System for Uniform Route-based Transportation Simulation

Final Design Document

Riley Frieburg, Jacob Fynboh,
Nicholas Seel, Justine Erich Recana

Project Iteration V
14th of December 2020

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to outline the bus simulation project we have been tasked with completing. In this document we will look at the design of the project, as well as the various requirements the client has put forth.

1.2 System Overview

This document is being created with our client in mind, as such should be readable from a non-technical point of view. The system is supposed to provide a realistic simulation of bus routes.

1.3 Design Objectives

The goal of the design is to provide a realistic simulation of bus routes with varying busses, and passengers. Users will be able to customize the simulation from the number of busses, number of passengers, the routes that may be taken, and the conditions that may occur in real life bus routes. The system must also be capable of summarizing the results of the simulations into logs, displaying the pure results of each run. There must also be a report that contains some of the same information as the logs, however it will be in an easier to read format and likely contains less raw data then the logs.

1.4 References

For more information check:

Requirements document for project, on Canvas.
Requirements document for this assignment, on Canvas.
Previous assignments on this topic (1-5)

2. Design Overview

2.1 Introduction

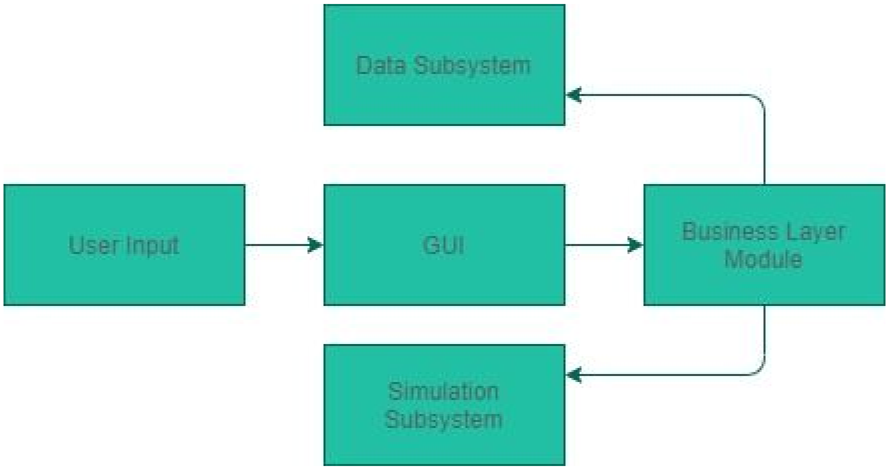
For this project will be implementing an object oriented design. Forming all the entities as objects should be a straightforward approach. This program should should not require a database or server in order to operate properly.

2.2 Environment Overview

We are building a program for the client, as such the program will run natively on the clients' local machines. Being able to store the results of the simulation locally should be sufficient in terms of the size of the memory.

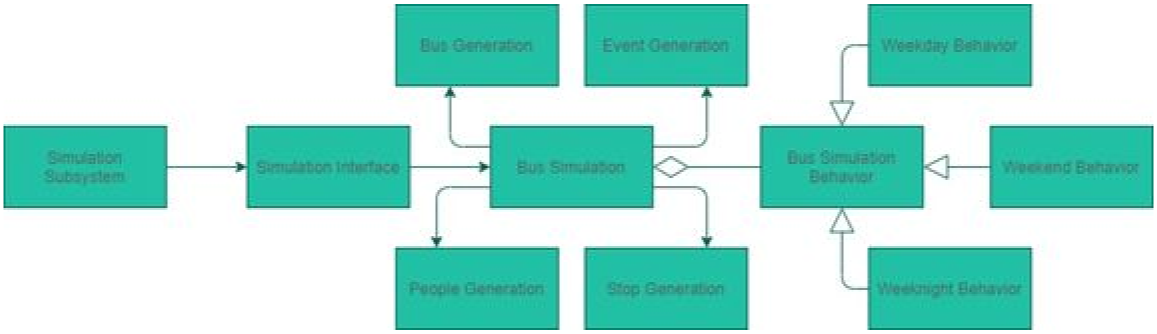
2.3 System Architecture

2.3.1 Top-level system structure of SURTS



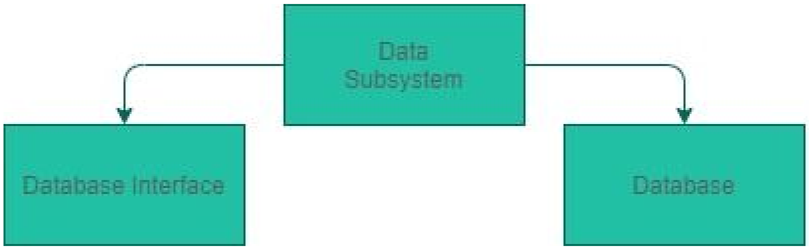
The system has two major components: Data Subsystem and Simulation Subsystem. The Simulation Subsystem runs the Bus Simulation of SURTS. The Data Subsystem runs independently where it stores any object in the domain for later retrieval. These two components can be accessed through the Business Layer Module.

2.3.2 Simulation Subsystem



The Simulation Subsystem consists of Simulation Interface that controls the Bus Simulation. The Bus Simulation is responsible for generating Bus, People, Event, and Stop, while a Strategy Behavioral Pattern is applied to Bus Simulation.

2.3.3 Data Subsystem



The Data Subsystem consists of Database Interface and Database. The interface controls the database which desired objects are to be restored and retrieved when needed.

2.4 Constraints and Assumptions

The main constraint is that the system shall be implemented using an OO language. This was chosen to make the implementation of the simulation and its variables much easier and straight-forward. Another constraint is that the simulation shall be processed in thirty seconds at a maximum when the hardware requirements are met. This was chosen so that the users can configure and run simulations in a timely manner.

Given that non-technical employees may be running the simulation, it is assumed that a graphical user interface would be ideal in this situation. Assuming the client has mostly Windows computers, we shall ensure that the program is compatible with Windows. It is assumed that the computers that will be running the simulation will have the memory required to execute the program, as well as have the storage required to save the logs. The logs will be in a text format which will require little memory in comparison to a simulation run. Another assumption is that the user has gone through the necessary progressions of the system, i.e. user logged in before running the simulation or generated a log after a simulation.

3. Interfaces and Data Stores

3.1 System Interfaces

3.1.1 Entity Creation Interface

Allows users to customize the entities they wish to include in the simulation. This interface will allow users to choose between busses, passengers, and routes. The busses will be customizable in terms of size, numbers, and where they are routed. Passengers can be customized in terms of numbers, as well as locations. Routes can also be pre-configured within the simulation, allowing for different numbers of routes as well as where that route will stop.

3.1.2 Log and Report Interface

The log and report features shall be able to observe the simulation, store the data, and return it in a human-readable format. Logs will display only raw data taken from the simulation itself; things like: number of busses used, number of passengers, and number of routes among other things. The report will then take the raw data shown in the logs and present the summarized information.

3.2 Data Stores

Data on the simulation runs will be stored locally on the computer. These records shall be accessible within the program and shall not be removed unless the user has specified the results should not be saved. We will be focused on storing only the logs from simulation runs, as reports can be generated from the log information, thus making saving of reports to be a waste of space when they can easily be regenerated at any time.

