

Paper #: 1005

# **Migrating Your System To Client/Server**

**By Neil Mehta  
Bradmark, Inc.**

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

The information technology industry has seen phenomenal rates of technological and computing discipline changes over the last decade. The advent of PC's and networks forever changed the ideas and methods not only for personal computing, but corporate computing as well.

Several technological waves have billowed since the dawn of corporate computing. We all started out using centralized computing on mainframes and mini-computers. When personal computers and local area networks gained prominence, the client to database server wave broke, empowering end users with a wealth of decision making information. In the latest wave, client to function (or application) servers are approaching. Understanding the options and nuances in these latter-day technology waves can be confusing. Question abound, like what exactly is client/server anyway? And, what's driving it? What business needs brought it into existence?

Other concerns involve what your going to do with you legacy solutions. You know, those time-tested and reliable systems that may now be bursting at the seams. Users may be clamoring for added functionality, or an easier to use system. What can be done to preserve that legacy investment in software, hardware and support – not to mention the business logic driving your organization tied up in those systems? As you know, it's not prudent to throw the baby out with the bath water! Preserve your legacy investment, migrating to client/server solutions in a manner that's both expedient and robust. Migrate in a manner that will preserve your baseline of existing end-user services, enhancing it with the latest client/server technologies.

Today, *Client/Server* promises to deliver the newest and hottest technology to our users and to us. The problem we face, however, is that the *legacy systems*, such as the HP3000 cannot just be turned off. We must find a way to migrate those systems to take advantage of today's state-of-the art technology. We must shift our thinking toward making the PC our front-end, and leaving the processing on the HP3000.

### **Client/Server ... What is it?**

Feeling lately like you've been hit by a technology tsunami? No wonder! We've been through at least three waves of computing since the 1960s, each with its own nuances and niceties: centralized computing; client to database server; and recently the client to function server wave.

Several years ago our industry began a paradigm shift toward a new way of data processing called *Client/Server*. Theoretically *Client/Server* is the way all computer systems operate. However, all the processing takes place within the confines of a single computer. In the early years of commercial data processing, data was entered and retrieved on computer via a card deck and reader/sorter. The cards were fed into a computer that manipulated the data, then printed the results on more cards. In that scenario, the reader/sorter was the *client* and the mainframe was the *server*. Then came *on-line data processing* and we again shifted our thinking to using this more advanced technology. But batch processing did not just go away though. Data processing professionals merely gave it a new coat of paint by finding better and smarter ways to use it. Batch processing became our *legacy system*, and we still use it.

The first wave of corporate computing crested using centralized computing comprised of a single-platform solution, usually a mini-computer or a mainframe. The data and the application code that manipulates it resided together in one place. These so-called "legacy" solutions were usually highly secure and manageable, and scaled well in the beginning. But, these solutions had a tendency to be cumbersome and slow to change, ignoring today's empowered users and the low-cost explosion of processing power on personal computers.

In the next wave, the client to database server solution surged. This is usually what most people think of if you mention "client/server." The term "client/server" actually predates most of these solutions. It arrived around 1987 with IBM's conception of Cooperative Processing and SAA, waned a year or so

later, then burst back on the scene with a vengeance. Client/server technology took hold due to the use of local-area networks (LANs) and distributed computing, both of which have been in use since the early 1970s. In client to database server solutions, a second processing platform (server) holds the data. The application code usually (but not always) lives and executes entirely on the client, an individual's PC, while the database stays on the central platform serving multiple clients. An example of this would be a Powersoft's PowerBuilder client linked to any SQL database server. You don't have to keep all the code on the client in these client to database server solutions. Application code can also reside within the database as stored procedures or triggers. SQL calls or commands issued on the client invoke tasks running on the server, such as those preserving referential integrity within the database.

The latest computing wave is rippling with client to function (or application) servers. Here the application function itself resides on at least two platforms. Some software functions go on the client, some go on one or more servers, with placement depending on application performance and security needs. These solutions have a tendency to scale better than its client to database predecessors due to division of labor.

