**SYSC 3303 Term Project Report - L4 Group 6**

**GitHub:** https://github.com/AshtonMohns/elevatorTiming

**Demo Video:** https://www.youtube.com/watch?v=iBuNZKiITTE&feature=youtu.be

**Recipient:** TA for SYSC 3303

**Client:** Roy Franks of Department of Systems and Computer Engineering Class of SYSC 3303

**Authors:**

1. Ashton Mohns: 101074479
2. Edmond Chow: 100883365
3. Jyotsna Mahesh: 101084851
4. Asifur Rahman: 101069183
5. Md Aiman Sharif: 101062765

**Date:** April 14, 2021

Shape, polygon

Description automatically generated

Table of Contents

[**1.0** **BREAKDOWN OF RESPONSIBILITIES:** 3](#_Toc69327907)

[**1.1** **ITERATION 1:** 3](#_Toc69327908)

[**1.2** **ITERATION 2:** 3](#_Toc69327909)

[**1.3** **ITERATION 3:** 4](#_Toc69327910)

[**1.4** **ITERATION 4:** 4](#_Toc69327911)

[**1.5** **ITERATION 5:** 5](#_Toc69327912)

[**2.0** **DIAGRAMS:** 6](#_Toc69327913)

[**2.1** **UML Class Diagram:** 6](#_Toc69327914)

[**2.2** **State Machine Diagram:** 9](#_Toc69327915)

[**2.3** **Sequence Diagram:** 10](#_Toc69327916)

[**2.4** **Timing Diagram:** 11](#_Toc69327917)

[**3.0** **DETAILED SET-UP AND TEST INSTRUCTIONS:** 12](#_Toc69327918)

[**3.1** **SETUP AND FORMATS:** 12](#_Toc69327919)

[**3.1.1** **SETUP INSTRUCTIONS:** 12](#_Toc69327920)

[**3.1.2** **Input.txt format:** 12](#_Toc69327921)

[**3.1.3** **Using Constants.java to modify behavior:** 13](#_Toc69327922)

[**3.2** **TEST INSTRUCTIONS** 13](#_Toc69327923)

[**3.2.1** **Unit Testing:** 13](#_Toc69327924)

[**3.2.2** **Integration Testing:** 13](#_Toc69327925)

[**3.2.3** **Acceptance Testing:** 14](#_Toc69327926)

[**4.0** **RESULTS FROM MEASUREMENT:** 16](#_Toc69327927)

[**5.0** **REFLECTION ON DESIGN:** 16](#_Toc69327928)

# **BREAKDOWN OF RESPONSIBILITIES:**

## **ITERATION 1:**

**Ashton Mohns:**

* Floor Subsystem and their related classes

**Edmond Chow:**

* Scheduler and Elevator Subsystem and their related classes

**Jyotsna Mahesh:**

* UML and Sequence Documentation

**Asifur Rahman:**

* UML, README and Testing

**Md Aiman Sharif:**

* Testing framework

## **ITERATION 2:**

**Ashton Mohns:**

* Thread3 Class Creation
* Map Class Creation

**Edmond Chow:**

* Thread2 Class Creation
* Map Class Creation
* ReqBox implementation for Schedular Subsystem

**Jyotsna Mahesh:**

* Floor Subsystem Creation
* Thread1 Class Creation
* ReqBox implementation for Floor Subsystem

**Asifur Rahman:**

* ElevatorSubSystem Creation
* Updating README file

**Md Aiman Sharif:**

* Drawing UML Class Diagram
* JUnit Testing

## **ITERATION 3:**

**Ashton Mohns:**

* Drawing Sequence Diagram
* Update and refactor code for Floor subsystem

**Edmond Chow:**

* JUnit testing
* Update threads and code for Elevator subsystem

**Jyotsna Mahesh:**

* Remodeling Floor Subsystem
* Drawing State Machines Diagram

**Asifur Rahman:**

* Remodeling of the ElevatorSubSystem
* Updating README file

**Md Aiman Sharif:**

* Drawing UML Class Diagram
* Updating Javadoc
* Updating Floor Manager’s Implementation

