

Paper #: 3045

The Importance of Middleware to Client-Server and the Internet

By C. Bradley Tashenberg

4265 San Felipe, Suite 800, Houston, TX 77027
(713) 621-2808

You can, of course, go client-server with your HP3000. However, everything you read and hear these days suggests that to go true client-server you have to abandon your HP 3000, move to UNIX or NT, and get a copy of Oracle, Sybase, Informix, SQL Server, or, for IBM users, DB2. Note that you're not allowed to pick just any relational database. It has to be one of the above, because these are the only ones being supported by the bigger client-server developers--SAP, Baan, PeopleSoft, Oracle Financials, etc.

To top it off, they've coined a new term for those of us who have elected to stay with their current systems rather than going to UNIX--we're called legacy systems users. The reason, I guess, is that we have older systems that operate in mission-critical environments. Well, doesn't that just make you feel great!

A year ago I sat in a Gartner session on legacy systems. It happened to be the best-attended session at the show, having 2,500 people. To begin the session the speaker asked three questions. First, How many people have a legacy system? All raised their hands. Second, How many are happy with your legacy systems? Almost all hands went up. And the third, How many of you are being forced to go client-server? The entire audience responded with raised hands.

This was not an HP 3000 show. Since Gartner tends to appeal strongly to the Fortune 500 companies, you could call it an industry show. And, most of those in attendance were CIOs, CFOs, technology managers, corporate MIS directors, and corporate and divisional managers--an excellent representation of corporate America today. Many market segments were represented. Attendees came from companies with IBM equipment, HP equipment, DEC equipment, SUN equipment--you name it and they had it. They shared one thing in common, however. They all had legacy systems and they were all being persuaded to go to client-server.

From this session I became aware that despite strong movements out there to move to UNIX, client-server, and relational database technologies, many are very happy with their current legacy environments. After watching the scenario over the past year, even Gartner agreed that the satisfaction with legacy systems is not weakening but is getting stronger. The systems are working, the users are happy, and business is getting done.

Revolutionary Technologies

Older technology is generally abandoned when newer technology is so wonderful that it makes the old completely obsolete. For example, when vacuum tube computers first came out, offering faster readers and printers and 16K of programmable memory, their predecessors, the unit record devices, became obsolete. Unit record devices had mechanical sorters, slow readers, slow printers, and drawers of replaceable metal wire boards that programmers had to wire in order to perform the program logic. Nobody questioned a move from the older to the newer technology--the newer was obviously superior. The same thing happened with the introduction of mainframe computers having condensed integrated circuitry, which replaced the vacuum tubes of the previous generation.

In the area of hardware technology, whenever people go back to an older technology, it's generally for economic reasons. They would unquestionably go to the new, if they could afford it. We see this all too frequently in the PC industry, where Moore's Law of 18-month obsolescence prevails. I like to refer to this as revolutionary change because the end result is to throw away the old and bring in the new, a syndrome promoted by ad agencies as "bigger, faster, better."

Evolutionary Technologies

The software industry, surprisingly enough, is different. Batch processing has been around since the days of unit record. Even though we've gone through several generations of hardware, people haven't thrown out batch processing. Even the advent of online processing didn't eliminate the use of batch

processing, especially in those areas in which batch processing was the most practical. I don't know of any payroll systems that process the payroll one employee at a time. Normally, it's processed in batch.

Similarly, online systems aren't going to be thrown out just because client-server applications and the Internet are in vogue. And nobody is going to throw out a client-server system to implement an Internet application. Client-server and Internet/ intranet applications, from a software perspective, are just another variation of online processing. Who knows...with the current movement to thin clients, intelligent terminals may be back in style. On the other hand, you might view database systems as a refinement or add-on (significant as it may be) to conventional methods of file management.

Sure, some software technologies have revolutionized the market, but their development has been evolutionary. Assembler languages evolved into third-generation languages (3GLs) like COBOL and FORTRAN, which further evolved into 4GLs like PowerHouse, Speedware, PowerBuilder, and Visual Basic. File systems evolved from simple tape and disk file management systems to network database systems like IMAGE and relational database systems like ALLBASE, Oracle, Sybase, etc.

Isn't software development supposed to promote evolutionary growth, so we can take advantage of our previous investments? New technologies are generally designed to help us service the programming needs of our future projects, but we don't have to throw away systems that are working successfully for us today.

