
Subject: Re: object methods as friends

Posted by [JD Smith](#) on Tue, 20 Apr 2004 17:05:31 GMT

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

On Mon, 19 Apr 2004 10:42:24 +1200, Mark Hadfield wrote:

```
> Zorch Tierod wrote:
>> Hey there,
>> I've got an application that uses two types of objects, each with
>> several different method procedures.
>>
>> What I'm trying to do is to calculate some stuff based on the
>> interaction of one of the objects with an array of the other type of
>> objects. Illustrative example (but not the REAL application!):
>>
>> sb = {bullet, param1:.....} ;protoype for bullet object
>> st = {target, param1:.....} ;prototype for target object
>>
>> target_array = obj_arr('target',10) ;make a bunch of targets
>> projectile = obj_new('bullet') ;make a bullet
>>
>> jtarget = which_hit(target_array, projectile) ;return which target is
>> hit
>>
>> My problem seems to be that the function 'which_hit' has no access to
>> the data of either the 'bullet' or the 'target' objects, because it is
>> not a method of either - and I can't make it a method of both...
>
> The standard way of dealing with this in IDL is to expose the properties
> you need via GetProperty & SetProperty methods. Yes, this can be a lot
> of work and, no, it doesn't allow for discrimination between friend and
> non-friend callers.
>
> I suppose you could have FriendGetProperty & FriendSetProperty methods
> which asked for authentication from the caller, but really, why would
> you bother? If you don't want non-friends to access certain properties,
> then just don't do it.
```

This isn't really a solution. Accessing a "friend" property via a method call like GetProperty is far slower than direct structure access (like 10-20x). This might not be a problem in general cases, but in the event-driven world of IDL widget objects, you might need to lookup info from a friend or composited class 10-100 times a second in an event callback.

Typically, to avoid this speed penalty, you feel motivated to create a local cache of the other object's info, which isn't really the best solution, since it requires you to know how frequently and when that

object's class data is changing. This has bitten me more than once. And moreover, if you receive a pointer to your friend or composited object's data, you have to remember what the policy on freeing that pointer is: probably you should leave it up to the other object (unless of course he created it specially for you!). I have too many sections of code that look like:

```
self.myObj->GetProperty,SOMEVAR=somevar  
self.somevar=somevar
```

I'm not sure what the solution to this is. Ideally, all object data access is encapsulated behind method calls: this is the thinking that likely went into IDL's OOP paradigm (and others, like SmallTalk). However, most OOP languages tend to relax this somewhat, for exactly this reason: sometimes you just need to get the data fast! It's very possible (likely even) to abuse this freedom, but it's also likely that without it you'll create brittle workarounds like locally caching important data.

One idea I like is the "Uniform access principle" which states that "All services offered by a module should be available through a uniform notation, which does not betray whether they are implemented through storage or through computation." I.e. make data access just like method (well, method function) calls:

```
class={myClass,$  
      INHERITS superclass,  
      PUBLIC Image: ptr_new(), $  
      other_private_data:0.0}
```

```
function myClass::PI_total  
  return,!PI*total(*self.Image)  
end
```

```
IDL> o=obj_new('myClass',dist(100))  
IDL> print,o->PI_total()  
IDL> myim=*o->Image()  
IDL> *o->Image()=dist(200)  
IDL> print,o->PI_total()
```

If later you decided to change the internal storage mechanism of Image, you could then make it a method call instead of just PUBLIC data. And of course these "pseudo" method calls would translate into instance structure dereference and not have the traditional speed penalty of method calls. Efficiency comparable to structure dereference may not be possible given the need to determine if the call is traditional or pseudo, but I'm sure this can be relegated to compile time rather than run-time.

Anyway, that's my proposal.

JD
