SYSC 3303 Project Report

Lab Section 3, Group 5

Brandon Hartford

Darren Holden

Callum Kirby

Logan MacGillivray

Christopher Molnar

Contents

[Team Member Responsibilities 1](#_Toc5727225)

[Diagrams 3](#_Toc5727226)

[Class Diagrams 3](#_Toc5727227)

[Scheduler Subsystem 3](#_Toc5727228)

[Floor Subsystem 4](#_Toc5727229)

[Elevator Subsystem 5](#_Toc5727230)

[Resources 6](#_Toc5727231)

[Elevator State Machine 7](#_Toc5727232)

[Timing Diagrams 8](#_Toc5727233)

[Elevator Hard Error 8](#_Toc5727234)

[Elevator Soft Error 8](#_Toc5727235)

[Scheduler 9](#_Toc5727236)

[Setup and Test Instructions 10](#_Toc5727237)

[Measurement Results 11](#_Toc5727238)

[Scheduability Analysis 13](#_Toc5727239)

[Design Reflection 14](#_Toc5727240)

[Schedular Subsystem 14](#_Toc5727241)

[Elevator Subsystem 14](#_Toc5727242)

[Floor Subsystem 14](#_Toc5727243)

# Team Member Responsibilities

The following tables (Tables 1 to 5 below) outline each team member’s responsibilities for each of the iterations.

|  |  |
| --- | --- |
| **Team Member** | **Responsibilities** |
| Brandon Hartford | * Programming the Floor system classes * Writing JUnit Test Cases * Documentation |
| Darren Holden | * Programming the Scheduler system classes * Writing documentation * Writing JUnit test cases * Performing end-to-end tests and debugging |
| Callum Kirby | * Programming the Elevator, ElevatorMotor, and ElevatorReciever * Writing JUnit Test Cases * Documentation |
| Logan MacGillivray | * Programming the Scheduler system classes * Drawing UML Class diagram |
| Christopher Molnar | * Programming SystemFile, Message, and Elevator State Machine * Created JUnit Tests * Writing documentation * Drawing the elevator state machine diagram |

Table : Iteration 1 Team Member Responsibilities

|  |  |
| --- | --- |
| **Team Member** | **Responsibilities** |
| Brandon Hartford | * Updating the Floor Subsystem |
| Darren Holden | * Updating the Scheduler subsystem |
| Callum Kirby | * Updating the Elevator subsystem |
| Logan MacGillivray | * Updating the README * Updating UML Class diagrams |
| Christopher Molnar | * Updating the Elevator subsystem * Elevator state machine diagram |

Table : Iteration 2 Team Member Responsibilities

|  |  |
| --- | --- |
| **Team Member** | **Responsibilities** |
| Brandon Hartford | * Timing Diagrams * Floor subsystem formatting cleanup |
| Darren Holden | * Debugging error modes * Preliminary performance timing |
| Callum Kirby | * Updating the Directions enum class * Automated test updates * UML class diagrams updates |
| Logan MacGillivray | * Updating the README * Updating UML Class diagrams * Error handling in the Scheduler |
| Christopher Molnar | * Implementing elevator error modes * Implementing error propagation through the system * Updating the state machine diagram |

Table : Iteration 3 Team Member Responsibilities

|  |  |
| --- | --- |
| **Team Member** | **Responsibilities** |
| Brandon Hartford | * Started work on the GUI |
| Darren Holden | * Error fixing * Improving timing accuracy |
| Callum Kirby | * Error fixing |
| Logan MacGillivray | * Updating UML Class diagrams |
| Christopher Molnar | * Error fixing * Updating the state machine diagram |

Table : Iteration 4 Team Member Responsibilities

|  |  |
| --- | --- |
| **Team Member** | **Responsibilities** |
| Brandon Hartford | * Working on the GUI * Refactor to scheduler |
| Darren Holden | * Testing * Writing JUnit tests * Writing the report |
| Callum Kirby | * Timing and Scheduling analysis * Summarizing Elevator subsystem |
| Logan MacGillivray | * Updating UML class diagrams with GUI components |
| Christopher Molnar | * Assisted with timing and scheduling analysis * Assisted with GUI |

Table : Iteration 5 Team Member Responsibilities

# Diagrams

## Class Diagrams

### Scheduler Subsystem

Figure 1 below shows the class diagram for the scheduler subsystem. This includes the support for the GUI system, as the GUI is drawn based on the data in the Scheduler.

