Internet Protocols and Applications

This chapter introduces some of the protocols and applications that have made the task of using the Internet both easier and very popular over the past couple of years. In fact, World Wide Web traffic, which is mostly HTTP, has outrun FTP as using the most bandwidth of all protocols across the Internet. Modern computer operating systems provide Web browser applications by default, some even provide Web servers, thus making it ever easier for end users and businesses to explore and exploit the vast capabilities of worldwide networked computing.

The World Wide Web (WWW)

The World Wide Web is a global hypertext system that was initially developed in 1989 by Tim Berners Lee at the European Laboratory for Particle Physics, CERN in Switzerland to facilitate an easy way of sharing and editing research documents among a geographically dispersed group of scientists.

In 1993 the Web started to grow rapidly which was mainly due to the National Center for Supercomputing Applications (NCSA) developing a Web browser program called Mosaic, an X Windows-based application. This application provided the first graphical user interface to the Web and made browsing more convenient.

Today there are Web browsers and servers available for nearly all platforms. You can get them either from an FTP site for free or buy a licensed copy. The rapid growth in popularity of the Web is due to the flexible way people can navigate through worldwide resources in the Internet and retrieve them.

The number of Web servers is also increasing rapidly and the traffic over port 80, which is the well-known port for HTTP Web servers, on the NSF backbone has had a phenomenal rate of growth too. The NSFNET, however, was converted back to a private research network in 1995; therefore comprehensive statistics of backbone traffic are not as easily available anymore, if they are at all.

Web Browsers

Generally, a browser is referred to as an application that provides access to a Web server. Depending on the implementation, browser capabilities and hence structures may vary. A Web browser, at a minimum, consists of an HTML interpreter and HTTP client which is used to access HTML Web pages. Besides this basic requirement, many browsers also support FTP, NNTP, e-mail (POP and SMTP clients), among other features, with an easy-to-manage graphical interface. Figure 251 on page 438 illustrates a basic Web browser structure.

