SYSC 3303C Real-Time Concurrent Systems
Final Report

Elevator System

Lab Section: C2
Group: 6

Due date: April 12, 2023

Mayilen Hanumunthadu 101107011
Abdul Kazal 101115331
Eric Benitez 101106903
Benjamin Cyiza 101137947

Table Contents

Breakdown of Responsibilities

Iteration 1:

Mayilen Hanumunthadu	Created Floor Subsystem, Sequence Diagram, FloorRequest (entity class)
Abdul Kazal	Created Elevator
Benjamin Cyiza	Created Scheduler
Eric Benitez	Created Unit Tests, UML class diagram

Iteration 2:

Mayilen Hanumunthadu	Created Elevator state diagram, updated UML diagram, Sequence diagram Updated ElevatorTest for state machines
Abdul Kazal	Updated Elevator.java for state machines, created Engine.java, ElevatorTest,EngineTest
Benjamin Cyiza	Updated Scheduler for state machines
Eric Benitez	Created Scheduler state diagram, updated UML class diagram, sequence diagram, integrated classes together updated SchedulerTest.java for state machines

Iteration 3:

Mayilen Hanumunthadu	Updated Tests, updated UML class diagram, Sequence diagram
Abdul Kazal	Scheduler car priority
Benjamin Cyiza	UDP integration for Scheduler and floor Subsystem
Eric Benitez	Updated Scheduler, UDP integration for Elevator

Iteration 4:

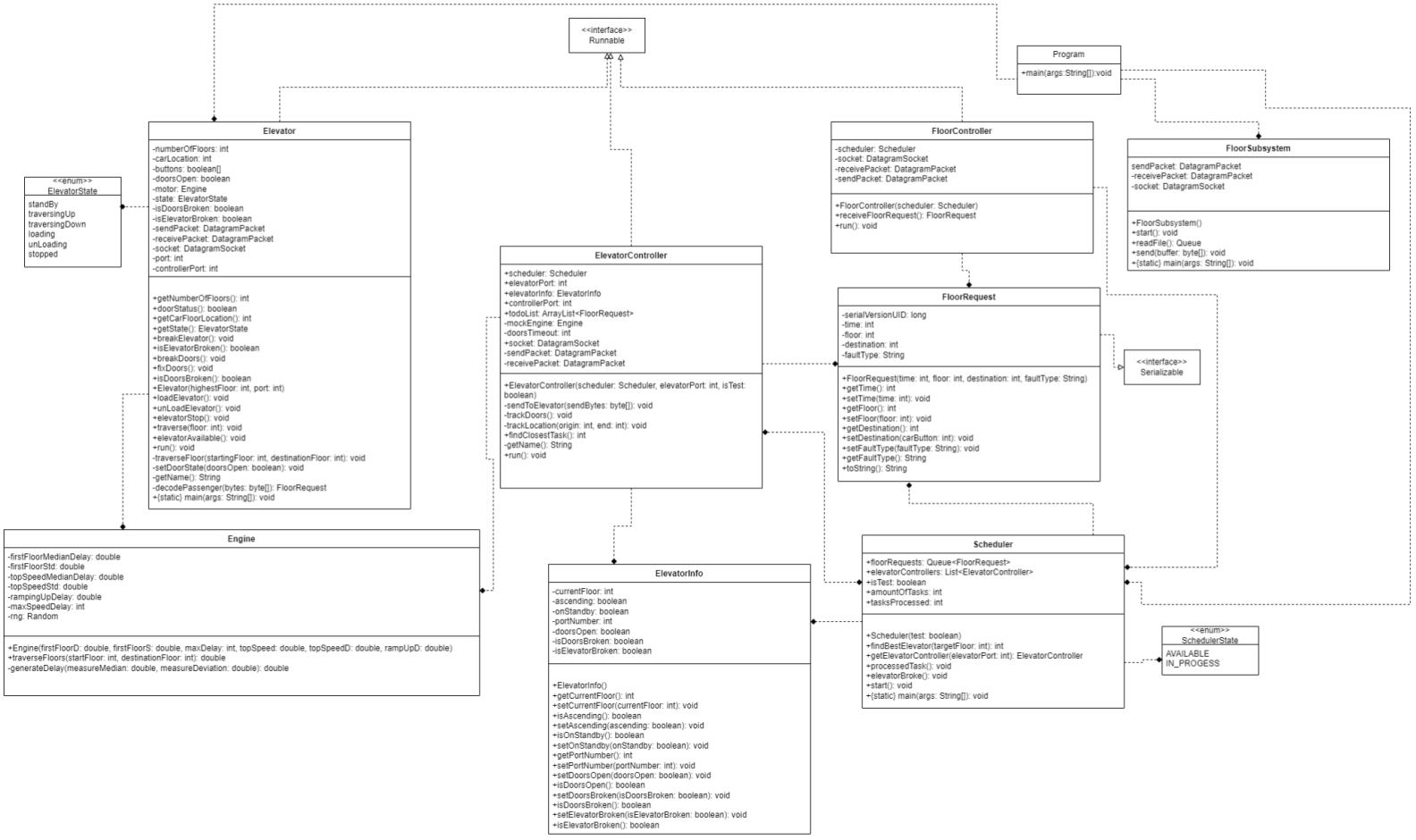
Mayilen Hanumunthadu	Updated unit tests, UML class Diagram, Sequence Diagram, Timing Diagrams
Abdul Kazal	Updated Scheduler, ElevatorController, Unit Tests
Benjamin Cyiza	Updated Scheduler, Elevator, Acceptance Tests
Eric Benitez	Updated Scheduler, FloorController,FloorSubsystem,Elevator Info

