Elevator Simulator System

SYSC 3303-A1, Team 3

April 12, 2023

Sarah Chow 101143033
Andre Hazim 101141843
Kyra Lothrop 101145872
Guy Morgenshtern 101151430
Bardia Parmoun 101143006

1.0 Breakdown of Responsibilities

1.1 Iteration 1

Name	ElevatorSimulator	ElevatorSimulatorTest
Andre Hazim	Elevator.java, Scheduler.java, ArrivedElevatorMessage.java	SimulatorTest.java
Bardia Parmoun	Buffer.java,Scheduler.java, Message.java	ElevatorTest.java, FloorTest.java, MockScheduler.java, SchedulerTest.java, SimulatorTest.java
Guy Morgenshtern	Floor.java, Simulator.java, Message.java, KillMessage.java, RequestElevatorMessage.java	BufferTest.java
Kyra Lothrop	Buffer.java, Elevator.java, ArrivedElevatorMessage.java, KillMessage.java	BufferTest.java
Sarah Chow	Floor.java, Simulator.java, RequestElevatorMessage.java	ElevatorTest.java, FloorTest.java, MockScheduler.java, SchedulerTest.java

Table 1: Summary of the team's contribution for iteration 1.

1.2 Iteration 2

Name	ElevatorSimulator	ElevatorSimulatorTest
Andre Hazim	Elevator.java, OpenDoorsMessage.java, MessageQueue.java	
Bardia Parmoun	MessageQueue.java, Scheduler.java, ElevatorController.java	ElevatorTest.java
Guy Morgenshtern	Elevator.java, OpenDoorsMessage.java, MessageQueue.java	
Kyra Lothrop	MessageQueue.java, Scheduler.java, ElevatorController.java	FloorTest.java, SchedulerTest.java
Sarah Chow	MessageQueue.java, Scheduler.java, ElevatorController.java	ElevatorTest.java, FloorTest.java, SchedulerTest.java

Table 2: Summary of the team's contribution for iteration 2.

1.3 Iteration 3

Name	ElevatorSimulator	ElevatorSimulatorTest
Andre Hazim	Implemented ClientRPC and updated the elevator to send UDP messages.	
Bardia Parmoun	Implemented ServerRPC and updated the scheduler to handle multiple elevators.	
Guy Morgenshtern	Implemented ClientRPC and updated the elevator to handle multiple requests at once.	
Kyra Lothrop	Updated the floor to send multiple requests.	Added JUnit tests for the scheduler.
Sarah Chow		Added jUnit tests for the floor and the elevator.

Table 3: Summary of the team's contribution for iteration 3.

1.4 Iteration 4

Name	ElevatorSimulator	ElevatorSimulatorTest
Andre Hazim	Updated the elevator to handle door interrupts and the elevator stuck cases.	Added regression and unit tests for the floor subsystem.
Bardia Parmoun	Updated the scheduler to handle different error cases and reroute the requests.	Added the elevator tests and implemented the MockServerRPC.
Guy Morgenshtern	Updated the elevator to handle door interrupts and the elevator stuck cases.	Added regression and unit tests for the floor subsystem.
Kyra Lothrop	Update the floor to handle sending multiple files with error cases.	Added JUnit tests for the scheduler.
Sarah Chow	Update the floor to handle sending multiple files with error cases.	

Table 4: Summary of the team's contribution for iteration 4.

1.5 Iteration 5

Name	ElevatorSimulator	ElevatorSimulatorTest	Demo
Andre Hazim	Developed the UI for the system.		Contributed to the project report.
Bardia Parmoun		Added performance tests to measure various aspects of the system.	Contributed to the project report.
Guy Morgenshtern	Developed the UI for the system.		Contributed to the project report.
Kyra Lothrop		Added performance tests to measure various aspects of the system.	Contributed to the project report.
Sarah Chow	Developed the UI for the system.		Contributed to the project report and prepared the demo video.

Table 5: Summary of the team's contribution for iteration 5.

2.0 Diagrams

2.1 UML Diagrams

2.1.1 Iteration 1

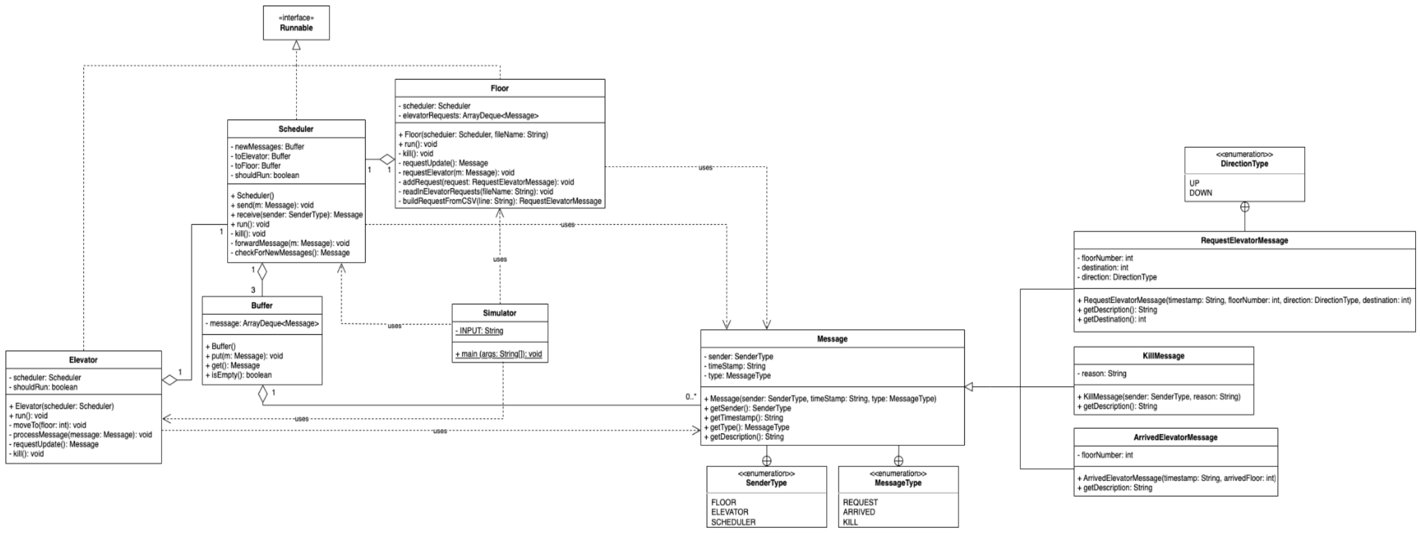


Figure 1: Complete UML diagram for the project after iteration 1.

2.1.2 Iteration 2

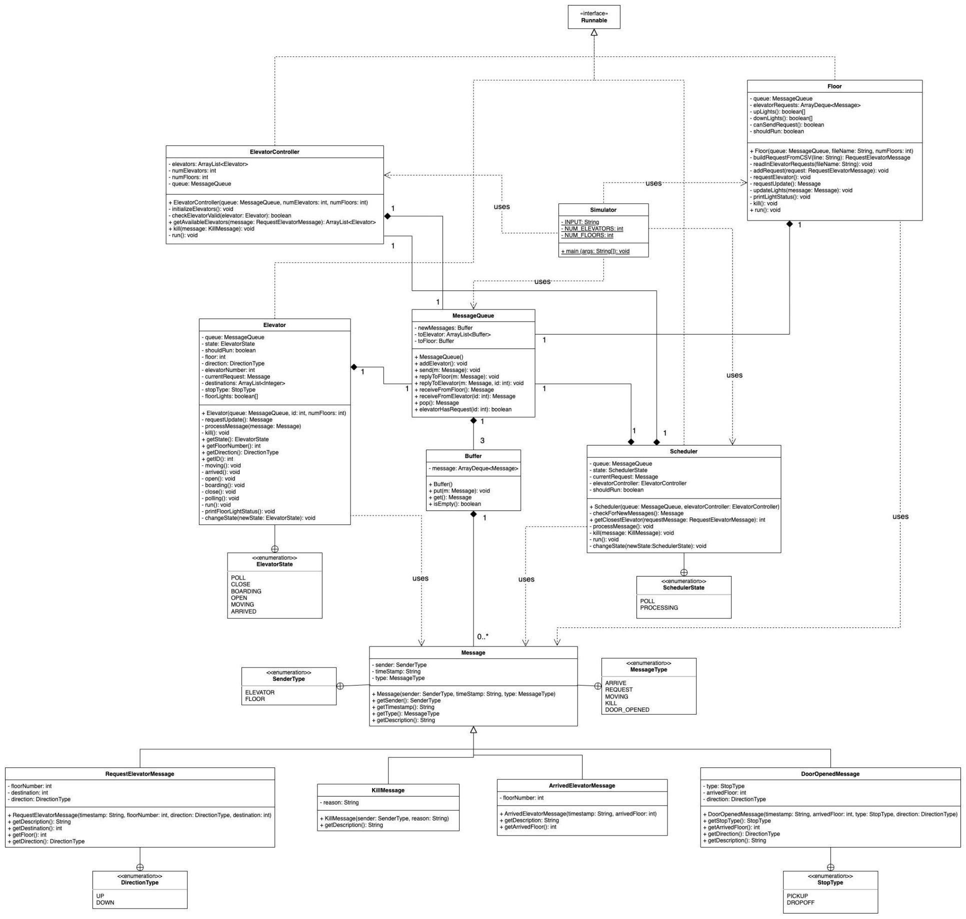


Figure 2: Complete UML diagram for the project after iteration 2.

