 object and
> invoque "datos". The result is the execution of method "datos" of the C2
> object.
>
> How can i do the same with IDL? It posible in IDL to cast to superclass?
>
> ----------
>
> Example:

>
> IDL> o1 = obj_new('c1')
> IDL> o2 = obj_new('c2')
> IDL> o1->datos
> clase1
> 1
>
> IDL> o2->datos
> class2
> 2
> thing 2
>
>
> I want to "cast" o2 from C2 to C1 class and invoque "datos". A real
> polymorphism "detects" that o2 really is an C2 object and that it had
> overriden "datos" method and invoque it.

I don't know a thing about JAVA, but didn't this just
DO what you want it to do? You invoked the DATOS method
on o2, IDL knows this is a c2 object, so called the correct
method. What did you want, if not that?

Now that I think about it, what may be confusing you
is how object methods get *attached* to objects. If you
create object c1 with a DATOS method. Then create object c2
by INHERITing c1, but do NOT create it with a DATOS method,
it will use the c1 DATOS method when DATOS is called. We
agree on this.

But now, if IN THAT SAME IDL SESSION, you create a DATOS method
for c2 (after you have already created an instance of c2 in that
IDL session), then when you call the DATOS method, the c1 DATOS
method will STILL be called! It is as if the c2 DATOS method
doesn't exist. You will have to do a .RESET to get the c2 DATOS
method attached to a c2 object. The IDL documentation is silent
on this little point. You learn it only by experience.

Could this be what is causing you problems? Otherwise, I think
IDL acts exactly like you hope it will act.

> To finshing, i think i answer my self :)

Ah, the secret of the IDL newsgroup. Reading is useless.
Understanding comes only from the act of hitting the SEND
button. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: What about real polymorphism ??
Posted by Karl Schultz on Thu, 09 Dec 2004 17:12:25 GMT
View Forum Message <> Reply to Message

"Antonio Santiago" <d6522117@est.fib.upc.es> wrote in message
news:cp9urgrpj1@defalla.upc.es...
> Sorry, i think my english is little poor :)
>
> Here are class C1 with a method called "datos" to print object data. It
> prints: name and num.
>
> C2 is a subclass of C1 and overrides "datos" method, that prints: name,
> num and think.
>
> In a language like java i can "cast" C2 object to a C1 object and
> invoque "datos". The result is the execution of method "datos" of the C2
> object.
>
> How can i do the same with IDL? It posible in IDL to cast to superclass?

Do you want:

c2->c1::datos

???

This will invoke the datos method defined in c1.

Karl

Subject: Re: What about real polymorphism ??
Posted by Antonio Santiago on Thu, 09 Dec 2004 17:32:41 GMT
View Forum Message <> Reply to Message

> Do you want:
>
> c2->c1::datos
>

> ???
>
> This will invoke the datos method defined in c1.
>
> Karl
>

Not exactly.

I want to view C2 as a C1 class object but when i invoque "datos" it
will be execuete the method defined in C2.
The problem is the point of view. I want to view like in java or other
OO language but IDL cant accept this point of view :(

With PERSON, MAN and WOMAN is more clear. I want to view a MAN or a
WOMAN only as a PERSON, something like a cast. Later when i execute
methods over this PERSON it will execute the "datos" method of WOMEN or
MAN depending of its subtype (that's polymorphism).
But i supossed this point of view is not possible in IDL.

My solution is to use an OBJ_ARR, put objects WOMAN and MAN in it and
consider that all are objects PERSON, then execute "datos" method on
every object of the array.

Thanks anyway for your time.

Subject: Re: What about real polymorphism ??
Posted by David Fanning on Thu, 09 Dec 2004 17:45:41 GMT
View Forum Message <> Reply to Message

Antonio Santiago writes:

> I want to view C2 as a C1 class object but when i invoque "datos" it
> will be execuete the method defined in C2.
> The problem is the point of view. I want to view like in java or other
> OO language but IDL cant accept this point of view :(

It is possible that no one who works with IDL
understands this point of view, I guess, but
I still don't get it. :-(

I think the problem I'm having is over the word "view".
What does that mean? How does one "view" an object?
If you want to know if c2 "is a" c1 object, it is easy
enough to tell:

```
PRINT, Obj_Isa(c2, Obj_Class(c1))
```

If you want to treat it "as if it were" a c1 object:

  c2 -> c1::Datos

How else would one "view" it?