## **ITERATION 4:**

**Ashton Mohns:**

* Implementing an intermediate host for forwarding door closes from floor to elevator subsystem
* Updating tests with the new request packets
* Helping with state diagrams

**Edmond Chow:**

* Debug testing
* Elevator Rebuild, Elevator Fault Handling
* Debug setting for console output

**Jyotsna Mahesh:**

* Upgrade file reader and Direction Enum
* Create Error Enum
* Update Print statements
* Documentation (sequence and state diagrams)

**Asifur Rahman:**

* Implementing ElevatorThread4 class for handling door close/open functionality
* Upgrading ElevatorBox for error handling
* Updating README file
* Modifying RequestHandler to meet the current criteria

**Md Aiman Sharif:**

* Floor door handling
* Updating packets sent by the floor subsystem to handle errors
* Updating Class diagrams

## **ITERATION 5:**

**Ashton Mohns:**

* GUI
* Acceptance testing
* Reflections

**Edmond Chow:**

* Documentation
* Integration Testing
* Refactor code, remove bugs
* Error handling unit tests

**Jyotsna Mahesh:**

* Created more Test Files
* Diagrams: Sequence, State Machine and timing
* Worked on the Report

**Asifur Rahman:**

* Creating and maintaining the report
* Diagrams: UML, Sequence
* Updating README file

**Md Aiman Sharif:**

* Designed the GUI- code
* Implemented GUI endpoints
* User Interface testing
* Worked on report

