System models for distributed systems

INF5040 autumn 2007

lecturer: Frank Eliassen

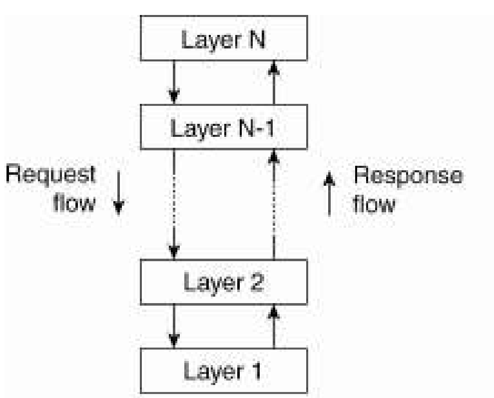
System models

- Motivation
  - illustrate common properties and design choices for distributed system in a single descriptive model
- Two types of models
  - **Architecture models:** define the main components of the system, what their roles are and how they interact, and how they are deployed in a underlying network of computers.
  - **Fundamental models:** formal description of the properties that are common to architecture models. Three fundamental models:
    - interaction models
    - failure models
    - security models

Architectural styles

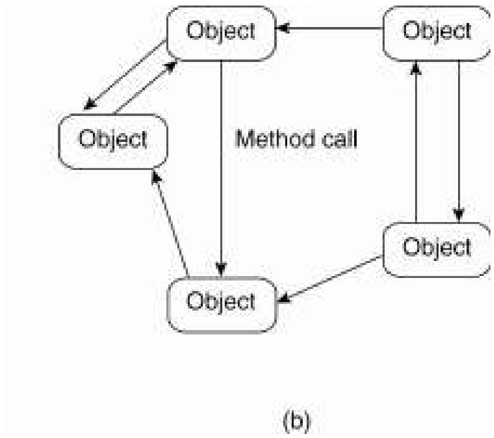
- Concern the logical organization of distributed systems into software components and connectors
  - Components are replaceable units within its environment
  - Connectors are mechanisms that mediate communication, coordination and cooperation among components
- Important architectural styles for DS
  - Layered architectures
  - Object-based architectures
  - Data-centered architecture
  - Event-based architectures

Layered architecture

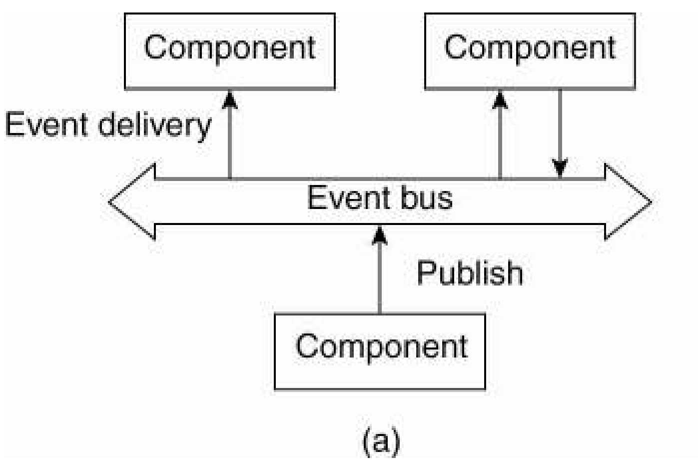


(a)

