L3G2 Final Report

By:

Aleksandar Veselinovic - 101118532

Dominique Giguere Samson - 100912460

Harrison Lee - 101111588

Ryan Nguyen - 101127319

Tyler Leung - 101108511

For:

Dr. Franks

TA:

Ismael Al-Shiab

Date:

April 12th, 2022

Contents

Breakdown of Responsibilities

The following table is an individual breakdown of each person's tasks throughout the project. It is important to note that most of the work was completed collaboratively with at least 3 group members present (design and coding). The collaborative work went smoothly with very few conflicts.

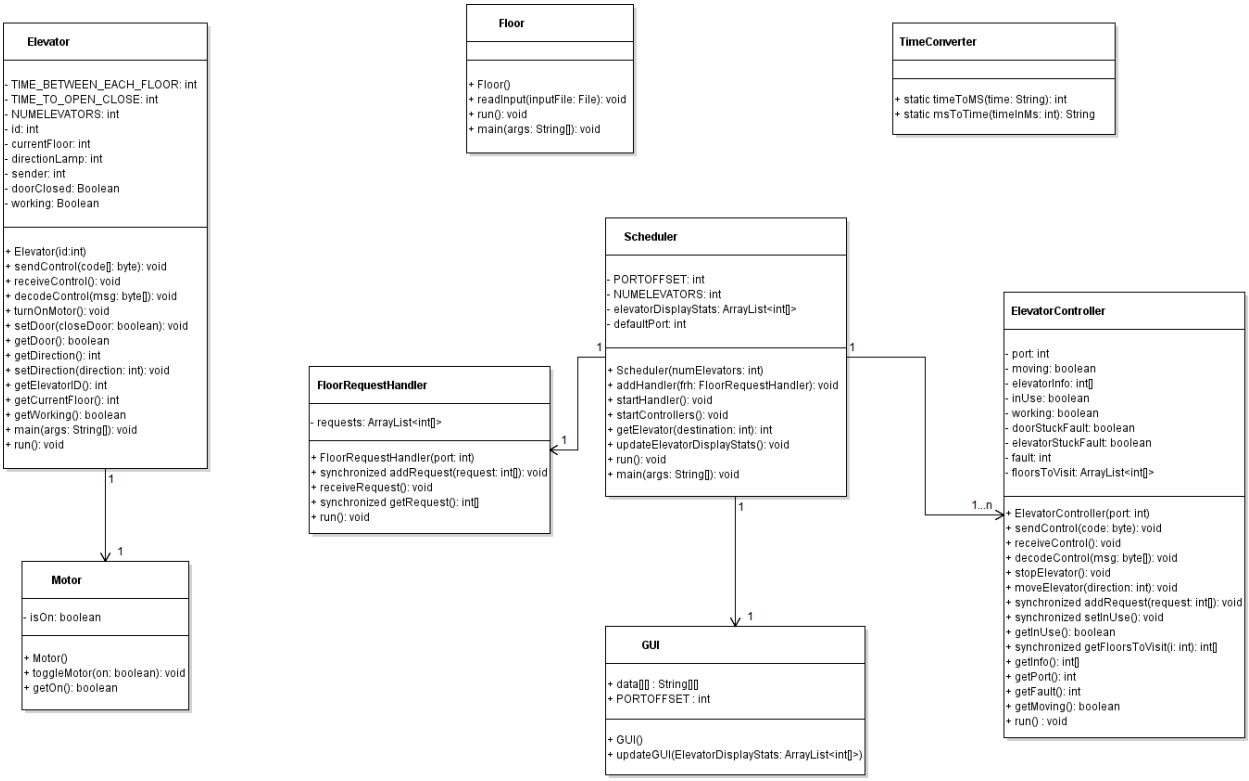
Iteration 1	
Aleksandar	Created scheduler class with synchronized get and put methods
Dominique	Created sequence diagram Created class diagram
Harrison	ABSENT
Ryan	Created elevator class Created test files
Tyler	Created floor class Created file reading methods
Iteration 2	
Aleksandar	Redesigned scheduler class to control elevator
Dominique	Updated sequence diagram Updated class diagram
Harrison	Created state machine diagrams Updated timing of elevator
Ryan	Updated elevator class with direction, the floor at, and moving components Create motor class Updated test cases to reflect the current iteration
Tyler	Created documentation for the iteration Created TimeConverter class used to display timestamps
Iteration 3	
Aleksandar	Created floor request handler class for scheduler subsystem Added UDP communication in scheduler subsystem Updated scheduler code to schedule multiple elevators
Dominique	Created the communication protocol between subsystems Created message decoding methods Created elevator controller class Updated scheduler code to control multiple elevators
Harrison	Updated elevator for UDP messaging Updated floor for UDP messaging
Ryan	Updated documentation Updated test cases to reflect the current iteration
Tyler	Updated class diagram Updated state machine Updated sequence diagram

