Applications

Architectures

Security Issues

The Web

Overview

- Client-server architectures
  - message-passing
  - remote procedure calls
- Security
  - cryptography
- Example distributed system: the world-wide web
- References:
  - Kurose & Ross: Ch 2.1
  - Tanenbaum: Ch 7.1, 7.6
  - Also:
    - Computer Networks and Internets, Second Edition; Douglas Comer; Prentice-Hall, 1999; ISBN 0-13-083617-6
    - Business Data Communications, Third Edition; Stallings & Van Slyke; Prentice-Hall, 1998; ISBN 0-13-594581-X

Characteristics of a distributed system

- Resource sharing
  - sharing hardware, software, data
  - resource management
- Openness:
  - standardized interfaces
  - vendor and op-sys independence
- Concurrency
- Scalability
  - as the demand for a resource increases, extend (add more)
  - counter-example: IP addresses
- Fault tolerance (availability)
  - adaptation to & recovery from errors
- Transparency
  - users should not be aware of the “distributedness”

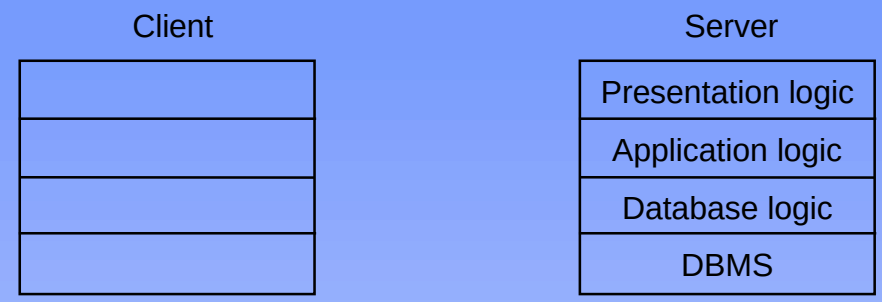
Resource sharing

- Client-server model:
  - resources have managers, users
  - servers are the *resource managers*, clients are *resource users*
  - clients send messages to servers; server returns results
- Object-based model:
  - resources are objects (data, code) with message-handling interface
  - *object managers* for classes of objects
  - clients send messages (invoke object methods), manager returns results (set properties of objects)
    - virtually the same as procedural client-server interfaces, although some object paradigms make method invocation implicit (instantiating the object invokes initialization methods that sent messages to object managers)

Client-server model

- Varying divisions of work between clients and servers
- Host-based processing:
  - all server, no client; eg mainframe and dumb terminals
- Server-based:
  - server does all computation, client does only presentation (eg GUI)
- Client-based:
  - client does as much as possible, server acts as central repository only
- Coöperative
  - “rational” balance between client and server
- Choice depends on workload, capability, availability etc.

… client-server, 2



Host-based processing

Server-based processing

Coöperative processing

Client-based processing

… client-server, 3

- Characteristics of client software
  - perceived as “the application” by user, need not be aware of “distributedness”
    - eg RealPlayer
  - client initiated directly by user
    - client initiates contact with one or more servers, as required
  - executes at user’s computer
  - runs as a standard program within the context of standard operating-system facilities

… client-server, 3

- Characteristics of server software
  - initiated automatically by operating-system startup sequence
  - handles many clients simultaneously
  - waits (blocking wait) to be contacted by clients
    - processes incoming client requests by responding to messages at predetermined Service Access Points
  - usually offers a single service
  - usually runs on dedicated hardware, possibly running special-purpose operating-system
    - “server-class” system
    - generally not the same as user workstations
- “Personal servers” are somewhat oxymoronic, but legitimate and useful

… client-server, 4

- Server handle multiple clients simultaneously
  - servers normally have only one well-known SAP
  - how can multiple clients be handled?
  - clients contact server at the well-known SAP, then are “handed off” to another access point
    - requires that servers be able to create client-handling processes or threads dynamically
  - servers can provide the same function through different protocols
    - classic example is a server that provides service through both TCP and UDP ports, giving clients a choice