Iteration 5:

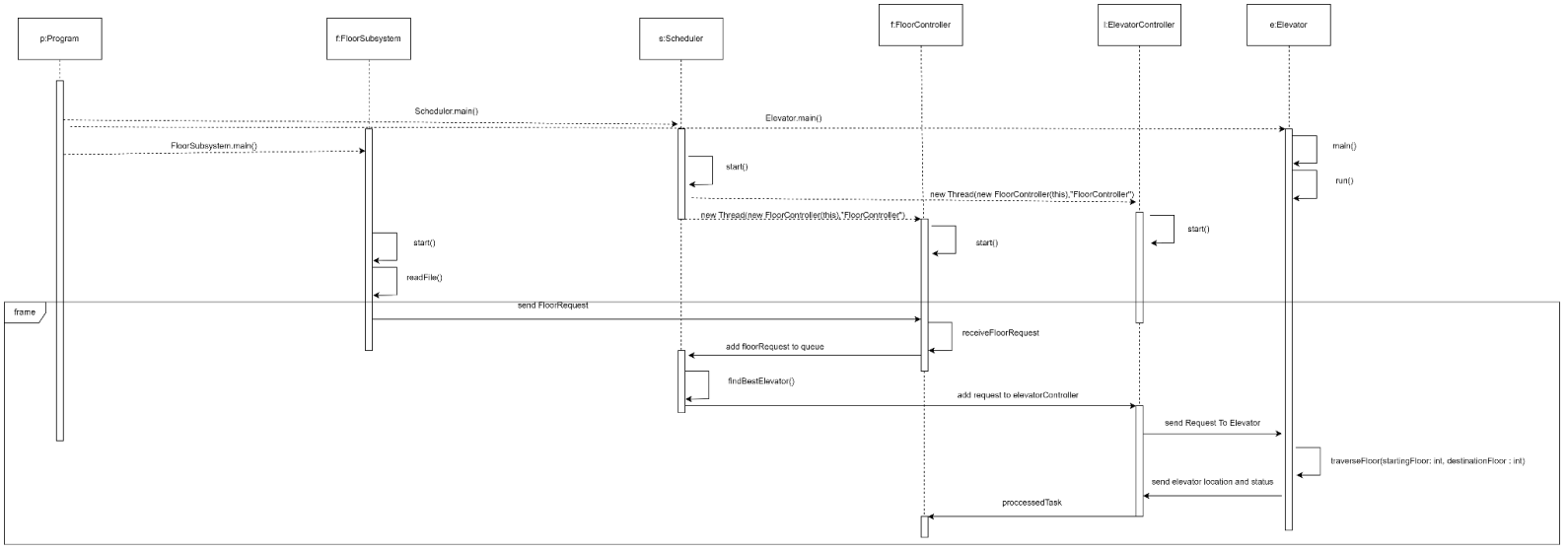
Mayilen Hanumunthadu	Updated UML class Diagram, Sequence Diagram
Abdul Kazal	Adjust unit test constructors, adding javadoc commenting and reviewing coding
Benjamin Cyiza	Updated Acceptance Tests, Debug
Eric Benitez	move elevator fault, multiple floor requests, clean up prints in scheduler

