11] An Elevator Control System

Functional Requirements of the system:

➢ User Requirements:

1- Each passenger can request car by pressing on Floor Button
2- Each passenger can choose floor by pressing on Car Button of specific floor number
3-Each passenger can call emergency by pressing on Emergency Button

➢ System Requirements:

1.1 – Floor button will be turned on if passenger pressed on it
1.2 – Indicator will send the floor position to Elevator controller system .
1.3 – Elevator controller system will add that request to the requests bank list
1.4 – Elevator controller system chooses the available and nearest car based on floor position
1.5 – Elevator controller system will send signal to the car chosen before
1.6 – Specific car will go to the floor
1.7 – if car reached the target floor , doors will open
1.8 – after 5sec doors will be closed , if there is an object between doors , doors will be opened again

2.1 – Car button will be turned on if passenger pressed on it
2.2 – Indicator will send the target floor position to Elevator controller system .
2.3 – Elevator controller system will add that request to the requests bank list
2.4 – After 5sec doors will be closed , if there is an object between doors , doors will be opened again
2.5 – car will be moving toward target floor until it reached the destination
2.6 – if car reached the target floor , doors will open
2.7 – after 5sec doors will be closed , if there is an object between doors , doors will be opened again

3.1 – Emergency button will be turned on if passenger pressed on it
3.2 – Indicator will send the floor position where emergency button was pressed to Elevator controller system .
3.3 – Elevator controller system will add that request
3.4 – Elevator controller system will handle the emergency by sending the car to nearest floor and then open the doors
3.5 – After 5sec doors will be closed , if there is an object between doors , doors will be opened again
3.6 -car will be stop or freeze until issues resolved

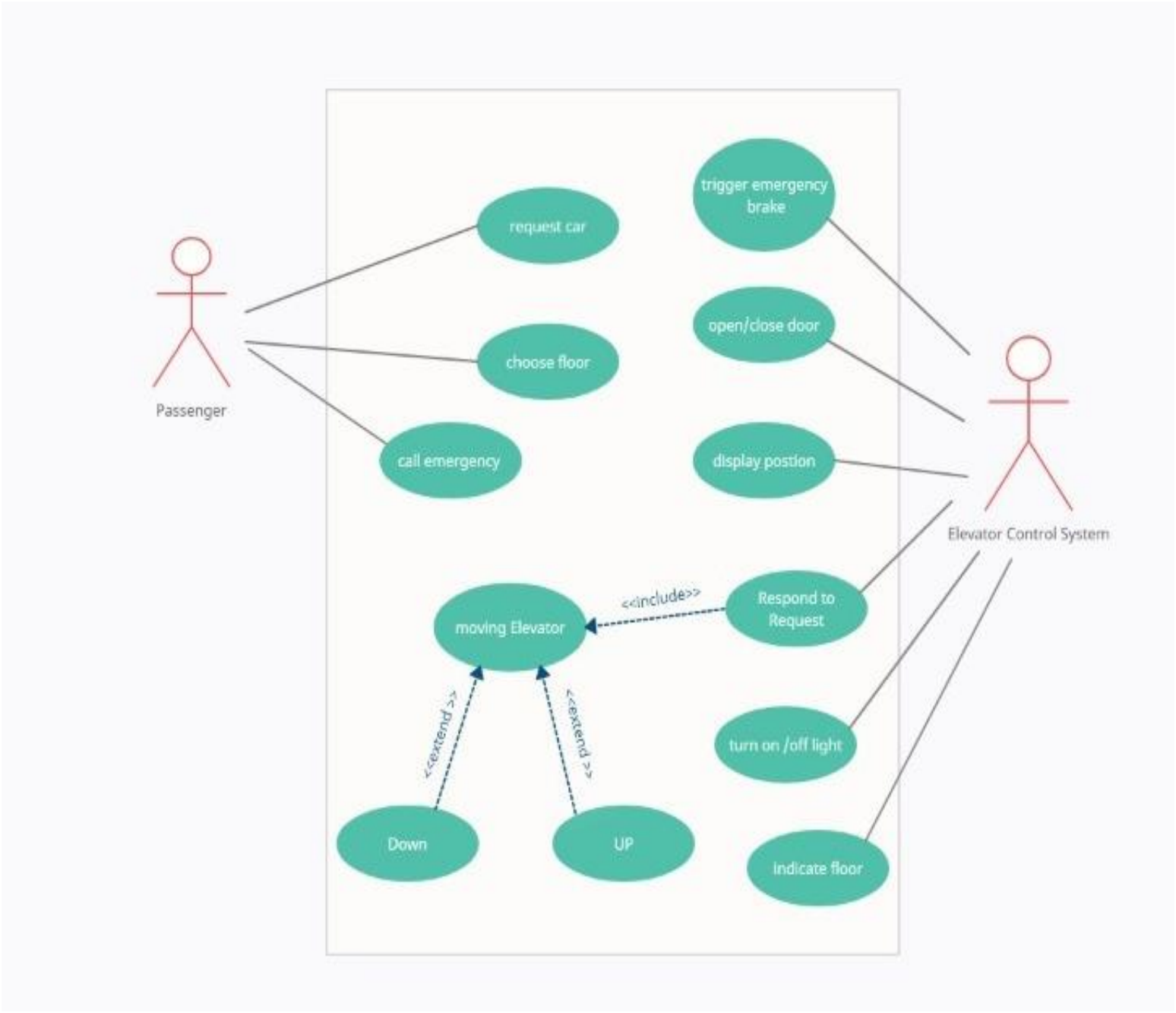
Non-Functional Requirements of the system:

1- Look and feel Requirements :
Colors used in user interface should be professional
2- Usability and humanity Requirements :
Control panel shall be easy to use on the first attempt by a passenger of the public without training.

3 - Performance Requirements:
* The system shall handle 3 requests simultaneously.
* The system shall, on average, work without failure for 30 days
* Closing door should be in 10 sec
* Car moving taking 2 sec to reach another floor
4 – Operational and environmental requirements :
Elevator controller system must work with electricity and emergency high power supply
5 - Maintainability and support Requirements:
* The Elevator controller system shall be able to be modified to cope with minor changes in the Elevator Bank system.
* The Elevator controller system shall be able to be modified to cope with a new car
6 - Culture Requirements
Numbers and instruction of control panel should be clear and formal in English

Diagrams of the system:

1-UseCase Diagram:



Use case description:

Identifier and name: Moving elevator.
Initiator: Elevator control system.
Goal: Elevator moving up or down when the passenger make request and the elevator reach to the target floor.
Pre-condition: The system has an empty place to receive a request or the light is working.
Post-condition: Reach the elevator to target floor.
Include: None.
Extend: Up and Down.
Assumption: The passenger must make request to the system.
Main success scenario:
1- The clicks the button to make request.
2- The indicator response to request.
3- The elevator moves from floor to another floor.
4- The elevator moves up or down depends on the request.
5- The elevator reaches to target floor.
6- The door will open.
7- After 5 second the door will close.
8- If happen object detection the door will open again.

Alternative scenario:
1- The clicks the button to make request.
2- The indicator response to request.
3- The elevator moves from floor to another floor.
4- The elevator moves up or down depends on the request.
5- The elevator reaches to target floor.
6- The door will open.
7- After 5 second the door will close.
8- If happen object detection the door will open again.
9- If the light cut off.
10- The elevator will reach to save floor.
11- The door will open.
12- After 5 second the door will close.

Identifier and name: Request car.
Initiator: Passenger.
Goal: Passenger makes request the car to move from floor to another floor until he reaches the floor he wants.
Pre-condition: The system has an empty place to receive a request or the light is working.
Post-condition: The request reaches to the system and the indicator response to request.
Include: None.
Extend: None.
Assumption: The passenger must make request to the system and the light is working.

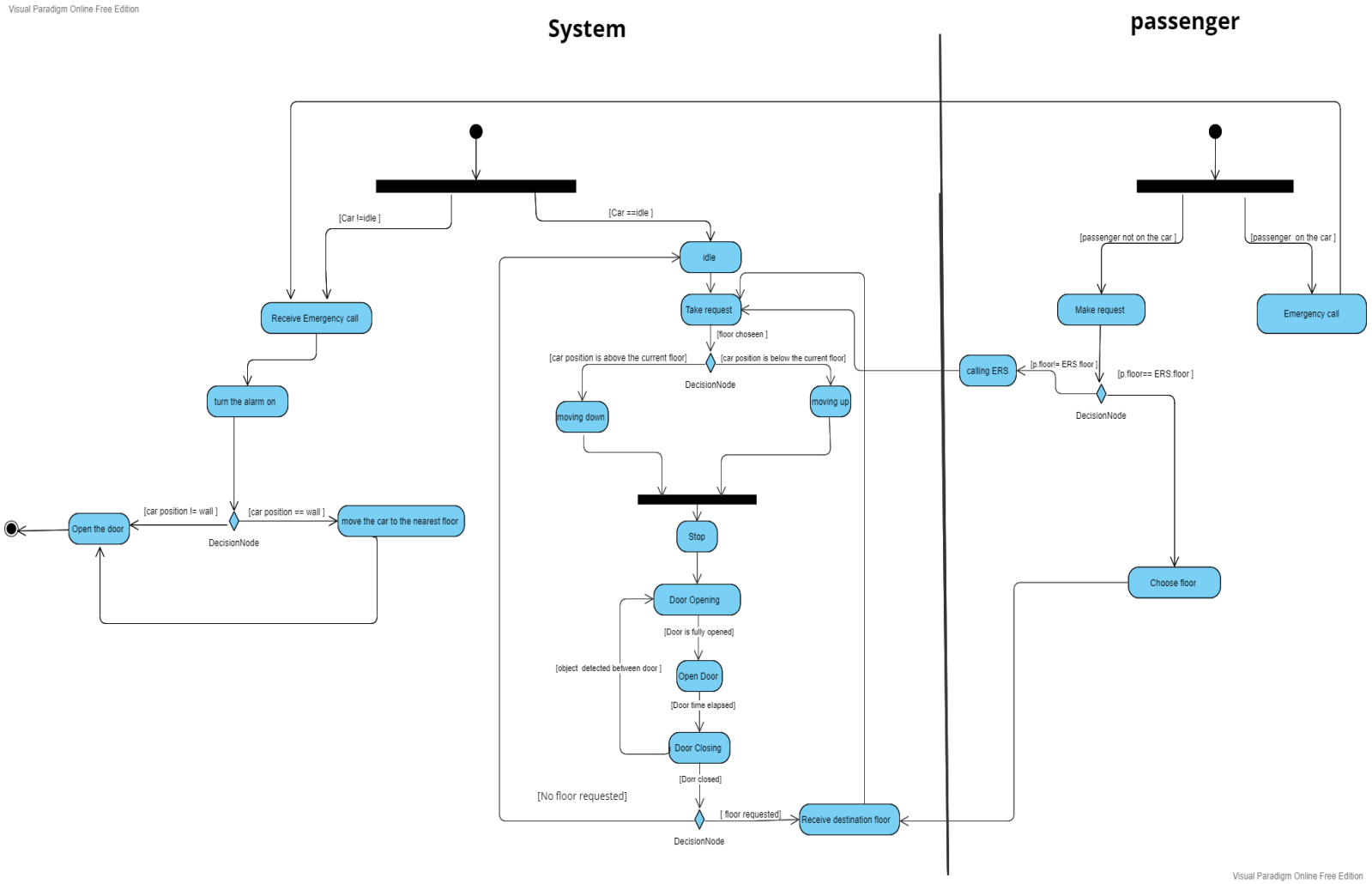
Main success scenario:

1- The clicks the button to make request.
2- The button is turn on.
3- The request records in the system
4- The indicator response to request.
5- The elevator moves from floor to another floor.
6- The elevator moves up or down depends on the request.
7- The elevator reaches to target floor.

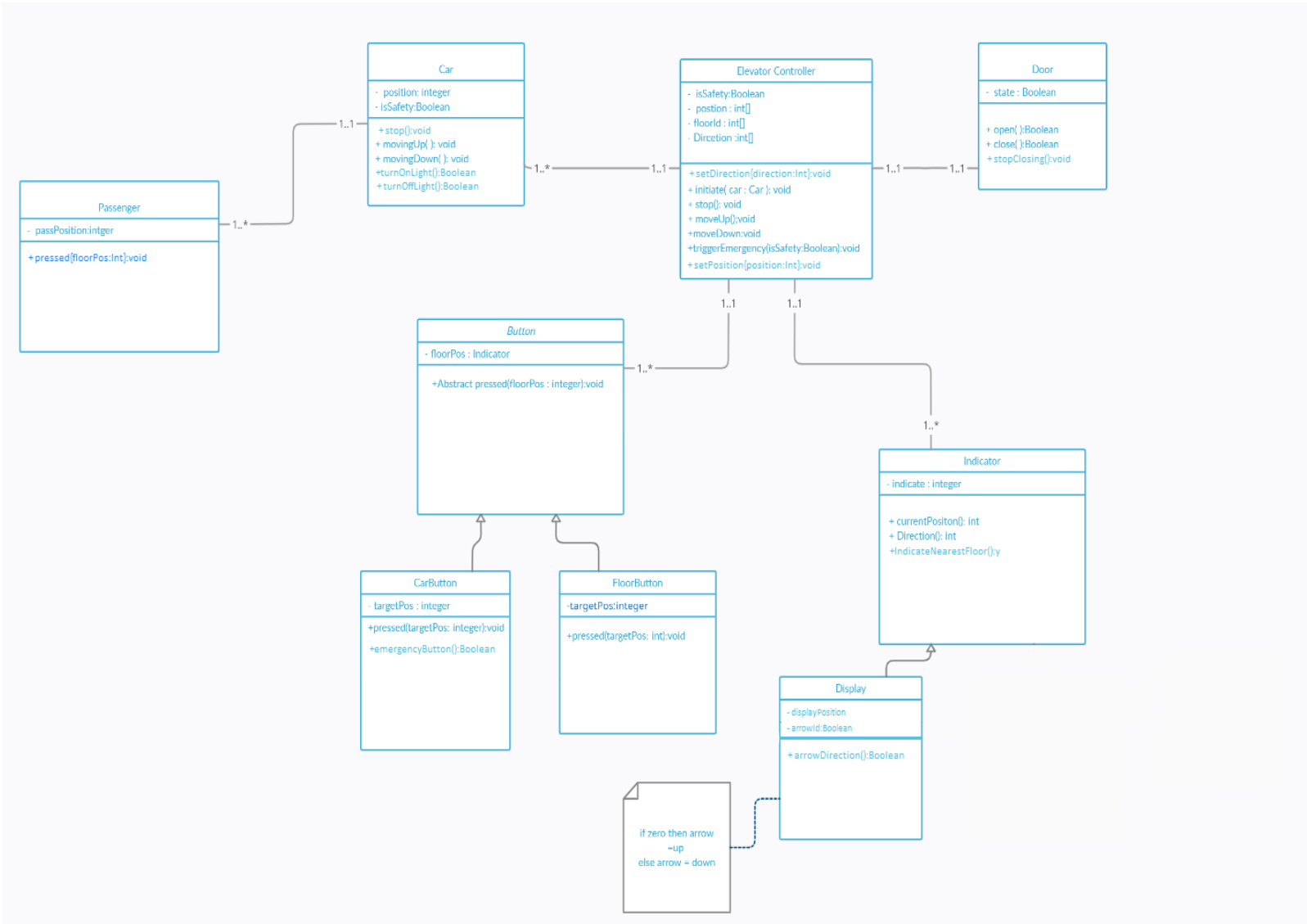
Alternative scenario:

1- The clicks the button to make request.
2- The button is turn on.
3- The request records in the system
4- The indicator response to request.
5- The elevator moves from floor to another floor.
6- The elevator moves up or down depends on the request.
7- The elevator reaches to target floor.
8- The door does not open.
9- Send to the sensor that the door does not open.
10- Try again to open the door.
11- If the door does not open.
12- The alarm will work.

