Introduksjon til Distribuerte System (DS)

INF5040 høst 2005

foreleser: Olav Lysne

Hva er et distribuert system?

- Definisjon [Coulouris & Emmerich]
  - Et distribuert system består av maskinvare- og programvare-komponenter lokalisert i et nettverk av datamaskiner som kommuniserer og koordinerer sine aksjoner kun ved å sende meldinger.

- Definisjon [Lamport]
  - Et distribuert system er et system som hindrer deg i å få gjort noe arbeid når en maskin du aldri har hørt om før, feiler.

Konsekvenser av distribuerte system

- Komponenter feiler uavhengig av hverandre
  - "delvis feiling" & ufullstendig informasjon
- Upålitelig kommunikasjon
  - Tap av forbindelse og meldinger. Bitfeil i meldinger.
- Usikker kommunikasjon
  - Mulighet for uautorisert avlytting og modifikasjon av meldinger
- Kostbar kommunikasjon
  - Kommunikasjon mellom datamaskiner har vanligvis lavere båndbredde, høyere latenstid, og koster mer, enn mellom uavhengige prosesser i samme maskin.
- Samtidighet
  - komponenter eksekverer i samtidige prosesser som leser og oppdaterer delte ressurser. Krever koordinering (samtidighetskontroll)
- Ingen global klokke
  - vanskeliggjør tett koordinering

Krav som leder til distribuerte system

- ressursdeling
  - muligheten til å benytte tilgjengelige ressurser hvor som helst
- åpenhet
  - et åpent system kan utvides og forbedres inkrementelt
- skalerbarhet
  - betjene flere brukere, gi kortere svartider
- feiltoleranse
  - opprettholde tilgjengelighet selv i tilfeller der komponenter har liten pålitelighet
- heterogenitet
  - nettverk og maskinvare, operativsystem, programmeringsspråk, implementasjon av forskjellige utviklere

Ressursdeling

- Muligheten til å benytte tilgjengelig maskinvare, programvare eller data hvor som helst i systemet
- Ressursforvaltere (managers) kontrollerer aksess, tilbyr skjema for navngiving, og kontrollerer samtidighet
- En ressursforvalter er en programvaremodul som forvalter en ressurs av en bestemt type.
- Ressursdelingsmodell beskriver hvordan
  - ressurser gjøres tilgjengelig
  - ressurser kan brukes
  - tjenesteyter og bruker interagerer med hverandre

Ressursdelingsmodeller

- Klient-tjener ressursmodell
  - Tjenerprosesser opptrer som ressursforvaltere, og tilbyr tjenester (samling prosedyrer).
  - Klientprosesser sender forespørsler til tjenere
- Objekt-basert ressursmodell
  - Enhver entitet innen en prosess modelleres som et objekt med et meldingsbasert grensesnitt som gir adgang til dets operasjoner.
  - Enhver delt ressurs modelleres som et objekt

Åpenhet

- Et åpent DS kan utvides og forbedres inkrementelt
- Krever en uniform interprosessmekanisme og at komponentgrensesnitt offentliggjøres (f.eks. gjenstand for standardisering)
  - IETF RFC: Protokollspesifikasjon (www.ietf.org)
  - OMG: grensesnittspesifikasjoner m.m. (www.omg.org)
- Nye komponenter må kunne integreres med (virke sammen med) eksisterende komponenter

Samtidighet

- Komponenter i DS eksekverer i samtidige prosesser
- Komponenter aksesserer og oppdaterer delte ressurser (f.eks. variable, databaser, device drivere)
- Integriteten til systemet kan brytes hvis samtidig oppdatering ikke koordineres
  - tapte oppdateringer
  - inkonsistent analyse
- Bevaring av integritet krever samtidighetskontroll hvor samtidig aksess til samme ressurs synkroniseres

