Systemutvikling – Fra krav til modellering av objekter

Høgskolen i Oslo og Akershus
Avdeling for ingeniørutdanning
08 sept. 2014

Yngve Lindsjørn- yngve.lindsjorn@hioa.no

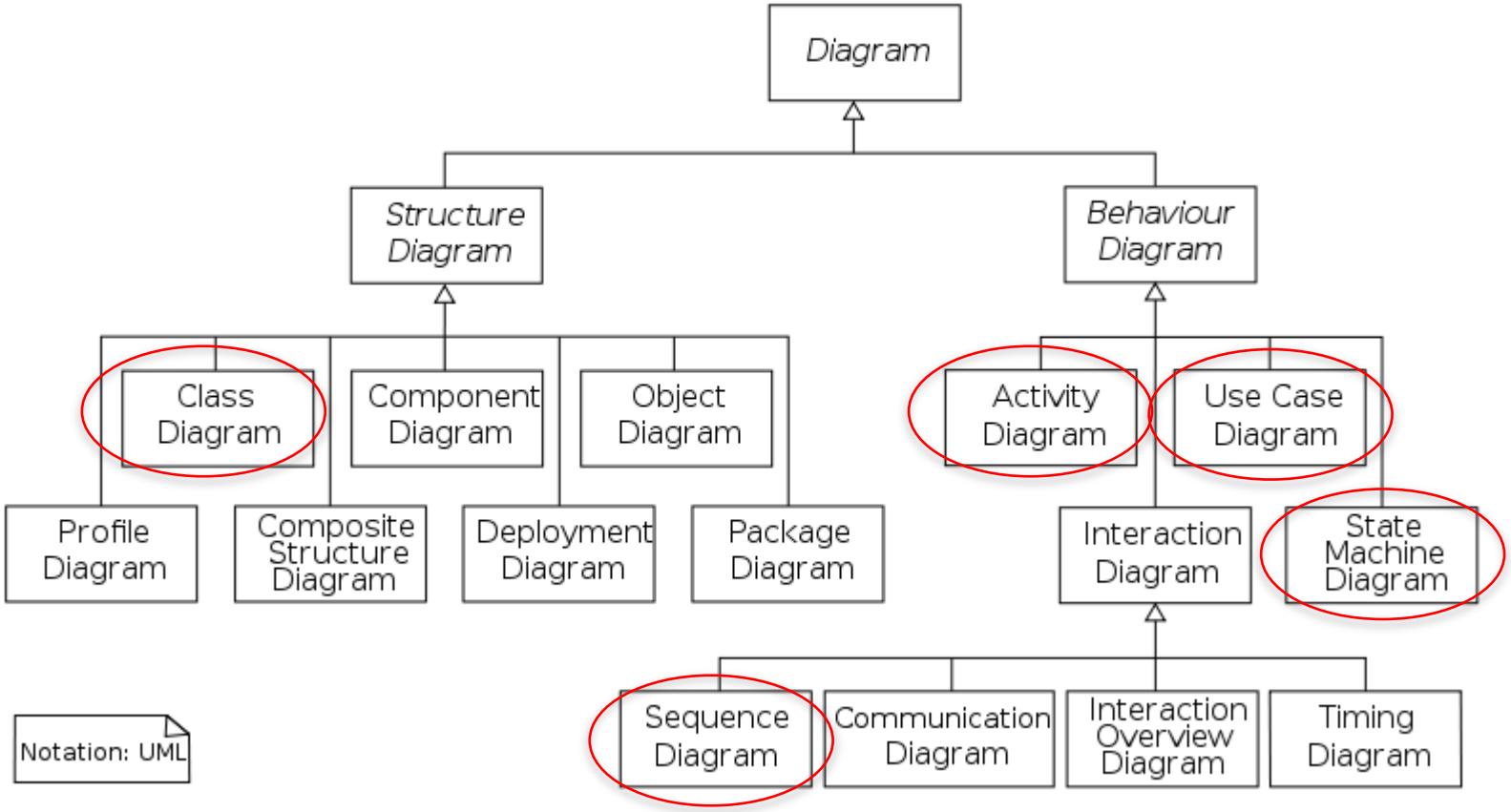
Temaer i dagens forelesning

- System-modellering og systemperspektiv
- Objektorientert perspektiv
- UML diagrammer
  - Bruksmønster (Use-Case)
  - Sekvensdiagram
  - Klassediagram
  - Aktivitetsdiagram
  - Tilstandsdiagram
- Modelldrevet tilnærming

Systemmodellering

- Det sentrale med systemmodellering er å utvikle abstrakte modeller av et system, der hver modell representerer ulike perspektiver av systemet
- Systemmodellering er viktig for å forstå funksjonaliteten i et system og modellene brukes til å kommunisere med kundene og til dokumentasjon
- Grafisk notasjon er viktig, og er i hovedsak basert på notasjoner gitt i UML (Unified Modeling Language)