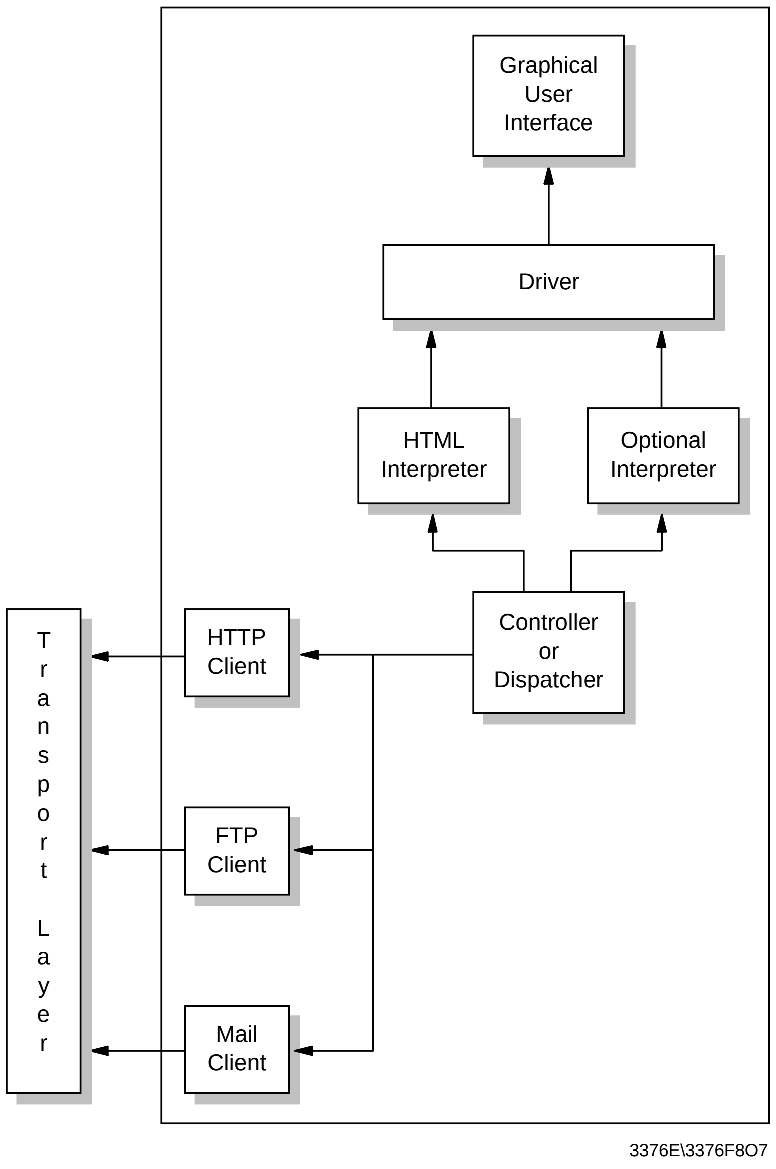

Figure 251. Structure of a Web Browser

As with many other Internet facilities, the Web uses a client/server processing model. The Web browser is the client component. Examples of Web browsers include Mosaic, Netscape Navigator, Microsoft Internet Explorer, or Sun HotJava browser. Web browsers are responsible for formatting and displaying information, interacting with the user, and invoking external functions, such as telnet, or external viewers for data types that Web browsers do not directly support. Web browsers have become the "universal client" for the GUI workstation environment, in much the same way that the ability to emulate popular terminals such as the DEC VT100 or IBM 3270 allows connectivity and access to character-based applications on a wide variety of computers. Web browsers are widely available for all popular GUI workstation platforms and are inexpensive.

Web Servers

Web servers are responsible for servicing requests for information from Web browsers. The information can be a file retrieved from the server's local disk, or it can be generated by a program called by the server to perform a specific application function.

There are a number of public-domain Web servers available for a variety of platforms including most UNIX variants, as well as personal computer environments such as OS/2 Warp and Windows NT. Some well-known public domain servers are CERN, NCSA httpd, and Apache servers.

IBM has released the Domino Go Webserver, a scalable, high-performance Web server that is available on OS/390 as Version 5 (including WebSphere Application Server as described in 8.6.4.2, “IBM Web Application Servers” on page 458). Domino Go Webserver V4.6.2.5 is available on many workstation platforms (AIX, Solaris, HP-UX, OS/2 Warp, Windows NT, and Windows 95). Both versions provide state-of-the-art security, site indexing capabilities, advanced server statistics reporting, PICS support, and relational database connectivity with Net.Data (see 8.6.4.1, “IBM Web Connectors” on page 457). Domino Go Webserver is the successor to IBM's well-known Internet Connection Secure Server (ICSS), and it is a Java-enabled Web server that supports the development and running of servlets (see 8.5, “Java” on page 449).

Web Server Application Technologies

As mentioned earlier, Web server can serve static (mere HTML pages) or dynamic (generated by a program upon invocation) content. This section discusses some commonly used technologies used to provide dynamic content and to facilitate interaction between a Web server and an application server that is not directly accessible to a client (Web browser).

Common Gateway Interface (CGI)

The Common Gateway Interface (CGI) is a means of allowing a Web server to execute a program that is provided by the Web server administrator, rather than retrieving a file. CGI programs allow a Web server to generate a dynamic response, usually based on the client's input. A number of popular Web servers support the CGI, and a variety of programming languages can be used to develop programs that interface with CGI. Unless using PERL, however, CGI programs are not easily portable across platforms.

Server-Specific APIs

Some Web servers offer specific APIs that allow developers to create programs that can be invoked for special purposes upon certain events. Those APIs are usually quite powerful but offer no portability across Web server platforms. The most popular server-specific APIs are Netscape Server API (NSAPI) and Microsoft Internet Information Server API (ISAPI).

Servlets

Based on Java, this technology allows the invocation of a Java program in the server's memory. This method is usually very portable across platforms and incurs little processing overhead. See 8.5, “Java” on page 449 for more details.

Server-Side Includes (SSI)

This is a technology that a Java-enabled Web server (meaning, a Web server with a servlet engine) can use to convert a section of an HTML file into an alternative dynamic portion each time the document is sent to the client's browser. This dynamic portion invokes an appropriate servlet and passes to it the parameters it needs. The replacement is performed at the server and it is completely transparent to the client. Pages that use this technology have the extension .shtml instead of .html (or .htm).

Java Server Pages (JSP)

This is an easy-to-use solution for generating HTML pages with dynamic content. A JSP file contains combinations of HTML tags, NCSA tags (special tags that were the first method of implementing server-side includes), <SERVLET> tags, and JSP syntax. JSP files have the extension .jsp. One of the many advantages of JSP is that it enables programmers to effectively separate the HTML coding from the business logic in Web pages. JSP can be used to access reusable components, such as servlets, JavaBeans, and Java-based Web applications. JSP also supports embedding inline Java code within Web pages.

