Subject: Re: Object Programming in IDL
Posted by David Fanning on Tue, 21 May 2002 14:22:51 GMT
View Forum Message <> Reply to Message

Graham Wilson (graham_wilson_1234@yahoo.ca) writes:

> Just to appease Craig, I have started a new thread so I can avoid putting my
> comments after David's 'gosh golly' post ;) I am interested in hearing
> others comments...

I can't quite make out from this whether you are at
all interested in my comments, but I feel compelled
nonetheless. (This is a professional affliction, I'm
afraid.)

> The first point that we should all be very clear on is that IDL is _NOT_
> a particularly good example of an object oriented language.

One of my favorite comments about IDL was made on this
newsgroup several years ago. A newcomer to it complained,
"My God, it seems as if IDL were a programming language
that has been cobbled together over a 20 year period of time!"
Well ... indeed.

I've said before, and I still feel this strongly, that the
amazing thing about IDL is not that the object oriented
programming features are not complete, but that they exist
at all. They certainly don't exist in PV-Wave, for example.

I've really no argument to make about IDL not being a
particularly good example of an object oriented language.
(My own OO experience is so limited that I am not sure
I would recognize a good example even if I saw one, so it
is easy to concede this point.) What I object to is the
notion that just because it is not complete, it is not
useful. Objects are amazingly useful to me, and are
becoming more so every day as I learn more about
how to use them.

> You can
> certainly emulate OOP concepts using IDL's objects and a select few
> functions/proceedures but if often defeats the purpose of the OOP style.

If by "purpose" you mean reusability, ease of maintenance, and/or
ease of extensibility, then I can't see how you can support this
statement. Every one on these "purposes" is enhanced by writing
object oriented code in IDL.

It is true that you often have to be creative in how
you program. For example, I often wish that widget events
and objects were more tightly coupled, and I have had to
work hard to find ways to incorporate event handling into
objects. But I don't begrudge the creativity. Indeed, it
is one of the things that I happen to like about programming.

> When someone mentions IDL objects, it is universally assumed that they
> really mean 'object graphics'...

I grant that this is a common misconception, but it is because
people insist on reading the IDL documentation instead of this
newsgroup. If there is a misconception here, then I have seriously
not been doing my job for the past several years.

But even if it were true, and most people believe object oriented
programming means object graphics, what difference does it make?
It seems a distinction without a difference to me. Most programs
involve graphics. If you start with the object graphics library,
you will soon enough see the usefulness of writing an object that
doesn't draw something in a window.

> With regard to writing object oriented code in IDL we are all rather stuck
> until RSI implements a more complete feature set.

Well, as I say, I don't think we are "all" stuck. I don't feel
stuck at all. I feel like I am moving forward and making great
progress. I guess this could be an illusion on my part, but if
it is, at least it gives me comfort. There are advantages, I guess,
to not knowing what is going on in the rest of the world. :-)

> I generally define
> polymorphism it as the ability to process objects differently depending on
> their data type or class. In this respect, the lack of operator overloading
> is an example where IDL fails to offer the full OOP tool set. Yes, you can
> overload methods, but operators should be no different. To compensate for
> this missing functionality one can write functions and/or procedures but
> this better described as an overlay and you must rely on a naming
> convention or a path precidence to avoid conflicts. Personally, I'd like
> to see true polymorphism (with overloading) and public/private methods
> sooner rather than later (is anyone at RSI listening?).

Anything that forces better naming conventions has got to be
a plus in my book, but I find little else to quibble over here.
(Although God knows I have no idea what I would do with operator
overloading. I'm still trying to figure out the Histogram
function.) I would *love* to see public and private methods,
although since so few people actually look at code, pretty much

any method you don't explicitly document becomes private. (Just
a small joke, sorry.)

People at RSI probably are listening, but they are not probably
the ones that can do anything about this. Better to write a formal
letter outlining your concerns. Always start the letter "I love
IDL but..." It has more effect that way. :-)

> For what it is worth, Matlab has a slightly more complete implementation of
> OOP. The one glaring (and annoying) feature missing from Matlab, however,
> is the absence of pointers and therefore dynamic structures/sizing. This,
> of course, is a grip for a different newsgroup...

Alas, no software that has been around for any length of time
is free of baggage that can make some ideas impossible (or at
the very least, not cost effective) to implement.

Cheers,

David

--
David W. Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Object Programming in IDL
Posted by James Kuyper on Tue, 21 May 2002 15:18:35 GMT

View Forum Message <> Reply to Message

Graham Wilson wrote:
...

> With regard to writing object oriented code in IDL we are all rather stuck
> until RSI implements a more complete feature set. I generally define
> polymorphism it as the ability to process objects differently depending on
> their data type or class. In this respect, the lack of operator overloading
> is an example where IDL fails to offer the full OOP tool set. Yes, you can
> overload methods, but operators should be no different. To compensate for
> this missing functionality one can write functions and/or procedures but
> this better described as an overlay and you must rely on a naming
> convention or a path precidence to avoid conflicts. Personally, I'd like
> to see true polymorphism (with overloading) and public/private methods
> sooner rather than later (is anyone at RSI listening?).

I won't try to defend IDL as an OO language; that's not it's heritage.
OO was tacked on long after the initial design, and it shows.

However, I think you're over-estimating the importance of operator
overloads. C++ has operator overloading, and the judgement of the
experts I've read seems to be that it can be more of a trap than a
useful feature. The basic criterion for deciding whether operator
overloads would be useful, is that if a user-defined type (UDT) is
intended to extend the concept of one of the basic types, then it's
reasonable to implment operator overloads for the UDT that correspond to
the operators that can be used the corresponding basic type. Othewise,
the potential confusion causes by operator overloads is more trouble
than the convenience is worth.

For instance, UDTs that represent extensions of the concept of an
arithmetic type, like quaternions or matrices, should overload the
arithmetic operators. Smart Pointer classes that represent extensions of
the concept of a pointer, should overload the unary operator* and
operator->, (and possibly some of the arithmetic ones, if they're meant
to point into an array). Container classes that represent extensions of
the concept of an array should overload operator[]. Function object
classes that extend the concept of a function should overload
operator(). However, that pretty much exhausts the list of things for
which operator overloads are a good idea. The use of operator<< for the
standard I/O classes, and of operator+ for the standard string classes,
was arguably poor design, though it's too well entrenched by now to remove.