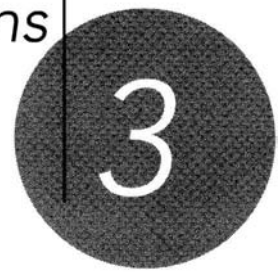
Computers, Networks, and Organizations



The major classes of networked applications discussed in Chapter 2 must be supported by an *infrastructure*—those things not specific to any application, but which support all applications. To conceptualize good applications, it helps immensely to have an appreciation for the capabilities and limitations of the infrastructure that supports them.

ANALOGY: *A new organization in the physical world would be supported by an infrastructure, including highways, postal and package delivery systems, banking, accountants and lawyers, etc. To establish the most successful organization, this infrastructure should be used most effectively, which requires understanding of what each provides and how they work in combination.*

A networked computing infrastructure includes computers (for computation, information processing, and user interface), a network for communication among these computers, and a lot of software. This equipment and software require administrative and operational support, and some of it is leased from and operated by service providers. The goal of this chapter is a deeper appreciation of how computers, networks, and organizations work together to support networked applications, including their development and deployment. The primary concern here is the physical infrastructure, whereas Chapter 4 describes the software infrastructure.

A networked computing infrastructure supports four important capabilities:

- *Communication across distance*: This is supported by the network (even within the same building), which allows computers to communicate data to one another.
- *Communication across time*: This is provided by computer storage. Deferred social and information management applications permit users to participate at times of their choosing, so they need to store information temporarily or permanently. Business applications typically capture and manage massive amounts of data.
- *Computation and logic*: These are defined by software programs. Most applications require internal logic governing how they react to user (or other) inputs, and many require numeric computations.
- *Human-computer interface*: Many applications (particularly social and information management) support users in their activity or job (including interaction, collaboration, information management, etc.). The application governs user interaction—called the *presentation*—by allowing the user to input data (through a keyboard, mouse, microphone, television camera, etc.) and extract information (graphics, video, audio, etc.).

The overall infrastructure and application are illustrated in Figure 3.1 for the simplest of situations: A single desktop computer providing the user interface and a second computer managing data. Any computer connected to a network is—from the network perspective—known as a *host*. The terminology originates from the observation that the computer "hosts" the software (both infrastructure and application). Since this book is concerned with networked computing, all computers of interest are hosts. (The term "host" was also used to connote a mainframe in the earlier days of centralized computing, leading to some terminology confusion.) The desktop computer is called a *client* host (or just client), and the computer managing the data is called the *server* host (or just server) (see Table 2.1 on page 18). The client and server hosts can communicate over the network. The preeminent public network—the Internet—is emphasized in this book. Together, the network and computers form the equipment (sometimes called hardware) por-

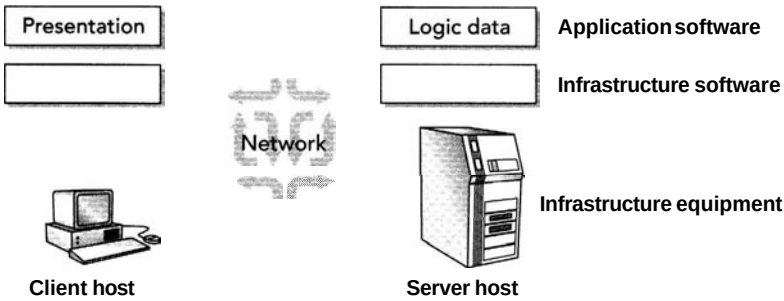


Figure 3.1 General structure of the infrastructure and application in the simplest case.

tion of the infrastructure. The software is equally important, including infrastructure software (Chapter 4) and application software (Chapter 6).

The partitioning of application software across hosts shown in Figure 3.1 is typical for a simple situation. The presentation-everything particular to the interaction with the user, including the human-computer interface—runs on the client host. Other aspects of the application, such as data management, computation, and logic, run in the server host. The portion of application software running on the client host is also called the client, and the portion running on the server host is also called the server. Scenarios involving more hosts are common and are discussed in Section 3.2.2.

3.1 Computing Systems

Any significant activity, such as a conference, business, or school, must be organized to be successful. Likewise, something as complex as a computing system also requires an internal organization to work properly (or to work at all). This organization takes the form of what engineers call a system. A *system* is a composition of subsystems that cooperate to accomplish some higher purpose. A *subsystem* is an element within the system that performs some narrower, well-defined function on behalf of the system and cannot be subdivided and still perform that function.

EXAMPLE: *A telephone system, consisting of switches, transmission systems, and telephones, is a system; these subsystems work together to provide telephone service. The government is a system consisting of executive, legislative, and judicial elements, and a bureaucracy, providing various services to the citizenry.*

In practice, no single person, group, or even organization can deal with the entire system as a whole. Thus, the purpose of decomposing the system into subsystems is to deal with the subsystems as individual units, independently of each other and independently of the system (insofar as practical).

3.1.1 The System Architecture

The *architecture* of a system encompasses its structure and organizing principles. An architecture has the three basic properties described in Table 3.1.

ANALOGY: *The architecture of a building is determined with its intended purpose (business, education, etc.) in mind. The subsystems are the rooms, halls, stairways, electricity, plumbing, etc. How these elements cooperate is determined partly by connections (the hallway has doors into each room, and the stairs connect the floors).*

Table 3.1 The basic elements of a system architecture.

Property	Description	Analogy
Decomposition	A partitioning of the system into individual subsystems that interact to realize the higher purposes of the system.	A government is partitioned into executive, legislative, and judicial branches.
Functionality	The specialized capabilities assigned to each subsystem supporting the overall system purposes.	The legislature makes laws, the executive branch enforces laws, and the judiciary determines the guilt or innocence of accused lawbreakers.
Interaction	How the subsystems communicate and cooperate to support the system purposes.	The executive branch informs the legislature of the need for new laws and brings accused lawbreakers before the judiciary.

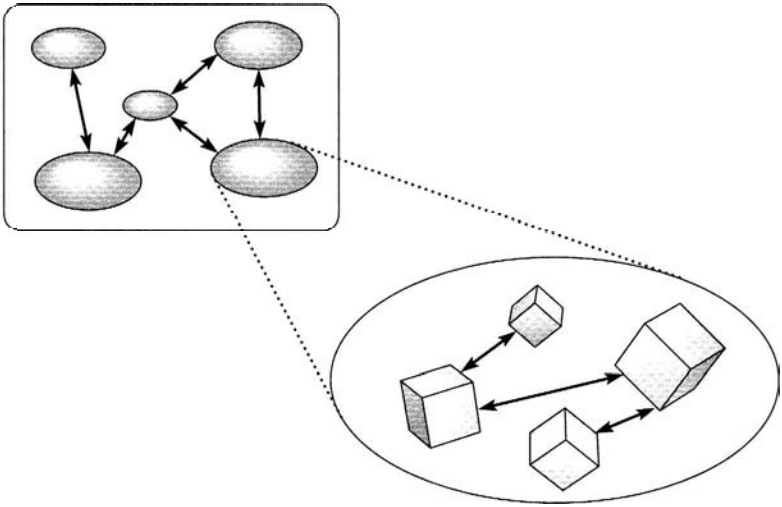


Figure 3.2 Illustration of hierarchical decomposition of a system into subsystems.

EXAMPLE: *A business process reengineering project (see "Business Process Reengineering" on page 58) defines an architecture. It is decomposed into activities, each activity with a specific functionality and a cooperation (flow of information and resources) among them.*

3.1.2 Decomposition of Systems

As illustrated in Figure 3.2, many systems have a *hierarchical* decomposition, meaning that subsystems are themselves systems with an internal decomposition into interacting subsystems.

