13

The World Wide Web

A Case Study in Interoperability

with Hong-Mei Chen

Flexibility was clearly a key goal. Every specification that was needed to ensure interoperability constrain[s] the Web's implementation. Therefore, there should be as few specifications as possible . . . and the necessary specifications should be made independently. . . . This would let you replace parts of the design while preserving the basic architecture.
— Tim Berners-Lee [Berners-Lee 96*b*]

In the not-too-distant future, anybody who doesn't have their own home page on the World Wide Web will probably qualify for a government subsidy for the home-pageless.
— Scott Adams, creator of *Dilbert*

Possibly the most dramatic example of the workings of the Architecture Business Cycle (ABC) can be found in the way in which the goals, business model, and architecture of the World Wide Web have changed since its introduction in 1990. No one—not the customers, the users, or the architect (Tim Berners-Lee)—could have foreseen the explosive growth and evolution of the Web. In this chapter, we interpret the Web from the point of view of the ABC and observe how changes in its architecture reflect the changing goals and business needs of the various players. We first look at the Web's origins in terms of its original requirements and players and then look at how its server-side architecture has changed as a result of the ABC.

Note: Hong-Mei Chen is an associate professor at the University of Hawaii's Department of Information Technology Management.

13.1 Relationship to the Architecture Business Cycle

The original proposal for the Web came from Tim Berners-Lee, a researcher with the European Laboratory for Particle Physics (CERN), who observed that the several thousand researchers at CERN formed an evolving human "web." People came and went, developed new research associations, lost old ones, shared papers, chatted in the hallways, and so on, and Berners-Lee wanted to support this informal web with a similar web of electronic information. In 1989, he created and circulated throughout CERN a document entitled *Information Management: A Proposal*. By October of 1990 a reformulated version of the project proposal was approved by management, the name World Wide Web was chosen, and development began.

Figure 13.1 shows the elements of the ABC as they applied to the initial proposal approved by CERN management. The system was intended to promote interaction among CERN researchers (the end users) within the constraints of a heterogeneous computing environment. The customer was CERN management, and the developing organization was a lone CERN researcher. The business case made by Berners-Lee was that the proposed system would increase communication among CERN staff. This was a very limited proposal with very limited (and speculative) objectives. There was no way of knowing whether such a system

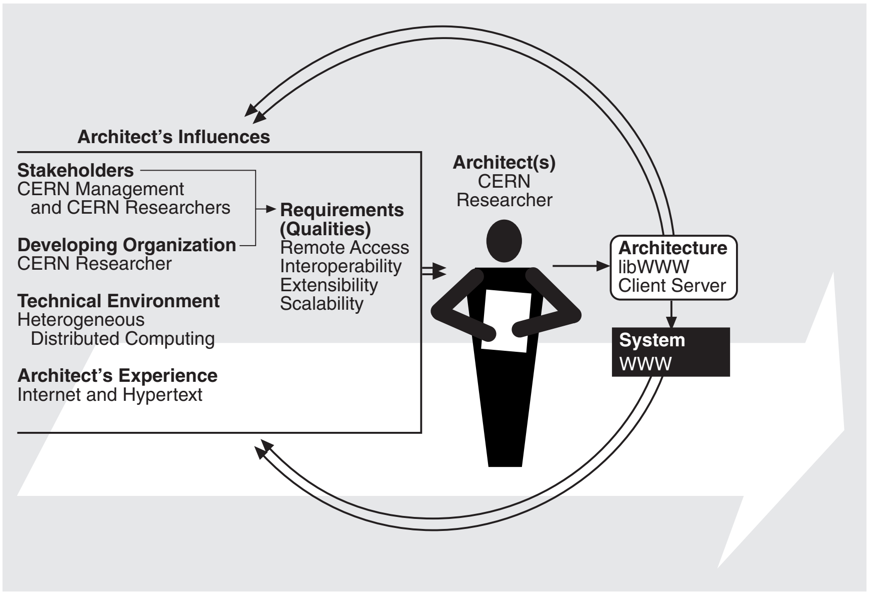

FIGURE 13.1 The original ABC for the Web

would, in fact, increase communication. On the other hand, the investment required by CERN to generate and test the system was also very limited: one researcher's time for a few months.

The technical environment was familiar to those in the research community, for which the Internet had been a mainstay since its introduction in the early 1970s. The net had weak notions of central control (volunteer committees whose responsibilities were to set protocols for communication among different nodes on the Internet and to charter new newsgroups) and an unregulated, "wild-west" style of interaction, primarily through specialized newsgroups.

Hypertext systems had had an even longer history, beginning with the vision of Vannevar Bush in the 1940s. Bush's vision had been explored throughout the 1960s and 1970s and into the 1980s, with hypertext conferences held regularly to bring researchers together. However, Bush's vision had not been achieved on a large scale by the 1980s: The uses of hypertext were primarily limited to small-scale documentation systems. That was to change.

CERN management approved Berners-Lee's proposal in October 1990. By November he had developed the first Web program on the NeXT platform, which meant he clearly had begun working on the implementation before receiving formal management approval. This loose coupling between management approval and researcher activity is quite common in research organizations in which small initial investments are required. By their nature, research organizations tend to generate projects from the bottom up more often than commercial organizations do, because they are dependent on the researchers' originality and creativity and allow far more freedom than is typical in a commercial organization.

The initial implementation of a Web system had many features that are still missing from more recent Web browsers. For example, it allowed users to create links from within the browser, and it allowed authors *and* readers to annotate information. Berners-Lee initially thought that no user would want to write HyperText Markup Language (HTML) or deal with uniform resource locators (URLs). He was wrong. Users have been willing to put up with these inconveniences to have the power of publishing on the Web.

13.2 Requirements and Qualities

The World Wide Web, as conceived and initially implemented at CERN, had several desirable qualities. It was portable, able to interoperate with other types of computers running the same software, and was scalable and extensible. The business goals of promoting interaction and allowing heterogeneous computing led to the quality goals of remote access, interoperability, extensibility, and scalability, which in turn led to libWWW, the original software library that supported Web-based development and a distributed client-server architecture. The realization of

TABLE 13.1 Web Growth Statistics

Date	Number of Web Sites	Percentage of *.com* Sites	Hosts per Web Server
6/93	130	1.5	13,000
12/93	623	4.6	3,475
6/94	2,738	13.5	1,095
12/94	10,022	18.3	451
6/95	23,500	31.3	270
1/96	100,000	50.0	94
6/96	252,000	68.0	41
1/97	646,162	62.6	40
1/98	1,834,710		16.2
1/99	4,062,280		10.6
1/00	9,950,491		7.3
1/01	27,585,719	54.68	4.0

Source: Used with permission of Matthew Gray of the Massachusetts Institute of Technology.

these properties in the original software architecture created an infrastructure that effectively supported the Web's tremendous growth (see Table 13.1). libWWW embodies strict separation of concerns and therefore works on virtually any hardware and readily accepts new protocols, new data formats, and new applications. Because it has no centralized control, the Web appears to be able to grow without bounds.

