**Software Design Document [IEEE 1016]**

**ECE 1140 - Group: Tovarish**

Jonah Belback, Lucas Connell, Devin James, Noah Lichstein, Elizabeth Novikova, Sam Pratley, Daniel Richardson

****

**Change Log**:

| **Change Date** | **Person** | **Affected Sections** | **Version** |
| --- | --- | --- | --- |
| 03/07/24 | Noah Lichstein | Document Creation | *1.0.0* |

**Table of Contents**

1. **Overview** 
   1. Scope
   2. Purpose
   3. Conformance
2. **Definitions**
3. **System Use Case Diagram**
4. **Key System Architecture and Design**
   1. Authority
   2. Vitality
   3. Route
   4. Wayside Distribution
   5. Timing
5. **Software Architecture**
   1. Software Architecture Diagram
6. **Module Design**
   1. **CTC**
      1. Use Case Diagram
      2. CTC Use Cases
      3. Software Class Diagram
   2. **Track Controller SW**
      1. Use case diagram
      2. Track Controller Uses Cases
      3. Software Class Diagram
   3. **Track Controller HW**
      1. Use Case Diagram
      2. Track Controller Use Cases
      3. Software Class Diagram
   4. **Track Model**
      1. Use case diagram
      2. Track Model Use Cases
      3. Software Class Diagram
   5. **Train Model**
      1. Use case diagram
      2. Train Model Use Cases
      3. Software Class Diagram
   6. **Train Controller SW**
      1. Use case diagram
      2. Train Controller Use Cases
      3. Software Class Diagram
   7. **Train Controller HW**
      1. Use case diagram
      2. Train Controller Use Cases
      3. Software Class Diagram
7. **System Deployment**
   1. Deployment Diagram
8. **Overview** 
   1. **Purpose**

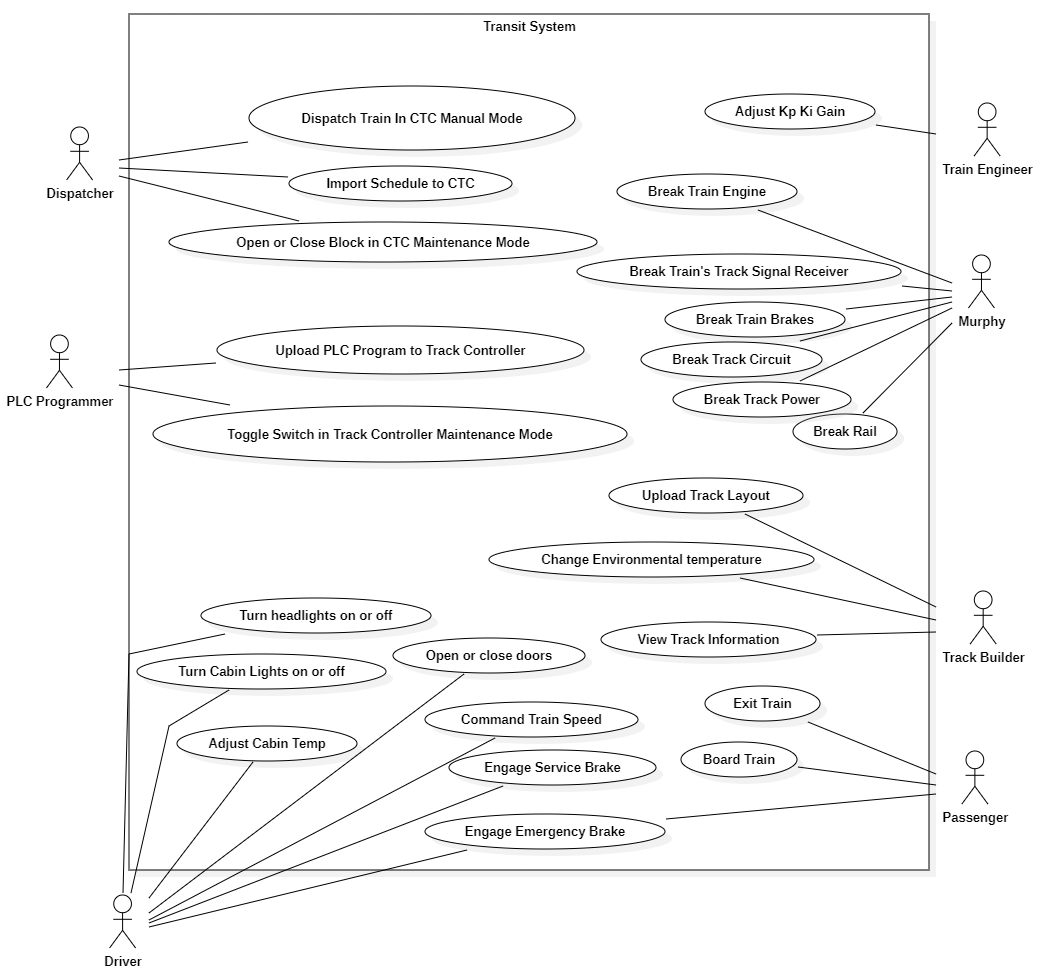
The purpose of this document is to provide an in-depth development outline of a train control system; including the structure of each module and the integration of each module as they relate to the system. The intended readers of this document are software developers as this document will provide an intimate understanding of the system to the point of reproducibility should it ever become necessary. The core design of this system is centered around safety and reliability. This system will employ standard software development practices such as rigorous testing and redundancy in vital safety architecture to ensure any possibility of system failure is as close to nil as possible.

* 1. **Scope**

This document outlines the implementation of a fully functioning train control system. This includes a CTC office, track controllers, a track model, train models, and a train controller. These modules will be integrated with one another to form an adaptable and safe train transit system. The scope of this project is rigidly defined and will not include any of the following functionality: a control system for the yard, the ability to request repairs or maintenance of the train, track, or any associated systems, the ticket sales system, any railway station system, or managing a roster of current employees or passengers.

* 1. **Conformance with IEEE Standards**
     1. IEEE/ISO/IEC 15026-3-2023: System Integrity Levels
     2. IEEE/ISO/IEC 24748-6-2023: Systems and Software Integration
     3. IEEE 1228-1994: Software Safety Plans
     4. IEEE 1016-1987: Recommended Practice for Software Design Descriptions
     5. IEEE/ISO/IEC 24765-2010: Vocabulary
     6. IEEE 1633-2008: Recommended Practice on Software Reliability

1. **Definitions** 
   1. **Vital**: Synonym of safety critical used primarily in locomotive applications
   2. **Red/Green Line**: The two lines of travel for this system. The two lines do not interact with each other except by sharing the same yard.
   3. **Block**: Sections of rail that are electrically isolated from each other for logistics purposes, any given rail line is composed of dozens such blocks and to preserve vitality no two trains can be in the same block. Blocks are typically denoted by a letter.
   4. **Authority**: The number of blocks a train shall move before it has to stop
   5. **Vital Speed**: The maximum safe speed a train shall move at through a given block
   6. Railroad Crossing: the intersection of a railway and a roadway
   7. **Signal Lights**: Signal lights exist to the side of tracks and primarily serve to indicate safety to the driver of a train.
   8. **Switch**: Switches are the boundary between three blocks such that there is one block A such that a train can get to blocks B and C from block A but a train can only go to block A from blocks B and C. switches have a state which determines whether or not a train goes to block B or C when leaving block A and must be set to the correct state when a train attempts to enter block A from blocks B or C.
   9. **Crew**: People who are responsible for the operation of a given train
   10. **Passengers**: Riders of the train
   11. **Station**: Certain blocks have a station on it, stations are where passengers board or depart from a train
   12. **Yard**: The yard is the area from which trains enter the track system, there is one yard that is connected to every line in the rail system.
   13. **Dispatcher**: Dispatchers work in the CTC office and use the CTC to administer schedules and commands to individual trains and switches for the purpose of administering convenient public transportation to the general public.
   14. **CTC**: The short name used in this document to refer to the software that the dispatcher uses to administer schedules. The CTC shall be responsible for a number of duties such as routing and ensuring that trains do not collide with other trains. The CTC shall NOT be a vital module. The CTC shall communicate with the track controllers and no other module
   15. **PLC code**: a set of binary instruction that act on certain inputs to to the module and produce certain outputs from the module, it consists of simple if/else, and, or, xor, and other binary operation
   16. **PLC Programmer**: The PLC programmer is responsible for writing the PLC code for the track controller, the PLC code will dictate whether or not it is safe for a train to continue doing what it's doing and will affect the signal sent to the train to maintain vitality. The PLC also describes the behavior of switches, railroad crossing and signal lights.
   17. **Track/Wayside Controller**: The track controller is the physical device that shall implement the PLC code written by the PLC programmer. The Track controller shall be a VITAL module, it shall signal to the train that its current course is not safe. The track controller shall communicate with the CTC and track model and no other module
   18. **Murphy**: Murphy is responsible for failures in both the track model and train model.
   19. **Track Model**: The track model shall be a piece of software for testing purposes before implementing the CTC, track controller, and train controller are implemented in the field. The track model shall be a full description of the entire track system including all its lines, all its blocks, and any and all extra factors associated with those blocks. The track model shall be acted on by the track controller and train model and shall send signals to both those modules according to those actions. This module shall have three simulated failures that affect each block independently: track circuit failure, broken rail, and loss of power. This module shall NOT be vital.
   20. **Train Model**: The train model shall be a piece of software for testing purposes before implementing the CTC, track controller, and train controller are implemented in the field. The train model shall be a complete description of the train that will be implemented in the field and moves around the simulated track in the track model to ensure the other modules are doing as they shall. The train model shall be acted upon by the train controller to implement a number of functions a train model shall do such as: moving, operating the doors and lights, heating the cabin, etc. the train model shall have three simulated failures: engine fail, signal pickup fail, brake fail. This module shall NOT be vital, it is assumed that irregardless of failure state the train controller will maintain vitality
   21. **Driver**: The crew member who responsible for driving the train and uses almost all of the functions of the train controller
   22. **Train Engineer**: The crew member who defines the PID parameters for the train controller to deliver the correct power to the train engine to ensure a stable system.
   23. **Train Controller**: The train controller shall be a piece of software or hardware that controls the train in the majority of its functions. The train controller shall act on certain stimuli to maintain vitality and should maintain the schedule from the CTC and comfort to the passengers when those aren’t against maintaining vitality. The train controller shall communicate with the train model and no other module. The train controller shall be a VITAL module.
   24. **Safe Stopping Distance (SSD)**: The safe distance (in blocks) in which a train can come to a complete stop using only the service brake, assuming the train is traveling through the minimum block size, at maximum speed, descending down the maximum possible descent grade.
   25. **Mesh points**: Where two sections of adjacent track are controlled by different track controllers, it is considered to be a mesh point.
2. **System Use Case Diagram**

****

**Figure 3.1:** System Use Case Diagram

1. **Key System Architecture and Design:**
   1. **Speed and Authority**
      1. **Authority** (unit: blocks) tells a train when to stop at a scheduled stop. In this Train Control System, authority is **not** vital (see 4.1.2). Authority is distributed to each train for a SSD block window at a time. This is to ensure that there are no authority messaging collisions/conflicts between trains, as the Track Controller maintains a SSD block window between each train. Once a train leaves a block, the CTC will send authority of 0 to that block.

At Time of Dispatch (Train is in Block 0 - Yard), the CTC will provide the following authorities for a train, assuming an SSD of 4:

| Train |  |  |  |  |
| --- | --- | --- | --- | --- |
| # Blocks to stop | # Blocks to stop - 1 | # Blocks to stop - 2 | # Blocks to stop - 3 | # Blocks to stop - 4 |
| Block 0 | Block 1 | Block 2 | Block 3 | Block 4 |

**Figure 4.1.1:** Demonstration of Authority

When the train enters Block 1, the CTC will revoke authority from Block 0 (Yard), and give authority to Block 5.

|  | Train |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | # Blocks to stop | # Blocks to stop - 1 | # Blocks to stop - 2 | # Blocks to stop - 3 | # Blocks to stop - 4 |
| Block 0 | Block 1 | Block 2 | Block 3 | Block 4 | Block 5 |

**Figure 4.1.1:** Demonstration of Authority

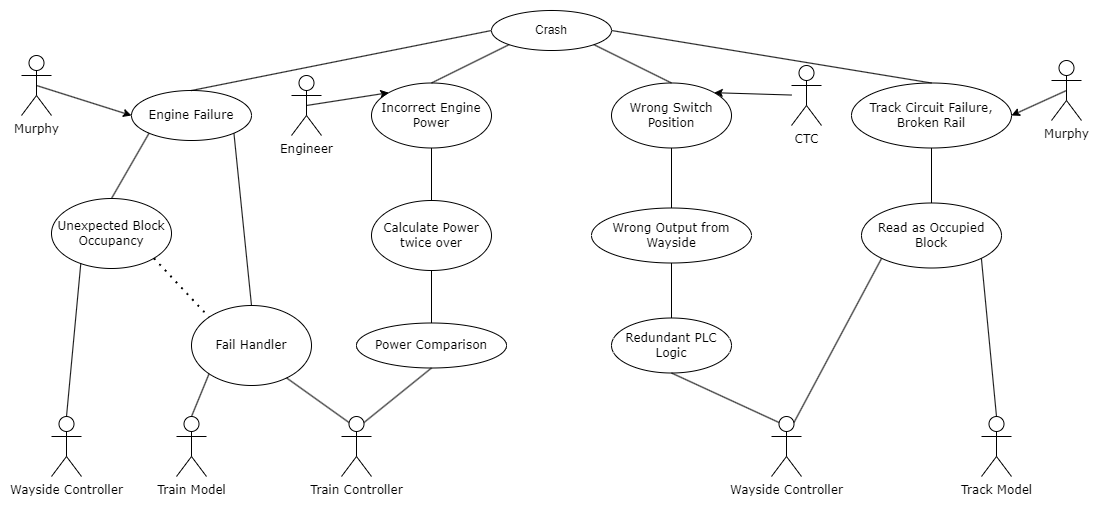
When the train is in any block X, and the station is within SSD blocks, no further authority will be suggested beyond the stop block until the train advances to the next stop.

| Train |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 2 | 1 | 0 | 0 | 0 |
| Block X | Block X + 1 | Block X + 2 | Block X + 3 (Stop) | Block X + 4 | Block X + 5 |

**Figure 4.1.1:** Demonstration of Authority

The train will not move past Block X + 3.

* + 1. **Speed** (unit: meters-per-second) tells a train the safe speed at which it is permitted to pass through any given block. Speed is suggested by the CTC (minimum: 0 mph, maximum: speed limit for a block). The Track Controllercontroller determines whether the train can continue through a block at the speed suggested by the CTC (suggested speed) or, if the train must stop within SSD blocks. For vitality purposes, the Track Controller will send a vital speed of 0 mph such that the Train Controller immediately slows the train down so that the train stops within SSD. If a train is to be stopped for scheduling purposes, it is commanded to stop by Authority (4.1.1).
  1. **Vitality**



**Figure 4.2.1:** System Failure and Remediation Diagram

* + 1. The system vitality is the collection of the procedures taken to ensure the train system does not enter a situation in which a crash can happen, as seen in **Figure 4.2.1**. There are several conditions that can occur that would cause the system to fail. To avoid such situations, safety procedures are taken by respective modules. In cases of engine failure, brake failure, or signal pickup failure, the remediation is controlled by the Train Model and Train Controller. Failures from the track, particularly track circuit failures and broken railes, will be processed by the Wayside Controller and Track Model. In order to avoid non-vital situations occurring from user actions, modules will perform safety checks before accepting the user’s input.
    2. **Track Controller Redundant PLC Logic:**For redundancy, the PLC will perform logic on the data twice in differing ways. These two methods of processing will execute in the same PLC file at the same time. When called, the PLC will receive the necessary boolean values including block occupancy. The PLC operations are performed twice in different ways but both with the same goal of preserving the system vitality. Once both operations are performed, the two decisions that the PLCs reach will be compared to each other. If they do not agree, then the command to stop will be given.
    3. **Train Model and Controller Fail Handling:**Use cases involving the Emergency and Service Brake, Commanded Speed, and Engine Power take Failures into consideration of its output in accordance with safety.
       1. **Train Model**

Murphy Breaks Engine, Murphy Breaks Signal Pickup, Murphy Breaks Brakes

* + - 1. **Train Controller SW**

Operate Emergency Brake, Update Setpoint Speed, Update Power

* + - 1. **Train Controller HW**

Update Emergency Brake State, Update Service Brake State, Calculate Engine Power and Exported Commanded Speed

* + 1. **Train Controller: Calculate Power Twice over and Compare:**

In order to ensure that a safe power level is being implemented to the engine, Power should be calculated twice over by the train controller in differing ways when a new power calculation is necessary. The resulting power values will then be compared. If the values match, the new power value will be accepted. If they do not match, there will be no change in engine power. If the power values do not match and the train is at risk of entering a non vital situation, the power will be set to 0.

* 1. **Route**
     1. When a train is scheduled by the dispatcher, it is scheduled in a queue that releases the train at the time of departure. If the destination selected by the dispatcher is a station, the train will make all stops (dictated by authority section 4.1) scheduled by the dispatcher, including the destination, then return to the yard. If the destination is a block that does not have a station, the train shall dwell at such block until the dispatcher re-schedules the train to another block.
     2. When a train stops at a station, the dwell time is 60sec. This is achieved by re-scheduling the train for departure from the station at t=arrival time + 60sec.
  2. **Wayside Distribution**



**Figure 4.4.1:** Wayside Distribution on the Green Line

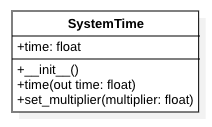
* + 1. Three track controllers will be used on the Green Line. The distribution of the three controllers is shown above in **Figure 4.4.1**. The sections: section A for ‘wayside A’, section B for ‘wayside B’ and section C for ‘wayside C’. This arrangement provides for the following:
       1. As shown in **Figure 4.4.1**, sections A and B and sections B and C overlap at mesh points. Overlapping at mesh points is used so that no track failures or occupancies cause a catastrophic train derailment or collision. For example, wayside B will receive block occupancies of the first SSD blocks in section C following the mesh point (and vice versa) so if within SSD blocks of either B or C, if there is a track failure or occupancy, the train will be commanded to stop in time.
       2. It is also important to note the handling of the switch between track sections F, G and Z (Switch A) and the switch between track sections N, M and R (Switch C). Switch A can be commanded to be toggled only by wayside A and Switch C can only be toggled by wayside C.



**Figure 4.4.2:** Wayside Distribution on the Red Line

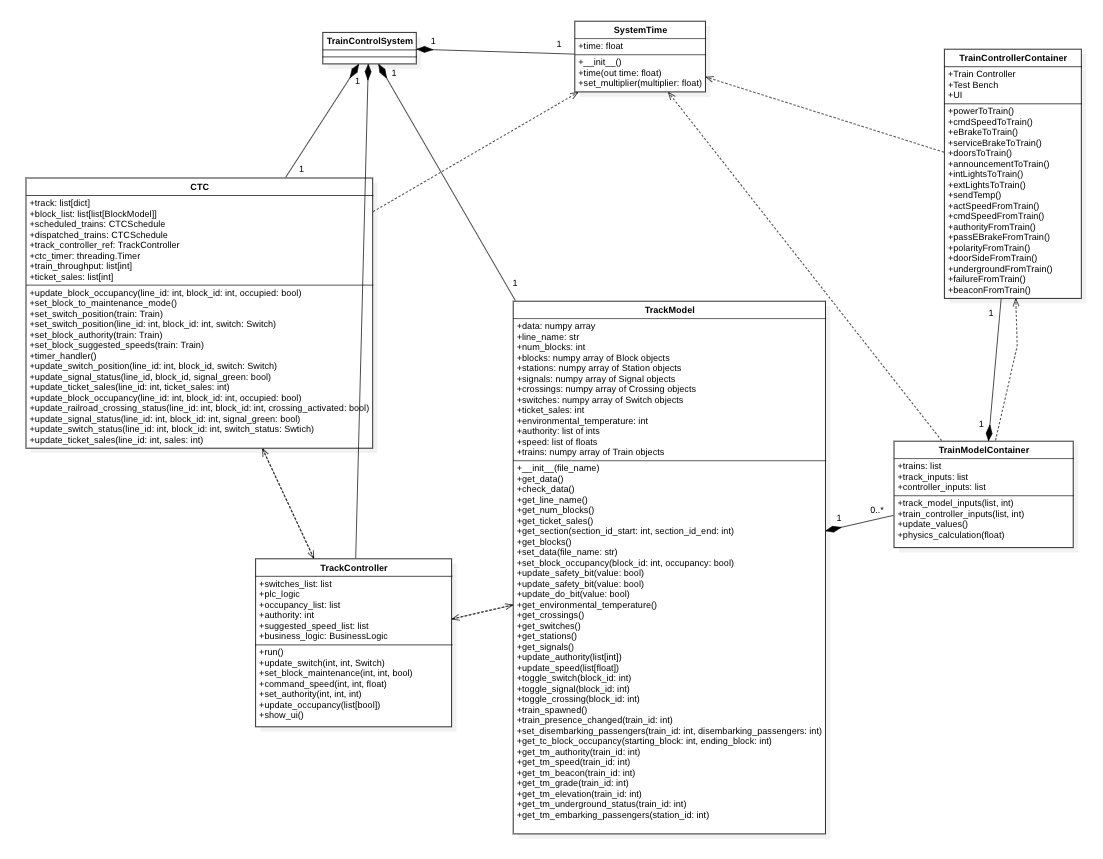
* + 1. Three track controllers will be used on the Red Line. The distribution of the three controllers is shown in **Figure 4.4.2**. The Sections: section A for ‘wayside A’, section B for ‘wayside B’ and section C for ‘wayside C’. This arrangement follows the same reasoning as the arrangement of the three controllers on the Green Line except that there are no switches at the mesh points.
  1. **Timing**

To ensure time synchronization across modules that rely on a time source, a custom time class will be injected into each module that requires it. This custom time class will use a threaded timer to increase the count of initial time by 1 second for each “system second” elapsed. One system second is defined as 1 “real-world” second divided by the time multiplier set for the system by a system user.

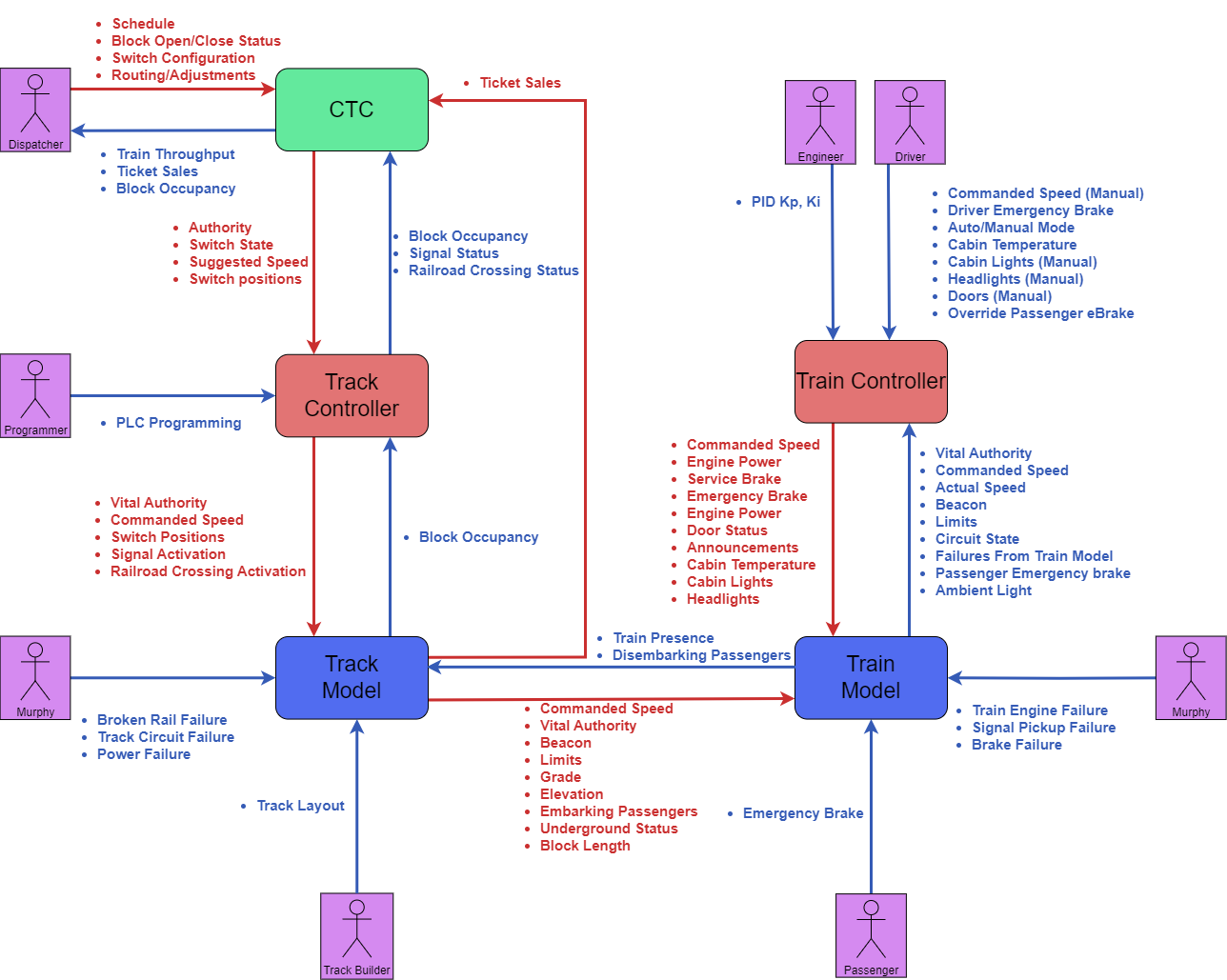


**Figure 4.5:** *SystemTime class diagram*

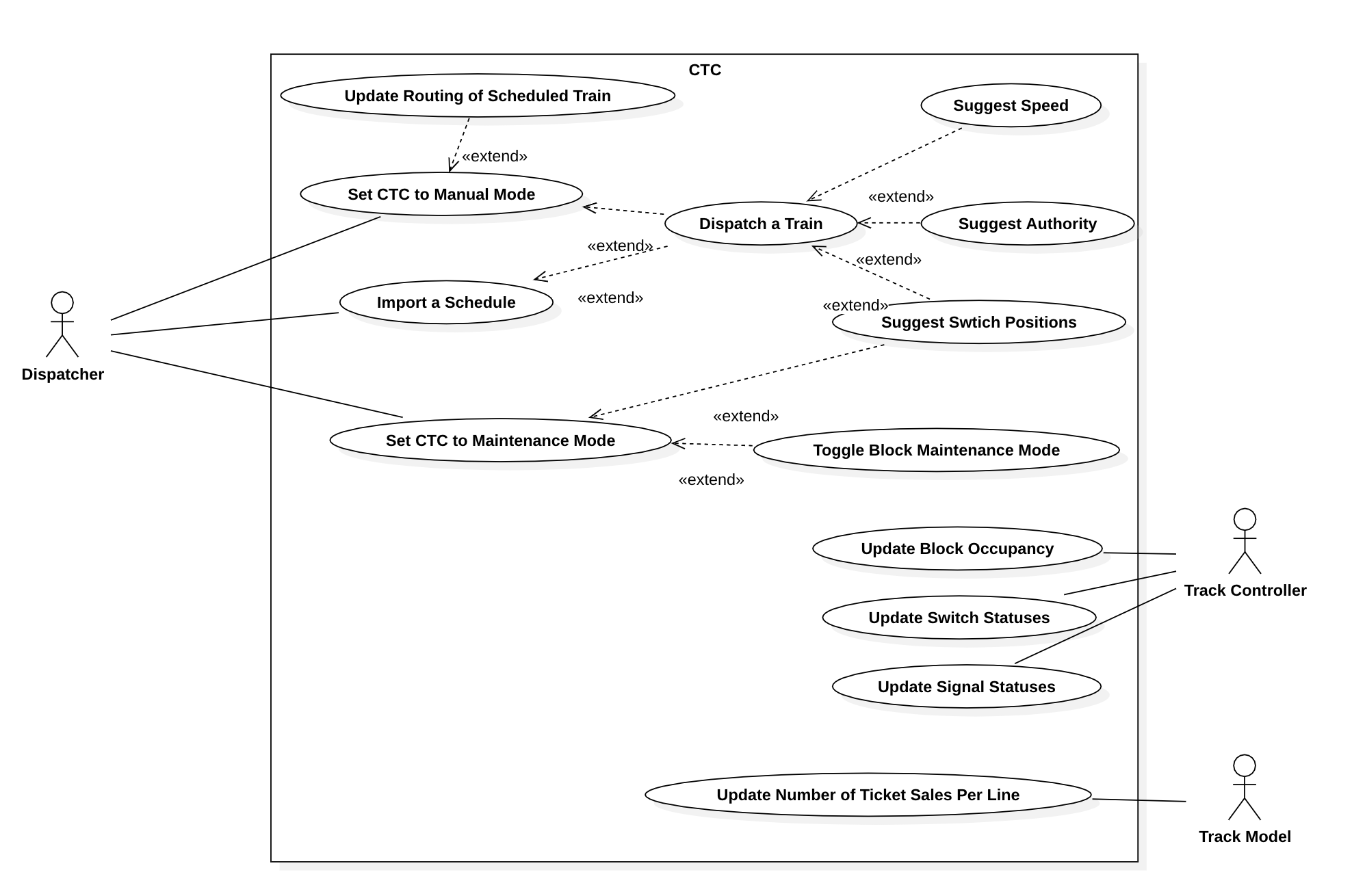
1. **Software Architecture**

****

**Figure 5.1** System Software Architecture Diagram

**Figure 5.1** System Data Architecture Diagram

1. **Module Design**
   1. **CTC**
      1. **Use Case Diagram**

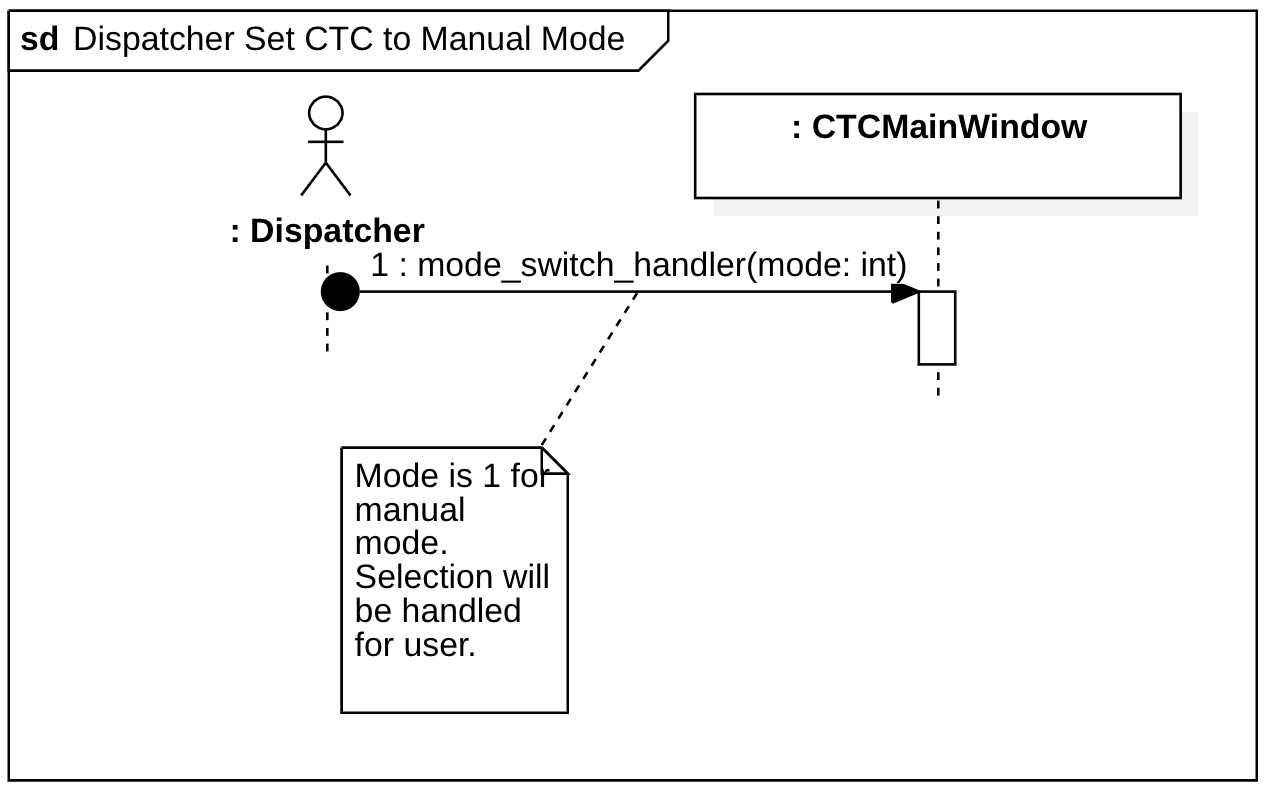


**Figure 6.1.1:** CTC Use Case Diagram

* + 1. **CTC Use Cases**
       1. **Set CTC to Manual Mode**

| Table 6.1.2.1 Set CTC to Manual Mode Use Case Description | |
| --- | --- |
| Actors | Dispatcher, CTCMainWindow |
| Description | 1. CTC is set to manual mode |
| Data | Current CTC Mode |
| Stimulus | Dispatcher sets CTC to manual mode |
| Response | CTC is set to manual mode and Dispatcher can manually dispatch trains. |

**Table 6.1.2.1:** Set CTC to Manual Mode Use Case Description

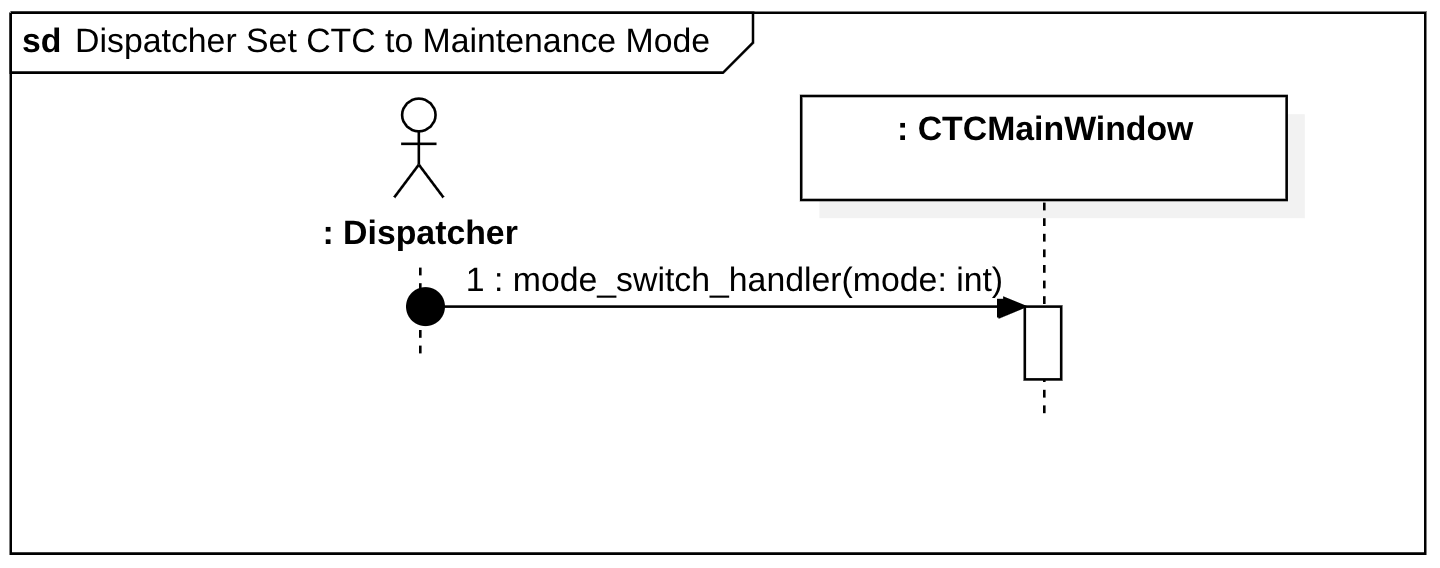


**Figure 6.1.2.1:** Set CTC to Manual Mode Sequence Diagram

* + - 1. **Set CTC to Maintenance Mode**

| Table 6.1.2.2 Set CTC to Maintenance Mode Use Case Description | |
| --- | --- |
| Actors | Dispatcher, CTCMainWindow |
| Description | 1. CTC is set to maintenance mode |
| Data | Current CTC Mode |
| Stimulus | Dispatcher sets CTC to maintenance mode |
| Response | CTC is set to maintenance mode and Dispatcher can set blocks to maintenance mode and set switch positions. Trains are not dispatchable until CTC is returned to manual mode. |

