Elevator Control System and Simulator

Final Project Submission

Jayson Mendoza, Jean-Pierre Aupont, and Jordan King

Lab C2 - Group 8

Department of Engineering, Carleton University

SYSC 3303 C: Real Time Concurrent Systems

Dr. Gregory Franks

April 12, 2023 @ 11:59pm

Table of Contents

Breakdown of Responsibilities

General Breakdown

Jayson	Jean-Pierre	Jordan
- All things FloorSystem and FloorSubsystem related (including input file handling) - All things Dispatcher related (UDP Networking) - All GUI research - Most of the GUI implementation - GUI Visuals/Symbols	- All things ElevatorSystem and ElevatorSubsystem related - Most things UML related - Additional JUnit testing - Some of the GUI Implementation (work on the middle view update panels) - Most of the GUI data display	- All things Scheduler and SchedulerAlgorithm related - All things related to general debugging and correctness between systems - All system measurements - Most of the report - Sending data to GUI

Iteration 0

Jayson	Jean-Pierre	Jordan
- Brainstorming + data collection (video recordings of elevators) - Report writing (creating a report and inputting analysed statistical data)	- Brainstorming + data collection (video recordings of elevators) - Data analysis (handling the relevant statistical calculations)	- Brainstorming + data collection (video recordings of elevators) - Data extraction (putting timestamps from the videos into spreadsheets)

Iteration 1

Jayson	Jean-Pierre	Jordan
- Brainstorming + design - UML Class diagram	- Brainstorming + design - UML Sequence diagram	- Brainstorming - Code (following designs and diagrams)

Iteration 2

Jayson	Jean-Pierre	Jordan
- Brainstorming + design - UML Class diagram (Floor System) - Code + JUnit Tests (Floor System, UDP networking	- Brainstorming + design - UML Class diagram (Elevator System + general organisation/maintenance) - Code + JUnit Tests	- Brainstorming + design - UML Class diagram (Scheduler) - UML State Machine diagram

Dispatcher)	(Elevator System)	- Code + JUnit Tests (Scheduler, general system debugging and correctness)

Iteration 3

Jayson	Jean-Pierre	Jordan
- Brainstorming + design - UML Class diagram (Floor System) - UML Sequence diagram - Code + JUnit Tests (Floor System, UDP networking over 3 processes)	- Brainstorming + design - UML Class diagram (Elevator System + general organisation/maintenance) - Code + JUnit Tests (Elevator System + additional Scheduler State JUnit tests)	- Brainstorming + design - UML Class diagram (SchedulerAlgorithm) - Code + JUnit Tests (Scheduler/Algorithm, general system debugging and correctness)

Iteration 4

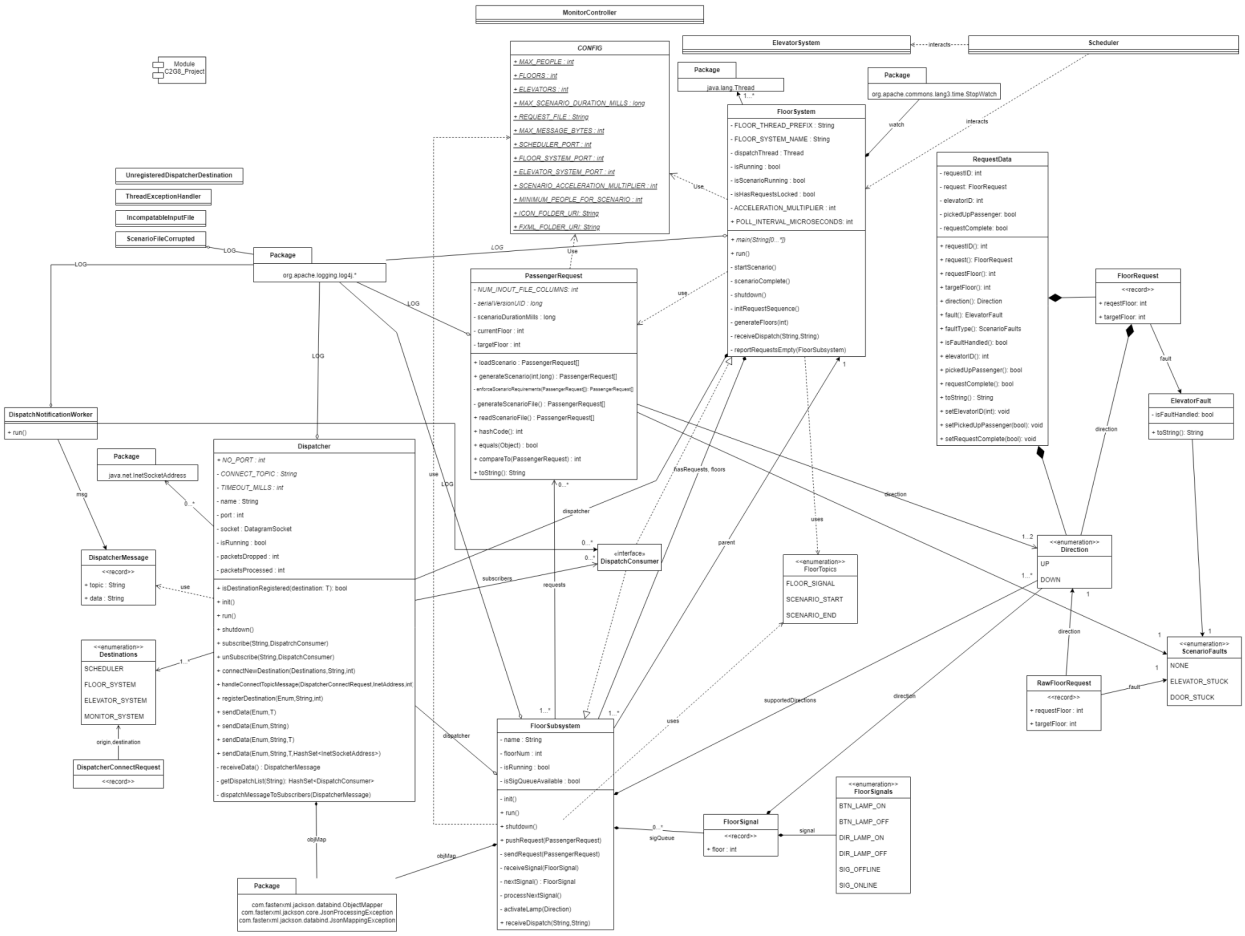
Jayson	Jean-Pierre	Jordan
- Brainstorming + design - UML Class diagram (Floor System) - Code + JUnit Tests (Floor System) - GUI Research and Implementation (for next iteration)	- Brainstorming + design - UML Class diagram (Elevator System + general organisation/maintenance) - UML Sequence diagram - UML Timing diagram - Code + JUnit Tests (Elevator System + JUnit tests for Elevator Errors)	- Brainstorming + design - UML Class diagram (SchedulerAlgorithm) - Code + JUnit Tests (Scheduler/Algorithm, general system debugging and correctness)

Iteration 5 (Final Submission)