> With PERSON, MAN and WOMAN is more clear. I want to view a MAN or a
> WOMAN only as a PERSON, something like a cast.

What do you think is preventing you from doing this?

> Later when i execute
> methods over this PERSON it will execute the "datos" method of WOMEN or
> MAN depending of its subtype (that's polymorphism).

Yes, and... I miss the point. This is *exactly* what IDL does.

> But i supossed this point of view is not possible in IDL.

Donno. I can't really tell what this point of view is yet. :-(

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: What about real polymorphism ??
Posted by Antonio Santiago on Thu, 09 Dec 2004 17:59:26 GMT
View Forum Message <> Reply to Message

Hi another time :)
this seems more a chat program than a news group, we are very quickly :)

This is only to talk because my problem was solutioned with the lastest
messages.

>> Later when i execute
>> methods over this PERSON it will execute the "datos" method of WOMEN or
>> MAN depending of its subtype (that's polymorphism).
>
>
> Yes, and... I miss the point. This is *exactly* what IDL does.

Thats ok. But you know that your object "is a" WOMEN or a MAN because:
1) you have created it "o=obj_new('WOMAN')" and then you know its methods or
2) you request its type with OBJ_ISA() funtion and then need to know it methos to invoque one.

I like the idea of use OBJ_ARR because i dont need to know the type of object it containes (WOMEN or MAN) o only know that this objects are PERSON's that has the method "datos". I can invoque that method on all this and the "polymorphism" does the rest.

Anyway if you come to Spain int he future and goes to Barcelona, call me and i invite you to have a beer, wine or a glass of watter if it is necesary.

Thanks for your time and to be patient with fiker newbie :)
Bye.

Subject: Re: What about real polymorphism ??
Posted by David Fanning on Thu, 09 Dec 2004 18:50:38 GMT
View Forum Message <> Reply to Message

Antonio Santiago writes:

> Anyway if you come to Spain int he future and goes to Barcelona, call me
> and i invite you to have a beer, wine or a glass of watter if it is
> necesary.

Well, I was just in Spain a couple of weeks ago. And had I known there was a beer waiting for me, I would have headed to Barcelona rather than to wonderful Toledo on the weekend. But Spain is lovely. I hope I can go back again soon. :-)

> Thanks for your time and to be patient with fiker newbie :)

We *love* newbies here. We get terribly tired preaching to the choir.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.

Subject: Re: What about real polymorphism ??
Posted by tam on Thu, 09 Dec 2004 20:11:19 GMT
View Forum Message <> Reply to Message

>
> Thats ok. But you know that your object "is a" WOMEN or a MAN because:
> 1) you have created it "o=obj_new('WOMAN')" and then you know its
> methods or
> 2) you request its type with OBJ_ISA() funtion and then need to know it
> methos to invoque one.
>
>
I suspect that the real difference between IDL and Java is that in Java
you need to deal with two different classes/types: In Java if you have some object O,
you have the actual class of O, but you also have the type associated
with the variable that references O, i.e., you can say:

```
Superclass o = new Subclass();
```

which creates an object that has an actual class of 'Subclass' but is referenced
as if it were an object of type 'Superclass'. Once you have this distinction
a language needs to have rules as to whether the methods that will be invoked
with a reference of the form
o.method() [or in IDL o->method()]
is the method associated with the Superclass or the Subclass. In Java it can be
either depending upon the details of the method.

In IDL (similarly in Smalltalk or Perl)
you don't have this distinction, since there are no typed variables.
The only class you need to worry about is the class of the object itself and
everything works polymorphically by default.

Of course occasionally you want to be able to explicitly use a superclass's methods
even though a subclass overrides it (e.g., when adding just a little functionality
to the method). In Java you do this with the 'super' prefix. In IDL you use
the name of the superclass.

```
o->Superclass::method()
```

Hope this clarifies things a bit.

