SOFTWARE DESIGN SPECIFICATION

FOR

AUTOMATIC ELEVATOR CONTROL SYSTEM

Prepared By :

Abhinandan (IIT2020119)

Shivam Harjani (IIT2020121)

Aditi (IIT2020138)

Kirti (IIT2020142)

Shashwat Mittal (IIT2020157)

Table Of Contents:

1. Introduction

This paper is a project report for the course Software Engineering at the Indian Institute of Information Technology Allahabad, Prayagraj. Throughout this report, a distributed real-time system – an elevator control system – is specified, designed, built, and simulated. Object Oriented Analysis and Design methods, in specific the Unified Modeling Language (UML) are used when designing the system.

The overall purpose of the Elevator System is to demonstrate the maximum passenger convenience and service within the constraints of available equipment. The software of the elevator controller shows how responsible it is for the safe and efficient operation of all of the other components within the elevator system. The controller's main goal is shown as how to handle input signals from other components and respond accordingly to the output signals. Two of the main computational obligations of the controller are to have a queuing system to log and process requests from passengers and to navigate the cabs of the elevators between floors in response to those requests.

The Java programming language has been selected for implementation so that an application can be executed in the form of an applet.

Section 2 of this report we abstract the overall outline of the elevator control system and the relationships and boundaries between components using the Architecture diagram. In section 3, the design of our elevator control system is presented from a static structural point of view, i.e. the Class diagrams, the Sequence diagrams and the State diagrams are presented and analyzed. Section 4 depicts the Execution Architecture of the project.

2. Conceptual Architecture/Architecture Diagram

We can make a rough draft of the elevator control system. We use the general model of an elevator system. The figures in Section 3 show the resulting component structure. This structure will be retained in essence during the whole specification process.

We refine the elevator control into two components. The new components are the car component "Elevator Control" and the component "Floors". "Floors" consists of four floor control systems, which observe the request button with its control light and forward

requests to the central control system. The component “Elevator Control” consists of three control components. Each of these components is assigned to subsystem of the Elevator.

Elevator runs in a hoistway built within the “Service core”. Service core is one of the most important aspects of high rise buildings. A core is a vertical space used for circulation and services. It may also be referred to as a circulation core or service core. A core may include staircases, elevators, electrical cables, water pipes and risers. The design of this is predominantly governed by the fundamental requirements of meeting fire-egress regulations, achieving basic efficiency in human movement, and creating an efficient internal layout. Typically the service core provides the principal structural element for both the gravity load-resisting system and lateral load-resisting system, with the latter becoming increasingly important as the height of the building increases. The core provides the stiffness to restrict deflections and accelerations to acceptable levels at the top of the building. The cost of a core for a typical high rise building is estimated to be around 35 to 40 percent of the total structural cost, or 4 to 5 percent of the total development cost.

The main component of the elevator control system is “Central Elevator Control”. This component realizes the strategy for serving the requests. Furthermore this component controls the other units, “Door Control” and “Motor Control”.

3. Logical Architecture (Class Diagram, Sequence Diagram, State Diagram)

Class Diagram:

Class diagram, one of the most commonly used diagrams in object-oriented systems, models the static design view for a system. The static view mainly supports the functional requirements of a system – the services the system should provide to the end users.

A class diagram shows a set of classes, interfaces, and collaborations and their relationships. Class diagrams involve global system description, such as the system architecture, and detail aspects such as the attributes and operations within a class as well. The most common contents of a class diagram are:

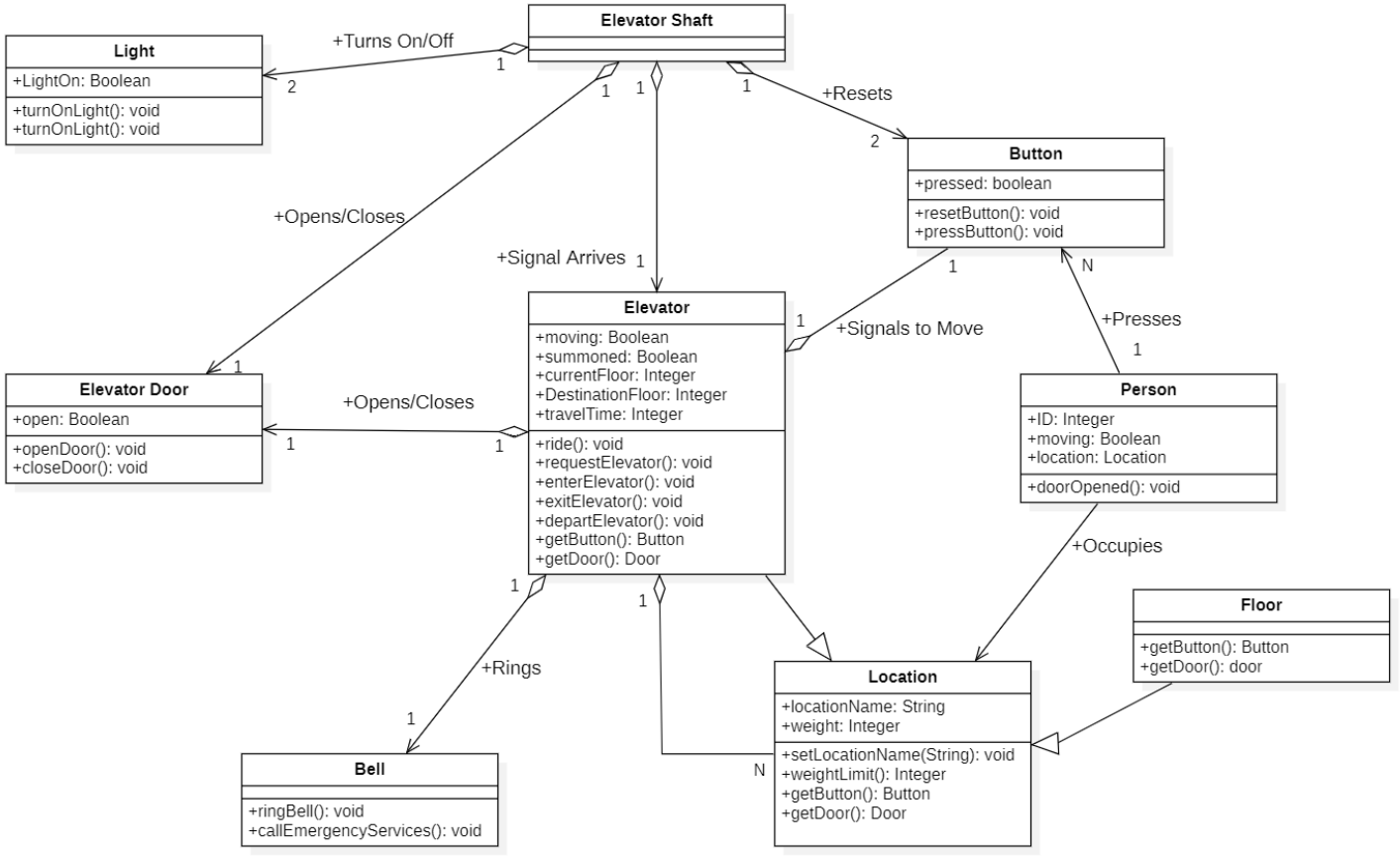
- Classes
- Interfaces

- Collaborations
- Dependency, generalization, and association relationships
- Notes and constraints

We have built this class diagram from the view of object composition of the system -

1. ElevatorControl: The central controlling object in the elevator system. ElevatorControl communicates and controls all other objects in the system.
2. Door: There are two doors in the system, the “god” object - the ElevatorControl – command the doors to open and close, according to the situation
3. Car: The car is being controlled to move up and down (at different speeds), to make stops at floors when necessary.
4. Button: The ElevatorControl class also controls the button class, which further generalizes two subclasses CarCallButton and HallCallButton. The control object communicates with the Button objects, get the information whether a button is pressed and in turn controls the illumination of Button lights
5. Indicator: There are two kinds of indicators in the system, the CarPositionIndicator and the CarDirectionIndicator (i.e. the CarLantern). The indicators are controlled to show the information about the current position and moving direction of the car.
6. Safety: Whenever an emergency happens according to the definition of emergency brake trigger in the requirement documentation, the ElevatorControl commands the Safety.