UML - diagrammer



Kilde: http://en.wikipedia.org/wiki/Unified_Modeling_Language#Structure_diagrams

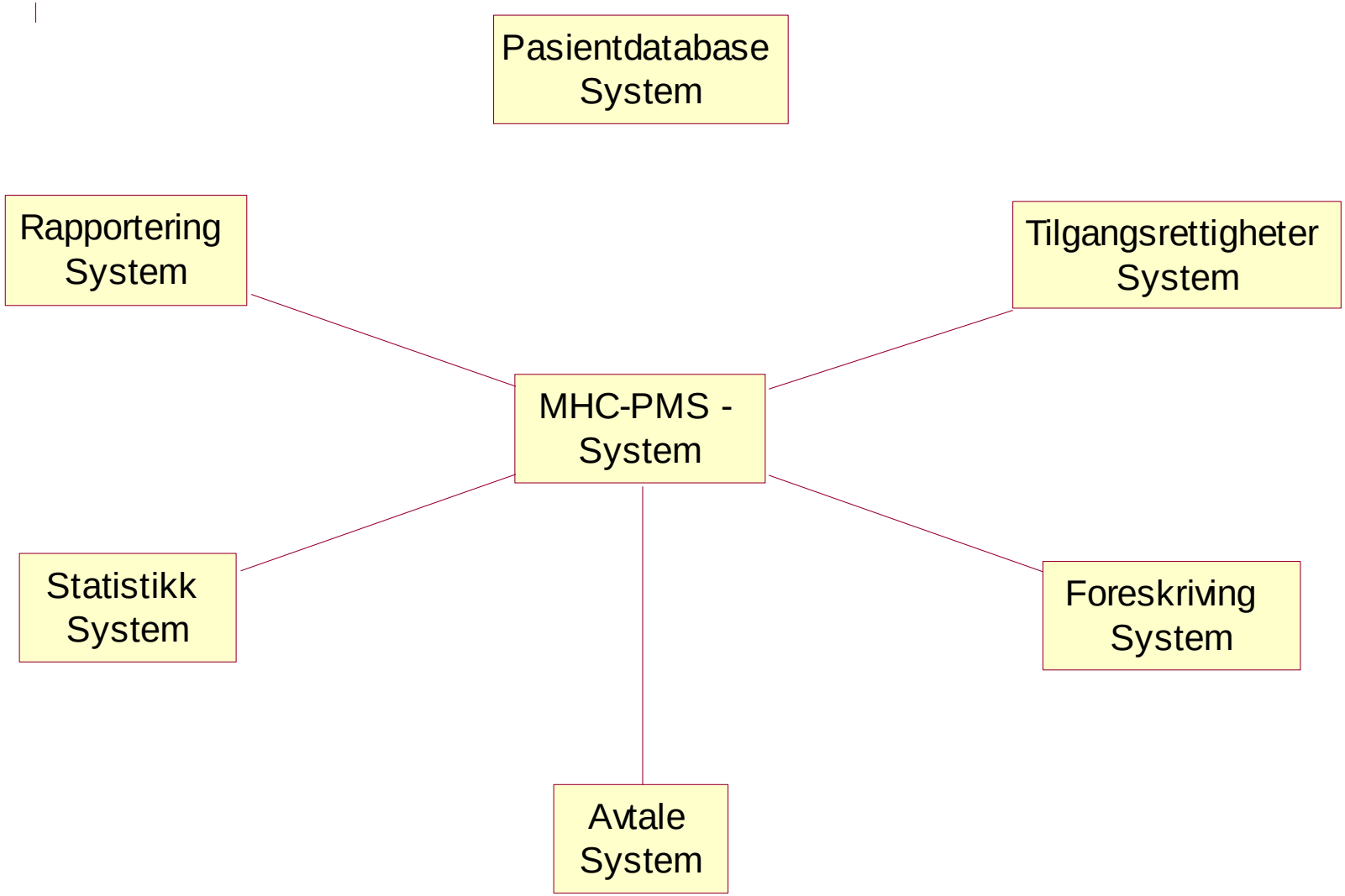
Modeller

- En modell er en abstrakt oversikt av et system
  - Ser bort fra alle detaljene
- Systemmodeller blir laget for å vise et systems
  - kontekst,
  - interaksjon,
  - struktur og
  - adferd (”behavior”).

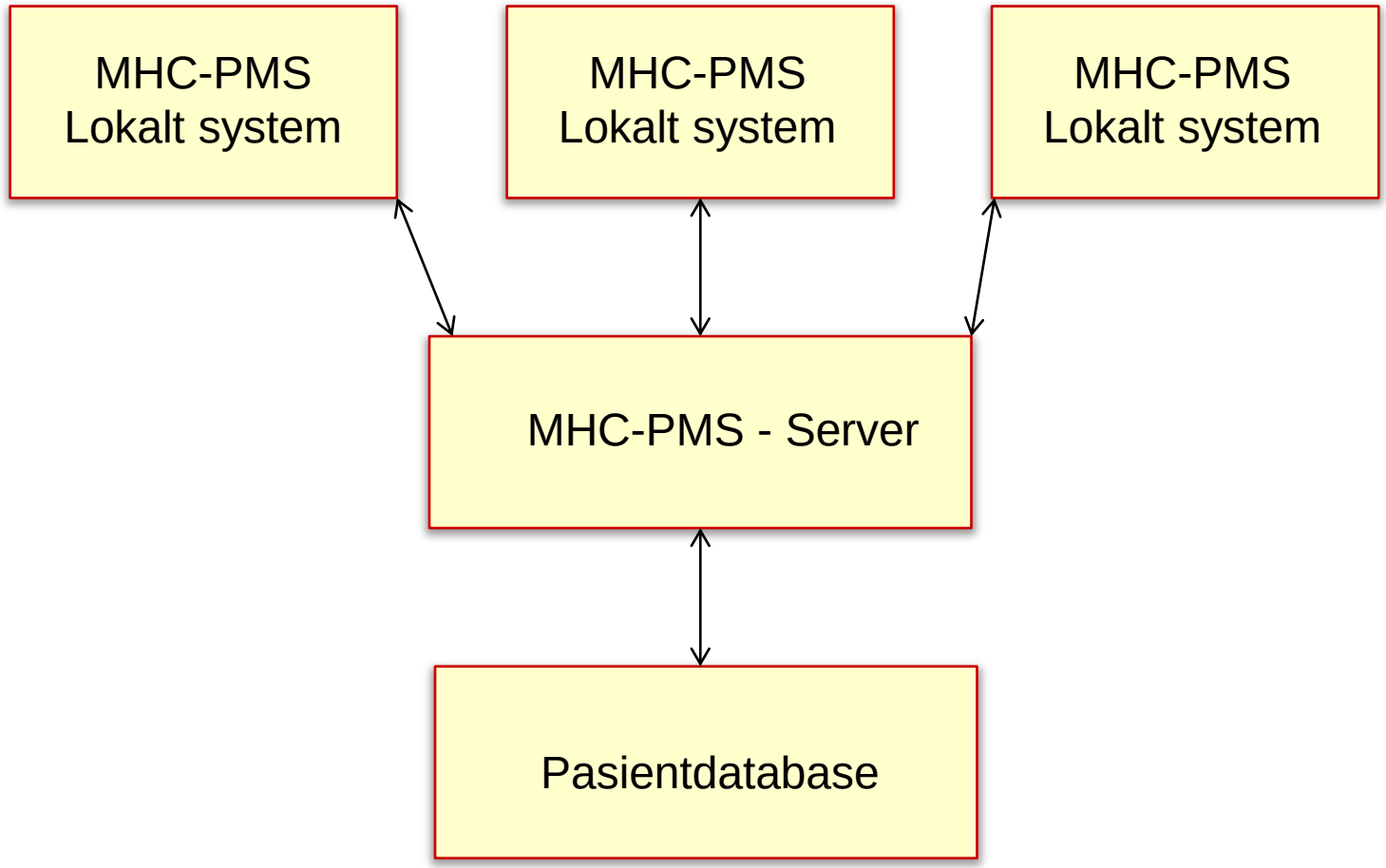
Kontekstmodeller

- Kontekstmodeller viser hvordan et system relaterer seg til andre systemer og prosesser

Kontekstmodell – MHC-PMS



Medical Health Care – Patient Management System



Beskrivelse av MHC-PMS: Lærebok s. 18 (kapittel 1.3.2)

Systemperspektiver

- 4 ulike perspektiver
  - **Eksternt perspektiv**, der du modellerer konteksten til systemet
  - **Interaksjonsperspektiv**, der du modellerer interaksjonen mellom et system og omgivelsene, eller mellom komponentene i et system
  - **Strukturelt perspektiv**, der du modellerer organisasjonen systemet inngår i eller datastrukturen som brukes av systemet
  - **Adferdsperspektiv**, der du modellerer systemets adferd og hvordan systemet reagerer på hendelser

Grafiske modeller

- Et godt hjelpemiddel i diskusjonen om systemet
  - Ikke komplette eller ukorrekte modeller kan være OK så lenge formålet er å bidra til diskusjon

Grafiske modeller

- Brukes ofte som en sentral del i dokumentasjon av et eksisterende system
  - Modeller bør representere systemet korrekt, men trenger ikke være komplett

Grafiske modeller

- En detaljert systembeskrivelse kan brukes til å implementere systemet
  - Modellen må både være korrekt og komplett

Prosessperspektiv

- Kontekstmodeller viser andre systemer og prosesser i omgivelsene, ikke hvordan systemet som utvikles brukes i omgivelsene
- Prosessmodeller viser hvordan systemet som utvikles, brukes i en videre sammenheng
- UML aktivitetsdiagram brukes til å definere forretningsprosessmodeller