Regards,
Tom McGlynn

Subject: Re: What about real polymorphism ??
Posted by Michael Wallace on Thu, 09 Dec 2004 20:25:22 GMT
View Forum Message <> Reply to Message

> I don't know a thing about JAVA, but didn't this just
> DO what you want it to do?

Quite a thread you guys have here. Anyway, I can't speak for objects in IDL since I haven't actually learned how to use them yet, although I keep planning to. I know exactly what Antonio is saying, so I'm going to try and give another example of how things work in Java. Maybe seeing this will help you IDL folks to better understand the Java side of the question. I'm going to be using Java code in my example, so I hope you can follow it. ;-)

I'll define three classes: Shape, Square and Circle. The Shape class defines a method called area() which returns the area of the shape. As a default, the area() method of Shape will return 0.0. Square and Circle both inherit from Shape and override the area() method.

```
public class Shape {
    public float area() {
        return 0.0;
    }
}

public class Square extends Shape {
    private float length;

    public float area() {
        return length * length;
    }
}

public class Circle extends Shape {
    private float radius;

    public float area() {
        return Math.PI * radius * radius;
    }
}
```

Now let's say that we had a utility class that works with Shapes and one of its methods creates a nice string representation of the shape for us.

```
public class ShapeUtil {
    public static String printShape(Shape s) {
        return "Shape area = " + s.area();
    }
```

}

This class is totally unaware of what kind of Shape was originally instantiated. The Shape could just be an instance of Shape or it could be an instance of Square or Circle. Even though we don't know the type of Shape sent in, each Shape knows its own type and so it calls the area() method specific to its type. If the Shape happens to be a Circle, the area() method of Circle is called. If Shape happens to be a Square, the area() method of Square is called. We don't need to know which specific Shape was instantiated in order to get the area of the Shape.

I believe the original poster was trying to say that in IDL when you have a case like this, where you only have knowledge of the superclass, the superclass method was getting called instead of the overriding subclass method even if the object was instantiated as the subclass.

Again, I don't know what happens in IDL since I haven't yet learned how objects work in IDL, but I believe this describes the behavior the original author was seeing. Hope all this makes sense. If not, feel free to ignore this post -- that'll be nothing new for me. ;-)

-Mike

Subject: Re: What about real polymorphism ??
Posted by David Fanning on Thu, 09 Dec 2004 20:52:48 GMT
View Forum Message <> Reply to Message

Michael Wallace writes:

> Quite a thread you guys have here. Anyway, I can't speak for objects in
> IDL since I haven't actually learned how to use them yet, although I
> keep planning to. I know exactly what Antonio is saying, so I'm going
> to try and give another example of how things work in Java. Maybe
> seeing this will help you IDL folks to better understand the Java side
> of the question. I'm going to be using Java code in my example, so I
> hope you can follow it. ;-)

Thank you for trying to shed light on this, Michael.
I can read your code well enough. What I can't follow
is why Antonio thinks this can't be done in IDL. :-)

Let me give you an example I use every day in IDL and
see if this isn't exactly the flavor of your example.

I have a draw widget object. I tell the object to
"draw itself" by calling its DRAW method. The draw
method does nothing more than call the DRAW methods

of any objects that happen to be in the draw widget's
container. That is to say, the draw widget object NEVER
knows what it is drawing! If I want something displayed
in the window, I just give it a DRAW method and plop it
into the Draw widget, which can always display it. It
doesn't have to know anything whatsoever about what kind
of objects populate its container.

