Carleton University

Term Project Report

Course: SYSC 3303

Real Time Elevator System

Group 1

Team Members:

- Nathan Fohkens - 100946190
- Ehren Julien-Neitzert - 101046053
- Naomi Lui-Hing - 101040800
- Manel Oudjida - 100945382
- Chris Wang - 100951354

Table of Contents

1. Purpose:

The purpose of this project is to implement an elevator control system with a multithreaded simulator. The system will be consisted into three parts: an elevator controller which represents the scheduler, simulator for elevator car such as lights, doors, buttons... And a simulator for the floors – buttons, lights…

2. Breakdown of responsibilities

2.1. Iteration 1

Initiated project with subsystems for scheduler, elevator, and floor; along with communication sockets that will be used to communicate between each subsystem in all future iterations.

1. Nathan Fohkens:
   - Responsible for project design, design insights, and concurrence with iteration 0
   - Has begun to work on Iteration 2 implementation

2. Ehren Julien-Neitzert:
   - Responsible for project implementation, project validation, and design insights
   - EventReader.java, Elevator.java, TestCommunicationSocket.java, Javadoc

3. Naomi Lui-Hing:
   - Responsible for project implementation, project validation, and design insights
   - Floor.java, javadoc

4. Manel Oudjida:
   - Responsible for project documentation, class diagrams, and design insights
   - ClassDiagram.png, project structure insights, javadoc

5. Chris Wang:
   - Responsible for Project organization, project structuring, project architecture, and project implementation
   - Main.java, ElevatorSubsystem.java, FloorSubsystem.java, SchedulerSubsystem.java, CommunicationSocket.java, Event.java, TestEventReader.java, javadoc

2.2. Iteration 2

Included State Machines into the design of both the scheduler and elevator subsystems.

1. Nathan Fohkens:
   - Responsible for Elevator Subsystem state machine design, Floor Subsystem state machine design and Scheduler Subsystem state machine.
   - Cooperative State Machine.png

2. Ehren Julien-Neitzert:
   - Responsible for the implementation of the Elevator Subsystem state machine design, Floor state Subsystem machine design and Scheduler Subsystem state machine design.
   - Package: scheduler floor, event, event.toScheduler

3. Naomi Lui-Hing:
   - Responsible for Elevator Subsystem state machine design, Floor Subsystem state machine design, project documentation
   - Javadoc, tests

4. Manel Oudjida:
   - Responsible for Elevator Subsystem state machine design, project documentation, sequence diagram and UML class diagram.
   - Javadoc , sequenceDiagram.png , UMLClassDiagram.png

5. Chris Wang:
   - Responsible for the implementation of the Elevator Subsystem state machine design, Floor state Subsystem machine design and Scheduler Subsystem state machine design.
   - Package: elevator, event.toElevator, floor, event, event.toScheduler, test

2.3. Iteration 3

Included multiple threads in the Scheduler, so the scheduler can handle event requests asynchronously.

1. Nathan Fohkens:
   - Responsible for Scheduling Algorithm
   - Package: scheduler

2. Ehren Julien-Neitzert:
   - Responsible for the implementation of the Scheduler side of the RPC, and the Serialization and Datagram/socket stuff.
   - Package: RPC Receiver, network, common, elevator, event, event.toElevator, event.toScheduler, floor, scheduler

3. Naomi Lui-Hing:
   - Responsible for Serialization and DatagramSocket implementations, testing, and project documentation.
   - Package: floor, elevator, network, tests

4. Manel Oudjida:
   - Responsible for testing, sequence diagram, readme and UML class diagram
   - Tests, Javadoc, sequenceDiagram.png , UMLClassDiagram.png

5. Chris Wang:
   - Responsible for the implementation of the Elevator side of the RPC, and testing.
   - Package: RPC Sender, test

2.4. Iteration 4 & 5

Included explicit fault (failure) state and fault reporting within the elevator state machine design to handle impossible cases.

1. Nathan Fohkens:
   - Responsible for project documentation, design insights, testing
   - Javadoc, Final report

2. Ehren Julien-Neitzert:
   - Responsible for project documentation, design insights, tests, recording of final working system for submission
   - Javadoc,

3. Naomi Lui-Hing:
   - Responsible for project documentation, class diagrams, design insights, Final Report
   - UpdatedClassDiagram.png, Javadoc, Final report

4. Manel Oudjida:
   - Responsible for project documentation, class diagrams, design insights, Final Report
   - UpdatedClassDiagram.png, Javadoc, Final report

5. Chris Wang:
   - Responsible for project documentation, design insights, fault handling, testing
   - Javadoc, ElevatorState.java, ElevatorFailureState.java, TestElevatorFault.java

3. Diagrams

All Diagrams discussed below can be found in their full size at the end of this document in *Appendix A: Full Sized Diagrams* starting on Page 14.

