SURTS
Design Document

Authors:
Jackson McEligot
Austin Wessel
David Ibarra
Connor Boehm

Group: A41

This page intentionally left blank.

Document Revision History

Date	Version	Description	Author
11/18/2020	1.0	Initial draft	Jackson McEligot
12/1/2020	1.1	OO Model, Revised attribute/method descriptions, Revised Architecture model	Jackson McEligot et. al.
12/2/2020	2.0	Dynamic Models, Logger, additional revisions	Jackson McEligot et. al.

This page intentionally left blank.

Contents

1 Introduction

This section addresses the purpose of this document including the intended audience, an introduction to the problem and a detailed view of the project's design. In the discussion, the design of the final system including several detailed diagrams will be described in detail.

1.1 Purpose

The Design Document detailed below has been created for the purpose of expressing a possible implementation of the requirements specified in the SRS document for the SURTS [1]. It details the interfaces and classes required to allow the implementation team to be able to create the system.

1.2 System Overview

SURTS is a system that will simulate uniform route-based transportation. Because transportation systems handle many diverse populations in motion, developing transit routes that meet the needs of the users while keeping costs low is often a challenge. This simulation will be used as a testing environment to determine optimal routes, fleet usage, and profitability. This document is intended for developers, project managers, testers, documentation writers, and all developers of this system. It is best to read through each section sequentially.

The software specified in this document is to be used by transportation organization staff, urban planners, statisticians, economists, and system administrators to develop and improve route planning for existing transportation organizations. By controlling different attributes of the simulation like traffic conditions and bus capacity, the users can test different scenarios and make decisions on the physical transit system based on that information. This system hopes to serve the transportation system in providing better insights to their staff who will then make changes to better serve their customers.

1.3 Design Objectives

This document is intended to cover enough of the design of the system so that it is clear to the implementation team how to start working on the implementation of the system. This document is also only intended to address the business layer of the system. Initial versions of this document do not intend to achieve complete design of the system. Instead, it aims to reduce the number of design choices made in the early stages of implementation. This system must be able to handle a simulation with thousands of passengers and hundreds of buses without maxing out CPU usage or filling a large amount of memory. When it comes to simulating variables and creating a model of student demand, the transit system should also be realistic and make the simulation as close as possible to the management of transit on campus as possible.

1.4 References

[1] Wessel, Austin, et al. *Software Requirements Specification for the SURTS*. (Software Requirements Document)

1.5 Definitions, Acronyms, and Abbreviations

- SURTS - System for Uniform Route-based Transportation Simulation
- BLM - Business layer module
- SRS - System Requirements Specification
- UUID - Universally unique identifier

- GUI - Graphical User Interface
- DPL - Data Persistence Layer

Refer to the Document Conventions or Glossary sections of the requirements document for the SURTS for any additional definitions or acronyms.

2 Design Overview

2.1 Introduction

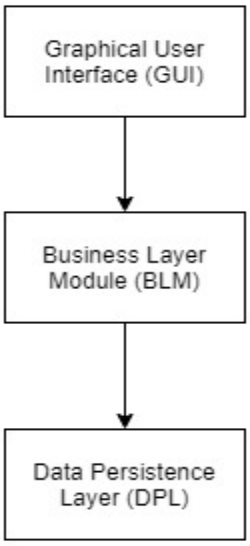
The system has been designed using an object oriented approach and demonstrates the layered architecture of the system. There are three layers to the system, of which we give details for the BLM. The design was created using LucidChart and GoogleDocs.

2.2 Environment Overview

The environment in which the SURTS system will be deployed is intended as a closed software application, not interacted with over a web-based application or internal server. It will reside in a user defined location on their individual machine.

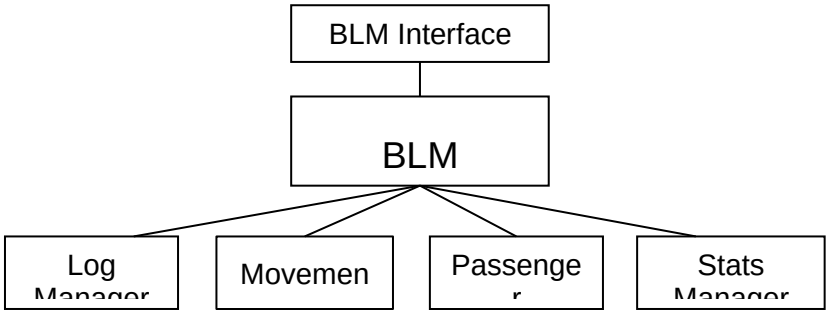
2.3 System Architecture

2.3.1 Top-level system structure of SURTS design



The System is composed of three major components, or layers: the GUI, which takes in requests from the user, and serves as the main source of user input to the system; the BLM, which describes the core logical functionality of the system; and the DPL (or Data Persistence Layer), which persists objects for the BLM. The GUI will take in requests from the user, and then the BLM will take the requests and use them to interact with the primary components of the simulation. The BLM will also be able to use the Persistence Layer to then store classes and information from the BLM.

2.3.2 ***BLM Interface subsystem***



The BLM interface is implemented by a concrete BLM Manager, which in turn utilizes the Passenger Generator, Log manager, and the Movement and Stats Managers.

The Business Layer Module is accessed by the GUI layer through the BLM Interface. This interface serves as the intermediary between the two layers, and the BLM manager is the concrete class for the interface. The BLM manager is then able to utilize requests from the user to create buses with the movement manager and generate a specified frequency of passengers for routes with the passenger generator. The Stats manager is then able to compile stats from the simulation to put into a report, and the log manager is able to generate logs for the simulation.

2.4 Constraints and Assumptions

Particular to the fact that we are designing only part of a layered application, we require that all requests come from the above GUI layer. We also assume that the data requests coming in from the GUI are well-formed and valid requests. We also assume that any authentication issues are handled between direct interaction between the GUI and the DPL, and that we don't need to handle authentication in the business layer. We also assume, for data persistence purposes, we can make an object persistent and retrieve any object from the persistence layer and be able to do so for any object. We are not restricted by languages or technologies at this time.

1. The System will be designed to be run on one machine at a time.
2. The Output of Reports is in a .txt or .csv format.
3. The GUI is assumed to be implemented in accordance with the designated BLM interface.
4. The user will be able to generate passengers based on a specified frequency for each stop. The user is assumed to have an understanding or estimation of the rate passengers will be expected at the stop. This can be changed for each simulation, so once the demand for passengers is specified, the simulation can be used to determine if a set of buses and routes is able to cope with that demand, in terms of bus capacity and passenger wait time.

5. There is only one report per simulation.

   Simulations have been suggested to be able to run several at a time, or less at a time, generating more or less reports. However, based on user-defined passenger generation, this may change, and we chose not to model passenger behavior in a probabilistic manner, which would fit the best for a multiple-simulation approach. Instead, we suppose that each simulation is significant, and that we would like to generate a report automatically after every simulation.