So, in Antonio's case, if he wants to treat his MEN
and WOMEN objects as "people", the more power to him.
Anyone who interacts with one of his "people" is going
to find the proper method called without him having to
do anything extra about it. That seems like perfect
polymorphism to me. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: What about real polymorphism ??
Posted by Mark Hadfield on Thu, 09 Dec 2004 23:00:31 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Michael Wallace writes:
>
>
>> Quite a thread you guys have here. Anyway, I can't speak for objects in
>> IDL since I haven't actually learned how to use them yet, although I
>> keep planning to. I know exactly what Antonio is saying, so I'm going
>> to try and give another example of how things work in Java. Maybe
>> seeing this will help you IDL folks to better understand the Java side
>> of the question. I'm going to be using Java code in my example, so I
>> hope you can follow it. ;-)
>
>
> Thank you for trying to shed light on this, Michael.
> I can read your code well enough. What I can't follow
> is why Antonio thinks this can't be done in IDL. :-)

I think Antonio's confusion came from his experience with languages like
C++ and Java, which have a different object model from IDL. In C++ and

Java, the fact that two classes have a method called Draw does not imply these methods are related. For them to be related, the classes have to have a common superclass which itself has a Draw method. The superclass's Draw method will often be virtual, ie. not implemented, but it has to exist.

Then if you want to put objects of different classes in a container, they all have to share a common superclass and when you put them into the container they kind-of-lose their identity (sorry for the technical terms here) So when you take an object back you have to cast it back its superclass before you can call its Draw method.

I hope nobody who knows anything about C++ and/or Java is reading this!

Or something like that. There are entire tomes devoted to this stuff. The thing is that in IDL you don't have to worry about all this and this can be confusing for people who learn about the more strictly-type languages first.

> So, in Antonio's case, if he wants to treat his MEN
> and WOMEN objects as "people", the more power to him.
> Anyone who interacts with one of his "people" is going
> to find the proper method called without him having to
> do anything extra about it. That seems like perfect
> polymorphism to me. :-)

Me too, but the strictly-typed people find it all a bit vague & scary.

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

Subject: Re: What about real polymorphism ??
Posted by David Fanning on Thu, 09 Dec 2004 23:22:26 GMT
View Forum Message <> Reply to Message

Mark Hadfield writes:

> I think Antonio's confusion came from his experience with languages like
> C++ and Java, which have a different object model from IDL. In C++ and
> Java, the fact that two classes have a method called Draw does not imply
> these methods are related. For them to be related, the classes have to
> have a common superclass which itself has a Draw method. The
> superclass's Draw method will often be virtual, ie. not implemented, but
> it has to exist.
>

> Then if you want to put objects of different classes in a container,
> they all have to share a common superclass and when you put them into
> the container they kind-of-lose their identity (sorry for the technical
> terms here) So when you take an object back you have to cast it back its
> superclass before you can call its Draw method.

Ahhh.

> I hope nobody who knows anything about C++ and/or Java is reading this!

No chance of that, I don't think.

>> So, in Antonio's case, if he wants to treat his MEN
>> and WOMEN objects as "people", the more power to him.
>> Anyone who interacts with one of his "people" is going
>> to find the proper method called without him having to
>> do anything extra about it. That seems like perfect
>> polymorphism to me. :-)
>
> Me too, but the strictly-typed people find it all a bit vague & scary.

Then what are they doing fooling around with IDL, which
is loose and messy and often inconsistent!? You've got
to embrace chaos to hang around here long. :-)

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: What about real polymorphism ??
Posted by Michael Wallace on Fri, 10 Dec 2004 00:49:33 GMT
View Forum Message <> Reply to Message

> I hope nobody who knows anything about C++ and/or Java is reading this!

Hmmm... should I mention that I'm primarily a Java programmer and use
IDL only on the side to do plotting and data analysis? Probably not.
Don't want to upset anything on the IDL newsgroup. Hey, did I think
that or type that? .... D'oh!

> Me too, but the strictly-typed people find it all a bit vague & scary.

IDL is vague! IDL is scary! While I love the loose typing for procedural programs, I can't stand it for objects. I must admit that even though I haven't programmed with objects in IDL, I have written several object graphics programs, and each time I write one I cringe a little because of what IDL calls an "object." Using them is so backwards in some respects. Maybe this is a wrong impression, but the more I learn, the more so-called IDL "objects" appear like "glorified structs" or maybe just "some stuff thrown together."