### **Client/Server ... What's Driving It?**

The power and flash of graphical user interfaces helped bring in the tide of these latter waves. Users value client/server architectures because they can combine the freedom of PC applications with the ability to access legacy data. With these new architectures, corporate decision makers felt user's PC skill sets could be fully leveraged. We all thought less powerful hardware could be used. We thought we'd save money over legacy hardware and system development. History has proven otherwise, however, especially when trying to scale these early systems beyond the workgroup or departmental level.

Within most organizations, these waves of client/server computing have billowed from the top down, more than from the bottom up with upper management making most of the requests. As such, it is a much more business-driven decision than a technology-driven one.

### **What happens to my legacy solutions?**

For the most part, business today still takes place using the legacy technology of mini-computers and mainframes moving towards client/server implementations. Legacy application code that took years to tune, tweak and streamline contains the core of "business logic" that drives these businesses in the form of payroll, accounts receivable, and the like. Organizations full of client/server zeal are eager to make the move, but hesitant to abandon the investment in hardware, software and business logic that's been the linchpin holding their business together.

How do you satisfy the need to empower employees with business information and flexibility, while preserving your investment in these legacy systems? Is scrapping the lot and starting anew the only option? Discarding those solutions for brand new development may be foolhardy. Experts warn that seventy percent of the client/server applications built today will be obsolete within two years. These applications are built expediently, not strategically. Pressure from management to meet deadlines frequently produces rabid, not rapid, application development.

One company with a client/server quandary had a comprehensive wage and personnel administration system which took five years to develop. The system includes around 2,000 programs connected to 60 databases serving some 60,000 people. Users needed the ability to easily modify information and create custom reports. Two options were available: rewrite the system entirely as a PC-based system or use the existing system. The entire rewrite would cost \$6 million requiring 100 man-years to accomplish. The existing system was already very functional - they just needed a graphical user interface to improve this functionality. Why re-write it?

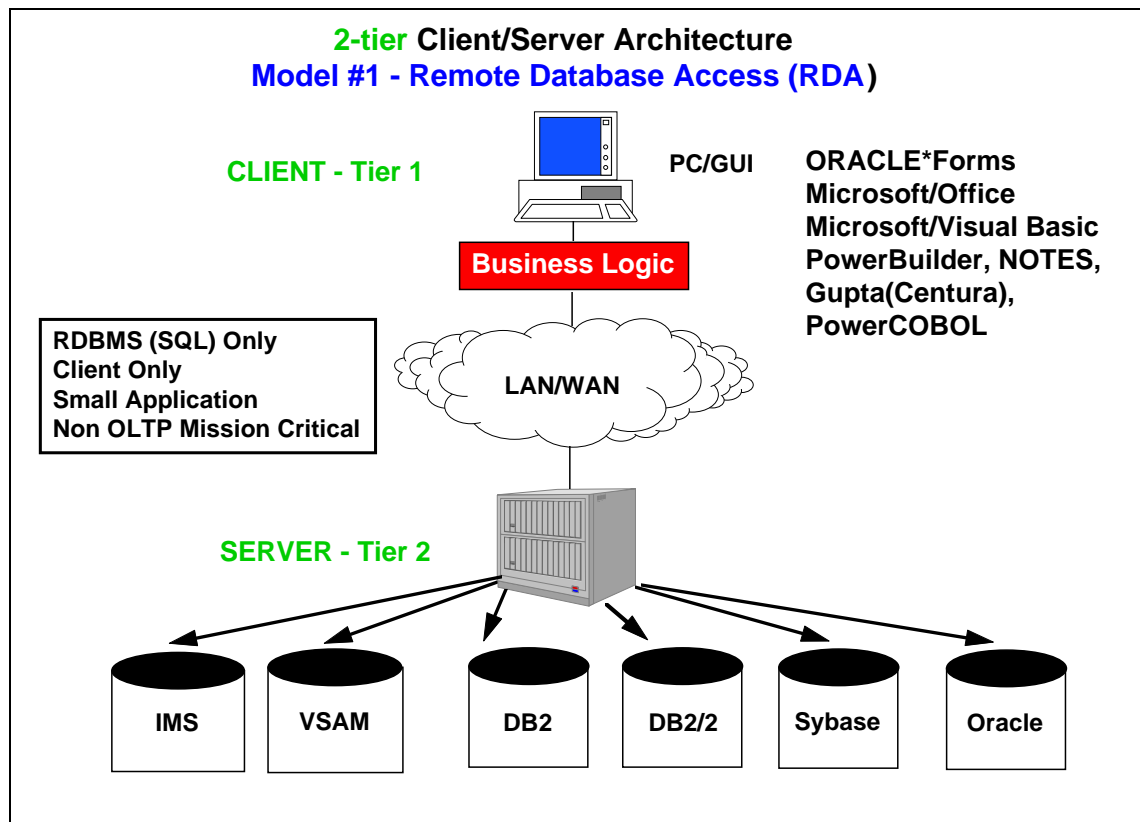
Building a distributed-computing infrastructure from scratch can be minefield — complex, costly, and risky. This company found the surest route through these land mines by taking advantage of middleware services, easing into client/server development.

