Rapport heisprosjekt

August Bjørlo Vetle Rasmussen

Mars 2021

Innholdsfortegnelse

1 Overordnet arkitektur **2**
1.1 Klassediagram . . . 2
1.2 Sekvensdiagram . . . 3
1.3 Tilstandsdiagram . . . 5

2 Moduldesign **5**
2.1 hww . . . 5
2.2 elevator . . . 5
2.3 door . . . 6
2.4 rp . . . 6

3 Testing **6**
3.1 Oppstart . . . 6
3.2 Håndtering av bestillinger . . . 6
3.3 Bestillingslys- og etasjelys . . . 7
3.4 Heis-dør . . . 7
3.5 Sikkerhet . . . 7
3.6 Robusthet . . . 8

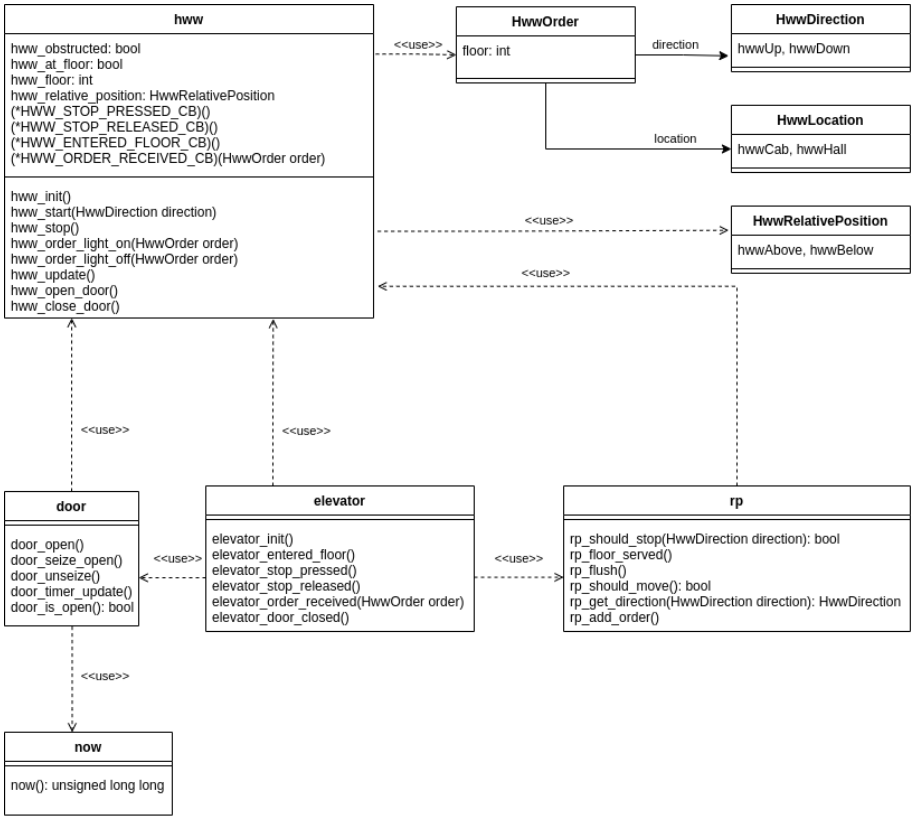
4 Diskusjon **8**

5 Konklusjon **8**

1 Overordnet arkitektur

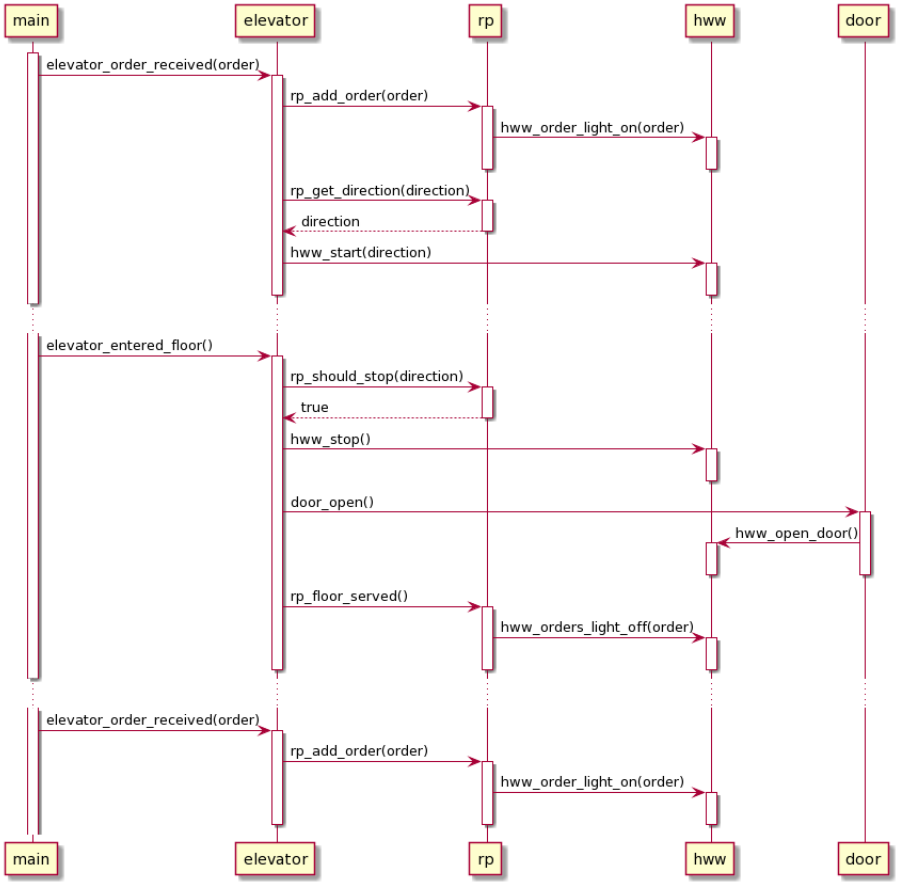
Helt overordnet er heisens styresystem delt i tre deler: et grensesnitt mot hardware, en tilstandsmaskin og hjelpemoduler for tilstandsmaskinen. Hardwaregrensesnittet er implementert i modulen hww (forkortelse for hardware.h wrapper). Wrapperen gjemmer bort polling-logikken og gir istedenfor muligheten til å definere callbacks for de ulike hendelsene som kan oppstå. hww fører til mer overhead, men gir et ryddigere grensesnitt og tydeligere innganger (entry points) for de øvrige modulene. Modulen elevator er implementasjonen av tilstandsmaskinen, og tar seg av den enkleste logiske styringen, men bruker hjelpemodulene door og rp (route planner) for mer kompliserte oppgaver relatert til styring av dør og behandling av bestillinger.