2.1.3 Iteration 3

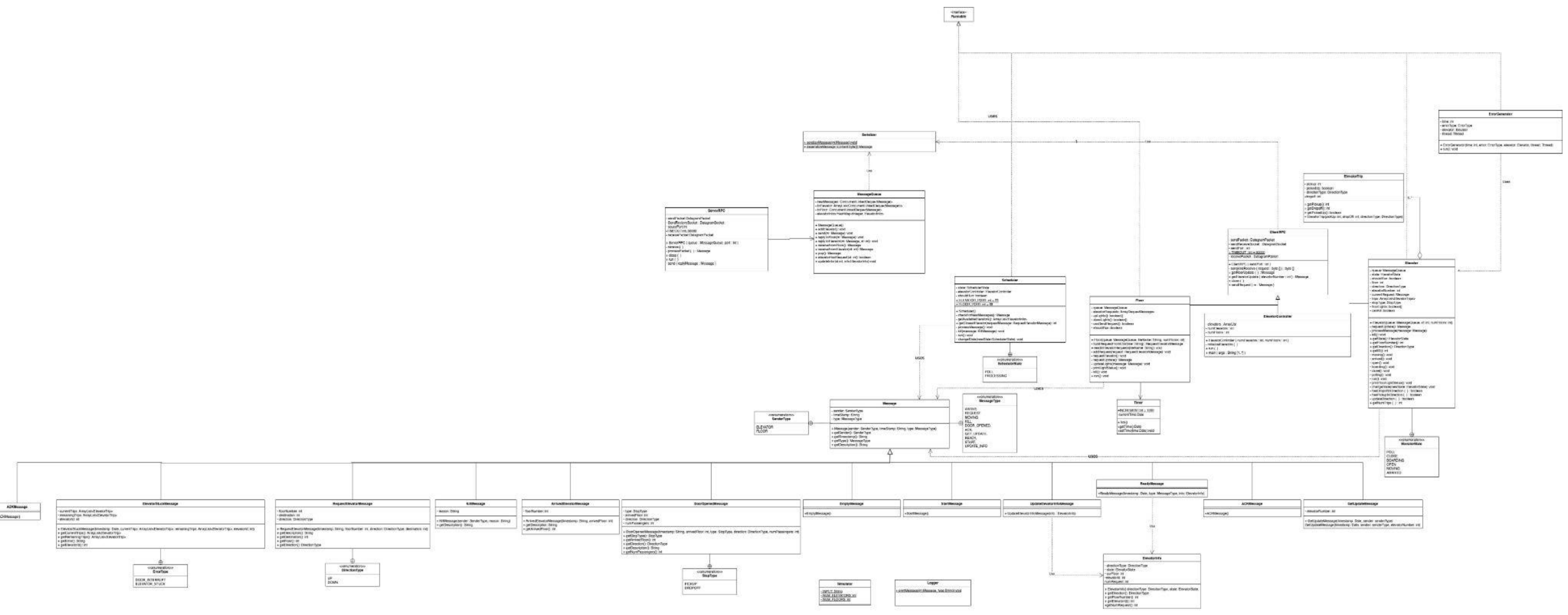


Figure 3: Complete UML diagram for the project after iteration 3.

2.1.4 Iteration 4

Figure 4: Complete UML diagram for the project after iteration 4.

2.1.5 Iteration 5

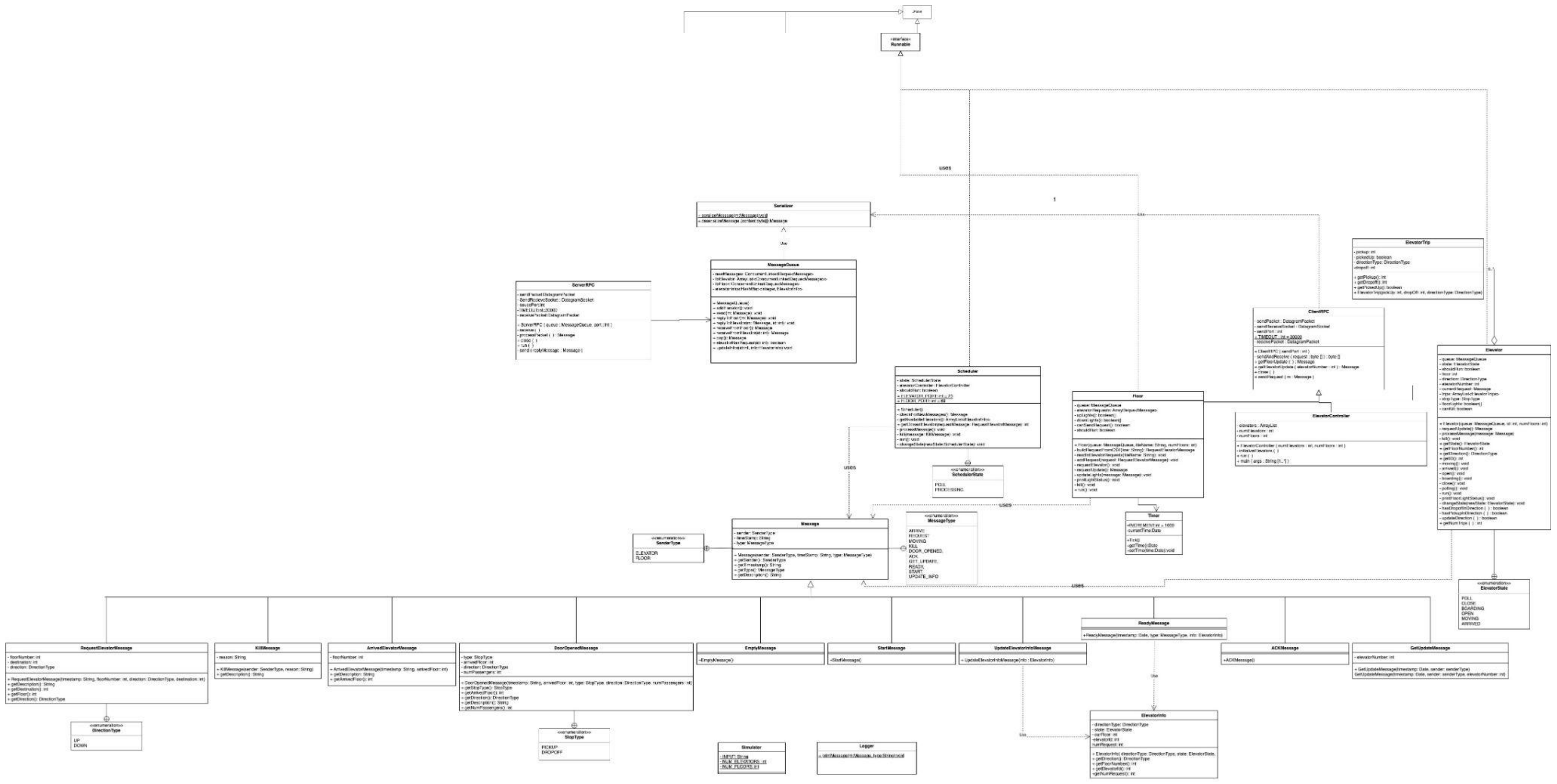


Figure 5: Complete UML diagram for the project after iteration 5.

2.2 State Machine Diagrams

2.2.1 Iteration 2

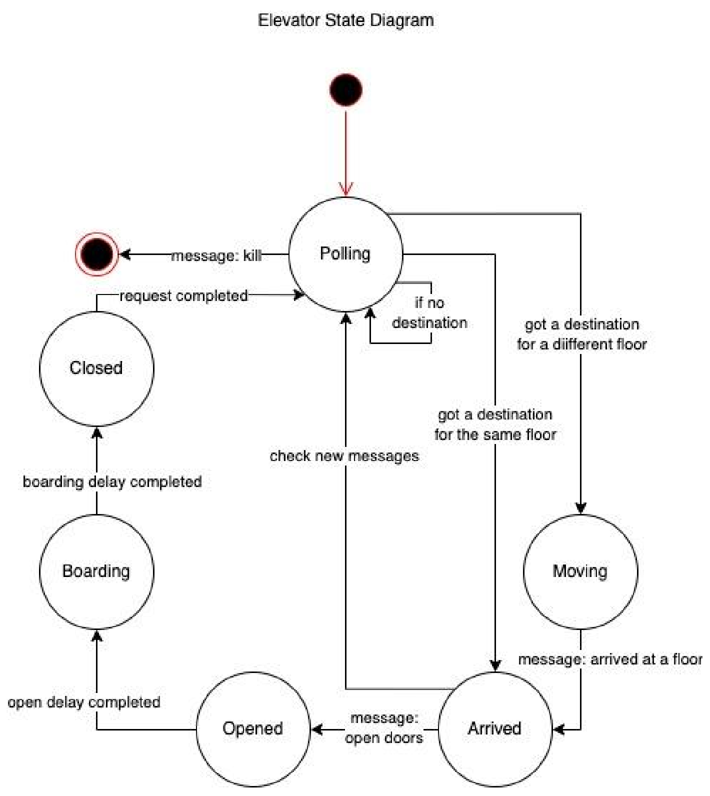


Figure 6: Complete Elevator state machine at iteration 2.

