Carleton University

Faculty of Engineering

Department of Systems and Computer Engineering

SYSC 3303 - Real-Time Concurrent Systems

Elevator Control System

Submitted by: L3-G3

Ezra Pierce

Jeong Won Kim

Kashish Saxena

Sunjeevani Pujari

Cameron Chung

April 14, 2021

CONTENTS

[**Breakdown of Responsibilities**](#_40z27hw5w29t) **2**

[Iteration 1](#_v0myisrwtoxc) 2

[Iteration 2](#_3nmlsnotvptp) 2

[Iteration 3](#_nwod1xj898nx) 2

[Iteration 4](#_ryrr0ihndzpw) 2

[Iteration 5](#_hol54p25u7kp) 2

[**Diagrams**](#_ou0e7kekc645) **3**

[UML Class Diagram](#_78re4uskkof3) 3

[UML Sequence Diagram](#_ozqg6ry2vzd5) 4

[Elevator State Machine](#_78re4uskkof3) 5

[Scheduler State Machine](#_78re4uskkof3) 5

[Scheduler Timing Diagrams](#_78re4uskkof3) 6

[Transient Error Timing Diagram](#_mbtzssi7onrk) 6

[Permanent Error Timing Diagram](#_eb720swnj27m) 7

[**Setup and Test Instructions**](#_jp6bcfqv395p) **7**

[**Measurement Results**](#_5c5h0joojuhw) **8**

[**Design Reflection**](#_jp6bcfqv395p) **8**

# 

## **Breakdown of Responsibilities**

### **Iteration 1**

Ezra Pierce - Implementation of Scheduler Class and Rough Draft of UML Class diagram

Jeong Won Kim - Implementation of Floor Class and JUnit Test

Cameron Chung - Implementation of UML Class, Sequence Diagram, and README file

Kashish Saxena - Implementation of Elevator Class and Review of UML diagrams

Sunjeevani Pujari - Implementation of Elevator Class and JUnit Test

### **Iteration 2**

Ezra Pierce - Code implementation of Scheduler & Elevator state machine, refactoring classes

Jeong Won Kim - Implementation of Scheduler & Elevator state machine

Cameron Chung - Implementation of Elevator state machine, refactoring UML diagrams

Kashish Saxena - Implementation of Elevator and Scheduler State Machine

Sunjeevani Pujari - Implementation of Elevator and Scheduler State Machine

### **Iteration 3**

Ezra Pierce - Refactoring Scheduler and Elevator Classes to work with UDP

Jeong Won Kim - Implementing Scheduler, Floor & Elevator UDP layer and JUnit tests

Cameron Chung - Refactoring Scheduler and Elevator, Implementing State machine JUnit tests

Kashish Saxena - Refactoring UML diagrams, Recatoring Elevator State Machine

Sunjeevani Pujari - Recatoring Scheduler state machine, Refactoring JUnit Tests

### **Iteration 4**

Ezra Pierce - Implementing Error Detection in Elevator

Jeong Won Kim - Implementing Error Detection in Scheduler

Cameron Chung - Implementing Code times intervals and Error Detection JUnit test

Kashish Saxena - Refactoring UML diagrams and Elevator state machine

Sunjeevani Pujari - Refactoring JUnit Tests and UML diagrams

### **Iteration 5**

Ezra Pierce - Added timing for scheduler performance, added fault injection into input file, general bug fixing from past iterations, added faults to GUI, Refactoring Unit Tests

Jeong Won Kim - Refactoring Unit Tests, Implemented Floor and Elevator GUI

Cameron Chung - Refactoring Unit Tests, Writing out Final Report

Kashish Saxena - Refactoring UML diagrams and state machines, Writing Final Report

