Rapport Heisprosjekt

Sigurd Digre og Simen Allum

Februar 2020

Innhold

1 Introduksjon

Dette er en rapport om heisprosjektet i faget TTK4235 - tilpassede datasystemer. Rapporten tar for seg systemets overordnede arkitektur, moduldesign og systemtesting. Til slutt diskuteres mulige svakheter ved systemet og mulige endringer som kunne vært gjort.

2 Overordnet arkitektur

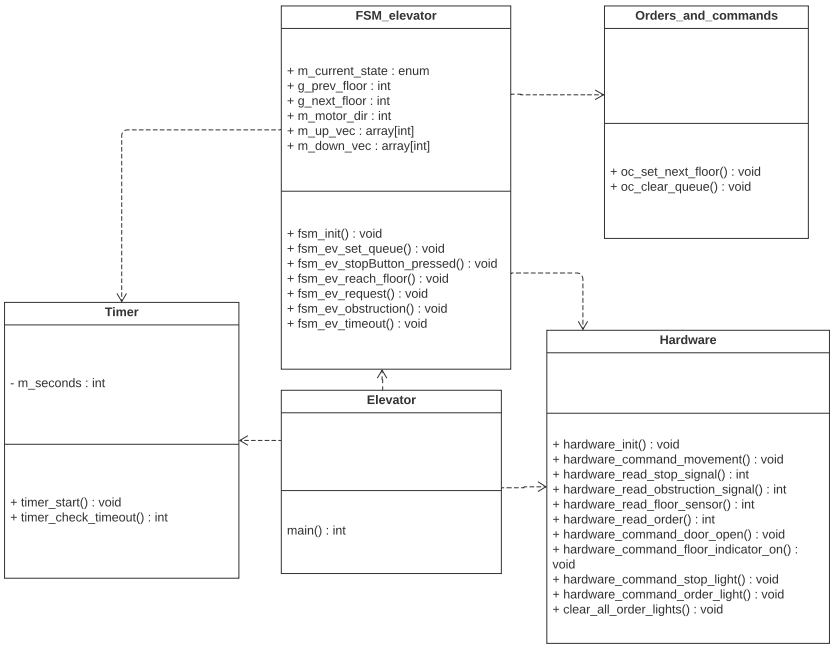
Systemet består av 5 klasser (figur 1). I elevator er det en while-loop som overvåker knapper og sensorer og kaller ulike tilstandsmaskiner basert på avlesninger. Elevator er derfor avhengig av alle klassene unntatt order_and_commands. Grunnen til at både Elevator og FSM_elevator er avhengig av klassen Timer er at overganger som blir gjort i FSM_elevator skal starte timeren, og while-loopen

i Elevator skal kunne overvåke timeren og sjekke om det er en timeout.

Alle overganger mellom de ulike tilstandene foregår i FSM_elevator, og denne klassen er derfor avhengig av hardware, timer og order_and_commands.

Order_and_commands inneholder to funksjoner. En funksjon for å slette alle bestillinger, og en for å bestemme hvilken etasje som er heisens neste. Grunnen til at det er kun FSM_elevator som har tilgang til order_and_commands er fordi den kun blir kalt fra tilstandsmaskinene ved en mulig overgang.

Det er to klasser som er avhengig av hardware. Det er Elevator og FSM_elevator. Elevator er avhengig av hardware siden while-loopen må kunne sjekke knapper og sensorers tilstand, og FSM_elevator da overgangene må endre kunne endre tilstanden til heisens hardware.

