Real-Time Concurrent Systems

ABSTRACT

The system will consist of an elevator controller (the Scheduler), a simulator for the elevator cars (which includes, the lights, buttons, doors, and motors) and a simulator for the floors (which includes, buttons, lights and last, but not least, people who are too lazy to take the stairs).

Group 7
Ismail, Zakaria
Liu, Patrick
Nguyen, Trong
Elmokdad, Hussein
Ngo Huu Gia, Bao

SYSC3303A
WINTER 2021

Table of Contents

1. Team Members

Group 7
- Ismail, Zakaria
- Liu, Patrick
- Nguyen, Trong
- Elmokdad, Hussein
- Ngo Huu Gia, Bao (Bobby)

2. Breakdown of Responsibilities

The distribution of work was coordinated using *Atlassian Jira Software* which can be viewed using this link: ECSS and for quick reference see Appendix A. Everyone took part in major code reviews and approved pull requests. Here below is a general summary:

Iteration 5 @version 5.0, 04/10/23	
Trong	• Implemented GUI for a whole system/console logging • Fixing timing bug of the Parser • Design script to auto-generate Elevator Events and fault • Timing diagram, Javadocs • Documentation (report writing, measurements, reflection)
Bobby	• Improved algorithm for Scheduler • UML class diagram, documentation (work distribution, README) • Resolved multiple elevator concurrency direction bug • Console UI integration with static model of domain
Hussein	• Improved Floor Subsystem • Fixing bugs of the floor subsystem • Addressed multiple floor/GUI component bugs in the system • Implemented data-transfer object for floor GUI
Patrick	• Updated the Parser functionality by adding new column representing the error • Updated Sequence diagram
Zak	• Improved, added unit tests, and fixed bug for Elevator subsystem • Handle packet lost simulation • Implement data-transfer objects for GUI • Added additional homing state • Added mode for disabled elevator state transition to timeout

Iteration 4 @version 4.0, 03/25/23	
Trong	• Implemented encode & decode interfaces for data transfer objects • Implemented floor request dispatching at a relative offset time • Documentation and Javadocs • Implement/improved distributed processes
Bobby	• Refactored Scheduler subsystem, implemented event-driven state machine • Developed optimal elevator job assignment algorithm • Unit testing of the elevator, scheduler, & floor subsystems • Documentation
Hussein	• Implement multiple floors and elevators each running on separate threads • Updated messaging interface to use UDP instead of RPC • Refactoring overall system communication
Patrick	• Refactored Scheduler subsystem, implemented event-driven state machine • Developed network communication interfaces for the Scheduler subsystem • Validated elevator car shaft traversal algorithm • Class Diagrams for subsystems • Added timestamp parser feature
Zak	• Refactored Elevator subsystem, designed • Implemented event-driven state machine • Developed elevator car shaft traversal algorithm • Added fault states, and associated elevator behaviours

Iteration 3 @version 3.0, 03/11/23	
Trong	• Implement Remote Procedure Calls using UDP • Implement distributed system processes • Documentation: Update class UML, sequence UML, Work Distribution Document, README • Javadocs/refactoring code smells
Bobby	• Improve elevator traversal to each floor sequentially • Create UI integration with Static Model of Domain (buttons, sensors…) • Preliminary elevator scanning algorithm
Hussein	• Implement multiple floors and elevators each running on separate threads • Integration of UI console output for better system view
Patrick	• Improved parser by sorting elevator request prior to sending to Floor
Zak	• Elevator subsystem refactoring & design • Initiate diagrams and generated system ideas

Iteration 2 @version 2.0, 02/27/23	
Trong	• Implement SchedulerState class, FloorState class, SchedulerStateTest, FloorStateTest class, and javadocs
Bobby	• Implement FloorState class, README, refactoring, documentation
Hussein	• UI improvements, refactoring
Patrick	• Update Elevator Location feature, UML, and State diagrams
Zak	• Proposing ideas and revisions

Iteration 1 @version 1.0, 02/04/23	
Trong	• Developed and implemented Elevator class • Documentation README, UML class & sequence, refactoring
Bobby	• Developed and implemented Scheduler class • Documentation README, UI logger
Hussein	• Developed and implemented Floor class, FloorTest, ElevatorTest
Patrick	• Developed and implemented Parser class, ElevatorRequest class, ParserTest, Exception class
Zak	• Integration of system and system testing • Overall revisions

3. Diagrams

3.1. UML Class diagram: Scheduler

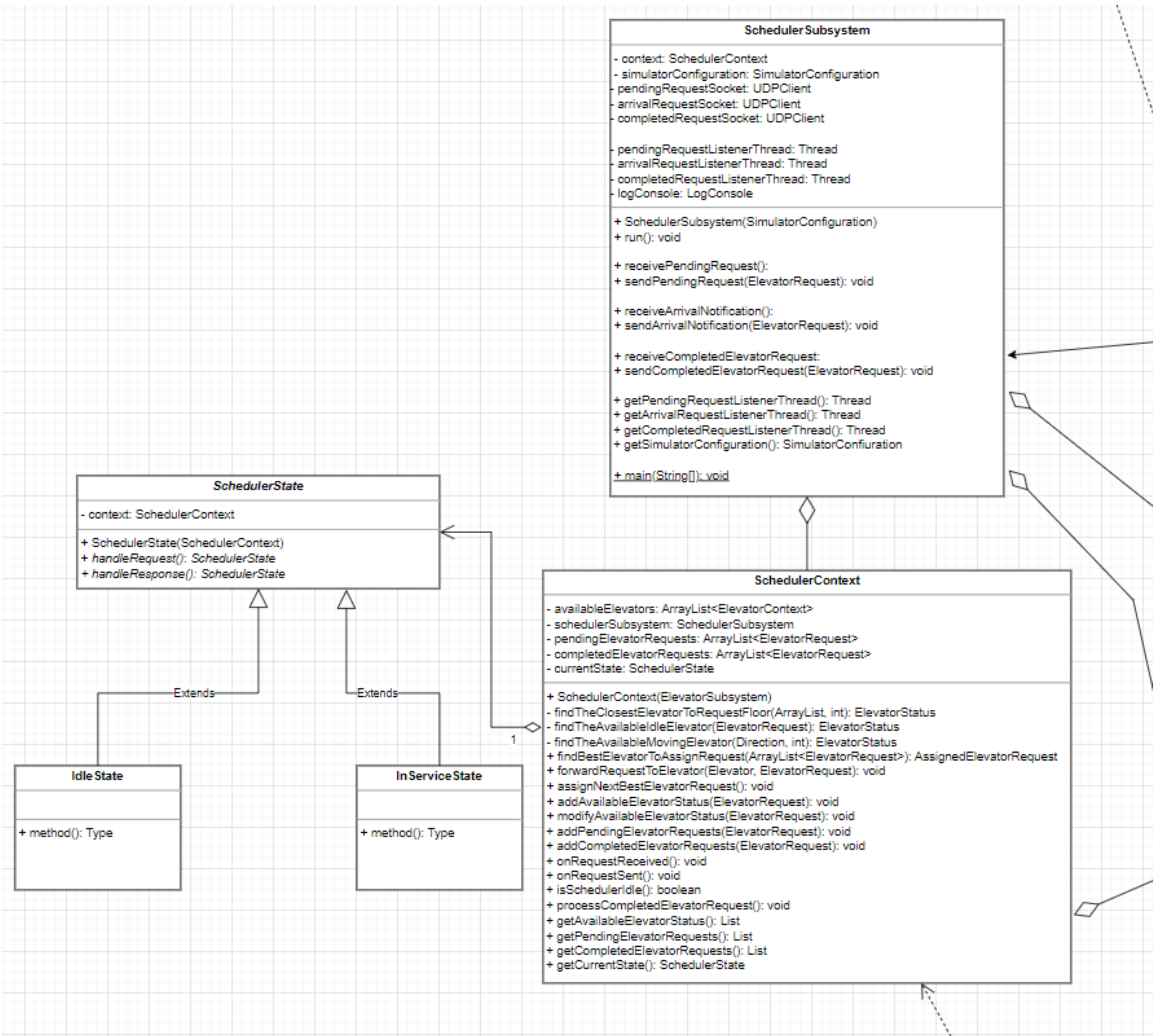


Figure 3.1. UML class diagram focusing on the Scheduler system.

3.2. UML Class diagram: Elevator

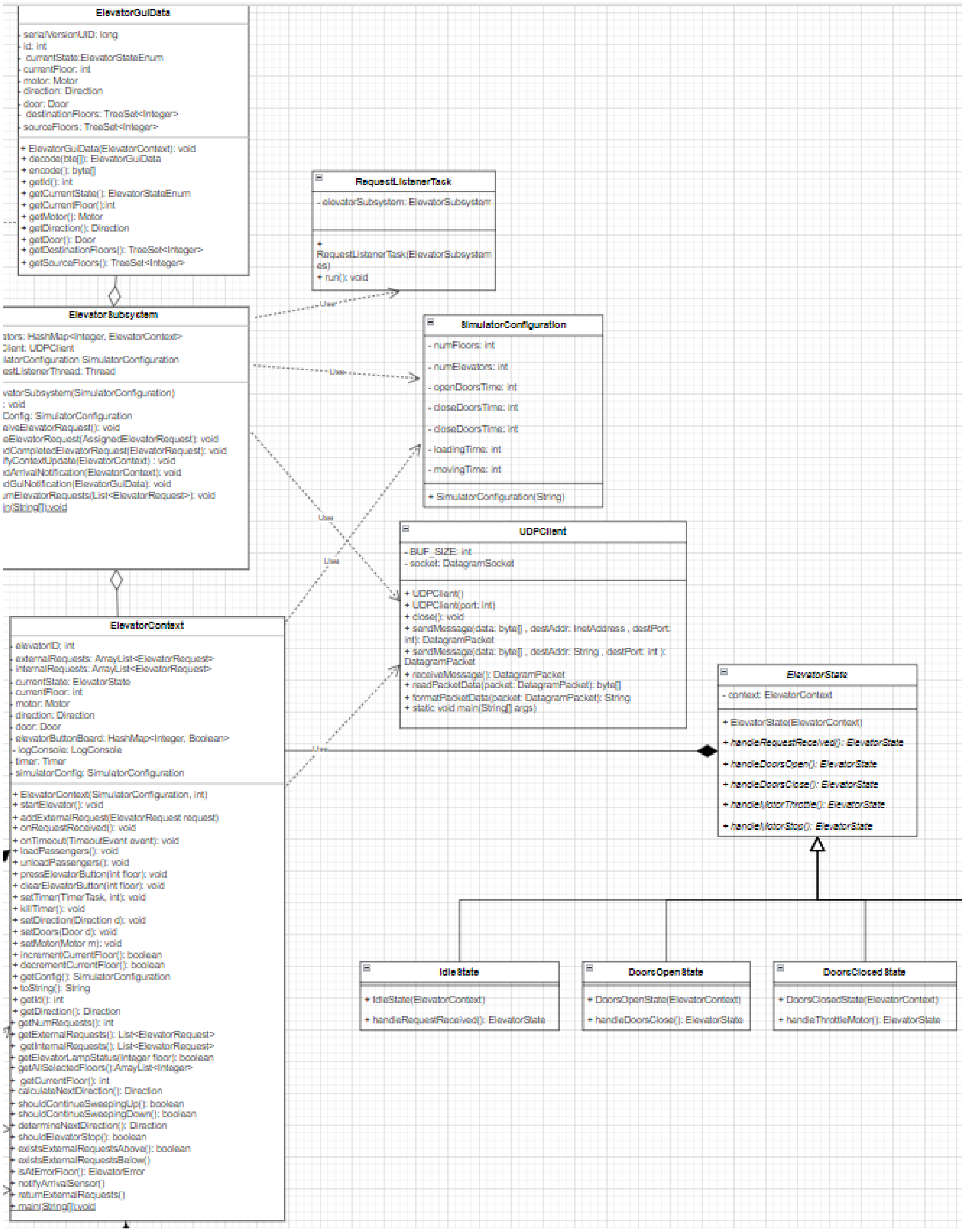


Figure 3.2. UML class diagram focusing on the Elevator system.