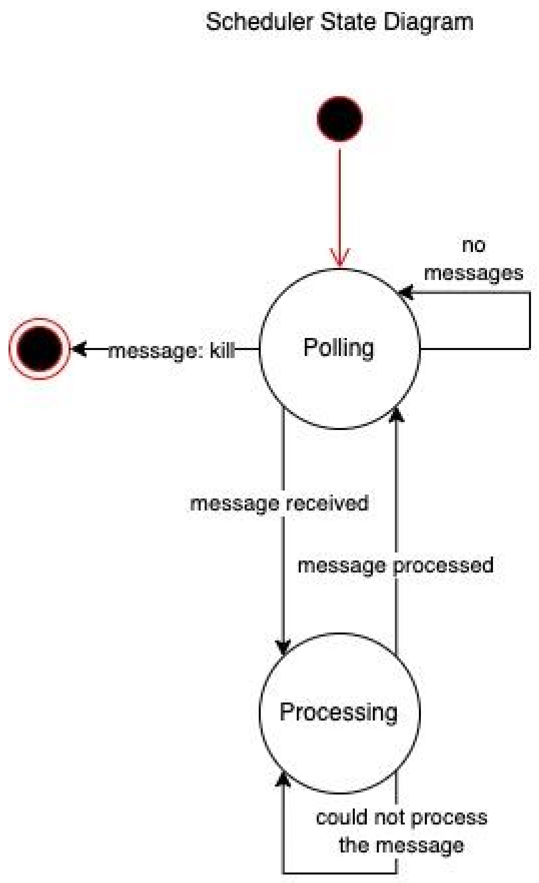


Figure 7: Complete Scheduler state machine at iteration 2.

2.2.2 Iteration 4

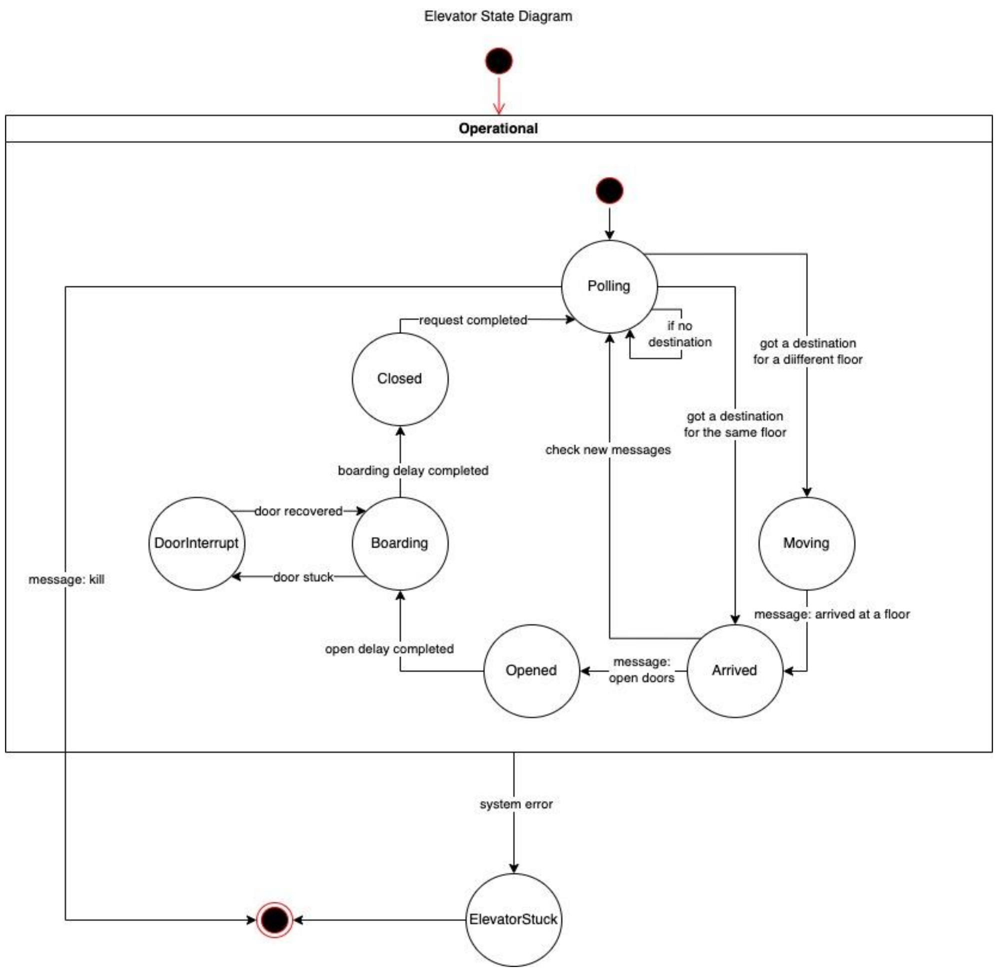


Figure 8: Complete Elevator state machine at iteration 4.

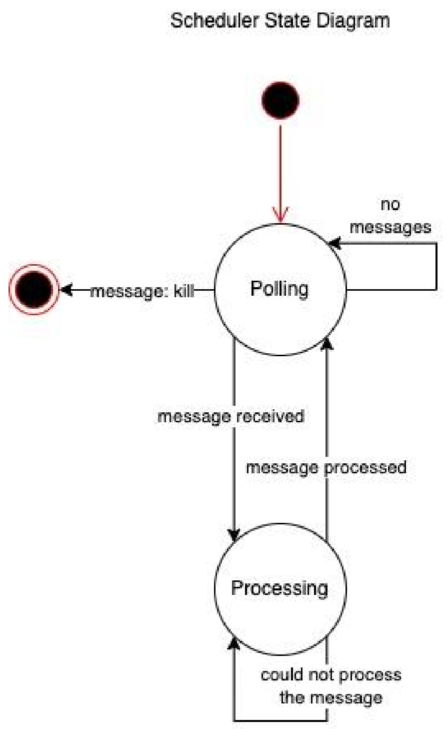


Figure 9: Complete Scheduler state machine at iteration 4.

