1.6 Case study: The World Wide Web

The World Wide Web [www.w3.org I, Berners-Lee 1991] is an evolving system for publishing and accessing resources and services across the Internet. Through commonly available web browsers, users retrieve and view documents of many types, listen to audio streams and view video streams, and interact with an unlimited set of services.

The Web began life at the European centre for nuclear research (CERN), Switzerland, in 1989 as a vehicle for exchanging documents between a community of physicists connected by the Internet [Berners-Lee 1999]. A key feature of the Web is that it provides a *hypertext* structure among the documents that it stores, reflecting the users' requirement to organize their knowledge. This means that documents contain *links* (or *hyperlinks*) – references to other documents and resources that are also stored in the Web.

It is fundamental to the user's experience of the Web that when they encounter a given image or piece of text within a document, this will frequently be accompanied by links to related documents and other resources. The structure of links can be arbitrarily complex and the set of resources that can be added is unlimited – the 'web' of links is indeed world-wide. Bush [1945] conceived of hypertextual structures over 50 years ago; it was with the development of the Internet that this idea could be manifested on a world-wide scale.

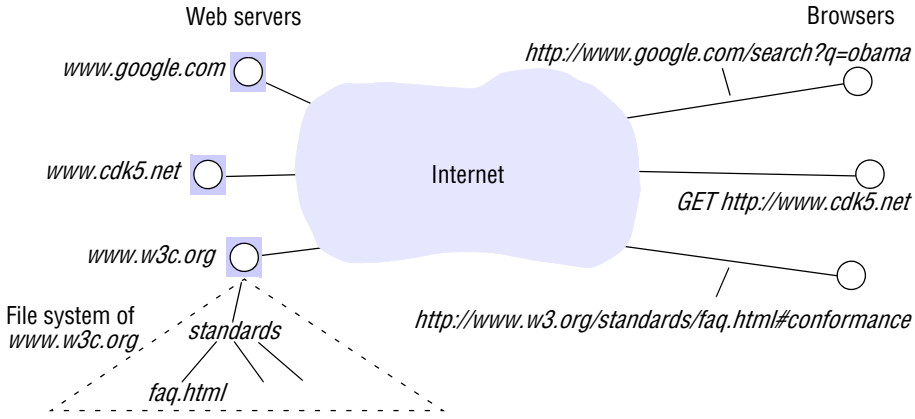
The Web is an *open* system: it can be extended and implemented in new ways without disturbing its existing functionality (see Section 1.5.2). First, its operation is based on communication standards and document or content standards that are freely published and widely implemented. For example, there are many types of browser, each in many cases implemented on several platforms; and there are many implementations of web servers. Any conformant browser can retrieve resources from any conformant server. So users have access to browsers on the majority of the devices that they use, from mobile phones to desktop computers.

Second, the Web is open with respect to the types of resource that can be published and shared on it. At its simplest, a resource on the Web is a web page or some other type of *content* that can be presented to the user, such as media files and documents in Portable Document Format. If somebody invents, say, a new image-storage format, then images in this format can immediately be published on the Web. Users require a means of viewing images in this new format, but browsers are designed to accommodate new content-presentation functionality in the form of 'helper' applications and 'plug-ins'.

The Web has moved beyond these simple data resources to encompass services, such as electronic purchasing of goods. It has evolved without changing its basic architecture. The Web is based on three main standard technological components:

- the HyperText Markup Language (HTML), a language for specifying the contents and layout of pages as they are displayed by web browsers;
- Uniform Resource Locators (URLs), also known as Uniform Resource Identifiers (URIs), which identify documents and other resources stored as part of the Web;
- a client-server system architecture, with standard rules for interaction (the HyperText Transfer Protocol – HTTP) by which browsers and other clients fetch documents and other resources from web servers. Figure 1.7 shows some web servers, and browsers making requests to them. It is an important feature that users may locate and manage their own web servers anywhere on the Internet.

Figure 1.7 Web servers and web browsers



We now discuss these components in turn, and in so doing explain the operation of browsers and web servers when a user fetches web pages and clicks on the links within them.

HTML • The HyperText Markup Language [www.w3.org II] is used to specify the text and images that make up the contents of a web page, and to specify how they are laid out and formatted for presentation to the user. A web page contains such structured items as headings, paragraphs, tables and images. HTML is also used to specify links and which resources are associated with them.

