SiMBTA

A simulation of the subway lines of the MBTA.

By: Hannah Clark, Raewyn Duvall, Andrew Stephens

Project Description

The project is to run a full simulation of the subway lines of the MBTA to observe patterns during busy hours and during service disruptions.

The MBTA often experiences events that affect people’s travel times. As many people rely on the service to get them where they need to go in a timely manner, it is the responsibility of the system’s managers to ensure that service disruptions do not inconvenience their riders an excessive amount. At the same time, they also have to work with a limited budget, and so have to figure out a way to deal with these disruptions in a cost-effective manner, whether it be running more trains or changing schedules to allocate trains to busier times. The project will be able to handle several different types of service disruptions, including construction projects, broken down trains, commuter congestion, and Red Sox games (or similar bursts in traffic).

We will use the MBTA’s data for average run times, which we will use to calibrate our simulation. There will be one clock governing the travel times in order to produce reasonably accurate time estimates. Passengers will be created with a randomly selected start and end point at a specific time, and will record the duration of their journey for use in calculating averages.

Concurrency will be used in this project by making each traveler and train a separate process. The purpose of this is to make the simulation as true to life as possible, where various travelers in the system also may move concurrently, except when being physically blocked, which our simulation will take into account. Each train will also be its own thread, and will allow certain passenger threads to be linked to it via message passing.

Minimum and Maximum Deliverable

The absolute minimum planned deliverable is a simulation of the Red Line, the results of which will be output to a text file. The maximum planned deliverable is a visualized simulation of all lines for which the MBTA provides accurate information (not the Green Line).

Because there is a large range of deliverables between those two points, we have decided on a progression that we feel will give us the best chance at producing useful results. After implementing the Red Line simulation, we plan to implement a the most basic visualization possible that still presents the information in a useful manner. We will then add a simulation of the Orange Line, including transfers between the lines. We will then add the remaining lines for which we have data, then improve the visualization.

Design Overview

We will be programming the simulation in Erlang. Initially, we were tied between Erlang and Python. Some parts of our design, such as the clock and how to keep the order of trains on a track lended themselves to concurrent programming design patterns for multi-threaded programs that we had previously encountered. For the clock, because our timing is event driven rather than system clock based, a condition variable would have been very useful for sending out ticks. For maintaining the train order, we could have had a module with many FIFO queues. However, Python would greatly limit the scale of the simulation because of the limits on the number of threads. Erlang was then considered because of the large number of processes. Also, passing messages is closer to how the components of our simulation behave in the real world. After laying out possible ways to create each component in Erlang, we determined that it would be feasible to do, and so we decided on Erlang so that we could have the most true to life an large scaled simulation.

We will have modules for trains, passengers, stations, the clock, the cartograph, and the main module for program startup. The cartograph is a set of functions that deliver information about stations, including which are reachable from a given direction and travel time between them. A train keeps track of the passengers currently on it; when it will arrive at the next station, if not delayed by a previous train or currently in a station; when it will leave the current station, when in a station; its capacity; and its direction. It also keeps track of the number of passengers that still wish to disembark and to board and the number that may still do so in the current minute, the values of which are only relevant while the train is in the station. A passenger keeps track of its current location, its endpoint, and its desired train to board. A station keeps track of the current trains, if any, on its inbound and outbound platforms; the trains approaching or waiting for a platform for both inbound and outbound directions; and the passengers in it. An “instance” of each of these modules is a process begun by the module’s start function that runs the module’s loop function.

