UNIT 1

INTRODUCTION

Examples of Distributed Systems–Trends in Distributed Systems – Focus on resource sharing – Challenges. Case study: World Wide Web.

A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages. The components interact with each other in order to achieve a common goal.

REAL TIME EXAMPLES OF DISTRIBUTED SYSTEMS:

There are somes pecific examples of distributed systems to illustrate the diversity and indeed complexity of distributed systems provision today.

Web Search

Web search has emerged as a major growth industry in the last decade, with recent figures indicating that the global number of searches has risen to over 10 billion per calendar month. The task of a web search engine is to index the entire contents of the World Wide Web. This is a very complex task, as current estimates state that the Web consists of over 63 billion pages and one trillion unique webaddresses. Given that most search engines analyze the entire web content and then carry out sophisticated processing on this enormous database, this task itself represents a major challenge for distributed systems design.Google, the market leader in web search technology, has put significant effort into the design of a sophisticated distributed system infrastructure to support search

Highlights of this infrastructureinclude:

• an underlying physical infrastructure consisting of very large numbers ofnetworked computers located at data centres all around the world;

• a distributed file system designed to support very large files and heavily optimizedfor the style of usage required by search and other Google applications

• an associated structured distributed storage system that offers fast access to very large datasets;

• a lock service that offers distributed system functions such as distributed lockingand agreement

• a programming model that supports the management of very large parallel anddistributed computations across the underlying physical infrastructure.

Massive Multiplayer Online Games(MMOGs)

Massively multiplayer online games offer an immersive experience whereby very large numbers of users interact through the Internet with a persistent virtual world. Leading examples of such games include Sony's EverQuest II and EVE Online from the Finnish company CCP Games.The number of players is also rising, with systems able to support over 50,000 simultaneous online players.The engineering of MMOGs represents a major challenge for distributed systems technologies, particularly because of the need for fast response times to preserve the user experience of the game.

A number of solutions have been proposed for the design of massively multiplayer online games:

1. Perhaps surprisingly, the largest online game, EVE Online, utilises a *client-server* architecture where a single copy of the state of the world is maintained on a centralized server and accessed by client programs running on players' consoles or other devices. To support large numbers of clients, the server is a complex entity in its own right consisting of a cluster architecture featuring hundreds of computer nodes.
2. Other MMOGs adopt more distributed architectures where the universe is partitioned across a (potentially very large) number of servers that may also be geographically distributed.

Financial Trading

The financial industry has long been at the cutting edge of distributed systems technology with its need, in particular, for real-time access to a wide range of information sources.

Note that the emphasis in such systems is on the communication and processing of items of interest, known as *events* in distributed systems, with the need also to deliver events reliably and in a timely manner to potentially very large numbers of clients who have a stated interest in such information items. Examples of such events include a drop in a share price, the release of the latest unemployment figures, and so on. This requires

a very different style of underlying architecture from the styles mentioned above and such systems typically employ what are known as *distributed event-based systems*.

This approach is primarily used to develop customized algorithmic trading strategies covering both buying and selling of stocks and shares, in particular looking for patterns that indicate a trading opportunity and then automatically responding by placing and managing orders.

<u>TRENDS IN DISTRIBUTED SYSTEMS</u>

Distributed systems are undergoing a period of significant change and this can be traced back to a number of influential trends:

- the emergence of pervasive networking technology;
- the emergence of ubiquitous computing coupled with the desire to support usermobility in distributed systems;
- the increasing demand for multimedia services;
- the view of distributed systems as a utility.

Pervasive networking and the modern Internet

The modern Internet is a vast interconnected collection of computer networks of many different types, with the range of types increasing all the time and now including, for example, a wide range of wireless communication technologies such as WiFi, WiMAX, Bluetooth and third-generation mobile phone networks. The net result is that networking has become a pervasive resource and devices can be connected at any time and in any place.

The Internet is also a very large distributed system. It enables users, wherever they are, to make use of services such as the World Wide Web, email and file transfer. The role of a *firewall* is to protect an intranet by preventing unauthorized messages from leaving or entering. A firewall is implemented by filtering incoming and outgoing messages. Internet Service Providers (ISPs) are companies that provide broadband links and other types of connection to individual users and small organizations. A *backbone* is a network link with a high transmission capacity, employing satellite connections, fibre optic cables and other high-bandwidth circuits.

