Programmer's UPDATE

VOLUME 7
NUMBER 9
DECEMBER 1989
$2.95

20 Working with The Lotus Add-In Toolkit

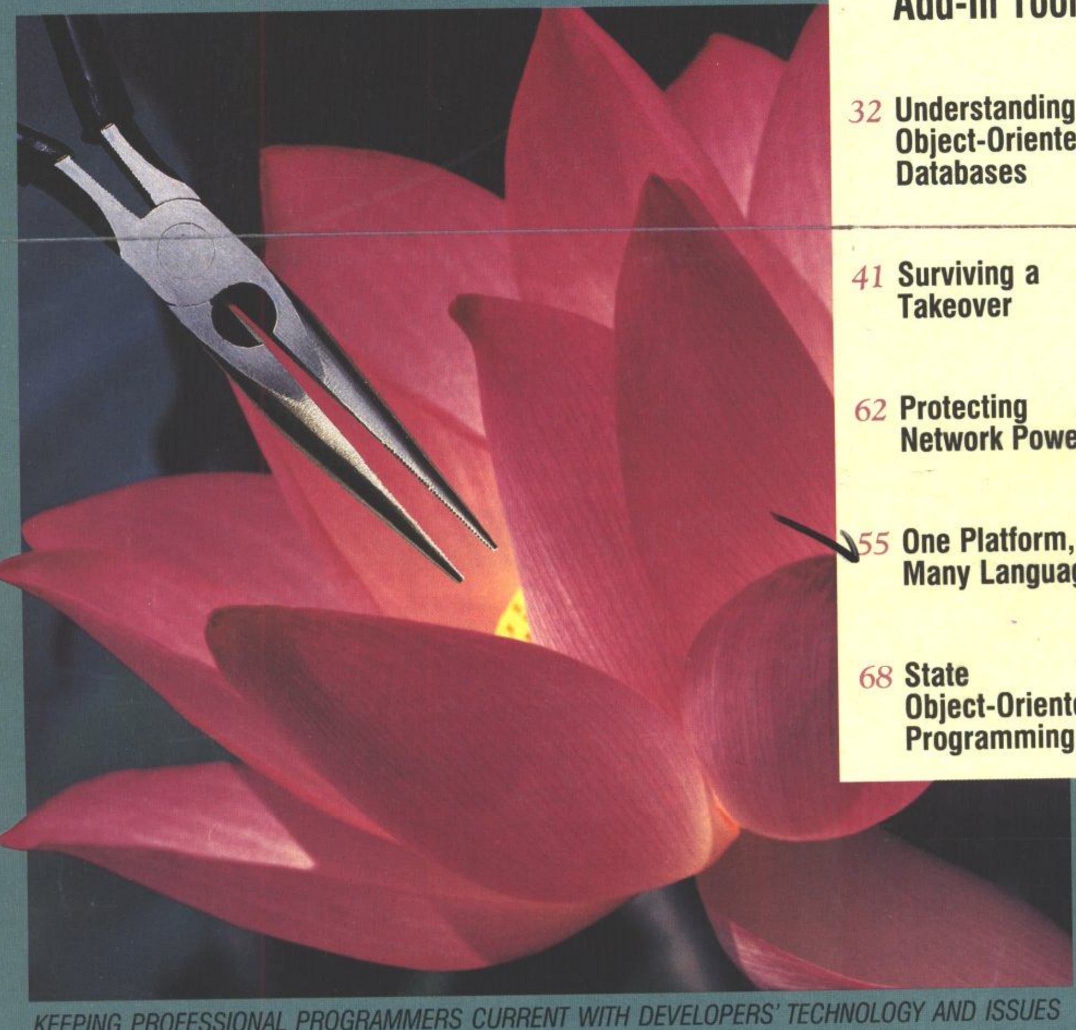
32 Understanding Object-Oriented Databases

41 Surviving a Takeover

62 Protecting Network Power

55 One Platform, Many Languages

68 State Object-Oriented Programming



KEEPING PROFESSIONAL PROGRAMMERS CURRENT WITH DEVELOPERS' TECHNOLOGY AND ISSUES

JENSEN & PARTNERS STORY
TopSpeed Compilers pg. 55

Niels Jensen, Jensen & Partners International

INTERVIEW

One Platform, Many Languages

How does the TopSpeed platform represent an important productivity concept for programmers?

We've noticed for some time now that a lot of people want to mix-and-match different programming languages. They might be doing a contract job where they have to provide some code in C and some in Modula-2 or Pascal, for instance. Or they may have code that they'd like to reuse somehow. Being able to link your tools in with a program you're writing in a different language could certainly increase your productivity. Also, newer languages such as Modula-2 can benefit quite a lot from this concept because they allow people to use the huge library of C code that already exists.