Skalerbarhet

- Et system er skalerbart hvis det forblir effektivt når det er en betydelig økning i mengden ressurser og antall brukere.
  - Internett: antall brukere og tjenester har vokst enormt
- Skalerbarhet betegner altså et systems egnethet til å handtere en økende last i fremtiden
- Krav om skalerbarhet leder ofte til en distribuert systemarkitektur (flere maskiner)

Skalerbarhet : utfordringer

- Kontrollere kostnader (ressurser)
  - Et system med n brukere er ressurs-skalerbarhet dersom antall ressurser som kreves for å underholde dem er høyst O(*n*)
- Kontrollere ytelsestap (når mengden data øker)
  - Et system er ytelses-skalerbart dersom tiden det tar å aksessere hierarkisk ordnede data er høyst O(log *n*) der *n* er mengden data
- Hindre at systemet slipper opp for programvareressurser:
  - Dimensjonere datastrukturer o.l. slik at systemet kan handtere fremtidige krav (vanskelig - jfr IP adresser)
- Unngå ytelsesflaskehalser
  - krever desentraliserte algoritmer (partisjonering, caching og replikering)

Feilhandtering

- Maskinvare, programvare og nettverk feiler!!
- DS må opprettholde tilgjengelighet selv i tilfeller der maskinvare/programvare/nettverk har liten pålitelighet
- Feil i distributerte system er partiell
  - gjør feilhandtering spesielt vanskelig
- Mange teknikker for å handtere feil
  - Deteksjon av feil (sjekksum o.l)
  - Maskering av feil (retransmisjon i protokoller, replikering ...)
  - Tolerere feil (som i web-lesere)
  - Gjenoppretting (“recovery”)
  - Redundans (replikere tjenester på feil-uavhengige måter)

Heterogenitet

- Variasjon og forskjeller som må handteres
  - nettverk
    - Internett-protokollen er implementert over mange ulike nettverk
  - maskinvare
    - forskjeller i data representasjon til datatyper på forskjellige prosessorer
  - operativsystem
    - API til samme protokoll og tjeneste varierer
  - programmeringsspråk
    - forskjellig representasjon av tegnsett og datastrukturer
  - implementasjon av forskjellige utviklere
    - sikre at ulike programmer kan kommunisere
      - krever enighet om en rekke ting (jfr standarder)

Transparens

- Transparens skjuler konsekvensene av distribusjon
- Transparens har forskjellige dimensjoner [ODP]
- Disse representerer ulike egenskaper et distribuert system kan ha (målestokk for å vurdere design av et system)

Aksesstransparens

- Muliggjør at lokale og fjerne ressurser/komponenter kan aksesseres ved bruk av identiske operasjoner
- Eksempel: Filsystem-operasjoner i NFS
- Eksempel: Navigering i www
- Eksempel: SQL-spørringer i distribuerte databaser

- Komponenter som ikke har transparent aksess kan ikke enkelt flyttes fra en maskin til en annen.