2.3 Sequence Diagrams

2.3.1 Iteration 1

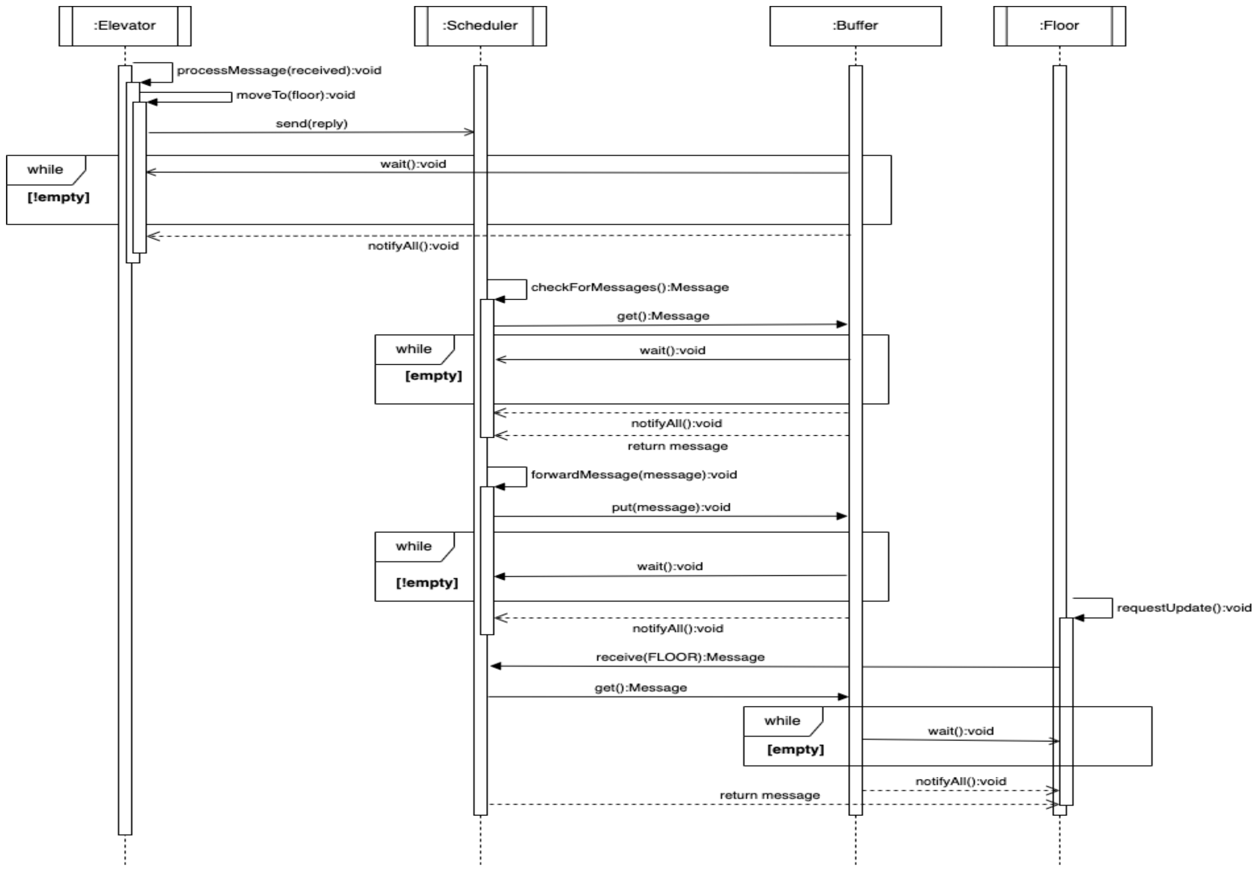


Figure 10: Sequence diagram demonstrating the elevator subsystem sending a message to the floor subsystem at iteration 1.

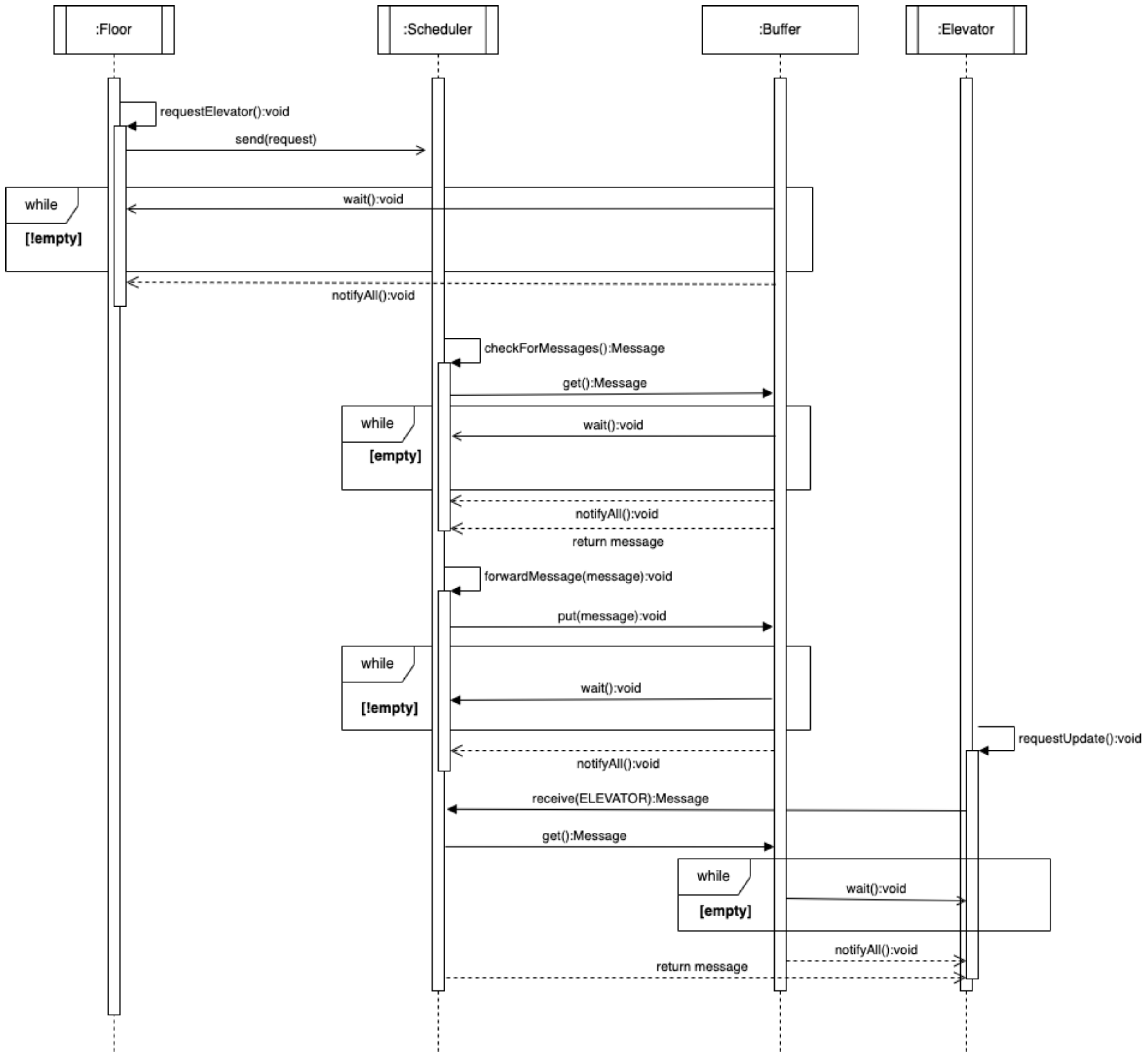


Figure 11: Sequence diagram demonstrating the floor subsystem sending a message to the elevator subsystem at iteration 1.

2.3.2 Iteration 2

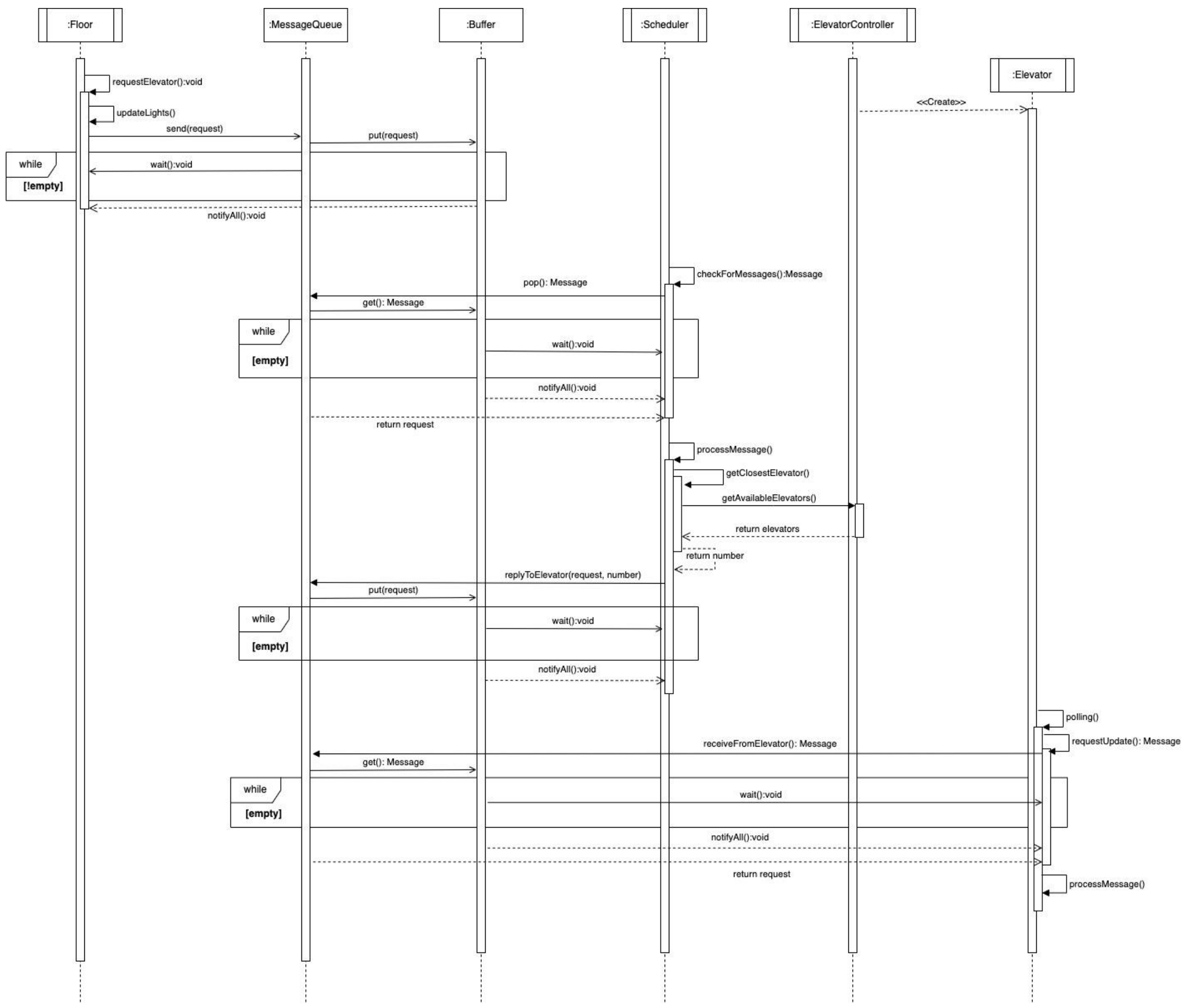


Figure 12: Sequence diagram demonstrating the floor subsystem sending a message to the elevator subsystem at iteration 2.