The OO programmer in me really wants to argue the comment of IDL having "perfect polymorphism," but I'll remain civil. Gotta remember I'm on different turf than normal in this newsgroup. ;-)

-Mike

Subject: Re: What about real polymorphism ??
Posted by Michael Wallace on Fri, 10 Dec 2004 01:08:31 GMT
View Forum Message <> Reply to Message

> Then what are they doing fooling around with IDL, which
> is loose and messy and often inconsistent!? You've got
> to embrace chaos to hang around here long. :-)

We wouldn't be fooling around with IDL if we didn't have to! But this is what I get for working in space sciences community. Of course, I make things less painful by doing the majority of the work in Java and calling IDL from Java only when I must! Some of you would probably laugh at the steps I take to make Java and IDL work together. ;-)

-Mike

Subject: Re: What about real polymorphism ??
Posted by David Fanning on Fri, 10 Dec 2004 01:28:25 GMT
View Forum Message <> Reply to Message

Michael Wallace writes:

> IDL is vague! IDL is scary! While I love the loose typing for
> procedural programs, I can't stand it for objects. I must admit that
> even though I haven't programmed with objects in IDL, I have written
> several object graphics programs, and each time I write one I cringe a
> little because of what IDL calls an "object." Using them is so
> backwards in some respects. Maybe this is a wrong impression, but the
> more I learn, the more so-called IDL "objects" appear like "glorified
> structs" or maybe just "some stuff thrown together."

Well, this is the literal truth when something gets to be
at least 20 years old. But isn't it amazing that a language
like IDL can exist for that long? It is probably a miracle
that it is not messier than it is.

> The OO programmer in me really wants to argue the comment of IDL having
> "perfect polymorphism," but I'll remain civil. Gotta remember I'm on
> different turf than normal in this newsgroup. ;-)

Well, most of the IDL object programmers I know don't
even know what polymorphism means (I admit I don't), so
"perfect polymorphism" is just something that trips lightly
off the tongue. Yes, it is messy and awful and "some stuff
thrown together", but it is also so much FUN! Can't really
say that about JAVA, I don't think. ;-)

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: What about real polymorphism ??
Posted by Antonio Santiago on Fri, 10 Dec 2004 07:09:14 GMT
View Forum Message <> Reply to Message

Ok boys.
All you are invited to take a beer in Spanish (i will need to break my
moneybox :))

I supose my problem is I'm customary to more typefied languages (i dont
know if this is the right sentence in english) like Java, C, ... and i
wanted to apply the same philosofy with IDL.

Really, in my actual job the applications are grown very fast (and big)
then i think is more usefull and powerfull to use an OO programing than
the traditional procedural method, with the corresponding lost of speed
and (perhaps) eficience.
In my short experience with IDL I only have programming with Object
Graphics, I'm a completly newbie with Direct Graphics (well, a couple of
sentence :) with it).
I supose that certain algorithms, image manipulation and some other data
process (more oriented to science like hydrology, astronomy, ... than
the computer science) are right resolve with procedural programing. But

when all this algorithms are all put together in a big application
that's the moment of the computer science and, personaly, i think is
more powerfull an OO programing.

Well, bye and remember all you (and me) are invited to take a beer :)
and typical spanish "TORTILLA DE PATATAS", mmm... :)

Subject: Re: What about real polymorphism ??
Posted by JD Smith on Fri, 10 Dec 2004 16:00:58 GMT
View Forum Message <> Reply to Message

On Thu, 2004-12-09 at 13:52 -0700, David Fanning wrote:
> Michael Wallace writes:
>
>> Quite a thread you guys have here. Anyway, I can't speak for objects in
>> IDL since I haven't actually learned how to use them yet, although I
>> keep planning to. I know exactly what Antonio is saying, so I'm going
>> to try and give another example of how things work in Java. Maybe
>> seeing this will help you IDL folks to better understand the Java side
>> of the question. I'm going to be using Java code in my example, so I
>> hope you can follow it. ;-)
>
> Thank you for trying to shed light on this, Michael.
> I can read your code well enough. What I can't follow
> is why Antonio thinks this can't be done in IDL. :-)
>
> Let me give you an example I use every day in IDL and
> see if this isn't exactly the flavor of your example.
>
> I have a draw widget object. I tell the object to
> "draw itself" by calling its DRAW method. The draw
> method does nothing more than call the DRAW methods
> of any objects that happen to be in the draw widget's
> container. That is to say, the draw widget object NEVER
> knows what it is drawing! If I want something displayed
> in the window, I just give it a DRAW method and plop it
> into the Draw widget, which can always display it. It
> doesn't have to know anything whatsoever about what kind
> of objects populate its container.
>
> So, in Antonio's case, if he wants to treat his MEN
> and WOMEN objects as "people", the more power to him.
> Anyone who interacts with one of his "people" is going
> to find the proper method called without him having to
> do anything extra about it. That seems like perfect
> polymorphism to me. :-)