# **DIAGRAMS:**

## **UML Class Diagram:**

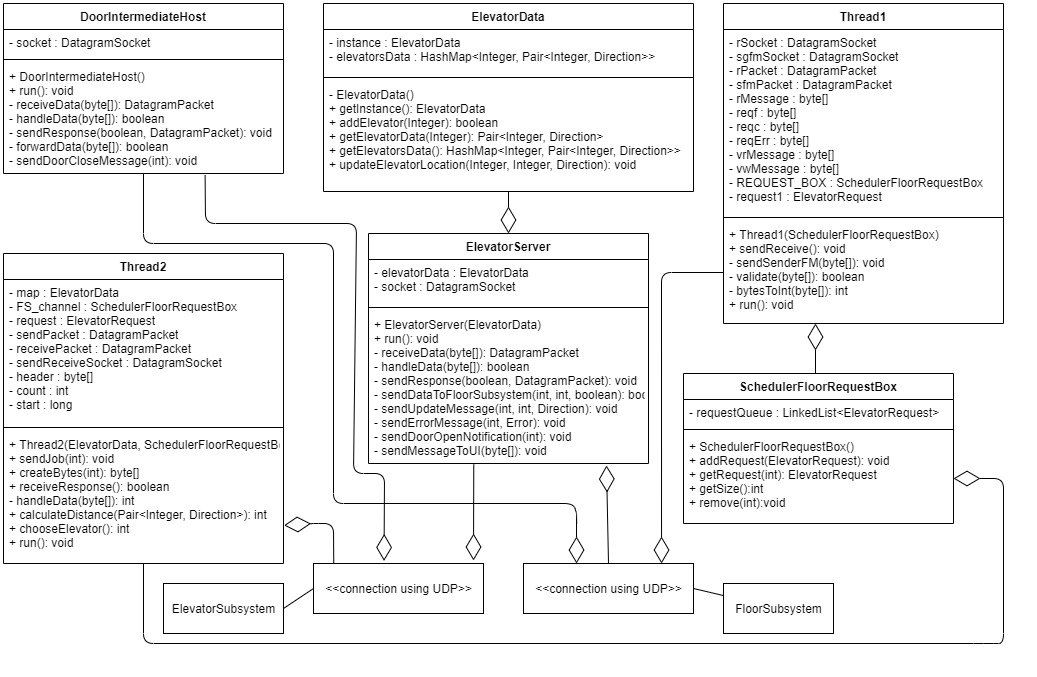
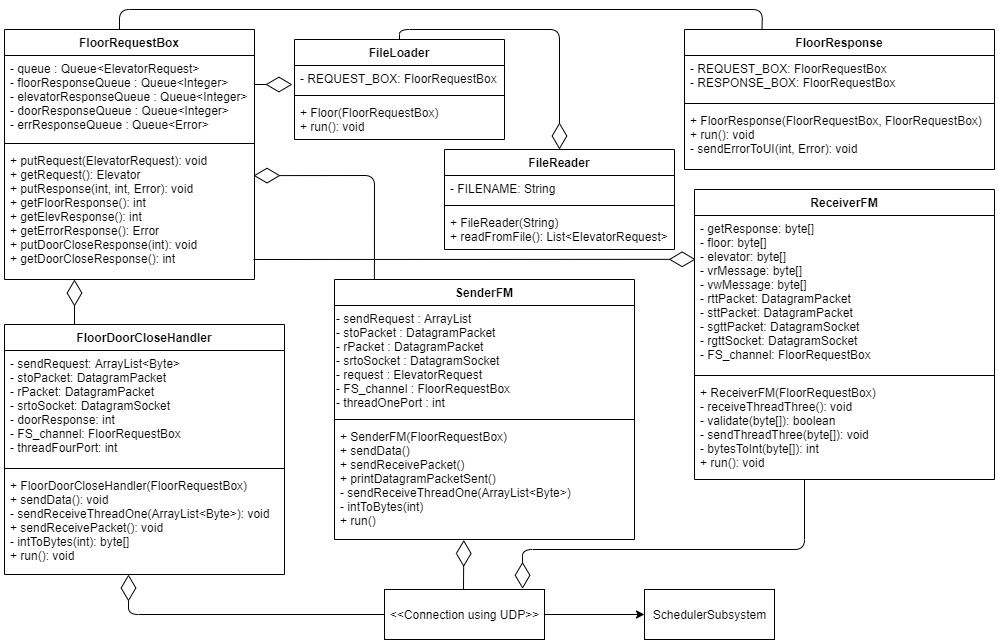
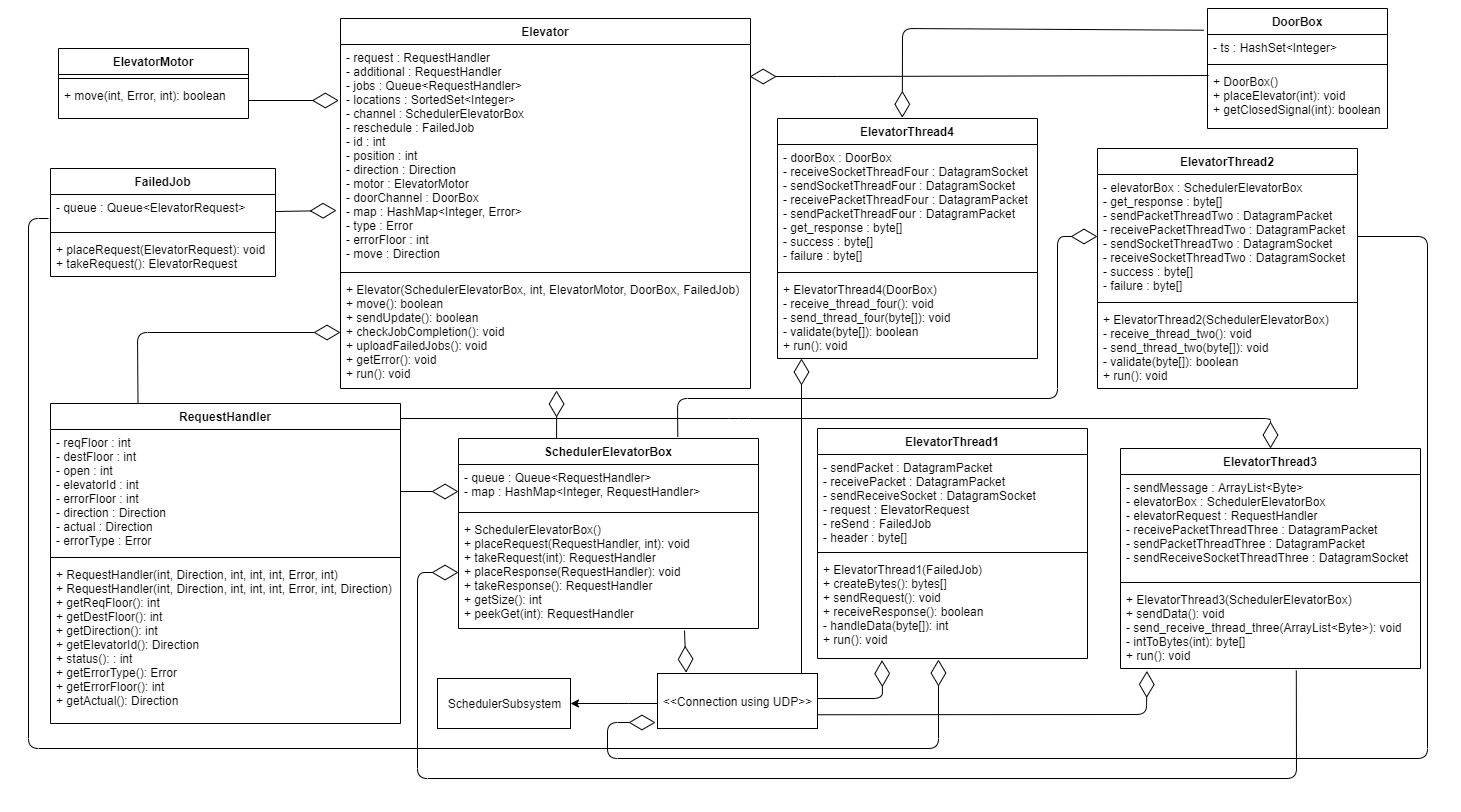