6. Users will want a set of buses that they can modify.

   To be able to create a simulation where one can see the transit system working in action, we decided to allow the user to create the set of buses that will be able to transport simulated passengers. Instead, we could have focused only on simulating a single bus at a time or focused on how the demand is met at one stop. Then, we need to be able to keep track of all of the information on each bus persistently over the course of the simulation. This is beneficial for how this system is designed, because we are primarily catering to the transportation organization, who, most significantly, wants to be able to see how they can possibly optimize their transportation system as a whole.

3 Interfaces and Data Stores

This section describes the interfaces into and out of the system as well as the data stores you will be including in your system.

3.1.1 System Interfaces

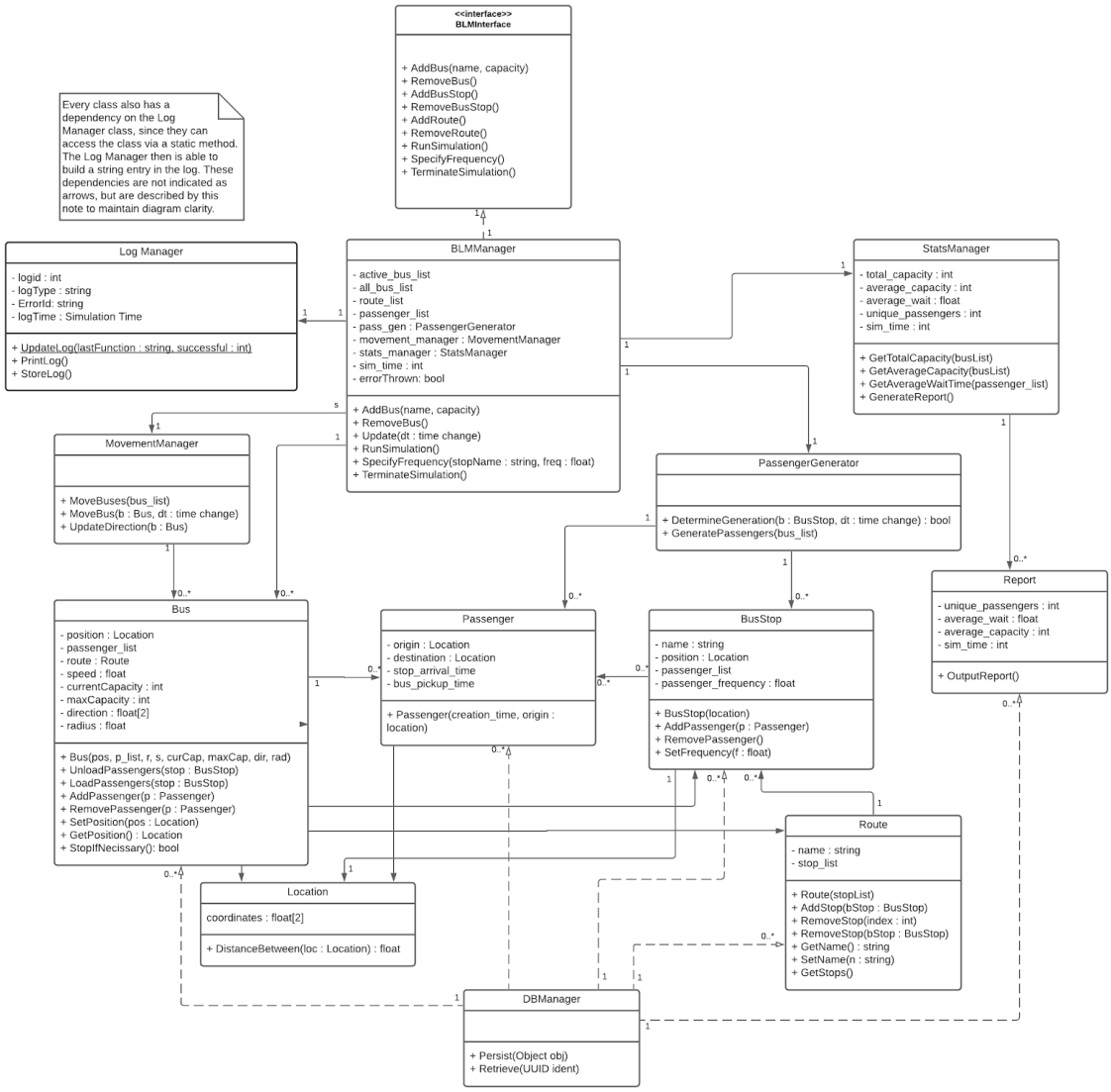
The GUI layer accesses the BLM layer through the BLM interface, as described above. Through this subsystem interface, the GUI layer can send user requests to the subsystem to configure different attributes of the simulation. In this way, the BLM subsystem interface can define passengers, buses, and routes for the simulation from user input. In turn, the BLM subsystem itself has access to the Data Persistence Layer below it via the database manager class.

3.1.2 Data Stores

As part of our system, all the references to stored data are kept inside of the BLM Manager. This includes information on the fleet of buses that is kept inside of the program, the number and attributes of passengers that are generated, and a list of all bus routes kept inside of the system. However, all the information and storage for the buses, passengers, and bus stops are stored inside of the persistent layer instead of in internal data stores.

4 Structural Design

4.1 Class Diagram



4.2 Class Descriptions

4.3 Classes in the BLM Interface Subsystem

4.3.1 Class: BLM Manager

- Purpose: *To act as an entrance point into the system*
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

4.3.1.1 Attribute Descriptions

1. Attribute: *active_bus_list*
   Type: *list of Bus objects*
   Description: *A list of all active buses within the system*

Constraints: *should hold initialized buses only, and be of a length less than or equal to that of all_bus_list*

2. Attribute: *all_bus_list*
   Type: *list of Bus objects*
   Description: *A list of all buses within the system, inactive included*
   Constraints: *should hold initialized buses only, and be of a length from 0-999*

3. Attribute: *route_list*
   Type: *list of Route objects*
   Description: *A list of all Route objects defined within the system*
   Constraints: *Should hold initialized route objects only*

4. Attribute: *passenger_list*
   Type: *list of Passenger objects*
   Description: *A list of all Passenger objects active in the system*
   Constraints: *should hold initialized Passenger objects only*

5. Attribute: *pass_gen*
   Type: *PassengerGenerator*
   Description: *Used to generate passengers at a user-defined frequency*
   Constraints: *None*

6. Attribute: *movement_manager*
   Type: *MovementManager*
   Description: *Manages all bus and passenger movement within the system*
   Constraints: *None*

7. Attribute: *stats_manager*
   Type: *StatsManager*
   Description: *Find the total and average capacity for purposes of generating a report*
   Constraints: *None*

8. Attribute: *sim_time*
   Type: *N/A*
   Description: *Stores current simulation time, increments as simulation progresses*
   Constraints: *Must be non-negative number, real number.*

4.3.1.2 Method Descriptions

1. Method: *AddBus()*
   Return Type: *N/A*
   Parameters: *name, capacity - Name of bus and max capacity of bus*
   Return value: *N/A*
   Pre-condition: *None*
   Post-condition: *New Bus in System*
   Attributes read/used: *active_bus_list, all_bus_list*
   Methods called: *Bus(name, capacity)*

   Processing logic:
   The new bus is created and added to the system using the Bus constructor.