CLASS DIAGRAM



Sequence Diagrams:

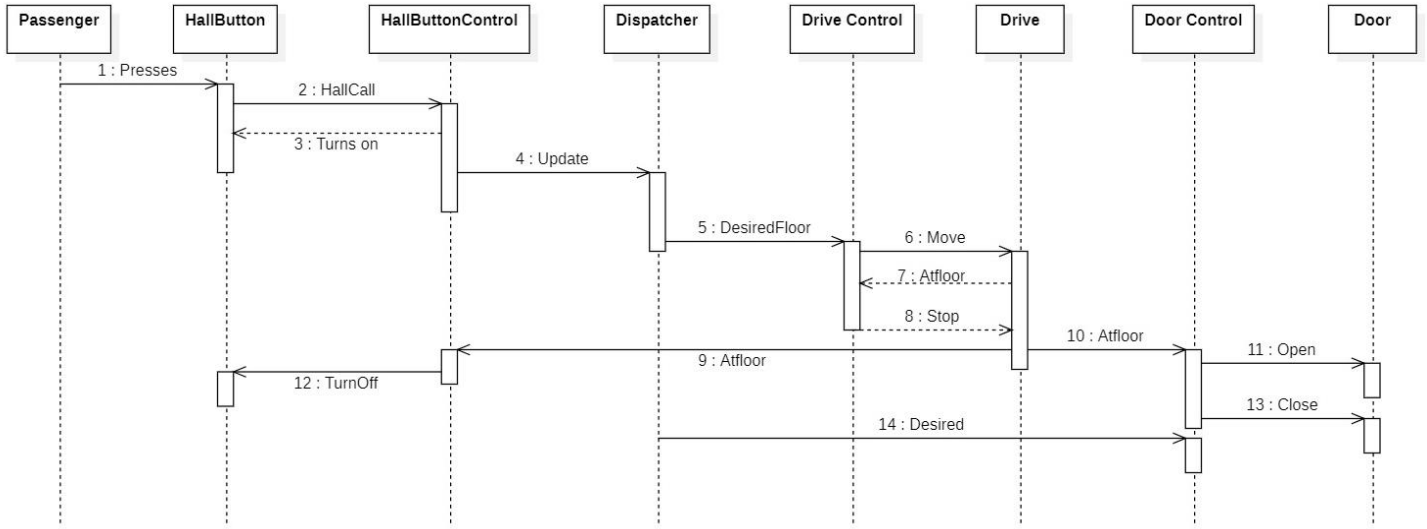
Sequence diagram is one kind of interaction diagram, which shows an interaction among a set of objects and their relationships. The purpose of the Sequence diagram is to document the sequence of messages among objects in a time based view. The scope of a typical sequence diagram includes all the message interactions for (part-of) a single use case. There may be multiple sequence diagrams per use case, one per use case scenario.

The objects in sequence diagrams are based on the class diagram from the software architecture view. The reason for doing that is we want to neither stay in the object construction view, in which the functions of objects are obscure and inadequate, nor go too further in the system architecture view, where many technical details obstruct a quick understanding of interaction among objects.

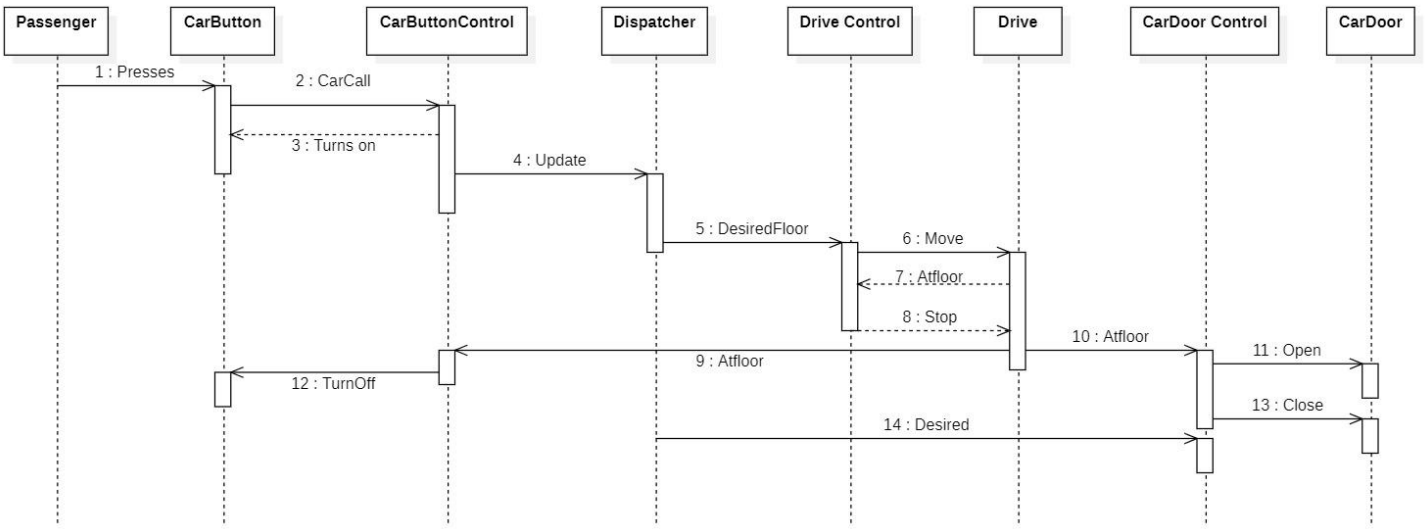
The state diagrams commonly contain:

- Objects
- Links
- Messages
- Response Time (especially useful in real-time systems)

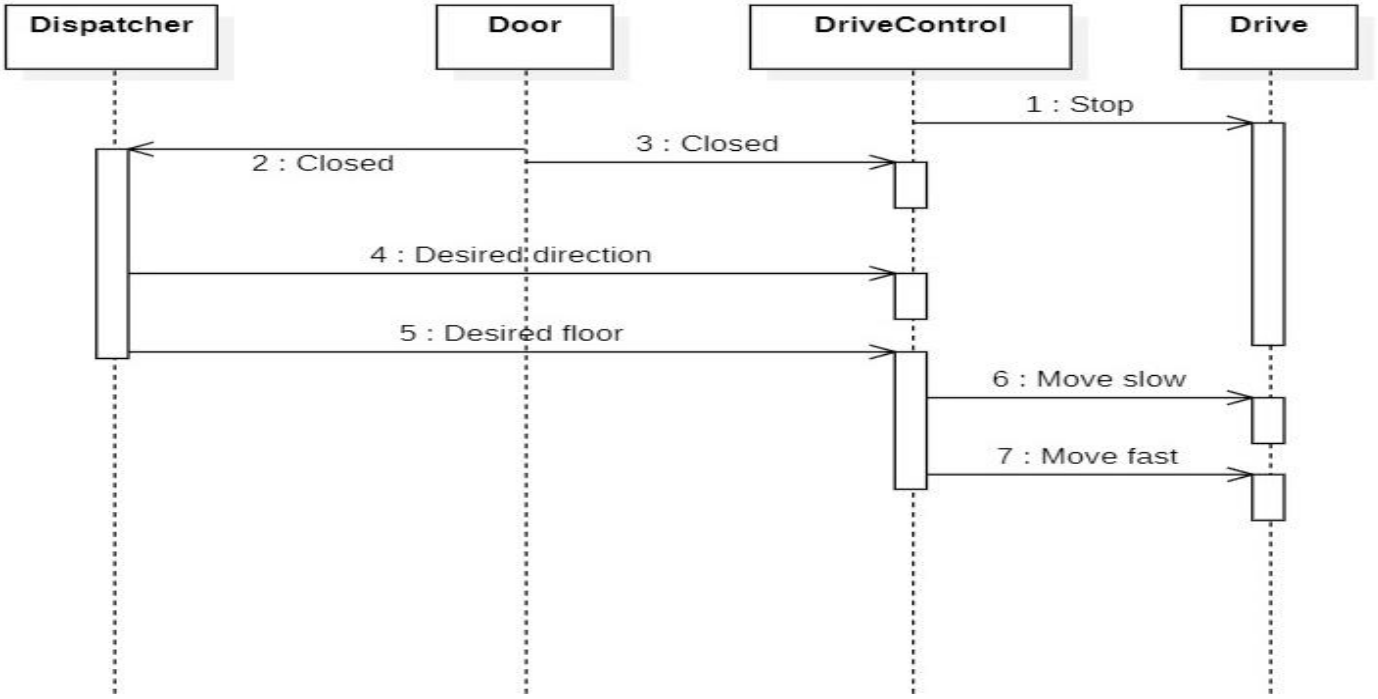
Sequence Diagram : Floor/Hall call service