4. Structural Design

4.1 Class Diagram

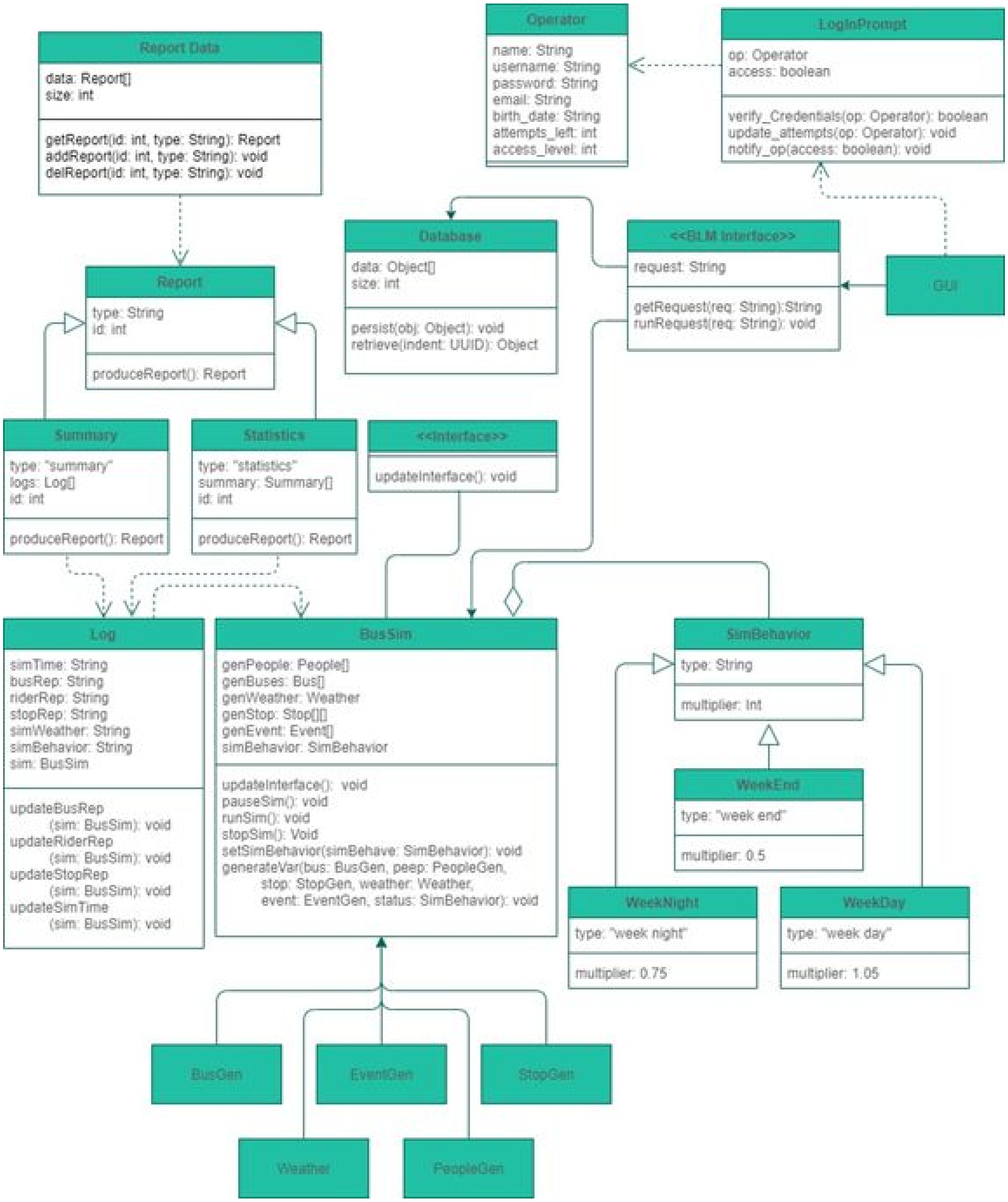


Figure 1: SURTS General UML Diagram

Figure 1 illustrates how the system would look like with BLM and Database Abstraction. The variables used by BusSim are dissected into parts for easy reading. BusSim will run and generate logs, which can be converted into different reports (e.g. summary and statistics report) for safekeeping. The operator can also choose to access the database to preserve any domain within the scope.

4.1.1 Class: LoginPrompt

Purpose: Assign the credentials to users who can access the system
Constraints:
username – format is given by the company using the system (no symbols)
password – length shall be in range 8 to 15
Persistent: Yes (always available)

4.1.1.1 Attribute Descriptions

1. Attribute: op
   Type: Operator
   Description: The operator object that holds the credentials of the user
   Constraints: None
2. Attribute: access
   Type: boolean
   Description: tells whether the credentials given are right for system access
   Constraints: true or false

4.1.1.2 Method Descriptions

1. Method: verify_Credentials
   Return Type: boolean
   Parameters: Operator
   Return value: True if the credentials are valid
   Description: Returns whether or not the credentials are valid
   Constraints: None
2. Method: update_attempts
   Return Type: void
   Parameters: Operator
   Return value: None
   Description: updates the current attempts to enter the correct credentials
   Constraints: None
3. Method: notify_op
   Return Type: void
   Parameters: boolean
   Return value: None
   Description: notifies the user if the access is granted or not
   Constraints: None

4.1.2 Class: Operator

Purpose: Store the credentials of the user
Constraints: None
Persistent: Yes (always available)

4.1.2.1 Attribute Descriptions

1. Attribute: name
   Type: String
   Description: The name of the operator
   Constraints: None
2. Attribute: username
   Type: String
   Description: The username of the operator
   Constraints: Not null
3. Attribute: password
   Type: String
   Description: The password of the operator
   Constraints: Not null
4. Attribute: email
   Type: String
   Description: The email for this operator
   Constraints: None
5. Attribute: birth_date
   Type: String
   Description: The birth date of this operator
   Constraints: None
6. Attribute: attempts_left
   Type: int
   Description: The number of remaining login attempts allowed for this operator
   Constraints: Non-negative
7. Attribute: access_level
   Type: int
   Description: A number representing the access level of this operator
   Constraints: None

4.1.2.2 Method Descriptions

None

4.1.3 Class: BusSim

Purpose: Run the Simulation
Constraints: None
Persistent: Yes (always available)

4.1.3.1 Attribute Descriptions

1. Attribute: genPeople
   Type: People[]
   Description: An array representing People

2. Attribute: genBuses
   Type: Bus[]
   Description: An array representing busses
   Constraints: None
3. Attribute: genWeather
   Type: Weather
   Description: The Weather of the simulation
   Constraints: None
4. Attribute: genStop
   Type: Stop[][]
   Description: A 2D array representing generated Stops in each Route
   Constraints: None
5. Attribute: genEvent
   Type: Event[]
   Description: An array holding the generated Events
   Constraints: None
6. Attribute: simBehavior
   Type: SimBehavior
   Description: The behavior of the Simulation
   Constraints: None

4.1.3.2 Method Descriptions

1. Method: updateInterface
   Return Type: void
   Parameters: None
   Return value: None
   Description: Updates the interface in the simulation
   Constraints: None
2. Method: pauseSim
   Return Type: void
   Parameters: None
   Return value: None
   Description: Pauses the simulation
   Constraints: None
3. Method: runSim
   Return Type: void
   Parameters: None
   Return value: None
   Description: Runs the simulation
   Constraints: None
4. Method: stopSim
   Return Type: void
   Parameters: None
   Return value: None

Description: Stops the simulation
Constraints: None

5. Method: setSimBehavior
   Return Type: void
   Parameters: SimBehavior
   Return value: None
   Description: sets the behavior of the simulation
   Constraints: None
6. Method: generateVar
   Return Type: void
   Parameters: BusGen, PeopleGen, StopGen, Weather, EventGen, SimBehavior
   Return value: None
   Description:sets every variable needed for the simulation
   Constraints: None

4.1.4 Class: Log

Purpose: Store data from simulation runs and allow for later access
Constraints: None
Persistent: Yes (always available)

4.1.4.1 Attribute Descriptions

1. Attribute: simTime
   Type: String
   Description: String representing the time of simulation
   Constraints: None
2. Attribute: busRep
   Type: String
   Description: Report returned by busses
   Constraints: None
3. Attribute: rideRep
   Type: String
   Description: Report returned by riders
   Constraints: None
4. Attribute: stopRep
   Type: String
   Description: Report returned by stops
   Constraints: None
5. Attribute: simWeather
   Type: String
   Description: Report regarding simulation weather
   Constraints: None

6. Attribute: simBehavior
   Type: String
   Description: Report regarding simulation behavior
   Constraints: None
7. Attribute: sim
   Type: BusSim
   Description: The bus simulation itself
   Constraints: None

4.1.4.2 Method Descriptions

1. Method: updateBusRep
   Return Type: void
   Parameters: BusSim
   Return value: None
   Description: Updates bus report
   Constraints: None
2. Method: updateRiderRep
   Return Type: void
   Parameters: BusSim
   Return value: None
   Description: Updates rider report
   Constraints: None
3. Method: updateStopRep
   Return Type: void
   Parameters: BusSim
   Return value: None
   Description: Updates stop report
   Constraints: None
4. Method: updateSimTime
   Return Type: void
   Parameters: BusSim
   Return value: None
   Description: Updates elapsed time in simulation
   Constraints: None