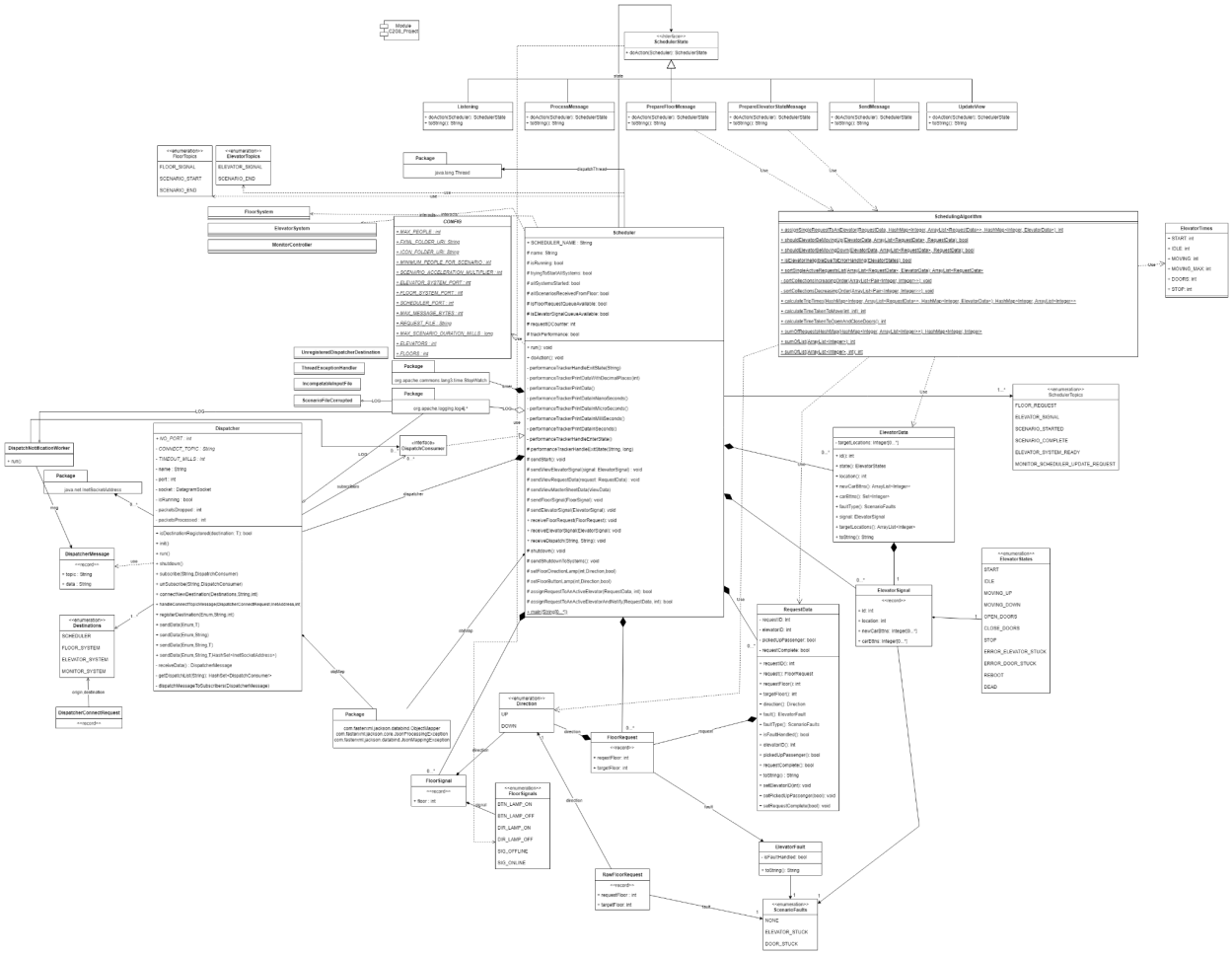
Jayson	Jean-Pierre	Jordan
- GUI research - GUI implementation - GUI Visuals/Symbols	- Organization and final cleanup of all UML diagrams (Class, Sequence, Timing) - GUI implementation - GUI Floor Request List	- All system measurements - Report - Sending data to GUI

UML Diagrams

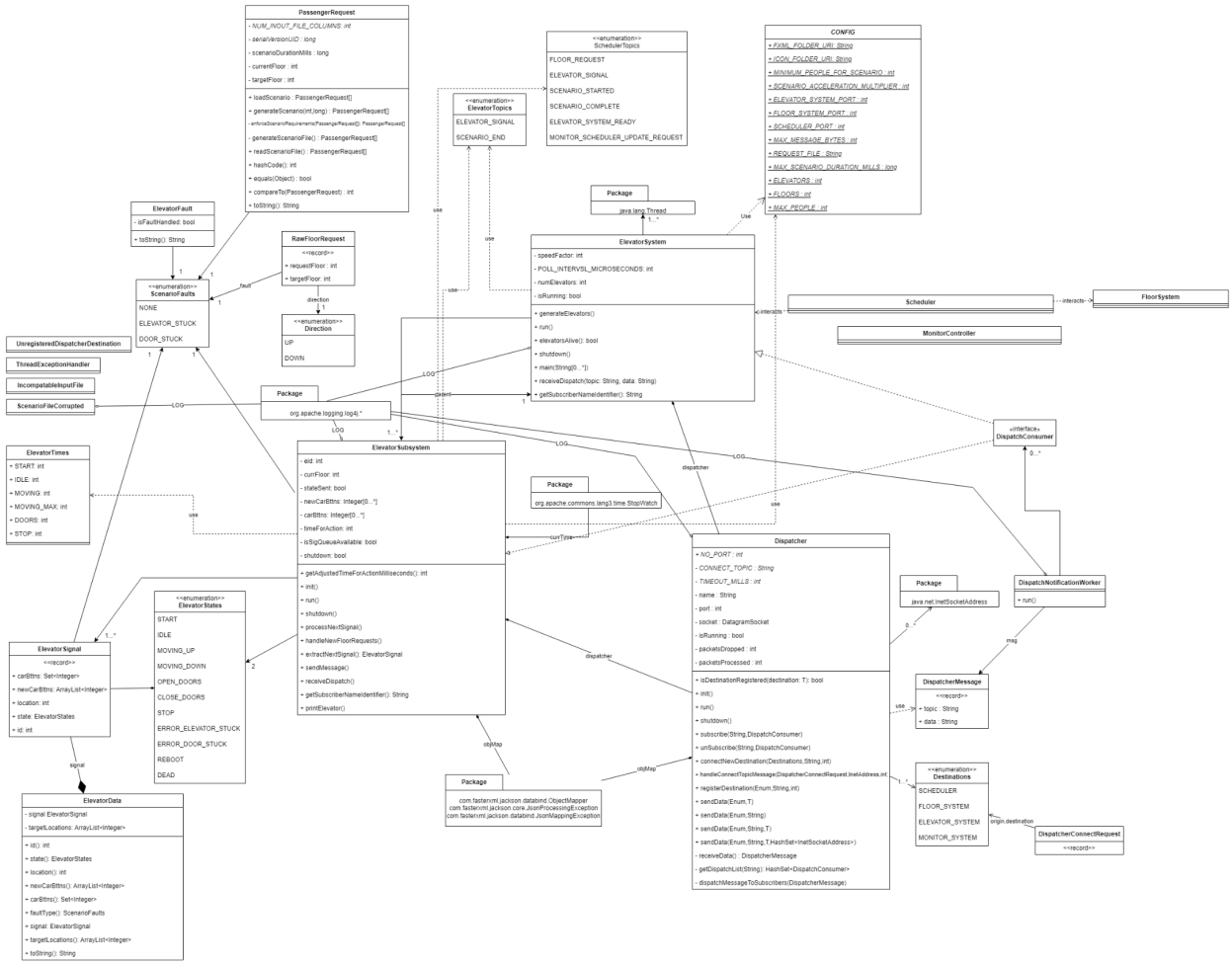
UML Class Diagram - Floor System (Full)



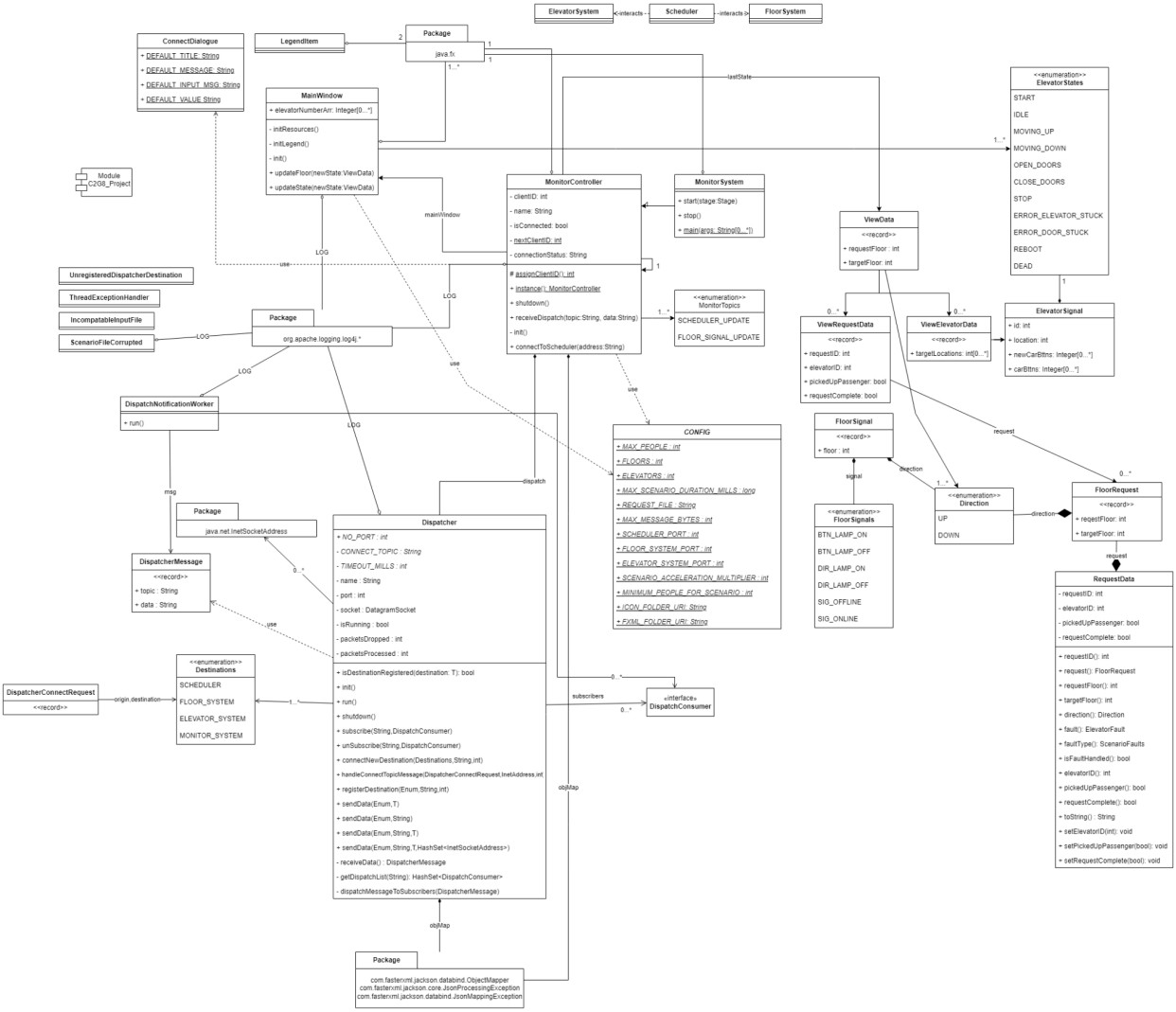
UML Class Diagram - Scheduler (Full)