**Table 6.1.2.2:** Set CTC to Maintenance Mode Use Case Description

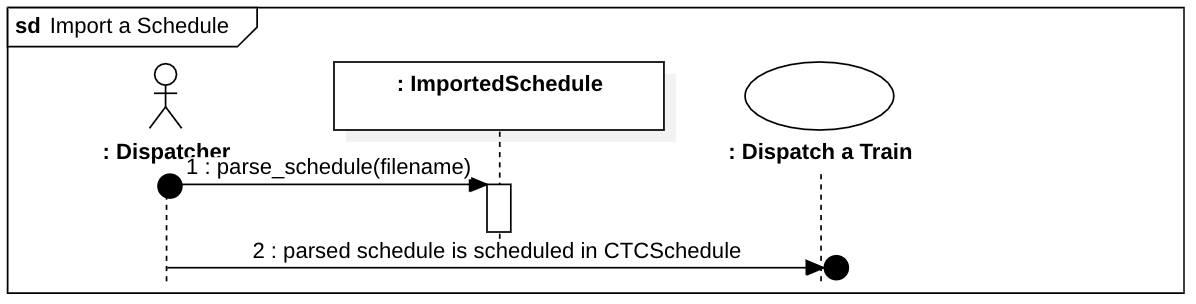


**Figure 6.1.2.2:** Set CTC to Maintenance Mode Sequence Diagram

* + - 1. **Import a Schedule**

| Table 6.1.2.3 Import a Schedule Use Case Description | |
| --- | --- |
| Actors | Dispatcher, ImportedSchedule, Dispatch a Train (use case) |
| Description | 1. Dispatcher provides ImportedSchedule with file path to schedule file’ 2. ImportedSchedule parses schedule for data validity    1. Each valid route is imported, invalid routes are discarded 3. Imported routes are scheduled for dispatch |
| Data | Importable Schedule: CSV File following correct format supplied with system delivery, imported routes: list of route objects to be scheduled in CTCSchedule for dispatch |
| Stimulus | Dispatcher imports a schedule in automatic mode |
| Response | Schedulable trains are scheduled |

**Table 6.1.2.3:** Import a Schedule Use Case Description

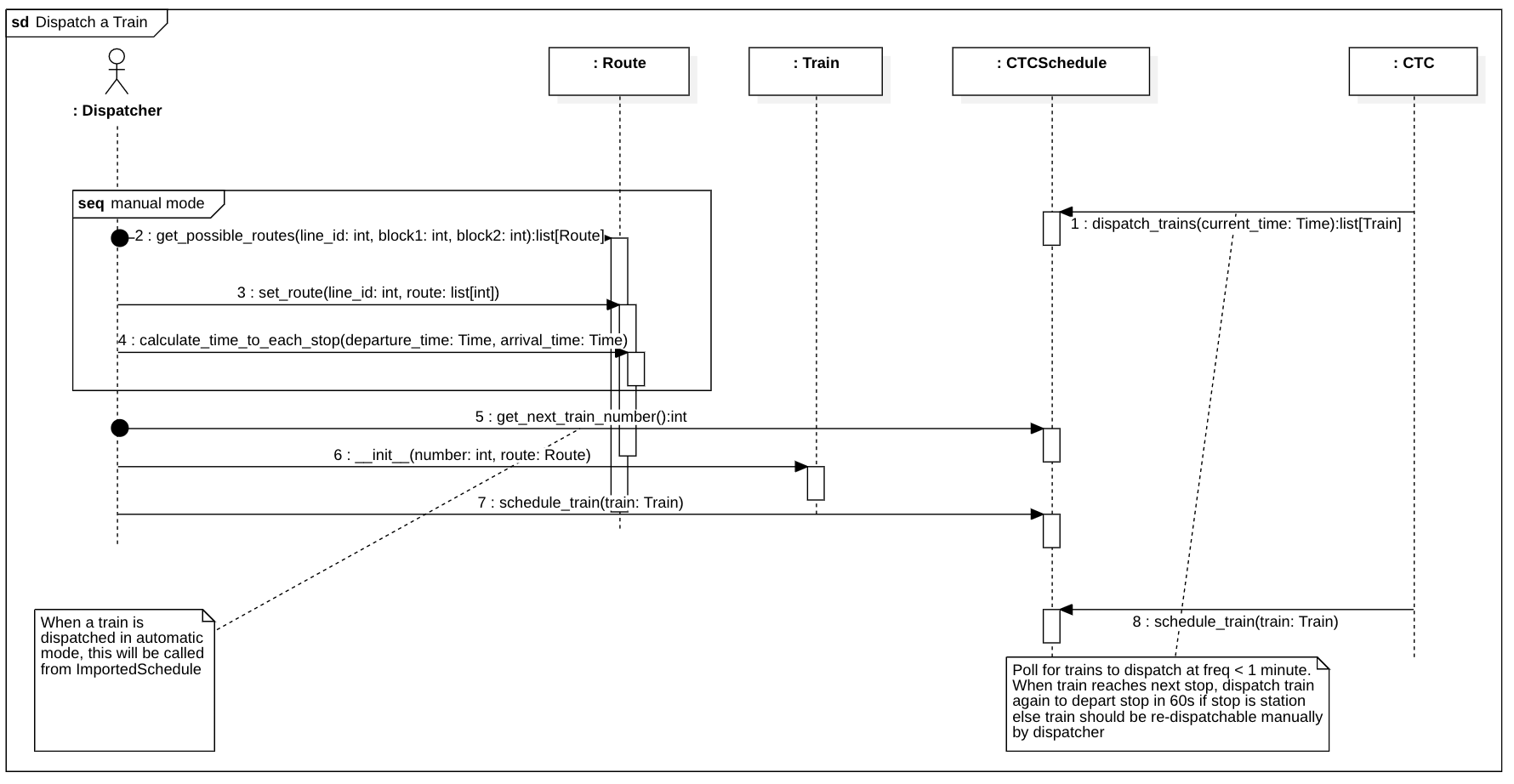


**Figure 6.1.2.3:** Import a Schedule Sequence Diagram

* + - 1. **Dispatch a Train**

| Table 6.1.2.4 Dispatch a Train Use Case Description | |
| --- | --- |
| Actors | Dispatcher, CTC, CTCSchedule |
| Description | 1. If manual mode dispatch:    1. All possible routes to desired destination are presented to Dispatcher    2. Dispatcher chooses route from set of possible routes    3. CTC Office calculates time to each destination; Dispatcher adjusts accordingly. Continue to 2. 2. All cases:    1. Next train number is requested from the CTCSchedule    2. Train object is created with route number, route    3. Train object is scheduled for dispatch at departure time |
| Data | Departure time, stop arrival times, route. |
| Stimulus | Whenever the Dispatcher uploads a schedule or the Dispatcher |
| Response | The train(s) will be scheduled if departure is in future relative to time scheduled or will be dispatched if schedule time is dispatch time. |

**Table 6.1.2.4:** Dispatch a Train Use Case Description

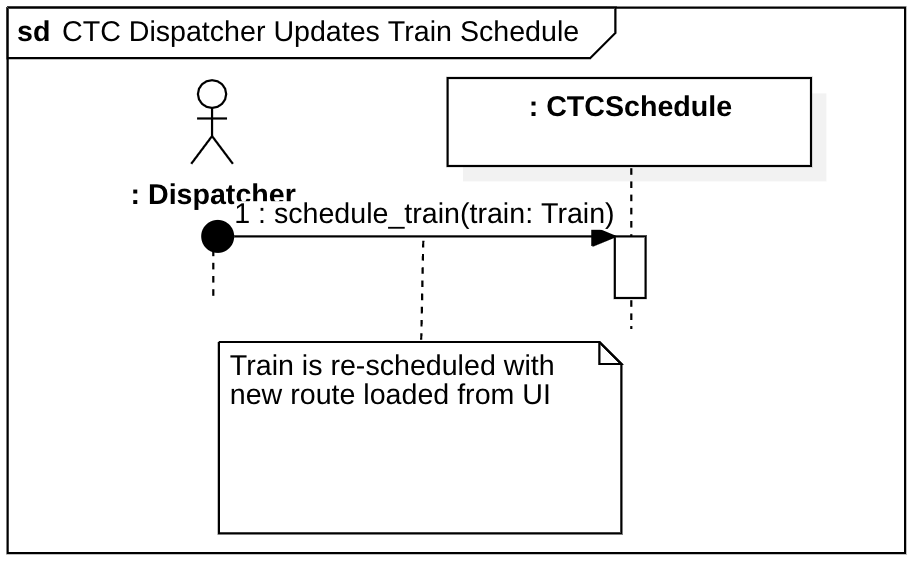


**Figure 6.1.2.4:** Dispatch a Train Sequence Diagram

* + - 1. **Update Routing of a Scheduled Train**

| Table 6.1.2.5 Update Routing of Scheduled Train Use Case Description | |
| --- | --- |
| Actors | Dispatcher, CTC |
| Description | 1. Dispatcher sends CTC updated route for a scheduled, but not dispatched, train. 2. CTC removes the original train that Dispatcher edits from the schedul queue and re-queues train with new route. |
| Data | Route object containing updated route and schedule |
| Stimulus | Whenever the Dispatcher re-schedules a train |
| Response | The CTC will dispatch the train according to the updated schedule. |

**Table 6.1.2.5:** Update Routing of Scheduled Train Use Case Description

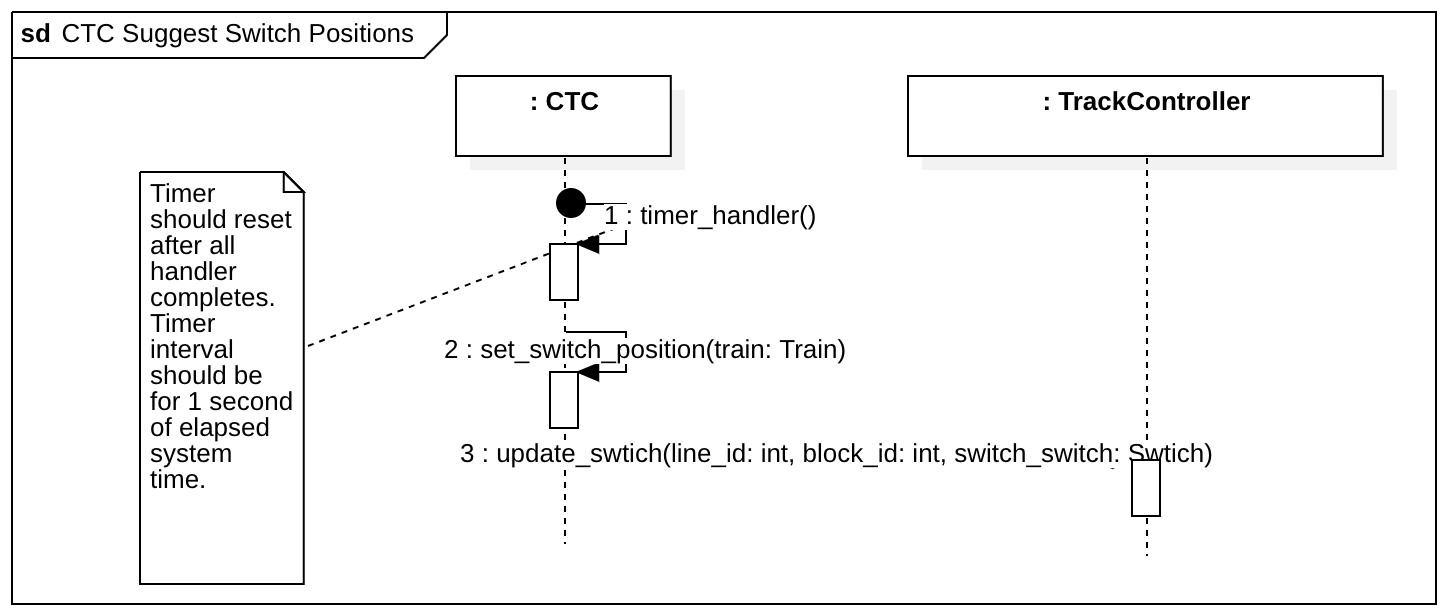


**Table 6.1.2.5:** Update Routing of Scheduled Train Sequence Diagram

* + - 1. **Suggest Switch Positions**

| Table 6.1.2.6 Suggest Switch Positions Use Case Description | |
| --- | --- |
| Actors | CTC, Track Controller |
| Description | 1. For each train currently running on a the tracks (in dispatched CTCSchedule), CTC checks if a switch position needs to be suggested (each train that is within a SSD block window of a switch) 2. For each switch position suggested where current\_switch\_position != suggested\_switch\_position, CTC sends Wayside suggested switch position |
| Data | Current switch positions, suggested switch positions, train routes and positions |
| Stimulus | Expires; handler called (timer is set according to 1s in system time) |
| Response | Switch positions for routing are suggested to Track Controller |

**Table 6.1.2.6:** Suggest Switch Positions Use Case Description

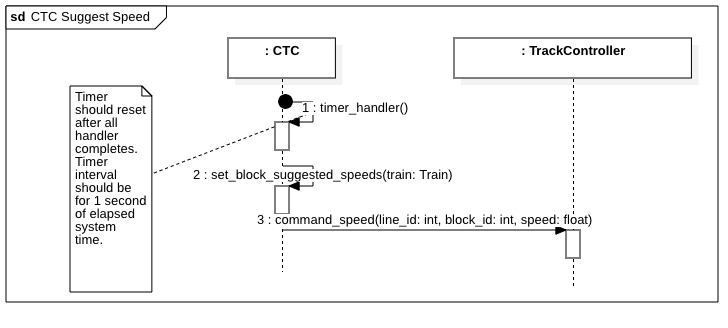


**Figure 6.1.2.6:** Suggest Switch Positions Sequence Diagram

* + - 1. **Suggest Speed**

| Table 6.1.2.7 Suggest Speed Use Case Description | |
| --- | --- |
| Actors | CTC, Track Controller |
| Description | 1. For each train currently running on the tracks (in dispatched CTCSchedule), CTC sets block suggested speeds for each of the blocks in the SSD block window along the train's route 2. Each block that has an updated suggested speed is sent to the track controller |
| Data | Current suggested speeds, updated suggested speeds, train Route and position |
| Stimulus | Timer Expires; handler called (timer is set according to 1s in system time) |
| Response | Suggested Block Speed Limits are sent to Track Controller |

**Table 6.1.2.7:** Suggest Speed Use Case Description

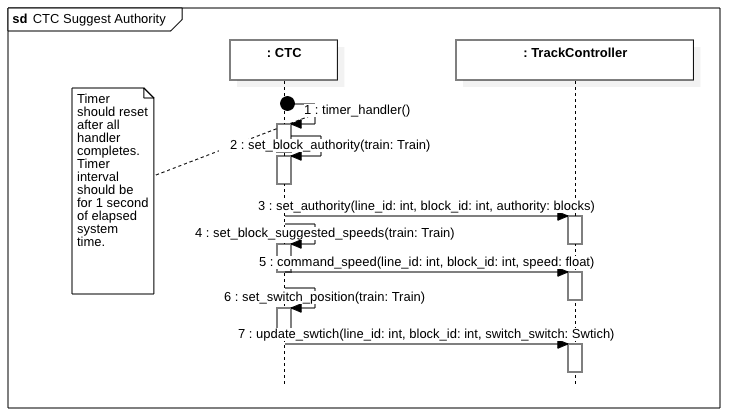


**Figure 6.1.2.7:** Suggest Speed Sequence Diagram

* + - 1. **Suggest Authority**

| Table 6.1.2.8 Suggest Authority Use Case Description | |
| --- | --- |
| Actors | CTC, TrackController |
| Description | 1. For each train currently running on the tracks (trains in dispatched CTCSchedule), CTC sets SSD +1 block window authority for each block. Each block’s authority message will relay the number of blocks until the train’s next scheduled stop. 2. Each block that has an updated suggested authority is updated in the track controller |
| Data | Current authority in each block, new authority in each block, train Route and position |
| Stimulus | Timer expires; handler called (timer is set according to 1s in system time) |
| Response | Authority for each block is sent to Track Controller |

**Table 6.1.2.8:** Suggest Authority Use Case Description

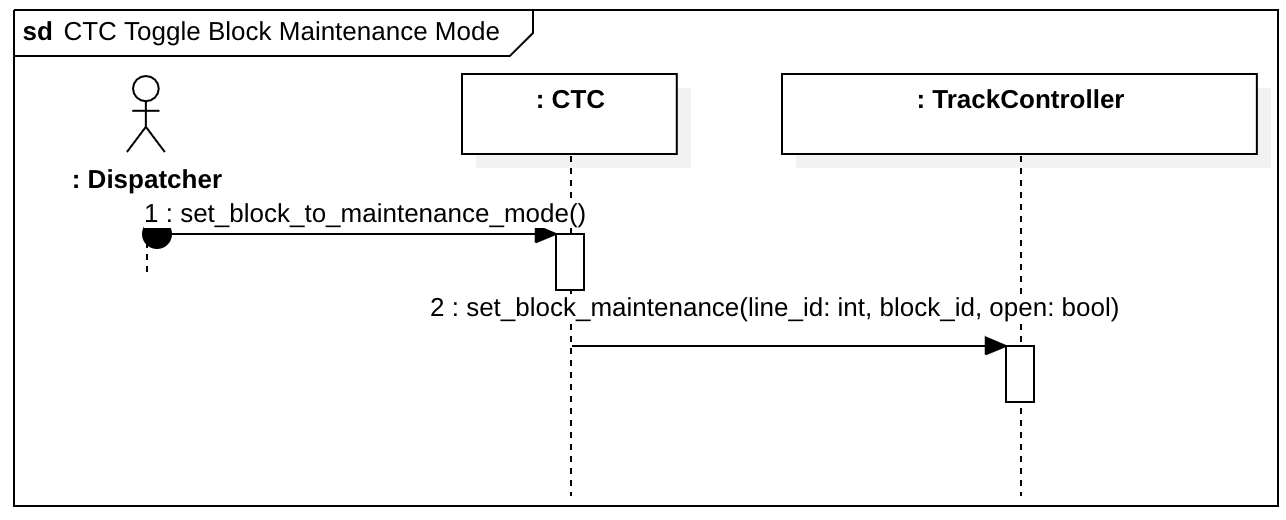


**Figure 6.1.2.8:** Suggest Authority Sequence Diagram

* + - 1. **Toggle Block Maintenance Mode**

| Table 6.1.2.9 Toggle Block Maintenance Mode Use Case Description | |
| --- | --- |
| Actors | Dispatcher, CTC, Track Controller |
| Description | 1. Block is set to maintenance mode or open |
| Data | Line id, Block id, block open status |
| Stimulus | Dispatcher sets CTC to maintenance mode and closes/opens block |
| Response | CTC updates Track Controller with block state |

**Table 6.1.2.9:** Toggle Block Maintenance Mode Use Case Description

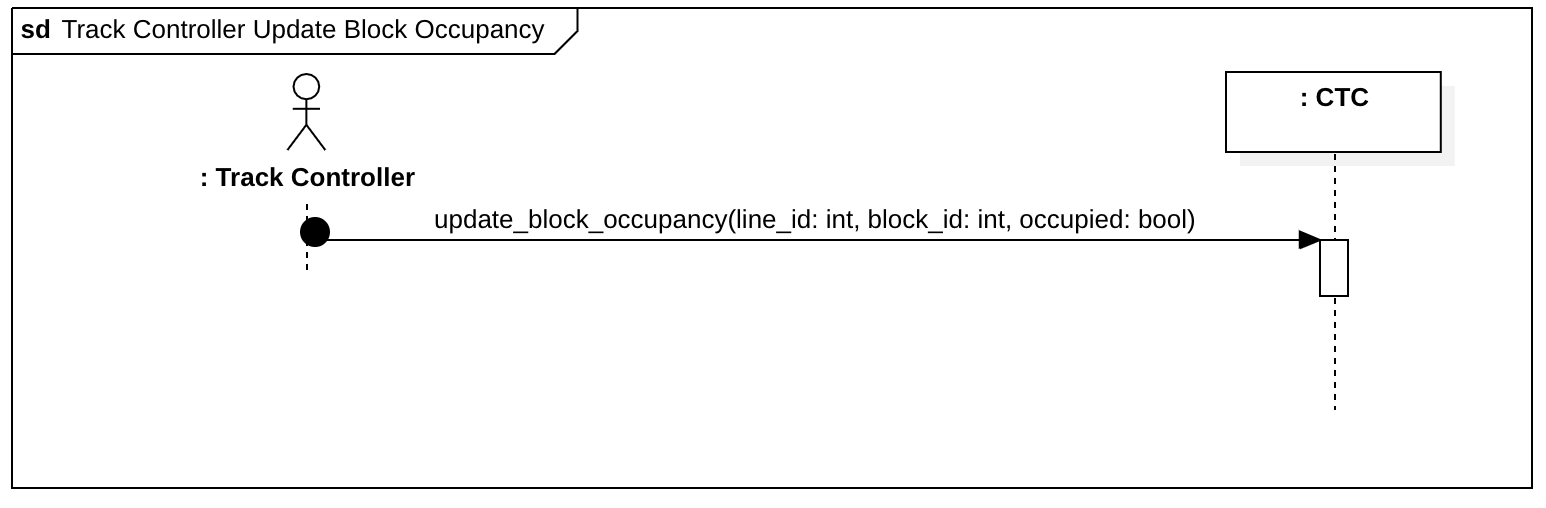


**Figure 6.1.2.9:** Toggle Block Maintenance Mode Sequence Diagram

* + - 1. **Update Block Occupancy**

| Table 6.1.2.10 Update Block Occupancy Use Case Description | |
| --- | --- |
| Actors | Track Controller, CTC |
| Description | 1. Block occupancy is updated in CTC (occupied or not occupied) when an occupancy state changes |
| Data | Line id, block id, occupancy state |
| Stimulus | A block occupancy state changes |
| Response | Track Controller updates CTC with block occupancy status |

**Table 6.1.2.10:** Update Block Occupancy Use Case Description

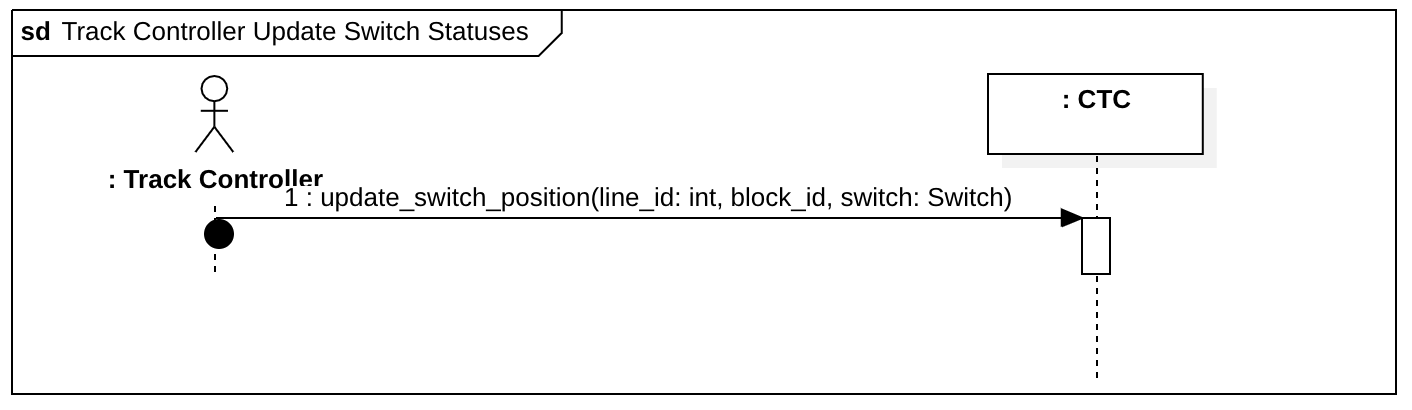


**Figure 6.1.2.10:** Update Block Occupancy Sequence Diagram

* + - 1. **Update Switch Statuses**

| Table 6.1.2.11 Update Switch Statuses Use Case Description | |
| --- | --- |
| Actors | Track Controller, CTC |
| Description | 1. Switch positions are updated in CTC |
| Data | Line id, block id, switch state (Switch object) |
| Stimulus | Track Controller changes switch state |
| Response | Track controller updates CTC with updated switch state |

**Table 6.1.2.11:** Update Switch Statuses Use Case Description

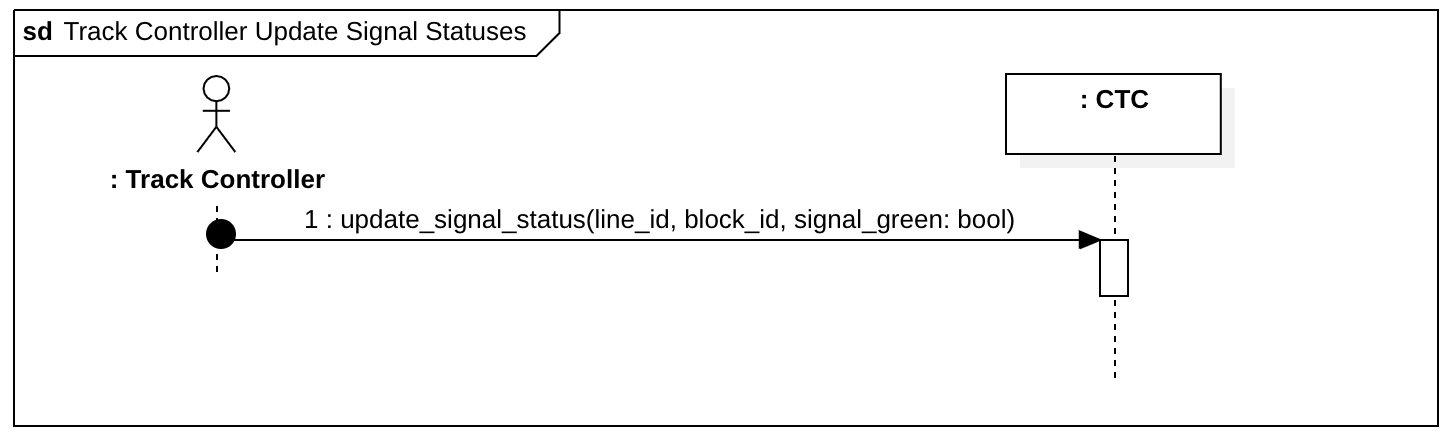


**Figure 6.1.2.11:** Update Switch Statuses Sequence Diagram

* + - 1. **Update Signal Statuses**

| Table 6.1.2.12 Update Signal Statuses Use Case Description | |
| --- | --- |
| Actors | Track Controller, CTC |
| Description | 1. Signal statuses are updated in CTC |
| Data | Line id, block id, signal is green boolean |
| Stimulus | Track Controller changes signal state |
| Response | Track Controller updates CTC with updated signal state |

**Table 6.1.2.12:** Update Signal Statuses Use Case Description

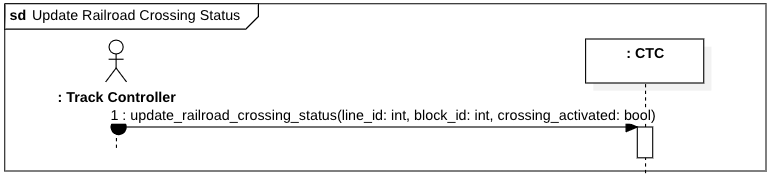


**Figure 6.1.2.12:** Update Signal Statuses Sequence Diagram

* + - 1. **Update Railroad Crossing Status**

| Table 6.1.2.13 Update Railroad Crossing Statuses Use Case Description | |
| --- | --- |
| Actors | Track Controller, CTC |
| Description | 1. Railroad Crossing statuses are updated in CTC |
| Data | Line id, block id, railroad crossing activation boolean |
| Stimulus | Railroad crossing is activated/deactivated |
| Response | Track Controller updates CTC with updated railroad crossing status |

**Table 6.1.2.13:** Update Railroad Crossing Statuses Use Case Description

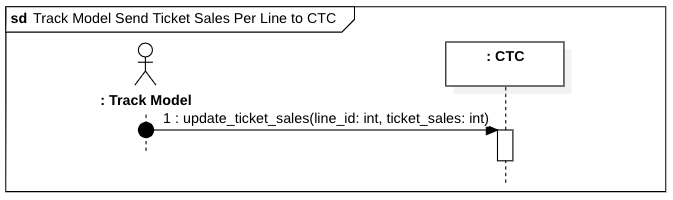


**Figure 6.1.2.13:** Update Railroad Crossing Statuses Sequence Diagram

* + - 1. **Update Number of Ticket Sales per Line**

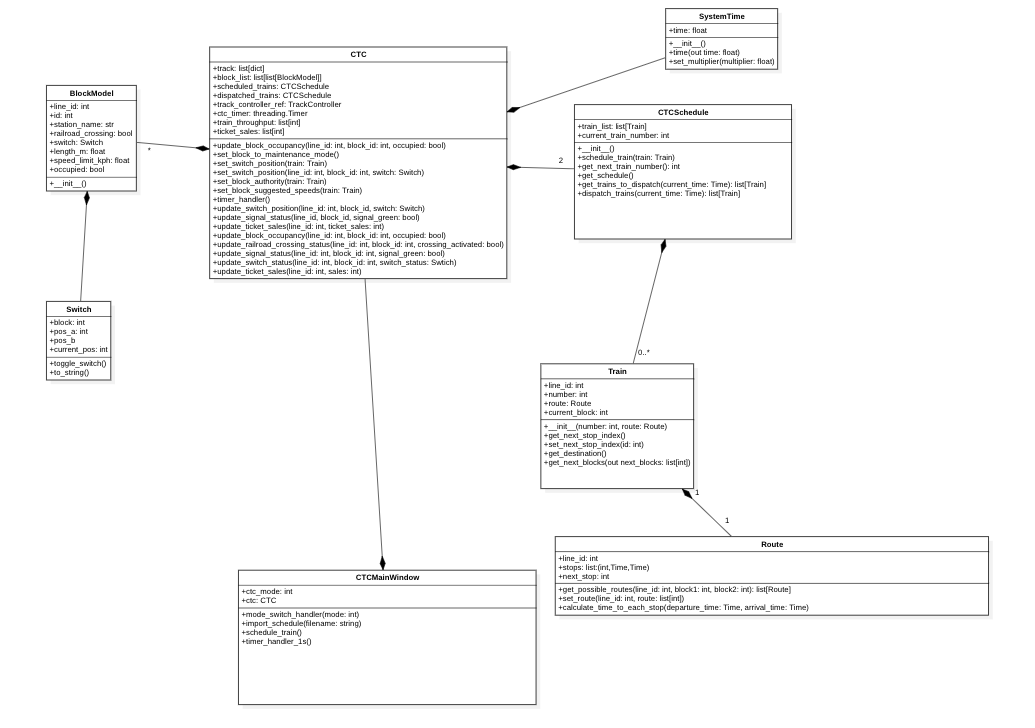
| Table 6.1.2.14 Update Number of Ticket Sales per Line Use Case Description | |
| --- | --- |
| Actors | Track Model, CTC |
| Description | 1. Number of ticket sales for a line is updated |
| Data | Line id, number of ticket sales |
| Stimulus | Update in number of ticket sales on a line |
| Response | Track Model updates CTC with number of ticket sales on a line |

**Table 6.1.2.14:** Update Number of Ticket Sales Per Line Use Case Description



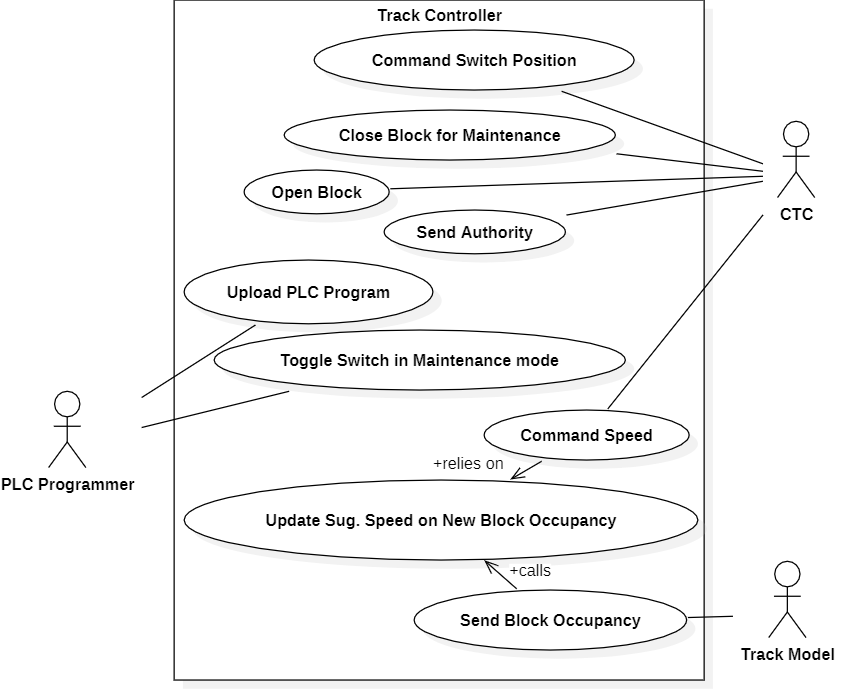
**Figure 6.1.2.14:** Update Number of Ticket Sales per Line Sequence Diagram