For too many years in data processing, we were in a mode of reinventing the wheel. That's what made the HP 3000 such a superlative system. From its inception in 1972, its success can be attributed to compatibility. Yes, it is one of the most reliable systems on the market. Yes, it has the widest range of processing power of any system out there. And, yes, it is the best-priced system per performance in its field. But what makes it great is the 20-plus years of upgrades and hardware improvements, with a consistent backward-compatible upgrade path that allowed us to move forward gracefully and painlessly. So, why do we have to change now? Maybe we don't!

A Client-Server Definition

One perspective of client-server is that in its most basic form, it is the integration of PCs (or more accurately the Microsoft Windows environment) into the distributed processing model of the professional data processing environment. For the past 20 years, these elements have coexisted as mutually nonexclusive data processing solutions. Generally, we view the computer industry as one entity, but there are two distinct industries--the PC market and the professional DP market. Think of the way you order computers. If you want a mainframe or midrange computer, you don't go over to your local MicroWarehouse, Computer City, or CompUSA; you go directly to the hardware vendor. If you want a PC, you go to a local retail store, which usually offers better deals.

The reason is simple. Mainframes and midrange computers require a professional staff to operate, whereas PCs are end-user machines. Now, however, PCs are found in almost every area of the company. The professional data processor can no longer ignore them, and must figure out a way to integrate them into the overall system model--voilà, client-server. (See Figure 1.)

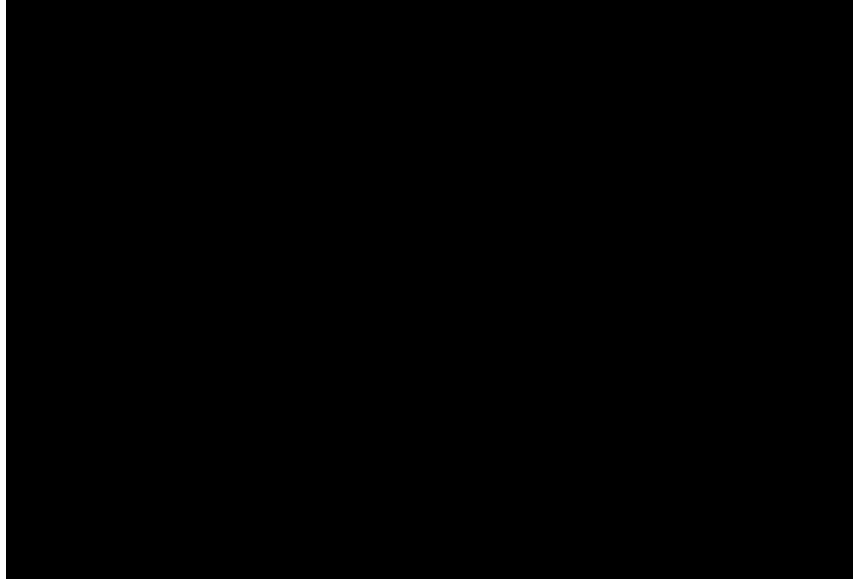


Figure 1: Client-Server

Attempts to Develop Universal Standards

In the mid 1980s and early 1990s, terminal emulators were developed as a feeble attempt to link PCs to the HP 3000 and other central computers. Emulators were originally designed to serve a short-term purpose, enabling the PC user to sign onto the HP 3000, using the PC as a terminal to a larger machine or using it to pass files back and forth between the PC and the HP 3000. But the linkage acted simply as a bridge between the two technologies and the options were limited: terminal emulation or RJE (remote job entry). Terminal emulators provided little integration. The PC couldn't operate at its full potential. A PC program could not talk directly to an application program on the HP 3000.

So what's the answer? The answer certainly isn't to throw away your HP 3000 and convert to UNIX, because UNIX doesn't have a direct solution either. One solution might be to get a copy of Oracle or Sybase, because these products do provide the linkage required to do client-server processing. Unfortunately, this is a proprietary solution. Another solution might be to use ODBC (Open Database Connectivity), but ODBC allows you to communicate only with the database, not with any of your HP 3000 programs. And, of course, since ODBC uses SQL, it communicates only through ALLBASE or the relational side of IMAGE/SQL.

So what makes UNIX, relational database technologies (such as Oracle, Sybase, and Informix), and ODBC so attractive? Is there a magic ingredient common to all of them? At first glance, it doesn't seem so. But their successes reveal a common theme. They all attempt to build a homogeneous enterprise-wide environment.

The UNIX Solution

First came the Open Systems Foundation and UNIX, requiring everyone to abide by a common operating system standard. Over time, older systems would be replaced by new UNIX-based ones, forming operating environment uniformity similar to what has taken place in the PC market through Microsoft DOS and Windows. Unfortunately, this just won't happen. The mainframers and midrange systems users see very little reason to move to UNIX. It's slow, it's different, it's obtuse, and it lacks the necessary support tool familiar to us HP 3000 users and professional DP users.

