Chp 2. System Models

Road Map

- 2.1. Introduction
- 2.2. Architectural models
  - 2.2.1. software layer
  - 2.2.2. system architectures
  - 2.2.3. variations
  - 2.2.5. design requirements
- 2.3. Fundamental models
  - 2.3.1. interaction model
  - 2.3.2. failure model
  - 2.3.3. security model
2.1. Introduction

- **Architectural Model**
  - Hierarchy/placement of component parts and relationships
  - e.g. Client-Server or Peer-Process models

- **Fundamental/Descriptive Model**
  - Specification/description of common properties and design issues of all kinds of architectural models
  - e.g. Interaction model, failure model, security model
2.2. Architectural models – 2.2.1. software layers

Platform
- The lowest-level hardware and software layers, which provide services to the upper Middleware and Application Layers
  - The design and implementation of the platform is independent of the design and implementation of the upper layers
  - Examples: SunSPARC/SunOS, Intel x86/Solaris, PowerPC/MacOS, Intel x86/Linux, Intel x86/Windows or NT
2.2. Architectural models – 2.2.1. software layers

- Middleware
  - Layer of software, which masks heterogeneity in DS & provides a convenient programming model to application programmers
  - It is represented by processes and objects in a set of computers that interact with each other to implement communication and resource-sharing support for distributed applications
  - Provides building blocks (classes, objects, interfaces, library modules) for constructing DS components
  - It abstracts device-level communication requirements via, e.g., RMI, RPC, request/send, notify, selective or group communication, broadcast, protection/security, data replication/integrity
  - Examples: Sun RPC, Java RMI, CORBA, DCOM

- Provide services used by application programs
  - Infrastructural services: facilities/libraries for naming, security, transactions, persistent storage, event notification
  - Domain-specific services: in the top layer
2.2. Architectural models – 2.2.2. system architectures

- Architectural design has a major impact on performance, reliability, and security.
- **Client-server model** – widely used, servers/clients on different computers provide services to clients/servers on different computers via request/reply messaging
  - servers could also become clients in some services, e.g. web servers are clients of DNS service
2.2. Architectural models –
2.2.2. System architectures

- **Peer-to-peer** – without any distinction between servers and clients, all processes interact and cooperate in servicing requests
  - Client-server model is direct and simple, but scales poorly because of the centralization of service provision and management
  - Hardware capacity and OS functionality of today’s desktops > yesterday’s servers; always-on broadband network connections
  - Collectively store and manage, e.g., the Napster, sharing digital music files
  - Whiteboard application: multiple peer processes interact to modify a shared picture file, interactions and synch. done via middleware layer
2.2. Architectural models – 2.2.3. variations

- **Multiple server** – a DS with multiple, interacting servers responding to parts of a given request in a cooperative manner. Service provision is via the partitioning and distributing of object sets, data replication, (or code migration). E.g., A browser request targeting multiple servers depending on location of resource/data OR replication of data at several servers to speed up request/reply turnaround time, and guarantee availability and fault-tolerance.
2.2. Architectural models – 2.2.3. variations

- **Proxy server and caches** – increasing availability and performance of the service by reducing the load on the wide-area network and web servers
- Caching is frequently used: a cache is a store of recently used data objects, up-to-date copy (validation)
2.2. Architectural models – 2.2.3. variations

- **Mobile code** – A variant of the C-S model, code migration/mobility allows DS objects to be moved to a client (or server) for execution/processing in response to a client request, e.g. migration of an applet to a local browser – avoiding delays and communication overhead
- **push model** – the server initiates the interaction by sending update data to clients’ applets to, say, refresh a stocks web page

a) client request results in the downloading of applet code

![Diagram](image1)

b) client interacts with the applet

![Diagram](image2)
2.2. Architectural models – 2.2.3. variations

- **Mobile agents** – a variant of the C-S model
  - A running program (both code and associated data) that travels from one computer to another in a network carrying out a task on someone’s behalf
  - Reduction in communication cost and time through the replacement of remote invocations with local one
  - E.g., a software installation-agent installing applications on different computers for given hardware-configs; or, price-compare-agent checking variations in prices for a commodity; or, a worm-agent that looks for idle CPU cycles in cluster-computing
2.2. Architectural models – 2.2.3. variations