* + 1. **Software Class Diagram**



**Figure 6.1.3:** CTC Software Class Diagram

* 1. **Track Controller SW**
     1. **Use Case Diagram**

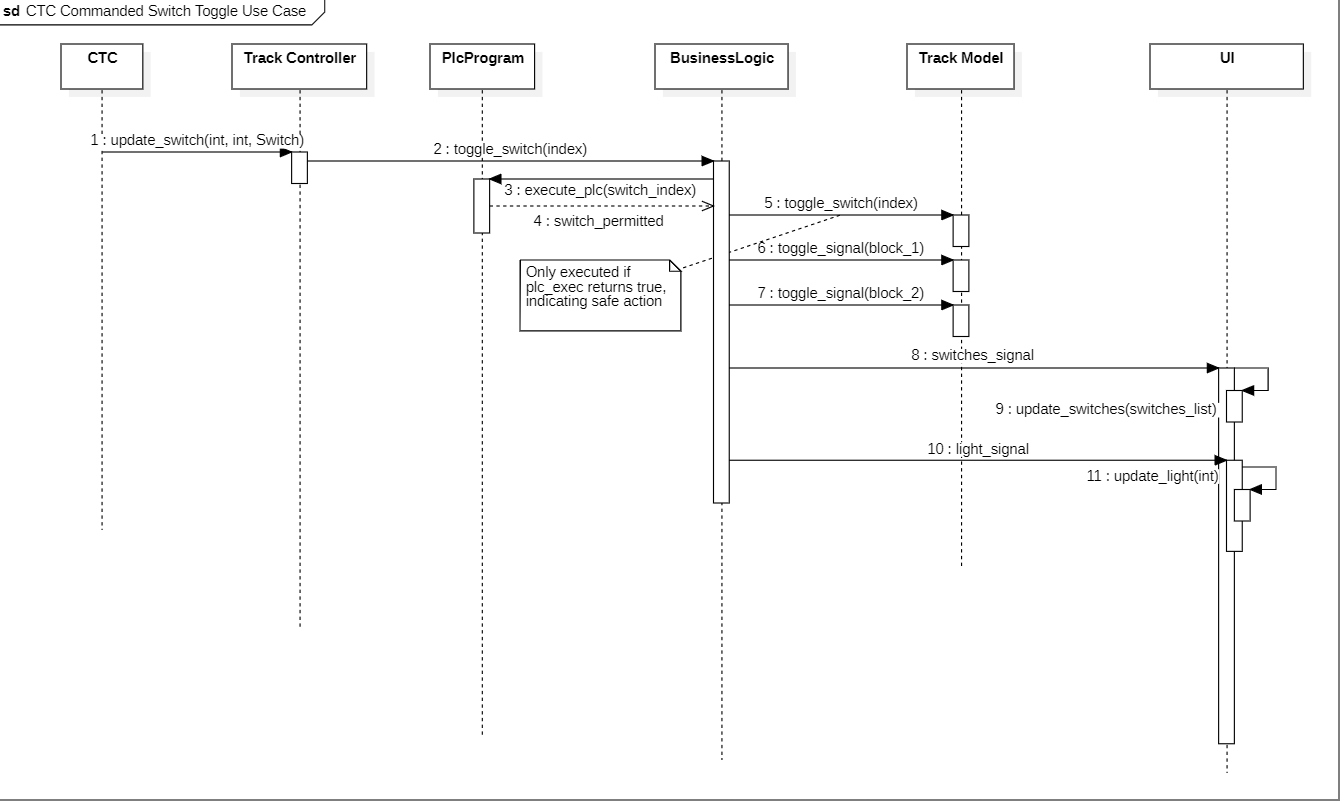


**Figure 6.2.1:** Track Controller System Use Case Diagram

* + 1. **Use Cases**
       1. **CTC Command Switch Position**

| Table 6.2.2.1: CTC Command Switch Position Use Case Description | |
| --- | --- |
| Actors | CTC |
| Description | In order to properly route trains, the CTC commands switch positions to the track controller which triggers the following actions:   1. CTC indicates a switch to be toggled by Track Controller 2. Track Controller executes plc program to determine if the switch can be toggled    1. If it is not safe, do nothing 3. If the switch can be toggled:    1. Toggle switch and associated signals on the Track Model    2. Update the Track Controller UI |
| Data | Int - line id, int block index, Switch - switch object |
| Stimulus | CTC calls update\_switch on Track Controller |
| Response | The Track Controller will execute the plc program to determine if it is safe to toggle the switch, and if it is, the switch will be toggled on the Track Model. |

**Table 6.2.2.1:** CTC Command Switch Position Use Case Description

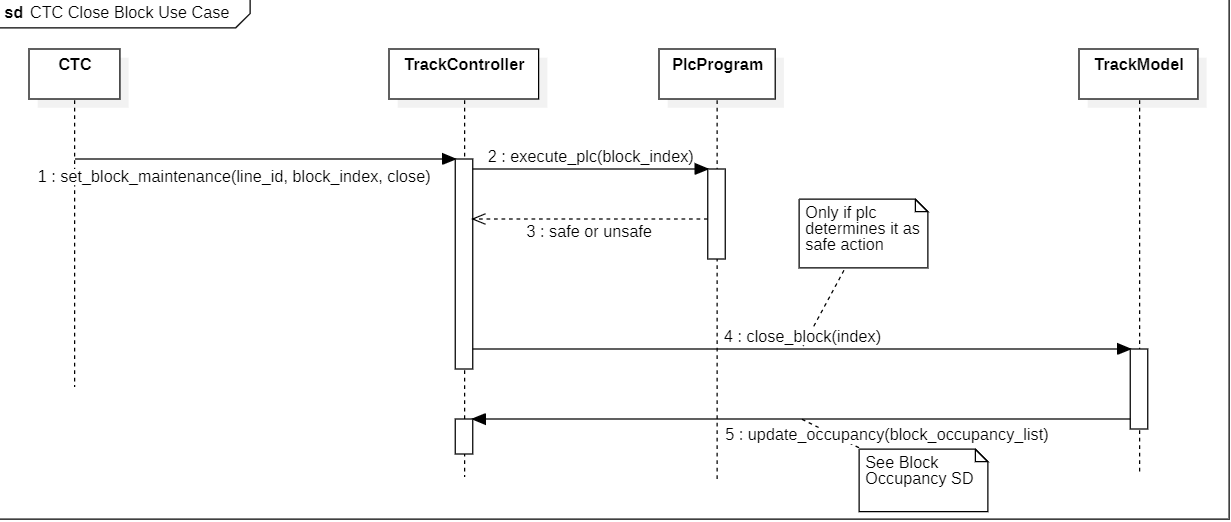


**Figure 6.2.2.1:** CTC Command Switch Position Use Case Sequence Diagram

* + - 1. **CTC Close Block for Maintenance**

| Table 6.2.2.2: CTC Close Block for Maintenance Use Case Description | |
| --- | --- |
| Actors | CTC |
| Description | In order to safely perform maintenance, the CTC must direct the closure of a block to the Track Controller which triggers the following actions:   1. CTC indicates a block to be closed to the Track Controller 2. The track controller executes the plc program to determine if this is a safe action    1. If it is not safe, do nothing 3. If it is safe to close the block, tell the Track Model to close the block |
| Data | Int line id, int block index, bool close |
| Stimulus | The CTC directs the Track Controller to close a block |
| Response | If it is safe to close the block, the track controller will close the block. Otherwise, nothing happens |

**Table 6.2.2.2:** CTC Close Block for Maintenance Use Case Description

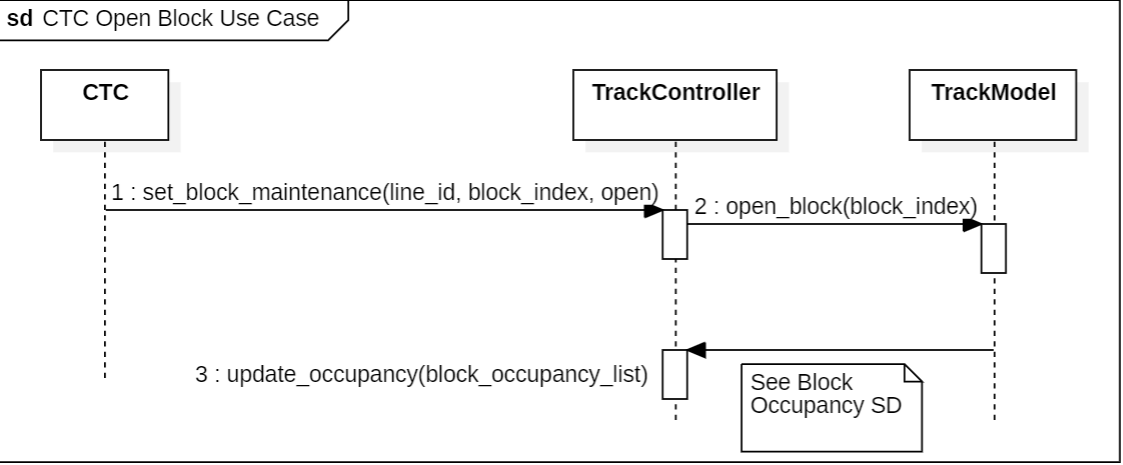


**Figure 6.2.2.2:** CTC Close Block for Maintenance Use Case Sequence Diagram

* + - 1. **CTC Open Block**

| Table 6.2.2.3: CTC Open Block Use Case Description | |
| --- | --- |
| Actors | CTC |
| Description | After block maintenance is completed, the CTC needs to re-open the block. This triggers the following actions:   1. The CTC tells the track controller to open the block 2. The track controller tells the track model to open the block  * This is always a safe action, so no further processing is required, the message is just passed along and the action is executed |
| Data | Int line id, int block index, bool open |
| Stimulus | The CTC tells the Track Controller to open a block |
| Response | The Track Controller opens the block on the Track Model |

**Table 6.2.2.3:** CTC Open Block Use Case Description

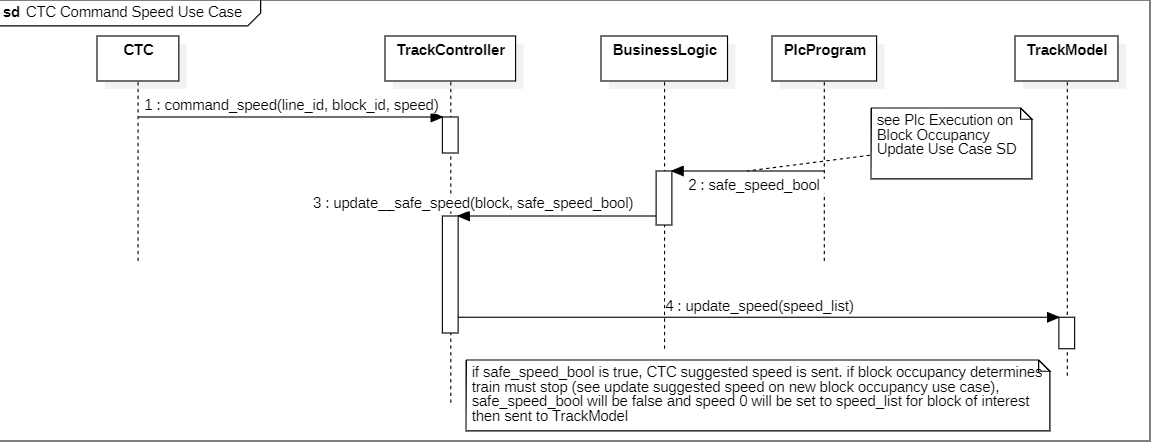


**Figure 6.2.2.3:** CTC Open Block Use Case Sequence Diagram

* + - 1. **CTC Command Speed**

| Table 6.2.2.4: CTC Command Speed Use Case Description | |
| --- | --- |
| Actors | CTC |
| Description | The CTC commands a speed for a train to travel throughout the system. This is sent on a block by block basis   1. The CTC commands a speed to a block to the Track Controller 2. The Track Controller, processing block occupancies from the Track Model, is continuously updating whether a train can safely travel into a block or not, sending a safe\_speed\_bool that is true if it’s safe to travel through the block or false if the train must not travel through the block 3. If safe\_speed\_bool is false, CTC commanded speed is not sent. Rather, 0 is sent by the Track Controller. 4. If safe\_speed\_bool is true, the CTC commanded speed for the block is sent. |
| Data | Bool: safe\_speed\_bool, int: line id, int: block id, float: speed, list: speed list |
| Stimulus | CTC Commands a speed, track controller receives new occupancy |
| Response | Safe speed is sent from Track controller to Track Model |

**Table 6.2.2.4:** CTC Command Speed Use Case Description

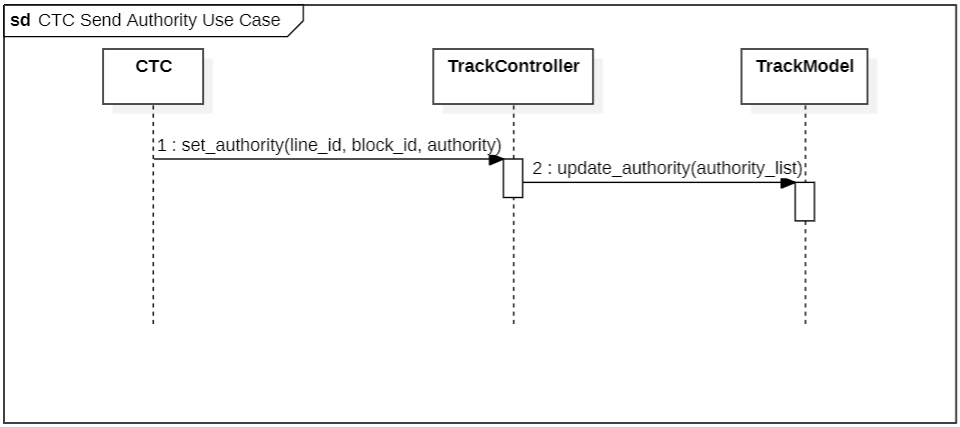


**Figure 6.2.2.4:** CTC Command Speed Use Case Sequence Diagram

* + - 1. **CTC Send Authority**

| Table 6.2.2.5: CTC Send Authority Use Case Description | |
| --- | --- |
| Actors | CTC |
| Description | The CTC updates authority on a block by block basis to ensure trains know where their next stop is. This is done through the following actions involving the CTC and Track Controller:   1. CTC sets authority for a certain block 2. The Track Controller passes the authority to the Track Model |
| Data | Int: line\_id, int: block\_id, int: authority, list: authority\_list |
| Stimulus | CTC sends authority for a block to Track Controller |
| Response | Track Controller updates a list containing current authorities and sends it to Track Model |

**Table 6.2.2.5:** CTC Send Authority Use Case Description

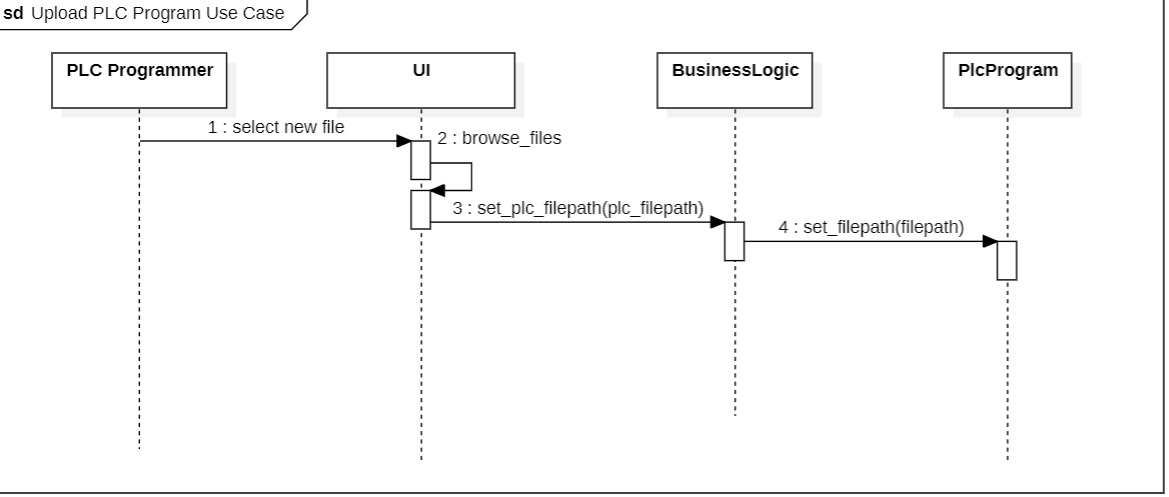


**Figure 6.2.2.5:** CTC Send Authority Use Case Sequence Diagram

* + - 1. **PLC Programmer Upload PLC Program**

| Table 6.2.2.6: Upload PLC Program Use Case Description | |
| --- | --- |
| Actors | PLC Programmer |
| Description | The PLC programmer must upload a PLC program at each instance of a wayside controller. This is done through the following actions:   1. The plc programmer selects a new file through the UI 2. The filepath of the selected file is used as the PLC program |
| Data | Str: filepath |
| Stimulus | The PLC Program selects a file from the file browser after clicking the upload file button |
| Response | The PLC program loads the plc file from the selected file’s filepath |

**Table 6.2.2.6:** Upload PLC Program Use Case Description

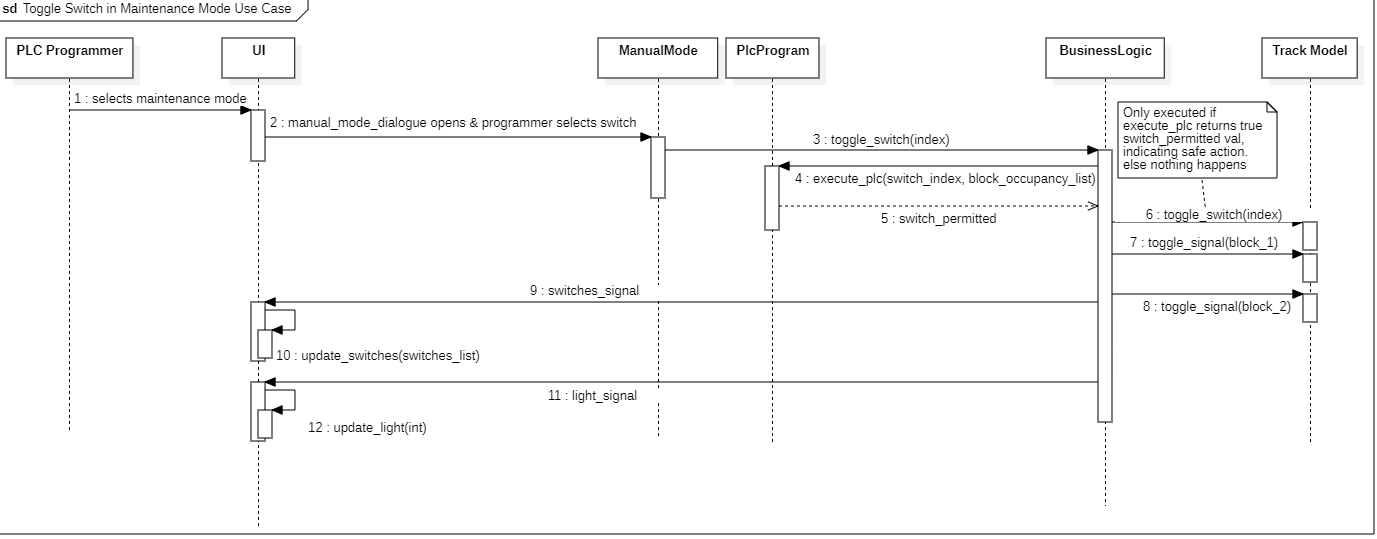


**Figure 6.2.7:** Upload PLC Program Use Case Sequence Diagram

* + - 1. **PLC Programmer Toggle Switch in Maintenance Mode**

| Table 6.2.2.7: Toggle Switch in Maintenance Mode Use Case Description | |
| --- | --- |
| Actors | PLC Programmer |
| Description | The PLC Programmer wishes to manually a toggle a switch through a maintenance mode dialogue by way of the following actions:   1. PLC Programmer selects maintenance mode 2. A dialogue appears with buttons for each switch that allow for a toggle 3. The programmer selects a switch to toggle 4. The PLC program determines if it is safe to toggle the switch based on block occupancy    1. If it is not safe, do nothing 5. If safe, track model is told to update switch and light statuses |
| Data | Int : switch\_index, list: block\_occupancy\_list, int: block 1 signal, int: block 2 signal, bool: switch\_permitted |
| Stimulus | PLC programmer selects a switch to toggle |
| Response | Track Controller either does or does not toggle the switch based on whether it is a safe action |

**Table 6.2.2.7:** Toggle Switch in Maintenance Mode Use Case Description

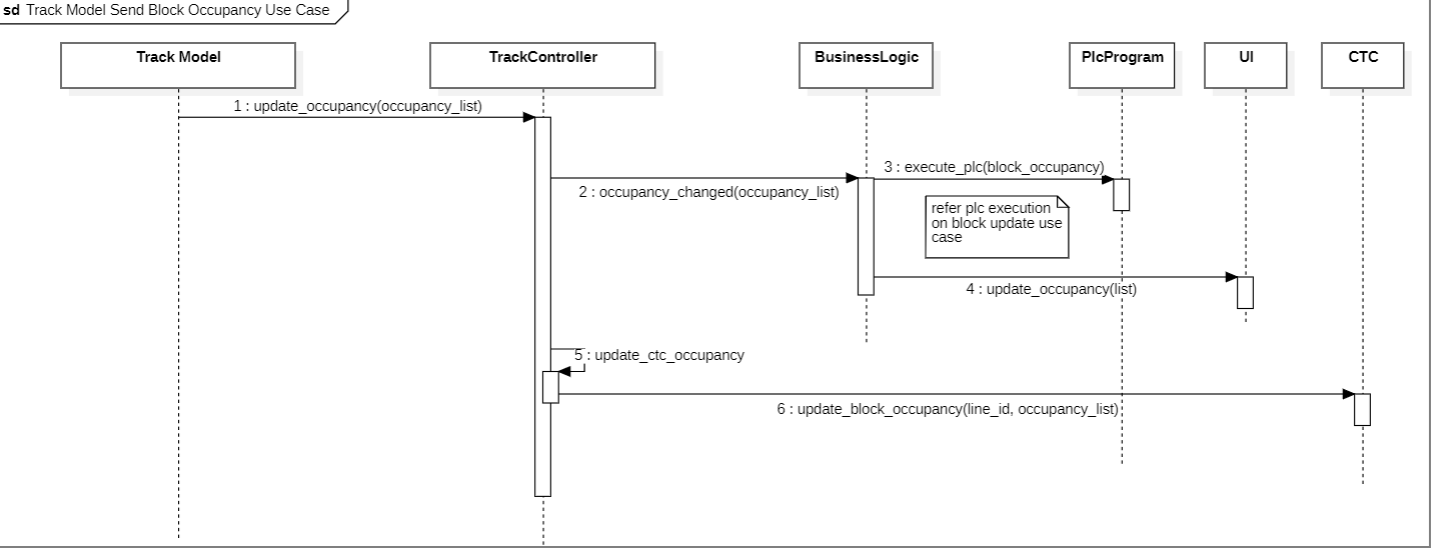


**Figure 6.2.2.7:** Toggle Switch in Maintenance Mode Use Case Sequence Diagram

* + - 1. **Track Model Send Block Occupancy**

| Table 6.2.2.8: Track Model Send Block Occupancy Use Case Description | |
| --- | --- |
| Actors | Track Model |
| Description | The Track Model sends a block occupancy to the Track Controller everytime a new block is occupied. This triggers the following actions:   1. The track controller sends the new occupancy list to its PLC program to be executed, determining which blocks are safe to be entered 2. The UI is updated with the new occupancy 3. The CTC is updated with the new block occupancies |
| Data | List: occupancy\_list, int line id |
| Stimulus | Track Model sends new block occupancy to Track Controller |
| Response | Track Controller executes PLC program, updates UI, and sends new occupancy to CTC |

**Table 6.2.2.8:** Track Model Send Block Occupancy Use Case Description

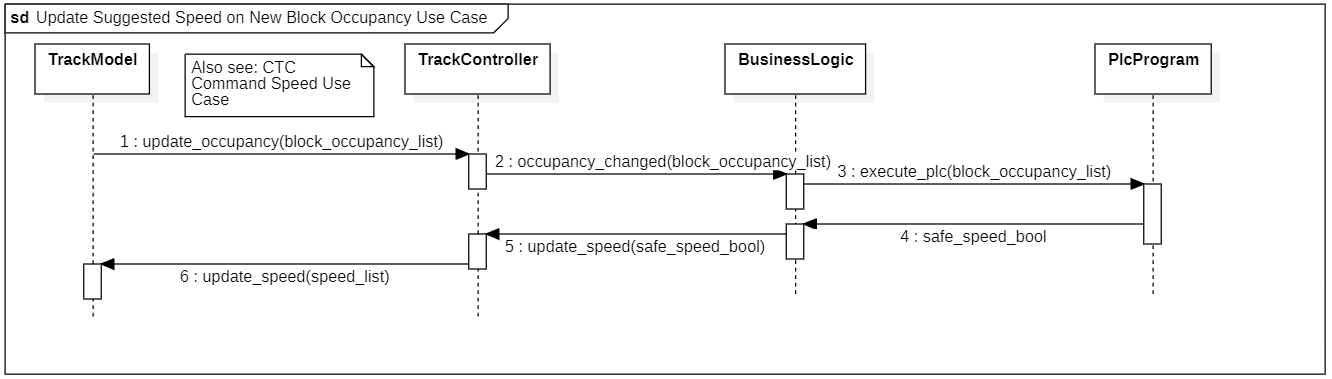


**Figure 6.2.2.8:** Track Model Send Block Occupancy Use Case Sequence Diagram

* + - 1. **Update Suggested Speed on New Block Occupancy**

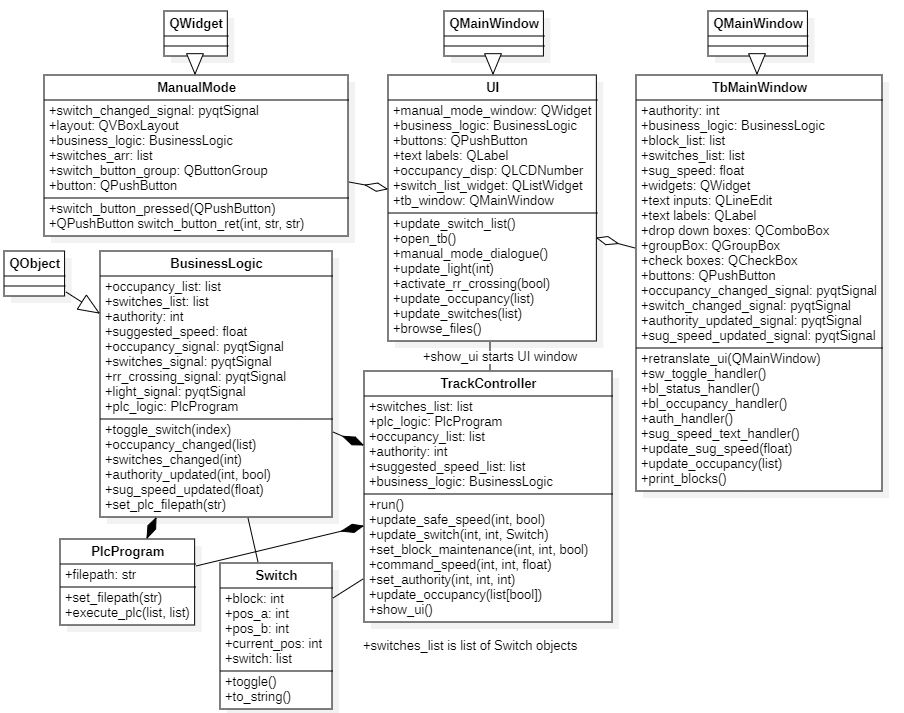
| Table 6.2.2.9: Update Suggested Speed on New Block Occupancy Use Case Description | |
| --- | --- |
| Actors | [internal] |
| Description | Any time block occupancy is updated, the PLC must re-evaluate what blocks are safe to travel through. This is relied on for several internal use cases including any switch toggle or speed command |
| Data | Block\_occupancy\_list, |
| Stimulus | Block occupancy is updated |
| Response | PLC program determines safe travel |

**Table 6.2.2.9:** Update Suggested Speed on New Block Occupancy Use Case Description

****

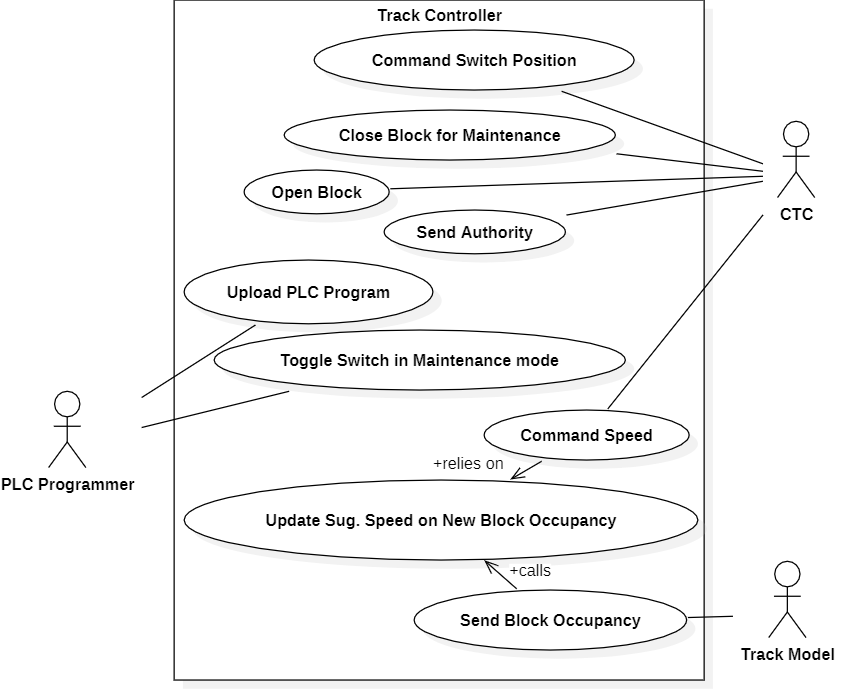
**Figures 6.2.2.9:** Update Suggested Speed on New Block Occupancy Use Case Sequence Diagram

* + 1. **Software Class Diagram**



**Figure 6.2.3:** Track Controller SW Class Diagram

* 1. **Track Controller HW**
     1. Use Case Diagram

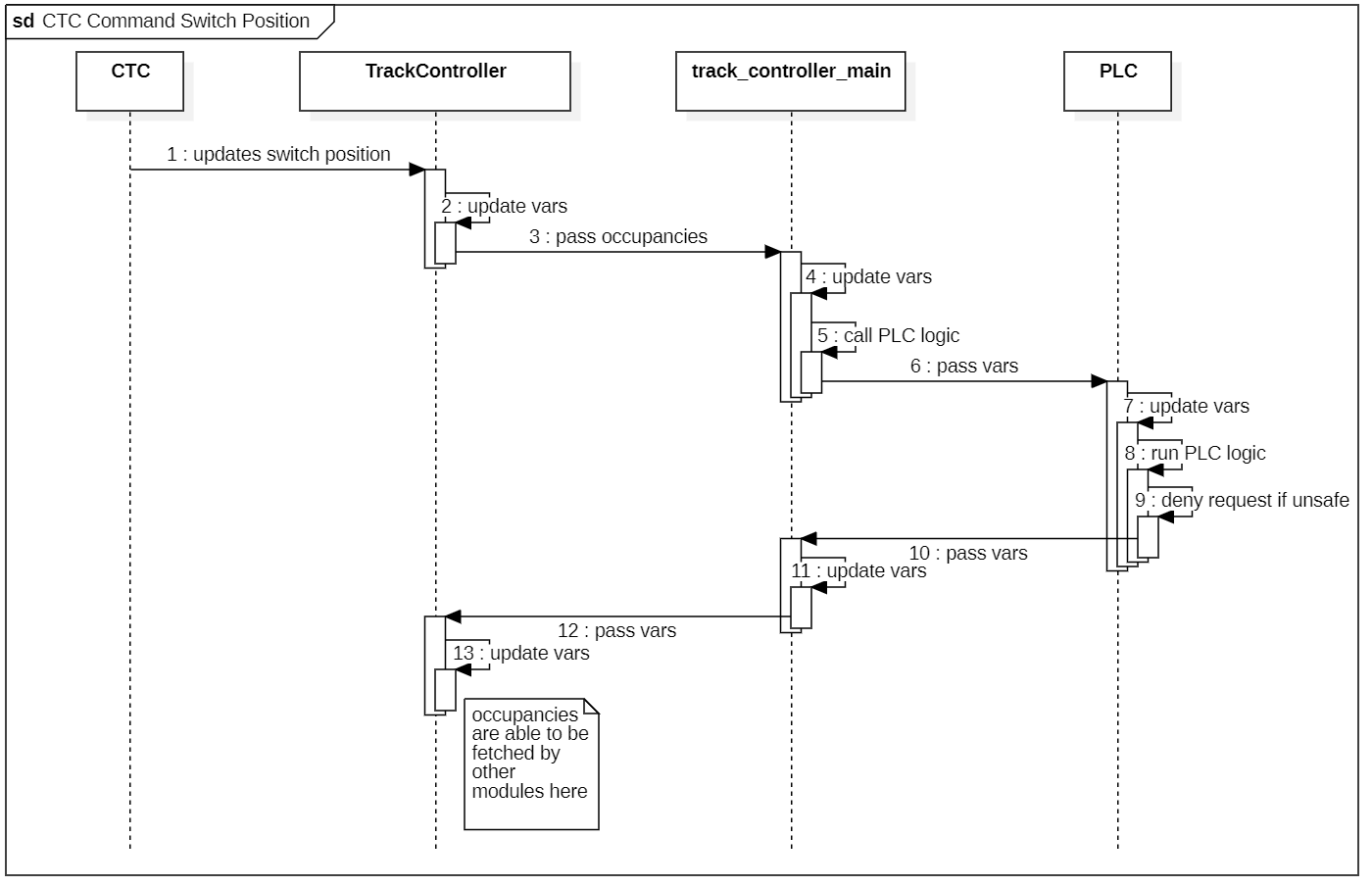


**Figure 6.3.1:** Track Controller HW Use Case Diagram

* + 1. **Use Cases**
       1. **CTC Command Switch Position**

| Table 6.3.2.1: CTC Command Switch Position | |
| --- | --- |
| Actors | CTC |
| Description | In order to properly route trains, the CTC commands switch positions to the track controller which triggers the following actions:   1. CTC indicates a switch to be toggled by Track Controller 2. Track Controller executes plc program to determine if the switch can be toggled    1. If it is not safe, do nothing 3. If the switch can be toggled:    1. Toggle switch and associated signals on the Track Model    2. Update the Track Controller UI |
| Data | Int - line id, int block index, Switch - switch object |
| Stimulus | CTC calls update\_switch on Track Controller |
| Response | The Track Controller will execute the plc program to determine if it is safe to toggle the switch, and if it is, the switch will be toggled on the Track Model. |

**Table 6.3.2.1:** CTC Command Switch Position Description

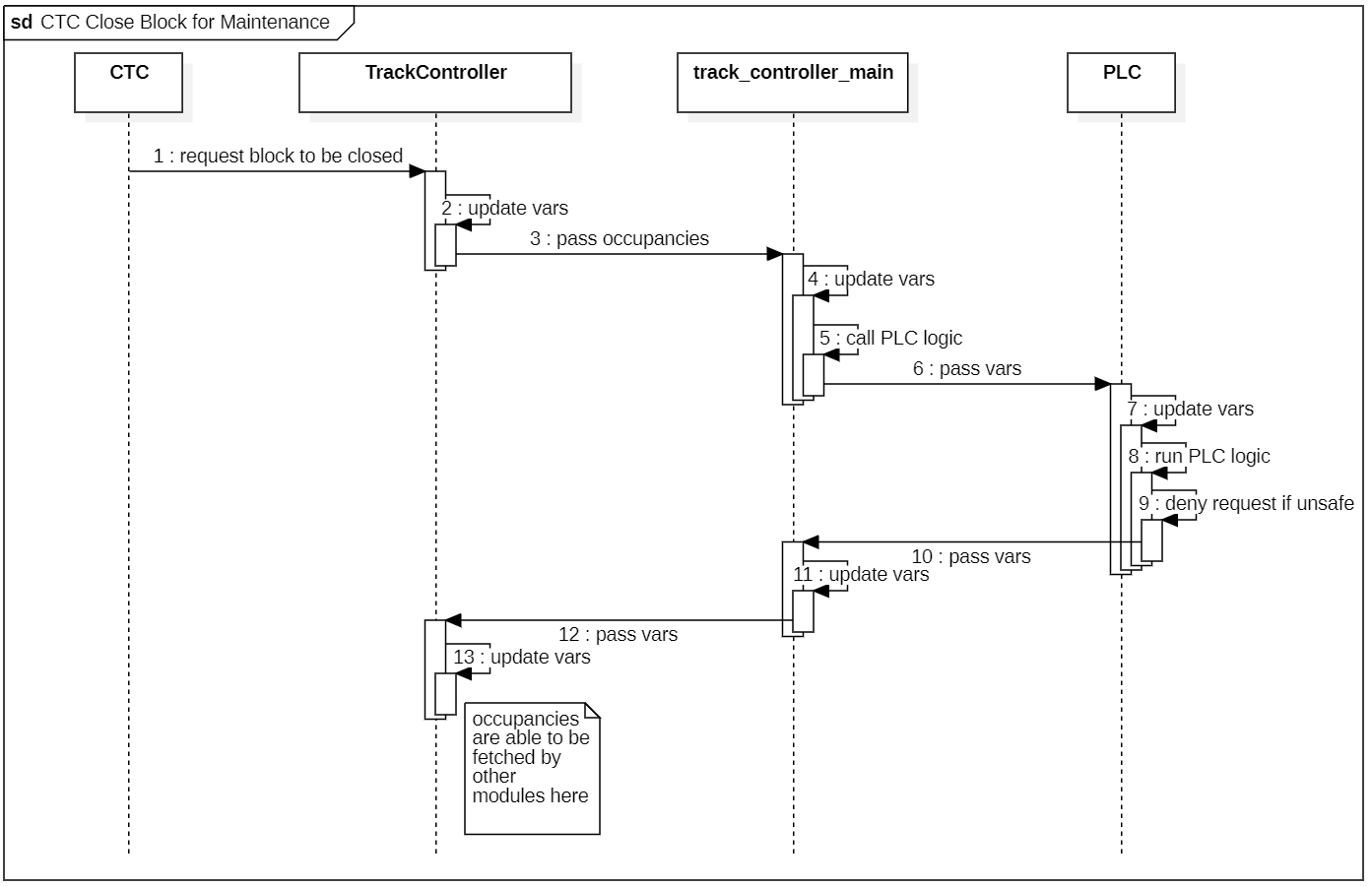


**Figure 6.3.2.1:** CTC Command Switch Position Sequence Diagram

* + - 1. **CTC Close Block for Maintenance**

| Table 6.3.2.2: CTC Close Block for Maintenance | |
| --- | --- |
| Actors | CTC |
| Description | In order to safely perform maintenance, the CTC must direct the closure of a block to the Track Controller which triggers the following actions:   1. CTC indicates a block to be closed to the Track Controller 2. The track controller executes the plc program to determine if this is a safe action    1. If it is not safe, do nothing 3. If it is safe to close the block, tell the Track Model to close the block |
| Data | Int line id, int block index, bool close |
| Stimulus | The CTC directs the Track Controller to close a block |
| Response | If it is safe to close the block, the track controller will close the block. Otherwise, nothing happens |

