GEORGE SCHUSSEL

WHICH LANGUAGE SHOULD YOU SPEAK?

This is the first of George Schussel's two-part discussion on choosing tools and languages for the client/server architecture.

Now that you're convinced that client/server is the architecture for the future, the next question is which application development tools should your organization be using to implement that architecture. Sticking to accepted market standards is always a safe approach, as you are ensured of greater availability of add-on utilities, training materials, and finished applications—with less risk, too.

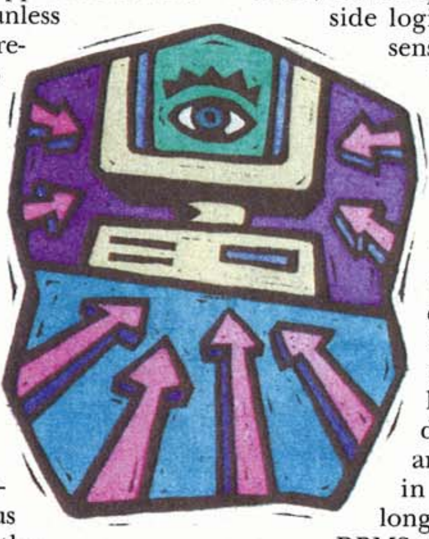
Choosing standard tools and languages in the 1970s was easy. Most custom applications ran on mainframes. These applications were typically built in COBOL, unless you were among the adventuresome 2 percent or so who tried something unusual like IPL or PL/1.

In the 1980s, COBOL was still the predominant business application language, but proprietary 4GLs like Natural, ADF, and FOCUS, offering higher productivity, emerged as COBOL competitors. Indeed, the major proprietary 4GLs built the user bases of thousands of companies.

If you were really avant-garde and used PCs for serious application building in the 1980s, then one of the flavors of the dBASE language was probably your language choice.

As an architecture, client/server has been around since Sybase started selling SQL Server in the late 1980s. When Gupta, Oracle, and others started selling SQL-based DBMS server engines, the idea of a relational DBMS for the server side of client/server computing became an accepted norm. While some people now look at object DBMS engines for handling complex data or difficult performance problems, the SQL engine approach, for typical business applications, has 90 percent of the market.

Meanwhile, client/server developers must now ask what language or tool facility will emerge as the standard for writing the client side of client/server applications.



The difference between regular (character) style programming and GUI or Windows programming is the difference between straight-line and event-driven programming. In a traditional COBOL-style world, the application program is in control. The program drives the user. In a GUI environment like Windows, it's the other way around. Because events can be invoked from all parts of the screen, the developer of a GUI application has to anticipate all potential actions of a user and in all potential orders. And other programs may also be running and interacting with this program.

Developing for such a situation is complex; most people should have an advanced facility to develop such applications, something easier than the C language. Surprisingly, until recently the vast majority of deployed or fielded client/server applications had their client-side logic written in C. That made sense, as native controls for popular GUIs, such as Motif and Windows, were supported in C and C++ environments.

With the advent of new Visual 4GLs like SQLWindows, Omnis, ObjectView, VisualWorks, and PowerBuilder, access to these GUI controls was made more easily available. You didn't need to know C. These products have features like Data Windows, which provide point-and-click access to data stored in SQL databases. You no longer need to program in the DBMS-specific API or SQL to access that relational data. In general, this new type of tool provides three principal types of services for building client/server applications:

··Interface Building: Creating what appears to the end user on a PC or workstation. This facility usually supports the most popular GUIs, such as Windows, Macintosh, and Motif. · Create Application Logic: This facility allows scripts to be attached to events. These scripts are normally written in the vendor's proprietary language. It's common to find scripting languages that look like BASIC, Smalltalk, C++, or Xbase. · Connect to Database or Source: Provides for communication with server DBMS where the application data resides. These connections are automatically generated and are in addition to such connectivity options as dynamic and static SQL, ODBC, or other types of remote procedure calls (RPCs). Based onthe findings from informal surveys, it appears to me that, until about a year ago, the majority of client/server applications were written in C or C++.

The surge in Visual 4GLs

Beginning in 1994, however, a tremendous surge in sales of Visual 4GLs occurred. I guess that a current survey of production client/server applications would yield the finding that Visual 4GL and C/C++ categories each have about 40 percent of the fielded client/server applications. Another popular technology base for Visual 4GLs is the Smalltalk language. Typified by tools such as Visual Works (ParcPlace Systems), Smalltalk-based technologies seem to account for about 15 percent of existing applications. There are significant differences in capability, ease of learning, and deployability of the various proprietary Visual 4GLs.

What they all do share in common is that each is based on a combination of proprietary languages and screen-painting approaches, with the goal of simplifying the development of client- based applications. This category is a descendant of the 4GLs of the 1980s.

With PowerBuilder and Visual Basic leading the way, these products have seen a huge surge in popularity over the last year. Where performance is paramount, however, C-based technologies still hold an important advantage. When training and productivity are the issues, Visual 4GLs will predominate.

Calculating the minuses in C++

Comparing C++ to COBOL is unfair to COBOL, which actually was a marvelous feat of engineering, given the technology of its day. The only marvelous thing about C++ is that anyone manages to get any work done in it at all. Fortunately, most good programmers know that they can avoid C++ by writing largely in C, steering clear of most of the ridiculous features that they'll probably never understand anyway. Usually, this means writing their own non-object-oriented tools to get just the features they need. Of course, this means their code will be idiosyncratic, incompatible, and impossible to understand or reuse. But, a thin veneer of C++ here and there is just enough to fool managers into approving their projects.

—Excerpted from *The UNIX-Haters Handbook*, edited by Simson Garfinkel, Daniel Weise, and Steven Strassmann, IDG Books, 1994.

Guidance for C and C++

Since COBOL is an industry standard that many vendors provide, and the same applies to C, it's tempting to think of C and C++ as the logical follow-on to the COBOL of the 1970s and 1980s. However, C technology is a lot tougher to master and at a lower level than COBOL.

The best analogy is to compare C and C++ application development to building applications in 370 assembler code. If you are thinking of using C++ for application development, it might be wise to read The UNIX-Haters Handbook first (see box).

The outlook for Smalltalk

Smalltalk, which is an object-oriented language, has the advantage of being a higher level in syntax than C++. Since the best current technology for building client-side tool sets is object-oriented, Smalltalk is a logical candidate for the underlying technology.

ParcPlace Systems with its Visual Works, IBM with its VisualAge, and other companies like Digitalk agree and are leading the development of this market.

Smalltalk, as the scripting language for this category of tool, is similar to a 3GL like COBOL in syntactic difficulty. Ed Yourdon, Chairman of DCI's Software World Conference, has analogized Smalltalk to the Apple Macintosh, a superior technology destined to play an important but minority role.

A newer COBOL thinks Visual

For a language that was originally developed back in 1960, COBOL has shown amazing longevity. Much of the source of this longevity, of course, is the investment in training and in knowledge of the estimated 2 million COBOL programmers in the United States alone.

Now, products that have been developed by Microfocus, Computer Associates, and IBM are providing visual and object-based enhancements for the COBOL community. Think of a visual programming environment like Visual Basic or PowerBuilder, but with COBOL for the scripting language and an ability to encapsulate COBOL code into objects that can be activated from a button.

An experienced COBOL programmer can pick up visual programming with one of these products in about a month's time.

That contrasts to the much more complete retraining and many months required to pick up a comparable skill level in C++, Smalltalk, or a tool like PowerBuilder.

George Schussel is the founder and chairman of Digital Consulting Inc. (DCI) in Andover, Mass. He can be reached at CompuServe 74407,2472.