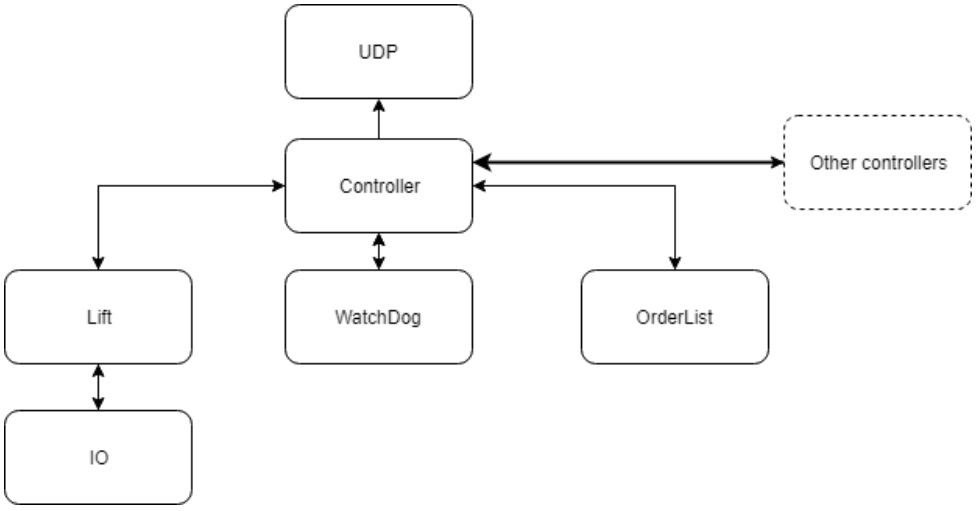
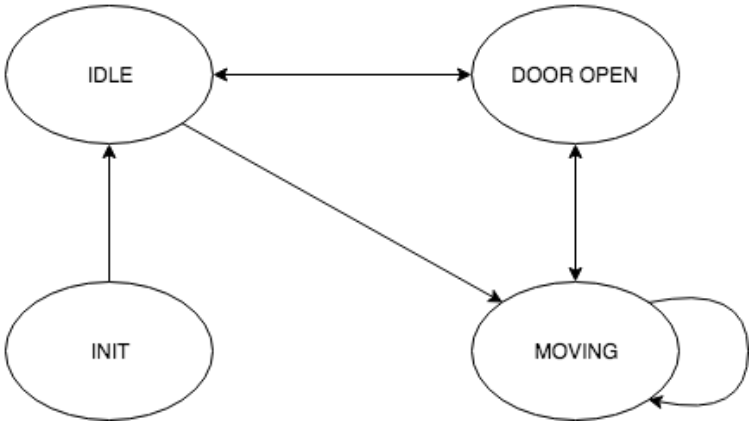
Designpresentasjon notater

Information flow diagram:



Lift FSM:



<u>Transitions Lift FSM:</u>

- INIT -> IDLE: Floor_sensor går høy
- IDLE -> MOVING: Det eksisterer en ordre i en annen etasje
- IDLE -> DOOR_OPEN: Det er en ordre i etasjen heisen har stått i
- DOOR_OPEN -> IDLE: Ingen ordre eksisterer, ordreliste er tom
- MOVING -> DOOR_OPEN: Etasjesensor høy, det eksisterer en ordre i denne etasjen
- DOOR_OPEN -> MOVING: Ny ordre i en annen etasje, ikke-tom ordreliste utenom etasjen heisen står i.
- MOVING -> MOVING: Heisen har ankommet en etasje, med en ordre lenger oppe/nede. Heisen går forbi.

- **Bestille heis med knappetrykk:**
    - Knapp gir beskjed til sin respektive controller Ci at den er trykket -- Controller.new_order(order)
    - Ci spør om kost for denne ordren fra alle controllere -- Controller.request_cost()
    - Etter svar fra alle på nettverket
        - Ci gir beskjed til Cj at Cj har fått ordren -- Controller.assign_order(order):
        - Cj acker til Ci
        - Ci gir beskjed til Ck om å starte timer
        - Ck acker til Ci
        - Ci sender infomelding til alle (logikk i mottaker bestemmer om lys skal på eller ikke) -- Controller.order_bid_completed(order)
            - Hvis cab call, kun ett lys skrus på
            - Hvis hall call, lys hos alle heiser på nettverket skrus på

