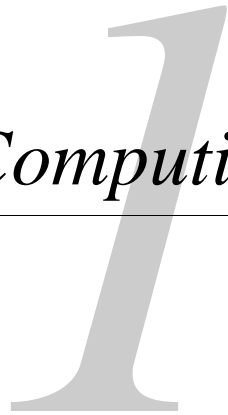


Distributed Computing



What this chapter covers:

- **Enterprise computing**
- **What is distributed computing?**
- **Business computing architectures**
- **One-Tier (Mainframe) architecture**
- **Two-Tier (Client/Server) architecture**
- **Three-Tier (Network/Multi-Tier) architecture**
- **Advantages and disadvantages of each architecture**

The lure of distributed computing is the promise of universal access and deployment of information on both the Internet and within company walls. Whether building and integrating large business applications, or placing applications on the Web and the Internet, a distributed architecture centered on middle tier processing is central to making information widely available and maintainable. The ability to develop, deploy, and manage business applications capable of supporting large-scale transaction processing on the Web and in the enterprise is made easier with a distributed architecture centered on application servers. Application servers provide robust features that insure the scalability and the reliability of the application—bringing data and processing that is distributed across a network together into an integrated, enterprise-wide system.

This book will focus on using PowerBuilder to build enterprise scale distributed applications using the Enterprise Application Server from Sybase. However, before tackling the techniques and capabilities of these products, it is important to have a good understanding of what is required when building large enterprise scale applications and why a multi-tier distributed architecture is best suited for the job. This chapter will describe distributed computing and the advantages and disadvantages of different business computing architectures.

1.1 *Enterprise and information technology*

Doing business in today's competitive marketplace requires that corporations constantly change and adapt to new pressures and demands. These demands are due, in part, to the globalization of the business market, the increased number of companies that are merging (requiring an integration of business processes and systems), and the accelerated pace of business and technology changes. Businesses can no longer focus on regional competition and local markets because improvements in technology, telecommunications, and the Internet have made the world an open market. New competitors can be as far away as the other side of the world. Rapidly meeting the needs of the market and customer has become very important. The ability of a company to identify the need for changes in the market place and be the first to make those changes will enable it to overtake its competition.

In today's business world, one of a company's most strategic assets is its information. Disseminating this information to appropriate users across the company requires integration of data that is currently spread across individual systems. If information is not used within the company to create a competitive advantage, then it is not being used to its full potential. In a marketplace that demands more services and value to secure repeat busi-

ness, a company must insure that more information is provided to customers and employees, and that the information is useful, complete, timely, and of a high quality.

This has forced many companies to re-invent themselves and the way they think about doing business. Being able to implement new strategies quickly, cut costs, improve services, and increase worker productivity has become very important to remaining competitive. Technology has become an integral part of a company's strategy for effectively managing the demands of competing with and staying ahead of the competition. Businesses simply need systems that can provide a flexible, reliable and integrated solution to their problems.

Enterprise technology investments are made to meet the following goals:

- increase productivity
- improve customer services
- improve communication and knowledge sharing
- automate manual processes
- reduce costs

1.2 Enterprise applications

An application is considered mission critical when it provides information and processing that is critical to the operation of a business. Other characteristics that a mission critical system typically possesses are a large user base, high availability (24x7), and heterogeneous data sources. Building enterprise applications requires more planning, integration, and scalability than a mid-sized or departmental application. Features such as load balancing, fail-over, and scalability are very important. System failures can result in the entire business being down—leading to the loss of millions of dollars. A distributed computing architecture allows a company to utilize technology in satisfying its corporate goals and focusing on its core business while providing a scalable and reliable computing environment.

1.3 What is distributed computing?

Distributed computing describes a computing architecture where portions of an application are executed on several machines connected over a network. When discussing distributed computing, we often refer to tiers—as in two-tier or three-tier architecture. A *tier* is a layer or level in computing architecture that can perform a particular level of processing. The mainframe environment is described as a single tier architecture because all the processing

takes place in a single location on the mainframe. Once we move processing from a single tier, we are dealing with distributed computing. Software is no longer one large, centralized application. Software solutions are instead divided into two or more applications that work together across process spaces and a network. These applications work together in a relationship where one application (the client) requests services from another application (the server). The client is the initiator, requesting services and resources from a server to complete its tasks. The server passively waits for these requests, performing the necessary tasks and returning any results to the client. This is called the *client/server model*.

Distributed computing is a more complex architecture that requires networking, middleware, and communication software to manage the separate processes and keep them working together. In a typical client/server architecture—which is also known as a *two-tier* architecture—processing takes place on both the client workstation and the database server. This is depicted in figure 1.1. In this architecture, the client typically handles the presentation and application logic while the database server handles data storage and retrieval. Communication between the two tiers is commonly done using SQL, allowing the client to request and change data.