Users may produce HTML by hand, using a standard text editor, but they more commonly use an HTML-aware 'wysiwyg' editor that generates HTML from a layout that they create graphically. A typical piece of HTML text follows:

```
<IMG SRC = "http://www.cdk5.net/WebExample/Images/earth.jpg">          1
<P>                                                                    2
Welcome to Earth! Visitors may also be interested in taking a look at the   3
<A HREF = "http://www.cdk5.net/WebExample/moon.html">Moon</A>.         4
</P>                                                                   5
```

This HTML text is stored in a file that a web server can access – let us say the file *earth.html*. A browser retrieves the contents of this file from a web server – in this case a server on a computer called *www.cdk5.net*. The browser reads the content returned by the server and renders it into formatted text and images laid out on a web page in the familiar fashion. Only the browser – not the server – interprets the HTML text. But the server does inform the browser of the type of content it is returning, to distinguish it from, say, a document in Portable Document Format. The server can infer the content type from the filename extension '.html'.

Note that the HTML directives, known as *tags*, are enclosed by angle brackets, such as <*P*>. Line 1 of the example identifies a file containing an image for presentation. Its URL is *http://www.cdk5.net/WebExample/Images/earth.jpg*. Lines 2 and 5 are directives to begin and end a paragraph, respectively. Lines 3 and 4 contain text to be displayed on the web page in the standard paragraph format.

Line 4 specifies a link in the web page. It contains the word 'Moon' surrounded by two related HTML tags, <*A HREF...*> and </*A*>. The text between these tags is what appears in the link as it is presented on the web page. Most browsers are configured to show the text of links underlined by default, so what the user will see in that paragraph is:

Welcome to Earth! Visitors may also be interested in taking a look at the Moon.

The browser records the association between the link's displayed text and the URL contained in the <*A HREF...*> tag – in this case:

http://www.cdk5.net/WebExample/moon.html

When the user clicks on the text, the browser retrieves the resource identified by the corresponding URL and presents it to the user. In the example, the resource is an HTML file specifying a web page about the Moon.

URLs • The purpose of a Uniform Resource Locator [www.w3.org III] is to identify a resource. Indeed, the term used in web architecture documents is Uniform Resource *Identifier* (URI), but in this book the better-known term URL will be used when no confusion can arise. Browsers examine URLs in order to access the corresponding resources. Sometimes the user types a URL into the browser. More commonly, the browser looks up the corresponding URL when the user clicks on a link or selects one of their 'bookmarks'; or when the browser fetches a resource embedded in a web page, such as an image.

Every URL, in its full, absolute form, has two top-level components:

scheme : scheme-specific-identifier

The first component, the 'scheme', declares which type of URL this is. URLs are required to identify a variety of resources. For example, *mailto:joe@anISP.net* identifies a user's email address; *ftp://ftp.downloadIt.com/software/aProg.exe* identifies a file that is to be retrieved using the File Transfer Protocol (FTP) rather than the more commonly used protocol HTTP. Other examples of schemes are 'tel' (used to specify a telephone number to dial, which is particularly useful when browsing on a mobile phone) and 'tag' (used to identify an arbitrary entity).

The Web is open with respect to the types of resources it can be used to access, by virtue of the scheme designators in URLs. If somebody invents a useful new type of 'widget' resource – perhaps with its own addressing scheme for locating widgets and its own protocol for accessing them – then the world can start using URLs of the form *widget:*.... Of course, browsers must be given the capability to use the new 'widget' protocol, but this can be done by adding a plug-in.

HTTP URLs are the most widely used, for accessing resources using the standard HTTP protocol. An HTTP URL has two main jobs: to identify which web server maintains the resource, and to identify which of the resources at that server is required. Figure 1.7 shows three browsers issuing requests for resources managed by three web servers. The topmost browser is issuing a query to a search engine. The middle browser requires the default page of another web site. The bottommost browser requires a web page that is specified in full, including a path name relative to the server. The files for a given web server are maintained in one or more subtrees (directories) of the server's file system, and each resource is identified by a path name relative to the server.

In general, HTTP URLs are of the following form:

http:// servername [:port] [/pathName] [?query] [#fragment]

where items in square brackets are optional. A full HTTP URL always begins with the string 'http://' followed by a server name, expressed as a Domain Name System (DNS) name (see Section 13.2). The server's DNS name is optionally followed by the number of the 'port' on which the server listens for requests (see Chapter 4), which is 80 by default. Then comes an optional path name of the server's resource. If this is absent then the server's default web page is required. Finally, the URL optionally ends in a query component – for example, when a user submits the entries in a form such as a search engine's query page – and/or a fragment identifier, which identifies a component of the resource.