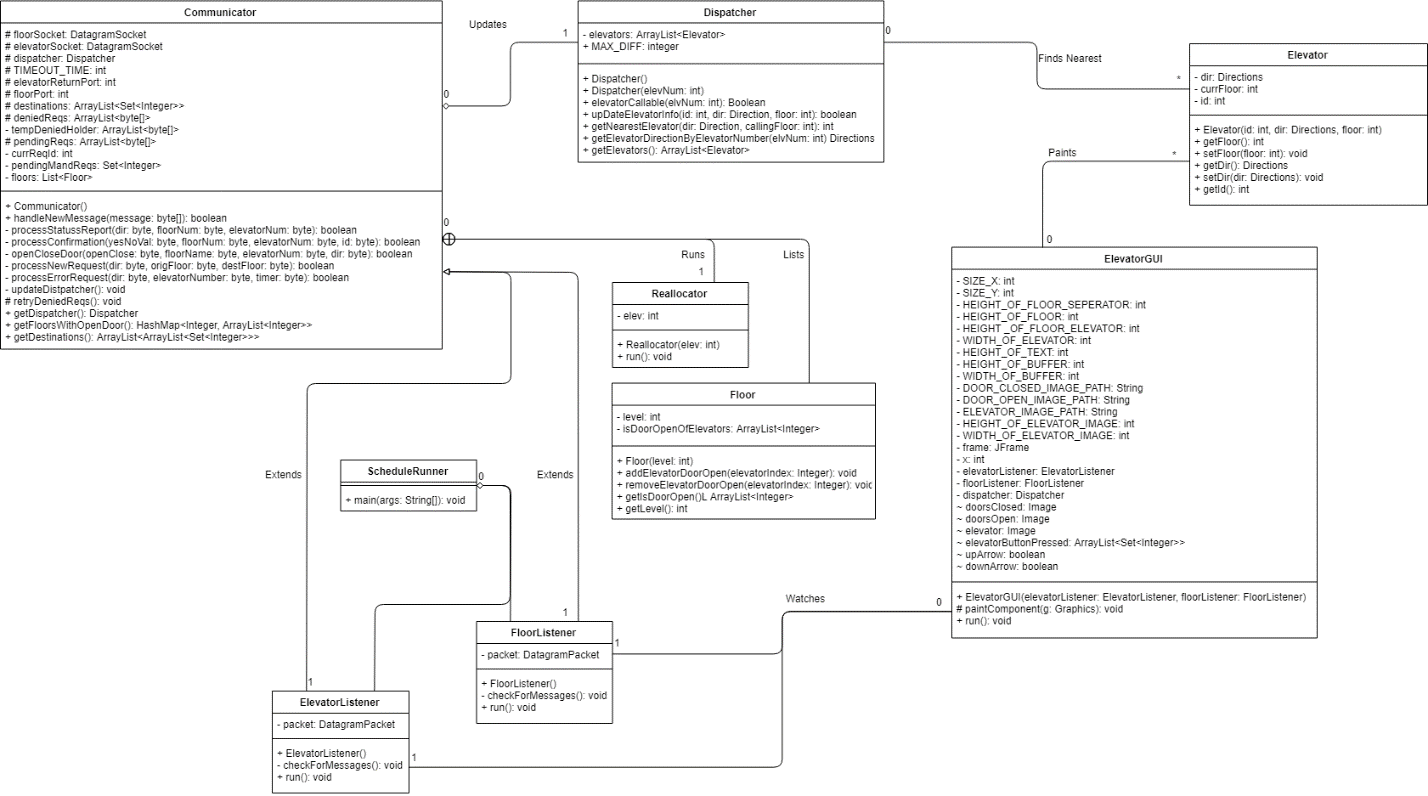


Figure : Scheduler Subsystem Class Diagram

### Floor Subsystem

Figure 2 below shows the class diagram for the floor subsystem.