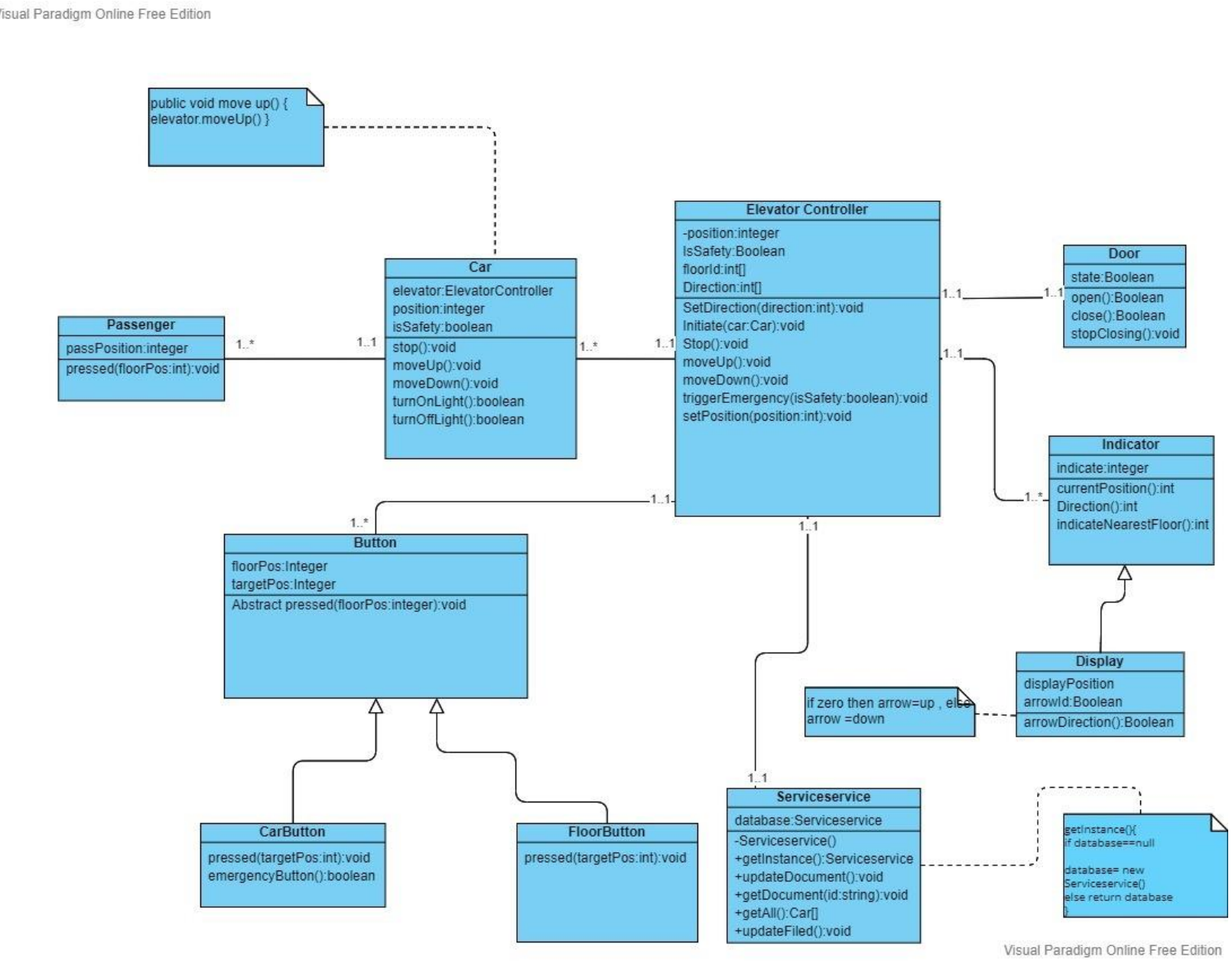
2-Activity Diagram:



3-Class Diagram:



Final class Diagram version with Design patterns:



Design patterns:

SINGLETON PATTERN

Context:

- It is very common to find classes for which only one instance should exist (singleton).
- OUR Example: Serviceservice

Problem:

- How do you ensure that it is never possible to create more than one instance of a singleton class(Serviceservice). And provide a global point of access to it.

Forces:

- The use of a public constructor cannot guarantee that no more than one instance will be created.
- The singleton instance must also be accessible to all classes that require it; therefore it must often be public.

Solution:

- Have the constructor private to ensure that no other class will be able to create an instance of the class singleton (Serviceservice).
- Define a public static method, The first time this method is called, it creates the single instance of the class "singleton" and stores a reference to that object in a static private variable.

Delegation Pattern

Context:

- When designing the methods "movingUp ()" and "movingDown ()" and "stop ()" in "Car" class, it makes us realize that there is another class that is "ElevatorController" has a method which provides the required service.
- Inheritance is not appropriate (e. g because the is -a does not apply)

Problem:

- How you most effectively make use of a that already exists in the other class?

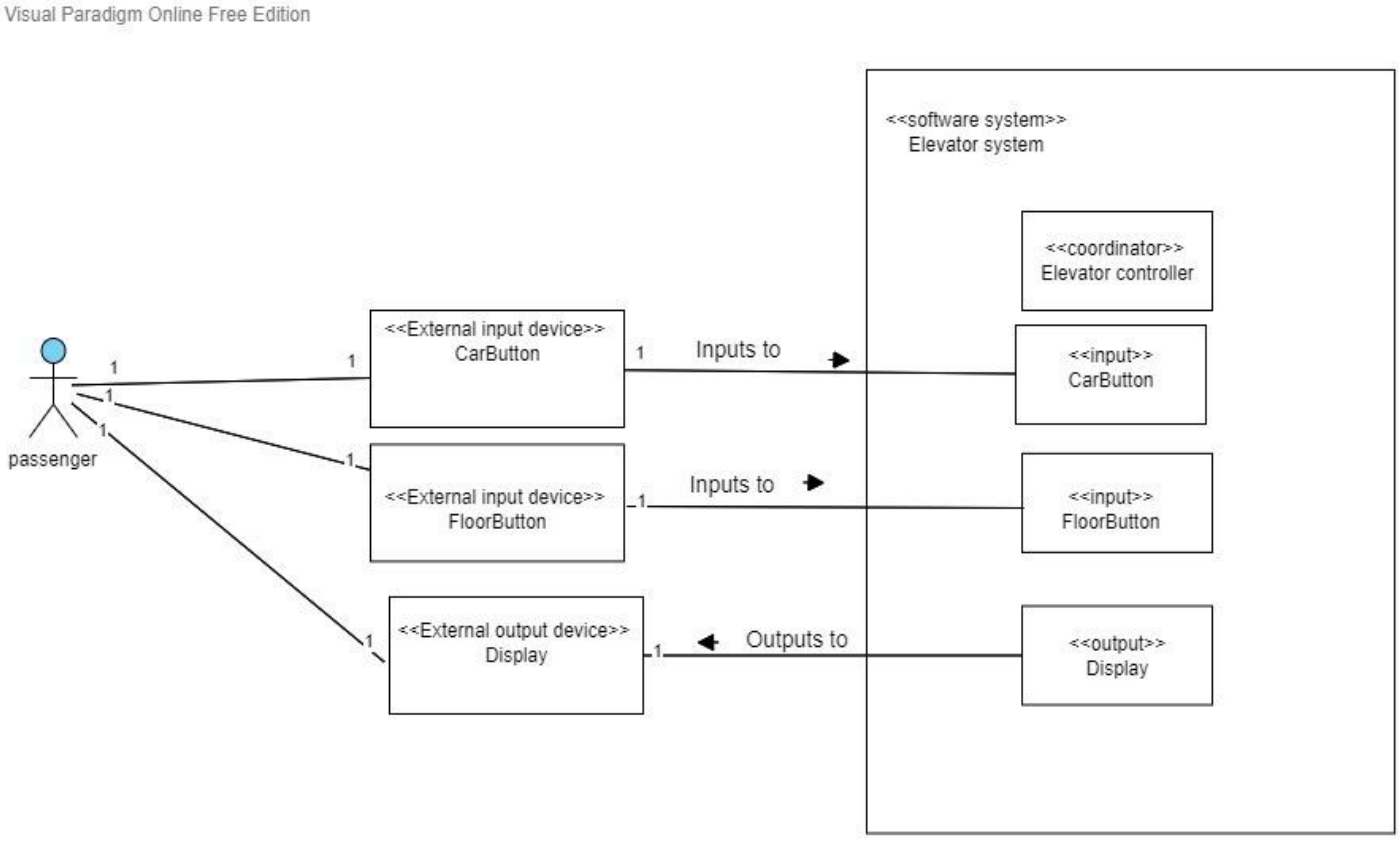
Forces:

- You want to minimize development cost reusing methods.

Solution:

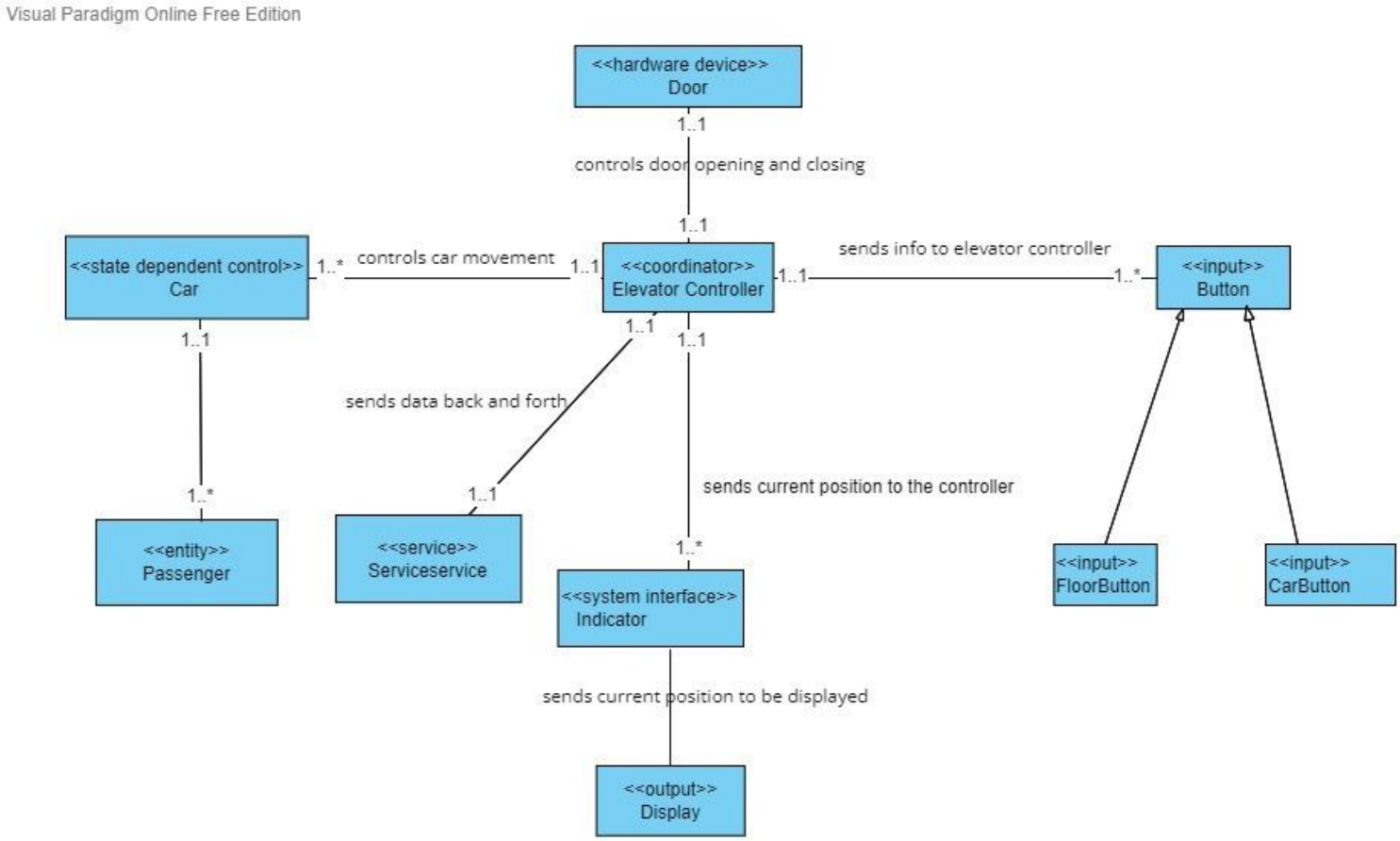
- The delegating the methods "movingUp ()" and "movingDown ()" and "stop ()" in delegator class "Car" calls methods "movingUp ()" and "movingDown ()" and "stop ()" in the delegate class "ElevatorController" to perform the required ask. An association must between the delegator and delegate classes.