… client-server, 5

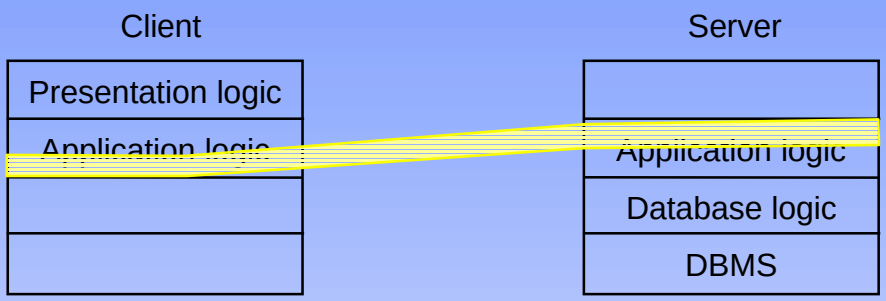
- Basic decisions to be made in creating a client-server system
  - what division of work?
    - where does the application logic reside?
  - what are the message formats and application protocols?
  - what transport services will be required?
    - connection-oriented or connectionless?
    - if CL, reliable and in-order or not?
      - if not, must handle in the application
  - what will be the SAPs for the server side?
  - what non-standard services might be required?
    - assuming a TCP/IP reference model, are there any services that would be classified as presentation or session?
      - are there any proprietary layers or sub-layers that can do the work?

Middleware

- In making the basic decisions, implementor will be re-implementing the same basic processes over and over
  - eg if there are common non-standard services required by applications
  - database-driven client-server systems have many common features
- Software engineering principle: whenever one is re-implementing, think: generalize, package, parameterize
  - in distributed and network-based systems, this means “think layer”

… middleware, 2

- Recall the example division of client-server systems (coöperative):

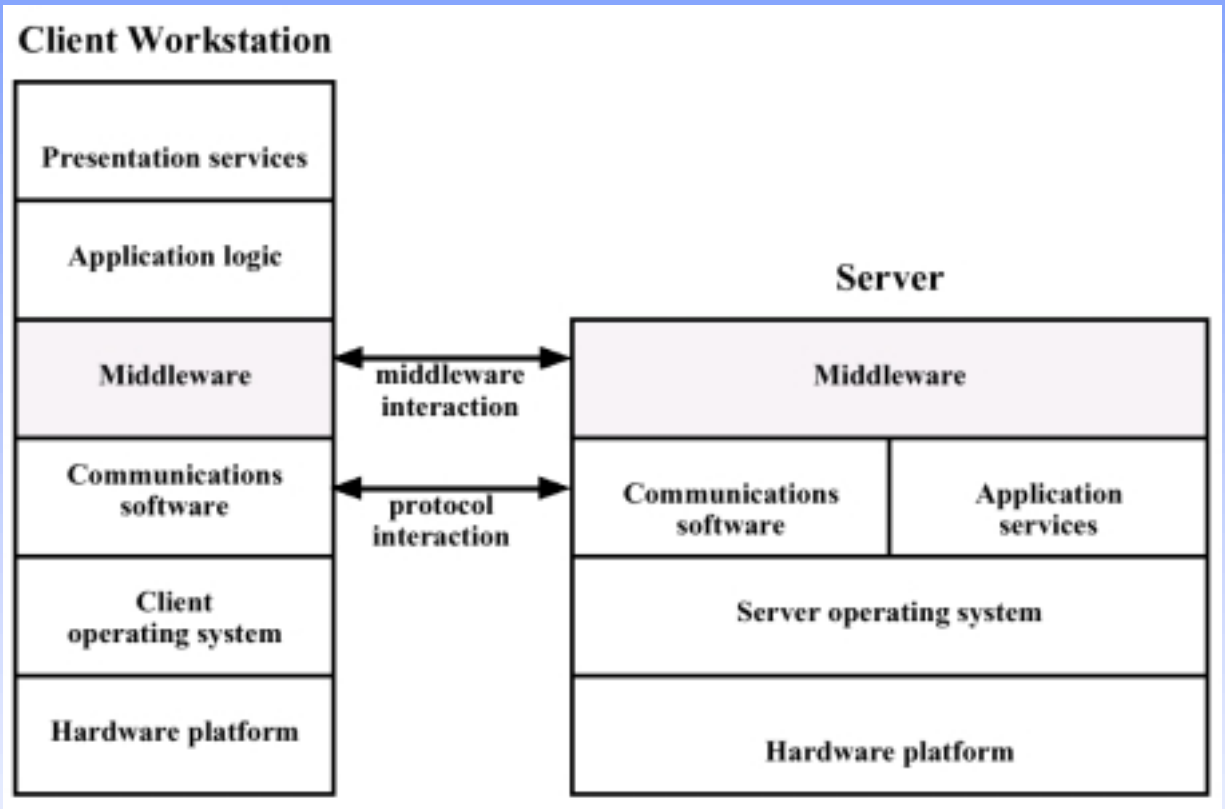


- The highlighted region tends to be duplicated in many applications, and can be isolated into a sub-layer or semi-layer
  - called *middleware,* or the middleware layer
  - middleware provides a standard set of services for the implementation of client-server applications