EXAMPLE: *In the government system, each branch of government is a subsystem internally decomposed into its own subsystems (agencies, departments, etc.).*

A computing system follows this hierarchical model. At the top level, the subsystems include hosts, network, and application and infrastructure software, but each of these subsystems is itself decomposed into internal subsystems. For example—focusing on the equipment—the network includes switches and communication

links, as will be discussed shortly. Each host has internal subsystems—one or more microprocessors (single-chip processing units), memory, and storage—that cooperate to execute a program. Each of these is a subsystem; for example, a microprocessor's functionality (running program instructions) requires an internal decomposition into subsystems (e.g., transistors).

ANALOGY: *Biology displays a similar hierarchical system architecture. An organism is composed of cooperating organs, an organ is composed of cooperating cells, and a cell internally has a membrane, nucleus, etc. Similarly, organizations have a hierarchical decomposition, with divisions composed of departments, departments composed of groups, etc.*

This hierarchical organization is necessary to contain the inherent complexity of a computing system, and it also leads directly to an organization of computer suppliers in which the end-user assembles a system from subsystems, and in turn subsystem suppliers assemble their subsystems from subsystems they purchase from other suppliers (see Chapter 5). Eventually, this decomposition must terminate—an element in a subsystem can no longer be decomposed. In this case, the element is said to be *atomic*, meaning it can't be subdivided.

Subsystems and Components

There is a substantial difference between custom designing a subsystem as part of a system design and purchasing a subsystem as a product from another company. In the former, the system functionality and interaction can be chosen freely to match the precise system requirements, and in the latter the subsystem must be accepted as is and the system designed around it. A subsystem that is purchased as a product from an outside company is called a *component*.

EXAMPLE: *With rare exceptions, a networked application uses commodity computers, peripherals, and networking equipment from outside vendors. From the application perspective, these subsystems are components; the system is designed around them, accepting their functionality and interaction as is.*

The importance of components to both equipment and software suppliers is discussed in Chapters 5 and 6.

3.1.3 Hosts and the Network

For a given application, an architecture of the equipment supporting the application must be designed. The architecture includes subsystems (hosts and the network interconnecting them), an assignment of application functionality to each host, and an understanding of how those subsystems interact. These subsystems supporting networked applications were shown in Figure 2.1 on page 17 and in Figure 3.1 on page 77.

Client/Server Architecture

Two simple architectures supporting networked applications are shown in Figure 3.3 (more complicated scenarios are considered in Section 3.2.2 on page 92). In the clientlserver architecture, hosts come in two flavors: clients and servers. Clients are the hosts that directly support the user and are usually desktop computers (most often PCs or Macintoshes, but sometimes also UNIX workstations). Servers are hosts that do not interface directly to users but provide computation or logic to the networked application, or manage storage. It is typical for many users to access a single server (through their respective clients) remotely across a network, and a server often has higher performance than a client to serve those multiple users. It is also common for a single client to access two or more servers, often to run different applications.

The clientlserver architecture is suitable for information management applications, in which the server stores and manipulates information and the clients pass requests for information from users and display results for them. Many direct- and publication-deferred social applications also use this client/server architecture (Chapter 2 defined the application classification used here). A primary advantage of a network connection is that many servers can be accessed from a single client, either simultaneously or at different times.

EXAMPLE: *The Web (see the sidebar "World Wide Web" on page 37) assigns information management functions to a Web server residing in a server host, and the presentation to a Web browser*

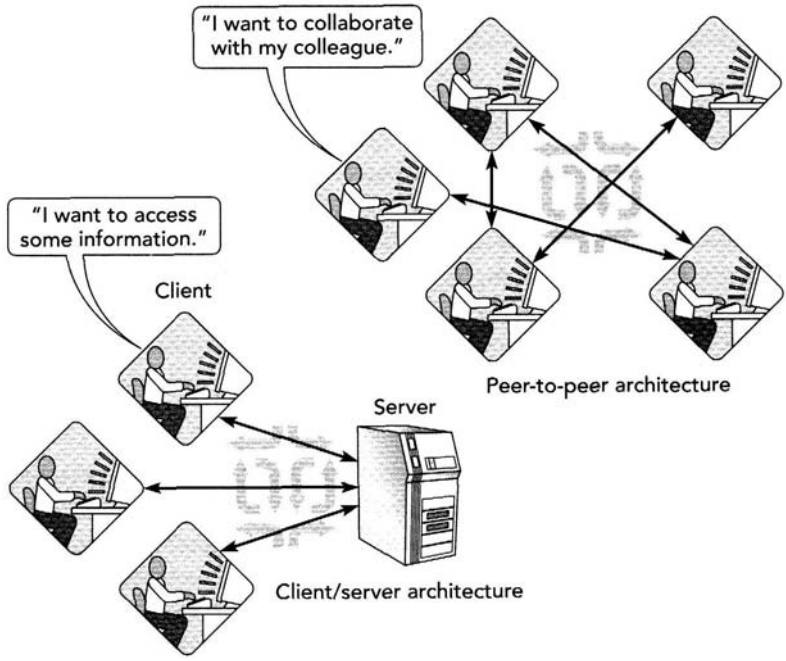


Figure 3.3 Two basic architectures for social and information management applications.

residing in the client host. The user uses a mouse to invoke hyperlinks, and these invocations are communicated over the network to the server host, which returns the requested page for the browser to display. A Web browser can access many Web servers. When the user invokes a hyperlink (by clicking highlighted text), this may cause the browser to shift to an alternative server host.

In summary:

- The primary function of the client is to accept instructions from the user, make requests of the server, and display responses from the server for the user.
- The primary function of the server is to respond to such requests, typically from many clients.

This gives client/server an *asymmetry* of function, where the client makes requests and the server satisfies requests.

EXAMPLE: In an airline reservation system, the travel agent is at a client, and the flight information is stored in a server, so the travel agent—through his client—can answer inquiries about flight times. When a bank customer visits an ATM to get cash, she is the user, the ATM is the client, and the bank's central mainframe is a server. The ATM accepts customers' requests for cash, clears them with the server, and then issues the cash.

A server must be available at all times, waiting for requests from clients. Clients, however, can come and go, since they always initiate the interaction.

ANALOGY: In a law practice, the customers are also called clients. The law office provides a service to multiple clients. A single person might be a client to both a lawyer and an accountant. A law office is always open for any potential client to request legal services, but the interaction is initiated by the client. Different clients may appear at different times.

While the Web illustrates the client/server architecture for information management, it is also natural for direct-*deferred* social applications because there is a storage (communication across time) function naturally assigned to the server.

EXAMPLE: Figure 3.4 illustrates an email application using a client/server architecture. The user can input her messages into an email client program on the client host, and the result is sent to the email server program on a server host. The server stores each message during the time that elapses between the originating user composing and sending a message and the recipient user reading the message, and it also routes the message to the intended recipient (not shown). The recipient user can retrieve and view the message using his own email client. An example of an email client is Eudora, and an example of an email server is a POP server ("POP" stands for "post office protocol").

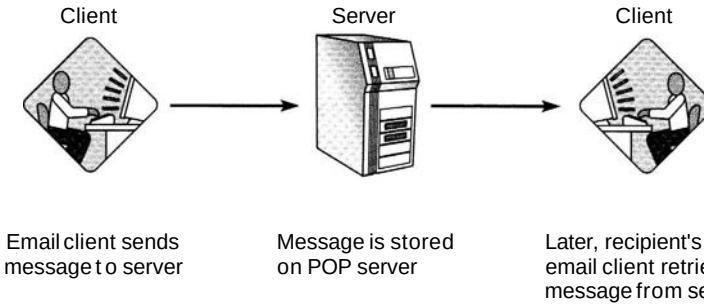


Figure 3.4 Using a client/server architecture to support an email application.

