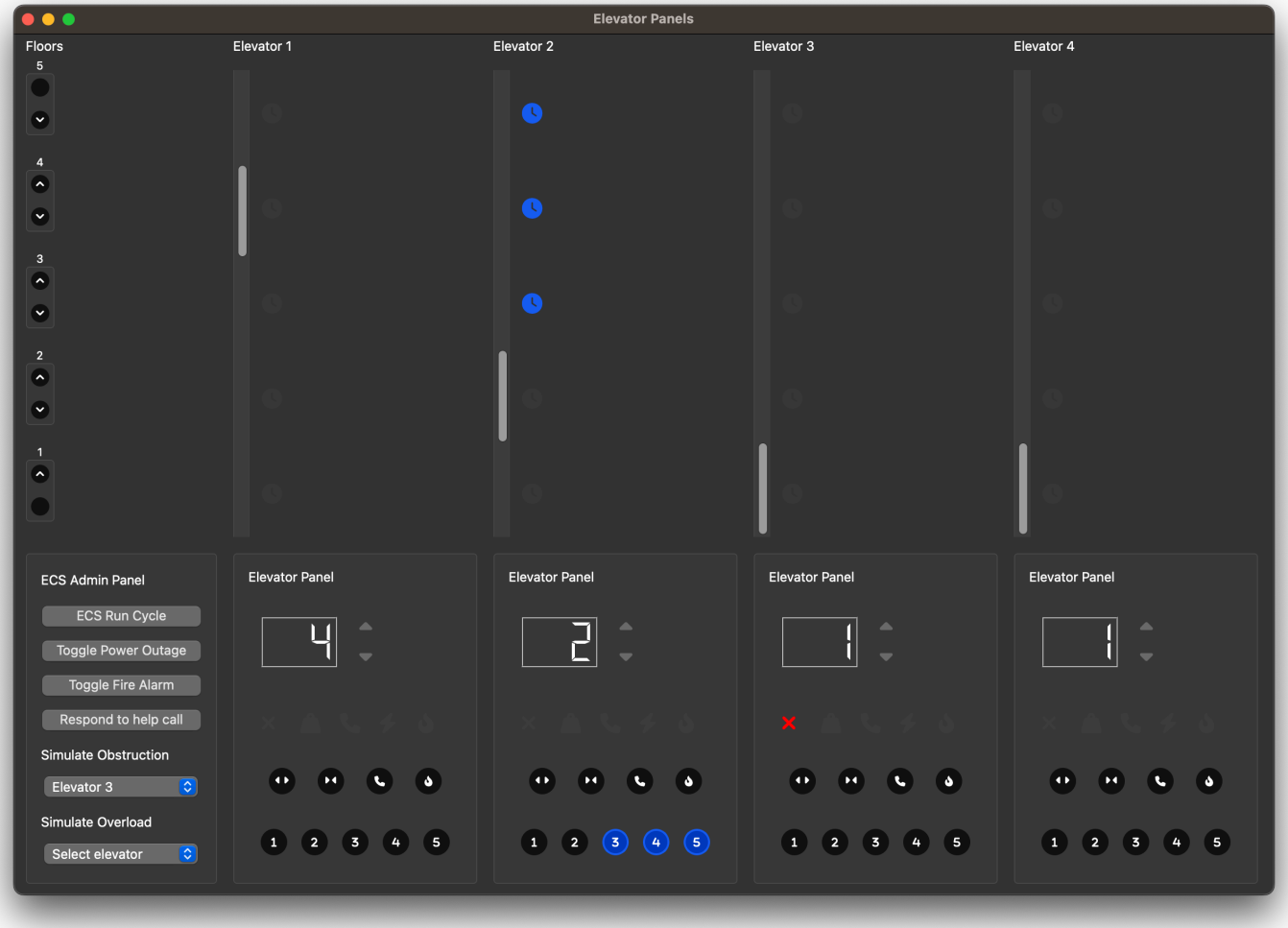
COMP 3004 Assignment 3

Name: Abdullah Mostafa
Student Number: 101008311
Date: Oct 15, 2023

Table of Contents

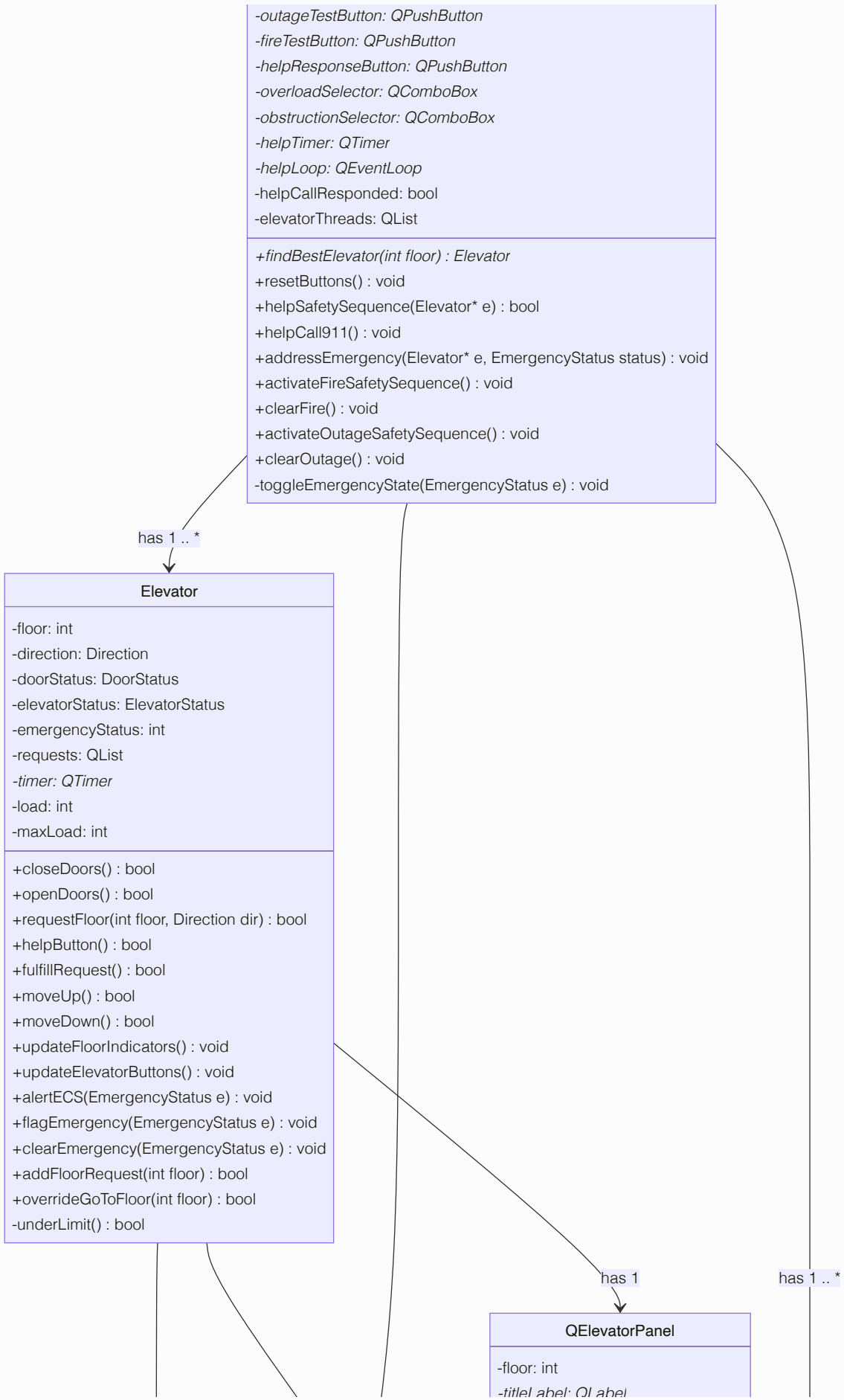
Elevator State Diagram

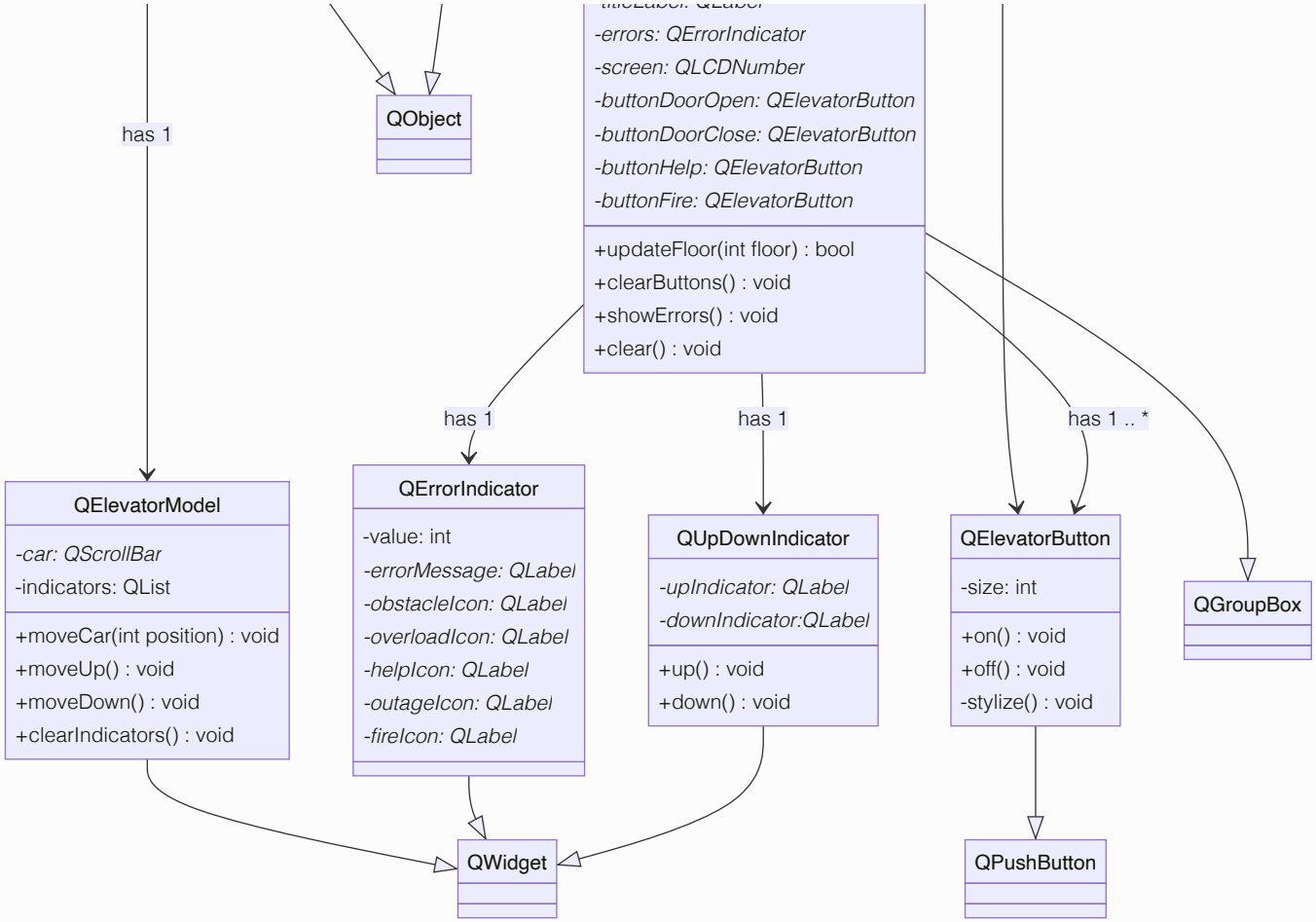
Elevator Control System Simulator



UML Class Diagram

ElevatorControlSystem
-numFloors: int -emergencyStatus: int


-outageTestButton: QPushButton
-fireTestButton: QPushButton
-helpResponseButton: QPushButton
-overloadSelector: QComboBox
-obstructionSelector: QComboBox
-helpTimer: QTimer
-helpLoop: QEventLoop
-helpCallResponded: bool
-elevatorThreads: QList
+findBestElevator(int floor) : Elevator
+resetButtons() : void
+helpSafetySequence(Elevator* e) : bool
+helpCall911() : void
+addressEmergency(Elevator* e, EmergencyStatus status) : void
+activateFireSafetySequence() : void
+clearFire() : void
+activateOutageSafetySequence() : void
+clearOutage() : void
-toggleEmergencyState(EmergencyStatus e) : void
has 1 .. *
Elevator
-floor: int
-direction: Direction
-doorStatus: DoorStatus
-elevatorStatus: ElevatorStatus
-emergencyStatus: int
-requests: QList
-timer: QTimer
-load: int
-maxLoad: int
+closeDoors() : bool
+openDoors() : bool
+requestFloor(int floor, Direction dir) : bool
+helpButton() : bool
+fulfillRequest() : bool
+moveUp() : bool
+moveDown() : bool
+updateFloorIndicators() : void
+updateElevatorButtons() : void
+alertECS(EmergencyStatus e) : void
+flagEmergency(EmergencyStatus e) : void
+clearEmergency(EmergencyStatus e) : void
+addFloorRequest(int floor) : bool
+overrideGoToFloor(int floor) : bool
-underLimit() : bool
has 1
has 1 .. *
QElevatorPanel
-floor: int
-titleLabel: QLabel



Traceability Matrix

Id	Requirement	Related Use Case	Fullfilled By	Description	Tested By
01	Elevator responds to external floor buttons	Passenger uses elevator	`Elevator::requestFloor(int Floor, Direction d)`	Allows the elevator to respond to external floor buttons	Pressing an external button and seeing if the elevator comes to that floor
02	Elevator responds to internal floor buttons	Passenger uses elevator	`Elevator::requestFloor(int Floor)`	Allows the elevator to respond to floor buttons in the elevator	Pressing a button on the elevator panel and seeing if the elevator goes to that floor
03	Elevator moves between floors	Passenger uses	`Elevator::fulfillRequest()`	This ensures the elevator car can	Engage with the elevator by calling it and go to a

		elevator		move between one floor to another by handling the requests	different floor to see if it moves
04	Elevator can open its doors	Passenger uses elevator	`Elevator::openDoors()`	Ensures elevator can open its doors	- Call an elevator by pressing an external floor button. - Use the door open button.
05	Elevator can close its doors	Passenger uses elevator	`Elevator::closeDoors()`	Ensures elevator can close its doors	- Engage with the elevator and see if the door closes after. - Use the door close button.
06	Elevator goes to ground floor during fire alarm	Fire alarm from building	`ElevatorControlSystem::activateFireSafetySequence()`	Allows for proper safety protocol during a fire alarm	- Press the fire button inside the elevator. - Trigger the building fire alarm through the ECS admin panel
07	Elevator goes to ground floor during power outage	Power outage in the building	`ElevatorControlSystem::activateOutageSafetySequence()`	Allows for proper safety protocol during a fire outage	Trigger building power outage through the ECS admin panel
08	Help button triggers emergency response	Control system receives Help signal	`ElevatorControlSystem::helpSafetySequence()`	Allows the system to respond to help calls	Help button is pressed in the elevator panel. - Can test it by not responding to the call and seeing if the ECS dials 911. - You can also test it by responding to the call from the ECS panel to address the emergency
09	Door obstacle prevents door from closing	Door obstacle detected	`Elevator::closeDoors()`	Ensures doors don't close when there is something	Engage with any operation that closes the elevator doors, i.e press door close button.

				obstructing the door from closing	- Trigger a simulated door obstacle in the ECS Admin Panel to see if doors don't close and the panel shows the error