Gartner projects that mainframes and midrange systems will be around for the next 20 years. And, by the way, last year was the best of the past five years in IBM sales. Furthermore, Bill Gates doesn't plan on being eliminated from the offerings. He would like NT to be the winner, if there is a winner in the operating system ring.

It is becoming quite clear that no one operating system is going to prevail by the end of the millennium. Therefore, it is safe to say that UNIX will not become the de facto standard for the industry and that we will be living in a world of heterogeneous systems for years to come. So we better accept them as a fact of life, and deal with them. (See Figure 2.)

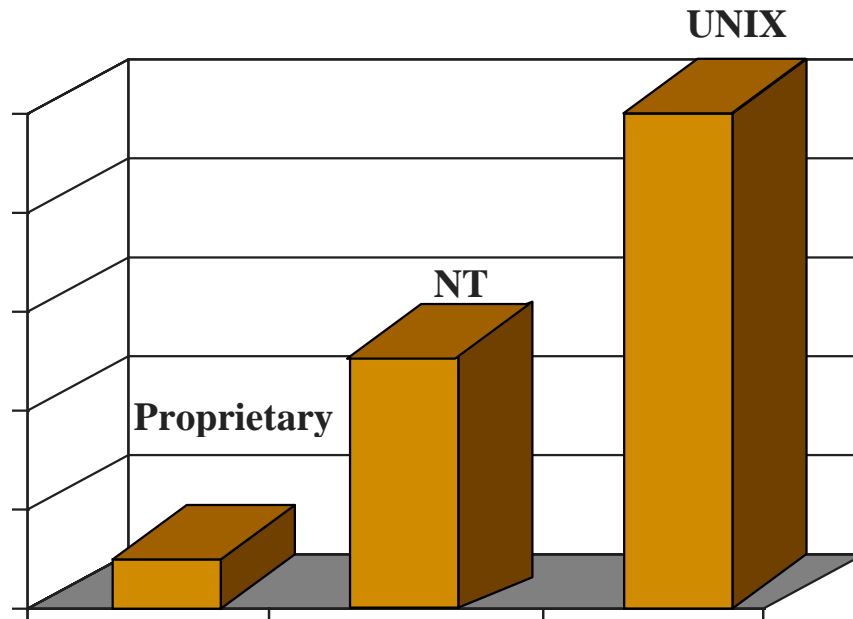


Figure 2: Next Generation of Open Systems

The Relational Database Solution

After the UNIX solution came the bandwagon moving to relational database technology. Since DB2 and Ingres were the early leaders in relational database technology, why has there been so much interest in Oracle and Sybase? Today, Oracle has a substantial lead in the RDBMS market, with the second-place vendor quite far behind.

When I first looked into this, I thought that it had something to do with Oracle's superior relational database technology. Surprise, surprise! It has nothing to do with the database technology. Rather, Oracle sits on 172 platforms and can make them behave as though they were one homogeneous environment.

Oracle offers the best middleware solution of any RDBMS vendor in the market, and supports more platforms than any other relational vendor. IBM has a great RDBMS, but it works only on IBM mainframes. Ingres works only under UNIX and doesn't offer non-UNIX ports. Sybase, like Oracle, understands the value of middleware and ports to as many platforms as practical, offering even greater flexibility across platforms for database operations.

Oracle and Sybase offer excellent ways to tie the enterprise together, but they are proprietary, not open system, solutions. And, what if you don't want to use a relational database? Maybe you want to use only IMAGE. Oracle won't allow that. To acquire the Oracle middleware (called SQL *Net), you have to buy all of Oracle or nothing. Sybase is no different.

The ODBC Solution

Last came ODBC, and from a PC developer's perspective, ODBC looked like a pretty good solution. Remember, to Bill Gates the only real computer market is the PC market. All else is arcane and unacceptable. Therefore, in his quest to develop universal standards, Gates assumed that all processing took place on the PC and that any external system would be nothing other than a file server.

According to this definition, ODBC appears to be a sound universal way to access information housed in a repository of different relational database technologies. Again, it is a middleware solution, but one specifically restricted to the database area. ODBC's PC-centric perspective, however, leads to some server limitations. First, ODBC will support only a two-tier architecture. This is an obtuse way of saying it can communicate only between a PC and one other computer at a time, because the database drivers that service the request are PC-resident and not server-resident. (I guess that leaves the Internet out, since most Internet users involve configuring a Web server between the external user and the company's mission-critical database.)