### **Middle... what?**

Middleware consists of sets of software services parceled out among different hardware platforms. These services connect the application, the operating system and network services on one or more system nodes to other platforms on the network. These services not only help connect client PCs to servers, they can also bridge the legacy world to the PC world.

Middleware services sit just above the network layers providing a wealth of critical program-to-program, communications, and data management services. Application developers can use a single set of application programming interfaces (APIs) to connect graphical development environments to legacy data and applications hosted on mini-computers or mainframes.

These services can even enable “peer” connectivity between disparate platforms without requiring specialized gateways. Mainframe or mini-computers, once viewed as “host-only,” can act as a client by employing middleware services. These new “hosts-as-clients” can then request information from another network node (or even run applications on another node) – whether the other node is another mini-computer, mainframe, or even a LAN server.



*Typical Two-Tier Architecture - ODBC Type*

These middleware services can bridge the gap between your legacy development efforts, and the bright horizon of client/server computing, letting developers use familiar languages and tools. The good news is that you don't have to build all this connectivity software yourself. There's a energetic market full of vendors eager to service you with their middleware tool solution.

### **Why not use ODBC? I thought it was basically free?**

I can hear the application developers crying, "Why buy middleware? What about using ODBC?!" Doesn't it already come with most of the client/server databases out there?" Remember the adage, "you get what you pay for." Advantages and disadvantages exist when using ODBC, especially as a application developer.

When using ODBC, your program makes a general request for data to the ODBC manager, an application on your PC that directs data traffic. The ODBC manager determines the designated database for your request and then calls the appropriate database "drive" program. This database driver then takes the request, translates it into a format the target database can understand, and then sends the formatted request to the database server to process. When the server is finished, all the results are shipped back to the PC. The extra software layering on ODBC generally uses a least common denominator approach. A common set of programming interfaces is published regardless of the back-end database. This common set of APIs usually reduces performance compared to direct access using proprietary APIs.

Another ODBC drawback is that developers don't usually have access to the SQL code generated by ODBC drivers. Therefore, developers cannot optimize the SQL code. ODBC documentation and support is poor to practically non-existent with the exception of online sources like CompuServe or the Internet.

Your particular database may not even be supported by ODBC drivers. If it is supported, the ODBC drivers may not be optimized for full performance. If your particular database doesn't come with ODBC support or is a performance pig, you might be tempted to use third-party ODBC drivers. The circumstances becomes a little sticky when you buy a third-party ODBC driver and then start running into access problems. Who do you blame? Is it your application programmer's problem, or the database vendor's problem? Or is the ODBC driver to blame? Problems can resort into finger-pointing performances.