Iteration 4	
Aleksandar	Improved the scheduling algorithm in the scheduler Improved synchronization in floor request handler Helped add fault handling
Dominique	Improved scheduler code to better control multiple elevators Improved elevator controller code to better control multiple elevators
Harrison	Created timing diagrams of the elevator errors Assisted with creating test cases for the fault handlers
Ryan	Created fault handlers for the door being stuck and the elevator not arriving at the proper floor Updated test cases to reflect the current iteration
Tyler	Designed fault handling Updated code for fault checking
Iteration 5	
Aleksandar	Updated fault handling from random occurrences to manually causing it to fault with the help of the input file
Dominique	Designed GUI framework Created GUI code Created time stamps
Harrison	Assisted in the design of GUI Assisted in improving GUI UX Documentation/reorganization of the program Measured performance of system Updated state diagram
Ryan	Updated fault handling from random occurrences to manually causing it to fault with the help of the input file Updated test cases to reflect the current iteration
Tyler	Designed GUI UX Improved GUI UX Updated class diagram

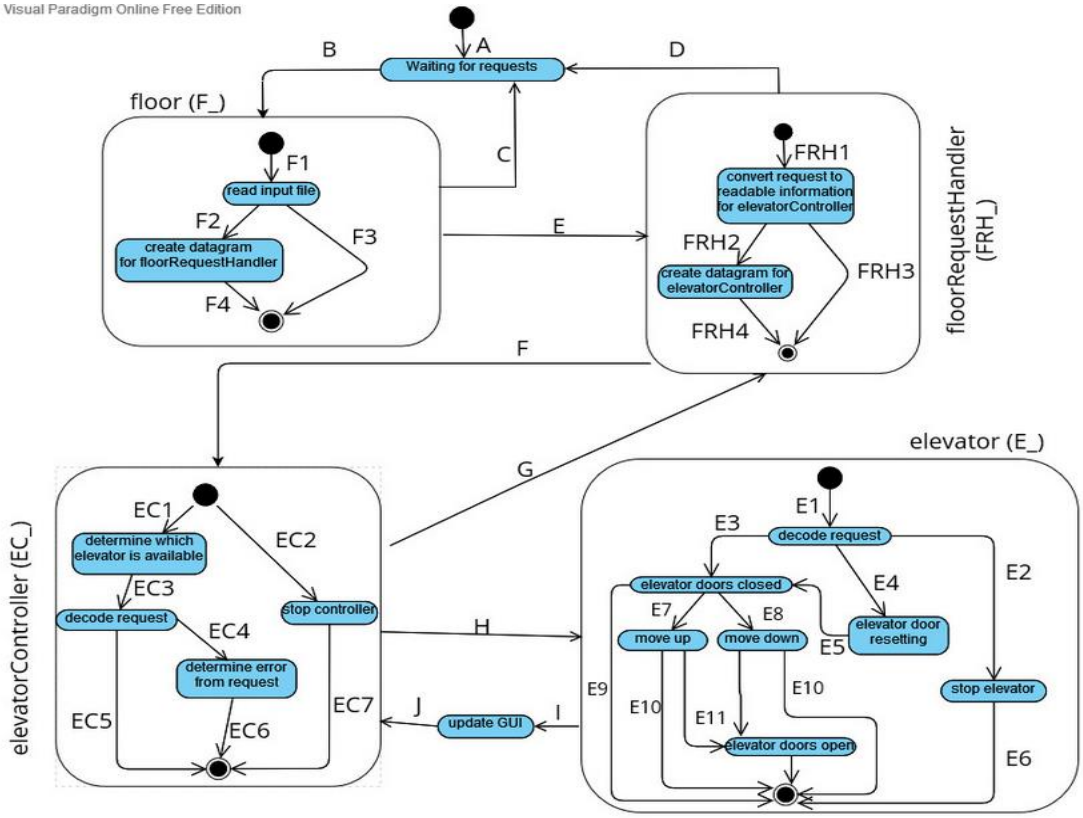
Diagrams

High resolution versions of all diagrams can be found in the “Diagrams” folder of the project folder.