2. Method: *RemoveBus(bus)*
   Return Type: *N/A*
   Parameters: *b of type bus*

Return value: *N/A*
Pre-condition: *a bus is in the bus list*
Post-condition: *bus b has been removed*
Attributes read/used: *active_bus_list, all_bus_list*
Methods called: *None*

Processing logic:
A bus is input and said bus is removed from either the active or all bus lists.

3. Method: *Update*
   Return Type: *N/A*
   Parameters: *dt - the incremental time change since last Update call*
   Return value: *N/A*
   Pre-condition: *None*
   Post-condition: *None*
   Attributes read/used: *sim_time*
   Methods called: *movement_manager.MoveBuses(active_bus_list)*

   Processing logic:
   The Update() method manages the movement of passengers and buses within the system.

4. Method: *RunSimulation()*
   Return Type: *N/A*
   Parameters: *None*
   Return value: *N/A*
   Pre-condition: *Simulation not active, sim_time == 0*
   Post-condition: *sim_time > 0*
   Attributes read/used: *sim_time*
   Methods called: *Update()*

   Processing logic:
   RunSimulation() handles the startup of the simulation.

5. Method: *SpecifyFrequency()*
   Return Type: *N/A*
   Parameters: *stopName - string, freq - float*
   Return value: *N/A*
   Pre-condition: *a BusStop with stopName exists*
   Post-condition: *BusStop with stopName has frequency freq*
   Attributes read/used: *active_bus_list, all_bus_list*
   Methods called: *N/A*

   Processing logic:
   Allows for setting of a passenger generation frequency (how often a passenger is generated at a particular stop) from the main interface.

4.3.2 Class: Movement Manager

- Purpose: *To handle movement of Bus and Passenger Objects within simulation*
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

4.3.2.1 Attribute Descriptions

None

4.3.2.2 Method Descriptions

1. Method: *MoveBuses(bus_list)*
   Return Type: *N/A*
   Parameters: *bus_list - A list of bus objects*
   Return value: *N/A*
   Pre-condition: *None*
   Post-condition: *All buses in bus_list and passengers in each bus object have been moved*
   Attributes read/used: *None*
   Methods called: *MoveBus()*

   Processing logic:
   MovementManager is passed a bus_list and then sets the position of buses and passengers based on the specific bus specific attributes by calling

2. Method: *Movebus(bus_list)*
   Return Type: *N/A*
   Parameters: *bus_list - A list of bus objects*
   Return value: *N/A*
   Pre-condition: *None*
   Post-condition: *Buses and Passengers passed in have moved*
   Attributes read/used: *None*
   Methods called: *Bus.SetPosition(), Passenger.SetPosition()*

   Processing logic:
   The MoveBus method simplifies MoveBuses() method, it is the one changing individual bus positions

3. Method: *UpdateDirection*
   Return Type: *N/A*
   Parameters: *b - bus object*
   Return value: *N/A*
   Pre-condition: *bus b is initialized*
   Post-condition: *b.direction has been set based on bus route*
   Attributes read/used: *b.route*
   Methods called: *b.SetDirection()*

   Processing logic:
   Called each time moveBuses is called to accurately upkeep bus direction

4.3.3 Class: Passenger Generator

- Purpose: *To handle generation of passengers at bus stops at a set frequency*
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

4.3.3.1 Attribute Descriptions

None

4.3.3.2 Method Descriptions

1. Method: *DetermineGeneration(b, dt)*
   Return Type: *bool*
   Parameters: *b - BusStop, dt - time change*
   Return value: *T or F*
   Pre-condition: *b exists and is initialized, dt is a valid positive time*

Post-condition: *None*
Attributes read/used: *b.passenger_frequency*
Methods called: *None*

Processing logic:
Determines if a passenger should be generated, based on time elapsed since last passenger generation

2. Method: *GeneratePassengers(b)*
   Return Type: *N/A*
   Parameters: *b -bus stop list*
   Return value: *N/A*
   Pre-condition: *None*
   Post-condition: *All bus stops have had passengers generated if they should have been*
   Attributes read/used: *None*
   Methods called: *DetermineGeneration()*

   Processing logic:
   Calls DetermineGeneration() for each bus stop in the bus stop list

4.3.4 Class: Stats Manager

- Purpose: *To model the relevant aspects of the physical tank that stores fuel*
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

4.3.4.1 Attribute Descriptions

1. Attribute: *total_capacity*
   Description: *Stores the total capacity of all buses in the system*
   Constraints: *should not be greater than sum of bus capacities*

2. Attribute: *average_capacity*
   Description: *average capacity of all buses throughout simulation*
   Constraints: *not greater than one hundred percent*

3. Attribute: *average_wait*
   Description: *average wait of all passengers at bus stops in system*
   Constraints: *Must be greater than or equal to zero*

4. Attribute: *unique_passenger*
   Description: *Total number of passengers passed through simulation*
   Constraints: *Greater than or equal to zero*

4.3.4.2 Method Descriptions

1. Method: *GetTotalCapacity(busList)*
   Parameters: *bus List*
   Return value: *N/A*
   Pre-condition: *bus list isn't empty*
   Post-condition: *None*
   Attributes read/used: *total_capacity*
   Methods called: *None*

   Processing logic:
   Gets sum of capacities of buses in bus list

2. Method: *GetAverageCapacity(busList)*
   Return Type: *N/A*
   Parameters: *busList - list of bus objects*
   Return value: *N/A*
   Pre-condition: *busList not empty*
   Post-condition: *average_capacity updated*
   Attributes read/used: *average_capacity*
   Methods called: *None*

   Processing logic:
   Gets average of bus capacities in bus list

3. Method: *GetAverageWaitTime(passenger_list)*
   Return Type: *N/A*
   Parameters: *passenger_list - list of passenger objects*
   Return value: *N/A*
   Pre-condition: *passenger_list not empty*
   Post-condition: *average_wait updated*
   Attributes read/used: *average_wait*
   Methods called: *None*

   Processing logic:
   Updates average_wait of all passengers in passenger_list/system

4. Method: *GenerateReport()*
   Return Type: *N/A*
   Parameters: *None*
   Return value: *N/A*
   Pre-condition: *all attributes non-zero*
   Post-condition: *None*
   Attributes read/used: *total_capacity, average_capacity, average_wait, unique_passengers*
   Methods called: *None*

   Processing logic:
   Generates report of attributes as declared.