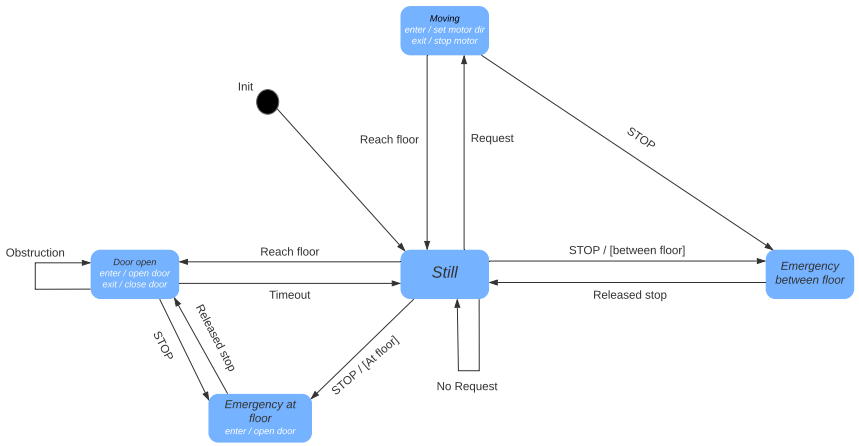


Figur 1: Klassediagram for arkitekturen

Figur 2: Sekvensdiagram for sekvensen i tabell 1

Tabell 1: Sekvens som visualiseres i figur 2

Delsekvens	Beskrivelse
1.	Heisen står stille i 2. etasje med døren lukket
2.	En person bestiller heisen fra 1. etasje
3.	Når heisen ankommer, går personen inn i heisen og bestiller 4. etasje
4.	Heisen ankommer 4. etasje, og personen går av
5.	Etter 3 sekunder lukker dørene til heisen seg



Figur 3: Tilstandsdiagram for heisen

Arkitekturen har fem tilstander (figur 3), og initialiseres inn til Still. Vi valgte å kun ha én tilstand for bevegelse (moving) istedet for to (opp, ned). Vi har derfor brukt en ekstra variabel for motorretning, da dette krevde mindre logikk enn det en ekstra tilstand ville gjort. Vi skiller også på Emergencytilfellene ved å ha to separate tilstander for når heisen er i en etasje og når den er mellom etasjer. Dette fordi døren her skal behandles forskjellig. Beksrivelse av tilstandene: (EAF = Emergency at floor, EBF = Emergency between floor)

Still Heisen har ingen bestillinger, og står stille. Den skal kunne nå denne fra alle de andre tilstandene. Kan ta bestillinger.

Door open Heisen må stå stille, være i en etasje, og ha nådd en destinasjon i en ordre. Kan ta bestillinger.

Moving Heisen er i bevegelse, enten oppover eller nedover. Kan ta bestillinger.

EAF Stopp-knappen må ha blitt trykket inn når heisen står stille i en etasje. Tar ingen bestillinger.

EBF Stopp-knappen må ha blitt trykket inn når heisen er i bevegelse. Tar ingen bestillinger.

Arkitekturen er designet på denne måten for å minimere avhengigheter samtidig som å sørge for oversiktlig og strukturert kode. Elevator-klassen er bevisst holdt på et minimumsnivå for å gjøre det lett å se gangen i programmet kun ved å studere denne klassen.

3 Moduldesign

3.1 Elevator

Det er i denne modulen main-funksjonen ligger. Elevator-modulen er holdt på et simpelt nivå for å øke lesbarhet og gjøre kodeflyten lettere å følge. Den enslige while-loopen gjør det lettere å følge koden for å detektere feil og å legge til ny funksjonalitet. Main-funksjonen initialiserer først heisen for så gå inn i en loopen som overvåker knapper og sensorer. Om det gjøres avlesninger av hardware som returnerer true kalles det ulike event-funksjoner i FSM-modulen.

3.2 FSM

Denne modulen lagrer tilstander. I modulen har vi de to globale variablene g_prev_floor og g_next_floor som inneholder forrige etasje og neste etasje og de interne modulvariablene m_current_state og m_motor_dir for å inneholde nåværende tilstand og motorretning. Den har også to arrays, m_up_vec og m_down_vec, som inneholder bestillinger. En for bestillinger som skal prioriteres i det heisen er på vei opp, og en på vei ned. Funksjoner for alle hendelser/overganger mellom ulike states finnes også her.

I modulen har vi fsm_init() som sørger for at heisen initialiseres på riktig måte. Den sjekker først om heisen står i enten første eller fjerde etasje, om dette er tilfelle initialiseres alle variable med riktig verdi. Om heisen er i en annen udefinert tilstand kjører heisen oppover til den får en positiv avlesning på en etasjesensor for deretter å stoppe heisen i etasjen og initialisere variablene med riktige verdier. Heisen vil ikke ta imot ordre før den er i en definert tilstand.

Køsystemet i heisen ligger implementert i denne modulen under funksjonen fsm_set_queue(). Køsystemet består av to array som har en fast lengde på fire posisjoner hver. Det ene arrayet inneholder orde som skal prioriteres når heisen er på vei opp, og det andre for ordre som skal prioriteres på vei ned. Ved å gjøre det på denne måten blir logikken veldig enkel, og det er lettere å feilsøke og rette opp i eventuelle feil. Man slipper å dynamisk allokere minne for nye ordre som måtte komme inn, som igjen hindrer at man løper tom for minne.