Class Diagram:



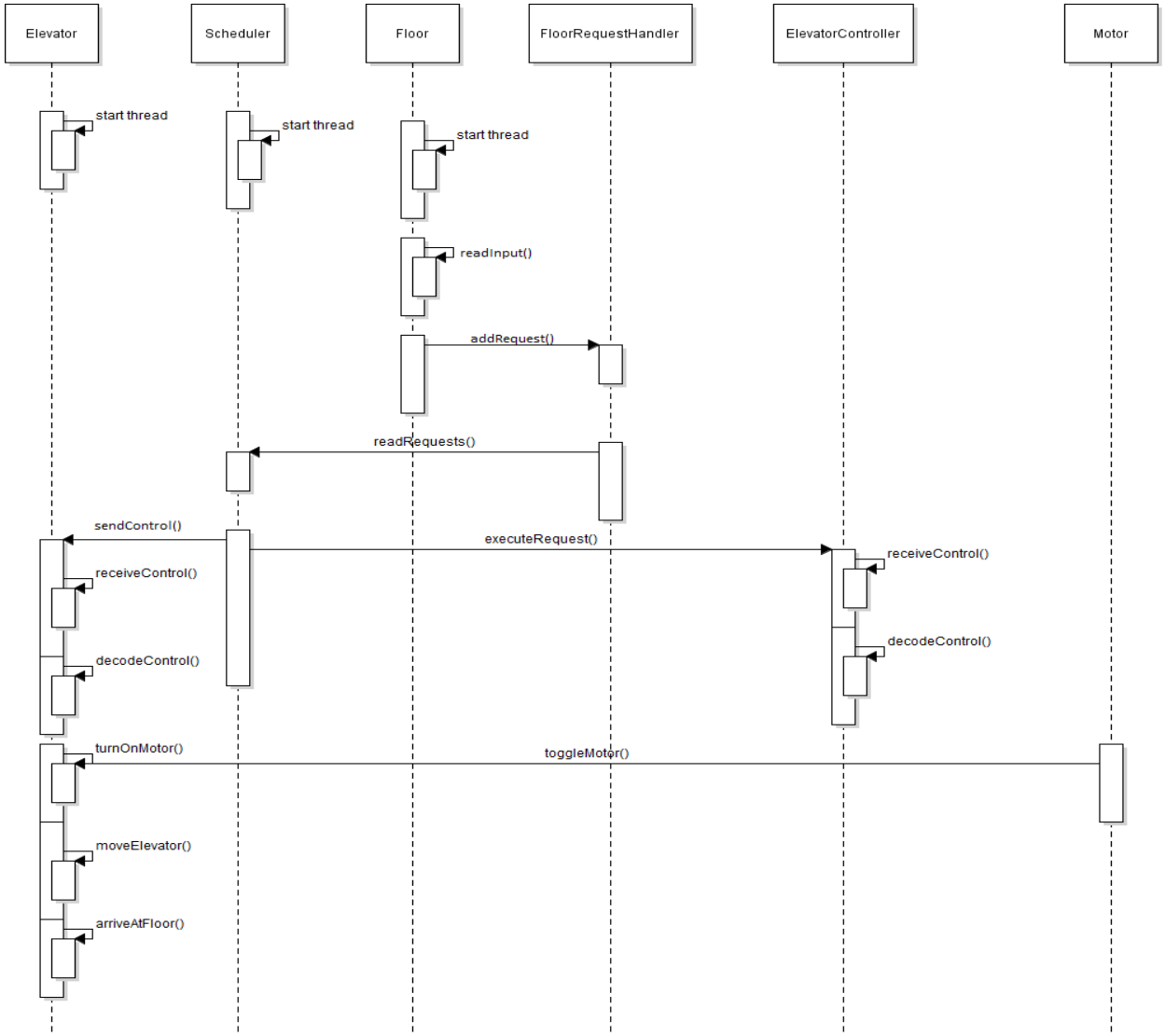
State Diagram:



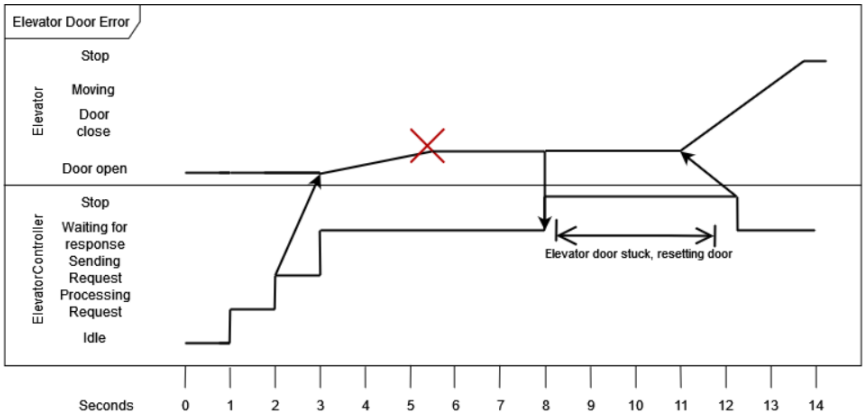
State Diagram Legend:

A – initialize scheduler, floor, elevator B – input file has been inserted C – input file was missing/incorrect format etc. thusignored, returning to idle state D – input file has incorrect information, thus ignored,returning to idle state E – sending datagram to floorRequestHandler F – sending datagram to elevatorController G – telling floorRequestHandler to shut down thiselevatorController due to hard fault H – sending instructions to corresponding elevator I – sending information back to update GUI J – telling elevatorController GUI is updated and now askingfor next instruction	EC1 – datagram from floorRequestHandler received EC2 – elevator has a hard fault/tell floorRequestHandler and stop controller EC3 – figure out which case the request is EC4 – [if error injection detected]/determine hard or soft fault EC5 – [if error injection not detected]/send instruction to elevator EC6 – send error to elevator EC7 – tell floorRequestHandler hard fault, shutting down
F1 – sending input file to be read F2 – [if input file passes error check]/create datagram to send F3 – [if input file fails error check]/ignore and return to idle state F4 – creating datagram of converted text	E1 – receive instruction from elevatorController E2 – hard fault instruction, shut down E3 – move elevator E4 – soft fault, doors stuck E5 – attempt to move elevator again E6 – telling GUI there is an error with elevator E7 – elevator is below floor/motor moves elevator up E8 – elevator is above floor/motor moves elevator down E9 – tell GUI doors are resetting E10 – tell GUI where elevator is and ask elevatorControllerif correct floor E11 – elevator has reached destination
FRH1 – receiving datagram from floor/elevatorController FRH2 – [if request is in proper format]/convert request FRH3 – [if request is not it proper format]/dismiss and return to idle state FRH4 – starting elevatorController to send datagram to	

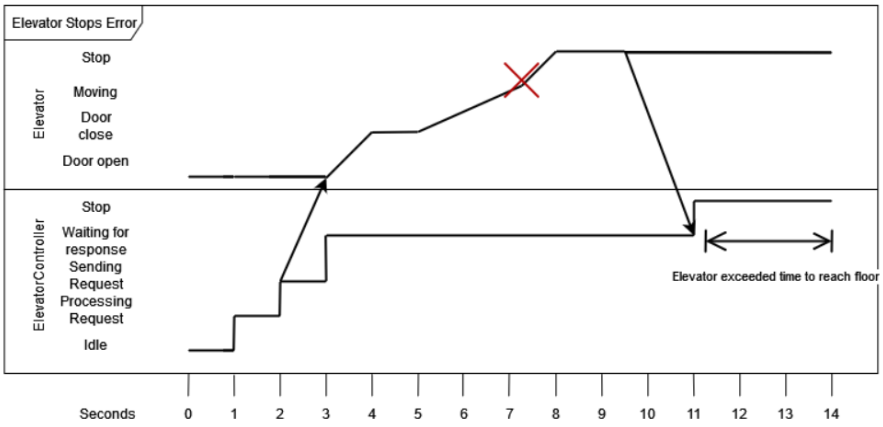
Sequence Diagram:



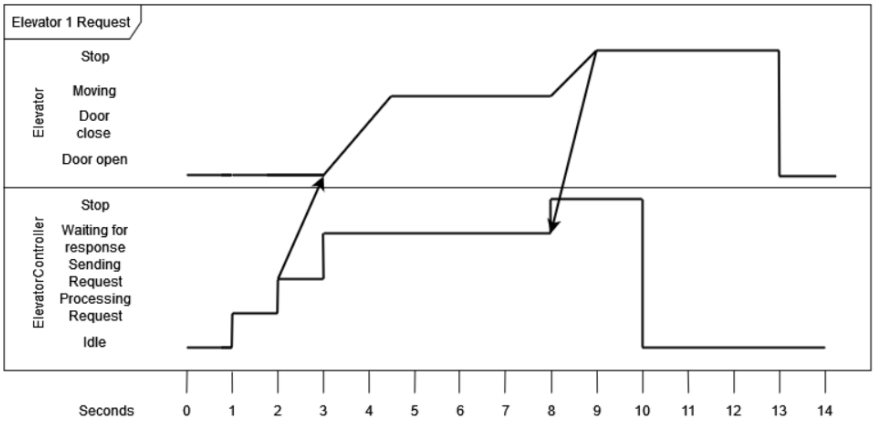
Timing Diagram 1: Elevator door gets stuck open and resets



Timing Diagram 2: Elevator stops moving and gets stuck, thus shutting down the elevator



Timing Diagram 3: Elevator processing 1 request with no errors



Set-Up and Test Instructions

To start the program, run the subsystems in the following order:

1. Elevator.java

2. Scheduler.java

3. Floor.java

Once Scheduler.java is running, you should see the GUI.