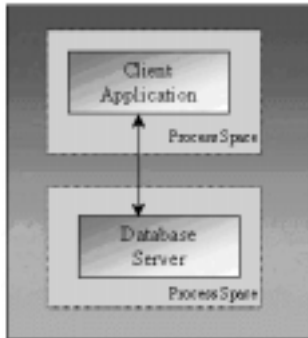


Figure 1.1
The client/server computing model

Three-tier architecture adds a middle tier—the *application server*—to the mix, creating three levels: client workstation, application server, and database server. Application processing takes place on the application server. This is illustrated in figure 1.2. In this context we use the term “application server” rather loosely—applying it broadly to the middle tier, where application processing takes place, rather than to the application server market that includes products like Enterprise Application Server.

In a three-tier architecture, it is possible for a tier to act as both a client and a server. The application server is a server (receiving requests from a client workstation) and is

also a client (making requests to the database server). This architecture is also referred to as a *multi-tier* and *n-tier* architecture because, as we will see in this chapter, it is often advantageous to add several more tiers to achieve better performance, availability, and scalability.

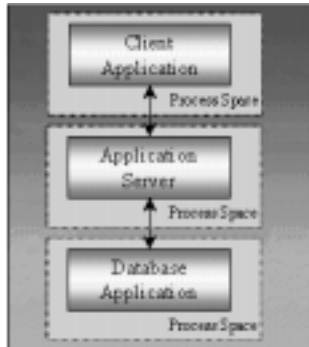


Figure 1.2
A three-tier computing model

As we continue, it is important to point out that the computing industry usually reserves the term “distributed computing” to describe a server-centric architecture with three or more tiers, and “client/server” to describe a two-tier computing architecture. We will do likewise in this book. A two-tier architecture, however, is still a form of distributed computing, and an *n-tier* architecture utilizes a client/server relationship.



Note

Distributed computing involves partitioning an application into logically separate components based on functionality and services so that they can be physically distributed between processes or machines.

1.3.1 Application partitioning

To divide an application so that processing can take place on several tiers, we need to divide the logic and determine where it will execute. *Application partitioning* is the process of dividing application logic and functionality into objects that can be deployed across a network, and is the foundation of distributed computing. Distributing an application involves dividing application logic into objects or components and placing them in one of three layers, based on their functionality:

- Presentation Logic
- Business Logic
- Data

Presentation logic

The *presentation layer* defines the user interface and describes how the user accomplishes tasks within an application. This includes data entry, presenting data, maintaining data, and initiating processes. The user interface can be as simple as a character-based, menu-driven system, or as complex as an event-driven GUI. The presentation layer should not contain business logic or transaction management logic. It should only contain a minimal amount of data validation to prevent presentation logic modifications due to changes on other tiers.

Some examples of presentation level services include:

- drag and drop
- searching and sorting
- selecting and deselecting rows
- entering or modifying data
- pushing a button
- choosing a menu item

Business logic

The *business rules layer* is where number crunching and business decisions are resolved, based on a set of operations designed to meet business needs. This involves validating information, storing information, calculating results, and looking up information that users request. Some examples of business logic include:

- The invoice number must contain six digits.
- A prescription without refills cannot be filled by a pharmacist.
- A tax return is invalid without a tax ID number.
- Claim costs are covered 100% when services are provided by a doctor in the network.

Data

The *data layer* includes such tasks as data definition and manipulation, data storage, management and retrieval, transaction processing, and SQL compilation. Although this

layer consists primarily of database services, others (e.g. file, print, and operating systems services) can also be located in this layer. These services are placed on resource servers designed to maximize sharing among all users. Some examples of data services include:

- data storage and retrieval
- management and sharing of specialized hardware, such as printers and faxes
- enforcement of referential integrity by preventing the deletion of a row that still is referenced by a foreign key in other tables
- locking a resource that is already in use



Note

Application partitioning can be defined as breaking up an application into one of three layers: presentation logic, business logic, and data, based on the type of functionality it provides.

Each layer is responsible for different tasks in an application, and every application is made up of these three layers—from simple spreadsheets to client/server and multi-tier applications. The problem is that most applications do not separate code along these boundaries, making it difficult to distribute logic. Object-oriented analysis and design divides an application into self-contained, specialized software components that are grouped together based on similar functionality or attributes. Application partitioning further divides application logic into application layers. Done correctly, an application can be designed using object-oriented techniques and application partitioning to provide a flexible foundation of reusable components that provide scalability to your system.