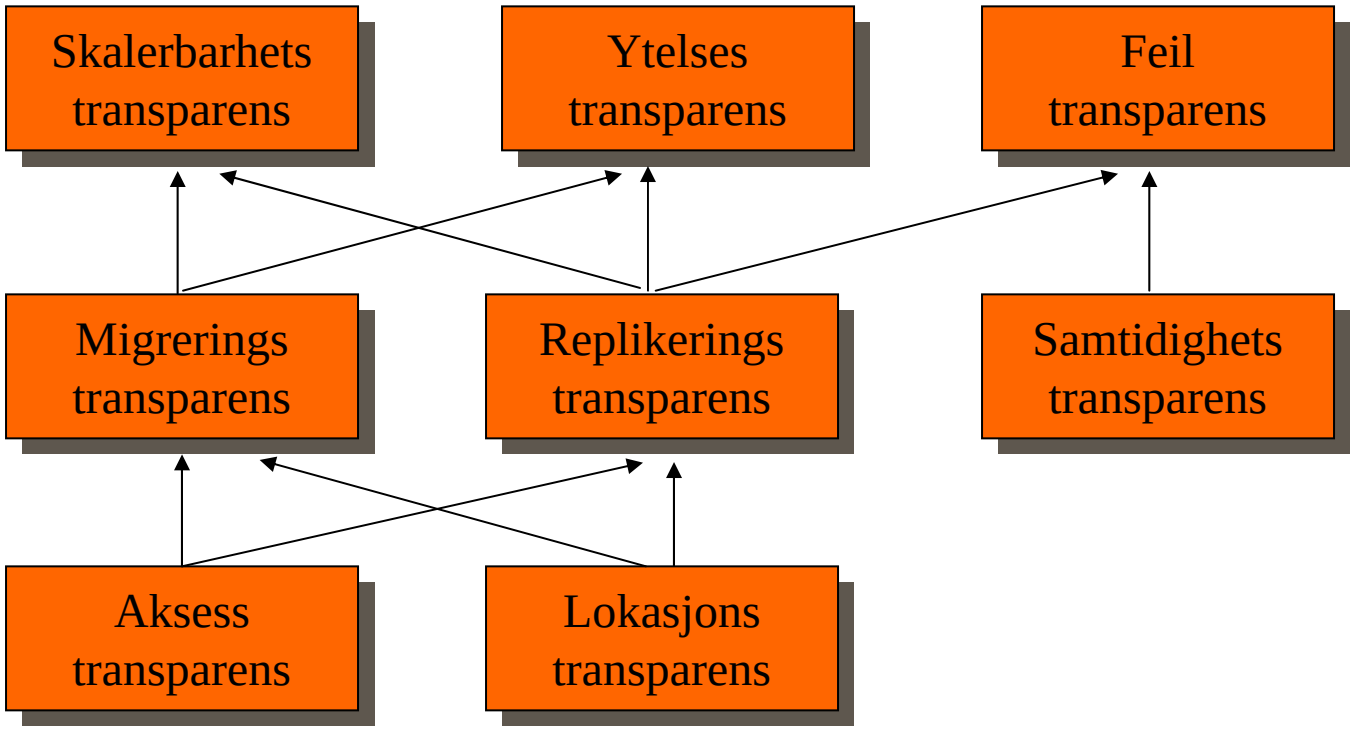
Lokasjonstransparens

- Muliggjør at ressurser/komponenter kan aksesseres uten kunnskap om deres lokasjon
- Eksempel: Filsystem-operasjoner i NFS
- Eksempel: Websider (URLer) i www
- Eksempel: Tabeller i distribuerte databaser

Andre transparensdimensjoner

- Samtidighetstransparens
- Replikeringstransparens
- Feiltransparens
- Migreringstransparens
- Ytelsestransparens
- Skaleringstransparens

Distribusjonstransparens



Mål for distribuert mellomvare

- Høste fordelene ved distribuerte system
- Skjule uønskede konsekvenser av distribusjon (transparens)
- Realisere portabilitet og interoperabilitet

Distribuerte
applikasjoner
og tjenester

PlattformUavhengig grensesitt

DISTRIBUERT MELLOMVARE

PlattformAvhengig grensesitt

Plattform 1 | Plattform 2 | . . . | Plattform n

- transaksjonsorientert (ODTP XA)
- meldingsorientert (IBM MQSeries)
- fjerne prosedyrekall (X/Open DCE)
- objekt-basert (CORBA, COM, Java)

Oppsummering

- Distribuert system:
  - maskinvare- og programvare-komponenter lokalisert i et nettverk av datamaskiner som kommuniserer og koordinerer sine aksjoner kun ved å sende meldinger
- Konsekvenser av distribuerte system
  - Komponenter feiler uavhengig av hverandre
  - Usikker kommunikasjon (sikkerhet)
  - Ingen global klokke
- Krav som ressursdeling, åpenhet, skalerbarhet, feiltoleranse og heterogenitet kan tilfredstilles av distribuerte system
- Mål for distribuert mellomvare
  - Høste potensielle fordeler av distribuerte system uten å måtte betale for alle dens utfordringer og problemer (transparens)