10	Overload limit in elevator	Overload detected	`Elevator::underLimit()`	Ensures the elevator doesn't move when it surpasses the load limit	Engage in any elevator operation where the door closes. - Trigger a simulated overload in the ECS Admin Panel to see if the elevator responds accordingly and shows the error
11	Update display floor number	Many	`QElevatorPanel::updateFloor(int floor)`	Ensures the elevator's display shows the according floor number	Engage with the elevator in any way that moves it up or down and see if the floor updates in the panel
12	Show warnings on the display	many	`QErrorIndicator::setErrorCode(int error)`	Ensures elevator's display shows the appropriate warning to the car passengers	Trigger each of the different emergency scenarios in the ECS admin panel or from the elevator panel and see if the error shows in the elevator car
13	Ring bell	many	`Elevator::fulfillRequest()`	Allows the elevator to ring a bell upon floor arrival	Have the elevator move to any floor and see if it dings
14	Illuminate floor buttons when pressed	Passenger uses elevator	`QElevatorButton::on()` and `QElevatorButton::off()`	Allows for the system to indicate the request has been put in	Interact with any floor button and see if it illuminates
15	Turn off Illuminated floor buttons when request is fulfilled	Passenger uses elevator	`Elevator::updateElevatorButtons()` and `Elevator::updateFloorIndicators()`	Allows for the system to clear fulfilled requests	Have the elevator go to the floor of the pressed button and see if the button illumination turns off

Design Patterns Used

Throughout the design of the Elevator controller simulator's a couple different design patterns were used to make the design more robust and organized.

The `ElevatorControlSystem` (ECS) is implemented as a Singleton Pattern, there is exactly one instance of the ECS class. This allows for a single point of control for all the elevators.

The way the elevators are controlled follows an Observer Pattern and a Command Pattern design. The ECS observes all the states of the different elevators, sensors, buttons etc and then commands each entity on the next instruction.

Use Case for Elevator Control System

Use case: Using an Elevator

Primary Actor: Passenger

Scope: Elevator control system

Level: User goal

Stakeholders and interests:

- Passenger: Wants to ride the elevator to a different floor
- Control System: Responds to the elevator requests and overlooks safety systems
- Building management: Ensures the elevator operates safely and handles emergencies if they were to occur

- Safety service: Provides help in emergency situations

Pre-conditions:

- The elevator is operational

Post-conditions:

- The elevator arrives at the floor that the passenger requested

Minimal Guarantees:

Success Guarantees:

Trigger: The passenger calls the elevator with the Up/Down button on the floor they want to move away from

Main Success Scenario:

1. The passenger presses Up/Down button on the floor to request the elevator
    a. The pressed button illuminates
    b. A request signal is sent to the control system to bring the car down to that floor
2. The elevator starts moving to the requested floor if it is not already there
    a. The bell rings when the elevator car arrives
    b. The floor indicator on the screen updates to the current floor
    c. The elevator announces what floor it is on
    d. The up/down button light on the floor to request the elevator turns off
    e. The doors on the elevator's car opens and remains open for 10 seconds
3. The passenger enters the elevator
    a. The passenger presses the floor they want to go to

b. The light on the pressed floor illuminates

c. The bell rings and the doors close

4. The elevator moves to the requested floor

    a. The bell rings when the elevator car arrives

    b. The floor indicator on the screen updates to the current floor

    c. The elevator announces what floor it is on

    d. The floor button light that was pressed turns off

    e. The doors on the elevator's car opens and remains open for 10 seconds

5. The passenger exits on the floor they wanted to go to

Extensions:

6. Help button is pressed

    a. Control system receives a help request

    b. Passenger is connected to the safety service through a voice connection

    c. If no response from the passenger or safety service within 5 seconds, 911 is called

7. Door obstacle is detected

    a. An audible alert is played

    b. The door is prevented from closing until the obstacle is clear from the door

8. Fire alarm was set off in the building or in the elevator

    a. The control system instructs the elevator to go to the nearest safest floor

    b. The elevator plays a warning message and shows it on the screen

9. Overload is detected

    a. An audible alert is played and the screen updates

b. The door is prevented from closing until the overload sensor detects a safe load

10. Door open button is pressed

    a. The doors remain open for another 10 seconds or until the button isn't being pressed anymore

11. Door close button is pressed

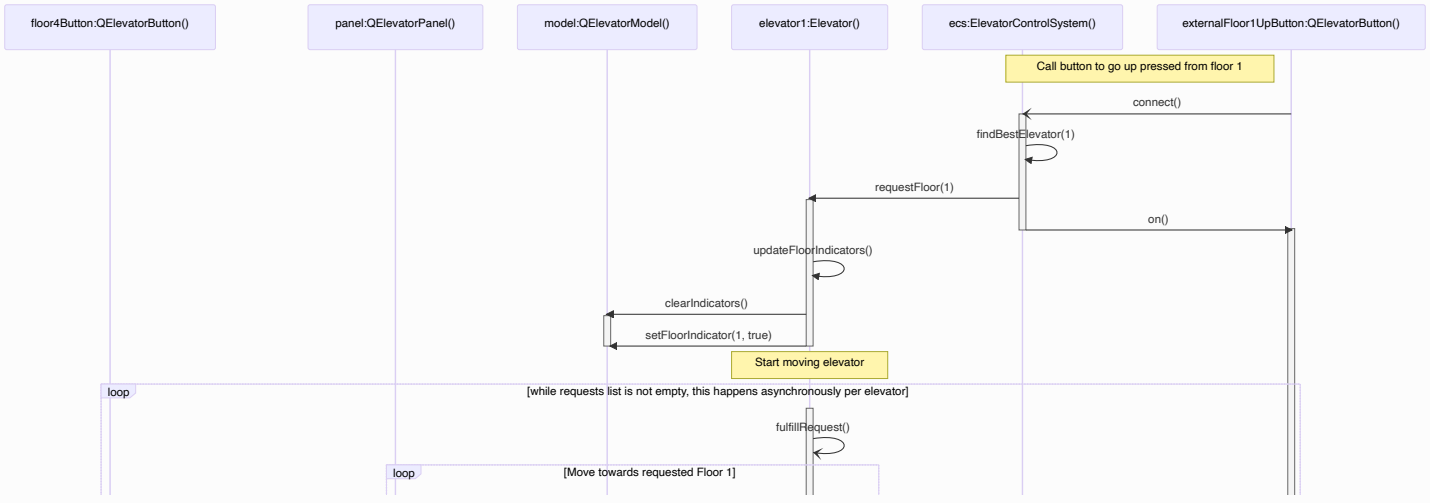
    a. The doors attempt to close right away

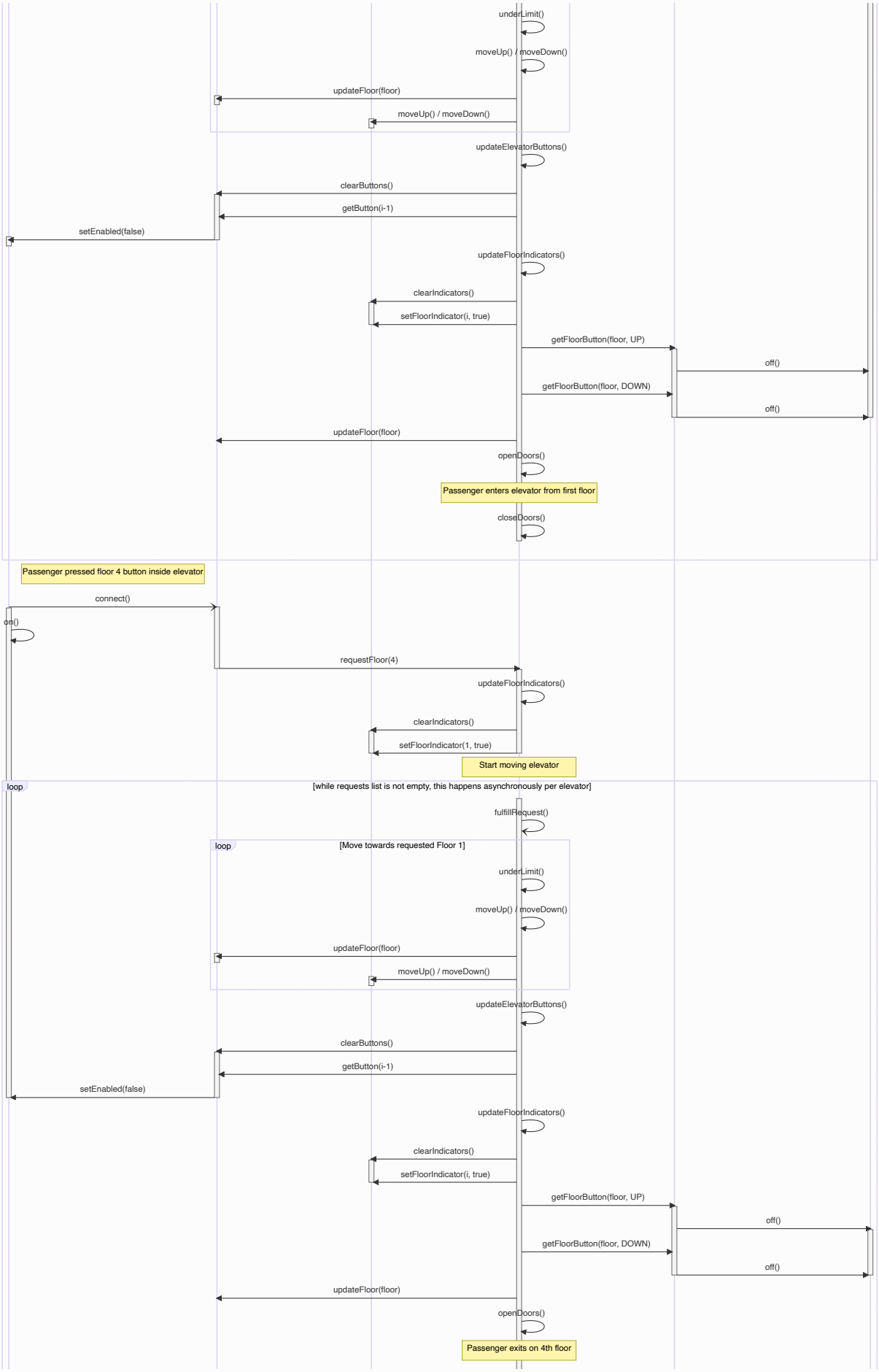
Sequence Diagrams

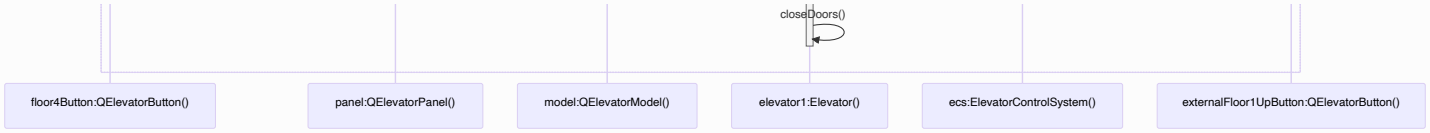
Success Scenario 1: Passenger requests elevator and rides it to floor *f*

- Passenger calls the elevator from an arbitrary floor *f*, the ECS will find the nearest available elevator `findBestElevator()` and add floor *f* to that elevator's requests array `QList<int> requests`. The Elevator will instruct itself to move through `fulfillRequest()`, continuously calling this method until the request is complete. Each time this method is called the elevator will move one floor if there is a request. A request is completed when the elevator arrives at the floor of the request, at this point the doors will opens.
- When the elevator reaches the floor of which the button was pressed from, the passenger enters and the door closes after 10 seconds. At this point in time the elevator and ECS is still polling `fulfillRequest()` however nothing is done as the requests queue is empty

- When the passenger presses a floor button to go to a different (or same) arbitrary floor *f*, that floor is added to the `requests` array, the elevator to start moving with `fulfillRequest()` again until the elevator reaches the requested floor.
- This design allows for a robust system that encompasses different scenarios as well without altering the flow of the system. For example an edge case would what if a passenger calls the elevator and the elevator is already on that floor? Or alternatively the passenger enters on floor 1, the door closes and passenger presses floor 1? In this system, the floor is added to the `request` array as normal, the Elevator will try fulfill that request. When the elevator sees its already at the destination floor then the request is deemed complete, at which the doors will then open.


floor4Button:QElevatorButton()
panel:QElevatorPanel()
model:QElevatorModel()
elevator1:Elevator()
ecs:ElevatorControlSystem()
externalFloor1UpButton:QElevatorButton()
Call button to go up pressed from floor 1
connect()
findBestElevator(1)
requestFloor(1)
on()
updateFloorIndicators()
clearIndicators()
setFloorIndicator(1, true)
Start moving elevator
loop
[while requests list is not empty, this happens asynchronously per elevator]
fulfillRequest()
loop
[Move towards requested Floor 1]