This brings us to the second and more fundamental ODBC limitation. All the data must be retrieved and brought across the LAN to the PC before the actual information requested can be filtered out. In heavy usage environments this puts a tremendous burden on the main system and the LAN, and could result in unbearable response times.

The Middleware Solution

In examining all the possible solutions, one thing becomes very clear. Among UNIX, the new leading relational database technologies, and ODBC, there is no one solution that allows a heterogeneous environment of different operating systems to behave as though it were homogeneous, without imposing significant outside influences. So, the only realistic solution is through the connectivity component. Not the operating system, not the database, not the language--but the connectivity component. (See Figure 3.)

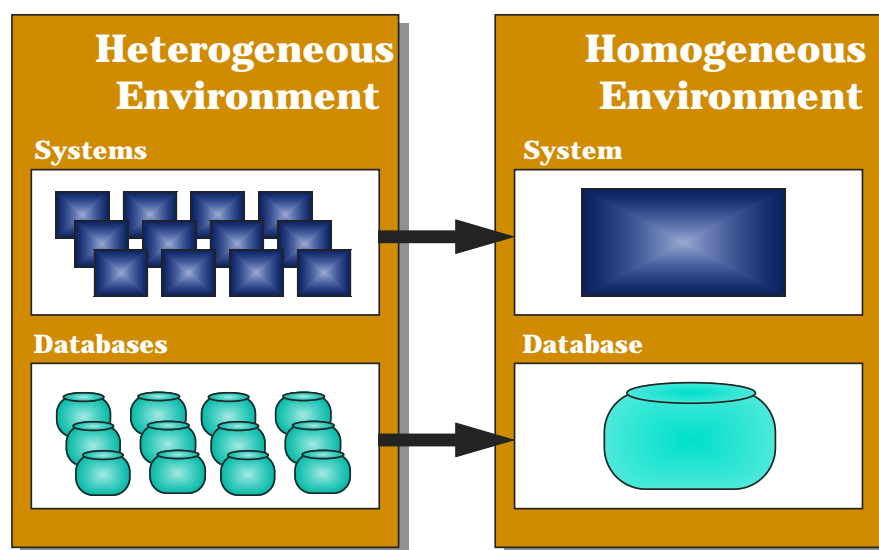


Figure 3: Enterprise Solution of Converging Technology

We're not talking about connectivity that allows the LAN or the WAN to function. That's the job of networking technology. We're talking several layers higher, in which a program on a PC (or another computer) can communicate directly with a program on the HP 3000. This form of connectivity, called middleware, is a true universal solution. Another way of defining middleware is to consider it an intercomputer or intracomputer messaging system through which messages can be sent and received between any two computers (PCs included), regardless of their unique operating environments. Effectively, middleware provides the glue that can bind different operating environments together.

The HP 3000: a Client-Server System

As for our HP 3000 problem, isn't this the answer we've been looking for? Those who said we had to go to UNIX and use relational technology in order to convert to client-server obviously got carried away.

Through middleware, you don't need to change hardware and operating systems. In fact, you don't even need to replace your mission-critical application. All you have to do is integrate the middleware messaging concept to be the principal interface between your current applications and your new PC application.

The simplest way to do this is to move the terminal processing code from your online applications, whether it be in VPLUS, DEL, or some other terminal processing technology, and replace it with a Visual Basic, PowerBuilder, Gupta, or C++ program on your PC. In so doing, you will also notice a substantial improvement in response time. Generally, moving the terminal screens and edit functions of your online environment onto the PC improves the overall application response time between 15 to 40 percent. And it should. You're not doing anything magical. You're simply using two computers to do the work previously done by one. Isn't that what distributed processing is all about? (See Figure 4.)