- **Legge en ordre ut på anbud med kostfunksjon:**
    - Controller Cj mottar forespørsel om kost fra controller Ci (i kan være lik j) -- Controller.request_cost()
    - Sjekker om denne ordreIDen (unik datastruktur med node og ID) er blitt utført tidligere -- OrderList.order_complete?(order):
        - Hvis ja, send en “abort”-melding tilbake. Dette kan skje ved nettverksfeil eller at en timer ikke fikk med seg at ordren ble utført.
        - Ellers, regn ut kost på vanlig måte (hvis cabcall vil alle andre gi uendelig kost)
    - Cj sender melding til Ci (med kost eller “abort”)

- **Timer går ut:**
    - En timer skal passe på en ordre -- WatchDog.add_timer(order, node_id)
    - Timer går ut før bekreftelse på utført ordre mottatt Controller.new_order(order).
        - Hvis ordren er en cab call, sjekker WatchDog at noden cab callen tilhører, er på nettverket.
            - Hvis ja, fortsett.
            - Hvis nei, sjekk nodelisten og vent til noden er på nettverket.
        - Hvis ordren er en hall call, fortsett.
    - Legger ordren ut i kostfunksjonen på ny med ordreID.
    - Vent på bekreftelse på at ordre er delegert til en ny WatchDog. (reply fra Controller.new_order(order))

- **Network partition**
    - Nettverket deles opp i to clusters.
    - Nodelista vil oppdage at nettverket forsvinner.
    - En ordre i en cluster med en watchdog i det andre clustere vil bli lagt ut på anbud på ny.
    - Watchdogen vil oppdage det samme, og vil legge ut en hall call på ny. En evt cab call vil legges på vent til riktig node ligger i nodelista.

- **Oppstart av node:**
    - Controller initialiserer systemet.
    - Controlleren lytter på UDP etter et nettverkscluster å koble seg opp mot. -- UDP.listen()
    - Timer starter:
        - Hvis timer > *threshold* [sec], operer heisen som vanlig, men med kun en på nettverket.
        - Hvis nettverk oppdages når timer < *threshold* [sec], kobler noden seg på nettverket og operer som vanlig.

- **Heis ankommer etasje:**
    - IO sender melding til Lift-- IO.at_floor.
    - Liften spør controller om den skal stoppe -- Controller.get_direction().
    - Controller sjekker ordreliste -- OrderList.orders_at_floor?(floor, direction).
    - Controller sender svar til lift -- IO.set_direction(state):
        - Hvis ja, ordren slettes fra ordrelista -- OrderList.order_done(order)
        - Lys må håndteres og beskjed må sendes til andre noder --Controller.order_done(order)
    - Oppdatere floor, direction (Internal state)

Module	Interface	Events
Controller	<u>Internt i node:</u> get_direction(floor) *reply* new_order(order) *reply* <u>Mellom noder:</u> assign_order(order) *reply* assign_watchdog(node,order) request_cost() *reply* order_bid_completed(order) *reply* order_done(order)	
Lift	respond_IO_event() *reply* set_light_button(button_type,floor) *no_reply*	
IO	set_light(light) *no_reply* set_door(state) *reply* set_direction(state) *reply*	at_floor button_pushed door_closed
OrderList	add_order(order) *reply* order_done (order) *reply* orders_at_floor? (floor,direction) *reply* get_all_orders() *reply* order_complete?(order) *reply*	first_order_added
WatchDog	add_timer (order, node_id) *reply* order_complete (order)	re-inject_order
UDP	start_link(node_id) *reply* listen() *no_reply* send() *no_reply*	

(hva kan jeg gjøre for andre / få som melding fra andre?)
(hva kan skje hos meg/hva genereres hos meg?)

Fault tolerance:
Watchdogs, periodisk consistency check av order_list -- er watchdogen til denne ordren fortsatt oppe, time-stamp + kryssjekk av utført orderID

Begrunnelse for nettverk:

- Etter vår kravspec er det ikke nødvendig å designe for skalering for over 30 heiser.

- TCP-forbindelse mellom alle noder (alle til alle) og dermed tar det lang tid å legge til ny node.
- Det er heartbeat-meldinger mellom BEAM'ene i clusteret, når man ikke får reply fra en antar resterende at denne BEAMen man ikke får svar fra ikke finnes, og vil fjernes fra clusteret. Det eneste vi må implementere, er logikk for at ulike clusters finner hverandre og kan koble seg opp igjen.
- Kontant sendehastighet ved at man i stedet for å senke hastighet ved tap, så antar man at noden er tapt og at man derfor heller kan prøve å fange den opp igjen (restart).

Presentasjon:
https://docs.google.com/presentation/d/1g_avcnsCrwCrFQTU2uZ-0qDFhutpMK_i_9cuFLBrzII/edit?usp=sharing