Client/server is also appropriate for direct-*immediate* social applications, when there is application logic that naturally resides on a server, such as consolidation of information for many users.

EXAMPLE: *In a chatroom application illustrated in Figure 3.5, many users can participate at their respective clients. When a user types, each client sends the text to the chatroom server, which aggregates that typing and returns a single presentation to all clients. Each user can see everything typed by everyone.*

Peer-to-Peer Architecture

In the alternative peer-to-peer architecture of Figure 3.3, there is no server but only desktop computers — called *peer hosts* because of their similarity of function — supporting the application and users. This architecture is appropriate for direct-immediate social applications, in which there is no communication across time necessitating storage on a server and no centralized application logic is needed.

EXAMPLE: *Peer-to-peer architecture is natural for audio and video conferencing. Data representing the audio or video is simply communicated over the network directly from one peer to another.*

ANALOGY: *In the public telephone system, when two telephones are used to support a conversation, they are peers. However, when one telephone is used to access bank account information, then the telephone is a client and the bank's computer is a server.*

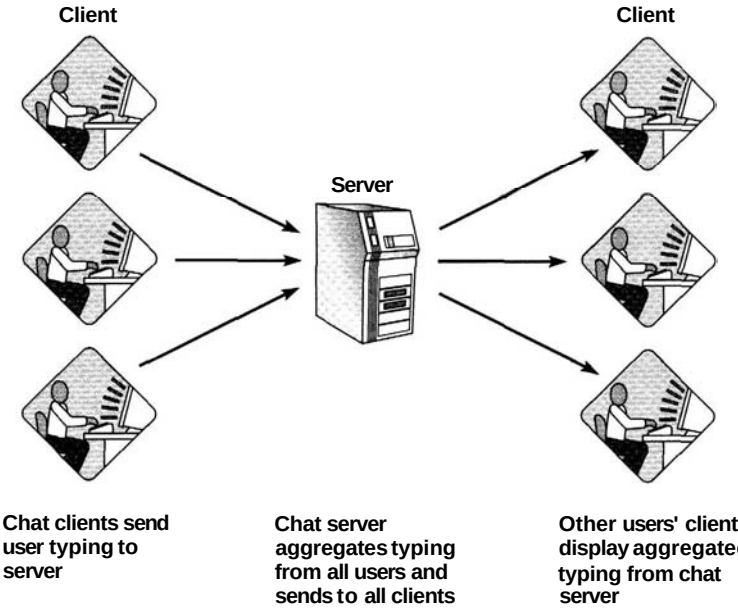


Figure 3.5 A chatroom application using a client/server architecture.

A distinguishing characteristic of the peer-to-peer architecture is the symmetry of function—each peer provides essentially the same functionality—as contrasted to the asymmetry in client/server. Architectures can also mix client/server and peer-to-peer. For example, many direct-immediate social applications have an information management aspect and a multiuser interaction aspect. Their respective desktop computers may serve as both peers (to interact with other peers directly) and as clients (to access the information stored in a server).

EXAMPLE: *In remote conferencing (see Section 2.3.3 on page 27), each user's desktop computer might act as a client to the server that manages the document edits coming from multiple users, while simultaneously acting as a peer for the voice and video aspect.*

A given desktop computer can be a client or a peer; it is the software that defines the host's role in the application. There is no

major distinction in the hardware between clients and servers, except as to the details of configuration (memory, storage, etc.) and performance and the requirement for user interface elements (display, keyboard, etc.) in the client.

The Network

A computer network is to hosts what the public telephone system is to people, allowing hosts to communicate data as necessary to support the application. Any host can communicate with any other host on that same network at any time. In the case of the world's largest network—the global Internet—this means that any host can communicate with tens of millions of other hosts. Each host on the Internet is assigned a *domain name* (for example, info.SIMS.Berkeley.EDU) that other hosts need in order to communicate with it.

Of course, limits must be placed to avoid chaos. One host requires *authorization* to communicate with another; it can't do whatever it pleases with another host without permission. Authorization must be accompanied by *authentication* that confirms the requesting host's identity. A host's owner controls which other hosts can access it and for what purposes. Techniques for authentication and authorization are discussed in Chapter 8.

ANALOGY: *The public streets enable Mary to travel anywhere in her car, including into her own garage. In principle, the streets allow any citizen to drive his car into Mary's garage, but only she is authorized to enter, and the garage door authenticates her by the key she carries.*

EXAMPLE: *A public Web server may authorize any host on the network to access its pages. On the other hand, access can be restricted, for example, to hosts in the same company. Another application running on the same host—one managing sensitive employee records—is only authorized to be accessed by the human resources department.*

The application pieces residing on different hosts may communicate by sending one another *messages* (very much like users in groupware—see Section 2.3.4 on page 28). A message is the small-

est unit of data that makes sense to the sender and recipient; by assumption, a fragment of a message is useless to the application.

EXAMPLE: *In an email application, one user sends an email message to another (note the use of the term "message"). The email message is the smallest unit of information that is meaningful to the sender and recipient. Delivering half the message to the recipient isn't useful. In the Web application, when the user requests a page of information to be displayed, that page is a message communicated from the Web server to the Web browser.*

People use the telephone network by initiating a "call" and holding a "conversation," and typically a given telephone participates in only one conversation at a time. The Internet differs from the telephone network in that one host doesn't have to "call" another host: It can simply send a message to one or to multiple hosts (or receive them from multiple hosts). It is also possible to set up something analogous to a "call" and hold a "conversation"—an ongoing bidirectional exchange of messages—in a service called a *session*. These options are discussed in Chapter 7.

Switching

Architectures are often hierarchical (see Section 3.1.2 on page 79), and the network is no exception. To the communicating hosts, the network structure is not evident, but internally it is decomposed into subsystems—access links, switches, and backbone links. This internal architecture will now be described.

Consider the problem of carrying messages from one host to another. Although the application interprets the message as information, the network considers it data (a collection of bits). To communicate data in a message from one host to another requires physical interconnections called *communication links*.

ANALOGY: *The wire that connects a residence to the telephone's central office is a communication link. It carries the voice (using a telephone) and data (using a modem) from the residence to the telephone's central office and also in the reverse direction.*

The detailed characteristics of communication links are discussed in Chapter 12. For present purposes, a link simply carries messages from one geographic location to another. It is impractical to directly connect each host with every other host by a dedicated communication link. In the global Internet, for example, there are millions of hosts, and having millions of connections to each host is nonsensical. This full interconnection is avoided by requiring each message to traverse multiple communication links and by forwarding that message from one link to another using *switching.*

ANALOGY: *A city avoids having a dedicated street between each pair of garages by adding intersections (analogous to switches). Driving a car between two garages requires traversing multiple streets (analogous to communication links), and at each intersection, choosing one from among the (typically three) alternatives is analogous to switching.*

A simple switched network is shown in Figure 3.6. Each host is connected to a switch by a single dedicated communication link, and switches are in turn interconnected by communication links. A message can be communicated between any pair of hosts with only one link connected to each host. The network has two distinct types of communication links:

- Each host has a single *access link* (analogous to a driveway), which interconnects it with the first switch, called an *access switch*.
- Switches are interconnected by *backbone links*—defined as links not directly connected to hosts. Each switch is typically connected to at most a few other switches. The goal is to avoid—with switches as well as with hosts—an explosion of connected links as the number of hosts on the network grows.