… middleware, 3

- Picture:

  From: Stallings & Van Slyke, Fig 14.8

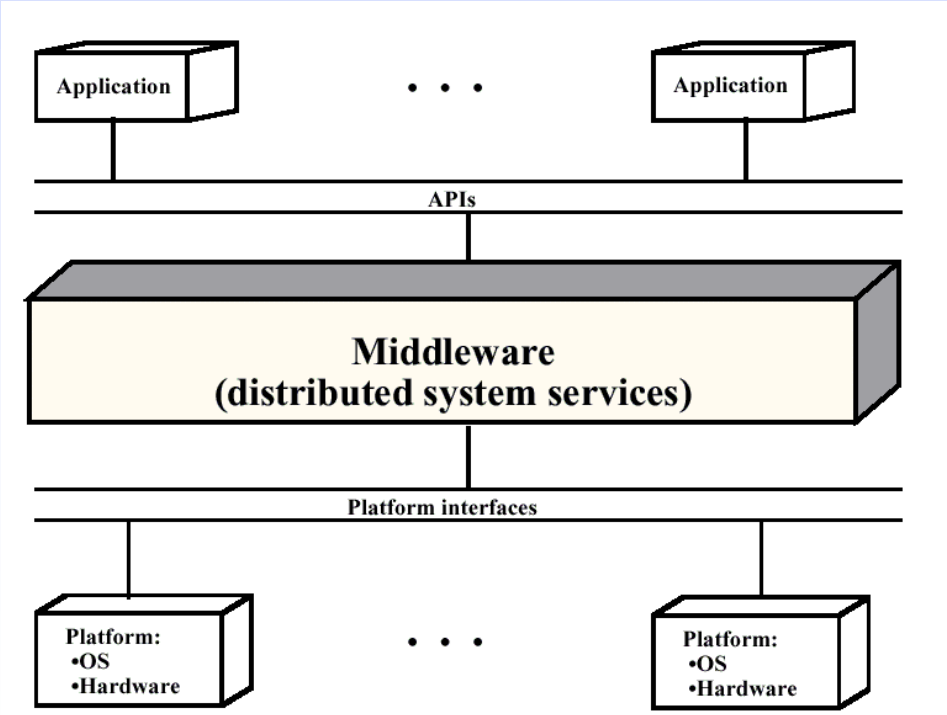


- Application services include database servers

… middleware, 4

- In commercial middleware, the reach of the services extends to database services and networking platforms
  - defines a standard interface between applications and databases, applications and network services
    - by building products with a modular architecture, different plug-in modules provide interoperability and independence between database and networking products

From Stallings & Van Slyke, Fig 14.9



Implementation

- How are client/server applications and middleware layers implemented?
- Message-passing:
  - send, receive
    - blocking (synchronous) or non-blocking (asynchronous)
    - reliable vs unreliable
  - puts the work in the protocols
- Remote Procedure Call (RPC):
  - encapsulate message-passing (reliable, blocking) with standard procedure-call
  - calling a procedure elsewhere
    - client calls a local procedure, which uses send/receive to communicate with procedure on server
    - values can be returned as usual
  - function-shipping: send the code to be executed
    - eg Java, PostScript

… implementation, 2

- Object-oriented methods:
  - client and server are objects that exchange messages
  - to request a service, a client object contacts an object request broker
    - broker has a list of all available servers (i.e. acts as a directory), locates available server objects, and passes along the service request
  - object communications can be RPC-based, message passing or use operating-system object paradigm (if available)

Current ~~buzzwords~~ trends

- COM, OLE, DCOM, COM+: Microsoft
  - COM: Component Object Module; operating-system software architecture
  - OLE: Object Linking and Embedding; object interaction on a single system
  - DCOM: Distributed COM
  - COM+: combination of COM & DCOM
- DCE: Open Software Foundation
  - DCE: Distributed Computing Environment; RPC-based middleware; a.k.a. DCE/RPC
- CORBA: Object Management Group(OMG)
  - CORBA: Common Object Request Broker Architecture
- Plus:
  - ActiveX: GUI objects based on COM
  - ADO (ActiveX Data Objects): data and database access based on ActiveX
  - SOM, DSOM: IBM OS/2's attempts
  - RDO, RDS, MDAC, blah blah blah .....