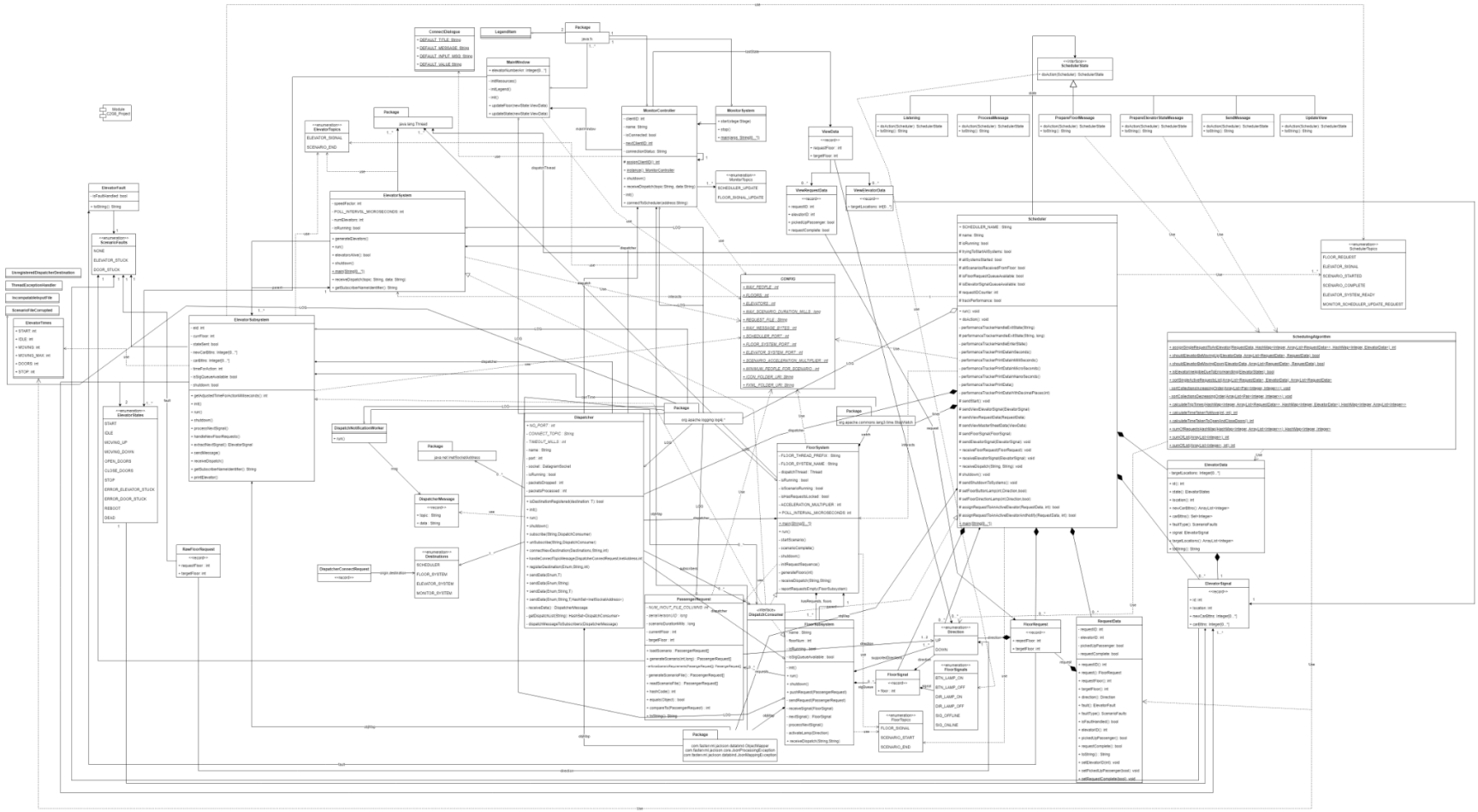
UML Class Diagram - Elevator System (Full)



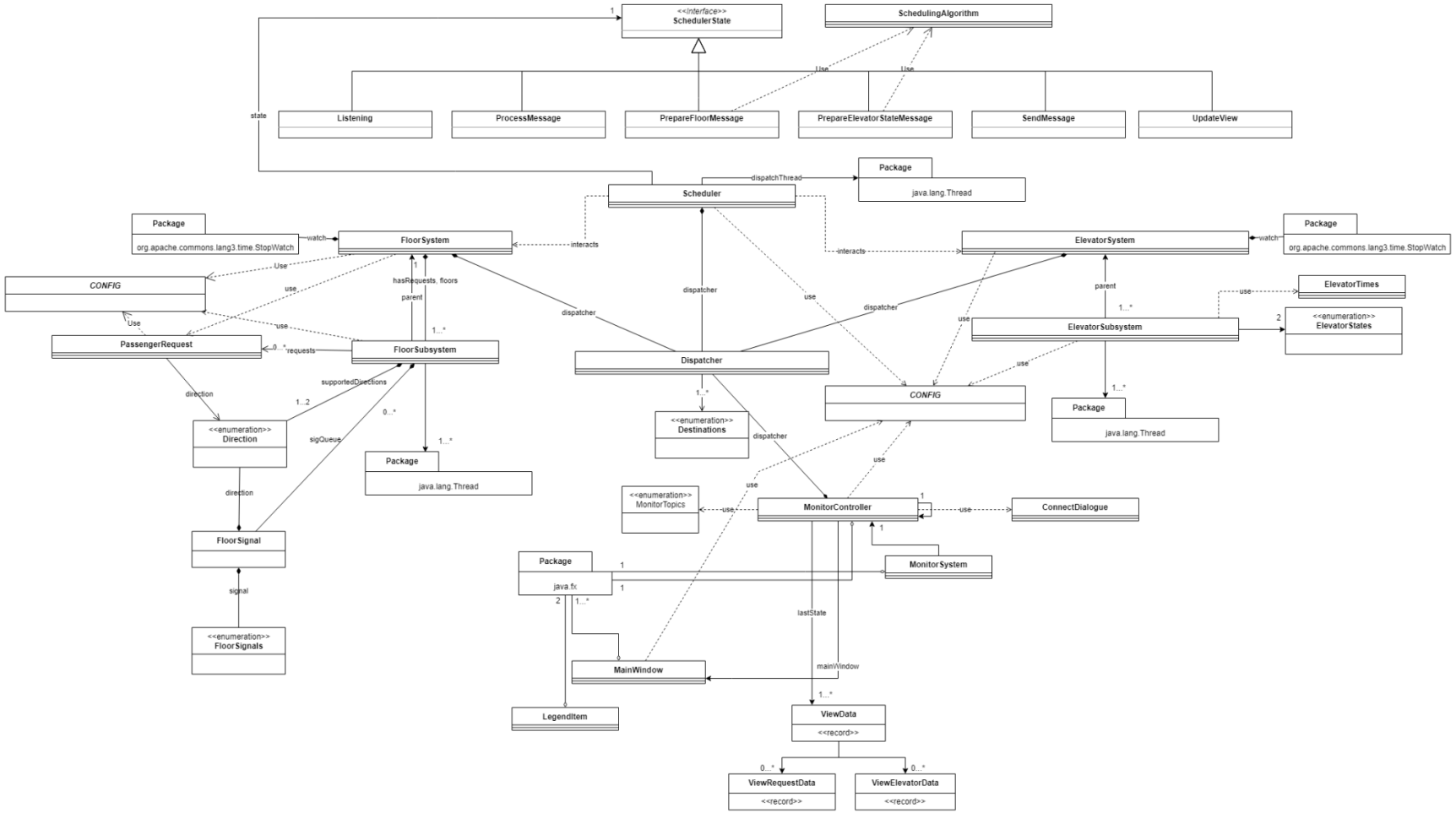
UML Class Diagram - Monitor (Full)



