Innledning

I dette dokumentet er det dokumentasjon av design for heisprosjektet i TTK4125. Det er (som sikkert kjent) et prosjekt der vi skal programmere en heis så den oppfører seg som en heis bør gjøre, med tanke på køsystem, obstruksjoner i heissjakt osv. Alt vi har i dette dokumentet er gjort før vi har begynt med selve programmeringen, og regnes derfor ikke som komplett dokumentasjon av sluttresultatet, men er utgangspunktet for hvordan vi kommer til å programmere dette. Det betyr at selv om sluttresultatet kanskje ikke blir helt likt, vil det uansett være lett å kjenne igjen strukturen som vi viser her i dette dokumentet.

Vi har beskrivelse av systemets funksjoner(use case), et overblikk på systemets arkitektur og beskrivelse av hver modul i systemet.

Innhold

1 Use cases

Use case blir brukt for å kunne beskrive hvilke funksjoner et system trenger for å kunne fungere som ønsket. Vi har funnet use casene ved å se på normale bruksområder for en heis og tenke på hvilke situasjoner man kan komme i.

Initialiser
Precondition:
Systemet er skrudd av
Trigger:
Systemet blir skrudd på
Main Success Scenario:
1. Heisen er i en etasje
2.Systemet går til idle-tilstand
Extensions:
1a Heisen er i en udefinert tilstand
.1: Heisen går nedover til etasje er nådd. Alle ordre blir ignorert. Returner til MSS step 1
1b Obstruksjon
.1: Systemet stopper
Success Guarantee:
Systemet er i idle-tilstanden
Minimal Guarantee:
Heisen beveger seg ikke

Bestill heis
Precondition:
Systemet er initialisert og og stopp er ikke aktivert
Trigger:
Bestillingsknapp blir aktivert
Main Success Scenario:
1. Etasje og retning blir lagt til i queueList og callButtonLamp blir aktivert
2. Systemet håndterer ordre i queueList
3. Heisen ankommer etasjen, floorLamp blir aktivert, og callButtonLamp blir deaktivert
4. Dørene åpnes (3s)
5. Dørene lukkes
Extensions:
1a Flere ordre samtidig
.1 Ordrene blir håndtert
4/5a Obstruksjon
.1: Heisen stopper
.2: Dørene holdes åpne
.3: Obstruksjonen blir fjernet, returner til MSS step 2
xa Use caset kan når som helst bli avbrutt av stopp-use caset.
Success Guarantee:
Heisen kommer til etasjen, dørene åpnes og lukkes
Minimal Guarantee:
Heisen beveger seg ikke

Kjør heis

Precondition:

Systemet er initialisert, stop er ikke aktivert

Trigger:

Etasjeknapp blir trykket

Main Success Scenario:

1. Etasje blir lagt til i queueList, orderButtonLamp blir aktivert
2. Systemet håndterer ordre i queueList
3. Heisen ankommer etasjen, orderButtonLamp blir deaktivert
4. Dørene åpnes
5. Dørene lukkes

Extensions:

4/5a Obstruksjon

.1: Heisen stopper

.2: Dørene holdes åpne

.3: Obstruksjonen blir fjernet, returner til MSS step 2

xa Use caset kan når som helst bli avbrutt av stopp-use caset.

Success Guarantee:

Heisen når riktig etasjen, dørene åpnes og lukkes

Minimal Guarantee:

Heisen beveger seg ikke

Stopp

Precondition:

Systemet er initialisert, stopp er ikke aktivert

Trigger:

Stopp-knappen blir trykket

Main Success Scenario:

1. Heisen stopper, stopLight blir aktivert
2. Skrur av alle bestillingslys
3. Køen blir slettet, og alle nye ordre ignoreres
4. Stopp-knappen blir deaktivert, returner til forrige Usecase

Extensions:

1 a. Stop trykket i en etasje

.1: Dørene holdes åpne

Success Guarantee:

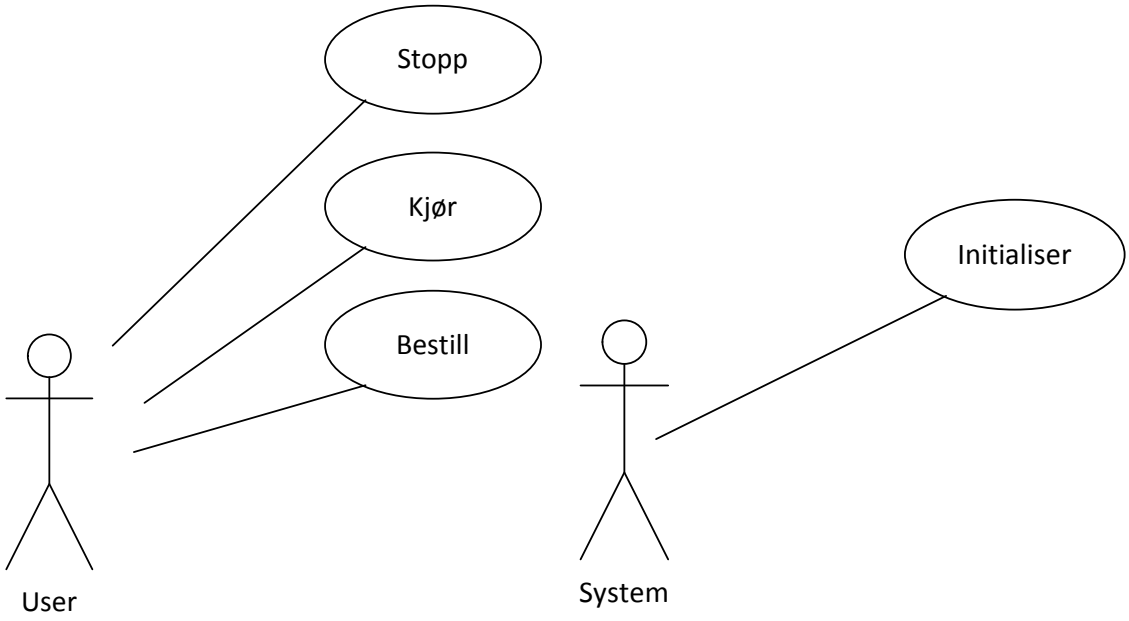
Returnerer til forrige Usecase, køen er slettet

Minimal Guarantee:

Heisen beveger seg ikke

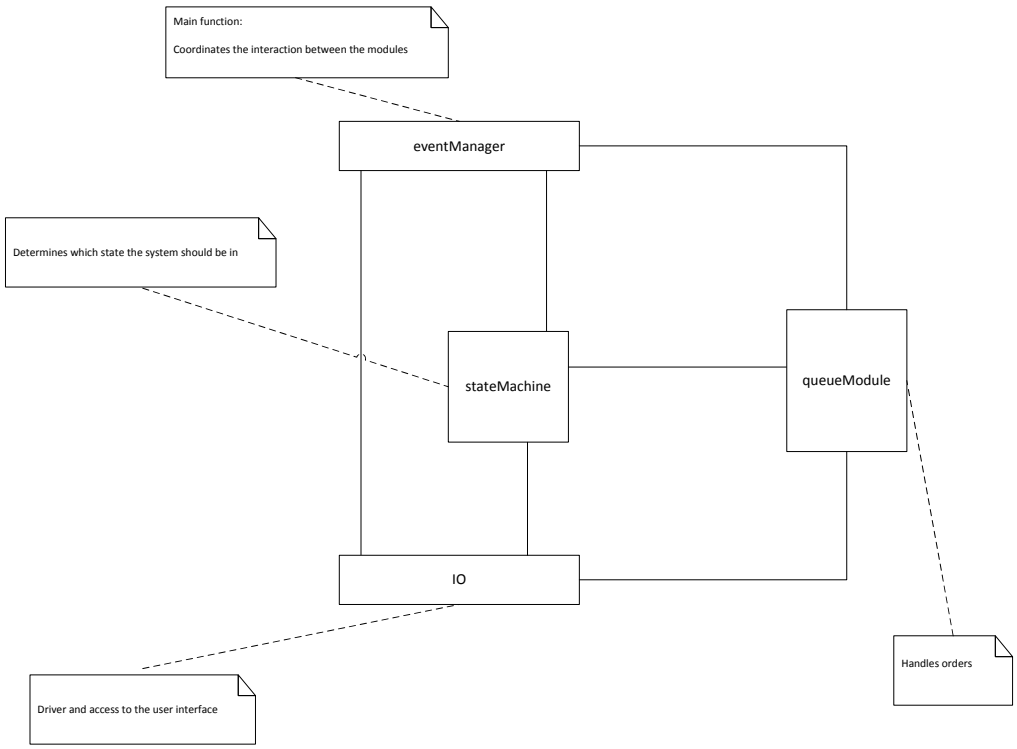
Aktører:

Bruker: personen som bruker heisen, både med å bestille og kjøre heisen. Bruker trenger ikke å gjøre begge deler for å være definert som bruker.
System: aktøren som håndterer heisstyringen



2 Overordnet systemarkitektur

Systemet består av IO som sender og mottar signaler mellom hardware og software. Den sier ifra til eventManager, som har oversikt over hendelsene som foregår. Den sender signaler til stateMachine, som styrer hvilken tilstand systemet er i. eventManager sender også signal til queueModule, som holder oversikt over hvilke etasjer som er bestilt, og om bruker skal opp eller ned.

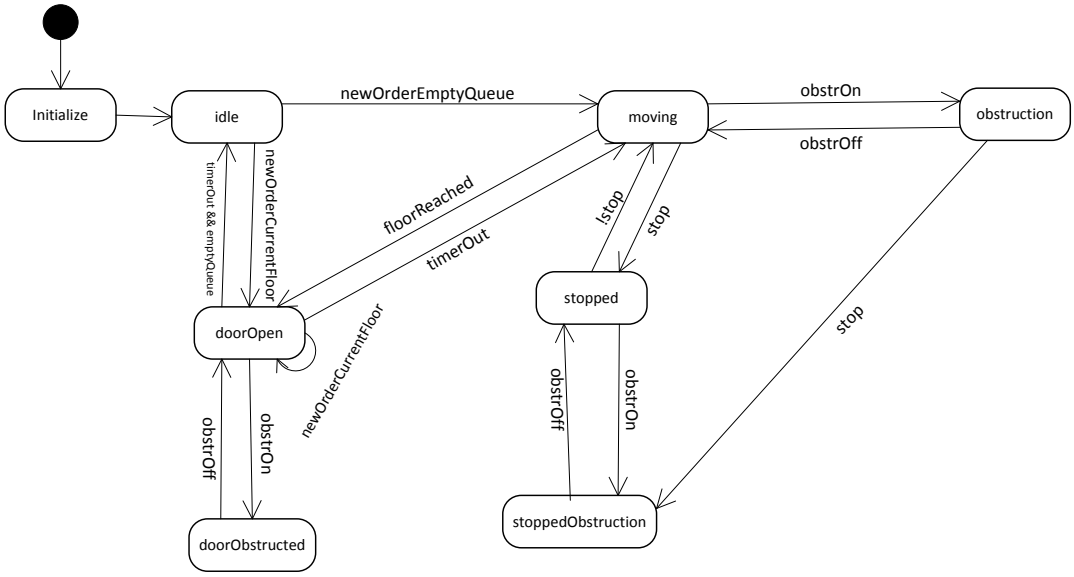


3 Moduler og grensesnitt

3.1 Moduler

eventManager er det som tradisjonelt er main-fila i programmet, og er "hovedmodulen", modulen som knytter alle de andre modulene sammen. Den ser etter informasjon fra IO og videresender informasjonen den eventuelt får til tilstandsmaskinen og kømodulen.