The clock module was also the focus of a major design decision. The initial thought was that the clock simply send a tick to all trains and passengers that have yet to begin their journey after a predetermined amount of time chosen to represent a minute within the simulation. That would have been very easy to implement. We realized however that there was another possible method, and event driven clock. Because of the possible scale of the simulation, a great deal of things need to be able to happen in a minute to be accurate. While events can happen very quickly, there still could be the possibility of having to make the simulation minute longer and slow the simulation down. Also, we realized that if no passengers start a journey and there is no train in the station in a given minute, then waiting seems pointless because nothing needs to happen. Therefore, we thought of using an event driven clock. A simulation minute would then last exactly as long as needed, which is long enough for passengers to enter the station and for a reasonable number of passengers to board and disembark trains that are in the station, if there are passengers needing to do so. This would be able to handle cases where a large amount of things need to happen in a simulation minute without slowing down simulation minutes where few things need to happen. This method, while a little less true to life at first glance, should be just as true to life because of the limits on boarding and disembarking, which will be based on what is feasible in the real world, if not more so because of the prevention of unrealistic limiting due to a simulation minute not being long enough to execute all needed behaviors. While the event based clock requires more complexity, it was the method chosen because of the possible speed advantages. The clock will keep track of all trains, the number of trains done with boarding and disembarkation for the current minute, the passengers who haven't entered stations yet, and the current time. At the beginning of each clock tick, the stations and trains will send their state to the output module. Passengers will send the output module their start and end stations and the time taken for the journey upon its completion.

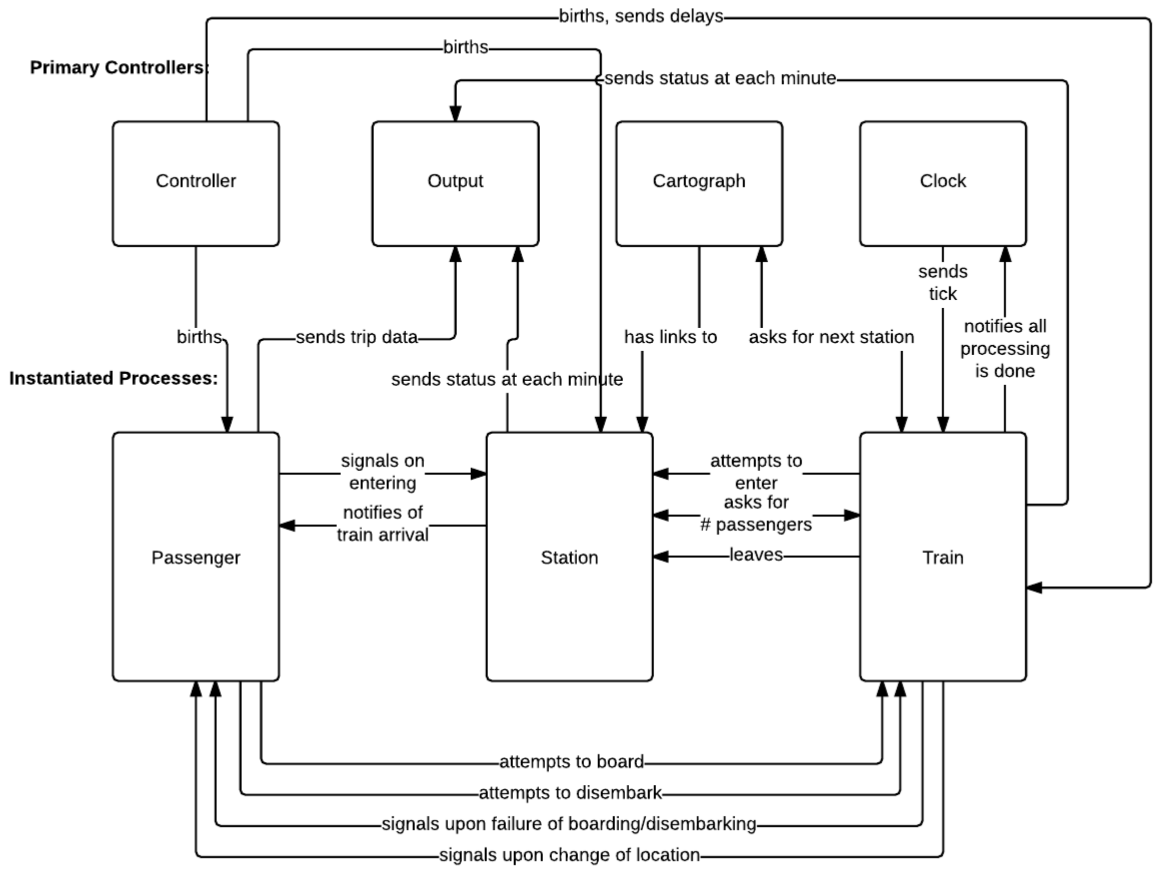
The main module will be used to start the simulation. It will read in the simulation specifications from a specified text file, start up the clock, and create instances of the other modules as needed for the simulation. It also communicates information about these modules to the clock and output modules. Main also contains a loop that causes train delays as indicated by the input file. Delays are specified in the input file to happen at given times for the corresponding train.