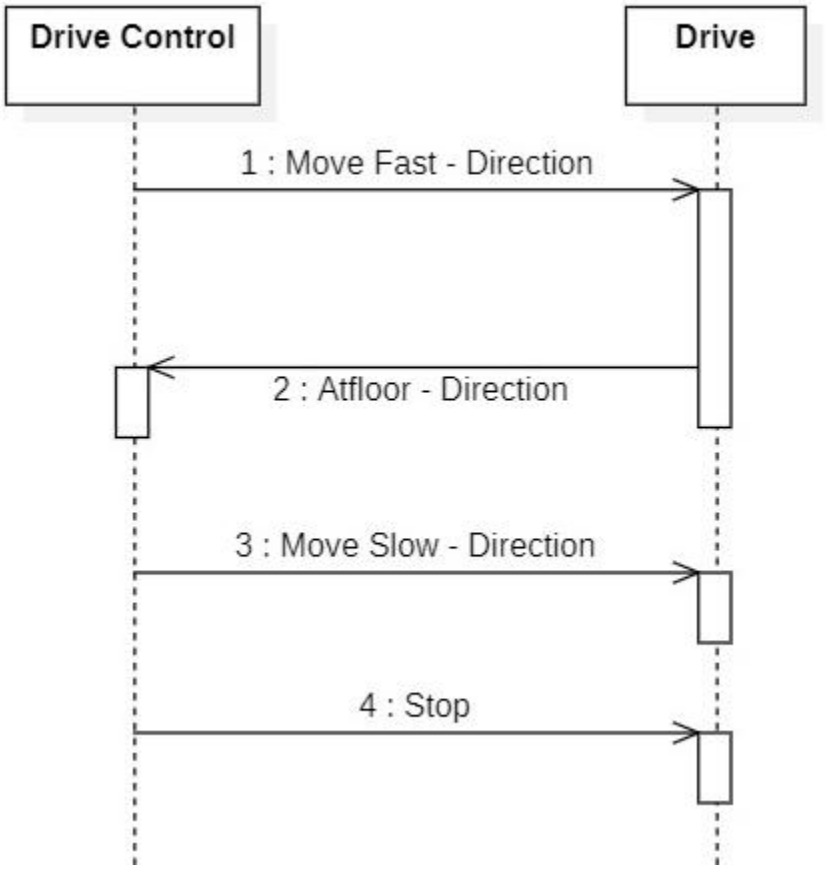
Sequence Diagram : Car call service



Sequence Diagram : Movement of car

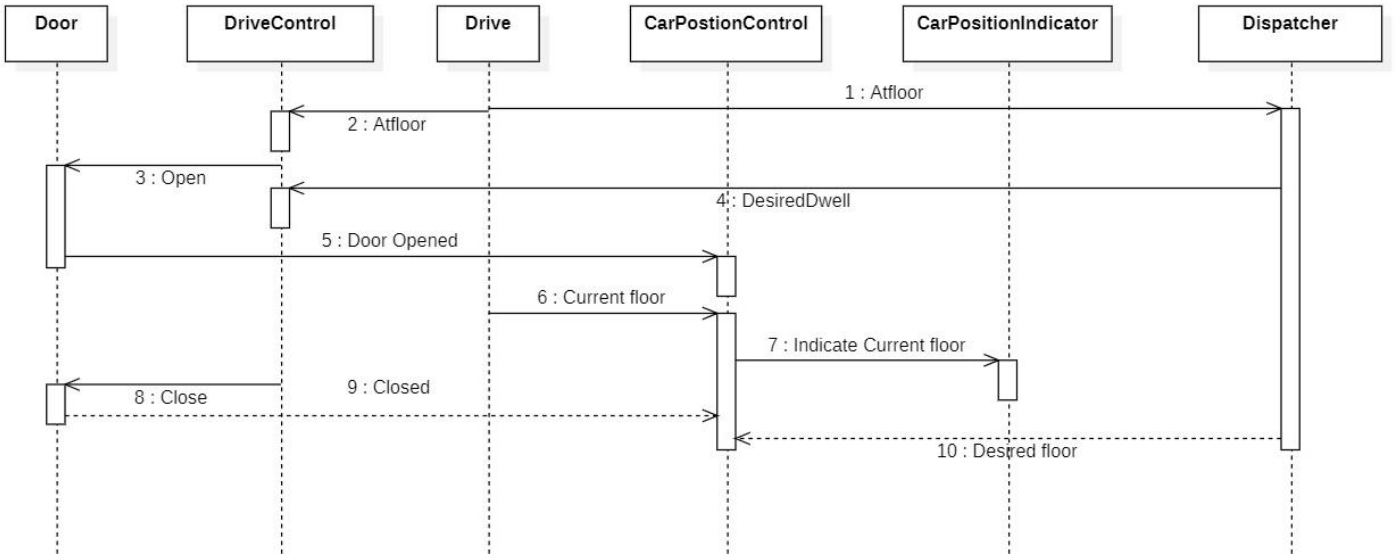


Stop to Slow then to Fast

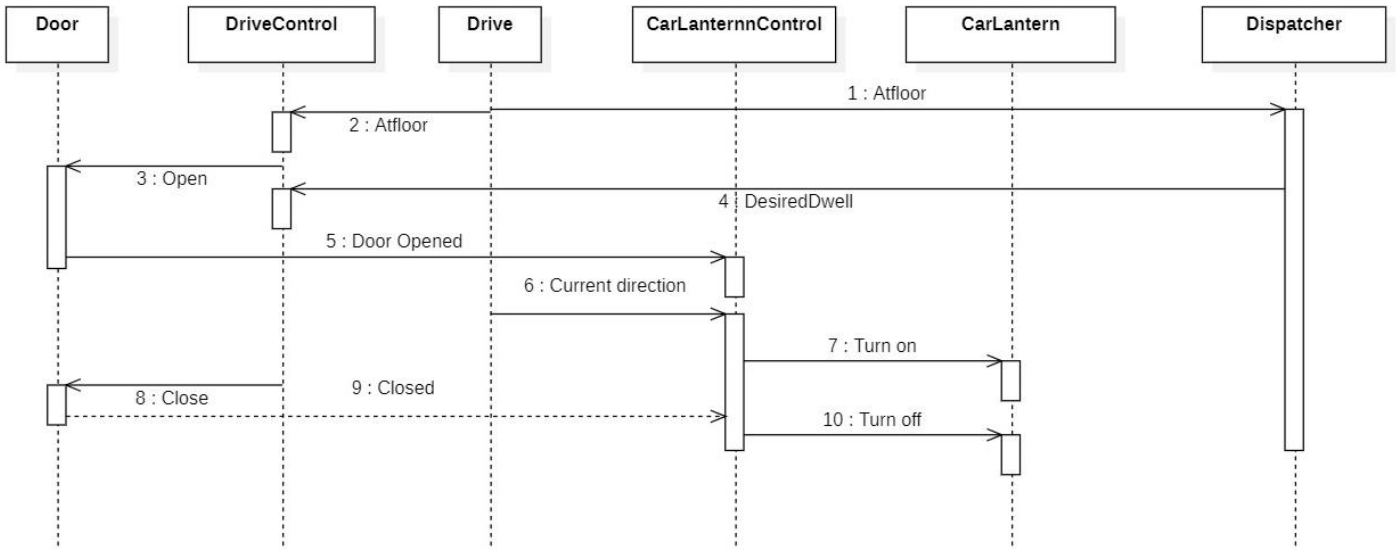