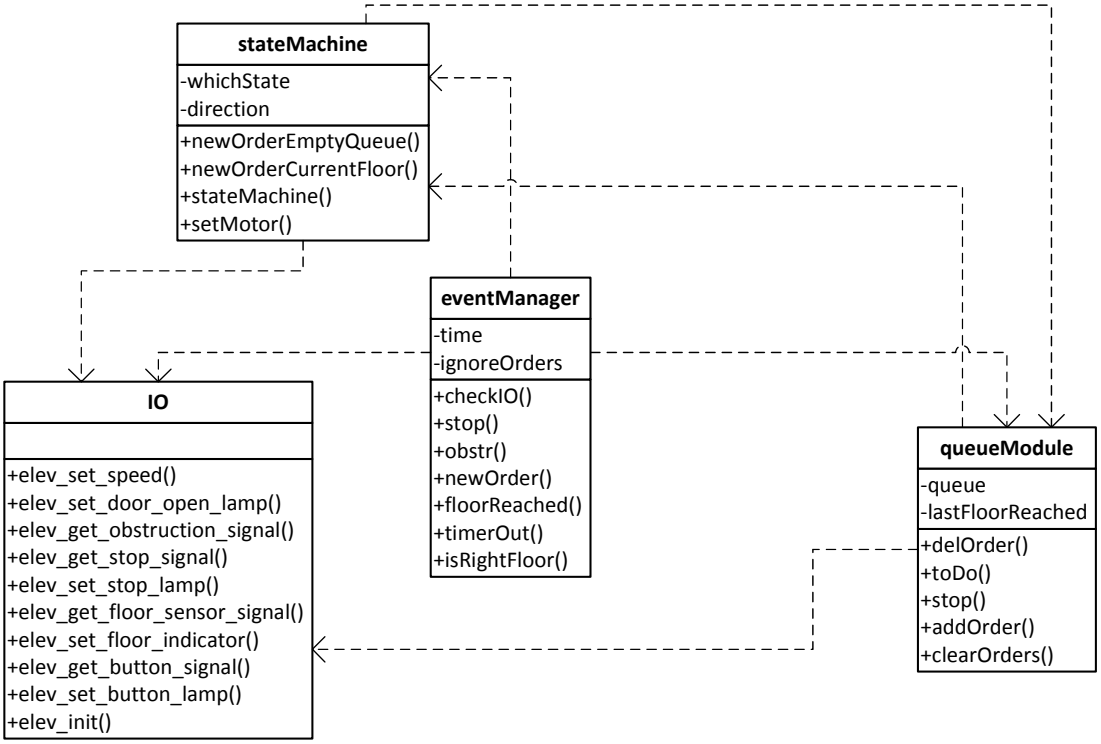
stateMachine er tilstandsmaskina, og holder oversikt over hvilken tilstand systemet befinner seg i. Den setter også neste tilstand.



Kømodulen, queueModule, har til enhver tid oversikt over hvilken etasje heisen besøkte sist og hvilken retning den reiser. Den har også oversikt over alle ordrene som har blitt gjort, og hvor mange det er igjen av køen.