We've noticed for some time now that a lot of people want to mix-and-match different programming languages.

How is all of this related to the idea of dynamic linking?

OS/2 incorporates the concept of a dynamic link library (DLL), which is simply a collection of procedure functions. Normally, if you have twenty utilities that all use the same C library, you end up having twenty copies of that library floating around with your .EXE files, thus generating a lot of redundancy on your disk. With OS/2's dynamic link library, you save a lot of disk space because only one copy physically exists.

We developed our system under OS/2 first, by the way, and then ported it down to DOS. We found OS/2 to be a very nice development platform because if you do naughty things like write in the code segment, OS/2 will trap it for you immediately.

One Platform

It's a very safe environment to work in, and you trap bugs that would have wrought havoc under DOS or gone unnoticed until something really awful happened.

When we decided that we wanted to keep this design under DOS as well, we simply wrote a dynamic link library loader that works under DOS.

Do you pay in terms of execution speed by using DLLs?

There is a tiny overhead, but it's unnoticeable. You also pay a little bit in load time because the program needs to build up some tables as a gateway to the dynamic link library. But all those things are very small compared to the benefits you receive.

More importantly, you can have a very complicated software system that uses dynamic linking. You would then be able to replace one dynamic link library with a newer, better version, so to speak, without having to rethink your entire program or change the whole thing in any way. So you'd get this big system where you could plug in a new editor or compiler just by adding another DLL.

And, as a matter of fact, that's how our multi-language system works: You have to tell the system when you install it what compilers are actually part of it. You also tell it what file extensions belong to each compiler so that, for instance, all files called .C or .H will be C files, and files called .MOD, .PAS, .ADA, or .ASM will belong to other compilers. The automatic make system is also aware of this information. So if you have written a Modula-2 program that uses some C libraries, when you say "make" to the system, the environment will automatically pick the C compiler for the C files, the Modula compiler for the Modula files, and the assembler compiler for the assembler files.

Being able to link your tools in with a program you're writing in a different language could certainly increase your productivity.

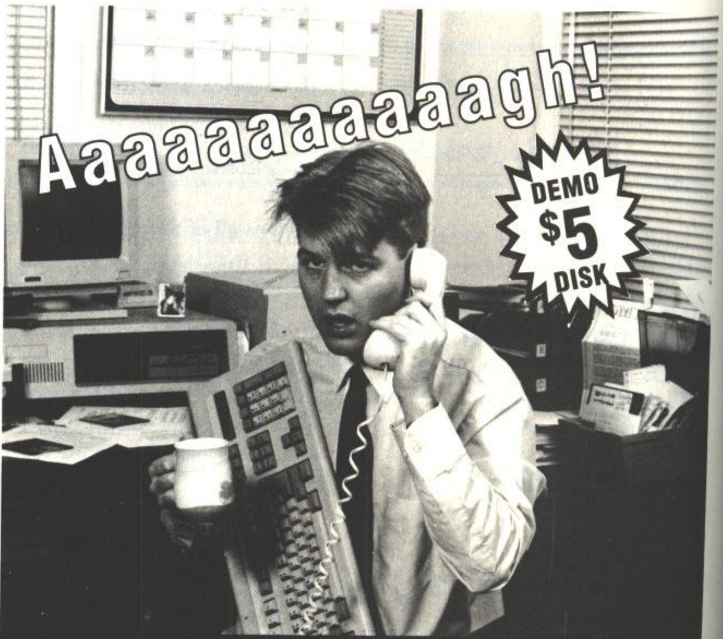
If you want to compile a program written in a single language, you simply point at the window that contains the source file and then press a "hot key" to compile it. If the extension is .ASM,

What do you mean 3 months to prepare the user's manual??

There is a better way... a much better way.... With DocuFlex, the documentation can be ready just a few days after the application's development is completed, not a few days before Rev .2 is ready for release.

DocuFlex, the automated documentation synthesizer, composes comprehensive instruction manuals for DataFlex, or any language based applications. Now available in version 4.0.

Also Available: HelpFlex...puts your entire instruction manual where it belongs... on-line with your application.



ROBITRON SOFTWARE RESEARCH

P.O. Box A, Rockmart, GA 30153, USA.
Phone: (404) 684-5855
FAX (404) 684-0468

Circle 63 on Reader Service Card.

it will select the assembler to compile it. Had the extension been C, it would have selected the C compiler automatically. It makes working with C and assembler quite pleasant because you just press a key whenever you want it compiled, assembled, or whatever.

The basic idea being to free you from mechanics?

Yes. Another important aspect of our system is that it tries to present the best of both worlds: You could compile your program fast for prototyping, then later use an optimizer switch, which would make the compilation take a bit longer, but would optimize the code to a very, very high standard.

