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