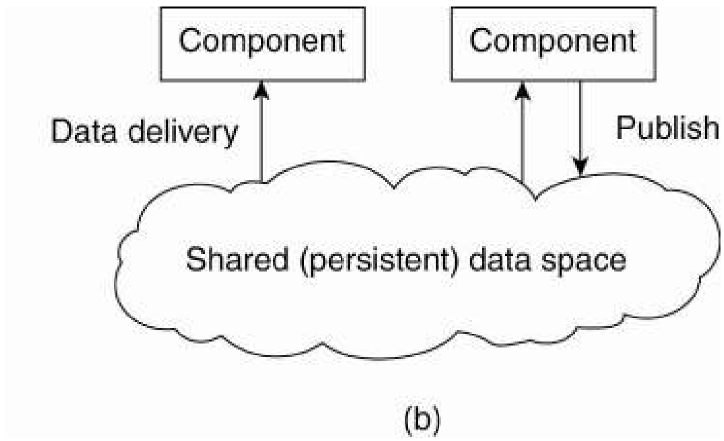
Object-based architecture



Event-based architecture

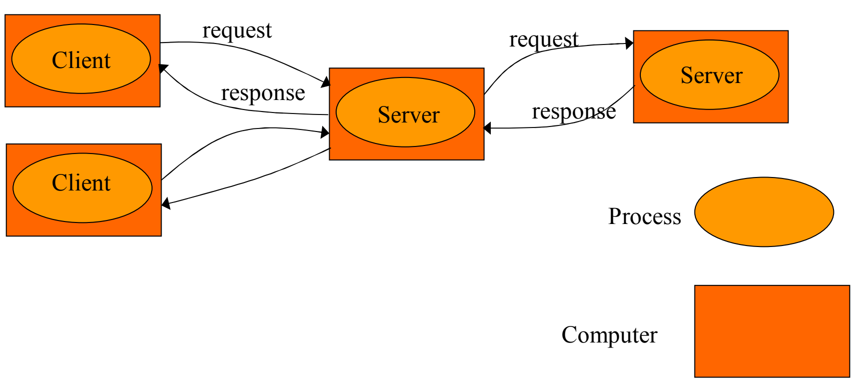


A data-centered architecture: Shared data-spaces



Client-server model

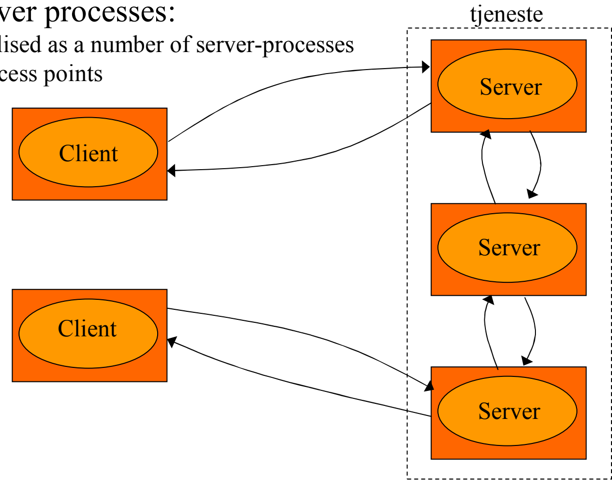
Known for more than 20 years, very popular in DS design



Variants of client-server (1)

Multiple server processes:

- server realised as a number of server-processes
- several access points

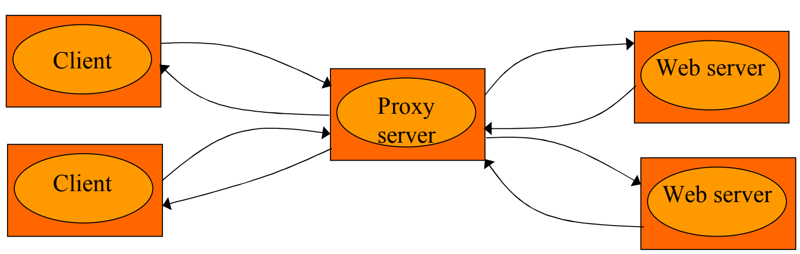


Variants of client-server (2)

Client/server model with proxy-server:

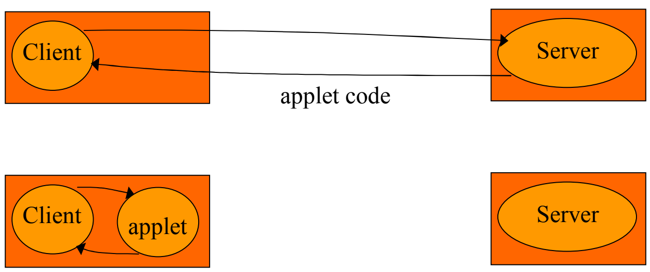
Cache: stores recently-used data objects that are closer to the client than the original objects themselves.