You can think of our system as being just like any other big software system. You run one .EXE file, called "TopSpeed" when you start, and then you take advantage of the different dynamic link libraries. One is the environment with all the tools—editor, etc.—another is the front-end for the C compiler, and another is the common back-end—the code generator which actually generates machine code for different languages. You can have a C front-end and a Modula-2 front-end, and both will be DLLs; both will use the common code generator.

So if we release a new compiler and you already have the system installed with the C or the Modula compiler, C++ would only take up another 120–140K on your disk, plus, of course, whatever memory any extra C++ libraries would require. All you would need is a new dynamic link library containing the front-end for C++.

That's an interesting selling point for multi-language developers.

It's also something that the people who develop programs like accounting packages can make use of. If someone who has a DLL which takes care of printing makes the printing utility better, all he'll need to do is ship customers a new file containing that dynamic link library. The customer can then just copy it into his system. The programmer only needs to worry about algorithms and programming, not how everything is tied together, what files depend on what, and so on. Whatever can be automated should be. That's what we've achieved in our system. For instance, generating these dynamic link libraries is difficult using Microsoft tools, but with our tools you just select from a menu where you want a DLL generated, and the system will do it for you automatically.

There may also be many people working on a project which contains 500 files of source code that depend on each other in a variety of different ways. The way TopSpeed works, the system will read a source file from the network or from a disk, automatically look at all the dependencies, build new libraries for you if necessary, and so on.

There are a lot of other features that would be nice, such as an editor that could collapse procedures to be viewed as an outline. That's something I think you'll see in future programming environments. But the most important thing is a clean integration. If this environment can do a lot of the work behind your back, and do it right every time, it makes room for creativity.

Are you interested in porting to Unix?

Yes, we will probably be moving into Unix as soon as we can free the necessary resources.

Is this multi-language system a benefit for people who want to develop tools to support programmers as well as a benefit for developers?

Absolutely, because those people can choose whatever language they want to develop their tool in. And once they have developed it for something like our C compiler, our Modula-2

The programmer only needs to worry about algorithms and programming, not how everything is tied together, what files depend on what, and so on.

users will also be able to take advantage of it. And so would our future Ada and C++ users. We're working on a formal third-party support program right now. We're already in contact with all the third-party people out there, and they're all very keen to support our prototype.

What similarities does your approach to languages share with Microsoft's?

Well, Microsoft says that they have the ability to reuse code, but if you talk to people trying to utilize it, you'll see it's quite difficult to achieve. You really need to have your whole system integrated in a nice way, and you also need to have all of your runtime support general across all of your languages. It doesn't help you to try to allocate memory between Pascal and C if they use two completely different schemes, because it will result in a fight over which gets what part of RAM.

So you're implying that Microsoft doesn't transparently take care of that?

Exactly. At least that is the information I have from people I've talked to.

Is there some kind of trade-off in order to get this shared codability? For example, C and Pascal treat function argument definition differently. C is very relaxed, but in Pascal you have to define everything. Is there any sort of language rewriting or sacrifice of adherence to standards that you had to do?

No, absolutely not. You see, the system itself does not care what language you have written the object in. Whether you have written two C functions and compiled them separately or you have written one in Modula-2 and the other in C does not matter whatsoever. Of course, as you mentioned, C allows you a variable number of parameters, where some other languages do not. But that could be implemented as an extension to those other languages if you absolutely wanted to have this feature of a variable number of parameters preceding the function.

Wouldn't the Pascal or Modula programs either become confused or confuse the C portion of your program when it did something the other part wasn't expecting?

The way that we've implemented it avoids all this confusion completely. If you want to call a whole bunch of Modula-2 segments from your C program, you would write a header file in C and specify with a pragma that certain functions are to be called with Modula-2 calling conventions. That's all there is to it.

From then on you can act as if those functions had actually been written in C. When you want to give compiler directives, pragmas—or whatever they're called in Modula-2—you do it exactly the same way that you would in our C or our Ada. We have taken the syntax of Ada pragmas to be used across all our languages, because in Ada the syntax for pragmas is part of the definition of the language.

Would there be any compatibility problems if you wanted to use the code created in your environment somewhere else?

Not as far as the language itself is concerned. The pragmas, of course, would not be the same as the pragmas that we have chosen. But that's in the nature of the problem because there is no definition or formal official standard for pragmas. We've simply chosen the Ada method because sooner or later Ada is going to be incorporated into this multi-language system.

The drawback that I've heard about Ada is that it's an "everything including the kitchen sink" language. Do you run into the objection that there are hundreds of things to learn?