We will deal with these core requirements, and others, in more detail now, returning to the structure of libWWW later in Section 13.3. There is no explicit requirement for ease of use in the original requirements, and it was not until the development of point-and-click browsers that the Web began its tremendous growth. On the other hand, the requirement for portability and the heterogeneous computing environment led to the introduction of the browser as a separate element, thereby fostering the development of more sophisticated browsers.

THE ORIGINAL REQUIREMENTS

The initial set of requirements for the Web, as established in the original project proposals, were as follows:

- *Remote access across networks.* Any information had to be accessible from any machine on a CERN network.
- *Heterogeneity.* The system could not be limited to run on any specific hardware or software platform.

- *Noncentralization.* In the spirit of a human web and of the Internet, there could not be any single source of data or services. This requirement was in anticipation that the Web would grow. The operation of linking to a document, in particular, had to be decentralized.
- *Access to existing data.* Existing databases had to be accessible.
- *Ability for users to add data.* Users should be able to "publish" their own data on the Web, using the same interface used to read others' data.
- *Private links.* Links and nodes had to be capable of being privately annotated.
- *Bells and whistles.* The only form of data display originally planned was display on a 24×80 character ASCII terminal. Graphics were considered optional.
- *Data analysis.* Users should be able to search across the various databases and look for anomalies, regularities, irregularities, and so on. Berners-Lee gave, as examples, the ability to look for undocumented software and organizations with no people.
- *Live links.* Given that information changes all the time, there should be some way of updating a user's view of it. This could be by simply retrieving the information every time the link is accessed or (in a more sophisticated fashion) by notifying a user of a link whenever the information has changed.

In addition to these requirements, there were a number of nonrequirements identified. For example, copyright enforcement and data security were explicitly mentioned as requirements that the original project would *not* deal with. The Web, as initially conceived, was to be a public medium. Also, the original proposal explicitly noted that users should not have to use any particular markup format.

Other criteria and features that were common in proposals for hypertext systems at the time but that were missing from the Web proposal are as follows:

- Controlling topology
- Defining navigational techniques and user interface requirements, including keeping a visual history
- Having different types of links to express differing relationships among nodes

Although many of the original requirements formed the essence of what the Web is today, several were not realized, were only partially realized, or their impact was dramatically underestimated. For example, data analysis, live links, and private link capabilities are still relatively crude to this day. These requirements have gone largely unfulfilled.

Adaptation and selective postponement of requirements are characteristic of unprecedented systems. Requirements are often lists of desirable characteristics, and in unprecedented systems the tradeoffs required to realize these requirements are often unknown until a design exists. In the process of making the tradeoffs, some requirements become more important and others less so.

The effect of one of the requirements turned out to have been greatly *underestimated*. Namely, the "bells and whistles" of graphics dominate much of today's

Web traffic. Graphics today carry the bulk of the interest and consume the bulk of the Internet traffic generated by the Web. And yet Berners-Lee and CERN management did not concern themselves with graphics in the initial proposal, and the initial Web browser was line oriented. Similarly, the original proposal eschewed any interest in multimedia research for supporting sound and video.

Some nonrequirements, as the ABC has been traversed, have also become requirements. Security, for one, has proven to be a substantial issue, particularly as the Web has become increasingly dominated by commercial traffic. The security issue is large and complex, given the distributed, decentralized form of the Internet. Security is difficult to ensure when protected access to private data cannot be guaranteed—the Web opens a window onto your computer, and some uninvited guests are sure to crawl through.

This has become even more relevant in recent years as e-commerce has begun to drive the structure and direction of the Web and a large number of ad hoc mechanisms have been created to facilitate it. The most obvious is simple encryption of sensitive data, typically via SSL (Secure Sockets Layer), seen in Web browsers as HTTPS (HyperText Transfer Protocol Secure). But this protocol only decreases the likelihood of others snooping on your private data while it is being transmitted over a public network. Other solutions—such as Microsoft's Passport—have you prove that you are who you say you are. (Chapter 4 discussed the various aspects of security, and Chapter 5 presented a set of tactics to achieve it.)

REQUIREMENTS COME AND GO

No one could have foreseen the tremendous growth of the Web, or of the Internet, over the past few years. According to recent statistics, the Web has been doubling in size every three to six months, from about 130 sites in mid-1993 to more than 230,000 sites in mid-1996 to 27 million in early 2001 (see Table 13.1). Figure 13.2 shows how the base communication paths for the Internet blanket the United States. Similarly, the number of Internet hosts—at least as counted by registered Internet Protocol (IP) addresses—grew from 1.3 million in 1993 to 9.5 million in early 1996.

Both the Web and the Internet have grown, but the Web has grown much faster as a whole. This can be seen in the final column of Table 13.1, where we see that the ratio of Internet hosts to Web servers keeps decreasing. This means that an ever-greater proportion of Internet hosts are becoming Web servers.

In addition to its enormous growth, the nature of the Web has changed, as indicated by the third column of Table 13.1. Although its beginnings were in the research community, it is increasingly dominated by commercial traffic (as indicated by Internet hosts whose names end in ".com"). The percentage of .com sites has leveled out at around 55%, but this is due mainly to the rise of other domains, such as .net and .biz, rather than to any decline in commercial activity.