Mobile and ubiquitous computing

Technological advances in device miniaturization and wireless networking have led increasingly to the integration of small and portable computing devices into distributed systems. These devices include laptop computers, mobile phones, Personal Digital Assistants(PDAs) etc.

The portability of many of these devices, together with their ability to connect conveniently to networks in different places, makes *mobile computing* possible. Mobile computing is the performance of computing tasks while the user is on the move, or visiting places other than their usual environment. In mobile computing, users who are away from their ‘home’ intranet (the intranet at work, or their residence) are still provided with access to resources via the devices they carry with them. They can continue to access the Internet; they can continue to access resources in their home intranet; and there is increasing provision for users to utilize resources such as printers or even sales points that are conveniently nearby as they move around. The latter is also known as *location-aware* or *context-aware computing*. Mobility

introduces a number of challenges for distributed systems, including the need to deal with variable connectivity and indeed disconnection, and the need to maintain operation in the face of device mobility.
Ubiquitous computing is the harnessing of many small, cheap computational devices that are present in users' physical environments, including the home, office and even natural settings. Ubiquitous and mobile computing overlap, since the mobile user can in principle benefit from computers that are everywhere. But they are distinct, in general. Ubiquitous computing could benefit users while they remain in a single environment such as the home or a hospital. Similarly, mobile computing has advantages even if it involves only conventional, discrete computers and devices such as laptops and printers.