- **Network computers** – a variant of the C-S model
  - each client computer downloads the OS and application software/code from a server, applications are then run locally and files/OS are managed by server
  - this way, a client-user can migrate from computer to computer and still access the server, and all updates are done at the server side
  - Local disks, if available, are used primarily as cache storage
  - Has low management and hardware costs per computer
2.2. Architectural models – 2.2.3. variations

- **Thin clients** – A variant of the C-S/Network computer model
  - each client computer, instead of downloading the OS and application software/code from a server, supports a layer of software with a window-based user interface and executing application programs on a remote computer
  - Main drawback: in highly interactive graphical applications such as CAD and image processing, delays and communication cost can eclipse any advantages
2.2. Architectural models – 2.2.5. design requirements

- **Performance**
  - Responsiveness: timely and consistent responses
  - Throughput: rate at which computational work is done
  - Load balancing: equal distribution of tasks and resources

- **Quality of service (QoS):** non-functional properties of systems that affect the quality of the service experienced by clients
  - Reliability: related to the failure model
  - Security: related to the security model
  - Performance: related to the interaction model
    - meet timeliness guarantees
    - Time-critical data, e.g. movie services
2.2. Architectural models – 2.2.5. design requirements

- Use of caching and replication (self-reading if interested)
  - Much of challenges (due to performance constraints) have been mitigated by use of caching and replication
  - Techniques for cache updates and cache coherence based on cache coherent protocols (e.g., web-caching protocol, a part of the HTTP cache coherent protocol)
  - Web-caching protocol:
    - Provided by browsers or proxy servers, with periodic update from web-server (dep. on performance requirements) – achieved by browser/proxy sending validation-request to servers,
    - Protocol: Servers set approximate expiry time-to-refresh and sends this time and the server-time (time of sending) as a part of the initial web-data requested/cached. Browser or proxy calculates: if(age = server-time + lengthof-cached-time > expiry-time) then need-request-cache-update – independent of respective clocks or sychronization
2.2. Architectural models – 2.2.5. design requirements

- Dependability issues (self-reading if interested)
  - As related to correctness, security, and fault tolerance
  - Fault tolerance: issues of fail-safe, graceful degradation, partial availability due to redundancy (of both hardware and software), dynamic reconfigurability
  - Architectural redundancy: multiple computers/storage, replicated data/programs/processes, alternative communication paths/switches, e.g., air traffic control systems, with multiple retransmission with Ack protocols
  - Security: models and protocols that ensure protection of sensitive data/programs/processes from attack/abuse, e.g., securing patient records on a server
2.3. Fundamental models

- Specification/description of common properties and design issues of all kinds of architectural models

- Aspects of DS to capture
  - Interaction model: communication and coordination between processes
  - Failure model: defines and classifies faults (hardware, software, network) for fault analysis and system design
  - Security model: defines and classifies possible cases of attack for analysis of threat and system design

  * modularity (of process/services) and openness make DS vulnerable
2.3. Fundamental models – 2.3.1. interaction model

- Two factors affecting interacting processes in DS
  - communication performance is a bottleneck
  - no sense/notion of global clock

- Performance of communication channels
  - Latency: time to transfer/receipt, setup time, OS service overhead
  - Bandwidth: total amount of information that can be transmitted over the network in a given time
  - Jitter: variation in the time taken to deliver a series of messages. Related to multimedia data
2.3. Fundamental models –
2.3.1. interaction model

- Computer clocks and timing events
  - Different clock drifts disallows a notion of ‘perfect global time,’ making timestamps differ in DS
    - Clock drift rate: relative amount that a computer clock differs from a perfect reference clock
  - Dealing with clock drift and clock synch can be accomplished using radio receivers to obtain times from a GPS within 1µsec accuracy and broadcast this time to other computers

- Variants of interaction Model
  - Synchronous DS: assumes lower/upper bounds on exec/comm times, and bounds on clock drift rates
  - Asynchronous DS: no assumptions about time, exec/comm times are unpredictable, and clock drift rates are arbitrary
  - Internet? Multimedia data stream?
2.3. Fundamental models –
2.3.1. interaction model