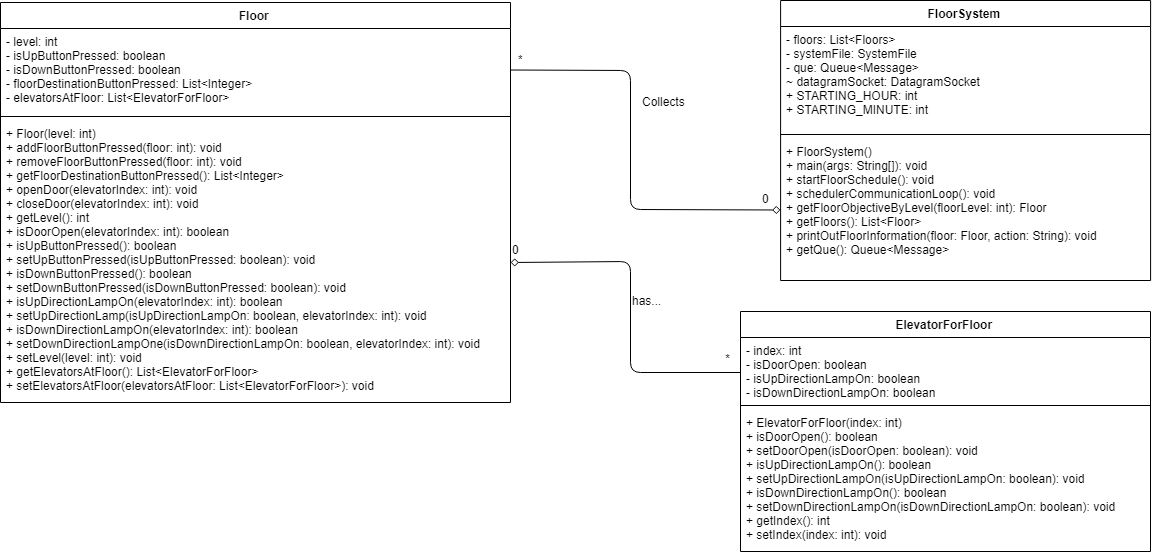


Figure : Floor Subsystem Class Diagram

### Elevator Subsystem

Figure 3 below shows the class diagram for the elevator subsystem.

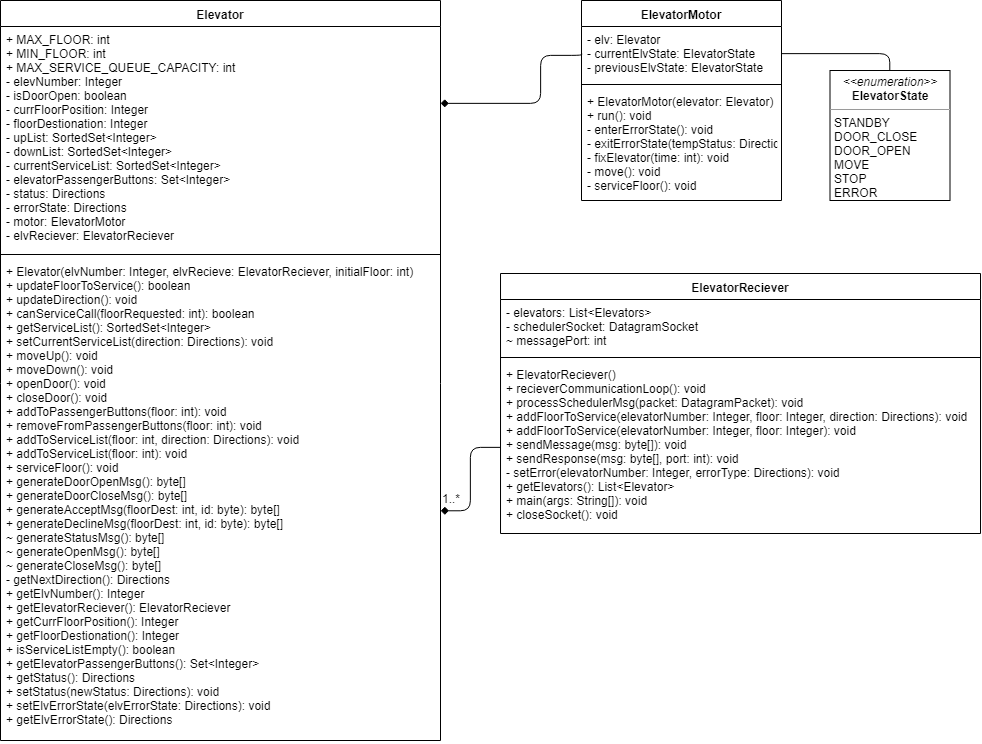


Figure : Elevator Subsystem Class Diagram

### Resources

Figure 4 below shows all the supporting classes that are used by the overall system. This includes share constant values, an enumerated type for directions, and the SystemFile class which reads the input file.

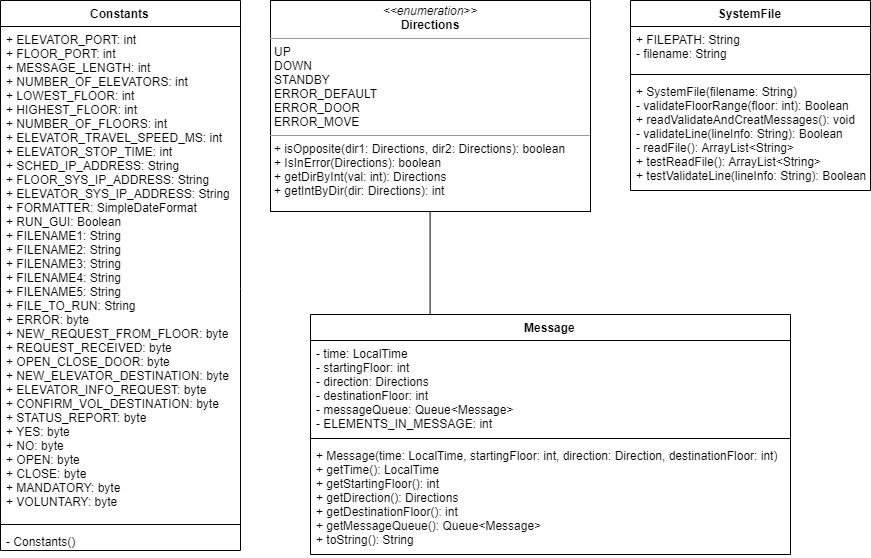


Figure : Class Diagram of Miscellaneous Resources

## Elevator State Machine

Figure 5 depicts the state machine that the elevator instantiates. The states are designed such that the elevator should only die if it receives a hard error, which is meant to be a fatal event.

