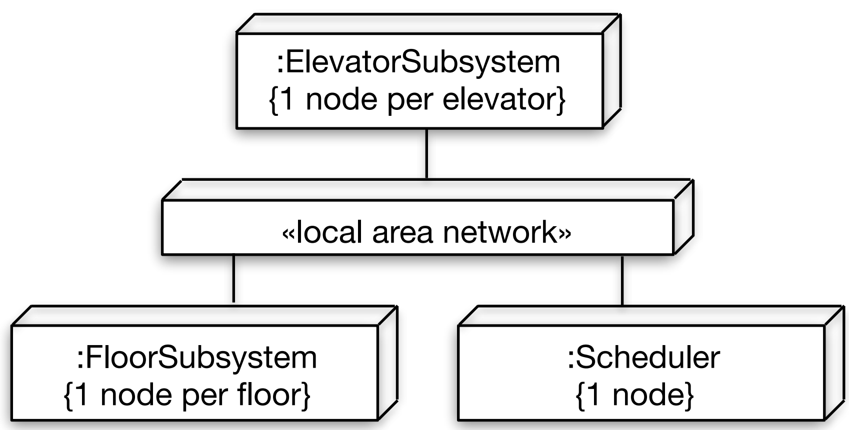
Carleton University
Department of Systems and Computer Engineering
SYSC 3303B RealTime Concurrent Systems Winter 2020

Project Specification

Teams will design and implement an elevator control system and simulator. The system will consist of an elevator controller (the Scheduler), a simulator for the elevator cars (which includes, the lights, buttons, doors and motors) and a simulator for the floors (which includes, buttons, lights and last, but not least, people who are too lazy to take the stairs). The elevator controller will have to be multi-threaded since it is expected to handle more than one car at a time. Your simulation should be configurable in terms of the number of floors, the number number of elevators, the time it takes to open and close the doors, and the time it takes to move between floors. The simulation will eventually run on multiple computers as part of the project will involve running the controller on its own machine with the simulator(s) running on a separate computer. The code will be written in Java, using the Eclipse IDE. *You must design your code to work in the lab environment provided!*



Note that there will be three separate programs (projects): the Elevator Subsystem (the cars), the Floor Subsystem (the floors in the building and the simulated users), and the Scheduler. Each program will run as a separate Win32 process, and the programs will ultimately communicate using DatagramSocket objects. These three components should run on three separate computers in the lab. You can decide whether each elevator and floor is run as a separate instance, or that the elevator and floor subsystems are one program that can simulate multiple cars and floors simultaneously. See the Reference Material on the course web site for hints in doing this in Eclipse.

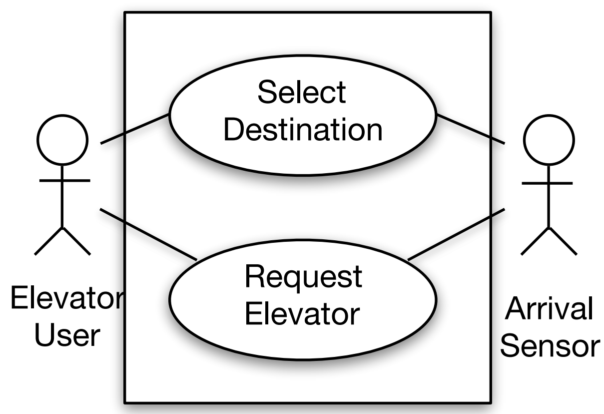
Your project is also expected to behave in “real-time” which means that elevators take time to more from floor to floor. For Project Iteration zero, your are to measure the time your favourite elevator (!) takes to move between floors. I suggest that you choose the elevators in the Dunton Tower, Minto Case or the Canal Building. I would avoid any pokey freight elevator (like the hydraulic elevator between blocks 2 and 3 in MacKenzie)!

Your team's code must demonstrate good programming style, and be well documented. For examples of "industrial quality" Java code, have a look at Sun's Java coding conventions, which can be found on the Oracle Web site. Unit testing using JUnit should be included for each subsystem. Java Doc documentation should also be provided.