4.1.5 Class: Summary

Purpose: Creates a summary from report class
Constraints: None
Persistent: Yes (always available)

4.1.5.1 Attribute Descriptions

1. Attribute: logs
   Type: logs[]

Description: An array representing logs

2. Attribute: id
   Type: int
   Description: Identification for each summary
   Constraints: None

4.1.5.2 Method Descriptions

1. Method: produceReport
   Return Type: Report
   Parameters: None
   Return value: None
   Description: Generates a report for the users to view
   Constraints: None

4.1.6 Class: Statistic

Purpose: Helps generate the statistics in the log
Constraints: None
Persistent: Yes (always available)

4.1.6.1 Attribute Descriptions

1. Attribute: summary
   Type: Summary[]
   Description: An array representing summaries
   Constraints: None
2. Attribute: id
   Type: int
   Description: A number representing the summaries in summary[]
   Constraints: Greater than 0

4.1.6.2 Method Descriptions

1. Method: produceReport
   Return Type: Report
   Parameters: None
   Return value: None
   Description: Generates a report for the users to view
   Constraints: None

4.1.7 Class: Report

Purpose: generate different Reports
Constraints: None
Persistent: Yes (always available)

4.1.7.1 Attribute Descriptions

1. Attribute: id
   Type: int
   Description: A number representing the report
   Constraints: Greater than 0

4.1.7.2 Method Descriptions

2. Method: produceReport
   Return Type: Report
   Parameters: None
   Return value: None
   Description: Generates a report for the users to view
   Constraints: None

4.1.8 Class: ReportData

Purpose: Stores and reports data for use in generating reports
Constraints: None
Persistent: Yes (always available)

4.1.8.1 Attribute Descriptions

1. Attribute: data
   Type: Report[]
   Description: An array of reports allowing access to past reports
   Constraints: None
2. Attribute: size
   Type: int
   Description: The number of records stored in data
   Constraints: Greater than 0

4.1.8.2 Method Descriptions

1. Method: getReport
   Return Type: Report
   Parameters: int, String
   Return value: None
   Description: Allows users to select from data
   Constraints: None
2. Method: addReport
   Return Type: void
   Parameters: int, String
   Return value: None
   Description: Allows users to add new reports to data
   Constraints: None
3. Method: delReport

Return Type: void
Parameters: int, String
Return value: None
Description: Allows users to remove reports from data
Constraints: None

4.1.9 Class: SimBehavior

Purpose: Simulates change in amount of people throughout the week
Constraints: None
Persistent: Yes (always available)

4.1.9.1 Attribute Descriptions

1. Attribute: SimBehavior
   Type: String
   Description: Reports time of week
   Constraints: None
2. Attribute: multiplier
   Type: int
   Description: Stores multiplier that dictates amount of riders and traffic.
   Constraints: None

4.1.10 Class: WeekDay

Purpose: sim behavior during weekdays
Constraints: None
Persistent: Yes (always available)

4.1.10.1 Attribute Descriptions

1. Attribute: week
   Type: String
   Description: Reports time of week
   Constraints: None
2. Attribute: multiplier
   Type: int
   Description: Stores multiplier that dictates amount of riders and traffic.
   Constraints: None

4.1.11 Class: WeekNight

Purpose: sim behavior during weeknights
Constraints: None
Persistent: Yes (always available)

4.1.11.1 Attribute Descriptions

3. Attribute: week
   Type: String

Description: Reports time of week
Constraints: None

4. Attribute: multiplier
Type: int
Description: Stores multiplier that dictates amount of riders and traffic.
Constraints: None

4.1.12 Class: WeekEnd

Purpose: sim behavior during weekends
Constraints: None
Persistent: Yes (always available)

4.1.12.1 Attribute Descriptions

1. Attribute: week
Type: String
Description: Reports time of week
Constraints: None
2. Attribute: multiplier
Type: int
Description: Stores multiplier that dictates amount of riders and traffic.
Constraints: None

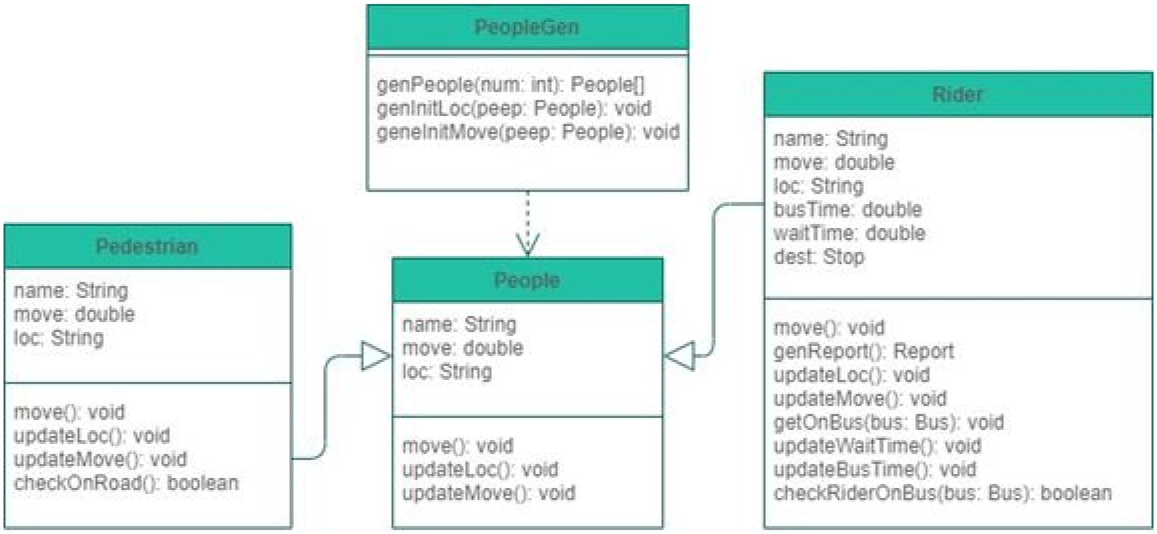


Figure 2: People Generation

Figure 2 illustrates different people to be generated in the simulation. Bothe riders and pedestrians have an impact on the performance of the simulation. Riders and pedestrians could move slower than expected, which will affect the simulation time and speed of the buses. Even though the system is based on rider-need transportation, the presence of the pedestrians also plays a part.

4.1.13 Class: People

Purpose: Base class for riders and pedestrians
Constraints: None
Persistent: Yes (always available)

4.1.13.1 Attribute Descriptions

1. Attribute: name
   Type: string
   Description: Name of people
   Constraints: None
2. Attribute: move
   Type: double
   Description: Tracks movement of people
   Constraints: None
3. Attribute: loc
   Type: string
   Description: Updates the location attribute
   Constraints: Must be within simulation

4.1.13.2 Method Descriptions

1. Method: move
   Return Type: void
   Parameters: none
   Return value: None
   Description: Moves users around simulation
   Constraints: None
2. Method: updateLoc
   Return Type: void
   Parameters: none
   Return value: None
   Description: Updates location
   Constraints: None
3. Method: updateMove
   Return Type: void
   Parameters: none
   Return value: None
   Description: Updates move values
   Constraints: None

4.1.14 Class: Rider

Purpose:
Constraints: None
Persistent: Yes (always available)

4.1.14.1 Attribute Descriptions

1. Attribute: name
   Type: string
   Description: Name of people
   Constraints: None
2. Attribute: move
   Type: double
   Description: Tracks movement of people
   Constraints: None
3. Attribute: loc
   Type: string
   Description: Shows location of passenger
   Constraints: Must be within simulation
4. Attribute: busTime
   Type: double
   Description: Time spent on the bus
   Constraints: Greater than 0
5. Attribute: waitTime
   Type: double
   Description: Time spent waiting
   Constraints: Must be greater than 0
6. Attribute: dest
   Type: Stop
   Description: Where the user is supposed to stop
   Constraints: None