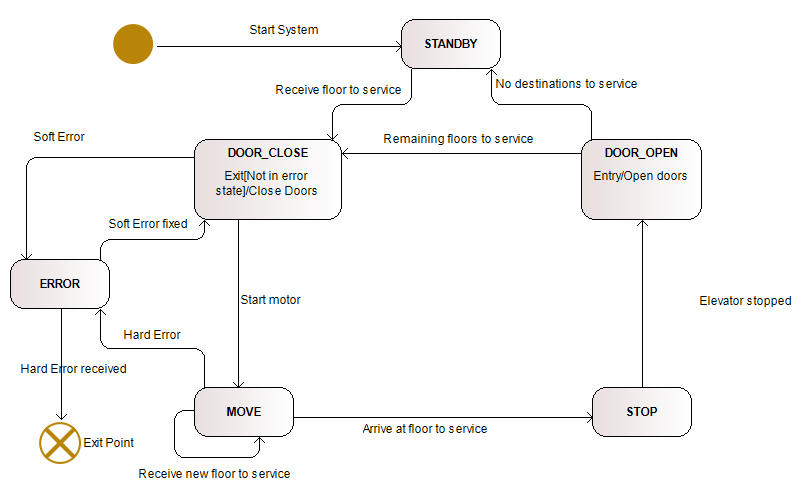


Figure : Elevator State Machine Diagram

## Timing Diagrams

### Elevator Hard Error

Figure 6 shows the timing diagram for when an elevator receives a hard error, the specific example being an error that causes the elevator to get stuck between floors.

