Group 22

Author: Yuxin Miao

Software Specifications

Elevator Control System

Table of Contents

[System Architecture 2](#_Toc44615228)

[Software Specifications 3](#_Toc44615229)

[Core functions introduction 3](#_Toc44615230)

[Class Controller\_Core() 3](#_Toc44615231)

[Class Elevator () 4](#_Toc44615232)

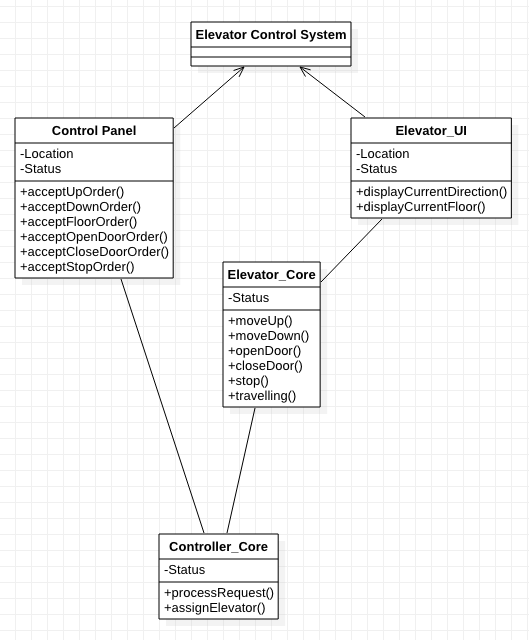
[Implementation Details 7](#_Toc44615233)

[S1 UI Implementation 7](#_Toc44615234)

[S2 Use Case Implementation 7](#_Toc44615235)

# System Architecture

The system architecture is shown below:



# Software Specifications

## Core functions introduction

This section lists and explains a few core functions, structures, and algorithms in the implementation of the elevator that may be referred in the later description. The names of functions may differ in real implementation, but they should perform the same algorithms.

Note that there are two algorithms in red below; they are the most critical and yet scalable functions of the system design. With these two functions, the system can be scaled to unlimited number of elevators in work and unlimited number of floors to serve.

### Class Controller\_Core()

Contorller\_Core(CC) is located between UI and Elevator, acting as a commander of the system. The CC is the processor that allocate and assign incoming request and operations to suitable elevators. The following properties and methods are the supporting key factors of this class.

More importantly, this controller\_core is scalable to more than 2 elevators using the same logic of core functions with a few more conditions.

#### Properties

* Elevators: This indicates the elevator needed for this system. In the original implementation, there are 2 of them.

#### Methods

* Goto function: This is the function that deals with the request coming from inside of the elevator with no direction but only target floor.
  + If the target floor is above the elevator’s current location, add this request to the UpQ (See below in “Elevator” Class).
  + If the target floor is below the elevator’s current location, add this request to the DnQ.
* Allocation algorithm (callElevator function): This is the function of allocating requests from floors’ control panel with both direction and their locating floor.

First, we have to define new terms called forward requests and backward requests.

A forward request is a request of an elevator if:

1. It has the same direction with that elevator.
2. It is located at a floor which the elevator can reach without changing direction.

A backward request is a request of an elevator if:

1. It has the same direction with that elevator.
2. But the elevator needs to switch direction to reach its floor.

Or

1. It has the same direction with that elevator.
2. But the elevator does not need to switch direction to reach its floor.

For example, if there is an elevator locating at 2nd floor with a direction of going up. Then a request of going up from 3rd floor would be its forward request. A request of going up from 1st floor or a request of going down from 3rd floor would be its backward request.

Supposing there are N elevators available, which in original implementation is 2, when a request from floor F with a direction of D comes, the following is how the algorithm works.

* Traverse through N elevators to see if the request is one of theirs forward request. If so, add this request to the first satisfying elevator’s waiting queue which has the same direction with the request, i.e. add a Up request from 2nd floor to the satisfying elevator’s UpQ.
* If it is not the forward request of any elevator, traverse through N elevators to see if any of them is stopped. If there are any stopped ones, add this request to the first satisfying elevator’s waiting queue which has the same direction with the request.
* If none of the above situation is met, add this request randomly with same probability to one of the elevator’s waiting queue which has the same direction with the request.

**If an extension of more elevators is needed to the system, modifying the N from 2 to whatever number is required should meets the requirement.**

### Class Elevator ()

This is the class of elevator, which refers to the same class in system architecture. To avoid ambiguity, here is a more detailed description of some core components of this class.