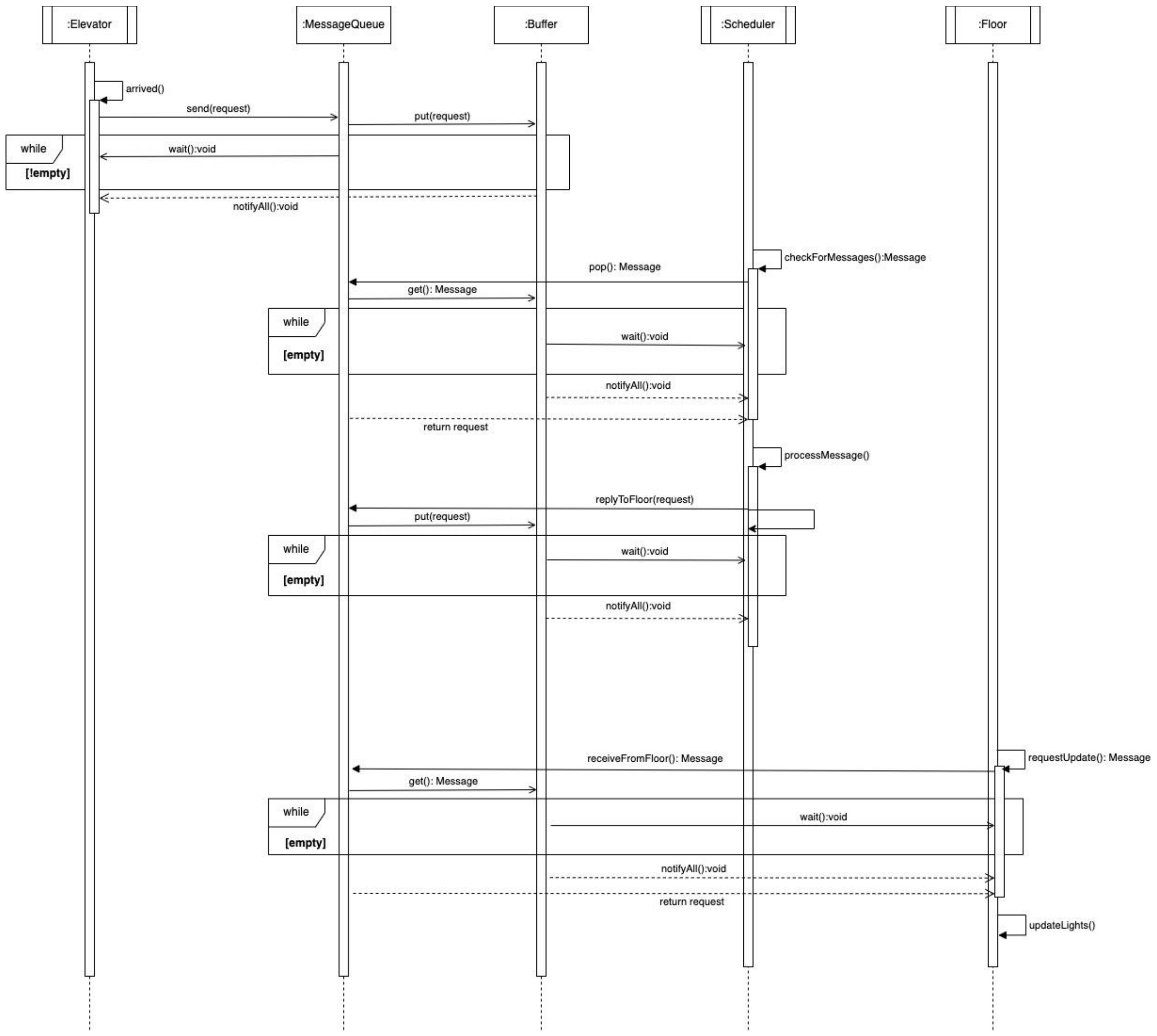


Figure 13: Sequence diagram demonstrating the floor subsystem sending a message to the elevator subsystem at iteration 2.

2.3.3 Iteration 3

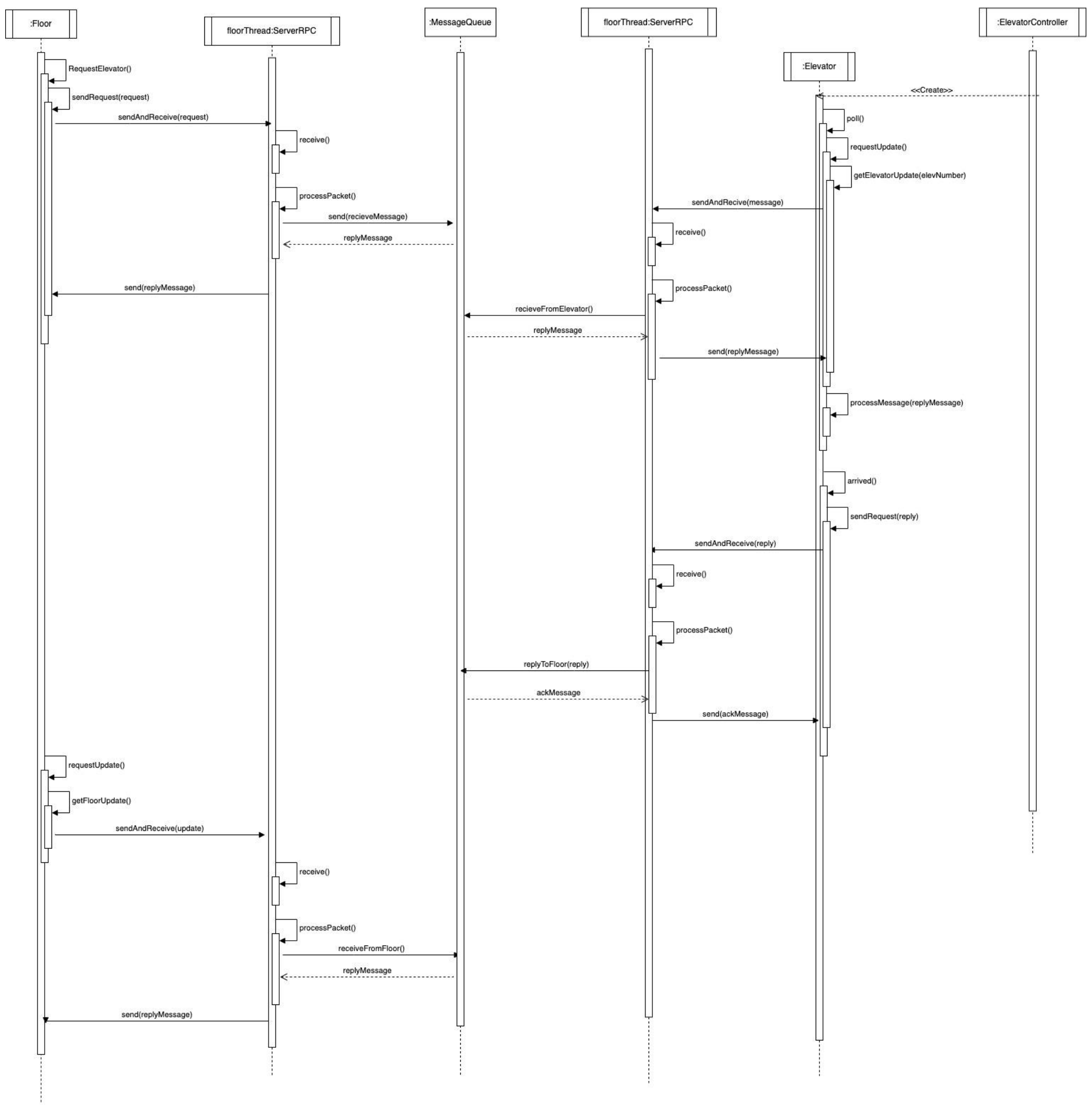


Figure 13: Sequence diagram demonstrating the elevator subsystem and the floor subsystem communicating with each other at iteration 3.