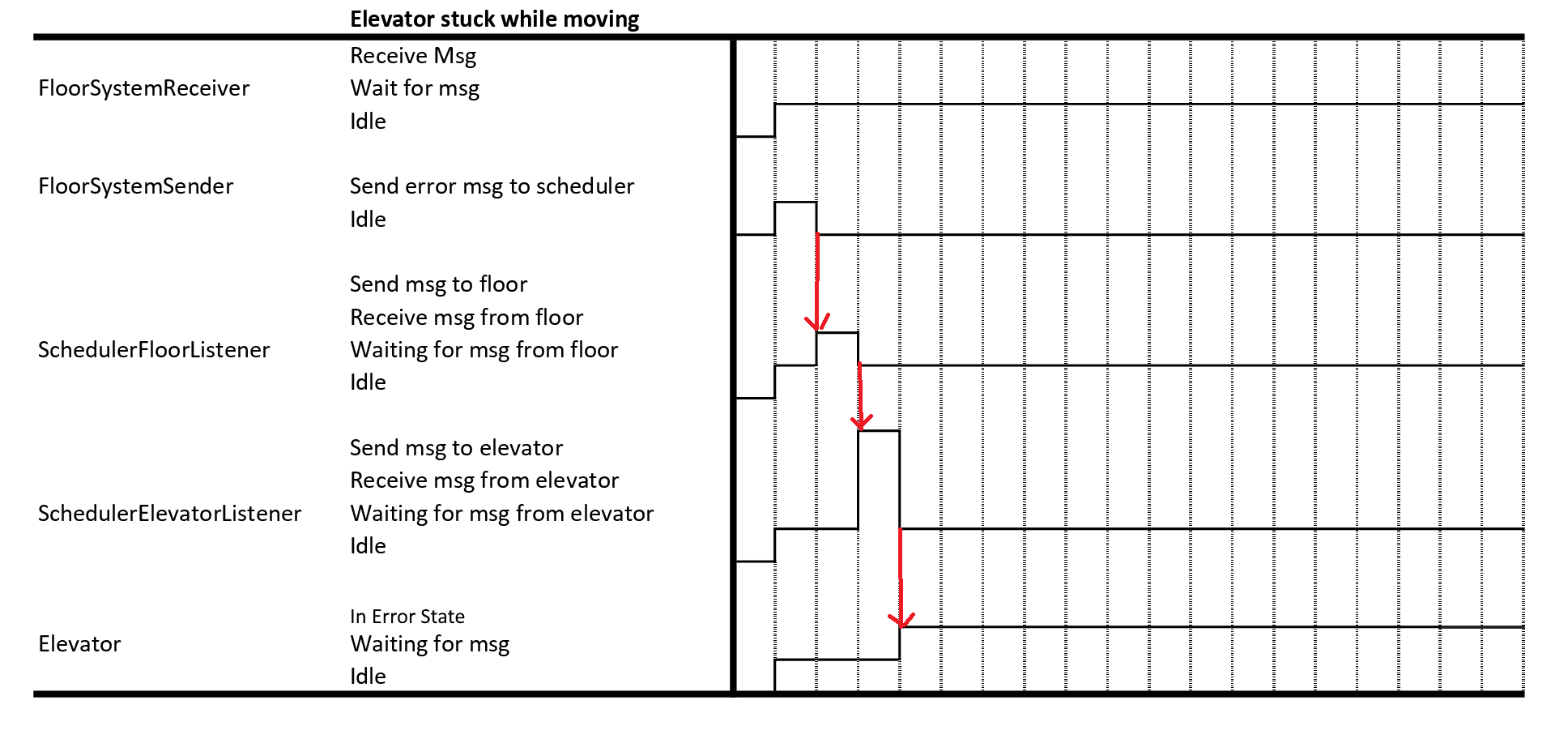


Figure : Timing Diagram for a Hard Error for an Elevator

### Elevator Soft Error

Figure 7 shows the sequence of events for when an elevator gets a soft error (e.g. if the doors get stuck open). A notable difference from Figure 6 is that the elevator eventually recovers in this case.