Once an application is partitioned, the layers will help determine where an object should be placed in a multi-tier application for maximum reuse and performance gains. Placement of the objects that make up each layer onto a particular tier can have a great impact on the performance and scalability of an application. Figure 1.3 shows how application logic can be placed in single tier, two-tier, and three-tier architectures.

Let's see how the three logical layers match up with the different architectures.

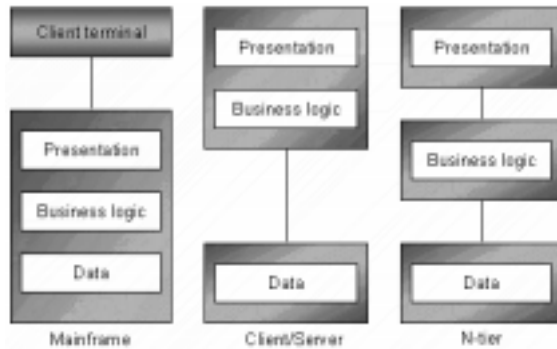


Figure 1.3
Logical and physical distribution of an application

1.4 Business application architectures

A *business application architecture* is determined by the number of tiers available for processing, and where each layer of functionality is placed. The architectures are influenced by the different technologies that are available.

There are three business application architectures:

- single tier or host-centric architecture
- two-tier or client/server architecture
- n-tier or network-centric architecture

1.4.1 The single tier or host-centric architecture

The *host-centric* computing model describes application processing that takes place on one machine or process, as depicted in figures 1.3 and 1.4. Centralized mainframe applications fit into this category. The mainframe is the single tier responsible for all the presentation management, processing, and data functionality. The terminals are used to display screens of data to the user and capture user input, but have no processing capabilities themselves. The mainframe does essentially everything, including supply terminals with presentation information.

Logic placement

In the *host-based* model there is typically no separation of logic into different processing layers. The presentation, business, and data layers are all centralized on a single machine so there is no compelling reason to divide the software into separate modules—the application logic is both logically and physically tied together. A typical mainframe program contains code that controls the interface, data access, and the business rules that

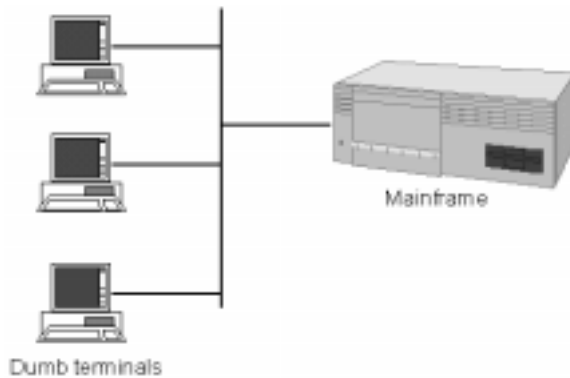


Figure 1.4
Host-centric computing architecture

are required to implement business functionality. Tying these layers together limits the reusability of application logic and causes duplication of rules in the software. This is one of the main reasons that maintaining systems on the mainframe is so difficult and why changes to business rules are so hard to implement. Enhancements usually involve finding all the code that references the rules that are being changed, as well as modifying the code that controls the interface or data access (that would not have been required otherwise).

Advantages and disadvantages

The mainframe provides a continuous computing environment that is able to support a large base of users and transactions. The constant availability of computing resources, combined with reliability and security, make the mainframe ideal for mission-critical applications. Its ability to provide reliable computing to large numbers of users has been proven over time, and is evident by the large number of businesses still running mainframe applications today and the continued growth in mainframe sales and services achieved by companies such as IBM. Other advantages found in the mainframe architecture are robust security and the ability to update software in a single place—eliminating distribution issues. Continuing to use these applications, however, comes at a heavy price in both dollars and flexibility. Mainframe solutions do not use open technologies, so companies are forced to use a single vendor. The vendor provides all of the hardware, software, and networking equipment required to support the mainframe environment and build applications. This equipment and software is often incompatible with other vendor implementations and is very expensive. While the mainframe excels at capturing and storing large amounts of data, long development and maintenance cycles make this architecture resistant to change. The time and resources required for maintenance and

enhancements do not help businesses quickly adapt to changing market trends or implement new strategies. The costs, time, and effort associated with correcting the Year 2000 problem, for example, illustrate the difficulties in making changes to business processes automated by the mainframe. In addition, the poor user interfaces and single tier processing do not allow end-users to make effective use of their time or analyze information outside of the host environment.