2.3.4 Iteration 4

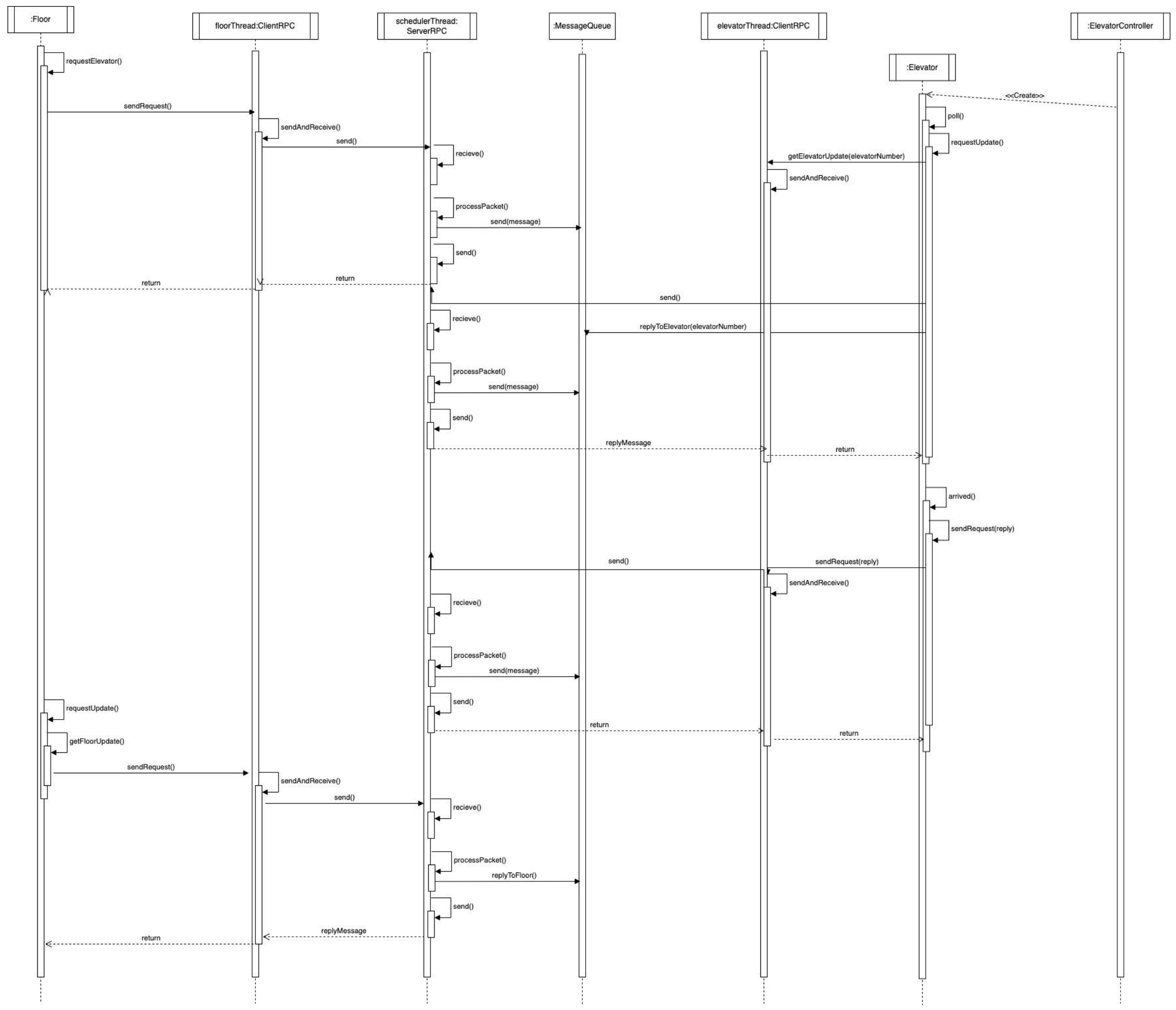
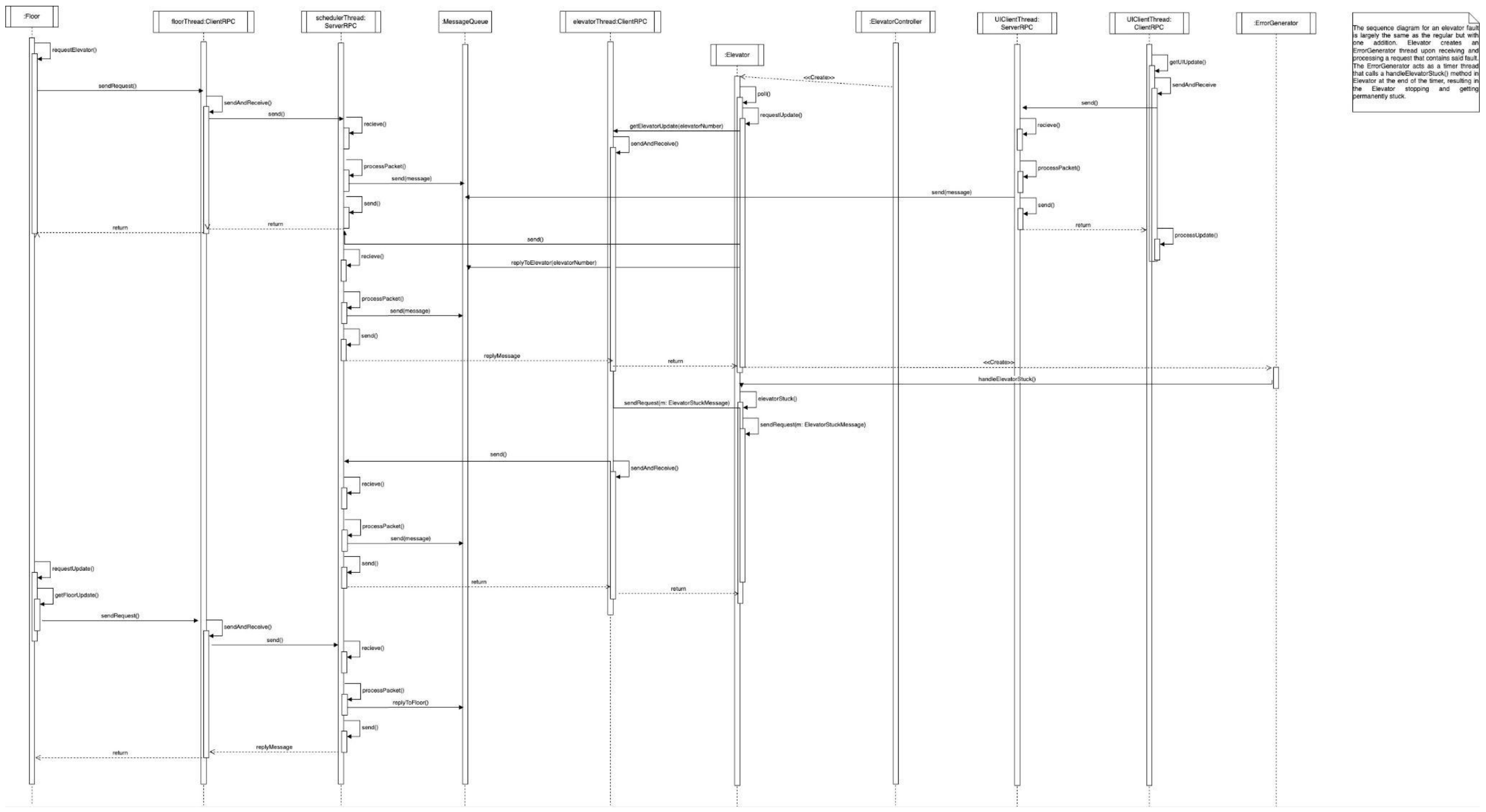