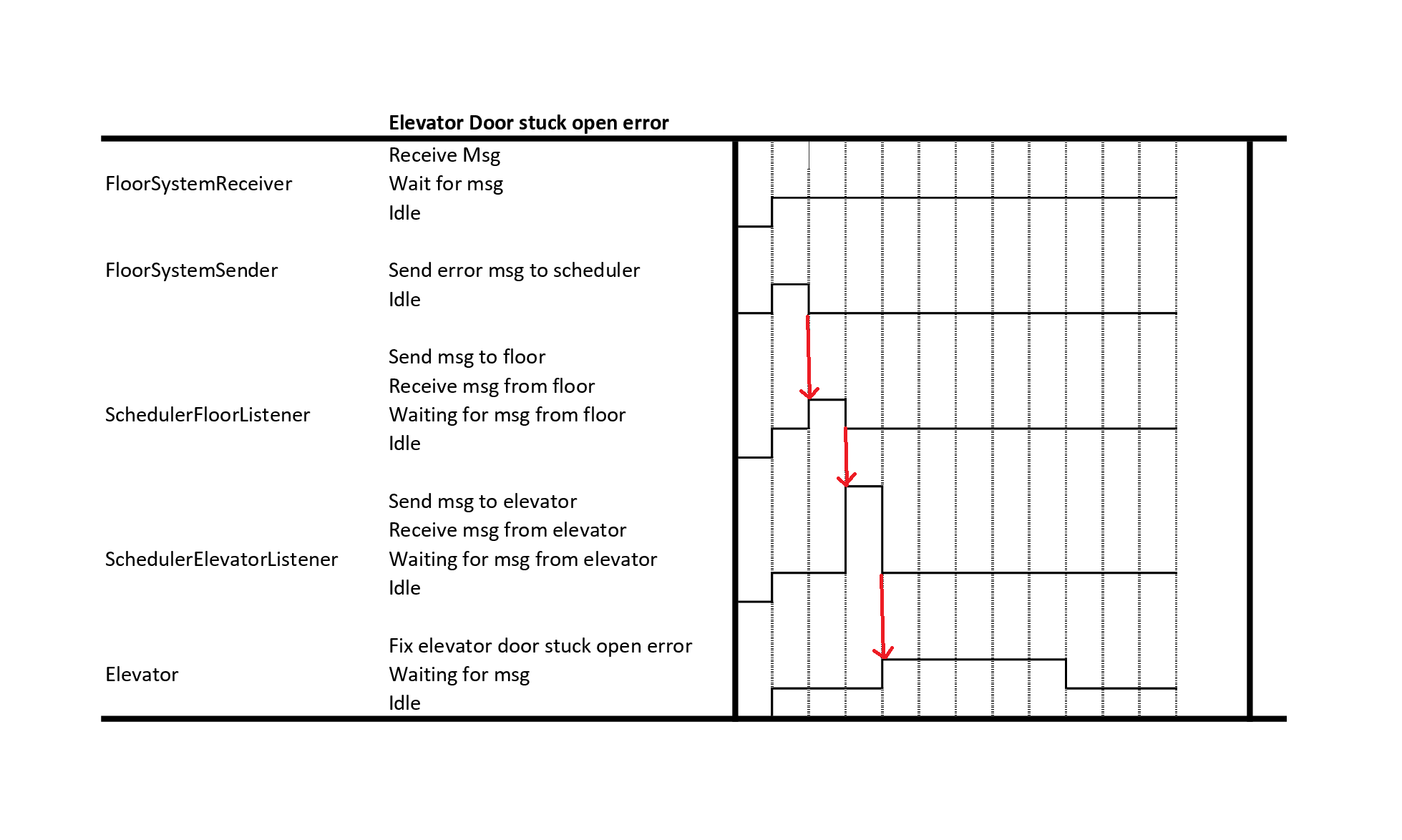


Figure : Timing Diagram for a Soft Error for an Elevator

### Scheduler

Figure 8 shows a timing diagram for the scheduler subsystem. Note that the red lines indicate a specific case when the ElevatorListener receives a message indicating that an elevator has experienced a hard error. It spawns a Reallocator thread which reallocates pending requests for that elevator. That thread terminates on completion, indicated by the red “X”.

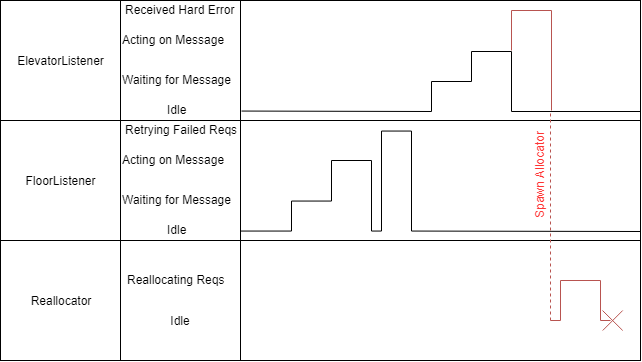


Figure : Scheduler Timing Diagram, The Red Lines Depict Behaviour for When an Elevator Experiences a Hard Error

# Setup and Test Instructions

1. Unzip project
2. Import the Java project into Eclipse
   1. Open Eclipse
   2. Create a Java Project (file->new->Java Project)
   3. Give the Project a Name (e.g. ElevatorProjectGroup5) and Finish
   4. Import File (File -> Import), select General - File System and click Next
   5. Select Browse for Current Directory, and find the Java project "ElevatorProject" in the unzipped folder from step 1
   6. Check the box next to ElevatorProject box to import all of the files
   7. For the “Into Folder”, select the Java Project you created and hit finish
   8. If a popup asking if you want to overwrite the “.classpath” appears, select “Yes to All”
3. Navigate in the Package Explorer to the Java Project you imported the files into, and open src
   1. Open ElevatorReciever.java from the elevatorSubsystem package
   2. Open SchedulerRunner.java from the scheduler package
   3. Open FloorSystem.java from the floorSubsystem package
4. To run the system on multiple computers, follow the next instructions, otherwise skip to step 5
   1. Repeat Steps 1 to 3 on each computer that is to be used.
   2. In Eclipse, open resources.Constants.java and update the following values:
      1. SCHED\_IP\_ADDRESS - This should be the IP address of the computer that will run the Scheduler
      2. FLOOR\_SYS\_IP\_ADDRESS - This should be the IP address of the computer that will run the Floor System
      3. ELEVATOR\_SYS\_IP\_ADDRESS - This should be the IP address of the computer that will run the Elevator System
   3. Repeat step b on each computer that is being used
5. The execution of the elevator demo can be customized to enable or disable the visualizations of the GUI.
   1. To enable set the ‘RUN\_GUI’ variable to ‘true’ in the constants Java file
   2. To disable set the ‘RUN\_GUI’ variable to ‘false’ in the constants Java file
6. There are numerous testing files provided within the system to simulate different scenarios of elevator requests. These files can be found in the ‘resources’ folder.
   1. To set a file as input change the ‘FILE\_TO\_RUN’ variable to the desired file name within the ‘resources’ directory
7. To run our project run ElevatorReciever.java, SchedulerRunner.java and floorSystem.java (in that order)
   1. The ElevatorReciever output will show the progress of the elevator (where its going and what state its in)
   2. The SchedulerRunner output will show the scheduler system processing requests
   3. The FloorSystem output will show the floor system sending new requests, as well as the status of the floors as they get updated
8. To access the JUnit tests, navigate to any package the is preceded with “test.”
   1. The JUnit tests are organized by the subsystem or component that they correspond to
      1. This is denoted in the package name
   2. To run a JUnit Test, navigate to the desired JUnit file and select “Run as…” and then “JUnit”

# Measurement Results

In order to measure the periods of certain tasks, we did not insert any additional instrumentation. Instead, we used the timestamps from existing the existing logs that get printed to the console by each component. This gives the following advantages:

* It doesn’t put the system under any additional load;
* We don’t have to worry about inserting new code which could cause issues;
* It utilizes the current implementation; and
* It already provides timing on all of the system functions.