3.3. UML Class diagram: Floor

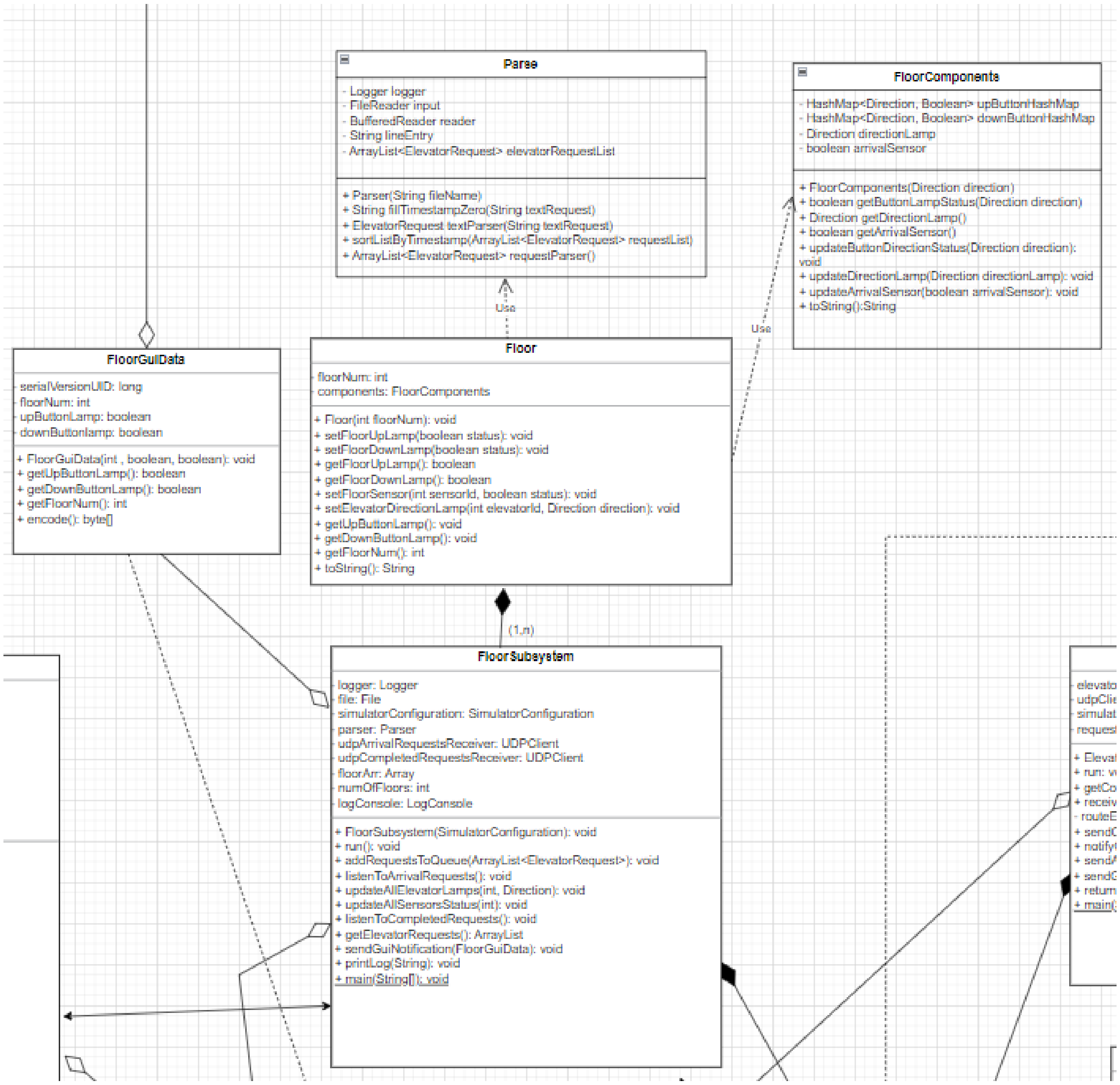


Figure 3.3. UML class diagram focusing on the Floor system.

3.4. UML Class diagram: Entire System

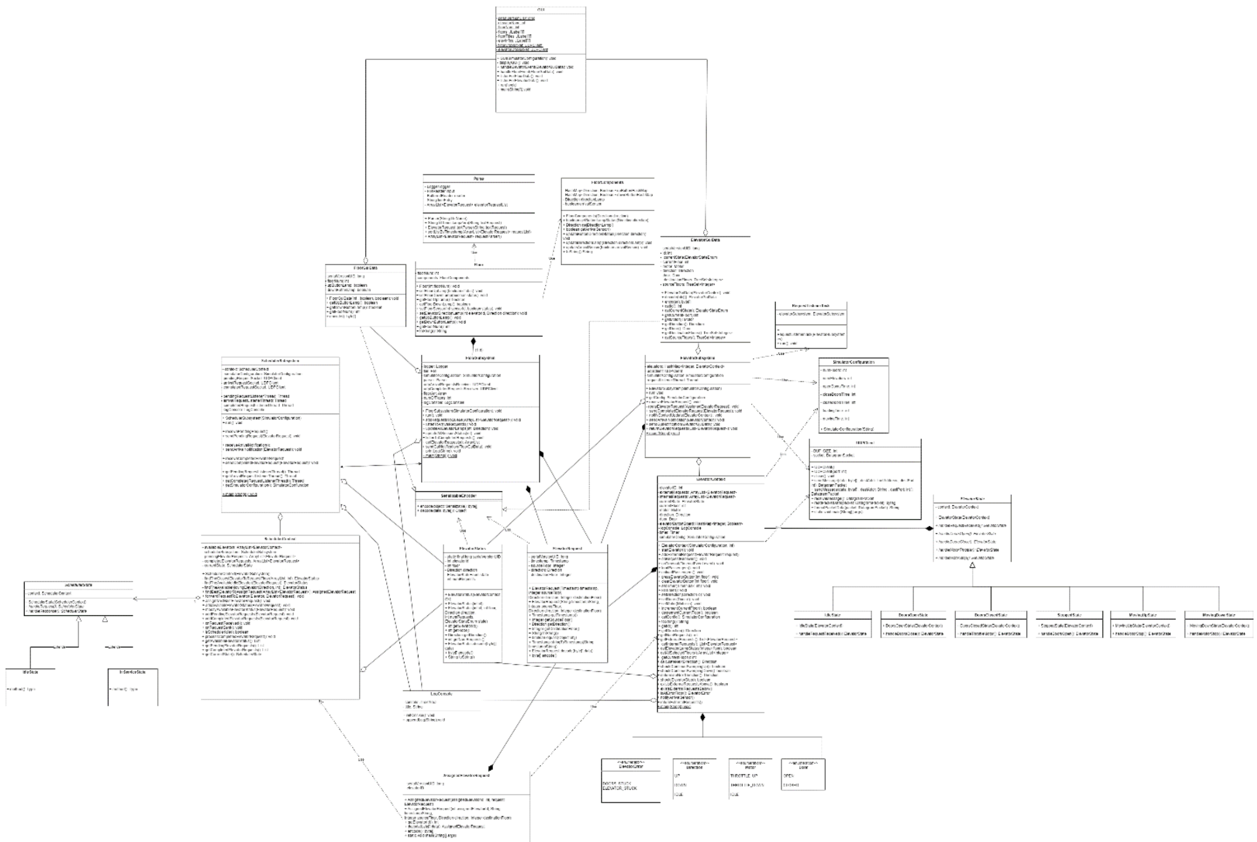


Figure 3.4. UML class diagram of the Elevator Control and Simulator system.

3.5. State Machine Diagram: Scheduler

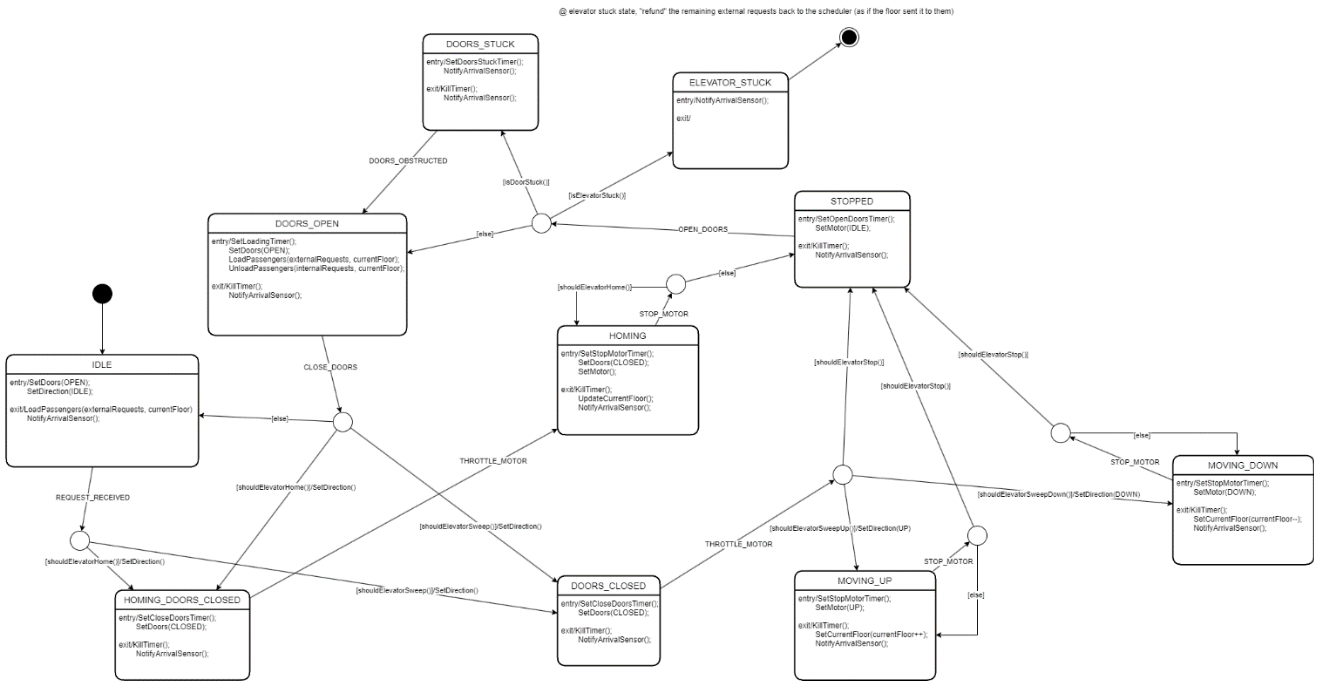


Figure 3.5. State machine diagram of the Scheduler.