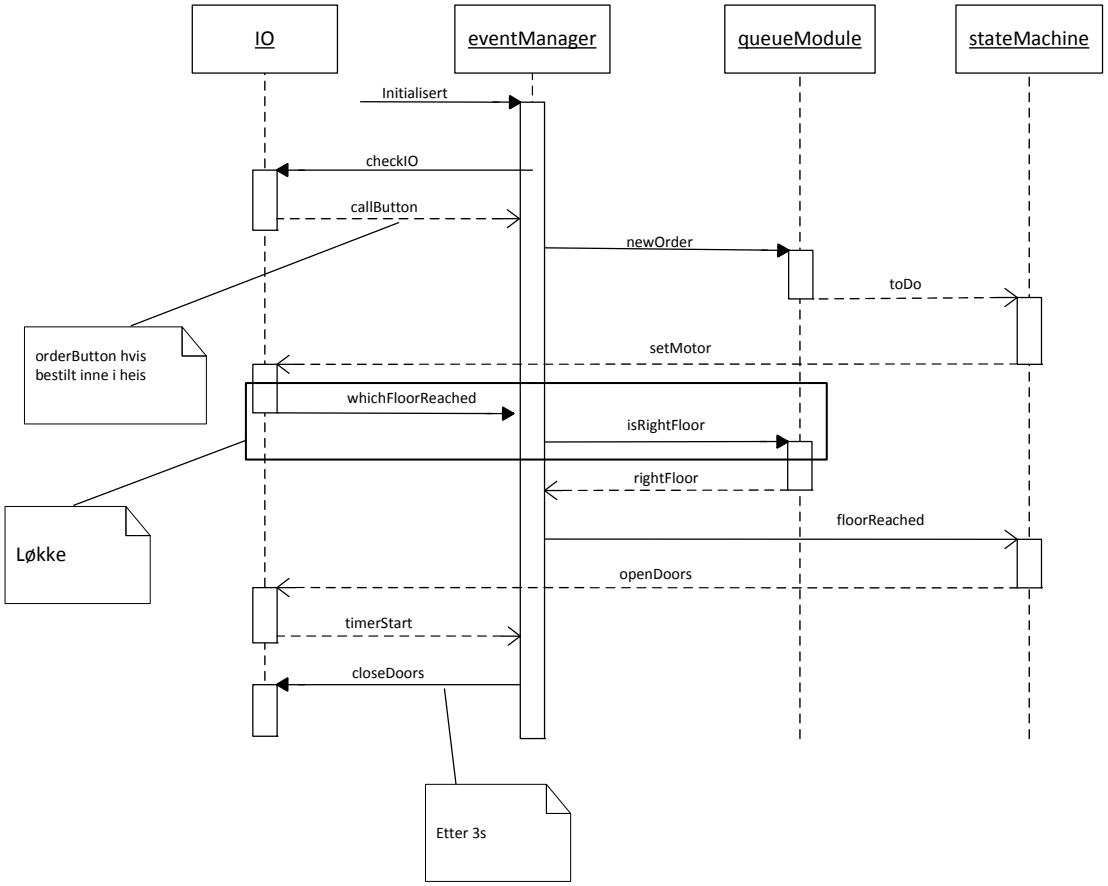
3.2 Klassediagram

Klassediagrammet beskriver objekttypene i et system og er en fin måte å få oversikt på hva som gjør hva og hvordan de henger sammen.