All team members should be familiar with all aspects of the code and diagrams for your group. Working in (at least) pairs for the programming, debugging, and developing the diagrams is recommended. If any team members are not pulling their weight, please notify the TAs and/or instructor as early in the term as possible so that this can be remedied. Part of the final project mark will be based on Graduate Attribute 6 (see below and the course web site).

Please note that this project specification may be revised as necessary.

Use Cases



Select Destination Use Case

Actors: Elevator User (primary), Arrival Sensor
Precondition: User is in the elevator
Description:

1. User presses an elevator floor button. The elevator button sensor sends the elevator button request to the system, identifying the destination floor the user wishes to visit.
2. The new request is added to the list of floors to visit. If the elevator is stationary; the system determines in which direction the system should move in order to service the next request. The system commands the elevator door to close. When the door has closed, the system commands the motor to start moving the elevator, either up or down.
3. As the elevator moves between floors, the arrival sensor detects that the elevator is approaching a floor and notifies the system. The system checks whether the elevator should stop at this floor. If so, the system commands the motor to stop. When the elevator has stopped, the system commands the elevator door to open.

4. If there are other outstanding requests, the elevator visits these floors on the way to the floor requested by the user. Eventually, the elevator arrives at the destination floor selected by the user.

Alternatives:

- If the elevator is at a floor and there is no new floor to move to, the elevator stays at the current floor, with the door open.
  Postcondition: Elevator has arrived at the destination floor selected by the user.

Request Elevator Use Case

Actors: Elevator User (primary), Arrival Sensor
Precondition: User is at a floor and wants an elevator.
Description:

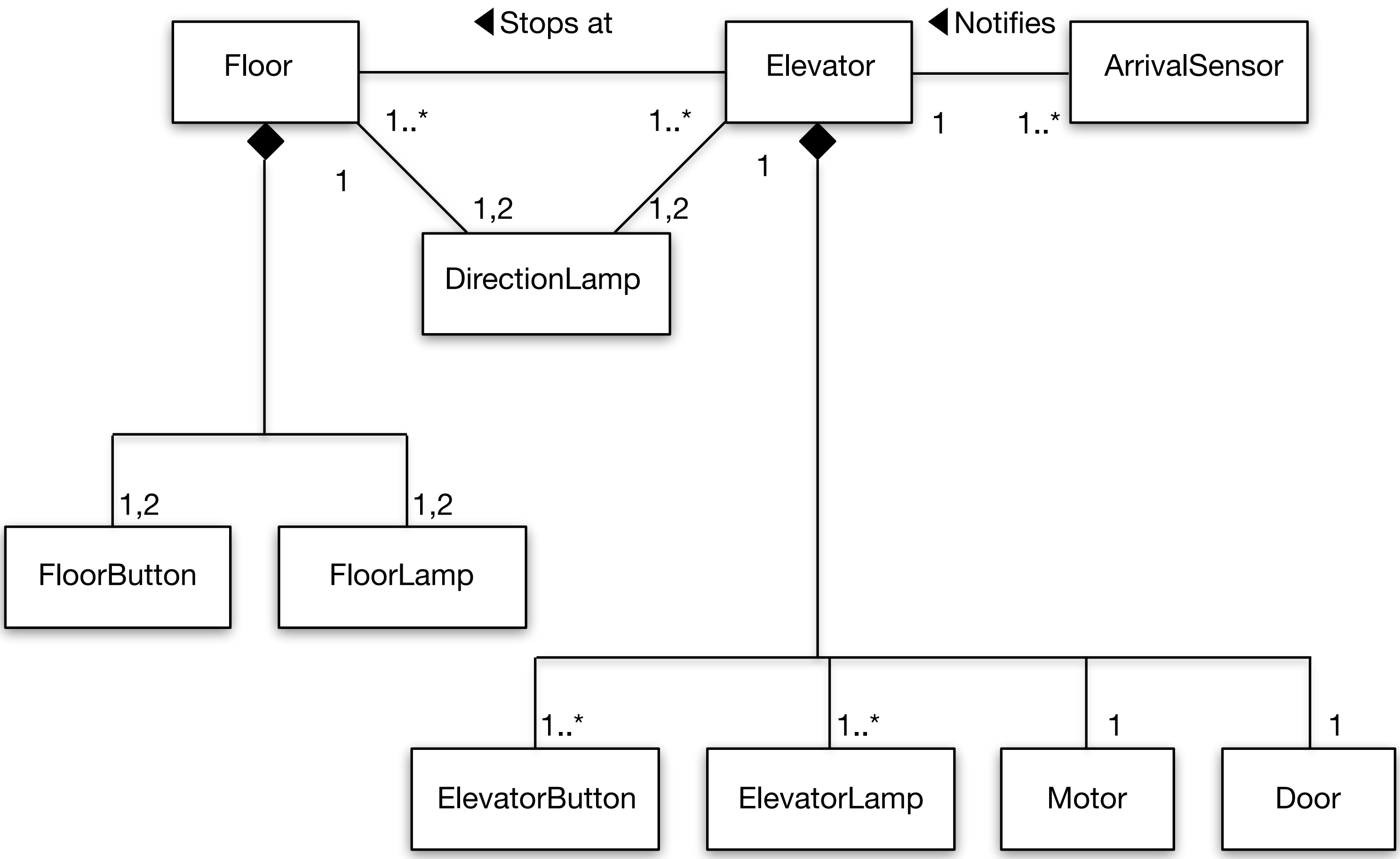
1. User presses the up floor button. The floor button sensor sends the user request to the system, identifying the floor number.
2. The system selects an elevator to visit this floor. The new request is added to the list of floors to visit. If the elevator is stationary, the system determines in which direction the system should move in order to service the next request.The system commands the elevator door to close. After the door has closed, the system commands the motor to start moving the elevator, either up or down.
3. As the elevator moves between floors, the arrival sensor detects that the elevator is approaching a floor and notifies the system. The system checks whether the elevator should stop at this floor, if so, the system commands the motor to stop. When the elevator has stopped, the system commands the elevator door to open.
4. If there are other outstanding requests, the elevator visits these floors on the way to the floor requested by the user. Eventually, the elevator arrives at the floor in response to the user request.