There are also two distinct types of switches:

- An *access switch* forwards messages from an access link to the appropriate backbone link (and vice versa).
- *Backbone switches* forward messages from one backbone link to another.

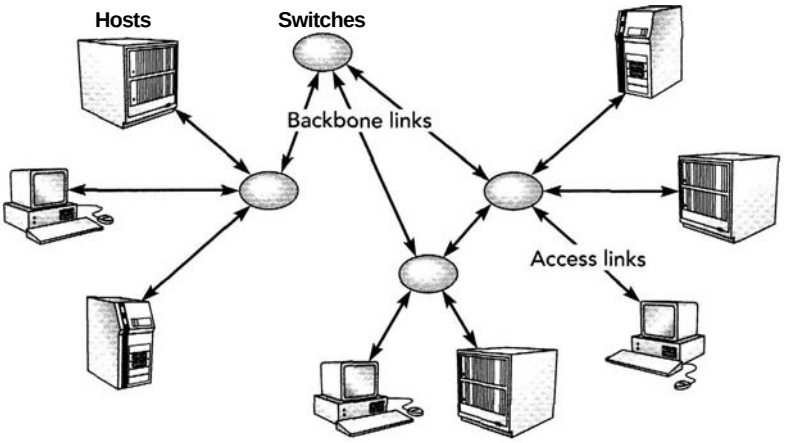


Figure 3.6 Introducing switches allows many hosts to be interconnected, with only one communication link per host.

The collection of backbone switches and backbone links is called the *backbone network*, and the access switches and access links are called the *access network*.

The particular pattern of interconnection of hosts and switches is called the *network topology.* The primary requirement is that from each host to all other hosts there must be at least one feasible *path*—consisting of switches and connecting links—that a message can traverse. As in the simple topology of Figure 3.6, there may be more than one such path, implying that switches must choose from among feasible alternatives. Each switch forwards a message to an appropriate output link, bringing it closer to the destination host, and switches must be coordinated so that the message eventually arrives at the intended recipient host (this function is called *routing* and is discussed in Chapter 11).

In practice, the network forwards not messages, but *packets*. A packet is typically a fragment of a message (a message may be composed of multiple packets), or a packet may be an entire message. For this reason, a computer network is called a *packet network*. The reason for the distinction between packets and messages is related to performance and is discussed in Chapter 10.

3.2 Client/Server Computing

The client/server architecture is popular today, particularly in business computing. However, nothing in the world of computers stands still, and the origins of client/server suggest where it may be heading.

3.2.1 Two-Tier Client/Server

In the centralized computing era (see Section 1.1 on page 2), applications commonly ran on a mainframe computer, and when users needed to interact with the application, this was supported by a terminal that displayed text (and only text). This terminal—called affectionately a "dumb" terminal—had no graphics, windows, menus, nor pointing device, but only a keyboard and text display. The dumb terminal was actually a major advance over *batch processing*, used in the 1960s and earlier, in which users submitted batch jobs using decks of punched cards. Multiple users—each at a dumb terminal—could simultaneously run their own applications on a single computer using *time-sharing*.

ANALOGY: *Dropping your clothes off at a laundry and picking them up later is analogous to batch processing. A laundromat—where you share the available washing machines with others and wait for your clean clothes before leaving—is analogous to time-sharing.*

Dumb terminals, which were directly wired to a computer, allowed a user to interact with an application running on a centralized computer. Later, Ethernet (and other local area network (LAN) technologies) enabled one computer to communicate with other computers. In the networked computing era, the desktop computer supplanted the dumb terminal in support of the user interface and communicated with the mainframe using the LAN.

The arrival of the desktop computer and LAN evolved into the client/server architecture. Solitary applications such as word processing and spreadsheets could execute directly on the desktop computer, but users also wanted centralized capabilities previously supported by time-sharing. These included

- Social applications enabled by multiple users accessing a centralized computer.
- Ability to share information with other users without carrying physical media such as floppy disks. In addition, information stored on a central host allows collaborative authoring and sharing of volatile information.
- Centralized administration of widely used applications, which removes that burden from users.

These capabilities evolved in the client/server architecture without giving up the advantages of desktop computers. The server became the focus for the sharing and backup of information and centralized administration of applications, while users retained the ability to process information in personalized ways or install their own applications on desktop clients.

File System

One of the primary functions of the infrastructure is the storage of data on behalf of users and applications. For this purpose, hosts have storage media (such as disks, tapes, and CD-ROM) and also a file system provided by every computer operating system (see more discussion of operating systems in Chapter 4). The file system is very familiar to personal computer users, who use it to store documents or spreadsheets in files, which are in turn stored in directories or folders. A file is a collection of data managed by some application (such as a word processor or spreadsheet), and the file system stores and retrieves files in a collection of folders. Folders make it easier for the application or user to keep track of related files, by grouping them under a single name.

Both the client and the server have a file system. However, it is particularly important in the server, which is often where most files reside. By centralizing files in the server, systematic backup of data (by making copies on tape) prevents its loss due to equipment failure. Also, this allows data to be shared by multiple users, at their multiple clients. File systems are sometimes distributed, meaning there is a single hierarchical structure of files physically stored in different computers and accessed over the network. A file system

does not know or care about what data is stored in the files—that is managed by applications.

3.2.2 Three-Tier Client/Server

The two-tier client/server architecture has been described, in which the first tier is the client and the second tier is the server. The combination of desktop computers and relatively inexpensive microprocessor-based servers enabled individual departments to operate their own applications (see Section 2.6.1 on page 53).

However, businesses also rely heavily on enterprise databases supporting mission-critical OLTP applications on mainframe computers. The data residing in these databases is useful for departmental applications and decision support (see "Decision Support" on page 62)—and so should be available to servers and desktop computers—but that data and the OLTP applications it supports are also mission critical. For reasons of security and reliability, these mainframes are not suitable as servers for general user access or running other client/server applications (see Chapter 8).

There emerged a three-tier architecture that distinguishes three distinct functions in enterprise applications, as illustrated in Figure 3.7 and listed in Table 3.2. The essence of the architecture is to define two types of more specialized servers—shared data and application logic.

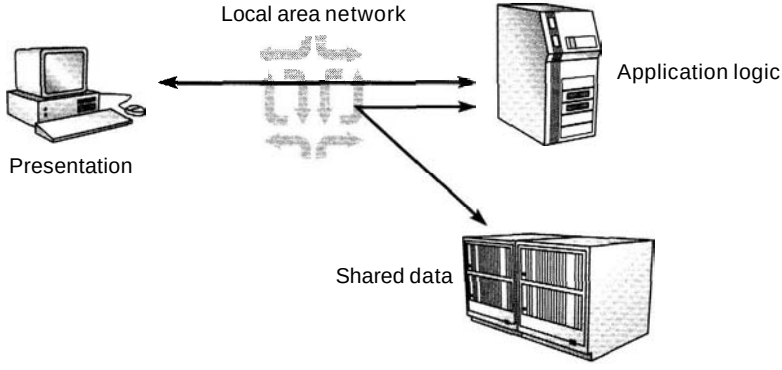


Figure 3.7 Three-tier client server.

Table 3.2 The partitioning of functionality in three-tier client/server computing.