UML Diagrams

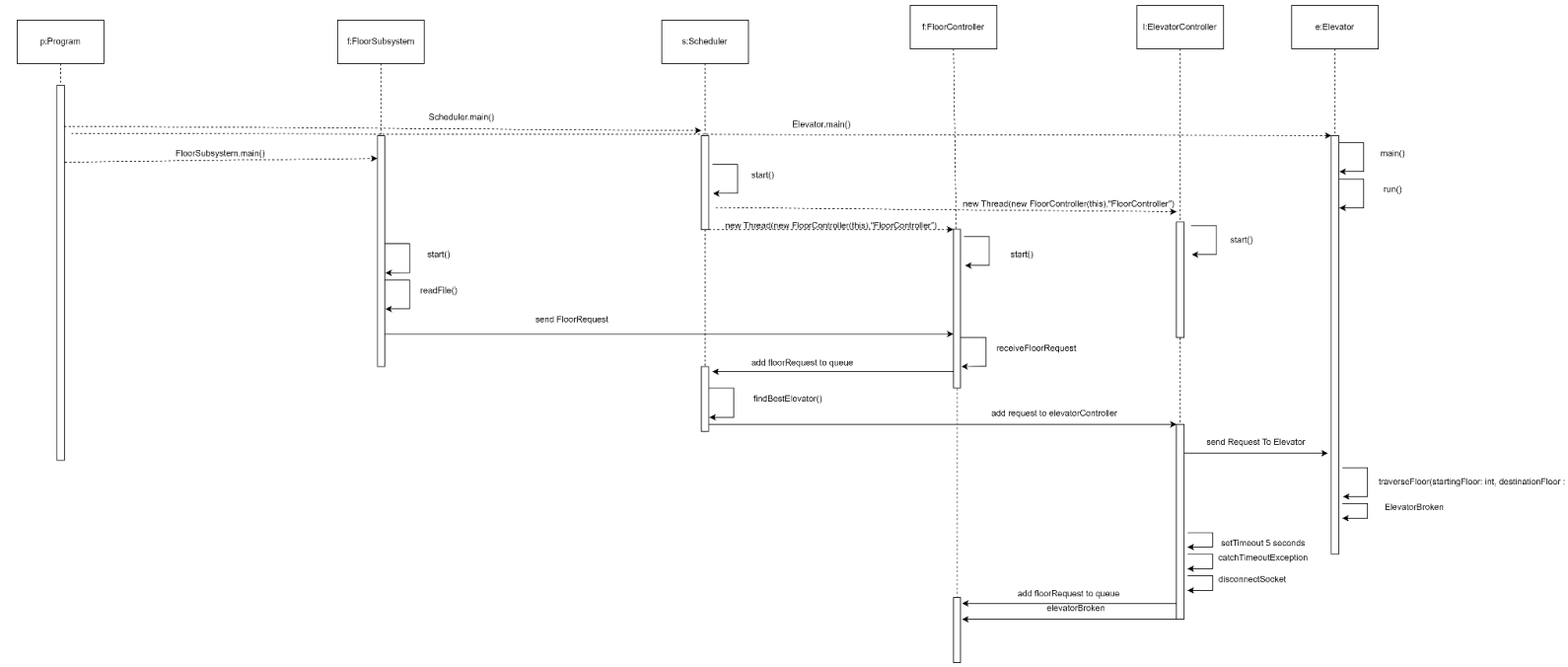
Class Diagram



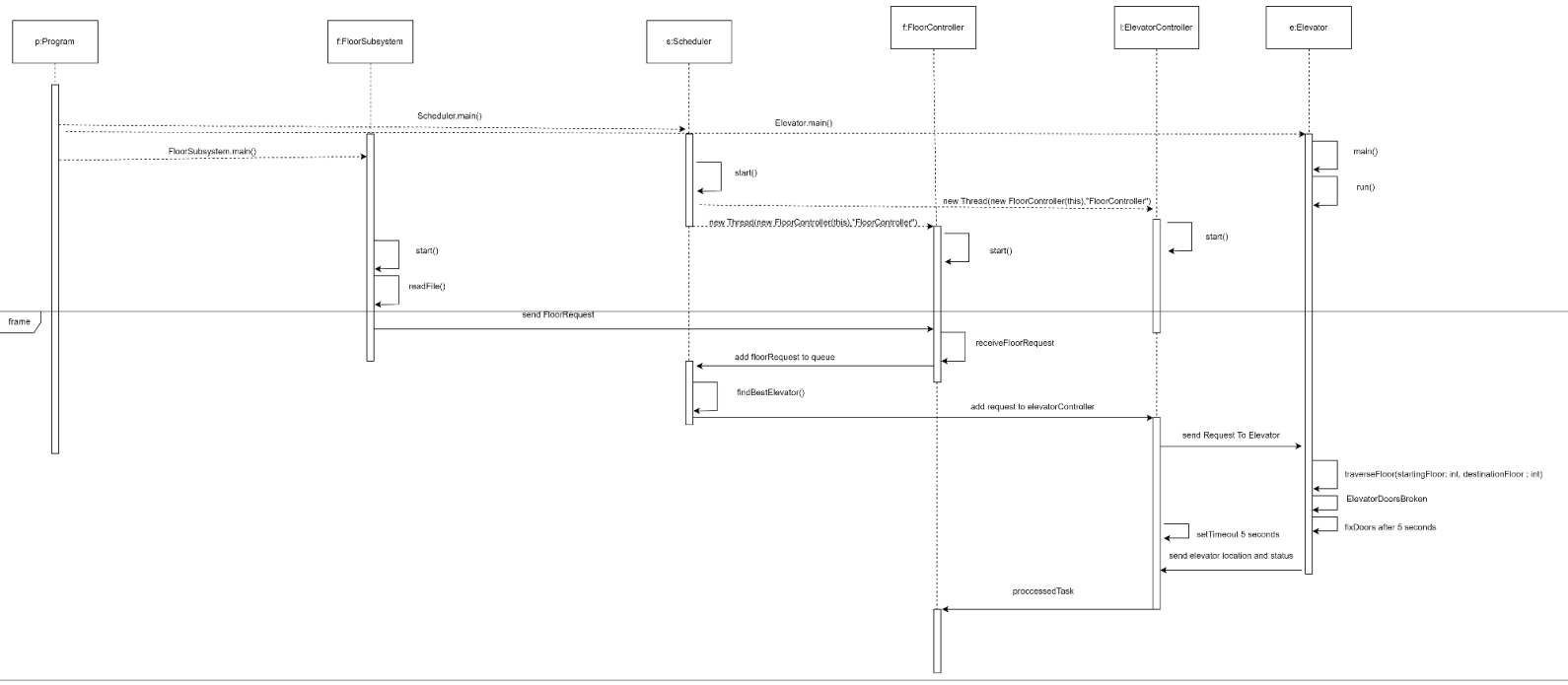
Sequence Diagram



Sequence Diagram for Broken Elevator Fault

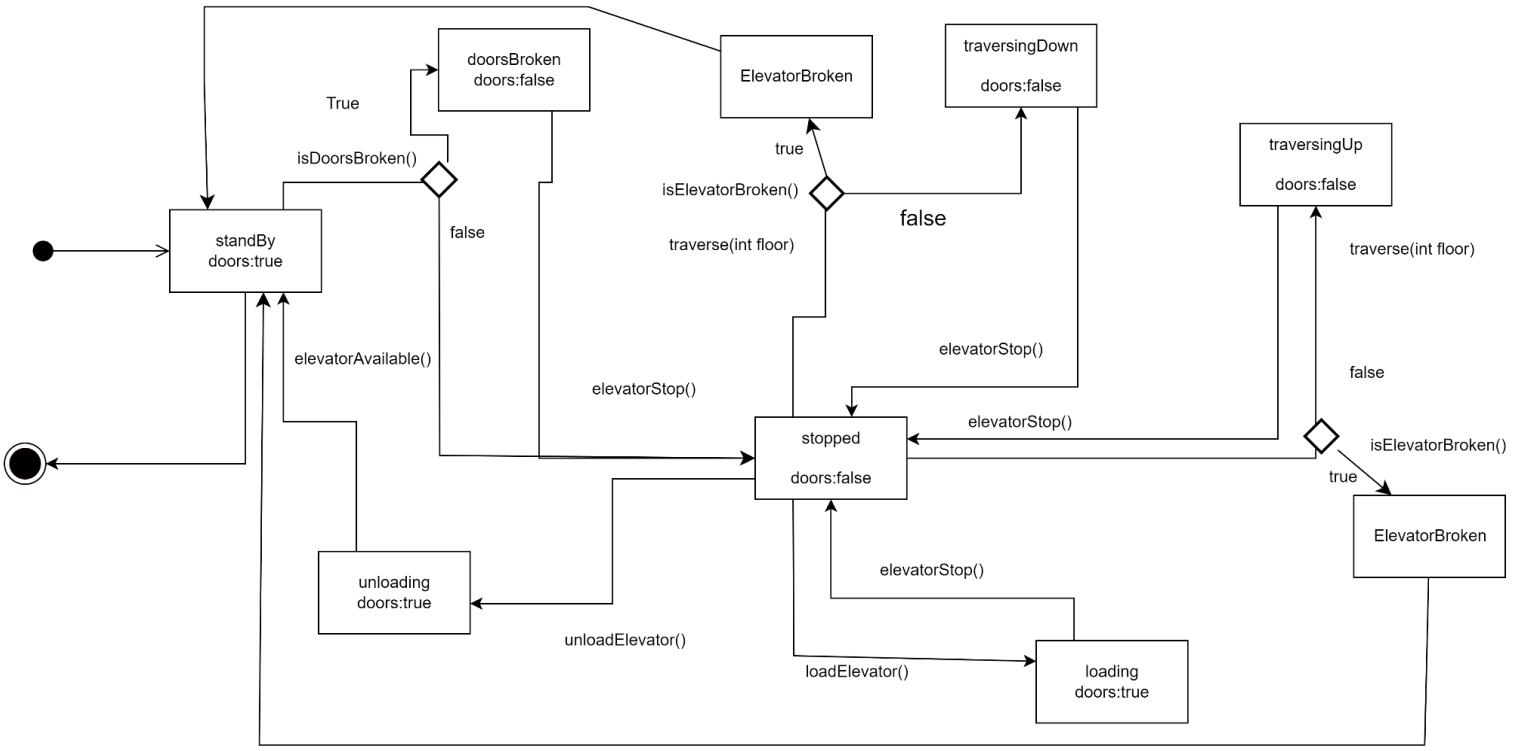


Sequence diagram for Elevator doors broken

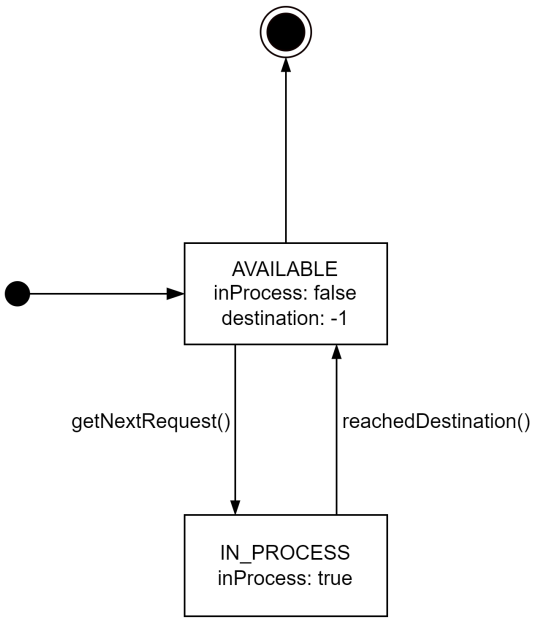


State Machine Diagram

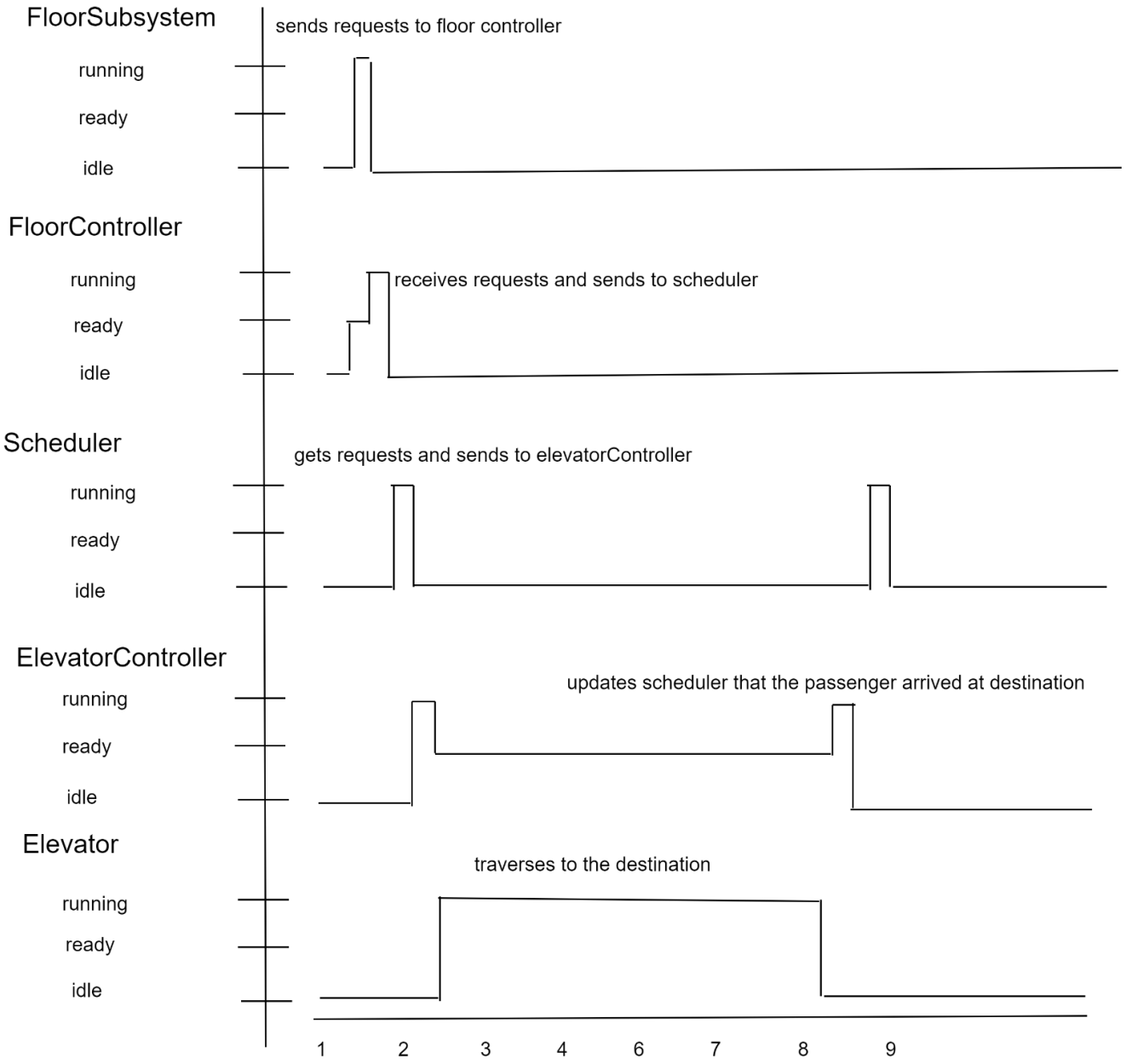
State Machine Diagram of the Elevator



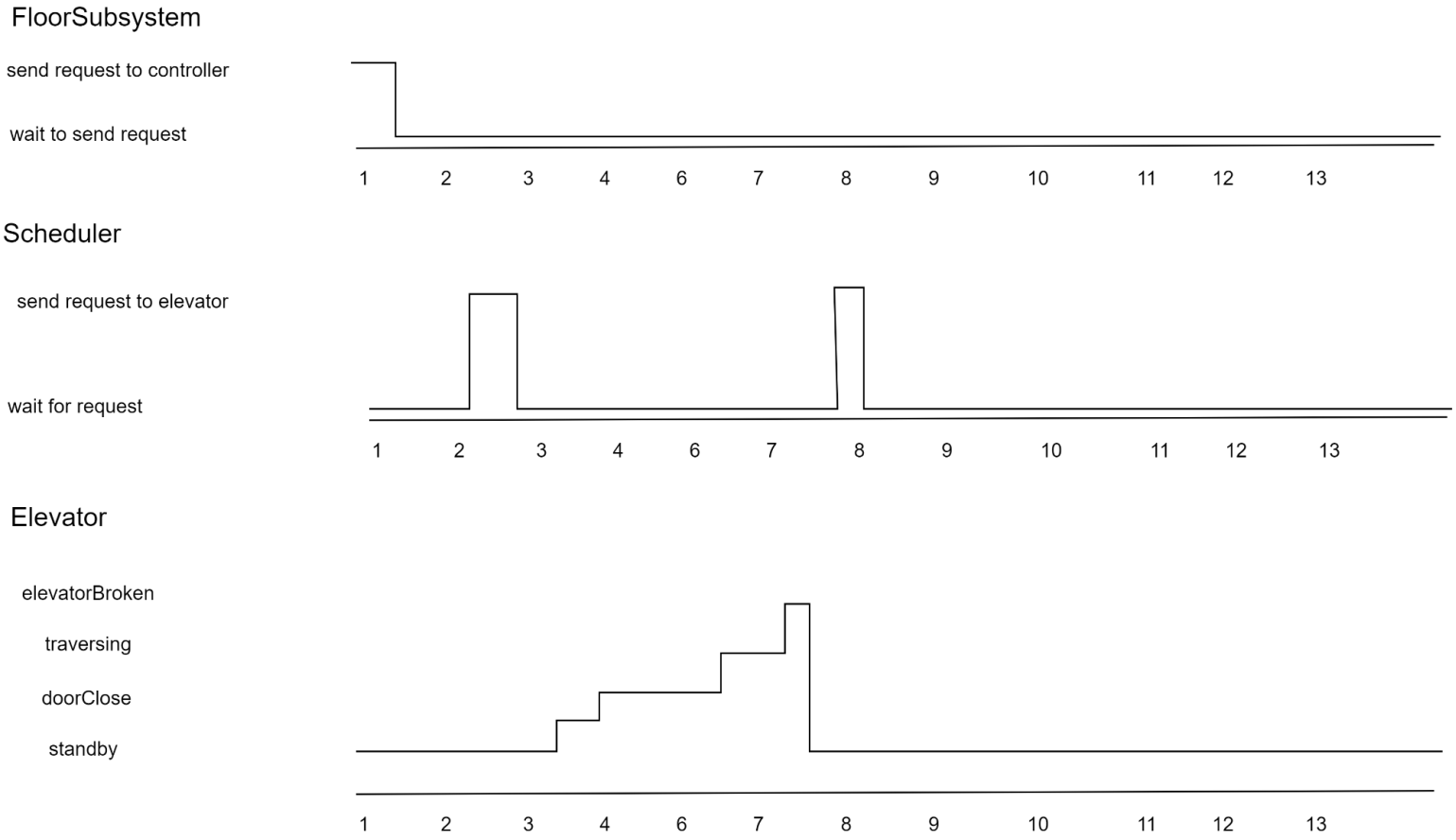