Alternatives:

- User presses the down floor button to move down. System response is the same as for the main sequence.
- If the elevator is at a floor and there is no new floor to move to, the elevator stays at the current floor, with the door open.

Postcondition: Elevator has arrived at the floor in response to user request.

Static Model of the Problem Domain



Each elevator has:

- **A set of elevator buttons:** The user presses one to select a floor.
- **A set of elevator lamps:** The lamps indicate the floor(s) which will be visited by the elevator.
- **An elevator motor:** The motor moves the elevator between floors.
- **An elevator door:** The elevator door also opens and closes the floor doors.

Each floor has:

- **Up and down floor buttons:** The user presses a button to request an elevator. Note that the top and bottom floors will only have one button.
- **Floor lamps**: The lamps indicated which buttons have been pushed.
- **Direction lamps:** Each elevator has a set of lamps to denote the arrival and direction of an elevator at a floor.
- **Arrival sensors**: Each elevator shaft at each floor has a sensor to detect the presence of an elevator.

Floor Subsystem Simulation

The floor subsystem is used to simulate the arrival of passengers to the elevators and for simulating all button presses and lamps. It should also test for the proper operation of the elevator, i.e., a passenger arrives, presses the elevator request button, waits for the arrival of an elevator (signalled by a lamp), then presses a floor button. The elevator chosen should eventually arrive at the floor selected.

Your floor subsystem should read an input file using the format shown below:

Time	Floor	Floor Button	Car Button
hh:mm:ss.mmm	n	Up/Down	n
14:05:15.0	2	Up	4

i.e., a time stamp, white space, an integer representing the floor on which the passenger is making a request, white space, a string consisting of either "up" or "down", more white space, then an integer representing floor button within the elevator which is providing service to the passenger. For example, in the table above, a passenger arrives at 2:05pm on the second floor of the building and wants to go up to the fourth floor. It is strongly advisable to maintain separate queues for each floor and direction for service requests since passengers may arrive at other floors prior to an elevator reaching the floor currently being serviced. If several requests are pending for an elevator, they can all be serviced at the same time once an elevator arrives. You can assume that each elevator has infinite capacity (if only that were true in Dunton!) so you do not have to keep track of the number of passengers in an elevator car.