**Table 6.3.2.2:** CTC Close Block for Maintenance Description

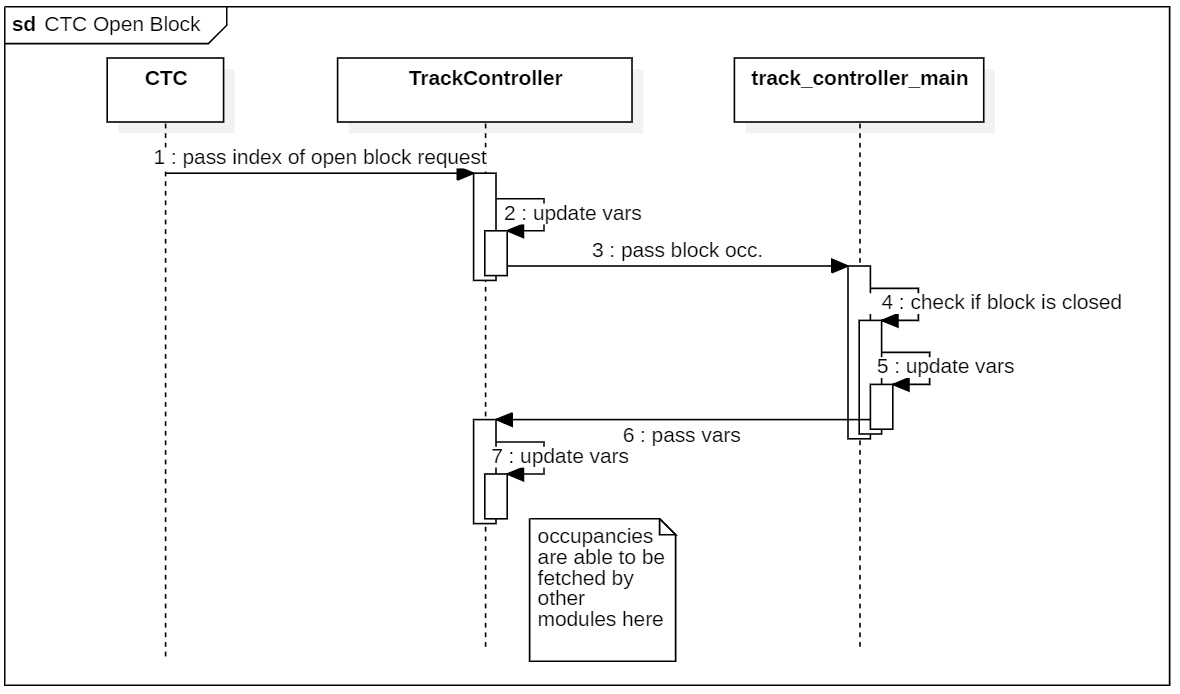


**Figure 6.3.2.2:** CTC Close Block for Maintenance Sequence Diagram

* + - 1. **CTC Open Block**

| Table 6.3.2.3: CTC Open Block | |
| --- | --- |
| Actors | CTC |
| Description | After block maintenance is completed, the CTC needs to re-open the block. This triggers the following actions:   1. The CTC tells the track controller to open the block 2. The track controller tells the track model to open the block  * This is always a safe action, so no further processing is required, the message is just passed along and the action is executed |
| Data | Int line id, int block index, bool open |
| Stimulus | The CTC tells the Track Controller to open a block |
| Response | The Track Controller opens the block on the Track Model |

**Table 6.3.2.3:** CTC Open Block Description

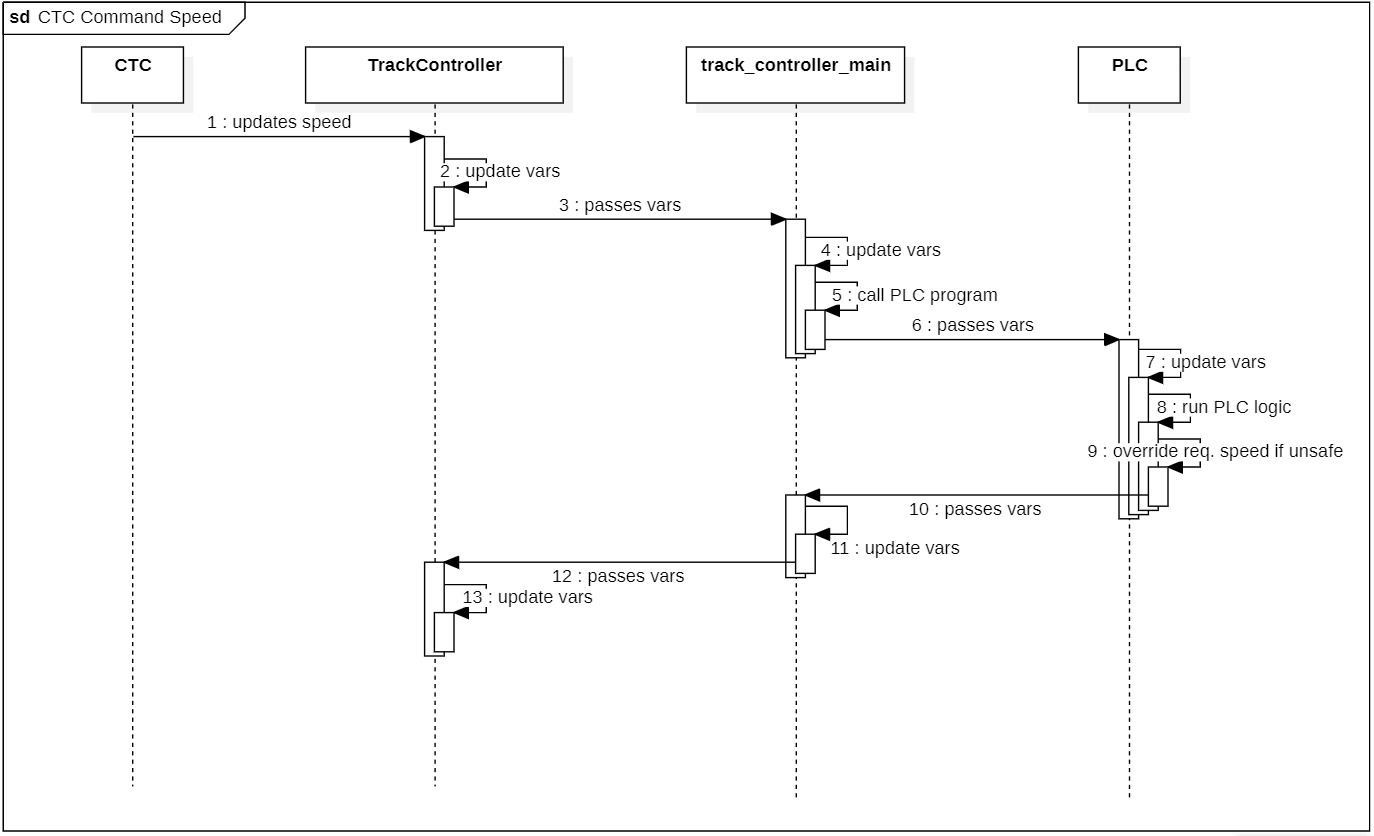


**Figure 6.3.2.3:** CTC Open Block Sequence Diagram

* + - 1. **CTC Command Speed**

| Table 6.3.2.4: CTC Command Speed | |
| --- | --- |
| Actors | CTC |
| Description | The CTC commands a speed for a train to travel throughout the system. This is sent on a block by block basis   1. The CTC commands a speed to a block to the Track Controller 2. The Track Controller, processing block occupancies from the Track Model, is continuously updating whether a train can safely travel into a block or not, sending a safe\_speed\_bool that is true if it’s safe to travel through the block or false if the train must not travel through the block 3. If safe\_speed\_bool is false, CTC commanded speed is not sent. Rather, 0 is sent by the Track Controller. 4. If safe\_speed\_bool is true, the CTC commanded speed for the block is sent. |
| Data | Bool: safe\_speed\_bool, int: line id, int: block id, float: speed, list: speed list |
| Stimulus | CTC Commands a speed, track controller receives new occupancy |
| Response | Safe speed is sent from Track controller to Track Model |

**Table 6.3.2.4:** CTC Command Speed Description

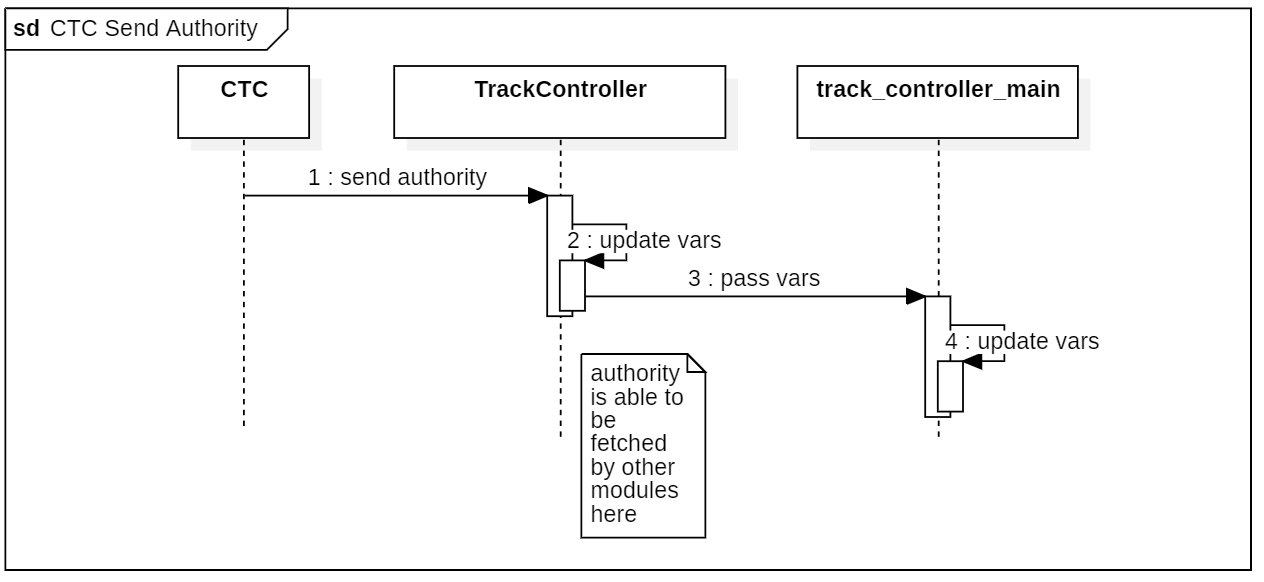


**Figure 6.3.2.4:** CTC Command Speed Sequence Diagram

* + - 1. **CTC Send Authority**

| Table 6.3.2.5: CTC Send Authority | |
| --- | --- |
| Actors | CTC |
| Description | The CTC updates authority on a block by block basis to ensure trains know where their next stop is. This is done through the following actions involving the CTC and Track Controller:   1. CTC sets authority for a certain block 2. The Track Controller passes the authority to the Track Model |
| Data | Int: line\_id, int: block\_id, int: authority, list: authority\_list |
| Stimulus | CTC sends authority for a block to Track Controller |
| Response | Track Controller updates a list containing current authorities and sends it to Track Model |

**Table 6.3.2.5:** CTC Send Authority Description

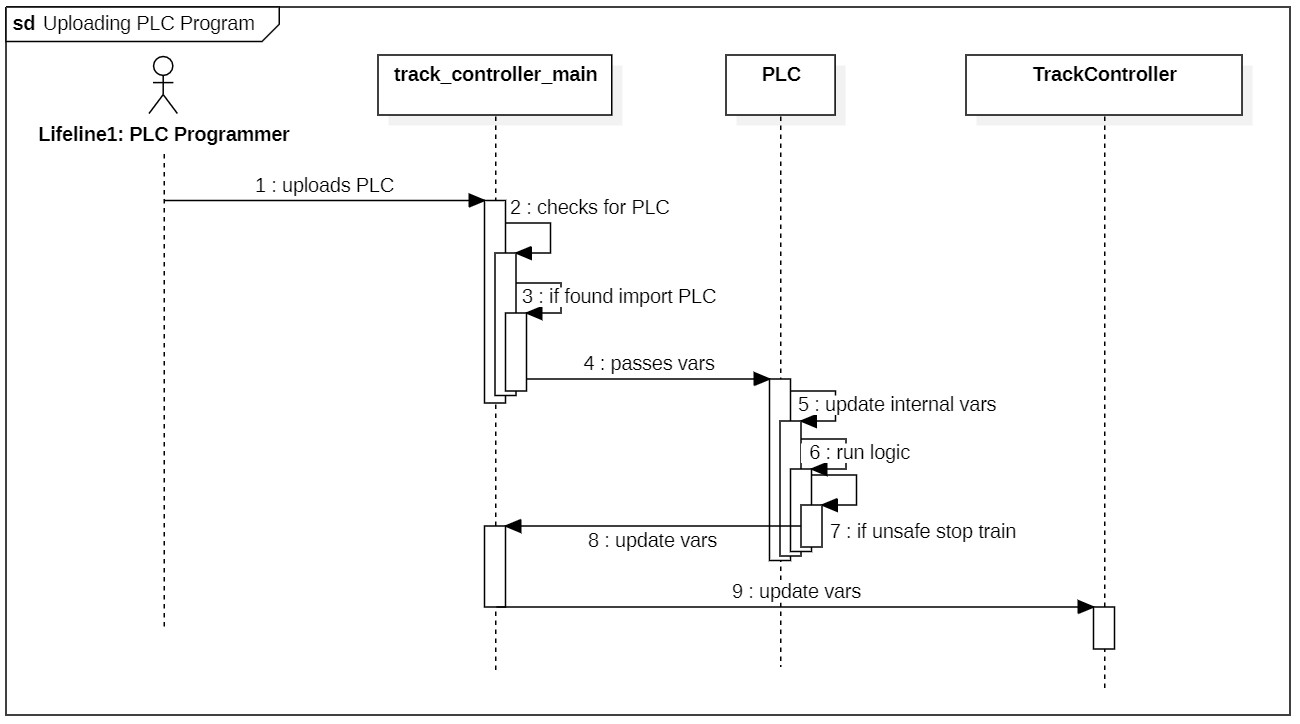


**Figure 6.3.2.5:** CTC Send Authority Sequence Diagram

* + - 1. **PLC Programmer Upload PLC Program**

| Table 6.3.2.6: Upload PLC Program | |
| --- | --- |
| Actors | PLC Programmer |
| Description | The PLC programmer must upload a PLC program at each instance of a wayside controller. This is done through the following actions:   1. The plc programmer selects a new file through the UI 2. The filepath of the selected file is used as the PLC program |
| Data | Str: filepath |
| Stimulus | The PLC Program selects a file from the file browser after clicking the upload file button |
| Response | The PLC program loads the plc file from the selected file’s filepath |

**Table 6.3.2.6:** Upload PLC Program Description

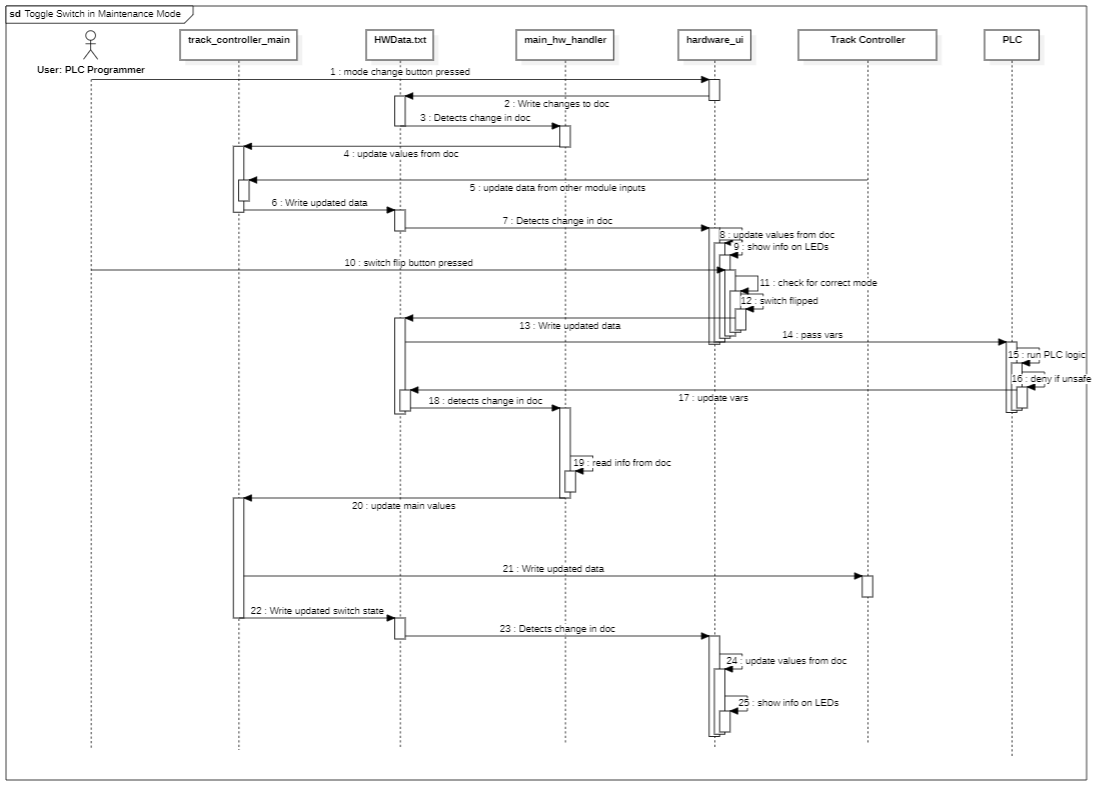


**Figure 6.3.2.6:** Upload PLC Program Sequence Diagram

* + - 1. **PLC Programmer Toggle Switch in Maintenance Mode**

| Table 6.3.2.7: Toggle Switch in Maintenance Mode | |
| --- | --- |
| Actors | PLC Programmer |
| Description | The PLC Programmer wishes to manually a toggle a switch through a maintenance mode dialogue by way of the following actions:   1. PLC Programmer selects maintenance mode 2. A dialogue appears with buttons for each switch that allow for a toggle 3. The programmer selects a switch to toggle 4. The PLC program determines if it is safe to toggle the switch based on block occupancy    1. If it is not safe, do nothing 5. If safe, track model is told to update switch and light statuses |
| Data | Int : switch\_index, list: block\_occupancy\_list, int: block 1 signal, int: block 2 signal, bool: switch\_permitted |
| Stimulus | PLC programmer selects a switch to toggle |
| Response | Track Controller either does or does not toggle the switch based on whether it is a safe action |

**Table 6.3.2.7:** Toggle Switch in Maintenance Mode Description

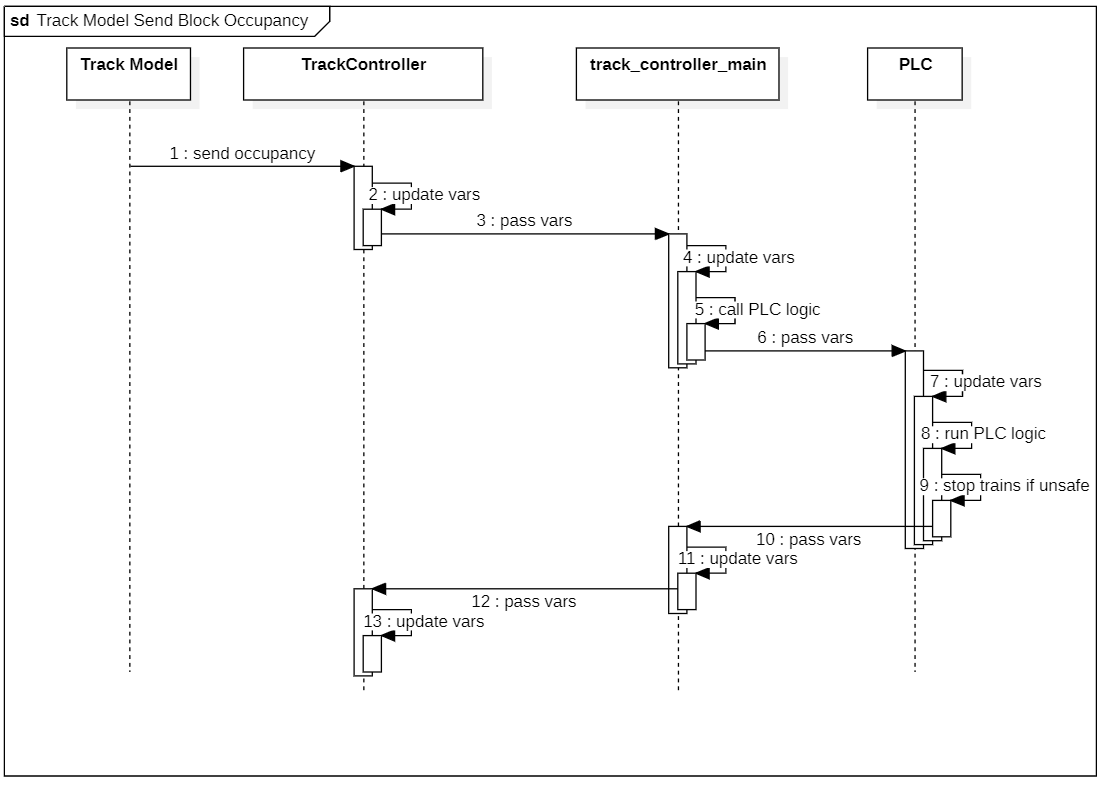


**Figure 6.3.2.7:** Toggle Switch in Maintenance Mode Sequence Diagram

* + - 1. **Track Model Send Block Occupancy**

| Table 6.3.2.8: Track Model Send Block occupancy | |
| --- | --- |
| Actors | Track Model |
| Description | The Track Model sends a block occupancy to the Track Controller everytime a new block is occupied. This triggers the following actions:   1. The track controller sends the new occupancy list to its PLC program to be executed, determining which blocks are safe to be entered 2. The UI is updated with the new occupancy 3. The CTC is updated with the new block occupancies |
| Data | List: occupancy\_list, int line id |
| Stimulus | Track Model sends new block occupancy to Track Controller |
| Response | Track Controller executes PLC program, updates UI, and sends new occupancy to CTC |

**Table 6.3.2.8:** Track Model Send Block occupancy Description

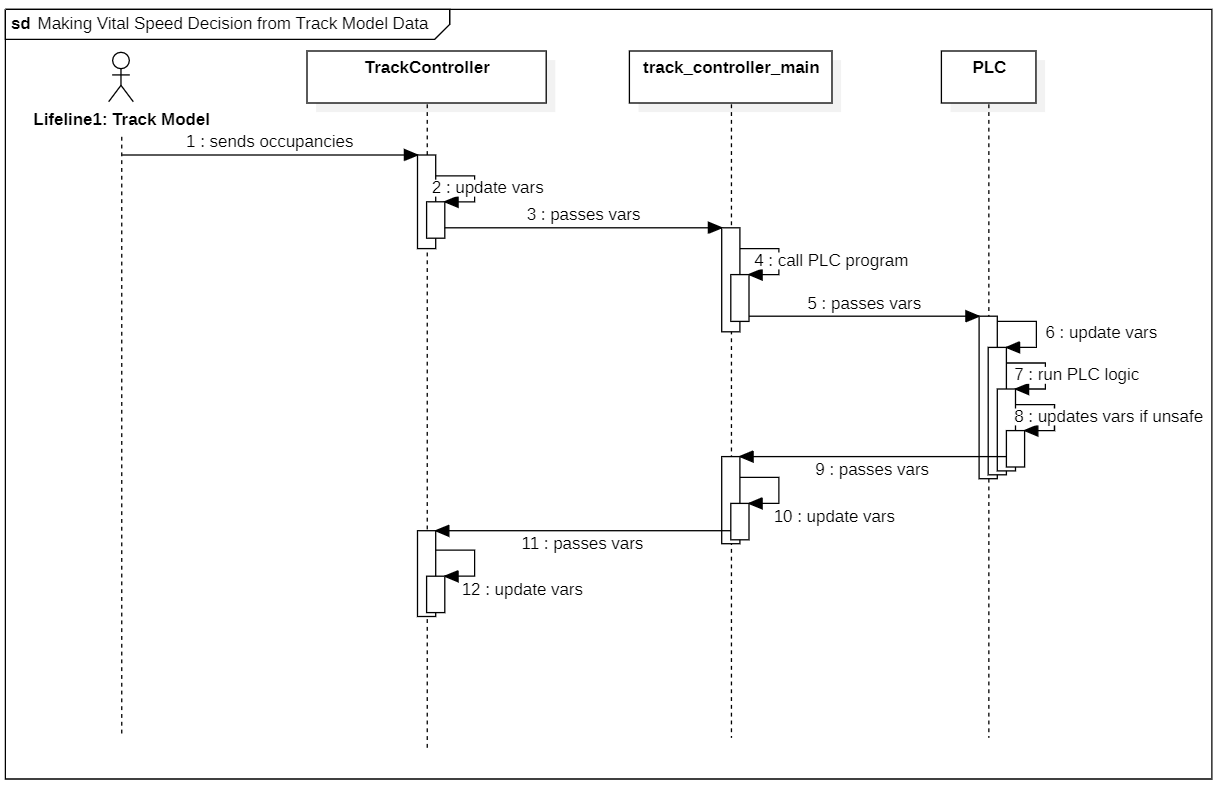


**Figure 6.3.2.8:** Track Model Send Block occupancy Sequence Diagram

* + - 1. **Update Suggested Speed on New Occupancy**

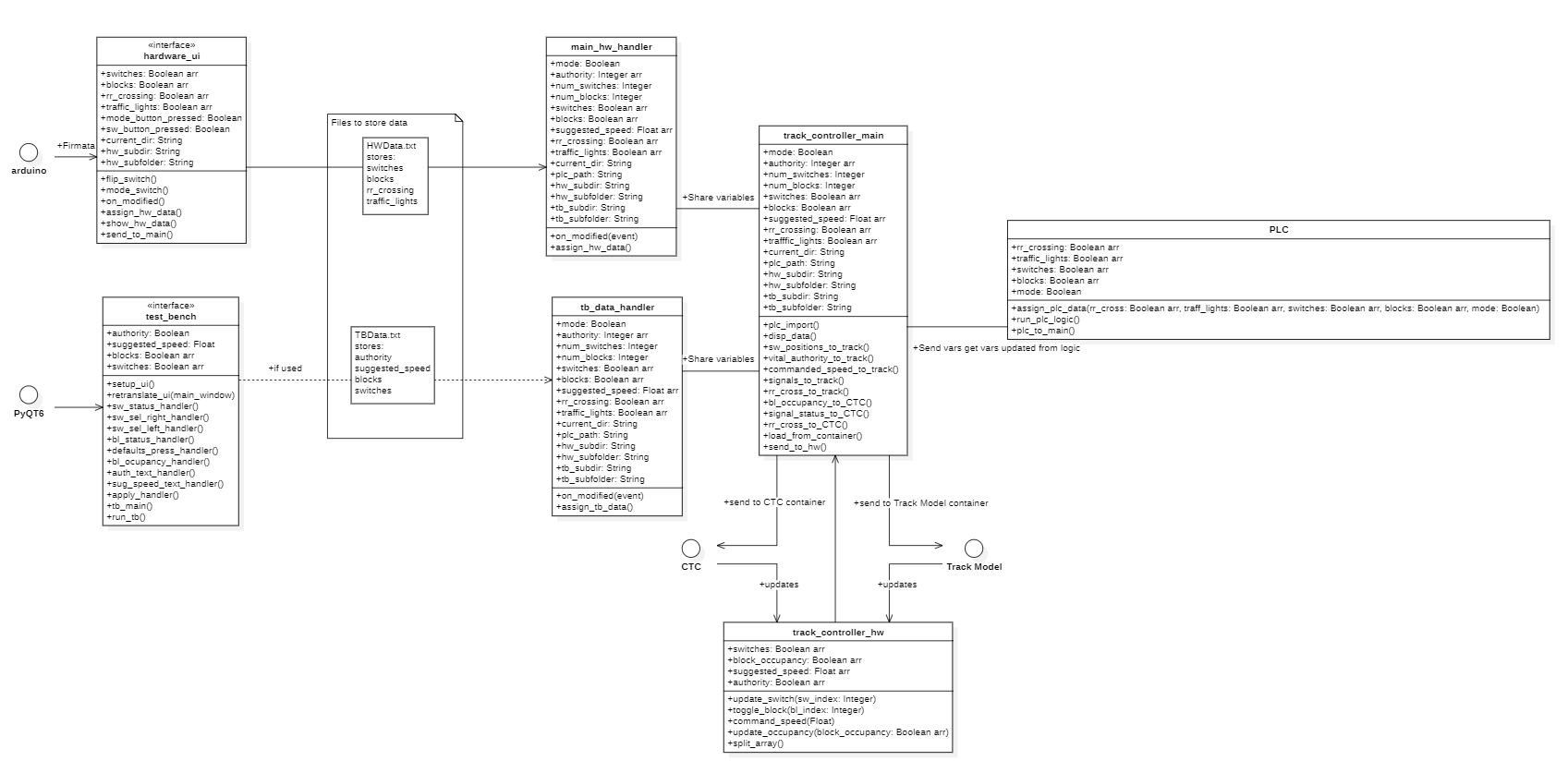
| Table 6.3.2.9: Update Suggested Speed on New Block Occupancy | |
| --- | --- |
| Actors | [internal] |
| Description | Any time block occupancy is updated, the PLC must re-evaluate what blocks are safe to travel through. This is relied on for several internal use cases including any switch toggle or speed command |
| Data | Block\_occupancy\_list, |
| Stimulus | Block occupancy is updated |
| Response | PLC program determines safe travel |