The confusion here is common for people coming from strictly typed languages like C++ and Java, for which you must always pre-declare the class (or common superclass) of objects you are using (e.g. the "Shape" object from Michael's example), and, at least in some languages, go specifically out of your way to get what I consider true poly-morphis (e.g. by using the "virtual" statement in C++).

Since IDL is entirely type agnostic, and doesn't require any pre-declarations in code which uses objects, *everything* is completely poly-morphic, i.e. all methods are pure virtual methods (which is one of the reasons the OO system of IDL is somewhat slower for many objects than others). You never need to know anything about the class of an individual object: it's up to the caller to ensure that it passes objects of classes which implement the methods to be called: the compiler will never complain if I compile a statement like this:

```
function obj_do,obj
  result=obj->SomeFunkyMethod(12)
  return,result
end
```

This is perfectly valid code, to which the compiler can find no objection. It doesn't know what "obj" is or is going to be, and it just hopes that the caller will pass a true object (instead of say the string "squirrel", or the value !PI), and further more an object which implements a SomeFunkyMethod function-method. This is the simultaneous joy and pain of type-free languages: the freedom to skip all those tedious type declarations, but the trouble that can get you into for large projects.

To reiterate, all method invocations are computed at run-time and not compile-time. This is similar to languages like Smalltalk, or Objective-C, but is fairly foreign to people used to working with type-driven OOP languages. It's entirely analogous to the non-existent type enforcement IDL provides for regular variables, but somehow this is less confusing for people than the typeless class analog.

All this is not to say that IDL OO paradigm is a perfect implementation. The method invocation speed is *slow*, yet data encapsulation amounts to "see no instance data", so you tend to cache pieces of information about objects for speed reasons, which *breaks* encapsulation, and furthermore leads to out-of-sync problems. There are no static or class variables, and no class methods, so you often finding yourself doing awkward things like:

```
dummy=obj_new('someclass')
new=dummy->Read(file)
```

obj_destroy,dummy

when a class method would have been much more natural. There is no generic way to refer to super-classes, which breaks method-chaining if you ever decide to substitute a different super-class, etc. But poly-morphism... that it does well.

JD

Subject: Re: What about real polymorphism ??
Posted by JD Smith on Fri, 10 Dec 2004 16:21:54 GMT
View Forum Message <> Reply to Message

On Thu, 2004-12-09 at 18:32 +0100, Antonio Santiago wrote:
>> Do you want:
>>
>> c2->c1::datos
>>
>> ???
>>
>> This will invoke the datos method defined in c1.
>>
>> Karl
>>
>
> Not exactly.
>
> I want to view C2 as a C1 class object but when i invoque "datos" it
> will be execuete the method defined in C2.
> The problem is the point of view. I want to view like in java or other
> OO language but IDL cant accept this point of view :(
>
> With PERSON, MAN and WOMAN is more clear. I want to view a MAN or a
> WOMAN only as a PERSON, something like a cast. Later when i execute
> methods over this PERSON it will execute the "datos" method of WOMEN or
> MAN depending of its subtype (that's polymorphism).
> But i supossed this point of view is not possible in IDL.