Design av distribuerte objekter

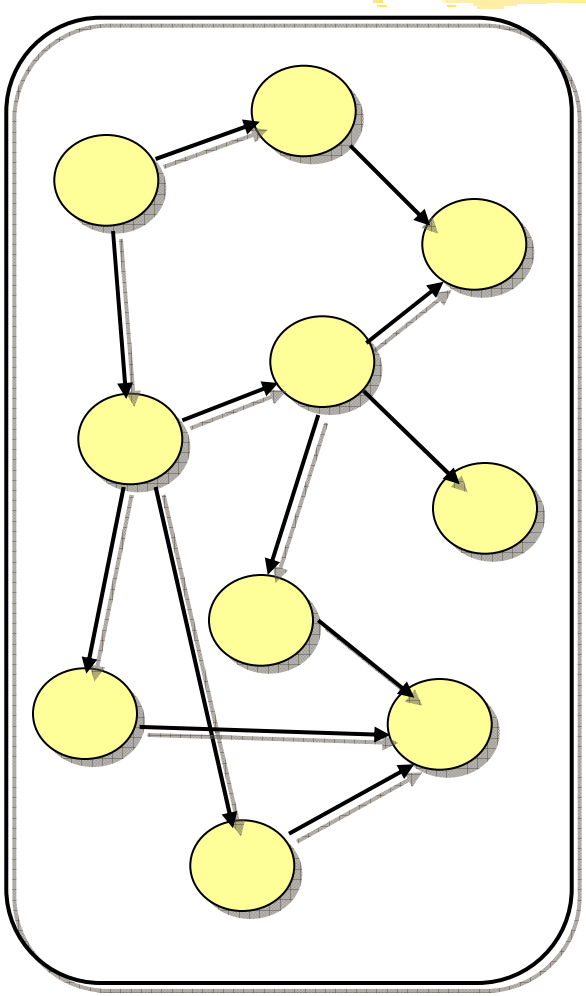
INF5040 høst 2005

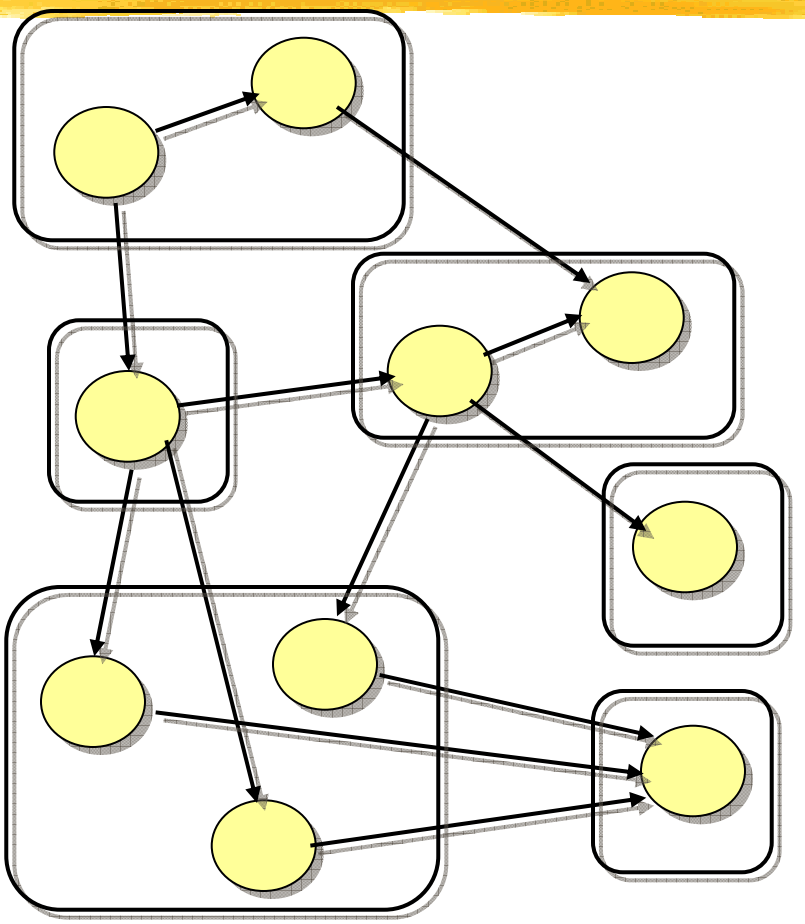
foreleser: Olav Lysne

Design av distribuerte objekter

- Mange har erfaring med design av lokale objekter som alle er lokalisert i kjøretidsomgivelsen til et OO programmeringsspråk
- Design av distribuerte objekter er forskjellig!
- I det følgende: Diskutere de viktigste forskjellene

Design av distribuerte objekter





Lokale vs distribuerte objekter

- Referanser
- Aktivisering/deaktivisering
- Migrering