You hear that a lot, and in many instances it's probably true. The big advantage of Ada, though, is that it is very machine-independent. You will get a much higher level of portability than with any other language today. It's also a very safe language to program in: You

OPUS SOFTWARE

"Without a doubt, OPUS Make is the hottest make utility on the market"
Tom Swan, PCWorld, 7/89

Opus Make

For professional programmers, OPUS Make is the superior choice in a make utility. It is the fastest most full-featured make utility there is. Features include:

- **DOS version uses only 3K memory!**
- **Multiple directory support.**
- **Supports: Polytron PVCS™**
  Burton Systems TLIB™
  Microsoft LIB™
  SLR Systems OPTLIB™
- **Generates automatic response files of unlimited legnth for LINK and LIB.**
- **OPUS MKMF included: An automatic dependency generator that fully understands C preprocessor directives.**

DOS and OS/2 executables for only: $99

MC, Visa, COD, PO
415-664-7901

1032 Irving Street, Suite 439, San Francisco, CA 94122
Trademarks owned by their respective companies

Circle 7 on Reader Service Card.

Get the best of both worlds.

The Berkeley Utilities
UNIX COMMANDS FOR MS·DOS

The latest product from OPENetwork brings the MS-DOS world a set of UNIX tools with a twist.

- 32 UNIX commands with all System V.3 and BSD 4.3 options plus a number of DOS-specific enhancements.
- Includes **awk, grep, find, make, diff, ed, sed,** and **sort**.
- Each utility is standalone, uses video attributes and has a built-in Help screen.
- Comprehensive manuals, including an introduction to UNIX for DOS users.

For IBM compatibles (hard disk recommended), 256K RAM, DOS 3.0 or higher.

Order The Berkeley Utilities now for $125.

Available from Programmer's Shop or OPENetwork.
Call our BBS for complete product information and demos.

OPENetwork
POWER TOOLS FOR POWER USERS

215 Berkeley Pl. (P2)
Brooklyn, NY 11217
Voice: 718·638·2240
BBS: 718·638·2239

Circle 31 on Reader Service Card.

can't make so many silly mistakes because the compiler will catch them.

People in the U.S. are probably curious about what you see as the real and potential market size for Modula-2, either here or abroad. How many copies are you selling and how many people do you expect to adopt Modula-2?

Well, of course I'd like to see most people programming in Modula-2 instead of C, though since we're releasing the C compiler, it isn't as important.

It's quite strange, but the U.S. seems to be more conservative than Europe when choosing programming languages. We have found that the market outside of the U.S. is very much more alive in Modula-2. In fact, the European space agency intends to write all the software for their space program in Modula-2.

Actually, over the past couple of months we have shipped thousands of systems to universities stocking up for next semester, so I think that's probably a good indicator of what will happen in the future with Modula-2.

You mean that if people use it in an academic environment, then they will use it in the real world because it's what they are used to?

Exactly. That was certainly one of the reasons that Turbo took off. People got acquainted with it at school.

At this point, are there any significant tool or library opportunities for developers writing to support the TopSpeed platform?

Well, we've had to cover a lot of the Modula-2 market ourselves because it is not as mature as the C market, but there will be the same general opportunities for developers as with any other environment. Because we are highly compatible with Microsoft C and Turbo C, before the product hits the market we will have the support of many third-party tool makers whose tools we have already verified will work with our compiler.

Does it seem with Modula-2 and Ada that you are fighting an uphill battle against the universe of C programmers?

Well, there are two kinds of C programmers. First there are those who know about more formal languages, like Pascal and Modula-2, but haven't used them because, before our Modula-2 came along, there weren't any good alternatives. The existing Modula-2 and Pascal compilers did not generate code that was fast enough, so they went with C. Many of these people are certainly ready to move into Modula-2, and it's beginning to happen.

The other kind of of C programmer does not even know what an enumerated type is, can't understand using local procedures, and so on. You'll probably never talk them into using a more modern language, and they'll be the ones who move to C++ instead of Modula-2.

Do you plan to add object-oriented extensions to any of your language products besides C++?

One thing that is always overlooked in discussions of object-oriented programming is that you really don't need a language like C++ or Turbo Pascal 5.5. Of course, doing object-oriented programming in Modula-2 would not be as easy now as it is in Turbo 5.5, but by adding the same kind of syntactical sugar, it would be.

I would not be surprised if you saw object-oriented extensions to our languages in the future. I think it's a good step forward, though it has always puzzled me that C programmers seem willing to accept such a level of abstraction when they're not willing to accept a switch to Pascal or Modula-2.

—*MKS*

Niels Jensen, a co-founder of Borland International, is the president of Jensen & Partners International. He has managed the acquisition and development of a number of products, including Turbo Pascal, SideKick, and SuperKey. □

"According to this, the next big trend will be Z80 multitasking GUI development in BASIC."