Subject: Why IDL Is Not My Favorite Platform (was Re: IDL alternatives?)
Posted by zowie on Thu, 01 Jun 1995 07:00:00 GMT
View Forum Message <> Reply to Message

Charles Cavanaugh (cavanaug@uars1.acd.ucar.edu) wrote:
: I read in various places (Mr. Deforest's above posting being one) about
: how IDL's API is so-o-o-o horrible. I do not understand this. To me,
: IDL seems like a Fortran 90 - Pascal morph, with dynamic typing,
: automatic variables, automatic garbage collection and a useful event-
: driven paradigm all thrown in the mix.

Yep. You about summed it up [except that the gc, at least in 3.5, is broken]. Throw in a few references to APL, shake well, and you've got it.

My gripe isn't with the concept of IDL -- IMHO, this type of tool (with APL-like vector processing, simple imperative structure, the things Mr. Cavanaugh revers to, etc.) is exactly what is needed for scientific applications.

When I complain about the API, it's because the whole structure resembles a large collection of hacks, each with a unique user interface, and with little or no forethought about generalization. One gets the feeling that each tool or feature is crufted onto a mass of features that just accreted over the kernel interpreter, rather than carefully executed in an obvious Right Way according to some master plan. Here are a few representative warts on the programmer interface, at the syntax, data structure, and standard library levels. The troubles with support at the compilation and debugging level have been explored in another parallel thread.

For an example in the language syntax itself [one of many], the vector processing is very nice -- but (at least, in 3.5 -- don't know if this has changed in 4.0) the n:m and * syntaxes don't generalize to higher dimensions. You're left having to program the innermost dimension of a large array operation as a vector, and then write a for loop to do the operation across the other dimensions: two completely different strategies for different aspects of the same large parallel operation. While the system works, it makes code less readable and makes the programmer's environment model that much more complicated. Given that one is implementing vectorized operations, one ought to generalize them better.

An example of poor data structure design: Matrices act like their transposes. Enough said. Another: that cruft about a double quote denoting a string constant OR an octal number. The command
print,"05 hello"
is a syntax error. Sure, you can work around it, but it's ugly

as sin, and it's just one more useless detail to have to remember.

An example of a poorly designed collection of utilities: the whole friggin' plotting package. Now, the package itself is pretty nice: there're semi-device-independent plotting commands and so forth that work on a wide variety of devices, with a large number of knobs to frob. The problem is that it isn't assembled or organized in a coherent manner, and so is very hard to use without resorting to trial and error.

To compare with a "good package" to do the same thing, get a copy of Symantec C++ 7.0 for Macintosh, or CodeWarrior 6 (which is, I believe, the latest version) from Metrowerks, and read the manuals on their plotting and window objects. The problems to be solved are comparable: both IDL and the C++ packages need to present a large number of tools to handle a complicated conceptual array of abstract objects (multiple coordinate systems and planes that are abstracted out of the screen, page, or what-have-you to be drawn on). All three packages are aimed at the academic and professional programming markets, so the users (ie programmers) can in each case be expected to have a similar range of background knowledge and experience. The two C compilers, each of which costs an order of magnitude less than IDL (and comes, of course, with a smaller standard library), are professionally conceived, designed, and integrated, in contrast to IDL's sprawling amorphous mass of disconnected parts.

Compare the IDL User's Manual and/or Reference Guide to the comparable portions of the Metrowerks and/or Symantec documentation on handling graphical objects. Compare the organization of the graphical tools in each system. Both Metrowerks and Symantec produce well-organized programmer interfaces, coherent suites of prefab routines, and documentation: the abstract concepts are presented clearly in the documentation, and adhered to in calling conventions throughout the libraries. This clear and well-organized quality is not just a product of the structure of the language, it is the result of careful thought by the development team.

In contrast, while the IDL User's Guide contains a very brief introduction to the 'screen coordinate system', it is not always clear which calls use what coordinate system by default. Each graphics built- in routine can take a subset of the graphics keywords that are supported by the current device, but not all relevant keywords are always available. One has a choice of specifying distances in the units of several coordinate systems ("device units","screen units", etc.), or having them automatically translated to Imperial units (inches). However, PostScript pixels come by default in SI sizes. Etc. There are enough differences in the behavior of each device that, in practice, it is difficult to write code that work on all devices

without explicitly writing subsections to handle each type of graphics output -- this need begs the question of why the complex device-independent package is needed in the first place.

Finally, the tools that are presented to the programmer are (as other posters have mentioned) somewhat inferior when compared to, for example, the THINK (Symantec) debugger for Macintosh or the GNU debugger for everything else. While some effort has been made at trap handling and re-entry, the traceing, stack manipulation, and breakpoint facilities (particularly in the widget libraries) leave much to be desired.

None of these problems individually would cause me to dislike the language as much as I do. In fact, despite all the hassles and heartache, I like the basic interpreter enough to use it (or something like it) for most of my scientific programming. The world needs something like this -- a bit of APL mixed in with a bit of FORTRAN with structure, and I'm willing to tolerate all the poor planning and disorganization to use what I see as fundamentally a good tool.

The deciding point for me has been what I perceive as RSI's money-grubbing tactics. IDL is extremely highly priced, especially when compared to other programming packages with a similar audience and a much better, slicker final product. Maintenance costs are high, too. Furthermore, once one purchases IDL and writes some code in it, one is obliged to continue to keep one's license up-to-date. One sales clerk smugly informed me that "If you allow your maintenance to expire, we normally require you to pay for maintenance for the entire elapsed period before you can re-instate it", a policy stricter than California's auto-registration laws! (She did magnanimously waive the re-instatement fee on our license, because RSI is dropping support for our platform (MIPS/Ultrix in the near future).

I don't mind paying top dollar for top notch products, but IMHO that is not what one gets when one buys IDL. If I, a professional astrophysicist and computer programmer, am spending lots of my time figuring out a particularly opaque set of library function calls, then (IMHO) the problem is with the library or its documentation. When I buy an expensive mechanic's set of Snap-On tools, I expect them to work better and more easily than the discount set at Pak-n-Save. Similarly, when I shell out $1500 for a single-user license to run an interpreter and development package, I expect that package to be cleanly designed and well documented, not crufted together from random bits and pieces on (apparently) an ad-hoc basis.

: OK, so maybe you have to specify a continuation character, but in C you
: have to suffix lines with a ';'. To be honest, I would rather put a '$'
: at the end of the few lines I continue than put a ';' at the end of nearly
: every line. But here I am digressing. (Let's not get started on RSI's
: business practices
: My point being : it aint LISP, it aint object-oriented, but it does well
: (mostly) what it was designed to do.

: But maybe I am in the dark about this whole 1970's interface (I have only
: been programming since 1989), and I am always open to change. So if you
: (or another API slammer) could show tangible evidence that IDL's programming
: interface is a lava lamp or mood ring compared to C's video-conferencing or
: big-house-with-no-backyard, I will take back all that I have said, and jump
: on [insert language X here]'s bandwagon.

: Charles
: --
: Charles Cavanaugh | "Words are very unnecessary, they can only do harm"
: cavanaug@ncar.ucar.edu | - Depeche Mode
: NCAR Boulder, CO, USA | "Facts all come with points of view"
: My opinions | - Talking Heads

--
Craig DeForest "My research group launched a rocket into space, and all I
got was this lousy T-shirt"