Security

- Distributed systems are especially susceptible to unauthorized access
  - the only really secure system is one with no networks or terminals, housed in a locked room
  - for useful commercial distributed systems, some form of protection against security threats is required
- Security threats (system perspective):
  - passive: end-users are (probably) unaware; eavesdropping or monitoring
  - active: interruption or modification of services or messages
- Security threats (user perspective)
  - secrecy: keeping messages secret
  - authentication: verifying identities
  - (non)repudiation: proving message transmission
  - integrity: message correctness

… Security: system threats

- Passive threats:
  - others recording or reading messages
  - others publishing message contents
  - monitoring traffic source/destination, patterns (length, frequency)
- Active threats:
  - inserting false messages
  - modifying (delete, delay) messages
  - impersonating an authentic user or host system
  - denial or hindering service
- Passive threats are hard(er) to detect, but easy(ier) to prevent
- Active threats are easy(ier) to detect, but hard(er) to prevent

… Security: system threats, 2

- Denial/hindrance of service can be just as damaging as falsifying or stealing information
  - *packet storms*: attacker sends constant stream of messages to the point of overloading host's ability to process
    - messages themselves can be legitimate eg ICMP, DNS
    - many attackers (thousands) can join together to create an overwhelming storm
  - no general solution, but some measures are possible
    - gateway or firewall to discard unwanted protocols
    - address filtering – discard packets based on origin

… Security: user threats

- Secrecy:
  - ensuring that message content is viewable only be intended recipients
- Authentication:
  - ensuring that senders and recipients are who they claim to be
  - providing a “signature” mechanism
- Integrity:
  - ensuring that message content is not modified or falsified completely
- Non-repudiation:
  - preventing a sender from denying content of a message
    - eg “that’s not my signature”

Physical security

- System threats (active, passive) require access to communications channels
  - passive requires only observation
  - active requires direct intervention
- Choice of physical media affects security:
  - copper-based cable is easy to tap passively (cables radiate EM), somewhat harder to tap actively
  - fiber-optic is extremely difficult to tap
  - radio-based (microwave, satellite) trivial to tap passively, active can be done but is non-trivial
- 802 LANs: “promiscuous” mode allows receiver to get all frames

Cryptography

- Basis for most security in distributed systems is cryptography
  - encoding messages so that only the intended recipient can read
- Two phases:
  - encryption: sender encodes message prior to transmission
  - decryption: recipient decodes (undoes encoding) after reception, prior to reading
- Cryptography is an “ancient science” (2000+ years); computing has advanced the field tremendously
- All methods are variations on:
  - substitution: replacing pieces of text with other pieces
  - transposition: changing the order of text
- Two methods dominate today: single-key and public-key

… cryptography, 2

- Terminology:
  - plaintext (P): message prior to encryption
  - ciphertext (C): message after encryption
  - key: additional input to encryption & decryption functions
  - encryption function notation:
    $$C = E_k(P)$$
  - decryption function notation:
    $$P = D_k(C)$$
  - decrypting an encrypted message:
    $$D_k(E_k(P)) \Rightarrow P$$
    (ie the encryption and decryption functions are inverses of each other)
- Example functions:

  E_k: $C = P \times k$; encryption
  where P is a numeric version of the text and k is the key

  D_k: $P = C \div k$; decryption

Single-key encryption

- Also called private-key, symmetric-key
- Basic idea:
  - sender encrypts, receiver decrypts with the same key, which must be kept secret; “shared secret” system
- Many variations, eg:
  - “one-time pad”: set of keys, different key for each message, used only once
- Best-known is the DES (Data Encryption Standard):
  - developed by IBM, standardized by US government (1977)
  - many allegations about NSA influence on key length (ie ability to “crack”), “backdoor” functions
  - key length reduced from 128 to 56 bits
  - not secure today, but still useful in many circumstances

