

---

Subject: Re: OO IDL

Posted by [marc schellens\[1\]](#) on Thu, 16 Sep 2004 05:13:50 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Robert Barnett wrote:

>  
> I'm curious about common ways to call differing versions of code. I have  
> implemented OO (Object Oriented) IDL to achieve this common task and  
> wanted to know what peoples thoughts might be.  
>  
> I have several routines, each which have many different versions. In  
> many cases, no version is any more recent than any other. It's more that  
> each version is applicable for different problems.  
>  
> The programs are in their own .pro files, with the filename and function  
> name being the same so that autoloading works. They are also in  
> lowercase so that autoloading works correctly. The version is just  
> appended onto the end like so:  
>  
> cost\_function\_mem.pro  
> cost\_function\_lb.pro  
> cost\_function\_sr.pro  
> ...  
>  
> simplex\_fast.pro  
> simplex\_slow.pro  
> ...  
>  
> ... and on it goes  
>  
> This means that I have to do lots of calls to CALL\_FUNCTION becuae I  
> only know what version I am to use at runtime.  
>  
> I'm having a play around with OO IDL and seeing if there is a way to do  
> this without using CALL\_FUNCTION, and seeing if there are any advantages  
> in doing so.  
>  
> The only way I can see to avoid the use of CALL\_FUNCTION is to create a  
> class for each function.  
>  
> mem::cost\_function  
> lb::cost\_function  
> sr::cost\_function  
> ...  
>  
> fast::simplex  
> slow::simplex  
> ...

What about the CASE statement?

CASE version of

```
'slow' : result = simplex_slow()  
'fast' : result = simplex_fast()  
ENDCASE
```

Or use a keyword in your functions:

```
result = simplex(VERSION='fast')
```

> It is now possible to call a cost function like so:

> cf -> cost\_function()

> Where cf could be

> cf = obj\_new('mem')

> cf = obj\_new('lb')

> cf = obj\_new('sr')

>

> Unfortunatley, this causes a maintainence issue with structures. I now

> also need to define

> mem\_\_define

> lb\_\_define

> sr\_\_define

> fast\_\_define

> slow\_\_define

> However, is it easy to write a trivial shell or perl script for

> generating these.

You don't want to create (and don't forget to destroy later) an object just to call a function! And how do you create it? Using CASE :-)

> It seems that both OO and CALL\_FUNCTION require the same number of lines of code aside from the maintainence of the OO structures.

>

> Some advantages of OO may be

> \* The ability for objects to inherit each other, thus being able to use each others methods.

> \* Each class has its own namespace, ensuring that all methods which are not in conflict with other versions

> \* Each class could have instance data, thus saving effort in passing information down the call stack and back again.

>

> Disadvantages

> \* It may not be entirely obvious where instance data comes from

Do you mean where the object was created?

It is as obvious as where any other data came from.

> \* It may not be entirely obvious which objects inherit each other

Just look at the OBJECTNAME\_\_DEFINE procedure.

> \* A change in class struct definitions requires IDL to restart.

or at least .RESET\_SESSION, but if you use a struct instead it is the same, see below.

> The advantages of OO, although desirable don't seem to have a huge  
> impact. Makes me wonder if anyone has an IDL OO success story.

Well, I think OO isn't exactly about your problem.  
OO is about (object's) data and methods (functions) which work on  
this (object's) data (SELF).

The advantages do have huge impact in programs where  
complex data structures are used.  
If you have only arrays of fixed size and fairly simple program  
logic, probably you don't need objects indeed.

marc

---