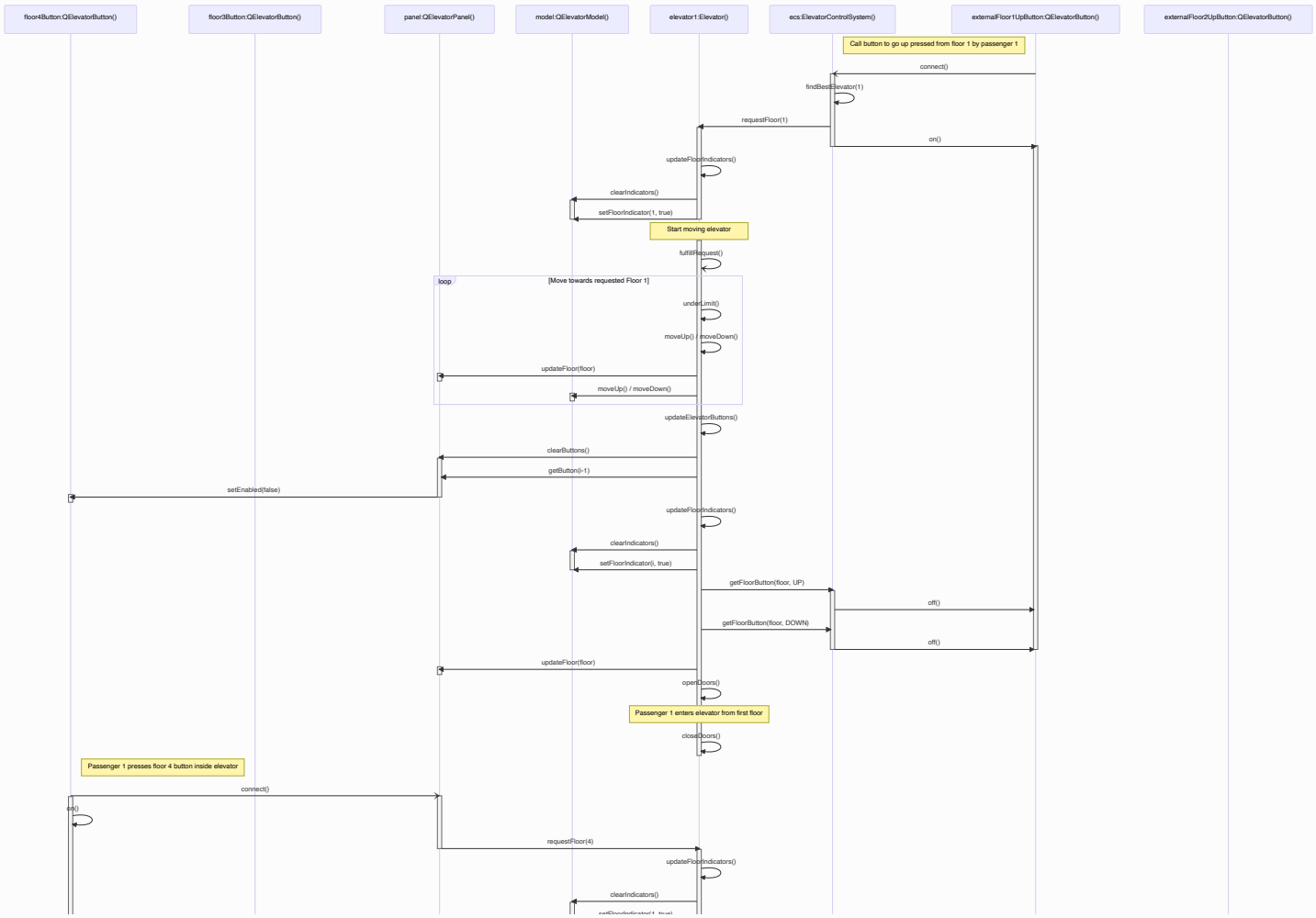

underLimit()
moveUp() / moveDown()
updateFloor(floor)
moveUp() / moveDown()
updateElevatorButtons()
clearButtons()
getButton(i-1)
setEnabled(false)
updateFloorIndicators()
clearIndicators()
setFloorIndicator(i, true)
getFloorButton(floor, UP)
off()
getFloorButton(floor, DOWN)
off()
updateFloor(floor)
openDoors()
Passenger enters elevator from first floor
closeDoors()
Passenger pressed floor 4 button inside elevator
connect()
on()
requestFloor(4)
updateFloorIndicators()
clearIndicators()
setFloorIndicator(1, true)
Start moving elevator
loop
[while requests list is not empty, this happens asynchronously per elevator]
fulfillRequest()
loop
[Move towards requested Floor 1]
underLimit()
moveUp() / moveDown()
updateFloor(floor)
moveUp() / moveDown()
updateElevatorButtons()
clearButtons()
getButton(i-1)
setEnabled(false)
updateFloorIndicators()
clearIndicators()
setFloorIndicator(i, true)
getFloorButton(floor, UP)
off()
getFloorButton(floor, DOWN)
off()
updateFloor(floor)
openDoors()
Passenger exits on 4th floor

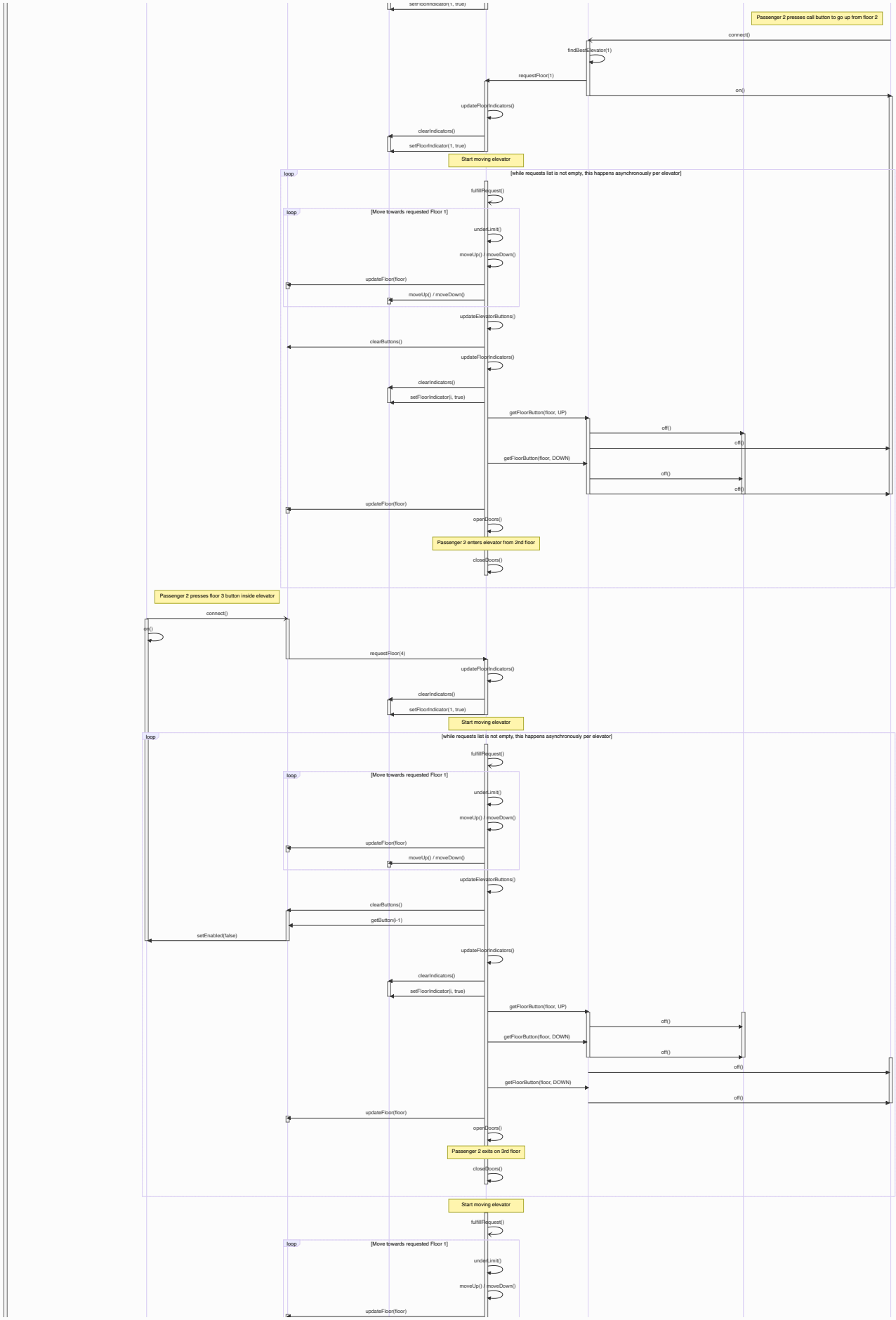

closeDoors()
floor4Button:QElevatorButton()
panel:QElevatorPanel()
model:QElevatorModel()
elevator1:Elevator()
ecs:ElevatorControlSystem()
externalFloor1UpButton:QElevatorButton()