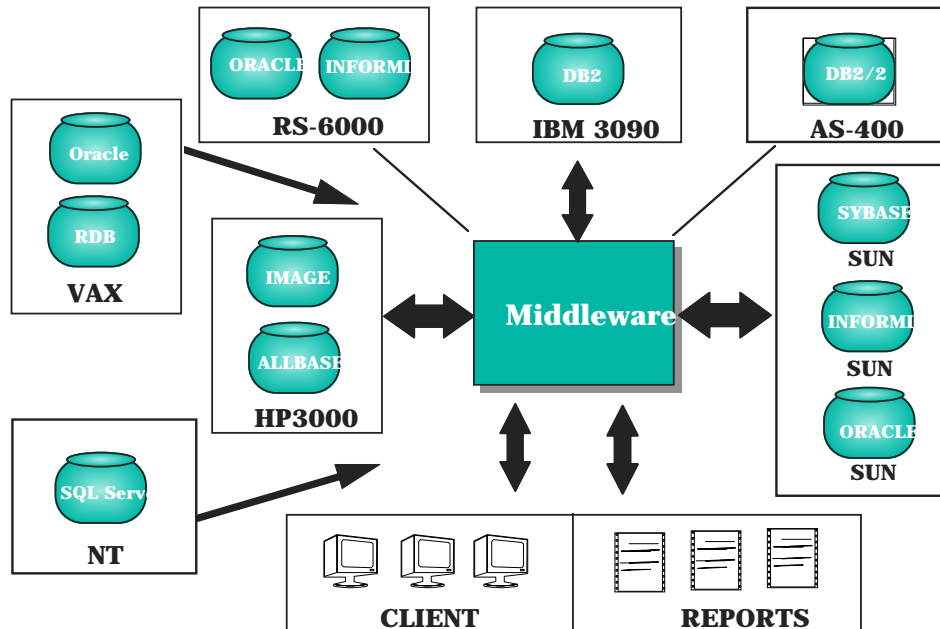


Figure 4: The Middleware Solution

The Middleware Criteria

Over the past year the market has been hit with a slew of new products professing to be middleware solutions to your HP 3000, client-server problems. And that's great. But because this area is new, suitable criteria are needed to measure the effectiveness of the different offerings.

Although more formal standards such as DCE and CORBA are being tested for practicality, I suggest the use of a few common sense standards in the interim, to avoid falling prey to product weaknesses. In general, a good middleware solution should offer the following capabilities:

- Connectivity across the operating environment
- Ability to integrate into your current applications
- Ability to retrieve information directly from the database
- Security provisions
- Simplicity of use
- Maximum flexibility
- The Connectivity Issue

Let's address these issues individually. First, if the middleware can't communicate through all the systems in your environment, it's not a practical solution. "All the systems" means that you should be able to communicate directly between the PCs in your environment and your HP 3000, as well as your NT server and HP-UX machine (if you have them). It's not uncommon these days to have an environment in which PCs are connected to an NT machine, which is connected to an HP-UX box, which is connected to an HP 3000, which might be further connected to an IBM mainframe (heaven forbid).

In this environment, the middleware should be able to support requests initiated from the PC that require information from or processing to be done on any of the other boxes. For example, let's assume that one of your PC users wants to retrieve manufacturing information from your HP 3000 and financial information from the mainframe. Is there any reason why this wouldn't be possible? No! Through an adequate middleware solution, you should be able to daisy-chain your way through all the systems in your network, gathering information and relating it back along the way. And it shouldn't take a lot of effort to do this. In client-server jargon this is referred to as n-tiered architecture.

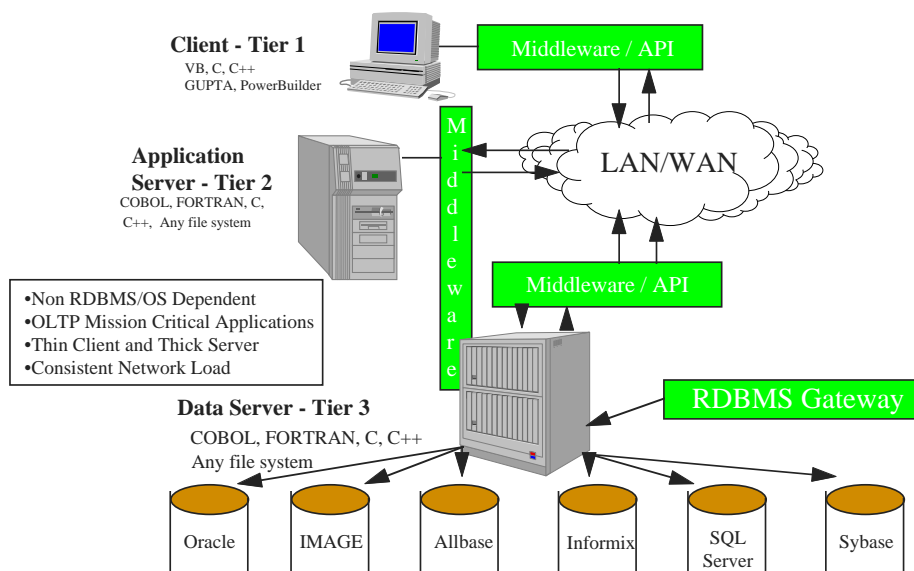


Figure 5: Three-Tier Architecture

Unfortunately, many so-called client-server middleware products don't support n-tiered architecture. They support only two-tier architecture. Some support three-tier, and with the advent of the Internet, three-tier will quickly become the minimum requirement, because nobody is going to make his main system a Web server. You might as well invite the hackers to your company information picnic. So, make sure that a package offers at least a three-tier solution, and preferably an n-tier solution. (See Figure 5.)