4.3.5 Class: Report

- Purpose: *To depict system attributes throughout simulation in a readable, useful way*
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

4.3.5.1 Attribute Descriptions

1. Attribute: *average_capacity*
   Description: *average capacity of all buses throughout simulation*
   Constraints: *not greater than one hundred percent*

2. Attribute: *average_wait*
   Description: *average wait of all passengers at bus stops in system*
   Constraints: *Must be greater than or equal to zero*

3. Attribute: *unique_passenger*
   Description: *Total number of passengers passed through simulation*
   Constraints: *Greater than or equal to zero*

4.3.5.2 Method Descriptions

1. Method: *OutputReport()*
   Return Type: *N/A*
   Parameters: *None*
   Return value: *N/A*
   Pre-condition: *attributes non-zero*
   Post-condition: *Report output*
   Attributes read/used: *unique_passengers, average_wait, average_capacity*
   Methods called: *a*

   Processing logic:
   Reads in attributes and outputs them in a readable format.

4.3.6 Class: Bus

- Purpose: *To act as a bus in our simulation, able to hold passengers and move them throughout simulation simultaneously.*
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

4.3.6.1 Attribute Descriptions

1. Attribute: *position*
   Description: *position of bus in system*
   Constraints: *of type Location, Location must be within the defined SURTS coordinate boundaries*

2. Attribute: *passenger_list*
   Description: *list of passengers*
   Constraints: *None*

3. Attribute: *route*
   Description: *route defined in system assigned to said bus*
   Constraints: *Route must exist within system*

4. Attribute: *speed*
   Description: *speed the bus moves through the simulation coordinates at*
   Constraints: *Greater than or equal to zero*

5. Attribute: *currentCapacity*
   Description: *passenger count of all passengers on bus*
   Constraints: *value equal to length of passenger_list*

6. Attribute: *maxCapacity*
   Description: *maximum number of passengers bus can hold*
   Constraints: *Greater than or equal to zero*

7. Attribute: *direction*
   Description: *unit vector of bus direction*
   Constraints: *Real numbers*

8. Attribute: *radius*
   Description: *radius used to determine proximity of bus to a bus stop*
   Constraints: *Must be non-negative, real number.*

4.3.6.2 Method Descriptions

1. Method: *UnloadPassengers(stop)*
   Return Type: *N/A*
   Parameters: *BusStop*
   Return value: *N/A*
   Pre-condition: *BusStop parameter must be a valid bus stop in the current simulation.*
   Post-condition: *Calls RemovePassenger(p, BusStop) for each passenger in this.passenger_list. When RemovePassengers(p, BusStop) returns true, decrement currentCapacity and remove p from this.passenger.list. When false, do nothing.*
   Attributes read/used: *passenger_list, currentCapacity*
   Methods called: *RemovePassenger(p,list)*

   Processing Logic:
   Serves to remove passengers from a bus when it arrives at their destination stop.

2. Method: *LoadPassengers(stop)*
   Return Type: *N/A*
   Parameters: *BusStop*
   Return value: *N/A*
   Pre-condition: *BusStop parameter must be a valid bus stop in the current simulation and currentCapacity must be less than maxCapacity.*
   Post-condition: *Calls AddPassenger(p, this.route) for each passenger in BusStop.passenger_list. When AddPassenger(p) returns true increment currentCapacity, add p to this.passenger_list, remove p from BusStop.passengerlist.*
   Attributes read/used: *passenger_list, currentCapacity, maxCapacity, route*
   Methods called: *AddPassenger(p,list)*

   Processing Logic:
   Serves to pick-up passengers from their stops and update both the passenger lists of both the stop and bus.

3. Method: *AddPassenger(p,list)*
   Return Type: *bool*
   Parameters: *Passenger, Bus.route*
   Return value: *true || false*
   Pre-condition: *N/A*
   Post-condition: *Returns true if p.destination is in list. Returns false otherwise.*
   Attributes read/used: *passenger_list, route*
   Methods called: *None*

   Processing Logic:
   Verifies whether or not a passenger should board a bus.

4. Method: *RemovePassenger(p,stop)*
   Return Type: *bool*
   Parameters: *Passenger, BusStop*
   Return value: *true || false*
   Pre-condition: *N/A*
   Post-condition: *Returns true if p.destination is equal to stop. Returns false otherwise.*
   Attributes read/used: *passenger*
   Methods called: *None*

   Processing Logic:
   Verifies whether or not a passenger should disembark a bus.

5. Method: *SetPosition(pos)*
   Return Type: *N/A*
   Parameters: *Position*
   Return value: *N/A*
   Pre-condition: *N/A*
   Post-condition: *The bus calling SetPosition has its position attribute updated to pos.*
   Attributes read/used: *position*
   Methods called: *None*

   Processing Logic:
   Relocates a bus to pos.

6. Method: *GetPosition()*
   Return Type: *Location*
   Parameters: *N/A*
   Return value: *N/A*
   Pre-condition: *N/A*
   Post-condition: *Return the position of the bus calling GetPosition()*
   Methods called: *None*

   Processing Logic:
   Returns a bus's position.

7. Method: *StopIfNecessary()*
   Return Type: *Bool*
   Parameters: *N/A*
   Return value: *N/A*
   Pre-condition: *Bus has valid route,*
   Post-condition: *If within stopping distance of a stop on its route and add/removed passengers*
   Methods called: *Location.DistanceBetween()*

   Processing Logic:
   Bus stopped it within stopping distance of a stop on its route and add/removed passengers

8. Method: *GetRadius()*
   Return Type: *float*
   Parameters: *None*
   Return value: *radius*
   Pre-condition: *None*
   Post-condition: *N/A*
   Attributes read/used: *radius*
   Methods called: *None*

   Processing logic:
   The radius of the bus is read from the radius attribute and returned.

9. Method: *SetRadius(newRadius)*
   Return Type: *float*
   Parameters: *newRadius - the new radius that the radius attribute will be set to.*
   Return value: *N/A*
   Pre-condition: *None*
   Post-condition: *N/A*
   Attributes read/used: *radius*
   Methods called: *None*

Processing logic:
The radius attribute is set to the value of newRadius.

10. Method: *Bus (pos, p_list, r, s, curCap, maxCap, dir, rad)*
    Return Type: *Bus*
    Parameters:
    - *pos - the initial location of the bus*
    - *p_list - the initial list of passengers on the bus*
    - *r - the route of the bus*
    - *s - the speed of the bus*
    - *curCap - the current capacity of the bus, length of p_list*
    - *maxCap - the maximum number of passengers the bus can hold*
    - *rad - the pickup radius of the bus, determines proximity to bus stops*

    Return value: *N/A*
    Pre-condition: *None*
    Post-condition: *N/A*
    Attributes read/used: *position, passenger_list, route, speed, currentCapacity, maxCapacity, direction, radius*
    Methods called: *None*

    Processing logic:
    A new bus is created and returned based on the passed in parameters.