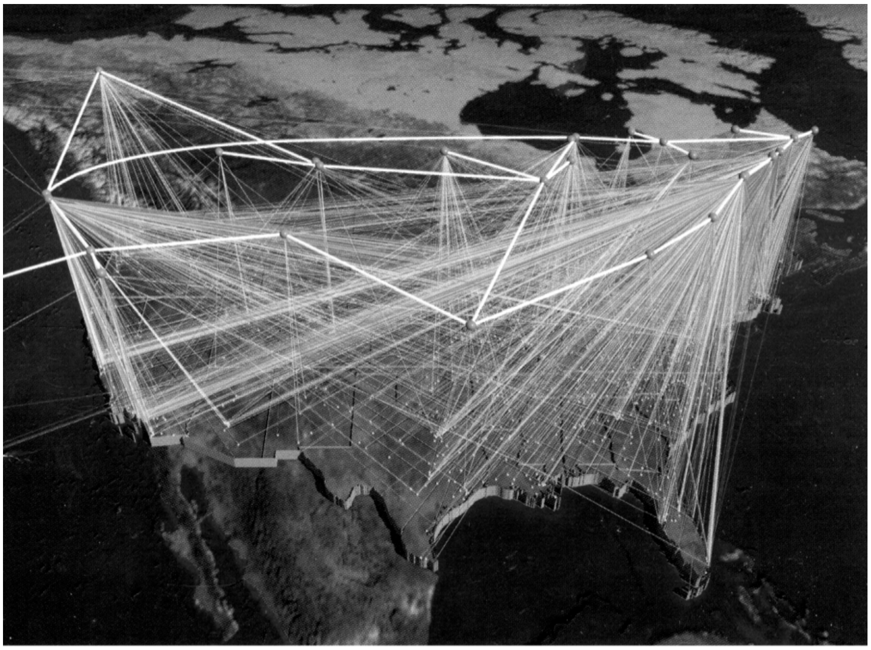

FIGURE 13.2 Internet backbones in the United States. Copyright 1996 by Donna Cox and Robert Patterson; produced at the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign. Used with permission.

The advent of easy, widespread access to the Web has had an interesting side effect. Easy access to graphics in a distributed, largely uncontrolled fashion has spawned the "cyberporn" industry, which has led to a new requirement: that content be labeled and access to content be controllable. The result is the platform for Internet content selection (PICS) specification, an industry-wide set of principles, and vendor implementations of them, that allows the labeling of content and flexible selection criteria. In this way, content producers are not limited in what they provide, but content consumers can tailor what they view or what they permit others to view according to their own tastes and criteria. For example, a parent can prevent a child from viewing movies other than those suitably rated, and an employer can prevent an employee from accessing non-business-related sites during business hours.

To see how far and how fast the Web has diverged from its original concept, imagine that Berners-Lee had proposed a requirement for restriction of content to prevent children from accessing pornography. The management of CERN would have tossed out his proposal without discussion. We return to this point about changing stakeholder concerns when we revisit the ABC for the WWW in Section 13.5.

13.3 Architectural Solution

The basic architectural approach used for the Web, first at CERN and later at the World Wide Web Consortium (W3C), relied on clients and servers and a library (libWWW) that masks all hardware, operating system, and protocol dependencies. Figure 13.3 shows how the content producers and consumers interact through their respective servers and clients. The producer places content that is described in HTML on a server machine. The server communicates with a client using the HyperText Transfer Protocol (HTTP). The software on both the server and the client is based on libWWW, so the details of the protocol and the dependencies on the platforms are masked from it. One of the elements on the client side is a browser that knows how to display HTML so that the content consumer is presented with an understandable image.

We now go into more detail about both the libWWW and the client-server architecture used as the basis for the original Web and that still largely pervades Web-based software. Section 13.4 will discuss how the architecture of the Web and Web-based software have changed in response to the e-commerce revolution.

MEETING THE ORIGINAL REQUIREMENTS: libWWW

As stated earlier, libWWW is a library of software for creating applications that run on either the client or the server. It provides the generic functionality that is shared by most applications: the ability to connect with remote hosts, the ability to understand streams of HTML data, and so forth.

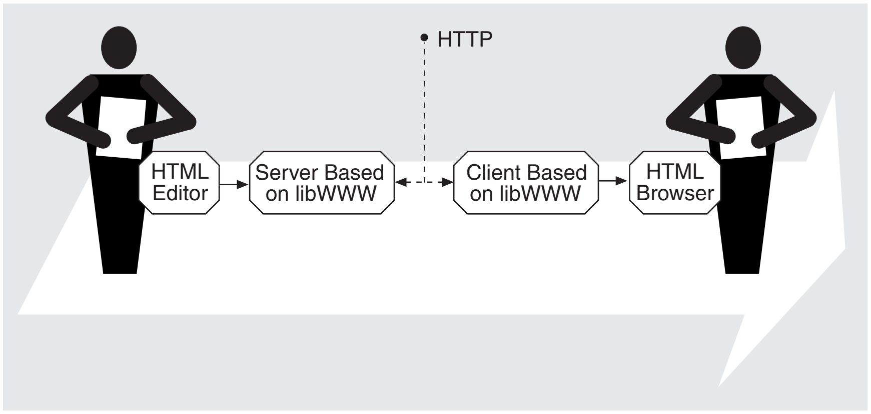

FIGURE 13.3 Content producers and consumers interact through clients and servers

libWWW is a compact, portable library that can be built on to create Web-based applications such as clients, servers, databases, and Web spiders. It is organized into five layers, as shown in Figure 13.4.

The generic utilities provide a portability layer on which the rest of the system rests. This layer includes basic building blocks for the system such as network management, data types such as container classes, and string manipulation utilities. Through the services provided by this layer, all higher levels can be made platform independent, and the task of porting to a new hardware or software platform can be almost entirely contained within the porting of the utilities layer, which needs to be done only once per platform.

The core layer contains the skeletal functionality of a Web application—network access, data management and parsing, logging, and the like. By itself, this layer does nothing. Rather, it provides a standard interface for a Web application to be built upon, with the actual functionality provided by plug-in modules and call-out functions that are registered by an application. *Plug-ins* are registered at runtime and do the actual work of the core layer—sending and manipulating data. They typically support protocols, handle low-level transport, and understand data formats. Plug-ins can be changed dynamically, making it easy to add new functionality or even to change the very nature of the Web application.