Current Application Issues

Middleware that doesn't provide a way for you to interface with your current applications is not a good solution. Every shop has systems that have grown old and need to be replaced, as well as systems that are working perfectly well and don't need to be replaced. So, why should you be forced to replace all of your systems, just because you've been pressured to go to client-server for the past five years? There are alternatives. (See Figure 6.)

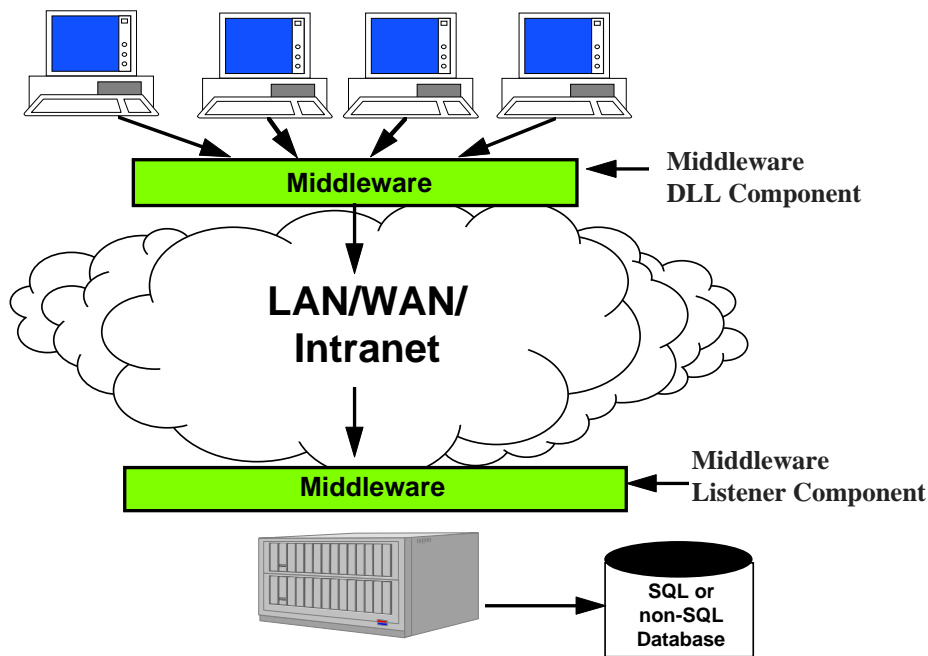


Figure 6: The Middleware Solution for Current Applications

Why take the revolutionary approach (and, by the way, the most expensive and most risky one)? Why reinvent the wheel if you don't need to? If an available tool allows you to move to client-server and yet retain your current working applications, wouldn't you want to use it? This is part of an evolutionary approach. Add on, rather than discard and redevelop. Besides, not every PC query is database-related. Your current applications may access KSAM files or maybe even MPE files. Nothing says information lookups in the client-server world require only relational database access. In some cases, you might not want to do any database access at all, but simply look up a spooler file or perform a SHOWJOB. Every access doesn't have to be a database access.

In examining different middleware products you'll probably note that different products use different client-server terminology, so don't be confused or fooled. Object-oriented technology is very much in vogue today. If you don't understand the vocabulary used, ask the vendor to explain what it means.

Generally, the terminology is simply a new way to explain a concept you already are familiar with. DP developers love to invent new terminology.

By the way, Geoffrey Moore, in his book *Crossing the Chasm*, states that in every software technology change, the mainstream of the market (represented by pragmatists and conservatives) requires an evolutionary solution and will not accept a revolutionary one. So, why should you? (See Figure 7.)

