AFTER IBM: THE TECHNICAL IMPERATIVE FOR CONSOLIDATION

Analysts foresee complexity-adverse buyers continuing to weed out secondary players, thereby encouraging a merger frenzy in a dash to the top.

By Clara Basile and Ellen Ullman

From July 1994 issue

Consolidation has come to the software industry. Novell buys WordPerfect. Aldus merges with adobe. Borland, slowly dwindling, sells its Quattro Pro Spreadsheet to Novell. Not content with gobbling up rival Norton Utilities, Symantec goes on to merge with Centran Point. These are but the most recent announcements in a steady process of acquisition and recombination.

After the early-1990s wave of business failures that saw PC hardware vendors reduced to a few key players, the consolidation movement has now come to the software side of the business.

Many analysts have discussed computer industry consolidation in economic terms. They have noted the effects of the recession, falling profit margins, downward price pressures. While economic factors certainly say a great deal about the difficulties of doing business in the late eighties and early nineties, any discussion of the computer industry must also take into account the most basic fact upon which the industry is built: technology.

This month, we look at the technical factors that are fueling the movement towards consolidation in the industry. We devote the entire piece to the technology scenario, providing a detailed look at the changes in enterprise computing that are affecting the climate for both buyers and sellers. Next month, we will discuss the effect of consolidation on technology investments. We will examine the way the market sees the various niches involved in enterprise systems.

After the Fall

The period 1987-90 saw not only a recession but also a fundamental shift in technology used by the enterprise. It marked the end of the era of the proprietary vendor and the movement into the mainstream of so-called "open systems." The fall from dominance of IBM is, of course, the hallmark of this technological shift. IBM's retrenchment affected the structure of the entire computer industry. And it profoundly changed the nature of business computing itself.

Before the fall of IBM, the problem of building enterprise systems was one of procurement: a company first had to select a vendor and, with that selection, came a complete set of tools. After IBM, the problem became one of integration: a business has to take pieces of the solution from multiple vendors and, out of that complexity, must somehow create a coherent, working set of applications.

The complexity of today's enterprise system is more than just a technical challenge; it also affects the way a business goes about buying the various pieces of the system. In a world with nearly endless choices, there are only so many variables anyone can consider. Buyers try to minimize technical difficulties by considering only the best-known vendors in each system niche; they want standard solutions; they eliminate choices.

There is an irony here. The promise of open systems was freedom from a single vendor, a wide range of competing solutions from which the buyer could choose the most effective product. As it turns out, however, there is only so much freedom that a working computer system can stand. In the long run, technical complexity is driving out the secondary set of vendors. The sheer difficulty of today's systems is creating an imperative towards consolidation.

The Great Age of Proprietary Systems

There are still a few developers around who will admit that, compared to today's mix-and-match environments, it was fairly easy to build a business application on a proprietary system like one from IBM. All the tools were in place; there was no question that they would work together. Compiler, debugger, dump analyzer. User interface, sort routines, standard libraries. All these elements of the development environment came from the system vendor or were 100% compatible with it. The same was true of the hardware. Developers did not have to wonder about the type of monitors, terminals, printers, and controllers that would be part of the application; the hardware requirements were part of the application design, and developers could simply write their code assuming that an IBM 3274 or AS/400 terminal would be on the other end.

In the all-IBM shop, the code could simply address the needs of the application without much explicit logic dealing with the environment itself. If there was an intractable system problem, IBM could send out a customer engineer-the fabled CE, an alert young man in white shirt and tie who would practically live on-site until the problem was solved.

The much disparaged "application backlog" came not from the proprietary nature of the system but from structural and managerial causes: the lack of consistent online access to the system for the programmer, lengthy turnaround times during development and debugging, and, above all, an historically adversarial relationship between systems people and their user departments. Once the developer was actually working on the application, however, the main problem was getting the code right for the user's needs.

In modern client/server systems, in contrast, an overwhelming portion of the code must handle interfaces among the disparate elements that make up the system. When an application designer sits down today to start designing a system, the first thing he or she must do is figure out a way to shield the programmer from the vagaries of multiple, incompatible technical environments. There may be several clients: DOS, Windows, OS/2, Mac, UNIX Motif. There can be multiple networking protocols: IPX, AppleTalk, Net BIOS, TCP/IP, SNA. There can be one or more networked file system: Novell, NFS. There may be several databases: SQL engines on UNIX or NT platforms, as well as older host-based data sources. Client platforms may be connected via some middleware or gateway product to various minicomputers and, finally, to a mainframe. Given all the variables, there is only one thing about the client/server environment that the designer knows for certain: if there is an intractable system problem, no nice CE from IBM in tie and white shirt will come to fix it.