Consider the URLs:

http://www.cdk5.net
http://www.w3.org/standards/faq.html#conformance
http://www.google.com/search?q=obama

These can be broken down as follows:

Server DNS name	*Path name*	*Query*	*Fragment*
www.cdk5.net	(default)	(none)	(none)
www.w3.org	standards/faq.html	(none)	intro
www.google.com	search	q=obama	(none)

The first URL designates the default page supplied by *www.cdk5.net.* The next identifies a fragment of an HTML file whose path name is *standards/faq.html* relative to the server *www.w3.org*. The fragment's identifier (specified after the '*#*' character in the URL) is *intro*, and a browser will search for that fragment identifier within the HTML text after it has downloaded the whole file. The third URL specifies a query to a search engine. The path identifies a program called 'search', and the string after the '?' character encodes a query string supplied as arguments to this program. We discuss URLs that identify programmatic resources in more detail when we consider more advanced features below.

Publishing a resource: While the Web has a clearly defined model for accessing a resource from its URL, the exact methods for publishing resources on the Web are dependent upon the web server implementation. In terms of low-level mechanisms, the simplest method of publishing a resource on the Web is to place the corresponding file in a directory that the web server can access. Knowing the name of the server *S* and a path name for the file *P* that the server can recognize, the user then constructs the URL as *http://S/P*. The user puts this URL in a link from an existing document or distributes the URL to other users, for example by email.

It is common for such concerns to be hidden from users when they generate content. For example, 'bloggers' typically use software tools, themselves implemented as web pages, to create organized collections of journal pages. Product pages for a company's web site are typically created using a *content management system*, again by

directly interacting with the web site through administrative web pages. The database or file system on which the product pages are based is transparent.

Finally, Huang *et al.* [2000] provide a model for inserting content into the Web with minimal human intervention. This is particularly relevant where users need to extract content from a variety of devices, such as cameras, for publication in web pages.

HTTP • The HyperText Transfer Protocol [www.w3.org IV] defines the ways in which browsers and other types of client interact with web servers. Chapter 5 will consider HTTP in more detail, but here we outline its main features (restricting our discussion to the retrieval of resources in files):

Request-reply interactions: HTTP is a 'request-reply' protocol. The client sends a request message to the server containing the URL of the required resource. The server looks up the path name and, if it exists, sends back the resource's content in a reply message to the client. Otherwise, it sends back an error response such as the familiar '404 Not Found'. HTTP defines a small set of operations or *methods* that can be performed on a resource. The most common are GET, to retrieve data from the resource, and POST, to provide data to the resource.

Content types: Browsers are not necessarily capable of handling every type of content. When a browser makes a request, it includes a list of the types of content it prefers – for example, in principle it may be able to display images in 'GIF' format but not 'JPEG' format. The server may be able to take this into account when it returns content to the browser. The server includes the content type in the reply message so that the browser will know how to process it. The strings that denote the type of content are called MIME types, and they are standardized in RFC 1521 [Freed and Borenstein 1996]. For example, if the content is of type 'text/html' then a browser will interpret the text as HTML and display it; if the content is of type 'image/GIF' then the browser will render it as an image in 'GIF' format; if the content type is 'application/zip' then it is data compressed in 'zip' format, and the browser will launch an external helper application to decompress it. The set of actions that a browser will take for a given type of content is configurable, and readers may care to check these settings for their own browsers.

One resource per request: Clients specify one resource per HTTP request. If a web page contains nine images, say, then the browser will issue a total of ten separate requests to obtain the entire contents of the page. Browsers typically make several requests concurrently, to reduce the overall delay to the user.

Simple access control: By default, any user with network connectivity to a web server can access any of its published resources. If users wish to restrict access to a resource, then they can configure the server to issue a 'challenge' to any client that requests it. The corresponding user then has to prove that they have the right to access the resource, for example, by typing in a password.

Dynamic pages • So far we have described how users can publish web pages and other content stored in files on the Web. However, much of the users' experience of the Web is that of interacting with services rather than retrieving data. For example, when purchasing an item at an online store, the user often fills out a *web form* to provide personal details or to specify exactly what they wish to purchase. A web form is a web

page containing instructions for the user and input widgets such as text fields and check boxes. When the user submits the form (usually by pressing a button or the 'return' key), the browser sends an HTTP request to a web server, containing the values that the user has entered.