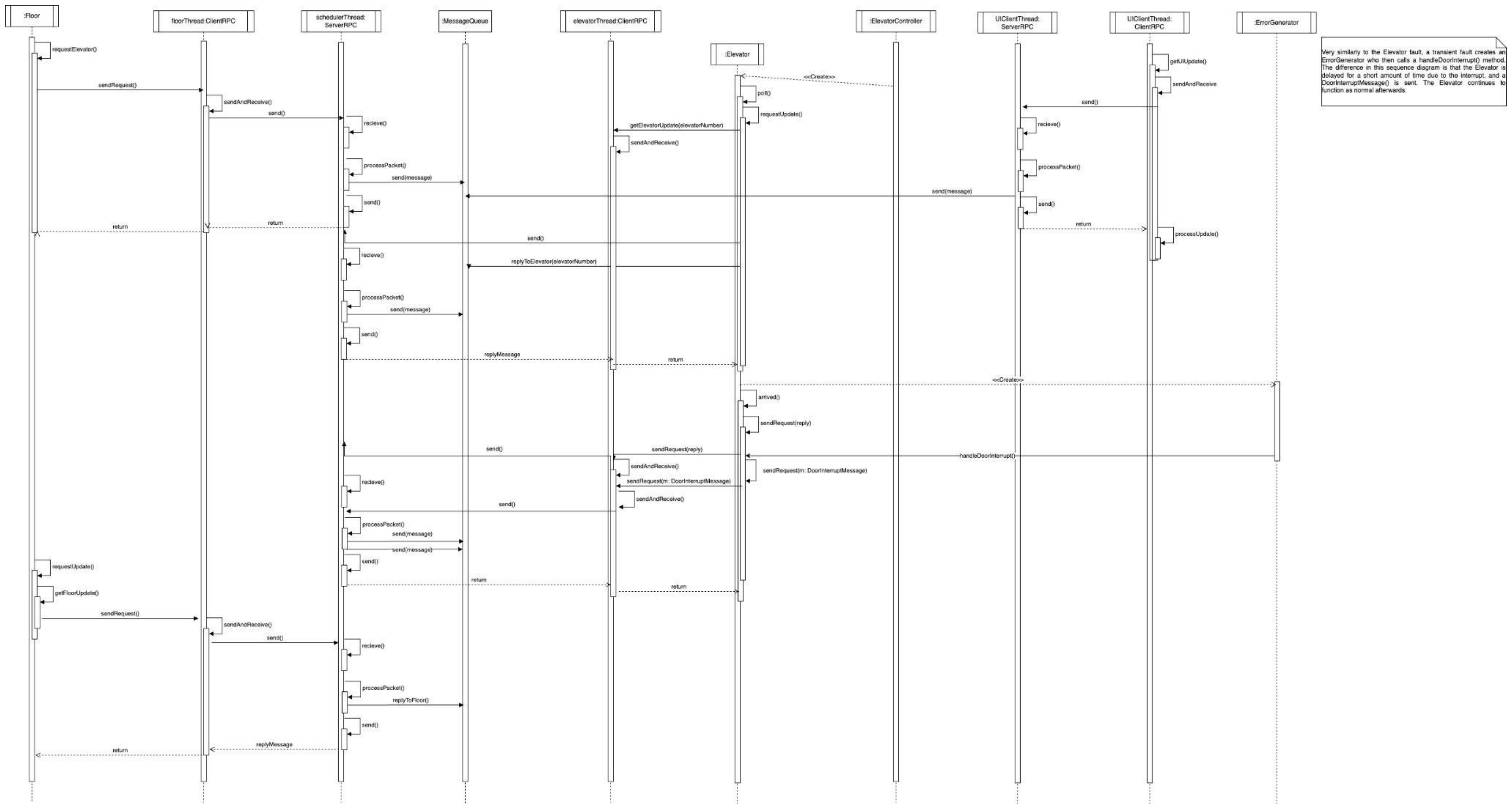
Figure 14: Sequence diagram demonstrating the elevator subsystem and the floor subsystem communicating with each other at iteration 4.

2.3.5 Iteration 5



The sequence diagram for an elevator fault is largely the same as the regular but with one addition. Elevator creates an ErrorGenerator thread upon receiving and processing a request that contains said fault. The ErrorGenerator acts as a timer thread that calls a handleElevatorStuck() method in Elevator at the end of the timer, resulting in the Elevator stopping and getting permanently stuck.

Figure 15: Sequence diagram demonstrating the elevator subsystem and the floor subsystem communicating with each other during a system fault at iteration 5.



Very similarly to the Elevator fault, a transient fault creates an ErrorGenerator who then calls a handleDoorInterrupt() method. The difference in this sequence diagram is that the Elevator is delayed for a short amount of time due to the interrupt, and a DoorInterruptMessage() is sent. The Elevator continues to function as normal afterwards.

Figure 16: Sequence diagram demonstrating the elevator subsystem and the floor subsystem communicating with each other during a transient fault at iteration 5.

2.4 Timing Diagrams

2.4.1 Iteration 4

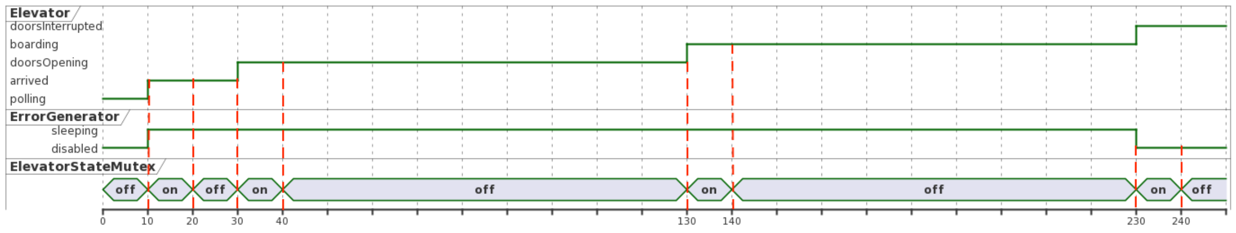


Figure 17: The timing diagram for the system in case of a transient fault.

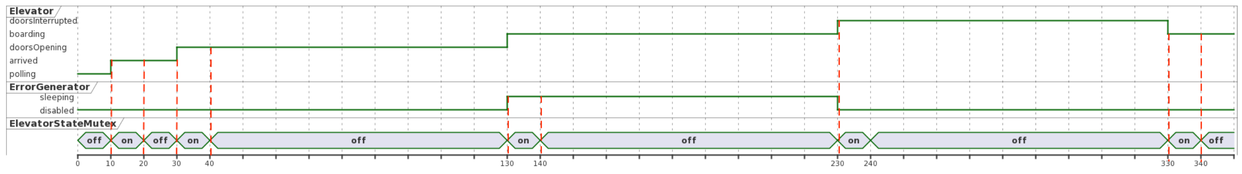


Figure 18: The timing diagram for the system in case of a system fault.

3.0 Set Up and Test Instructions

The elevator simulator program was made in eclipse and set up is required to run the simulator and tests properly.

3.1 Set Up Instructions

1. Unzip the submission file.
2. Navigate the eclipse IDE.
3. Navigate the File menu.
4. Open the "Open Project from File System.." option.
5. Select the root folder for the project "ElevatorSimulator".
6. Click the "Finish button".
7. Select the project folder.
8. Once the project folder is open navigate to the "./src/ElevatorSimulator/" and run these files in the following order.
    - Scheduler/Scheduler.java
    - Elevator/ElevatorController.java
    - UI/UIGenerator.java
    - Floor/Floor.java

To run the system with multiple devices:

1. Run the Scheduler.java on your machine to find your IP address.
2. On every other machine, update the PUBLIC_IP parameter of “Message/ServerRPC.java” file with the IP address that you obtained.
3. Update the main function for “UIGenerator”, “Floor”, and “ElevatorController” by changing their connection type to “ConnectionType.REMOTE”.
4. Start the subsystems on each machine with the same order as before.

3.2 Test Instructions

The ElevatorSimulator.Test package has been dedicated to testing. The testing framework that was used in this project is JUnit 5.
For each subsystem there is a dedicated test class to test them separately.

- ElevatorIntegrationTest: Tests the general flow for the elevator.
- ElevatorUnitTest: Unit Tests for the elevator.
- FloorTest: testing the floor subsystem alone.
- SchedulerTest: testing the scheduler subsystem alone.
- PerformanceUnitTest: measuring specific elevator events.
- PerformanceIntegrationTest: measuring the system as a whole.

Navigate to any of these classes and run them as a JUnit test.

4.0 Results of Measurements

In iteration 0, measurements from the elevators in Canal were gathered as a baseline for this project. In the last iteration, measurements from the elevator simulation were gathered and the two datasets were compared.

4.1 Measuring the times for a real elevator

Floor Number	**Door opening (s)**	**Boarding (s)**	**Door closing (s)**	**Going up (s)**	**Going down (s)**
1	2.53	5.87	4.05	6.71	-
2	1.94	6.09	3.90	7.05	6.53
3	2.77	5.95	4.05	7.06	6.91
4	2.20	4.91	4.02	7.55	7.46
5	2.06	6.07	3.93	7.39	7.38
6	2.12	6.02	4.03	7.58	7.45
7	1.73	6.13	4.15	-	7.08
Mean ($\overline{X}$)	2.193	5.863	4.019	7.223	7.135
Sample Variance (S^2)	0.125	0.184	0.007	0.116	0.137
Standard Deviation (S)	0.353	0.429	0.083	0.341	0.370
Confidence Interval	1.87 <x< 2.19	5.47 <x< 5.86	3.94 <x<4.02	6.91 <x< 7.22	6.79 <x< 7.14

Table 6: Measurements captured from the Canal elevator at iteration 0.