Tier	Description	Analogy
Shared data	Enterprise mission-critical data shared by multiple applications; a server (often, but not necessarily, a mainframe) satisfies data queries from the application logic.	In a restaurant, the refrigerators store the ingredients (analogous to data). The kitchen prepares meals (analogous to information) on request from the waiters.
Application logic	Embodies the unique behavior and functionality of an application; runs in an application server (frequently more than one application server per shared data server).	Operation of the restaurant requires offering the customer (analogous to user) choices, eliciting his order, satisfying requests. The waiter (frequently more than one per kitchen) serves the customer.
Presentation	Displays information to the user by vision and sound; receives user input and requests by voice, keyboard, or mouse.	The table service (dishes, plates, glasses) present the meal to the customer, who uses silverware to eat it. The customer verbally makes requests of the waiter.

The application logic, which mediates between shared data and presentation, manipulates the shared data. It also takes inputs and requests from the presentation, decides what needs to be done, decides what shared data should be accessed or must be updated, manipulates that data appropriately, and responds to the presentation. The shared data answers queries from the application logic, and the application logic determines what data is stored and what queries are needed.

EXAMPLE: A Web-based bookseller (see the sidebar "amazon.com: On-Line Merchant" on page 70) might have mission-critical shared data, including inventory, the status of active orders and back orders, customer information, etc., stored in a database server. A separate application server runs the application logic and Web server that interact with the presentation—the Web browser supporting the customer. The application logic is responsible for determining what is presented on customer screens and what database queries are necessary to generate those screens. It also accepts customer input and updates the database.

There are several reasons to divide things this way:

- The presentation — dedicated to a single user — is natural to assign to the client for better interactivity.
- The application logic supports multiple users and thus is naturally assigned to a server. As the number of users increases, more application servers can be added, all accessing a common shared data server.
- The shared data server may support multiple applications and by dedicating a host can support more application users. It can also have a protected administrative environment for security and reliability (see Chapter 8).
- Keeping application logic out of the client is natural for applications accessed by the citizenry (such as consumer electronic commerce) because it avoids installation of special software.

The three-tier client/server architecture has a many-to-one (or at least few-to-one) relationship at each level: There are many clients per application server and potentially multiple application servers per shared data server.

Shared Data Tier: Database Management

The third tier — shared data — is usually supported by an off-the-shelf server product called a database management system (DBMS) purchased from a specialized vendor (such as Oracle, Sybase, Informix, or IBM). A DBMS assumes a specific structure for the data — one that is appropriate for the application.

EXAMPLE: In business applications, data often takes the form of numbers and character strings (see Section 2.6 on page 52). A character is a single letter in the alphabet, and a character string is a collection of such characters, for example, "San Francisco". Thus, employee or customer information would include character strings for name, address, city, etc., and numbers for telephone number, age, salary, etc. Another example of structured data is a document, which has headings and subheadings, paragraphs, figures, etc.

A database is a file containing interrelated data with a specific predefined structure. Specifically, a data element is a digital encoding

of a character string or number, such as a customer name or telephone number in the previous example. Each customer has the same set of data elements, although the value of the data elements will be different for different customers. The set of data elements corresponding to a customer is called a record, and a database might consist of a collection of records, one for each customer.

EXAMPLE: An application that accesses stock prices might provide the most recent bid and asking price of a stock. It would likely use a DBMS to manage a large set of current and historical stock prices. An application that manages sales records might use a DBMS to manage customer contact and historical sales information.

An alternative to a DBMS is an application that directly manages data in a file with a structure of its choosing. The DBMS offers compelling advantages when large amounts of structure data are managed:

- The DBMS can isolate the application from changes in computer systems. In many situations the data itself (for example, the customer data of a company) needs to survive the replacement of a computer system.
- The DBMS can provide many standard operations on data needed by many applications. Premier among these is the search mentioned in Section 2.4.1 on page 40, which in the case of a database is called a query.

  Often a common data repository must be accessed by multiple users and applications, and thus it is appropriate to separate the database and its management from those applications. That way, applications can be added or removed without affecting the data. Also, the DBMS can take care of many complications arising when different applications try to access the same database at the same time.
- Frequently data is a fundamental asset of an organization, and its safety and integrity are important. The DBMS provides many features that enhance the integrity of data. For example, it can prevent the loss of critical data when a computer fails.

The most common modern databases are relational, which implies that they structure data in two-dimensional forms (with rows and columns) called tables. The table captures the relationship among different types of data. Although this model may seem limiting, it is not, because of the ability to store data in multiple columns and also in multiple tables. The application accesses the data in the database through the query.

EXAMPLE: A relational table storing data about tourism in four cities is shown in Figure 3.8. This table indicates the number of tourists by accommodation and year. Each row corresponds to a set of attributes (year, city, and type of accommodation). A typical query might be "tell me how many tourists stay in resorts in all cities farther from San Francisco than Oakland." The application would recast this into the database query for the relational table in Figure 3.8 as "tell me how many tourists stay in resorts in Oakley and Albany." Since the database is not aware of the interpretation of the data, it cannot translate "farther from San Francisco than Oakland" into "Oakley and Albany," but that would be inferred by the application, perhaps by consulting a separate geographical database.

Because all data is stored in relational tables, queries take standard forms that depend on this structure. Other types of databases are discussed in Chapter 6, and more advanced uses of databases are discussed in the sidebar "Data Warehouses and OLAP."

3.2.3 Thin and Ultrathin Clients

The amount of application functionality in the client is controversial. One extreme — called a fat client — contains a lot of functionality, and the opposite is called a thin client. The three-tier client/server architecture, by keeping application logic out of the client, results in a thin client.

There is increasing concern about the administrative costs associated with desktop computers — not the least of which is time users spend maintaining their own software. The thin client moves toward central administration and avoids separately administering and upgrading many desktop computers. This is one motivation for the

Year	City	Accommodation	Tourists
2002	Oakley	Bed & Breakfast	14
2002	Oakley	Resort	190
2002	Oakland	Bed & Breakfast	340
2002	Oakland	Resort	230
2002	Berkeley	Camping	120000
2002	Berkeley	Bed & Breakfast	3450
2002	Berkeley	Resort	390800
2002	Albany	Camping	8790
2002	Albany	Bed & Breakfast	3240
2003	Oakley	Bed & Breakfast	55
2003	Oakley	Resort	320
2003	Oakland	Bed & Breakfast	280
2003	Oakland	Resort	210
2001	Berkeley	Camping	1115800
2003	Berkeley	Bed & Breakfast	4560
2003	Berkeley	Resort	419000
2003	Albany	Camping	7650
2003	Albany	Bed & Breakfast	6750

Figure 3.8 An example of a relational table.

network computer (NC)-a highly simplified desktop machine (see the sidebar "An Ultrathin Client: The Network Computer (NC)").

3.2.4 The Future of Client/Server

Three-tier client/server architecture is only one of many possibilities for partitioning application functionality among hosts. A generaliza-

Data Warehouses and OLAP

Often an organization will have multiple operational database management systems but need to develop a "big picture" of the overall operation, not only at the present time but also historically (see "Knowledge Management" on page 63). A data warehouse is a very large database accumulated by systematically capturing data from multiple databases. Data warehouses are best understood in the context of business applications.

A data warehouse can store huge amounts of data, but ways to analyze this massive data are needed. On-line analytical processing (OLAP) applications present multidimensional views of data stored in two-dimensional relational tables.

EXAMPLE: A three-dimensional view of the data stored in the two-dimensional relational table in Figure 3.8 is shown in Figure 3.9. While this representation is equivalent in supporting queries, it is often better suited to the presentation of data to a human who wants to understand relationships or trends in the data.