3.6. Sequence Diagram: Error Scenarios

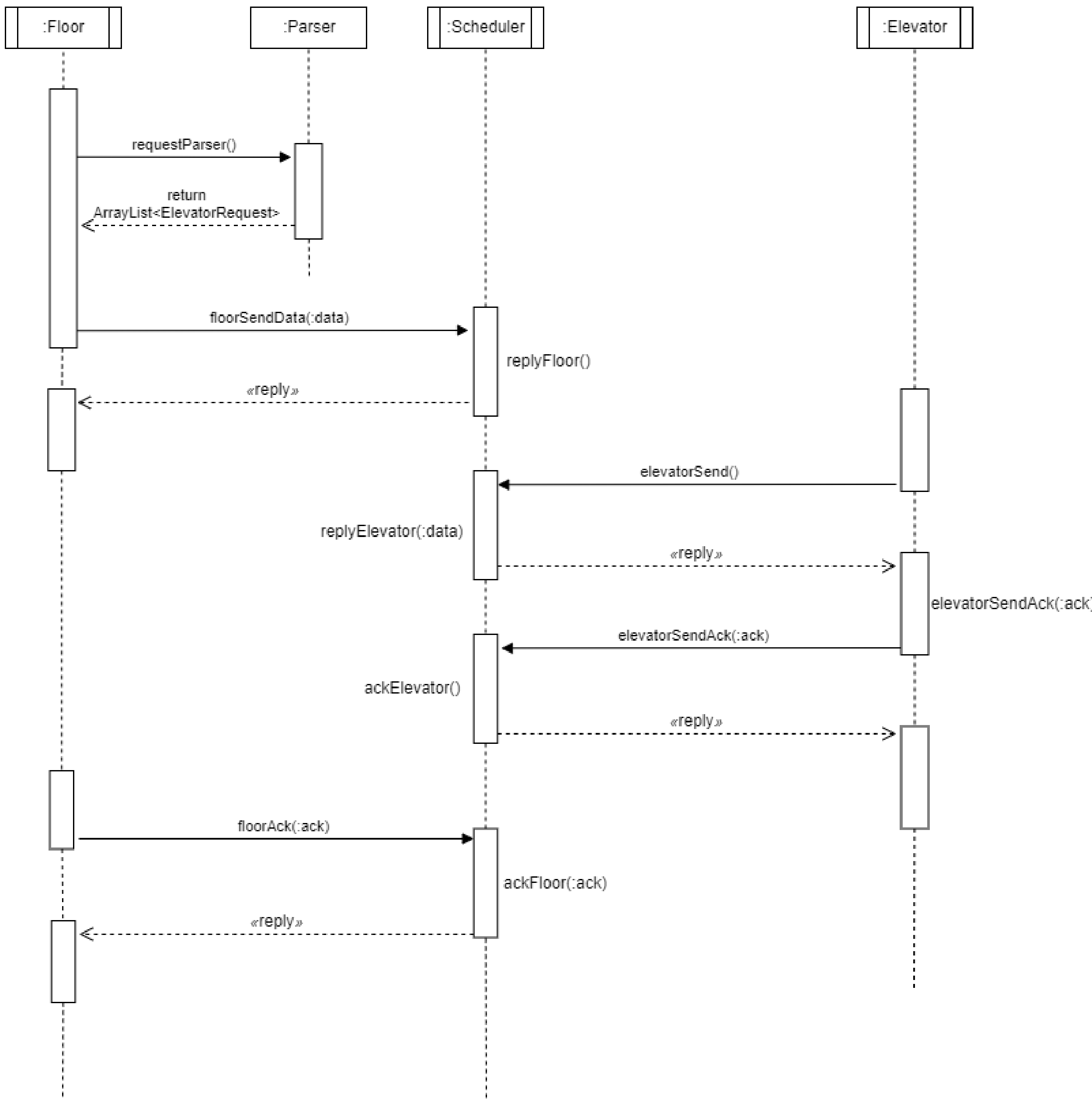


Figure 3.6. Sequence diagram of the system.

3.7. Timing Diagram: Scheduler

3.7.1. Timing Diagram: No faults (happy-path)

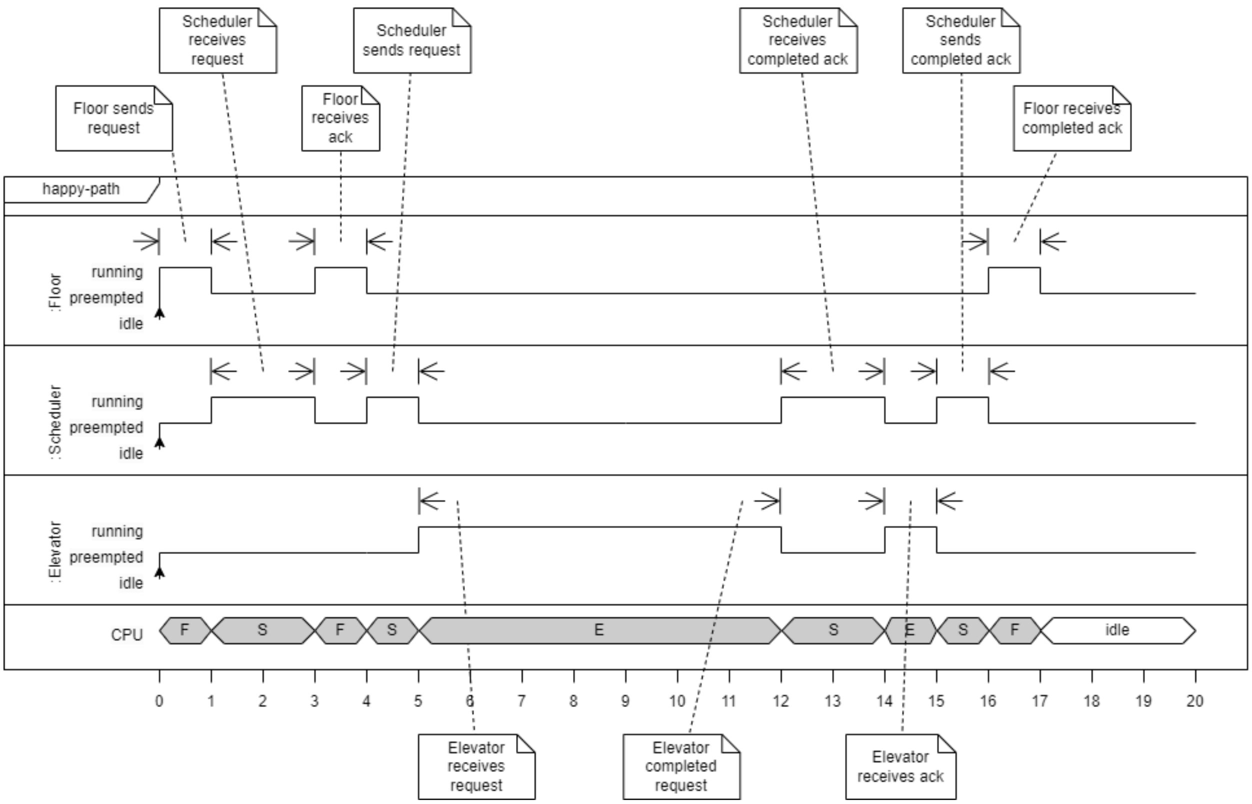


Figure 3.6.1. Timing diagram for no fault occurrence in the system.

3.7.2. Timing Diagram: Soft-fault (door-obstruction)

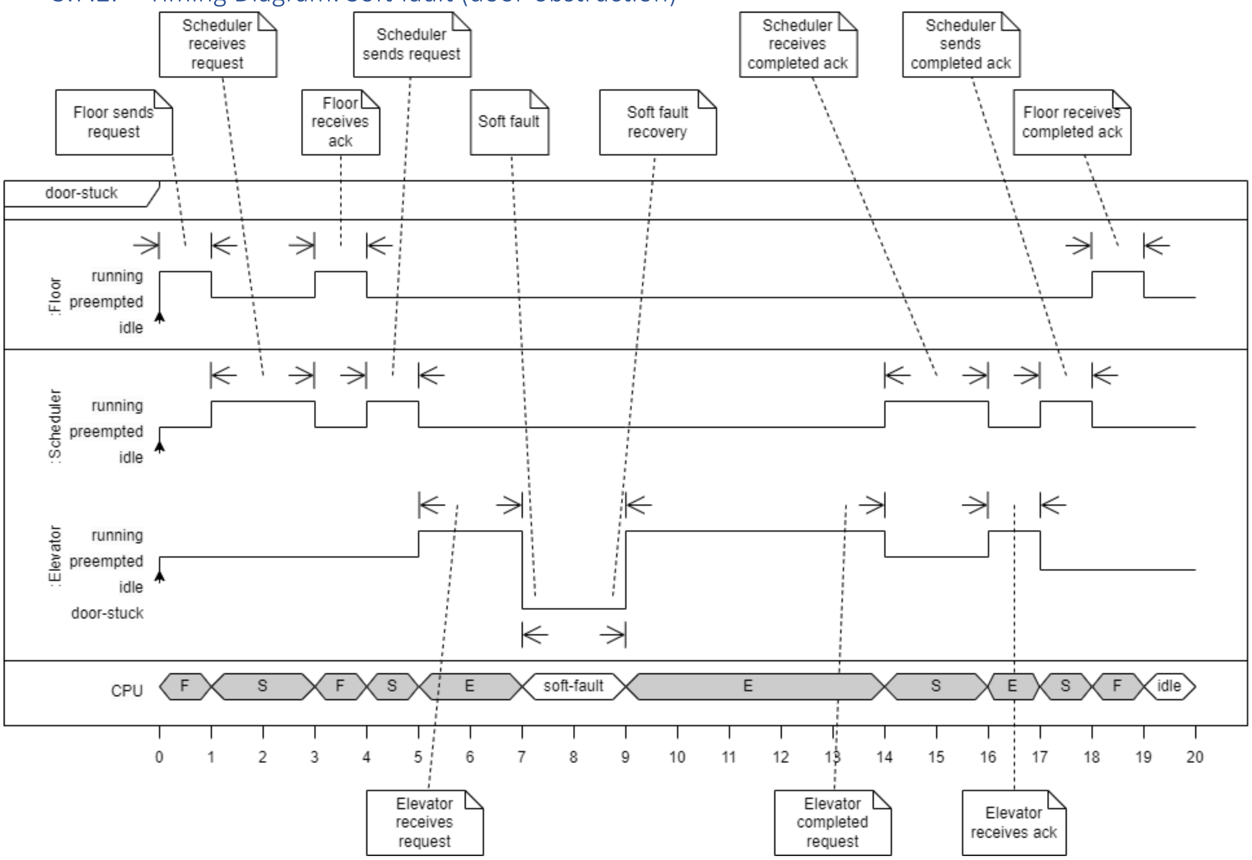


Figure 3.6.2. Timing diagram for soft fault occurrence in the system.