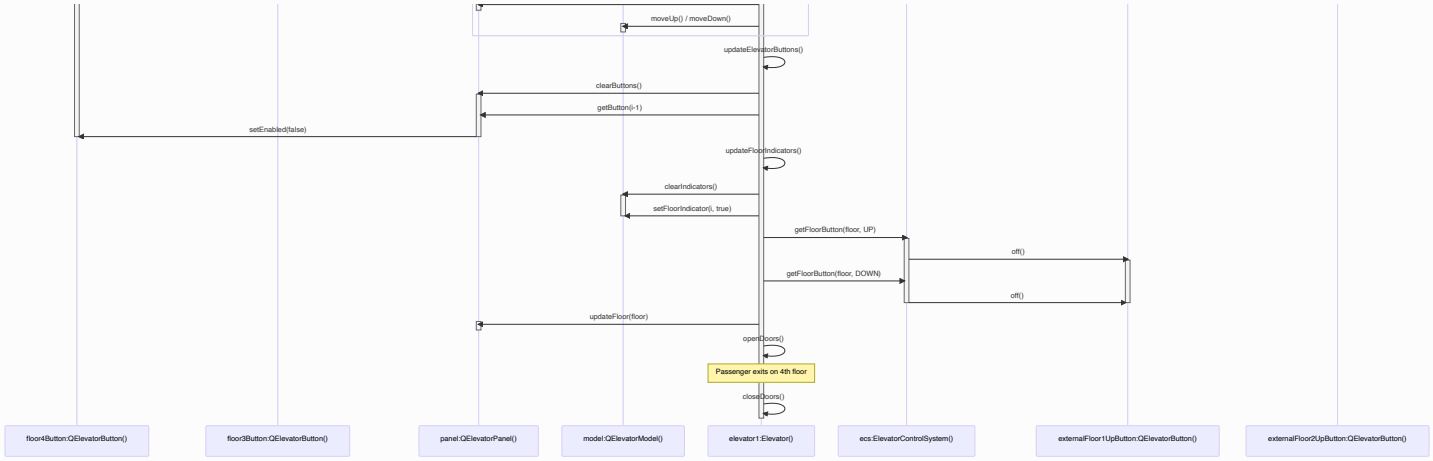
Success Scenario 2: Passenger *A* on 1st floor requests elevator to floor 4, at the same time Passenger *B* on 2nd floor requests elevator to go to floor 3.

- Note: This scenario builds off *Success Scenario 1*, please read description for that one first to have better understanding of how the system functions.
- This scenario assumes both Passengers use the same elevator

- When Passenger *A* gets on the elevator and chooses to go to the 4th floor, that is added to the elevator's `requests` array. The Elevator will then instruct itself to move floor by floor until it reaches the requested floor. `requests = [4]`
- During this time Passenger *B* calls the elevator from the 2nd floor. The Elevator adds the second floor to the `requests` so now `requests = [2, 4]`.
- The elevator reaches the second floor so 2 is removed from the `requests` array as that floor request has now been fulfilled. `requests = [4]`
- When passenger three chooses to press the button to the third floor, that is also added to the requests array, so now `requests = [3, 4]`.
- Now the elevator ascends to the 3rd, then fourth floor fulfilling those two floor requests.

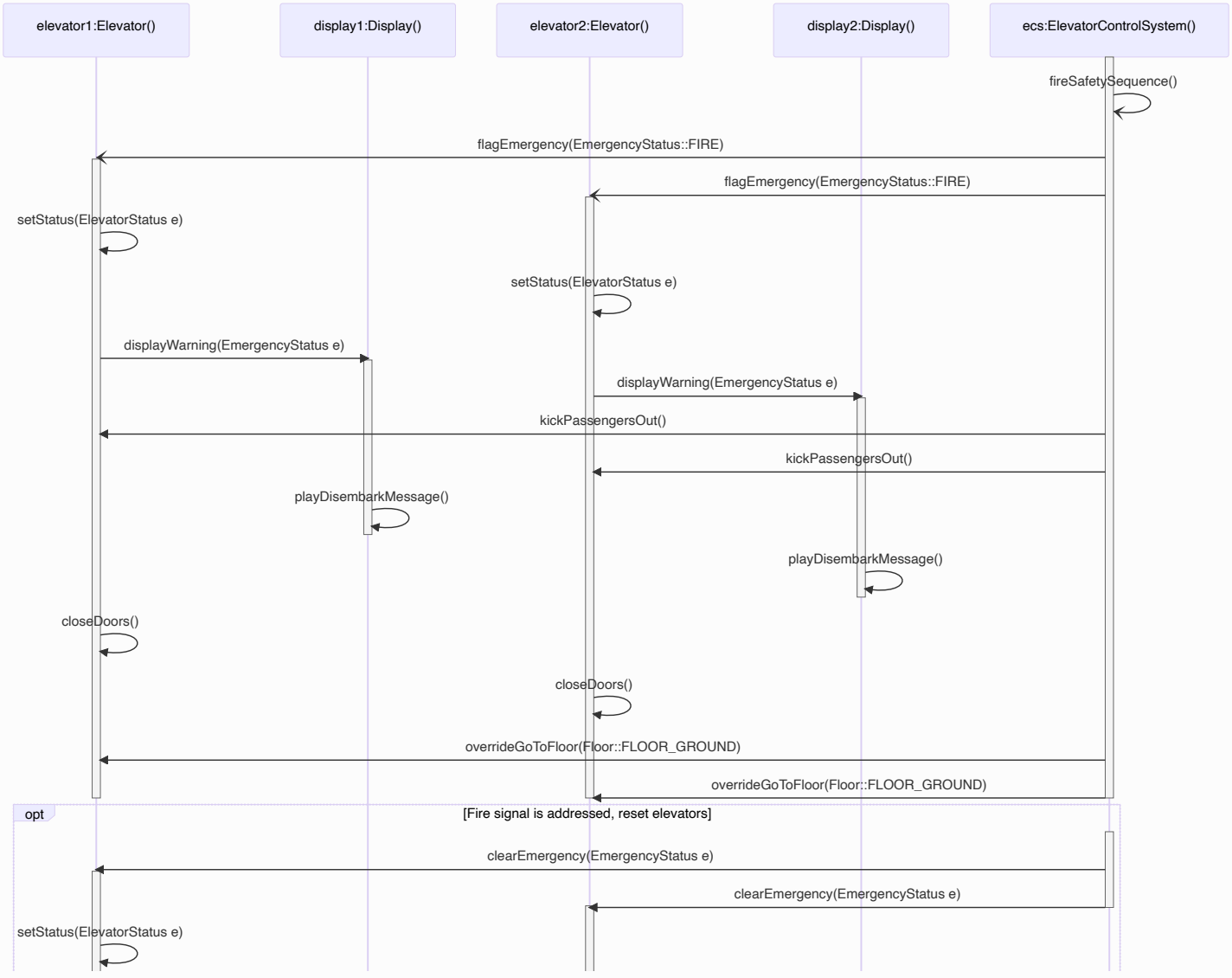

floor4Button:QElevatorButton()
floor3Button:QElevatorButton()
panel:QElevatorPanel()
model:QElevatorModel()
elevator1:Elevator()
ecs:ElevatorControlSystem()
externalFloor1UpButton:QElevatorButton()
externalFloor2UpButton:QElevatorButton()
Call button to go up pressed from floor 1 by passenger 1
connect()
findBestElevator(1)
requestFloor(1)
on()
updateFloorIndicators()
clearIndicators()
setFloorIndicator(1, true)
Start moving elevator
fulfillRequest()
loop
[Move towards requested Floor 1]
underLimit()
moveUp() / moveDown()
updateFloor(floor)
moveUp() / moveDown()
updateElevatorButtons()
clearButtons()
getButton(i-1)
setEnabled(false)
updateFloorIndicators()
clearIndicators()
setFloorIndicator(i, true)
getFloorButton(floor, UP)
off()
getFloorButton(floor, DOWN)
off()
updateFloor(floor)
openDoors()
Passenger 1 enters elevator from first floor
closeDoors()
Passenger 1 presses floor 4 button inside elevator
connect()
add()
requestFloor(4)
updateFloorIndicators()
clearIndicators()

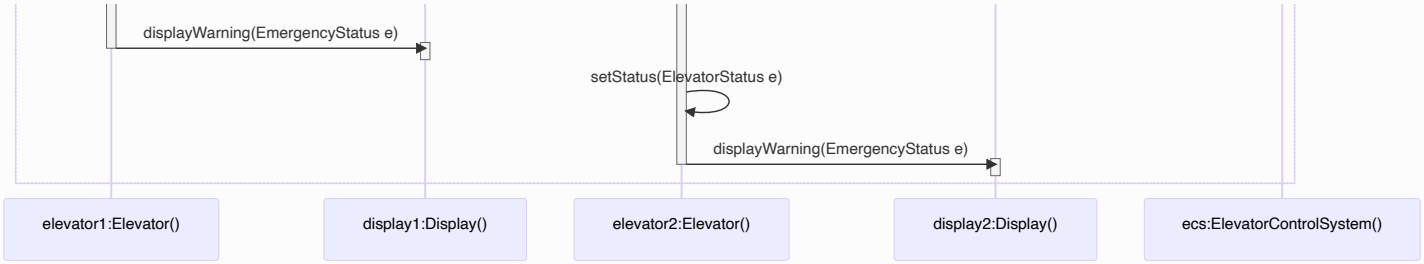

setFloorIndicator(1, true)
Passenger 2 presses call button to go up from floor 2
connect()
findBestElevator(1)
requestFloor(1)
on()
updateFloorIndicators()
clearIndicators()
setFloorIndicator(1, true)
Start moving elevator
loop
[while requests list is not empty, this happens asynchronously per elevator]
fulfillRequest()
loop
[Move towards requested Floor 1]
underLimit()
moveUp() / moveDown()
updateFloor(floor)
moveUp() / moveDown()
updateElevatorButtons()
clearButtons()
updateFloorIndicators()
clearIndicators()
setFloorIndicator(i, true)
getFloorButton(floor, UP)
off()
off()
getFloorButton(floor, DOWN)
off()
off()
updateFloor(floor)
openDoors()
Passenger 2 enters elevator from 2nd floor
closeDoors()
Passenger 2 presses floor 3 button inside elevator
connect()
on()
requestFloor(4)
updateFloorIndicators()
clearIndicators()
setFloorIndicator(1, true)
Start moving elevator
loop
[while requests list is not empty, this happens asynchronously per elevator]
fulfillRequest()
loop
[Move towards requested Floor 1]
underLimit()
moveUp() / moveDown()
updateFloor(floor)
moveUp() / moveDown()
updateElevatorButtons()
clearButtons()
getButton(i-1)
setEnabled(false)
updateFloorIndicators()
clearIndicators()
setFloorIndicator(i, true)
getFloorButton(floor, UP)
off()
getFloorButton(floor, DOWN)
off()
off()
getFloorButton(floor, DOWN)
off()
updateFloor(floor)
openDoors()
Passenger 2 exits on 3rd floor
closeDoors()
Start moving elevator
fulfillRequest()
loop
[Move towards requested Floor 1]
underLimit()
moveUp() / moveDown()
updateFloor(floor)