#### Properties

* UpQueue(UpQ): This is a queue that stores all the requests of going up in a certain floor of a certain elevator.
* DownQueue(DnQ): This is a queue that stores all the requests of going down in a certain floor of a certain elevator.
* Physical properties: This is the abstract of physical features of an elevator needed to do basic functions. This may include current floor, current direction, etc. Please implement these features as required by your own need.

#### Methods

* Physical movement: This may generate movement functions such as up, down, stop, open, close, etc. These functions modify the properties of the elevators.
* The close and open functions update the status of corresponding buttons on the panels when the function is performed.
* Stop: This is the function call by the elevator\_core (below) to make sure the elevator enters the status of stop and update the corresponding panels.
* emergencyCall and emergencyRecovery: These are the pair of functions to be activated when the stop button is press from inside the elevator. The UpQ and DnQ of the elevator is cleared and all the buttons inside the elevator will be disabled or enabled depending on the current state of the stop button. When both elevators are in the state of emergency, the outside panel will also be disabled until on of the elevator exit from the emergency state.
* UpdatePanel: This is the function that updates the outside common panel of buttons and display info automatically, especially after the movements or actions are done.
* **Travelling algorithm (inPosition function):** This is the core function of an elevator in terms of how it moves and clears its requests. The algorithm can be viewed as follow diagram. This algorithm is scalable to no matter how many floors there are.

**Whenever and wherever an elevator reaches a certain floor with a close status, it performs the travelling algorithm before doing anything. This means that, after an elevator close its door at a certain floor, even if no movement is performed, the following algorithm shall be called again.**

When an elevator is at a certain floor with a certain direction it towards, there are always 6 different possible request it needs to process, which should have already been added to their UpQ and DnQ by Controller\_Core through Allocation() mentioned above.

1. An Up request from current floor.
2. An Up request from higher floor.
3. An Up request from lower floor.
4. A Down request from higher floor.
5. A Down request from current floor.
6. A Down request from lower floor.

The elevator checks the request by following order:

* If the elevator is either stop or going up, it checks the possible requests in the order of 1 2 3 4 5 6.
* If the elevator is going down, it checks the possible request in the order of 4 5 6 1 2 3.

Whenever the requests exist during the check, the elevator exits the searching process and detail with the first found request by following logic.

* If the request is from the current floor, the elevator stops (changes the direction property) and opens the door.
* If the request is from above, the elevator goes up.
* If the request is from below, the elevator goes down.

To scale this system to more than 3 floors, no modification is needed to this algorithm as it does not depend on the exact floor of the elevator but rather the relative position between the elevator and its next request to be served.

#### Stateflow Elevator\_Core

In the implementation, this elevator\_core acts as a role of representing real physical actions of an elevator. The graph consists of 7 basic states and can be extended. It takes input signal from controller\_core and calls the functions defined in Elevator class.

1. Sleep

This is a state will the elevator starts with and rests when all current requests are fulfilled.

1. Floor 1

This indicates the elevator locating at floor 1 and door closed.

1. Floor 2

This indicates the elevator locating at floor 2 and door closed.

1. Floor 3

This indicates the elevator locating at floor 3 and door closed.

1. Open

This indicates that the elevator stops at a certain floor and opens the door

1. Close

This indicates that the elevator closes the door and goes back to state of current floor when exit this state

For a clearer description of how this stateflow functions, below are some essential transition between states. X indicates any possible state.

* Sleep -> Floor

This transition is triggered by the controller\_core to wake up the target elevator when a new request is added to the waiting queue while the elevator is stopped.

* Floor -> Sleep

This transition happens when all the requests in both waiting of an elevator have been cleared.

* Floor -> Floor

This transition only happens between adjacent floors and is triggered by the Travelling algorithm of the elevator when an Up or Down movement is needed to meet the request.

* Floor -> Open

This transition happens when there is a request from the same floor of where the elevator is located.

* Sleep -> Open

This transition happens when the open button is pressed when the elevator is stopped, i.e. the stateflow is in the state of Sleep.

* Open -> Close

This happens automatically after a preset length of time or can be halted when the Open button is pressed, and the OPEN signal is sent. This is also the only outgoing transition from the state of Open.

* Open -> Open

This happens when the open button is pressed when the door is already open, which reset the timer of door opening.