3.1. UML Class Diagram

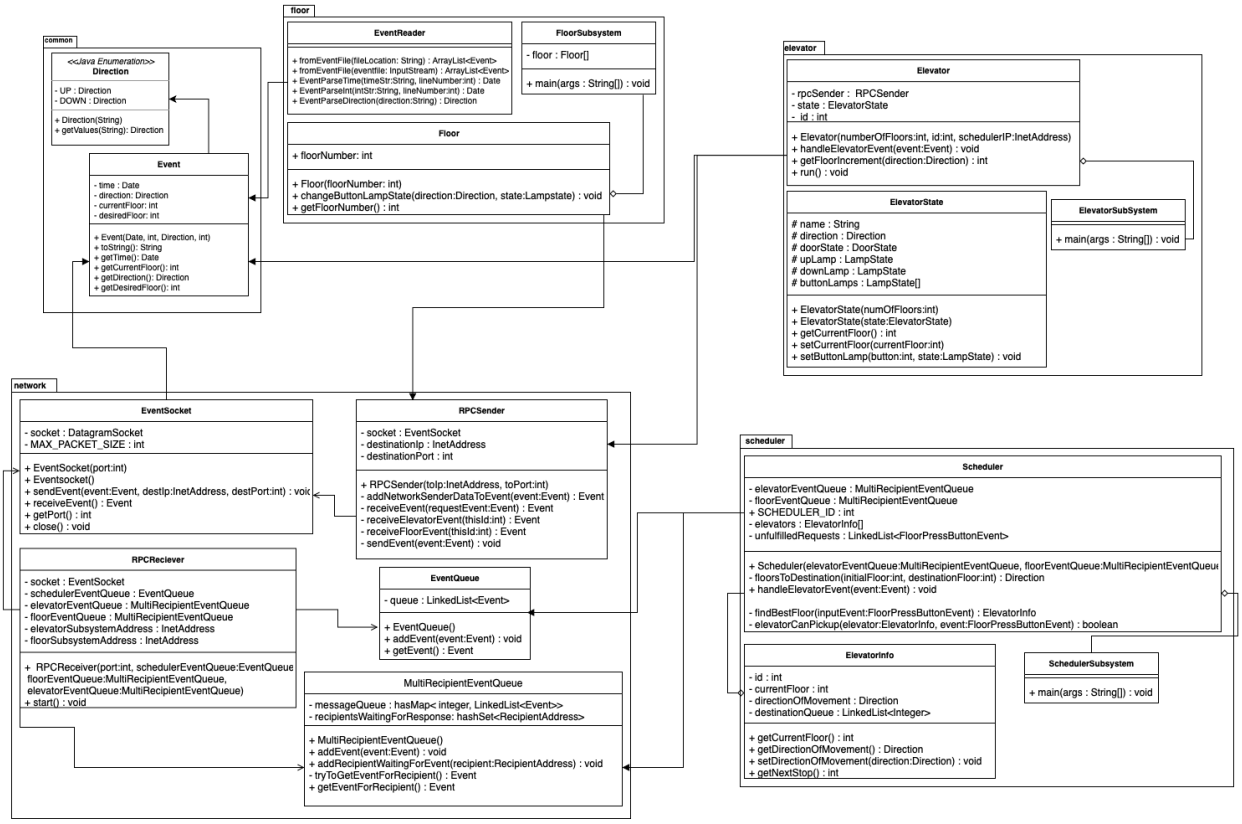


Figure 1: UML Class Diagram of Elevator Project

3.2. State Machine Diagrams

3.2.1. Elevator State Machine

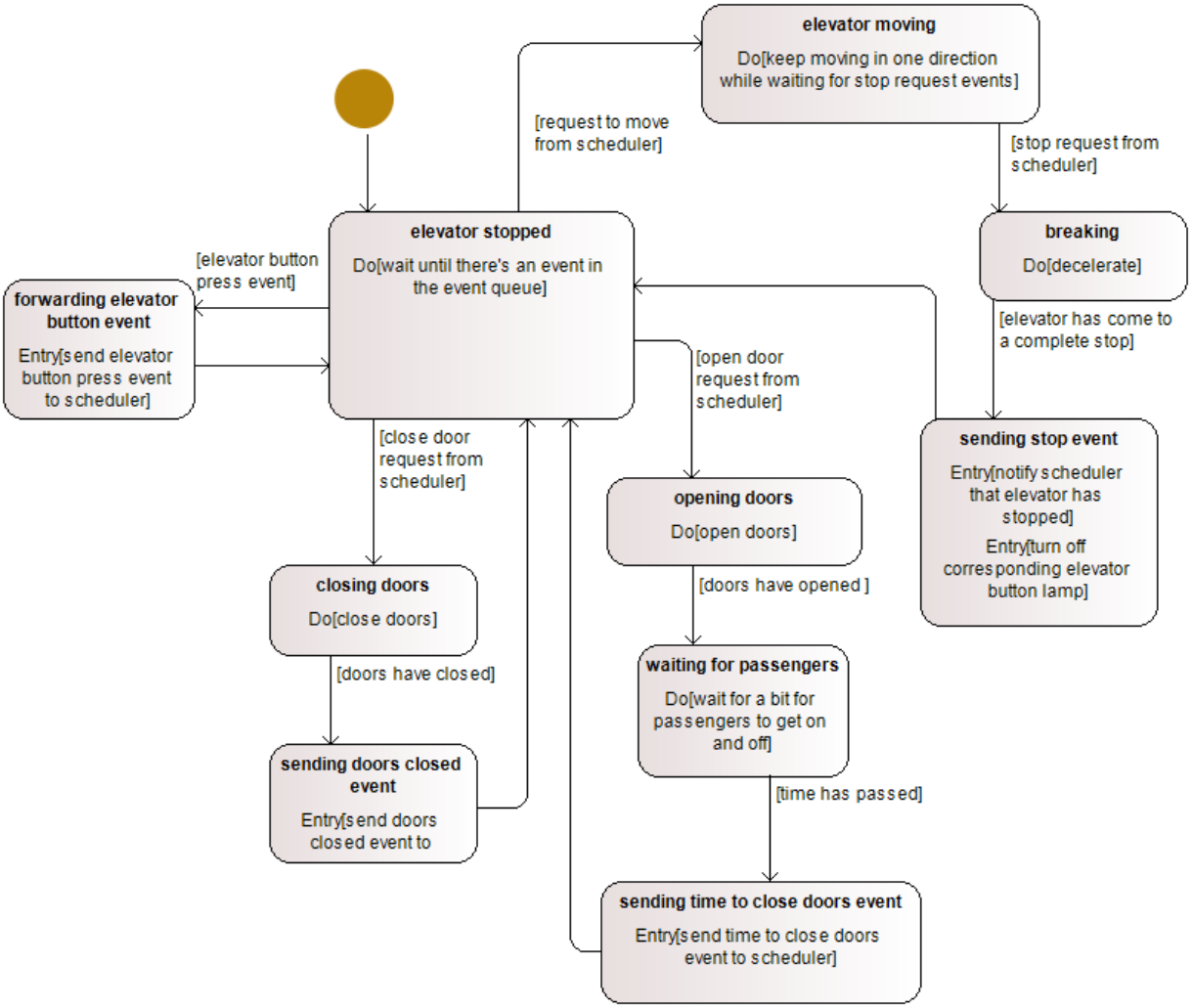


Figure 2: State Machine of Elevator Subsystem

3.2.2. Scheduler State Machine

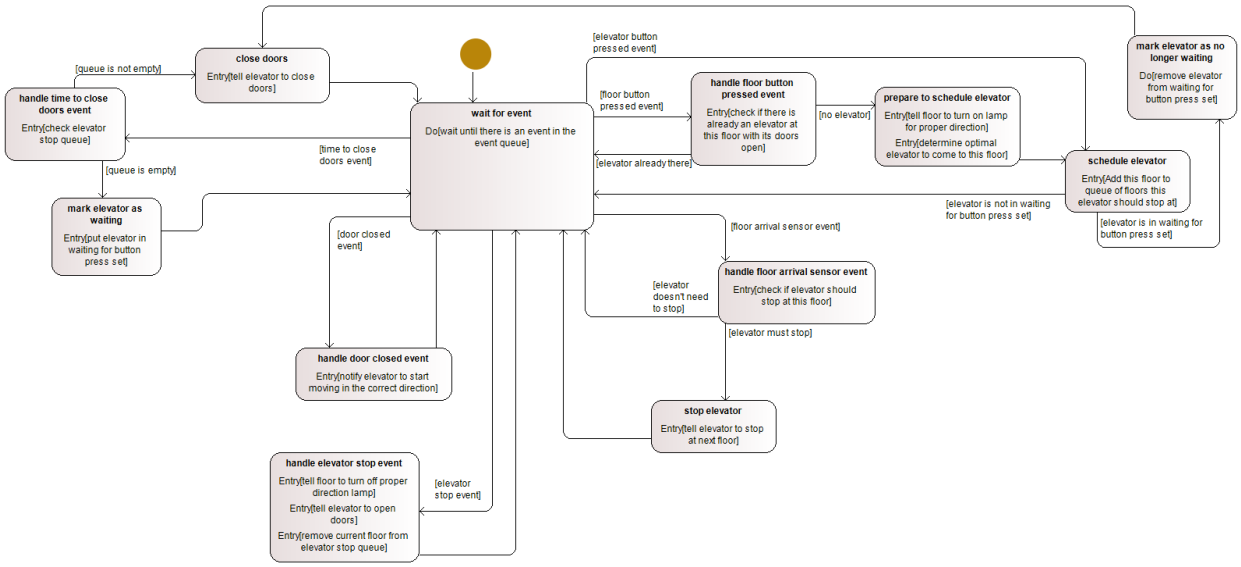


Figure 3: State Machine of Scheduler Subsystem

3.2.3. Cooperative State Machine

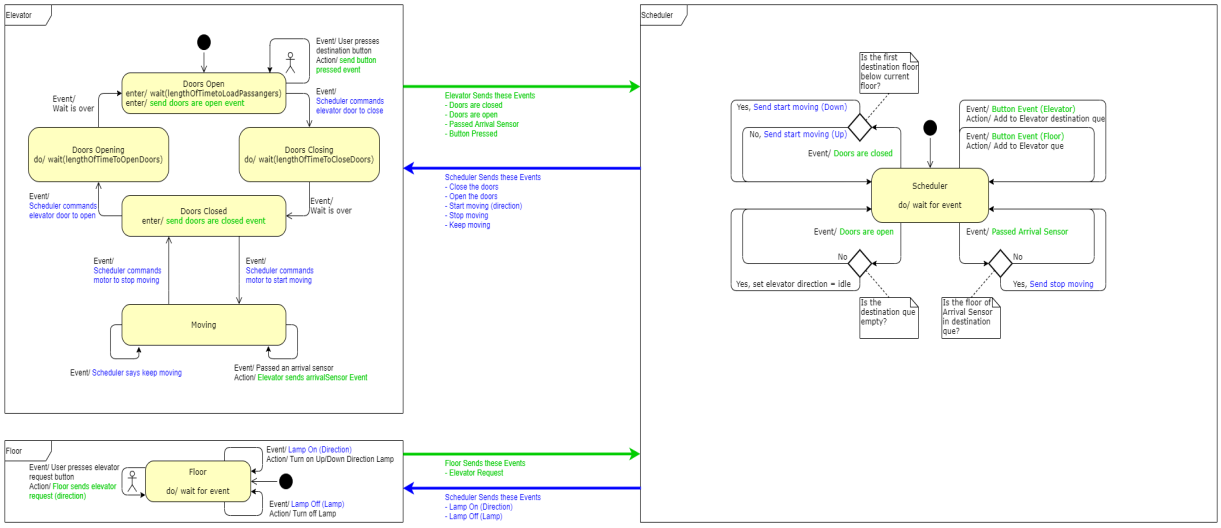


Figure 4: Cooperative State Machine of Entire Project

3.3. Sequence Diagrams

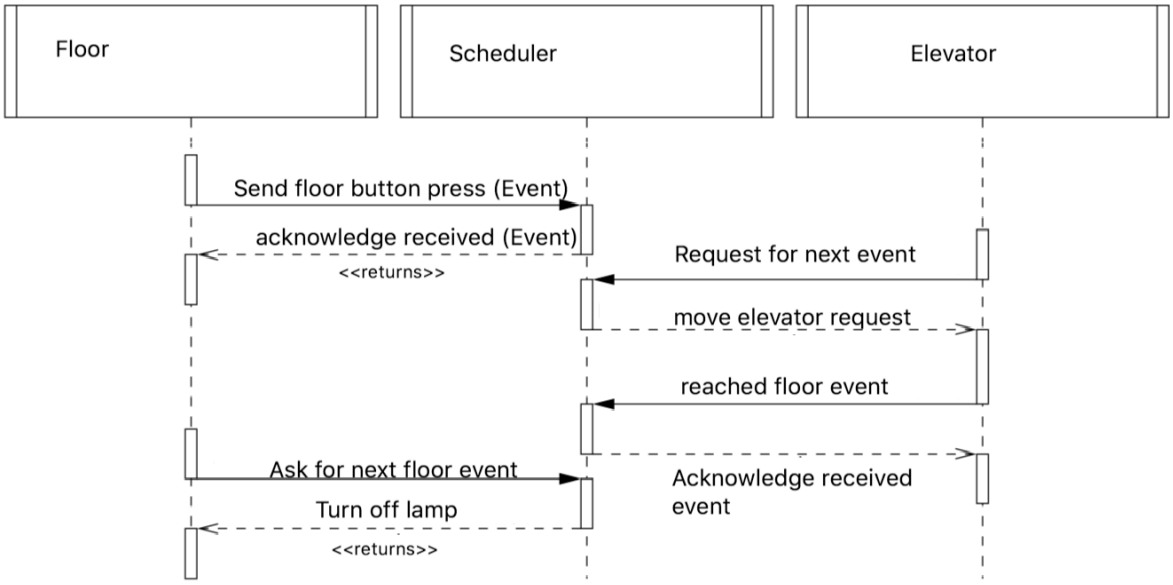


Figure 5: Sequence Diagram of Elevator System

3.4. Timing Diagram

Timing Diagram excluded from this report per instructor's direction:
"*There is no need for a GUI or timing as you did not have to instrument your code.*"

4. Set up and Test Instructions

1. Import project from archive file into Eclipse

2. From Eclipse