For the measurements, the scheduler subsystem was run on one computer, and both the elevator and floor subsystems were both run on another computer. This means for timing the arrival sensor (which is a measurement between the elevator and floor subsystems), the system times were in synch and the values could just be compared. For measurements involving the scheduler, the difference in system times with the elevator and floor subsystem had to be considered. By analyzing the system clocks, the difference was determined. This difference was subtracted from the times for the elevator and floor systems, since that was the system that was ahead.

Three measurements were taken, for the arrival sensors interface, the elevator buttons interface, and the floor buttons interface. The arrival sensors interface was measured as the time it takes the open door request from the elevator to reach the floor, as that request is sent on the elevator’s arrival. The elevator buttons interface was measured using the time it takes for the scheduler to send the new destinations to the elevator after the elevator picks someone up. The floor buttons interface was measured as the time it takes for a new request from the floor to be received by the schedular, but not including the time it takes to process the request. These timings are seen in Tables 6, 7, and 8 below. Gray columns have been adjusted as detailed in the captions.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Start Time** | **Start (mS)** | **End Time** | **End (mS)** | **Period (mS)** |
| 2019.04.09-09.40.00.0055 | 55 | 2019.04.09-09.40.00.0056 | 56 | 1 |
| 2019.04.09-09.40.04.0805 | 805 | 2019.04.09-09.40.04.0808 | 808 | 3 |
| 2019.04.09-09.40.04.0963 | 963 | 2019.04.09-09.40.04.0965 | 965 | 2 |
| 2019.04.09-09.40.08.0058 | 58 | 2019.04.09-09.40.08.0061 | 61 | 3 |
| 2019.04.09-09.40.09.0714 | 714 | 2019.04.09-09.40.09.0717 | 717 | 3 |
| 2019.04.09-09.40.11.0310 | 310 | 2019.04.09-09.40.11.0312 | 312 | 2 |
| 2019.04.09-09.40.12.0967 | 967 | 2019.04.09-09.40.12.0970 | 970 | 3 |
| 2019.04.09-09.40.16.0063 | 63 | 2019.04.09-09.40.16.0065 | 65 | 2 |
| 2019.04.09-09.40.16.0219 | 219 | 2019.04.09-09.40.16.0221 | 221 | 2 |
| 2019.04.09-09.40.17.0170 | 170 | 2019.04.09-09.40.17.0172 | 172 | 2 |
| 2019.04.09-09.40.19.0314 | 314 | 2019.04.09-09.40.19.0316 | 316 | 2 |
| 2019.04.09-09.40.22.0470 | 470 | 2019.04.09-09.40.22.0472 | 472 | 2 |
| 2019.04.09-09.40.24.0864 | 864 | 2019.04.09-09.40.24.0866 | 866 | 2 |
| 2019.04.09-09.40.24.0924 | 924 | 2019.04.09-09.40.24.0926 | 926 | 2 |
| 2019.04.09-09.40.35.0616 | 616 | 2019.04.09-09.40.35.0618 | 618 | 2 |

Table : Arrival Sensors Interface Request Periods

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Start Time** | **Start (mS)** | **End Time** | **End (mS)** | **Period (mS)** |
| 2019.04.09-09.43.03.0505 | 3505 | 2019.04.09-09.43.04.0200 | 3510 | 5 |
| 2019.04.09-09.43.08.0252 | 8252 | 2019.04.09-09.43.08.0953 | 8263 | 11 |
| 2019.04.09-09.43.08.0498 | 8498 | 2019.04.09-09.43.09.0199 | 8509 | 11 |
| 2019.04.09-09.43.13.0251 | 13251 | 2019.04.09-09.43.13.0951 | 13261 | 10 |
| 2019.04.09-09.43.14.0756 | 14756 | 2019.04.09-09.43.15.0458 | 14768 | 12 |
| 2019.04.09-09.43.16.0502 | 16502 | 2019.04.09-09.43.17.0204 | 16514 | 12 |
| 2019.04.09-09.43.20.0704 | 20704 | 2019.04.09-09.43.21.0412 | 20722 | 18 |
| 2019.04.09-09.43.28.0629 | 28629 | 2019.04.09-09.43.29.0337 | 28647 | 18 |