The clock also serves as a rendezvous point for the entire program. The next minute is not triggered until all operations in the previous minute are complete. Before this happens, we take this opportunity to record all data about the current state of the program using an output module. This module exports information about all of the trains, stations, and passengers to a text file, which aggregates this data over the course of the simulation. At the end of the simulation this data is loaded into a web page for a simple visualization. We choose to output data at the end of every minute, rather than as it happens, in order to make sure that messages do not get output in an incorrect order (for example, data from train 5 at minute 3 being output after data from train 2 at minute 4). The output module will receive messages from trains and stations each minute and from passengers at the end of their journeys and handle all output to prevent the slowing of the simulation. It's sole purpose is to provide consistent and organized output. It only keeps track of the number of stations and trains in the simulation and the backlog of its output.

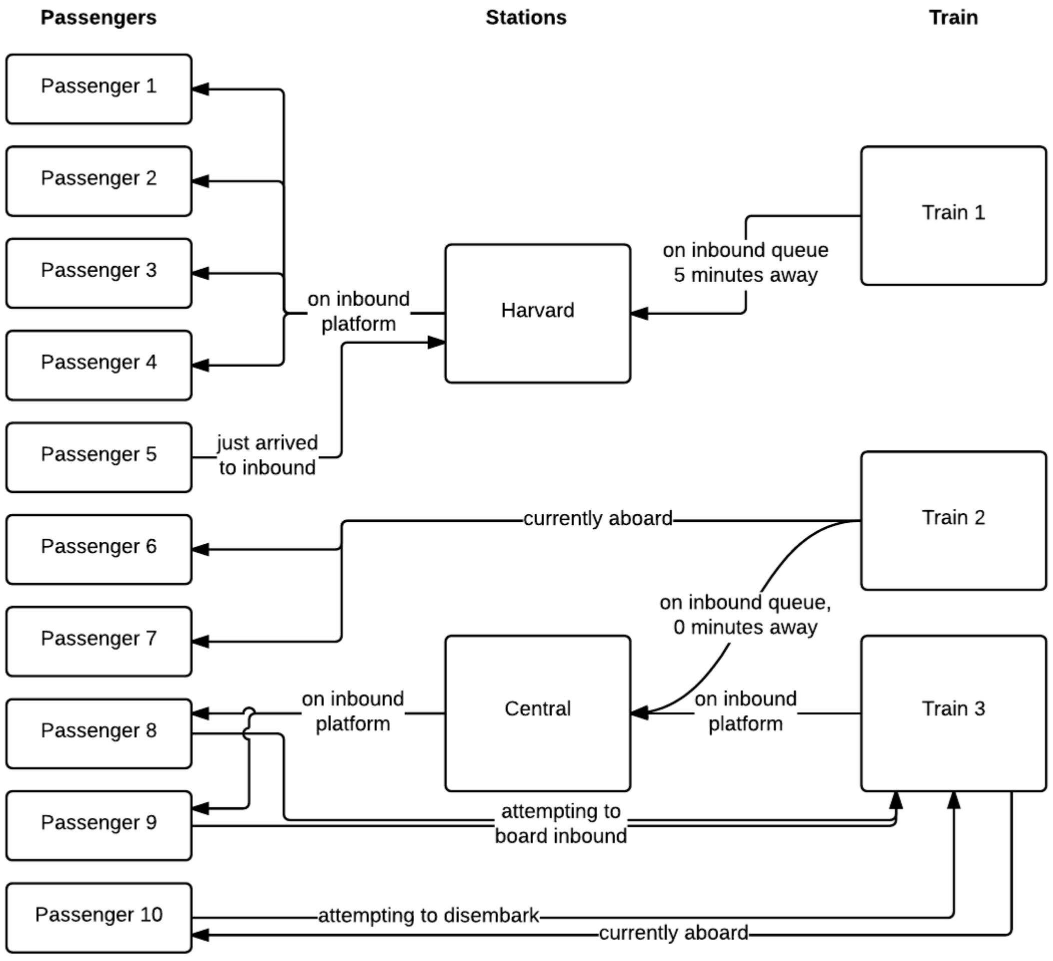
The visualization will be built in HTML using jQuery. The file will be parsed when the web page loads, and the page will contain a view of the red line and a slider for the user to change the current minute being viewed. Each train will be displayed as a graphic with a number, indicating the number of passengers on board, located either at a station or between two stations. Each station will also be displayed with the number of passengers currently waiting. Each passenger that finishes his/her journey at a particular minute will be displayed as such, along with the time it took him or her to travel from their origin station.