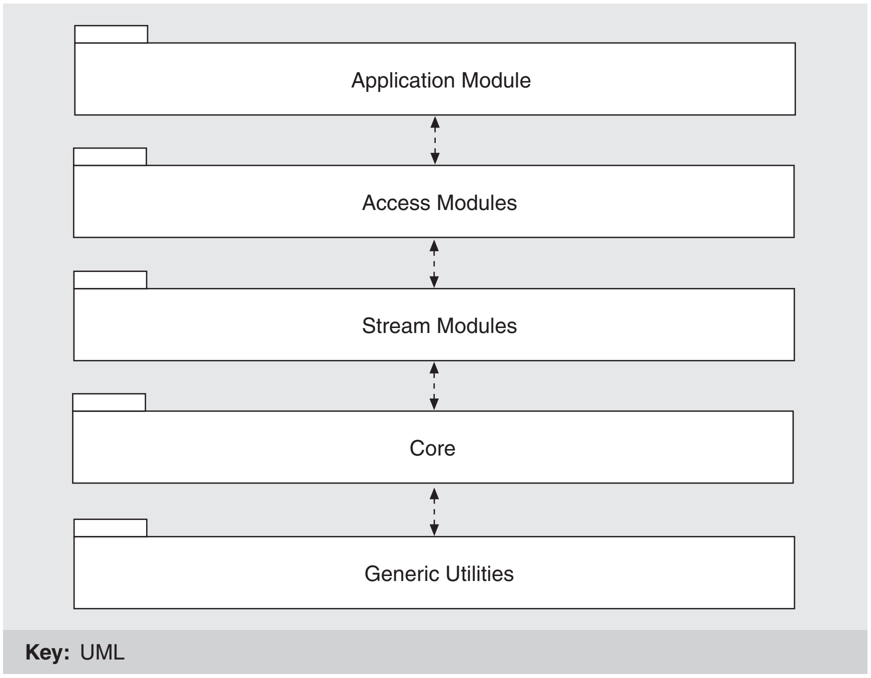

FIGURE 13.4 A layered view of libWWW

Call-out functions provide another way for applications to extend the functionality provided in the core layer. They are arbitrary application-specific functions that can be called before or after requests to protocol modules.

What is the relationship between the generic utilities and the core? The generic utilities provide platform-independent functions, but they can be used to build any networked application. The core layer, on the other hand, provides the abstractions specific to building a Web application.

The stream layer provides the abstraction of a stream of data used by all data transported between the application and the network.

The access layer provides a set of network-protocol-aware modules. The standard set of protocols that libWWW originally supported are HTTP—the underlying protocol of the World Wide Web; Network News Transport Protocol (NNTP)—the protocol for Usenet messages; Wide Area Information Server (WAIS)—a networked information retrieval system; File Transfer Protocol (FTP), TELNET, rlogin, Gopher, local file system, and TN3270. Many of these are becoming rare, but others, such as HTTPS (HTTP Secure) have been added. It is relatively simple to add new protocol modules because they are built upon the abstractions of the lower layers.

The uppermost layer, consisting of the Web application modules, is not an actual application but rather a set of functionality useful for writing applications. It includes modules for common functionality, such as caching, logging, and registering proxy servers (for protocol translation) and gateways (for dealing with security firewalls, for example); history maintenance, and so on.

LESSONS FROM libWWW

As a result of building libWWW and the many applications that rest on it, several lessons have been learned. These lessons have derived in part from the developers' experience in trying to meet the requirements that we listed in Section 13.2—that Web-based tools be heterogeneous, support remote access across networks, be noncentralized, and so forth. However, the requirement that turned out to be the most challenging was supplying unforeseen bells and whistles. That is, allowing the features of Web-based applications to grow has driven many decisions in libWWW and has led to the following lessons:

- *Formalized application programming interfaces (APIs) are required*. These are the interfaces that present the functionality of libWWW to the programs built on top of it. For this reason, APIs should be specified in a language-independent fashion because libWWW is meant to support application development on a wide variety of platforms and in many languages.
- *Functionality and the APIs that present it must be layered*. Different applications will need access to different levels of service abstraction, which are most naturally provided by layers.

- *The library must support a dynamic, open-ended set of features*. All of these features must be replaceable, and it must be possible to make replacements at runtime.
- *Processes built on the software must be thread safe*. Web-based applications must support the ability to perform several functions simultaneously, particularly because operations such as downloading large files over a slow communication link may take a considerable amount of real time. This requires the use of several simultaneous threads of control. Thus, the functionality exposed by the APIs must be safe to use in a threaded environment.

It turns out that libWWW does not support all of these goals as well as it might. For example, the libWWW core makes some assumptions about essential services, so not all features can be dynamically replaced. Furthermore, libWWW is meant to run on many different platforms, and so it can not depend on a single-thread model. Thus, it has implemented pseudothreads, which provide some, but not all, of the required functionality. Finally, most current Web applications do not support dynamic feature configuration; they require a restart before new services can be registered.

AN EARLY CLIENT-SERVER ARCHITECTURE USING libWWW

In Figure 13.5 we show a deployment view of a typical Web client-server that was built using libWWW services. A module decomposition view is also shown for the HTTP client and server components of the deployment view. The figure makes a few points about libWWW. First, not all parts of a client-server are built from it. For example, the user interface is independent. Second, the names of the managers do not directly correspond to the names of the layers: Although the access manager, protocol manager, and stream manager are clearly related to the access and stream layers, the cache manager uses the services of the application layer. The stream managers in the client-server pair manage the low-level communications, thus ensuring transparent communication across a network for the other parts of the system.

The user interface (UI) manager handles the look-and-feel of the client‘s user interface. However, given the open-ended set of resources that a WWW system can handle, another element, the presentation manager, can delegate information display to external programs (viewers) to view resources known by the system but that the UI manager does not directly support. For example, most Web viewers use an external program to view PostScript or .pdf files. This delegation is a compromise between the competing desires of user interface integration (which provides for a consistent look-and-feel and hence better usability) and extensibility.

The UI manager captures a user’s request for information retrieval in the form of a URL and passes the information to the access manager. The access manager determines if the requested URL exists in cache and also interprets history-based