Hypertext Transfer Protocol (HTTP)

HTTP 1.1 is a *proposed standard protocol*. Its status is *elective*. It is described in RFC 2068. The older HTTP 1.0 is an *informational protocol* and described in RFC 1945.

The hypertext transfer protocol is a protocol designed to allow the transfer of hypertext markup language (HTML) documents (please see 8.3, “Hypertext Markup Language (HTML)” on page 448). HTML is a tag language used to create hypertext documents. Hypertext documents include links to other documents that contain additional information about the highlighted term or subject. Such documents may contain other elements apart from text, such as graphic images, audio and video clips, Java applets, and even virtual reality worlds (which are described in VRML, a scripting language for that kind of elements). See 8.3, “Hypertext Markup Language (HTML)” on page 448 for more information on HTML.

Overview of HTTP

HTTP is based on request-response activity. A client, running an application called a browser, establishes a connection with a server and sends a request to the server in the form of a request method. The server responds with a status line, including the message's protocol version and a success or error code, followed by a message containing server information, entity information and possible body content.

An HTTP transaction is divided into four steps:

1. The browser opens a connection.
2. The browser sends a request to the server.
3. The server sends a response to the browser.
4. The connection is closed.

On the Internet, HTTP communication generally takes place over TCP connections. The default port is TCP 80, but other ports can be used. This does not preclude HTTP from being implemented on top of any other protocol on the Internet, or on other networks. HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used.

Except for experimental applications, current practice requires that the connection be established by the client prior to each request and closed by the server after sending the response. Both clients and servers should be aware that either party may close the connection prematurely, due to user action, automated timeout, or program failure, and should handle such closing in a predictable fashion. In any case, the closing of the connection by either or both parties always terminates the current request, regardless of its status.

In simple terms, HTTP is a stateless protocol because it keeps no track of the connections. To load a page including two graphics for example, a graphic-enabled browser will open three TCP connections: one for the page, and two for the graphics. Most browsers, however, are able to handle several of these connections simultaneously.

This behavior can be rather resource-intensive if one page consists of a lot of elements as quite a number of Web pages nowadays do. HTTP 1.1, as defined in RFC 2068, alleviates this problem to the extent that one TCP connection will be established per type of element on a page, and all elements of that kind will be transferred over the same connection respectively.

However, if a request depends on the information exchanged during a previous connection, then this information has to be kept outside the protocol. One way of tracking such persistent information is the use of cookies. A cookie is a set of information that is exchanged between a client Web browser and a Web server during an HTTP transaction. The maximum size of a cookie is 4 KB. All these pieces of information, or cookies, are then stored in one single file and placed in the directory of the Web browser. If cookies are disabled, that file is automatically deleted. A cookie can be retrieved and checked by the server at any subsequent connection. Because cookies are regarded as a potential privacy exposure, a Web browser should allow the user to decide whether to accept cookies or not and from which servers. While cookies merely serve the purpose of keeping some kind of state for HTTP connections, secure client and server authentication is provided by the Secure Sockets Layer (SSL) which is described in 5.7, “Secure Sockets Layer (SSL)” on page 331.

HTTP Operation

In most cases, the HTTP communication is initiated by the user agent requesting a resource on the origin server. In simplest case, the connection is established via a single connection between the user agent and the origin server as shown in Figure 252.

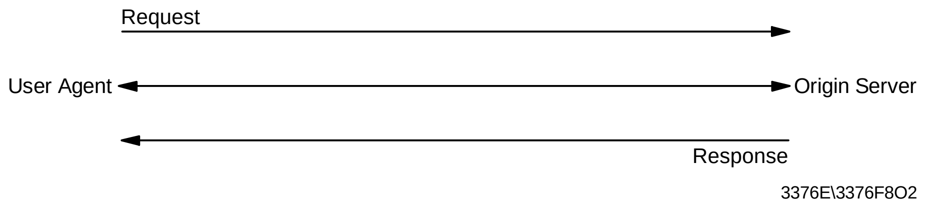

Figure 252. HTTP - Single Client/Server Connection

In some cases, there is no direct connection between the user agent and the origin server. There are one or more intermediaries between the user agent and origin server such as a proxy, gateway, or tunnel. Requests and responses are evaluated by the intermediaries and forwarded to the destination or another intermediary in the request-response chain as shown in Figure 253.