Fast to Slow then to Stop

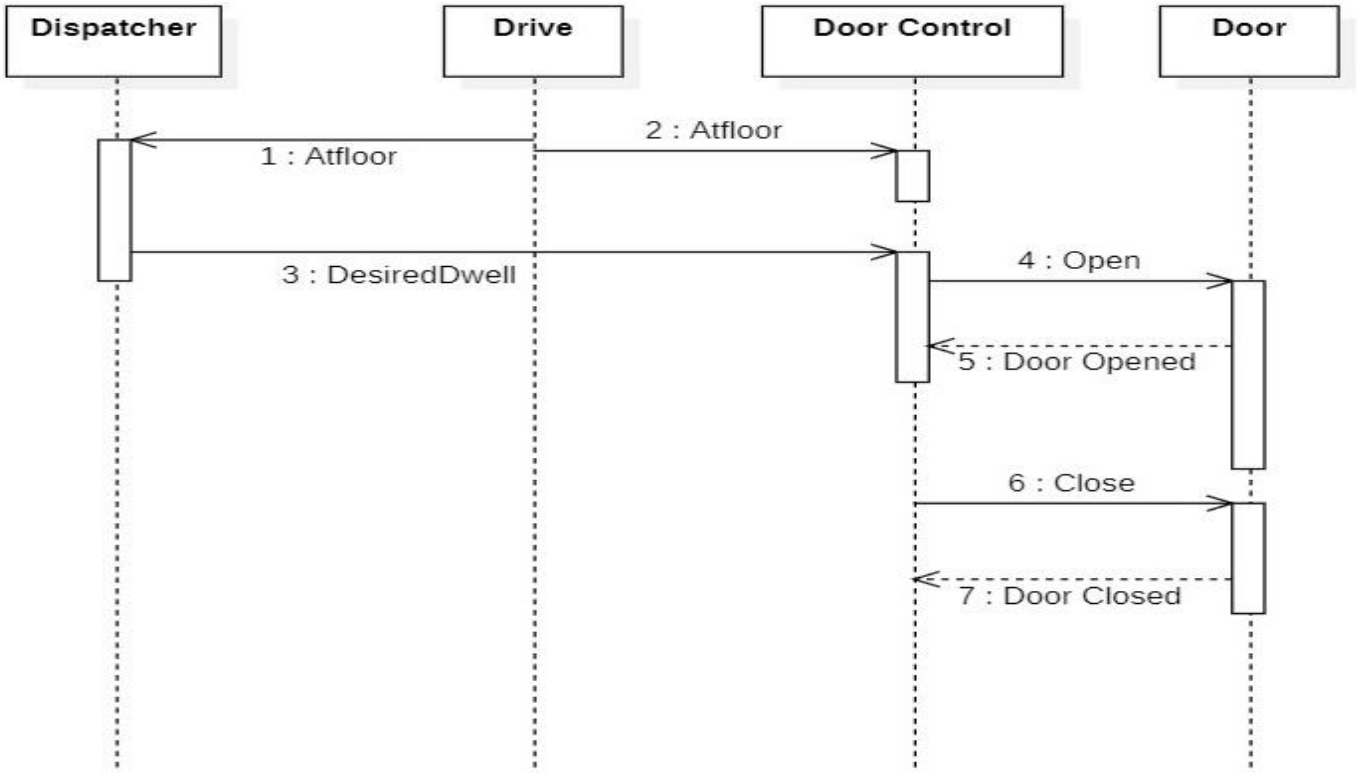
Sequence Diagram : Car position Indicator



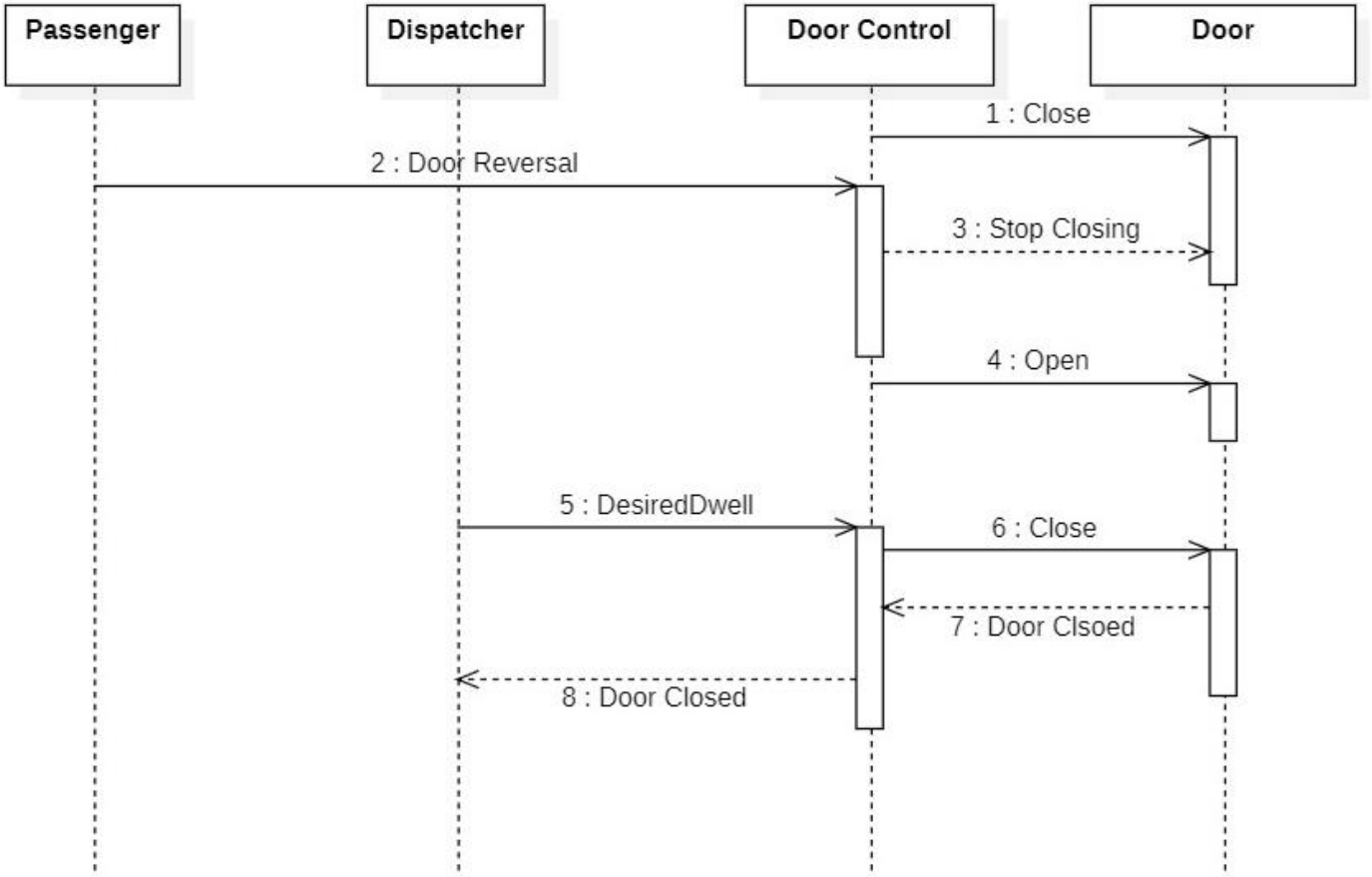
Sequence Diagram : Direction of car movement