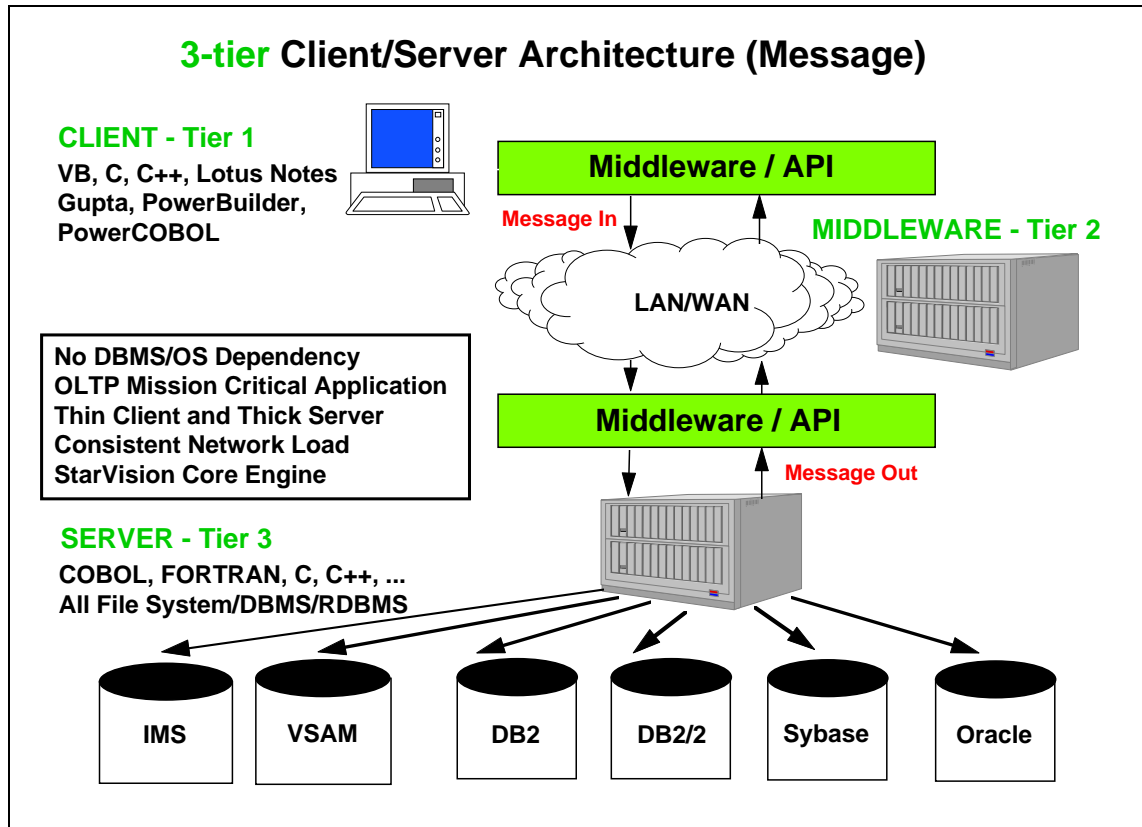
### **What about SQL?**

I can hear others wondering, "Isn't SQL used to access databases?" This is true, but the Structured Query Language (SQL) falls short when trying to use it alone to access a database programmatically. SQL cannot allocate program memory. It cannot address program variables. It cannot perform loop processing. In short, SQL was not designed to be a full functioning programming language.

Some SQL databases support stored procedures allowing flow control and some rudimentary programming. SQL statements can also be embedded within other programming languages giving developers more programmatic control over database access. SQL and ODBC are useful when generating ad hoc requests, but generally fall down on the job when used programmatically, especially if the ODBC driver has not been optimized by the database vendor.

Proprietary APIs from database vendors can provide programmatic access to database systems and other data sources. Disadvantages can arise when using vendor-supplied APIs or ODBC drivers, however. Programs written with these APIs are usually limited to accessing only one type of database - coincidentally, the one provided by the vendor who wrote the API. The drawback?

If you need to connect or move to another type of database, be prepared to re-write all your database programming from scratch.