… single-key encryption, 2

- DES is secure for medium security, but definitely can be broken by “exhaustive enumeration”
  - 56-bit keys means 7×10^{16} keys
  - hardware-based solutions can search all possible keys in hours or days
- Double encryption (56-bit keys twice) sounds good ($2^{112} \cong 5 \times 10^{33}$ keys), but has been shown to be no more effective than single (ref: Tanenbaum)
- Triple encryption (using only two different keys) appears to be secure today

… single-key encryption, 3

- Problems with DES:
  - insecure for determined “crackers”
  - key management
- Key management: consider
  - group of *n* users, each of whom wants to communicate with each of the others
  - needs lots of keys:

$$(n-1)+(n-2)+\ldots+2+1 \equiv \sum_{i=1}^{n} i \equiv \frac{n(n-1)}{2}$$

  - for *n*=1000:

$$\frac{1000(999)}{2} = 500{,}000$$

  - managing 500,000 keys is nightmarish: consider just the initial delivery

Public-key encryption

- Quite different from private-key methods
- Each user *U* has two keys
- Notation:
  - E_U is *U*'s public key, used by everyone to encrypt messages to be sent to *U*
  - D_U is *U*'s private (secret) key, used by *U* to decrypt messages sent to *U*
- E_U and D_U must be chosen together as a pair so that $D_U(E_U(P))$ is *P*
  - essential that D_U cannot be derived or inferred from E_U
  - choosing keys relies on number theory, prime numbers and related mystery
- Public-key encryption solves the key-distribution problem – no keys to distribute!

… public-key encryption, 2

- So, sender S uses the recipient *R*'s public key (eg published on a Web page) and computes $C = E_R(P)$
- R uses its private key to compute $P = D_R(C)$
- The best-known public-key system is called RSA (Rivest-Shamir-Adelman)
  - key selection starts with two 100-digit prime numbers
    - key lengths of 512 to 768 bits suggested
    - see Kurose & Ross Ch 7.2 for an example
  - multiplies them, then solves some modulo-arithmetic equations
  - deriving a private key from a public key is computationally intractable (equivalent to prime-factoring a 200-digit number): no known (published) algorithms exist

… public-key encryption, 3

- Conclusion: RSA is secure
- Benefits compared to DES:
  - trivial key management
  - no shared secrets
  - no prior arrangement required between a sender and receiver
- Problems:
  - generating keys is extremely time-consuming
  - algorithms for encryption are slow – two or three orders slower than DES
- In many applications RSA is used only to exchange DES keys, which are then used to exchange data
- Some of the RSA problems are overcome with other public-key algorithms such as Elliptical Curve
  (ref. www.certicom.com/research/history.html)

Digital signatures

- Basic cryptographic techniques can guarantee secrecy
- What about authentication?
  - DES can be used for authentication, since the key is secret
  - public-key cannot be used as is, since anyone can encrypt a message
- However, can reverse the public–private roles in RSA to achieve an authentication scheme
- Suppose that sender S is sending to R, and R wants to be assured that the sender really is S
  - S uses its private key to encode a short message, *Sig*, such as “I am S”
  - S then appends this signature to a regularly-encoded message for R

… digital signatures, 2

- R receives the message encoded with its public key plus the signature
  - R decrypts the message as usual
  - R decrypts the signature with S's public key
  - if the result of this is intelligible, it must have come from S, since only S knows the private key that goes with S's public key
- Symbolic:
  - S computes $C = E_R(P)$ and $C_{Sig} = D_S(Sig)$; transmits $C \,||\, C_{sig}$
  - R computes $P = D_R(C)$ and $Sig = E_S(C_{Sig})$
- This scheme has a flaw: C_{Sig} can be extracted and appended to other text
  - masquerading
  - solution: put C_{Sig} in P prior to $E_R(P)$

Message digests

- Digital signatures work, but problems:
  - computing RSA encryptions is slow and cumbersome
  - often, we need authentication independent of secrecy (message isn't secret, just need to guarantee sender)
- Message digests solve these problems:
  - function MD(P) computes a unique hash function of P (eg 128-bit number)
  - sender S "signs" the digest by computing $C_{digest} = D_S(MD(P))$
  - S transmits $P \,||\, C_{digest}$
  - receiver R computes MD(P) and $E_S(C_{digest})$
  - the message is authentic and unmodified if the computed $MD(P)$ matches the transmitted one
- MD5 is a standard Internet message-digest function