4.3.7 Class: Passenger

- Purpose: *Act as a passenger in the simulation*
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

4.3.7.1 Attribute Descriptions

1. Attribute: *origin*
   Description: *a bus stop the passenger arrives at*
   Constraints: *Must be an existing bus stop within the system*

2. Attribute: *destination*
   Description: *a bus stop the passenger intends to depart at*
   Constraints: *Must be a bus stop that has an active bus route connected to the originating bus stop*

3. Attribute: *stop_arrival_time*
   Description: *Simulation time that the passenger arrived at their origin stop*
   Constraints: *Must be a positive, real number less than or equal to the active simulation time*

4. Attribute: *bus_pickup_time*
   Description: *Simulation time that the passenger boarded a bus*
   Constraints: *Must be a positive, real number less than or equal to the active simulation time*

4.3.7.2 Method Descriptions

1. Method: *Passenger(creationTime, originStop)*
   Return Type: *Passenger*
   Parameters: *creationTime, originStop - Simulation time of passenger creation; bus stop the passenger first arrives at*

Return value: *N/A*
Pre-condition: *None*
Post-condition: *None*
Attributes read/used: *None*
Methods called: *None*

Processing logic:
A new passenger object is created and returned.

4.3.8 Class: Location

- Purpose: *To store a point in the simulation coordinates*
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

4.3.8.1 Attribute Descriptions

1. Attribute: *coordinates - float[2]*
   Description: *Point within the system depicting a bus stop*
   Constraints: *Point must exist within the system and each coordinate must be a real number*

4.3.8.2 Method Descriptions

1. Method: *DistanceBetween(secondLocation)*
   Return Type: *float*
   Parameters: *secondLocation - A location object to compare the distance between*
   Return value: *N/A*
   Pre-condition: *both local coordinates and secondLocation coordinates exist*
   Post-condition: *None*
   Attributes read/used: *coordinates*
   Methods called: *secondLocation.GetCoordinates()*

   Processing logic:
   The distance between two locations is determined through the formula derived through Pythagorean's Theorem.

2. Method: *getCoords()*
   Return Type: *tuple*
   Parameters: *N/A*
   Return value: *(this.coordinates[0],this.coordinates[1])*
   Pre-condition: *N/A*
   Post-condition: *N/A*
   Attributes read/used: *coordinates*
   Methods called: *None*

   Processing logic:
   Returns a tuple containing the calling locations coordinates.

3. Method: *setCoords(x,y)*
   Return Type: *N/A*
   Parameters: *floats containing the x and y coordinates*
   Return value: *N/A*
   Pre-condition: *x and y must be within allowed range of simulation.*
   Post-condition: *this.coordinates[0] = x. this.coordinates[1] = y.*
   Attributes read/used: *coordinates*

Methods called: *None*

Processing logic:
Sets the coordinates of a location.

4.3.9 Class: Bus Stop

- Purpose: *Act as a pickup/dropoff location of passengers by buses*
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

4.3.9.1 Attribute Descriptions

1. Attribute: *name*
   Description: *Name associated with this bus stop*
   Constraints: *Must be a unique name within the system and be of type string*

2. Attribute: *position*
   Description: *position of bus stop in system, is of type Location*
   Constraints: *Must be of type Location, and be within the defined SURTS coordinate system*

3. Attribute: *passenger_list*
   Description: *List of passengers currently waiting at this bus stop*
   Constraints: *Cannot contain passengers that have not already been generated; all passengers in list must be known to the system*

4. Attribute: *passenger_frequency*
   Description: *Frequency of passenger arrival at bus stop during simulation run*

4.3.9.2 Method Descriptions

1. Method: *SetPosition(pos)*
   Return Type: *N/A*
   Parameters: *N/A*
   Return value: *N/A*
   Pre-condition: *pos is of type location.*
   Post-condition: *BusRoute.position is set to pos.*
   Methods called: *None*

   Processing Logic:
   Sets the position of the calling bus to pos.

2. Method: *AddPassenger(p)*
   Return Type: *N/A*
   Parameters: *Passenger*
   Return value: *N/A*
   Pre-condition: *p.destination is not equal to this.position.*
   Post-condition: *Add passenger to this.passenger_list.*
   Attributes read/used: *passenger_list*
   Methods called: *None*

   Processing Logic:
   Adds a passenger at a stop to the stop's passenger list.

3. Method: *RemovePassenger(p)*
   Return Type: *N/A*

Parameters: *Passenger*
Return value: *N/A*
Pre-condition: *p is in this.passenger_list*
Post-condition: *Remove p from this.passenger_list*
Attributes read/used: *passenger_list*
Methods called: *None*

Processing Logic:
Removes a passenger from a stop's passenger list.

4. Method: *SetFrequency(rate)*
Return Type: *N/A*
Parameters: *float*
Return value: *N/A*
Pre-condition: *N/A*
Post-condition: *Sets the calling stop's passenger_frequency to rate.*
Attributes read/used: *passenger_frequency*
Methods called: *None*

Processing Logic:
Updates a stop's passenger_frequency to rate.

4.3.10 Class: Route

- Purpose: *To store the designated bus stop traversal of one or more buses within the simulation*
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

4.3.10.1 Attribute Descriptions

1. Attribute: *name*
Description: *Name associated with this route*
Constraints: *Must be a unique route name within the system and only contain alphanumeric characters*

2. Attribute: *stop_list*
Description: *List of bus stops existing along this bus route.*
Constraints: *All stops in the list must exist within the system. No stop may be connected to itself. List must be longer than one element.*

4.3.10.2 Method Descriptions

1. Method: *Route(stopList)*
Return Type: *Route*
Parameters: *stopList - List of bus stops along the route*
Return value: *N/A*
Pre-condition: *All bus stops exist within the system*
Post-condition: *New route within the system*
Attributes read/used: *stop_list*
Methods called: *None*

Processing logic:
The new route is created and added to the system.

2. Method: *AddStop(bStop, routeIndex)*
Return Type: *bool*

Parameters: *bStop, routeIndex - Bus stop to be added; position in route to add the stop at*
Return value: *true/false*
Pre-condition: *bStop exists within the system*
Post-condition: *New bus stop within route*
Attributes read/used: *stop_list*
Methods called: *None*

Processing logic:
The specified stop is added to the route at the specified index, pushing existing stop at said index back by one spot.

3. Method: *RemoveStop(stop)*
   Return Type: *bool*
   Parameters: *stop - Either Name of bus stop, or position of bus stop within route to be removed*
   Return value: *true/false*
   Pre-condition: *stop exists within the system and within the route*
   Post-condition: *route is one stop shorter*
   Attributes read/used: *stop_list*
   Methods called: *None*

   Processing logic:
   The method removes the specified stop either by comparing the name of each stop, or by removing a stop at a specified index within the route's stop_list. The return bool is whether the provided stop was found within the stop_list and removed.