4.1.14.2 Method Descriptions

1. Method: move
   Return Type: void
   Parameters: none
   Return value: None
   Description: Moves users around simulation
   Constraints: None
2. Method: updateLoc
   Return Type: void
   Parameters: none
   Return value: None
   Description: Updates location
   Constraints: None
3. Method: updateMove
   Return Type: void
   Parameters: none
   Return value: None
   Description: Updates move values

Constraints: None

4. Method: genReport
   Return Type: Report
   Parameters: none
   Return value: None
   Description: Generates a report
   Constraints: None
5. Method: getOnBus
   Return Type: void
   Parameters: Bus
   Return value: None
   Description: Loads rider on bus
   Constraints: Rider must be near bus
6. Method: updateWaitTime
   Return Type: void
   Parameters: none
   Return value: None
   Description: Changes the value of the wait time for the passenger
   Constraints: None
7. Method: updateBusTime
   Return Type: void
   Parameters: none
   Return value: None
   Description: Changes the value of the bus time for the passenger
   Constraints: None

4.1.15 Class: Pedestrian

Purpose: Provide the simulation with pedestrians
Constraints: None
Persistent: Yes (always available)

4.1.15.1 Attribute Descriptions

1. Attribute: name
   Type: string
   Description: Name of people
   Constraints: None
2. Attribute: move
   Type: double
   Description: Tracks movement of people
   Constraints: None
3. Attribute: loc
   Type: string
   Description: Shows location of passenger

4.1.15.2 Method Descriptions

1. Method: move
   Return Type: void
   Parameters: none
   Return value: None
   Description: Moves users around simulation
   Constraints: None
2. Method: updateLoc
   Return Type: void
   Parameters: none
   Return value: None
   Description: Updates location
   Constraints: None
3. Method: updateMove
   Return Type: void
   Parameters: none
   Return value: None
   Description: Updates move values
   Constraints: None

4.1.16 Class: PeopleGen

Purpose: Generate new people entities
Constraints: None
Persistent: Yes (always available)

4.1.16.1 Method Descriptions

1. Method: genPeople
   Return Type: People[]
   Parameters: int
   Return value: None
   Description: Creates new people
   Constraints: None
2. Method: genInitLoc
   Return Type: void
   Parameters: People
   Return value: None
   Description: Creates location for people
   Constraints: None
3. Method: genInitMove
   Return Type: void
   Parameters: People
   Return value: None
   Description: Creates move for people
   Constraints: None

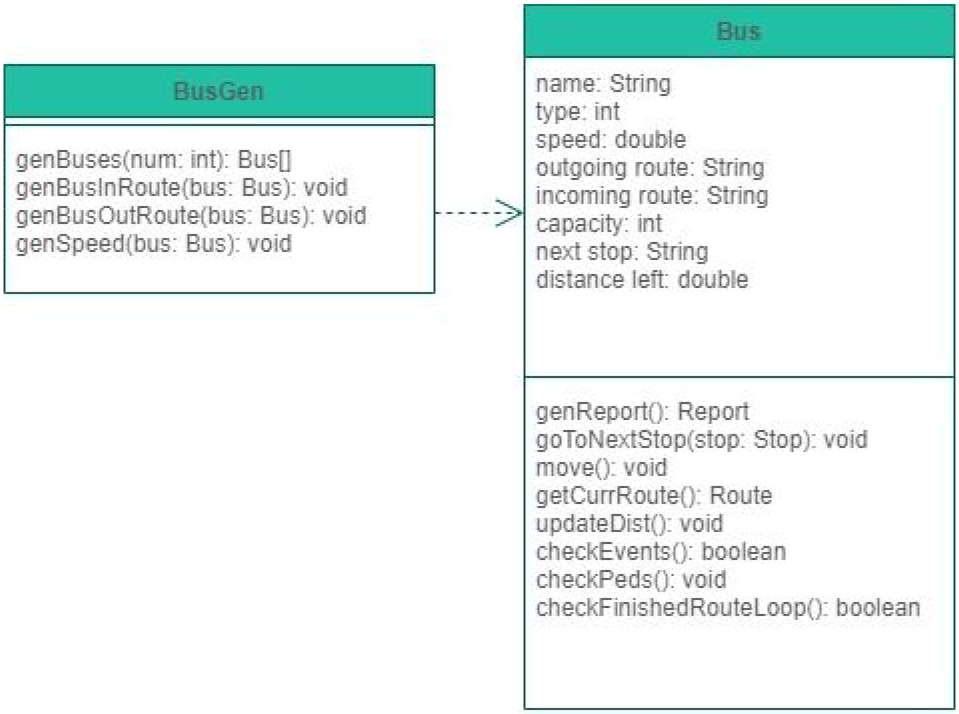


Figure 3: Bus Generation

Figure 3 illustrates the creation and the generation of buses. Each bus has its own ingoing and outgoing routes. It can also track which stop is next and how much seats are available to prevent bus overloading.

4.1.17 Class: Bus

Purpose: Provide the simulation with busses
Constraints: None
Persistent: Yes (always available)

4.1.17.1 Attribute Descriptions

1. Attribute: name
   Type: String
   Description: Name of bus
   Constraints: None
2. Attribute: type
   Type: int
   Description:
   Constraints: None
3. Attribute: speed
   Type: double
   Description: Tracks speed of bus
   Constraints: Greater than 0

4. Attribute: outgoing route
   Type: string
   Description: Route bus will take after leaving start
   Constraints: None
5. Attribute: incoming route
   Type: string
   Description: Route bus will take going back to start
   Constraints: None
6. Attribute: capacity
   Type: int
   Description: Room on bus
   Constraints: Greater than 0
7. Attribute: next stop
   Type: string
   Description: Next planned bus stop
   Constraints: None
8. Attribute: distance left
   Type: double
   Description: How far the bus needs to drive
   Constraints: Nonnegative

4.1.17.2 Method Descriptions

1. Method: genReport
   Return Type: Report
   Parameters: none
   Return value: None
   Description: Returns a report
   Constraints: None
2. Method: move
   Return Type: void
   Parameters: none
   Return value: None
   Description: Moves busses around simulation
   Constraints: None
3. Method: move
   Return Type: void
   Parameters: none
   Return value: None
   Description: Moves the bus
   Constraints: None
4. Method: getCurrRoute
   Return Type: void
   Parameters: Stop
   Return value: None

Description: Returns value of route that the stop is on
Constraints: None

5. Method: updateDist
   Return Type: void
   Parameters: none
   Return value: None
   Description: Updates distance
   Constraints: None
6. Method: checkEvents
   Return Type: boolean
   Parameters: none
   Return value: None
   Description: Returns true or false if a new event has occurred
   Constraints: None
7. Method: checkPeds
   Return Type: void
   Parameters: none
   Return value: None
   Description: Checks for pedestrians
   Constraints: None
8. Method: checkRiderOnBus
   Return Type: boolean
   Parameters: none
   Return value: None
   Description: Returns true or false if a new passenger has boarded
   Constraints: None

4.1.18 Class: BusGen

Purpose: Generate new bus entities
Constraints: None
Persistent: Yes (always available)

4.1.18.1 Method Descriptions

1. Method: genBuses
   Return Type: Bus[]
   Parameters: int
   Return value: None
   Description: Creates new busses
   Constraints: Nones
2. Method: genBusInRoute
   Return Type: void
   Parameters: bus
   Return value: None
   Description: Creates a new bus incoming route

3. Method: genBusOutRoute
   Return Type: void
   Parameters: bus
   Return value: None
   Description: Creates a new bus outgoing route
   Constraints: None
4. Method: genSpeed
   Return Type: void
   Parameters: bus
   Return value: None
   Description: Creates a new bus speed
   Constraints: None