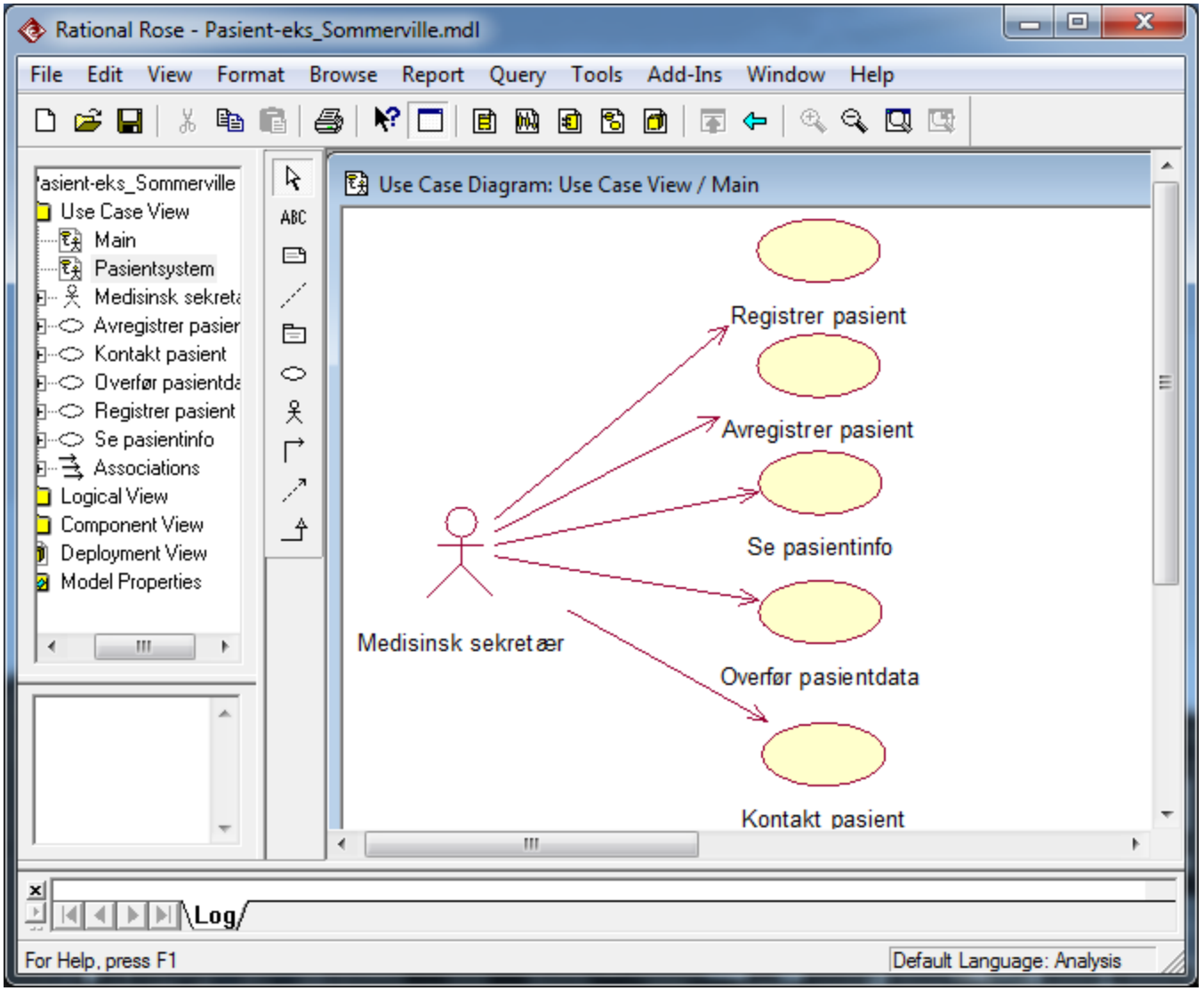
Interaksjonsmodeller

- Modeller av bruker-interaksjoner er viktig for å identifisere brukerkrav
- Bruksmønstre og sekvensdiagrammer er sentrale teknikker og diagrammer som brukes i interaksjons-modeller

Bruksmønster- og sekvensdiagram

- Bruksmønster diagrammer (Use Case diagrams) og sekvensdiagrammer brukes til å beskrive interaksjonen mellom brukere og systemer.
- Bruksmønstre beskriver interaksjonen mellom et system og eksterne aktører
- Sekvensdiagrammer legger til mer informasjon for hvert bruksmønster og viser også interaksjon mellom objekter

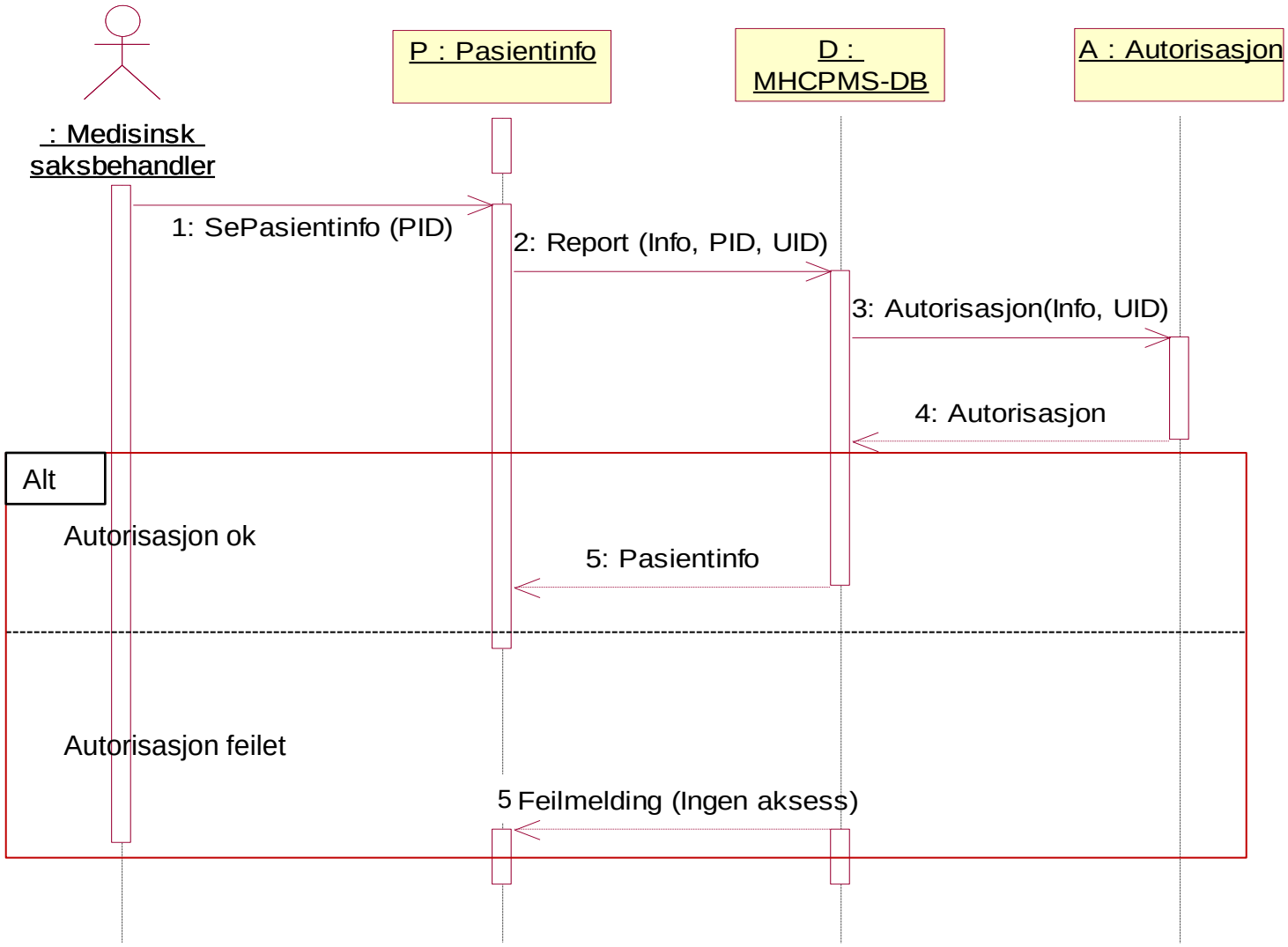
Bruksmønstre i MHC-PMS som involverer medisinsk saksbehandler