State Machine Diagram of the Scheduler



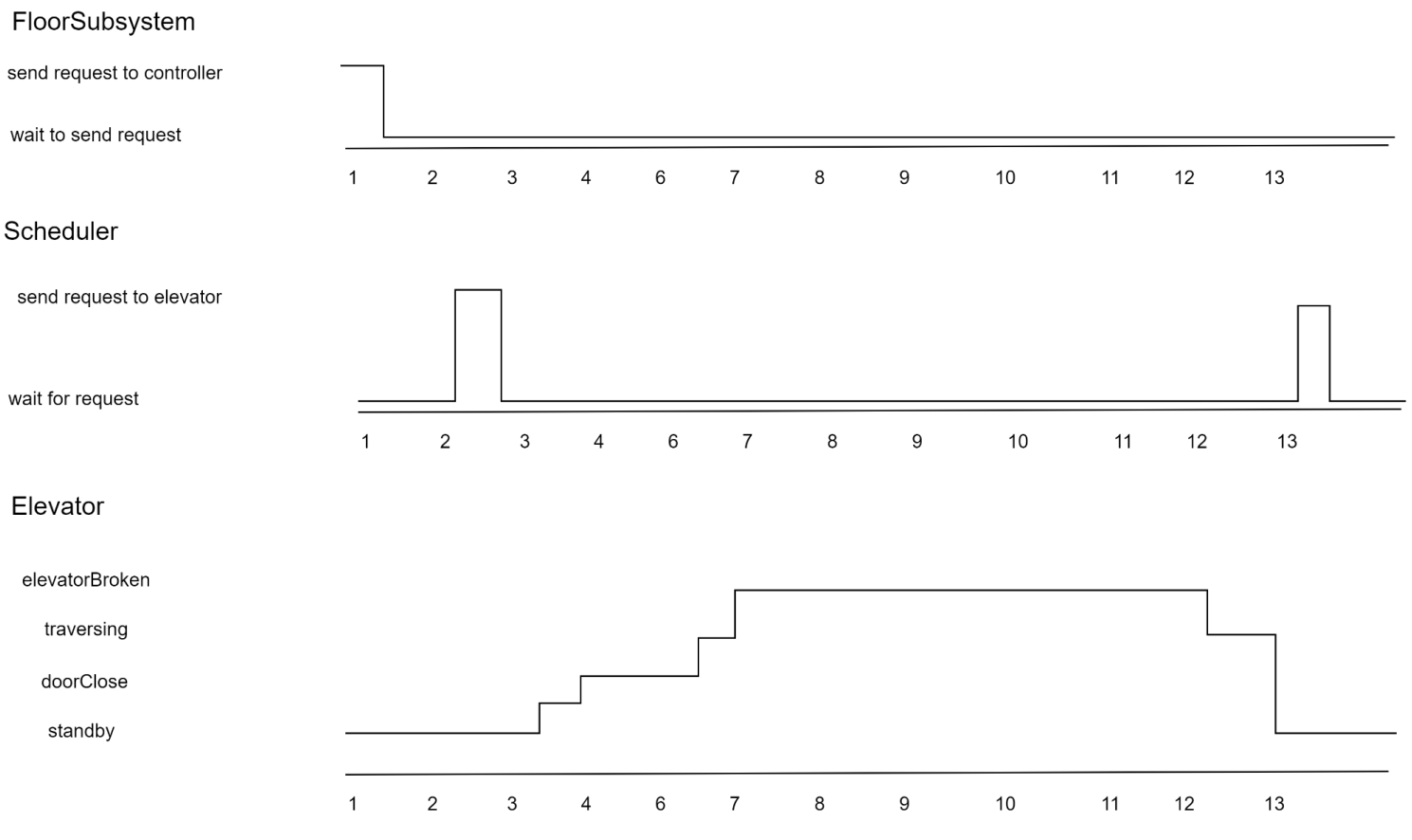
Timing Diagram



Timing diagram for broken Elevator fault



Timing diagram for broken Elevator door fault



Set up and Tests

1. Extract the zip file to your desktop.
2. In eclipse: File -> Import -> Projects from Folder or Archive ->
3. Click "archive" and select the zip file containing this README.md.
4. Click "Finished"

In order to run unit tests:
1. right click the "tests" package in the project explorer pane
2. click Run As -> JUnit Test

In order to run the program:

The scheduler, Elevator, FloorSubsystem can each be individually started by right clicking their respective .java files
and clicking Run As -> Java Application

The FloorSubsystem must be started last as the simulation begins when it's done initialising.

Alternatively, to run the program:
right click Program.java and click Run As -> Java Application

Measurements

These simulation performance metrics were read from the console output of the Scheduler. The program starts with the floorSubsystem saving the list of requests in the scheduler. The scheduler will then proceed with selecting the appropriate elevator and send the passenger request to the appropriate elevator. For each of the requests, the elevator will communicate its location and door status to keep track of its location. At the end of all requests,the system will output the total time for all requests to complete along with the average for each request.

```
The program took 92 seconds to run

AVG request time was 13 seconds
```

Reflection

Overall, the design had to be reworked to better allow adjustments and the implementation of added features. There were alot of issues initially with having a limited and few classes. The design was beneficial as it allowed us to implement controllers that were used solely for the communications between the different subsystems. By implementing controllers, it allowed us to quickly build up the features with minimal refactoring. If we were to redo this program, we would start with having controllers in place from the start so that when we do the UDP integration, there would be alot less refactoring and help keep the code organised.