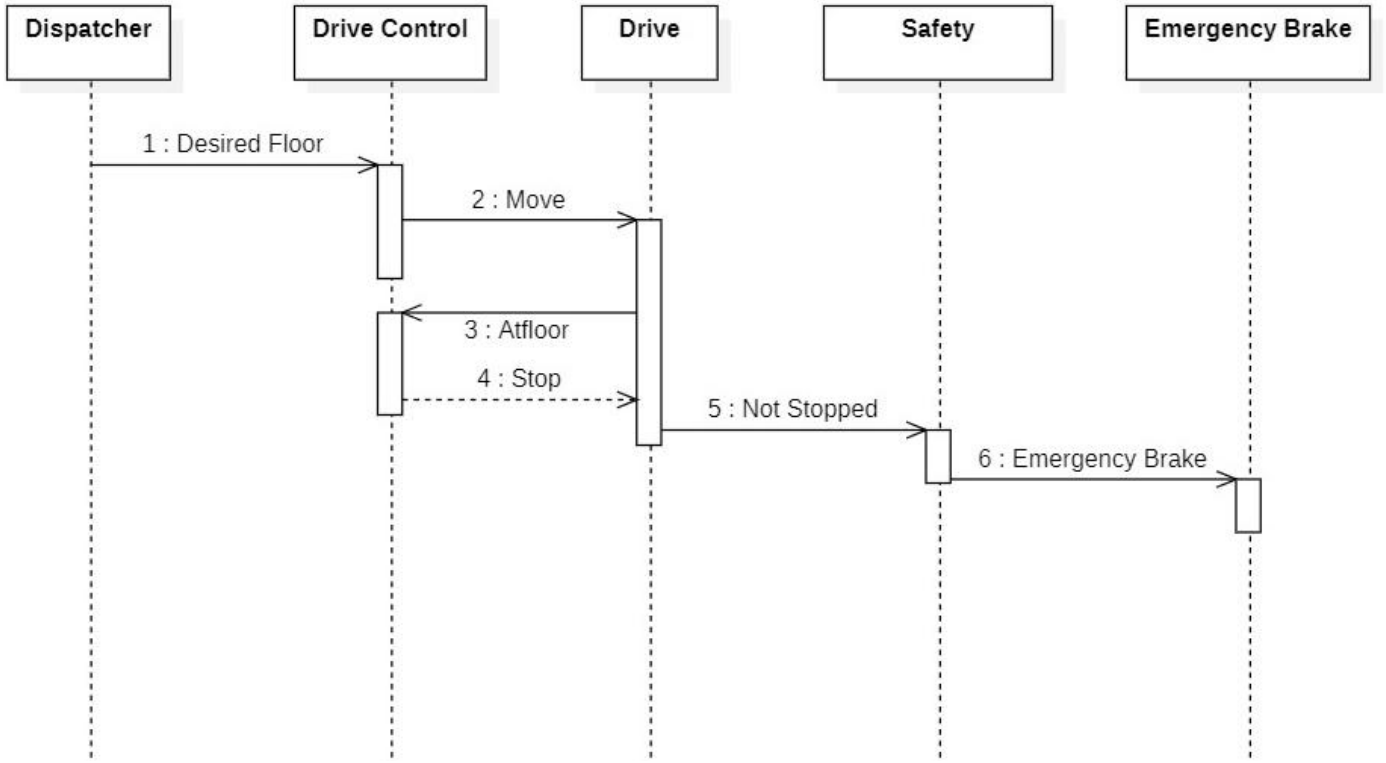
Sequence Diagram : Open/Close the doors



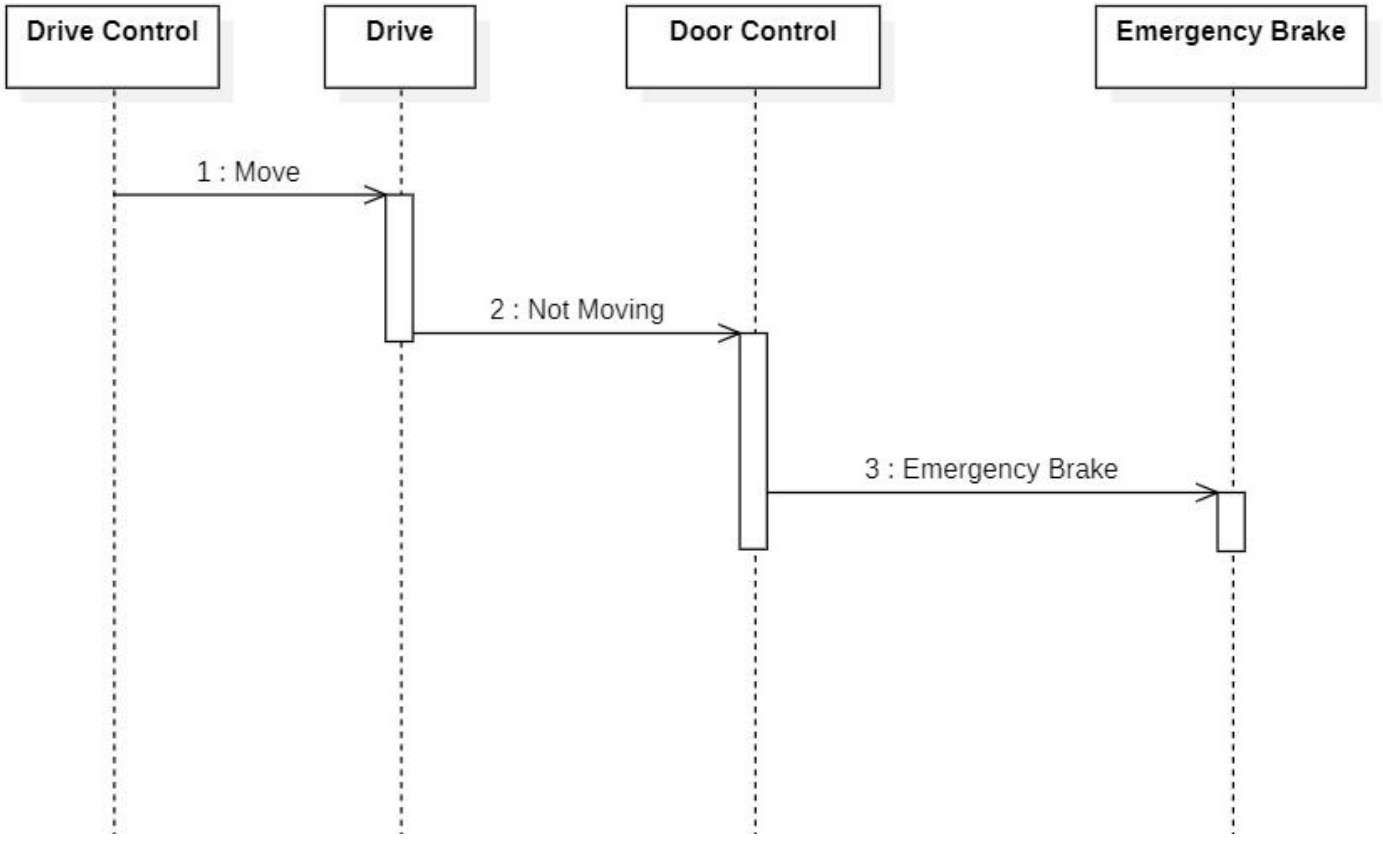
Sequence Diagram : Door reversal



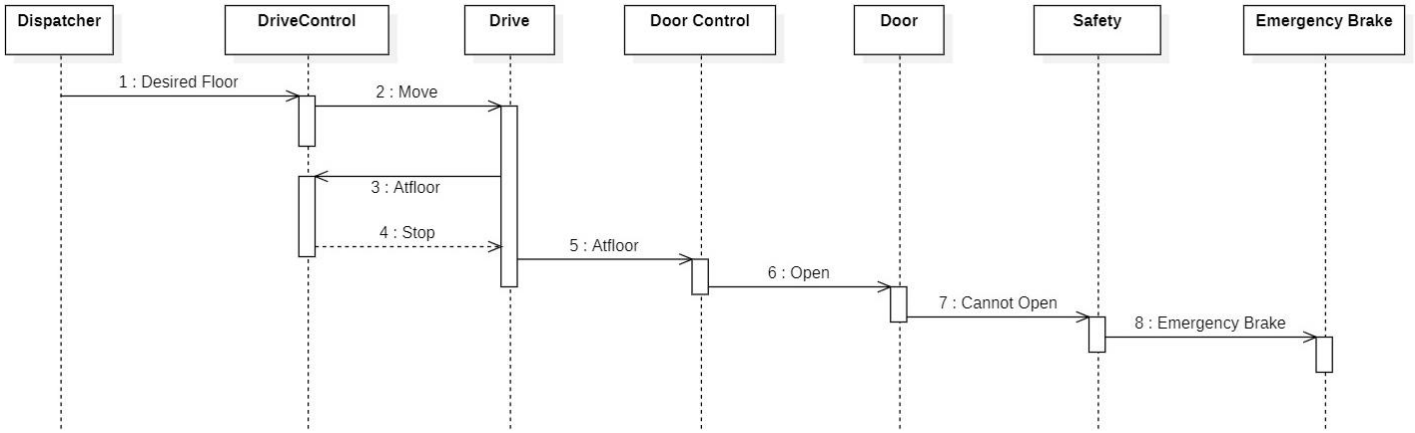
Sequence Diagram : Trigger emergency brake