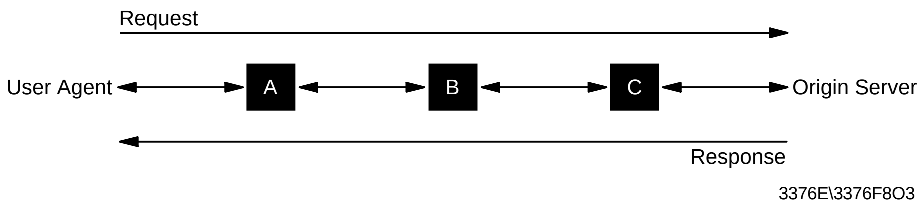

Figure 253. HTTP - Client/Server Connection with Intermediaries Between

As described in 5.3.4, “Application Level Gateway (Proxy)” on page 284, a proxy can handle the content of the data and therefore modify the data accordingly. When a request comes to a proxy, it rewrites all or part of the message and forwards the message to the next destination. A gateway receives the message and sends the message to the underlying protocols with an appropriate format. A tunnel does not deal with the content of the message, therefore it simply forwards the message as it is.

Proxies and gateways in general can handle caching of HTTP messages. This can dramatically reduce the response time and IP traffic on the network. Since tunnels cannot understand the message content they cannot store cached data of HTTP messages. In the previous figure (Figure 253), if one of the intermediaries (A,B and C) employs an internal cache for HTTP messages, the user agent can get response from the intermediary if it is previously cached from the origin server in the response chain. The following figure (Figure 254) illustrates that A has a cached copy of an earlier response from the origin server in the response chain. Hence, if the server response for the request is not already cached in the user agent's internal cache, it can directly be obtained from A.

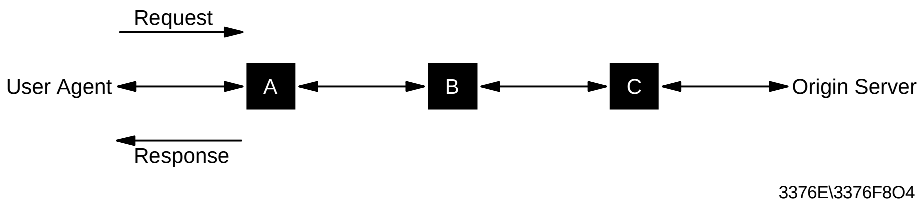

Figure 254. HTTP - Cached Server Response

Caching is not applicable to all server responses. Caching behavior can be modified by special requests to determine which server responses can or cannot be cached. For this purpose, server responses can be marked as non-cachable, public or private (cannot be cached in a public cache). Cache behavior and cachable responses are discussed in 8.2.2.11, “HTTP Caching” on page 447.

Protocol Parameters

Some of the HTTP protocol parameters are given below. Please refer to RFC 2068 for the full list and details:

HTTP Version

HTTP uses a <major>.<minor> numbering scheme to indicate the versions of the protocol. The further connection will be performed according to the protocol versioning policy. The <major> number is incremented when there are significant changes in protocol such as changing a message format. The <minor> number is incremented when the changes does not effect the message format.

The version of HTTP messages is sent by an HTTP-Version field in the first line of the message. The HTTP-Version field is in the following format. (Please refer to RFC 822 for augmented Backus-Naur Form.)

```
HTTP-Version   =   "HTTP" "/" 1*DIGIT "." 1*DIGIT
```

Uniform Resource Identifiers (URI)

Uniform Resource Identifiers are generally refer to as WWW addresses and combination of Uniform Resource Locators (URL) and Uniform Resource Names (URN). In fact, URIs are strings that indicate the location and name of the source on the server. Please see RFC 2068 and 2396 for more detail about the URI and URL syntax.

HTTP URL

The HTTP URL scheme allows you to locate network resources via HTTP protocol. It is based on the URI Generic Syntax and described in RFC 2396. The general syntax of a URL scheme is shown below:

```
HTTP_URL  =  "http" "//" host  [ ":" port ]  [ abs_path ]
```

The port number is optional. If it is not specified, the default value is 80.