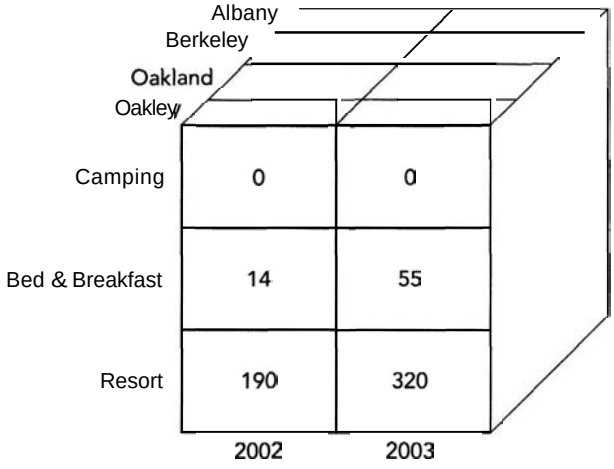


Figure 3.9 An example of an OLAP presentation of the data in Figure **3.8.**

An Ultrathin Client: The Network Computer (NC)

Several approaches to the NC share a desire to simplify desktop computers and move users' data and applications to centralized servers with central administration. The hope is to reduce the life cycle costs of the desktop computer. At its most extreme, the NC is a graphic-display engine. In this ultrathin client, even the presentation is moved to a server, and there is no application-specific software running in the client. Generic graphic-display capabilities can be exploited by all applications.

Many NC proposals include a suite of standard Internet applications — especially a Web browser — resulting in a networked information appliance (an appliance is a terminal with a specific purpose, as discussed in Chapter 5). Another approach is to allow application-specific presentation or logic to reside in the client but to dynamically load it into the client when needed (using mobile code--see Chapter 9).

The thin-client architecture limits a user's ability to install applications, so it is most appropriate when using a limited set of standard applications (as in point-of-sale terminals, airline counters, and simple data entry). One

tion is multitier client/server, which structures the application in four or more tiers. The declining cost of servers based on microprocessor technology encourages approaches that trade more hosts for other benefits such as better performance or reduced administrative costs.

There are two major unmet needs that the current client/server architectures do not address sufficiently:

- Organizational needs change quickly, and it is also important to quickly deploy applications supporting new business opportunities. This implies an ability to build new applications faster than if they were built from scratch. New applications need to be designed, implemented, and deployed by mixing and matching existing components (databases, presentation, etc.). Client/server systems typically couple the tiers too closely to make this feasible, and one role of additional tier(s) is to mediate in a way that is more flexible.
- Enterprise and cross-enterprise applications require data integration across different departments and different enterprises (see Section 2.6.2 on page 56). Client/server computing has

evolved primarily as a vehicle to support departmental applications and by itself does not supply the needed data integration.

Major vendors are addressing these issues, making their own proposals for architectures beyond client/server.

weakness of the desktop computer—security—is where the thin client or NC offers value. When many people may have physical access, restrictions on access to applications and data is easier with thin clients.

3.3 Internet, Intranet, Extranet

No, this isn't a children's rhyme. It's a special set of terminology used when the internet packet-switching technology is applied in several contexts. Following widespread use of local area networks, the internet technology was invented to create a wide area network by interconnecting preexisting LANs. An internet (lowercase) thus designates a "network of networks," including standard ways to interconnect networks as well as equipment and software implementing these standards. The Internet (uppercase) is a specific internet; namely, the large one that is global in extent, accessible to the citizenry, and subject to much attention in the news media.

3.3.1 Intranets

The same internet technologies are used to construct private networks — called intranets — for exclusive use within an enterprise. An intranet and its suite of applications are often used to improve internal communications and collaboration while protecting proprietary information. Other proprietary network technologies and applications can be used for this purpose; however, this is becoming less attractive as the internet technologies provide a suite of existing solutions.

EXAMPLE: General Motors Corporation deployed a satellite-based intranet connecting over 9,000 locations, including all its dealerships. It replaces many volumes of paper-based manuals and daily service bulletins with remote interactive access to the same information. Later, the same intranet will allow customers to customize automobiles and receive delivery in just a few days, or summon help if stranded on the road [Pan98].

Inevitably, intranet users want access to the Internet from their desktop computers, for example, to exchange email outside the

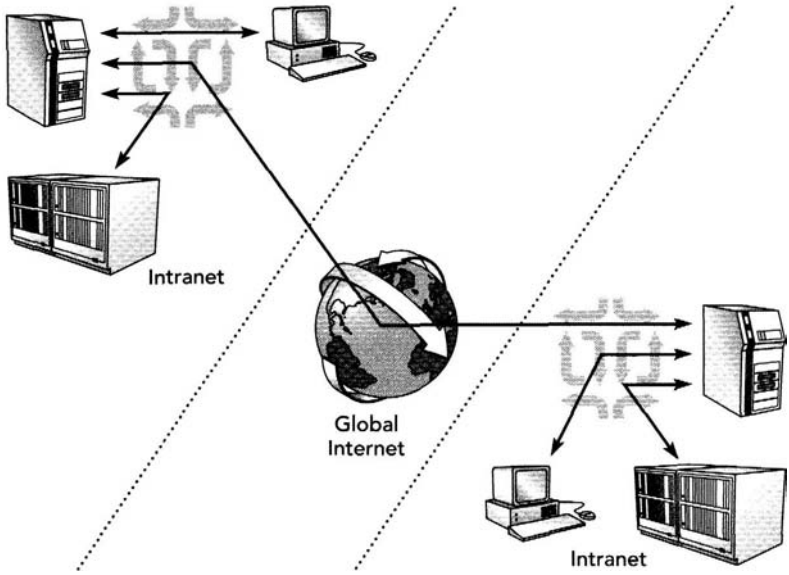


Figure 3.10 An extranet encompasses one or more intranets and the Internet.

organization or access the Web. For this purpose, the intranet is connected to the Internet in a way that does not compromise internal proprietary information, using a firewall (see Chapters 8 and 11). The firewall creates a protected enclave by enforcing restrictions on whether and how internal users can access the Internet and Internet citizenry can access the intranet. Figure 3.10 illustrates a pair of intranets and their connection to the Internet. The dotted lines denote the boundary between intranet and Internet, and all communication links traversing those dotted lines are firewall protected. An organization may also have internal firewalls to enforce restrictions on how employees in one department access resources in other departments.

3.3.2 Extranets

Frequently an organization has two or more locations—each with an intranet—that are geographically separated, and wants to join them in a single intranet. One option is private communication links

among the locations, but this is relatively expensive. Another option is to connect the intranets through the Internet, as shown in Figure 3.10. This compromises the security of any messages traversing the Internet, because it's an unprotected domain outside the organization's administrative control. Confidentiality can be preserved using encryption (see Chapter 8), which can hide message content from anyone without a secret key.

ANALOGY: A bank has vaults to store cash securely but must sometimes transport cash through public streets (analogous to the Internet) from one vault (analogous to an intranet) to another. For this purpose, the bank creates a mobile protected environment with an armored car and armed guards to prevent theft. Access to the armored car requires a key (analogous to an encryption key), possessed only by the trusted guards.

An Internet incorporated into an intranet, as shown in Figure 3.10, is called an extranet. Similarly, when different companies use the Internet to interconnect intranets for cross-enterprise applications such as electronic commerce, this is also called an extranet.

EXAMPLE: The Automotive Network Exchange (ANX) electronic commerce initiative from U.S. automobile companies (see Section 2.6.3 on page 64) is based on a large extranet. Designed by Electronic Data Systems (EDS), the network employs standard internet technologies. It offers numerous security and performance guarantees that ensure it is "'business quality."' Although the ANX network is separate from the public Internet, it does connect many firms' intranets securely [Jan97].