3.7.3. Timing Diagram: Hard-fault (elevator-stuck)

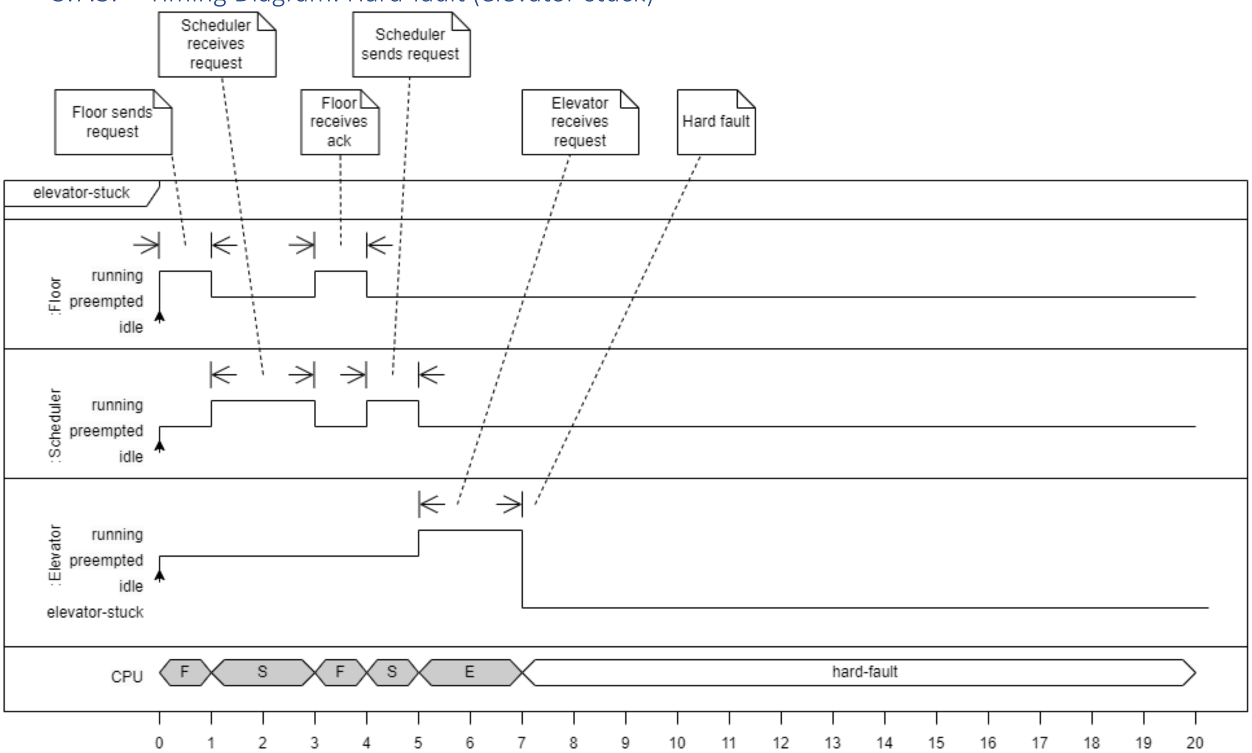


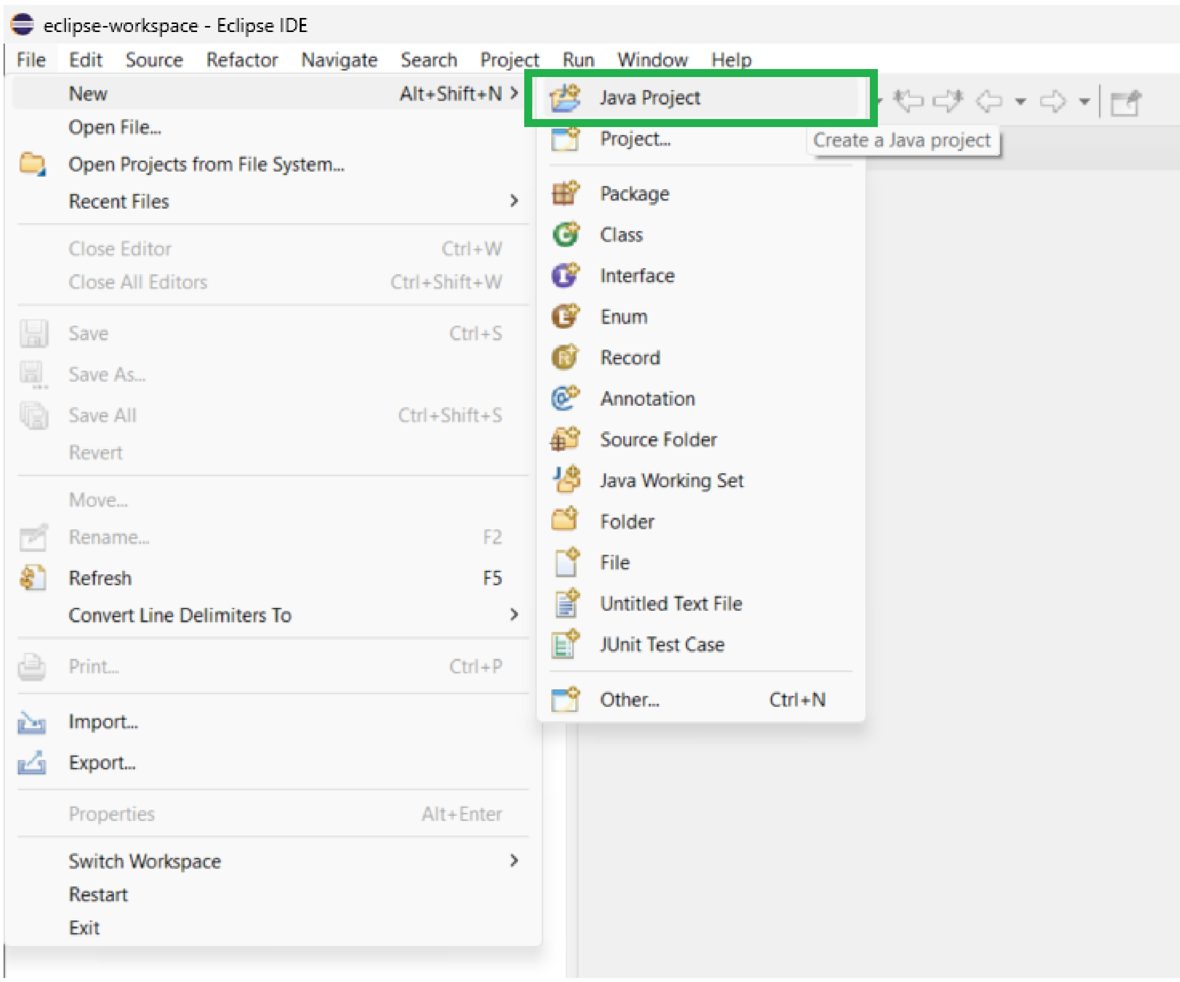
Figure 3.6.3. Timing diagram for hard fault occurrence in the system.

4. Set-up and test instructions

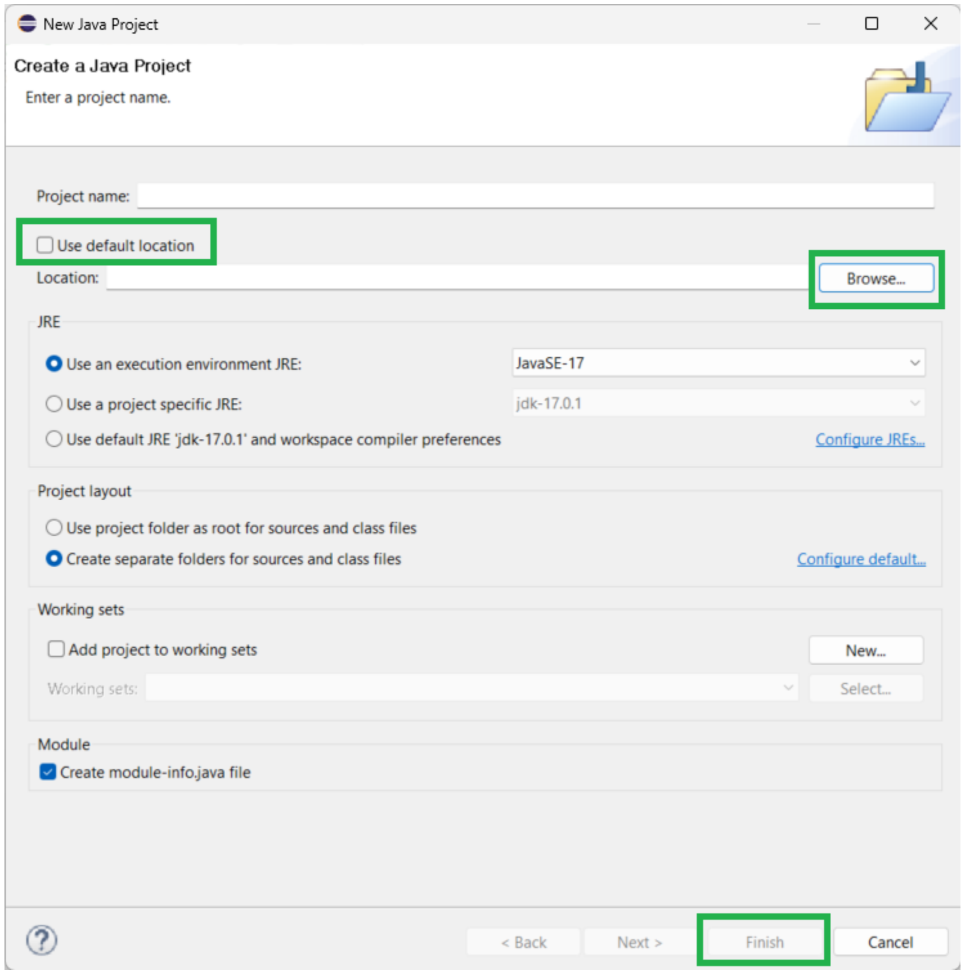
4.1. Setting up the Application

1. Download the .zip file containing the project
2. Unzip the compressed folder in a known directory
3. Open Eclipse IDE
4. Create a Java project

   - File > New > Java Project



5. Click on “Java Project” menu option to navigate to “New Java Project”

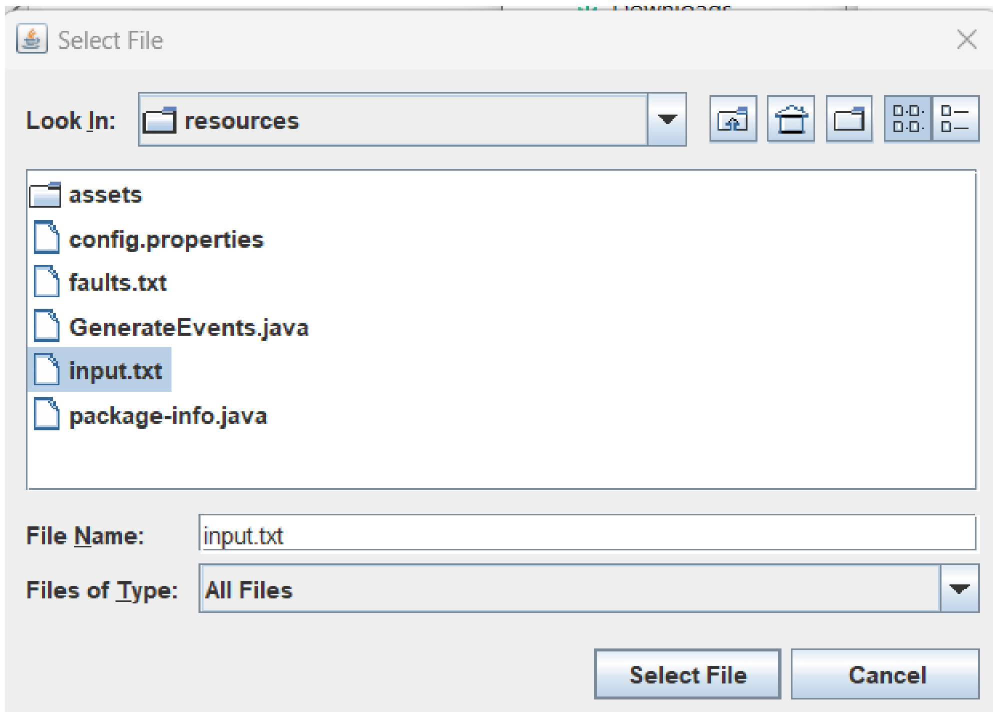


6. Untick “Use default location”
7. Click “Browse…”
8. Navigate to the extracted compressed fold containing the project
9. “Project name” field should auto-populate to *elevatorControlSystemAndSimulator*
10. Click “Finish”
11. Update the module name, if required and build file path as required
12. Expand directory and navigate to Main class
    - elevatorControlSystemAndSimulator > src > main > java > Main

4.2. Executing the Application

4.2.1 Executing using a single process

1. Right-click on Main
   - Run as > Java Application
2. Select “input.txt” for default running parameters Otherwise, select “faults.txt” to execute automated hard and soft faults in the application. Otherwise, select any other custom input file to use your own inputs.



4.2.2 Executing as 4 separate processes

1. Right-click on ElevatorSubsystem.java and Run as > Java Application

2. Right-click on SchedulerSubsystem.java and Run as > Java Application

3. Right-click on GUI.java and Run as > Java Application

4. Right-click on Subsystem.java and Run as > Java Application

5. Select “input.txt” for default running parameters Otherwise, select “faults.txt” to execute automated hard and soft faults in the application. Otherwise, select any other custom input file to use your own inputs.