It's very possible:

```
============================================================

pro Man::AboutMe
  print,FORMAT='(%"Dude, I am a %d yr old man, and I like %s.")',self.age, $
        self.activity
end
```

```
function Man::Init,age
  if ~self->Person::Init(age) then return,0
  self.activity='watching football'
  return,1
end

pro man__define
  m={MAN, $
     INHERITS PERSON, $
     BELT_SIZE: 0.0, $
     ACTIVITY: ''}
end


pro Woman::AboutMe
  print,FORMAT= $
        '(%"Pleased to meet you.  I am a %d yr old woman, and I enjoy %s.")', $
        self.age, self.activity
end

function Woman::Init,age
  if ~self->Person::Init(age) then return,0
  self.activity='gardening'
  return,1
end

pro woman__define
  m={WOMAN, $
     INHERITS PERSON, $
     SHOE_SIZE: 0.0, $
     ACTIVITY: ''}
end

function Person::Init,age
  self.age=age
  return,1
end

pro person__define
  p={PERSON, $
     AGE: 0}
end
```

==========

Compile, and then try:

```
IDL> people=[obj_new('man',22),obj_new('woman',31)]
```

IDL> for i=0,1 do people[i]->AboutMe
Dude, I am a 22 yr old man, and I like watching football.
Pleased to meet you. I am a 31 yr old woman, and I enjoy gardening.

Sorry for the flagrant stereo-typing.

JD

Subject: Re: What about real polymorphism ??
Posted by JD Smith on Fri, 10 Dec 2004 16:23:19 GMT
View Forum Message <> Reply to Message

On Fri, 2004-12-10 at 12:00 +1300, Mark Hadfield wrote:
> David Fanning wrote:
>> Michael Wallace writes:
>>
>>
>>> Quite a thread you guys have here. Anyway, I can't speak for objects in
>>> IDL since I haven't actually learned how to use them yet, although I
>>> keep planning to. I know exactly what Antonio is saying, so I'm going
>>> to try and give another example of how things work in Java. Maybe
>>> seeing this will help you IDL folks to better understand the Java side
>>> of the question. I'm going to be using Java code in my example, so I
>>> hope you can follow it. ;-)
>>
>>
>> Thank you for trying to shed light on this, Michael.
>> I can read your code well enough. What I can't follow
>> is why Antonio thinks this can't be done in IDL. :-)
>
> I think Antonio's confusion came from his experience with languages like
> C++ and Java, which have a different object model from IDL. In C++ and
> Java, the fact that two classes have a method called Draw does not imply
> these methods are related. For them to be related, the classes have to
> have a common superclass which itself has a Draw method. The
> superclass's Draw method will often be virtual, ie. not implemented, but
> it has to exist.

Did you mean "abstract" here? Virtual methods are what all IDL methods
are: overrideable in a sub-class.

JD

Subject: Re: What about real polymorphism ??
Posted by on Fri, 10 Dec 2004 17:25:35 GMT

very clear now, i agree with you saying that : "IDL is entirely type
agnostic, and doesn't require any pre-
declarations in code which uses objects, *everything* is completely
poly-morphic, i.e. all methods are pure virtual methods ", and the example
explain it quite well. Furthermore I have tried to add the same method
"AboutMe" to the <<super-class>> person like this:

```
pro Person::AboutMe
  print,FORMAT= $
        '(%"This doesn`t work and I am not a %d yr old woman, and I don`t
enjoy %s.")', $
        self.age, self.activity
end
```

and the result is the same:
> Dude, I am a 22 yr old man, and I like watching football.
> Pleased to meet you. I am a 31 yr old woman, and I enjoy gardening.

and it means that polymorphism is working, because is using the methos of
man and woman. I agree as well saying "There are no static or class
variables,
and no class methods", I mean IDL doesn't have many things of pure OO
programming so you can't do things like singleton pattern (desing patterns
by Erich Gamma, Richard Helm, ... for more information about this) i think
even IDL couldn't do many of the basics patterns of this paradigm but
anycase it didn't born to
do that, and works quite well in many other things (Visualization, matrix
operators, etc.).