    2.1 Select 'File' menu
    2.2 Select 'Import' menu item
    2.3 Under 'General' select 'Projects From File System or Archive File'
    2.4 Select 'Archive...'
    2.5 Locate 'Real-Time-Systems-Project.zip' as Import source.
    2.6 Click 'Finish'

3. Set up input file:
    3.1 Rename the text file that contains all the events to read from to floorEvents.tsv (an example is provided)
    3.2 Place floorEvents.tsv in the project root (similar to the example)
    3.3 floorEvents.tsv does not need to be a tab separated file, any whitespace character separation will work
    3.4 However the extension of floorEvents.tsv MUST be a .tsv file (again please follow the example)
    3.5 This workaround is because of how we are supposed to ingest the event file was never explicitly specified

4. From within the project "Real-Time-Systems-Project"
    4.1 Right click on the SchedulerSubsystem.java in the Scheduler package in Eclipse IDE
    4.2 Select "Run As" then "Java Application" to run the application
    4.3 Right click on the ElevatorSubsystem.java in the Elevator package in Eclipse IDE (on any computer)
    4.4 Select "Run As" then "Java Application" to run the application (You will need to enter the IP of the Scheduler which will appear in the console when you run the SchedulerSubsystem)
    4.5 Right click on the FloorSubsystem.java in the Floor package in Eclipse IDE (on any computer)
    4.6 Select "Run As" then "Java Application" to run the application (You will need to enter the IP of the Scheduler which will appear in the console when you run the SchedulerSubsystem)
    4.7 Right click on the tests package in Eclipse IDE
    4.8 Select "Run As" then "JUnit Test" to run the tests of the application

5. Important File Explanations

5.1. Packages

Common: Contains classes shared by multiple systems/subsystems.