moveUp() / moveDown()
updateElevatorButtons()
clearButtons()
getButton(i-1)
setEnabled(false)
updateFloorIndicators()
clearIndicators()
setFloorIndicator(i, true)
getFloorButton(floor, UP)
off()
getFloorButton(floor, DOWN)
off()
updateFloor(floor)
openDoors()
Passenger exits on 4th floor
closeDoors()
floor4Button:QElevatorButton()
floor3Button:QElevatorButton()
panel:QElevatorPanel()
model:QElevatorModel()
elevator1:Elevator()
ecs:ElevatorControlSystem()
externalFloor1UpButton:QElevatorButton()
externalFloor2UpButton:QElevatorButton()

Safety Scenario: Fire

- ECS invokes the Fire singal sequence. Both elevators are set into emergency mode with the fire signal flag. This disables the elevators, updates the screen and plays a message telling passengers to disembark. The doors then close and make their way to the ground floor. This is done via the elevator's method `overrideGoToFloor(Floor::FLOOR_GROUND)` , overriding any requests in the `requests` array.
- The ECS then resets the elevators back into an operational state when the fire signal is addressed

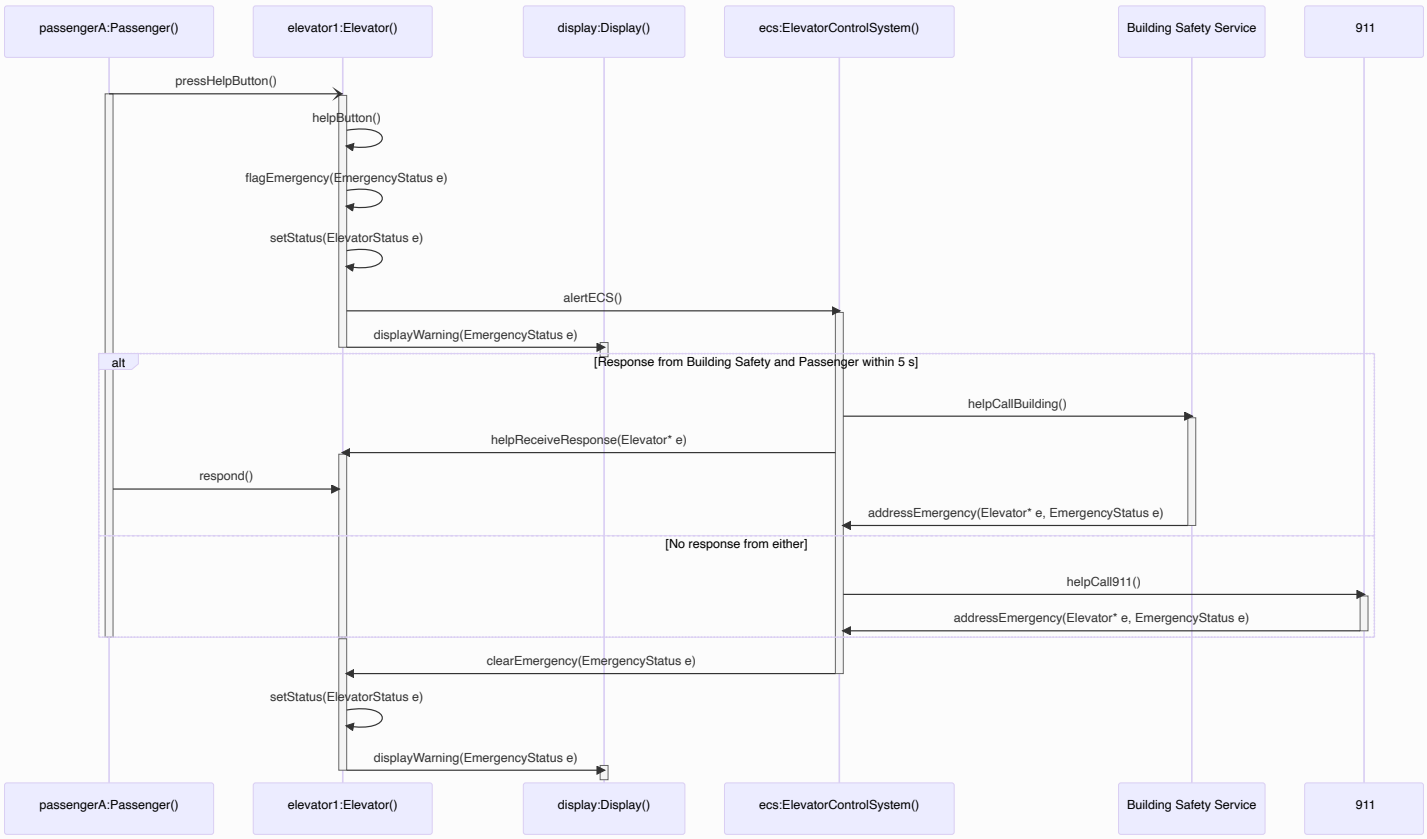

elevator1:Elevator()
display1:Display()
elevator2:Elevator()
display2:Display()
ecs:ElevatorControlSystem()
fireSafetySequence()
flagEmergency(EmergencyStatus::FIRE)
flagEmergency(EmergencyStatus::FIRE)
setStatus(ElevatorStatus e)
setStatus(ElevatorStatus e)
displayWarning(EmergencyStatus e)
displayWarning(EmergencyStatus e)
kickPassengersOut()
kickPassengersOut()
playDisembarkMessage()
playDisembarkMessage()
closeDoors()
closeDoors()
overrideGoToFloor(Floor::FLOOR_GROUND)
overrideGoToFloor(Floor::FLOOR_GROUND)
opt
[Fire signal is addressed, reset elevators]
clearEmergency(EmergencyStatus e)
clearEmergency(EmergencyStatus e)
setStatus(ElevatorStatus e)



Safety Scenario: Help button is pressed

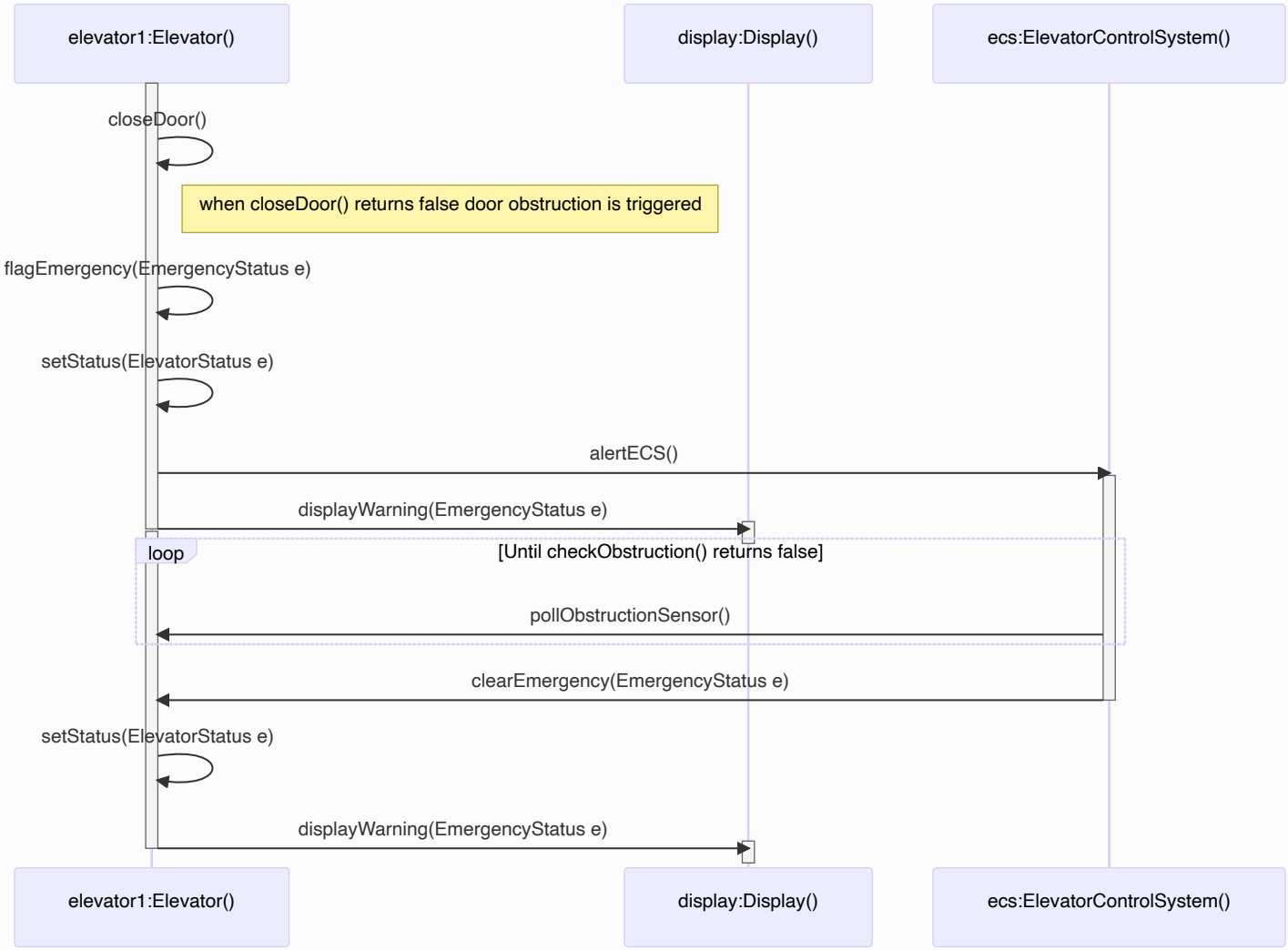
- When A passenger presses the help button, an emergency procedure is initiated. elevator flags the error with `flagEmergency(EmergencyStatus::HELP)`, this value gets added to the elevator's `emergencyStatus` (rather than set). The elevator's `alertECS()` method then alerts the ECS.
- When the `emergencyStatus` is changed and the value isn't zero, the `elevatorStatus` value is set to error mode and becomes inoperative until the problem has been addressed. `emergencyStatus` can only be cleared by the ECS when conditions are deemed safe.
- *Note*: Each EmergencyStatus is a different bit value so you can set multiple