4. Method: *GetName()*
   Return Type: *string*
   Parameters: *None*
   Return value: *N/A*
   Pre-condition: *Name of route has been specified*
   Post-condition: *N/A*
   Attributes read/used: *name*
   Methods called: *None*

   Processing logic:
   The name of the route is retrieved from the name attribute and returned.

5. Method: *SetName(newName)*
   Return Type: *void*
   Parameters: *newName - The new name that the route name attribute will be set to*
   Return value: *N/A*
   Pre-condition: *None*
   Post-condition: *the route name attribute has changed to newName*
   Attributes read/used: *name*
   Methods called: *None*

   Processing logic:
   The name attribute is set equal to the newName parameter.

6. Method: *GetStops()*
   Return Type: *list of bus stops*
   Parameters: *None*
   Return value: *N/A*
   Pre-condition: *None*
   Post-condition: *N/A*

Attributes read/used: *stop_list*
Methods called: *None*

Processing logic:
The list of stops associated with the route is retrieved from stop_list and returned.

4.3.11 Class: Log Manager

- Purpose: *To track and output status of the system during simulation run*
- Constraints: *None*
- Persistent: *Yes*

4.3.11.1 Attribute Descriptions

1. Attribute: *logId*
   Description: *unique identifier for this log*
   Constraints: *Must be unique within system*

2. Attribute: *logType*
   Description: *Indicates type of log. Error, warning, etc.*
   Constraints: *Must be a string.*

3. Attribute: *errorId*
   Description: *Identifies the type of error for lookup purposes.*
   Constraints: *Must be an already defined error id*.

4. Attribute: *logTime*
   Description: *Indicates at what time in the simulation the log was created*
   Constraints: *must be greater than 0.*

4.3.11.2 Method Descriptions

1. Method: *UpdateLog(lastFunction, successful)*
   Return Type: *None*
   Parameters: *lastFunction, successful — string indicating last function call; boolean indicating success or failure of said function call*
   Return value: *N/A*
   Pre-condition: *Simulation running, last function known*
   Post-condition: *LogManager attributes changed*
   Attributes read/used: *logId, logType, errorId, logTime*
   Methods called: *PrintLog(), StoreLog()*

   Processing logic:
   Updates information stored in LogManager, and prints/stores the attributes if the provided function call was unsuccessful.

2. Method: *PrintLog()*
   Return Type: *None*
   Parameters: *None*
   Return value: *N/A*
   Pre-condition: *Terminal/Console available for writing*
   Post-condition: *I/O stream written to.*
   Attributes read/used: *logId, logType, errorId, logTime*
   Methods called: *None*

   Processing logic:

Writes information stored in attributes to the console/terminal

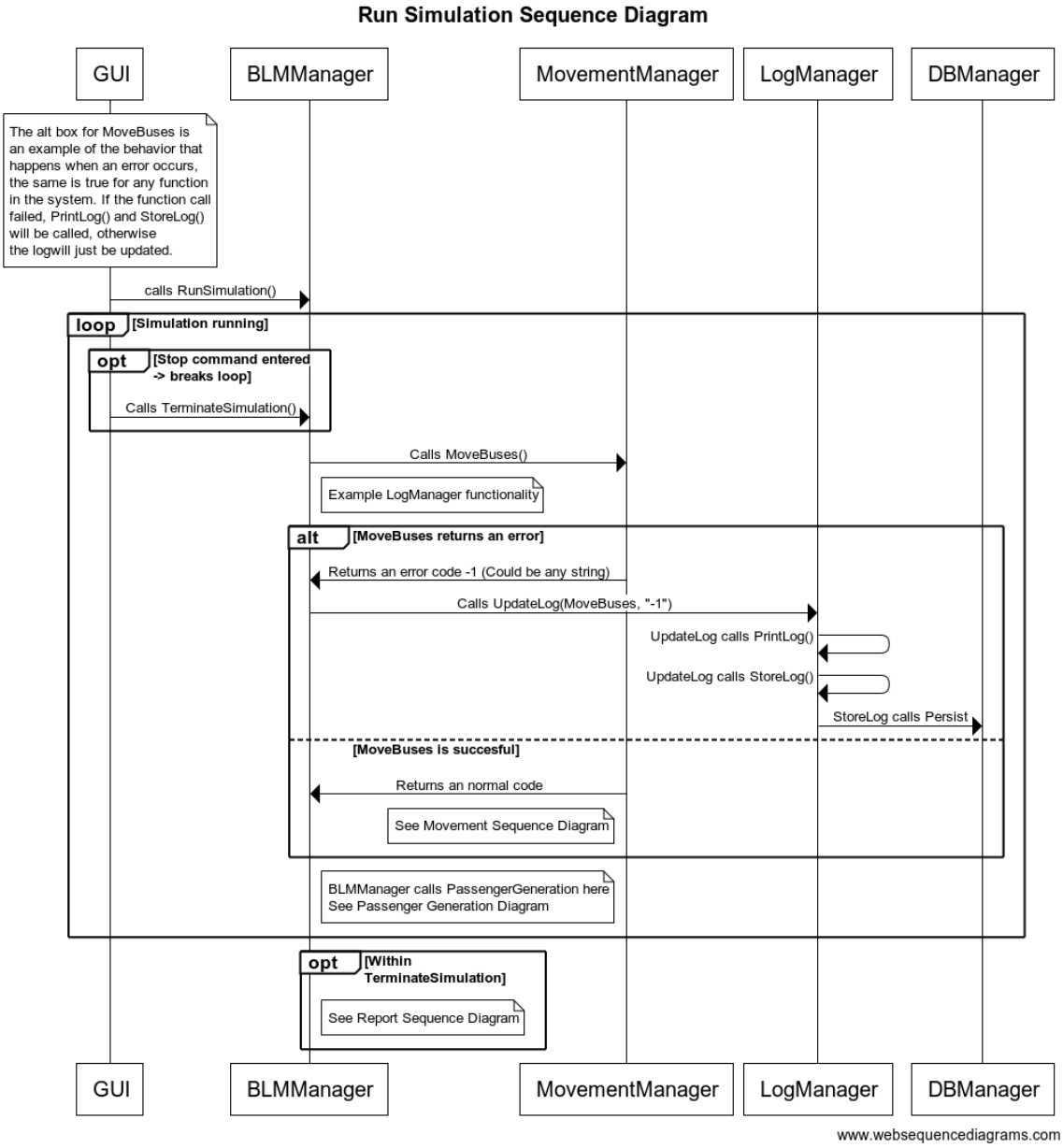
3. Method: *StoreLog()*
   Return Type: *None*
   Parameters: *None*
   Return value: *N/A*
   Pre-condition: *None*
   Post-condition: *N/A*
   Attributes read/used: *stop_list*
   Methods called: *None*

   Processing logic:
   The information stored in the attributes is written to an external location for later access.