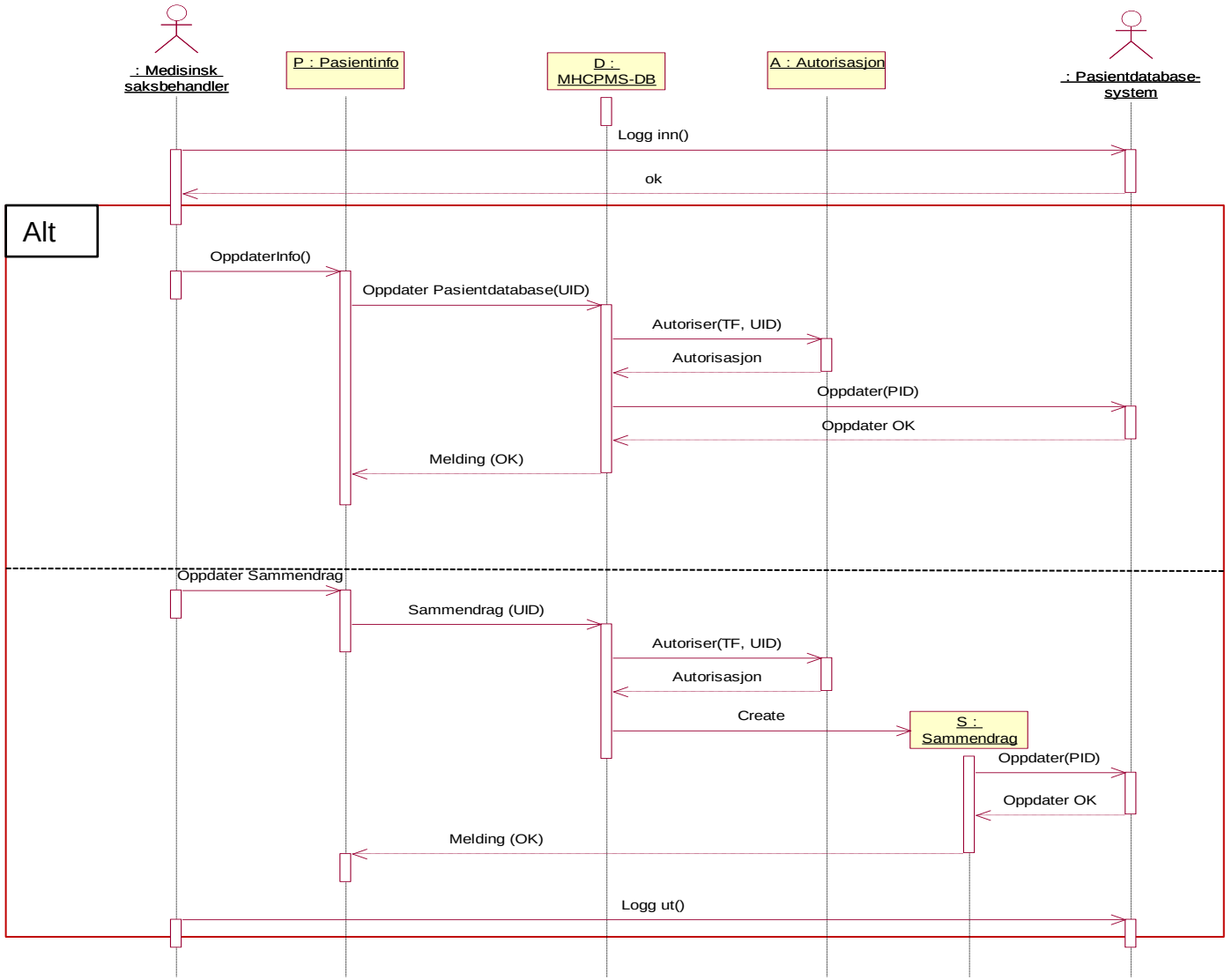
Sekvensdiagrammer

- Sekvensdiagrammer brukes til å modellere interaksjonen mellom aktørene og objektene i et system
- Et sekvensdiagram viser sekvensen av interaksjoner som skjer under et gitt bruksmønster (spesielt for hovedløp)
- Et sekvensdiagram angir hvordan metodene i objektene anvendes

Sekvensdiagram “Se pasientinfo ”



Sekvensdiagram for “overfør pasientdata”



Strukturelle modeller

- Strukturelle modeller av et programvaresystem viser organiseringen av systemet ved å vise systemets komponenter og relasjonen mellom komponentene
- Klassediagrammer brukes til å definere den statiske strukturen av klasser og deres assosiasjoner

Klassediagrammer

- Klassediagrammer blir brukt i utviklingen av systemmodeller for å vise klasser i systemet og assosiasjoner mellom disse klassene
- En klasse kan bli sett på som en generell definisjon (mønster) av objekter som er instanser av klassen
- En assosiasjon mellom to klasser angir at det er en forbindelse mellom disse klassene
- Ved utvikling av modeller i den tidlige fasen av systemutviklingsprosessen, vil objekter som regel representere (være en modell av) noe i den virkelige verden, som en pasient, en doktor, en bil eller en bankkonto.

Eksempel

class Bank

I en bank skal vi kunne legge inn en ny kunde, fjerne en kunde, sette inn penger på en kundes konto og finne forvaltningskapitalen til banken (summen av alle beløpene på alle kontoene)