Table : Elevator Buttons Interface Request Periods – Subtract 690mS from the Floor System Time to Synch the System Clocks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Start Time** | **Start (mS)** | **End Time** | **End (mS)** | **Period (mS)** |
| 2019.04.09-11.57.36.0570 | 35820 | 2019.04.09-11.57.35.0827 | 35827 | 7 |
| 2019.04.09-11.57.37.0536 | 36786 | 2019.04.09-11.57.36.0793 | 36793 | 7 |
| 2019.04.09-11.57.38.0535 | 37785 | 2019.04.09-11.57.37.0792 | 37792 | 7 |
| 2019.04.09-11.57.38.0535 | 37785 | 2019.04.09-11.57.37.0895 | 37895 | 110 |
| 2019.04.09-11.57.38.0536 | 37786 | 2019.04.09-11.57.38.0096 | 38096 | 310 |
| 2019.04.09-11.57.38.0536 | 37786 | 2019.04.09-11.57.38.0297 | 38297 | 511 |
| 2019.04.09-11.57.38.0736 | 37986 | 2019.04.09-11.57.38.0499 | 38499 | 513 |
| 2019.04.09-11.57.39.0539 | 38789 | 2019.04.09-11.57.38.0795 | 38795 | 6 |
| 2019.04.09-11.57.39.0540 | 38790 | 2019.04.09-11.57.38.0900 | 38900 | 110 |
| 2019.04.09-11.57.39.0541 | 38791 | 2019.04.09-11.57.39.0101 | 39101 | 310 |

Table : Floor Buttons Interface Request Periods – Subtract 750mS from the Floor System Time to Synch the System Clocks

# Scheduability Analysis

The system had mandatory timing deadlines that it had to meet in order to meet the requirements of the client, seen in Figure 9.

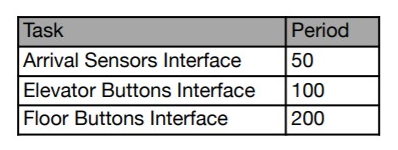


Figure : Deadline Requirements for the System

From the measurements above, we came to the results seen in Table 9. As seen in Table 9, both the arrival sensors interface and the elevator buttons interface meet the target period with no issues. This is not the case for the floor buttons interface. On average, the floor buttons interface met its deadline. However, in the worst case it took two and a half times as long as the deadline gave.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | Period (mS) | | | |
| Minimum | Maximum | Average | Target |
| Arrival Sensors Interface | 1 | 3 | 2.2 | 50 |
| Elevator Buttons Interface | 5 | 18 | 12.1 | 100 |
| Floor Buttons Interface | 6 | 513 | 183.9 | 200 |

Table : Minimum, Maximum, and Average Periods for Each of the Tasks

This issue can be traced back to a design decision in the scheduler. The requests for the floor buttons interface get handled by the FloorListener thread in the scheduler. This thread is also responsible for retrying requests that failed to get assigned to an elevator. The decision to have the FloorListener do this was made since requests coming from the floor systems were much less frequent than ones coming from the elevators, and to reduce the number of threads running in the system. However, this can cause the floor messages to be delayed when there are failed requests to deal with. This could be remedied by moving the function of retrying failed requests to a separate. This would prevent that function from blocking the reception of requests from the floor system. Running an extra thread may have an impact on the other tasks, but its doubtful that such an impact would be big enough to cause deadline issues.

# Design Reflection

## Schedular Subsystem

One thing that is good about it is that it has threads to separately listen to the floor and elevator subsystems. This gives the system some flexibility and allows it deal with the two systems in different ways.

One thing that might have been changed would be to have each of the scheduler processes run as a different thread. This would allow different priorities to be set to different tasks, which would allow the schedular to handle certain tasks quicker or slower as need be. This would have a performance impact, especially if the schedular received a burst of many messages all at once. Testing would have to be done to determine if the benefits are worst the cost.

The structures for maintaining pending requests and final elevator destinations are a bit complicated. It may have been possible to use a less convoluted method for tracking this information, perhaps with a specially constructed class.

## Elevator Subsystem

We were content in the compatibility of the elevator subsystem and being able to clearly break it into three keys parts – Motor, Listener, and State Machine. All of these were compacted into a standard ‘Elevator’ object allowing us to easily test the class at run time.

Breaking the elevator apart this way allowed for us to run more efficient elevator timing wise, as the elevator needed to be able to both listen for new communication from the scheduler, as well as move to service current requests. Breaking the subsystem into these two main threads allowed us to better simulate how a real elevator system would work.

Because the elevator object is constructed on these subclasses, it involves a lot of method calls outside of the class – which can get a little difficult traceability wise for someone who is not familiar with the system. To improve on the design of the subsystem, designing the elevator with this in mind and clearly identifying method names would be preferred. Otherwise, we believe the elevators operate in an optimal way.

## Floor Subsystem

The floor subsystem handles the floor requests by reading through the file and using a scheduled executor to schedule the requests based on whenever they are supposed to occur. I think it is a good way to do it because each request has its own thread to send out the request to the scheduler because if there are many at the same time they will occur quickly.

One thing I would change about the system is to abstract out sending the packets to the scheduler. There are a few methods that do a lot of stuff. It would be better to create another class and break down those methods so that it is easier to understand.