Ordre legges inn i de to arrayene basert på hvilken retning heisen kjører/kjørte i. Om det kommer ordre utenfra heisrommet i de to endepunktene i første og fjerde etasje legges disse inn i henholdvis ned arrayet og opp arrayet. Slik vil en opp-ordre i første etasje bli prioritert når heisen er på vei, og en ordre i

fjerde blir prioritert når heisen er på vei opp. Bestillinger i et av endepunktene blir behandlet på et naturlig tidspunkt som skaper flyt i heisdriften. Ordre fra heisrommet blir lagt i riktig array basert på motorretningen til heisen og forrige etasje heisen var i. Ved å ha implementert køsystemet på denne måten fører det også til at køsystemet vil være skalerbart til flere etasjer da logikken vil være lik uavhengig av antallet etasjer.

I fsm_set_queue() sørges det også for at en bestilling som gjøres i en etasje der heisen står med døren åpen ikke regnes som en ekte bestilling da heisen allerede er i etasjen med døren åpen. Bestillinger som kommer fra egen etasje blir derfor ignorert om døren er åpen.

3.3 Orders and commands

I denne modulen finner vi to hjelpefunksjoner som har med ordre å gjøre. Heisen er primært styrt av variablen g_next_floor som inneholder neste etasje heisen skal kjøre til. Vi har derfor implementert en funksjon oc_set_next_floor() som tar inn motorretning, siste avleste etasje og bestillingsarrayene samt en referanse til variablen g_next_floor. Basert på retningen den har, vil heisen først sjekke alle potensielle bestillinger foran seg, og i samme retning. Deretter sjekkes alle mulige bestillinger i arrayet for motsatt retning, så til slutt de resterende bestillingene i arrayet i bevegelsesretningen. Om ingen av arrayene inneholder noen bestillinger vil g_next_floor settes til NO_ORDERS som betyr at heisen skal stå stille. På denne måten sørges det for at heisen blir mer effektiv da den f.eks prioriterer en 4 ned-bestilling om den står i tredje etasje før den kjører ned.

3.4 Timer

Timer-modulen har to funksjoner; en for å starte timeren, og en for å sjekke timerens tilstand. Timer_start() lagrer tidspunktet under modul-variabelen m_seconds i det den blir kalt. Og timer_check_timeout() sammenligner den lagrede tiden under m_seconds med gjeldene tid og returnerer TRUE om tiden er større enn 3 sekunder. timer_check_timeout() kalles en gang i hver løkkeiterasjon i Elevator, noe som gjør at man sjekker timeren tilnærmet kontinuerlig. Ved å implementere timeren på denne måten tillater man at timer_start() kan kalles flere ganger på rad da den kun oppdaterer gjeldene tidspunkt i m_seconds variablen. Dette designet gjør også at m_seconds blir en intern modul-variabel som fører til at modulene er svakere koblet.

3.5 Hardware

Hardware-modulen er utdelt kode i dette prosjektet, og vi viser derfor til medfølgende doxygen-dokumentasjon for informasjon om modulens funksjonaliteter.

4 Testing

Testing av heisen funksjonalitet er gjort via fem ulike testcaser som gjenspeiler kravspesifikasjonen. De fem ulike testcasene er som følgende:

T1 Fokus: oppstart, etasjelys, ordrelys, prioritering og dør
Heisen står mellom 1. og 2. Programmet blir startet. Mens heisen er på vei opp til andre, får den en opp-bestilling i 3.

Når heisen er i 2. får den en opp-bestilling i 2. Mens døren er åpen får den en opp-bestilling i 1., deretter en innside-bestilling til 3.

Heisen kjører opp og får en ned-bestilling i 4. Etter stopp i 3. får den innside-bestilling til 2.

Etter den har passert 3. på vei ned får den en ned-bestilling i 3.

T2 Fokus: Urealistiske startbetingelser
Heisen står i endestoppbryter nederst.

T3 Fokus: Prioritering av bestillinger
Heisen står i 3. Får innside-bestilling til 1.

På vei ned får den en opp-bestilling i 2.

Deretter opp- / ned-/ innside-bestilling i og til 3. Til slutt en innside-bestilling til 4.

T4 Fokus: Håndtering av stopp-knapp
Heisen skal stå i 4. Bestillinger er som følgende: 3 ned-bestilling, 2 opp-bestilling og 1 innside-bestilling.

Mellom 4. og 3. klikkes stopp-knappen. Mens stopp er holdt inne bestilles det på nytt en opp-bestilling i 2.

Stopp slippes. Stopp trykkes igjen og slippes.

Det kommer innside-bestilling til 4. Deretter ned-bestilling i 3. og ned-bestilling i 2..

Når heisen kommer til 3. trykkes stopp mens døren er åpen.

Stopp slippes.

Vente tre sekunder på at døren skal lukkes. Deretter trykkes stopp igjen.