1.1 Klassediagram

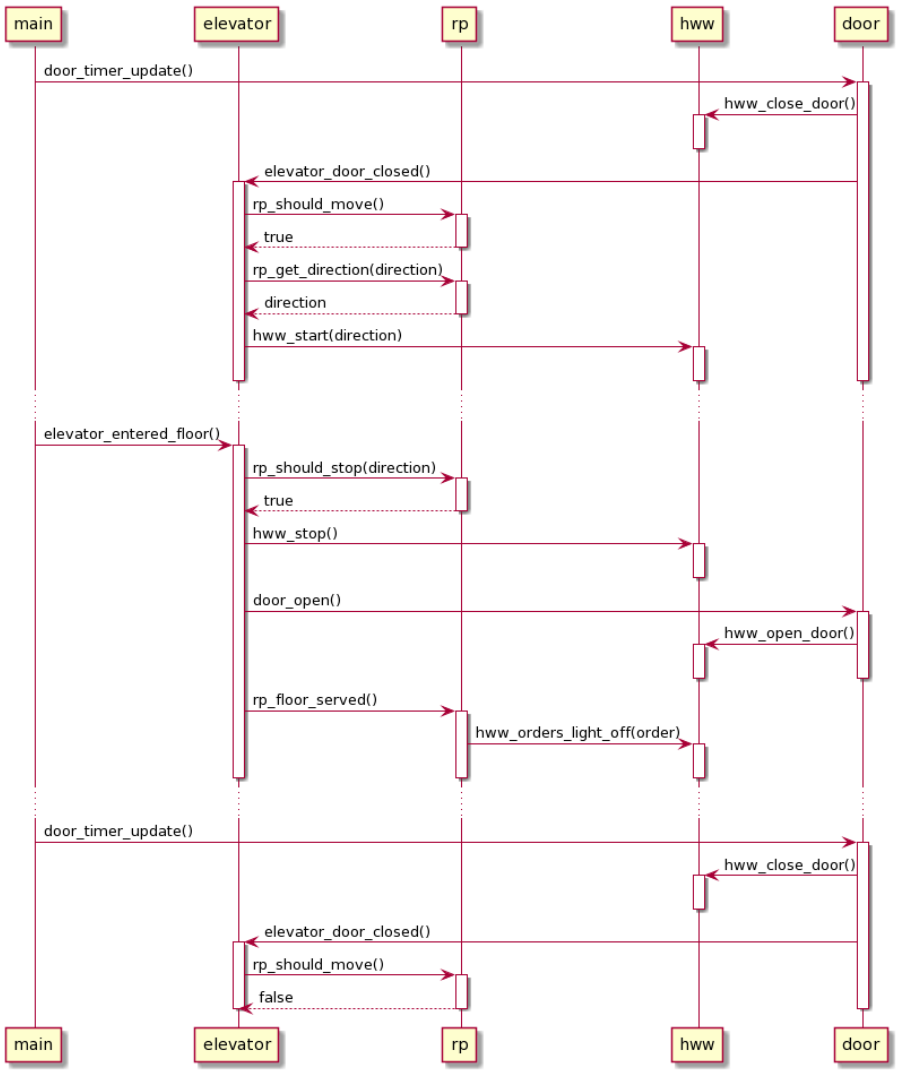


Figur 1: Klassediagram tegnet med app.diagrams.net

1.2 Sekvensdiagram

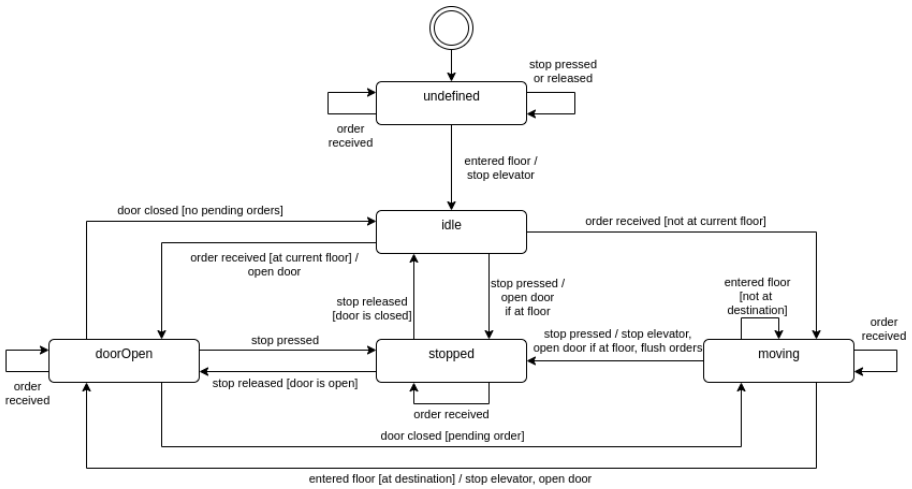


Figur 2: Sekvensdiagram del en tegnet med plantuml.com. Sekvensene hopper over lesing av tilstander fra hww.



Figur 3: Sekvensdiagram del to tegnet med plantuml.com. Sekvensene hopper over lesing av tilstander fra hww.

1.3 Tilstandsdiagram



Figur 4: Tilstandsdiagram tegnet med app.diagrams.net. Diagrammet gjenspeiler elevator-modulen

2 Moduldesign

Informasjonsutvekslingen mellom modulene gjøres ved hjelp av funksjoner. Dette gir løsere kobling, og dermed enklere debugging og bedre modularitet. Hardwaretilstander som obstruksjon er globale variable, men når sentrale tilstander som stoppknapp og etasjesensorer endres, er det definert callbacks. Disse callbackene kan defineres som en selv vil, men i denne implementasjonen er callbackene definert til å kalle events i tilstandsmaskinen elevator.