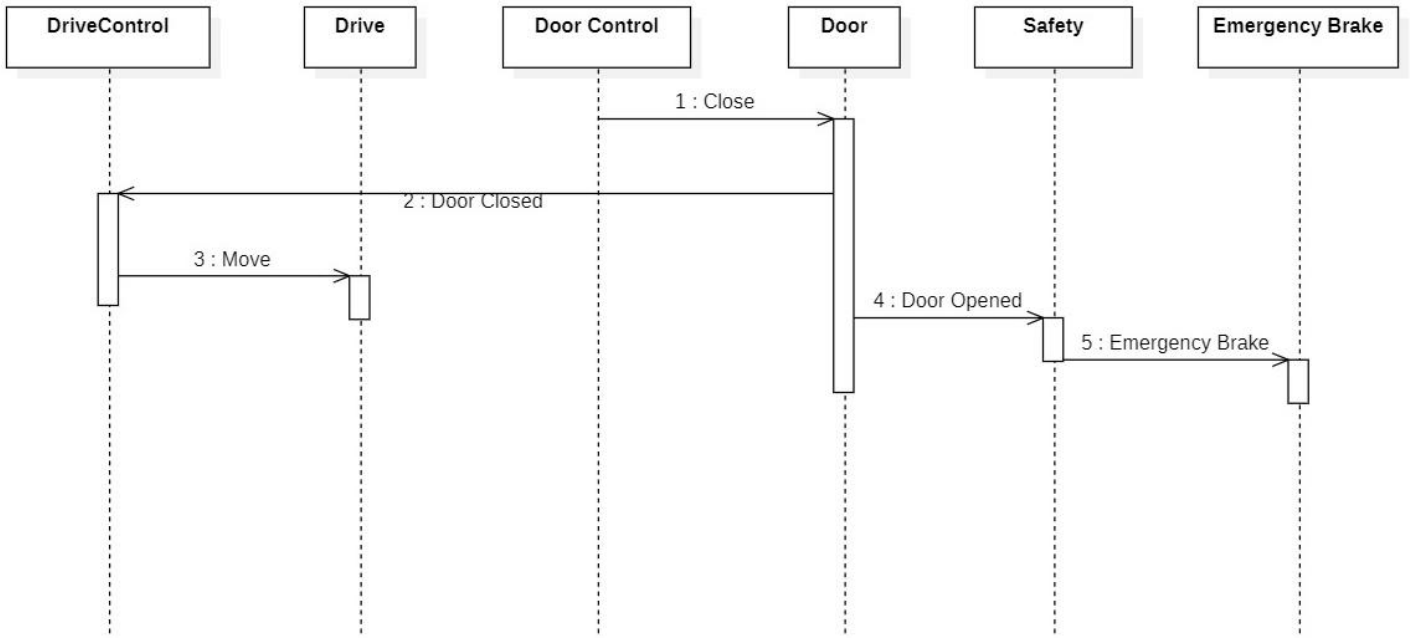
Emergency Brake - The car won't stop at desired Floor