1.4.2 The two-tier or client/server architecture

In a two-tier architecture, software is no longer one large, centralized application. Software solutions are broken into two applications that work together in a client/server relationship. The typical client/server implementation is used to describe a custom business application written in C++, or a 4GL—like PowerBuilder—to access a database server over a local area network (LAN) using SQL. This is illustrated in figure 1.5. The application is used to solve business problems—usually at the departmental level—and may receive information from an existing mainframe system through batch feeds and file transfers.

The client workstation, unlike the terminal in the host-centric model, plays a large part in the client/server architecture. It typically is responsible for handling the presentation (accepting and displaying data) and processing of business logic. Additional responsibilities include communicating with the Database Management System (DBMS) and managing transactions. The database server's duties include providing concurrent access to data by multiple users, accepting SQL requests and calls to stored procedures, and enforcing referential integrity and data consistency.

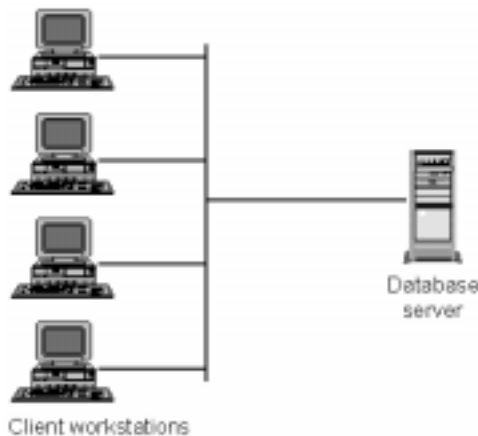


Figure 1.5
The client/server architecture

Advantages of client/server

Client/server architecture and the PC introduced many new capabilities and advantages that now make business computing more productive and powerful than a centralized computing architecture for many types of applications. The open solutions philosophy that came with client/server technology promised cross-vendor interoperability between products and reduced costs in hardware and software. The cost savings have improved the bottom line of corporations over the course of time; but the real benefits are the ability of applications to adapt quickly to changes in business requirements, and the increased productivity of both developers and end-users.

The use of multi-tasking workstations allow end-users to perform additional tasks while they are waiting for the server to process requests—utilizing the processing power of the desktop computer. In addition, the integration of desktop applications (word processor, spreadsheet, etc.) with custom client/server systems increases the productivity of the end-users. Data captured in client/server applications does not have to be re-entered into text editors or spread sheets for additional analysis. Often these components can be built into the client/server system using inter-process communication mechanisms or third party controls.

Client/server development is done using 4GL tools like PowerBuilder, enabling rapid development of applications that allow users to receive essential enhancements and functionality when they need them. Third party controls, object-oriented techniques, and object frameworks also allow for an increase in code re-use and sharing, which can decrease the amount of time it takes to develop an application. These benefits enable a business to reach its goals and become more competitive. Some other advantages include:

- GUI interfaces that make applications easier to use
- easy access to data and ad-hoc reporting capabilities that allow end-users to make more informed decisions
- sharing of network resources such as files, printers, faxes and databases
- interoperability with multiple tools, databases and platforms that leads to greater flexibility in designing a client/server system

Presentation logic and data placement

In a distributed computing architecture, the physical separation of machines forces developers to partition logic. When deciding where to place logic, proximity is a big factor in the performance of a distributed system. Placement of the presentation logic and the data layers is easy—the client workstation is the obvious choice for handling presen-

tation logic. The operations required by presentation logic are very closely tied with the input and output devices of the client. Placing presentation logic on the client eliminates needless network traffic that would be generated if the server had to handle interface logic. In addition, the large memory resources taken up by graphics and interfaces would be a waste of valuable server memory and processing time, especially when these resource demands are multiplied by a large number of clients.

The data is best handled by a DBMS that is designed to handle multiple client requests, and has advanced algorithms to handle data searching and retrieval. Allowing the DBMS to handle data searches and retrievals eliminates network traffic by passing only the information requested over the wire. A DBMS is also the obvious choice for handling the data storage, data management, and the transactional integrity needs of an application. In addition to data services, servers can enable sharing of files, printers, communications devices, and high-powered processors. Placing the data management and file services onto more powerful back-end machines takes advantage of the increased processing power, and utilizes sharing of resources across the company.

Business logic placement

It is the placement of business logic in the two-tier architecture that can be a difficult decision. There are two choices for the placement of business processing, and each choice has a great effect on the overall architecture—creating either a *fat client* or a *fat server* (also called a two and a half-tier) architecture. Each approach has its own set of advantages and disadvantages.

Fat client

The fat client approach is probably the most common two-tier architecture, made popular by 4GL tools like PowerBuilder and Visual Basic. In a fat client architecture, most if not all the business logic is placed on the client workstation (as depicted in figure 1.6).