2.1 hww

hww (hardware.h wrapper) er som nevnt en callback-basert wrapper for hardware. Sentralt er funksjonen hww_update() som leser alle inputs, setter globale tilstander, kaller callbacks og setter bl.a. stopplys. Modulen inneholder også noen typedefs, som blant annet enn mer komplett bestillingstype, og noen enums.

2.2 elevator

elevator er en tilstandsmaskin, og utgjør hoveddelen av styringssystemet. Figure 4 viser de fem tilstandene elevator kan ha, og overgangene mellom dem. Fordelen med å bruke en tilstandsmaskin til hovedstyringen av systemet er at det er "lett" å få den til å speile spesifikasjonen. Det har også gjort det enklere å debugge fordi ulovlige overganger kan lett håndteres og varsles.

2.3 door

door er også en tilstandsmaskin og tar seg av logikken forbundet med døren. Modulen har funksjoner elevator (eller andre) kan kalle for å åpne døra, men door holder selv kontroll på når døra lukkes, basert på tid og obstruksjon. I tillegg har også modulen offentlige funksjoner for å ignorere tidstaking under for eksempel nødstopp. Når døren lukkes kalles elevator_door_closed() slik at elevator kan fortsette å betjene bestillinger. I likhet med hww har også door en update()-funksjon som kalles i main loop for å sjekke om det er på tide å lukke døren.

2.4 rp

rp (route planner) håndterer bestillinger, og kan instruere elevator hvor den skal stoppe og kjøre. Modulen håndterer dessuten bestillingslys. Bestillinger lagres i et array med fast størrelse og behandles som en FIFO-kø. Det er mange muligheter for å lagre bestillinger, men siden hastighet ikke har så mye å si i C, velges array som er et av de enkleste alternativene. rp prioriterer bestillinger sånn at heisen bytter retning minst mulig. Det betyr at heisen kun stopper om en bestilling er i samme retning som heisen allerede kjører eller er den siste bestillingen i nevnte retning.

3 Testing

3.1 Oppstart

Vi sikret at punkt O1 var tilfredsstilt ved å starte opp programmet med heisen i litt forskjellige posisjoner (mellom etasjer, i en etasje etc.). Hvis heisen var mellom to etasjer, så kjørte heisen ned til neste etasje, og hvis heisen var i en etasje stod den stille.

Punkt O2 ble sikret ved å sette mange bestillinger mens heisen var i mellom to etasjer ved oppstart. Heisen ignorerte alle bestillingene og forholdt seg til vanlig oppstartsprosedyre.

3.2 Håndtering av bestillinger

For å teste punkt H1 satte vi så mange bestillinger som mulig, og satte nye bestillinger etter hvert som bestillinger ble betjent. Punktet ble sikret ettersom alle bestillinger ble tatt.

For å teste punkt H2 opprettet vi en intern ordre til en øvrig etasje, og deretter opprettet vi en ekstern ordre med retning ned i etasjen under den øvrige. Heisen ignorerte den eksterne ordren på veg opp som forventet.

H3 sikret vi ved å sette ekstern bestilling både ned og opp i en etasje. Når heisen stoppet i etasjen, så ble begge ordrene ekspedert.

H4 sjekket vi ved å sette bestillinger, og se at heisen stod stille etter at alle bestillinger var utført.

3.3 Bestillingslys- og etasjelys

Testet punkt L1 ved å sette interne og eksterne bestillinger, for deretter å se at bestillingslys ble slukket ved utførte bestillinger.

Punkt L2 sikret vi ved å se at bestillingslys kun ble slått på ved å opprette en bestilling, og å se at etter bestillingen ble ekspedert, så ble det tilhørende bestillingslyset slukket.