Figure 1: UML Class Diagram for Scheduler Subsystem

Figure 2: UML Class Diagram for the Floor Subsystem

Figure 3: UML Class Diagram for Elevator Subsystem

## **State Machine Diagram:**

Diagram

Description automatically generated

Figure 4: State Machine Diagram for Scheduler Subsystem

## A picture containing calendar Description automatically generated**Sequence Diagram:**

Figure 5: Sequence Diagram showing error handling in all three subsystems

## **Diagram, schematic Description automatically generatedTiming Diagram:**

Figure 6: Timing Diagram for Scheduler Threads

[Was informed by the prof that exact timings were not required]

# **DETAILED SET-UP AND TEST INSTRUCTIONS:**

## **SETUP AND FORMATS:**

## **SETUP INSTRUCTIONS:**

1. **Import the project archive file into Eclipse**
2. **From Eclipse**

* Select "File"
* Select "Import"
* Select "General"
* Select "Projects from File System or Archive File" under "General"
* Select "Archive..."
* Locate "L4G6\_Milestone5.zip" as Import Source
* Click "Finish"

1. **From within the project open "ElevatorTiming"**

Before starting the code running, if you wish to change the times for elevators to move and/or doors to open, you can change the values in the sharedObjects/Constants.java file. Both values are time in milliseconds. Please do not reduce too close to 0, as there are potential dropped packets in this case. If you encounter any errors while running the code, please ‘clean’ and ‘refresh’ the Project. In order to run the program, you can follow one of below listed set of steps:

* 1. The main files for each subsystem can be run individually and have the outputs be shown on separate consoles.

Run the files in this order:

* 1. Run the UserInterfaceMain.java file from the main package.
  2. Run the SchedulerMain.java from the main package.
  3. Run the MainEM.java file from the main package.
  4. Run the MainFM.java file from the main package.
  5. The main files can be run simultaneously with all their outputs being shown on the same console.

Run the Main.java file from the main package.