An extranet also allows employees unfettered access to their company's intranet while traveling, as illustrated in Figure 3.11. Individual hosts are allowed intranet access through the Internet. An extranet also allows individual consumers limited and secure access to intranet resources—for example, in selling goods over the Internet—while preventing theft of confidential information (such as credit card numbers). For this purpose, Web browsers support limited extranet capabilities.

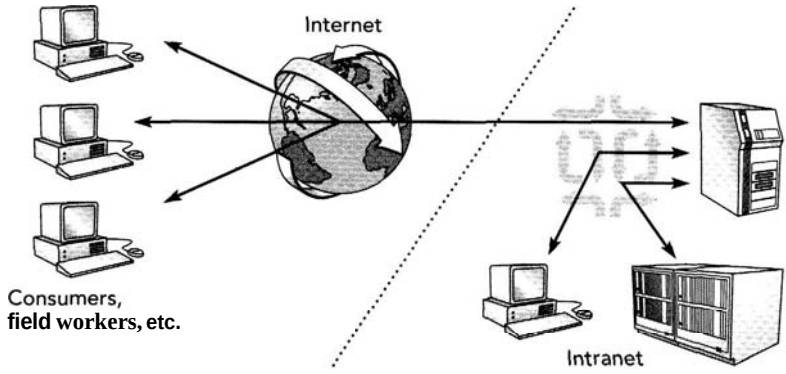


Figure **3.11** An extranet can extend an intranet to field workers and customers.

Nomadic and Mobile Access

As networked computing applications become widely used, network access becomes important not only at home and work but also while traveling. Access while traveling is called nomadic access. Access while actually moving—as in a car, bus, or train—is called mobile access.

EXAMPLE: Today's telephone system provides nomadic access with pay telephones and mobile access with cellular telephones. There are some nascent services providing nomadic and mobile access to the Internet, but for the most part this is an infrastructure waiting to be built.

Nomadic and mobile users want the same suite of applications as fixed users. Radio communications technologies supporting mobile access are more limiting than the fiber-optic technologies supporting fixed access (see Chapter 12). A technical limitation on portable devices is battery technology, which limits the time of operation and processing power.

EXAMPLE: A sales force is naturally nomadic, while sales automation applications enable the representatives to demonstrate on-line catalogs, check inventory and delivery schedules, and place orders.

The latter functions require access to mission-critical business processes, where security and confidentiality are important.

3.3.3 Internet Applications

While the internet is a "network of networks," it also provides a set of popular social and information management applications. When people in an organization speak of having an intranet, they often refer to these standard internet applications as well as using internet networking technologies. These applications include those discussed in Section 2.3 on page 19, such as email, newsgroups, chatrooms, listservers, the Web, etc.

The Web as the Presentation Tier

The Web was conceived as an information management application but has become the basis for a wider class of applications (see the sidebar "World Wide Web" on page 37). The Web browser includes capabilities useful for describing the presentation portion of almost any application, such as formatted text and graphics, and various ways to capture user input, including forms, dialog boxes, radio buttons, etc. Rather than reimplementing these capabilities, applications can simply incorporate a Web browser for presentation. (This is a thin-client architecture—see Section 3.2.3 on page 96.) A major advantage of incorporating the existing browser software is the reduction in development effort, but it also addresses a far more important practical problem. In an uncoordinated environment, such as individual consumer access to an intranet, getting application-specific programs installed on the client is logistically difficult and a deterrent to users.

3.4 Networked Computing and the Organization

In an organizational context, the networking of applications is integrally tied to the organizational structure. The life cycle of the application—its conception, fruition, and maintenance—is also integrally tied to the organizational mission.

3.4.1 Rationale for Networked Computing

Why should applications be networked rather than run on a single computer? Although networked computing has compelling advantages, the rationale is a bit complicated. Social applications, which require that every user has a desktop computer (client or peer) to support the human-computer interface and presentation, are inherently networked. What about other applications? Two important justifications are scalability and administration.

Scalability

Scalability is the ability of an application to support increasing levels of activity and capability. For example, information management or consumer electronic commerce should have no inherent limitation on the number of users or on the amount of information or goods and services for sale. A successful application should not self-limit because of some technological bottleneck. A single computer has limited capability and, as the level of activity increases, eventually exhausts some resource (processing power, storage capacity, etc.). Scalability requires at minimum multiple hosts.

EXAMPLE: A Web server may initially use a single host, but as the number of users accessing the server increases, eventually the communications capability or processing power is exhausted. A second host (then a third, fourth, etc.) can accommodate more users.

ANALOGY: In principle, a company could hire a single employee, but as the company grows, eventually that employee can't keep up and it is necessary to hire more employees. Ways must be found to divide the work among those employees, and doubtless they will need to interact.

Scalability also means the costs of the infrastructure grow no more quickly than key application activity and capability performance attributes (see Chapter 10).

Administration

Administration refers to the ownership and operation of an application; this requires workers to install, operate, and maintain it. Thus,

the cost of the infrastructure — the emphasis of scalability — must be augmented by salary and other costs. Even where a single host may achieve the required performance, this may not be administratively desirable or cost effective: Some administrative justifications for networked computing are listed in Table 3.3.

Table 3.3 Administrative justifications for networked computing.

Issue	Description	Analogy
Administrative specialization	Each department finds it advantageous to administer its own dedicated infrastructure. Today, hardware is inexpensive in relation to salaries, so it makes sense to minimize administrative expenses by proliferating computers with more specialized functionality, if that reduces administrative costs.	An enterprise is broken down into specialized functional units (departments).
Administrative compartmentalization	For cross-enterprise applications, it is not administratively practical to share computing resources. Each company owns and operates its own hosts, permitting them to interact by communicating over the network. This also makes it easier to allocate costs to each company.	Even when two companies need to cooperate, each hires its own workers rather than sharing them.
Locality	It is desirable to store information near where it is generated (administratively and geographically) and generate the presentation near where it is used (drawing upon multiple sources) (see Chapter 6).	A factory is located near transportation hubs and natural resources, but a retail outlet is located near customers.
Sharing	Certain information is needed by many users and applications. It should be maintained and updated in one place but made available to users and applications across the enterprise. Conversely, each user or department should access information maintained by multiple sources.	A firm may have many customers. A firm may have many suppliers.
Security	Information must be reliable, available, and protected against unauthorized access, while selected information is accessible on the outside. A single host can selectively make information or interaction available to other hosts. This is much more secure than sharing a host (see Chapter 8).	A firm wants its own dedicated building. Locating two firms in the same building is less secure.

In the centralized computer era—when computers were large and expensive—it was important to use computers efficiently, for example, by sharing them over many applications. Today, the cost of a substantial microprocessor-based server (including storage and peripherals) is comparable to a few months' loaded salary. Minimizing administrative costs is paramount, even if this means more hosts. Two administrators maintaining the same host—or worse, sharing a single host across two or more administrative units-introduces complications, so it often makes sense to dedicate servers to individual administrative units or even single applications. There can also be compelling reasons to centralize, such as an enterprise database that must be shared among applications. Thus, in practice there is often a mixture of centralization and distribution.

For cross-enterprise applications, the impact of the administrative boundaries on the partitioning of application and data becomes even more constraining. Compartmentalization—allowing access to data and applications by internal users while keeping out external users—is an overriding consideration.

3.4.2 The Application Life Cycle

The conceptualization, development, deployment, and maintenance of a business application must satisfy many stakeholders and (sometimes conflicting) objectives. In a business context, an application may be purchased off the shelf, developed internally, or developed under contract to a professional services firm (the considerations are discussed further in Chapters 5 and 6). The stakeholders whose influences and needs should be taken into account in application development include [BCK98]

- Management within the application development organization: Whether the development is internal or by a professional services company, managers will be concerned about the cost and time to completion.
- End-users: The users of the application and their management will be concerned about cost of acquisition, features, quality, administrative and operational costs, and flexibility to meet

changing requirements (see Section 2.8.4 on page 73). There may also be different criteria applied by management and users.