Proxy server: cache that is shared between several clients



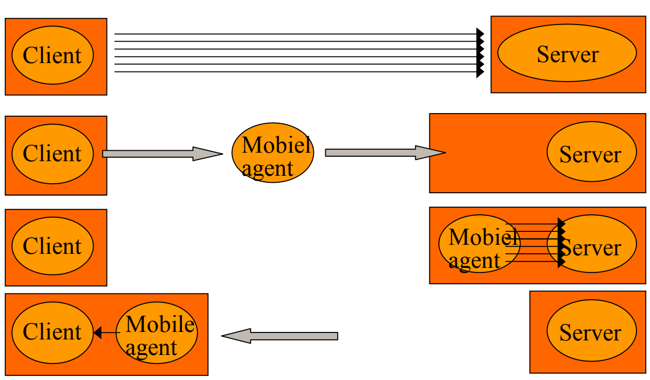
Variants of client-server (3)

Mobile code (applets) . Enables e.g., "push-model": the server invokes the client



Variants of client-server (4)

Mobile agents . Program (code + data) that migrates between computers and executes a task on behalf of someone.

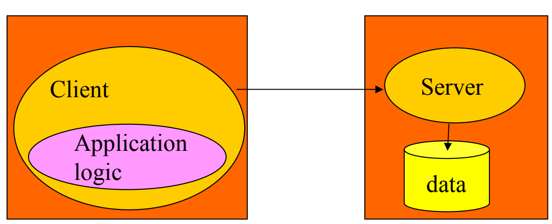


Thin vs thick clients

- "Historically" the trend for application architectures has switched between thick and thin clients
- One variant is network PCs
- Trend today?
  - seems to be thin clients
  - small handheld clients (WAP, PDA, …)
  - ubiquitous computing

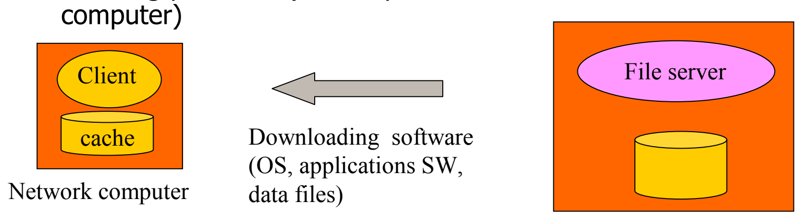
Thick clients

- Identical software installed in all clients
- Allows individual installation and configuration
- Always available, simple licensing
- Problems
  - users can invalidate the installation
  - difficult to keep software "up to date"



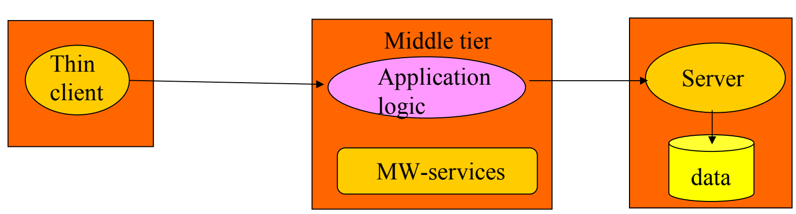
Network-computer/PC

- Network computers attempt to avoid installation problems in client
  - Software is downloaded to client as needed
  - Configuration can be determined in advance (maintained in one place)
- Problems
  - availability (server fails)
  - licensing problems (license pool in stead of license for each computer)



Thin clients

- Thin clients attempt to avoid installation problems in client
  - Thin client: software layer that supports GUI (X.11 server, Web-browser, …)
  - application program executes on remote application server (cf. JEE)
- Problems
  - availability (server fails)
  - licensing problems (licensing pool rather than license for each machine)
  - highly interactive applications



Decentralized architectures

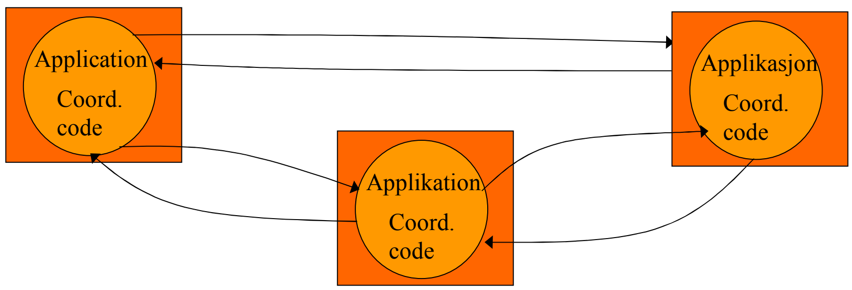
peer-to-peer systems:

Processes have identical roles : each process acts both as a client and a server

Examples:

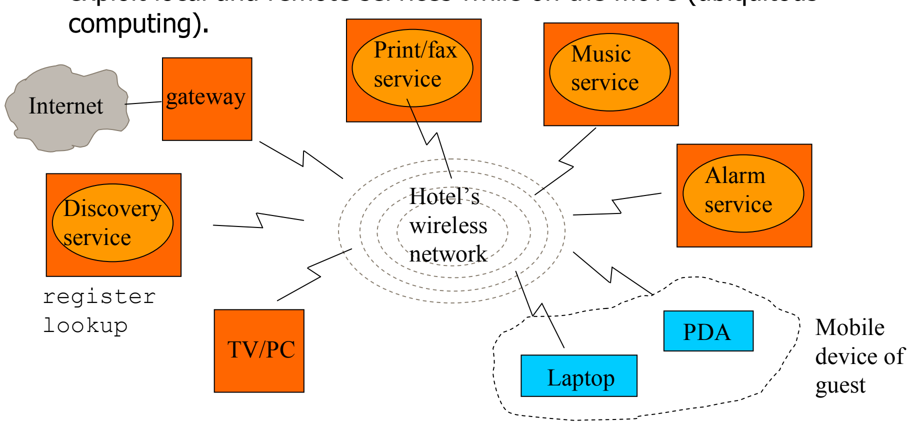
file-sharing systems, cooperation systems (CSCW) as “whiteboard” applications

interactive network based games



Spontaneous networks

- Clients carry mobile devices (laptop, PDA, ….) between different network environments (hotel network, airport network, …) and can exploit local and remote services while on the move (ubiquitous computing).



Fundamental models

- Properties shared by all architecture models
  - communicates by sending messages across a network
  - requirements of performance, reliability, and security
- Fundamental models
  - abstracts over unnecessary details
  - used to address questions like
    - what are the most important entities in the system?
    - how do they interact?
    - what are the characteristics that affect their individual and collective behaviour?
- The purpose of fundamental models
  - to make explicit all relevant assumptions about the system we are modeling
  - to find out what is generally feasible and not feasible under the given assumptions

Fundamental models

- Aspects of distributed systems we want to express
  - Interaction model
    - processes, messages, coordination (synchronisation and ordering)
    - must reflect that messages are subject to delays, and that delay limits exact coordination and maintenance of global time
  - Failure model
    - defines and classifies failures that can occur in a DS
    - basis for analysis of effects of failures and for design of systems that are able to tolerate failures of each type while continuing to run correctly
  - Security model
    - defines and classifies security attacks that can occur in a DS
    - basis for analysis of threats to a system and for design of systems that are able to resist them

Two variants of the interaction model

- Synchronous distributed systems
  - the time to execute each step of a process has known lower and upper bounds
  - each message transmitted over a channel is received within a known bounded time
  - each process has a local clock whose drift rate from real time has a known bound
- Asynchronous distributed systems
  - the time to execute each step of a process can take arbitrarily long
  - each message transmitted over a channel can be received after an arbitrarily long time
  - each process has a local clock whose drift rate from real time can be arbitrarily large

Significance synchronous vs asynchronous DS

- Many coordination problems have a solution in synchronous distributed systems, but not in asynchronous
  - e.g., "The two army problem" or "Agreement in Pepperland" (see [Coulouris])
- Often we assume synchronousness even when the underlying distributed system in essence is asynchronous
  - Internet is in essence asynchronous but we use timeouts in protocols over Internet to detect failures
  - based on estimates of time limits
  - but: design based on time limits that can not be guaranteed, will generally be unreliable