## **Input.txt format:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Time** | **RequestedFloor** | **Direction** | **DestinationFloor** | **ErrorType** | **ErrorFloor** |

**Time:** when the request was sent

**RequestedFloor:** use up to 22 floors

**Direction: “**Up” or “Down” inputs only [Case does not matter]

**DestinationFLoor:** use up to 22 floors

**ErrorType: “**NONE”, “ARRIVAL”, “TIME”, “DOOR” inputs only [Case does not matter]

**ErrorFloor:** floor where you want the error to happen

## **Using Constants.java to modify behavior:**

A wide variety of factors in behavior of the system can be edited through manipulation of variables in the constants.java file as listed below

* **Boolean debug:** when debug is false, the system will output less information to the consoles to increase readability in general cases. When debug is set to true, more information will be outputted to consoles for troubleshooting.
* **Int TIMEOUT\_MILLIS**: sets how long is required for a socket time out to occur, not extremely low time out values will cause system to fail
* **Int elevator:** change to adjust the number of elevators that will be running in the system
* **Int MOVE\_TIME:** This value is used to adjust how long it will take an elevator to move between floors
* **Int DOOR\_TIME:** This value is used to control how long a door will stay open when they open
* **String FILENAME:** This will be the name of the input file read by the system

**Int FLOOR\_PORT, Int ELEVATOR\_SERVER\_PORT,** and **Int HEADER** should not be edited as they are constants used by the system for regular operations.

## **TEST INSTRUCTIONS**

## **Unit Testing:**

* The **ElevatorThreadTests**, **ElevatorServerTest**, **SenderFMTest**, **ReceiverFMTest**, **Thread1Test** are to test the threads functionalities individually.
* The **FloorRequestBoxTest**, **SchedulerElevatorBoxTest**, **SchedulerFloorRequestBoxTest**, **DoorBoxTest** are to test if the functionalities of the box classes are working as intended.
* **ElevatorFaultTest** checks if the system can handle timer and arrival sensor faults.
* **FloorResponseTest** Checks if the system can properly handle and clear door errors
* **ElevatorMotorTest** ensures that the movement of elevator is performed as expected.
* **ElevatorRequestTest** checks if the elevator request objects handle the data properly.
* **FailedJobTest** checks if the failed process can be reprocessed by the scheduler.
* **FloorDoorCloseHandlerTest** checks if the thread sending the packets properly in the right manner.

Upon writing the unit tests, minor bugs were found and patched in the code. These included inconsistencies in handling requests and slower response times than expected. We have since patched the bugs and the unit tests succeed.

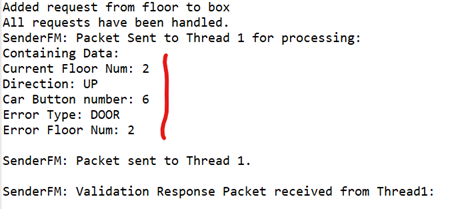
## **Integration Testing:**

No actual testing file was made to monitor integration testing. In order to test the integration between the various subsystem components the following steps were done:

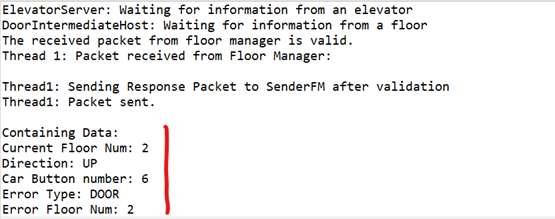
1. In the Constants.java file, set the Boolean variable debug to true, to enable
2. Run the program as normal.
3. At the end of the run of the program you can examine the console logs of the 3 subsystems.
4. To see how connections between components work, one can compare the outgoing packet logs, to the incoming packet logs for the adjacent subsystem

For example, to test that the floor subsystem is properly sending requests to the scheduler

You can look at the values in the packet being sent by the floor subsystem



And compare it to what the scheduler receives on its end



And if the data sent out by the floor matches what the scheduler reads in, then we know that the integration between the two components is successful.

Another tool we have to use in making sure that integration between subsystems is the packet validation for every packet sent between subsystems. The packet validation allows us to see at a glance if a packet that was sent between subsystems is being constructed improperly allowing faster validation. The packet validation was done by comparing the format of the packet in matched the pre-agreed format for that type of packet.