- Maintenance organization: The staff maintaining the application, whether internal or external, is concerned about the maintenance costs and resilience to adding new features.
- Suppliers and customers: If a business application impacts suppliers and customers, they will be concerned about features, ease of use, etc.

Keeping in mind these stakeholders, the following subsections describe some phases in the application life cycle [Boo94, McC97]. Many large application developments ultimately fail—in the sense that they are never deployed—but avoiding this requires at minimum a well-thought-out and well-executed process. A caution is in order: The process is never as clean and linear as might be implied here; at minimum, these phases overlap, and they may even have to be repeated if the outcome of a later stage is not satisfactory.

Conceptualization

Conceptualization establishes the basic objectives. In a business application, this is an aspect of understanding and refining the business process or commerce, taking into account the organization of workers, the application functionality, and the interaction of workers and application. The conceptualization includes vision (what new function is to be accomplished?) and assumptions (what are fixed points that cannot be changed?). Typically, a business case has to be formulated to convince management that the investment in the development and deployment of the application is warranted. Important objectives include activity and capability performance parameters and how those parameters may evolve over time.

A useful validation tool is low-cost experiments. Using whatever means available, stitching together a prototype to validate basic ideas and assumptions is worthwhile. The human-computer interfaces to the application can be roughly prototyped and tried out on users. The earlier any major problems can be identified and addressed, the more likely the project will be successful. A prototype is also useful for selling the vision to top management.

Analysis

Once it is decided to move ahead, the next phase is the analysis of the application in its organizational context. The analysis is best described by its outcome, which is a description of what the application does in a form that can be reviewed by stakeholders, and in detail sufficient to allow them to make suggestions for change and judge whether development is advisable. It considers the application implementation details only to the extent necessary to validate the analysis; that is, establish that the behavior of the application meets reality.

A useful technique to use during analysis is scenarios. A scenario represents a typical usage of the application and specifies external events and the actions of both the people and the application in response to those events. The scenarios chosen should represent a reasonably complete set, so that no major objectives are neglected.

Analysis does not result in highly detailed specifications. That is best reserved for a process of "iterative refinement" during the design evolution phase. It is important to avoid moving ahead with architecture design before the analysis phase has reached sufficient maturity, or at least the analysis may have to be revisited after greater understanding is developed in the architecture.

A major requirement for modern business applications is flexibility to meet changing needs. The business environment will change as new products and services are introduced or organizations are split or merged. Thus, it is a mistake to overanalyze the near-term needs and ignore or compromise the needed flexibility. In the analysis and architecture phases, a major consideration should be trying to anticipate how application needs may change in the future.

Architecture Design

The application is a system performing the specific functions identified by analysis (see Section 3.1.1 on page 78). The system architecture phase requires the decomposition of the application into hardware and software subsystems, the functionality of those subsystems, and their interaction. Often, the application itself is only one subsystem embedded within a larger social or business system (see Section 1.2 on page 7). Often, this larger system context will

include more than one networked application (see Section 2.6.2 on page 56).

Architecture is critical because it forms the basis of a divide-and-conquer strategy in the subsequent design evolution phase, where individual subsystems are farmed out to different programmers or groups of programmers. One goal is to make subtasks as independent as possible, avoiding excessive coordination. Insufficient attention to or poor design of the architecture is a frequent cause of project failure. At the other extreme, a poorly conceived architecture can constrain the programmers excessively, so the application cannot meet objectives or has insufficient flexibility.

An application architecture is primarily of concern to the implementers; it is not generally visible to or of concern to users. Its primary role is structuring the development process.

Development Evolution

Although not evident from the previous description, the outcome of architecture design is often a software program that is a very incomplete implementation of the system but incorporates major subsystems without details filled in. A major activity in the application development is programming, in which the nascent architectural description is successively refined into a prototype of the production system. Usually it is best to avoid going directly from architecture to production system, but rather to start with the architecture and incrementally add detailed capabilities through successive refinement.

Both the architecture and development evolution phases are described further in Chapter 6.

Testing

Testing an evolving implementation is crucial to uncover shortcomings and flaws **[KFN93].** It does not await a complete production system, but rather is an integral part of the design evolution. With a well-conceived architecture, subsystems can be tested independently, with many problems uncovered and repaired. The merger of interacting subsystems—called integration—requires further testing of their successful interaction.

Once the first prototype of a production system is available, it can be tried out in an environment that approximates the intended usage, called an alpha test. Major problems needing repair are often identified at this stage, so it is "off-line" and users must be cooperative and tolerant. Once the problems are shaken out and repaired, it is typical to move to a second phase of testing called the beta test in an environment as close as possible to intended operational conditions. Ideally, beta testing is performed in a production context but, again, with cooperative and tolerant users. After completing beta testing, the application is ready for deployment.

Deployment

For a business application, the deployment phase includes the establishment of the human organization and the hardware infrastructure (network, hosts, and installation of software on the hosts) and user training. Not infrequently, deployment requires a conversion from some previous operational process and includes extensive planning to avoid major problems and outages, as well as special measures to convert and import relevant data to the new application. Deployment requires advance planning that is every bit as rigorous as design of the software, and total deployment costs frequently exceed the development cost.

Operations and Maintenance

Once successfully deployed, the application moves into the operations and maintenance phase. Operationally, vigilance of human administrators is necessary to take care of problems as they arise and, especially, to maintain security (see Chapter 8). In addition, an application requires continual maintenance by a group of programmers, for two reasons. First, no amount of testing can detect and repair all problems, but inevitably more arise in the operational phase, where conditions not anticipated in testing may arise. Evaluation in the operational phase may uncover fundamental problems, such as failure to meet all functional requirements or user needs. Further, the performance aspects of the application, as it grows to accommodate more users or transactions, are difficult to test fully. Problems must be repaired as they are observed. Second, the operational requirements typically evolve with time, and thus continual development is required to add new capabilities or change func-

tionality to meet changing needs. One reason the architecture design is critical is that a well-conceived architecture is far easier to maintain (both repair and upgrade).

3.5 Open Issue: What Lies beyond Client/Server Computing?

The clientlserver architecture is not an endpoint in the evolution of computing, but a stepping stone. But a stepping stone to what? The origin — and the strength as well — of clientlserver computing is in allowing users on desktop computers to access legacy applications on mainframes or other servers. It was a logical and incremental progression and not a clean break with the past.

Much more radical departures from the past are discussed in Chapter 9, such as distributed object management. These suggest an amorphous architecture, in which applications can search for interesting subsystems from a wide variety of sources and configure themselves more dynamically. There may also be mobile subsystems (called agents) that actively roam the globe looking for interesting goods or information on the user's behalf. These architectures are analogous to the global commercial marketplace, where there are a multiplicity of buyers and sellers continually seeking opportunities and participating in opportunistic transactions that provide personal benefit. This vision will entail a sophisticated infrastructure that doesn't exist today, as the clientlserver architecture is too simplistic to support it.

Further Reading

[OHE96a] is an accessible and excellent source of additional detail on the clientlserver software technologies, including current industry activity, [Wat95] covers the same basic material in much less technical detail, with the needs of managers especially in mind. The process of developing new applications is discussed by [BCK98, McC97, Pre96, Boo94], where the latter served as the basis for the description here. Testing methodologies are covered in [KFN93].