emergency statuses at once. (For example `EmegencyStatus::OBSTRUCTION|EmegencyStatus::OVERLOAD = 0x6` This value can also be decoded to find out which emergency statuses have been set.)



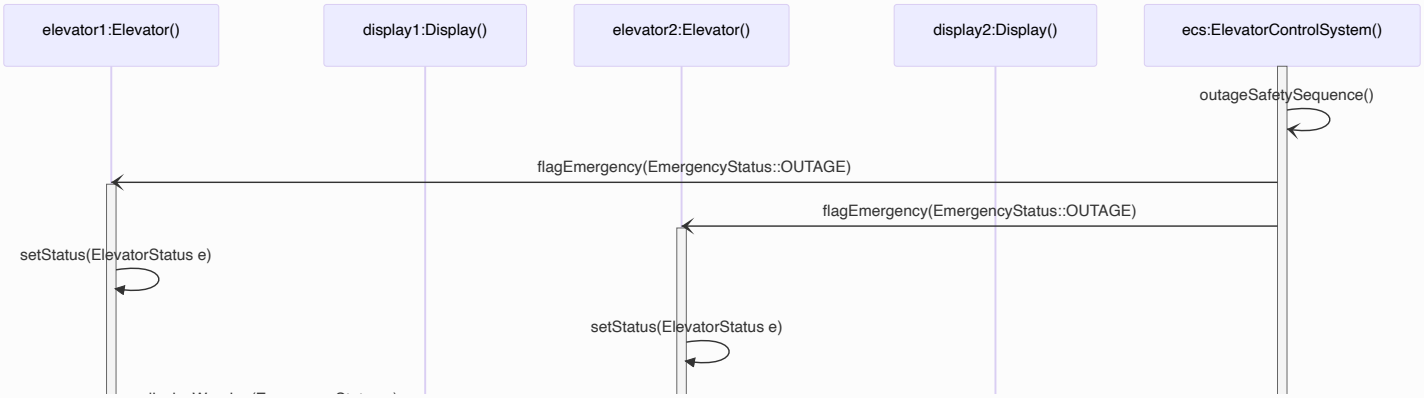
Safety Scenario: Door Obstruction

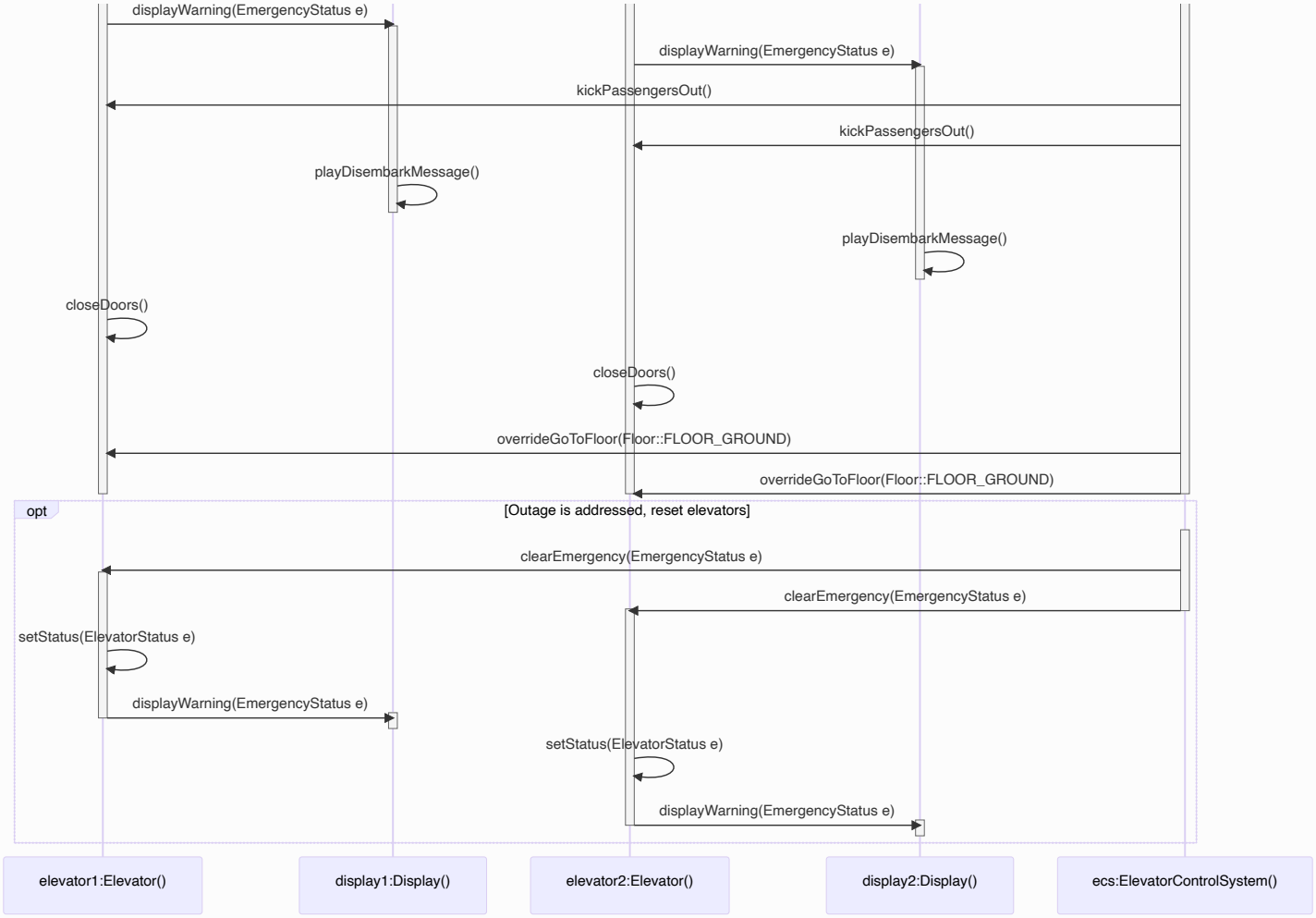
- Elevator will attempt to close the door, `closeDoor()` will check safety sensors such as the light curtain obstruction sensor, as well as the overload sensor. If any of these sensors are triggered `closeDoor()` fails and returns false, flagging an emergency and alerting the ECS. Elevator gets set to `ElevatorStatus::ERROR` and will remain non-operational. At this point the ECS will keep checking the obstruction sensor until the obstruction has been cleared. The ECS will clear the emergency flags the elevator is deemed safe (obstruction cleared) and the elevator will return back to an operating state.


elevator1:Elevator()
display:Display()
ecs:ElevatorControlSystem()
closeDoor()
when closeDoor() returns false door obstruction is triggered
flagEmergency(EmergencyStatus e)
setStatus(ElevatorStatus e)
alertECS()
displayWarning(EmergencyStatus e)
loop
[Until checkObstruction() returns false]
pollObstructionSensor()
clearEmergency(EmergencyStatus e)
setStatus(ElevatorStatus e)
displayWarning(EmergencyStatus e)
elevator1:Elevator()
display:Display()
ecs:ElevatorControlSystem()