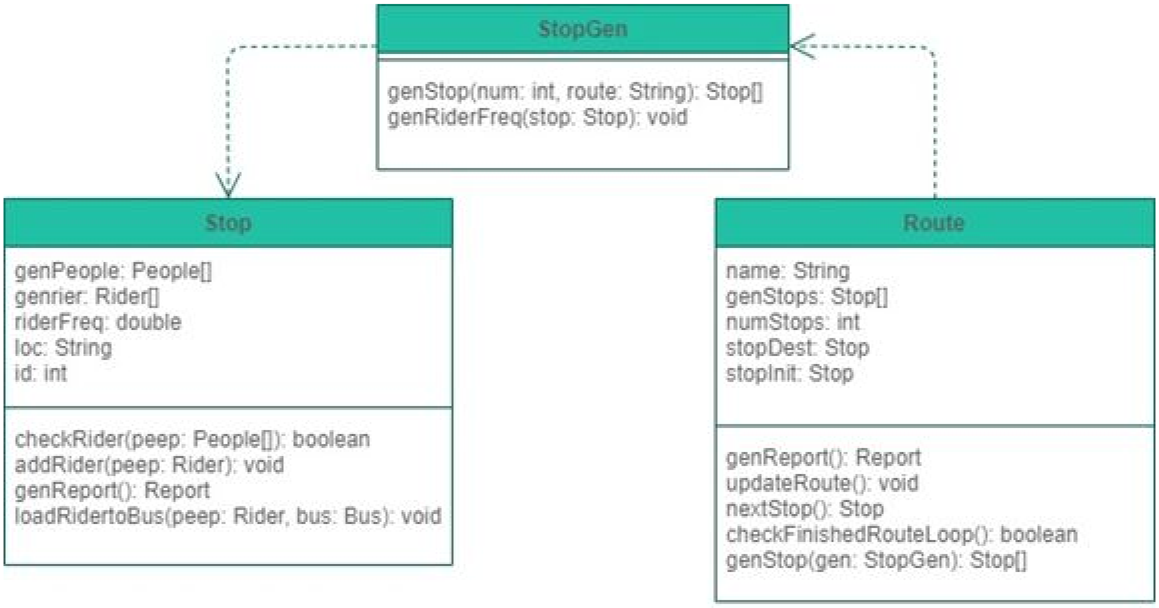

StopGen
genStop(num: int, route: String): Stop[]
genRiderFreq(stop: Stop): void
Stop
genPeople: People[]
genrier: Rider[]
riderFreq: double
loc: String
id: int
checkRider(peep: People[]): boolean
addRider(peep: Rider): void
genReport(): Report
loadRidertoBus(peep: Rider, bus: Bus): void
Route
name: String
genStops: Stop[]
numStops: int
stopDest: Stop
stopInit: Stop
genReport(): Report
updateRoute(): void
nextStop(): Stop
checkFinishedRouteLoop(): boolean
genStop(gen: StopGen): Stop[]

Figure 4: Stop Generation

Figure 4 illustrates the creation and the generation of stops as well as the relationship between the route and the generated stops.

4.1.19 Class: Stop

Purpose: Provide the simulation with bus stops
Constraints: None
Persistent: Yes (always available)

4.1.19.1 Attribute Descriptions

1. Attribute: genPeople
   Type: People[]
   Description: Array of people
   Constraints: None
2. Attribute: genRider
   Type: Rider[]
   Description: Array of riders
   Constraints: None
3. Attribute: riderFreq
   Type: Double
   Description: Frequency of riders appearing
   Constraints: None
4. Attribute: loc
   Type: double
   Description: Location of stop
   Constraints: None
5. Attribute: id
   Type: int
   Description: Stop id
   Constraints: None

4.1.19.2 Method Descriptions

1. Method: checkRider
   Return Type: boolean
   Parameters: people
   Return value: None
   Description: Reports if a rider is present
   Constraints: Nones
2. Method: addRider
   Return Type: void
   Parameters: Rider
   Return value: None
   Description: Adds a rider to a stop
   Constraints: None
3. Method: genReport
   Return Type: Report
   Parameters: None
   Return value: None

Description: Generates a report regarding the stop
Constraints: None

4. Method: loadRidertoBus
Return Type: void
Parameters: rider, bus
Return value: None
Description: Adds a passenger to a bus
Constraints: None

4.1.20 Class: StopGen

Purpose: Generate new stops within the simulation
Constraints: None
Persistent: Yes (always available)

4.1.20.1 Method Descriptions

1. Method: genStop
Return Type: Stop[]
Parameters: int, String
Return value: None
Description: Creates a new stop
Constraints: Nones
2. Method: genRiderFrequency
Return Type: void
Parameters: Stop
Return value: None
Description: Creates rider frequency value for stop
Constraints: None

4.1.21 Class: Route

Purpose: Create a path for each bus to follow in the simulation
Constraints: None
Persistent: Yes (always available)

4.1.21.1 Attribute Descriptions

1. Attribute: name
Type: String
Description: name of the route
Constraints: Must be within the system's scope
2. Attribute: genStops
Type: Stop[]
Description: a collection of Stops to create a bus Route
Constraints: None

3. Attribute: numStops
   Type: int
   Description: number of Stops generated for each Route
   Constraints: None
4. Attribute: stopDest
   Type: Stop
   Description: the final Stop for the Route
   Constraints: None
5. Attribute: stopInit
   Type: Stop
   Description: the initial Stop for the Route
   Constraints: None

4.1.21.2 Method Descriptions

1. Method: genReport
   Return type: Report
   Parameters: None
   Return value: Route Report
   Description: Report for each Route
   Constraints: None
2. Method: updateRoute
   Return type: void
   Parameters: None
   Return value: None
   Description: updates the name of the Route
   Constraints: must be within the system's scope
3. Method: nextStop
   Return type: Stop
   Parameters: None
   Return value: Stop
   Description: gets the next Stop of the bus
   Constraints: None
4. Method: checkFinishedRouteLoop
   Return type: boolean
   Parameters: None
   Return value: None
   Description: checks if the bus finished the entire route loop
   Constraints: None
5. Method: genStop
   Return type: Stop[]
   Parameters: StopGen
   Return value: a list of Stops generated
   Description: sets a list of generated Stops for the Route
   Constraints: None

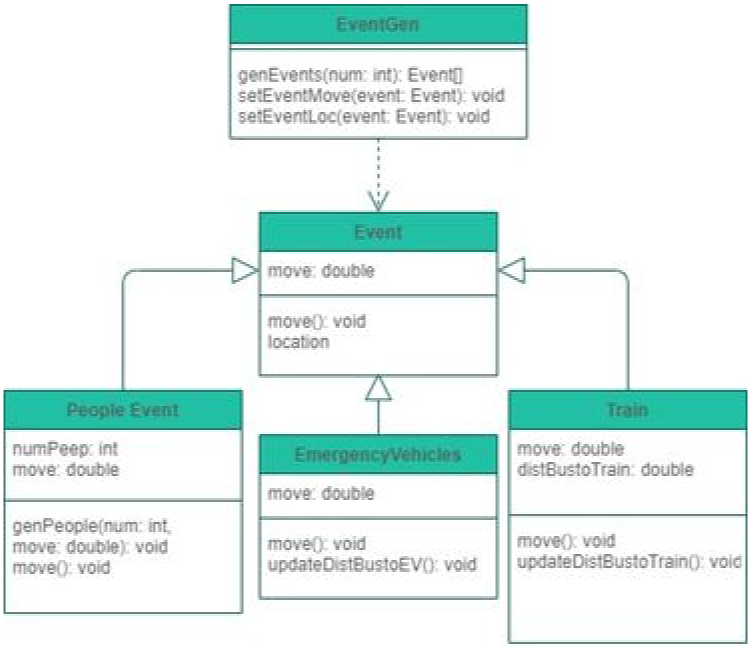


Figure 5: Event Generation

Figure 5 illustrates the creation and generation of different events that will influence the simulation

4.1.22 Class: Event

Purpose: generate different events
Constraints: None
Persistent: Depends on the user

4.1.22.1 Attribute Descriptions

1. Attribute: move
   Type: double
   Description: the movement of the event
   Constraints: None

4.1.22.2 Method Descriptions

1. Method: move
   Return type: void
   Parameters: None
   Return value: None
   Description: moves the event
   Constraints: None
2. Method: setLocation
   Return type: void
   Parameters: double, double
   Return value: None
   Description: sets the starting point of the event

4.1.23 Class: EventGen

Purpose: Generates a couple of Events in the simulation
Constraints: None
Persistent: Depends on the user