## **Acceptance Testing:**

For acceptance testing, we went and reviewed the UI and corresponding console outputs to ensure the following; first, that the elevator requests are being distributed logically; and second, that all requests are being handled correctly by both the backend and the UI.

Initially, the jobs are distributed as per table 1 below. The jobs are distributed initially to available elevators, then assigned based on the current location and direction of the elevators. The first request in ELEVATOR 1 is a door error, which can be recovered from. This error causes a recoverable fault, so jobs are not allocated, but are forced to wait.

The second elevator also has an error, but this is an elevator timer error, which is a hard fault. When this occurs, the elevator is shut down since it cannot recover. When this happens, the jobs from ELEVATOR 2 are sent back to the scheduler subsystem to be reassigned. The scheduler subsystem looks at the updated location and directions of the elevators and reassigns them as per Table 2 below.

When looking at the UI, we can see the doors open on each floor required for the elevators to handle the jobs assigned to it. We can also see the UI light up to display both the expected faults. Similarly, we can see the output logs from both the elevators and floor subsystems to match the expected results for each of the elevators. Because of these factors, the acceptance testing has proven the code works in an acceptable manner for the UI and overall functionality of the system.

Table 1: Initial job distribution

|  |  |  |  |
| --- | --- | --- | --- |
| **ELEVATOR 1** | **ELEVATOR 2** | **ELEVATOR 3** | **ELEVATOR 4** |
| 14:05:15.000 2 Up 20 Door 2 | 14:10:15.000 16 Down 2 Time 7 | 14:15:15.000 1 Up 19 None 0 | 14:20:15.000 19 Down 1 None 0 |
| 14:25:15.000 1 Up 16 None 0 |  |  |  |
| 14:35:15.000 2 Up 8 None 0 |  |  |  |

Table 2: Job redistribution after hard fault

|  |  |  |  |
| --- | --- | --- | --- |
| **ELEVATOR 1** | **ELEVATOR 2** | **ELEVATOR 3** | **ELEVATOR 4** |
| 14:05:15.000 2 Up 20 Door 2 | FAULT - STOPPED | 14:15:15.000 1 Up 19 None 0 | 14:20:15.000 19 Down 1 None 0 |
| 14:25:15.000 1 Up 16 None 0 |  |  | 14:10:15.000 16 Down 2 None 7 |
| 14:35:15.000 2 Up 8 None 0 |  |  | 14:30:15.000 17 Down 2 None 0 |
|  |  |  | 14:40:15.000 9 Down 1 None 0 |

# **RESULTS FROM MEASUREMENT:**

**Door opening & closing time:** = 9.39 seconds

**All floors:**

v1 = = 1.59 m/s

v2 = v3 = = 1.428 m/s

v4 = = 1.244 m/s

**Average Velocity** = 1.4225 m/s

# **REFLECTION ON DESIGN:**

Overall, this project went well. There were portions where we attempted to get ahead of ourselves with the work. For example, in Milestone 2, we attempted to refactor the project to use UDP rather than boxes, which would have been the requirement for Milestone 3. This ended up making a lot of excess work on ourselves and we ended up having to submit without properly having the whole code working.

If we had more time to redo portions, we would likely work on redoing the elevator classes, as they became fairly jumbled over the course of a few milestones in order to get them working by the deadlines. The UI turned out decently, as it runs asynchronously through remote procedure calls from the scheduler subsystem. It properly displays the current location and status of each elevator, though is partially hard-coded due to the minimal time for getting Milestone 5 running before the demonstration. The scheduler logic turned out decently for distributing jobs, but it was tougher to create than we would have liked due to inconsistent information regarding how new jobs should be handled (i.e., if they should be automatically given to stopped elevators or just the closest elevator even if it is busy, etc.).

It has been a great group to work with, even though the many stressful weeks of getting code complete. Thankfully, we have not faced any problems with any of the group members not coordinating or cooperating to get the work done. Everyone was cooperative and worked hard to get each of the project iterations done successfully and put in their full time and effort to get it done as a group. Overall, this project could have gone better if we had more time, but we believe that we put in our absolute best effort in this project. Thank you for taking the time to read our final project report.