- Persistens
- Latenstid for metodeanrop
- Samtidighet
- Kommunikasjon
- Sikkerhet
- Mange fallgruber lurer her!!

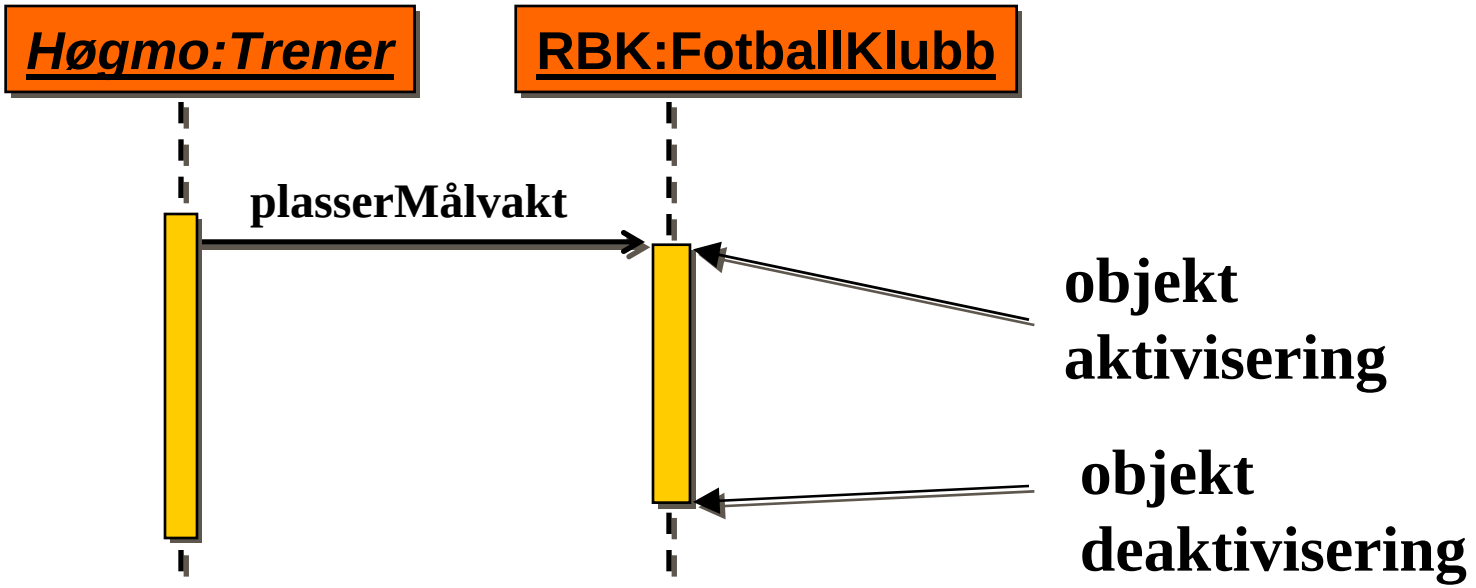
Objektreferanser

- Referanser til objekter i OOPS er vanligvis pekere til primærlageradresser
  - noen ganger kan de endres til referanser (C++)
  - andre ganger ikke (Smalltalk, Java)
- Referanser til distribuerte objekter er mer kompleks
  - lokasjonsinformasjon
  - sikkerhetsinformasjon
  - referanse til objekttype
- Referanse til distribuerte objekt er større (f.eks. 350 byte i Orbix)