A good reference for OO programming is the book thinking in C++, is for C++
programmers but the ideas of OO programming are very well explained, you can
find it in : http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html

  Good thread and thanks all.
        Esteban .

news:1102695714.17270.2.camel@turtle.as.arizona.edu...
> On Thu, 2004-12-09 at 18:32 +0100, Antonio Santiago wrote:
>>> Do you want:
>>>
>>> c2->c1::datos
>>>
>>> ???

```
>>>
>>>  This will invoke the datos method defined in c1.
>>>
>>>  Karl
>>>
>>
>>  Not exactly.
>>
>>  I want to view C2 as a C1 class object but when i invoque "datos" it
>>  will be executete the method defined in C2.
>>  The problem is the point of view. I want to view like in java or other
>>  OO language but IDL cant accept this point of view :(
>>
>>  With PERSON, MAN and WOMAN is more clear. I want to view a MAN or a
>>  WOMAN only as a PERSON, something like a cast. Later when i execute
>>  methods over this PERSON it will execute the "datos" method of WOMEN or
>>  MAN depending of its subtype (that's polymorphism).
>>  But i supossed this point of view is not possible in IDL.
>
>  It's very possible:
>
>  ============================================================
>
>  pro Man::AboutMe
>    print,FORMAT='(%"Dude, I am a %d yr old man, and I like %s.")',self.age,
$
>          self.activity
>  end
>
>  function Man::Init,age
>    if ~self->Person::Init(age) then return,0
>    self.activity='watching football'
>    return,1
>  end
>
>  pro man__define
>    m={MAN, $
>       INHERITS PERSON, $
>       BELT_SIZE: 0.0, $
>       ACTIVITY: ''}
>  end
>
>
>  pro Woman::AboutMe
>    print,FORMAT= $
>          '(%"Pleased to meet you.  I am a %d yr old woman, and I enjoy
%s.")', $
>          self.age, self.activity
```

```
>  end
>
>  function Woman::Init,age
>    if ~self->Person::Init(age) then return,0
>    self.activity='gardening'
>    return,1
>  end
>
>  pro woman__define
>    m={WOMAN, $
>       INHERITS PERSON, $
>       SHOE_SIZE: 0.0, $
>       ACTIVITY: ''}
>  end
>
>  function Person::Init,age
>    self.age=age
>    return,1
>  end
>
>  pro person__define
>    p={PERSON, $
>       AGE: 0}
>  end
>
>  ============================================================
>
>  Compile, and then try:
>
>  IDL> people=[obj_new('man',22),obj_new('woman',31)]
>  IDL> for i=0,1 do people[i]->AboutMe
>  Dude, I am a 22 yr old man, and I like watching football.
>  Pleased to meet you.  I am a 31 yr old woman, and I enjoy gardening.
>
>  Sorry for the flagrant stereo-typing.
>
>  JD
>
>
```

Subject: Re: What about real polymorphism ??
Posted by Mark Hadfield on Sun, 12 Dec 2004 22:07:56 GMT
View Forum Message <> Reply to Message

JD Smith wrote:
> On Fri, 2004-12-10 at 12:00 +1300, Mark Hadfield wrote:
>

>> I think Antonio's confusion came from his experience with languages like
>> C++ and Java, which have a different object model from IDL. In C++ and
>> Java, the fact that two classes have a method called Draw does not imply
>> these methods are related. For them to be related, the classes have to
>> have a common superclass which itself has a Draw method. The
>> superclass's Draw method will often be virtual, ie. not implemented, but
>> it has to exist.
>
>
> Did you mean "abstract" here? Virtual methods are what all IDL methods
> are: overrideable in a sub-class.

Yeah, that's what I meant. Something like that, anyway :-)

Actually, given my shaky knowledge about these matters, it's probably
stretching it a bit to say I "meant" anything.

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

Subject: Re: What about real polymorphism ??
Posted by marc schellens[1] on Tue, 14 Dec 2004 07:11:37 GMT
View Forum Message <> Reply to Message

> and no class methods", I mean IDL doesn't have many things of pure OO
> programming so you can't do things like singleton pattern (desing patterns
> by Erich Gamma, Richard Helm, ... for more information about this) i think
> even IDL couldn't do many of the basics patterns of this paradigm but
> anycase it didn't born to
> do that, and works quite well in many other things (Visualization, matrix
> operators, etc.).

A singleton is very possible with IDL:
Just use a global variable (also as your access point) and
check its existance in the constructor.

m