The advantages of a fat client architecture are ease of development and the potential reuse of code by writing logic in encapsulated objects. Using this approach, however, often results in poor programming habits. Because developers are not forced to partition presentation and business logic, they often combine them in visual components. Coupling the presentation logic and business logic together limits the reusability of the code and makes modifications more difficult. Other applications requiring different presentation styles are forced to rewrite the business logic if they need the same functionality, which duplicates code and increases the difficulty of making changes to business rules. When presentation and business logic are tightly coupled, it makes moving the application to a



Figure 1.6
Two-tier (fat client) architecture

three-tier architecture a difficult task because the logic must first be separated into visual and non-visual components.

Even if good object-oriented programming is enforced, the fat client implementation is not completely responsive to making rapid changes to functionality. This is due to the difficulty in having to redeploy the application to each client workstation. Redeploying an application can be a major effort for a large number of users, slowing down the ability to get changes out. The redeployment effort also causes disruptions to the user community when updates need to be made to the client software. This can negatively impact productivity and cause versioning problems if some client workstations are not running the newest software.

Applications that require the client to handle most of the business processing will need more powerful computers to serve as client workstations—to handle the heavier processing loads and more complex systems—thereby increasing hardware costs.

A fat client approach also places a heavy load on the network due to the high volume of SQL statements sent between the client and the server. As applications grow larger and more complex and more users are added to the system, performance degradation occurs. This is due to the demands placed on network bandwidth, as more SQL statements are executed and larger SQL result sets are transferred. Sharing cached data or database connections is not possible in a two-tier architecture, because each client application is separate. This limits the scalability of the application, since there is no way to reduce the load on the DBMS.

The fat client architecture does not provide a secure environment. With data access software on the client workstation and database privileges given to users, an end-user can access and manipulate data outside of the application, bypassing business rules and other security measures.

Fat server

The fat server (or two and a half-tier) architecture describes applications that use stored procedures to help solve problems inherent in the two-tier architecture. Stored procedures are pre-compiled SQL statements and logic that implement business functionality. They are stored and run on the database server. Stored procedures can accept arguments and return values and result sets like a function. Client workstations can call stored procedures in a *remote procedure call* (RPC)-like fashion to perform tasks. Figure 1.7 illustrates how an application is partitioned in a two and a half-tier architecture.



Figure 1.7
2 1/2-tier (fat server) architecture

One of the benefits of using this architecture is lower network traffic. Lots of SQL statements and processing can be initiated with a single request rather than sending multiple SQL statements and result sets over the wire. The fat server architecture allows for less powerful client workstations and an easier deployment of changes, since stored procedures are maintained on the server. This approach also increases code re-use among different applications that use the same database, since these applications can call the same stored procedures.

Writing stored procedures forces developers to separate presentation logic from business logic, but it unfortunately ties the business logic to the database server. Stored procedures are written using vendor specific DBMS languages that are not compatible with each other. Using stored procedures and triggers locks companies into a single vendor solution, similar to the mainframe architecture. It also creates application code that is not very portable because it must be run on the DBMS. Furthermore, stored procedures cannot handle data that is not stored in a relational table format, making access to files or legacy data difficult.

An increase in performance can be obtained using a fat server architecture for applications with a limited number of users. However the fat server architecture does not scale well as the number of users and processing loads increase. These increases place greater demands on the database server resources that are used to handle executing stored procedures, maintaining connections, and performing data retrieval and management. The heavy use of these resources leads to bottlenecks and performance degradation, which cannot always be adequately solved by increasing server capabilities through hardware and software upgrades.

Additional limitations

Additional problems with a two-tier architecture that are independent of the business logic placement include:

- The DBMS maintains an active connection for each client consuming memory and server resources even when it is not in use.
- Increased demand to support connectivity to heterogeneous data sources due to company mergers and varying departmental implementation choices is not handled well. Client/server cannot guarantee data integrity with transactions that span more than one database, since transaction support is provided by a DBMS.
- DBMS vendors support different SQL extensions that make their solutions more attractive, but reduce the ability to re-use code in a heterogeneous environment.
- Without complicating the architecture with replication and duplication of data, the DBMS and data sit on one machine. The server can be boosted with additional resources and memory, but eventually even a single powerful machine will have difficulty maintaining adequate performance levels with a large number of users.

1.4.3 The n-tier or network-centric architecture

The distributed computing model extends the two-tier architecture used in the client/server model by adding a third tier to the architecture, where execution of business logic can be

handled. This additional tier is placed in between the client and the database server, and is called an application server—or the middle tier (see figure 1.8).