Each test class is runnable as a JUnit Test program to run our test cases.

Measurements

AVG Time of Elevator to travel between floors (from doors opening to doors opening once reaching next floor): (6.2 + 6.5 + 9.7 + 8.5 + 10.6 + 7.7 + 10.7 + 12.7 + 10.2 + 8.4 + 11 + 13.2 + 8.2)/13 ≈ 9.5 seconds (s)

Velocity (V): Distance = 4 meters (m), Time ≈ 14.5 - 9.5 = 5s

V = 4/5 = 0.8 m/s

Acceleration (A): Initial Velocity = 0m/s, Final Velocity = 23.76 m/s, Time ≈ 19.8s

A = 23.76/19.8 = 1.2m/s ^{2}

Standard Deviation (*S*) = $\sqrt{\frac{0.5}{2-1}} = 0.707$

Schedulability Analysis

The program on the first run takes 6 minutes and 6 seconds to completely execute. After doing multiple runs, the execution time averages out to around 5 minutes 44 seconds. This was done with the average travel time between each floor set to 4.5 seconds, which is reduced from our calculated time.

Reflections

Our request handling scheduler algorithm has room for improvement, currently, the scheduler will dispatch requests to a free elevator, if none are free then the elevator is picked based on the direction it is traveling, and if it has not passed the request's floor. If all four elevators fail to satisfy these conditions, the request is dispatched to the default elevator. The first elevator begins as the default elevator, and when it receives a request through being the default elevator, the default elevator is incremented to the second elevator. This method is fairly good at load balancing, as most of the elevators will be occupied but the load balancing could be improved by processing the requests out of order, and only dispatching the requests once the conditions are met. There could be scenarios where a request could repeatedly fail to satisfy the dispatching conditions, and the passenger will end up waiting. We can combat this by adding a priority scheme to the request handling. For example, after a request has been skipped 3 times for failing to meet dispatch conditions, its priority will increase, forcing it to be dispatched when the next elevator is ready.

The elevator controller's scheduler can also be improved. The elevator controllers process their requests sequentially, by going to a passenger's pick up and drop off floors, then processing the next passenger's request. To improve the performance of the system the elevator controller could stop at all floors of interest while travelling to its destination. For example, if the first passenger arrived on floor 1 with destination floor 7, while a second passenger is on floor 3 with destination floor 5. The elevator will travel to floor 1 to pick up the first passenger, then head to floor 3 to pick up the second passenger, then head to floor 5 to drop off the second passenger and finish on floor 7. Instead of picking up the first passenger and traveling from the 1st floor to the 7th floor before picking up the second passenger.

The GUI (Graphical User Interface) that was designed was one of the more enjoyable features of this project. It was designed well enough that it displayed all the relevant information as well as leaving room for many improvements. Currently, it is a simple window that displays each state as words. This could be improved by displaying the design of an elevator shaft which displays the direction it will be traveling as well as the floor location that it will be traveling to. It is also not as dynamic as it could be. The initial table that displays all the data is set to four elevators and twenty-two floors. This could be improved by allowing for the GUI to determine how many elevators there are as well as how many floors there are dynamically. These improvements would allow for an ideal GUI for an elevator system displaying the information in a visual pleasing and accurate format.

Testing the elevator system was a component that our team had difficulty doing and was one of the least enjoyed parts. Since it was our first time working with UDP, it was an inconvenience to test each method, as most of our functions relied on other methods to finish. For example, the moveElevator function in our elevator controller class interacts with our elevator class sending and receiving packets. If we create an elevator object in the test class and have it run continuously to receive packets, the system will not work properly as many packets were sent back and forth. Also, some of our functions did not have return variables, figuring out how to test those methods was challenging as we did not have a way to get the changes conducted by the method. In some cases, we had getter methods for some field variables that we could use in our test cases to check if the variable was used and changed within the method we want to test. To sum it up, some methods were not testable as the setup was not practical and simply did not use that method to validate the functionality of it.