PKI – public-key infrastructure

- Public-key cryptography is generally more secure than private-key methods
  - there is still one major problem: ensuring that a public key is “correct”, i.e. that is it the proper key of an intended recipient
  - personal contact for key distribution is undesirable (*a priori* contact)
  - publishing eg via a web page or e-mail signature is not reliable
    - how can the authenticity of the web page or e-mail sender be assured?
  - solution: Certificate Authorities (CA) create, store and publish certificates
    - CA is responsible for verifying identity of certificate holder
    - users must trust CAs, eg banks, governments, any trusted brand name
      - certificate is only as good as the CA
    - there are technical standards for CAs, but no standards for being a CA

Using cryptographic security

- Where does cryptography fit into the architecture of a distributed system?
  - OSI: presentation layer, possibly others
  - in the absence of a PL…?
  - don't want unnecessary duplication (eg every application)
  - TCP/IP has some *de facto* standards
- SSL: Secure Sockets Layer
  - fits in as a (semi-)layer between application and transport;
    - creates a secure session
  - application programs use the SSL API instead of the ordinary socket API
    - in the context of the Web, the Web browser is the application program
  - permits clients to authenticate servers by inspecting server digital signature
    - requires server owners to undergo "personal" authentication

… using cryptographic security, 2

- SSL:
  - in a Web context, documents are provided from a different SAP (443 instead of 80)
    - document reference uses “https” instead of “http” to direct to the SSL port
  - SSL can provide client authentication, but in practice does not
- SET: Secure Electronic Transactions
  - developed by the financial sector (credit card organizations) to facilitate secure purchase and other financial transactions
  - restricted to use for finance, not general-purpose session security
  - requires certificates from purchaser, vendor and financial institution
    - browser wallet, merchant server and acquirer gateway
  - SET is quite complex, many protocols

… using cryptographic security, 3

- IPSec: IP Security
  - security at the network layer
  - replacement/significant modification to the IP layer (RFC 2401, 2411 & dozens more)
    - must apply to IPv4 and IPv6 and all future versions
    - hugely complex, requires replacement of every IP layer on the planet before it becomes really useful
  - no transport layer (or higher) can do anything about *IP spoofing:* hosts that falsify the origin of IP datagrams by modifying data in the protocol headers
    - IPSec is designed to prevent this kind of security failure

… using cryptographic security, 4

- E-mail security
  - secure e-mail requires
    - secrecy of content
    - message integrity
    - sender authentication
  - all of these can be accomplished with combinations of encryption, signatures and digests
    - SSL or IPSec could handle some, but not all aspects
  - add-in and complementary products exist for most major e-mail clients
    - still a problem with acquiring trusted keys
    - existing CAs do issue “personal certificates” but they are of questionable value

The world wide web (Web)

- The Web is probably the dominant application in use on the Internet (or e-mail?)
- What exactly is the Web?
  - collection of documents that contain text, images and other data, plus references to other documents (hypertext, linked text, etc)
  - documents are made available by servers
  - documents are viewed by client programs (i.e. Web browsers)

… Web: architecture

- The Web architecture is simple client-server:
  - client establishes connection to a server
  - client issues command to retrieve a document
  - server sends requested document
  - client browser displays retrieved document
- The Web is defined by huge set of protocol standards, most notably:
  - HTTP: HyperText Transfer Protocol governs interaction between client browser and server
  - HTML: HyperText Markup Language defines presentation of documents (ie interaction between browser and user)
    - new versions regularly: Dynamic HTML (DHTML), eXtensible HTML (XHTML), VRML; see www.w3c.org for details

… Web: server

- Conceptually, Web servers are very simple:
  - server listens to TCP port 80 (a TCP/IP formal well-known port)
  - client sends a request for one document
  - if the document is available, the server sends it to the client, and disconnects
  - client-server interactions are *stateless:*
    - each request is a separate TCP connection
    - no information is preserved between successive connections
      - but, “cookies”!
  - HTTP (the protocol defining the interaction) could be considered a client-server middleware protocol
- Historically, much of the work in the Web was done by the clients
- “Server-side” actions are evolving, becoming very complex