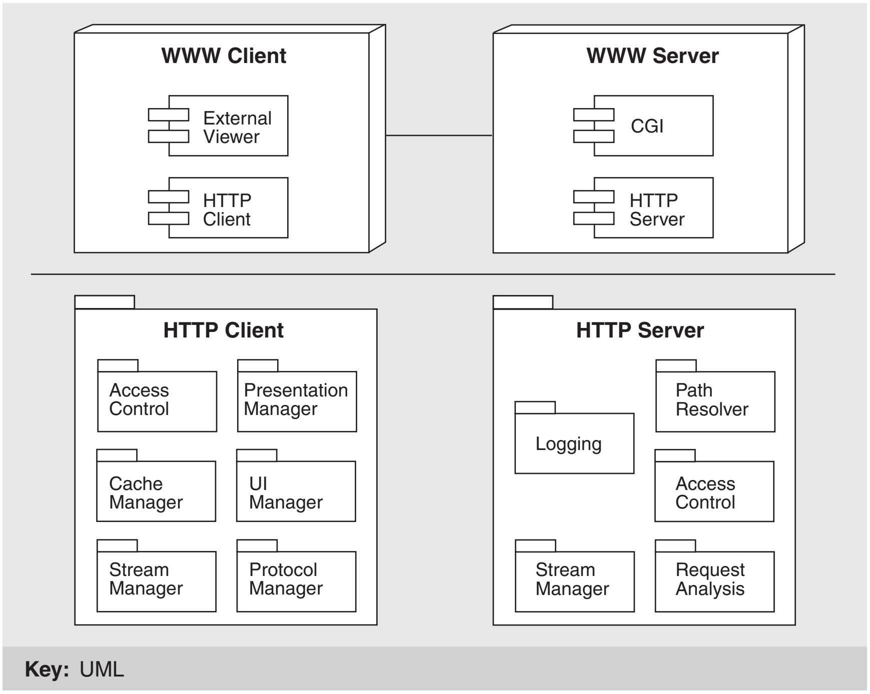

FIGURE 13.5 Deployment view of a Web client-server with a module decomposition view of the HTTP client and server components

navigation (e.g., "back"). If the file is cached, it is retrieved from the cache manager and passed to the presentation manager for display to either the UI or an external viewer. If it is not cached, the protocol manager determines the type of request and invokes the appropriate protocol suite to service it. The client stream manager uses this protocol for communicating the request to the server. Once it receives a response from the server in the form of a document, this information is passed to the presentation manager for appropriate display. The presentation manager consults a static view control configuration file (mimerc, mailcap, etc.) to help it map document types to external viewers.

The HTTP server ensures transparent access to the file system—the source of the documents that the Web exists to transfer. It does this either by handling the access directly (for known resource types) or through a proxy known as common gateway interface (CGI). CGI handles resource types that a native server cannot handle and handles extension of server functionality, as will be discussed next. Before these extensions, the available WWW servers implemented a subset of defined HTTP requests, which allowed for the retrieval of documents, the retrieval of document meta-information, and server-side program execution via CGI.

When a request is received by the server stream manager, its type is determined and the path of the URL is resolved via the path resolver. The HTTP server consults an access list to determine if the requesting client is authorized for access. It might initiate a password authentication session with the client to permit access to secured data. Assuming authentication, it accesses the file system (which is outside the server boundary) and writes the requested information to the output stream. If a program is to be executed, a process is made available (either new or polled) through CGI and the program is executed, with the output written by the server stream manager back to the client.

In either case, CGI is one of the primary means by which servers provide extensibility, which is one of the most important requirements driving the evolution of Web software. CGI became such an important aspect of Web-based applications that we now discuss this topic at greater length.

COMMON GATEWAY INTERFACE

Most information returned by a server is static, changing only when modified on its home file system. CGI scripts, on the other hand, allow dynamic, request-specific information to be returned. CGI has historically been used to augment server functionality: for input of information, for searches, for clickable images. The most common use of CGI, however, is to create *virtual documents*—documents that are dynamically synthesized in response to a user request. For example, when a user looks for something on the Internet, the search engine creates a reply to the user's search request; a CGI script creates a new HTML document from the reply and returns it to the user.

CGI scripts show the flexibility of early architectures which were based on libWWW. In Figure 13.5, CGI is shown as external to the HTTP server. CGI scripts are written in a variety of languages, some of which are compiled (C, C++, Fortran) and some of which are interpreted (perl, VisualBasic, AppleScript, etc.). These scripts allow a developer to extend a server's functionality arbitrarily and, in particular, to produce information that the server will return to the user.

However, because scripts may contain any functionality written in C, perl, and so on, they represent an enormous security hole for the system on which they are installed. For example, a script (which runs as a process separate from the server) might be "tricked" into executing an arbitrary command on the host system on behalf of a remote user. For this reason, server-side scripts such as CGI have led to a new requirement for increased security. The use of HTTPS to address this requirement will be described in the next section.

Probably the most important additional feature that CGI brought to the Web architecture is that it allows users to "put" information into the Web, in contrast to the "get" operation that servers normally provide. Although the requirement to put in information was listed in the original World Wide Web project requirements, it has not been fully achieved. CGI allows users to put information only in application-specific ways, such as adding it to a database by filling out a form.

Table 13.2 How the WWW Achieved Its Initial Quality Goals

Goal	How Achieved	Tactics Used
Remote Access	Build Web on top of Internet	Adherence to defined protocols
Interoperability	Use libWWW to mask platform details	Abstract common services Hide information
Extensibility of Software	Isolate protocol and data type extensions in libWWW; allow for plug-in components (applets and servlets)	Abstract common services Hide information Replace components Configuration files
Extensibility of Data	Make each data item independent except for references it controls	Limit possible options
Scalability	Use client-server architecture and keep references to other data local to referring data location	Introduce concurrency Reduce computational overhead

CGI solved many problems inherent in the original design of libWWW—principally because it provided much needed server extensibility to handle arbitrary resources, allowed users to put data in limited ways—it also had several substantial shortcomings. The security issue was one; another was portability. CGI scripts written in VisualBasic, AppleScript, and C Shell work on Windows, Macintosh, and UNIX, respectively. These scripts cannot be (easily) moved from one platform to another.

ACHIEVING INITIAL QUALITY GOALS

Table 13.2 describes how the Web achieved its initial quality goals of remote access, interoperability, extensibility, and scalability.

13.4 Another Cycle through the ABC: The Evolution of Web-Based E-Commerce Architectures

The incredible success of the Web has resulted in unprecedented interest from business and hence unprecedented pressure on the architecture, via the ABC. Business requirements have begun to dominate Web architecture. Business-to-business and business-to-consumer Web sites have fueled most of the innovation in Web-based software.

The original conception of the Web was as a web of documents, in keeping with its hypertext roots. E-commerce, however, views the Web as a web of data, and these different views have led to some tensions. For example, "pushing" data to a user is difficult; the most common technique for updating data is to reload it at specified periods rather than to rely on the change of data to force a screen update. Another is the back button on a browser, which in certain circumstances may result in stale data being displayed on a screen.