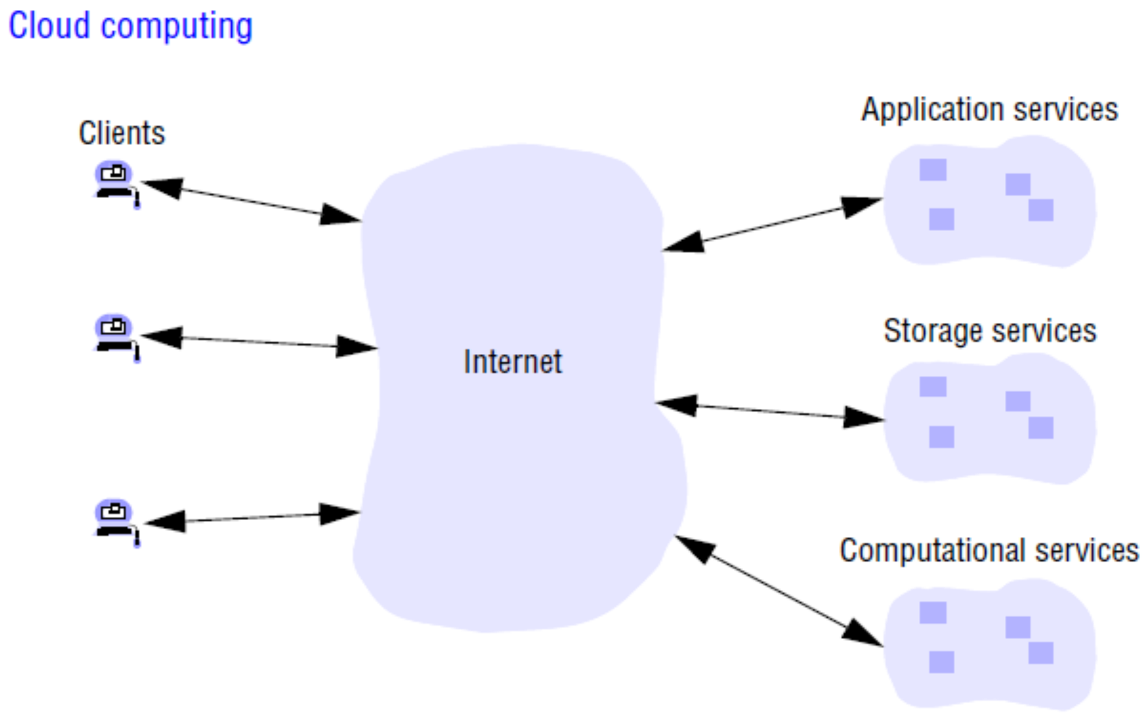
Distributed multimedia systems

Another important trend is the requirement to support multimedia services in distributed systems. Multimedia support can usefully be defined as the ability to support a range of media types in an integrated manner. One can expect a distributed system to support the storage, transmission and presentation of what are often referred to as discrete media types, such as pictures or text messages. A distributed multimedia system should be able to perform the same functions for continuous media types such as audio and video; that is, it should be able to store and locate audio or video files, to transmit them across the network to support the presentation of the media types to the user and optionally also to share the media types across a group of users. The benefits of distributed multimedia computing are considerable in that a wide range of new services and applications can be provided on the desktop.
Webcasting is an application of distributed multimedia technology. Webcasting is the ability to broadcast continuous media, typically audio or video, over the Internet.

Distributed computing as a utility

With the increasing maturity of distributed systems infrastructure, a number of companies are promoting the view of distributed resources as a commodity or utility, drawing the analogy between distributed resources and other utilities such as water or electricity.
This model applies to both physical resources and more logical services:
Physical resources such as storage and processing can be made available to networked computers, removing the need to own such resources on their own. At one end of the spectrum, a user may opt for a remote storage facility for file storage requirements and/or for backups.
Software services can also be made available across the global Internet using this approach.
The term *cloud computing* is used to capture this vision of computing as a utility. A cloud is defined as a set of Internet-based application, storage and computing services sufficient to support most users' needs, thusenabling them to largely or totally dispense with local data storage and application software.

A *cluster computer* is a set of interconnected computers that cooperate closely to provide a single, integrated high performance computing capability. The overall goal of cluster computers is to provide a range of cloud services, including high-performance computing capabilities, mass storage and richer application services such as web search.



FOCUS ON RESOURCE SHARING

Users are so accustomed to the benefits of resource sharing that they may easilyoverlook their significance. We routinely share hardware resources such as printers, dataresources such as files, and resources with more specific functionality such as searchengines.
In practice, patterns of resource sharing vary widely in their scope and in how closely users work together. At one extreme, a search engine on the Web provides a facility to users throughout the world, users who need never come into contact with one another directly. At the other extreme, in *computer-supported cooperative working*(CSCW), a group of users who cooperate directly share resources such as documents in a small, closed group. The pattern of sharing and the geographic distribution of particular users determines what mechanisms the system must supply to coordinate users' actions.

The term *service is used* for a distinct part of a computer system that manages a collection of related resources and presents their functionality to users and applications. For example, we access shared files through a file service; we send documents to printers through a printing service; we buy goods through an electronic payment service. The only access we have to the service is via the set of operations that it exports. For example, a file service provides *read*, *write* and *delete* operations on files. The fact that services restrict resource access to a well-defined set of operations is in part standard software engineering practice. But it also reflects the physical organization of distributed systems. Resources in a distributed system are physically encapsulated within computers and can only be accessed from other computers bymeans of communication. For effective sharing, each resource must be managed by a program that offers a communication interface enabling the resource to be accessed and updated reliably and consistently.

CHALLENGES IN DISTRIBUTED SYSTEMS:

Major challenges faced in a distributed system are:

- Heterogeneity
- Openness
- Security
- Scalability
- Failure Handling
- Concurrency
- Transparency
- Quality of Service

Heterogeneity

The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks. Heterogeneity (that is, variety and difference) applies to all of the following:

- networks;
- computer hardware;
- operating systems;
- programming languages;
- implementations by different developers.

The term *middleware* applies to a software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages. In addition to solving the problems of heterogeneity, middleware provides a uniform computational model for use by the programmers of servers and distributed applications. Possible models include remote object invocation, remote eventnotification, remote SQL access and distributed transaction processing.The term *mobile code* is used to refer to program code that can be transferred from one computer to another and run at the destination – Java applets are an example. Code suitable for running on one computer is not necessarily suitable for running on another because executable programs are normally specific both to the instruction set and to the host operating system.