*Typical Three-Tier Architecture.*

#### Client/Server Checklist

Jumping into the client/server arena can produce salvation or peril. Where should you begin? What pitfalls should you avoid? The list below can help you get started.

This list is not meant to be comprehensive by any means, but can help you along the path to distributed computing. Walk through the questions, checking off the items that apply to you and your organization. When you're done, you'll have a launching pad of recommendations and ideas to navigate you.

#### Can you start from scratch?

- ◇ I can start from scratch. My previous hardware and software investment has been minimal. Place a checkmark beside recommendations #1 and #2.
- ◇ I can't start from scratch, I've got legacy mainframe or mini-computer systems, hardware, and data. Checkmark recommendation's #2 and #3.
- ◇ I can't start from scratch, I've got legacy first generation client-server systems, hardware, and data. I want to migrate to more robust solutions. Checkmark recommendation's #2 and #3.

#### Do you have a environment mixed with legacy systems and hardware?

Determine the answers to the following questions up front:

#### Where should the data reside that needs to be created or updated?

**Where should the data reside that only needs to be read?** -- If user's don't need real-time data distributed databases with historical information may serve your needs.

**How many users will be using the system at peak times?**

**Where are these users located?**

Plan a gradual migration to client/server.

Review you existing systems.

Separate the data entry portion of the system from the core processing logic - that portion containing business logic that's been thoroughly tested standing the test of time.

Deploy a middleware solution that will connect your user's graphical environment to the core part of your legacy system.

Migrate further portions of the legacy system to a distributed solution over time without having to rewrite the user interface.

**What are your user's needs? What functionality do they need?**

My users need to make ad-hoc requests.

- ◇ My users need real-time access to the data for ad-hoc requests. Checkmark recommendation's #3 and #5.
- ◇ My users do not need real-time access to the data. Historical or snapshot data will suffice. Checkmark recommendation's #3, #5 and #6.
- ◇ My legacy database vendor doesn't have a connectivity solution that will communicate with my PCs. Checkmark recommendation's #3 and #6.
- ◇ My users need a system that's easier to use. Checkmark recommendation #4.
- ◇ My users need an on-line transaction processing system with the ability to update during business hours. Checkmark recommendation's #7 and #8.

### **Recommendations**

- ◇ Investigate PC based solutions.
  - The hardware costs are much cheaper.
  - The system's ease-of-use is much better.
  - User productivity is much higher.
  - New application development time is much shorter.
- ◇ Consider 3-tier architecture with client, database server(s), and application (or business logic) server(s).
  - Reduces process load on database server(s).
  - Reduces the amount of business logic programming necessary on the clients.
  - Can be scaled more readily.
- ◇ Consider utilizing middleware services to connect legacy information and systems to client PCs.
- ◇ Develop or deploy graphical user environments based on Microsoft Windows.

Once trained in one application, the learning curve for newer applications is reduced significantly.

- ◇ For ad-hoc requests from your users, check with the database vendor for ODBC connectivity to your database server for all the various types of client computers in your organization.
- ◇ Investigate database gateway or middleware services. These service PC clients with requests from legacy systems and data.
- ◇ Plan a scaled-down pilot using application platforms taking advantage of symmetrical-multiprocessing systems (SMP) like Windows NT. But beware! Know what the pilot will prove and what it won't. If your application will have to scale to hundreds of seats or to several remote sites, the pilot won't prove much. A major portion of your proof of concept will still be ahead of you.
- ◇ Ask your application tool vendors for help in finding people who have built similarly structured applications. Ask these people what their performance experiences have been, and what pitfalls to avoid.
- ◇ Investigate database gateway or middleware services connecting heterogeneous hosts.

### **Summary**

There is a sense of urgency to move applications currently running on the HP3000 to a Client/Server environment. To do so in one fell swoop is very costly in financial terms, programming resources and training of the users. With the concept, you can move at the speed you need to. You are in control of what processes you want to make Client/Server and how you want to do so. Most importantly, you do not have to change your existing database design or platform.