4.1.23.1 Attribute Descriptions

None

4.1.23.2 Method Descriptions

1. Method: genEvents
   Return type: Event[]
   Parameters: int
   Return value: Events that are generated
   Description: generates multiple events
   Constraints: None
2. Method: setEventMove
   Return type: void
   Parameters: Event
   Return value: None
   Description: sets the move for each Event
   Constraints: None
3. Method: setEventLoc
   Return type: void
   Parameters: Event
   Return value: None
   Description: sets the location of each Event
   Constraints: None

4.1.24 Class: PeopleEvent

Purpose: generates a people event in the simulation
Constraints: None
Persistent: Depends on the user

4.1.24.1 Attribute Descriptions

1. Attribute: numPeep
   Type: int
   Description: number of people generated in the event
   Constraints: None
2. Attribute: move
   Type: double
   Description: the movement of the people generated
   Constraints: None

4.1.24.2 Method Descriptions

3. Method: genPeople
   Return type: void
   Parameters: int, double
   Return value: None
   Description: generates people in an event
   Constraints: 20 =< num < 200000 people generated
4. Method: move
   Return type: void
   Parameters: None
   Return value: None
   Description: moves the people in the even
   Constraints: None

4.1.25 Class: EmergencyVehicles

Purpose: generates an emergency vehicle in the simulation
Constraints: None
Persistent: Depends on the user

4.1.25.1 Attribute Descriptions

1. Attribute: move
   Type: double
   Description: the movement of the EV
   Constraints: None

4.1.25.2 Method Descriptions

1. Method: move
   Return type: void
   Parameters: None
   Return value: None
   Description: moves the EV in the simulation
   Constraints: None
2. Method: updateDistBustoEV
   Return type: void
   Parameters: None
   Return value: None
   Description: updates the distance between the bus and the EV
   Constraints: None

4.1.26 Class: Train

Purpose: generates a train event in the simulation
Constraints: None
Persistent: Depends on the user

4.1.26.1 Attribute Descriptions

1. Attribute: move
   Type: double
   Description: the movement of the train
   Constraints: None
2. Attribute: distBusToTrain
   Type: double
   Description: train’s distance from the closest bus by the track
   Constraints: >= 0.0 and <= 1000.0 meters

4.1.26.2 Method Descriptions

1. Method: move
   Return type: void
   Parameters: None
   Return value: None
   Description: moves the train in the simulation
   Constraints: None
2. Method: updateDistBustoTrain
   Return type: void
   Parameters: None
   Return value: None
   Description: updates the distance between the bus and the train
   Constraints: None

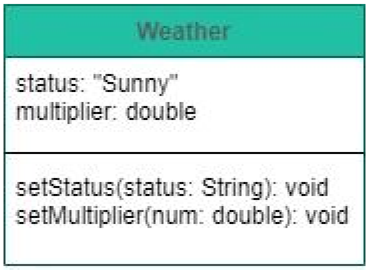

Weather
status: "Sunny"
multiplier: double
setStatus(status: String): void
setMultiplier(num: double): void

Figure 6: Weather

Figure 6 illustrates the creation of weather that will affect the performance of buses and passengers in the simulation.

4.1.27 Class: Weather

Purpose:

Constraints: None

Persistent: Yes (always available)

4.1.27.1 Attribute Descriptions

1. Attribute: status
   Type: String
   Description: tells what kind of weather is available in the simulation
   Constraints: None
2. Attribute: multiplier
   Type: double
   Description: sets the multiplier based on the status of the weather
   Constraints: 0.0 and positive double and < 5

4.1.27.2 Method Descriptions

1. Method: setStatus
   Return type: void
   Parameters: String
   Return value: None
   Description: sets the status in the simulation
   Constraints: Can only be modified based on the available weather conditions pre-defined in the system
2. Method: setMultiplier
   Return type: void
   Parameters: double
   Return value: None
   Description: sets the multiplier for the movement of the simulation
   Constraints: 0.0 and positive double and < 5

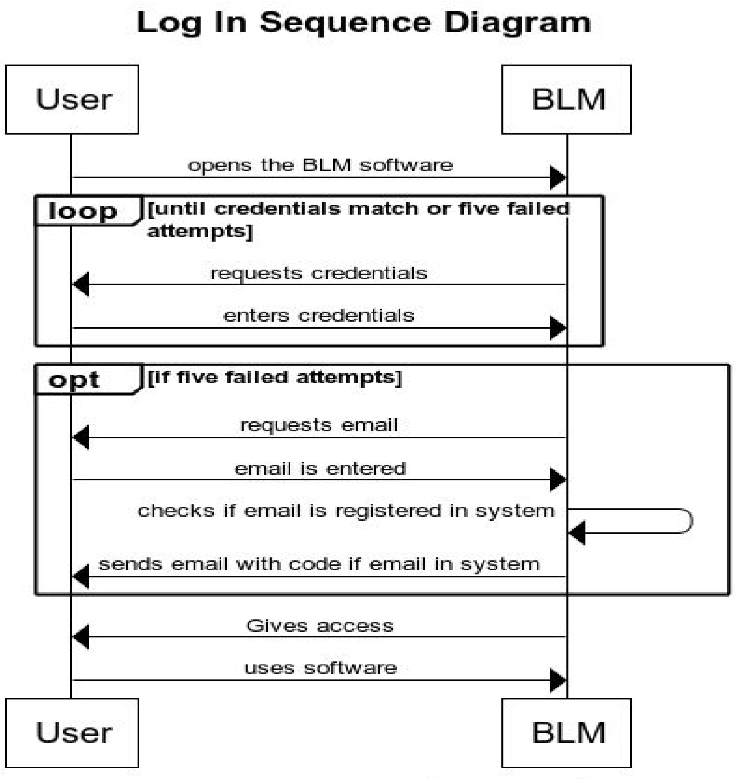
4.2 Design Reasoning

The overall design of the BLM is in an object oriented style. Due to the complexity of the simulation, OO was the chosen coding style since it would make it easier to implement the simulation with objects since there are many moving parts in the simulation. Another reason OO was the best choice is because it would be easier to change attributes in the simulation if we were able to use interfaces and abstract classes so busses for example could hold varying amounts of passengers.

An OO Design Pattern was used in the Class Diagram. The authors have decided to use a Strategy Pattern, one of the Behavior Patterns in OO Design. We intended to use the Strategy Pattern since the fluidity of the transportation in real life depends on what time of day it is. Most of the time, the bus transportation is busy during the weekdays, while average in weeknights and slow during the weekends. This will inform the bus company how many busses to produce to meet the consumers' demands. The Strategy Pattern is also expendable, which means we can add more simulation behaviors such as the Holidays and the Parades. The addition of behaviors will greatly affect the runtime of the simulation without affecting the structure of the system itself.

5. Dynamic Model

5.1 Scenarios



Sequence Diagram for login and logout (modified from Assignment 3)

5.1.1 Login/Logout

Use Case Name #1: Log In

Summary: The user input their user ID and password

Basic Course of Events:

1. The user opens the system’s software
2. The user types the user ID and the password
3. The system validates the user’s ID and password of the user

Alternative Paths: In step 4, if the user ID is incorrect, the system will show an error message and will ask the user again.

Exception Paths: In step 4, if the password is incorrect, the user will get five (5) attempts after which the system will provide an error message and will ask the user to enter a code that is emailed to an entered email if it is registered in the system.

Extension Points: The users can use the system for extended period until they log out

Trigger: The user wants to use the system.

Assumptions: The user is an accredited user of the system and received a username and password to gain access to the system.
Precondition: The SURTS is working properly.
Postcondition: The user will have access to the system.