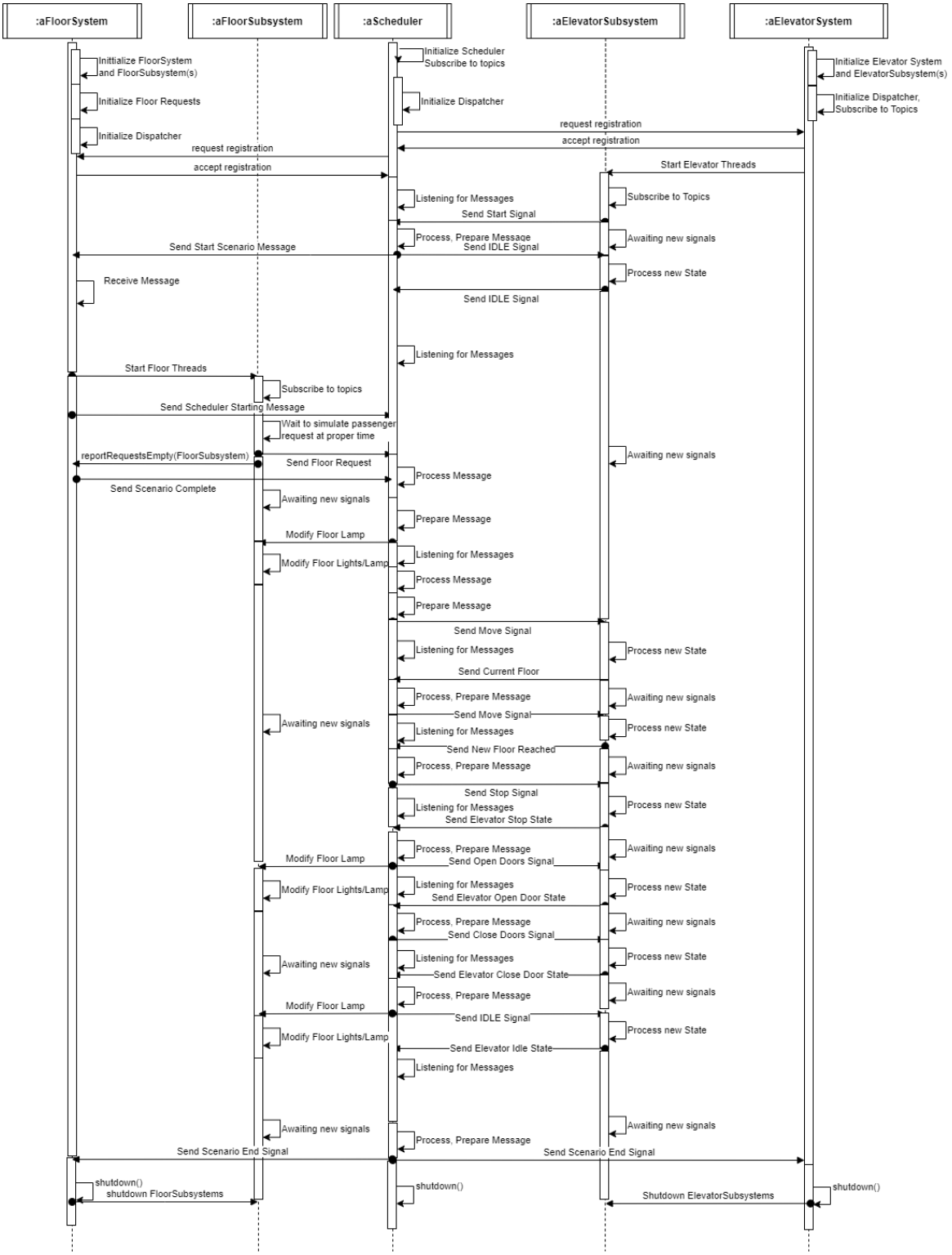
UML Class Diagram - Everything (All 3 Systems)



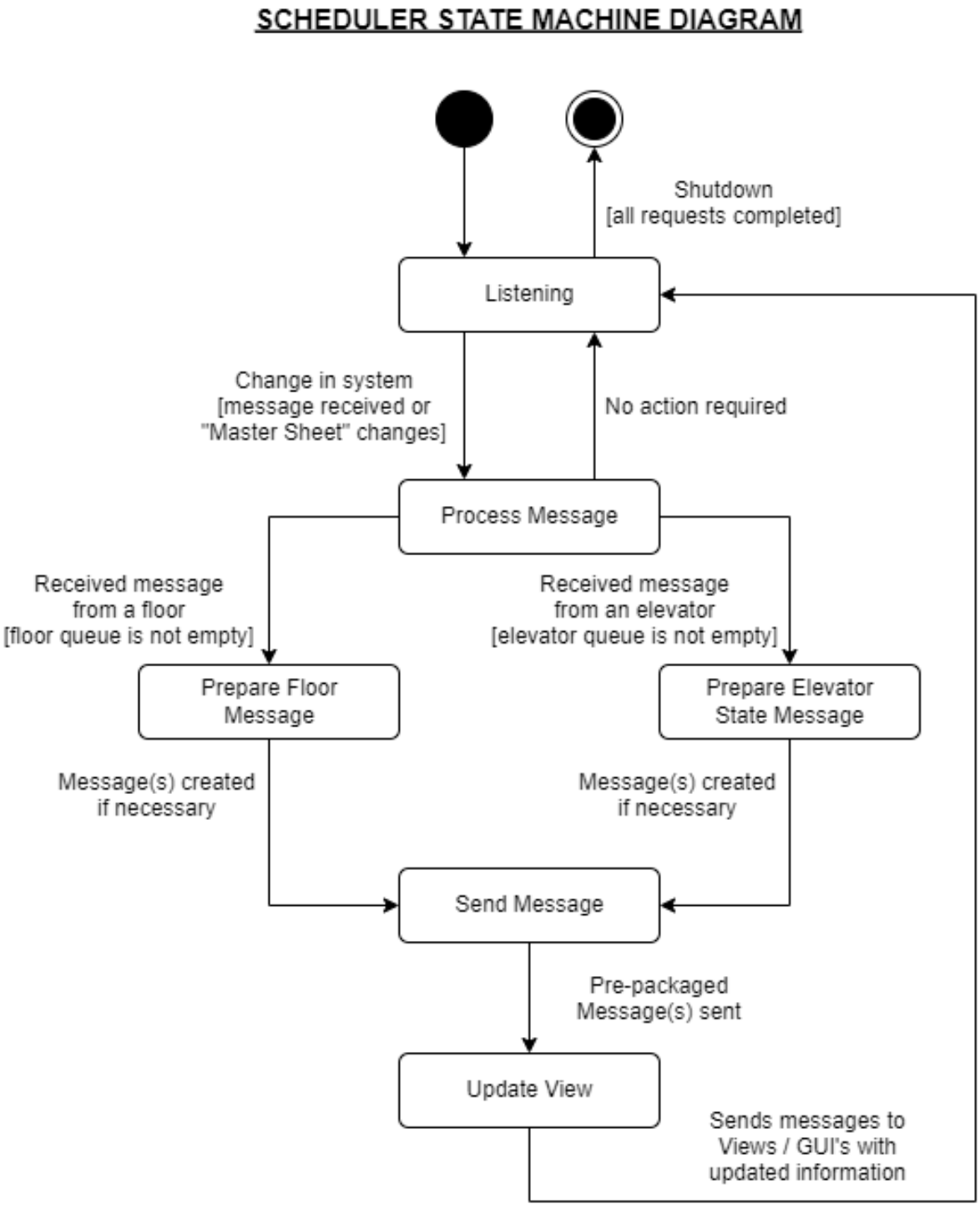
UML Class Diagram - Simplified (Important Class Names Only)