You can assume that there is no "down" button at the lowest floor, nor is there an "up" button at the highest. Furthermore, you can assume that are as many floor buttons in an elevator car as there are floors serviced by the elevator, no more and no less. You should assume that roughly one half of all passengers arrive at the ground floor (although you could also assume that people can enter a building at more than one floor like here at Carleton if you so like). You should also assume that people choose floors somewhat randomly when they enter the elevator at a ground floor. Finally, not all passengers who enter an elevator at a non-ground floor will exit at a ground floor (i.e., some passengers will go from floor 6 to floor 2).

Scheduler Specification

The scheduler is responsible for accepting input from all of the sensors shown in the diagram above, and sending indications (to devices such as lamps) and commands (to devices such as the motor and door). It is responsible for routing each elevator to requested floors and coordinating elevators in such a way to minimize waiting times for people moving between floors (hint: observe where the elevators are in the Canal Building). You definitely don't want to serve requests in a first-come, first-served order, but you do want to make sure that all requests are served in a timely fashion (i.e., prevent starvation).

The scheduler system also has to be prepared to handle possible faults and failures in the system. Some of these faults include, but are not limited to, doors not opening or closing, elevator cars getting stuck between floors, and packets being lost on the LAN.

Elevator Subsystem Specification

The elevator subsystem consists of the buttons and lamps inside of the elevator used to select floors and indicate the floors selected, and to indicate the location of the elevator itself. The elevator subsystem is also used to operate the motor and to open and close the doors. Each elevator has its own elevator subsystem.

For the purpose of this project, the elevator subsystem responds to the scheduler to control the motor and to open the doors. The elevator subsystem also has to monitor the floor subsystem for destination requests (button presses inside of the elevator car, rather than button presses at the floors) from the input file. Button presses are to be rerouted to the scheduler system. Lamp (floor indications) from button pushes do not have to originate from the scheduler. Rather, when the elevator subsystem detects a button request, it can then light the corresponding lamp. When the elevator reaches a floor, the scheduling subsystem signals the elevator subsystem to turn the lamp off.

User Interface

Not much of a user interface is needed initially. For the final project iteration, you will have to have some sort of output to show the position of all of the elevators in the system at any given time. Full marks will be awarded for graphical output.

The only other user interface is to load the input file into the system. You should also be able to provision the number of elevators and the number of floors before your simulation starts. Start up the three parts, then load the trace into the system. Must be able to configure the number of floors and the number of elevators. Stress test to see when it breaks (I.e, how many elevators can run at the same time? Maybe you'll be lucky and you won't hit a limit).

Development Process

The project will be developed using an iterative, incremental process. The result of each iteration will be the release of an executable piece of software that constitutes a subset of the final elevator controller and simulator. The software will be grown incrementally from iteration to iteration to become the final system. Note that when submitting iteration "*n*", it is absolutely not acceptable to include code for iteration "*n* + 1", etc. During the project iteration meetings held during the lab periods, you will be expected to show your iteration "*n* - 1" code in operation.

Iteration 0 — Measure a Real Elevator

In order to populate some of these values, your team should pick your favourite (set of) elevator(s) and measure the times. However, once you pick an elevator or a set of elevators(s), stick with it. You will need to measure the time it takes to load and unload the elevator (i.e., the time between the doors opening and closing) and time time it takes to move between floors. Note that real elevators accelerate and decelerate as they start and stop, so the time it takes to move between two floors depends on whether the car needs to stop or start. I suggest that you measure the time it takes to move between two adjacent floors and time it takes to move between multiple floors in order to find the maximum speed the elevator moves at and the rate of acceleration and deceleration. You should assume that there is a difference in height of four (4) metres between each floor (i.e., 22 steps times 7 inches per step).

To save your time, and to annoy your fellow students, press all the destination buttons in a car to measure the time between adjacent floors and them time to load and unload the elevator. Just be sure to have a stop watch with you when you do this! Conversely, pick a time during the day when the elevators aren't busy to time runs between multiple floors (don't forget to keep track of the number of floors the car passes!). Collect data for several trips then decide whether the mean, median, or perhaps even the mode is best for your project (but be prepared to justify your choice). This is a golden opportunity to put some of that physics and statistics knowledge you thought that you would never use again to good use. If you are truly lazy, you can use a smart watch to do all the recording for you. For the image below, splits 1 and 3 correspond to me entering and leaving an elevator, and split 2 is the time taken to move between floors. Split 4 is junk data because I pressed the stop button after I had left the elevator.