Elevator: Contains classes used by the elevator subsystem.

Event: Contains classes that represent Events passed between subsystems.

Floor: Contains classes used by the floor subsystem

Main: Legacy code from when the code ran in one main function. Can be ignored.

Network: Contains classes that are used to communicate over the local network via remote procedure calls

Scheduler: Contains classes that are used by the scheduler subsystem

Tests: Contains test cases to ensure the entire system is functioning properly. Uses JUnit 4.

5.2. Classes

ElevatorSubsystem.java:
Entry point for the elevator subsystem

FloorSubsystem.java:
Entry point for the floor subsystem

SchedulerSubsystem.java:
Entry point for the floor subsystem

6. Measurement Results

Measurement Results for scheduler excluded from this report per instructor's direction:
"*There is no need for a GUI or timing as you did not have to instrument your code.*"

7. Reflection on the design

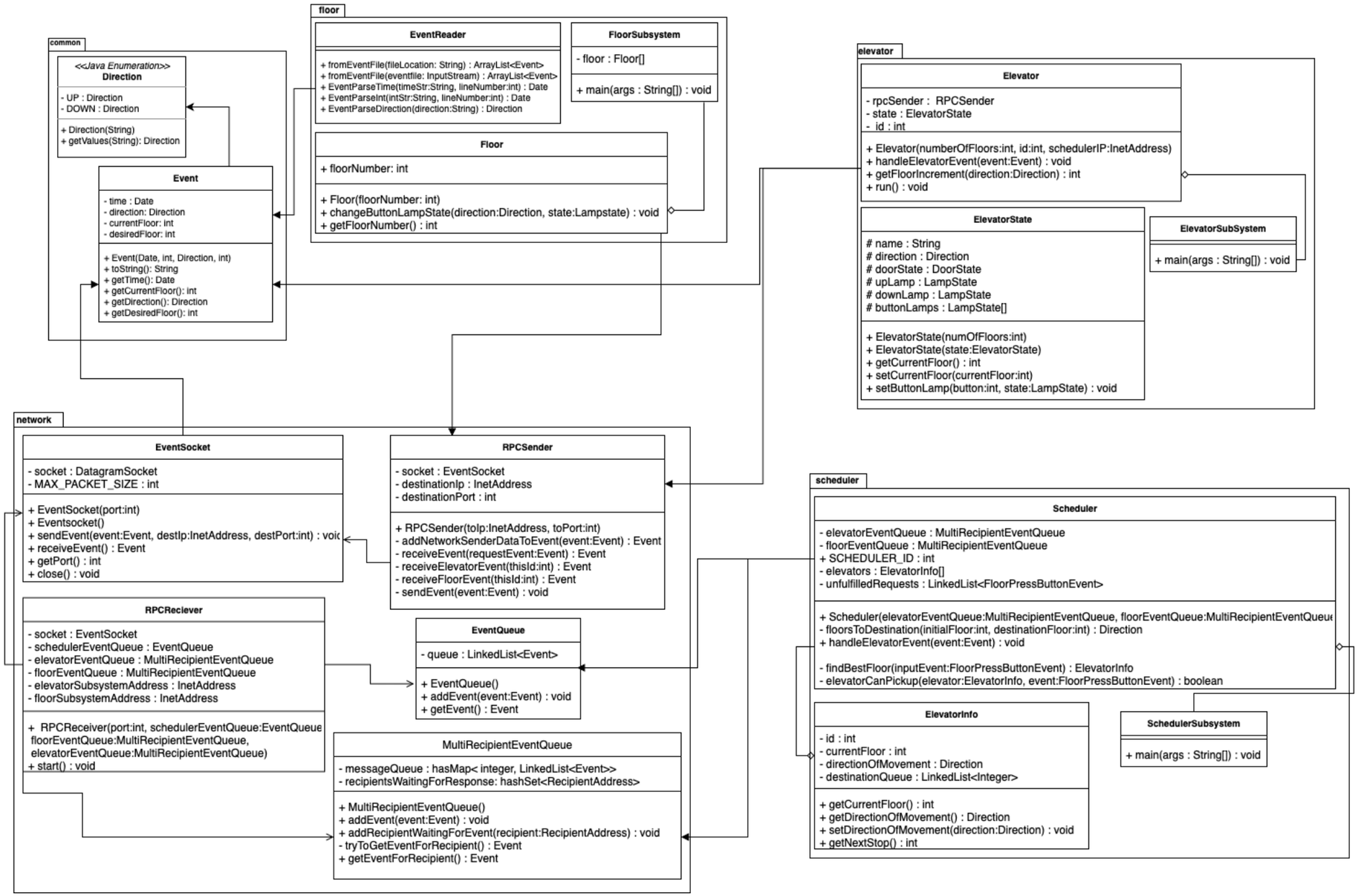
As a group, we are content with the design choices made throughout the term regarding this project. The elevator state machine and the ThreadPrinter class were both items that we found to be both well deigned and well implemented in the code as written; meanwhile the Scheduler and Event classes were already rewritten once and could still benefit for additional changes.