The *virtual machine* approach provides a way of making code executable on a variety of host computers: the compiler for a particular language generates code for a virtual machine instead of a particular hardware order code. For example, the Java compiler produces code for a Java virtual machine, which executes it by interpretation. The Java virtual machine needs to be implemented once for each type of computer to enable Java programs to run.

Openness

The openness of a computer system is the characteristic that determines whether the system can be extended and reimplemented in various ways. The openness of distributed systems is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs.

Systems that are designed to support resource sharing in this way are termed *open distributed systems* to emphasize the fact that they are extensible. They may be extendedat the hardware level by the addition of computers to the network and at the softwarelevel by the introduction of new services and the reimplementation of old ones, enablingapplication programs to share resources. A further benefit that is often cited for opensystems is their independence from individual vendors.

In short,

- Open systems are characterized by the fact that their key interfaces are published.
- Open distributed systems are based on the provision of a uniform communication mechanism and published interfaces for access to shared resources.

• Open distributed systems can be constructed from heterogeneous hardware and software, possibly from different vendors. But the conformance of each component to the published standard must be carefully tested and verified if the system is to work correctly.

Security
Many of the information resources that are made available and maintained in distributedsystems have a high intrinsic value to their users. Their security is therefore of considerable importance. Security for information resources has three components: confidentiality , integrity and availability
In a distributed system, clients send requests to access data managed by servers, which involves sending information in messages over a network.
For example:
1. A doctor might request access to hospital patient data or send additions to that data.
2. In electronic commerce and banking, users send their credit card numbers across the Internet.
In both examples, the challenge is to send sensitive information in a message over a network in a secure manner. But security is not just a matter of concealing the contents of messages – it also involves knowing for sure the identity of the user or other agent on whose behalf a message was sent. In the first example, the server needs to know that the user is really a doctor, and in the second example, the user needs to be sure of the identity of the shop or bank with which they are dealing. The second challenge here is to identify a remote user or other agent correctly. Both of these challenges can be met by the use of encryption techniques developed for this purpose.
Following two security challenges have not yet been fully met:
Denial of service attacks: Another security problem is that a user may wish to disrupt a service for some reason. This can be achieved by bombarding the service with such a large number of pointless requests that the serious users are unable to use it. This is called a *denial of service* attack.
Security of mobile code: Mobile code needs to be handled with care. Consider someone who receives an executable program as an electronic mail attachment: the possible effects of running the program are unpredictable.

Concurrency
The process that manages a shared resource could take one client request at a time. But that approach limits throughput. Therefore services and applications generally allow multiple client requests to be processed concurrently. To make this more concrete, suppose that each resource is encapsulated as an object and that invocations are executed in concurrent threads. In this case it is possible that several threads may be executing concurrently within an object, in which case their operations on the object may conflict with one another and produce inconsistent results. For example, if two concurrent bids at an auction are 'Smith: $122' and 'Jones: $111', and the corresponding operations are interleaved without any control, then they might get stored as 'Smith: $111' and 'Jones: $122'. The moral of this example is that any object that represents a shared resource in a distributed system must be responsible for ensuring that it operates correctly in a concurrent environment. This applies not only to servers but also to objects in applications.

Therefore any programmer who takes an implementation of an object that was not intended for use in a distributed system must do whatever is necessary to make it safe in a concurrent environment. For an object to be safe in a concurrent environment, its operations must be synchronized in such a way that its data remains consistent. This can be achieved by standard techniques such as semaphores, which are used in most operating systems.

Transparency
Transparency is defined as the concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components. The implications of transparency are a major influence on the

design of the system software. The ANSA Reference Manual [ANSA 1989] and the International Organization for Standardization's Reference Model for Open Distributed Processing (RM-ODP) [ISO 1992] identify eight forms of transparency. We have paraphrased the original ANSA definitions, replacing their migration transparency with our own mobility transparency, whose scope is broader:

Access transparency enables local and remote resources to be accessed using identical operations.

Location transparency enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).

Concurrency transparency enables several processes to operate concurrently using shared resources without interference between them.

Replication transparency enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.

Failure transparency enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware orsoftware components.
Mobility transparency allows the movement of resources and clients within a system without affecting the operation of users or programs.