Use Case Name #2: Log Out
Summary: The users can log out to the system if they are not actively using it.
Basic Course of Events:

1. Completion of use case Log In
2. The user will choose to press log out if wanted

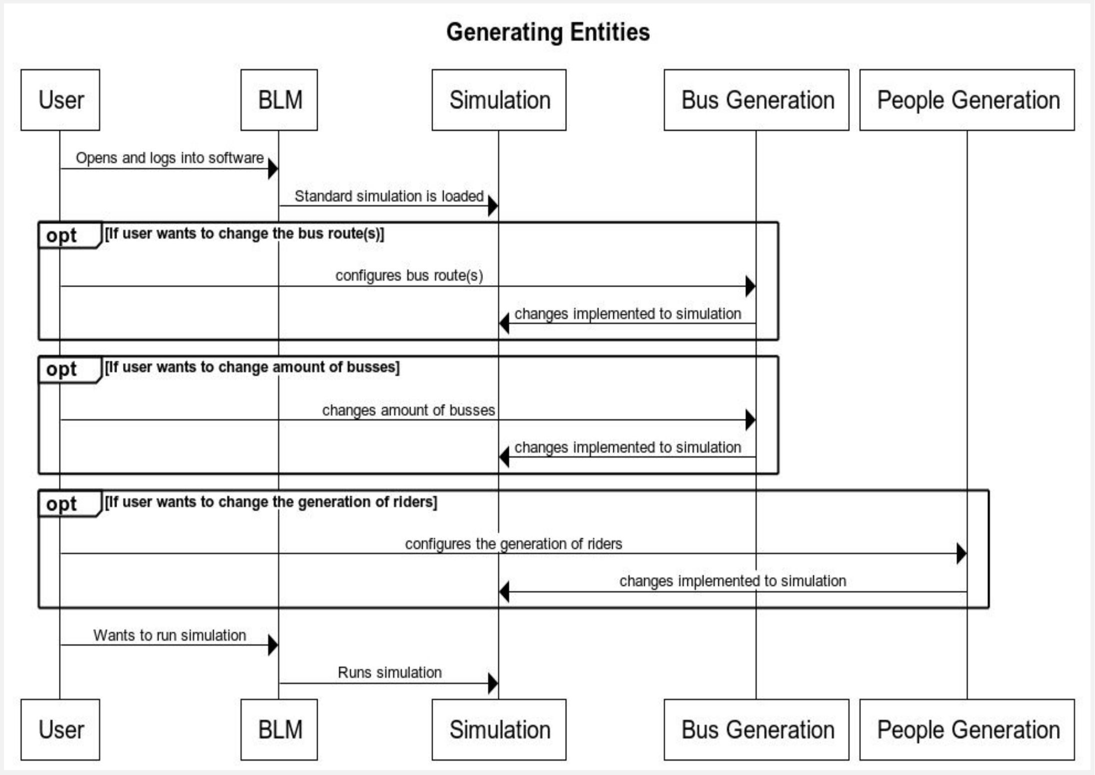
Alternative Paths: In step 2, if the user cancels to log out, the system will stay active.
Exception Paths: In step 2, if the user confirms to log out while the simulation is running, the SURTS will ask the user if they want to terminate the simulation and log out. If the user confirms, the simulation will terminate, the changes will not be saved, and the system will close. If they want to cancel logging out, the simulation will continue running.
Extension Points: None
Trigger: The user wants to not use the system at some point in time.
Assumptions: The user has completed the use case Log In
Precondition: The Log In use case has been successfully completed.
Postcondition: The system will close.

5.1.2 Running Simulation

Use Case Name #3: Run Simulation
Summary: The system will run the default simulation using the default information given by users.
Basic Course of Events:

1. Completed the use case Log In
2. The system will generate a default weather based on the current weather in real life
3. The system will ask the users about the passenger, bus, and route information
4. The user enters the information.
5. The system will generate a default simulation based on the information given.

Alternative Paths: None
Exception Paths: None
Extension Points: None
Trigger: None.
Assumptions: The user completed the use case Log In
Precondition: The Log In use case has been successfully completed.
Postcondition: The default simulation will be running by the system.


Generating Entities
User
BLM
Simulation
Bus Generation
People Generation
Opens and logs into software
Standard simulation is loaded
opt
[If user wants to change the bus route(s)]
configures bus route(s)
changes implemented to simulation
opt
[If user wants to change amount of busses]
changes amount of busses
changes implemented to simulation
opt
[If user wants to change the generation of riders]
configures the generation of riders
changes implemented to simulation
Wants to run simulation
Runs simulation
User
BLM
Simulation
Bus Generation
People Generation

Sequence diagram for generating routes, busses, and passengers (modified from Assignment 3)

5.1.3 Generating Entities

Use Case Name #3: Generate Routes

Summary: Employees within the transportation department run simulations of bus routes with options to configure the simulations and receive the results.

Basic Course of Events:

1. Employee loads simulation.
2. Employee configures routes, riders, and buses.
3. Employee runs simulation.
4. Simulations complete
5. Output is returned for the employee to view.
6. Final report is returned for the employee to review.

Alternative Paths: Employee does not need to change configuration of simulation, skip step 2.

Exception Paths: None

Extension Points: Multiple simulations are ran, after step 4 we return to step 1 and proceed as previously.

Trigger: Employee wants to run a simulation.

Assumptions: The employees want some sort of configuration of busses, riders, and routes. Employees are capable of configuring the simulation.

Precondition: The simulation has already been configured at the time of running.

Postcondition: The program has found what the employee was looking for in the simulation.

Use Case Name #4: Generate Buses
Summary: The user can generate a desired number of buses in the system
Basic Course of Events:

1. Completion of use case Log In
2. The system will ask the user how many buses to be generated
3. The user enters the number of buses
4. The system will ask the user what type each bus is
5. The user enters the type of each bus
6. The system will generate the buses according to the information given

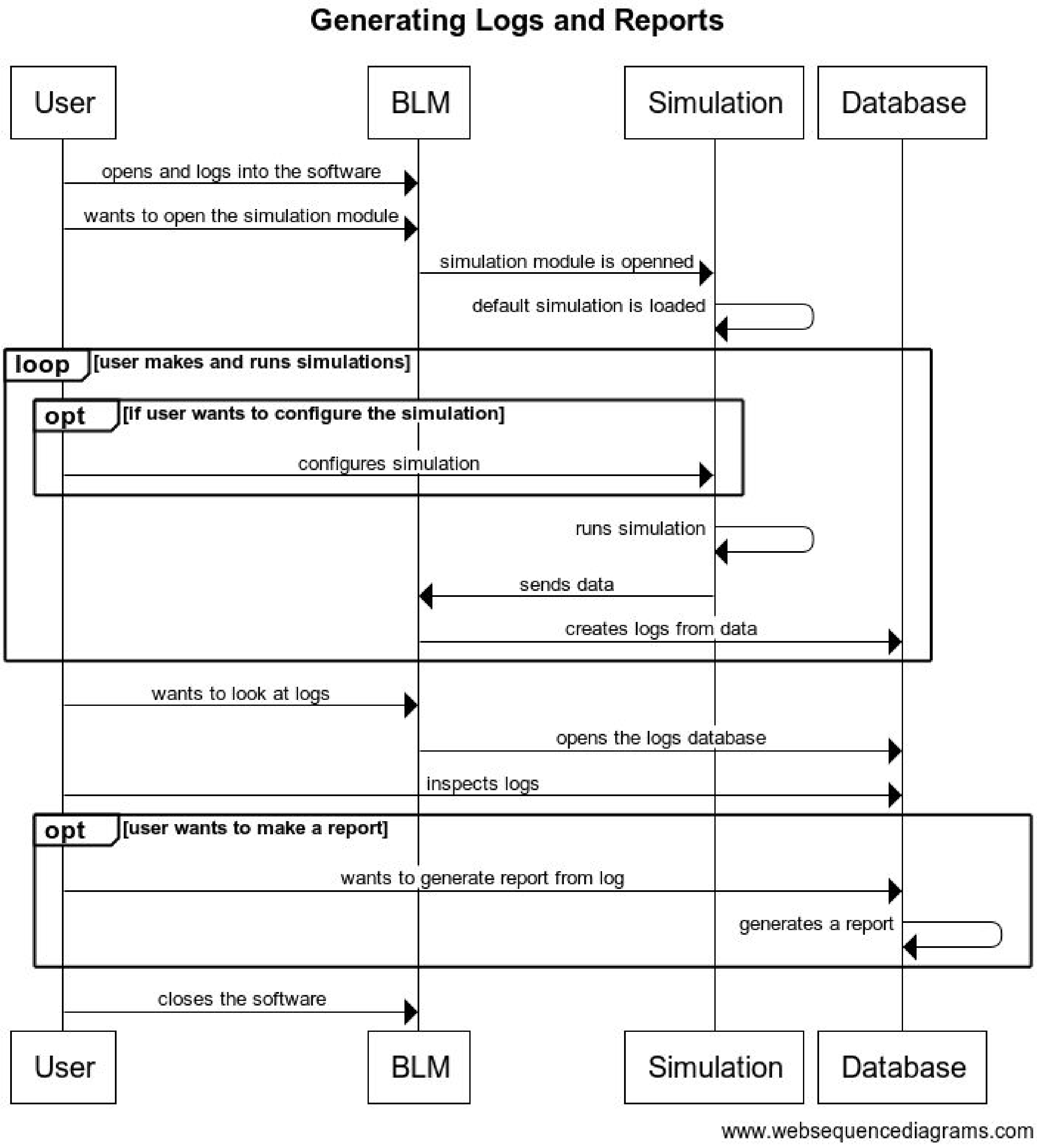
Alternative Paths: In steps 3 and 5, the user will have the option to cancel entering the information. If the user chooses to cancel giving the information, the simulation will run in default.
Exception Paths: None
Extension Points: None
Trigger: The user wants to generate buses.
Assumptions: The user has completed the use case Log In.
Precondition: The Log In use case has been successfully completed.
Postcondition: Bus Information Report will be created.