**Table 6.3.2.9:** Update Suggested Speed on New Block Occupancy Description

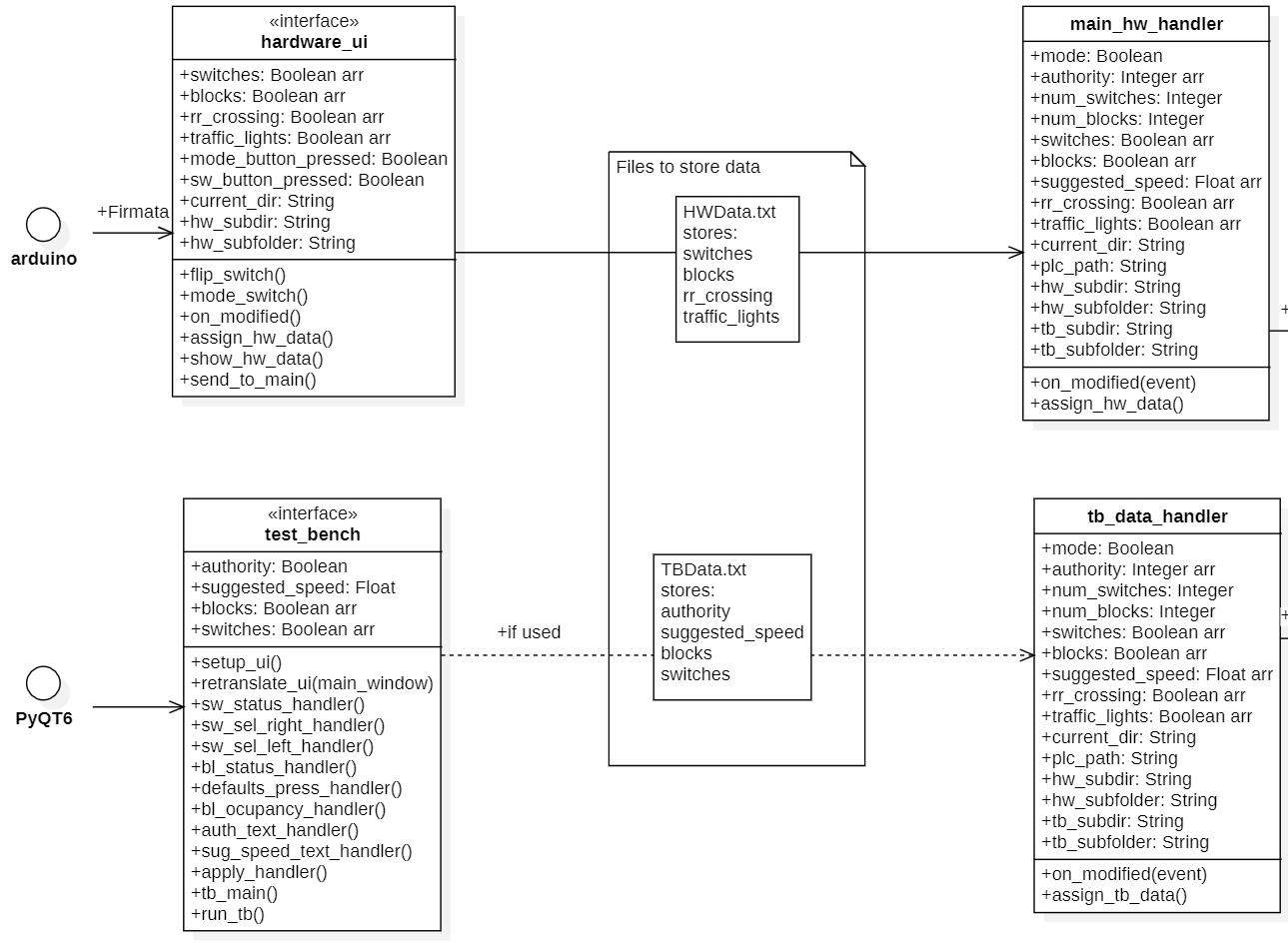


**Figure 6.3.2.9:** Update Suggested Speed on New Block Occupancy Sequence Diagram

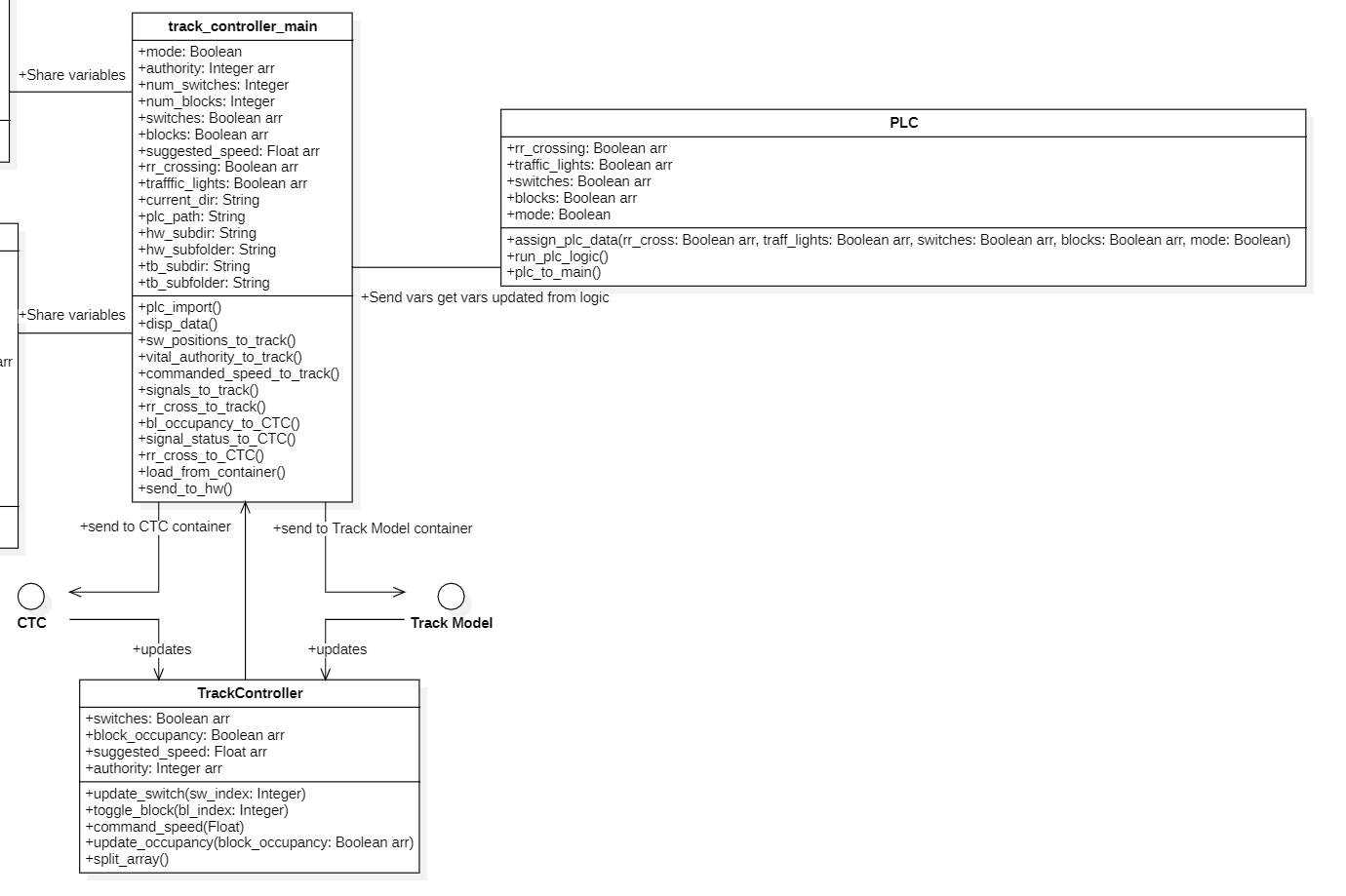
* + 1. **Software Class Diagram**



**Figure 6.3.3.1:** Track Controller HW Class Diagram Zoomed Out View

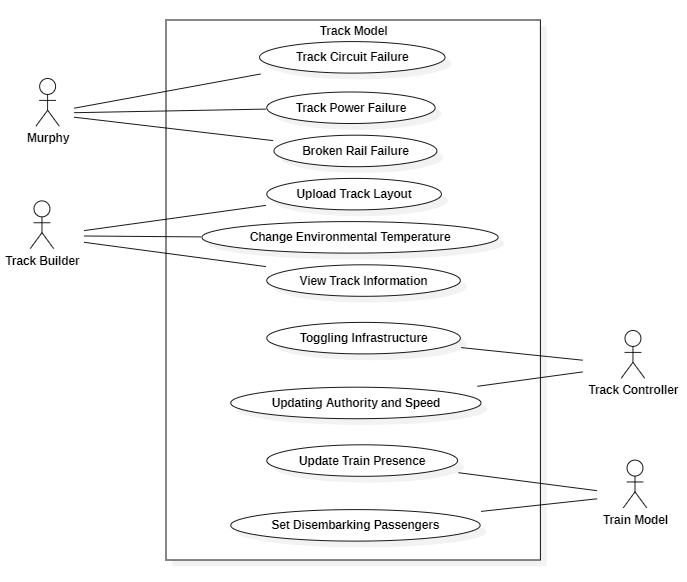


**Figure 6.3.3.2:** Track Controller HW Class Diagram Zoomed in View 1

****

**Figure 6.3.3.3:** Track Controller HW Class Diagram Zoomed in View 2

* 1. **Track Model**
     1. **Use Case Diagram**



**Figure 6.4.1:** Track Model Use Case Diagram

* + 1. **Use Cases**
       1. **Track Circuit Failure**

| Table 6.4.2.1: Track Circuit Failure | |
| --- | --- |
| Actors | Murphy |
| Description | 1. Murphy selects a block  2. Murphy toggles the failure value  3. Track model block occupancy attribute is toggled  4. Changes are shown on map and table |
| Data | Numpy Array element – Block (self.occupancy) |
| Stimulus | Any time a failure toggle-button is pressed |
| Response | The Track Model will update its block occupancy and this will be shown on the UI table and map |

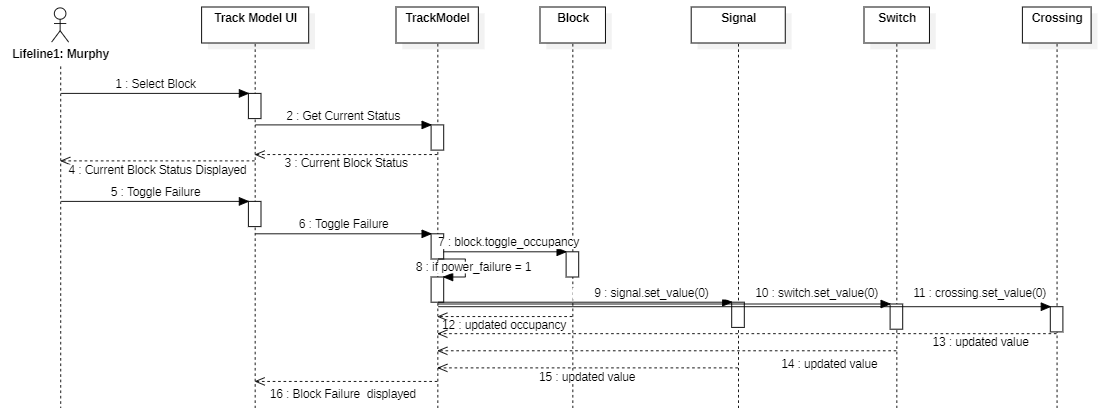
**Table 6.4.2.1:** Testing Track Failures Use Case Description

**Figure 6.4.2.1:** Track Circuit Failure Sequence Diagram

* + - 1. **Track Power Failure**

| Table 6.4.2.2: Track Power Failure | |
| --- | --- |
| Actors | Murphy |
| Description | 1. Murphy selects a failure mode to test (broken rail, track circuit failure, power failure)  2. Murphy selects a block  3. Murphy toggles the failure value  4. Track model block occupancy attribute is toggled  5. Infrastructure on blocks go to defaults (signals red, switch left, crossings activated)  6. Changes are shown on map and table |
| Data | Numpy Array element – Block (self.occupancy)  Numpy Array element – Signal (self.value)  Numpy Array element – Switch (self.value)  Numpy Array element – Crossing (self.value) |
| Stimulus | Any time a failure toggle-button is pressed |
| Response | The Track Model will update its block occupancy and infrastructure values, this will be shown on the UI table and map |

**Table 6.4.2.2:** Track Power Failure Use Case Description

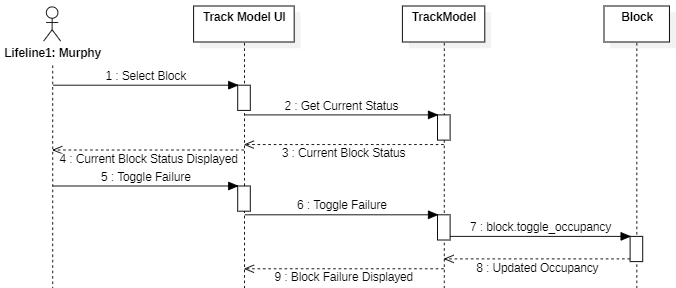
****

**Figure 6.4.2.2:** Track Power Failure Sequence Diagram

* + - 1. **Broken Rail Failure**

| Table 6.4.2.3: Broken Rail Failure | |
| --- | --- |
| Actors | Murphy |
| Description | 1. Murphy selects a block  2. Murphy toggles the failure value  3. Track model block occupancy attribute is toggled  4. If it is a power failure, infrastructure on blocks go to defaults  5. Changes are shown on map and table |
| Data | Numpy Array element – Block (self.occupancy) |
| Stimulus | Any time a failure toggle-button is pressed |
| Response | The Track Model will update its block occupancy and this will be shown on the UI table and map |

**Table 6.4.2.3:** Broken Rail Failure Use Case Description

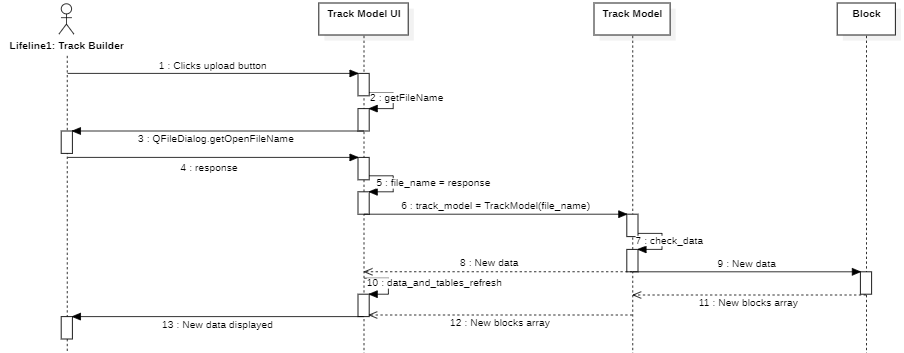


**Table 6.4.2.3:** Broken Rail Failure Use Case Description

* + - 1. **Loading Track Layouts**

| Table 6.4.2.4: Loading Track Layouts | |
| --- | --- |
| Actors | Track Builder |
| Description | 1. Open-file dialog opens  2. Track Builder selects an Excel file  3. A new Track Model Object is initialized with the data from the Excel |
| Data | Track Model Object |
| Stimulus | Any time the upload button is pressed |
| Response | The Track Model will update its block and infrastructure data and this will be shown on the UI table and map |

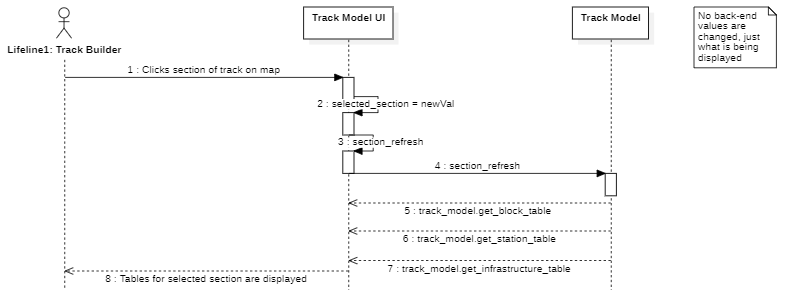
**Table 6.4.2.4:** Loading Track Layouts Description Use Case Description

**Figure 6.4.2.4:** Loading Track Layouts Sequence Diagram

* + - 1. **Viewing Track Information**

| Table 6.4.2.5: Viewing Track Information | |
| --- | --- |
| Actors | Track Builder |
| Description | 1. Track Builder selects a section of the track on the dynamic map  2. The tables update to display the currently-selected section information |
| Data | Track Model Object |
| Stimulus | Any time a new section is selected |
| Response | UI tables will grab most recent data from the Track Model Object |

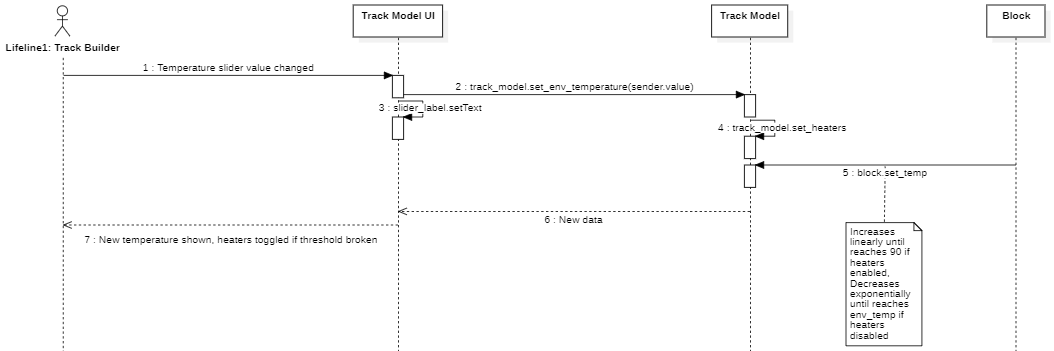
**Table 6.4.2.5:** Viewing Track Information Use Case Description



**Figure 6.4.2.5:** Viewing Track Information Sequence Diagram

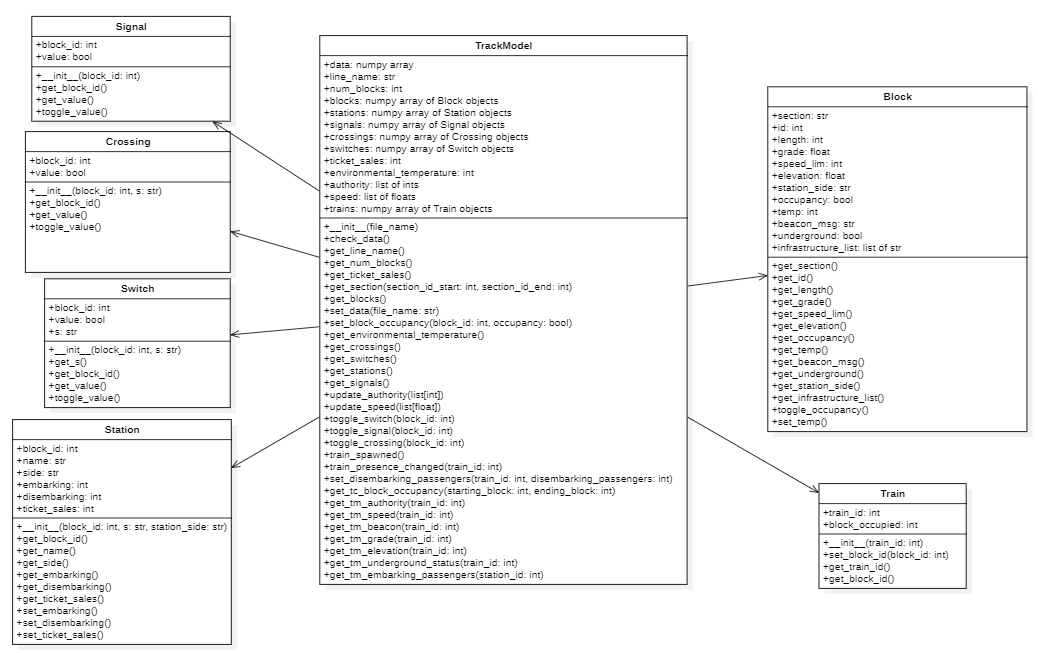
* + - 1. **Changing Track Temperature**

| Table 6.4.2.6: Changing Track Temperature | |
| --- | --- |
| Actors | Track Builder |
| Description | 1. Track Builder uses slider to change environmental temperature  2. New temperature is displayed in table and map  3. Track heaters are automatically enabled/disabled based on threshold |
| Data | Track Model Object |
| Stimulus | Any time the temperature slider is moved |
| Response | UI tables will grab most recent data from the Track Model Object |

**Table 6.4.2.6:** Changing Track Temperature Use Case Description

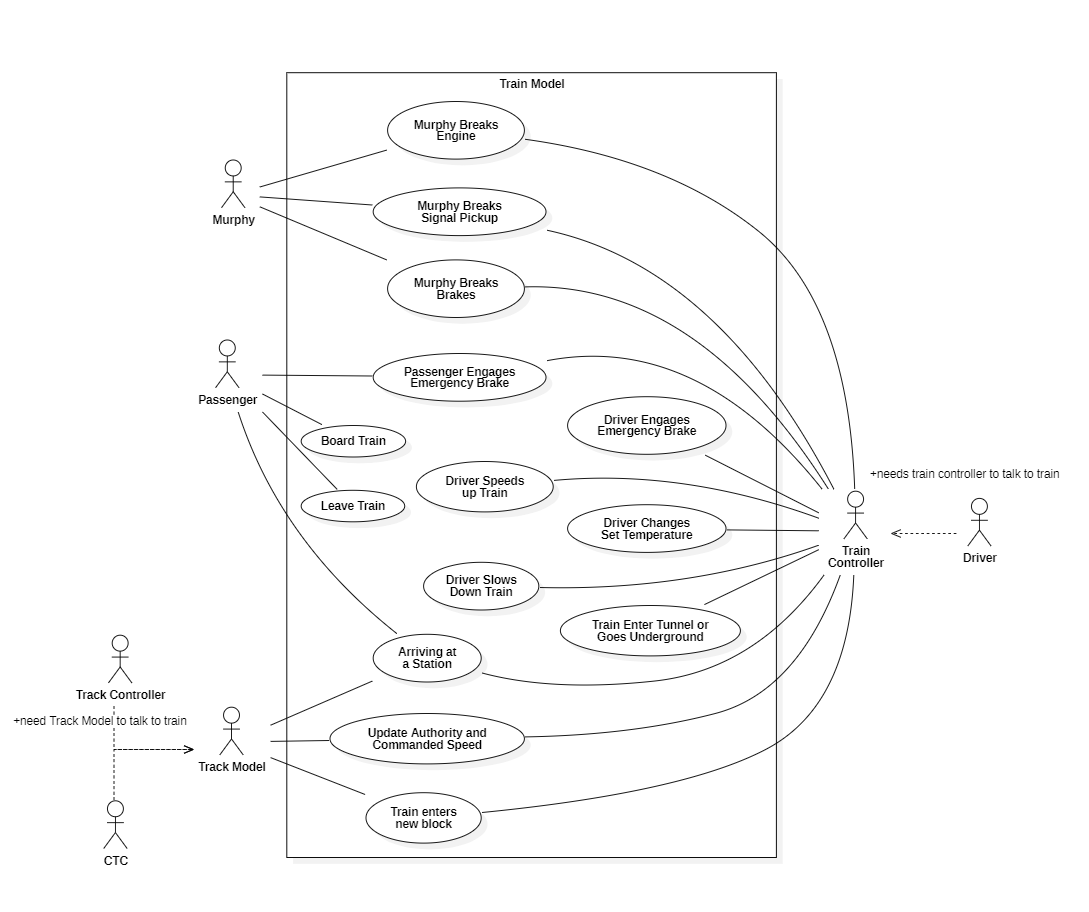
**Figure 6.4.2.6:** Changing Track Temperature Sequence Diagram

* + 1. **Software Class Diagram:**



**Figure 6.4.6:** Track Model Class Diagram

* 1. **Train Model**
     1. Use case Diagram:

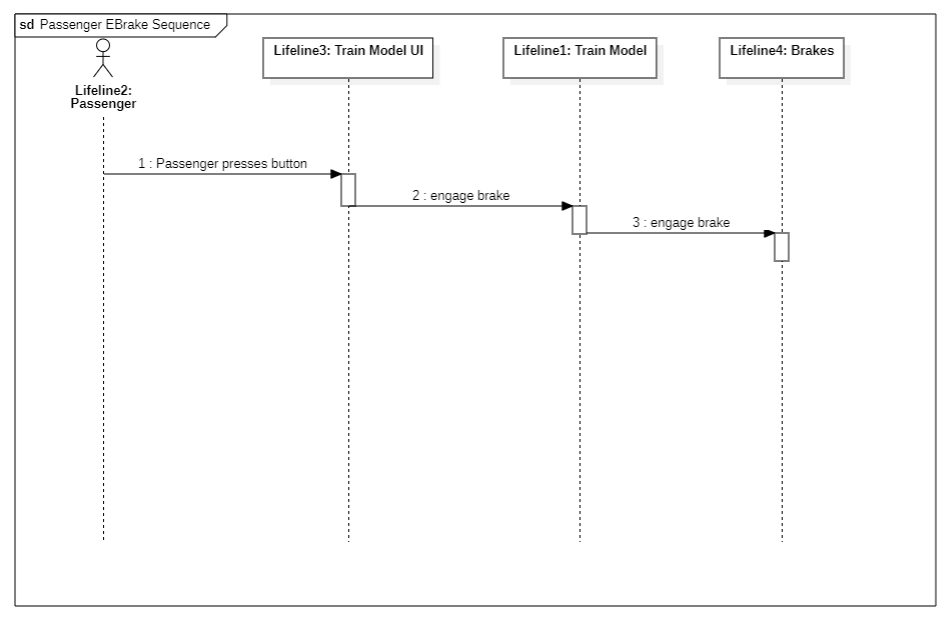


**Table 6.5.1:** Train Model Use Case Diagram

* + 1. **Use Cases**
       1. **Passenger engages emergency brake**

| Table 6.5.2.1:Passenger engages emergency brake Use Case Description | |
| --- | --- |
| Actors | Passenger |
| Description | 1. Passenger decides to engage the emergency brake. 2. Passenger presses the emergency brake button. 3. The Train Model applies the correct force to slow down the train. |
| Data | Bool - passenger emergency brake |
| Stimulus | Whether or not the emergency brake button for the passenger has been pressed |
| Response | If the button is pressed: enable emergency brake  Else: do nothing |

**Table 6.5.2.1:** Passenger engages emergency brake Description



**Figure 6.5.2.1:** Passenger engages emergency brake Sequence Diagram

* + - 1. **Murphy breaks the engine**

| Table 6.5.2.2:Murphy breaks the engine Use Case Description | |
| --- | --- |
| Actors | Murphy, Train Controller |
| Description | 1. Murphy decides to brake the engine 2. Murphy brakes the engine 3. The Train Model communicates the engine failure to the Train Controller 4. The Train Controller engages the emergency brake 5. The Train Model applies the correct force to slow down the train. |
| Data | Bool - engine failure  Bool - driver emergency brake |
| Stimulus | Murphy checking/unchecking the engine fail checkbox |
| Response | Set engine fail according to engine fail checkbox state  If the failure has occurred engage emergency brake |

**Table 6.5.2.2:** Murphy breaks the engine Description

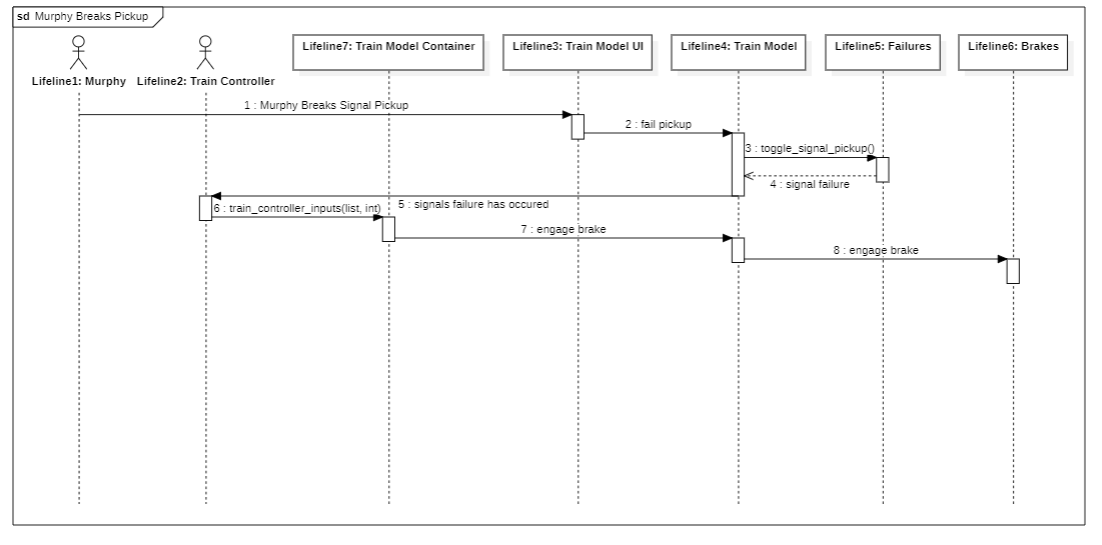


**Figure 6.5.2.2:** Murphy breaks the engine Sequence Diagram

* + - 1. **Murphy breaks the signal pickup**

| Table 6.5.2.3:Murphy breaks the signal pickup Use Case Description | |
| --- | --- |
| Actors | Murphy, Train Controller |
| Description | 1. Murphy decides to break the signal pickup 2. Murphy break the signal pickup 3. The Train Model communicates the signal pickup failure to the Train Controller 4. The Train Controller engages the emergency brake 5. The Train Model applies the correct force to slow down the train. |
| Data | Bool - signal pickup failure  Bool - driver emergency brake |
| Stimulus | Murphy checking/unchecking the signal pickup fail checkbox |
| Response | Set engine fail according to signal pickup fail checkbox state  If the failure has occurred engage emergency brake |

**Table 6.5.2.3:** Murphy breaks the signal pickup Description

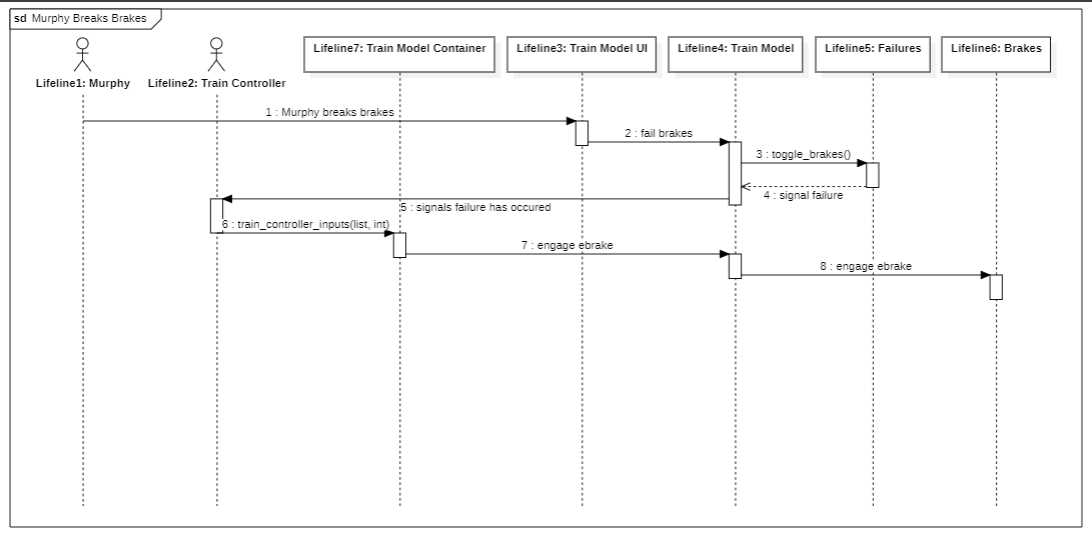


**Figure 6.5.2.3:** Murphy breaks the signal pickup Sequence Diagram

* + - 1. **Murphy breaks the brakes**

| Table 6.5.2.4: Murphy breaks the brakes Use Case Description | |
| --- | --- |
| Actors | Murphy, Train Controller |
| Description | 1. Murphy decides to break the brakes 2. Murphy breaks the brakes 3. The Train Model communicates the brake failure to the Train Controller 4. The Train Controller engages the emergency brake 5. The Train Model applies the correct force to slow down the train. |
| Data | Bool - brake failure  Bool - driver emergency brake |
| Stimulus | Murphy checking/unchecking the brake fail checkbox |
| Response | Set engine fail according to brake fail checkbox state  If the failure has occurred engage emergency brake |

**Table 6.5.2.4:** Murphy breaks the brakes Description

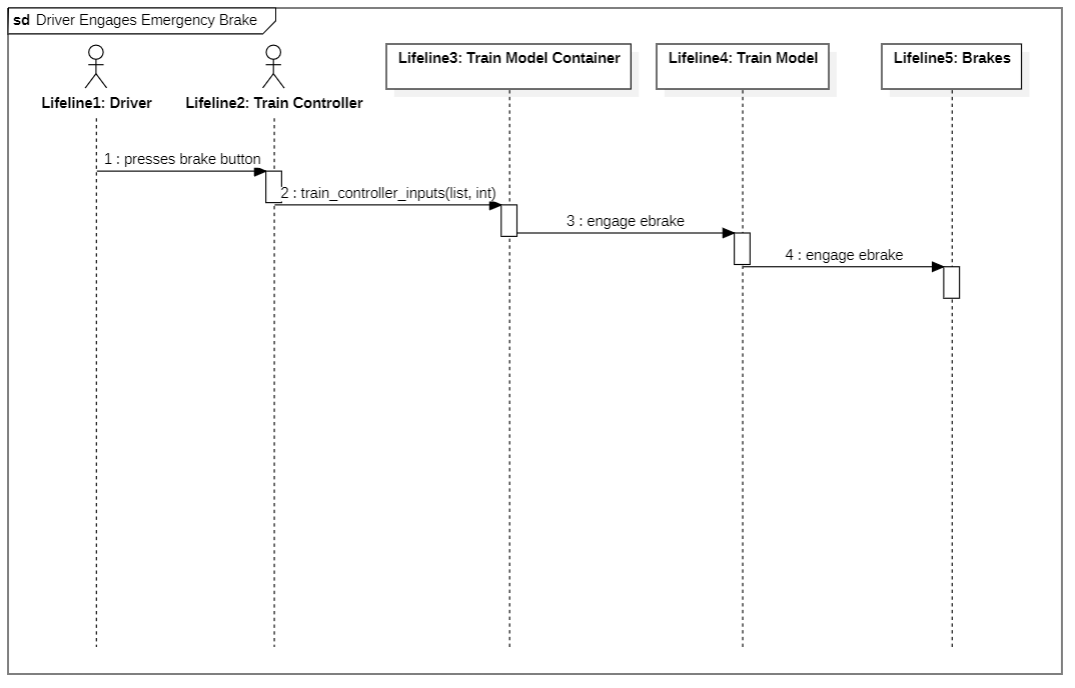


**Figure 6.5.2.4:** Murphy breaks the brakes Sequence Diagram

* + - 1. **Driver engages emergency brake**

| Table 6.5.2.5: Driver engages emergency brake Use Case Description | |
| --- | --- |
| Actors | Driver, Train Controller |
| Description | 1. Driver decides to engage the emergency brake. 2. Driver presses the emergency brake button. 3. Train controller communicates the activation of the emergency brake to the train model. 4. The Train Model applies the correct force to slow down the train. |
| Data | Bool - driver emergency brake |
| Stimulus | Driver presses emergency brake button on train controller |
| Response | Engage emergency brake |

**Table 6.5.2.5:** Driver engages emergency brake Description

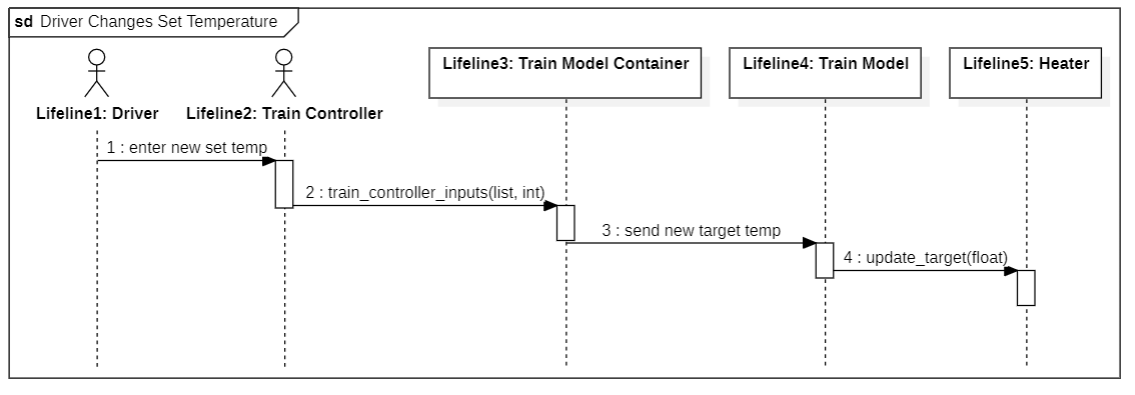


**Figure 6.5.2.5:** Driver engages emergency brake Sequence Diagram

* + - 1. **Driver changes set temperature**

| Table 6.5.2.6: Driver changes set temperature Use Case Description | |
| --- | --- |
| Actors | Driver, Train Controller |
| Description | 1. Driver changes the the target temperature 2. Train Controller sends new target temperature to Train Controller 3. Train model changes the initial temperature, target temperature, and time constant. 4. Train changes temperature according to the time that has passed |
| Data | Float - initial temperature  Float - target temperature  Float - time since target was set  Float - time constant |
| Stimulus | Driver sets new temp on train controller |
| Response | Update target temperature to the temp input by the driver, update initial\_temp to the current temp at time of updating, update time since target was set to 0, update time constant to appropriate value for first order differential decay |