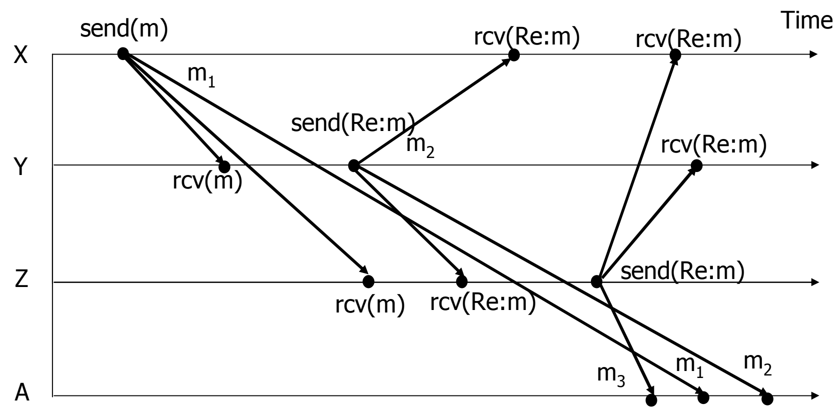
Ordering of events

- distributed coordination protocols have a need for ordering of events in time ("happened before"-relationship)
  - events: sending and receiving messages
  - example: update of replicated data must generally be done in the same order in all replica
  - difficult to use *physical clock* in computers for coordination (e.g.,. clock values in messages)
    - have limited time resolution and ticks with different rates (clock drift)
    - basic properties of message exchange limit the accuracy of the synchronization of clocks in a DS [Lamport 78]
  - possible to describe logical ordering of events even without accurate clocks by using *logical clocks* [Lamport78]

Logical clocks

- Principle
  - If two events happens in the same process, then they occur in the same order as in the process that observed them
  - When a message is transmitted between two processes, the event "send message" will always happen before the event "receive message"
- *Happened-before* relationship
  - is derived by generalizing the two relationships above such that if `x`, `y` and `z` are events and `x` "happened-before" `y` and `y` "happened before" `z`, then `x` "happened-before" `z`
- logicial clocks extends the idea above
  - more later in the course

Example: e-mail exchange



Failure model

- Definition of in which way failures may occur in distributed systems
- provides a basis for understanding the effects of failures
- Definition of failure model of a service enables construction of a *new* service that hides the faulty behaviour of the service it builds upon
  - example: TCP on top of IP
    - TCP: reliable byte-stream service
    - IP: unreliable datagram service

Specification of failure model

- Specification of failure models requires a way to describe failures
- One approach is to classify failure types (Cristian, 1991) (Hadzilacos & Toueg, 1994)
  - Omission failures
  - Arbitrary failures
  - Timing failures
- Model:

Omission failure (1)

- A process or channel fails to perform actions that it is supposed to do

Failure class	Affects	Description
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never the other end's incoming message buffer.

Omission failure (2)

Failure class	Affects	Description
Send-omission	Process	A process completes a *send*-operation, but the message is not put in the outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but the process does not receive it.

Omission failure (3)

- Usual assumption that a server has "**fail-stop**" failure model
  - the server crashes in a "nice" way
    - it halts completely
    - other servers may detect it has failed
  - if the server nevertheless fails in a different way, the software that uses the server, may fail in unpredictable ways
- It is difficult to detect omission failures for processes in an asynchronous system

Arbitrary failure (Byzantine failure)

- Process or channel may exhibit arbitrary behaviour when failing,
  - send/receive arbitrary messges at arbitrary intervals
  - a process may halt or perform "faulty" steps
  - a process may omit to respond now and then
- By adopting a byzantine failure model, we can attempt to make systems that are "ultra-reliable" (handles HW failures, and provide guaranteed response times)
  - control systems in air planes
  - patient monitoring systems
  - robot control systems
  - control systems for nuclear power plants

Timing failure

- Applicable in synchronous distributed systems
  - responses that are not available to clients in a specified time interval
  - timing guarantees requires guaranteed access to resources when they are needed
- Examples:
  - control and monitoring systems, multimedia systems

Failure class	Påvirker	Beskrivelse
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time
Performance	Process	Process exceeds the bounds on the interval between two processing steps
Performance	Channel	A message's transmission takes longer than the stated bounds

Summary

- Two types of system models
  - **Arcitecture models**: defines the components of the system, the way they interact, and the way the are deployed in a network of computers
    - client-server models (many variants)
    - peer processes (P2P)
    - spontaneous networks (mobility)
  - **Fundamental models**: formal description of the properties that are common to all architecture models
    - interaction models
    - failure models
    - security models (not covered in this course, but see e.g., INF3190)