Performance transparency allows the system to be reconfigured to improve performance as loads vary.

Scaling transparency allows the system and applications to expand in scale without change to the system structure or the application algorithms.

The two most important transparencies are access and location transparency; theirpresence or absence most strongly affects the utilization of distributed resources. They are sometimes referred to together as *network transparency*.

Quality of Service

Once users are provided with the functionality that they require of a service, such as the file service in a distributed system, we can go on to ask about the quality of the service provided. The main nonfunctional properties of systems that affect the quality of the service experienced by clients and users are *reliability*, *security* and *performance*.

Adaptability to meet changing system configurations and resource availability has been recognized as a further important aspect of service quality.

Reliability and security issues are critical in the design of most computer systems. The performance aspect of quality of service was originally defined in terms of responsiveness and computational throughput, but it has been redefined in terms of ability to meet timeliness guarantees, as discussed in the following paragraphs.

Some applications, including multimedia applications, handle *time-critical data* – streams of data that are required to be processed or transferred from one process to another at a fixed rate. For example, a movie service might consist of a client program that is retrieving a film from a video server and presenting it on the user's screen. For a satisfactory result the successive frames of video need to be displayed to the user within some specified time limits. In fact, the abbreviation QoS has effectively been commandeered to refer to the ability of systems to meet such deadlines. Its achievement depends upon the availability of the necessary computing and network resources at the appropriate times.

Scalability:
Distributed systems operate effectively and efficiently at many different scales, ranging from a small intranet to the Internet. A system is described as *scalable* if it will remain effective when there is a significant increase in the number of resources and the number of users. The number of computers and servers in the Internet has increased dramatically.

The design of scalable distributed systems presents the following challenges:
Controlling the cost of physical resources: As the demand for a resource grows, it should be possible to extend the system, at reasonable cost, to meet it. For example, the frequency with which files are accessed in an intranet is likely to grow as the number of users and computers increases. It must be possible to add server computers to avoid the performance bottleneck that would arise if a single file server had to handle all file access requests.

Controlling the performance loss: Consider the management of a set of data whose size is proportional to the number of users or resources in the system – for example, the table with the correspondence between the domain names of computers and their Internet addresses held by the Domain Name System, which is used mainly to look up DNS names such as www.amazon.com.

Preventing software resources running out: An example of lack of scalability is shown by the numbers used as Internet (IP) addresses .

Avoiding performance bottlenecks: In general, algorithms should be decentralized to avoid having performance bottlenecks. We illustrate this point with reference to the predecessor of the Domain Name System, in which the name table was kept in a single master file that could be downloaded to any computers that needed it. That was fine when there were only a few hundred computers in the Internet, but it soon became a serious performance and administrative bottleneck.

Failure Handling
Failures in a distributed system are partial – that is, some components fail while others continue to function. Therefore the handling of failures is particularly difficult. The following techniques for dealing with failures has been in implementation:

Detecting failures: Some failures can be detected. For example, checksums can be used to detect corrupted data in a message or a file.

Masking failures: Some failures that have been detected can be hidden or made lesssevere.
Two examples of hiding failures:
1. Messages can be retransmitted when they fail to arrive.
2. File data can be written to a pair of disks so that if one is corrupted, the other may still be correct.

Tolerating failures: Most of the services in the Internet do exhibit failures – it would not be practical for them to attempt to detect and hide all of the failures that might occur in such a large network with so many components. Their clients can be designed to tolerate failures, which generally involves the users tolerating them as well.

Recovery from failures: Recovery involves the design of software so that the state of permanent data can be recovered or 'rolled back' after a server has crashed.

Redundancy: Services can be made to tolerate failures by the use of redundant components. Consider the following examples:
1. There should always be at least two different routes between any two routers in the Internet.
2. In the Domain Name System, every name table is replicated in at least twodifferent servers.
3. A database may be replicated in several servers to ensure that the data remains accessible after the failure of any single server; the servers can be designed to detect faults in their peers; when a fault is detected in one server, clients are redirected to the remaining servers.

Distributed systems provide a high degree of availability in the face of hardware faults. The *availability* of a system is a measure of the proportion of time that it is available for use. When one of the components in a distributed system fails, only the work that was using the failed component is affected.