Mens stopp holdes inne trykkes det innside-bestilling til 1. Stopp slippes.

T5 Fokus: Obstruksjonsbryter
Heisen står i 1. med døren lukkes og har ingen bestillinger. Obstruksjonsbryteren settes aktivt. Innside-bestilling til 2.

Når den kommer til 2. ventes det 10 sekunder og bekrefter at døren holdes åpen. Det gjøres følgende bestillinger: opp- / ned-/ innside-bestilling i og til 2.

Deretter ned-bestilling i 4., innside-bestilling til 1. Obstruksjon skrus av.

Heisen skal kjøre til 4., for deretter å kjøre til 1.

Krav/Test	T1	T2	T3	T4	T5	Krav oppfylt
O1	x					x
O2	x					x
O3		x				x
H1	x					x
H2			x			x
H3			x			x
H4	x		x			x
L1	x		x			x
L2	x		x			x
L3	x		x			x
L4	x		x			x
L5	x		x			x
L6				x		x
D1	x		x			x
D2	x		x			x
D3				x		x
D4					x	x
S1	x		x			x
S2	x		x			x
S3	x		x			x
S4				x		x
S5				x		x
S6				x		x
S7				x		x
R1					x	x
R2						(x)
R3	x					x
Y1						(x)

Det er flere tester som huker av flere av kravspesifikasjonene. Men hver test har ulike fokusområder og vi krysser derfor ikke av alle kravene testen faktisk oppfyller. Punktene R2 og Y1 er vanskelige å oppnå direkte gjennom tester, men kan sies å være oppfylt indirekte gjennom kjøring av systemet flerfoldige ganger.

5 Diskusjon

5.1 Svakheter i systemet

Vi har valgt å ikke inkludere FSM i orders_and_commands siden det kun er én av variablene (g_next_floor) som skal endres. Ved å la være å inkludere FSM-klassen i orders_and_commands har vi brukt et "smutthull" ved å unngå å ha

en gjensidig avhengighet mellom disse to klassene, noe som ville gitt dårligere kodekvalitet. Dette baserer vi på ideen om å minimere antall avhengigheter mellom klasser. Dette kunne vært løst på en annen måte ved å legge innholdet i funksjonen flere steder i FSM-klassen. Koden hadde da blitt mindre oversiktlig og lesbar da oc_set_next_floor() kalles flere steder fra FSM-klassen. Ved å skille ut dette i en egen funksjon i en egen klasse har vi også sørget for at endringer i FSM-klassen ikke påvirker oc_set_next_floor().

Under testing fant vi en oppførsel hos systemet vårt som kan minne om en logisk brist. Det er en spesifikk situasjon der oppførselen ikke er som forventet. Situasjonen går som følger:

1. Heisen står i 4 etasje.
2. Det kommer en opp-bestilling i 2.
3. Personen som har bestilt heisen i 2. går på heisen og gjør en innside-bestilling til 3.
4. Rett før døren lukkes kommer det er person løpende inn i heisen og gjør en innside-bestilling til 1.
5. Heisen kjører ned til 1.

Dette er en situasjon som ikke er helt heldig. Da heisen er bestilt oppover fra 2. burde innside-bestillingen til 3. prioriteres før bestillingen i 1. Grunnen til at dette skjer i vårt system er at køsystemet er implementert slik at heisen alltid prioriterer en bestilling i samme retning som heisen kjører. Når heisen kommer ovenfra og ned i 2. har den motorretning nedover, og prioriterer dermed orde i ned-arrayet. I vårt system med kun fire etasjer vil ikke dette være et meget stort problem, da innside-bestillingen til personen som har bestilt heisen til 2. blir fullført innen kort tid uansett. Det kunne dermed blitt et problem om heisen hadde betydelig fler etasjer og personen som bestilte heisen på utsiden fikk sin ordre utsatt. Vi har valgt å ikke gjøre noe med "problemet" da denne heisen kun har fire etasjer, og systemet med prioritert retning gir fordeler i alle andre situasjoner. Gjennom testing av systemet har vi ikke funnet noen andre svakheter ved systemets oppførsel.

5.2 Alternative løsninger

Gjennom prosjektet har vi vært innom flere ulike løsninger på måter å løse ting på. Mye tid gikk med innledningsvis til å planlegge tilstander, klasser og sekvenser og vi hadde en god idé om hovedprogramflyten før vi startet med moduldesign. En alternativ rute til kø-systemet kunne vært å ha en sorterings-algoritme på et stort array som inneholdt en odretype, istedenfor valg-logikk på to mindre arrays.