**Table 6.5.2.6:** Driver changes set temperature Description

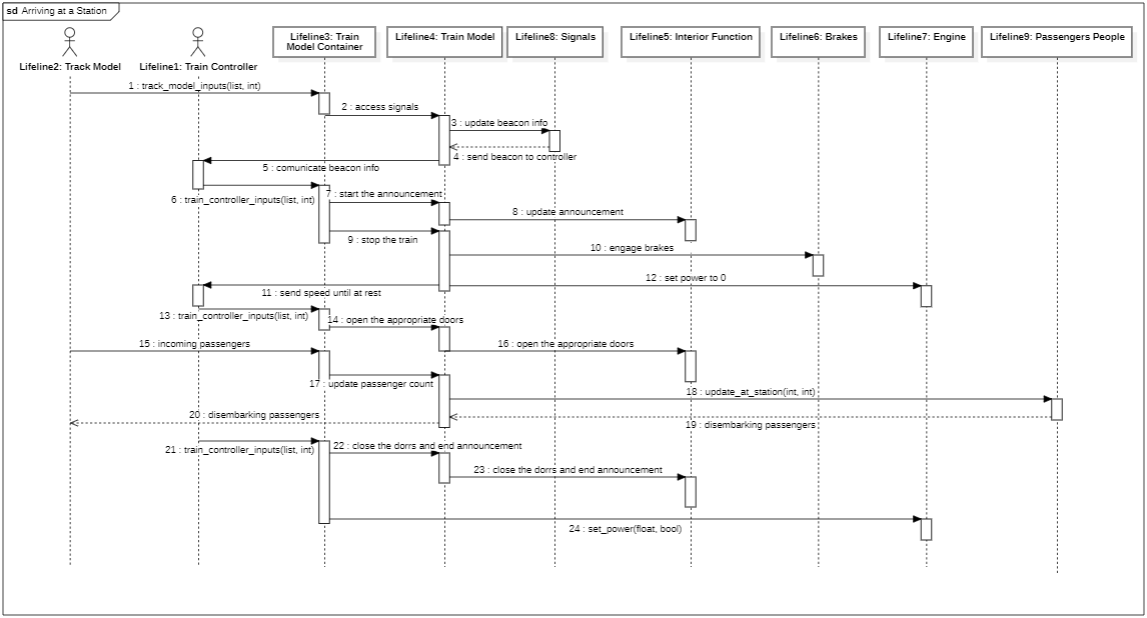


**Figure 6.5.2.6:** Driver changes set temperature Sequence Diagram

* + - 1. **Arriving at a station**

| Table 6.5.2.7: Arriving at a station Use Case Description | |
| --- | --- |
| Actors | Track Model, Train Controller, Passenger |
| Description | 1. Train receives information from the beacon and passes it to the train controller 2. Train controller determines whether the doors should open on the left, right, or both sides of the train and sends a signal to the train model to begin the announcement for this station 3. The train controller engages the service brake to stop at the station 4. Once at rest the train controller sends a signal to the train model to open the doors and the 5. Some passengers leave the train and some board the train 6. track model sends the amount of incoming passengers to the train model 7. The train model opens the doors and sends the amount of disembarking passengers to the track model 8. After some time the train controller sends the signal to close the doors and end the announcement 9. The train model closes the doors 10. The train controller increases the power to the engine 11. The train model performs the proper calculations and begins to increase speed |
| Data | Bool - left\_doors  Bool - right\_doors  String - announcement Bool - service\_brake  Float - power |
| Stimulus | Receiving beacon information from the track |
| Response | Slow down to stop at station and start announcement  Open doors at station  Close doors, end announcement, speed up |

**Table 6.5.2.7:** Arriving at a station Description



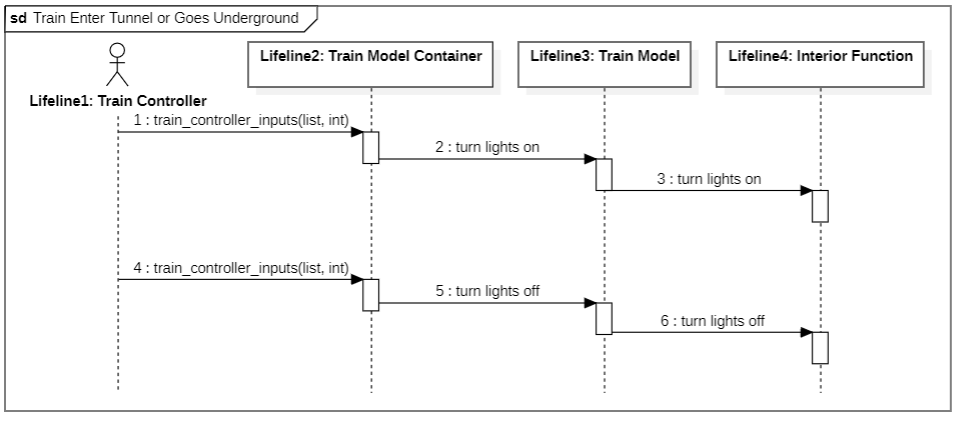
**Figure 6.5.2.7:** Arriving at a station Sequence Diagram

Note: the use cases Board Train and Leave Train are done entirely within this use case

* + - 1. **The train enters a tunnel/goes underground**

| Table 6.5.2.8: The train enters a tunnel/goes underground Use Case Description | |
| --- | --- |
| Actors | Train Controller |
| Description | 1. The train controller determines the current block is in a tunnel/is underground 2. The train controller send the signal to turn the headlights on 3. The train model turns on the headlights. 4. After some time the train controller determines the current block is not in a tunnel/underground |
| Data | Bool - exterior\_lights |
| Stimulus | Train controller determines block is in tunnel/is underground |
| Response | Turn on exterior lights |

**Table 6.5.2.8:** The train enters a tunnel/goes underground Description

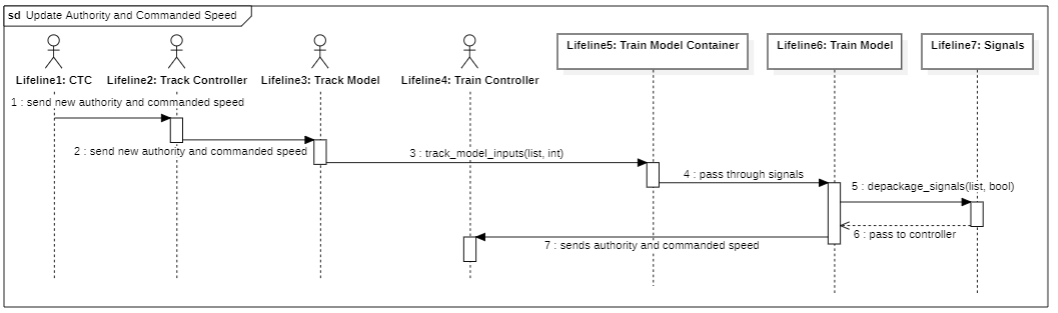


**Figure 6.5.2.8:** The train enters a tunnel/goes underground Sequence Diagram

* + - 1. **Update Authority/Commanded Speed**

| Table 6.5.2.9: Update Authority/Commanded Speed Use Case Description | |
| --- | --- |
| Actors | CTC, Track Controller, Track, Model, Train Controller |
| Description | 1. The CTC determines a new authority and commanded speed for a given instance of the train model 2. The CTC sends a signal to a track controller where the train model currently is to update the authority of specific blocks. 3. The track controller sends a signal to the track model to update the signal to the train model of specific blocks 4. The track model sends the authority and commanded speed to the train model 5. The train model passes the new authority and commanded speed to the train controller |
| Data | Int - commanded speed  Int - authority |
| Stimulus | Change in data received through the train signal pickup antenna |
| Response | Pass the incoming signals to the train controller |

**Table 6.5.2.9:** Update Authority/Commanded Speed Description

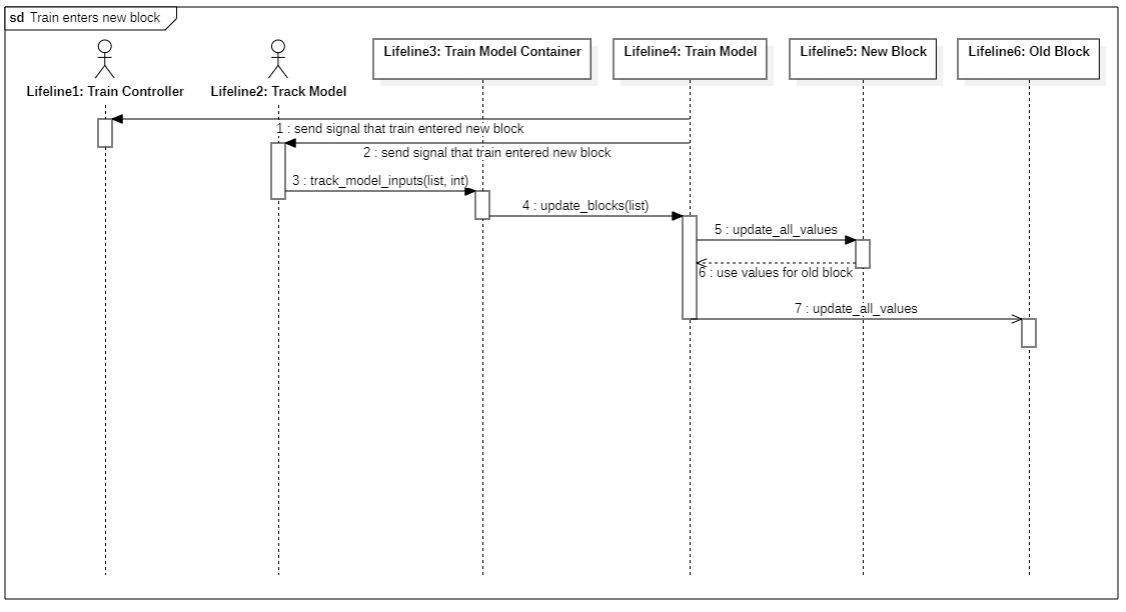


**Figure 6.5.2.9:** Update Authority/Commanded Speed Sequence Diagram

* + - 1. **Train enters new block**

| Table 6.5.2.10: Train enters new block Use Case Description | |
| --- | --- |
| Actors | Track Model, Train Controller |
| Description | 1. As part of the train models physics calculations it calculates how far along a block it is 2. When the train model calculates it has crossed into a new block it sends a signal to the train controller and the track model 3. The track model sends information about the new block to the train model and updates signal to the train model accordingly 4. The train model updates block information and incoming signals accordingly and applies related information to physics calculations. |
| Data | Float - Block Length  Float - Grade  Float - Elevation |
| Stimulus | Train crosses into new block |
| Response | Signal to the train controller and track model that a new block has been entered. Update the information of the block the train is occupying. |

**Table 6.5.2.10:** Train enters new block Description

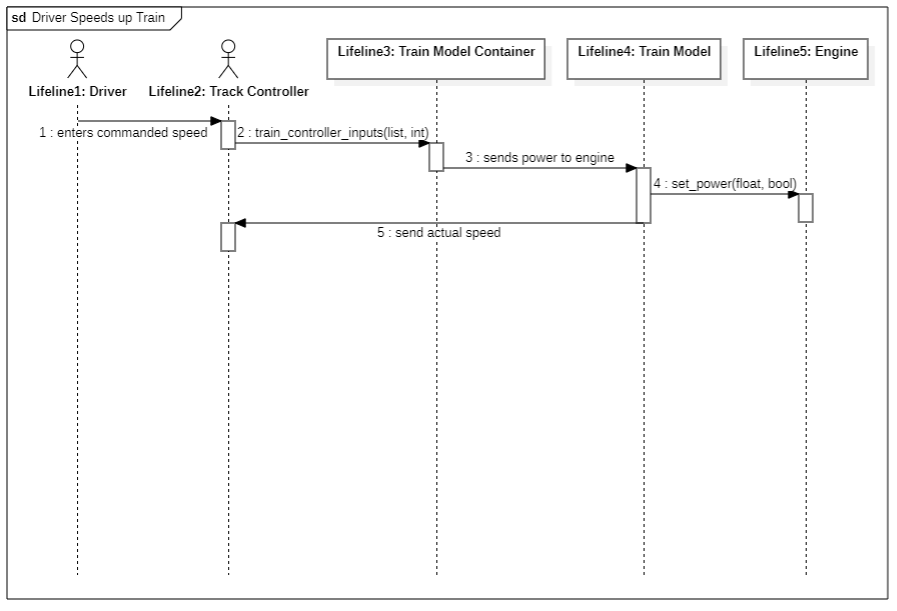


**Figure 6.5.2.10:** Train enters new block Sequence Diagram

* + - 1. **Driver Speeds up Train**

| Table 6.5.2.11: Driver Speeds up Train Use Case Description | |
| --- | --- |
| Actors | Driver, Train Controller |
| Description | 1. Driver enters desired speed into train controller 2. Train controller either calculates the power to send to the engine 3. The train controller send the power to the engine 4. The train model applies the input power to the physics calculation 5. The train model send the actual speed of the train to the train controller |
| Data | Float - power |
| Stimulus | Driver inputs new speed |
| Response | Update the power to the engine to calculate the new speed. Send speed back to train controller so I can calculate the new power |

**Table 6.5.2.11:** Driver Speeds up Train Description

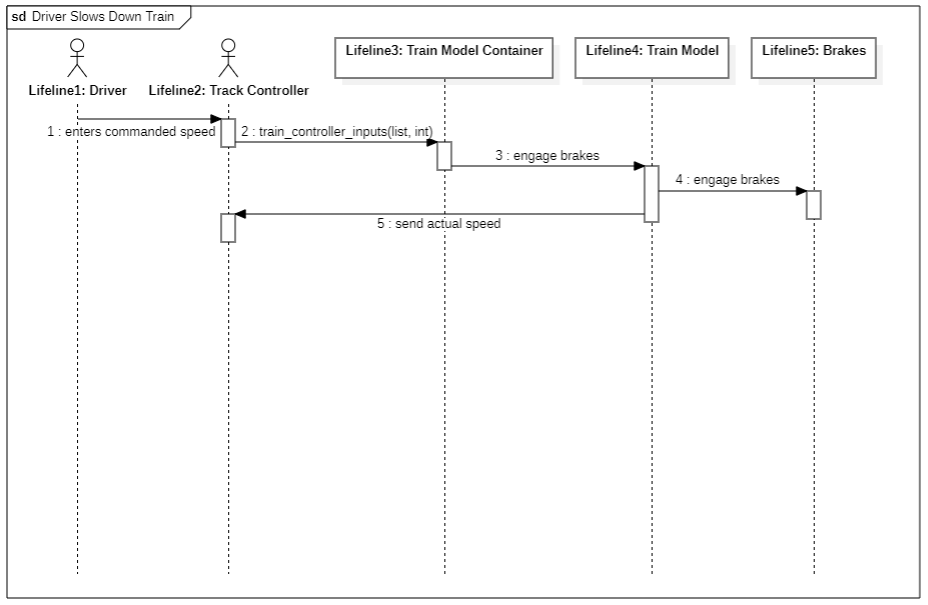


**Figure 6.5.2.11:** Driver Speeds up Train Sequence Diagram

* + - 1. **Driver Slows Down Train**

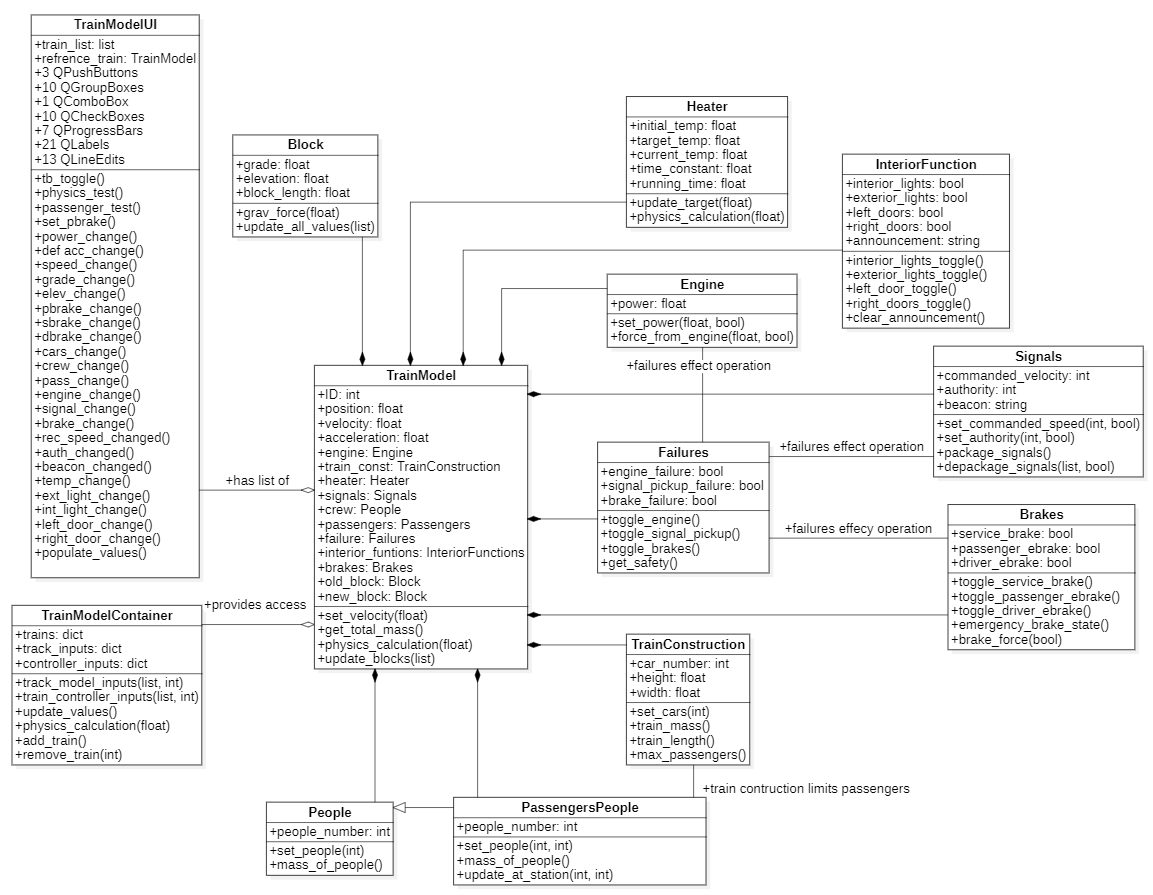
| Table 6.5.2.12: Driver Slows Down Train Use Case Description | |
| --- | --- |
| Actors | Driver, Train Controller |
| Description | 1. Driver enters desired speed into train controller 2. The train controller send the signal to activate the service brake to the train mode 3. The train model activates the service brake 4. The train model send the actual speed to the train controller |
| Data | Float - power |
| Stimulus | Driver inputs new speed |
| Response | Set the brakes and cut power to the engine. Send velocity back to the train controller so I can disable the brakes when desired speed is reached. |

**Table 6.5.2.12:** Driver Slows Down Train Description



**Figure 6.5.2.12:** Driver Slows Down Train Sequence Diagram

* + 1. **Software Class Diagram**

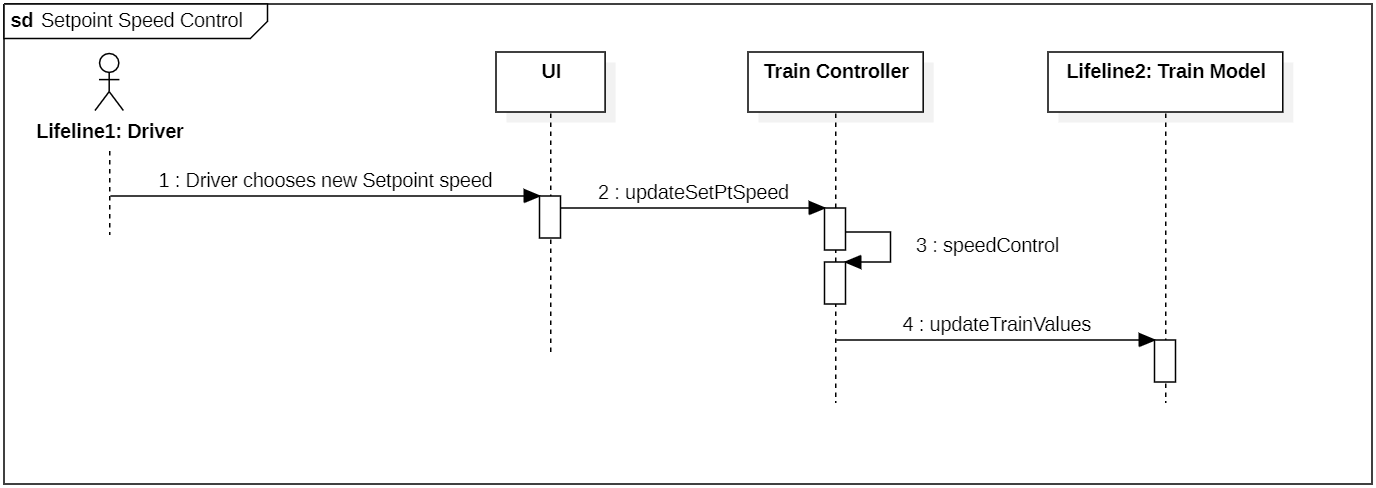


**Figure 6.5.3:** Train Model Class Diagram

* 1. **Train Controller SW**
     1. **Use case diagram**
     2. **Use Cases**
        1. **Driver updates setpoint speed**

| Table 6.6.2.1 Driver Updates Setpoint Speed | |
| --- | --- |
| Actors | Driver |
| Description | 1. Driver enters new setpoint speed into train controller UI 2. New setpoint speed value is processed to determine vitality    1. If new speed is found to be nonvital, no changes are made    2. If new speed is vital, setpoint speed is accepted in the system   If new value is accepted:   1. Train Controller runs speed and power calculations to determine new engine power to achieve speed. 2. Send new values to the train model |
| Data | Float - setPtSpeed |
| Stimulus | Driver changes setpoint speed |
| Response | Speed up or slow down the train if the system remains in vital state. Do nothing otherwise. |

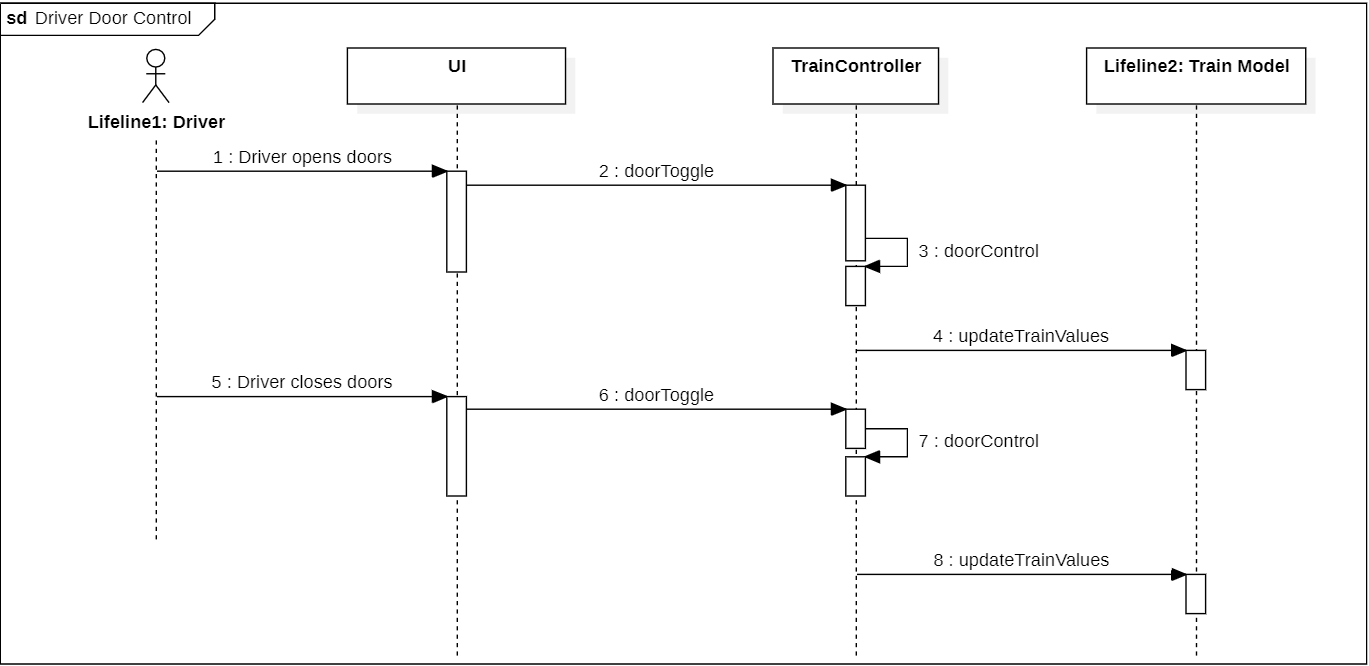
**Table 6.6.2.1:** Driver updates setpoint speed description

**Figure 6.6.2.1:** Driver updates setpoint speed sequence diagram

* + - 1. **Driver operates doors**

| Table 6.6.2.2 Driver Operates Doors | |
| --- | --- |
| Actors | Driver |
| Description | 1. System checks door side and current door state 2. If doors are closed, system opens doors according to door side 3. If doors are open, the system closes doors. 4. Send new door state values for respective doors to the train model |
| Data | Bool - doorState  Int - doorSide  Bool - doorL  Bool - doorR |
| Stimulus | Driver toggles doors |
| Response | Open doors if currently closed. Close doors if currently open. |

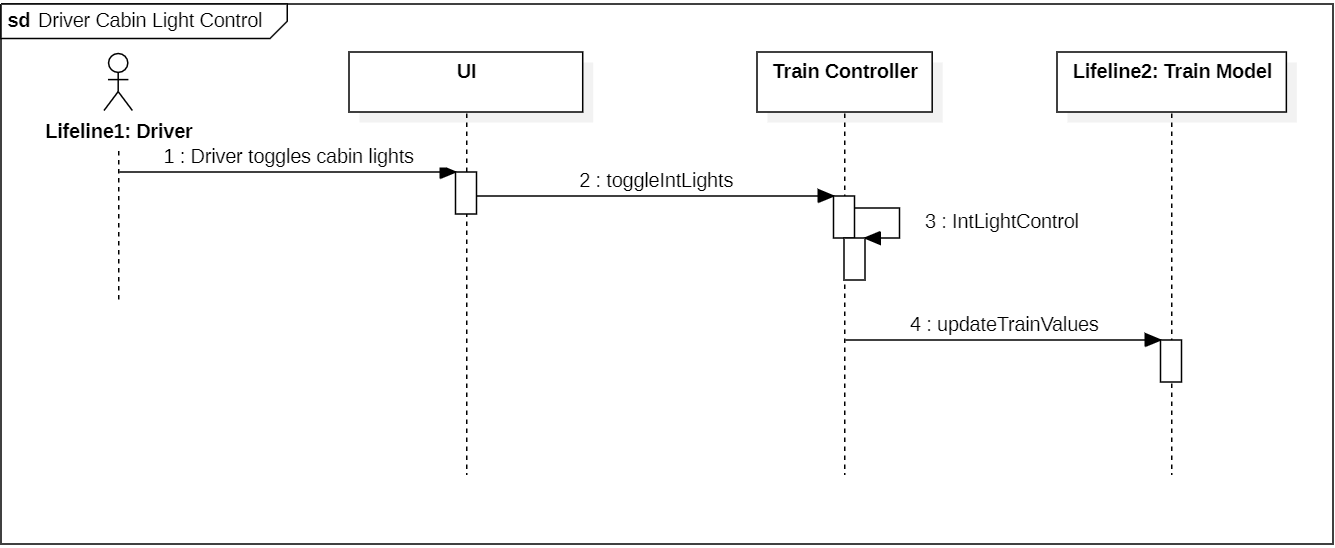
**Table 6.6.2.2:** Driver operates doors description

**Figure 6.6.2.2:** Driver operates doors sequence diagram

* + - 1. **Driver operates cabin lights**

| Table 6.6.2.3 Driver Operates Cabin Lights | |
| --- | --- |
| Actors | Driver |
| Description | 1. Check current state of lights 2. If cabin lights are on, set cabin lights off 3. If cabin lights are off, set cabin lights on 4. Send new cabin lights value to train model |
| Data | Int - intLights |
| Stimulus | Driver toggles interior lights lightswitch |
| Response | Turn cabin lights on if currently off. Turn cabin lights off if currently on. |

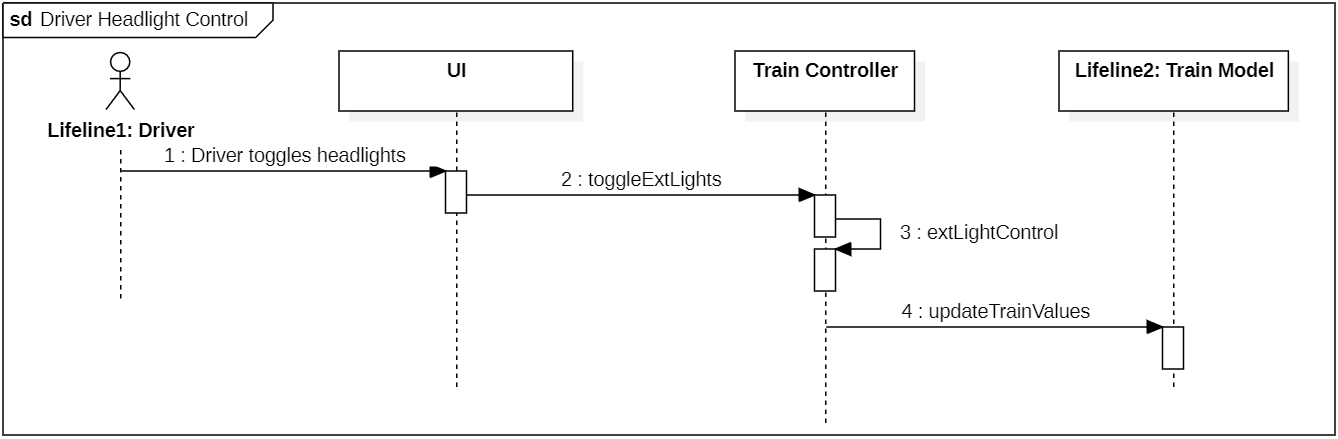
**Table 6.6.2.3:** Driver operates cabin lights description

**Figure 6.6.2.3:** Driver operates cabin lights sequence diagram

* + - 1. **Driver operates headlights**

| Table 6.6.2.4 Driver Operates Headlights | |
| --- | --- |
| Actors | Driver |
| Description | 1. Check current state of headlights 2. If currently off, set on 3. If currently on, set off 4. Send new value to train model |
| Data | Int - extLights |
| Stimulus | Driver toggles exterior lights lightswitch |
| Response | Turn headlights on if currently off. Turn headlights off if currently on. |

**Table 6.6.2.4:** Driver operates headlights description

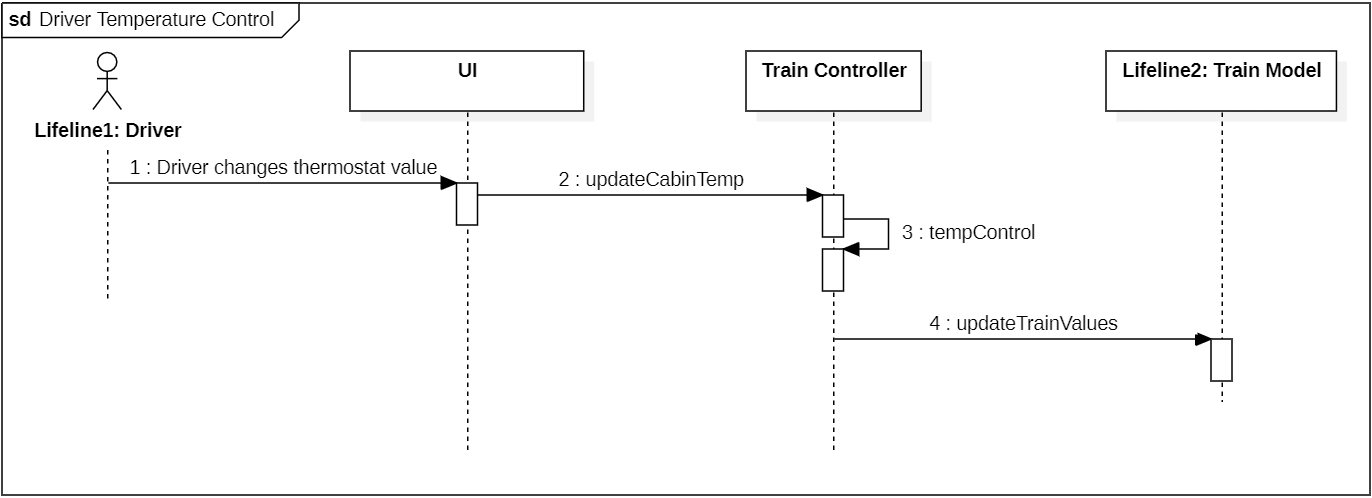


**Figure 6.6.2.4:** Driver operates headlights sequence diagram

* + - 1. **Driver sets cabin temperature**

| Table 6.6.2.5 Driver Sets Cabin Temperature | |
| --- | --- |
| Actors | Driver |
| Description | Send the new temperature setpoint value to the train model |
| Data | Float - cabinTemp |
| Stimulus | Driver changes cabin temperature |
| Response | Warm or cool cabin based on new value |

**Table 6.6.2.5:** Driver sets cabin temperature description

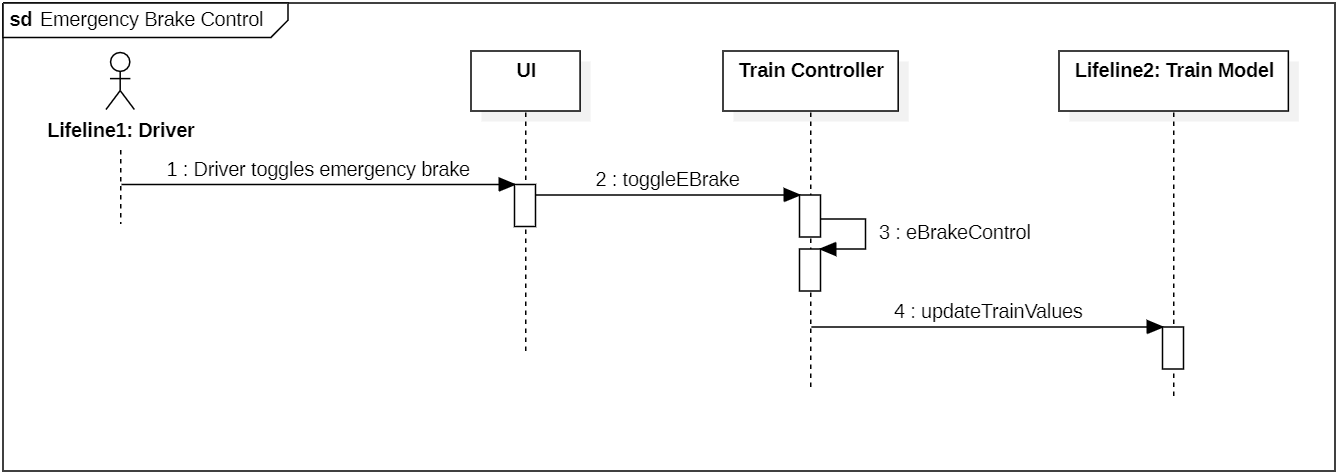


**Figure 6.6.2.5:** Driver sets cabin temperature sequence diagram

* + - 1. **Driver operates emergency brake**

| Table 6.6.2.6 Driver Operates Emergency Brake | |
| --- | --- |
| Actors | Driver |
| Description | 1. Check state of emergency brake   If emergency brake is disabled:   1. Enable emergency brake 2. Set engine power to 0 3. Train decelerates at allowed emergency brake deceleration rate   If emergency brake if enabled:   1. Disable emergency brake 2. Allow train to accept normal operation values 3. Resume train operation   2. Send new values to train model |
| Data | Bool - eBrake |
| Stimulus | Driver toggles emergency brake |
| Response | Enable emergency brake if currently disabled and stop the train. Disable emergency brake if currently enabled and allow the train to operate normally. |