Class Diagram with External Classes:

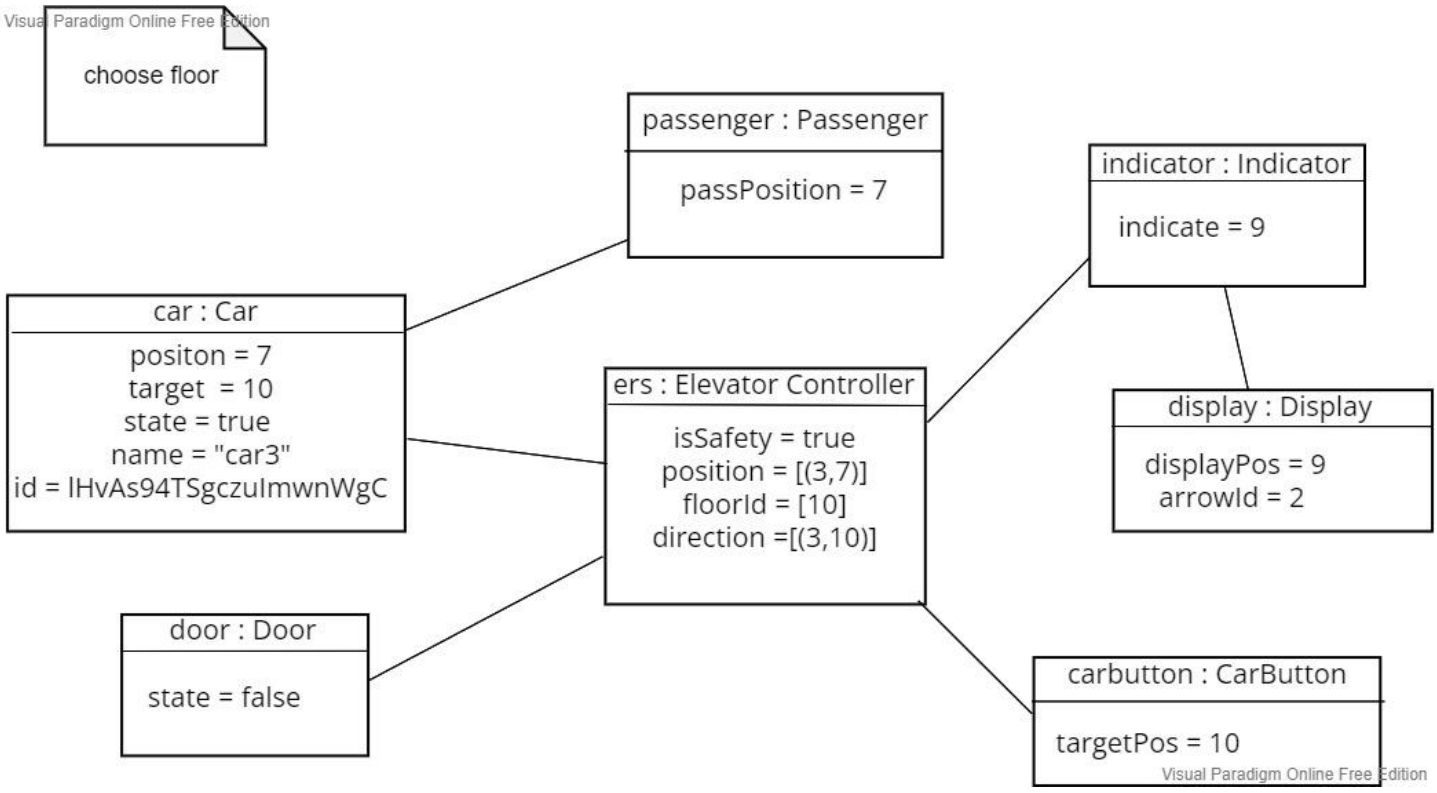


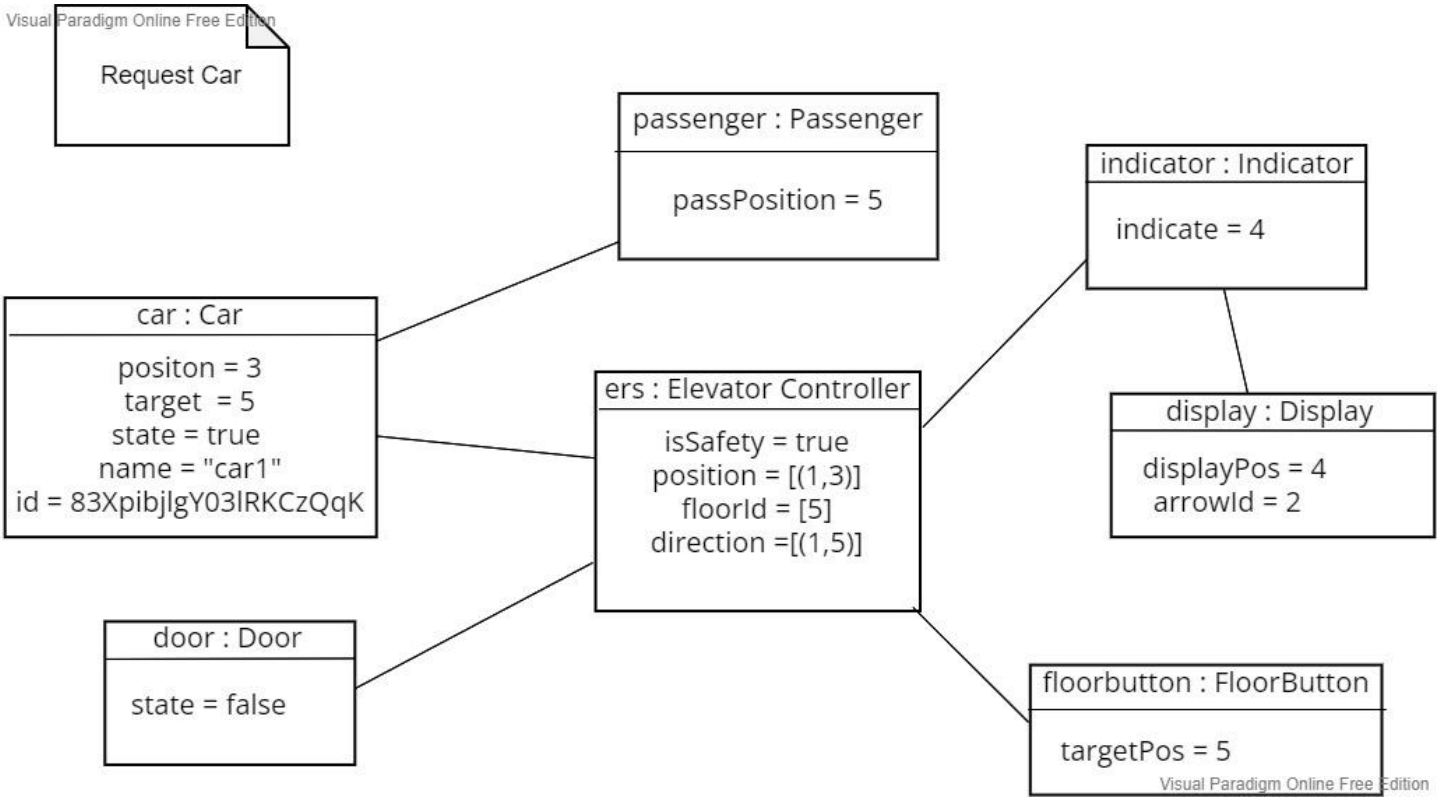
Elevator control system External classes and boundary classes

Class diagram with stereotypes:



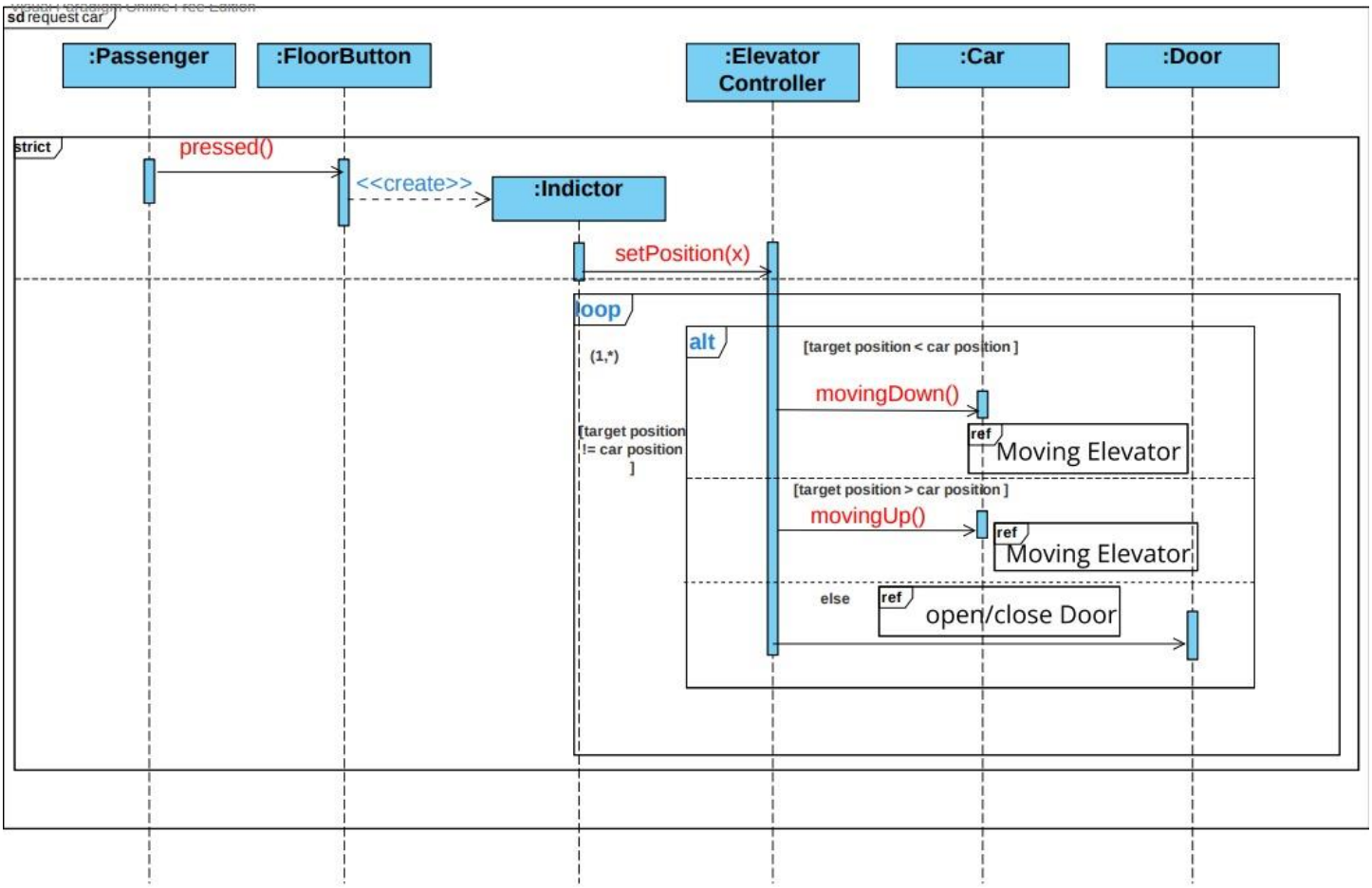
Object diagrams:



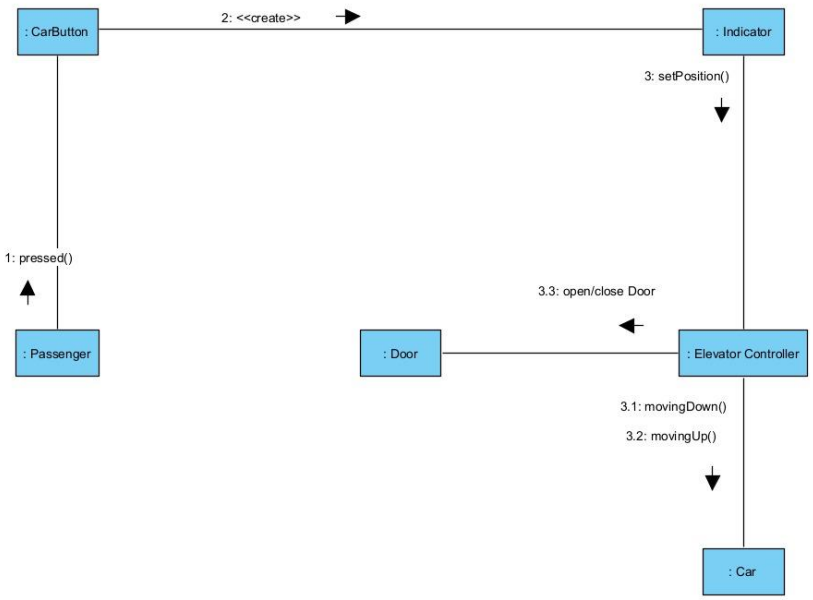


4- Sequence & collaboration 1st version Diagrams:

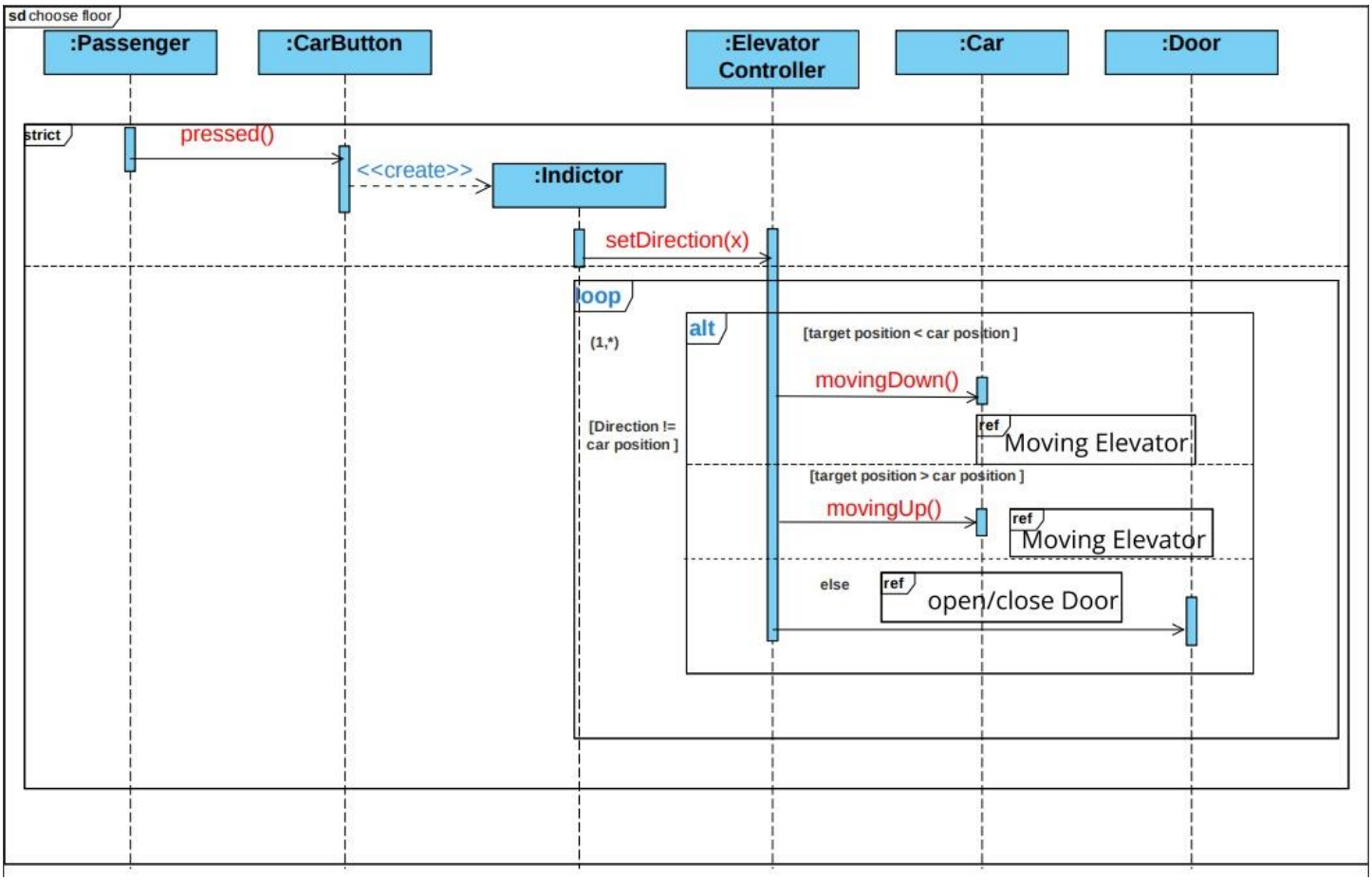
1- Request car:



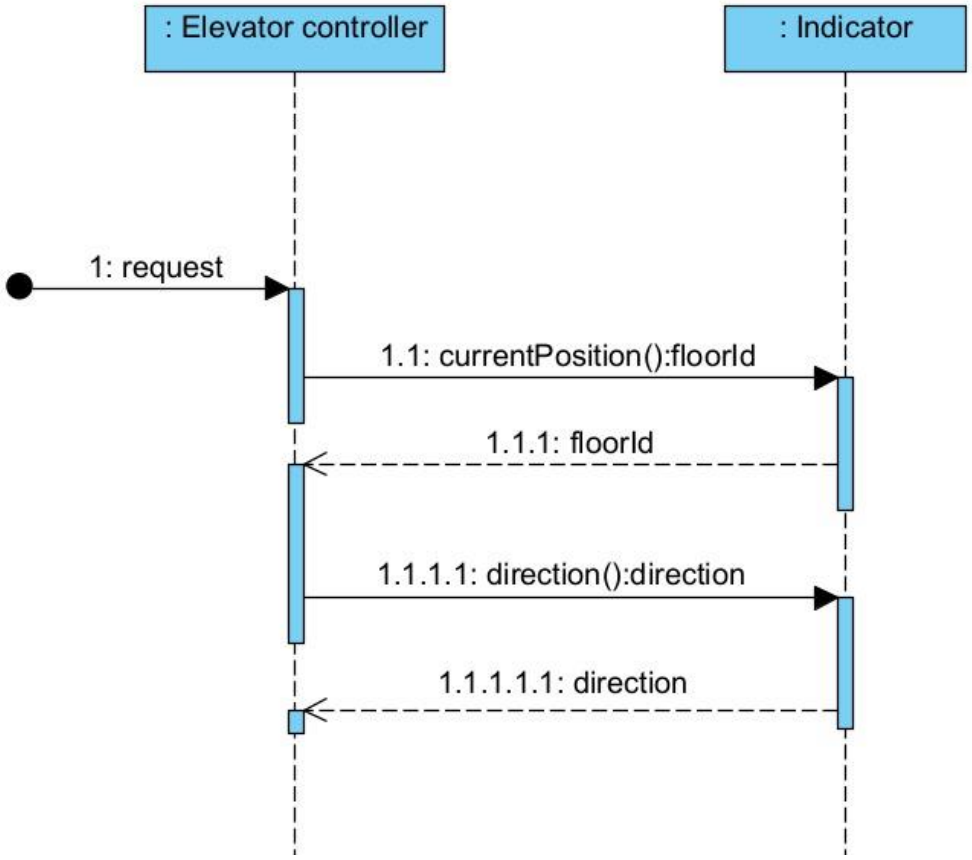
Collaboration :



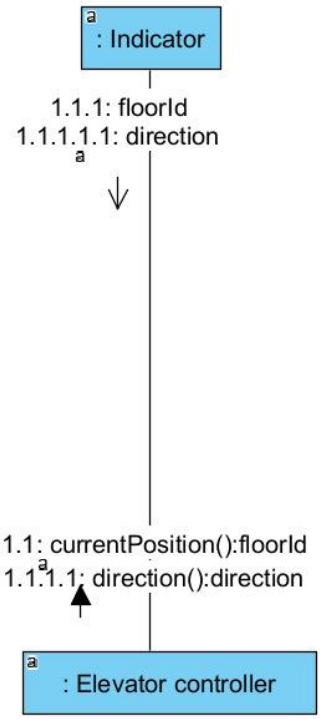
2-Choose floor:



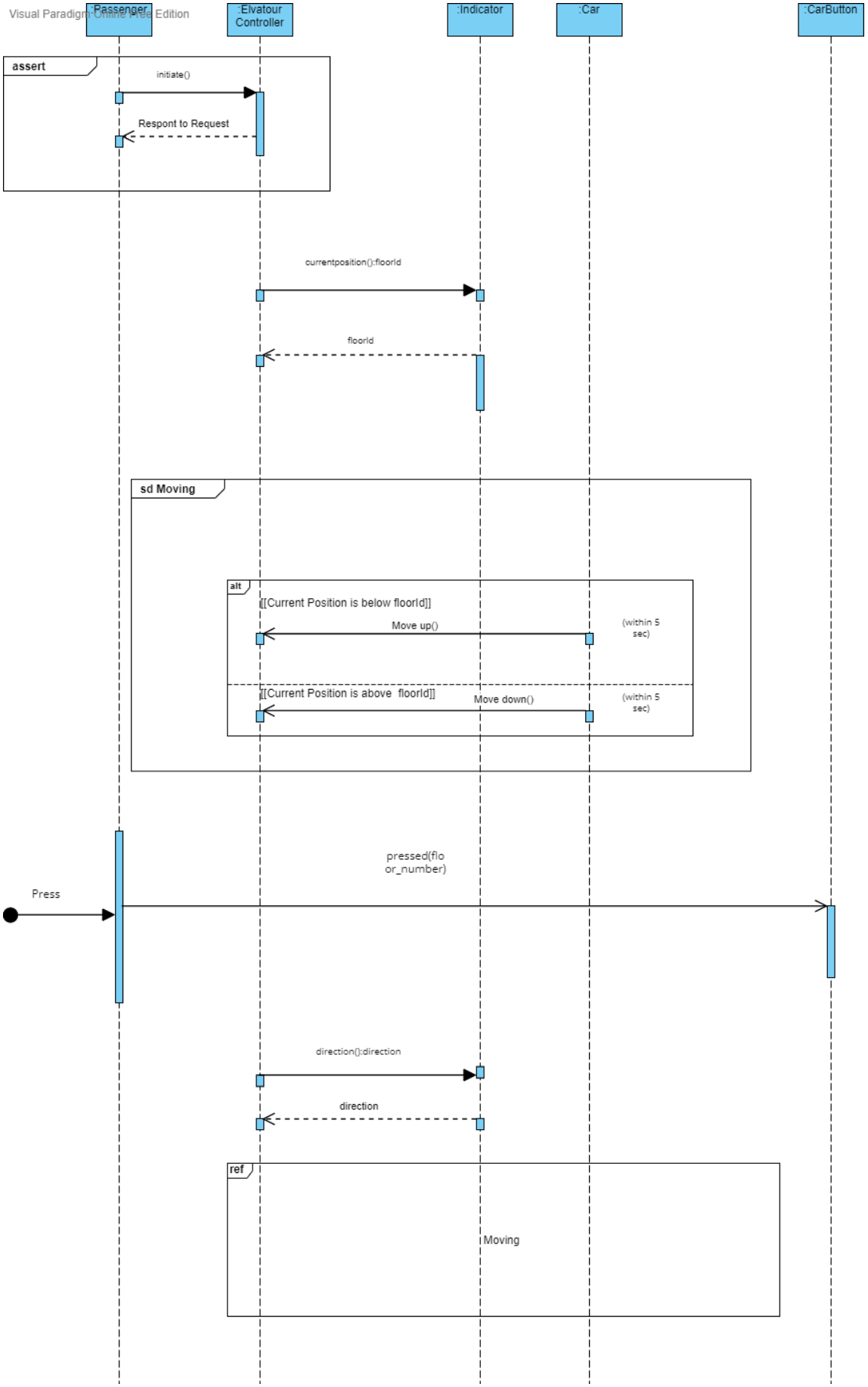
3- Indicate floor:



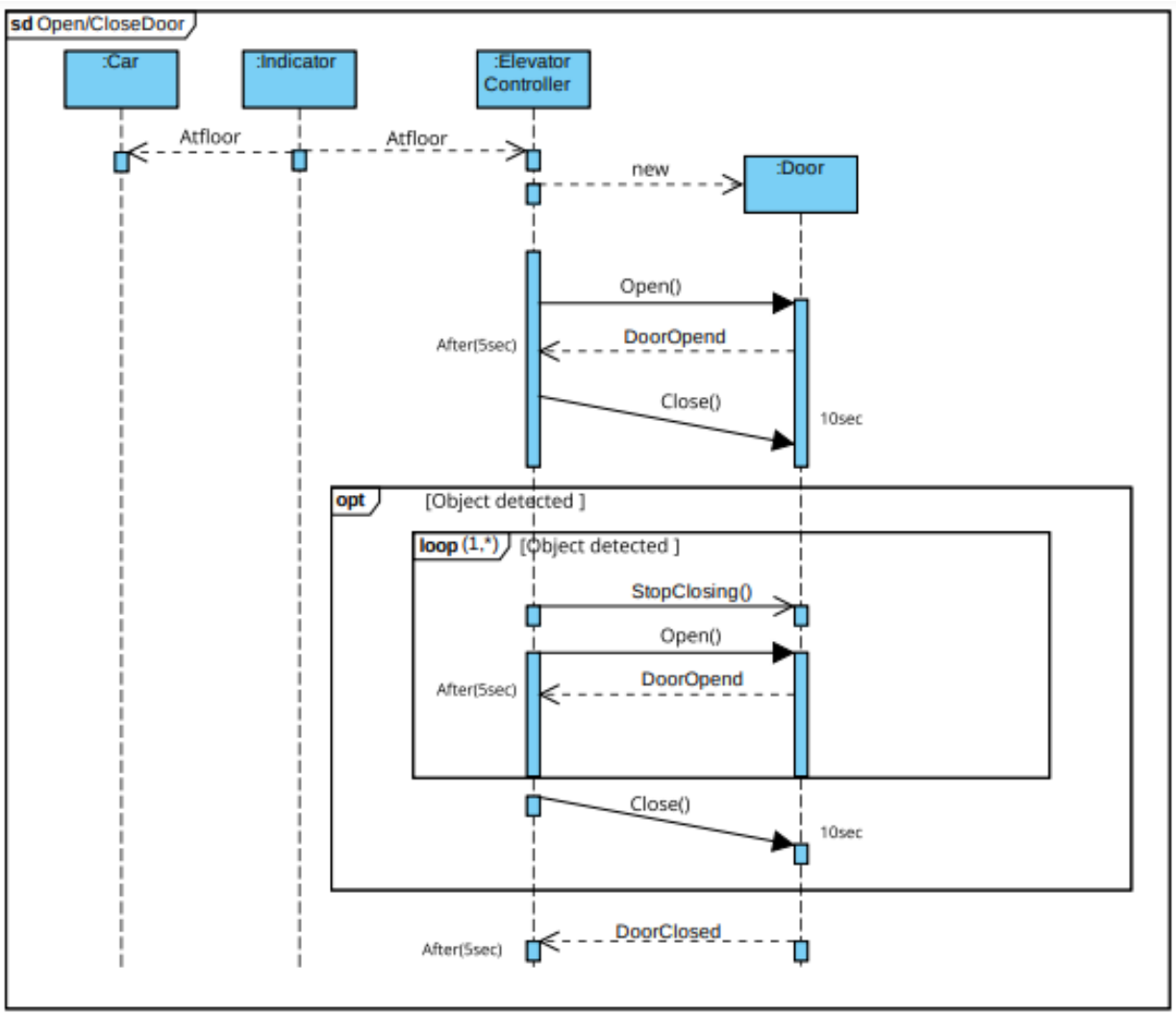
Collaboration:



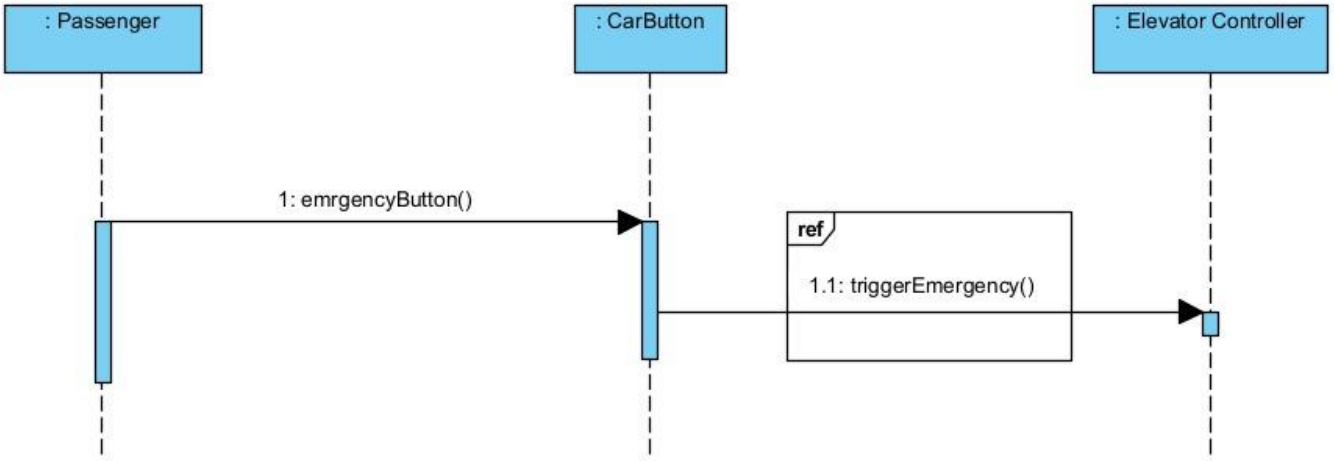
4-Moving Elevator:



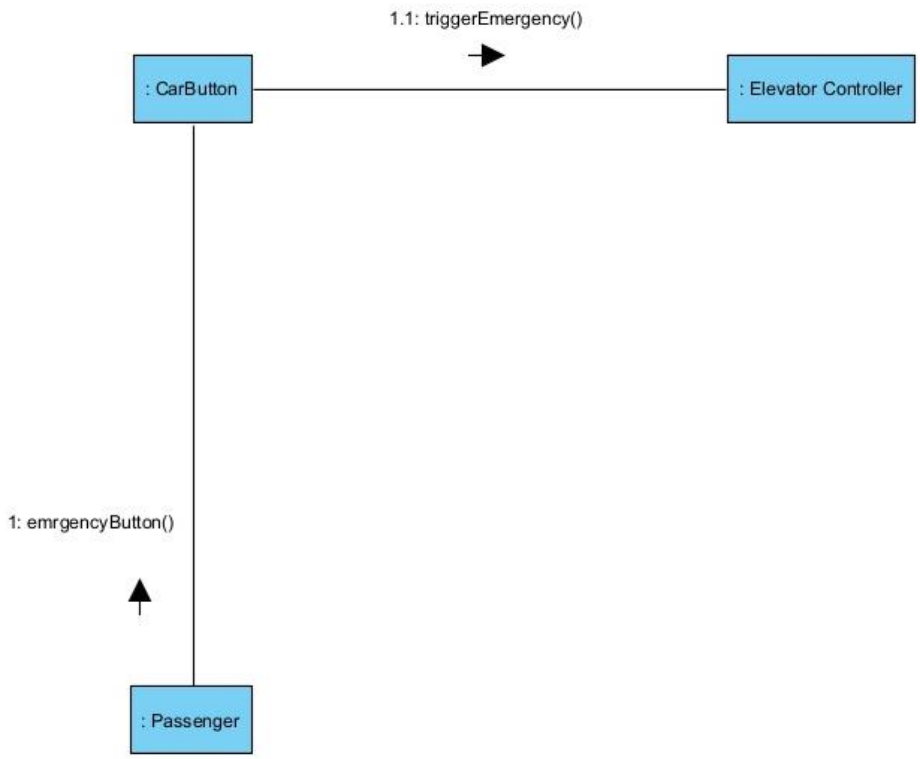
5-Open/Close door:



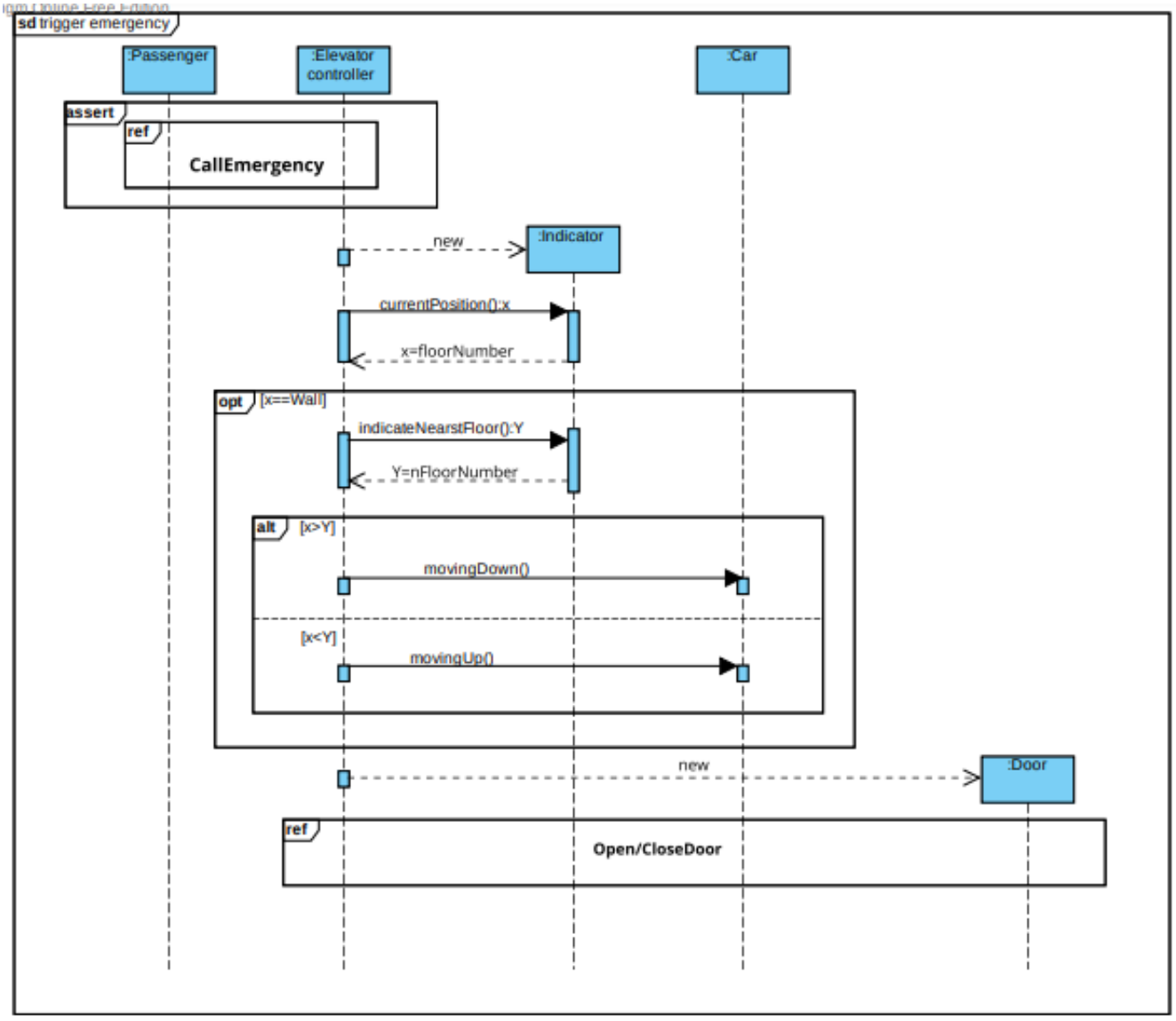
6- Call emergency:



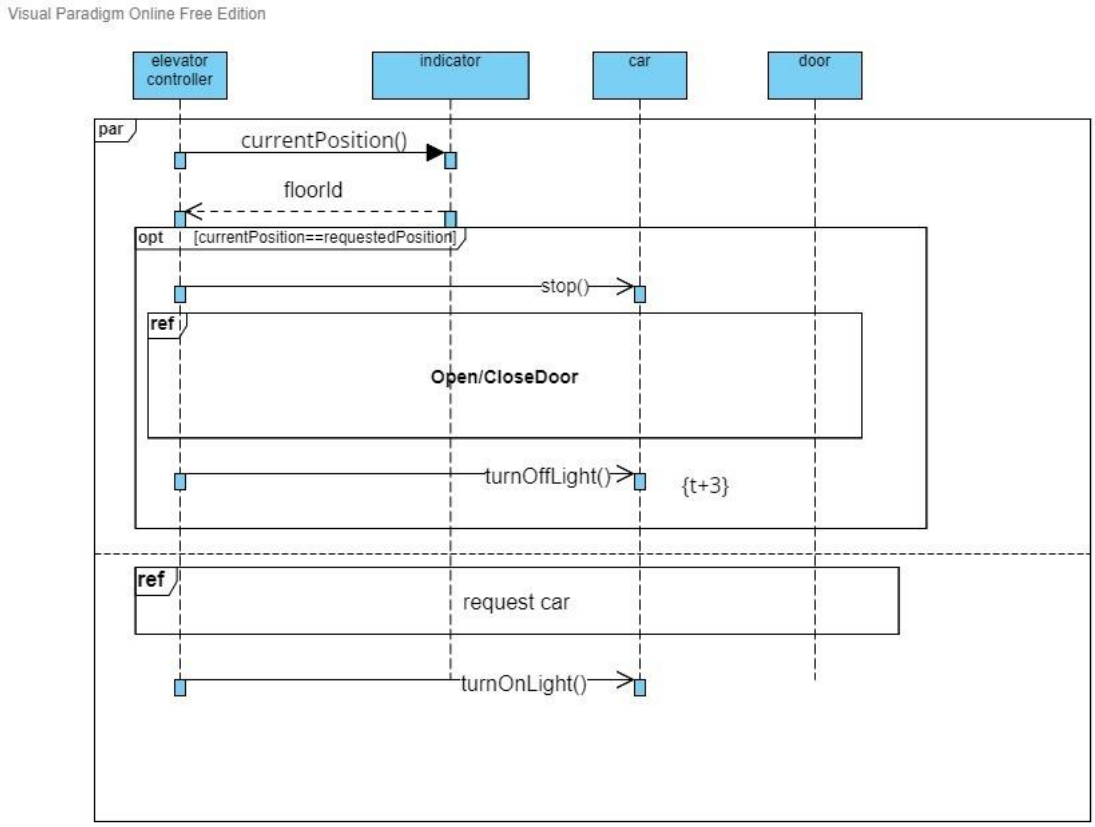
collaboration:



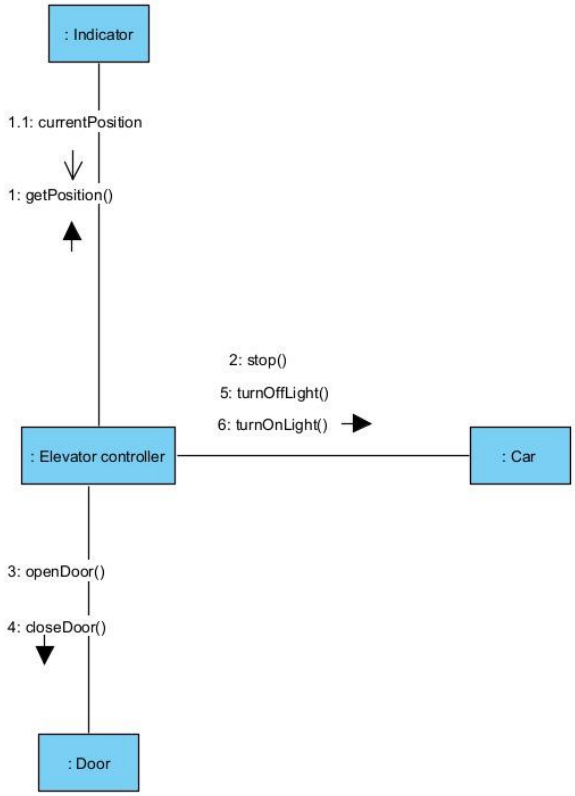
7-Trigger emergency:



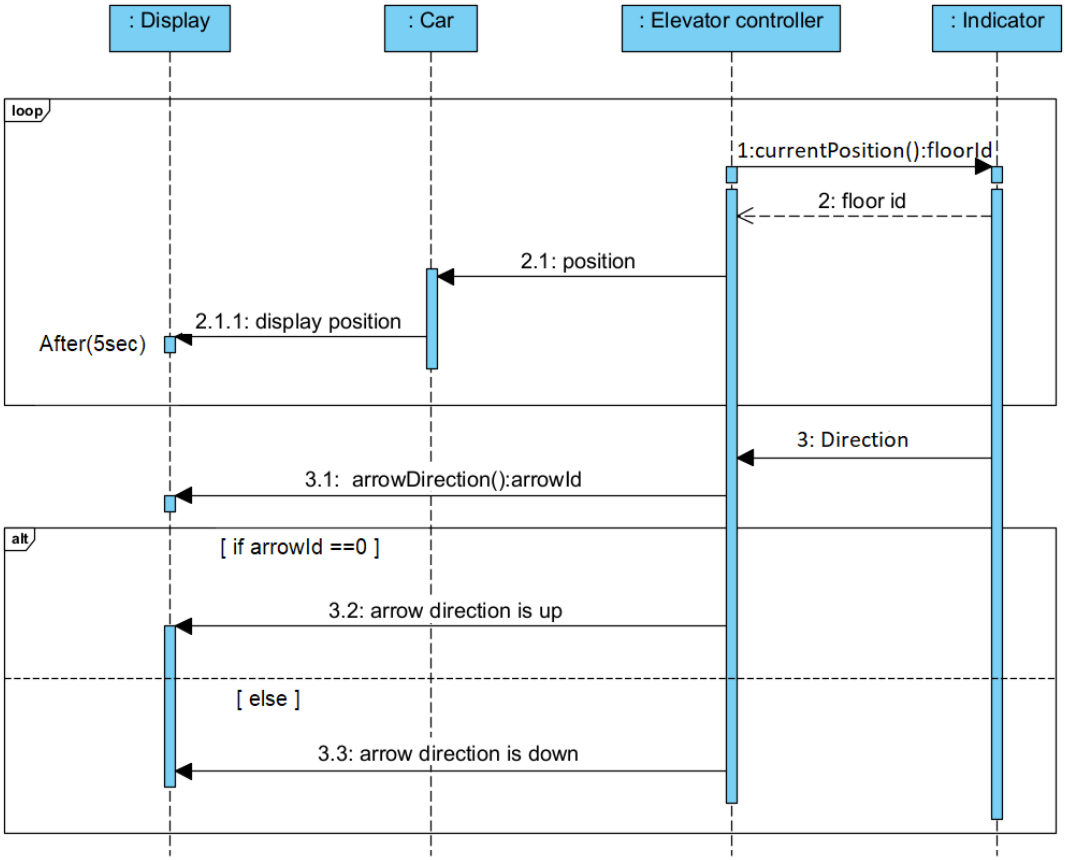
8- Turn on/off light



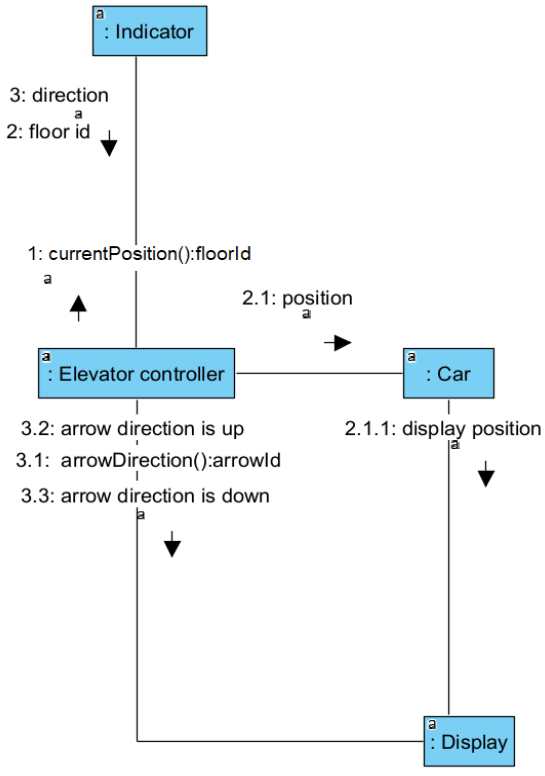
Collaboration:



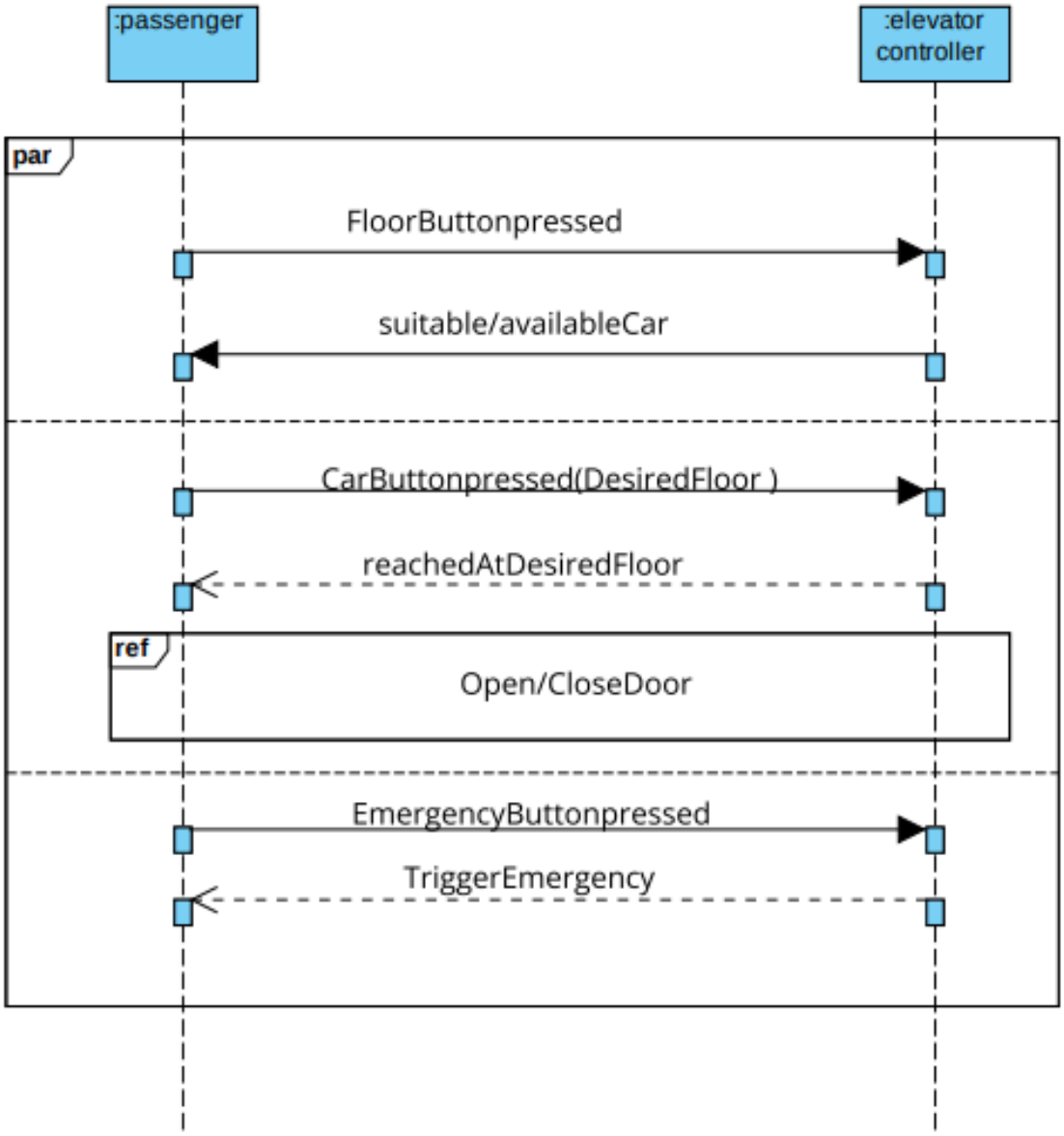
9- Display position:



Collaboration:

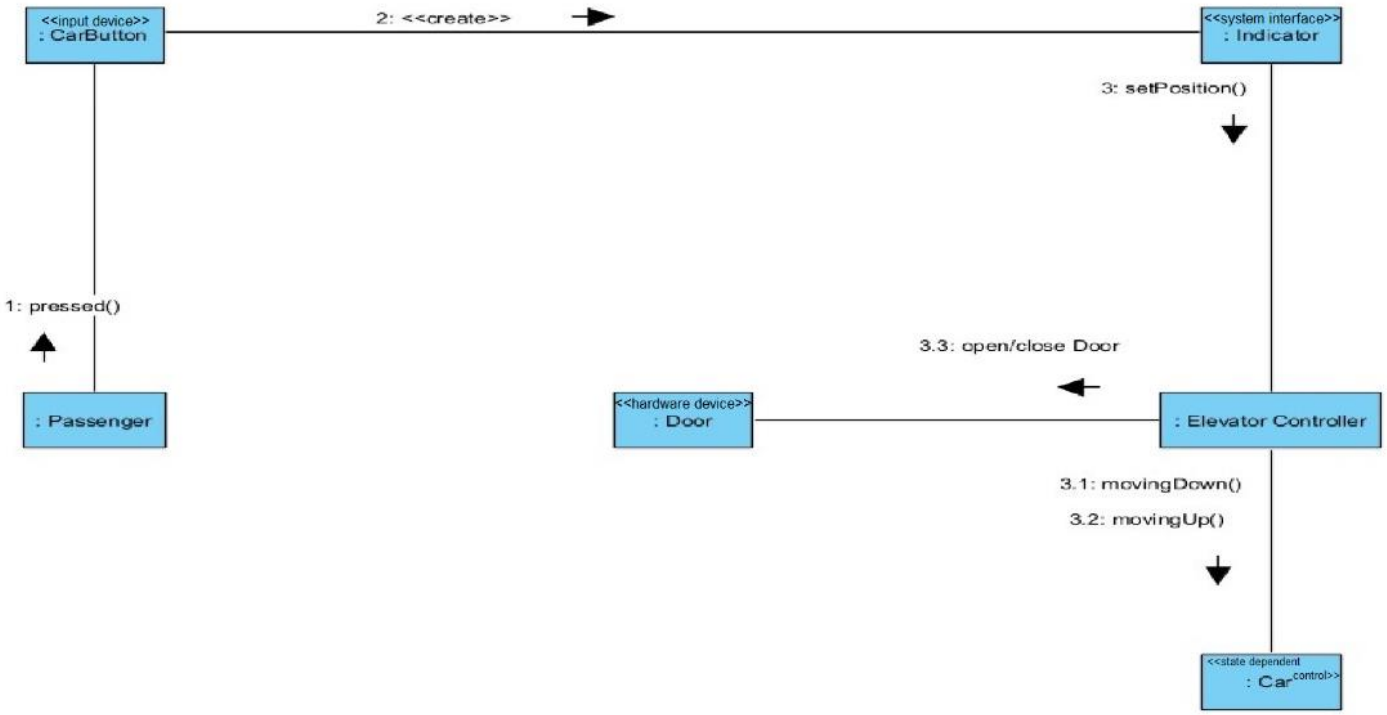


System Sequence Diagram:

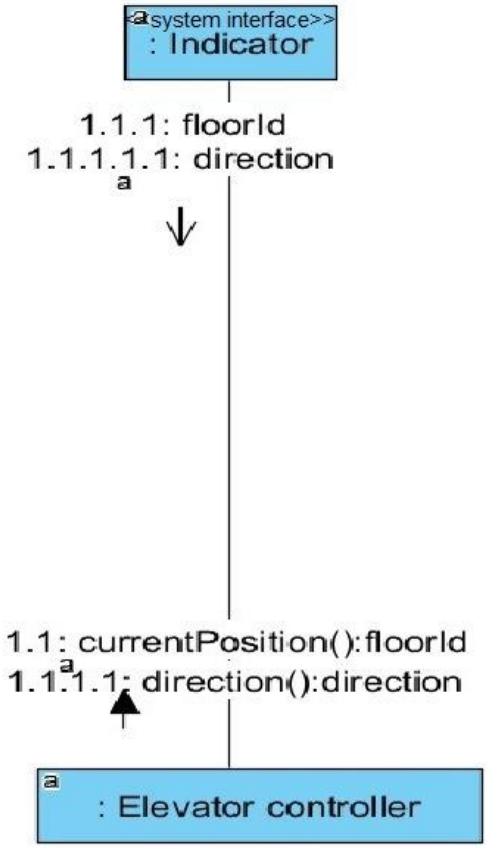


Collaboration 2nd Diagrams :

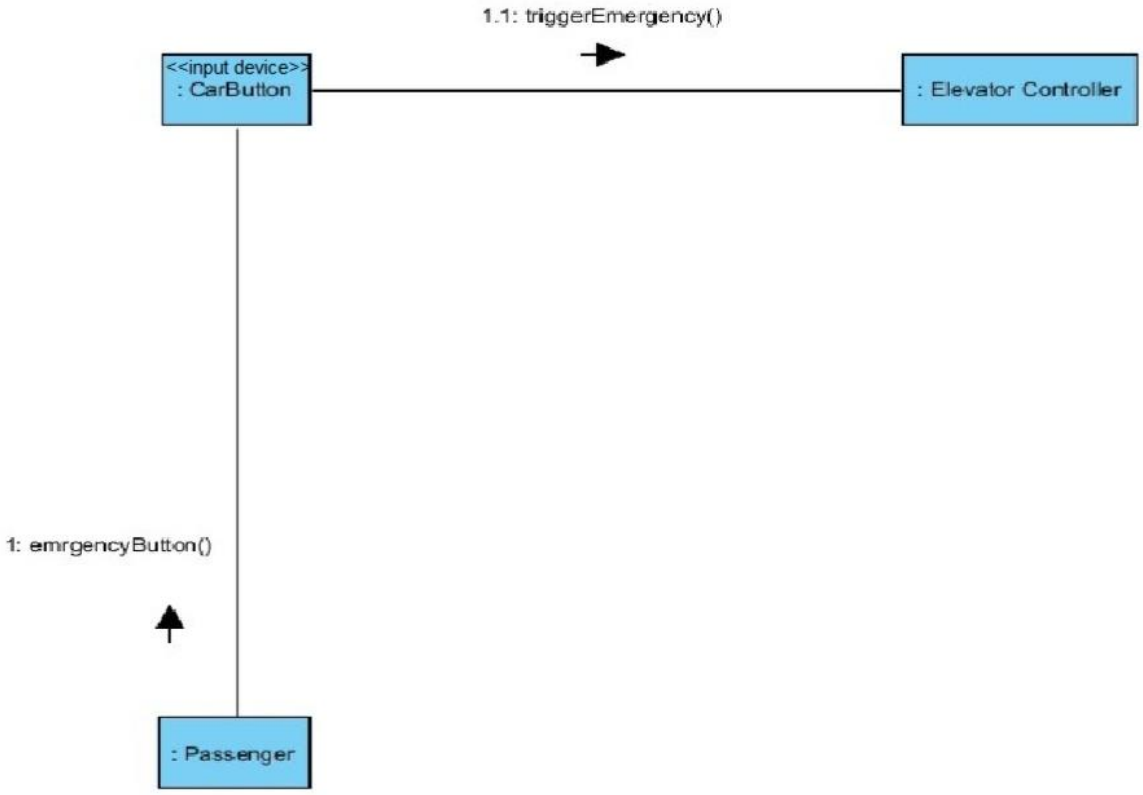
1- Request car:



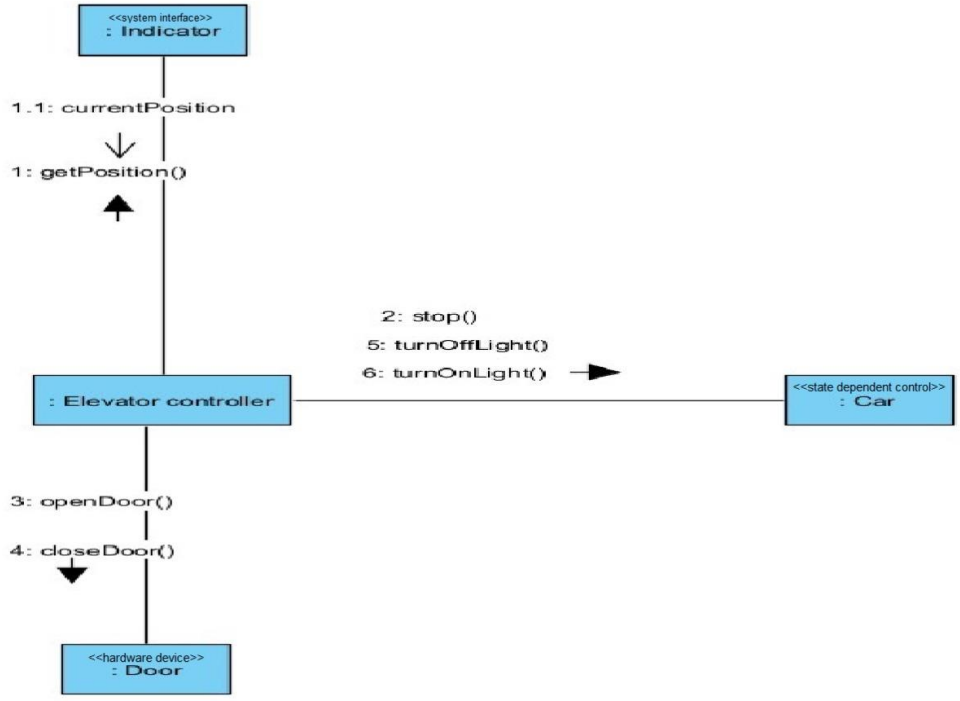
2- Indicate floor:



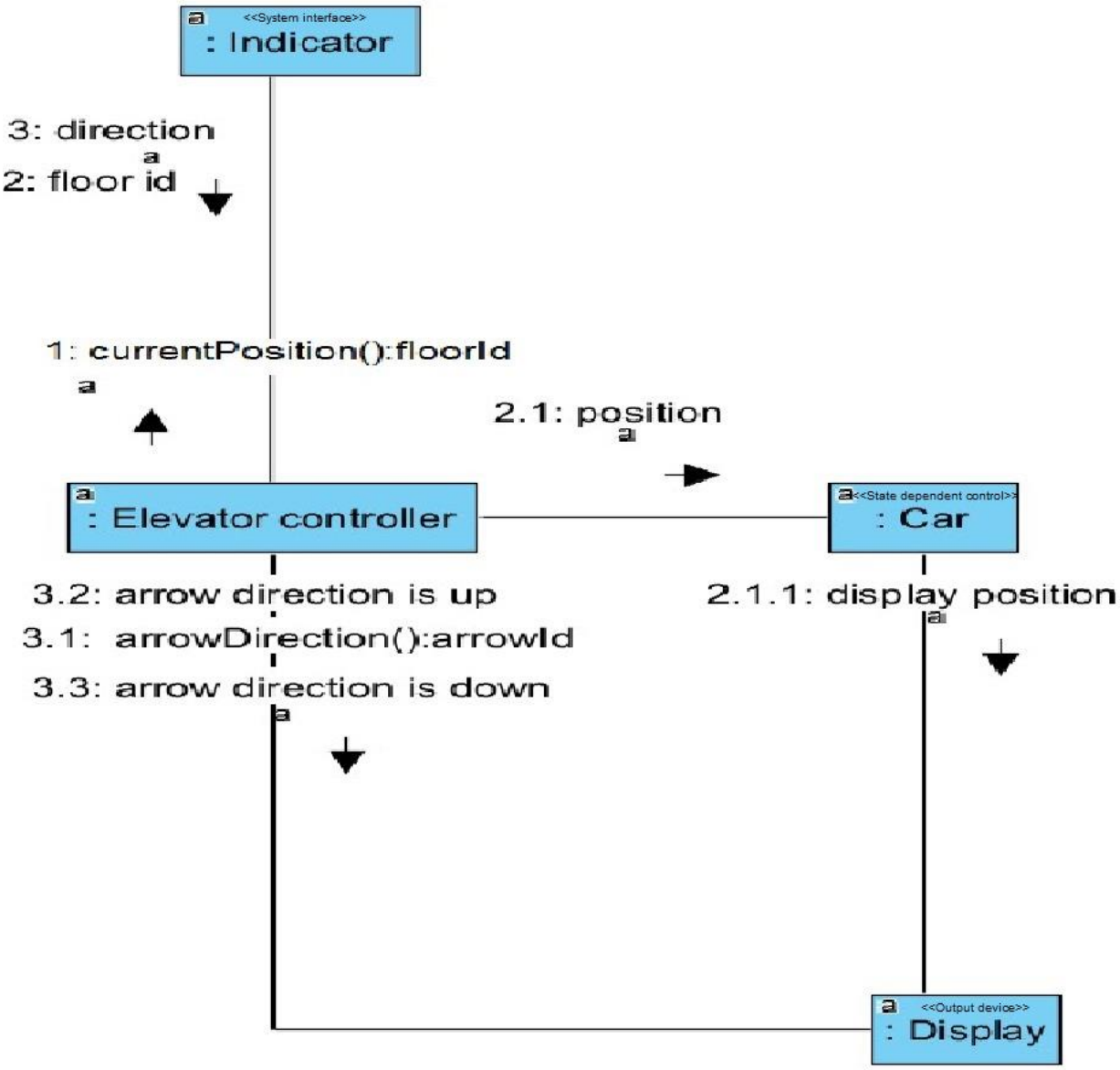
3- Call emergency:



4- Turn on/off light:

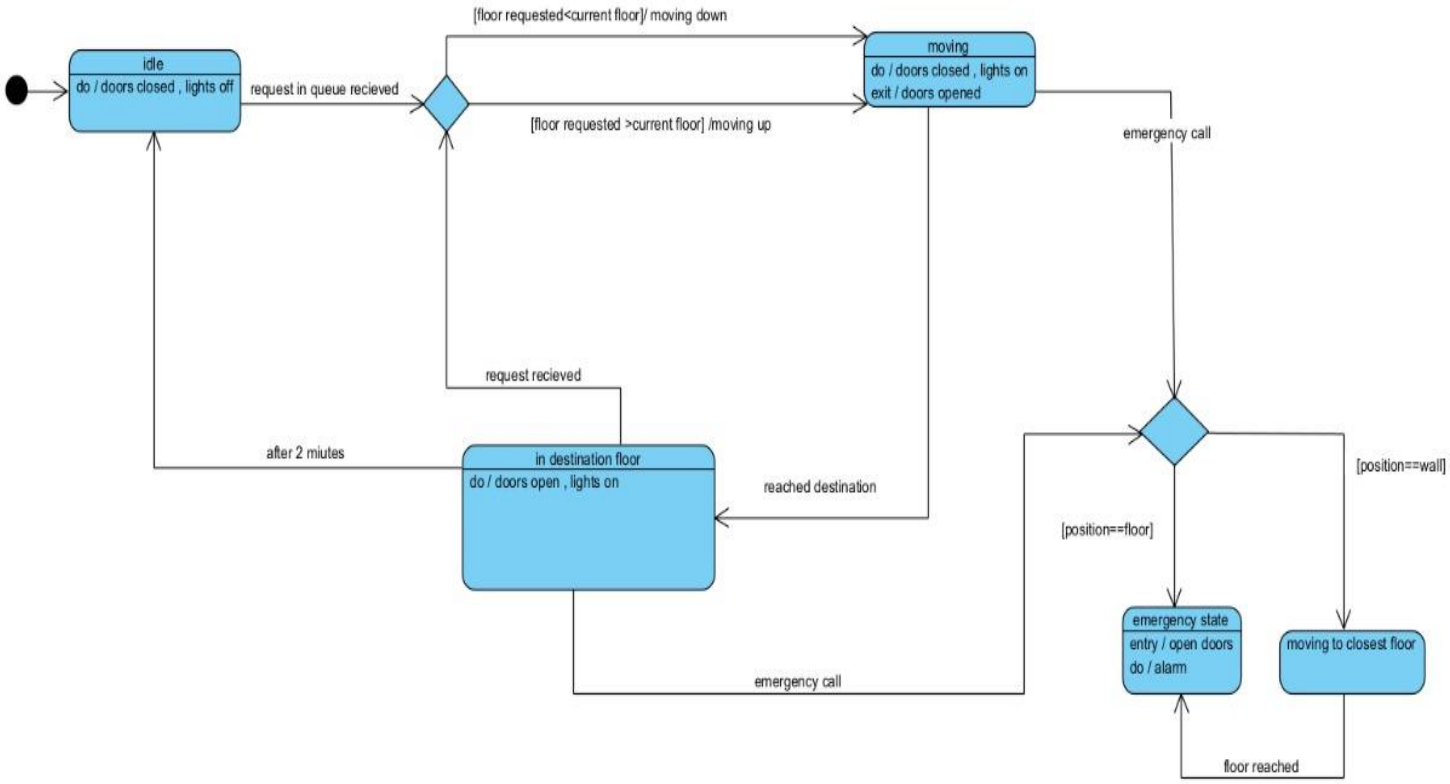


5- Display position:

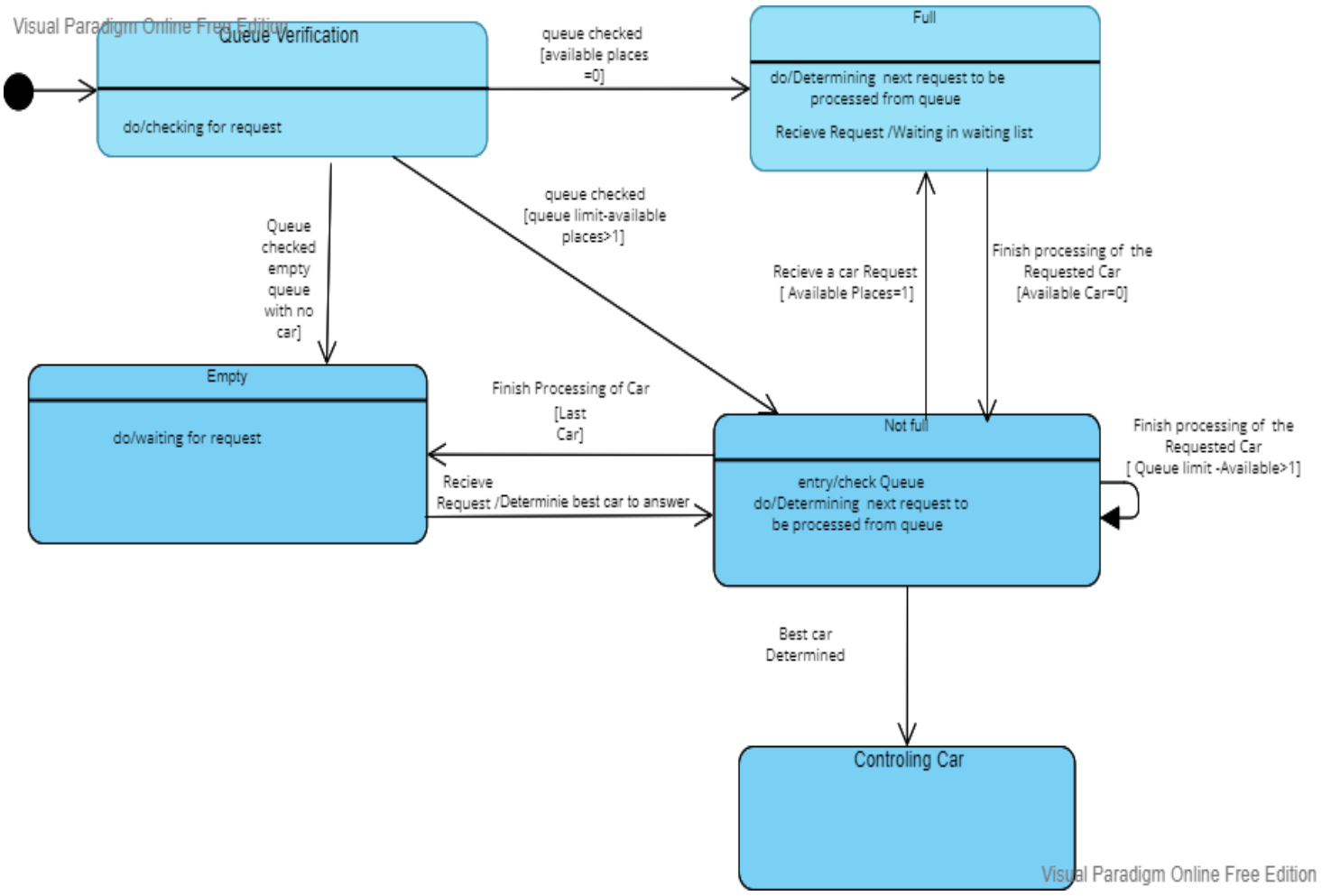


5-State Machine Diagrams:

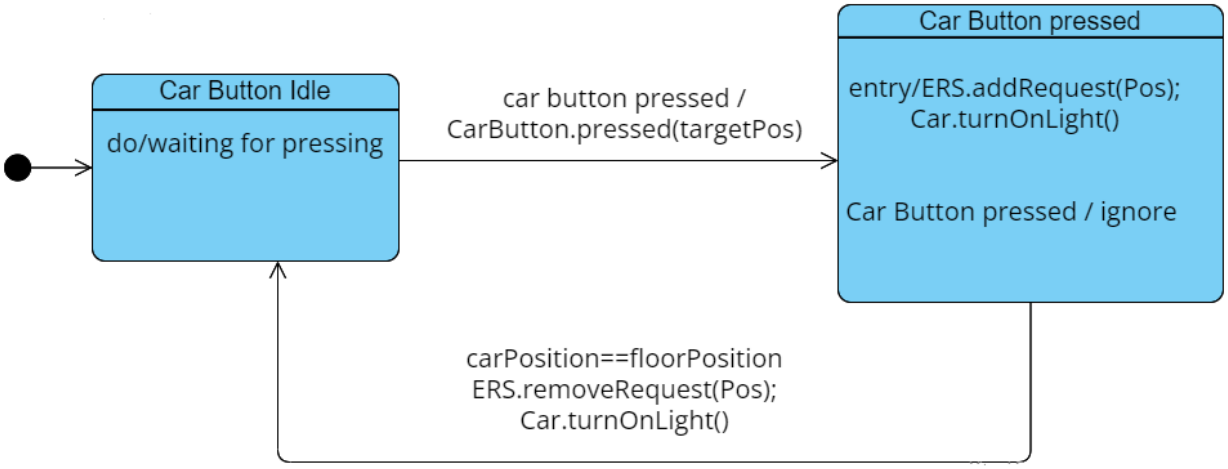
1- Car Controller:



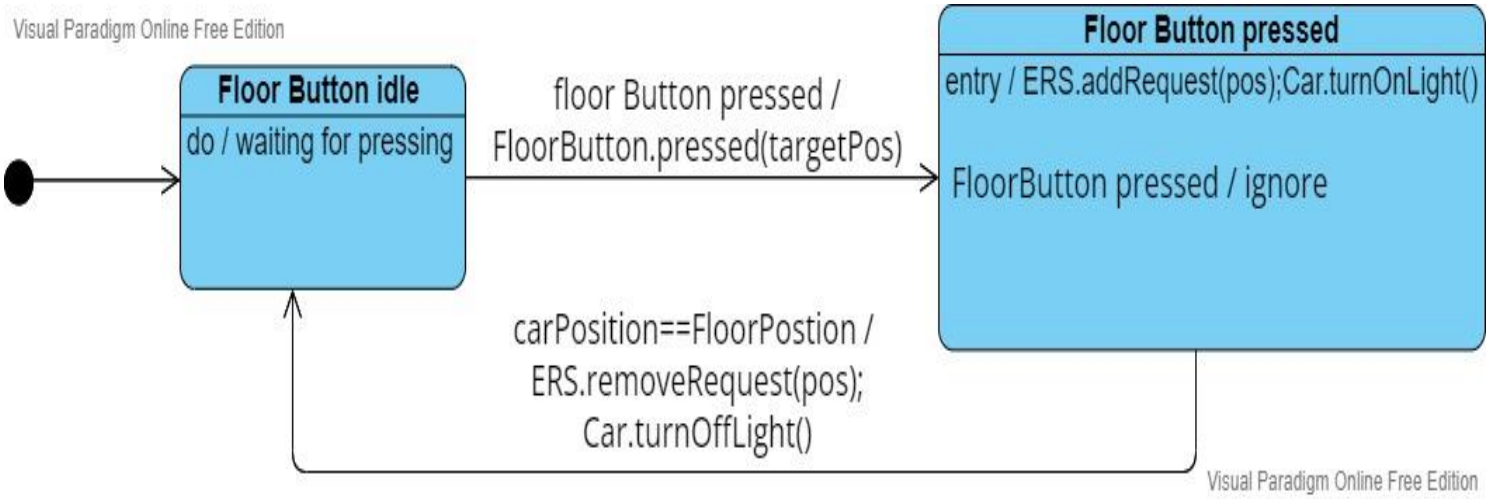
2- Bank list:


Visual Paradigm Online Free Edition
Queue Verification
do/checking for request
queue checked
[available places
=0]
Full
do/Determining next request to be
processed from queue
Recieve Request /Waiting in waiting list
Queue
checked
empty
queue
with no
car]
queue checked
[queue limit-available
places>1]
Recieve a car Request
[Available Places=1]
Finish processing of the
Requested Car
[Available Car=0]
Empty
do/waiting for request
Finish Processing of Car
[Last
Car]
Not full
entry/check Queue
do/Determining next request to
be processed from queue
Finish processing of the
Requested Car
[Queue limit -Available>1]
Recieve
Request /Determinie best car to answer
Best car
Determined
Controling Car
Visual Paradigm Online Free Edition

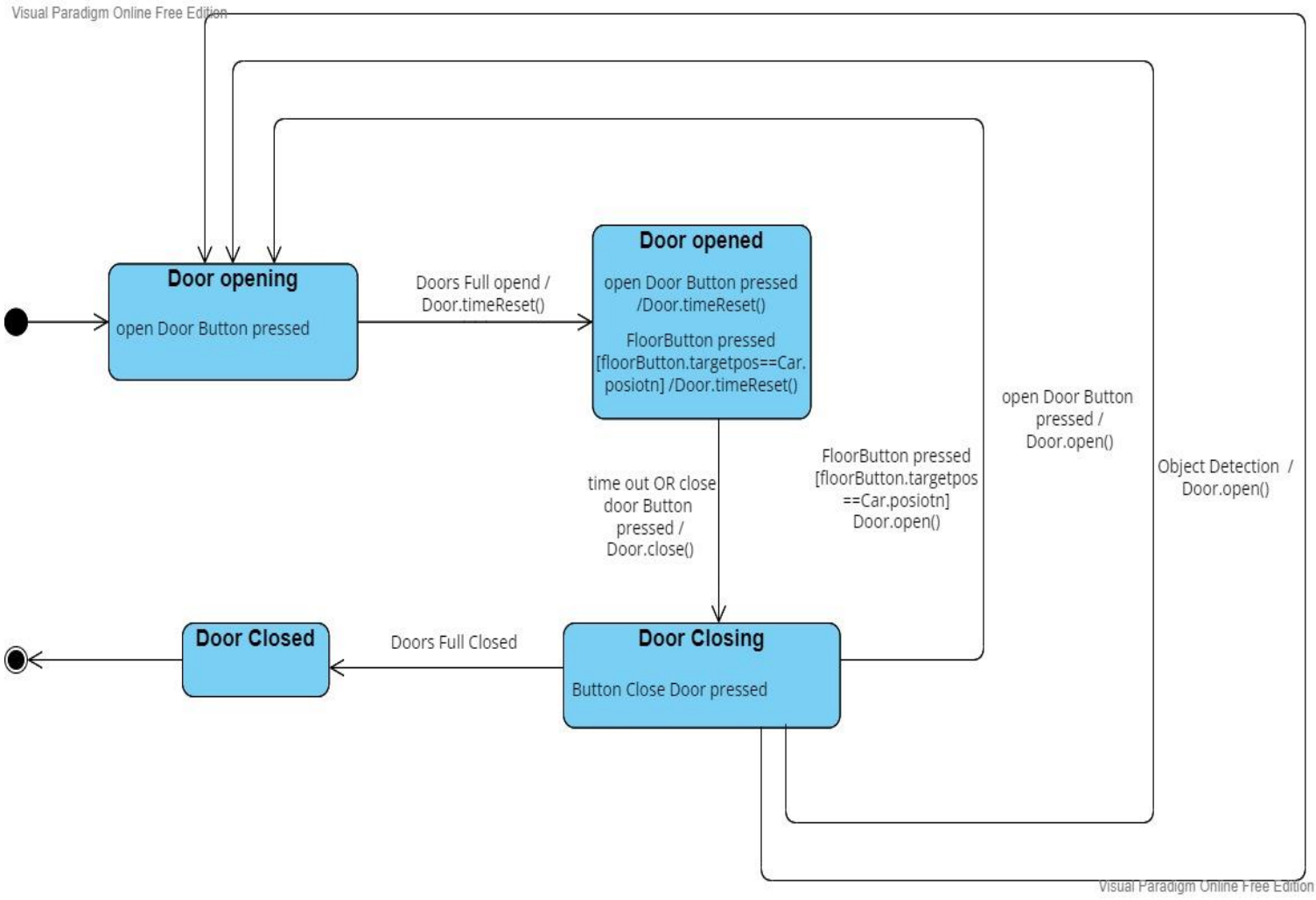
3- Car request:



4- Floor request:



5-Open/Close door:



6- Car button:

7- Floor button:

Stimuli/Response Identification (State Transition Table):

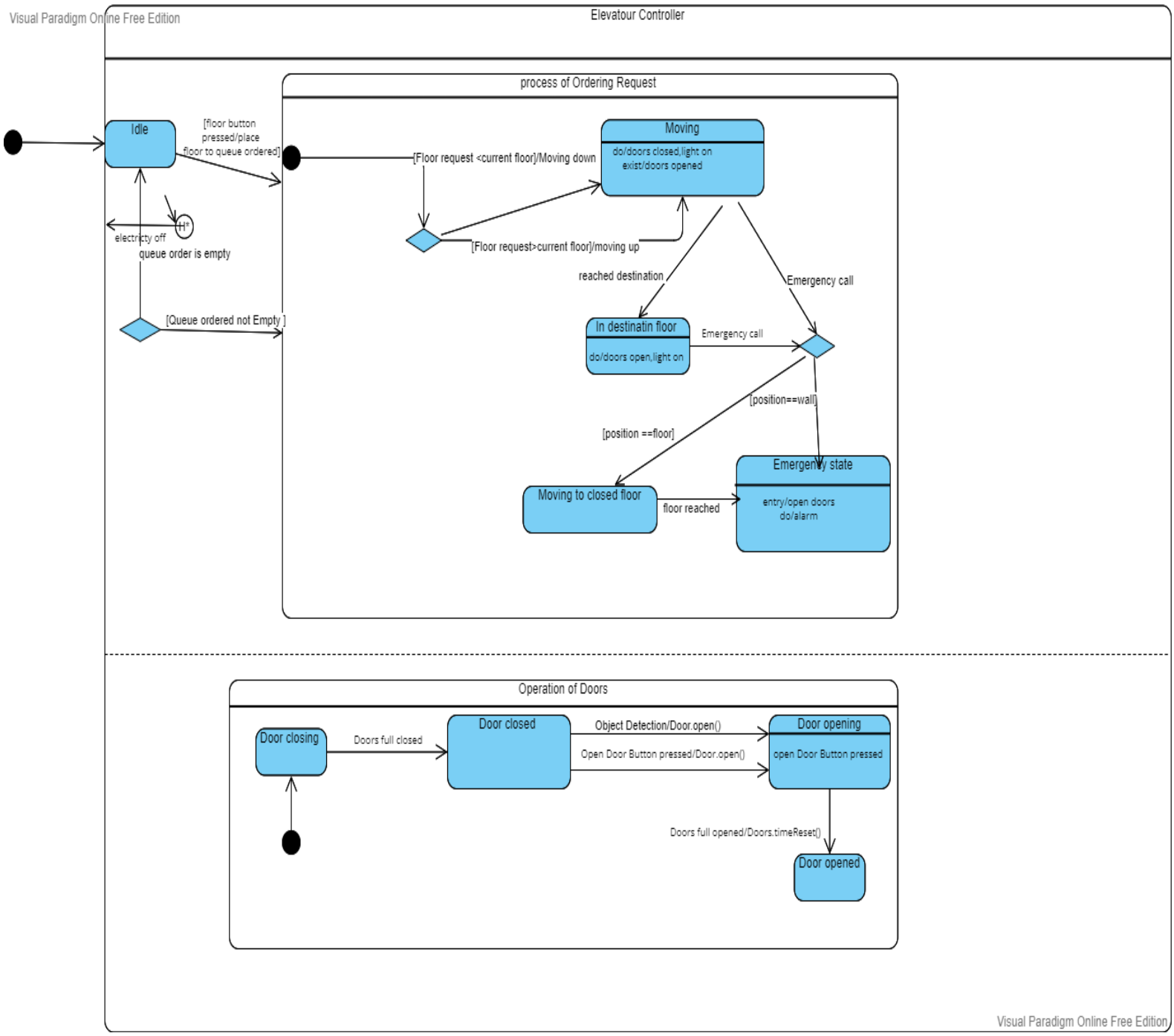
Opening and closing Door:

State/Event	Doors Full opened	Time out or close Door Button	Doors Full Closed	Floor Button pressed	Open Door Button pressed	Object Detection
Door Opening	Door Opened/Door.timeReset()	-	-	-	-	-
Door Opened	-	Door Closing/PressedDoor.close()	-	-	-	-
Door Closing	-	-	Door Closed	Door Opening	Door opening /Door.Open()	Door Opening/Door.Open()
Door Closed	-	-	-	-	-	-

Car request:

State/event	Car Button Pressed	car position==Floor position
Car Button Idle	Car Button pressed/car Button.pressed(targetpos)	-
Car Button pressed	-	Car Button Idle/ERS.removeRequest(pos); Car.TurnOnlight

State System Machine Diagrams:



System Architecture:

Model-View-Controller (MVC)

What is MVC ?

MVC is a software architectural pattern that is being used for solving problems about the architectural of application. It divide the application into three parts view, controller and model.

Type script:

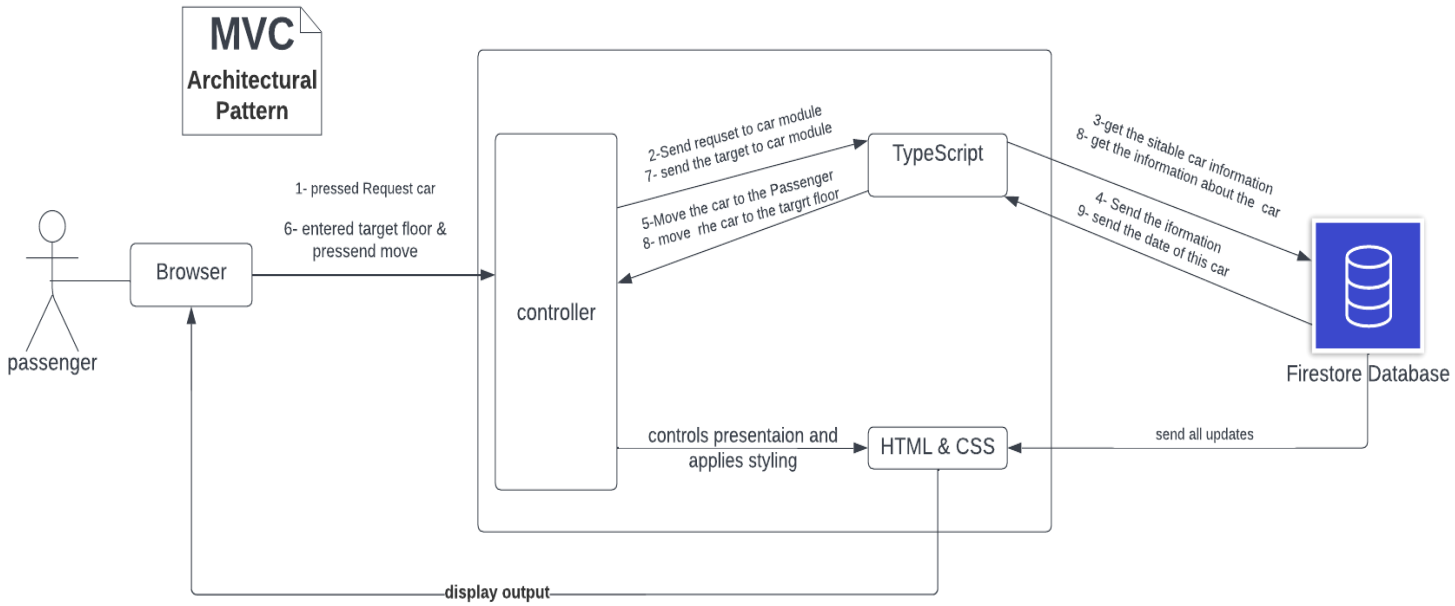
The typescript component contains all the data-related logic that the user works with. It manage the data and rule of the application.

Html & CSS:

The view component is the UI of the application that responsible for represents the information in such readable way for the user like tables and diagrams.

Controller:

The controller components is the interface between View and Model that handle requests. It accept the request from user then send it for either View or Model .It has business-related logic.



Hierarchical/ Multilevel Control Architectural Pattern