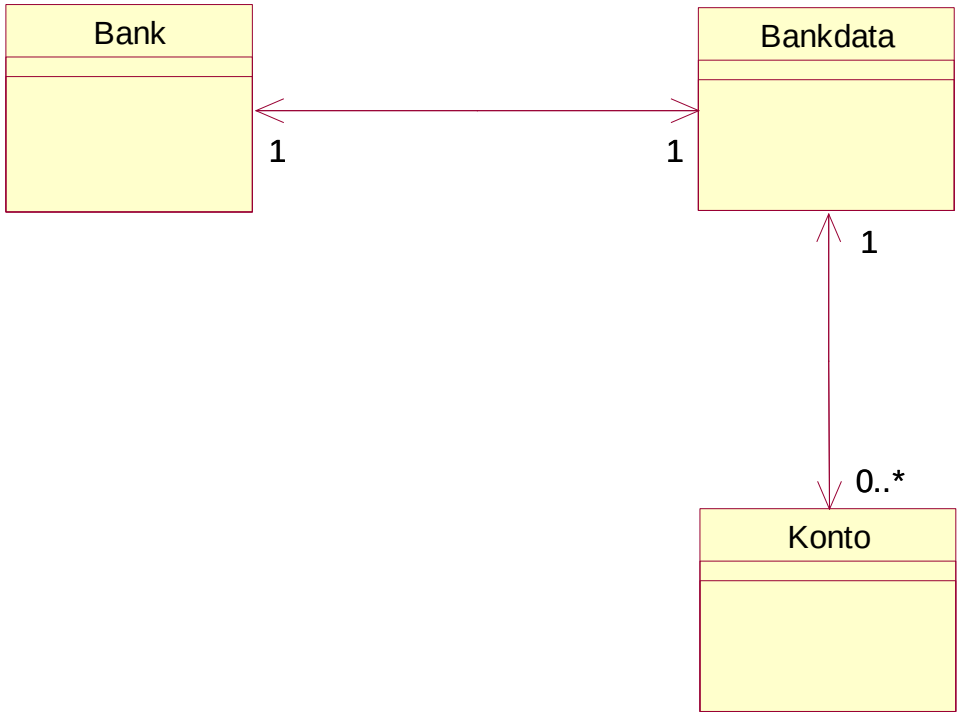
class BankData

Alle kontoene blir administrert av klassen BankData

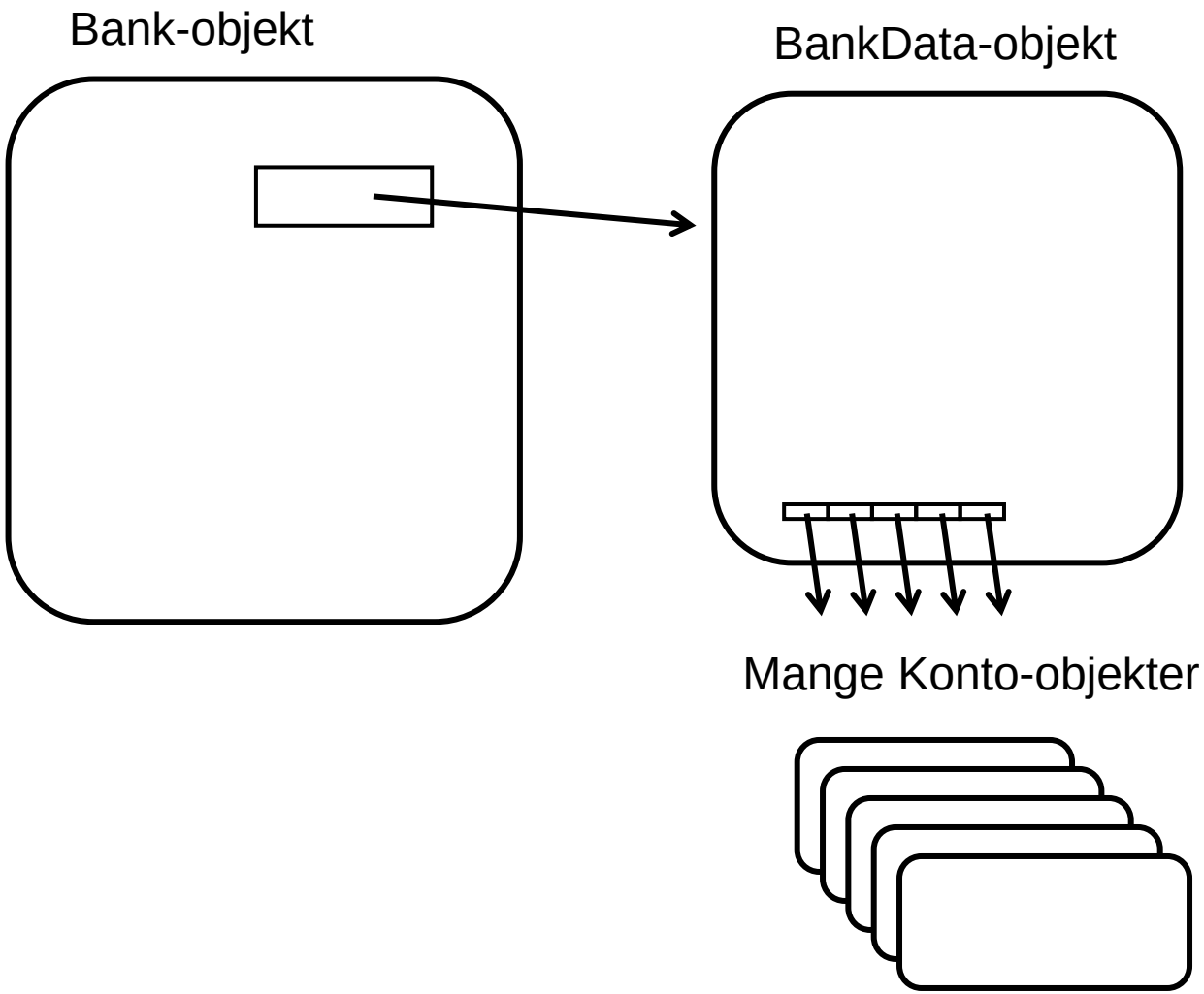
class Konto

Attributter (data) og metoder for en konto

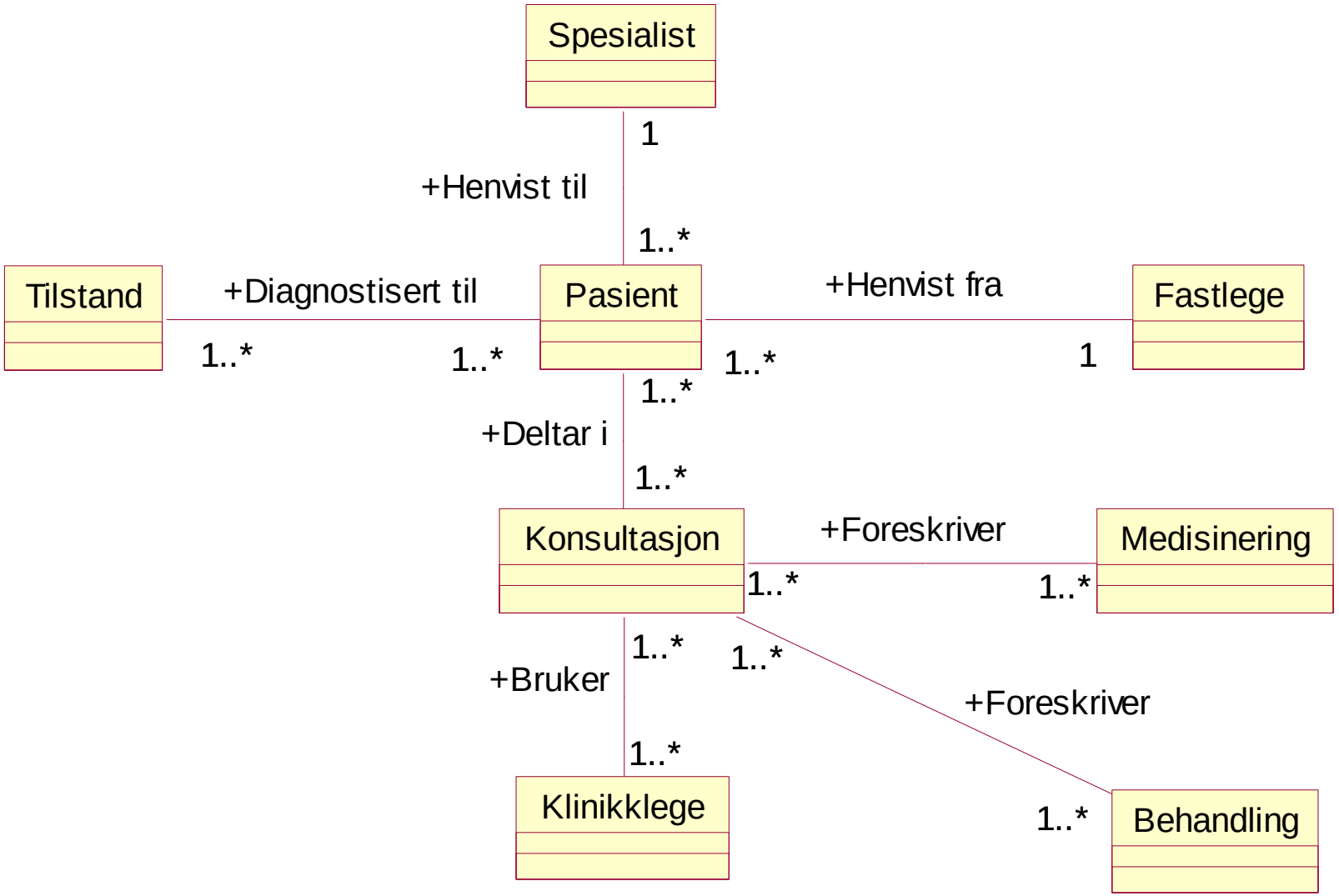
UML – klassediagram



Java datastruktur



Klasser og assosiasjoner i MHC-PMS



Klassen konsultasjon med flere detaljer

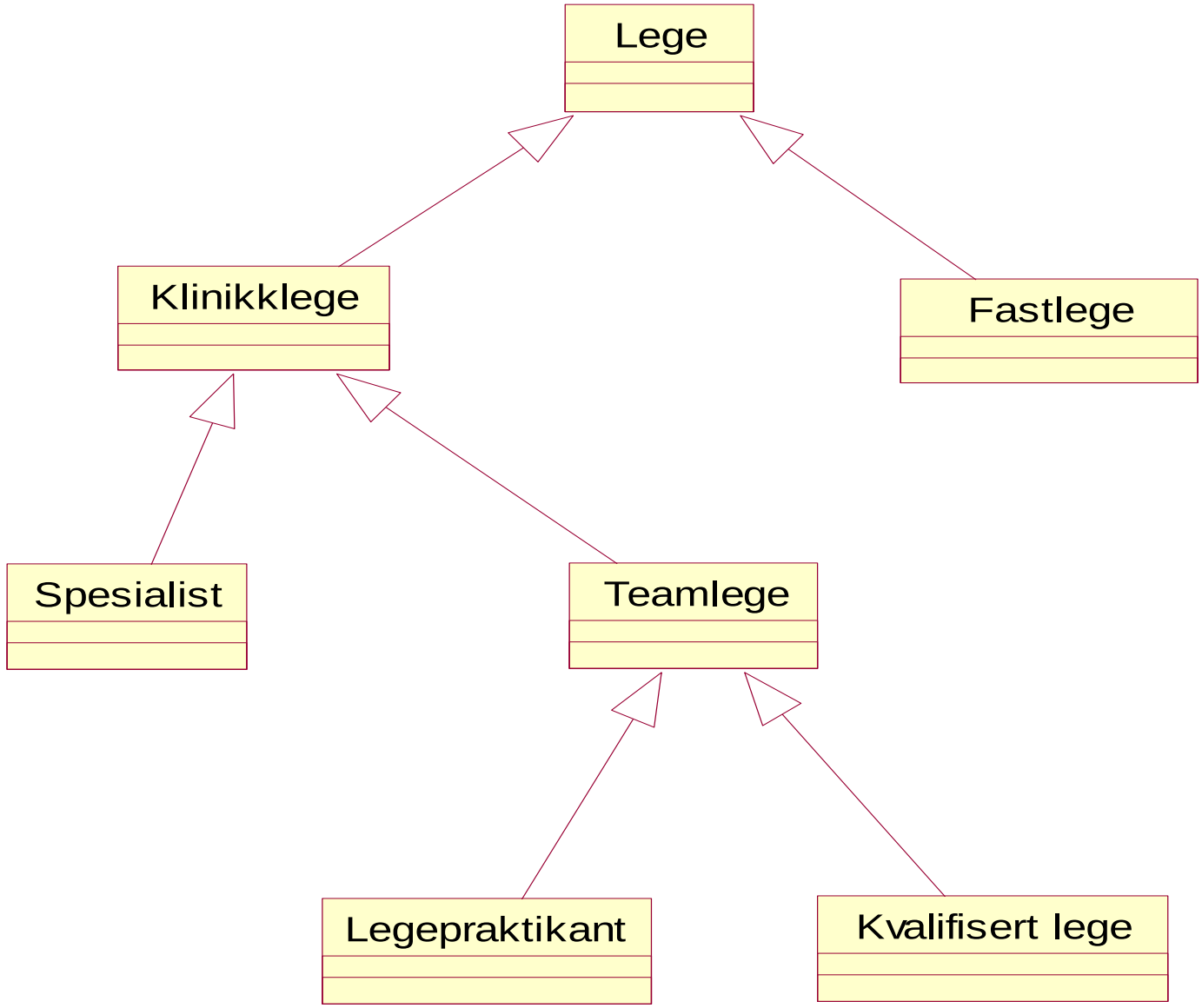
Generalisering

- Generalisering er en teknikk vi benytter oss av for å håndtere kompleksitet
- I stedet for å lære alle detaljene til hvert eneste objekt plasserer vi objektene i mer generelle klasser (person, dyr, bil, hus etc.) og fokuserer på hva som er karakteristisk ved disse klassene