Aktivisering/deaktivisering

- Objekter i OOPS er i primærlageret i tiden fra de opprettes til de slettes
- Dette passer ikke alltid like bra for distribuerte objekter
  - antall objekter
  - objekter kan brukes over lang tid
  - noen tjenermaskiner må tas ned av og til uten at applikasjonene stoppes
- Distribuerte objektimplementasjoner blir
  - lest inn i primærlageret (aktivisering)
  - fjernet fra primærlageret (deaktivisering)

Aktivisering/deaktivisering



Aktivisering/deaktivisering

- Spørsmål som oppstår:
  - "oppbevaringssted" for implementasjoner ("repository")
  - assosiasjon mellom objekter og prosesser
  - eksplisitt vs implisitt aktivisering
  - når deaktivisere objekter?
  - hvordan handtere samtidige anrop
- Hvem bestemmer svarene på spørsmålene over?
  - Designer?
  - Programmerer?
  - Administrator?

Persistens

- Tilstandsløse vs tilstandsfulle objekter
- Tilstandsfulle objekter må lagre sin tilstand mellom
  - objekt-deaktivisering og
  - objekt-aktivisering
- i et persistent lager
- Kan oppnås bl.a. ved
  - lage ekstern representasjon for filsystem
  - avbilde til relasjonsdatabase
  - objekt database
- Avgjøres ved objekt design

Objekters livsløp

- Objekter i OOPS lever i én virtuell maskin
- Distribuerte objekter kan opprettes på forskjellige maskiner
- Distributerte objekter kan kopieres eller flyttes (migreres) fra en maskin til en annen
- Fjerning av objekter ved “søppelsamling” er vanskelig i en distribuert omgivelse
  - Java RMI: “reference counting”
  - Jini: “leases”
- Livsløp må vurderes ved design av distribuerte objekter

Latenstid til anrop

- Å utføre et lokalt metodeanrop krever noen hundre nanosekunder
- Et fjernt metodeanrop krever mellom 0.1 og 10 millisekunder, eller mer
- $\Rightarrow$ grensesnitt til distribuerte objekter må konstrueres på en slik måte at
  - metoder utfører større prosesseringsoppgaver
  - de ikke krever svært hyppige anrop

Parallellitet

- Utførelse av objekter i OOPS
  - sekvensiell
  - samtidig (concurrent) m/flere tråder
- Distribuerte objekter eksekverer i parallell
- Kan brukes til å akselerere beregningene

Kommunikasjon

- Metodekall i OOPS er synkrone
- Alternativer for distribuerte objekter:
  - synkrone kall (“synchronous requests)
  - enveis-kall (“oneway requests”)
  - utsatt synkrone kall (“deferred synchronous requests”)
  - asynkrone kall (“asynchronous requests”)
- Hvem avgjør for hvert kall?
  - designer av tjeneren?
  - designer av klienten?
- Hvordan dokumentere?

Sikkerhet

- Sikkerhet i OO applikasjoner kan handteres på sesjonsnivå
- Objekter i OOPS trenger ikke skrives på en bestemt måte
- For distribuerte objekter
  - Hvem utsteder metodekallet?
  - Hvordan kan vi vite at klienten er den han hevder å være?
  - Hvordan kan vi avgjøre om vi skal innvilge klienten retten til å bruke tjenesten?
  - Hvordan kan vi bevise at vi har levert tjenesten for seinere å kunne kreve betaling for bruk av tjenesten?

Oppsummering

- Design av distribuerte objekter er forskjellig fra design av program der alle objekter er i samme prosess
- Mange fallgruber!!