Sunjeevani Pujari - Refactoring Timing diagrams, Writing Final Report

## **Diagrams**

### **UML Class Diagram**

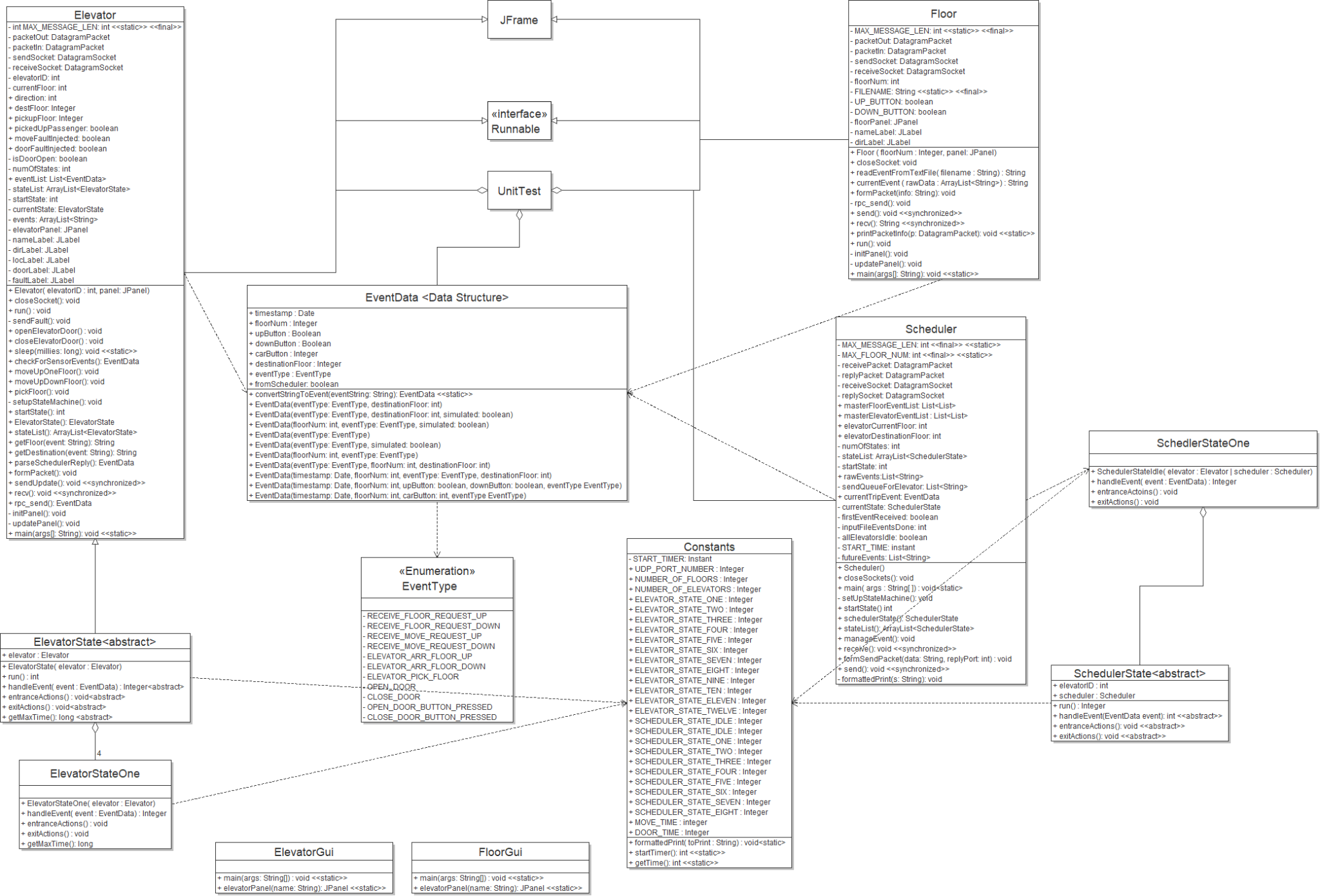


Figure 1: UML Class Diagram

### 

### **UML Sequence Diagram**

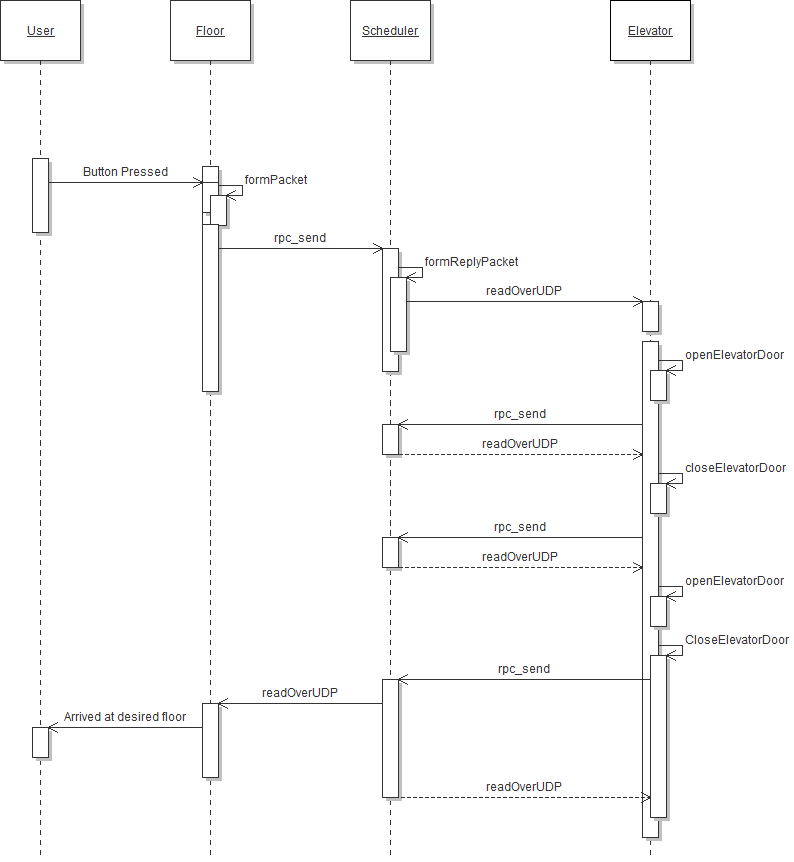


Figure 2: UML Sequence Diagram

### **Elevator State Machine**

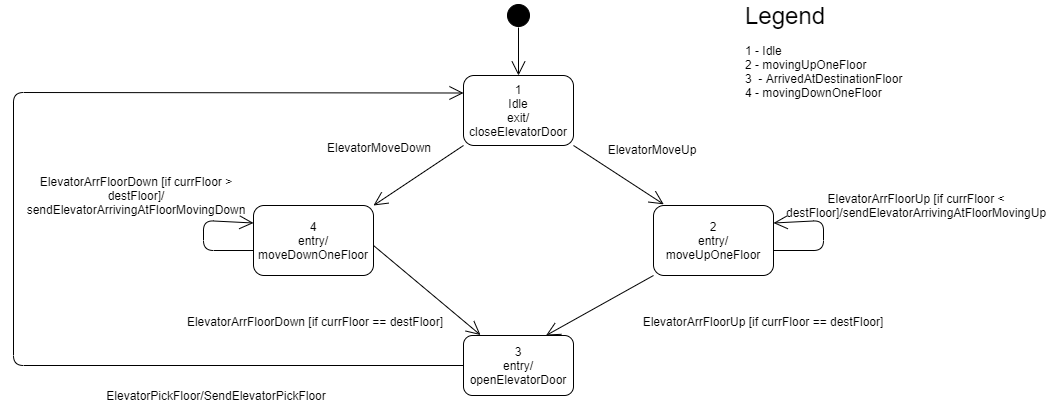


Figure 3: Elevator State Machine

### **Scheduler State Machine**

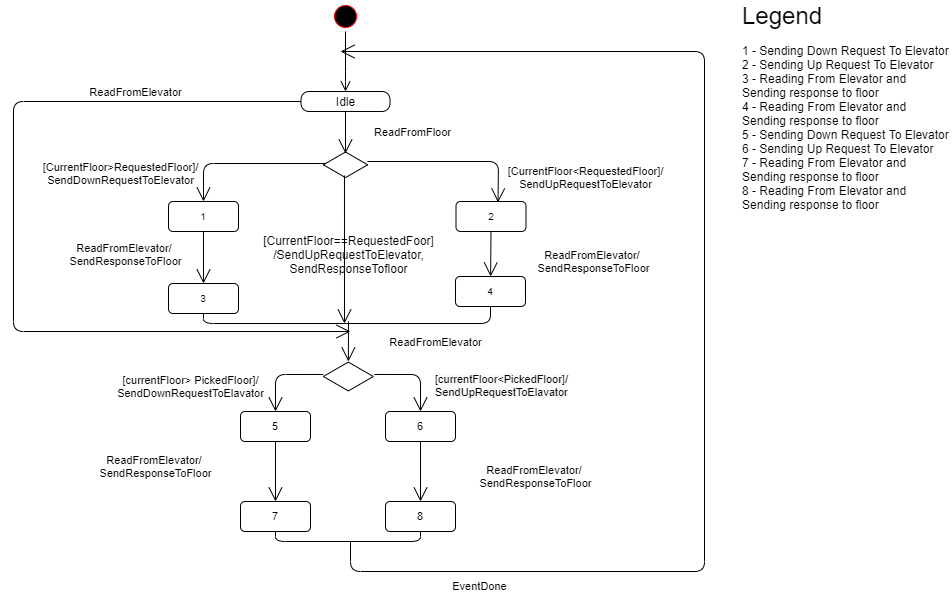


Figure 4: Scheduler State Machine

### **Scheduler Timing Diagrams**

#### Transient Error Timing Diagram

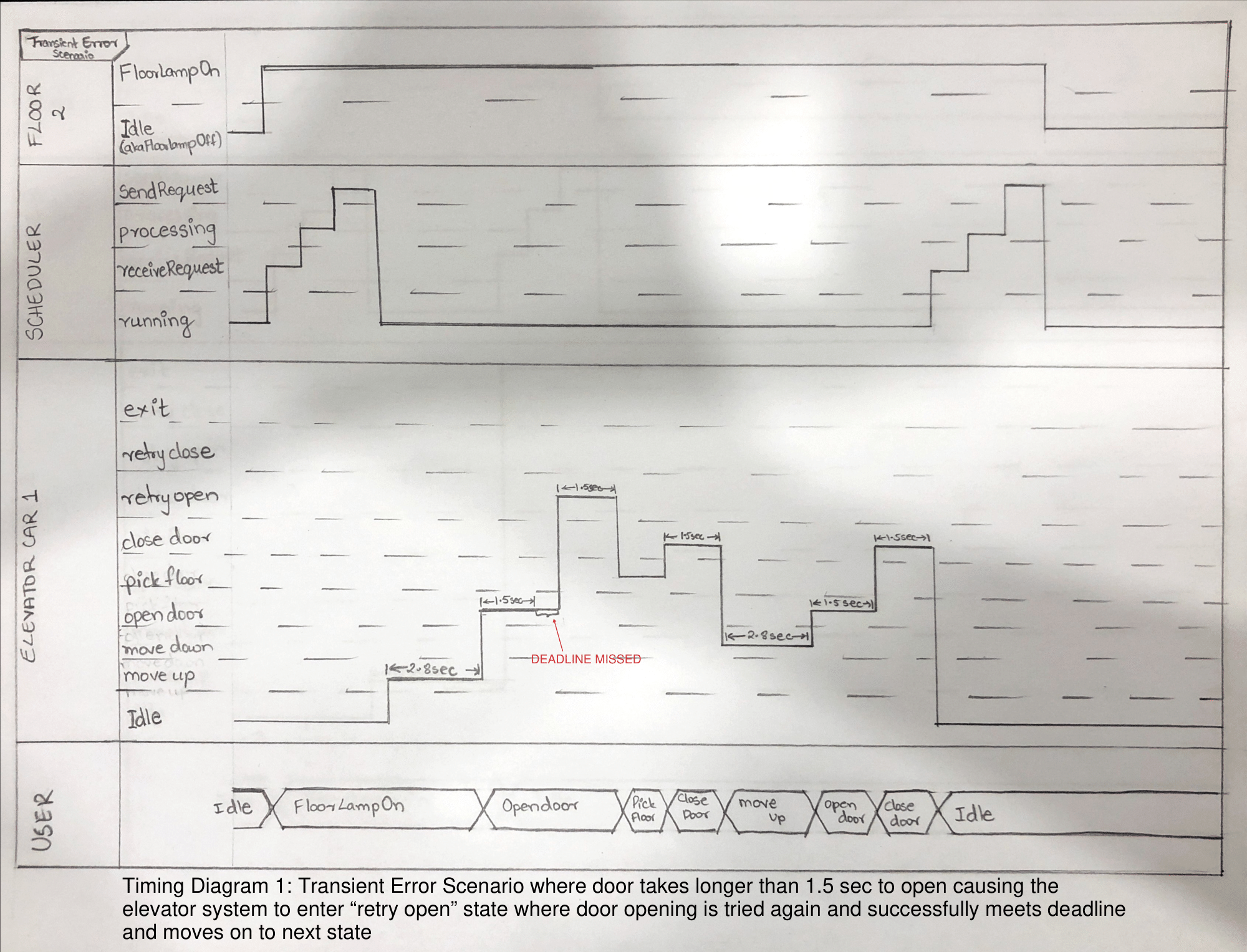


Figure 5: Transient Error Timing Diagram

#### 

#### Permanent Error Timing Diagram

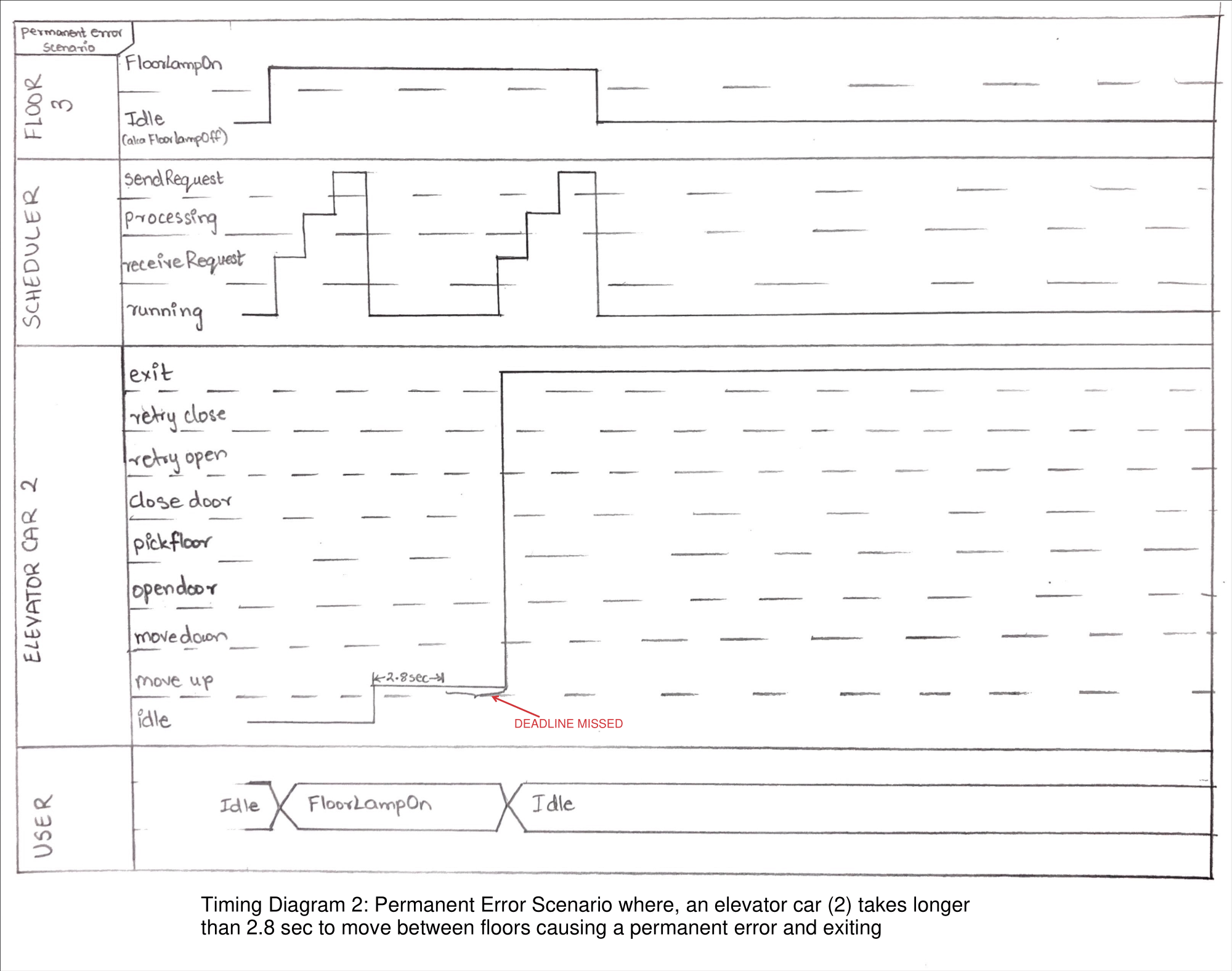