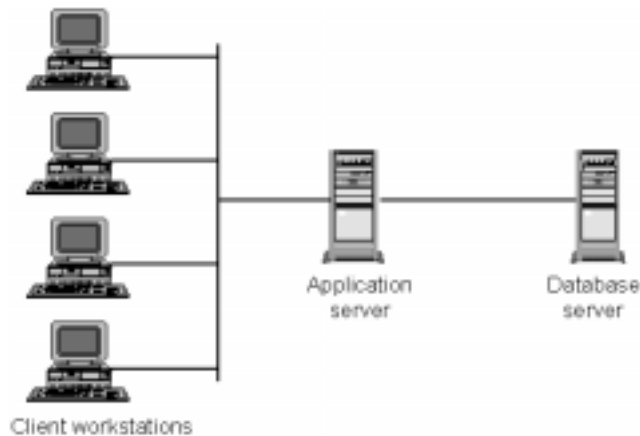


Figure 1.8
Three-tier architecture

With the addition of the application server, processing can be distributed more evenly. The ability to distribute application processing away from the client *and* the database server can be very important to system performance. The application server takes over much of the processing that was split between the client and database server in the two-tier model, including communicating with the DBMS, executing business logic, and managing transactions. The database server is no longer needed to perform services other than those associated with database management and data storage—reducing the processing demands placed on it. The biggest changes take place on the client, which is no longer responsible for connecting to the database or performing the majority of the processing (as it is in the client/server architecture). Client workstations can be left to manage interface processing and initiate server requests, reducing the amount of power required by the workstation and the amount of software that needs to be installed and configured—without losing the advantages of local processing and the use of desktop applications.

Logic placement

Like the two-tier model, the three-tier model forces the developer to decide where to place and execute application logic. Placement of the presentation logic and the data is handled similarly to the two-tier model and for the same reasons. What the three-tier model provides is another option to consider for the placement of the business logic.

Rather than make our decision harder, it actually makes it easier. The distributed model alleviates the fat client and fat server problems by thinning both the client and the database server and moving the processing to a middle tier on the application server (see figure 1.9).



Figure 1.9 Three-tier application partitioning

This provides the advantages of a fat server model—centralizing the logic and removing it from the client for easy maintenance and deployment—while reducing the processing load and potential bottlenecks that can occur on the database. It also solves the problems of a fat client model by allowing changes to business logic to be deployed by changing the middle tier. This avoids versioning and redeployment issues on the client workstation, while freeing the logic from being tied to the DBMS and its constraints.

This architecture, however, is not without problems. Network traffic can cause performance issues since there are more requests placed over the wire than in a two-tier application. Another problem is the need for users to get immediate responses to some of their work without having to make a request to the server. For example, performing basic data validation and type checking features over the network may annoy users who need quicker responses to speed up data entry. Although the goal is to place business logic on to the middle tier, it is probably not accurate to assume that all of it will be deployed this way. When deciding where to place objects containing business logic, always consider the application server the first choice. These guidelines can help you determine if another tier is more appropriate.

Place the object on the client when:

- the business logic is very stable
- the user requires instant responses from the system

Place the object on the application server when:

- the business rules are used by many applications

- the business rules change frequently
- the business rules contain sensitive information
- the application logic requires increased processing power

Place the object on the database server when:

- the business logic is tightly coupled with the data maintained by the server and is not processing intensive

Advantages of moving to an n-tier architecture

An *n-tier* architecture provides many of the advantages of the mainframe and two-tier systems, while solving many of the limitations found in each. The multi-tier architecture solves the limitations that are imposed by a two-tier architecture, providing the increased scalability, reusability, and maintainability required by larger mission-critical applications and found in a mainframe architecture. While relieving systems of some of the drawbacks of the two-tier model, the movement to an *n-tier* architecture manages to retain many of the advantages of moving to client/server in the first place. The architecture not only preserves these initial investments but also allows for the integration of heterogeneous computing environments including existing legacy systems and client/server applications. This is illustrated in figure 1.10.

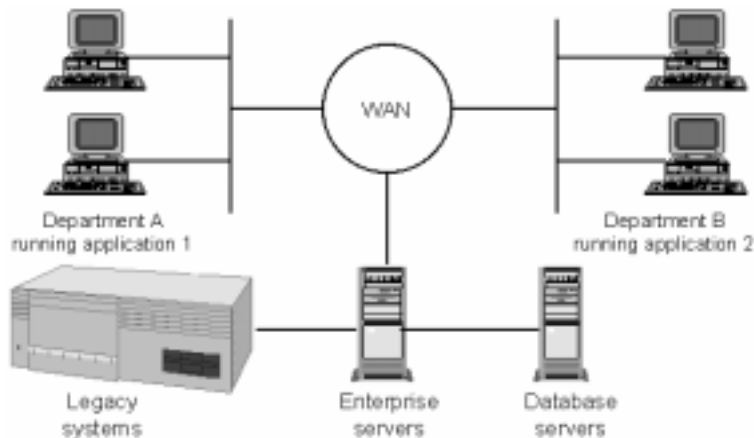


Figure 1.10
The Enterprise Solution

Web-enabling the enterprise

The *n-tier* architecture is essential to Web-enabling the enterprise. Departments that are located across remote geographic areas can take advantage of the Internet to access

corporate resources and information. Businesses can provide better services to customers by allowing them to access data and information that is pertinent to them. More sensitive applications, and those that do not require Internet access, can be run on the internal private network. All of these applications can use the same application servers and business logic, allowing for maximum reusability (see figure 1.11).