- Dette gir oss muligheten til å se at ulike objekter har felles karakteristikker, for eksempel at både pasient og lege er personer

Generalisering

- Det er ofte nyttig å undersøke klassene i et system for å se om det er mulighet for generalisering
- I objektorienterte språk, som Java, er generalisering en del av språket – gjennom såkalt arvemekanisme (”inheritance”)
- Attributter og operasjoner (metoder) som er assosiert med ”superklasser” er også assosiert med ”subklasser” gjennom arv. Subklassene vil så legge til mer spesifikke attributter og operasjoner

Eksempel - generalisering



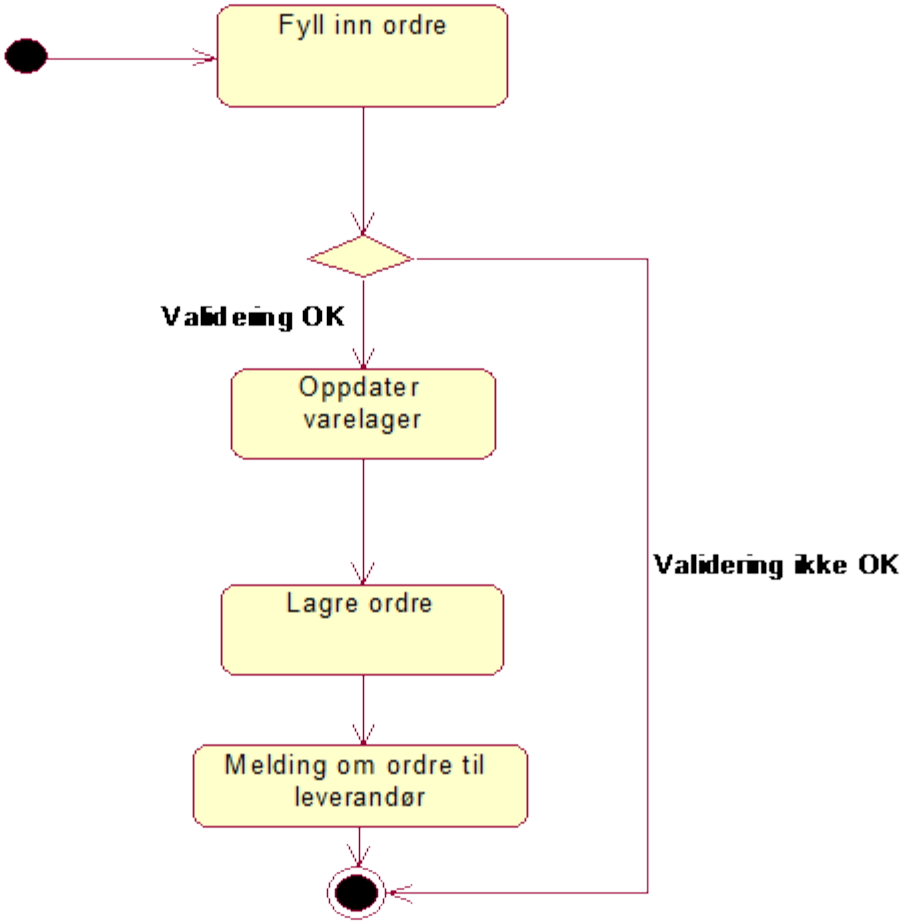
Adferdsmodeller (”behavioral” models)