The new requirements of e-commerce are stringent and quite different from the original requirements presented in Section 13.2:

- *High performance*. A popular Web site will typically have tens of millions of "hits" per day, and users expect low latency from it. Customers will not tolerate the site simply refusing their requests.
- *High availability.* E-commerce sites are expected to be available "24/7." They never close, so must have minimal downtime—perhaps a few minutes per year.
- *Scalability.* As Web sites grow in popularity, their processing capacity must be able to similarly grow, to both expand the amount of data they can manage and maintain acceptable levels of customer service.
- *Security.* Users must be assured that any sensitive information they send across the Web is secure from snooping. Operators of Web sites must be assured that their system is secure from attack (stealing or modifying data, rendering data unusable by flooding it with requests, crashing it, etc.).
- *Modifiability.* E-commerce Web sites change frequently, in many cases daily, and so their content must be very simple to change.

The architectural solution to these requirements is more about *system* architecture than simply software architecture. The components that populate the system come from the commercial marketplace: Web servers and Web clients of course, but also databases, security servers, application servers, proxy servers, transaction servers, and so forth.

A typical reference architecture for a modern e-commerce system is shown in Figure 13.6. The browser/user interaction function is usually fulfilled by a Web browser (but it could be a kiosk, a legacy system with a Web connection, or some other Web-enabled device). The business rules and applications function is typically fulfilled by application servers and transaction servers. The data services layer is typically fulfilled by a modern database, although connections to legacy systems and legacy databases are also quite common. This scheme is often

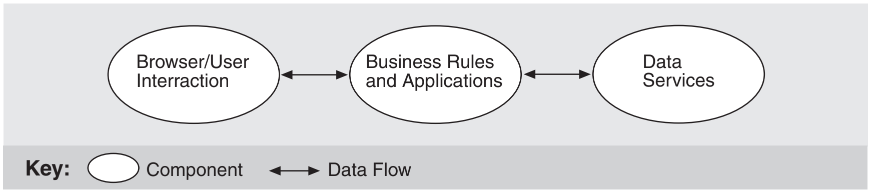

FIGURE 13.6 An e-commerce reference architecture

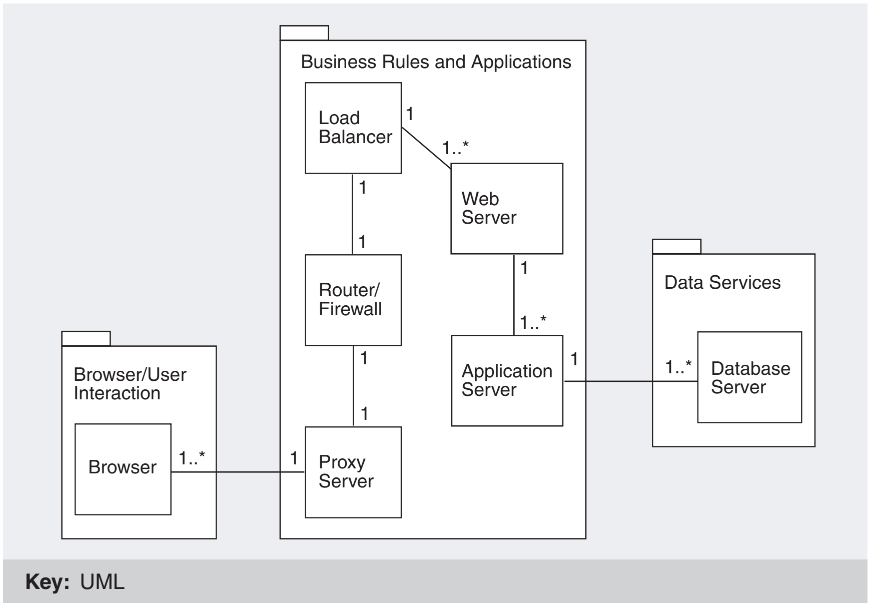

FIGURE 13.7 A typical e-commerce system

referred to as an *n*-tier architecture (here, *n* = 3). A *tier* is a partitioning of functionality that may be allocated to a separate physical machine.

A typical implementation of an e-commerce system architecture consists of a number of tiers, each consisting of a coherent grouping of software (typically customized commercial components) and hardware. Such a configuration is given in Figure 13.7, which shows how software is allocated to hardware.

The figure is annotated with the functional elements from Figure 13.6 to reinforce the notion that a single function in the reference architecture may map to multiple tiers in a typical e-commerce architecture. The two parts of Figure 13.5 occur here as elementary components: the Web browsers (clients) and the Web servers, respectively, reflecting the evolution toward component-based systems in which the internal component structure is less relevant.

We will now discuss each of the elements in Figure 13.7, along with the qualities that each helps to achieve.

WEB BROWSERS FOR MODIFIABILITY

An end user typically initiates a request for information by interacting with a Web browser. Modern Web browsers support user interface modifiability in a wide variety of ways, the most obvious of which has not changed from the inception of the Web: The user interface that the browser supports is not hardwired but it is specified via HTML. At least, it used to be. Nowadays there are many other technologies for creating sophisticated user interfaces. XML, Flash, ActiveX, and Java applets are just a few of the methods by which the standard palette of Web interactors (graphics and hot spots) are widened to provide fully programmable interactive interfaces via browsers.

HTTPS FOR SECURITY

Once the user has submitted a request, it must be transmitted to a target Web site. This transmission may be via HTTP or, for sensitive information such as credit card or identification numbers, HTTPS (HTTP Secure). HTTPS uses Netscape's Secure Sockets Layer as a subprotocol underneath HTTP. It uses a different port (443 instead of the standard port 80 that HTTP uses) to request TCP/IP services in an encrypted form. SSL uses a 128-bit public/private key pair to encrypt the data, and this level of encryption is considered adequate for the exchange of small amounts of commercial information in short transactions.

PROXY SERVERS FOR PERFORMANCE

Requests from individual browsers may first arrive at a proxy server, which exists to improve the performance of the Web-based system. These servers cache frequently accessed Web pages so that users may retrieve them without having to access the Web site. (Caches carry out the tactic of "multiple copies.") They are typically located close to the users, often on the same network, so they save a tremendous amount of both communication and computation resources. Proxy servers are also used by companies that want to restrict their employees' access to certain Web sites. In this case the proxy server is acting somewhat like a firewall.