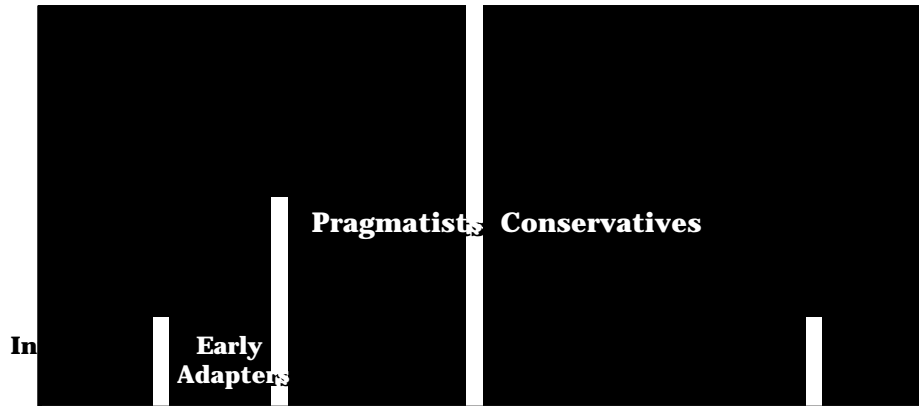


Figure 7: Moore's New Technology Acceptance Curve

The Database Retrieval Issue

As stated earlier, ODBC is a great idea. Unfortunately, it wasn't properly implemented. Requiring drivers to reside on the client defeats the purpose of client-server. The server is being used as a dump file server, which isn't much different from the mainframers and HP 3000 professionals using PCs as dumb terminals. The client-server idea is to split data processing tasks between the PC and the server so the processing load is shared equally. The user-based tasks should naturally be performed on the client, and the system-based tasks should naturally be performed on the server. Sometimes common sense prevails, and sometimes it doesn't.

In any case, placing most of the processing burden on the client (which ODBC tends to do) does not take advantage of the distributed processing power that could be achieved if the drivers resided on the server. For one thing, placing the drivers on the server would significantly cut down the processing load on the network, speeding up the overall user response time. More important, it wouldn't limit the client-server architecture to a two-tier model. (See Figure 8.)

Simply moving the ODBC drivers onto the server makes it possible for ODBC to perform n-tier processing. Let's take the example of an ODBC request that wants information from an Oracle database residing on an HP-UX server, an IMAGE database on an HP 3000, and a DB2 database residing on an IBM mainframe. Under the current ODBC two-tier architecture, there would have to be three separate requests, addressing each of the platforms individually. If the drivers were on each of the associated systems, a single request would have sufficed. All the filtering could have been done at the server level, with only the necessary results returned back to the client.

Not a bad idea. In fact, the idea is so good that all the new ODBC suppliers are altering their software to be server-resident--except one. And, I'll bet you can't guess who that one is? I'll give you a hint. Its name starts with an M and ends with a T. In defense of their position, though, remember this firm believes that nothing significant exists outside of the PC environment.

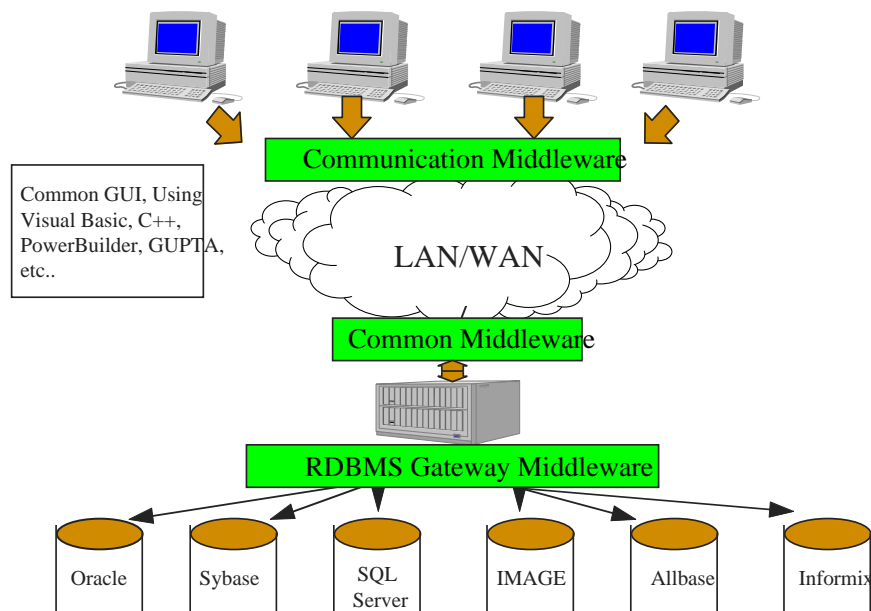


Figure 8: Server-Based Data Access

Since the drivers of the new ODBC developers are now being placed on the systems on which they belong, the next step is to develop new drivers that allow nonrelational access as well as relational access through ODBC. I like to refer to this as the next generation of ODBC. Users don't know or care which database ODBC is talking to, so long as it is an SQL-knowledgeable database.

Under such conditions, why can't ODBC talk to a KSAM file or a TurboIMAGE database? It could do so if an ODBC driver were to be developed for it. Of course, this is where we get into the issue of open systems standards versus practicality.