- En adferdsmodell er en modell av dynamikken i et system ved kjøring av systemet. Modellen viser hva som skjer ved respons på stimuli.
- To typer stimuli
  - **Data**. Systemet fanger opp data som må prosesseres
  - **Hendelser**. En hendelse som trigger systemet og fører til prosessering. En hendelse har ofte assosierte data i tillegg, men ikke nødvendigvis

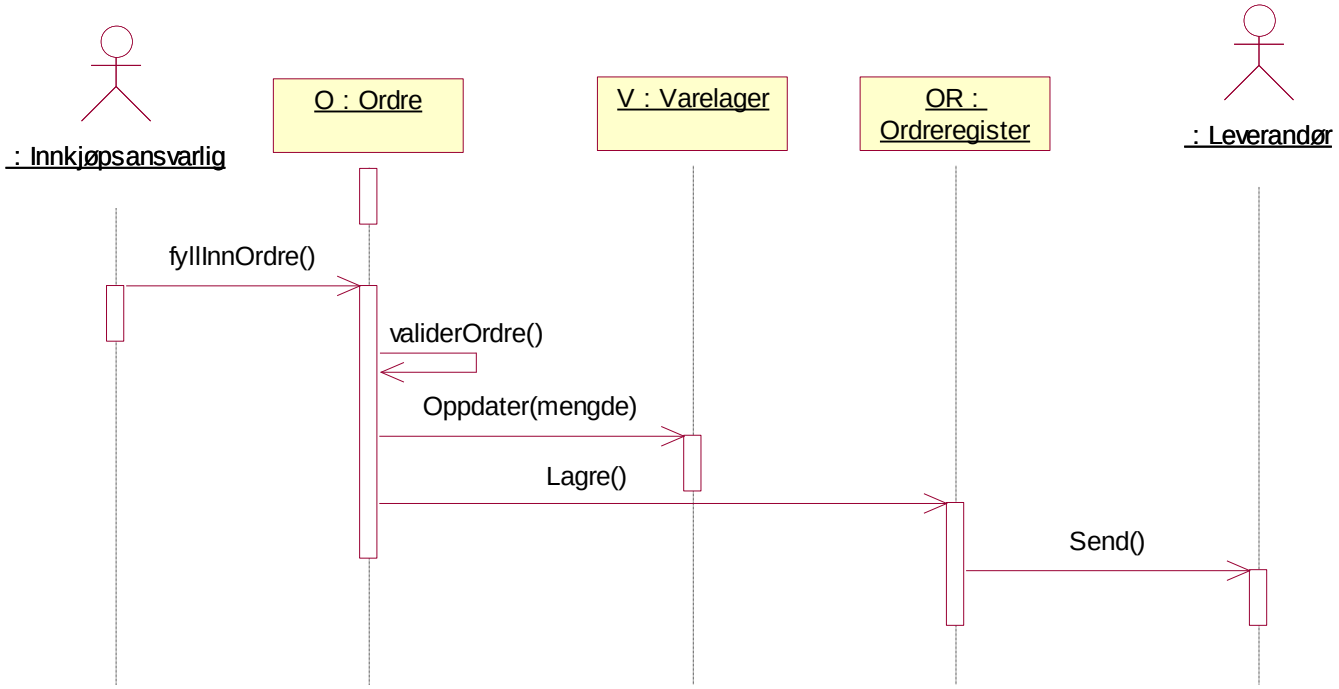
Datadrevet modellering

- Mange forretningssystemer er primært drevet av data. Systemene blir kontrollert av input av data, med få eksterne hendelser
- Datadrevne modeller viser sekvensen av handlinger som er involvert i prosesseringen av ”inputdata” og generering av ”outputdata”
- Modellene er spesielt nyttige under kravspesifikasjonsfasen og brukes til å vise ”end-to-end” prosessering av systemet