ROUTERS AND FIREWALLS FOR SECURITY

Requests from the browser (or proxy server) then arrive at a router, located on the e-commerce provider's network, that may include a firewall for security. (Alternately the router may pass HTTP requests on to a separate firewall.) The router may implement network address translation (NAT), which translates an externally visible IP address into an internal IP address. The IP address for any return traffic from the Web server is translated so that it appears to have originated from the externally visible site, not from the internal IP address. NAT is one of the techniques used in load balancing, as we will discuss shortly.

The purpose of the firewall is to prevent unauthorized information flows or accesses from the outside world, an example of the "limit access" tactic. There are several types of firewall, the most common being *packet filters* and *application proxies*. Packet filters examine the TCP and IP headers of each incoming packet and, if any bad behavior is detected (such as an attempt to connect via an unauthorized port or to send nonconforming file types), the packet is rejected. Packet filter firewalls are appropriate for Web-based communication because they examine each packet in isolation—there is no attempt to maintain a history of previous communication.

Application proxy firewalls are, as their name suggests, application specific. They typically understand application protocols and hence can filter traffic based on known patterns of behavior. An application proxy may, for example, refuse an HTTP response unless an HTTP request was recently sent to that site. These firewalls can be much slower than packet filter firewalls because they rely on keeping a certain amount of history information on hand and their processing tends to be more complex.

LOAD BALANCING FOR PERFORMANCE, SCALABILITY, AND AVAILABILITY

A load-balancing component is an integral part of any important e-commerce Web site, because it supports performance, scalability, and availability. The job of the load balancer is to distribute the "load"—incoming HTTP and HTTPS requests—among a pool of computers running Web servers. (Recall from Chapter 5 that load balancing follows from the tactic of "introducing physical concurrency.") The load balancer may simply (and transparently) redirect the request to another computer, or it may respond to the Web client and instruct it to redirect the request to a different server. While this redirection is transparent to the end user, it results in an additional roundtrip of communication.

In choosing which computer to redirect the traffic to, the load balancer may select in a round-robin fashion, or its choices may be based on known processing or load characteristics of each computer to which it is connected. Because the load balancer is acting as a proxy for the pool of computers, we can add to that pool without changing any external interface. In this way the load balancer supports performance scalability, known as horizontal scaling (adding more instances of a given resource).

In addition, the load balancer may monitor the liveness of each of its computers and, if one of them goes down, simply redirect traffic to the others in the pool. In this way it supports availability.

WEB SERVERS FOR PERFORMANCE

Next the HTTP or HTTPS request reaches the Web server. Early Web servers, such as those described in Figure 13.5, were typically single threaded. Modern versions are multithreaded, utilizing a pool of threads, each of which can be dispatched to handle an incoming request. A multithreaded server is less susceptible to bottlenecks (and hence long latency) when a number of long-running HTTP or

HTTPS requests (such as credit card validations) arrive because other threads in the pool are still available to serve incoming requests. This is the performance tactic of “introduce concurrency.”

Vertical scaling (adding more powerful instances of a given resource) can be accomplished by replacing existing servers with more powerful machines that will run more threads simultaneously.

Upon analyzing the request, the Web server will send it to an application server that can respond, typically using the services of a database to do so.

Chapter 16 will discuss Enterprise JavaBeans, a modern implementation approach for Web servers.

APPLICATION SERVERS FOR MODIFIABILITY, PERFORMANCE, AND SCALABILITY

From the Web server the request is forwarded to an application server. “Application server” is a broad (some would say ill-defined) term for a class of applications that run in the “middle” of the *n*-tier architecture—business rules and applications. These servers implement business logic and connectivity, which dictate how clients and servers interact. The trend toward application servers has allowed significant portions of functionality to be moved from old-style “fat” clients into the middle tier. Also, they have allowed databases to concentrate on the storage, retrieval, and analysis of data without worrying about precisely how that data will be used.

Application servers at the low end typically offer an integrated development environment (IDE) and runtime server. IDEs support a programming model, such as COM (or, more recently, .NET), CORBA, or J2EE (discussed in Chapter 16). Many application servers also support a set of commonly used services for quickly creating business and e-commerce applications, such as billing, inventory, work flow, and customer relations management. At the upper end in terms of cost, complexity, and functionality are transaction processing and transaction monitoring. Transaction monitors and processors interact with databases and manage tasks like distributed transactions (including combining data from multiple sources), queuing, transaction integrity, and workload balancing (much like the load balancer mentioned earlier).

DATABASES FOR PERFORMANCE, SCALABILITY, AND AVAILABILITY

Finally, the request for service arrives at the database, where it is converted into an instruction to add, modify, or retrieve information. Modern database architectures share many of the qualities of the entire e-commerce system presented in Figure 13.7. They frequently use internal replication for performance, scalability, and high availability. They may use caching for faster performance.

TABLE 13.3 How the Web e-Commerce Architecture Achieves Its Quality Goals

Goal	How Achieved	Tactics
High Performance	Load balancing, network address translation, proxy servers	Introduce concurrency; increase resources; multiple copies
High Availability	Redundant processors, networks, databases, and software; load balancing	Active redundancy; transactions; introduce concurrency
Scalability	Allow for horizontal and vertical scaling; load balancing	Abstract common services; adherence to defined protocols; introduce concurrency
Security	Firewalls; public/private key encryption across public networks	Limit access; integrity; limit exposure
Modifiability	Separation of browser functionality, database design, and business logic into distinct tiers	Abstract common services; semantic coherence; intermediary; interface stability

13.5 Achieving Quality Goals

Together the elements we have described allow the Web-based e-commerce system to achieve its stringent quality goals of security, high availability, modifiability, scalability, and high performance. How they do this is shown in Table 13.3.

13.6 The Architecture Business Cycle Today

If we look at the current state of the Web after several cycles through the ABC, we see a number of phenomena.

- Several types of organizations provide the technical environment. They can be divided into service providers and content providers. Service providers produce the software that makes the Web—browsers, servers, databases, application servers, security technologies (such as firewalls), transaction servers, networks, and routers. Content providers produce the data for the Web. There is heavy competition in all of these areas.
- A number of open-source projects, aside from the W3C, have come to prominence in the development of the Web, particularly the Apache project.
- CERN has had no special role in the evolution of the Web.
- Web-enabled languages, particularly Java, are changing the way functionality is developed and delivered over the Web. (See Chapter 18 for an example of how Web-based applications are built using Enterprise JavaBeans.)

- The emergence of the Web as a distributed development environment has given rise to several new organizations and products. For example, UDDI (Universal Description, Discovery, and Integration) provides distributed Web-based registries of Web services. These services can be used as building blocks for distributed Web-based applications.

