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