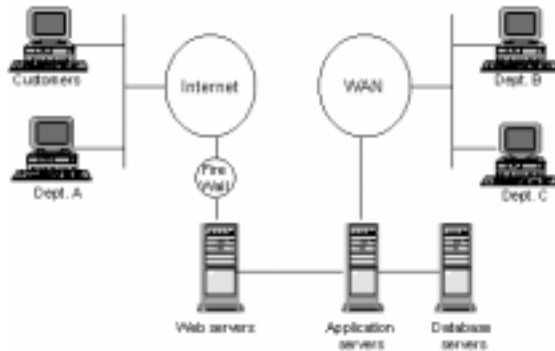


Figure 1.11
The Web-enabled Enterprise

1.4.4 Application server features

The move from a two-tier to an n-tier architecture is made to increase the scalability, reusability, and maintainability of an application. Let's examine the following features touted as advantages of a distributed application and explore why they are important:

- scalability
- connection pooling
- load balancing
- availability
- reusability
- maintainability
- integration

Scalability

Scalability is the ability of an application to perform acceptably under the stress of a production environment, maintaining a consistent level of performance as the demands placed on it increase. A system developed today should be able to grow with the business and handle the increasing workload without having to rebuild the application. Instead of rewriting the system, the increased demands are met by adding additional hardware. If an application is scalable, the application can handle increased demands by allowing addi-

tional servers (vertical scaling) or additional workstations (horizontal scaling) to be added as needed.

An application built on an n-tier architecture can handle both vertical and horizontal scaling. An application server can handle large numbers of users by managing server resources more efficiently (e.g., database connection pooling, instance [component] pooling, data caching). As more users and workstations are added to the architecture, the application server can scale by increasing the hardware capabilities on the server machine without impacting the client install base. When the load and the user base increases to the point where a single server cannot handle the processing, additional application servers can be added to the architecture (load balancing).

Database connection pooling

As an application expands in a two-tier architecture, and the number of users increases beyond what was anticipated, the processing load that the system is under increases. Eventually the increase in users and processing break the threshold that the application was designed to handle. Because the DBMS requires a connection for each user and must be deployed on a single machine with a finite number of resources, the system eventually reaches a point where it cannot be expanded.

The multi-tier architecture breaks this limitation by allowing the middle tier to pool database connections. Using *database connection pooling* a single connection can be shared by several client workstations. The theory behind connection pooling is that a client does not require the use of the connection all the time, so a database connection is only accessed when it is needed. This allows other clients to use the connection in a round-robin fashion. The application server software handles the connection management transparently and the connection is given to a client when an actual data request is made. This technique eliminates the necessity of maintaining one active connection for each client and limits the number of physical connections to a database server. This decreases the processing load and utilization of resources on both the application server and the database server, allowing the application to handle more users.

Load balancing

As processing demands increase, it is easier to balance the load by adding additional application servers to a multi-tier architecture than by trying to squeeze more processing out of the database server by increasing the server processing capacity. Replicating software in the middle tier across several servers allows client requests to be routed to the machine that has the least processing demands placed on it. Figure 1.12 illustrates this. The ability to do this seamlessly requires portable code that is not tied to an application

implementation, DBMS, or particular platform, so that it can be distributed across several machines. This makes the architecture very flexible and scalable. The ability to add to the middle tier is why this architecture is also known as an n-tier architecture. Adding additional application servers also enables the architecture to take advantage of *fail-over* and *fault tolerance*. These features enable the system to handle the crash of an application server by routing requests destined for the downed server to other available servers.

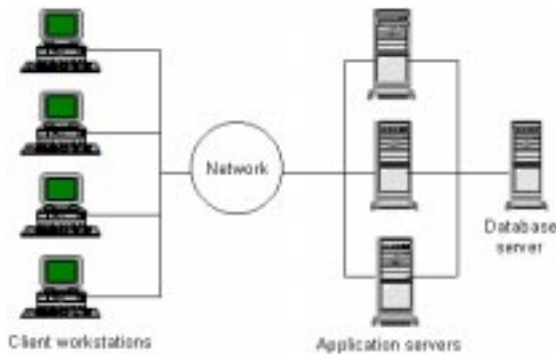


Figure 1.12
Load-balancing and fail-over



Note

The performance of an application involves every aspect of the system architecture, including the client software, middleware, network bandwidth, and application and database servers.