Figure 13.8 shows the ABC for the Web today.

The customers are the software server and browser providers and the service and content providers. The end users are the people of the world. The architect's role is provided by the W3C and other consortia such as UDDI, the Apache project, and several influential companies—Sun, Microsoft, and AOL/Netscape. The remainder of the ABC is the same except that the technical environment now includes the Web itself, which adds an upward compatibility requirement to the qualities.

We discussed the return cycle of the ABC in Section 1.1. The existence of a system creates new business opportunities for both the developing organization and its customers. In the World Wide Web case, the developing organization, CERN, decided that nuclear research, not Web activity, was its main business, so the business opportunities created by the return loop of the ABC were filled by other organizations.

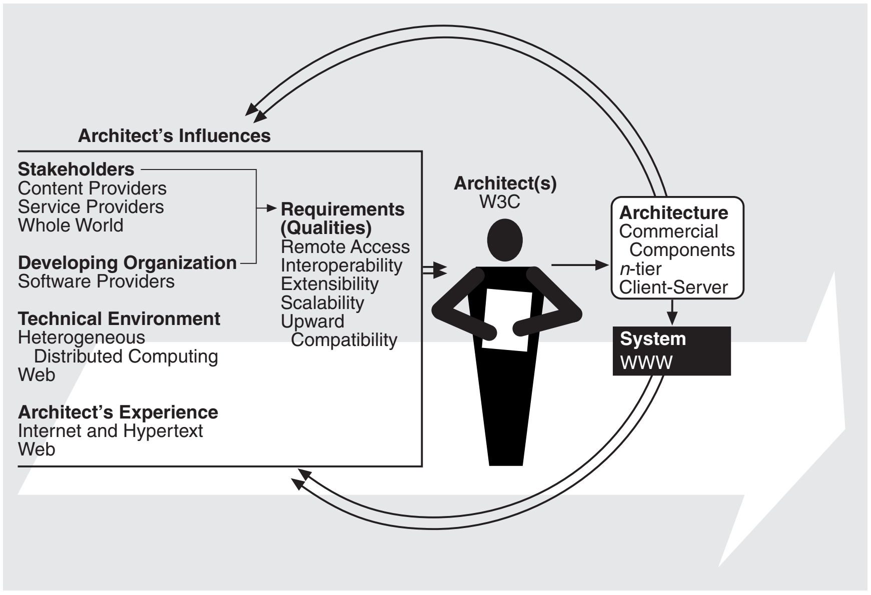

FIGURE 13.8 The current ABC for the Web

13.7 Summary

The Web has been so successful because of the manner in which the desired qualities were realized in its architectural structures, and in how these structures have been reinvented in the face of dramatic new requirements. The success of the Web has meant that the ABC has been traversed multiple times in just a few years, with each traversal creating new business opportunities, new requirements, and new technical challenges.

How the Web Has Changed the Business World: A Look at Amazon.com

When Amazon.com opened its virtual doors in 1995, it was already an order of magnitude bigger than the average bookstore, carrying more than 1 million titles. It was not like "brick-and-mortar" bookstores in other ways as well, and these differences all stemmed from the fact that Amazon was an e-business, delivering the message and products via the Web.

Being an e-store meant that Amazon could change the world (at least, the business world). For example, it meant that Amazon could sell books created by small, independent writers and publishers since it did not bear the costs of publishing. It meant that it could change the ways in which people bought books, with online user reviews, synopses, personal recommendation services, e-mail notifications of new books by a user's favorite author, and so forth. It also meant that Amazon could keep its prices low since it avoided most of the costs of traditional retail operations by outsourcing the majority of its operations.

A shopper at Amazon.com receives customized, personalized service such as suggestions for books similar to those the customer has browsed or purchased. Amazon can do this only because of its enormous IT infrastructure and data-mining ability.

Rather than a simple purchaser and reseller of books, Amazon is a "middleman" and an information broker. It has succeeded in creating a loyal and ever-growing community of sellers and buyers, not just of books. Amazon is a hub, collecting a percentage of every sale made and receiving commissions on referrals to other Web sites.

Ultimately Amazon's IT infrastructure has little to do with books. Amazon realized this early on and was able to transform itself into a retailer of toys, cell phones, drugs, cameras, software, car parts, pet supplies—virtually anything that could be sold to the public and shipped around the world. None of this would have been possible without the Web's infrastructure.

Today, Amazon.com claims to be the world's largest online store serving customers in more than 220 countries. It has five international Web sites and approximately 20 million registered customers (almost the population of Canada!). Repeat business is 70%, which is unheard of in retailing. At the time of this writing, Amazon hasn't made an annual profit, but it expects to be profitable in 2003 and beyond.

Although it is far from the only one, Amazon is perhaps the most dramatic example of how the Web has changed the world (at least the world of retailing).

— RK

13.8 For Further Reading

Readers interested in discovering more about hypertext should see [Bush 45] and the special issue of the CACM devoted to hypertext [CACM 88].

Information on the Web's history and growth can be found primarily on the Web. We used [Berners-Lee 96*a*], [Gray (*http://www.mit.edu/people/mkgray/net*)], and [Zakon (*http://www.zakon.com/robert/internet/timeline*)].

Much of the detail about libWWW comes from the W3C Reference Library at *http://www.w3.org/pub/WWW/Library.*

For a good general discussion of network security issues and cryptography, including all aspects of Web security, see [Stallings 99]. A good discussion of performance issues in e-commerce systems may be found in [Menasce 00].

The architectural style used in Web-based applications is treated in [Fielding 96]. A comparison of modern Web server patterns may be found in [Hassan 00], from which we adapted the client-server architecture shown in Figure 13.5.

Netcraft's May 2001 survey of Web server usage can be found at *http://www.netcraft.com/survey/.*

13.9 Discussion Questions

1. We have identified a number of qualities that made the WWW successful: interoperability, portability, remote access, extensibility, and scalability. Which of these do you think contributed most substantially to the Web's success? If any of these qualities had been sacrificed, would the Web still have been successful? What tradeoffs did these quality goals entail in the architecture of applications based upon libWWW?

2. The Web did not have performance as one of its early quality goals, which is unusual for a successful system. Why do you think the system was successful anyway? What, if anything, does this say about the future of computing?

3. What patterns and tactics can you discern in the portions of the architecture shown in Figures 13.4, 13.5, and 13.7?