5 Dynamic Models

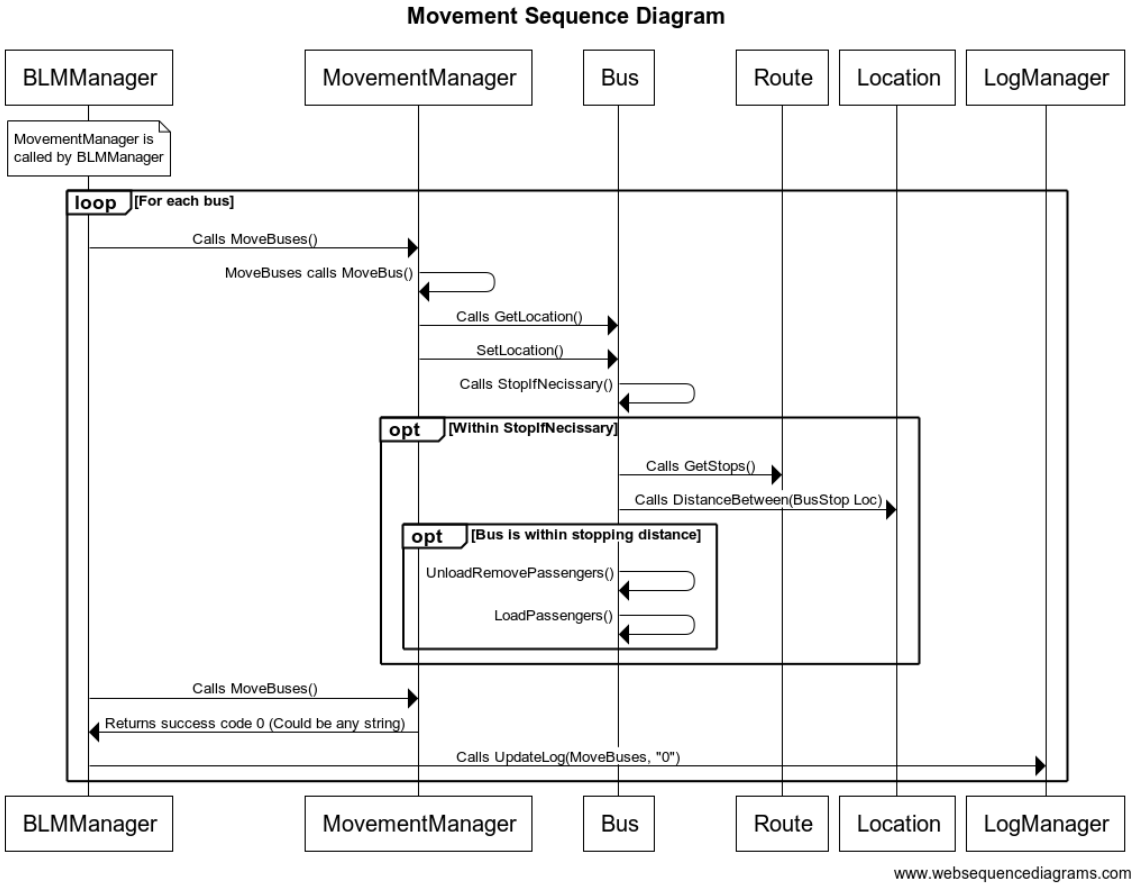
5.1 Scenarios

5.1.1 Run Simulation Sequence Diagram



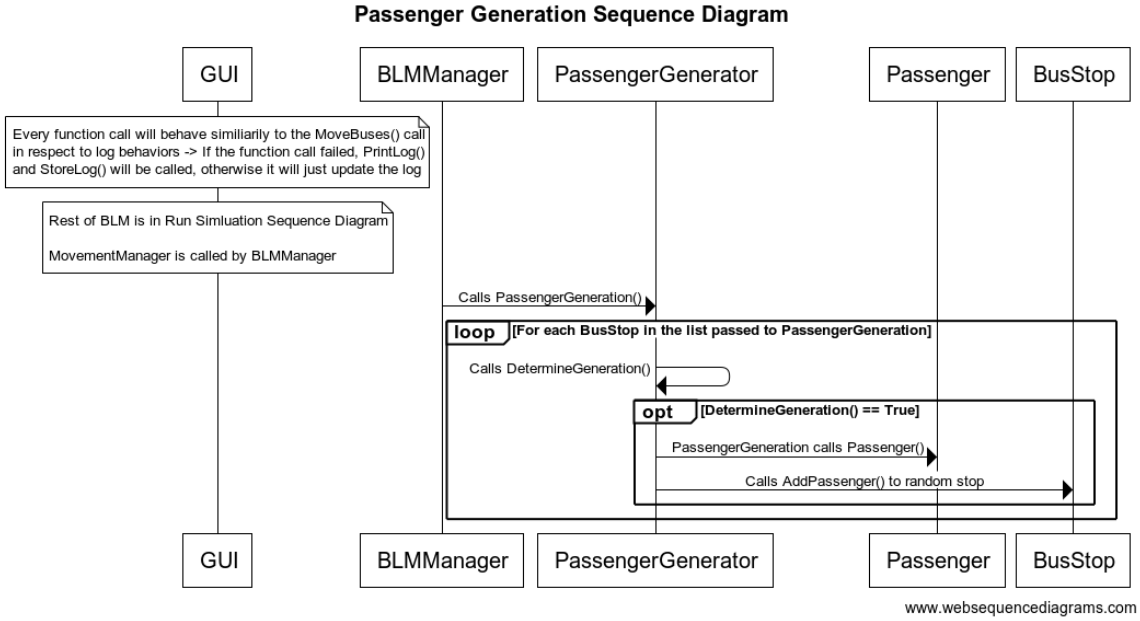
Shows a sequence of calls for the simulation of buses and passengers in the program, and the logging/reporting of the data collected. For this purpose, the GUI is included to demonstrate the relevant interaction with the BLM layer. The sequences for Movement and Reports are partitioned into the following two sequence diagrams (5.1.1.2, 5.1.1.2, 5.1.1.3). Covers Use Cases 1 & 2: Simulate Buses and Routes; Produce Logs and Generate Reports. (Wessel et al, SURTS SRS).