Since the result of the request depends upon the user's input, the server has to *process* the user's input. Therefore the URL or its initial component designates a *program* on the server, not a file. If the user's input is a reasonably small set of parameters it is often sent as the *query* component of the URL, using the GET method; alternatively, it is sent as additional data in the request using the POST method. For example, a request containing the following URL invokes a program called 'search' at www.google.com and specifies a query string of 'obama': *http://www.google.com/search?q=obama.*

That 'search' program produces HTML text as its output, and the user will see a listing of pages that contain the word 'obama'. (The reader may care to enter a query into their favourite search engine and notice the URL that the browser displays when the result is returned.) The server returns the HTML text that the program generates just as though it had retrieved it from a file. In other words, the difference between static content fetched from a file and content that is dynamically generated is transparent to the browser.

A program that web servers run to generate content for their clients is referred to as a Common Gateway Interface (CGI) program. A CGI program may have any application-specific functionality, as long as it can parse the arguments that the client provides to it and produce content of the required type (usually HTML text). The program will often consult or update a database in processing the request.

Downloaded code: A CGI program runs at the server. Sometimes the designers of web services require some service-related code to run inside the browser, at the user's computer. In particular, code written in Javascript [www.netscape.com] is often downloaded with a web page containing a form, in order to provide better-quality interaction with the user than that supported by HTML's standard widgets. A Javascript-enhanced page can give the user immediate feedback on invalid entries, instead of forcing the user to check the values at the server, which would take much longer.

Javascript can also be used to update parts of a web page's contents without fetching an entirely new version of the page and re-rendering it. These dynamic updates occur either due to a user action (such as clicking on a link or a radio button), or when the browser acquires new data from the server that supplied the web page. In the latter case, since the timing of the data's arrival is unconnected with any user action at the browser itself, it is termed *asynchronous*. A technique known as *AJAX* (Asynchronous Javascript And XML) is used in such cases. AJAX is described more fully in Section 2.3.2.

An alternative to a Javascript program is an *applet*: an application written in the Java language [Flanagan 2002], which the browser automatically downloads and runs when it fetches a corresponding web page. Applets may access the network and provide customized user interfaces. For example, 'chat' applications are sometimes implemented as applets that run on the users' browsers, together with a server program. The applets send the users' text to the server, which in turn distributes it to all the applets for presentation to the user. We discuss applets in more detail in Section 2.3.1.

Web services • So far we have discussed the Web largely from the point of view of a user operating a browser. But programs other than browsers can be clients of the Web, too; indeed, programmatic access to web resources is commonplace.

However, HTML is inadequate for programmatic interoperation. There is an increasing need to exchange many types of structured data on the Web, but HTML is limited in that it is not extensible to applications beyond information browsing. HTML has a static set of structures such as paragraphs, and they are bound up with the way that the data is to be presented to users. The Extensible Markup Language (XML) (see Section 4.3.3) has been designed as a way of representing data in standard, structured, application-specific forms. In principle, data expressed in XML is portable between applications since it is *self-describing*: it contains the names, types and structure of the data elements within it. For example, XML may be used to describe products or information about users, for many different services or applications. In the HTTP protocol, XML data can be transmitted by the POST and GET operations. In AJAX it can be used to provide data to Javascript programs in browsers.

Web resources provide service-specific operations. For example, in the store at amazon.com, web service operations include one to order a book and another to check the current status of an order. As we have mentioned, HTTP provides a small set of operations that are applicable to any resource. These include principally the GET and POST methods on existing resources, and the PUT and DELETE operations, respectively. for creating and deleting web resources. Any operation on a resource can be invoked using one of the GET or POST methods, with structured content used to specify the operation's parameters, results and error responses. The so-called REST (REpresentational State Transfer) architecture for web services [Fielding 2000] adopts this approach on the basis of its extensibility: every resource on the Web has a URL and responds to the same set of operations, although the processing of the operations can vary widely from resource to resource. The flip-side of that extensibility can be a lack of robustness in how software operates. Chapter 9 further describes REST and takes an in-depth look at the web services framework, which enables the designers of web services to describe to programmers more specifically what service-specific operations are available and how clients must access them.

Discussion of the Web • The Web's phenomenal success rests upon the relative ease with which many individual and organizational sources can publish resources, the suitability of its hypertext structure for organizing many types of information, and the openness of its system architecture. The standards upon which its architecture is based are simple and they were widely published at an early stage. They have enabled many new types of resources and services to be integrated.