Figure 6: Permanent Error Timing Diagram

## **Setup and Test Instructions**

Setup Instructions:

To get this all ready, open the folder "project->ElevatorSim->src->elevatorsim". Then you will have to start all three subsystems separately. First run the Scheduler.java main() function. Pin this console, it will print out the final time taken at the end. Then run the Floor.java main() function. Pin this console as well. Then open the folder “project->ElevatorSim->src->elevator” and run Elevator.java main() function. Enter the number of elevators (4). Then on the Floor console enter the number of floors (22). The system will now start the simulation, you will see the GUI updating. The system reads the events from ‘events.txt’. At the end the Scheduler console will print out the time it has taken.

Test Instructions:

The tests are built using JUnit. You will need to have JUnit installed. Then you can run the UnitTest.java file as a JUnit test suite.

## **Measurement Results**

In our multiple tests, we measured the system to run through all of the events at a total elapsed time of 227358 milliseconds, 227340 milliseconds, 227362 milliseconds, 227354 milliseconds, 227345 milliseconds, and 227352 milliseconds. This comes to the program running all of the inputs on an average elapsed time of 3 minutes and 47 seconds.

## **Design Reflection**

The goal of this project was to implement an elevator control system and a simulator that can handle multiple elevator cars and floors that are able to run on multiple computers simultaneously. Throughout the development stages of the project, our team was successfully able to implement all the major functionalities of the system and implement a well designed Scheduler subsystem that properly manages all the requests and deals with errors in an appropriate manner to make the system more efficient. Additionally, the Graphical User Interface of the system provides well-designed descriptive information of the status of each floor and elevator car in the system and updates automatically every time the state of either a floor or elevator car is changed. The GUI also updates every time an error occurs in order to notify the administrator of any fixes required for the system.

Although the developed system is functioning well, our team believes that we could improve the efficiency of the system by using data structures and Mutexes as a way to share requests between subsystems like we did in Iteration 1 rather than using the UDP layer so that we can securely transport request information between subsystems. Our team would also like to improve our test harness to perform more unit tests to check the performance of all the functionalities of the system. On top of that, more rigorous tests can be added to test the limits of the system such as looking at the number of requests the system can handle before causing an error.

Overall, Our team is satisfied with the outcome of this project and is grateful for the experience and things that we've learnt throughout the process.