4.3. Configuration of the Application

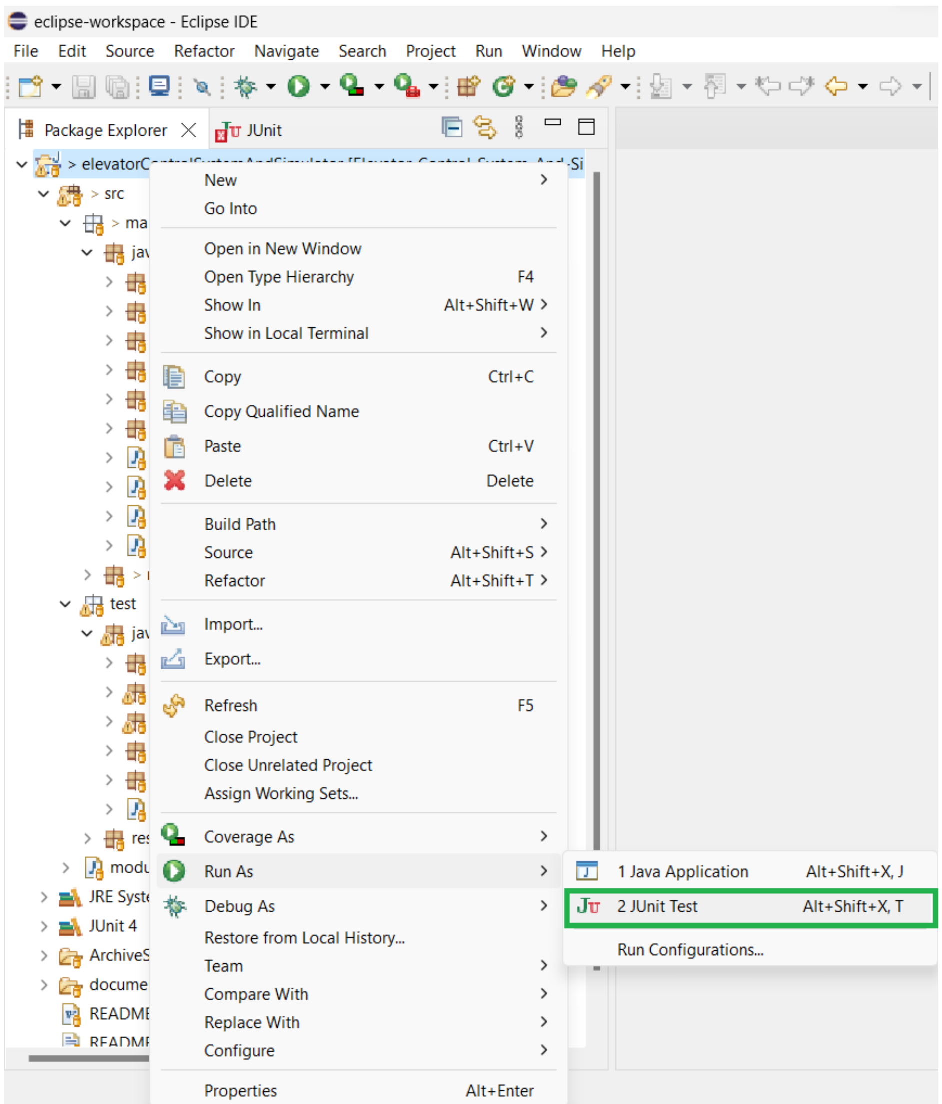
This application offers the ability to customize to enable dynamic numbers of elevators and floors. There are also configurable features that allow the application to run the various components on separate computers. The configurations can be inspected under:

- elevatorControlSystemAndSimulator > src > main > java > resources > config.properties

4.4. Testing Instructions

Assuming that JUnit4 library has been imported correctly.

1. Right-click on the main directory name: elevatorControlSystemAndSimulator
2. To run all the unit testing
   - Run as > JUnit Test
3. All unit test cases should pass



5. Results from measurements

5.1. Experimental procedure

Objective

The objective was to gather and record timing data for the time it takes an elevator to move between floors and to load/unload car. We decided to use the elevators located in the Canal Building during minimal usage hours for optimal consistency and maximize efficiency.

Questions

We are looking to determine the maximum speed of the elevator, the rate of acceleration for the elevator, and the average loading/unloading time.

Assumption

A difference in height of 4 meters between each floor.

Observation

The approach we took was to get a variety of times by going onto multiple floors and noting the time to travel between various floor differences, as seen in **Table 5.1**. For most measurements, we took 6 measurements of the same type to add some statistical backing in our findings. We also collected the boarding/exit time for when passengers would ideally load and unload the elevator car, as seen in **Table 5.2.**

5.2. Data reduction methods and results

Analyze Data

Interpreting the data. We decided to perform a geometric mean of the trials as it is the most appropriate for series that exhibit serial correlation meaning that one time frame will affect another timeframe. This is the case when we measured floor-by-floor as opposed to traveling multiple floors without stopping. Performed a standard deviation based on sample sized. However, the arithmetic and geometric means did not really differ dramatically in our dataset. Then we perform a confidence interval with an α = 0.05 for our sample size to satisfy the common 95% benchmark for confidence.

Sample Calculations

We measured for a difference of one (1) floor the distance between floors is 4 meters and the time travel is (10.479±00.242) s. And our largest timeframe was measured for six (6) floors difference giving us as travel time of (22.187±00.387) s. Therefore, we can assume that the maximum velocity can be represented as the following equation,

$$\Delta v = \frac{\Delta d}{\Delta t} = \frac{(6\,floor)(4\,\frac{m}{floor}) - (1\,floor)(4\,\frac{m}{floor})}{22.187\,s - 10.479\,s} = 1.708\,\frac{m}{s}$$

However, since this was our maximum range, we can assume that 1.708 m/s is maximum velocity.

We assumed to measure the time it takes to move between two adjacent floors and time it takes to move multiple floors. And, assuming that initial velocity is 0. We can model the relationship between velocity and distance in the following equation.

$$\Delta d = \left(\frac{v + v_0}{2}\right) t \Rightarrow t = \frac{\Delta d}{\left(\frac{v + v_0}{2}\right)} = \frac{4\ m}{\left(\frac{\left(1.708\,\frac{m}{s} + 0\right)}{2}\right)} = 4.676\ s$$

Thus, using the equation for acceleration,

$$\Delta a = \frac{\Delta v}{\Delta t} = \frac{1.7108\,\frac{m}{s}}{4.6762\ s} = 0.365\,\frac{m}{s^2}$$

5.3. Interpretation of data (synthesis) and discussion

Conclusion

Therefore, we can state that the maximum speed of the elevator is (1.708±0.895) m/s. The rate of acceleration for the elevator to be (0.365±0.112) m/s^2. And the average loading/unloading time to be (4.766±0.085) s, as seen in **Table 5.3.** Reference **Table 5.4.** for a tabulated summary of requested parameters outline as per objective requirement.

There was quite a handful of assumptions that went into interpreting and applying real world dynamics towards theorical predictions and relationships. Note that real elevators accelerate and decelerate as that start up and stop, so the time it takes to move between two floors depends on whether the car needs to stop or start. From the raw data, we made some mathematical and statistical trade-offs to best highlight the main objective to be able to measure tangible metrics of real-time mechanics and attempt to translate them into real-time process that can be executable via a code source. There are many edge cases that were not captured due to time and resource constraints. However, this just highlights the minute details that contribute to creating real-time systems.

Table 5.1. Collected sample timing between different floors with details on starting floor, direction, and car button pressed on the elevator.

Trial No.	**Time**	**Floor Button**	**Starting Floor**	**Car Button**
	hh:mm:ss.mmm	Up/Down	n	n
Floor diff = 1				
1	00:00:10.150	Up	1	2
2	00:00:10.200	Up	2	3
3	00:00:10.660	Up	3	4
4	00:00:10.580	Up	4	5
5	00:00:10.600	Up	5	6
6	00:00:10.700	Up	7	7
GEOMEAN	00:00:10.479			
STDEV	00:00:00.242			
CONFIDENCE	00:00:00.254			
Floor diff = 2				
1	00:00:12.530	Up	1	3
2	00:00:12.210	Up	4	6
3	00:00:13.090	Up	5	7
4	00:00:12.420	Down	4	2
5	00:00:12.190	Down	5	3
6	00:00:13.140	Down	6	4
GEOMEAN	00:00:12.591			
STDEV	00:00:00.422			
CONFIDENCE	00:00:00.443			
Floor diff = 4				
1	00:00:17.440	Up	3	7
2	00:00:17.340	Up	2	6
3	00:00:17.570	Up	2	6
4	00:00:16.530	Down	7	3
5	00:00:17.090	Down	7	2
6	00:00:17.180	Down	6	2
GEOMEAN	00:00:17.188			
STDEV	00:00:00.367			
CONFIDENCE	00:00:00.386			
Floor diff = 6				
1	00:00:22.420	Up	1	7
2	00:00:22.310	Down	7	1
3	00:00:22.520	Up	1	7
4	00:00:21.450	Down	7	1
5	00:00:22.330	Up	1	7
6	00:00:22.110	Down	7	1
GEOMEAN	00:00:22.187			
STDEV	00:00:00.387			
CONFIDENCE	00:00:00.406			

Table 5.2. Collected data for the boarding and exit times of the elevator.

Trial No.	Boarding/exit Time	Door Open + Idle Time	Load/Unload Time
1	00:00:08.070	00:00:03.330	00:00:04.740
2	00:00:08.090	00:00:03.350	00:00:04.740
3	00:00:08.010	00:00:03.360	00:00:04.650
4	00:00:08.020	00:00:03.110	00:00:04.910
5	00:00:08.120	00:00:03.360	00:00:04.760
6	00:00:08.130	00:00:03.340	00:00:04.790
GEOMEAN	00:00:08.073	00:00:03.307	00:00:04.766
STDEV	00:00:00.050	00:00:00.098	00:00:00.085
CONFIDENCE	00:00:00.053	00:00:00.103	00:00:00.089

Table 5.3. Summary table of all the statistical analysis performed on the data of the timing between elevator floors collected with sample size = 6 and α = 0.05.

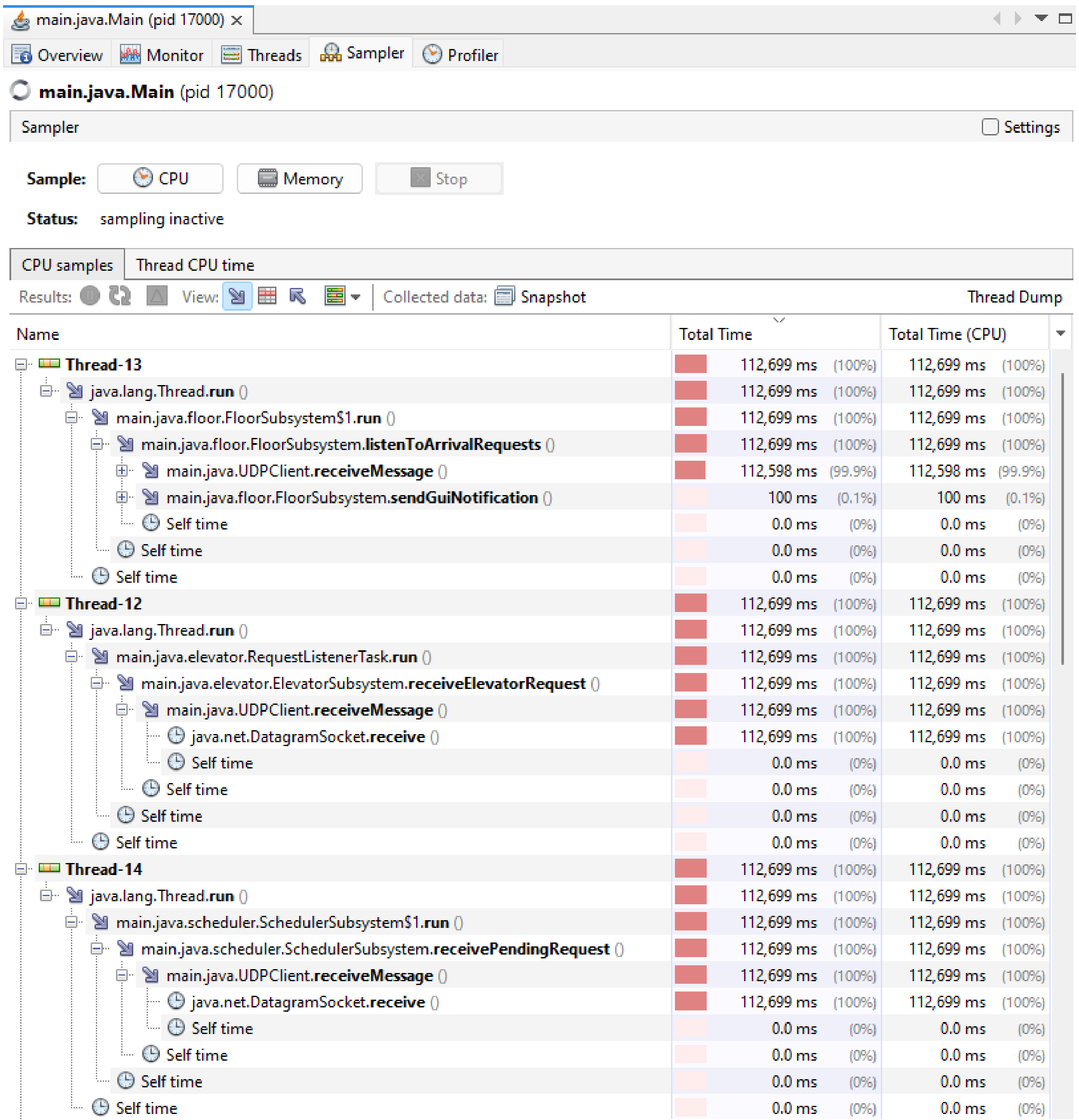
Floor difference	GEOMEAN hh:mm:ss.mmm	STDEV hh:mm:ss.mmm	CONFIDENCE hh:mm:ss.mmm
1	00:00:10.479	00:00:00.242	00:00:00.254
2	00:00:12.591	00:00:00.422	00:00:00.443
4	00:00:17.188	00:00:00.367	00:00:00.386
6	00:00:22.187	00:00:00.387	00:00:00.406
Boarding/exit	00:00:08.073	00:00:00.050	00:00:03.651
Door Open	00:00:03.307	00:00:00.098	00:00:00.103
Load/Unload	00:00:04.766	00:00:00.085	00:00:00.089