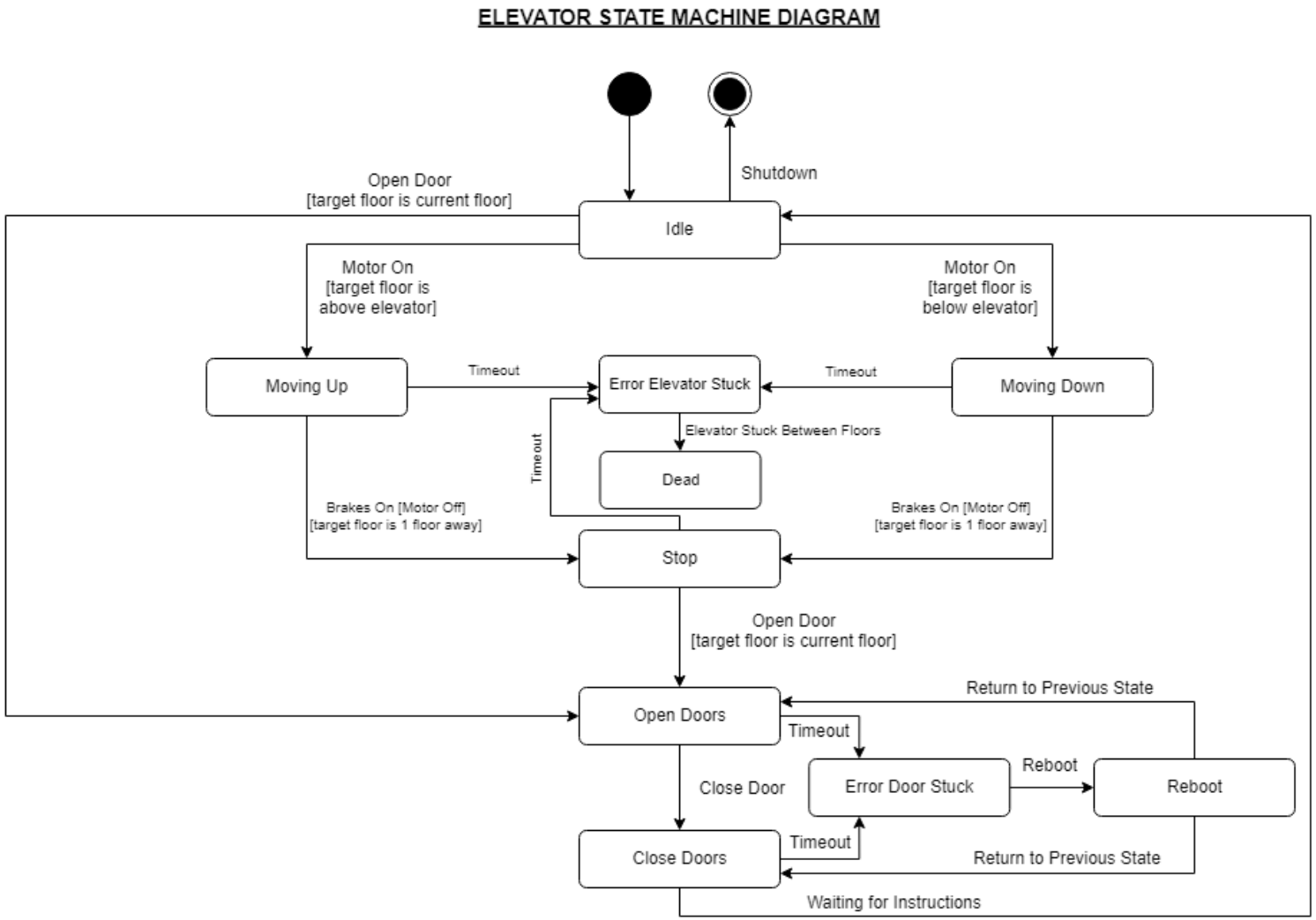
UML Sequence Diagram - System



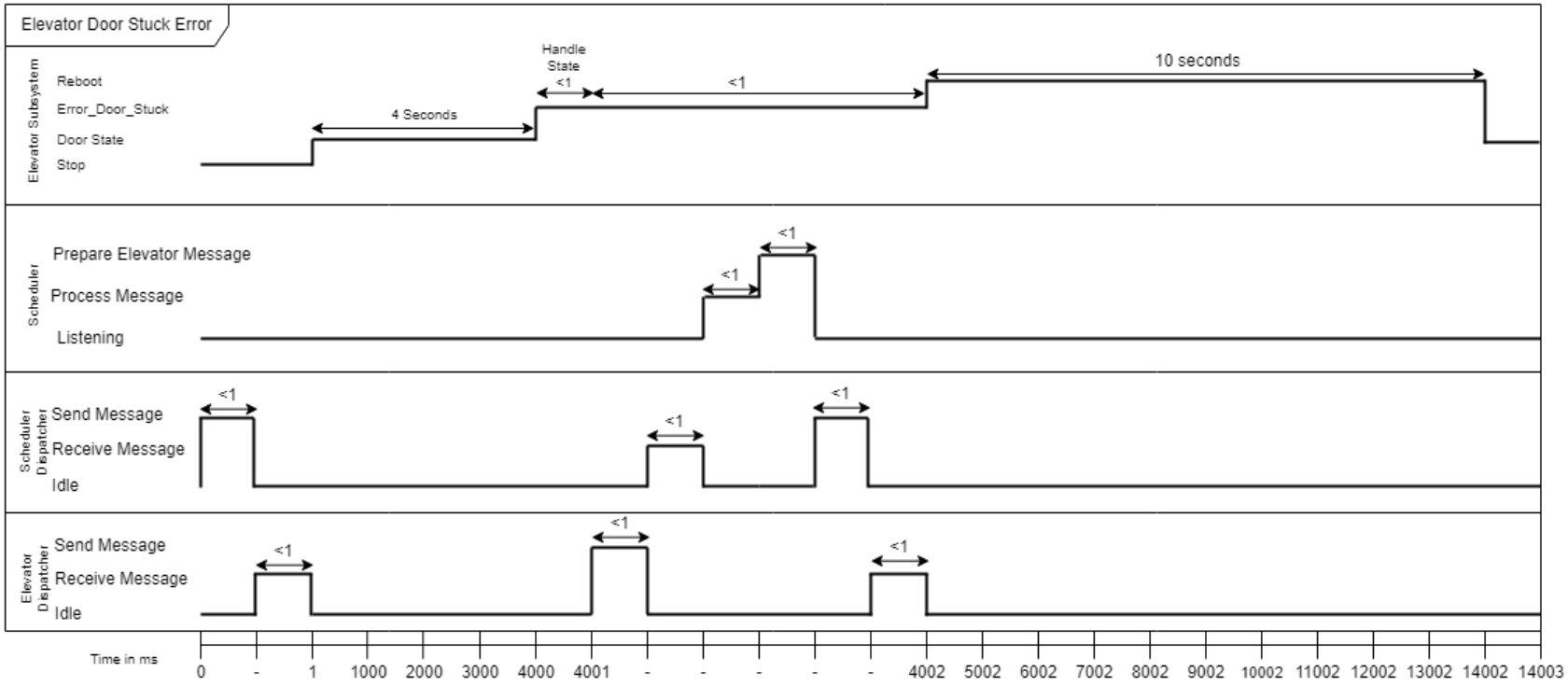
UML State Machine Diagram - Scheduler



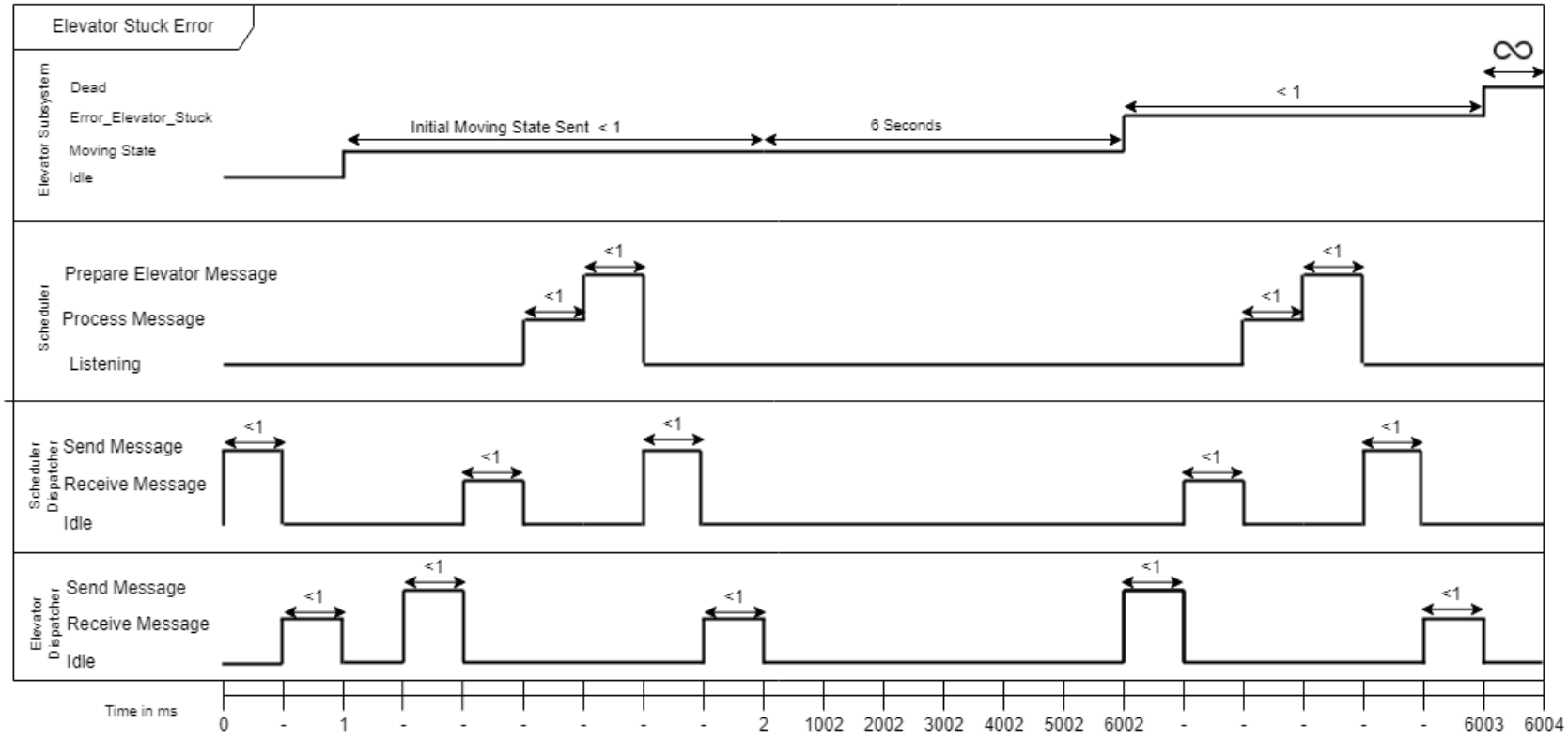
UML State Machine Diagram - Elevator



UML Timing Diagram - Elevator Door Stuck Error



UML Timing Diagram - Elevator Stuck Error



Instructions on How To Run the System

Setting up & Running the Program

NOTE: Our project compiles and runs with jdk-18.0.2.1.

NOTE: Once started, the elevator and scheduler have a 2 minute timeout to connect to the rest of the system. If the system (FloorSystem, Scheduler, and ElevatorSystem) are not running within the 2 minutes, then you will need to close everything and restart from step 4.

1. Unzip file to folder or you can ask us for access from GIT to clone the iteration.
2. Import the project into eclipse.
3. Ensure Maven dependencies have finished downloading and installing
4. In the “src/C2G8_Project/” folder, run “ElevatorSystem.java”. It will load and start trying to connect to the scheduler
5. In the “/” (root) folder, run the Maven “RunMonitorSystem.launch”.
    a. Once the window opens go to File (top left) and click “Connect” to connect to the Scheduler and receive updates. You can close the message box that appears saying that it is connecting.
6. In the “src/C2G8_Project/” folder, run “Scheduler.java”. It will establish connections with all components and then start the scenario.
7. In the “src/C2G8_Project/” folder, run “FloorSystem.java”. It will load and wait for a connection to the scheduler.

Testing the Program (with JUnit)

1. Make Setup instructions are complete
2. (Faster Version) Run ./test/src/C2G8_Project as a JUNIT test.
   OR…
   (Safer Version) Run all the tests files in ./test/src/C2G8_Project individually by right-clicking one and selecting “Run as…” -> “JUnit Test”, then repeating for the 8 test files in the folder.

Note: All tests should pass successfully. If a test fails or hangs, the issue is likely that a process (FloorSystem, ElevatorSystem, or Scheduler) was created and is trying to bind to a port that is already in use in another test case. You can run that test independently by right clicking on the test that failed (or the file that contains the test), selecting “Run as..”, and then selecting “JUnit Application”. Sometimes you may need to close eclipse and reopen it as the thread that is failing to bind properly may have gotten lost in

eclipse, making it unable to be closed unless you close eclipse entirely. Otherwise, the test should run and pass just fine.

(Optional) Changing the Scenario Speed

1. Locate the CONFIG.java file at SYSC3303_Project/src/C2G2_Project/Config.java.
2. Change the “SCENARIO_ACCELERATION_MULTIPLIER” variable to the desired integer value. The default (and lowest) value is 1.

(Optional) Changing the Elevator Speed Values

1. Locate the ElevatorTimes.java file at SYSC3303_Project/src/C2G2_Project/ElevatorTimes.java.
2. Change the value of following integer variables:
    a. “MOVING”, which is how long it takes the elevator to accelerate 1 floor.
    b. “MOVING_MAX”, which is the max speed of the elevator, reached after travelling 1 floor.
    c. “DOORS”, which is how long it takes the doors to open, or how long it takes the doors to close.
    d. “STOP”, which is how long it takes the elevator to decelerate and stop (all occurring within 1 floor).

(Optional) Randomly Generating a New Input file

1. Locate the CONFIG.java file at SYSC3303_Project/src/C2G2_Project/Config.java.
2. To change the number of floors in the system (which are each their own thread), change the “FLOORS” variable. The default is 22. The minimum number of floors is technically 2, however both types of elevator faults will only appear if there are at least 10 floors. The max number of floors is dependent on how many threads your computer can handle, and since floor threads are relatively passive, testing upwards of 100 floors still works just fine.