To measure the elevator doors opening, the timer would start as soon as the doors opened until they reached the fully open position. Measuring the doors idling, the timer started when the doors fully opened and stopped when the doors began to close. To get the time of doors closing, the timer would start when the doors stopped idling and would stop when the doors fully closed. Measuring the time it took for the elevator to go up started when the timer felt the movement of the elevator begin. The timer would stop when the movement of the elevator stopped. The same measurement technique was used for going down as well. No open door or close door buttons were used during the timing process. To measure the acceleration, we measured the elevator time for multiple floors. The maximum acceleration occurred between 2 floors and the maximum speed occurred between 6 floors. We also measured the elevator boarding time based on the number of people moving in/out the elevator. With 92% accuracy, these results can be represented in the following equation: $0.76x + 5.23$ where x is the number of people moving in/out. To measure the extra delay added by the elevator after being interrupted another set of measurements were conducted. After being interrupted the elevator added between 2-3s of delay to idle time.

To ensure the accuracy of the measurements, the data was collected individually for each component to measure each step of the elevator data. This included the doors opening, idle time, door closing, going up and going down. Additionally, the data was considered for multiple floors of the elevator for additional data points. To ensure consistency, all the elevator idle times for the 7 floors were measured without anyone going in/out the elevator. Additional measurements for the idle time were conducted based on varying numbers of passengers entering or leaving the elevator. The time for going up and going down was measured to ensure the time was not impacted by gravity and the data for multiple floor travel times were collected.

4.2 Measuring the times for the simulated elevator

4.2.1 Measuring important events for the elevator

Floor Number	**Door opening (s)**	**Boarding (s)**	**Door closing (s)**	**Going up (s)**	**Going down (s)**
1	3.216	5.986	3.218	7.293	-
2	3.218	5.989	3.217	7.298	7.294
3	3.216	5.986	3.216	7.294	7.293
4	3.218	5.981	3.218	7.289	7.291
5	3.216	5.978	3.217	7.289	7.289
6	3.211	5.986	3.215	7.290	7.294
7	3.213	5.984	3.216	-	7.294
Mean ($\overline{X}$)	3.215	5.984	3.217	7.292	7.293
Variance (S^2)	0.00	0.000	0.00	0.000	0.000
Std. Dev. (S)	0.000	0.000	0.000	0.004	0.002
Conf. Interval	3.213 <x< 3.217	5.981 <x< 5.987	3.216 <x<3.218	7.289 <x< 7.295	7.291 <x< 7.294

Table 7: Measurements captured from the Elevator Simulator.

These values were all captured through various unit tests located under the “PerformanceTest” package of the “ElevatorSimulatorTest”. The goal of these tests was to replicate the measurements that were captured in iteration 0 of the project. Each test started the simulator with 7 floors and pressed all the elevator buttons on the way up and down and measured the time it took for the elevator to stay in a given state. The elevator itself was given the values that were collected in iteration 0 for its actions. In other words the following variables were set:

Variable	**Value (in milliseconds)**
MOVE_DELAY	(7223 + 7135) / 2 = 7179
DOOR_DELAY	(4019 + 2193) / 2 = 3106
BOARDING_DELAY	5863
DOOR_INTERRUPT_DELAY	(2 + 3) / 2 = 2500

Table 8: Updating the parameters of the simulator based on the data captured in iteration 0.

As previously shown the measured values are very close to the ideal values that were set through these variables. All measurements had a variance of 0.000 meaning the values stayed consistent throughout the elevator’s run.

4.2.2 Measuring the overall performance of the system

To measure the overall performance of the system, a set of integration tests were created under the performance package. These tests measured the time it took the system to handle various requests.

Test Number	**# of Requests**	**# of Elevators**	**Has an error**	**Time (in ms)**	**Time per Requests**
1	1	1	False	39265	39265 / 1 = 39265 ms
2	6	1	False	192150	192150 / 6 = 32025 ms
3	6	2	False	114648	114648 / 6 = 19108 ms
4	6	2	True	148933	148933 / 6 = 24822 ms
5	21	4	False	453546	453546 / 21 = 21597 ms
6	21	4	True	714937	714937 / 21 = 34044 ms

Table 9: The measurements obtained after running the simulator on a full set of requests.

As shown in the table, the system was tested under various circumstances. The tests yielded a relatively consistent result with the average time per request ranging from 19 to 40 seconds depending on the circumstances.

The first test measured the performance of the elevator with a single request that spans multiple floors. The second test measured the performance of the elevator with multiple elevators that can be grouped together. The third and fourth tests tested the same against requests against 2 elevators to see if there are any performance improvements. As shown, test 3 showed a substantial improvement. Test 4 also simulated a few transient and system faults to see their effect on the overall performance of the system. Although the performance was affected as the result of this test, it still remained within a reasonable range. Tests 5 and 6 measured the performance of the elevator against many requests that all came at similar times. These simulated the system under stress. Both of these tests were run against 4 elevators but they both still remained in a reasonable time.

Conclusions: Since the elevator algorithm is designed in a way to group multiple requests, it can be seen that it is more beneficial for the system to handle multiple requests at once since on average that will lower the time to handle each request. It can also be soon that the error cases (those that stop the elevators) increase the time per request substantially since they remove a functioning elevator from the system. Finally, it can also be seen that even under the stress test the system managed to still handle the requests within a reasonable time indicating that the scheduling algorithm is expandable.

5.0 Reflections on Design

This project consisted of several design changes throughout the iterations to make the simulator as efficient and clear as possible.

5.1 Aspects We Enjoyed

We feel that our UDP implementation was designed well. By separating the client and server UDP into their own classes called ClientRPC and ServerRPC respectively, we were able to keep all the algorithm and logical components of Elevator, Floor, Scheduler, and UI isolated from the passing of data. Floor, UI, and Elevator all extend ClientRPC, and a ServerRPC thread is created for each. The only thing left to do is to add a global variable for the port of each client.

Our tests are very exhaustive and test both individual functions within classes and integration tests of the full system. The design of our tests was very helpful in identifying bugs and also in incremental development.
We are also happy with our scheduler algorithm. Using a CScan algorithm as the base ensures that almost all the requests being sent to the elevator are “good” requests, meaning they are on the way of the elevator and the trip is in the same direction. Improvements were made to the algorithm by making an effort to balance requests between elevators so that no one elevator is swamped. This resulted in an algorithm that ensures no starving and a solid throughput of requests.

5.2 Aspects To Be Redesigned

An aspect to be redesigned is our implementation for the state machines in the elevator and the scheduler. Currently, we implement our state machine using a variation of the table method. The table method increases the cyclomatic complexity since there are many branching if statements which correspond to the states in our diagram. With thoughts of reimplementation, the state pattern would be a much more effective method in lowering the cyclomatic complexity and allow for more states to be added with ease. The state pattern would help get rid of flags and replace them with more civilized classes.

Another redesign consideration is with the scheduling algorithm. The problem with the scheduler is multiple people request the same floor and only certain people get picked. Depending on the time of the request some of them will get assigned to an elevator and some will get assigned to another. For the redesign, the scheduler should be able to allow for all the people that requested the floor to use the same elevator.

6.0 Test Files

All the test files are located in the attached java files under: “src/ElevatorSimulatorTest”

The JUnit tests consists of the following files:

- Package ElevatorSimulatorTest.ElevatorTest
    - ElevatorIntegrationTest.java
        - The elevator integration test is the test for the general flow for the elevator.
    - ElevatorUnitTest.java
        - The elevator unit test is the test for individual functionalities of the elevator.

- Package ElevatorSimulatorTest.FloorTest
    - FloorParseTest.java
        - The floor parse test is the unit tests for the floor subsystem that handles tests requests with stuck faults and interrupt faults.
    - FloorTest.java
        - The floor test is the unit tests for the floor subsystem.
- Package ElevatorSimulatorTest.PerformanceTest
    - PerformanceIntergrationTest.java
        - Integration tests to measure the performance of the entire project when running various test files.
    - PerformanceUnitTest.java
        - Unit tests to replicate the behaviour of the elevator and measure the time it takes for the elevator to stay at a given state.
- Package ElevatorSimulatorTest.SchedulerTest
    - SchedulerTest.java: Unit tests to validate the behaviour of the scheduler is as expected. It also verifies the scheduling algorithm works as expected.

7.0 Code

The source code for the project is located under attached files under: “src/ElevatorSimulator”

- Package ElevatorSimulator.Elevator:
    - The elevator package keeps track of all the files that are related to the elevator such as the elevator states and elevator trips.
- Package ElevatorSimulator.Scheduler:
    - The scheduler package keeps track of the scheduler and its states.
- Package ElevatorSimulator.Messages:
    - The messages package keeps track of the different messages that are being sent between the different components.
- Package ElevatorSimulator.Messaging:
    - The messaging package keeps track of all the files related to connecting the components together. This class includes the files like ServerRPC and ClientRPC that are in charge of implementing the UDP functionality.
- Package ElevatorSimulator.UI:
    - The UI package is in charge of maintaining all the code related to the UI.