We found that the success of our Elevator State Machine began with the guidelines we used regarding state patterns in the book *Design Patterns: Elements of Reusable Object-Oriented Software* by Gamma et al. The implementation of these state patterns resulted in a more modular and extendible code; as such, only minor refactoring was required to implement a fault state for illegal transitions in Iteration 4. We were also very pleased with our ThreadPrinter class; enabling us to easily print to the console while keeping all print statements separated by thread, making debugging easier and the overall output easier to read.

Throughout the duration of this project we made substantial changes to both the Scheduler and the Event classes. We were often found it difficult to trace code in these particular classes, and often had to ask each other in order to determine what each function is supposed to do. If we were to re-write the entire project, the scheduler would be the first thing to be re-written.

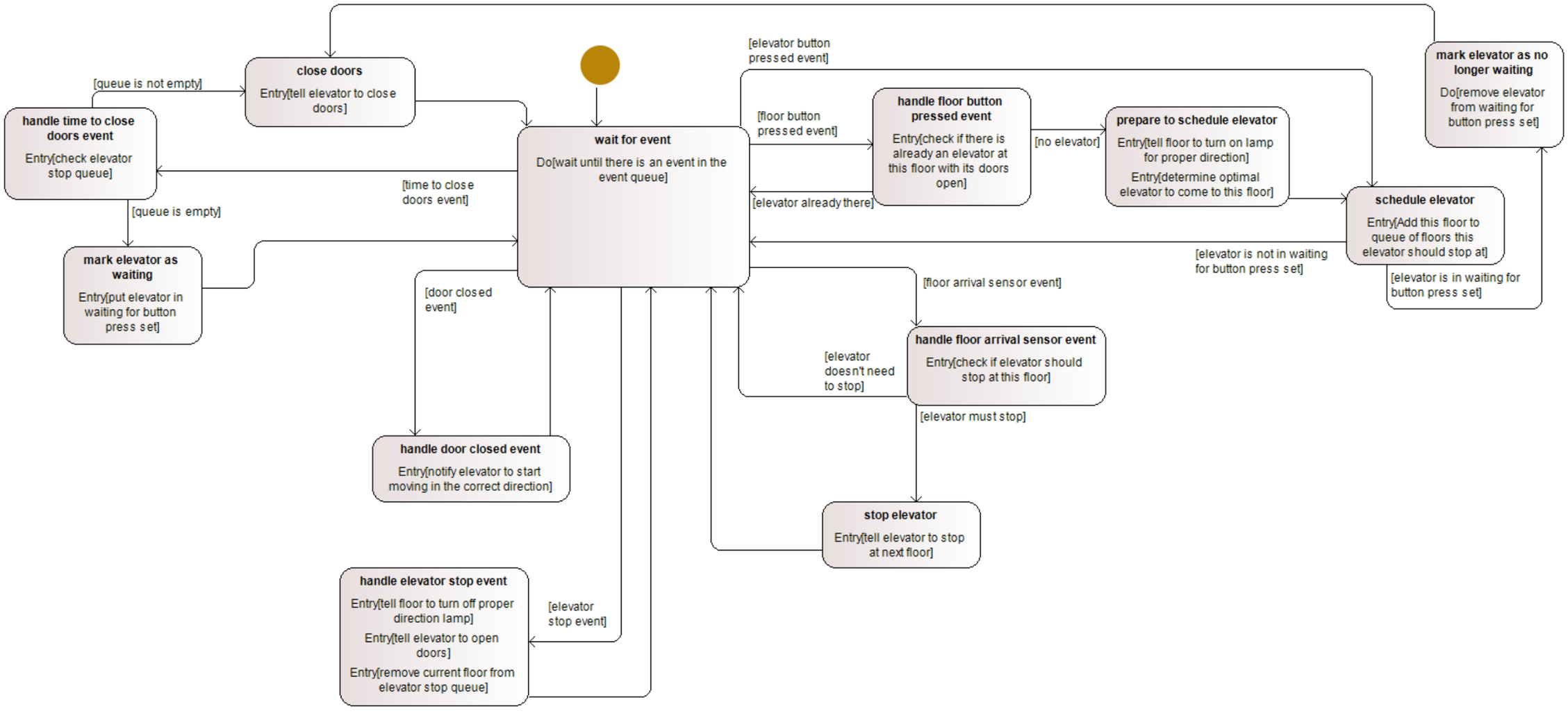
Appendix A. Full Sized Diagrams

Appendix A.1: Figure 1: UML Class Diagram

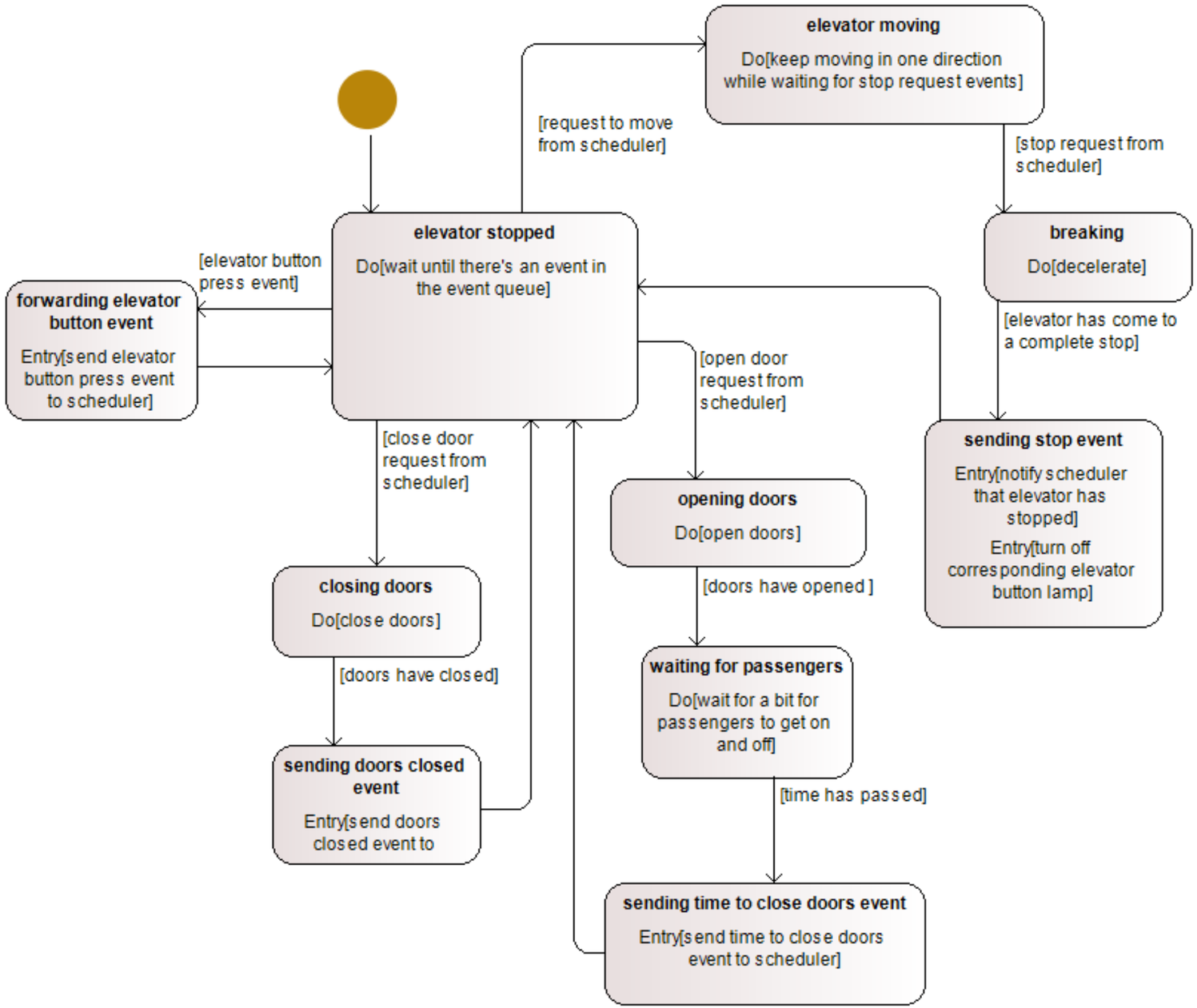


Appendix A.2: **State Machine Diagrams**

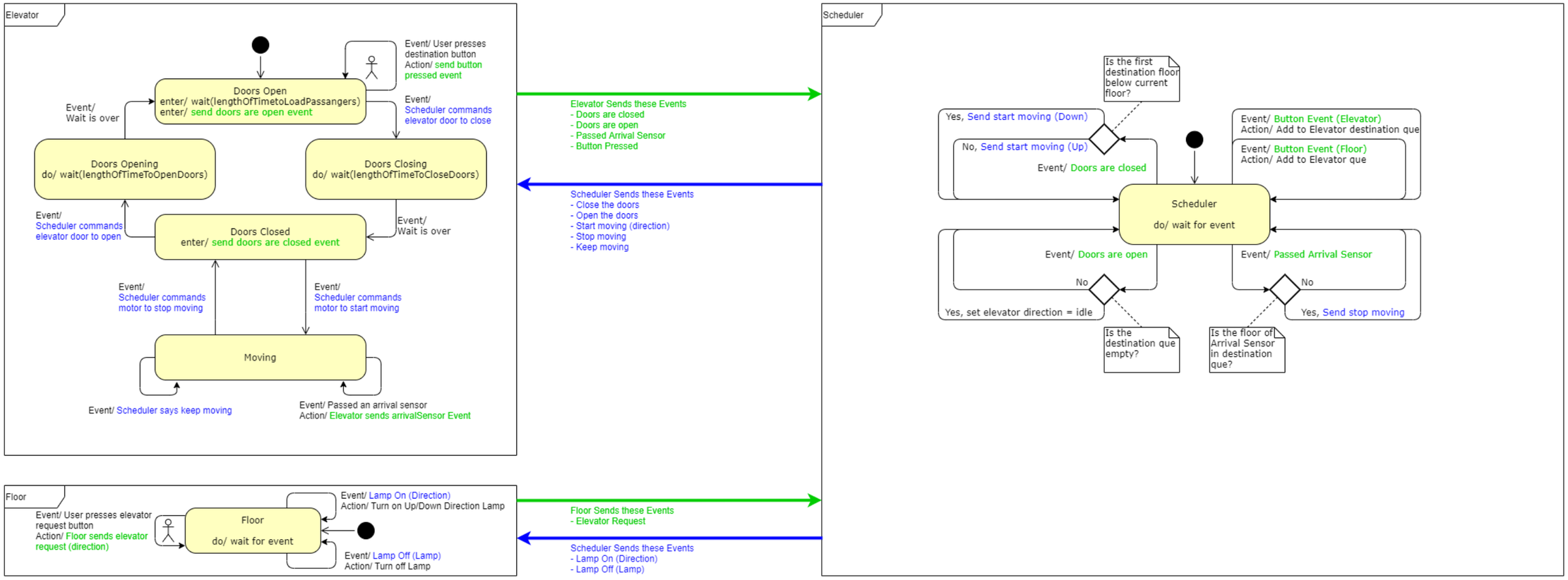
Appendix A.2.1: Figure 2: Elevator State Machine