Availability

The n-tier architecture is used to run mission critical applications for an enterprise. These applications must be available and reliable in order for the business to function. This architecture can provide high availability by having software deployed on the application server and replicated over several servers to handle fail-over and fault tolerant capabilities, as well as load balancing. When a single application server goes down, client requests can be routed away from the faulty server to the remaining functional servers.

Reusability

An application server centralizes business logic and database access, much the way a database server centralizes database management and data storage. The goal is to place business processing where it makes the most sense, and where it can be widely used by many applications. In a three-tier architecture, business logic becomes centralized on an application server, thus increasing the maintainability and scalability of a system. Since the application server manages the objects used to handle business processing, several applications can connect to the application server and access the functionality. By centralizing application processing onto servers, we are able to get true code reuse. Instead of reusing an object that was created for one application by compiling it into another application, we can actually re-use the same object class and instance on an application server between applications.

Maintainability

Centralizing the business logic onto application servers makes changes to the software more manageable. Business logic can be changed in an object(s) and installed on the application server, avoiding a recompile of several different applications that use the object. It also allows the object to be used without having to redeploy new versions of the software to the client workstation, as long as the interface to the object—function prototypes—between the client and the server has not changed. By enabling applications to share code—similar to using a stored procedure—we can insure that applications are consistently running the same versions of the software. This provides greater consistency throughout the company, because all client workstations are updated with enhancements and business modifications at the same time that the changes are applied to the server.

Integration

Access to heterogeneous data sources is improved in a three-tier architecture. An application server can access several different data sources using database drivers and legacy data connectivity middleware—like Sybase's Direct Connect products. The data from all these data sources is handled and processed before the client application ever receives it, allowing any pre-processing to occur. This allows applications to use data from several sources transparently, including multi-vendor Relational Database Management System (RDBMS) and existing legacy applications on the mainframe. Funneling database connections through an application server allows connection pooling to be used, and makes database changes easier since the client workstation no longer contains direct access to the database or SQL statements.

Some other advantages include:

- Transactions that span multiple data sources are possible using distributed transaction coordinators and TP monitors like Tuxedo and Microsoft Distributed Transaction Coordinator (DTC).
- Off-loading processing and reducing connections decreases bottlenecks on the database server.
- Client hardware and processing requirements are reduced.
- Increased security is possible by introducing another layer between users and corporate data/resources.

Disadvantages of the n-tier architecture

Despite all the excitement in the industry over distributed computing, it is not a perfect solution. It will not solve all your IT problems, and can actually make them worse. Distributed applications are more complicated than the client/server applications we are used to designing and building. The middleware and dialog between client and server is more advanced than the SQL database-querying typical of most client/server systems. The network infrastructure must provide stable and reliable communications while handling the additional traffic created by a distributed application.

Programming in a distributed environment creates many new challenges that a developer must learn and master. Applications must be written so that moving data across the network is transparent to users, and network traffic must be minimized so as not to tie up network resources. Applications must be able to gracefully handle the increase in the number of potential failure points in an application, including network errors, lost connections, and server bottlenecks. Developers will need to learn how to design and build server components that can take advantage of transaction processing, instance pooling, and other advanced features provided by application server software.

Lastly, distributed computing demands a greater degree of up front analysis and planning in order to get all the advantages out of an application and justify the added complexity. Most applications built today make liberal use of rapid application development (RAD) techniques, and quickly piece together a system in order to meet tight deadlines.

1.5 Summary

In this chapter, we looked at some of the reasons that companies are evaluating multi-tier applications as the next architecture for their large enterprise systems. Businesses look at technology to help solve their problems, increase productivity, and reduce costs. After evaluating the different business architectures that are used today, the n-tier model pro-

vides the type of computing environment that can meet the demands of large-scale Enterprise Applications. The n-tier architecture solves many of the limitations of a two-tier model, including scalability and the need to address changes quickly with less impact on the overall business. It also allows heterogeneous data sources to be combined and for applications to be placed on the Internet and the Web.

This book will focus on using PowerBuilder and the Jaguar CTS component of Sybase Enterprise Application Server (EAServer) to build enterprise distributed applications. Jaguar CTS is poised to help the industry move forward into the new age of component transaction servers and Web OLTP (On-Line Transaction Processing), providing a robust platform for building applications.