Safety Scenario: Power Outage

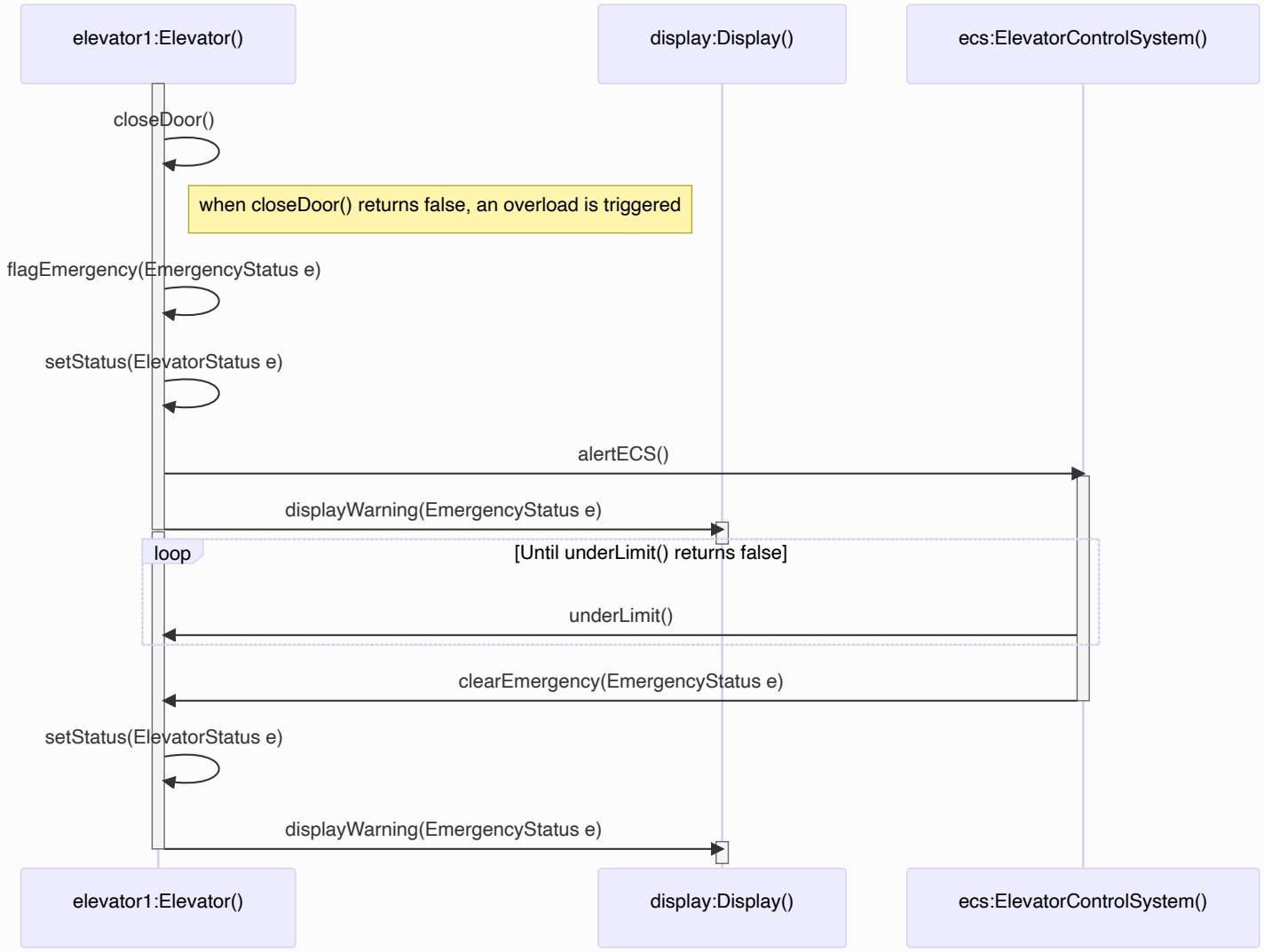
- ECS invokes the power outage sequence. Both elevators are set into emergency mode with the power outage flag. This disables the elevators, updates the screen and plays a message telling passengers to disembark. The doors then close and make their way to the ground floor. This is done via the elevator's method `overrideGoToFloor(Floor::FLOOR_GROUND)` , overriding any requests in the `requests` array.
- The ECS then resets the elevators back into an operational state when the power outage is addressed



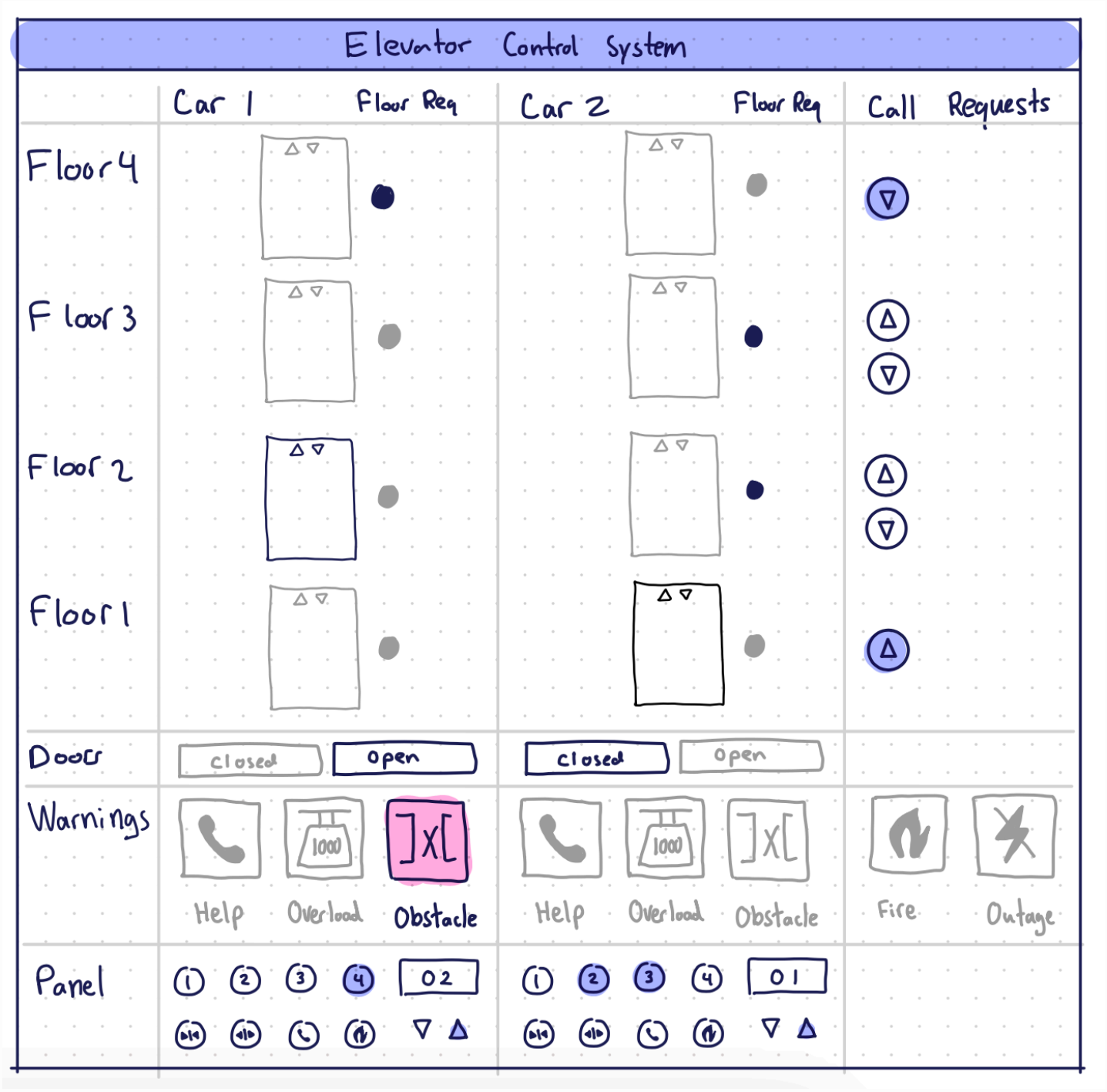

displayWarning(EmergencyStatus e)
displayWarning(EmergencyStatus e)
kickPassengersOut()
kickPassengersOut()
playDisembarkMessage()
playDisembarkMessage()
closeDoors()
closeDoors()
overrideGoToFloor(Floor::FLOOR_GROUND)
overrideGoToFloor(Floor::FLOOR_GROUND)
opt
[Outage is addressed, reset elevators]
clearEmergency(EmergencyStatus e)
clearEmergency(EmergencyStatus e)
setStatus(ElevatorStatus e)
displayWarning(EmergencyStatus e)
setStatus(ElevatorStatus e)
displayWarning(EmergencyStatus e)
elevator1:Elevator()
display1:Display()
elevator2:Elevator()
display2:Display()
ecs:ElevatorControlSystem()

Safety Scenario: Overload

- Elevator will attempt to close the door, `closeDoor()` will check safety sensors such as the light curtain obstruction sensor, as well as the overload sensor. If any of these sensors are triggered `closeDoor()` fails and returns false, flagging an emergency and alerting the ECS. Elevator gets set to `ElevatorStatus::ERROR` and will remain non-operational. At this point the ECS will keep checking the obstruction sensor until the overload isn't present. The ECS will clear the emergency flags the elevator is deemed safe (no overload) and the elevator will return back to an operating state.


elevator1:Elevator()
display:Display()
ecs:ElevatorControlSystem()
closeDoor()
when closeDoor() returns false, an overload is triggered
flagEmergency(EmergencyStatus e)
setStatus(ElevatorStatus e)
alertECS()
displayWarning(EmergencyStatus e)
loop
[Until underLimit() returns false]
underLimit()
clearEmergency(EmergencyStatus e)
setStatus(ElevatorStatus e)
displayWarning(EmergencyStatus e)
elevator1:Elevator()
display:Display()
ecs:ElevatorControlSystem()

GUI Prototype



State Diagrams

Elevator State Diagram