HTTP Message

HTTP messages consist of the following fields:

Message Types

A HTTP message can be either a client request or a server response. The following string indicates the HTTP message type:

```
HTTP-message   =   Request | Response
```

Message Headers

HTTP message header field can be one of the following:

- General header
- Request header
- Response header
- Entity header

Message Body

Message body can be referred to as entity body if there is no transfer coding has been applied. Message body simply carries the entity body of the relevant request or response.

Message Length

Message length indicates the length of the message body if it is included. The message length is determined according to the criteria that is described in RFC 2068 in detail.

General Header Fields

General header fields can apply both request and response messages. Currently defined general header field options are as follows:

- Cache-Control
- Connection
- Date
- Pragma
- Transfer-Encoding
- Upgrade
- Via

Request

A request messages from a client to a server includes the method to be applied to the resource, the identifier of the source, and the protocol version in use. A request message field is as follows:

```
Request  =  Request-Line
            *( general-header | request-header | entity-header )
              CRLF
              [ message-body ]
```

Please refer to RFC 2068 for detailed information.

Response

HTTP server returns a response after evaluating the client request. A response message field is as follows:

```
Request  =  Request-Line
            *( general-header | request-header | entity-header )
              CRLF
              [ message-body ]
```

Please refer to RFC 2068 for detailed information.

Entity

Either the client or server might send Entity in the request message or response message unless otherwise indicated. Entity consists of the following:

- Entity header fields
- Entity body

Connections

A significant difference between HTTP 1.1 and earlier versions of HTTP is that HTTP 1.1 uses persistent connection as default. In other words, the server maintains a persistent connection. In earlier version implementations, a separate TCP connection is established for each URL and clients have to make multiple requests for images and associated data on the same URL. This approach was causing congestion and performance problems on the network. Persistent HTTP connections have a number of advantages.

Method Definitions

Currently defined methods are as follows:

Safe and Idempotent Methods
Methods considered not to cause side effects are referred to as *safe.*. Idempotent methods are GET, HEAD, PUT and DELETE.

OPTIONS
This method allows the client to determine the options or requirements associated with a source or capabilities of a server, without any resource retrieval.

GET
This method allows the client to retrieve the data which was determined by the request URI.

HEAD
This method allows the client to retrieve metainformation about the entity which does not require you to transfer the entity body.

POST
The post function is determined by the server.

PUT This method is similar to the post method with one important difference which is the URI in post request identifies the resource that will handle enclosed entity.

DELETE
This methods requests that the server delete the source determined by the request URI.

TRACE
Trace method allows the client to see how the message was retrieved at the other side for testing and diagnostic purposes.

Status Code Definitions

The status code definitions are as follows:

Informational (1xx)
Informational status codes indicate a provisional response. Currently defined codes are as follows:

- 100 Continue
- 101 Switching Protocols

Successful (2xx)
This class of codes indicates that a particular request was successfully received, understood and accepted. Currently defined codes are as follows:

- 200 OK
- 201 Created
- 202 Accepted
- 203 Non-Authoritative Information
- 204 No Content
- 205 Reset Content
- 206 Partial Content

Redirection (3xx)
This class of codes indicates that an action is required from the user agent in order to complete the request. Currently defined codes are as follows:

- 300 Multiple Choices
- 301 Moved Permanently
- 302 Moved Temporarily
- 303 See Other
- 304 Not Modified
- 305 Use Proxy

Client Error (4xx)

This class of codes indicates client errors. Currently defined codes are as follows:

- 400 Bad Request
- 401 Unauthorized
- 402 Payment Required
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 406 Not Acceptable
- 407 Proxy Authentication Required
- 408 Request Timeout
- 409 Conflict
- 410 Gone
- 411 Length Required
- 412 Precondition Failed
- 413 Request Entity Too Large
- 414 Request-URI Too Long
- 415 Unsupported Media Type

Server Error (5xx)

This class of codes indicate client errors. Currently defined codes are as follows:

- 500 Internal Server Error
- 501 Not Implemented
- 502 Bad Gateway
- 503 Service Unavailable
- 504 Gateway Timeout
- 505 HTTP Version Not Supported

Access Authentication

HTTP provides an authentication mechanism to allow servers to define access permissions on resources and clients to use these resources. The authentication method can be one of the following:

Basic Authentication Scheme

Basic authentication is based on user IDs and passwords. In this authentication scheme, the server will permit the connection if only the user ID and password are validated. In basic authentication, user IDs and passwords are not encrypted. They are encoded in base64 format (see 4.8.3.5, “Base64 Encoding” on page 204).

