Subject: Re: Object based/oriented IDL ? Ever likely ?
Posted by Ken Knighton on Fri, 22 Mar 1996 08:00:00 GMT
View Forum Message <> Reply to Message

Paul Schopf <schopf@gsfc.nasa.gov> wrote:
> Now if you want to discuss encapsulation and inheritance, I agree,
> IDL is not OOP, but these are data structures, and they are very
> easy to use, and you can write functions for them, etc. etc.

IDL is not an OOL, but it does have a number of features that make it possible to do things traditionally supported by OOLs although perhaps not with the same elegance:

1) Polymorphism

   a. Functions/procedures can be called with a variable number of formal parameters.

   b. Since identifiers are dynamically typed, a single func/pro can be devised that performs an operation on a variety of input argument types.

2) Inheritance

   a. Keyword inheritance (_EXTRA) can be used to allow a wrapper routine to "inherit" the capability of another routine and add functionality to it. For example, one can write wrapper routines to the PLOT procedure that do some other things but allow the calling program to utilize all of the present and future PLOT command keywords without having to keep track of all of them.

   b. Virtual funcs/pros: Use of the EXECUTE, CALL_PROCEDURE, and CALL_FUNCTION routines allows a programmer to set up "classes" that have user replaceable member routines by allowing the calling program to pass the names of the new member routines into the "class" via arguments or to place them into a common block for the "class".

   c. Various keywords to widget routines give the programmer the ability (with the above techniques) to do almost everything that OOLs do as far as widgets are concerned. This includes the ability to have constructors/destructors (notify_realize and kill_notify), different instantiations of a class (this is what an object is) by keeping a widget's information in a state variable rather than a common block, encapsulation of data in a widget's state variable, and etc.

3) Encapsulation (data hiding)

IDL doesn't have global variables (except for system variables) and therefore automatically has data hiding unless one uses common blocks. Even so, common blocks are only "public" to routines that reference the common block. A judicious use of modular programming techniques, naming conventions, the @ command to include "header" files, and other techniques give the programmer the ability to do a pretty good job of encapsulating data in the module that needs it. If there are going to be various instantiations of a module (objects), then it is possible to use widget state variables or handles to provide data encapsulation at the object level.

4) Operator overloading

IDL doesn't support this. Although it might be convenient to overload operators as far as command line users go, from the application developer standpoint (the guys that use C++, SmallTalk, and other OOLs), many feel that operator overloading is a bad practice and attempt to avoid it.

It is unfortunate that some of IDL's truly powerful features tend to be hidden or unknown to the majority of users. It is also unfortunate that RSI doesn't use them in most of the code they supply with IDL. It would be nice to have a bunch of tools supplied with IDL that were written in IDL using excellent software engineering practices and the powerful techniques that are already available in the language. That way, users would have examples to go by when creating their own cool software.

Finally, IDL is used in two ways: interactively and for application development. I feel that many improvements can be made for each mode of operation, but an improvement for one mode is not necessarily a desirable improvement for the other. I look at IDL from an application developer standpoint.

>
>
> BTW, a 3000 line IDL app is EXTREMELY long.
>

Sounds rather small to me, but that's the way it goes with perspectives. :-)

Ken Knighton   knighton@gav.gat.com   knighton@cts.com
General Atomics
San Diego, CA