Punktene L3, L4 og L5 sikret vi ved å observere etasjelys imens heisen var i bevegelse. Vi så at kun ett etasjelys var tent om gangen, og at hvilket etasjelys dette var endret seg når heisen ankom en ny etasje.

For å teste stopplyset i L6 holdt vi inne stoppknappen, og sjekket at lyset kun var slått på så lenge vi holdt inne knappen.

3.4 Heis-dør

D1 sikret vi ved å påtrykke en bestilling og se heisdøra åpnet seg i 3 sekunder da heisen ankom bestillingens tilhørende etasje.

Vi sikret at heisdøra var lukket med mindre noe skjedde (bestilling, stopp etc.) ved å se at etter en utført bestilling, så var døra lukket. Dette sikret punkt D2.

Sikret punkt D3 ved å holde inne stoppknappen mens heisen var i en etasje og se at døren var åpen så lenge knappen var trykket, pluss 3 sekunder etterpå.

For å sikre punkt D4, så åpnet vi døra ved å trykke stoppknappen mens heisen var i en etasje, og deretter satte vi obstruksjonsbryteren til å være aktiv. Vi observerte så at heisdøra forholdt seg åpen så lenge bryteren var aktiv, pluss 3 sekunder etterpå.

3.5 Sikkerhet

Vi sikret punkt S1 ved å sette mange bestillinger imens døra var åpen. Så lenge døra var åpen, så stod heisen stille.

For å sikre punkt S2, stoppet vi heisen utenfor en etasje og så at heisdøra forble lukket. Dette var det eneste potensielle tilfellet hvor heisdøra kunne åpnet seg utenfor en etasje.

Punkt S3 sikret vi ved å observere at heisen aldri kjørte seg fast, hverken i øvre eller nedre ende av heisbanen.

For å sikre punkt S4, så trykket vi stoppknappen mens heisen var i bevegelse, og så at heisen stoppet momentant.

S5 sikret vi ved å sette mange bestillinger, for så å trykke på stoppknappen. Dette fjernet alle bestillingene.

For punkt S6, så holdt vi inne stoppknappen mens vi påtrykket bestillinger. Alle forsøk på bestillinger ble ignorert, dermed var dette punktet sikret.

Punkt S7 sikret vi ved å påtrykke stoppknappen og deretter se at heisen stod stille til vi satte bestillinger.

3.6 Robusthet

Vi testet punkt R1 ved å sette obstruksjonsbryteren aktiv imens heisen beveget seg mellom 2 etasjer. Obstruksjonsbryteren hadde kun innvirkning på heisen etter at heisdøra åpnet seg ved at døra forholdt seg åpen så lenge bryteren var aktiv.

Vi testet punkt R2 indirekte ved at gjennom alle andre tester vi hadde gjort, så krasjet aldri programmet. For å teste det direkte, holdt vi programmet gående i lang tid imens vi påtrykket svært mange bestillinger. Siden programmet aldri krasjet, så var punktet tilfredsstilt.

Punkt R3 sikret vi ved å sette en bestilling etter at heisen hadde kommet til en definert tilstand ved oppstart. Ettersom heisen alltid visste hvilken retning den skulle gå for å ekspedere bestillingen, så må den nødvendigvis også ha visst hvor den var opprinnelig.

4 Diskusjon

Implementasjonen bærer preg av en del boilerplate, spesielt i forbindelse med tilstandsmaskinene. I elevator er det utvilsomt verdt det (som nevnt i subsection 2.2), men door kunne sannsynligvis vært implementert på halvparten linjene med mer "tradisjonell" kode.

I elevator er det noe overlapp mellom tilstandene, noe som strider imot konseptet DRY (don't repeat yourself). Dette kunne muligens vært løst med enter/exit-handlinger i tilstandene, men det er ikke åpenbart hvordan dette implementeres i C uten mye boilerplate.

Den lengste modulen i prosjektet er hww. Dette er også et tydelig eksempel på boilerplate, ettersom hww ikke hadde vært strengt nødvendig.

5 Konklusjon

Prosjektet har tydelig demonstrert hvor viktig nøye planlegging i starten av et prosjekt er. Etter flere implementasjonsiterasjoner er det liten tvil om det.