Digest Authentication Scheme

Digest authentication scheme is an extension to HTTP and described in RFC 2069. In this authentication scheme, the user ID and a digest containing an encrypted form of the password are sent to the server. The server computes a similar digest and grants access to the protected resources if the two digests are equal. Notice that if the digest authentication is enabled, what is sent over the network is not simply an encrypted form of the password, which could be decrypted if one had the correct key, but is a one-hash value of the password, which cannot be decrypted. So digest authentication provides a higher level of security than the base-64 encoded password. Unfortunately, digest authentication is not yet supported by all browsers.

Content Negotiation

In order to find the best handling for different types of data, the correct representation for a particular entity body should be negotiated. Actually, the user might handle this by himself or herself but this sometimes is not the best way of representation. There are three types of negotiation:

Server-Driven Negotiation

The representation for a response is determined according to the algorithms located at the server.

Agent-Driven Negotiation

If the representation for a response is determined according to the algorithms located.

Transparent Negotiation

This is a combination of both server-driven and agent-driven negotiation. It is accomplished by a cache that includes a list of all available representations.

HTTP Caching

One of the most important features of HTTP is caching capability. Since HTTP is distributed information-based protocol, caching can improve the performance significantly. There are number of functions that came with the HTTP 1.1 protocol to use caching efficiently and properly.

In most cases, client requests and server responses can be stored in a cache within a reasonable amount of time, to handle the corresponding future requests. If the response is in the cache and accurate, there is no need to request another response from the server. This approach not only reduces the network bandwidth requirement but also increases the speed. There is a mechanism that the server estimates a minimum time in which the response message will be valid. That means, an *expiration* time is determined by the server for that particular response message. Therefore, within this time the message can be used without referring to the server.

Consider that this time is exceeded and there is a need for that response message. The data inside the message might have been changed or not after the expiration date. To be able to ensure whether the data is changed or not, a *validation* mechanism is defined as follows:

Expiration Mechanism

In order to decide whether the data is fresh or not, an expiration time should be determined. In most cases, the origin server explicitly defines the expiration time for a particular response message within that message. If this is the case, the cached data can be used to send from cache for subsequent requests within the expiration time.

If the origin server did not define any expiration time, there are some methods to estimate/calculate a reasonable expiration time (such as the Last-Modified time). Since this is not originated from the server, they should be used cautiously.

Validation Mechanism

When the expiration time is exceeded, there is a possibility that the data is stale. In order to ensure the validation of the response message, the cache has to check with the origin server (or possibly an intermediate cache with a fresh response) whether the response message is still usable. HTTP 1.1 provides conditional methods for this purpose.

When an origin server sends a full response, it attaches some sort of validator to the message. This will then be used as a *cache validator* by the user agent or the proxy cache. The client (user agent or the proxy cache) generates a conditional request with a cache validator attached to it. The server then evaluates the message and responds with a special code (usually, 304 (Not Modified)) and no entity body. Otherwise, the server sends the full response (including the entity body). This approach avoids an extra round-trip if the validator does not match and also avoids sending the full response if the validator matches.

Please refer to RFC 2068 for more details about HTTP caching.

Hypertext Markup Language (HTML)

HTML is one of the major attractions of the Web. It has an architected set of tags that should be understood by all Web browsers and Web servers, although as new features are added to HTML, they may not be supported by older Web browsers. These tags are device independent. The same document can be sent from a personal computer, an AIX or UNIX machine, or a mainframe, and the Web browser on any client machine can understand the HTML tags and build the data stream to display it on the target device. HTML tags describe basic elements of a Web document, such as headers, paragraphs, text styles, and lists. There are also more sophisticated tags to create tables and to include interactive elements, such as forms, scripts or Java applets.

Once document writers and programmers have mastered HTML, those skills are applicable to any operating system on any machine, provided that it has a Web browser.

Since HTML supports hypertext, it allows document writers to include links to other HTML documents. Those documents might be on the same machine as the original, or they might be on a machine on another network on the other side of the world; such is the power of HTML links.

The Extensible Markup Language (XML)

The Extensible Markup Language (XML) describes a class of data objects called XML documents which are stored on computers, and partially describes the behavior of programs that process these objects. XML is an application profile or restricted form of SGML. The goal of XML is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.