Security Provisions

Let's move on to another important area, security. When online environments consisted simply of dumb terminals chained to central systems, security was easy to handle. Today, with environments of 2,000 to 10,000 PC, Internet, and intranet hookups through internal Web servers, a slew of video terminals, and a maze of networked systems, all accessing information within the company's enterprise, security problems become very important. It goes without saying that anyone can put a "sniffer" on the network and intercept any communication. Security, therefore, must be the tightest when sensitive information is being passed across the network. (See Figure 9.)

There are gobs of security packages to help secure your network, but the most effective way to do so is to integrate your security into the middleware function. Since the middleware layer is already in place for communicating between programs on the PC, the HP 3000, and other networked systems, it is natural that security be an optional by-product of the messaging service. The information can then be easily encrypted and compressed at the sending location and decrypted and decompressed at the receiving location. Why add another external, non-integrated layer to the process, complicating the issue and adding to processing overhead?

The other advantage of having the security as middleware is that you can easily configure it in, when you want to use it, and out, when you don't. It becomes a simple parameter request of the application as it passes through the middleware.

In any event, don't implement a middleware solution without having a security provision in place.

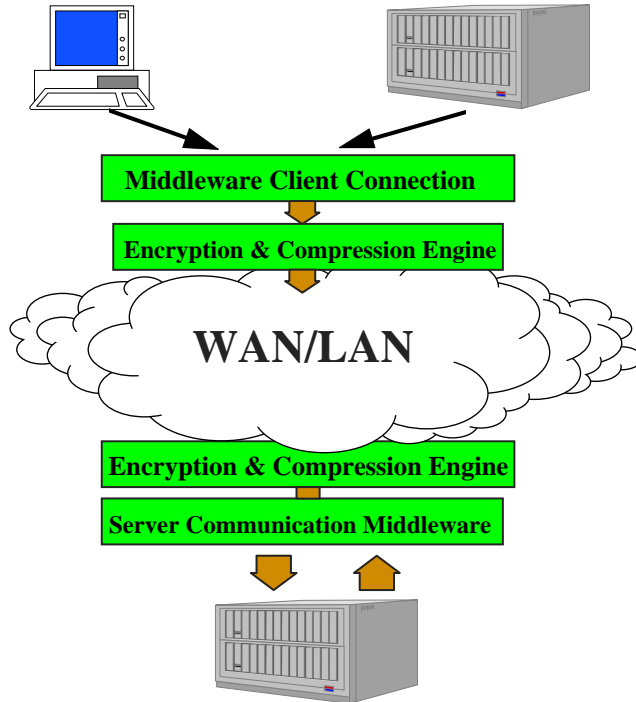


Figure 9: Retaining Security Across the Network

Simplicity and Flexibility

Like anything else, the simpler middleware is to install and use, the faster it is to implement. You probably have enough problems just keeping your head above water and adequately servicing your request backlog, so use a product that allows you to be up and running in the shortest time.

Flexibility is probably an even more important issue, since in most shops there is no stability. Today, management may decide to buy an HP-UX machine. Tomorrow, they may want to install Oracle on it. A month or so down the line, they may decide to try a copy of Oracle on an HP 3000 because they have decided to standardize on Oracle. Then, next year they may want to try out NT and replace Oracle with Sybase or SQL Server. How do you exist comfortably in this kind of environment?

You need tools that allow maximum flexibility. A good middleware product should allow you to configure your software so you can be relatively hardware platform and database technology independent. It should also allow you to migrate from your current HP 3000 online environment into a client-server environment at a pace at which you feel comfortable, not one that governs you.

Summary

There is no reason why you can't go client-server effectively with your HP 3000. As a matter of fact, client-server on the HP 3000 makes a lot of sense. It's faster than an equivalent HP-UX system, it's more reliable, it has more administrative tools, and it is cheaper to implement.

Like so many other companies, yours may have budgeted \$4 to \$10 million to implement SAP or some other client-server package. So you probably don't want to disappoint management by saving them a few million dollars! But--what the heck--they might be interested. On the other hand, why should you

be denied the opportunity to get yourself involved with all that is associated with a package solution--all new hardware, new applications, new networks, new maintenance tools, data conversion programs, extensive parallels, technical training, user training, management training, and cutovers? And, after it's all done, you can start to attack the change requests and reporting backlogs that always follow. Won't it be fun?

There is an alternative. You could keep your HP 3000 and use a middleware solution.

Brad Tashenberg is co-founder of StarVision, the author of DBGENERAL software, and president of Bradmark Technologies.