**Table 6.6.2.6:** Driver operates emergency brake description

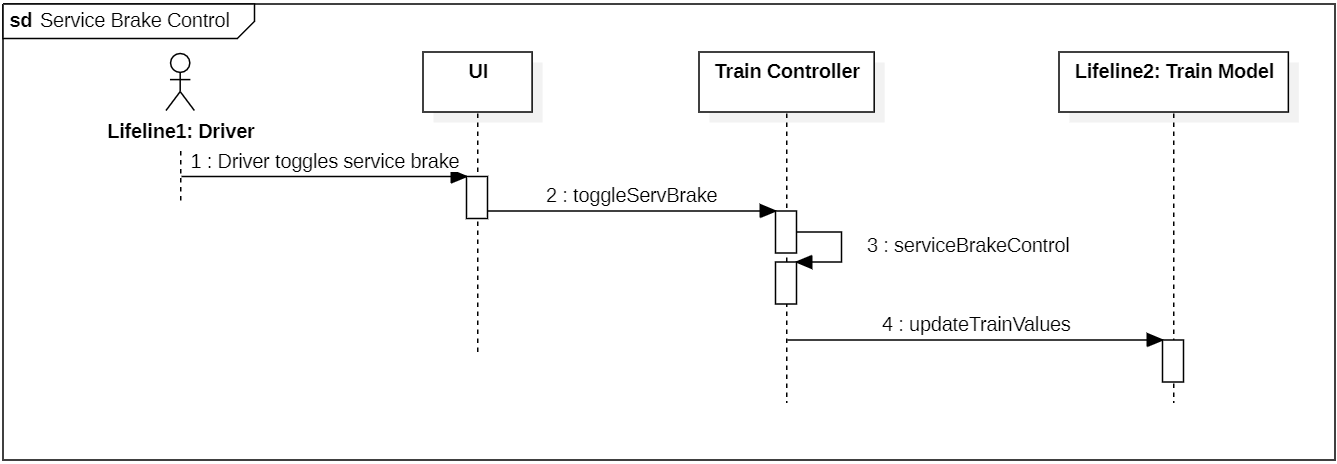


**Figure 6.6.2.6:** Driver operates emergency brake sequence diagram

* + - 1. **Driver operates service brake**

| Table 6.6.2.7 Driver Operates Service Brake | |
| --- | --- |
| Actors | Driver |
| Description | 1. Check state of service brake   If service brake if disabled:   1. Enable service brake 2. Set Engine power to zero 3. Train decelerates at allowed service brake deceleration rate   If service brake is enabled:   1. Disable service brake 2. Allow train to resume regular operation based on previous values   2. Send new values to train model |
| Data | Bool - servBrake |
| Stimulus | Driver toggles service brake |
| Response | Enable service brake if currently disabled and slow down the train. Disable service brake if currently enabled and resume normal operation. |

**Table 6.6.2.7:** Driver operates service brake description

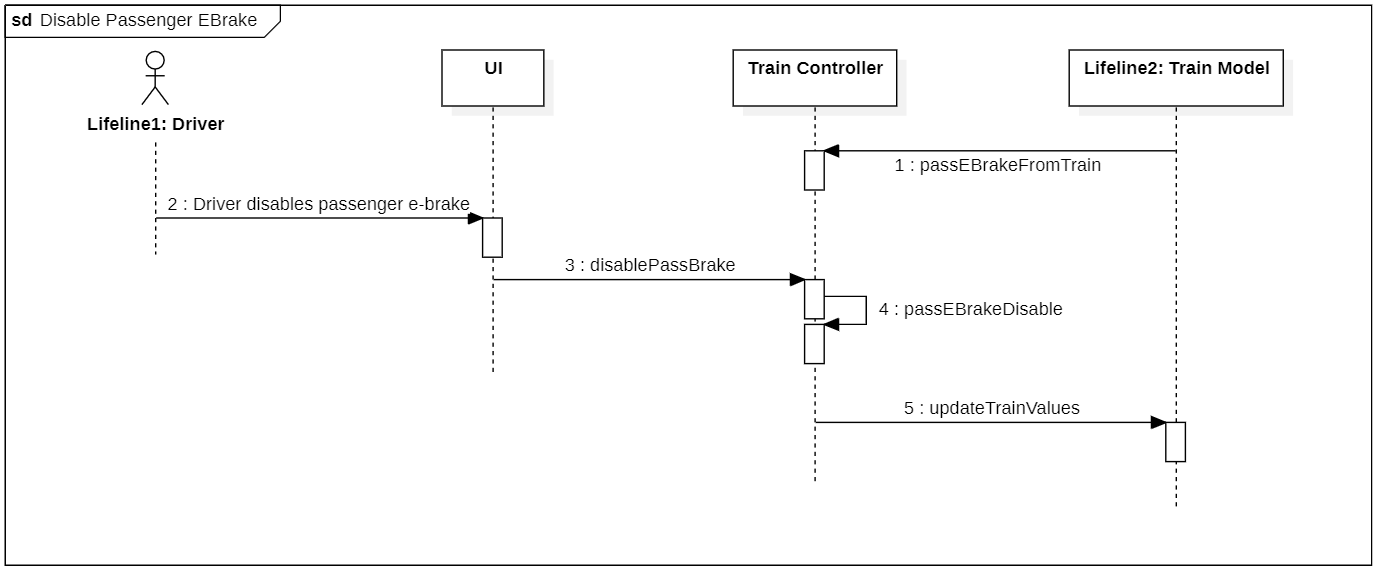


**Figure 6.6.2.7:** Driver operates service brake sequence diagram

* + - 1. **Driver disables passenger emergency brake**

| Table 6.6.2.8 Driver Disables Passenger Emergency Brake | |
| --- | --- |
| Actors | Driver |
| Description | 1. Passenger Emergency Brake is disabled 2. Train can return to normal operation 3. Send new values to train model |
| Data | Bool - passEBrake |
| Stimulus | Driver toggles off passenger eBrake |
| Response | Passenger Emergency Brake is disabled and the train resumes normal operation. |

**Table 6.6.2.8:** Driver disables passenger brake description

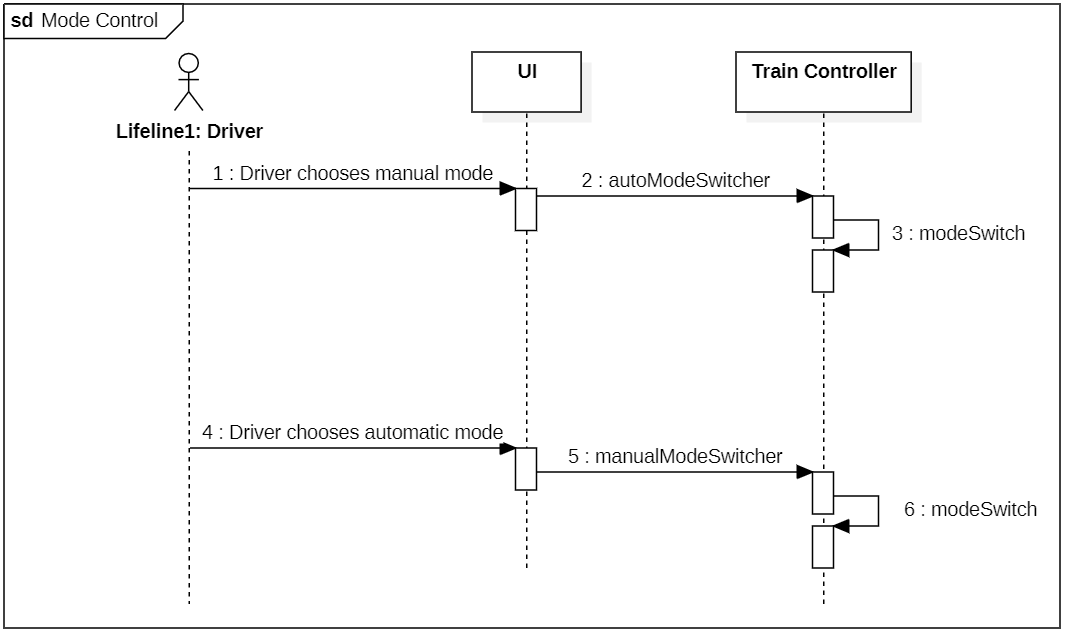


**Figure 6.6.2.8:** Driver disables passenger brake sequence diagram

* + - 1. **Driver chooses operation mode**

| Table 6.6.2.9 Driver Chooses Operation Mode | |
| --- | --- |
| Actors | Driver |
| Description | 1. Driver chooses manual mode    1. System is set to manual operation    2. Driver is free to control automatic controls 2. Driver chooses automatic mode    1. System is set to automatic operation    2. Automatic controls are controlled by train control |
| Data | Bool - mode |
| Stimulus | Driver toggles between automatic and manual mode |
| Response | If Driver chooses manual mode, the driver is able to control temperature, doors, lights, etc. If Driver chooses automatic mode, all operations are controlled by train controller |

**Table 6.6.2.9:** Driver chooses operation mode description

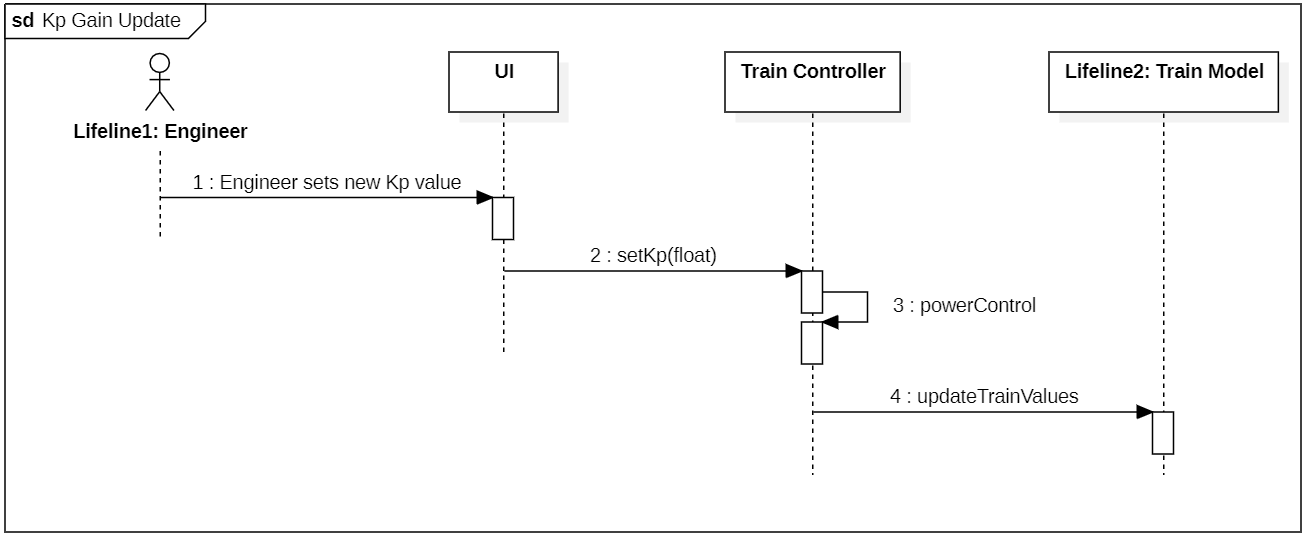


**Figure 6.6.2.9:** Driver chooses mode of operation sequence diagram

* + - 1. **Engineer updates Kp value**

| Table 6.6.2.10 Engineer Updates Kp Value | |
| --- | --- |
| Actors | Engineer |
| Description | 1. Engineer enters new Kp value 2. Train Controller checks if new value is stable   If stable:   1. New power is calculated according to new value 2. New power is sent to train |
| Data | Float - Kp |
| Stimulus | Engineer changes Kp value |
| Response | Engine power is increased or decreased if value is accepted. No change otherwise |

**Table 6.6.2.10:** Engineer updates Kp value description

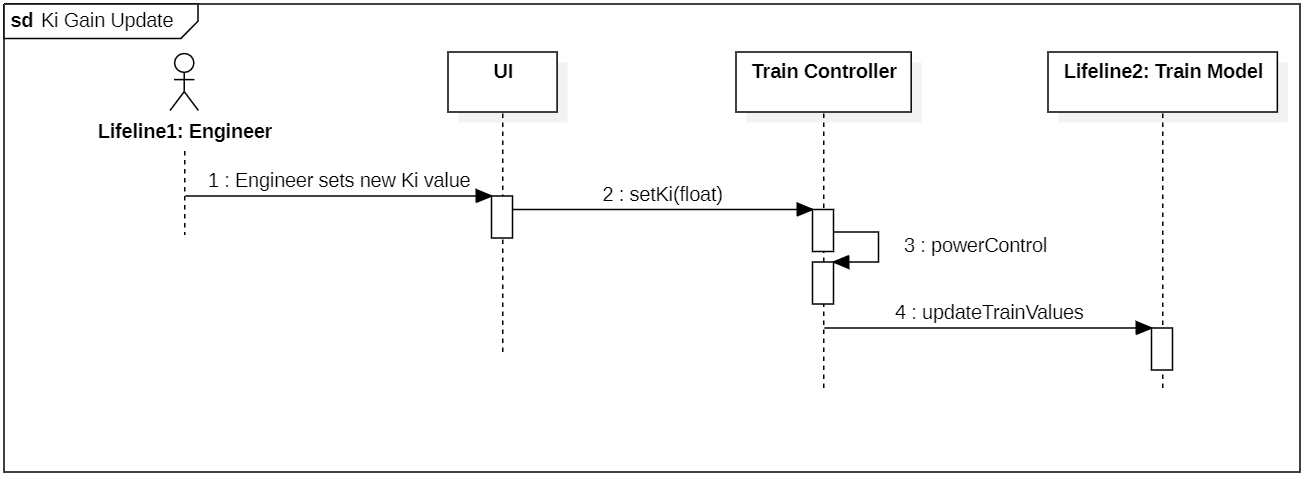


**Figure 6.6.2.10:** Engineer Updates Kp value sequence diagram

* + - 1. **Engineer updates Ki value**

| Table 6.6.2.11 Engineer Updates Ki Value | |
| --- | --- |
| Actors | Engineer |
| Description | 1. Engineer enters new Ki value 2. Train Controller checks is new value is stable   If stable:   1. New power is calculated according to new value 2. New power is sent to train |
| Data | Float - Ki |
| Stimulus | Engineer changes Ki value |
| Response | Engine power is increased or decreased if value is accepted. No change otherwise. |

**Table 6.6.2.11:** Engineer updates Ki value description

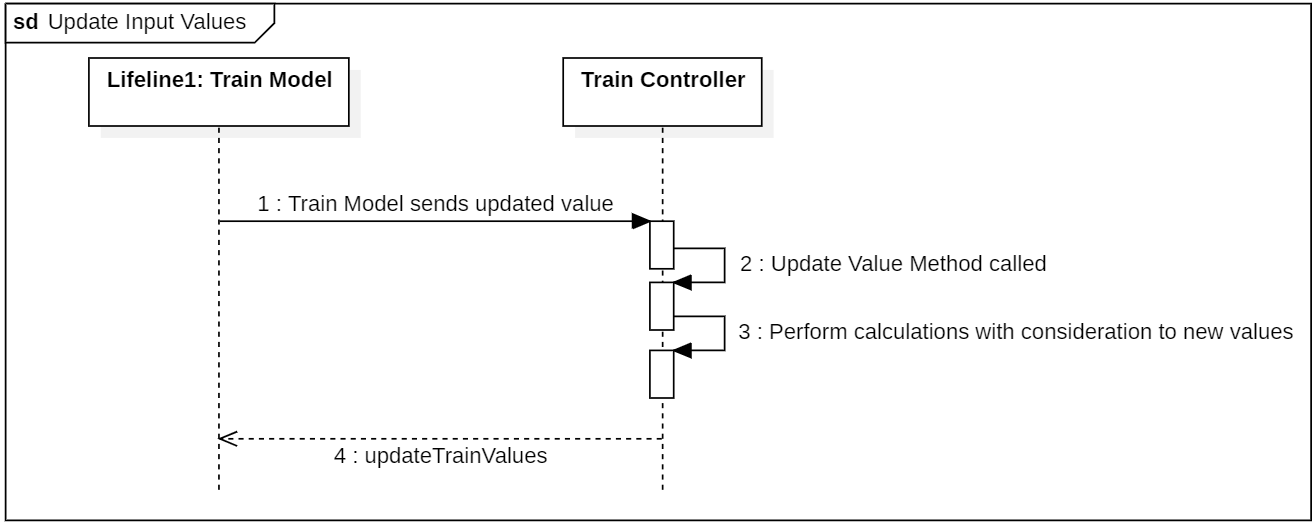


**Figure 6.6.2.11:** Engineer updates Ki value sequence diagram

* + - 1. **Train Model updates input values**

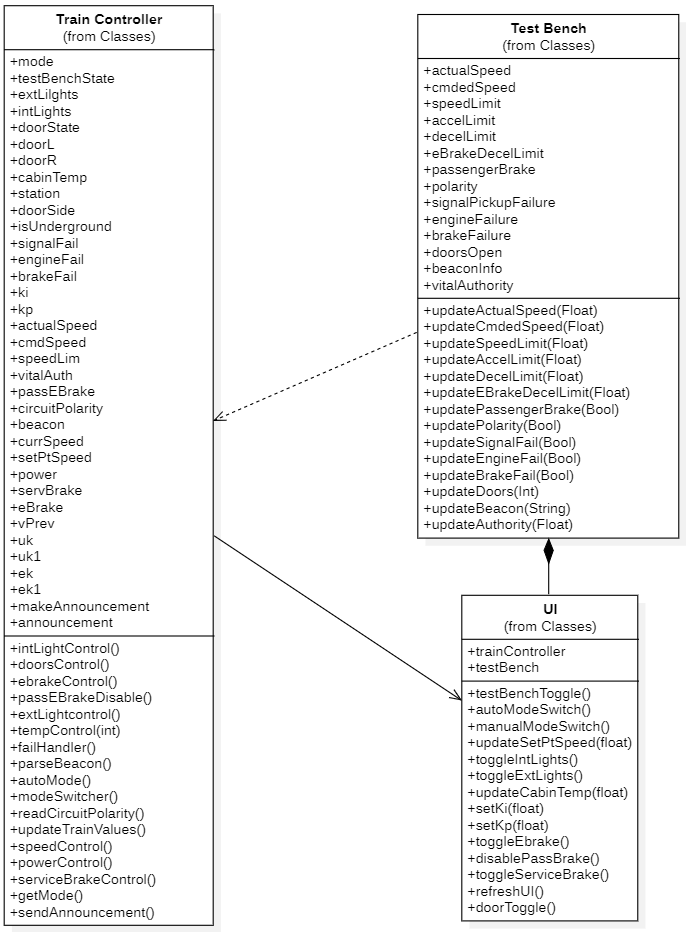
| Table 6.6.2.12 Train Model Updates Input values | |
| --- | --- |
| Actors | Train Model |
| Description | 1. Train Model calls an update function in the container class 2. The values related to the called function are updated 3. Any control functions are then called with accordance to new values. |
| Data | Float - actSpeed,  Float - cmdSpeed,  Float - vitalAuth,  Bool - passEBrake,  Bool - circuitPolarity,  Int - doorSide  String - beacon  Bool - signalFail  Bool - engineFail  Bool - brakeFail  String - announcement |
| Stimulus | Train Model sends any updated value |
| Response | Train Controller operates according to new values. |

**Table 6.6.2.12:** Train Model updates input values description



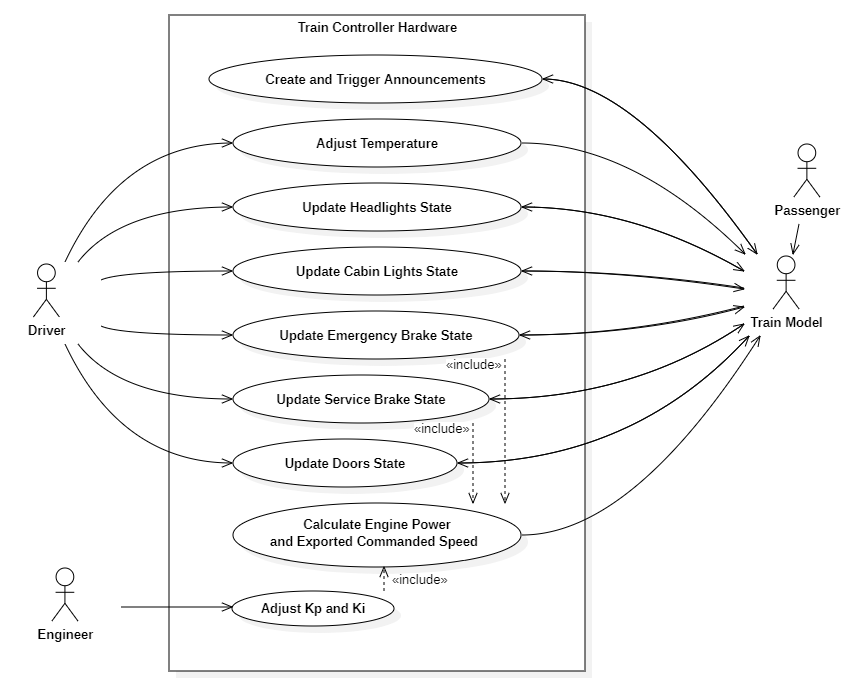
**Figure 6.6.2.12:** Train Model updates input values sequence diagram

* + 1. **Software Class Diagram**



**Figure 6.6.3:** Train Controller SW Class Diagram

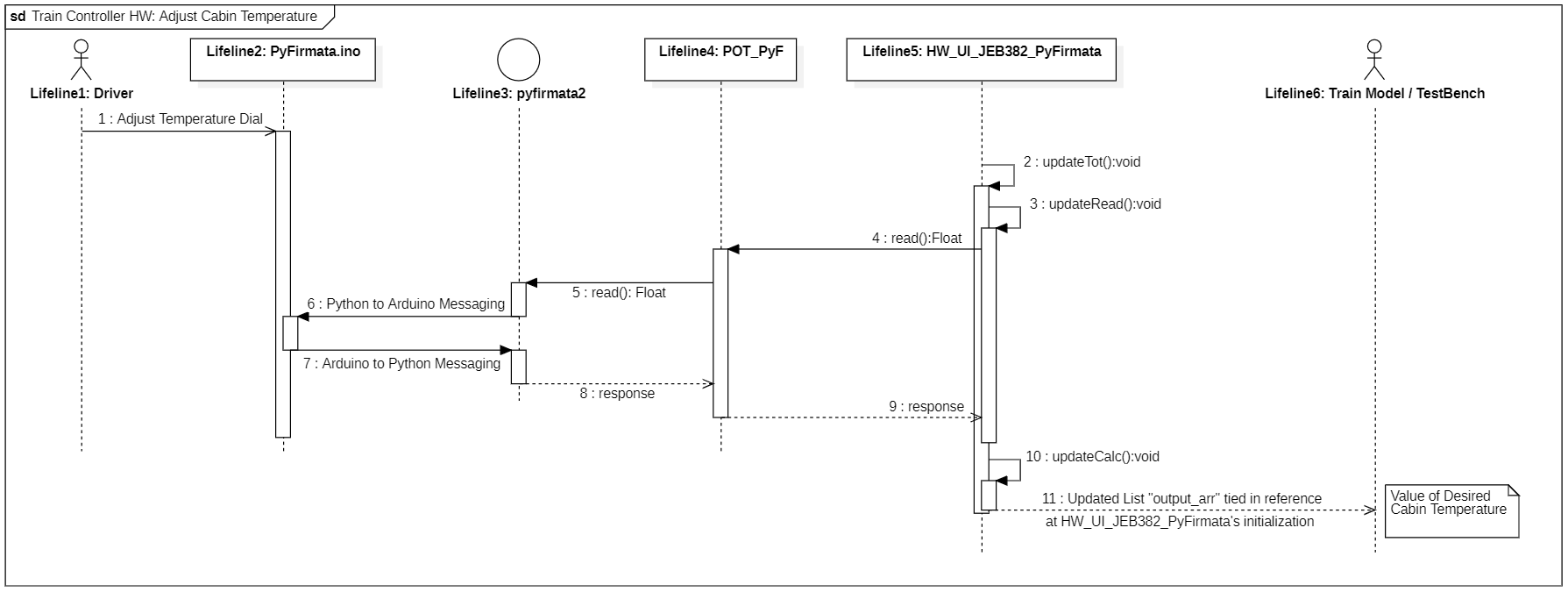
* 1. **Train Controller HW**
     1. **Use case diagram**



**Figure 6.7.1:** Train Controller HW Use Case Diagram

* + 1. **Module use cases**
       1. **Update Cabin Temperature**

| Table 6.7.2.1.1 Adjust Cabin Temperature Use Case Description | |
| --- | --- |
| Actors | Driver, Train Controller, Train Model |
| Description | 1. Driver gives Train Controller desired temperature 2. Outputs desired Temperature value to Train Model 3. Sets and keeps Train to desired Temperature |
| Data | Floating Point Value - State of Temperature Dial on Driver’s Controller adjusted with set Scale and Boundaries |
| Stimulus | Whenever Train is pulled from Yard or adjusted by Driver. |
| Response | The Train Controller will either output the initial set temperature when the train is pulled out of the yard, or output the temperature given by the driver to the Train Model. The Train Model then adjusts and keeps the cabin temperature to its given value. |

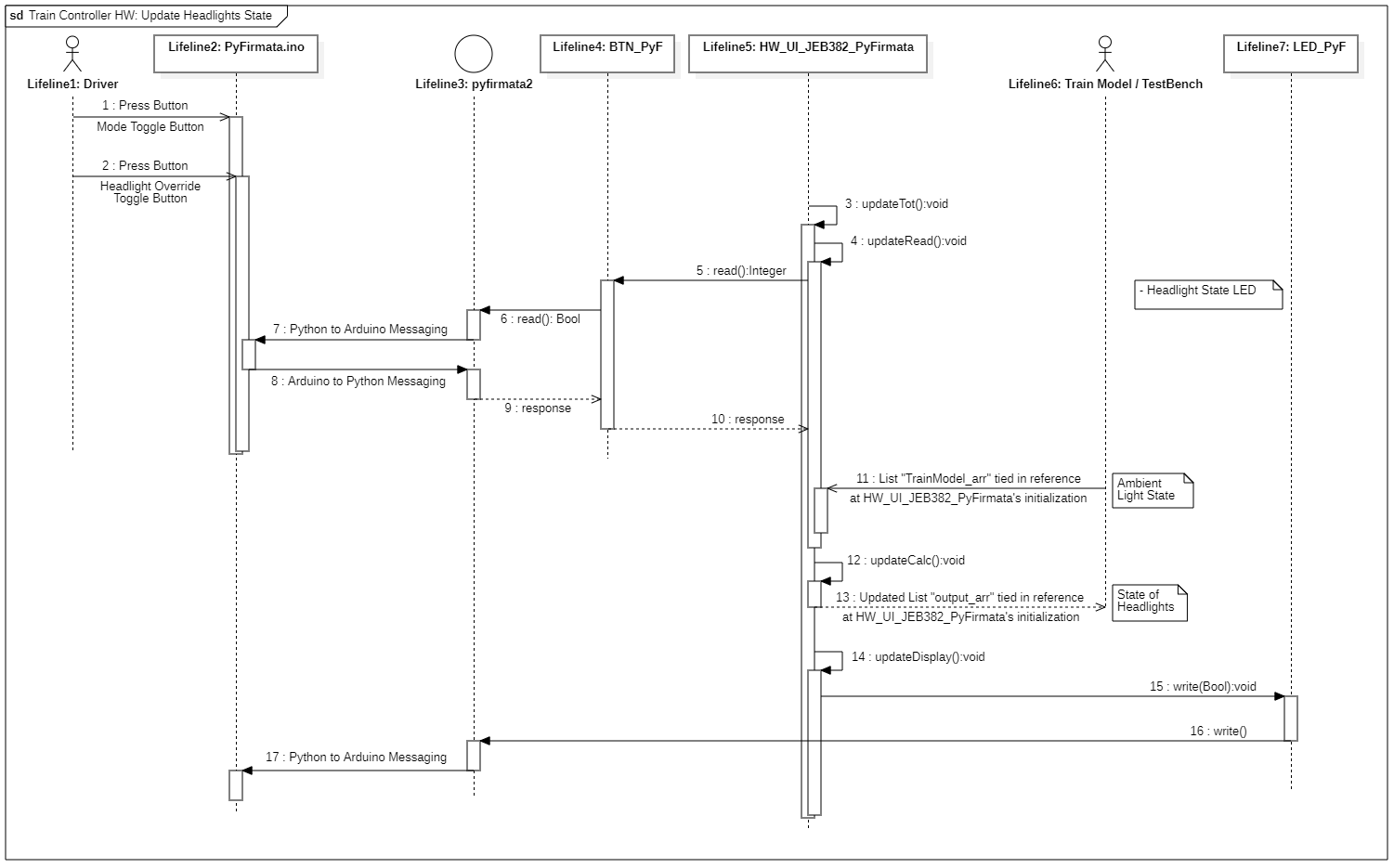
**Table 6.7.2.1.1:** Adjust Cabin Temperature Description

**Figure 6.7.2.1.2:** Adjust Cabin Temperature Sequence Diagram

* + - 1. **Update Headlight State**

| Table 6.7.2.2 Update Headlight State Use Case Description | |
| --- | --- |
| Actors | Driver, Train Controller, Train Model |
| Description | 1. Driver gives Train Controller States of Operation including the Headlight Override and the Mode Button Status 2. The Train Model gives the State of Ambient light based on a threshold of whether Train is in Darkness 3. Output to Train Model the correct Boolean Calculation on State of the Headlights 4. Sets train’s Headlights and Driver’s indicator LED to set Status |
| Data | 1. Integer - Mode Button Status 2. Integer - Headlight Override Button Status; considered in Manual Mode 3. Boolean - Ambient Light Status 4. Boolean - Driver LED Cabin Light Status Indicator |
| Stimulus | Whenever a train’s Ambient Light sensor indicates a change or is adjusted by the driver. |
| Response | The Train will either turn on the Headlights in accordance with Ambient Light, such as on underground or at night or off during the day, in Auto Mode or be overwritten by the Driver in Manual Mode. Alert the Driver of the output Status with an Indicator LED on their console. |

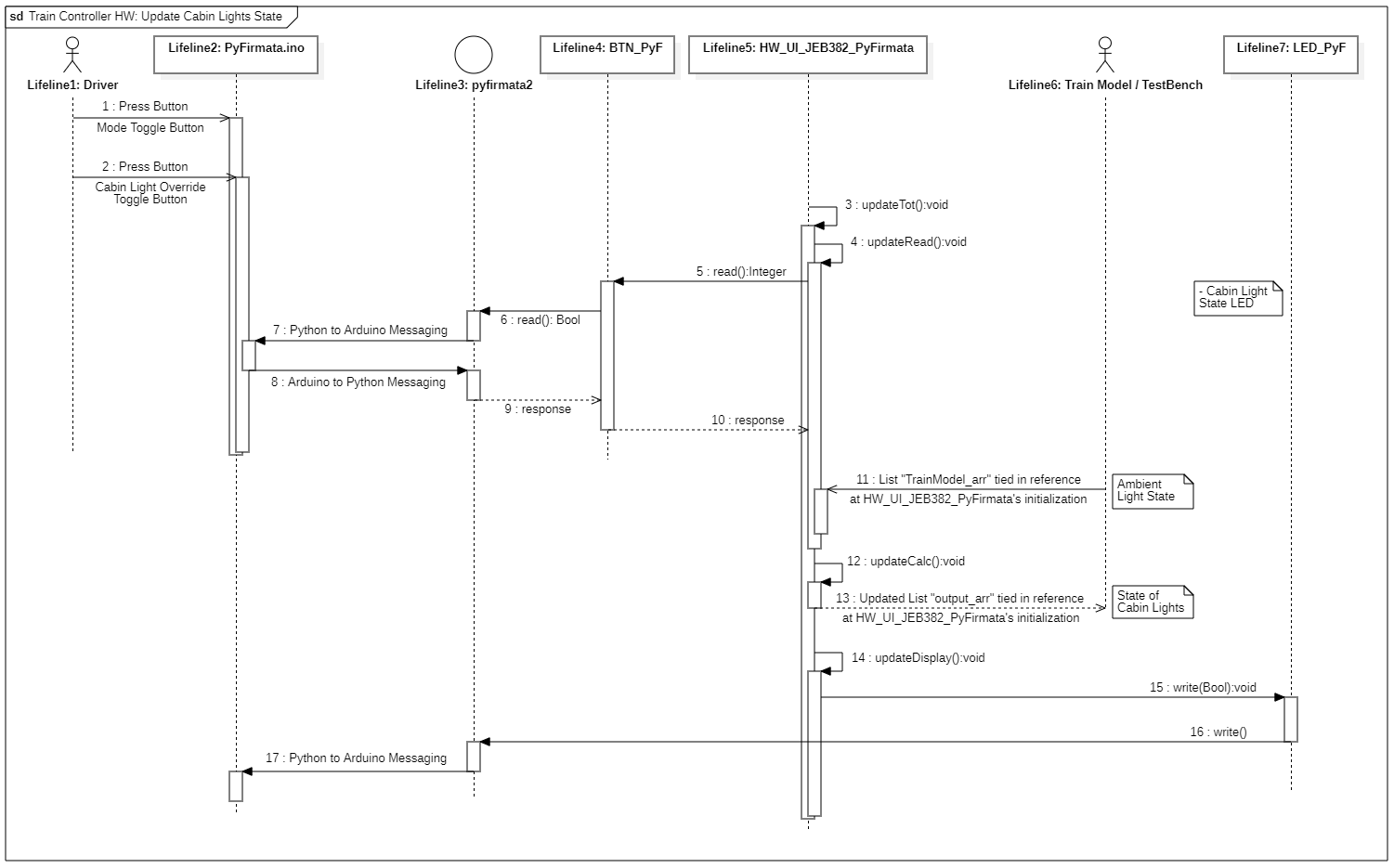
**Table 6.7.2.2:** Update Headlight State Description

**Figure 6.7.2.2:** Update Headlight State Sequence Diagram

* + - 1. **Update Cabin Light State**

| Table 6.7.2.3 Update Cabin Light State Use Case Description | |
| --- | --- |
| Actors | Driver, Train Controller, Train Model |
| Description | 1. Driver gives Train Controller States of Operation including the Cabin Light Override and the Mode Button Status 2. The Train Model gives the State of Ambient light based on a threshold of whether Train is in Darkness 3. Output to Train Model the correct Boolean Calculation on State of the Cabin Lights 4. Sets train’s Cabin lights and Driver’s indicator LED to set Status |
| Data | 1. Integer - Mode Button Status 2. Integer - Cabin Light Override Button Status; considered in Manual Mode 3. Boolean - Ambient Light Status 4. Boolean - Driver LED Cabin Light Status Indicator |
| Stimulus | Whenever a train’s Ambient Light sensor indicates a change or is adjusted by the driver. |
| Response | The Train will either turn on the Cabin Lights in accordance with Ambient Light, such as on underground or at night or off during the day, in Auto Mode or be overwritten by the Driver in Manual Mode. Alert the Driver of the outputted Status with an Indicator LED on their console. |

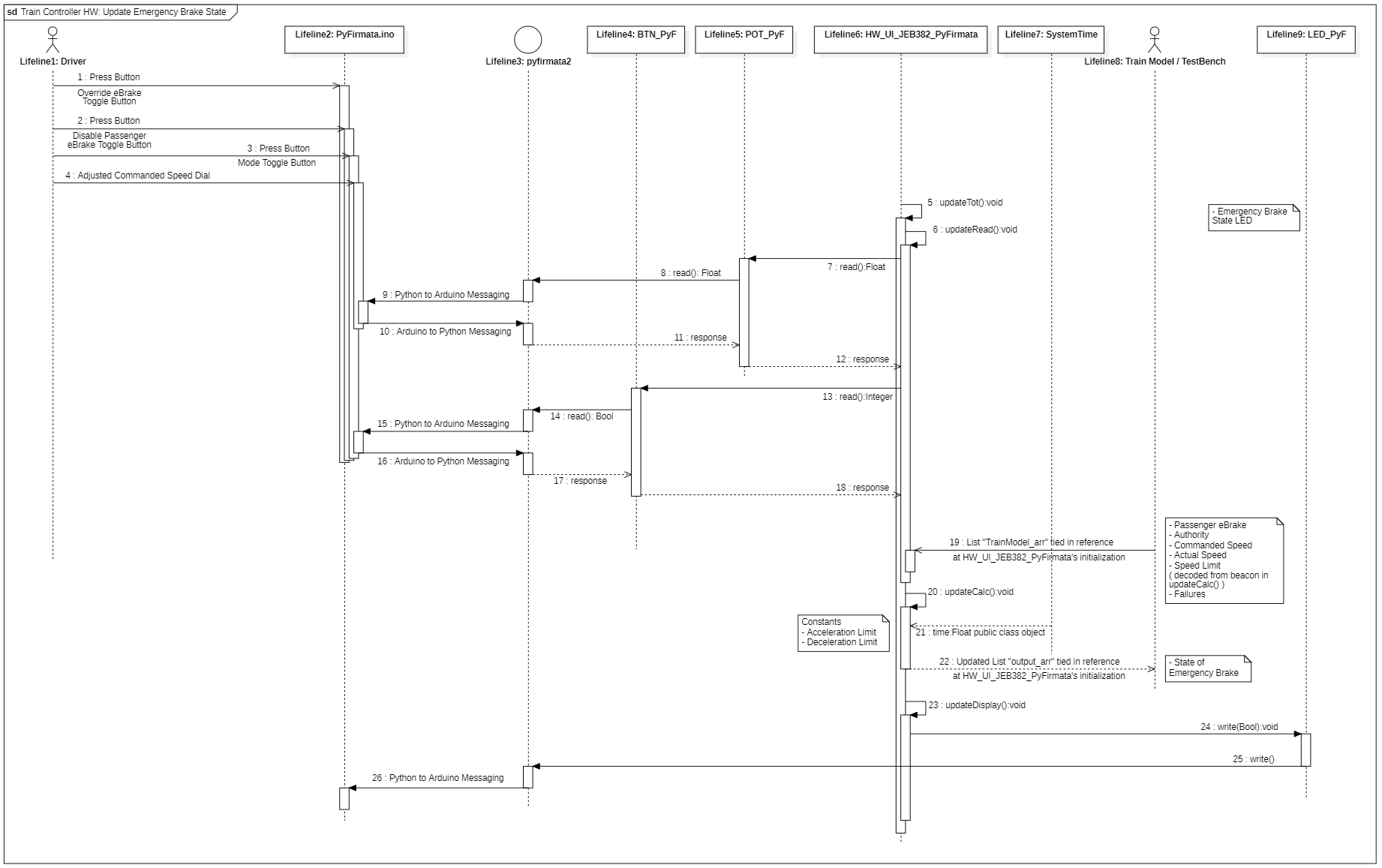
**Table 6.7.2.3:** Update Cabin Light State Description