Stats Splits Time in Zones

Splits ▲	Time	Cumulative Time	Moving Time	Distance	Avg Speed	Avg HR	Max HR
1	0:09.2	0:09.2	0	0	0	74	74
2	0:14.3	0:23.5	0	0	0	73	74
3	0:09.3	0:32.8	--	0	0	78	78
4	0:07.1	0:39.8	--	0	0	78	78
Summary	**0:39.8**	**0:39.8**	**0**	**0**	**0**	**75**	**78**

Forerunner 235
Software: 7.90.0.0
Elev Corrections ?: Disabled ▾
Summary Data: Original

Work Products for Iteration #0:

- Your raw data for the time it takes an elevator to move between floors and to load/unload a car. Save this as one or more tables in an Excel spreadsheet (you can use open office/numbers, but export as excel). You should be able to show whether you have collected enough samples to have confidence in your estimates for speed, acceleration and loading times.
- Your estimated values for the rate of acceleration and deceleration of an elevator car, and the maximum speed of the car. To estimate the distance between floors, count the number of steps. A typical riser is seven inches in height. The total time to move between floors one floor at a time, not including loading times, should be higher than the time it takes to go between floors without stops. You can assume that the elevator decelerates at the same rate.
- Your estimated time it takes to load and unload a car. You should probably take note whether there is a significant difference in the loading time of a car if there are a lot of people getting in and out or not (Hint: the Dunton elevators may be your best bet here!). This may not be an average, but a function of the number of people moving in and out of the elevator. You can include this information in your excel spreadsheet, or submit it as a separate PDF file.
- Finally, answer the three questions in the quiz for Iteration 0. You will need to submit the maximum speed for the elevator, the rate of acceleration for the elevator, and the average loading/unloading time. Only one person per group needs to do this (assuming I have figured out how to correctly configure this in CU Learn).

Iteration 1 — Establish Connections between the three subsystems.

Using Assignment 1, create three threads, one for each subsystem identified above. The Floor subsystem and the Elevators are the clients in the system; the Scheduler is the server. The Floor subsystem is to read in events using the format shown above: Time, floor or elevator number, and button. Each line of input is to be sent to the Scheduler. The elevators will make calls to the Scheduler which will then reply when there is work to be done. The Elevator will then send the data back to the Scheduler who will then send it back to the Floor. For this iteration the Scheduler is only being used as a communication channel from the Floor thread to the Elevator thread and back again. It is only necessary to create a test case showing that your program can read the input file and pass the data back and forth. Note that in a future iteration, there will be multiple elevator cars all running independently of each other, and there will be three programs running on separate computers, so be sure that you allow for this configuration when you design your code, or you will have to refactor your design at a later point in time.

You probably don't want to pass text strings from the Floor subsystem to the rest of the system. You should develop a data structure which only passes then necessary information representing the hardware device (floor button, elevator button, etc.) to the scheduler. If you choose to work ahead, it would be a good idea to tag this "version" in GIT for the purposes of the demo to the TA's and for submission in CU Learn.

Work Products for Iteration #1:

- "README.txt" file explaining the names of your files, set up instructions, etc.
- Breakdown of responsibilities of each team member for this iteration.
- UML class diagram and sequence diagrams.
- Detailed set up and test instructions, including test files used
- Code (.java files, all required Eclipse files, etc.)

Iteration 2 – Adding the Scheduler and Elevator Subsystems.

The goal of this iteration is to add the state machines for the scheduler and elevator subsystems assuming that there is only one elevator. However, you should bear in mind that for the next iteration, your system is expected to coordinate between the elevators in order to maximize the number of passengers carried over time (i.e., the throughput). Note that the elevator subsystem is used to notify the scheduler that an elevator has reached a floor, so that once an elevator has been told to move, the elevator subsystem also has to be informed so that it can send out messages back to the scheduler to denote the arrival by an elevator. You can either maintain a single event list or have separate tasks for each elevator. Perhaps you can think of another way of doing it too.