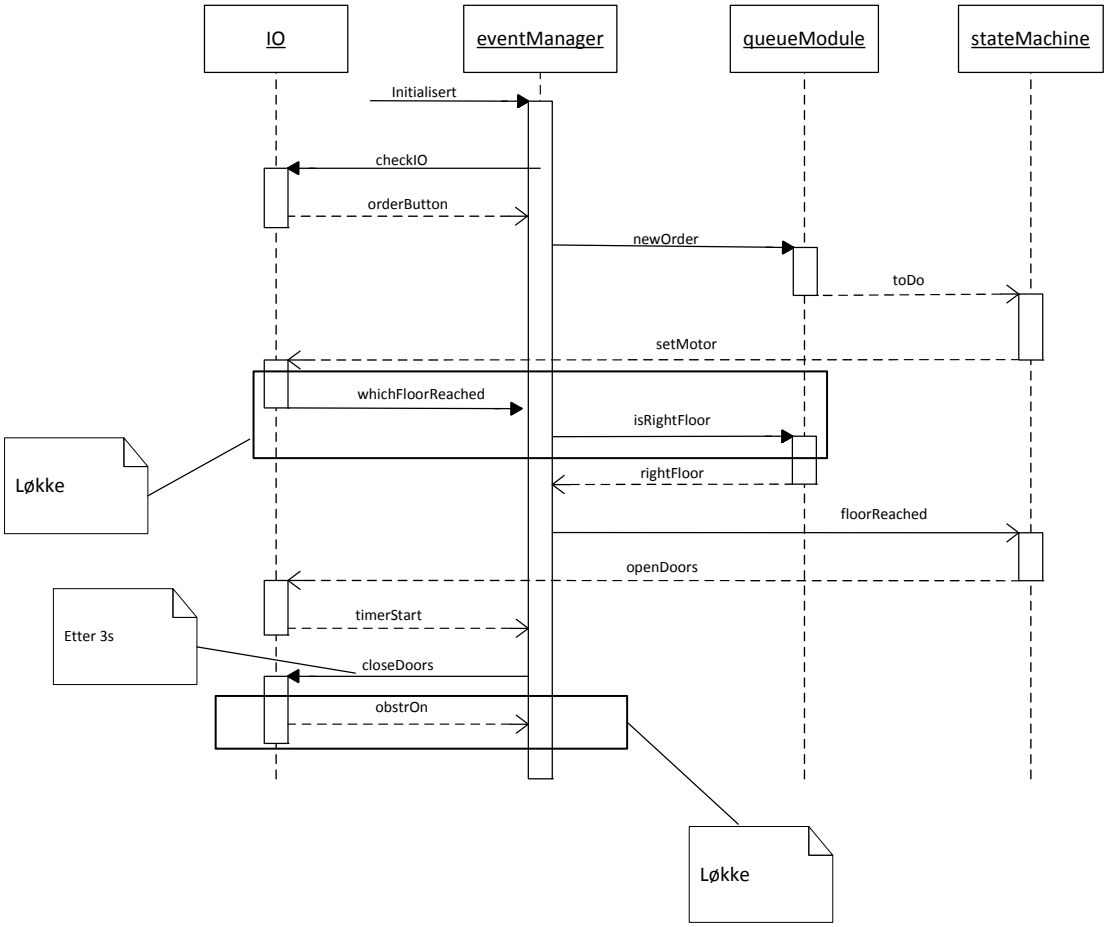


4 Sekvensdiagram

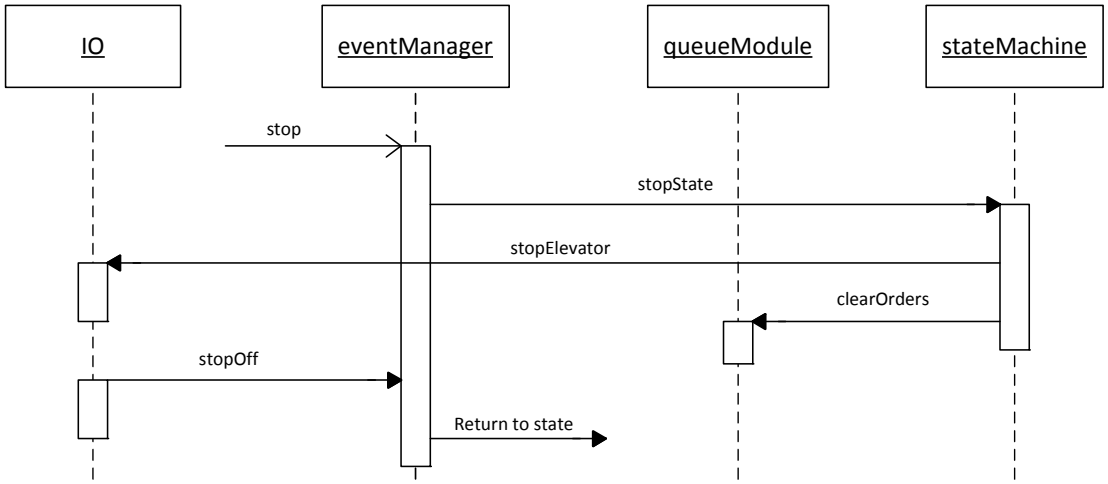
Sekvensdiagram for bestilling og kjøring av heis. Legg merke til at forskjellen i kjøringa bare er om det er en etasjeknapp eller en bestillingsknapp på innsida av heisen. Vi har allikevel valgt å ha det som to forskjellige use cases ettersom det er to uavhengige prosesser som ikke nødvendigvis henger sammen(man kan bestille heis uten å gå inn i den, og man kan ta heisen uten å ha bestilt den).



Sekvensdiagram for kjøretid når det er obstruksjon i døra når den åpnes. Det er en utvidelse i både bestill og kjør heis-use casene.



Sekvensdiagram for kjøretid når stop-knappen er aktivert.



5 Avsluttende kommentarer

Å bruke UML er en fin måte å få oversikt over et system før man begynner å programmere det. Det får deg til å tenke på hva som trengs og hva som ikke trengs, og vi vil påstå at det mest sannsynlig vil føre til noen smarte løsninger som vi ellers ikke ville ha funnet. Nå som vi har gjort UML-delen, føler vi at vi er mer klar for å programmere.