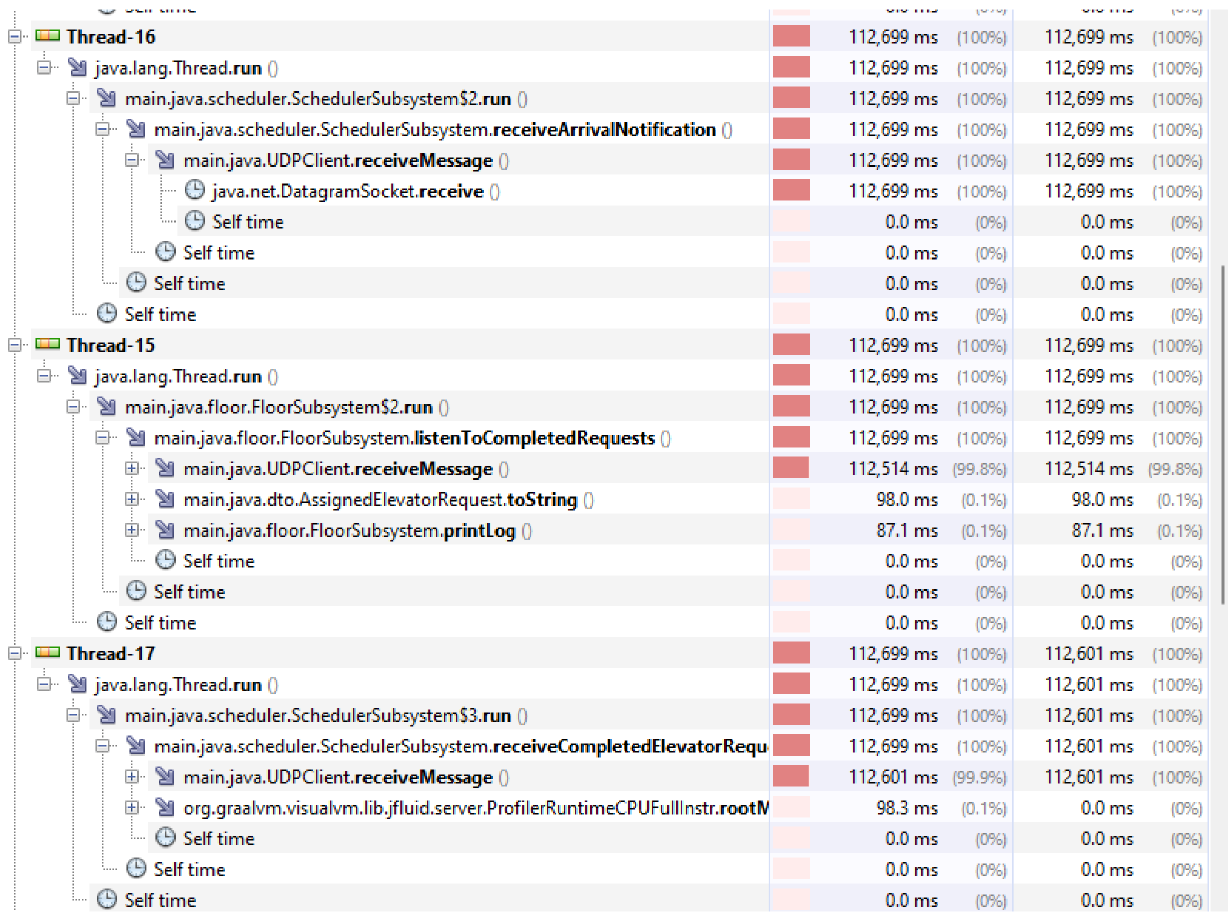
Table 5.4. Summary table of results and interpretation calculated from raw metric date of the elevator times. NB. Error values were calculated using propagation of uncertainty.

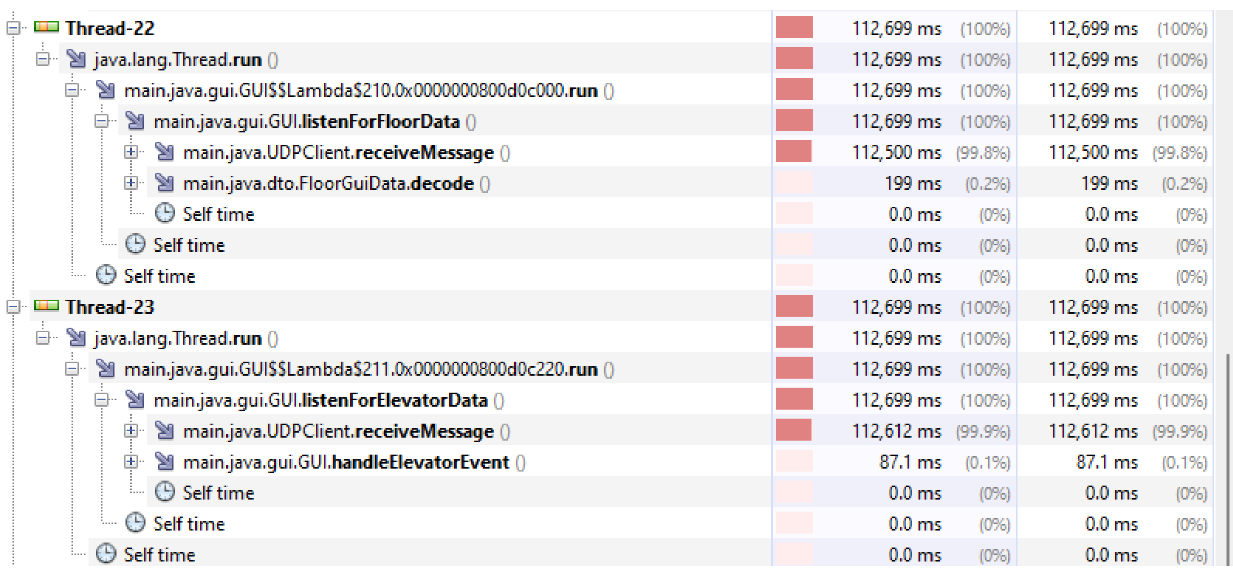
	VALUE	ERROR
AVG SPEED [m/s]	1.708	± 0.895
d TIME [s]	4.676	± 0.111
ACCELERATION [m/s2]	0.365	± 0.902
LOAD/UNLOAD [s]	4.766	± 0.085

5.4. VisualVM Sampling of Concurrent Thread Run-Time Analysis

Java VisualVM is a tool that provides a visual interface for troubleshooting and profiling Java applications while they are running on a Java Virtual Machine (JVM). It can be used to view detailed information about the application's performance and to improve it. CPU profiling is a command that returns detailed data on method-level CPU performance and shows the total execution time and number of invocations for each method. Java VisualVM instruments all the methods of the profiled application, and threads emit the "method entry" and "method exit" events when entering and exiting a method, respectively, which are processed in real-time.







This allows us to visualize the most CPU taxing threads in our application. It is no surprize that most of these processes involve message passing, decode/encode data, and handle events within the system.

6. Schedulability Analysis

6.1. System Throughput

The program has detailed logging with timestamps to measure temporal performance metrics. There is a system in place that can accelerate time for testing purposes, but the measurements provided were conducted without time acceleration. During time measurements, the time formatting was temporarily changed to show milliseconds for precision but reverted to avoid cluttering the logging.

Table 6.1. System throughput of the elevator simulator.

Trial No.	**Time** [HH:mm:ss.mmm]
1	00:03:42.966
2	00:04:03.005
3	00:03:53.000
4	00:03:37.000
5	00:03:47.547
6	00:03:46.706
AVERAGE	00:03:48.371
STDEV	00:00:08.917
CONFIDENCE	00:00:09.358
AVERAGE [ms]	228371
STDEV [ms]	8917
CONFIDENCE [ms]	9358
Num. Request Completed (C)	30
AVERAGE THRU TIME (X=C/T)	0.000131365
STDEV	0.00336436
CONFIDENCE	0.003205813

6.2. Service Time

It's important to note that the estimate of the average service time based on the demo input file may not accurately represent the true average service time in a real-world scenario with a different set of variables. However, it can provide a baseline for further optimization and improvement.

Regarding the implementation of a more sophisticated system for tracking the average service time, machine learning systems can be trained on large datasets to improve predictions and estimate more complex patterns in the data. However, it's important to consider the cost of implementing such a system and whether it's worth the additional resources required to maintain and improve it. It may also be necessary to collect additional data and modify the system as the environment changes to maintain accurate predictions.

Table 6.2. Service time of the elevator simulator.

Trial No.	**Time** [HH:mm:ss.mmm]
1	00:03:42.966
2	00:04:03.005
3	00:03:53.000
4	00:03:37.000
5	00:03:47.547
6	00:03:46.706
AVERAGE (B)	00:03:48.371
STDEV	00:00:08.917
CONFIDENCE	00:00:09.358
Num. Request Completed (C)	30
AVERAGE SERVICE TIME (S=B/C)	00:00:07.612
STDEV	00:00:00.297
CONFIDENCE	00:00:00.312
AVERAGE SERVICE TIME (S) [ms]	7612
STDEV [ms]	267
CONFIDENCE [ms]	312

6.3. System Utilization

Utilization refers to the percentage of time that an elevator is being used to perform work or provide service. In the context of queuing theory, utilization is the ratio of the average service time to the average time between arrivals of an elevator.

Utilization is an important metric in analyzing the performance of queuing systems because it affects both the throughput (rate of processing requests) and the service time (time taken to process a request). When the utilization of a server is high, it can lead to longer service times and reduced throughput, as the CPU is working at its capacity and may not be able to keep up with the incoming requests. On the other hand, when the utilization is low, the server may be underutilized and not performing to its maximum potential.

Table 6.3. System utilization of the elevator simulator.

Utilization (U = X * S)	0.999951833

6.4. Waiting Queue Time and System Response Time

It's important to note that the true value of the average queuing time will be affected by many factors, such as the number of active elevators, the capacity of each elevator, the scheduling algorithm, and the distribution of floor requests over time. Therefore, the estimated value based on the console log analysis should be considered with caution and may not accurately reflect the true value of the average queuing time in all scenarios.

Table 6.4. Waiting Queue Time and System Response Time of the elevator simulator.

Trial No.	**Time** [HH:mm:ss.mmm]	**Average number of requests in Queue** (N)
1	00:00:25.230	3
2	00:00:36.317	2
3	00:00:44.502	3
4	00:00:24.662	4
5	00:00:20.663	5
6	00:00:14.395	1
AVERAGE (W)	00:00:27.628	3
STDEV	00:00:10.940	1.41421356
CONFIDENCE	00:00:11.481	1.48412611
Num. Request Completed (C)	30	
AVERAGE RESPONSE Time (R = W/C)	00:00:09.209	
STDEV	00:00:00.365	
CONFIDENCE	00:00:00.383	

Over any time, interval, the area between the arrival and completion functions represents the accumulated time W in system during that interval, measured in request-seconds (or request-minutes, etc.). System response time is the time taken by a system to respond to a user request or input. It includes the time taken for the system to receive the request, process it, and provide a response. In general, a shorter response time is considered better as it implies that the system can process requests quickly and efficiently.