* Close -> Floor

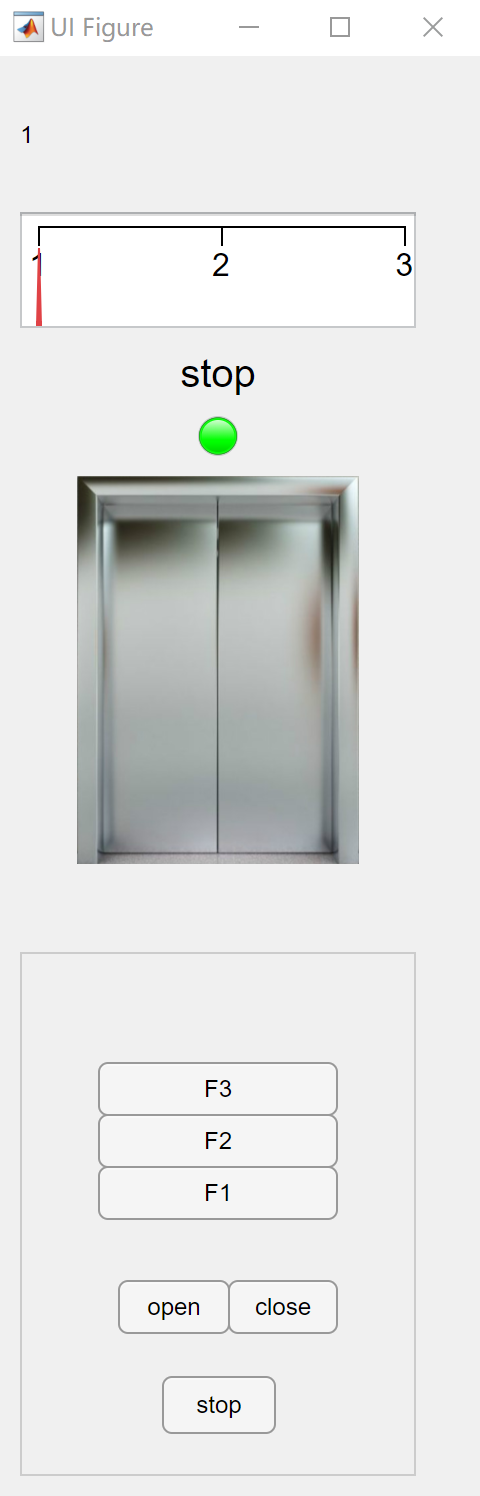
This transition happens automatically after the functions in the close state return and will go back to the state where the elevator is right now located.

## Implementation Details

This part shows how the code implements use cases and requirement in practice. As the elevator system is a system of high integration, it would be way to abundant to use separate examples to show all the use cases and requirement entries; thus I will use a widely covering example with its sequence diagram to show how the use cases are implemented.

