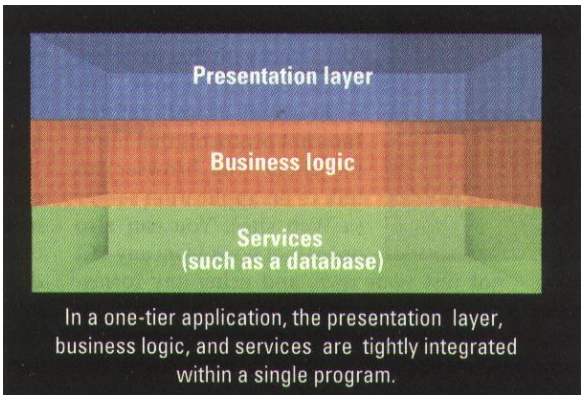
Understanding Client/Server Computing

By Will Fastie
PC Magazine; "Enterprize Computing;"

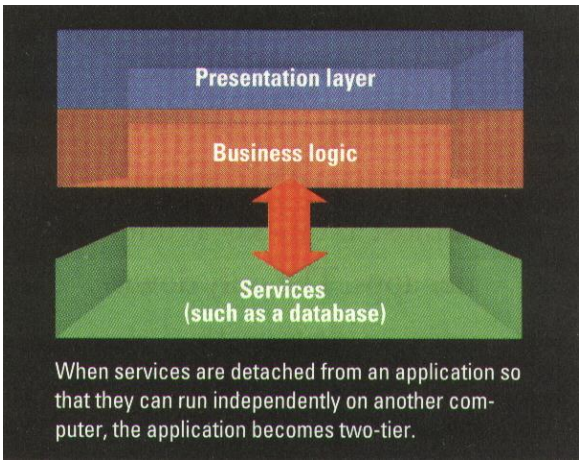
Back in the eighties when network equipment vendors first started to hawk their wares, the term *client/server* was used to conjure up a host of fancy capabilities offered by the new hardware. Today, client/server has a very specific meaning that has nothing to do with hardware at all. Even so, most folks still associate the term client with the physical computers they are using and the term server with some other physical computer to which they are connected. There's a glimmer of truth there, to be sure, but today client/server means much more.



In a one-tier application, the presentation layer, business logic, and services are tightly integrated within a single program.

We're going to cover the essentials of client/server computing. We'll define one-tier, two-tier, three-tier, and *n*-tier client/server program architectures; we'll explain the difference between a thin client and a fat client; and we'll discuss the middleware concept. But first, you need to understand a few things about general software architecture. Most application programs have three major layers. On top is the *presentation layer*, which provides the human/machine interaction (the user interface). The presentation layer handles input from the keyboard, mouse, or other device and output in the form of screen displays. In the middle is the *application* or *business logic*, the functionality that gives the application program its character. Application logic makes the difference between an order entry system and an inventory control system, for example; it is often called *business logic* because it contains the business rules that drive a given enterprise. The bottom layer provides the generalized services needed by the other layers, including such things as file services, print services, communications services, and, perhaps most important, database services.

For simplicity, we will confine this discussion just to those programs needing database services. That's what most architects first think of when they hear the term *client/server*, but the general rules apply to any kind of service.



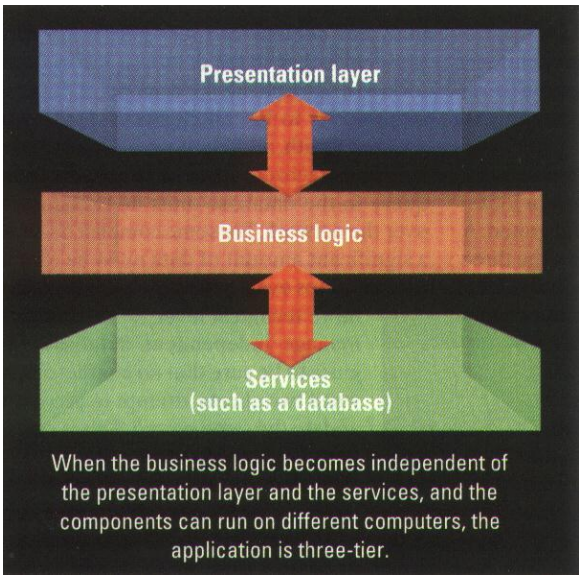
When services are detached from an application so that they can run independently on another computer, the application becomes two-tier.

The number of *tiers* in a client/server application is determined by how tightly or loosely the three program layers are integrated. A one-tier application is one in which the three program layers are tightly connected. In particular, the presentation layer has intimate and detailed knowledge of the database structure. The application layer is often interwoven with both the presentation layer and the services layer. All three layers, including the database engine, almost always run on the same computer.

One-tier applications are easy to design and write, especially with today's modern software tools. It's also possible to create a multiuser one-tier application by running the application on several PCs and having them share the database. The database can be stored either on one of the PCs (a *peer* solution) or on a file server. Note that each PC running the application has its own copy of the database engine; only the data is shared, not the database logic itself.