Little's law is a fundamental theorem in queuing theory, which relates the average number of elevator requests in a queuing system to the arrival rate of elevators and the average time they spend in the system. It states that the long-term average number of elevators in a stable system (that is, one in which the arrival rate equals the departure rate) is equal to the product of the average arrival rate and the average time that a elevator spends in the system. In mathematical notation, Little's law can be expressed as: $L = \lambda W$

where L is the long-term average number of elevator requests in the system, λ is the average arrival rate of elevator request entry and W is the average time that an elevator spends in the system. Little's law is widely used in operations research, industrial engineering, and computer science, to estimate the performance of queuing systems, to design efficient production lines, and to optimize computer networks, among other applications.

6.5. Transit Time

The program will determine the average transit time by analyzing the console log to identify when passengers are picked up and when they are delivered to their destination. The measurement will not consider overlapping paths with other requests for simplicity, and the results will only be used as estimates for the timing diagrams.

Table 6.5. Transit Time of the elevator simulator.

Trial No.	**Time** [HH:mm:ss.mmm]
1	00:00:16.612
2	00:00:13.736
3	00:00:24.226
4	00:00:20.795
5	00:00:19.610
6	00:00:14.486
AVERAGE (T)	00:00:18.244
STDEV	00:00:04.032
CONFIDENCE	00:00:04.231
AVERAGE (T) [ms]	182244
STDEV [ms]	4032
CONFIDENCE [ms]	4231
Num. Request Completed (C)	30
AVERAGE TRANSIT TIME/request	6074.8
STDEV	134.4
CONFIDENCE	141.0333333

The note explains that the reason why the individually measured transit times and queueing times do not add up to the average service time is because the latter is based on multiple elevators with temporal acceleration, whereas the former two are measured for a single elevator. Additionally, the average service time is calculated based on an average of 7.5 requests per elevator.

7. Reflections on Design

7.1 System Features

The elevator subsystem features a wide range of states and dynamic state transitions. For example, when the elevator is at a DOORS_CLOSED state, it can transition back to the DOORS_OPEN state in the event that a request at the same floor and same direction is received. This example behaviour is modeled after a common scenario observed at the Canal Building elevators, where when a passenger presses a floor button just after the elevator door closes, the elevator promptly opens its doors instead of leaving the passenger behind. An exhaustive description of all the state transitions can be found on the state diagram submission attached in the project.

A feature included in the elevator subsystem and the scheduler subsystem is that there is ability to re-schedule tasks to another elevator in the event of a fault. When an elevator reached a DOORS_STUCK or ELEVATOR_STUCK state, it will return all the requests for the passengers that it has not yet picked up for re-assignment to another elevator. This allows the elevator system to service requests smoothly without suffering from large delays due to faults, regardless of the duration of the fault. For example, if the DOOR_STUCK event occurred on an elevator, the passengers that the elevator was assigned to pickup would not suffer from any delays due to the fault because their request would be re-scheduled to the next best available elevator; only the passengers inside of the elevator at the time will suffer from delays from the fault.

The elevators' movement algorithm is modeled after the C-SCAN disk scheduling algorithm and the scheduler's algorithm complements this. The goal of the elevator is to keep moving as high as possible until there are no more requests to serve and then to keep moving as low possible until there are no more requests to serve. This allows passengers going in the same direction to be picked up in larger batches by a single elevator, which is very efficient.

A key feature of the scheduler's algorithm is that it will never assign two or more elevators to service requests with the same-source and same-direction at the same time. Any same-source and same-direction requests that have both not been picked up yet are guaranteed to be service3d at the same time by the same elevator.

The floor subsystem has been implemented to fire requests at wall-clock time based on the timestamps described in the input file.

The simulation as whole is highly configurable. The user can customize the elevator loading, moving, doors open, doors closed, & doors stuck fault durations. The user can also customize the number of elevators and floors from the configuration file. Moreover, the host addresses and ports of the Elevator, Scheduler, Floor, and GUI subsystems can be configured.

The simulation follows a distributed architecture and can run as 4 different processes interconnected on a Local Area Network. Each subsystem consists of a control class which handles the network communications between each subsystem and entity classes which contain the subsystem's application data.

7.2 Improvements

The group is satisfied with the project's design choices, which were based on the single responsibility principle, making enhancements easy to implement as the project progressed. The design is configurable and flexible, requiring minimal modifications to accommodate varying configurations. However, the group acknowledges the need for automated testing, as the testing framework initially developed was not updated in accordance with the project.

The first improvement suggested is to use generics to implement serializable encode/decode once, and then have all data transfer objects inherit a parent class with this implementation to avoid rewriting the method.

However, there is a scalability issue with the current approach of opening a new network port each time new data needs to be sent. This approach is not scalable because there is a limited number of ports on a computer. The reason for opening new ports is to avoid having to parse and figure out what kind of object it is, but a potential improvement would be to wrap data transfer object packets with a header to indicate the type of object that it is, and then route the object to the correct method accordingly.

This could be achieved by wrapping the object in a superclass with an identifier that tells you how to "route" the object when you send it to a host. For example, ElevatorRequest objects could be wrapped with an attribute String value "pendingRequest", which is like a URI resource in REST API interfaces. When the scheduler detects an object with attribute "pendingRequest", it takes the packet and treats it like a pending request. Similarly, a packet sent to the scheduler wrapped with "arrivalNotification" will be treated as an ElevatorStatus arrival notification.

Another improvement suggested is to change the scheduler to not use a state machine, as it does not fit nicely and feels hammered into the current approach. Instead, a different approach could be used, where each request is a thread competing for a critical section. The request should wait if there is no ideal elevator yet and notify all waiting threads on every arrival notification elevator status update. However, there is no defined new approach for this yet.

4. Appendices

4.1. Appendix A: Work Distribution

Iteration 5

ECSS-112	Documentation checklist	Bobby Ngo	Bobby Ngo	IN PROGRESS	Unresolved	Apr 10, 2023	Apr 10, 2023
ECSS-111	Duplicate print out for completed tasks for schedulerSubsystem	Trong Nguyen	Trong Nguyen	IN PROGRESS	Unresolved	Apr 10, 2023	Apr 10, 2023
ECSS-110	12 to 00-hour time	Trong Nguyen	Trong Nguyen	DONE	Done	Apr 10, 2023	Apr 10, 2023
ECSS-109	Add homing state	Zakaria Ismail	Zakaria Ismail	IN PROGRESS	Unresolved	Apr 10, 2023	Apr 10, 2023
ECSS-108	Implement floor GUI components	Trong Nguyen	Trong Nguyen	DONE	Done	Apr 9, 2023	Apr 10, 2023
ECSS-107	Allow commit of files in resources directory w/o forcing	Zakaria Ismail	Zakaria Ismail	DONE	Done	Apr 4, 2023	Apr 4, 2023
ECSS-106	Set timestamp dates to work with daylight savings time	Zakaria Ismail	Zakaria Ismail	IN REVIEW	Unresolved	Apr 4, 2023	Apr 4, 2023
ECSS-105	Add test mode to disable elevator state transition via timeout	Zakaria Ismail	Zakaria Ismail	DONE	Done	Apr 4, 2023	Apr 4, 2023
ECSS-104	Send arrival notif at every state change	Zakaria Ismail	Zakaria Ismail	DONE	Done	Apr 3, 2023	Apr 3, 2023
ECSS-103	Implement GUI code	Trong Nguyen	Trong Nguyen	DONE	Done	Mar 31, 2023	Apr 9, 2023
ECSS-102	Measure the performance of the Scheduler subsystem	Unassigned	Zakaria Ismail	TO DO	Unresolved	Mar 29, 2023	Mar 29, 2023
ECSS-101	Add descriptive logging of events and elevator context	Unassigned	Zakaria Ismail	TO DO	Unresolved	Mar 29, 2023	Mar 30, 2023
ECSS-100	Add javadoc	Trong Nguyen	Zakaria Ismail	IN REVIEW	Unresolved	Mar 29, 2023	Apr 10, 2023
ECSS-99	Handle packet loss in all subsystems	Zakaria Ismail	Zakaria Ismail	TO DO	Unresolved	Mar 29, 2023	Mar 30, 2023
ECSS-98	Verify and improve job scheduling algorithm	Bobby Ngo	Zakaria Ismail	DONE	Done	Mar 29, 2023	Apr 10, 2023
ECSS-97	Scheduler Subsystem	Unassigned	Zakaria Ismail	DONE	Done	Mar 29, 2023	Apr 10, 2023
ECSS-96	Handle new error column value from incoming Elevator Request messages	Zakaria Ismail	Zakaria Ismail	DONE	Done	Mar 29, 2023	Apr 4, 2023
ECSS-95	Update Parser to handle error column in input file	Patrick Liu	Zakaria Ismail	DONE	Done	Mar 29, 2023	Apr 4, 2023
ECSS-94	Add unit tests	Zakaria Ismail	Zakaria Ismail	IN PROGRESS	Unresolved	Mar 29, 2023	Apr 10, 2023

Iteration 4

ECSS-93	Verify elevator job queueing algorithm	Zakaria Ismail	Zakaria Ismail	TO DO	Unresolved	Mar 29, 2023	Apr 1, 2023
ECSS-92	Elevator Subsystem	Unassigned	Zakaria Ismail	TO DO	Unresolved	Mar 29, 2023	Mar 29, 2023
ECSS-91	Create design doc for Floor subsystem	Unassigned	Zakaria Ismail	TO DO	Unresolved	Mar 29, 2023	Mar 30, 2023
ECSS-90	Floor Subsystem	Unassigned	Zakaria Ismail	TO DO	Unresolved	Mar 29, 2023	Mar 30, 2023
ECSS-89	Create design doc for GUI Subsystem	Trong Nguyen	Zakaria Ismail	DONE	Done	Mar 29, 2023	Apr 9, 2023
ECSS-88	GUI Subsystem	Trong Nguyen	Zakaria Ismail	TO DO	Unresolved	Mar 29, 2023	Mar 31, 2023
ECSS-87	Data transfer object for Floor Subsystem -> GUI Subsystem	Hussein El Mokdad	Zakaria Ismail	DONE	Done	Mar 29, 2023	Apr 10, 2023
ECSS-86	Data transfer object for Elevator Subsystem -> GUI Subsystem	Zakaria Ismail	Zakaria Ismail	DONE	Done	Mar 29, 2023	Mar 30, 2023
ECSS-85	Integration & Interfacing	Unassigned	Zakaria Ismail	TO DO	Unresolved	Mar 29, 2023	Mar 29, 2023
ECSS-84	Completed requests are received after all requests have been sent	Unassigned	Hussein El Mokdad	TO DO	Unresolved	Mar 22, 2023	Mar 30, 2023
ECSS-82	Scheduler in the elevator subsystem is not the same as the one in scheduler subsystem	Unassigned	Hussein El Mokdad	TO DO	Unresolved	Mar 19, 2023	Mar 30, 2023
ECSS-81	Elevator doesn't open and close doors when picking up users	Unassigned	Hussein El Mokdad	DONE	Done	Mar 18, 2023	Mar 19, 2023
ECSS-80	Improve logging with multiple elevators	Trong Nguyen	Hussein El Mokdad	DONE	Done	Mar 18, 2023	Apr 10, 2023
ECSS-79	Improve logging detail in the UDP class	Hussein El Mokdad	Hussein El Mokdad	TO DO	Unresolved	Mar 17, 2023	Apr 4, 2023

Iteration 3

ECSS-78	Update Work Distribution Document	Trong Nguyen	Trong Nguyen	IN PROGRESS	Unresolved	Mar 11, 2023	Mar 11, 2023
ECSS-77	Update README	Trong Nguyen	Trong Nguyen	IN PROGRESS	Unresolved	Mar 11, 2023	Mar 11, 2023
ECSS-76	Update State Diagrams	Trong Nguyen	Trong Nguyen	DONE	Done	Mar 11, 2023	Mar 11, 2023
ECSS-75	Update Sequence UML	Trong Nguyen	Trong Nguyen	DONE	Done	Mar 11, 2023	Mar 11, 2023
ECSS-74	Update UML	Trong Nguyen	Trong Nguyen	DONE	Done	Mar 11, 2023	Mar 11, 2023
ECSS-73	Create method: sort elevatorRequest by time stamp before adding to arraylist	Patrick Liu	Patrick Liu	DONE	Done	Mar 10, 2023	Mar 10, 2023