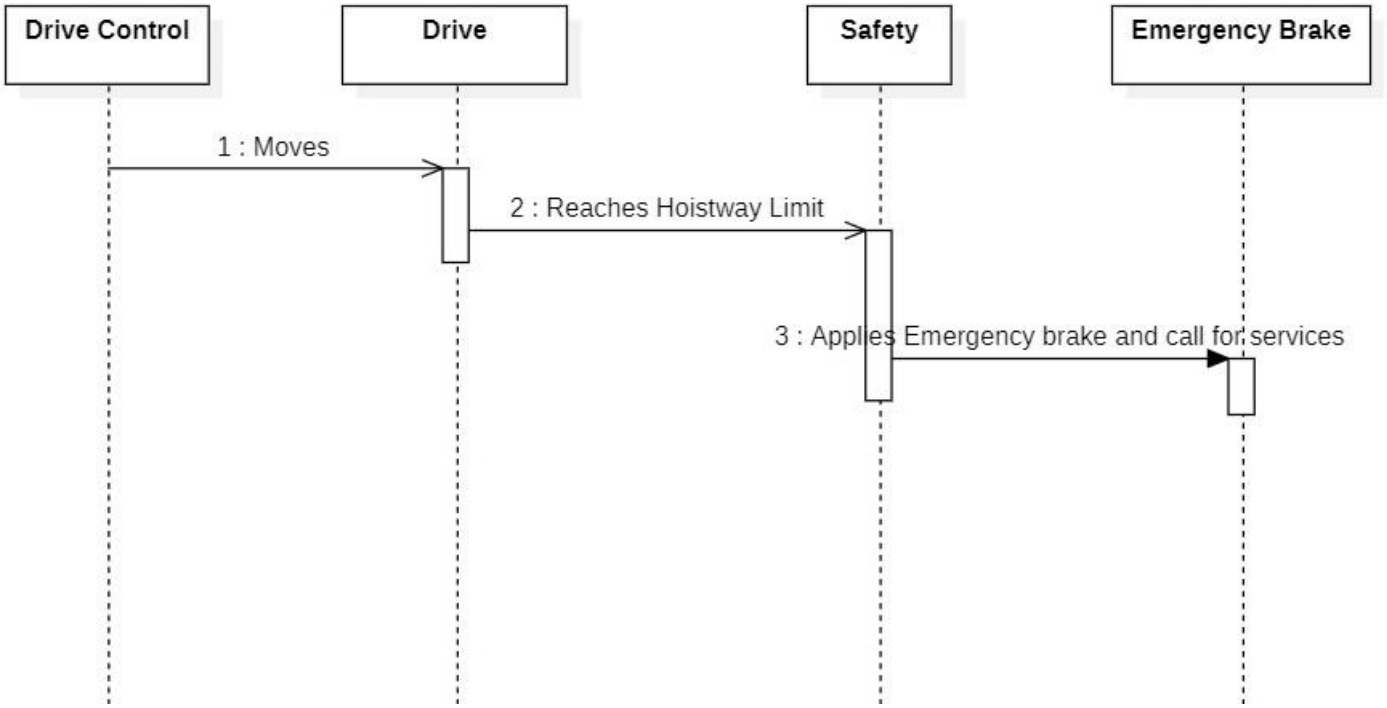
Emergency Brake - The car won't move



Emergency Brake - The doors won't open when the elevator stops at desired floor



Emergency Brake - The doors open when the elevator is moving

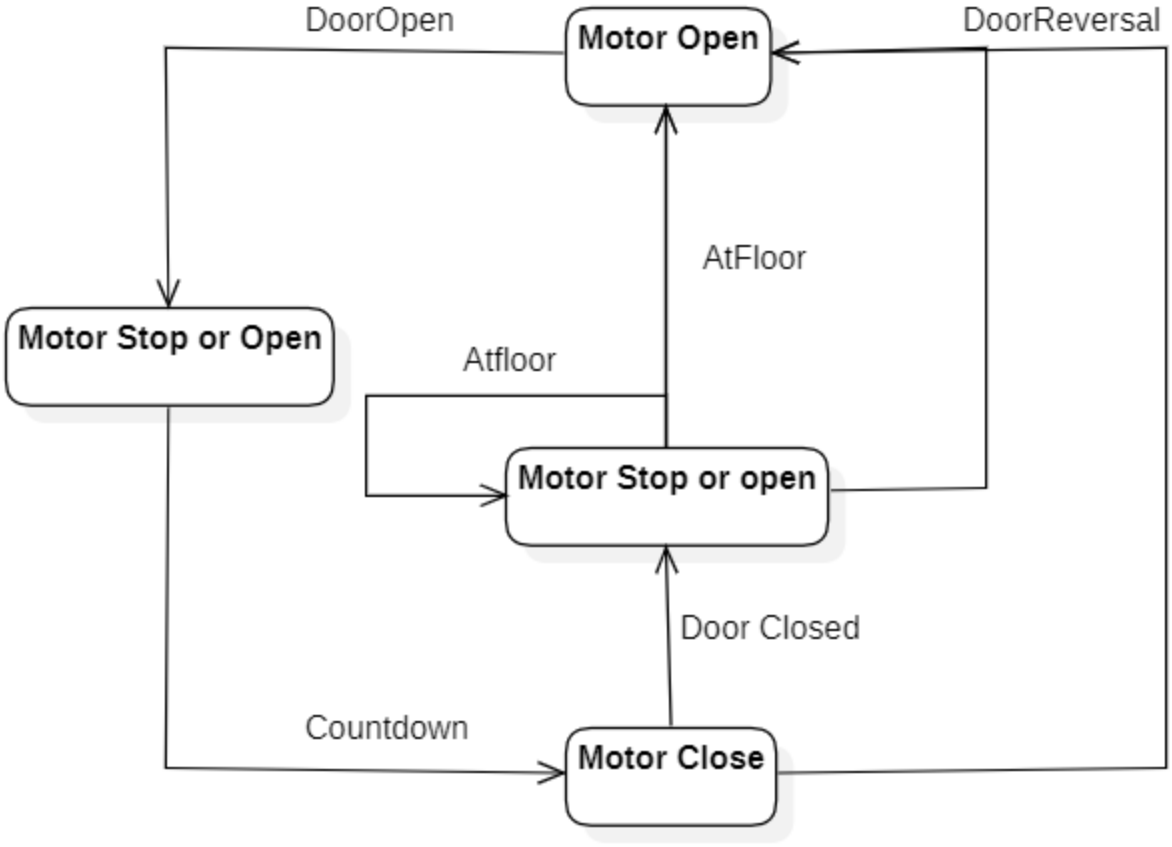


Emergency Brake - The elevator keeps going when hoistway limit is reached

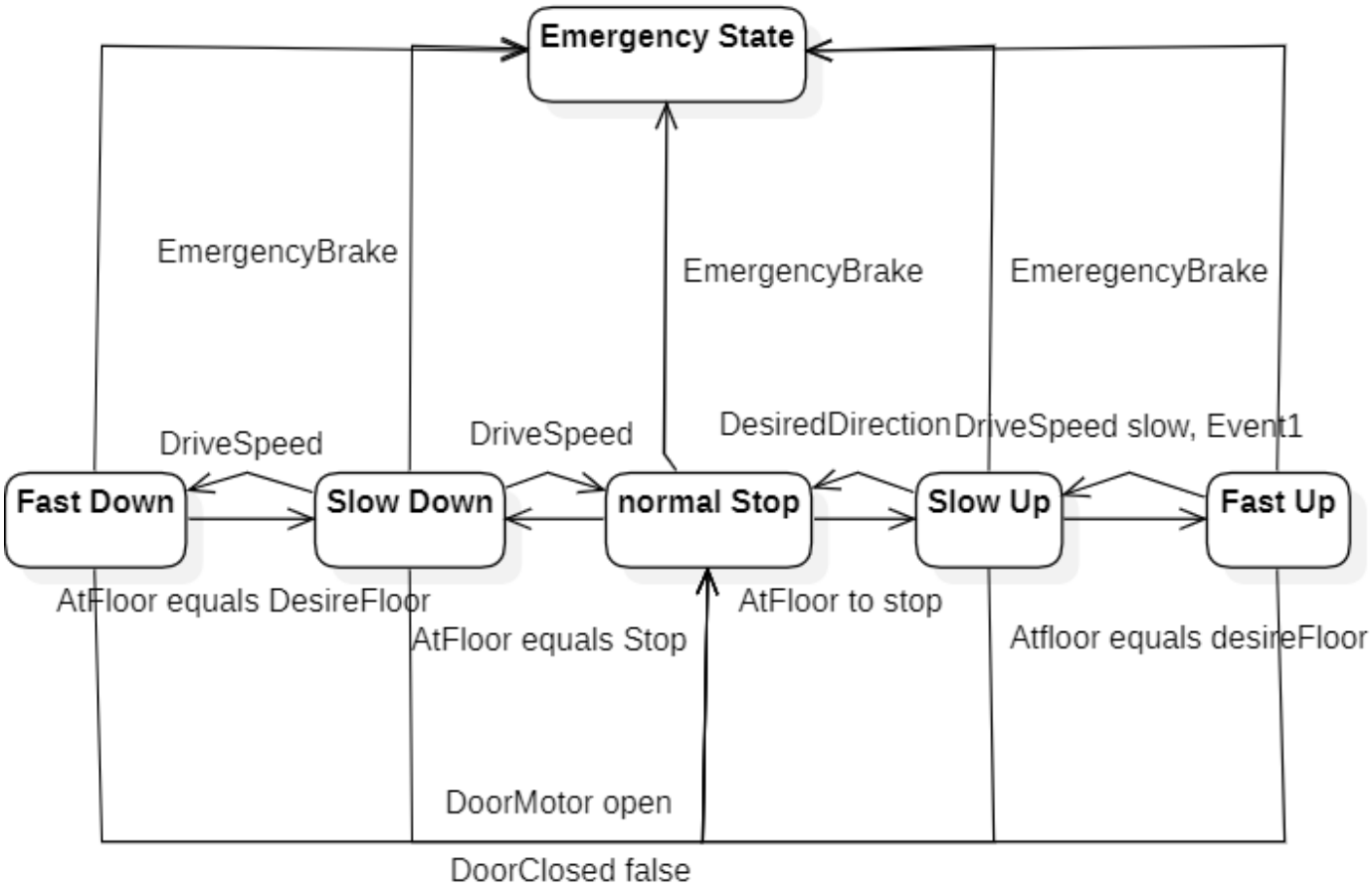
State Diagrams:

A State chart diagram shows a state machine. Usually the state machine in a state chart models the behavior of a reactive object, whose behavior is best characterized by its response to events dispatched from outside its context. The object has a clear lifetime whose current behavior is affected by its past. State chart diagrams are important for constructing executable systems through forward and reverse engineering.