**Figure 6.7.2.3:** Update Cabin Light State Sequence Diagram

* + - 1. **Update Emergency Brake State**

| Table 6.7.2.4 Update Emergency Brake State Use Case Description | |
| --- | --- |
| Actors | Driver, Train Controller, Train Model |
| Description | 1. Driver gives Train Controller States of Operation including the Emergency Brake Override and the Disable Passenger Button Status in either Mode 2. Driver gives desired commanded speed in Manual Mode and the Mode Button Status 3. The Train Model gives passenger Emergency Brake Button Status and relevant Engine Power and Exported Commanded Speed information either from sensors or Beacon information and an indexed Block Information Look-up-Table. Including Authority, CTC Commanded Speed, Speed Limit, and Failure Status. 4. Output to Train Model if from either states of override buttons from Driver and Passenger, or if the Braking Force of the Emergency Brake is needed in the calculation of Engine Power and Exported Commanded Speed; if the Emergency Brake is to be engaged 5. Sets train’s Cabin lights and Driver’s indicator LED to set Status |
| Data | 1. Integer - Emergency Brake Override Button Status 2. Integer - Disable Passenger Emergency Brake Button Status 3. Integer - Mode Button Status 4. Floating Point Value - State of Commanded Speed Dial on Driver’s Controller adjusted with set Scale and Boundaries; considered in Manual Mode 5. Floating Point Value - A managed indication of time to calculate acceleration and deceleration to compare to limits 6. Integer - Passenger Emergency Brake Button Status 7. Integer - Authority given by Train Model from CTC 8. Floating Point Value - Commanded Speed given by Train Model from CTC 9. Floating Point Value - Actual Speed 10. String - Beacon Information 11. Floating Point Value - Speed Limit decoded from Beacon Information and an indexed Block Information Look-up-Table 12. Boolean - Signal Failure Status 13. Boolean - Engine Failure Status 14. Boolean - Brake Failure Status 15. Boolean - Driver LED Emergency Brake Status Indicator 16. Boolean - Driver LED Disable Passenger Emergency Brake Status Indicator |
| Stimulus | Whenever the Driver Toggles the Emergency Brake Button in either mode, Driver disables the Passenger Emergency Brake, Driver adjusts Commanded speed in Manual Mode, the Passenger engages their Emergency Brake, or Engine Power and Exported Commanded Speed is calculated to need the braking force of the Emergency Brake. |
| Response | Train will either engage or disable Emergency Brake and alert the Driver of the outputted Status with an Indicator LED on their console. |

**Table 6.7.2.4:** Update Emergency Brake State Description

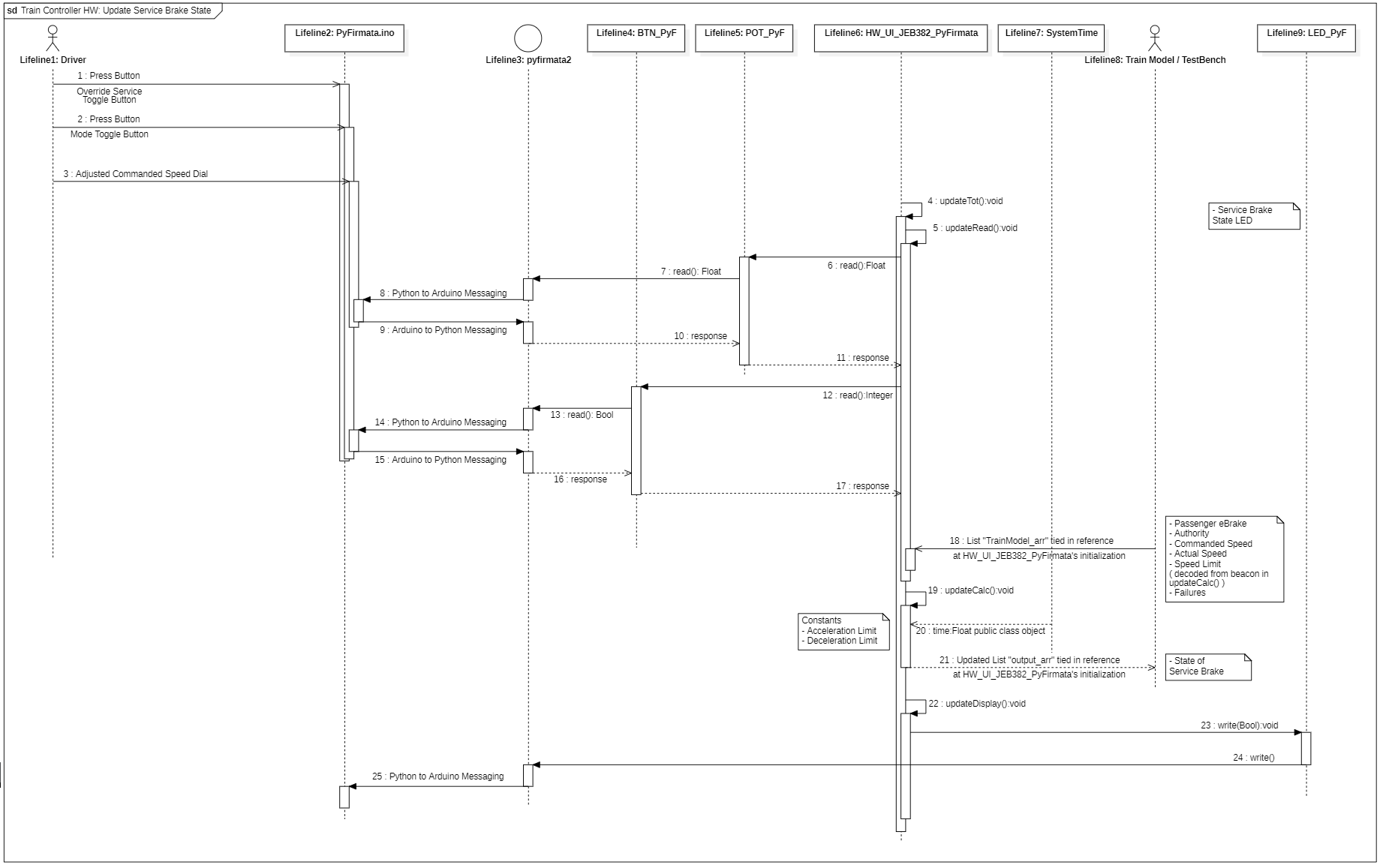


**Figure 6.7.2.4:** Update Emergency Brake State Sequence Diagram

* + - 1. **Update Service Brake State**

| Table 6.7.2.5 Update Service Brake State Use Case Description | |
| --- | --- |
| Actors | Driver, Train Controller, Train Model |
| Description | 1. Driver gives Train Controller States of Operation including the Service Brake Override and the Mode Button Status 2. Driver gives desired commanded speed in Manual Mode 3. The Train Model gives passenger Service Brake Button Status and relevant Engine Power and Exported Commanded Speed information either from sensors or Beacon information and an indexed Block Information Look-up-Table. Including Authority, CTC Commanded Speed, Speed Limit, and Failure Status. 4. Output to Train Model if from either states of override buttons from Driver and Passenger, or if the Braking Force of the Service Brake is needed in the calculation of Engine Power and Exported Commanded Speed; if the Service Brake is to be engaged 5. Sets train’s Cabin lights and Driver’s indicator LED to set Status |
| Data | 1. Integer - Service Brake Override Button Status 2. Integer - Mode Button Status 3. Floating Point Value - State of Commanded Speed Dial on Driver’s Controller adjusted with set Scale and Boundaries; considered in Manual Mode 4. Floating Point Value - A managed indication of time to calculate acceleration and deceleration to compare to limits 5. Integer - Authority given by Train Model from CTC 6. Floating Point Value - Commanded Speed given by Train Model from CTC 7. Floating Point Value - Actual Speed 8. String - Beacon Information 9. Floating Point Value - Speed Limit decoded from Beacon Information and an indexed Block Information Look-up-Table 10. Boolean - Signal Failure Status 11. Boolean - Engine Failure Status 12. Boolean - Brake Failure Status 13. Boolean - Driver LED Service Brake Status Indicator |
| Stimulus | Whenever the Driver Toggles the Service Brake Button in either mode, Driver disables the Passenger Service Brake, Driver adjusts Commanded speed in Manual Mode, the Passenger engages their Service Brake, or Engine Power and Exported Commanded Speed is calculated to need the braking force of the Service Brake. |
| Response | Train will either engage or disable Service Brake and alert the Driver of the output Status with an Indicator LED on their console. |

**Table 6.7.2.5:** Update Service Brake State Description

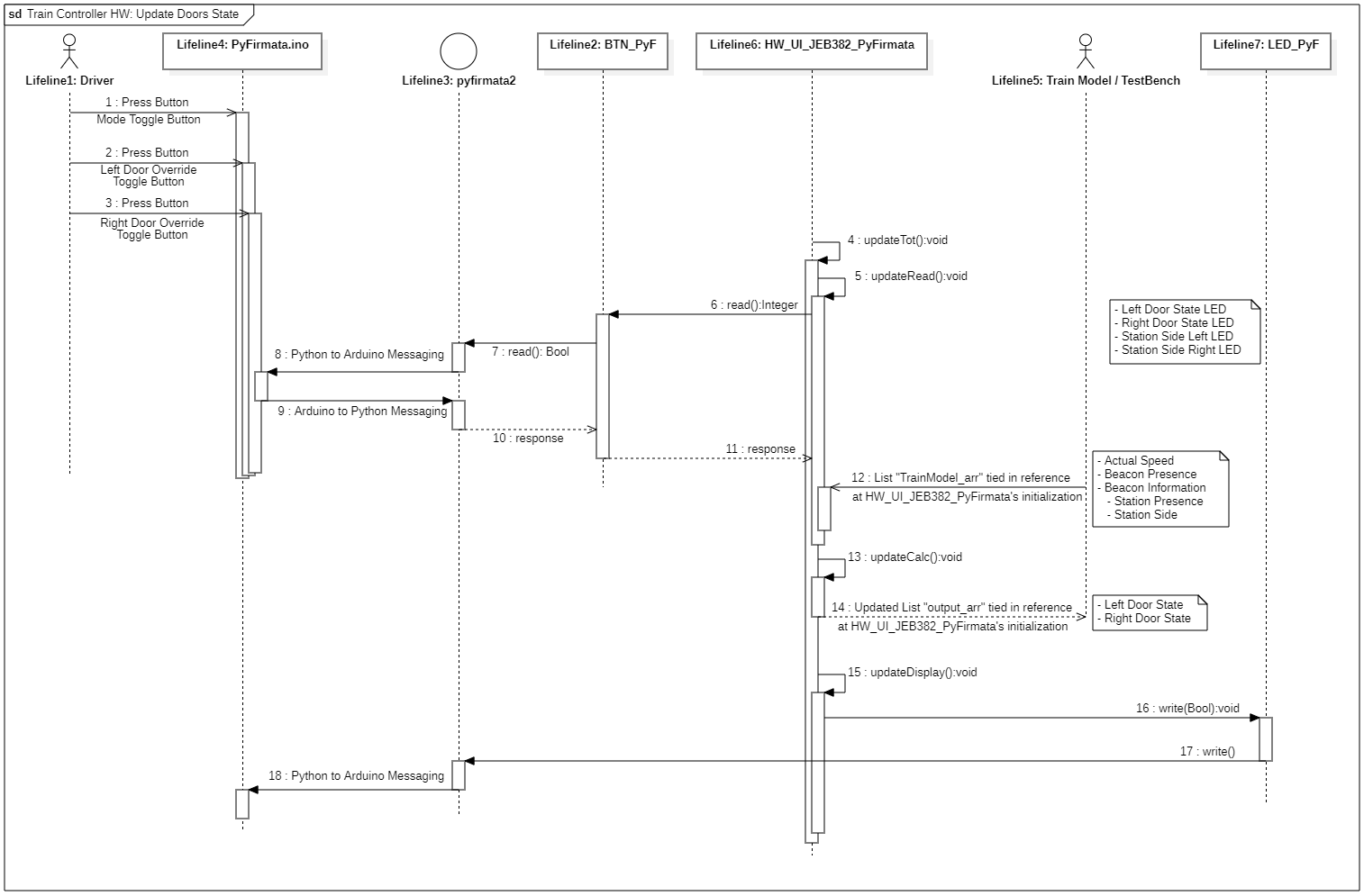


**Figure 6.7.2.5:** Update Service Brake State Sequence Diagram

* + - 1. **Update Doors State**

| Table 6.7.2.6 Update Doors State Use Case Description | |
| --- | --- |
| Actors | Driver, Train Controller, Train Model |
| Description | 1. Driver gives Train Controller States of Operation including both side door’s Overrides and the Mode Button Status 2. Train Model gives Actual speed and picks up Beacon Information and passes it to the Train Controller which decodes it with an index Look-up-Table to find Station Information including if a Station is approaching and it’s Docking Side 3. Output to Train Model the correct Boolean Calculation on State of Both Doors 4. Open or Close both sides of train doors and set status of Driver’s indicator LEDs |
| Data | 1. Integer - Mode Button Status 2. Integer - Left Door Override Button Status 3. Integer - Right Door Override Button Status 4. Floating Point Value - Actual Speed given by Train Model 5. String - Beacon Presence and Information that when decoded through an indexed Block Information Look-up-Table gives Station Presence and Docking Side 6. Boolean - Driver LED Left Door Status Indicator 7. Boolean - Driver LED Right Door Status Indicator 8. Boolean - Driver LED Station Side Left Status Indicator 9. Boolean - Driver LED Station Side Right Status Indicator |
| Stimulus | Whenever the Driver Toggles either Door Override Button in Manual Mode or when the train reaches a station indicated by a Beacon in Auto Mode. |
| Response | The train will either open or close either side door in accordance with the Station Information decoded from the beacon in Auto Mode or is overwritten by the Driver in Manual Mode. Alert the Driver of the outputted status of the doors and the Approaching Station’s Docking Side with Indicator LEDs on their console. |

**Table 6.7.2.6:** Update Doors State Description

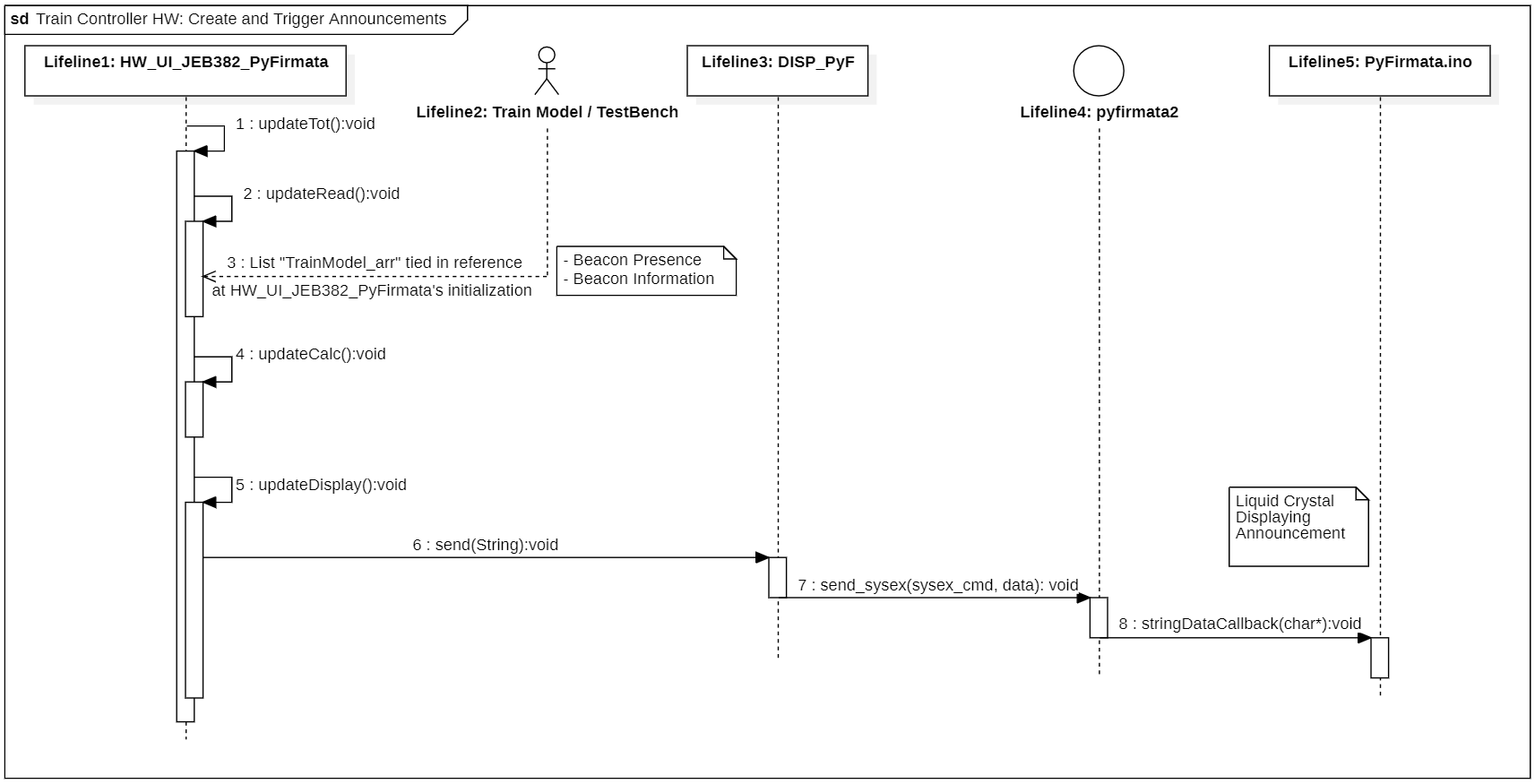


**Figure 6.7.2.6:** Update Doors State Sequence Diagram

* + - 1. **Create and Trigger Announcements**

| Table 6.7.2.7 Create and Trigger Announcements Use Case Description | |
| --- | --- |
| Actors | Train Controller, Train Model |
| Description | 1. Train Model picks up Beacon Information and passes it to the Train Controller which decodes it with an index Look-up-Table to find Station Information including if a Station and it’s name 2. Output to Train Model if there's an announcement and what to Announce on LCD screen |
| Data | 1. String - Beacon Presence and Information that when decoded through an indexed Block Information Look-up-Table gives Station Presence and Name 2. String - Export to LCD screen; Announcement Made; if black (“”), no Announcement made |
| Stimulus | Whenever the Train Model passes over a Beacon and passes it to the Train Controller. |
| Response | The train will either announce an approaching station or remain blank |

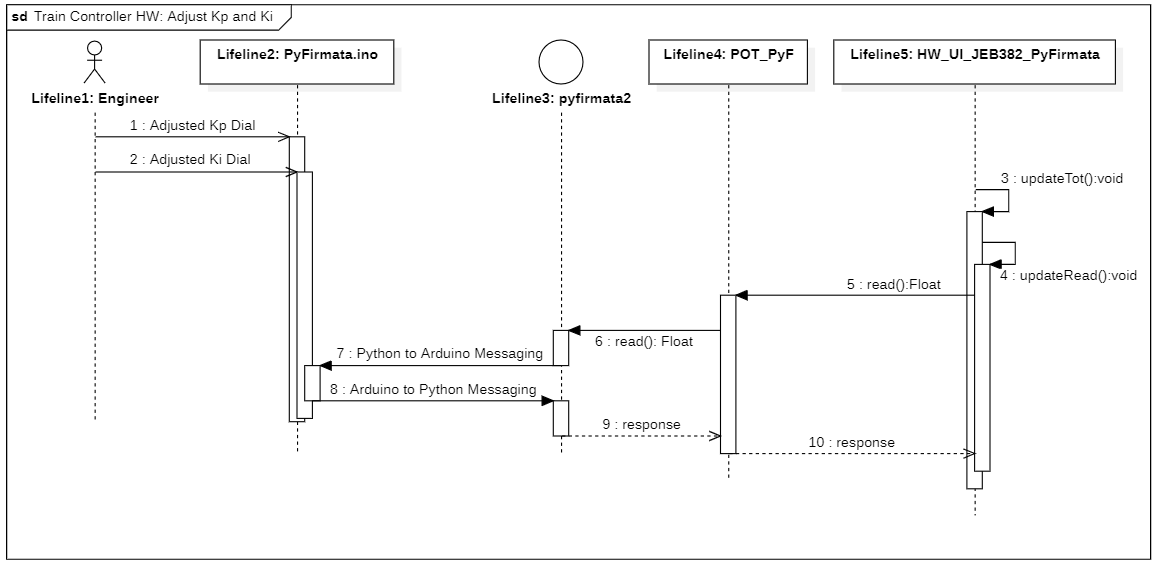
**Table 6.7.2.7:** Create and Trigger Announcements Description

**Figure 6.7.2.7:** Create and Trigger Announcements Sequence Diagram

* + - 1. **Adjust Kp and Ki**

| Table 6.7.2.8 Adjust Kp and Ki Use Case Description | |
| --- | --- |
| Actors | Engineer, Train Controller |
| Description | 1. Engineer unlocks case over the dials for Kp and Ki and connects laptop to system 2. Adjusts them to desired value printed out to laptop’s console 3. Engineer relocks case |
| Data | 1. Floating Point Value - State of Kp Dial on Engineer’s Controller adjusted with set Scale and Boundaries 2. Floating Point Value - State of Ki Dial on Engineer’s Controller adjusted with set Scale and Boundaries |
| Stimulus | Whenever the Engineer adjusts Kp and Ki during maintenance. |
| Response | The train controller will store the data of the dials for future engine power calculations. |

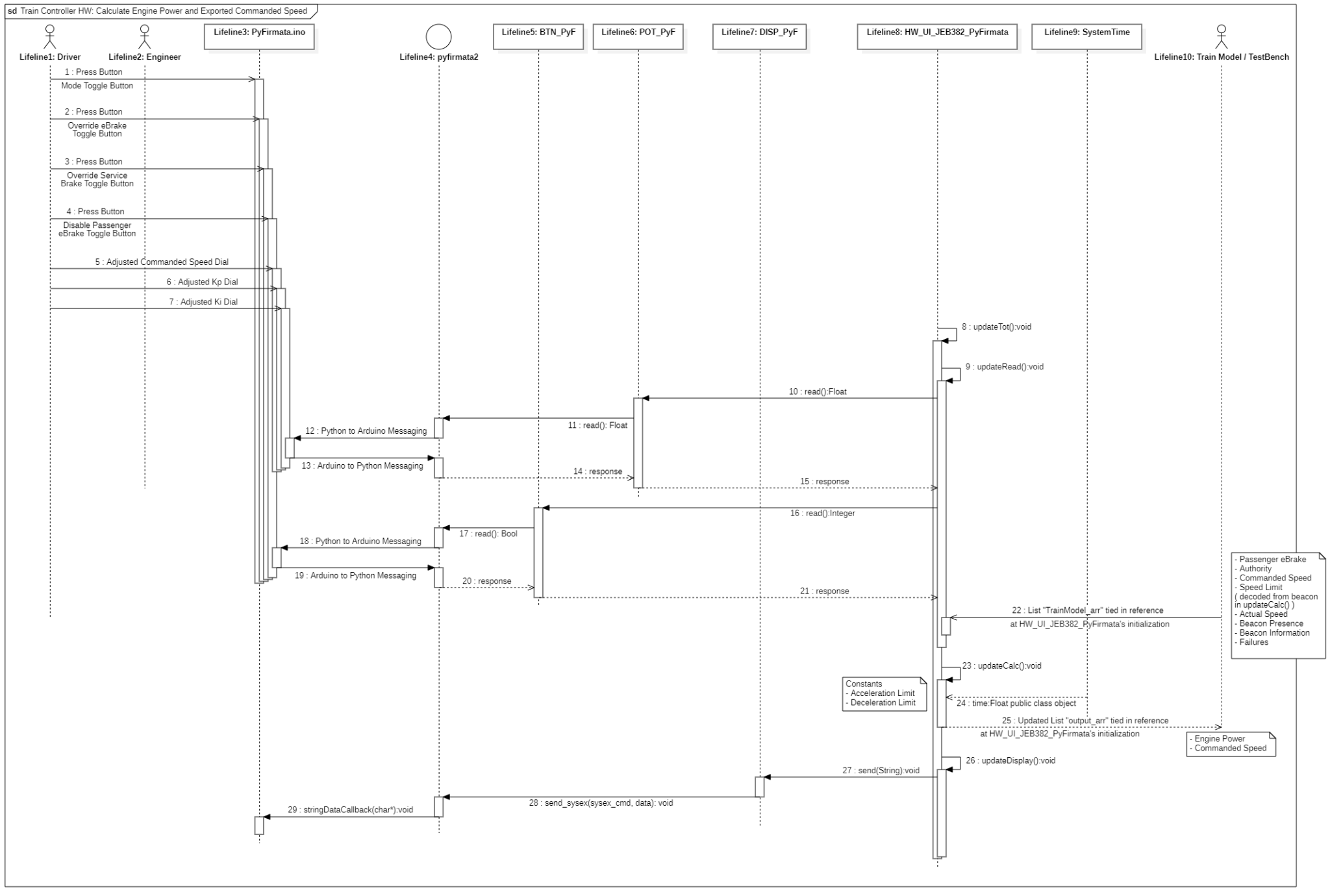
**Table 6.7.2.8:** Adjust Kp and Ki Description

**Figure 6.7.2.8:** Adjust Kp and Ki Sequence Diagram

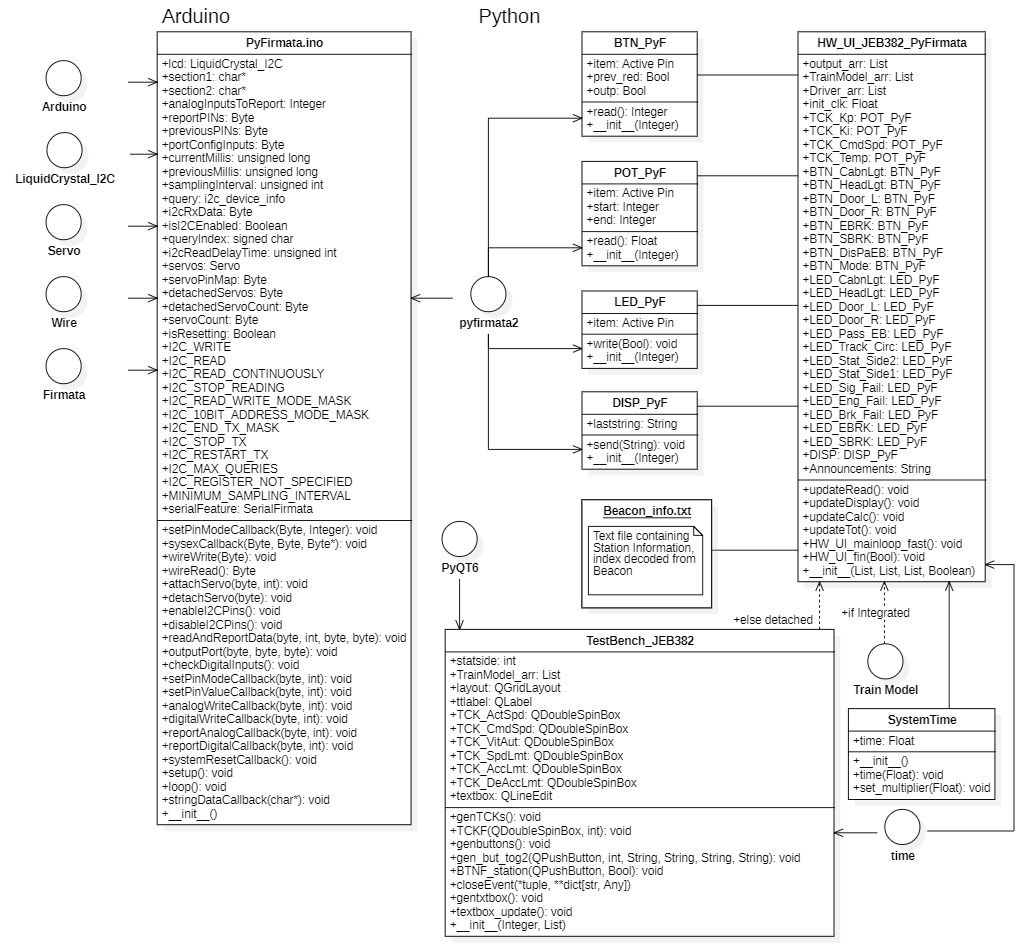
* + - 1. **Calculate Engine Power and Exported Commanded Speed**

| Table 6.7.2.9 Calculate Engine Power and Exported Commanded Speed Use Case Description | |
| --- | --- |
| Actors | Driver, Engineer, Train Controller, Train Model |
| Description | 1. Driver gives Train Controller States of Operation including the Emergency Brake Override, Service Brake Override, Disable Passenger Emergency Brake, and Mode Button Status. Also desired Commanded Speed in Manual Mode. 2. The Train Model gives Passenger Emergency Brake Button Status and relevant Information either from sensors or Beacon information and an indexed Block Information Look-up-Table. Including Authority, CTC Commanded Speed, Speed Limit, and Failure Status. 3. Output to Train Model the correct calculated Engine Power and Commanded Speed and Display Exported Commanded Speed and Actual Speed on LCD screen on Driver Console |
| Data | 1. Integer - Emergency Brake Override Button Status 2. Integer - Disable Passenger Emergency Brake Button Status 3. Integer - Mode Button Status 4. Floating Point Value - State of Commanded Speed Dial on Driver’s Controller adjusted with set Scale and Boundaries; considered in Manual Mode 5. Floating Point Value - State of Kp Dial on Engineer’s Controller adjusted with set Scale and Boundaries 6. Floating Point Value - State of Ki Dial on Engineer’s Controller adjusted with set Scale and Boundaries 7. Floating Point Value - A managed indication of time to calculate acceleration and deceleration to compare to limits 8. Integer - Passenger Emergency Brake Button Status 9. Integer - Authority given by Train Model from CTC 10. Floating Point Value - Commanded Speed given by Train Model from CTC 11. Floating Point Value - Actual Speed 12. String - Beacon Information 13. Floating Point Value - Speed Limit decoded from Beacon Information and an indexed Block Information Look-up-Table 14. Boolean - Signal Failure Status 15. Boolean - Engine Failure Status 16. Boolean - Brake Failure Status 17. Floating Point Value - Exported Engine Power 18. Floating Point Value - Exported Commanded Speed 19. String - Export to LCD screen; Exported Commanded Speed and Actual Speed |
| Stimulus | Whenever the train is in Motion or is desired to be in Motion. |
| Response | Train Controller exports desired Speed and required Engine Power to Train Model. |

**Table 6.7.2.9:** Calculate Engine Power and Exported Commanded Speed Description



**Figure 6.7.2.9:** Calculate Engine Power and Exported Commanded Speed Diagram

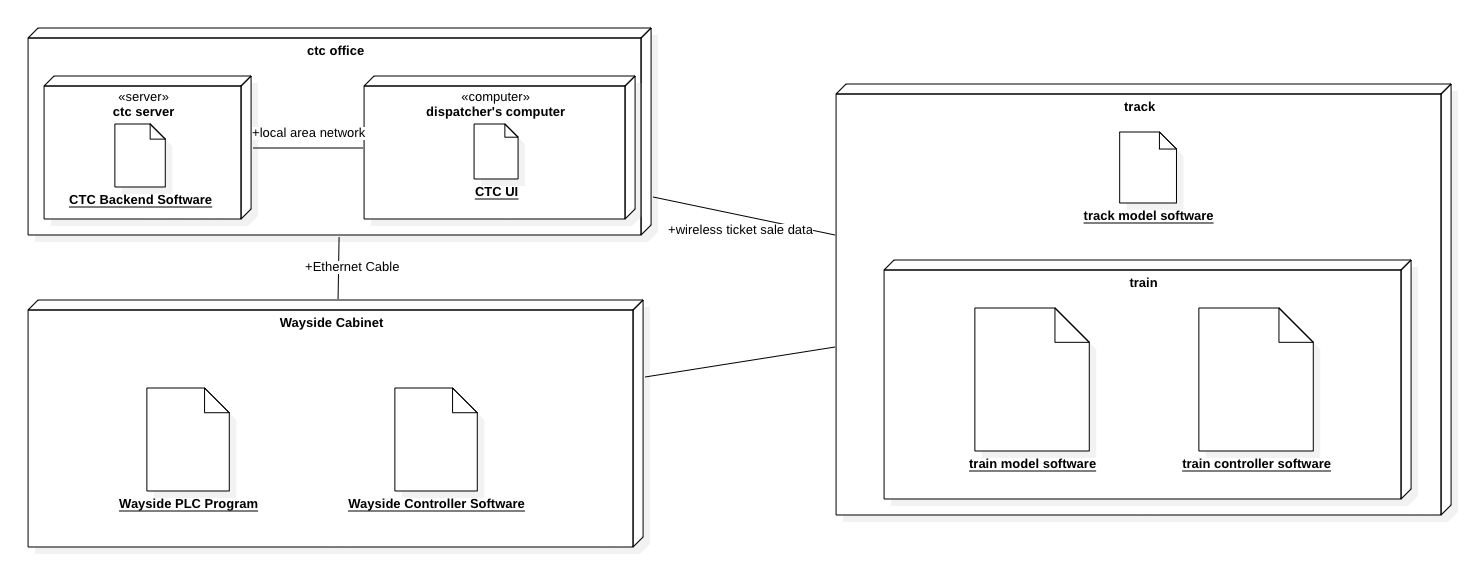
* + 1. **Software Class Diagram**

**Figure 6.7.3:** Train Controller HW Class Diagram

1. **System Deployment**

Although this Train Control System in reality is a simulation being deployed to one PC, in reality, the physical connections must be considered as they constrain the design of the system. The CTC office has only one physical link to the wayside controllers and a wireless networked connection to the Track Model to receive ticket sales data. The clear separation, and grouping of artifacts - what models the physical world, and what controls it - is made clear through the system deployment diagram (Fig. 7.1) - showing where each sub-module is deployed in reality.

* 1. **Deployment Diagram**



**Figure 7.1** System Deployment Diagram