3. To change the number of elevators in the system (which are each their own thread), change the “ELEVATORS” variable. The default is 4. The minimum number of elevators is 1. The max is dependent on how many threads your computer can handle, however since elevator threads are relatively active, 100 threads seems to be a good max limit.
4. To change the number of passenger requests in the system (the number of times a person requests an elevator), change the “MAX_PEOPLE” variable. The default is 15. The minimum number of passengers is 0. The max is any integer such that MAX_PEOPLE / MAX_SCENARIO_DURATION is roughly greater than 60.
5. To change the period of time in which passengers can request an elevator, change the “MAX_SCENARIO_DURATION_MILLS” variable. Note that this variable is a duration in milliseconds. The default is 60 seconds (60*1000). The minimum scenario duration is MAX_PEOPLE * 60. The max is any long value.

6. Once all variables are changed, you must remove the link to the input file (called InputFile.txt by default, located in the root directory, SYSC_3303_Project/InputFile.txt) by doing any one of the following:
    a. Deleting the input file.
    b. Renaming the input file to something other than what is listed in the “REQUEST_FILE” variable in the CONFIG.java file.
    c. Changing the name of the file listed by the “REQUEST_FILE” variable in the CONFIG.java file to either: the input file that you wish to run, or a file that doesn’t yet exist in the root directory.

    As long as the “REQUEST_FILE” variable does not contain the name of a file in the root directory, the FloorSystem will generate a new input file when it is run next.

Measurement Results

Default Values

The default values used in measuring the results are the default values that were handed in with the final deliverable (visible in the CONFIG.java file). Those values are:

- Demo runs at normal (1x) speed
- 22 floors
- 4 elevators
- 15 “passengers” (or requests from floors)
- Requests come in over a 60 second time-frame
- 1 of the requests is an elevator door fault
- 1 of the requests is an elevator stuck fault (which means we will lose an elevator by the end of the demo)

Any changes or deviations from these default values will have been mentioned in the section it relates to.

Iteration	Raw Data (ms)	Raw Data (sec)	Time (Decimal)	Minutes	Seconds
1	321022	321.022	5.350367	5	21.022
2	233091	233.091	3.88485	3	53.091
3	294117	294.117	4.90195	4	54.117
4	233125	233.125	3.885417	3	53.125
5	300140	300.14	5.002333	5	0.14
6	269155	269.155	4.485917	4	29.155
7	345193	345.193	5.753217	5	45.193
8	255171	255.171	4.25285	4	15.171
9	273142	273.142	4.552367	4	33.142
10	230093	230.093	3.834883	3	50.093
11	251121	251.121	4.18535	4	11.121
12	274126	274.126	4.568767	4	34.126
13	270111	270.111	4.50185	4	30.111
14	286120	286.12	4.768667	4	46.12
15	214114	214.114	3.568567	3	34.114
16	275126	275.126	4.585433	4	35.126
17	232102	232.102	3.868367	3	52.102
18	226098	226.098	3.7683	3	46.098
19	293133	293.133	4.88555	4	53.133
20	277126	277.126	4.618767	4	37.126
21	201084	201.084	3.3514	3	21.084
22	279145	279.145	4.652417	4	39.145
23	347182	347.182	5.786367	5	47.182
24	230096	230.096	3.834933	3	50.096
25	270114	270.114	4.5019	4	30.114
26	265111	265.111	4.418517	4	25.111
27	249102	249.102	4.1517	4	9.102
28	283145	283.145	4.719083	4	43.145
29	244118	244.118	4.068633	4	4.118
30	224092	224.092	3.734867	3	44.092

Average Time	4.414786	= 4:25 (mm:ss)
Minimum Time	3.3514	= 3:21 (mm:ss)
Maximum Time	5.786367	= 5:47 (mm:ss)

Figure 1: A Table consisting of the data for how long it takes to complete a demo using default values, with 30 iterations.

Entire System

Using the default values, the program was run 30 times with different input files each iteration, and with a timer recording the results.

On average over all iterations, the program took 4 minutes and 25 seconds to successfully complete. The minimum amount of time that it took for an iteration to complete was 3 minutes and 21 seconds, while the maximum amount of time that it took for an iteration to complete was 5 minutes and 47 seconds (see figure 1). The margin of error is around ± 13 seconds, so the confidence interval at 95% is 4 minutes and 12 seconds to 4 minutes and 38 seconds.