Use Case Name #5: Generate Riders
Summary: The user can generate a desired number of riders in the system
Basic Course of Events:

1. Completion of use case Log In
2. The system will ask the user how many riders to be generated
3. The user will enter the number of riders
4. The system will ask the user how fast a set of riders needing transportation moves
5. The user will enter the passenger movements
6. The system will randomize the location of the riders
7. The system shall move the riders to the bus stops based on the movement information given to riders

Alternative Paths: In steps 3 and 5, the user will have an option to cancel entering the information. If the user chose to cancel giving the information, the simulation will be in default. In steps 6, the user will have the option to move the riders to the desired locations.
Exception Paths: None
Extension Points: None
Trigger: The user wants to generate riders
Assumptions: The user has completed the use case Log In.
Precondition: The Log In use case has been successfully completed.

Postcondition: A passenger report will be created.


Generating Logs and Reports
User
BLM
Simulation
Database
opens and logs into the software
wants to open the simulation module
simulation module is openned
default simulation is loaded
loop
[user makes and runs simulations]
opt
[if user wants to configure the simulation]
configures simulation
runs simulation
sends data
creates logs from data
wants to look at logs
opens the logs database
inspects logs
opt
[user wants to make a report]
wants to generate report from log
generates a report
closes the software
User
BLM
Simulation
Database
www.websequencediagrams.com

Sequence diagram for generating logs and reports (modified from Assignment 3)

5.1.4 Generating Logs and Reports

Use Case Name #6: Log Generation

Summary: The system shall logs based on the performance of the simulation

Basic Course of Events:

1. Completion of use case Log In
2. Completion of use case Buses Transporting Riders
3. The system will automatically produce logs based on the reports given by the use case Buses Transporting Riders

Alternative Paths: In step 3, the user shall have the option to stop the simulation while running. If the user chose to pause the simulation, the buses will stop in their current locations. The user then will have an option to continue, restart, or stop the bus simulation.

Exception Paths: None

Extension Points: None

Trigger: None

Assumptions: The user has completed the use case Log In and Buses Transporting Riders

Precondition: The Log In use case has been successfully completed.

Postcondition: Log report will be created.

Use Case Name #7: Generating Reports

Summary: After the simulation completes an accurate report of the simulation is returned by the program for the employee to review.

Basic Course of Events:

1. Simulation terminates.
2. Information from the simulation is gathered.
3. Information is organized into a report.
4. Report is returned for the employee to review.

Alternative Paths: None

Exception Paths: If the employee decides the report is no longer needed the steps will terminate after step 1.

Extension Points: If multiple simulations are needed to be compiled into one report, stop at step 2 and return to 1 and repeat as needed.

Trigger: A simulation starting.

Assumptions: Simulation will return accurate results each time it is called.

Precondition: The simulation is successful.

Postcondition: The program has found what the employee was looking for and returns an accurate summarization on the outcome of the simulation.

6. Non-Functional Requirements

Performance Requirements

Ample amounts of processing power and memory will be needed for all instances of the system. These requirements can vary due to the size of the simulation that is being conducted to efficiently run it. Running the simulation is where much of the allocation of hardware is focused on while the rest is put into the actual running of the software. Simulations are typically run in less than thirty seconds if the computer has the ample resources, otherwise some simulations can take a little longer to run than usual.

The system must state how much memory (SSD preferable than HDD, although HDD is preferred to store data because it is cheaper), RAM, and CPU power are needed to run the simulation. This will ensure that the system will run at its best performance since it will use the recommended or best resources available. A fast and reliable network or internet (Wi-fi 5 and above are preferable) are needed if massive backups are desired.

Safety Requirements

The system must not modify the original data in any way. In addition, the system will be returning accurate results that will not lead to a loss in revenue for the company. If the hardware does not have ample amounts of processing power and memory, a warning message will appear that will allow the user to continue at the risk of possible damage to the hardware. A similar message appears if the user does not have enough space in storage.

The system will notify the users who can access the stored data if a data is being overridden, added, or deleted. This will prevent the modification of data in an unprofessional manner.

Security Requirements

The system must have a login system in order to protect the data. We must also ensure that there is no opportunity for outside groups to access the data being used in the simulations.

The system shall do the following:

- Upon the starting of the program, the user will be prompted to either create an account or login in upon starting up the software.
- The creation of an account will require a company's given email, a username, and a password of 8-15 characters.
    - If the username is forgotten, the system will use the email that was used to create an account to send the right username
    - If the password is forgotten, then an email that will allow a password change is sent. This is possible if the user decides to

either let the system send a reset password link through email immediately or if the user decides to use the 5 attempts allotted for their account.

- If the email is forgotten, contact the administrator immediately.

Software Quality Attributes

This product is created to be user-friendly to users that are not very tech savvy by having a simplified yet modernized, user-friendly interface with help tools that can point confused users in the right direction. It must also be stable and reliable with unexpected inputs resulting in a series or warnings as referred to unexpected behavior or crashing. The design must display error or warning prompts to notify users if such a thing has happened, with a message pointing to the next steps.. The simulations that are created allow for high levels of adaptability and flexibility that can help reliably simulate real-life bus transportation. The software shall perform simulations in at most 30 seconds with an ample amount of resources. Results of the simulation must be tested to provide realistic and accurate data.

Business Rules

Administrators are the only users who can conduct and give the ability to conduct simulations. This is reinforced by requiring a privilege attribute on each account be checked to allow certain functions that the admins only have access to. By default, anyone within their system can look at the data from the reports and logs associated with the software.

The design must provide different access levels. Each user must have an access level given to them by the administrators. Different access levels prevent system compromise, and this procedure leads to accountability and security.

7. Supplementary Documentation

7.1 Glossary

- **Entities**: In this case an existence in the program.
- **Graphical User Interface**: Allows the user to use a point and click method of operating the software
- **Hard Disk Drive (HDD)**: mass storage device that stores data on magnetic platters
- **Human-Readable**: Data that can be naturally read by humans
- **Local Machines**: On the computer the simulation is being run on.
- graphical user interface
- **Logs**: A file that records what occurred within a program or event
- **Natively**: Program will run without any external layer required, lowering complexity.
- **Object Oriented Language**: A programing language that implements objects and their methods in code to create software programs
- **Object Oriented (OO)**: Using a method which enables a system to be modelled after a set of objects.
- **Point and Click**: A user interface style where a user points and clicks with the mouse or other input device in order to initiate a function
- **Process Logic**: cause and effect explanation of a process
- **Processing Power:** Refers to the amount of calculations that can be processed, dependent on hardware used.
- **Raw data**: Primary data collected from a source
- **Solid State Drive (SSD)**: type of mass storage device that stores data by using flash memory