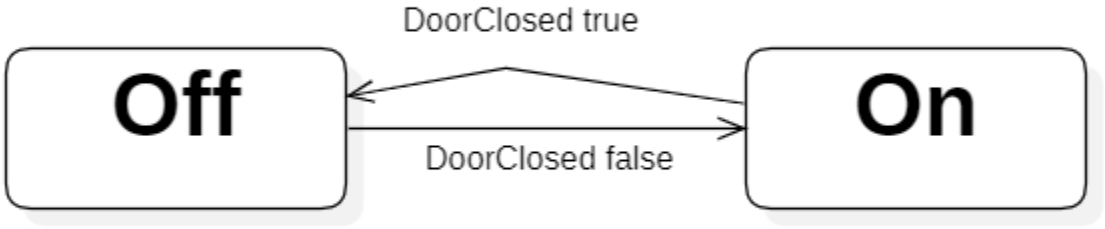
State Diagram : Door control



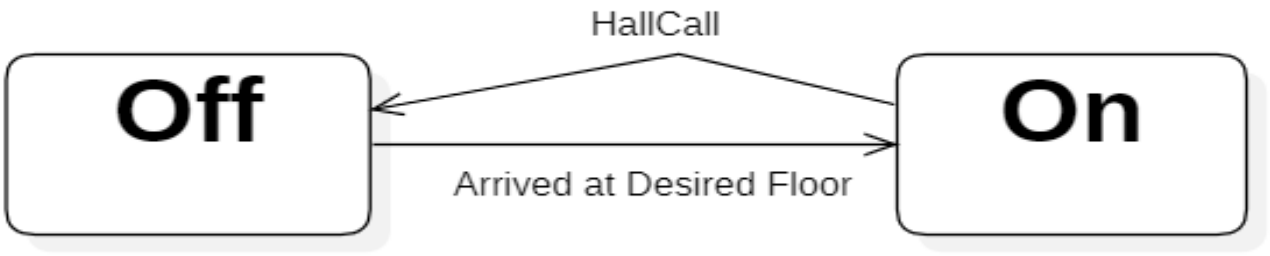
State Diagram : Drive control



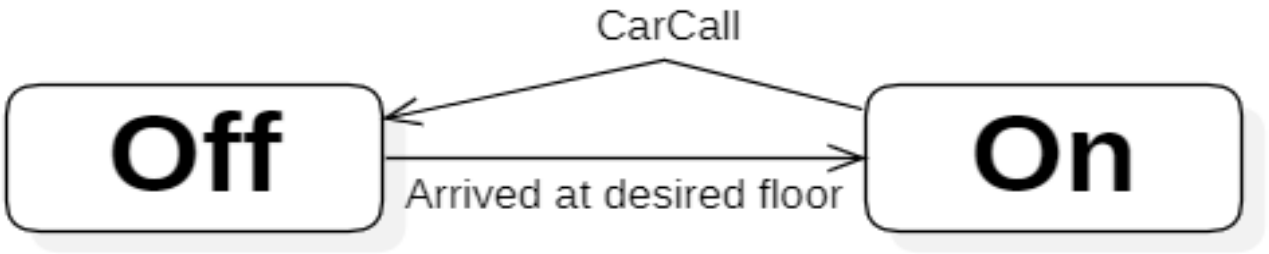
State Diagram : Lantern control



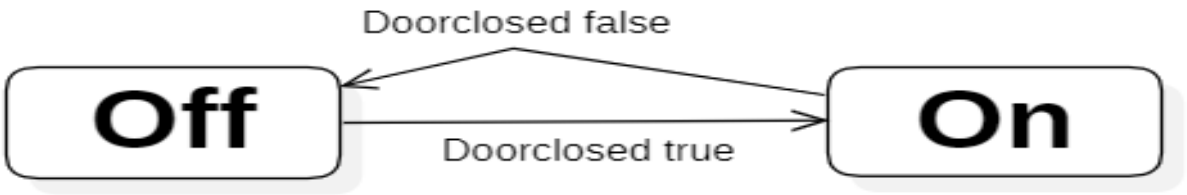
State Diagram : HallButton Control



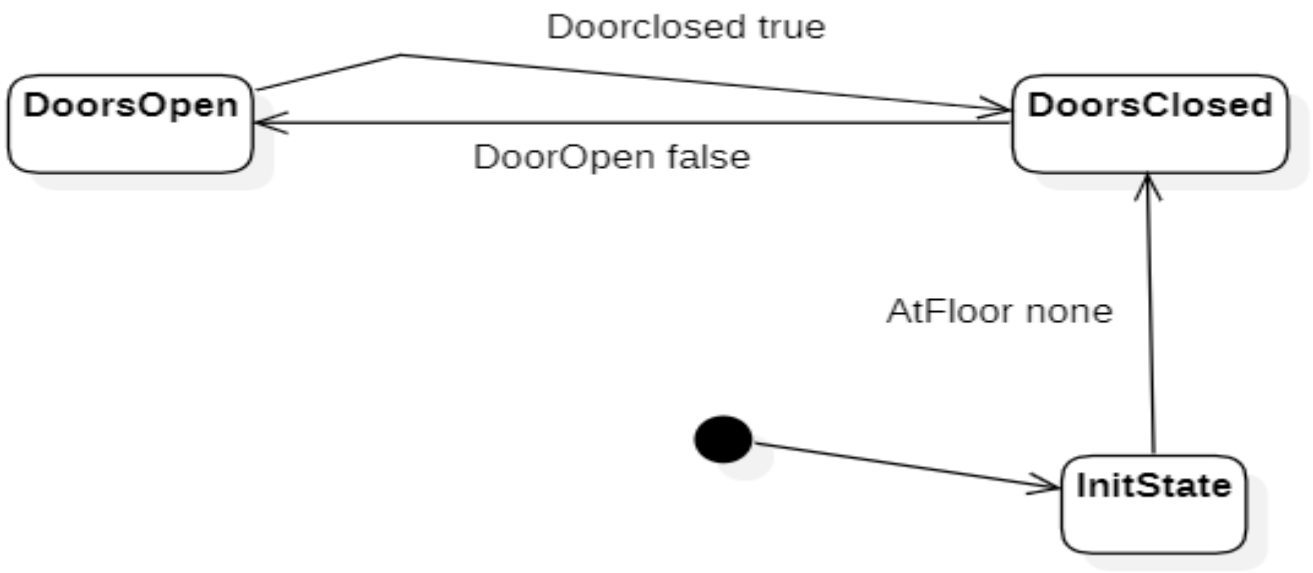
State Diagram : CarButton Control



State Diagram : CarPosition Control



State Diagram : Dispatcher



Basic Classes Involved:

Floor Button

A request is made by the passenger when he/she presses the up or down button outside the elevator.

Method: Call_Lift

Description: It schedules the request in the scheduler.

Elevator Button

A request is made inside the elevator. The passenger can select the floor to which they wish to go.

Method: Destination_Foor

Description: It moves the list to the floor requested by the passenger.

State

It has three values: MOVING, STOPPED and IDLE. It tells the state of the lift.

Method: Check_state

Description: Checks the state of the lift whether it is moving, stopped or idle.

Door

Opens and Closes the doors.

Method: Open_Door

Description: Opens the Door and calls the Arrival_Tone method.

Method: Close_Door

Description: Closes the Door

Method: Stay_Open

Description: It directs the doors to remain open.

Floor

Has a floor ID so that the system can return the floor number to the door system and the display.

Method: Check_Floor

Description: Checks for the state of the elevator and returns the current floor in case of idle and stopped state and returns the approaching floor in case of moving state.

Status

Checks the number of people in the elevator from the person and checks the total weight in the elevator and in case of an overload, it requests the door to be open until weight is reduced below the capacity.

Method: Weight

Description: Checks the current weight of the lift.

Method: Person

Description: Checks the number of people in the lift.

Method: Overload

Description: If current weight or number of persons is greater than overload then Stay_Open is called until overload is reduced.

Display

Show the floor, no of people and total weight in the elevator. It shows the current floor in idle and stopped state and the approaching floor in moving state.

Method: GetStatus

Description: Gets floor no, no of people and total weight in the elevator from Check_Floor, Person and Weight method respectively and displays them.

Emergency Button

Stops the moving lift at the nearest floor in the moving direction and sends an alert to the admin.

Method: emergency_btn

Description: Stops the Lift in at the nearest floor in the moving direction

Method: Alert_Admin

Description: Sends an alert to the admin about the lift being stopped due to emergency

Bell

Makes sound on the opening of doors. Makes an alert sound in case of overload.

Method: Arrival_Tone

Description: Makes the sound on the opening of doors

Method: Emergency_Tone

Description: Makes noise till it stops at the nearest floor.

Scheduler

Decide to serve which request and which floor to send first. Keep a queue of requests. It uses scheduling algorithms

Method: Request_floor

Description: Schedules the request from different floors using a scheduling algorithm for the lift to go.

4. Execution Architecture

Runtime environment required is any device supporting 17.0.1 version of JDK and 11.1.2 version of java to run the application.

4.1 Reuse and Relationships to Other Products

NIL

5. Design Decisions and Tradeoffs

The design decision to use two screens separately, one for displaying the real time visualization of the elevator and the other to the user as a controller. It may have been possible to get all the information on one screen. However, using two screens will provide a better user interface and be less crowded. The design would go as follows -

1. A person is on a particular floor. Suppose Ground Floor. He wants to go to the 5th Floor. So he clicks on the elevator button with upwards direction.

2. This will be referred to as the ExternalRequest. The direction and floor on which the user clicked the button, i.e. source floor, will be included in this Request. If there are any available requests, the elevator will check them and then handle this request according to its priority. The elevator arrives at the origin floor, which is the 0th or ground floor. The person steps into the elevator.

3. The person enters the elevator. The person then presses the 5th floor button in the elevator to indicate the elevator to go to the 5th floor.

4. This will be the internal request. So the internal request will be having only the floor to which the person wants to go to i.e. the destination floor. The elevator moves to the fifth floor. And the person then exits the elevator.

5. If suppose when the elevator is moving from the ground floor to the fifth floor and it reaches the first floor. At this moment suppose another person on the second floor wants to go in the UP direction. Then the elevator will stop for this request and the person on the second floor will enter the elevator. Suppose he presses the 4th floor button. Then the elevator will first stop at the 4th floor which was the destination of the

person who entered on the second floor. Later the elevator stops on the fifth floor and the person from the ground floor exits.

6. If suppose when the elevator is moving from the ground floor to the fifth floor and it reaches the first floor. At this moment suppose another person on the second floor wants to go in the DOWN direction. Then the elevator will not stop for this request immediately. Elevator will first go to the fifth floor where the person from the ground floor will exit. Elevator will then go to the second floor. The person will enter the elevator and press 0. The elevator will then move to the zeroth floor.

When it comes to links, one possible trade off is to use buttons instead of menu items. The inclusion of buttons to navigate between displays is a design decision made to improve visibility. Text links on the menu bar at the bottom of the screen can be difficult to navigate. Navigation from screen to screen will be easier if buttons with informative names are used instead of text links in the menu bar.

The user will be able to tell where he is navigating thanks to descriptive labeling. The text links in the display's menu bar are smaller than the buttons. As a result, the user will have a simpler time locating the mechanisms required to go from one screen to the next.