- The pepperland divisions $A$ and $O$ need to agree on which of them will lead the charge against the $M$, and when the charge will take place.
- In asynchronous pepperland, the messengers are very variable in their speed.
- In synchronous pepperland, the divisions know some useful constraints: every message takes at least $min$ minutes and at most $max$ minutes to arrive.
- The leading division sends a message ’charge!’, then waits for $min$ minutes, then it charges.
- The other division waits for 1 minute after receipt of the message.
- Its charge is guaranteed to be after the leading division’s, but no more than $(max - min + 1)$ after
2.3. Fundamental models – 2.3.1. interaction model

- Event ordering: interested in order of events
  - Messages delivered as such because of independent delays in delivery
  - If X, Y, Z have synch’ed local clocks, user A may see: t1 < t2 < t3
  - Local clocks can’t be synch’ed, hence Lamport proposed logical time for logical ordering of events – without recourse to clocks
2.3. Fundamental models – 2.3.2. failure mode

- Taxonomy of failures
  - Due to processes & communication, under 3 headings of omission failure, arbitrary failure, and timing failure

- Omission Failures
  - Process omission: crash
    - asynchronous: timeout indicate non-responding
    - synchronous: timeout indicate crash, called fail-stop
  - Comm omission: send-omission, receive-omission or channel-omission
2.3. Fundamental models – 2.3.2. failure mode

- Arbitrary / Byzantine failures: worst possible failure semantics
  - mainly process failure, undetectable
- Timing failures: timeliness constraint not met
  - Applicable to synchronous DS only (why?)
- Masking failures: possible to construct reliable services from components that exhibits failure. A service masks a failure either by hiding it altogether (replicas held by multiple servers) or by converting it into a more acceptable type of failure (checksum to mask corrupted messages by converting an arbitrary failure into an omission failure)

- Reliability of one-to-one communication
  - Validity: eventual delivery of queued message from outbound to inbound buffer
  - Integrity: message buffered on receipt is exact copy of the message queued and sent
# 2.3. Fundamental models – 2.3.2. failure mode

<table>
<thead>
<tr>
<th>Class of failure</th>
<th>Affects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail-stop</td>
<td>Process</td>
<td>Process halts and remains halted. Other processes may detect this state.</td>
</tr>
<tr>
<td>Crash</td>
<td>Process</td>
<td>Process halts and remains halted. Other processes may not be able to detect this state.</td>
</tr>
<tr>
<td>Omission</td>
<td>Channel</td>
<td>A message inserted in an outgoing message buffer never arrives at the other end’s incoming message buffer.</td>
</tr>
<tr>
<td>Send-omission</td>
<td>Process</td>
<td>A process completes a <code>send</code>, but the message is not put in its outgoing message buffer.</td>
</tr>
<tr>
<td>Receive-omission</td>
<td>Process</td>
<td>A message is put in a process’s incoming message buffer, but that process does not receive it.</td>
</tr>
<tr>
<td>Arbitrary</td>
<td>Process or channel</td>
<td>Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.</td>
</tr>
</tbody>
</table>
2.3. Fundamental models – 2.3.3. security mode

- Security of a DS can be achieved by
  - securing processes and comm channels
  - protecting objects (resources) they encapsulate against unauthorized access

- Protecting objects: access rights specify who is allowed to perform the op. of an object
  - Principle: authority associated with invocation and result; user or process
2.3. Fundamental models – 2.3.3. security mode

- **Enemy model**: used for the static analysis of security threats, evaluation of risks, and consequences
  - Capable of sending/reading/copying any messages

- Threats to processes: not difficult for the enemy to forge source address
  - Client/server do not have reliable knowledge of each other’s identity

- Threats to communication channels: copying, altering or inserting incorrect data into message streams to deceive or replicate unauthorized transactions (e.g. banking)
2.3. Fundamental models – 2.3.3. security mode

Defeating security threats: solution to the above two threats - secure channels
- Cryptography: science of keeping message secure via encryption/decryption of messages and public/private keys/codes
- Authentication: shared encrypted secret as identification info between server and clients
- Secure Channels: A service layer that uses encryption and authentication techniques on top of communication services. Channels connect a pair of processes
  - reliable identification for both clients and server
  - privacy/integrity of data
  - physical / logical timestamp to prevent replay or reordering of messages
2.3. Fundamental models – 2.3.3. security mode

- Other Threats
  
  - Denial of Service: Bombarding processes in DS with excessive requests/invocations for services beyond server capability – overloading physical resources, or delay of authorized users
  
  - Mobile Code: Trojan horse code, e.g., virus, popularly transmitted in email attachments