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.