The Complexity Problem

To see how complex the development environment has become, one only needs to look at the range of tools that today's programmer must master just to build a small, departmental application. Figure 1 compares a mix of tools used in the 1970s to those in standard use today. The Pick-based minicomputer of the seventies came fully equipped for application development: operating system, programming tools, database, and query language were all included. Right out of the box, the system was ready for the programmer to sit down and write code. The created application would only work on this vendor's hardware--the problem of proprietary systems--but the hardware was part of a package designed to get the user started without further attention to the platform itself.

Today's client/ server application, on the other hand, can run on hardware from any number of vendors. But to get that freedom from a particular hardware platform, the user has inherited a new set of problems. In place of the single vendor, there are at least five. Instead of a system-in-a-

box, there is a small roomful of heavy boxes filled with fat manuals.

The situation is even more complex when we move from an application contained within a department to one that reaches across the organization. Figure 2 is based on an application used in 1993 at the U.S. House of Representatives. It shows a query that originates on a Macintosh in a representative's office and accesses data about pending legislation that resides on a mainframe. From the user's viewpoint, the application seems simple, but the system manager knows better. This single query crosses three operating systems and three communications protocols. Even if we forget about the several hardware and networking products involved, the one SQL statement relies upon products from six vendors. When something goes wrong, as it inevitably will, it is the system manager who must sort out the cause.

If client/server systems are so difficult to build and maintain, why do organizations bother? Are these complicated systems worth all the trouble? A full accounting of the benefits of client/server enterprise computing is beyond the scope of this article, but it must be said that, no matter what the technical difficulties, there will be no return to the days of the proprietary vendor. Client/server greatly empower the end user. The systems offer a vastly improved human interface. They let the enterprise tailor hardware and software platforms to particular application needs. Above all, their modular nature allows theoretically unlimited access to data across a global organization. For these and many other reasons, client/server architecture represents a technological improvement that will continue to advance.

However, the companies building client/server systems must minimize complexity if their systems are to operate with any reliability. And the foremost method of limiting complexity is to eliminate variables. In corporate purchasing, this is done by eliminating from consideration products that are not commonly used. It is a modern variation on the "safe Blue" approach of the IBM era. In the days of Big Blue dominance, the saying was, "No one ever got fired for buying IBM." Now the saying might be, "No one ever got fired for buying what everyone else is buying."

The Virtual Proprietary System

In each niche of the client/server system, corporate buyers consider two or three products, usually the ones that are already the most successful. This behavior is not cowardice but prudence. When system problems arise, buyers know that there are other systems nearly like theirs; they have put together a system that is similar in configuration to other enterprise systems. In so doing, these corporate buyers are creating a multiple-vendor, open system that begins to resemble a proprietary system in that each element is an "expected" one. As much as possible, odd brands of hardware and software are left out of the puzzle. As in the good old days of proprietary systems, the programmer can assume that certain pieces will be there.

Complexity-adverse purchasing on the part of corporate buyers reinforces the market leadership of the top vendors, which in turn reinforces the tendency of buyers to select the leaders. Eventually, secondary vendors lose market share and are forced out of the market. (See Figure 3.) The result is technology-driven consolidation.

Vendors left in the market must begin to make their product offerings converge around a standard set of features and interfaces. Buyers demand that client/server products adhere to common interface standards, such as the ones emerging in classes of software ranging from SQL engine databases to network management, from messaging to object handling. These standards, in essence, recreate in software the seamless interfaces once offered by proprietary hardware.

The increasing standardization also makes it difficult for new vendors to enter the market by differentiating themselves from the leaders. Again, the result is a reinforced movement towards consolidation. Over time, under the pressure of standardization, vendors run the risk of creating a commodity product, which allows new vendors to enter the market through aggressive pricing.

Now that client/server is firmly in the mainstream, it's too late for vendors to gain much market share by offering "better" features. For example, database vendor Informix may offer a parallel query server not available from its competitors, but the imperative to keep down the number of system variables may be greater than the need for this feature. OS/2 provides a relatively compact, stable, multitasking desktop environment not available from any other source, but one more client platform may be one too many for the system manager. In the complex universe of client/server systems, a few good features here and there may not be worth the price of another vendor, another maintenance contract, another set of programming interfaces, another boxful of manuals, one more round of installation and training.

Next month, we will examine what this winnowing process means for investors. We'll look at the vendors that users and stock buyers are beginning to leave out of the picture. And we'll look at the fortunate vendors who have established themselves in the enterprise, the ones who have achieved the right balance of market share, adherence to standards, and technological edge.

Clara Basile co-founded Avalon Capital Management with partners Dave Rahn, Bruce Erickson and Bill Oberman. Avalon is a northern California investment firm that provides personalized investment portfolios for individuals.

Ellen Ullman is a software engineering design consultant and principal at NeoLogica. San Francisco-based NeoLogica specializes in new-product services for start-up and established technology companies.