5.1.1.1 **Movement Sequence Diagram**



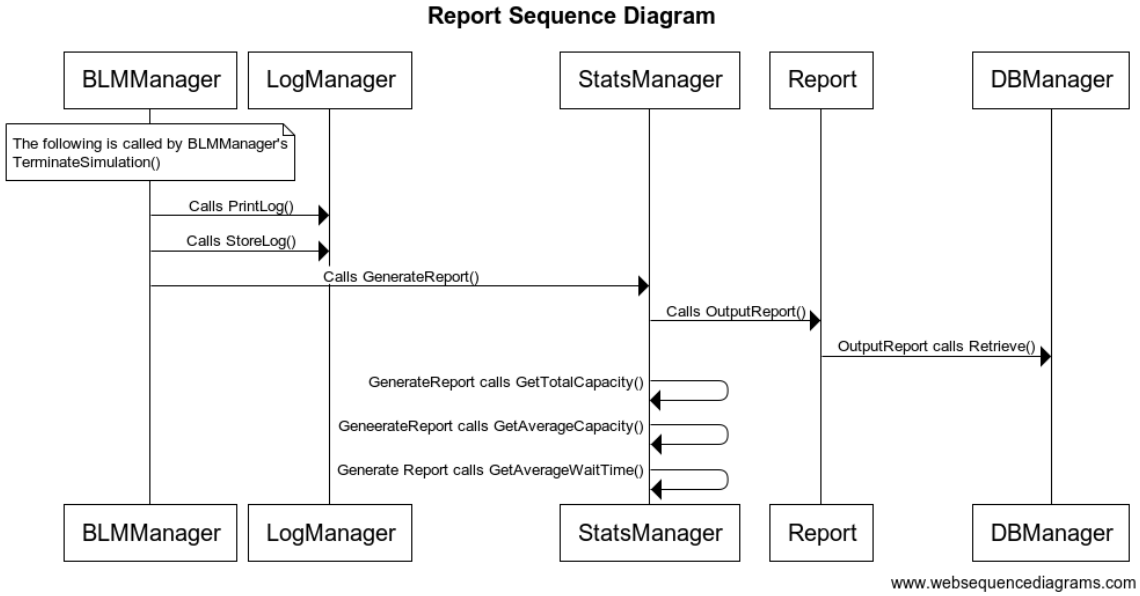
Sub-diagram that shows the sequence of calls that primarily affect the movement of objects in the simulation.

5.1.1.2 Passenger Generation Sequence Diagram



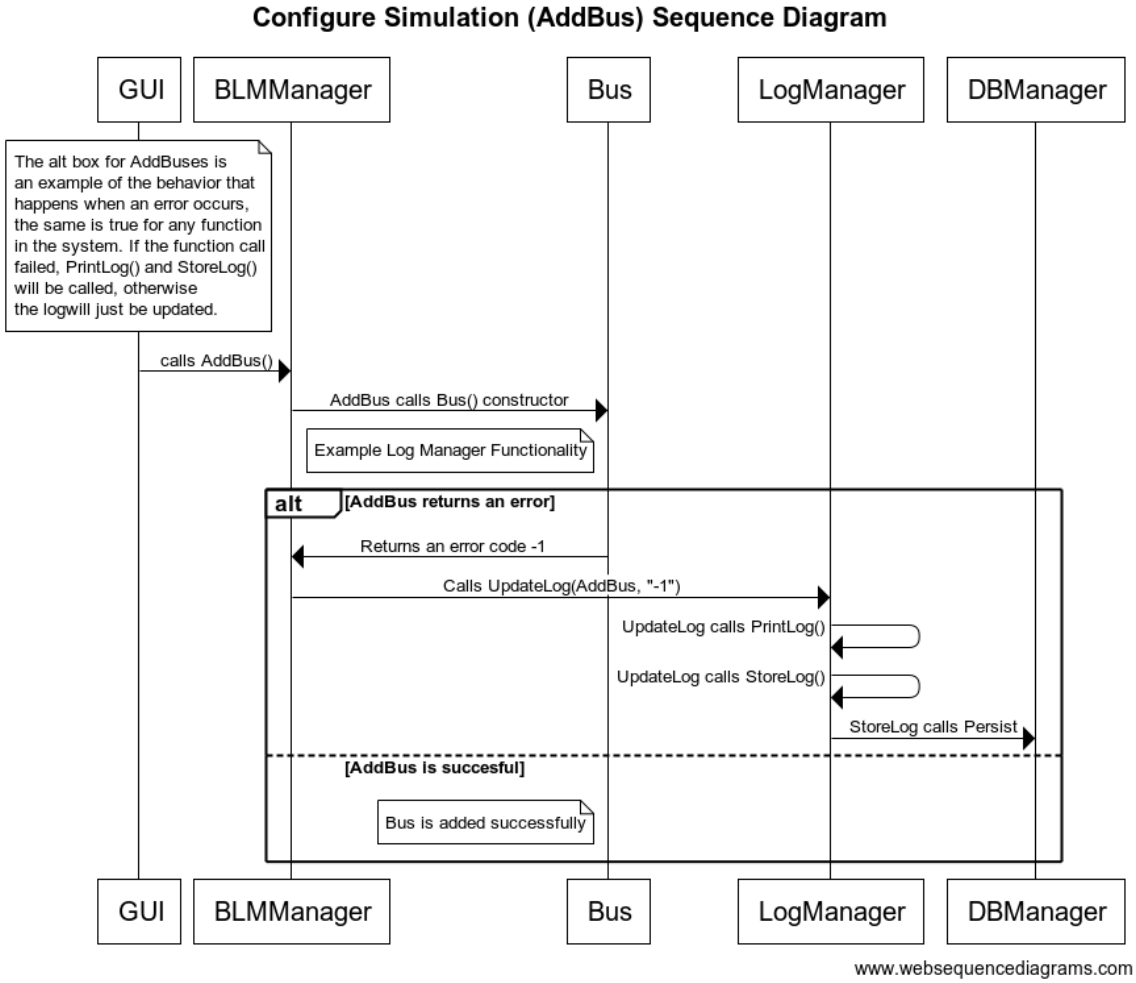
Sub-diagram that shows the sequence of calls used to generate the passengers.

5.1.1.3 Report Sequence Diagram



Sub-diagram that shows the sequence of calls that are used to make a report using logs. Once the logs are generated, administrators can also view the logs outside of the system allowing them to complete the task of using logs to debug the system.

5.1.2 Configure Simulation (AddBus) Sequence Diagram



Shows the sequence of calls for configuring a setting, in this case adding a bus. The error code -1 is an arbitrary code used to show the application of the error log reporting. This sequence shows the behavior of modifying any mutable setting and can be applied to any other setting such as removing a bus stop. Covers Use Case 3: Configure Simulation Parameters. (Wessel et al, SURTS SRS). Also allows administrators to perform the task of manipulating simulation parameters.

6 Non-functional requirements

The system must be capable of handling a simulation with thousands of passengers and hundreds of buses without maxing out CPU usage or filling up a large amount of memory or storage space. Thus, when we store large numbers of buses, passengers, and stops, we will store the information in the Data Persistence Layer, which we assume to handle memory as such. There may also be security concerns with the acquisition and storage of data provided by individual transportation service organizations, so we cover this aspect of the design by limiting the scope of the system, and only outputting a limited amount of information in the report when providing a summary of simulations that were run. We keep in mind that information like the passenger frequency, or the list of buses, which may be private to the organization. In the future, the design can change to accommodate whatever information is needed in the report, so it is given to users who are allowed access to the system.

7 Supplementary Documentation

No additional documentation is provided at this time.