HTTP

- HTTP defines client/server interactions
- Simple:
  - client sends commands in plain text
  - minimal command-set:
    - GET: get a page
    - HEAD: get page header
    - PUT: store a page
    - POST: append to a page
      - post is used by clients to send information to the server, eg form fields
    - DELETE: remove a page
    - LINK, UNLINK: create or remove connections between pages
  - response to a command is a status line, plus other information, if applicable (in particular, the contents of the requested page)

… Web: client

- The real power of the Web comes from the versatility of HTML documents
  - HTML is a tagged language that allows actions to be associated with the content
    - a document author can instruct a browser how to display and process the document
    - documents can contain references to other documents; called *links* or *hyperlinks*
    - hyperlinks can be traversed (followed) as easily as moving within the containing document
- HTML is a markup language in the same style as Script, GML, SGML, LaTex, …
  - Web browsers can adjust presentation to local conditions (screen resolution, colour availability, etc.)
  - the antithesis of WYSIWYG
  - creates a conflict between author and browser users

HTML

- HTML defines how to display/process a page
- Important component of this is the definition and syntax of a hyperlink
  - Uniform Resource Locator (URL), defines:
    - the name (identifier) of the page
    - where the page is located
    - how to access the page (how to process its contents)
- URL has three parts:
  scheme*: *// location:port / document-name
- Some existing *schemes:* http, ftp, file, news, gopher, mailto, telnet
  - defines what the document is, and implicitly how to process:
    - for http, get an HTML document
    - for FTP, connect to an FTP server
    - for mailto, start e-mail UA
    - etc

… HTML, 2

- HTML documents can be prepared “by hand” with a text editor
  - most office tools have “save as HTML” feature, which work to varying degrees
  - many HTML authoring systems exist, FrontPage, DreamWeaver, Adobe, etc
    - most have limitations or impose document organization constraints
    - hand-editing is still the method of choice for long-term development
- HMTL forms can solicit user input:
  - forms allow input fields to be defined
  - UI elements such as buttons can invoke an HTTP post operation
  - values can be passed to external programs for processing

Web scripting

- Dynamic HTML (server creating HTML pages “on the fly”) is common
  - reference to a URL causes some form of program code to be run, which determines what content should be returned in response to the URL
- Web browsers are fully programmable
  - give greater control over the operation of the browser
    - defeats the purpose of tagging, to some degree
  - JavaScript is the *de facto* standard language for browser scripting

Client-side and server-side

- Scripting can occur on the server for dynamic page content, and on the client
  - what's the difference?
- Client-side scripting
  - controls the operation of the browser
    - form-field content validation
    - dynamic changes to appearance: roll-over highlighting, bolding
    - menu bars (flyouts, popups)
    - field contents (eg repopulating a list-box)
  - once a page is loaded, client-side scripts can run without a network connection
  - related: *applets* are programs that are invoked via an *applet* tag in the HTML
    - the programs are fetched from a server, or installed at the client in advance
    - applets can only do what a browser lets them do

… client-side and server-side, 2

- Server-side scripting
  - used to cause a server to return differing page contents
  - any situation where a given URL will return different page contents from time to time probably requires server-side scripting
  - several technologies exist:
    - CGI (common gateway interface)
    - Perl: popular on Unix
    - ASP (Active Server Pages) for MS IIS
    - if page content is contained in databases, an *application server* works with the HTTP server to process DB queries and create pages
      - eg Cold Fusion
      - DB access can also be programmed directly with ASP
    - Java servelets

Scripts and SAPs – an editorial comment

- Much of the development with scripting is inappropriate
  - it is an overload of functionality on TCP port 80
    - developers create complex server-side scripts to perform computations and transmit results back to the client
    - this is often coupled with complex client-side scripting to interpret the results
  - this is the precise definition of a distributed client-server application
    - if the application is not inherently a “document-transfer and presentation” application, why use HTTP, HTML and port 80?
    - the answer seems to be that the public believes everything has to run in a Web browser

… editorial, 2

- A better approach would be to use a dedicated SAP for the application
  - the effort required to develop a client-side program and a dedicated server is not much more that developing the client-side and server-side scripts
    - this is especially true for applications that are not easily modelled with the "document fetch and display" paradigm
  - distribution of the client programs would be required, but this can be done via the standard web
    - the time to acquire a program may be significant, but it occurs only once, whereas client-side scripts have to be fetched repeatedly
- Conceptually, there is little difference between a Web-based document that invokes a server-side program, and invoking the program directly via a dedicated SAP