Multiuser one-tier applications are okay until the number of users gets large. The problem is that all the database work is performed in each client. For example, if the program needs to list all clients with the last name of Smith, all the information (indexes and data records) needed to resolve that query to the database must be transferred over the network. For some complex queries, a significant amount of data must be examined, perhaps even the entire database. At a more technical level, it is difficult to manage independent database engines to be sure that no conflict arises when two clients attempt to access or update the same record. This is called the *record-locking problem*.

The solution for the performance and technical problems associated with multiuser one-tier programs is two-tier client/server architecture. This type of application is written in mostly the same way as a one-tier application, with the exception that the database service (engine) no longer runs with the client. Instead, it runs on the server computer that contains the database itself. Some method is used to communicate between the application logic layer and the database service.



When the business logic becomes independent of the presentation layer and the services, and the components can run on different computers, the application is three-tier.

Today, the method that's most commonly used is the Structured Query Language (SQL), a language that can encapsulate complex database queries in relatively compact sentences. The SQL statement is sent to the database server, which performs the work locally on that physical computer and returns only the relevant information (the answer) to the client PC. Only the database services are separated from the application in a two-tier design. The presentation and business logic layers remain very intertwined, and both continue to have intimate knowledge of the database.

Two-tier applications are a bit more complex to write, but most development tools are equipped with a great deal of support in their integrated development environments to facilitate rapid programming. The tools are so good that two-tier programs can be developed almost as quickly as one-tier apps. The only downside is cost. Most tools provide database engines that can handle one-tier designs (such as the Jet engine, used with Access and Visual Basic), but two-tier applications require separate database products such as Oracle, IBM DB2, Sybase, or Microsoft SQL Server.

Three-tier client/server applications, as you may have guessed, break all three program layers into independent sections. We have already seen how the database service can run on a different computer. in three-tier designs, the business logic itself becomes a service, and that service can also be run on its own computer. When the business logic becomes a service, it is usually called an *application server* or simply *app server*.

Quite often, the app server runs on the same physical server hardware as the database. Colocation can sometimes benefit performance, so it is usually a first choice. But a key advantage of having an app server is that it can be located in the best place to serve the need.

If you run a business with sales regions, for example; some regions might have many salespeople sharing an app server that's running on a separate physical computer. However, the one salesperson in a remote region might run both the client and the app server on a single PC. No matter how the client and app server are configured, all app

servers might receive database services from a centralized computer, perhaps a mainframe. And no matter how the app server is deployed, all users operate the application in exactly the same way from the client. The locations of the app server and the database server are irrelevant to the user.

In a three-tier client/server application, the presentation layer usually does not have intimate knowledge of the database. Instead, the presentation layer communicates with its application server using a predifined message strategy.

The best way to understand this is to think about the Web. Your browser knows absolutely nothing about the structure of the database at Amazon.com, for example, but you are nevertheless able to interact with that database when you order a book. This is the result of well-defined Internet protocols that allow the client (the browser) to talk to an application server (a Web site).

Three-tier applications are much more difficult to build than two-tier apps. The biggest obstacle is that the software tools' integrated development environments are not aware of the three-tier model the way they are about the two-tier model. As a result, much more hand-coding is required to write a three-tier application. Three-tier apps are also harder to design, because they are somewhat abstract compared with their more direct two-tier counterparts. Software tool vendors are starting to release new versions that tout three-tier or n-tier development support, but it's not mature development technology just yet.

One point of interest is that a two-tier relationship exists between an app server and its database server. In other words, even though the client in a three-tier application doesn't have intimate database knowledge, the app server does.

The term "*n*-tier" is quite in vogue right now. In truth, three tiers are the maximum;"n-tier" refers to the fact that an application server can request services from many other services, and that the services themselves can ask for services as needed to respond properly to the client's original request. The message flows can get pretty complex.

Middleware is the term often used to describe the application or business logic present in an app server. Unfortunately, the same term is also used to describe generic services. For example, to communicate, the three tiers may use a messaging system. This is true middleware in the sense that it functions between, or in the middle of, the three tiers. When someone uses the term *middleware*, be sure you know what specific meaning is intended.

When designing and building a three-tier client/server application, most developers make conscious decisions about how much of the business logic to move into the independent app server. A good example of this is *input-data validation*. Most users like immediate feedback when a datum is invalid. If the app server lives on a different physical computer, there can be a delay before the presentaion layer in the client reports that an entry is invalid. To keep good performance, validation rules often live with the presentation layer in the client.

If little or no business logic remains in the client, meaning that it is exclusively a presentation layer, then the client is called *thin*. If the client has lots of business logic, then it is *fat*. The best example of a thin client is your Web browser. It is so thin that it

can connect to entirely different applications of which it has no knowledge and still provide a suitable user interface. The whole concept of the network computer hinges on building the cheapest, smallest device that will run a Web browser.

Client/server computing is really much more than a simple separation of a user's PC from a server computer. Hopefully the definitions provided here can give you a leg up the next time you're caught in a jargon-laden meeting.