Scheduler

The scheduler data was measured in two different situations: one with the default values, and the other with 100 passengers sending requests over a 400 second timeframe (same ratio as demo values of 15 passengers and 60 seconds) with all other default values being the same. The data for both will be provided in charts below (figure 2 & 3), but the measurements are similar enough that - for the sake of keeping everything consistent - the analysis will only be done on the demo that uses the default values.

The scheduler's data was analysed by its two different state paths that it can traverse (with the only differences being underlined):

1. Listening → Processing Message → <u>Prepare Floor Message</u> → Send Message → Update View → (back to Listening)
2. Listening → Processing Message → <u>Prepare Elevator State Message</u> → Send Message → Update View → (back to Listening)

When the scheduler received a message that required it to traverse the first state path (the floor message path), the average time it took to finish the path was 0.812 milliseconds, with the minimum time being 0.065 milliseconds, and the maximum time being 25.146 milliseconds.

When the scheduler received a message that required it to traverse the second state path (the elevator message path, which is traversed much more frequently than the floor path), the average time it took to finish the path was 2.139 milliseconds, with the minimum time being 0.163 milliseconds, and the maximum time being 31.554 milliseconds.

Note that on the graph, the highlighted yellow values are likely the most important, as those are the "worst case" values when receiving a message. Also note that the "Including total time listening" section is useful for finding out information such as, for example, how often a message is received by the scheduler (which is on average every 0.85-1.05 seconds, given the current values of the system).

15 Passengers requests over 60 seconds at 1x speed	Listening (total time)	Listening (one iteration)	Process Message	Prepare Floor Message	Prepare Elevator State Message	Send Message	Update View
# of Data Points	294	294	294	15	279	294	294
Average (second)	1.05127496	7.56E-06	1.37E-05	0.001633	0.000306	0.00035	0.000134
Average (milli)	1051.27496	0.007565	0.013687	1.633107	0.306139	0.349912	0.134432
Average (micro)	1051274.96	7.564626	13.68741	1633.107	306.1394	349.9116	134.4323
Average (nano)	1051274960	7564.626	13687.41	1633107	306139.4	349911.6	134432.3
Min	0.0035	0.0001	0.0012	0.104	0.0064	0.0015	0.0558
Max	7003.0696	1.4747	1.2071	17.8197	12.4118	8.4203	2.6317
Median	495.55285	0.0002	0.0045	0.1286	0.213	0.28405	0.1018
Mode	#N/A	0.0002	0.005	#N/A	0.247	0.2756	0.0915

Not including total time listening:		
Average time to go through elevator message state path (milliseconds):	0.812	ms
Minimum time to go through elevator message state path (milliseconds):	0.065	ms
Maximum time to go through elevator message state path (milliseconds):	26.146	ms
Average time to go through floor message state path (milliseconds):	2.139	ms
Minimum time to go through floor message state path (milliseconds):	0.163	ms
Maximum time to go through floor message state path (milliseconds):	31.554	ms

Including total time listening:		
Average time to go through elevator message state path (milliseconds):	1052.079	ms
Minimum time to go through elevator message state path (milliseconds):	0.068	ms
Maximum time to go through elevator message state path (milliseconds):	7027.741	ms
Average time to go through floor message state path (milliseconds):	1053.406	ms
Minimum time to go through floor message state path (milliseconds):	0.166	ms
Maximum time to go through floor message state path (milliseconds):	7033.148	ms

Figure 2: A Table consisting of the data for how long it takes to complete each state in the scheduler using default values. Default time measured is in milliseconds.

100 Passengers requests over 400 seconds at 1x speed	Listening (total time)	Listening (one iteration)	Process Message	Prepare Floor Message	Prepare Elevator State Message	Send Message	Update View
# of Data Points	1068	1068	1068	100	968	1068	1068
Average (second)	0.84489017	2.47E-06	4.9E-06	0.000381	0.000271	0.00022	8.59E-05
Average (milli)	844.890167	0.002474	0.004901	0.381161	0.270618	0.219663	0.085916
Average (micro)	844890.167	2.474345	4.901124	381.161	270.6176	219.6625	85.91564
Average (nano)	844890167	2474.345	4901.124	381161	270617.6	219662.5	85915.64
Min	0.0011	0.0001	0.0006	0.0903	0.0067	0.0002	0.0308
Max	5491.9788	1.3982	1.2188	16.6614	12.4477	7.8459	2.4435
Median	456.50765	0.0003	0.002	0.1465	0.23485	0.1647	0.0686
Mode	1.2523	0.0003	0.002	0.1157	0.1799	0.1444	0.0626

Not including total time listening:		
Average time to go through elevator message state path (milliseconds):	0.584	ms
Minimum time to go through elevator message state path (milliseconds):	0.038	ms
Maximum time to go through elevator message state path (milliseconds):	25.354	ms
Average time to go through floor message state path (milliseconds):	0.694	ms
Minimum time to go through floor message state path (milliseconds):	0.122	ms
Maximum time to go through floor message state path (milliseconds):	29.568	ms

Including total time listening:		
Average time to go through elevator message state path (milliseconds):	845.471	ms
Minimum time to go through elevator message state path (milliseconds):	0.039	ms
Maximum time to go through elevator message state path (milliseconds):	5515.935	ms
Average time to go through floor message state path (milliseconds):	845.582	ms
Minimum time to go through floor message state path (milliseconds):	0.123	ms
Maximum time to go through floor message state path (milliseconds):	5520.148	ms

Figure 3: A Table consisting of the data for how long it takes to complete each state in the scheduler using 100 passenger requests over 400 seconds. Default time measured is in milliseconds.

Elevator

The Elevator data was measured using the default values, with a timer recording the results. Three different aspects of the elevator were measured: how long it takes to receive (Extract) the message, how long it takes to process a message, and how long it takes the elevator to handle an entire state (from receiving a message to sending a message).

To extract a message, it takes, on average, 0.0078 milliseconds, with the minimum and maximum times being 0.0016 milliseconds and 0.0475 milliseconds, respectively. To process a message, it takes, on average, 0.2686 milliseconds, with the minimum and maximum times being 0.0008 milliseconds and 20.1632 milliseconds, respectively (see figure 4).

The total amount of time spent in a state is different for each state, depending on the amount of time it takes the elevator to complete the action related to the state. The “IDLE” state is the only state that is measured that does not have an expected time associated with it. As such, the average time spent in IDLE was 0.6503 milliseconds while the minimum and maximum times spent in that state are 0.1427 milliseconds and 2.9821 milliseconds, respectively. For the other states that have an expected time associated with them, in general, the minimum time spent in that state was 10 milliseconds less than the expected time, while the maximum time spent in that state was 4 milliseconds more than the expected time. Big variances in the average are due to varied expected times for the MOVING states, as well as errors occurring in that state that cause the elevator to “reboot”.

Default Values	IDLE			MOVING_UP			MOVING_DOWN			STOP			OPEN_DOORS			CLOSE_DOORS		
	EXTRACT	PROCESS	TOTAL	EXTRACT	PROCESS	TOTAL	EXTRACT	PROCESS	TOTAL	EXTRACT	PROCESS	TOTAL	EXTRACT	PROCESS	TOTAL	EXTRACT	PROCESS	TOTAL
# of Data Points	28	28	28	122	122	103	64	64	52	23	23	22	24	24	24	24	24	24
Average (second)	7.60357E-06	0.000973	0.00065	8.23E-06	0.00022	3.19227168	7.33E-06	7.19E-05	3.34496491	7.24E-06	0.000132	7.00148586	9.28E-06	0.000126	3.41869891	6.87E-06	8.9E-05	3.00188357
Average (milli)	0.007603571	0.972829	0.650304	0.008232	0.219682	3192.27168	0.007325	0.071903	3344.96491	0.007239	0.131809	7001.48586	0.009283	0.126463	3418.69891	0.006867	0.089038	3001.88357
Average (micro)	7.603571429	972.8286	650.3036	8.231967	219.682	3192271.68	7.325	71.90313	3344964.91	7.23913	131.8087	7001485.86	9.283333	126.4625	3418698.91	6.866667	89.0375	3001883.57
Average (nano)	7603.571429	972828.6	650303.6	8231.967	219682	3192271681	7325	71903.13	3344964912	7239.13	131808.7	7001485864	9283.333	126462.5	3418698913	6866.667	89037.5	3001883567
Min	0.0038	0.2905	0.1427	0.0016	0.0008	2989.5449	0.003	0.0009	2996.0213	0.0038	0.0012	7000.0683	0.0046	0.0352	3001.4748	0.0048	0.0409	3001.4694
Max	0.0198	4.868	2.9821	0.0455	20.1632	5000.4044	0.0222	1.137	5000.354	0.0154	1.5539	7002.8048	0.0475	0.645	13003.036	0.0119	0.215	3003.1669
Median	0.00675	0.52185	0.41385	0.0065	0.0018	2998.9827	0.0069	0.0016	2999.32325	0.0067	0.0598	7001.7564	0.007	0.08465	3001.82375	0.00635	0.06465	3001.779
Mode	0.0058	#N/A	#N/A	0.0063	0.0018	#N/A	0.0066	0.0014	#N/A	0.0049	#N/A	#N/A	0.007	#N/A	#N/A	0.005	#N/A	#N/A

Average time to extract a message:	0.0078 ms	Average time to process a message:	0.2686 ms
Minimum time to extract a message:	0.0016 ms	Minimum time to process a message:	0.0008 ms
Maximum time to extract a message:	0.0475 ms	Maximum time to process a message:	20.1632 ms

Figure 4: A Table consisting of the data for how long it takes to complete each state in the elevator using default values. Default time measured is in milliseconds.