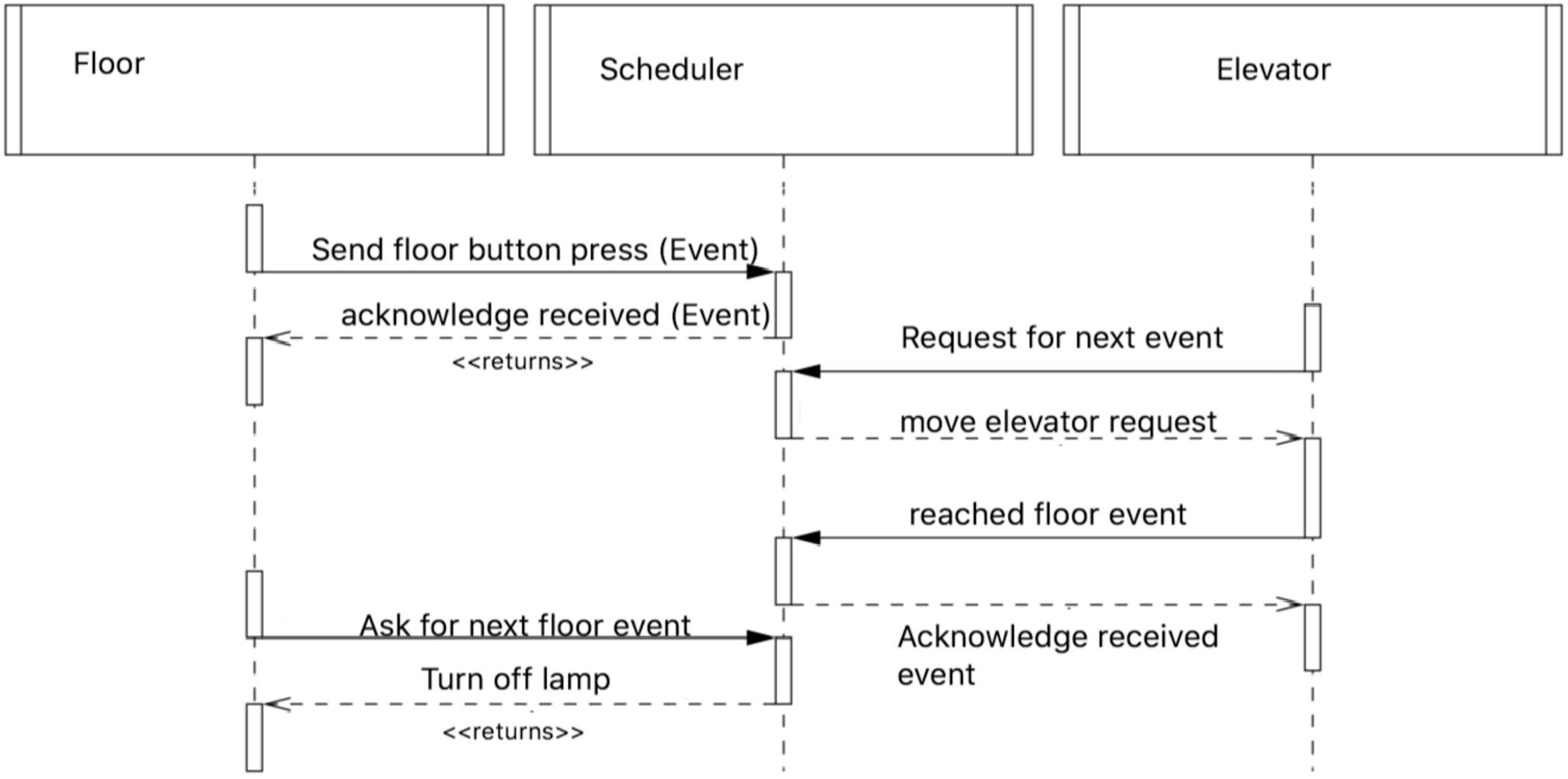
Appendix A.2.2: Figure 3: Scheduler State Machine



Appendix A.2.3: Figure 4: Cooperative State Machine



Appendix A.3: **Figure 5: Sequence Diagrams**



Appendix A.4: **Timing Diagram**

Timing Diagram excluded from this report per instructor's direction:
"*There is no need for a GUI or timing as you did not have to instrument your code.*"