CASE STUDY: WORLD WIDE WEB

The World Wide Web is an evolving system forpublishing and accessing resources and services across the Internet. Through commonlyavailable web browsers, users retrieve and view documents of many types, listen toaudio streams and view video streams, and interact with an unlimited set of services.The Web began life at the European center for nuclear research (CERN),Switzerland, in 1989 as a vehicle for exchanging documents between a community ofphysicists connected by the Internet [Berners-Lee 1999]. A key feature of the Web is thatit 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.

The Web is an *open* system: it can be extended and implemented in new ways without disturbing its existing functionality . First, its operation is based on communication standards and document or content standards that are freely published and widely implemented.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.

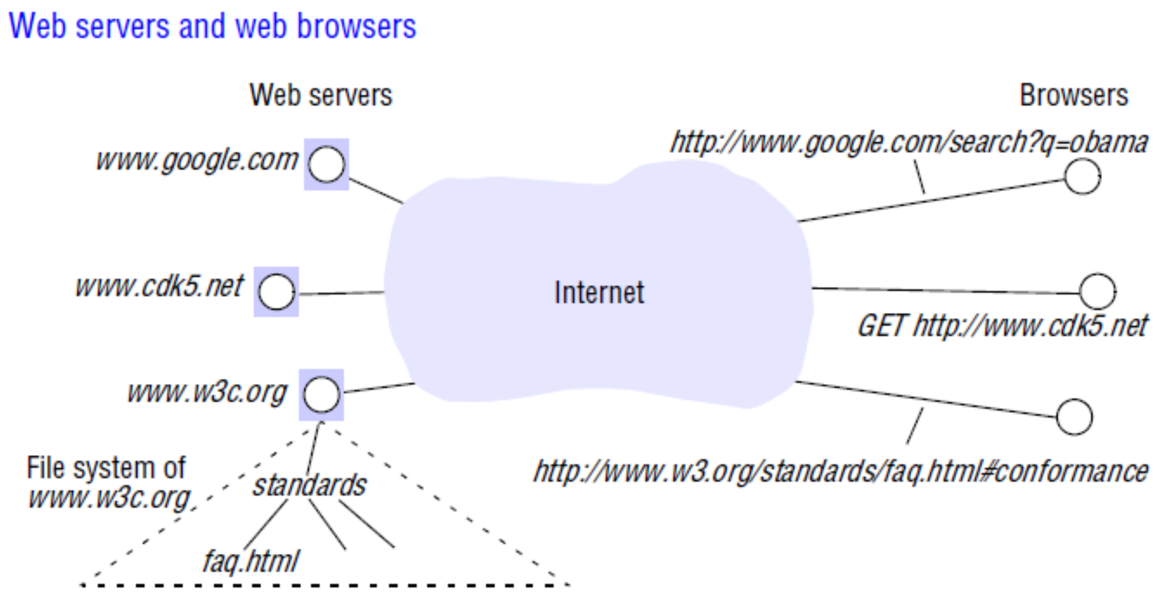
The Web has moved beyond these simple data resources to encompass services, such as electronic purchasing of goods. It has evolved without changing its basic architecture. The Web is based on three main standard technological components:

• 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.

HTML
The HyperText Markup Language is used to specify the textand images that make up the contents of a web page, and to specify how they are laidout and formatted for presentation to the user. A web page contains such structured itemsas headings, paragraphs, tables and images. HTML is also used to specify links andwhich resources are associated with them.

URLs
The purpose of a Uniform Resource Locator 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.HTTP URLs are the most widely used, for accessing resources using the standardHTTP protocol. An HTTP URL has two main jobs: to identify which web servermaintains the resource, and to identify which of the resources at that server is required.



Web servers and web browsers
Web servers
Browsers
www.google.com
www.cdk5.net
www.w3c.org
Internet
http://www.google.com/search?q=obama
GET http://www.cdk5.net
http://www.w3.org/standards/faq.html#conformance
File system of www.w3c.org
standards
faq.html

HTTP

The HyperText Transfer Protocol defines the ways in which browsers and other types of client interact with web servers.

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 theresource, 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.