Work Products for Iteration #2:

- "README.txt" file explaining the names of your files, set up instructions, etc.
- Breakdown of responsibilities of each team member for this iteration
- UML class diagram and sequence diagrams.
- State machine diagram for the scheduler and elevator subsystems.
- Detailed set up and test instructions, including test files used
- Code (.java files, all required Eclipse files, etc.)

Iteration 3 – Multiple Cars and System Distribution.

For this iteration, you are to use what you have learned from Assignments 2 and 3 and split your system up into three (or more) separate programs that can run on three separate computers and communicate with each other using UDP. Your task in Assignment 3 was to implement a simple version of a remote procedure call, so you are encouraged to recycle that code here. The Scheduler will now be used to coordinate the movement of cars such that each car carries roughly the same number of passengers as all of the others and so that the waiting time for passengers at floors is minimized. Hint: if a passenger wants to go down and there is an elevator above the passenger, then that elevator should service the passenger rather than having a second elevator go up instead. The state machines for each car should execute independently of each other, but they will all have to share their position with the scheduler. The scheduler will choose which elevator will be used to service a given request.

Work Products for Iteration #3:

- "README.txt" file explaining the names of your files, set up instructions, etc.
- Breakdown of responsibilities of each team member for this and previous iterations.
- Any diagram changed from the previous iteration's UML class diagram or sequence diagrams.
- Detailed set up and test instructions, including test files used.
- Reflection on how concurrency control at the scheduler changed from Iteration 2 to Iteration 3.
- Code (.java files, all required Eclipse files, etc.)

Iteration 4 – Adding Error detection and correction.

For this iteration, you will be adding code for detecting and handling faults. To this end, you will have to add timing events so that if the timer goes off before an elevator reaches a floor, then your system should assume a fault (either, the elevator is stuck between floors, or the arrival sensor at a floor has failed). Similarly, you should detect whether a door opens or not, or is stuck open. In Iteration 5 below, your elevator status output should show these faults. A door which has not closed should be regarded as a transient fault, so your system should be able to handle this situation gracefully (unlike the LRT). However, the floor timer fault should be regarded as a hard fault and should shut down the corresponding elevator.

You must submit code to enable us to see that your elevator scheduler can deal properly with the faults shown above (i.e., you must be able to inject these faults into the system). I suggest that you inject these faults using the input file (so you will have to modify its format and be able to show to us how it works).

Work Products for Iteration #4:

- "README.txt" file explaining the names of your files, set up instructions, etc.
- Breakdown of responsibilities of each team member for this and previous iterations
- Any diagram changed from the previous iteration's UML class diagram or sequence diagrams.
- Timing diagrams showing the error scenarios for this iteration
- Detailed set up and test instructions, including test files used
- Code (.java files, all required Eclipse files, etc.)

Iteration 5 – Measuring the Scheduler and predicting the performance.

The last stage of the project is to add a display console showing where each of the elevators is in real time and displaying any faults (if any). The idea is to have output suitable for the concierge sitting at the desk in the front lobby to refer to. This part of the project can be part of the floor subsystem if you so choose, or you may want to develop a separate subsystem altogether.

Your last task is to measure the performance of your simulation. To do so, measure how long your system takes to run the entire input file in terms of CPU or elapsed time, not simulated time. Try to determine the average time it takes your system to “move” one event on average. Include the variance (standard deviation).

Work Products for Iteration #5:

- “README.txt” file explaining the names of your files, set up instructions, etc.
- Breakdown of responsibilities of each team member for this and previous iterations
- Any diagram changed from the previous iteration’s UML class diagram or sequence diagrams.
- Detailed set up and test instructions, including test files used.
- Your estimate of the cost of handling an event.
- Code (.java files, all required Eclipse files, etc.)

Final Project Presentation

Your team will give a 15 minute presentation to a TA of iteration #5. It must show four elevators running with 22 floors with arrivals at levels one and two (i.e., the Dunton Tower). These arrivals must also eventually depart from either level one or two. To simulate people who arrive at level 2, then go down only to go up again, assume they depart and arrive at level 1 at the same time. Your demonstration should also show stuck doors on one elevator and a stuck elevator on a second. It is not necessary to demonstrate annoyed passengers typically found in Dunton. The goal of the presentation is to show us how awesome your team and project are, and to submit your final copy of your work using CULearn.