This design covers all of the components of the original minimum deliverable, with a simple visualization for the data following the simulation. The simulation only uses the Ashmont branch of the red line, so the path is linear.

Module Diagram:



Object Diagram:



At this particular moment in time, we examine the relationship between 10 passengers, 2 stations, and 3 trains. Of course, in the actual simulation there would be many more of all of these, but for the sake of simplifying the diagram only a few are shown. Passengers 1-4 were all birthed into existence by the master controller at various points in the last few minutes. Each one has a definite destination in mind, and by querying the cartograph they were each able to determine that inbound was the correct direction to go to get to their destination. The station is aware of their existence, and has them in a list that it plans to signal once an inbound train arrives. Passenger 5 has just arrived to the station, and so it notifies the station accordingly. It will then be added to the list of people to be notified when an inbound train arrives. Meanwhile, Train 1 has left Porter and is in the queue for Harvard, but still needs to wait for 5 ticks before it can begin boarding (as it will not have arrived at the station before then).

While these passengers are busy arriving and waiting, Passenger 6 and 7 are traveled to Park St for a romantic dinner in the North End. They are currently passengers on Train 2, which means that no station has reference to their PIds, but Train 2 does. Train 2 is currently on the track between Harvard and Central, but cannot enter the station because Train 3 is currently on the platform, and therefore Train 2 is waiting on a queue. As soon as Train 3 leaves, Train 2 will enter and notify its passengers that it has arrived in Harvard. As Passenger 6 and 7 are going to Park St, they will disregard this message.

At the same time, Train 3 is on the platform and boarding passengers. Central station has already sent a message that an inbound train is available to all the passengers on the inbound platform, allowing the passengers to request the ability to board the train. Depending on how crowded the train is and how many people are trying to disembark, the train may deny this request. For example, Passenger 10 is currently on Train 3 but wants to get off so he can get to his stand-up comedy show on time. As long as he is not blocked by the other passengers attempting to disembark, he will be successful. Otherwise he will have to wait a minute before trying again. Passengers 8 and 9 will only be able to board once all of the disembarking passengers have successfully done so.

There were no delays added to this simulation, so the controller has no delays to send to the train and is no longer in existence.

File Formats

Input

```
Minute num
train Direction:dir Station:station Passengers:num
train Direction:dir Approaching:station Passengers:num
station Name:station Passengers:num AshTrain:bool AleTrain:bool
passenger Start:station End:station Began:timeNum Duration:num
```

Output

```
train directionAtom startTimeInt capacityInt numDelaysInt
delay timeInt delayLengthInt
passenger countInt startTimeInt startStationAtom endStationAtom

Note: Delays apply to the train immediately preceding.
A numDelaysInt of zero for a train indicates there are no delays following.
The number of delays after a train must match numDelaysInt exactly.
```

Development Plan

So far, a detailed layout of the design for the simulation has been written up. We have divided the work on the modules as follows:

Cartograph	All together
Main	Hannah
Clock	Hannah
Station	Raewyn
Passengers	Raewyn

Train	Andrew
Output	Hannah
Visualization	Andrew

We have thoroughly thought out how the modules communicate with each other, and in doing so, wrote up general interfaces we will be working with in the design layout. The timeline will be as follows:

Mon 11/3	Define specific interfaces
	Write Cartograph module
	Start individual module work
Sat 11/15	Individual modules completed
	Combine and start testing simulation
	Refined design document
Wed 11/19	Agreed upon changes to original individual modules completed
	New Erlang modules completed
	Begin visualization
Sat 11/22	Finalize simulation
	Start Final Report