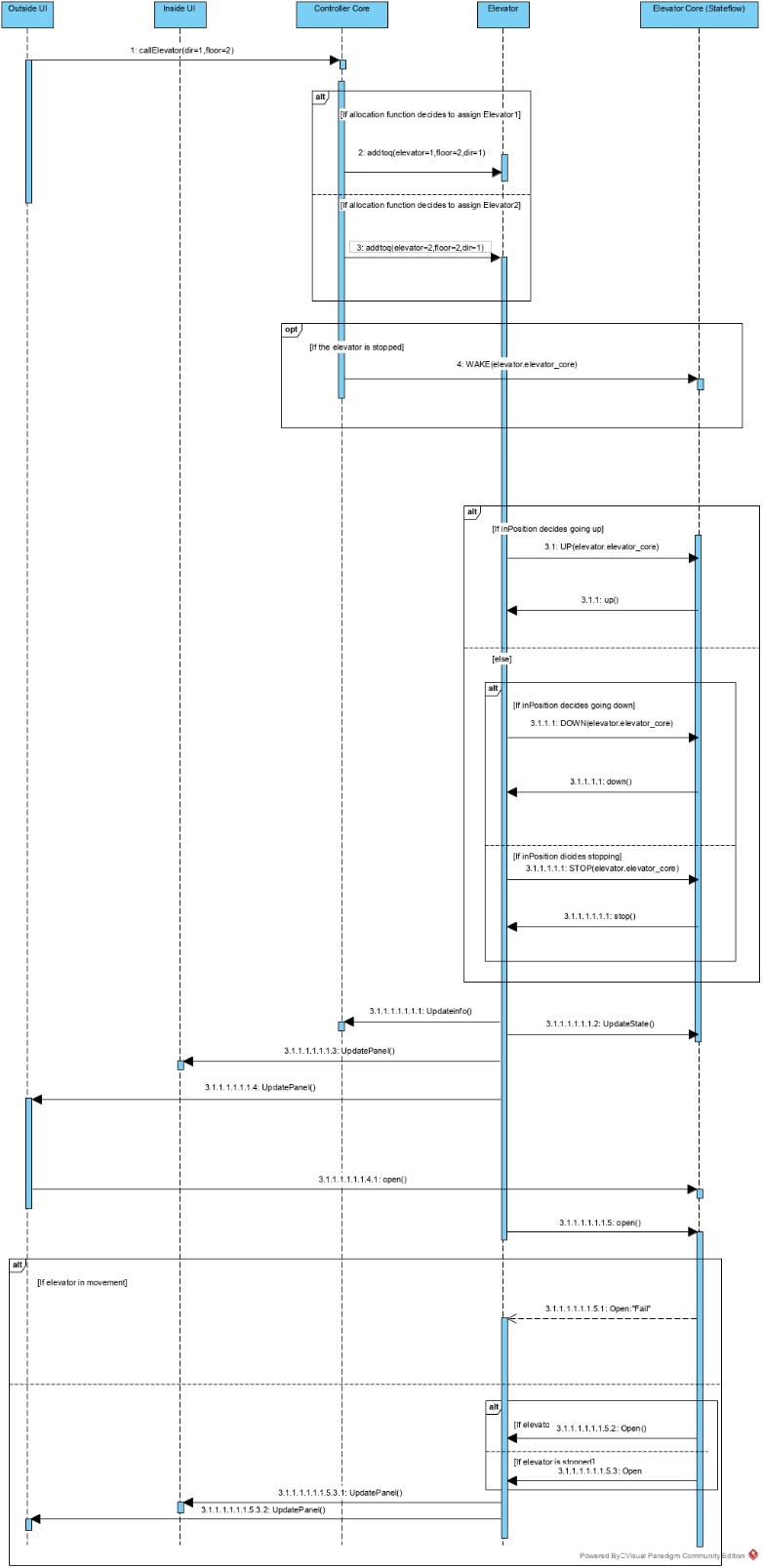
### S1 UI Implementation

* S1.1.1: First as required, the control panel inside the elevator has 6 buttons (R1.1.2) while the panel outside has two (R1.1.1).
* S1.1.2: Every button will be lit when its pressed and will dim when the floor is reach or the action has been done. (R1.2.1)
* S1.1.3: Both panels have either a meter or a number to indicate the current location of the elevator. On the inside panel, a light switch between green to red and a door opening and closing indicate the status of door open and door closed, respectively. On the outside panel, the red line shown will turn in the up arrow and down arrow depending on the corresponding elevator’s direction, with a door opening and closing indicating the status of door open and door closed.(R2.1.1)

### S2 Use Case Implementation

Here is a big sequence diagram of how the system deals with requests from inside and outside panel.



* S2.1: When a request of UP comes from 2nd floor
  + S2.1.1: The controller core here assign the request to suitable elevator via Allocation() (R4.1.2)
  + S2.1.2: All requests must go through the Allocation() (R4.1.1)
  + S2.1.3: All requests to elevator can only be distributed through Allocation() (R3.2.1)
  + S2.1.4: The conditions the allocations needed to make decision come directly from retrieving the target elevator’s data (R3.1.1)
* S2.2 After the request has been assigned
  + S2.2.1: The elevator moves according to the inPosition() function by sending signal to elevator core
  + S2.2.2: The elevator core calls the function to control motor written in Elevator class (R3.3.1)
  + S2.2.3: As implemented in the Stateflow mentioned above, the door will open only when the elevator is stopped because inPosition() only gives out the STOP signal when the elevator reaches a certain floor and before any further movement. This prevents the elevator to open in the middle of any movement (R3.3.2)
  + S2.2.4: As implemented in the Stateflow, there is no such a state representing floors over 3 of beneath 1, thus the elevator will only operates between 1st and 3rd floor. Also as implemented in inPosition, as no request from over 3 of beneath 1 comes, the elevator will not continue to move up at 3rd or move down at 1st. (R3.3.3 R3.3.4)
  + S2.2.5: After a request is fulfilled, the elevator will automatically update all the panels both inside and outside the elevator(R3.1.2)
* S2.3 When an open button is pressed manually, as shown in the sequence and implemented in the stateflow, the signal will be directly passed to the elevator core without going into controller core. (R3.2.1)
* S2.4 When an stop button is pressed, the stop() function mentioned above will be called to stop the elevator until it is pressed again (R1.1.2)