Finally, we will have a little contest to see whose elevator system can move all of the passengers from an input file supplied to you in the least amount of simulated time. Note that this time value is the simulated time that all passengers take, not the actual runtime of the simulation as was measured earlier. We shall assume that all the elevators are working for the entire period. Your system should allow the time it takes to move between floors and the time it takes to load/unload an elevator to be specified so that everyone is on an equal footing. The winner gets bragging rights and perhaps a good reference if you should choose to work for Otis.

Final Project Presentation Work Products:

You must submit all of the following:

1. A report consisting of:
    - Team number and team members
    - Table of contents
    - Breakdown of responsibilities of each team member for each iteration
    - All diagrams:
        - UML class diagrams for the three components.
        - A State Machine diagram for the scheduler.
        - Sequence diagrams showing all the error scenarios.
    - Detailed set up and test instructions.
    - Results from your measurements for both the real elevator, used for determining the simulated time, and from running your simulation, for determining the cost per event.

- Reflection on your design - what parts do you like and what parts should be redone.

2. Test files for all iterations.
3. Code (.java files, all required Eclipse files, etc.)

Submission and Grading:

Please refer to CULearn for all submission dates.

Work products for each iteration are to be submitted using cuLearn. Submit using your project team's number, (one submission per team) by the date/time specified. You can include any number of files in your submission. Ensure that all your files are accepted. Please submit PDF instead of Word, LibreOffice, Pages, etc. documents.

The project is worth 25% of your final grade. Iterations #1 through #4, and the demo are worth two marks each (total of 10 marks). 10 project marks are for the final deliverables. These 20 marks are for the team. However, based on the participation of each member, each individual's mark could be higher, the same, or lower than the team mark. The final five marks are assigned to each individual for their participation in the biweekly meetings (see below).

BiWeekly Meetings:

Each team will have an approximately biweekly 20 minute meeting with a TA to discuss their progress. The meetings are mandatory, and will be held during the labs. The TA's will expect a demo of each iteration during the meeting. The meeting schedule will be posted on the web site.

The teams will be graded using the following rubric:

Graduate Attrirbute		Performance Level	Level 1	Level 2	Level 3	Level 4
		Level Descriptor	**Beginning**	**Developing**	**Accomplished**	**Exemplary**
6	Individual and team work	6.1 Personal and group time management	*Deadlines often missed. Several projects or assignments appear hurried or inadequately addressed. No evidence of time management planning.*	*Some deadlines missed. Some tasks or assignments appear hurried or inadequately addressed. Some evidence of time management planning.*	*No important deadlines missed. Few tasks or assignments appear hurried or inadequately addressed. Good time management planning.*	*No deadlines missed. Tasks and assignments always excellently prepared and presented. Tasks occasionally completed ahead of schedule. Leadership in time planning.*
		6.2 Group culture, group dynamics	*Not dependable. Little or no trust from teammates. Dishonest or evasive interactions with teammates. Poor conflict resolution.*	*Usually dependable and trusted by teammates. Mostly ethical behavior towards project and teammates. Minor conflicts.*	*Dependable and trusted by teammates. Ethical open interactions with teammates. Good conflict resolution.*	*Dependable and trusted by team mates. Ethical and open interactions with teammates. Leads team in ethical and responsive behavior. Leadership in conflict resolution.*
		6.3 Leadership: initiative and mentoring, areas of expertise, and interdisciplinary teams	*No initiative. No effort to mentor group members or take ownership of an area of expertise.*	*Some initiative. Some effort to mentor group members and take ownership of an area of expertise.*	*Good initiative and self-motivation. Regular effort to mentor group members. Clear command of an area of expertise.*	*Self-motivated, with regular demonstration of initiative. Constructively directs other team members and helps them to improve at their tasks. High degree of knowledge in an area of expertise.*

Change Log

1.0: Initial version.

1.1: Revise Submission and Grading. Add rubric for Biweekly Meetings.

1.2: Add deadlines for the inputs.

1.3: Arrival sensors moved to elevator form floor subsystem in Iteration 1.

2.0: Revise project steps.

2.1: Revised sequencing to match assignments. Removed RMS.