Elevator Expected Times

Expected time values for our system was based off of the measured elevator values from iteration 0. Currently, the values in our system are as follows (and can be found in the ElevatorTimes.java file):

- Accelerating from a stationary position (MOVING) takes 5 seconds to get up to max speed. This is also how long it takes to move one floor while accelerating.

- Moving at max speed (MOVING_MAX) takes 3 seconds to move from one floor to the next
- Stopping (STOP) takes 7 seconds. This includes the time it takes for the elevator to slowly lock into position before opening the doors.
- Opening and closing doors (DOORS) take the same amount of time, 3 seconds each.

In iteration 0, we measured that with 95% confidence, opening a door takes between 3.3466 to 3.3768 seconds, with a margin or error of 0.0072 seconds. Closing a door takes between 3.3845 to 3.6955 seconds, with a margin or error of 0.0504 seconds. In an attempt to simplify the program, we simplified the values by rounding them down, and grouped them together due to them being so close in nature. For the sake of the demo, it is expected that passengers are able to load while the doors are opening and closing.

Schedulability

Floor

Schedulability for the floor is not much of an issue as each floor is run on its own thread, and asynchronous message handling is done by the dispatcher. The floor checks if a message has been sent to it, then checks if it needs to send a message, then repeats. Message sending and receiving is handled by the dispatcher, so there are no issues with schedulability.

Scheduler

The scheduler takes, in the worst case scenario, about 31.5 milliseconds to process an event, however it's expected time is less than 3 milliseconds per event. Going off of the worst case time, a scheduler is able to receive and process at most 31 messages per second without missing any deadlines. Any additional messages that are received while the scheduler is processing a message are not lost, but rather handled by the dispatcher and put into a shared queue that the scheduler can access when it is done. Since dispatchers run in their own thread, they can wait until it's their turn to save the message for the scheduler, making sure nothing is missed.

Elevator

Elevators are similar to floors in the sense that they too run in their own thread, with asynchronous message handling done by the dispatcher. However, where floors can technically send multiple messages per second (if someone is spamming the button), elevators cannot. Elevators can send at most 2 messages per second. This is the case where an elevator transitions into IDLE (message one), and then immediately into MOVING (which sends another message at the beginning of the MOVING state). Since elevators also use dispatchers, there isn't any risk of missing any messages.

Reflection on Overall Software Design

Jayson

I was responsible for the initial overall system design which was proposed and changed in our early meetings and group coordination. I was also the developer and designer for the FloorSystem and subsystems, Monitor System, Passenger Requests (Scenario) and the Dispatcher.

The overall design was to have a “System” executable that would be responsible for the setup of each individual component within its responsibility. In particular the floor and elevator subsystems would be contained within the overall system along with a dispatcher that would allow each program to communicate with each other. All systems would also draw from a single set of constants using the CONFIG class so that we can coordinate details about the scenario, its timing, ports, and other information of shared importance. Overall this worked well because it separated the responsibility for each system component from others making it easier to debug.

Dispatcher Component

The dispatcher component was the most central part of the system and standardised all UDP communications between components in an asynchronous single directional manner. The overall concept was that the dispatcher would implement a type of publish/subscribe message system that used topics with associated data payloads. System components would use the dispatcher to send messages and payloads with a particular topic to a destination. There may be more than one dispatcher that is associated with a destination allowing multiple monitors to receive the same message. When the dispatcher receives a message to send it would wrap the message into a new data structure containing the topic and payload and serialise the data using the jackson serialisation tool from apache into JSON text. Dispatchers receiving the message would look at a list of subscribers and call a function that must be implemented through an interface that the dispatch user had to implement. To accomplish this the dispatcher would run in it’s own thread as a runnable and constantly check for new messages and deal with them asynchronously. To ensure that the dispatcher can always continue to receive messages, work was offloaded to a worker thread for dispatching the message to subscribers. Overall this design was very successful and its standardised documentation and consistency was very helpful in debugging our growing system through the iterations. While initially difficult to test due to its asynchronous nature, it was extremely helpful for testing other systems since we could then simulate inputs to a system in unit tests by injecting the expected messages using a test dispatcher before validating the behaviour of the system under test.

Floor System Component

The floor system's design was to have the system create the dispatcher, one subsystem for each floor, and either load or generate a scenario. This worked well and had no issues. What worked well was how we delegated each request to the floor subsystem representing the floor of origin for the passenger request and had those dispatched to the scheduler at the correct scenario time. The floor system coordinated the scenario time using a stopwatch so that all subsystems stayed synchronous and actioned the requests in the proper order. However, the design is a bit wasteful given the overall design in hindsight. Since the dispatcher does work using worker threads it was not necessary to have each floor system with its own thread causing a lot of un-necessary overhead. If we used subsystems passively within the floor system instead of making them a thread this would have worked well since it didn't benefit from multi threading in any way due to its synchronous nature.

Passenger Request (Scenario Generation and Loading)

This was a passive class used by the floor system responsible for the generation and loading of scenarios and each passenger request. It was designed to generate a random scenario if one was absent from the filesystem and then enforce hard scenario standards imposed by our project criteria. This was very successful and scalable because we found that at times we had to make changes to our scenario and it helped us quickly generate new ones meeting the changed criteria without much extra work. It's fault tolerance was also quite useful early on because it enabled us to generate valid scenarios as a fallback if the file was missing or corrupt.

Jean-Pierre

Elevator System Component

I worked mainly on the Elevator side of this project and handled all sorts of different situations using a State-Driven system. We learned of the hierarchical way of doing State Machines which most likely could have been used for this implementation as well, however since this was implemented prior to this information and the elevator states remained fairly simple the switch was not done.

Each elevator is independent of each other such that one won't know about the other and no interactions can occur between each other. This was advantageous as adding more elevators was a matter of creating more ElevatorSubsystem objects. In addition when it comes to testing, if one Elevator passes all the tests, then all elevators automatically pass the tests no matter the amount created.

There were two types of tests made for the elevators, one which revolved around the Elevator System capabilities in handling the Elevator Subsystems, such as generating them and shutting them down. Then there were the tests for the Elevator Subsystems whose main goal was to test all the possible states an elevator can attain. While the Elevator System testing was not too big,

the Elevator Subsystem took a lot of space due to having a variety of possible outcomes/states, although it was necessary to make sure everything was working properly.

Aside from the Elevator Component I worked a lot on multiple diagrams, and mapping out our whole scenario. At the beginning these diagrams were quite helpful as when they were being built I could tell if something didn't quite fit. Naturally these design fixes led to our later successes. Overall I am very happy with our design choices and the communication method for our Major Components.

Jordan

Scheduler Component

I worked mainly on anything related to the scheduler, including the algorithm to determine what elevator is chosen for a passenger request.

What I like about the scheduler is that by using the state machine design pattern and separating the scheduler's functionality into states, it made it a lot easier to organise the code in a logical way (as well as debug it). However, I wish I had the knowledge that I do now about how to make more effective state machines, and about hierarchical state machines, as I would have likely implemented something a bit cleaner if I had.

What I liked about the algorithm that I created for the elevator assignment is that it can be boiled down to two cases that make the entire algorithm function. Originally it was meant to be a bit more complex, but we were told that it could be something simple, so it became simple. What I don't necessarily like about the usage of the algorithm is that once a request is assigned to an elevator, it will stay assigned to that elevator even if another elevator happens to become a slightly better option. The only reassigning is done with elevators that become "dead".

I also did all of the measurements for the system using a regular timer (stopwatch) in the program. Since this was the last thing done to the program, I was not too worried about the design of the measurement mechanisms other than trying to make sure that I'm not interfering too much with the processes. In retrospect, I could have made the measurement timer into its own class, which may have made it a bit easier to implement on other parts of the system. I also didn't fully understand what was being asked to be measured, since there were seemingly inconsistent requests for measurements throughout the various course files, so I may have measured the program too much on a micro scale as opposed to a macro scale (which isn't necessarily bad, but was a bit more time consuming).

Finally, since I did the debugging between systems and got to experience a lot of how the systems interacted with each other, I can say that I like the overall design of the core communication flow of the system. I think, as a group, we did a pretty good job in creating a message flow or sequence that makes sense for this project.