Aktivitetsdiagram ordreprosessering



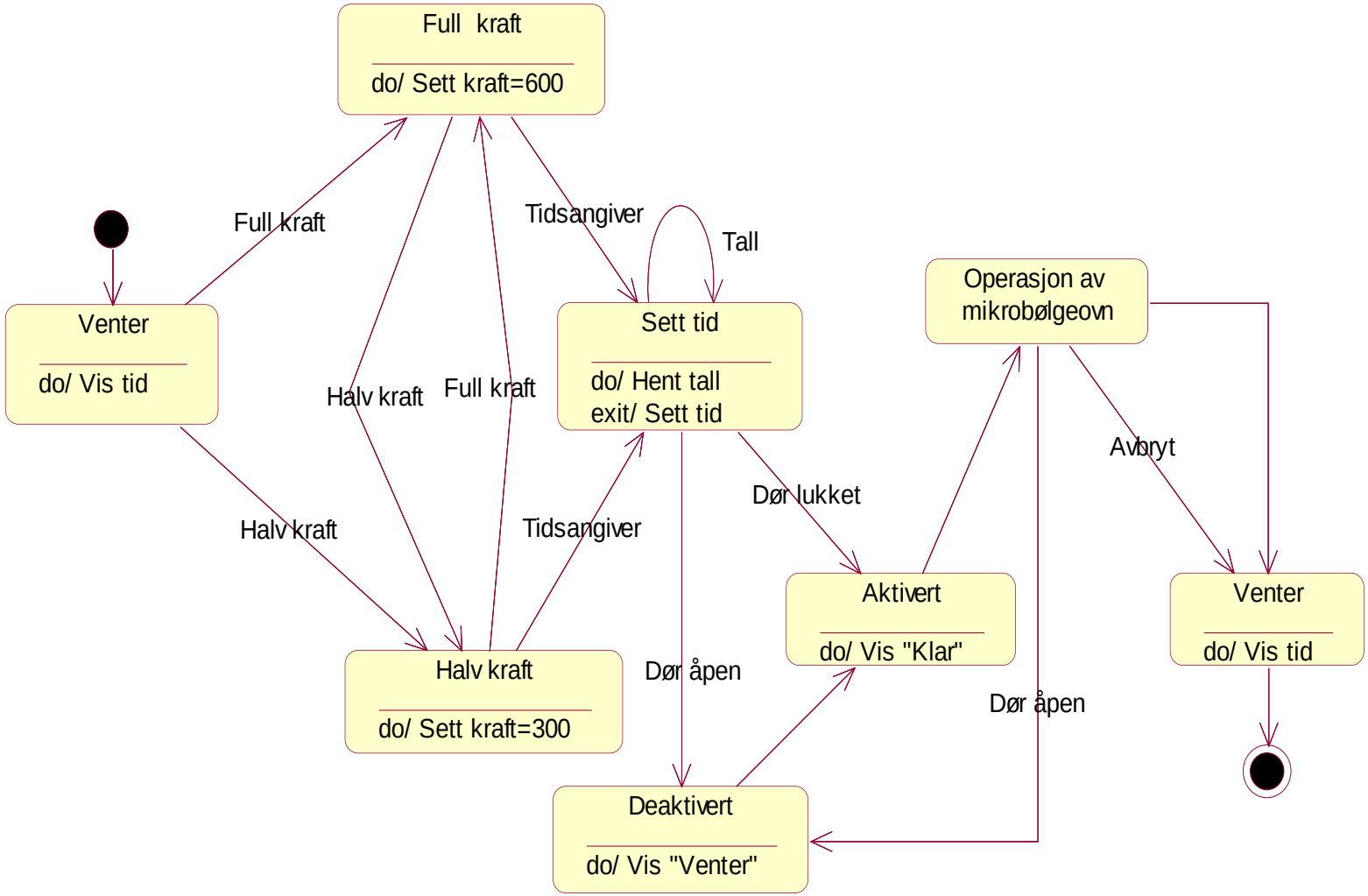
Ordreprosessering som sekvensdiagram



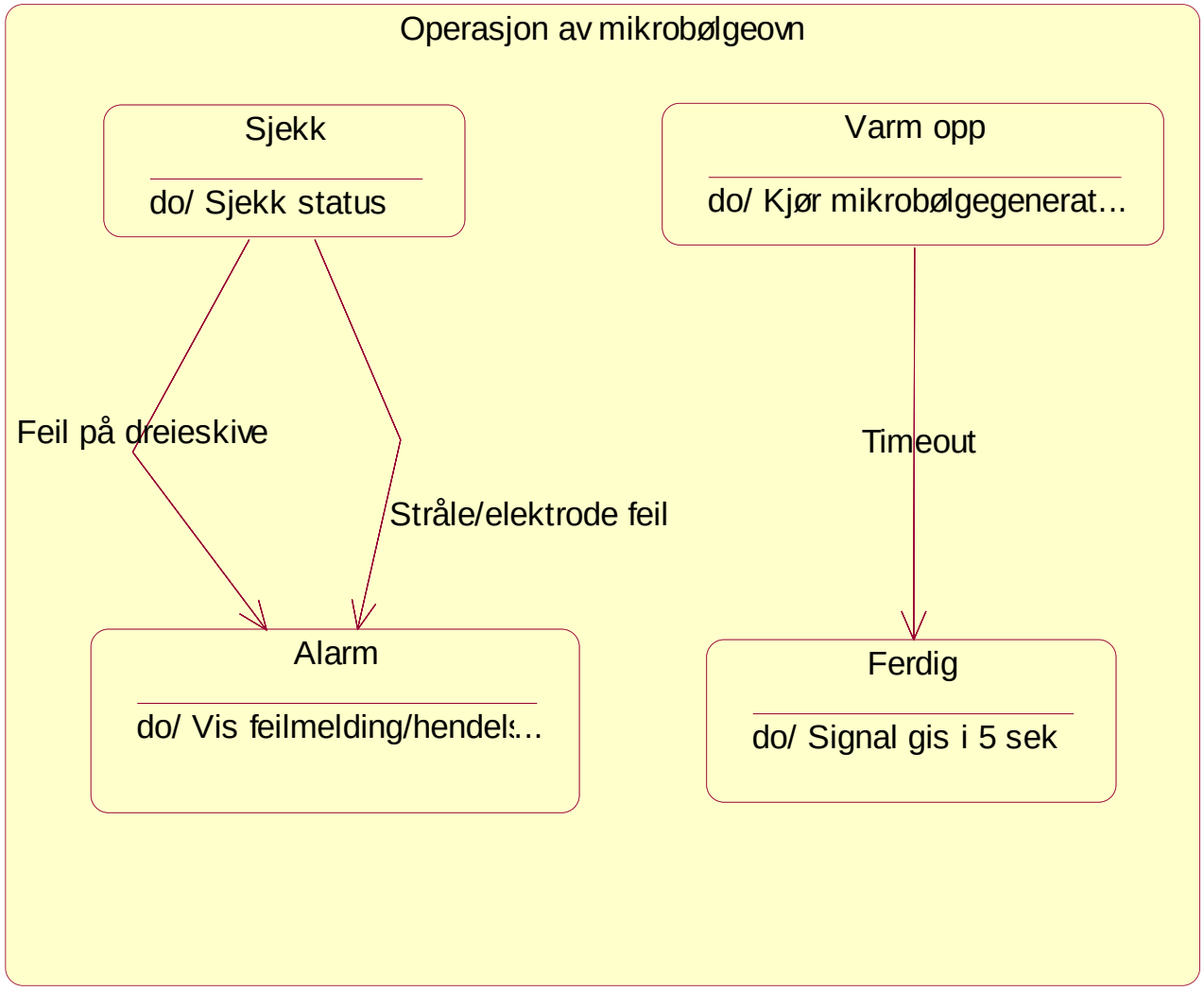
Hendelsesdrevet (“Event-driven”) modellering

- Mange systemer er drevet av ”hendelser”, med minimal data prosessering
- Modellene viser hvordan et system reagerer på eksterne og interne hendelser
- Baseres på antagelsen av at systemet har et endelig antall tilstander og at hendelser (stimuli) fører til at systemet går fra en tilstand til en annen

Tilstandsdiagram for mikrobølgeovn



Tilstandsdiagram for virkning av ovn



Tilstander og stimuli for mikrobølgeovn

Stimuli	Beskrivelse
Halv kraft (300 W)	Brukeren har trykket halv kraft knappen
Full kraft (600 W)	Brukeren har trykket full kraft knappen
Tidsinnstilling	Brukeren har trykket en av de forhåndsgitte tidsinnstillingene
Tall	Brukeren har trykket et tall på tastaturet
Åpen dør	Døra er ikke lukket
Dør lukket	Døra er lukket
Start	Brukeren har trykket startknappen
Kanseller	Brukeren har trykket kanseller knappen

Modelldrevet systemutvikling

- I modelldrevet utvikling (model-driven engineering - MDE) er det modellene i seg selv som er målet og “output” fra utviklingsprosessen
- Kode og programmer blir generert automatisk fra modellene
- Tilhengere av MDE hevder at dette gjør at ingeniører og utviklere ikke trenger å kunne detaljer innen programmering, plattformer etc.

Bruk av modell-drevet utvikling

- Fordeler
  - Høyere abstraksjonsnivå
  - Automatisk kodegenerering gjør det enklere og rimeligere å tilpasse systemer til nye plattformer

Bruk av modell-drevet utvikling

- Ulemper
  - Abstraksjonsmodeller er ikke nødvendigvis det riktige for implementering
  - Mange hevder koden er lite effektiv og endringer i koden er nødvendig
  - Vanskelig å holde modellen oppdatert ved endringer og feilretting

“Executable” UML

- Den fundamentale ideen bak modell-drevet systemutvikling er at det er mulig med en komplett automatisk transformasjon fra modell til kode.

- Dette er mulig innefor UML 2, og kalles Executable UML eller xUML
  - En **domenemodell** identifiserer de overordnede prinsippene, Domenemodellen bruker UML **klassediagram** og inkluderer objekter, attributter og assosiasjoner.
  - Klassemodeller, der klasser defineres sammen med attributter og metoder (operations i UML)
  - **Tilstandsmodeller** hvor et tilstandsdiagram assosieres med hver klasse og brukes til å beskrive livssyklusen til klassen

Smidige metoder og modell-drevet utvikling

- Mange brukere av modell-drevet utvikling hevder at metoden understøtter en iterativ tilnærming og derfor passer godt innen smidig metodikk
- Svært omfattende modellering tidlig i prosessen er i motsetning til idealene til “agile manifesto”, og det er få utviklere innen smidig metodikk som føler seg komfortabel med modell-drevet utvikling