ECSS-72	Implement distributed processes - method can be execute on different computers	Trong Nguyen	Trong Nguyen	DONE	Done	Mar 5, 2023	Mar 10, 2023
ECSS-71	Create method: Fill elevatorRequest String's timestamp position with 0s to 3 digits	Patrick Liu	Patrick Liu	DONE	Done	Mar 5, 2023	Mar 5, 2023
ECSS-70	Action for workflow and State	Unassigned	Bobby Ngo	TO DO	Unresolved	Mar 5, 2023	Mar 5, 2023
ECSS-69	Documentation	Trong Nguyen	Zakaria Ismail	IN PROGRESS	Unresolved	Mar 1, 2023	Mar 11, 2023
ECSS-68	Create multiple floors and elevators with each elevator being a separate thread	Hussein El Mokdad	Hussein El Mokdad	IN REVIEW	Unresolved	Mar 1, 2023	Mar 5, 2023
ECSS-67	Make the elevator move to each floor instead of teleporting	Bobby Ngo	Bobby Ngo	DONE	Done	Mar 1, 2023	Mar 5, 2023
ECSS-66	Define simple error handling	Unassigned	Trong Nguyen	TO DO	Unresolved	Feb 27, 2023	Mar 1, 2023
ECSS-65	Improve parser	Patrick Liu	Trong Nguyen	DONE	Done	Feb 27, 2023	Mar 5, 2023
ECSS-63	Implement scheduling sweeping algorithm + testing	Zakaria Ismail	Trong Nguyen	TO DO	Unresolved	Feb 27, 2023	Mar 11, 2023
ECSS-62	Console UI integration with Static Model of Domain	Bobby Ngo	Trong Nguyen	IN PROGRESS	Unresolved	Feb 27, 2023	Mar 6, 2023
ECSS-61	Implement Remote Procedure Calls between Classes	Trong Nguyen	Trong Nguyen	DONE	Done	Feb 27, 2023	Mar 10, 2023
ECSS-60	Data structure for reading timestamp events and input in the system	Trong Nguyen	Trong Nguyen	IN PROGRESS	Unresolved	Feb 27, 2023	Mar 7, 2023
ECSS-59	Iteration 3	Unassigned	Zakaria Ismail	TO DO	Unresolved	Feb 22, 2023	Feb 22, 2023

Iteration 2

ECSS-58	Javadocs + final edits	Unassigned	Trong Nguyen	IN REVIEW	Unresolved	Feb 23, 2023	Feb 23, 2023
ECSS-57	Proposal to refactor Elevator state machine implementation	Unassigned	Zakaria Ismail	TO DO	Unresolved	Feb 22, 2023	Feb 22, 2023
ECSS-56	Move Parser package into Floor package	Unassigned	Zakaria Ismail	DONE	Done	Feb 21, 2023	Feb 23, 2023
ECSS-55	Sending Elevator Location back to the Scheduler After Reaching Stop State	Unassigned	Patrick Liu	DONE	Done	Feb 20, 2023	Feb 22, 2023
ECSS-54	Improve Elevator State machine logic	Unassigned	Bobby Ngo	DONE	Done	Feb 20, 2023	Feb 22, 2023
ECSS-52	Implement SchedulerState enum based on StateMachine diagram	Unassigned	Trong Nguyen	DONE	Done	Feb 18, 2023	Feb 21, 2023
ECSS-51	Implement ElevatorState enum based on StateMachine diagram	Unassigned	Trong Nguyen	DONE	Done	Feb 18, 2023	Feb 21, 2023
ECSS-50	SchedulerState - Unit testing state machine implementation	Trong Nguyen	Trong Nguyen	DONE	Done	Feb 8, 2023	Feb 20, 2023
ECSS-49	ElevatorState - Unit Testing state machine implementation	Trong Nguyen	Trong Nguyen	DONE	Done	Feb 8, 2023	Feb 20, 2023
ECSS-48	Unit testing - Iteration 2	Unassigned	Trong Nguyen	DONE	Done	Feb 8, 2023	Feb 20, 2023
ECSS-47	ELEVATOR - State machine diagram	Trong Nguyen	Trong Nguyen	DONE	Done	Feb 8, 2023	Feb 20, 2023
ECSS-46	SCHEDULER - State machine diagram	Trong Nguyen	Trong Nguyen	DONE	Done	Feb 8, 2023	Feb 20, 2023
ECSS-45	README - update	Bobby Ngo	Trong Nguyen	DONE	Done	Feb 8, 2023	Feb 22, 2023
ECSS-44	Distribution of Responsibilities	Bobby Ngo	Trong Nguyen	DONE	Done	Feb 8, 2023	Feb 22, 2023
ECSS-43	UML Sequence Diagram - update	Patrick Liu	Trong Nguyen	DONE	Done	Feb 8, 2023	Feb 22, 2023
ECSS-42	UML Class Diagram - update	Patrick Liu	Trong Nguyen	DONE	Done	Feb 8, 2023	Feb 22, 2023
ECSS-41	Documentation - Iteration 2	Trong Nguyen	Trong Nguyen	IN PROGRESS	Unresolved	Feb 8, 2023	Feb 22, 2023
ECSS-40	Update UML Class diagram	Bobby Ngo	Bobby Ngo	DONE	Done	Feb 4, 2023	Feb 4, 2023
ECSS-39	Improve console logging UI	Hussein El Mokdad	Bobby Ngo	DONE	Done	Feb 4, 2023	Feb 22, 2023
ECSS-38	Review the output logging and add improvements to the presentation if possible	Bobby Ngo	Zakaria Ismail	TO DO	Unresolved	Feb 4, 2023	Feb 4, 2023
ECSS-37	Add sleep timer to allow the user to see the system interactions	Bobby Ngo	Zakaria Ismail	TO DO	Unresolved	Feb 4, 2023	Feb 4, 2023

Iteration 1

Type	Key	Summary	Assignee	Reporter	P	Status	Resolution	Created	Updated
	ECSS-1	Setup tasks	Bobby Ngo	Bobby Ngo		DONE	Done	Jan 15, 2023	Feb 3, 2023
	ECSS-2	Iteration 1 Implementation	Unassigned	Patrick Liu		TO DO	Unresolved	Jan 18, 2023	Jan 25, 2023
	ECSS-4	Submit Iteration 1 on BrightSpace	Trong Nguyen	SYSC3303 Project...		TO DO	Unresolved	Jan 18, 2023	Feb 4, 2023
	ECSS-5	Elevator class	Trong Nguyen	SYSC3303 Project...		DONE	Done	Jan 18, 2023	Feb 2, 2023
	ECSS-6	Scheduler Class	Bobby Ngo	SYSC3303 Project...		DONE	Done	Jan 18, 2023	Jan 29, 2023
	ECSS-7	Floor class	Hussein El Mokdad	SYSC3303 Project...		DONE	Done	Jan 18, 2023	Jan 29, 2023
	ECSS-8	Complete README.txt	Bobby Ngo	SYSC3303 Project...		IN REVIEW	Unresolved	Jan 18, 2023	Feb 3, 2023
	ECSS-9	Iteration 1 - Breakdown of Responsibilities	Trong Nguyen	SYSC3303 Project...		DONE	Done	Jan 18, 2023	Feb 3, 2023
	ECSS-10	Iteration 1 - UML Class Diagram & Sequence Diagrams	Trong Nguyen	SYSC3303 Project...		DONE	Done	Jan 18, 2023	Feb 4, 2023
	ECSS-11	Verify JavaDoc & Meaningful Comments	Trong Nguyen	SYSC3303 Project...		DONE	Done	Jan 18, 2023	Feb 4, 2023
	ECSS-12	Verify Unit Testing for all Classes	Trong Nguyen	SYSC3303 Project...		IN PROGRESS	Unresolved	Jan 18, 2023	Feb 4, 2023
	ECSS-13	Initial Environment Setup	Zakaria Ismail	Zakaria Ismail		DONE	Done	Jan 20, 2023	Jan 29, 2023
	ECSS-14	Configure CI/CD Automated Testing	Unassigned	Zakaria Ismail		TO DO	Unresolved	Jan 21, 2023	Jan 24, 2023
	ECSS-15	Iteration 0 Results Processing	Trong Nguyen	Zakaria Ismail		DONE	Done	Jan 21, 2023	Feb 3, 2023
	ECSS-16	Configure Logging	Unassigned	Zakaria Ismail		TO DO	Unresolved	Jan 21, 2023	Jan 26, 2023
	ECSS-17	Input File Parser and Tests	Patrick Liu	Patrick Liu		DONE	Done	Jan 23, 2023	Feb 2, 2023
	ECSS-18	Fix/Setup Relative Filepath for input.txt file	Unassigned	Zakaria Ismail		DONE	Done	Jan 24, 2023	Jan 29, 2023
	ECSS-19	Setup config.properties file to select input file	Unassigned	Zakaria Ismail		TO DO	Unresolved	Jan 24, 2023	Jan 26, 2023
	ECSS-20	Unit test main.parser.Parser class	Patrick Liu	Zakaria Ismail		DONE	Done	Jan 24, 2023	Feb 3, 2023
	ECSS-21	Unit test main.java.dto.ElevatorRequest class	Patrick Liu	Zakaria Ismail		DONE	Done	Jan 24, 2023	Feb 3, 2023
	ECSS-22	README file	Trong Nguyen	Bobby Ngo		IN PROGRESS	Unresolved	Jan 25, 2023	Feb 3, 2023
	ECSS-23	Create DOCs folder and include UML files	Hussein El Mokdad	Hussein El Mokdad		DONE	Done	Jan 25, 2023	Feb 4, 2023
	ECSS-24	.gitignore for IntelliJ	Zakaria Ismail	Zakaria Ismail		DONE	Done	Jan 25, 2023	Jan 25, 2023
	ECSS-25	Setup custom exception class with centralized error messages and refactor exception throws	Unassigned	Zakaria Ismail		TO DO	Unresolved	Jan 26, 2023	Jan 26, 2023
	ECSS-26	Unit test main.java.floor.Floor class	Unassigned	Zakaria Ismail		DONE	Done	Jan 29, 2023	Feb 3, 2023
	ECSS-27	Integrate Iteration 1 Components	Unassigned	Zakaria Ismail		TO DO	Unresolved	Jan 29, 2023	Feb 3, 2023
	ECSS-28	Phase 1 Component Integration - Parser, Floor, Elevator, Scheduler	Zakaria Ismail	Zakaria Ismail		DONE	Done	Jan 29, 2023	Feb 2, 2023
	ECSS-30	Bi-directional Communication Implementation	Zakaria Ismail	Zakaria Ismail		DONE	Done	Feb 2, 2023	Feb 4, 2023
	ECSS-31	Iteration 2 setup	Unassigned	Trong Nguyen		TO DO	Unresolved	Feb 3, 2023	Feb 3, 2023
	ECSS-32	Verify that program can read the input file and pass the data back and forth	Zakaria Ismail	Zakaria Ismail		IN PROGRESS	Unresolved	Feb 3, 2023	Feb 4, 2023
	ECSS-33	Standardize string output to use String.format for variables instead of + concat	Zakaria Ismail	Zakaria Ismail		TO DO	Unresolved	Feb 3, 2023	Feb 4, 2023
	ECSS-34	Added some more content to the readme	Unassigned	Trong Nguyen		DONE	Done	Feb 4, 2023	Feb 4, 2023
	ECSS-35	Implement remaining test cases (scheduler, floor, and elevator)	Hussein El Mokdad	Hussein El Mokdad		IN PROGRESS	Unresolved	Feb 4, 2023	Feb 4, 2023
	ECSS-36	Documentation breakdown of the responsibilities of each team member for iteration 1	Trong Nguyen	Trong Nguyen		IN PROGRESS	Unresolved	Feb 4, 2023	Feb 4, 2023