The Web's success belies some design problems. First, its hypertext model is lacking in some respects. If a resource is deleted or moved, so-called 'dangling' links to that resource may still remain, causing frustration for users. And there is the familiar problem of users getting 'lost in hyperspace'. Users often find themselves confused, following many disparate links, referencing pages from a disparate collection of sources, and of dubious reliability in some cases.

Search engines are a highly popular alternative to following links as a means of finding information on the Web, but these are imperfect at producing what the user specifically intends. One approach to this problem, exemplified in the Resource

Description Framework [www.w3.org V], is to produce standard vocabularies, syntax and semantics for expressing metadata about the things in our world, and to encapsulate that metadata in corresponding web resources for programmatic access. Rather than searching for words that occur in web pages, programs can then, in principle, perform searches against the metadata to compile lists of related links based on semantic matching. Collectively, the web of linked metadata resources is what is meant by the *semantic web.*

As a system architecture the Web faces problems of scale. Popular web servers may experience many 'hits' per second, and as a result the response to users can be slow. Chapter 2 describes the use of caching in browsers and proxy servers to increase responsiveness, and the division of the server's load across clusters of computers.

1.7 Summary

Distributed systems are everywhere. The Internet enables users throughout the world to access its services wherever they may be located. Each organization manages an intranet, which provides local services and Internet services for local users and generally provides services to other users in the Internet. Small distributed systems can be constructed from mobile computers and other small computational devices that are attached to a wireless network.

Resource sharing is the main motivating factor for constructing distributed systems. Resources such as printers, files, web pages or database records are managed by servers of the appropriate type. For example, web servers manage web pages and other web resources. Resources are accessed by clients – for example, the clients of web servers are generally called browsers.

The construction of distributed systems produces many challenges:

Heterogeneity: They must be constructed from a variety of different networks, operating systems, computer hardware and programming languages. The Internet communication protocols mask the difference in networks, and middleware can deal with the other differences.

Openness: Distributed systems should be extensible – the first step is to publish the interfaces of the components, but the integration of components written by different programmers is a real challenge.

Security: Encryption can be used to provide adequate protection of shared resources and to keep sensitive information secret when it is transmitted in messages over a network. Denial of service attacks are still a problem.

Scalability: A distributed system is scalable if the cost of adding a user is a constant amount in terms of the resources that must be added. The algorithms used to access shared data should avoid performance bottlenecks and data should be structured hierarchically to get the best access times. Frequently accessed data can be replicated.

Failure handling: Any process, computer or network may fail independently of the others. Therefore each component needs to be aware of the possible ways in which

the components it depends on may fail and be designed to deal with each of those failures appropriately.

Concurrency: The presence of multiple users in a distributed system is a source of concurrent requests to its resources. Each resource must be designed to be safe in a concurrent environment.

Transparency: The aim is to make certain aspects of distribution invisible to the application programmer so that they need only be concerned with the design of their particular application. For example, they need not be concerned with its location or the details of how its operations are accessed by other components, or whether it will be replicated or migrated. Even failures of networks and processes can be presented to application programmers in the form of exceptions – but they must be handled.

Quality of service. It is not sufficient to provide access to services in distributed systems. In particular, it is also important to provide guarantees regarding the qualities associated with such service access. Examples of such qualities include parameters related to performance, security and reliability.

EXERCISES

1.1 Give five types of hardware resource and five types of data or software resource that can usefully be shared. Give examples of their sharing as it occurs in practice in distributed systems. *pages 2, 14*

1.2 How might the clocks in two computers that are linked by a local network be synchronized without reference to an external time source? What factors limit the accuracy of the procedure you have described? How could the clocks in a large number of computers connected by the Internet be synchronized? Discuss the accuracy of that procedure. *page 2*

1.3 Consider the implementation strategies for massively multiplayer online games as discussed in Section 1.2.2. In particular, what advantages do you see in adopting a single server approach for representing the state of the multiplayer game? What problems can you identify and how might they be resolved? *page 5*

1.4 A user arrives at a railway station that they has never visited before, carrying a PDA that is capable of wireless networking. Suggest how the user could be provided with information about the local services and amenities at that station, without entering the station's name or attributes. What technical challenges must be overcome? *page 13*

1.5 Compare and contrast cloud computing with more traditional client-server computing? What is novel about cloud computing as a concept? *pages 13, 14*

1.6 Use the World Wide Web as an example to illustrate the concept of resource sharing, client and server. What are the advantages and disadvantages of HTML, URLs and HTTP as core technologies for information browsing? Are any of these technologies suitable as a basis for client-server computing in general? *pages 14, 26*