IEEE Software Design Descriptions

Version (1.0)

Team Members:

Karthik Raja

Zachary McPherson

Ricky Zhao

Manelin John Rajan

Thomas Eckrich

Jiahao Li

Original Issue Date: **10/24/2024**

Document Number: **PAAC-01**

Version Number	Author	Date	Reviewed By	Signature
1.0	Karthik Raja	10/22/2024	Ricky Zhao	
1.0	Zachary McPherson	10/22/2024	Karthik Raja	
1.0	Ricky Zhao	10/22/2024	Zachary McPherson	
1.0	Thomas Eckrich	10/22/2024	Jiahao Li	
1.0	Jiahao Li	10/22/2024	Manelin John Rajan	
1.0	Manelin John Rajan	10/22/2024	Thomas Eckrich	

Software Design Descriptions 1.0

Final Approval

Name	Date	Signature
Dr. Joseph Profeta	**10/24/2024**	
Mr. Anuj Lele	**10/24/2024**	
Mr. Thomas Gallagher-Teske	**10/24/2024**	

This document represents the current design (Version 1.0) of the railway control system. As is normal in the design development process, the design details are not complete on the first writing. The Design Analysis phase that follows the SDD will assess the consistency, completeness and feasibility of the design. In addition, during the Design Analysis phase evaluation of third-party products will be performed to determine their fit and usability. After the Design Analysis phase, the SDD will be updated to reflect any necessary changes.

Contents

1. Overview

1.1 Purpose

The purpose of this Software Design Document (SDD) is to present a detailed description of the architecture, system, and key functionalities of the train transit system that will serve the people of Pittsburgh, facilitating their movement around Allegheny County. This document will serve as a reference for both software developers and stakeholders involved in this system as it will be proposed to the Port Authority of Allegheny County (PAAC) for approval upon its completion.

1.2 Scope

The scope of this SDD is to describe the system's different modules, interactions, and vital design decisions. This includes detailing the different diagrams corresponding to each sub-module and how time is represented in the system. Additionally, this document will also cover the interaction between software and hardware modules ensuring that the overall design successfully meets the function and non-functional requirements.

1.3 Intended Audience

The intended audience for this SDD includes:

- **Stakeholders**: To ensure that the system meets the functional and non-functional requirements.
- **Software Developers**: To fully comprehend the system's architecture, design decisions, and implementation.

1.4 Conformance

This SDD document complies with clause 4 and 5 of the IEEE 1016-2009 standards with the following:

- **Clause 4**: Design Description Information

  This clause describes the system's design approaches and decisions such as, how the system is structured, behaves, and interacts with other sub-modules.

- **Clause 5**: Design Viewpoints

  This clause describes the system's requirements and rules that must be met and making sure all parts of the system are part of those requirements.

2. Definitions

For the purpose of this SDD document, the following definitions and terms apply.

2.1 CTC – Centralized Traffic Control

A railway signaling system responsible for managing decision for train routing.

2.2 PLC – Programmable Logic Controller

Industrial-grade computers designed to complete tasks that involve the usage of logic functions.

2.3 PAAC – Port Authority of Allegheny County

The transit agency that operates public transportation in the Allegheny County area.

2.4 COTS – Commercial-Off-The-Shelf

Refers to ready-made hardware or software products that are available to the public.

2.5 Suggested

The values are determined by non-vital systems in accordance with the schedule.

2.6 Commanded

The values determined by vital systems in accordance with safety limitations.

2.7 Setpoint

A target value set as input by the user.

2.8 Beacon

A device located at both ends of each train station that transmits static information to the train via the train’s antenna.

2.9 Block

A length of track consisting of various components.

2.10 Vital

Refers to the systems responsible for maintaining the safety of the system.

2.11 Shall

Expresses certainty about an action that will occur in the future.

2.12 Train Yard

Refers to the origin point where trains begin and end their route. It stores all the trains that run on the railway system.

3. Conceptual Model for Software Design Descriptions

3.1 Software Design in Context

3.1.1 CTC Office

The CTC Office acts as the “brain” of the railway system. From this office, the dispatcher will be able to schedule, route, monitor, and maintain the railway system by communicating via the Wayside Controller. The CTC is responsible for determining suggested speed, authority, and signal commands based on the schedule. Moreover, the dispatcher needs an accurate representation of the track in order to keep track of the trains in the system. Taking these parameters into consideration, the CTC must send suggestions to Wayside Controller while also receiving information about the rest of the system.

3.1.2 Wayside Controller

The Wayside Controller is the logical unit of the system in which processes data and makes vital decisions based on the current status of the track. The Wayside Controller uses PLC programs to handle the commanding of switches, crossings, lights, speed, and authority.

3.1.3 Track Model

The Track Model acts as a software implementation of a physical railway. The track model needs to monitor the occupancies of the blocks on the track as well as current states of the switches, crossings, and lights. Moreover, the track model must act as a medium for communication between the Wayside Controller and the Train Controller. Furthermore, there are three types of failures that may be toggled by Murphy to test the failure response of the railway system. Taking these into consideration, the track model shall accurately represent the commands received from the Wayside, report the current statuses of the track, and relay information to the train model.

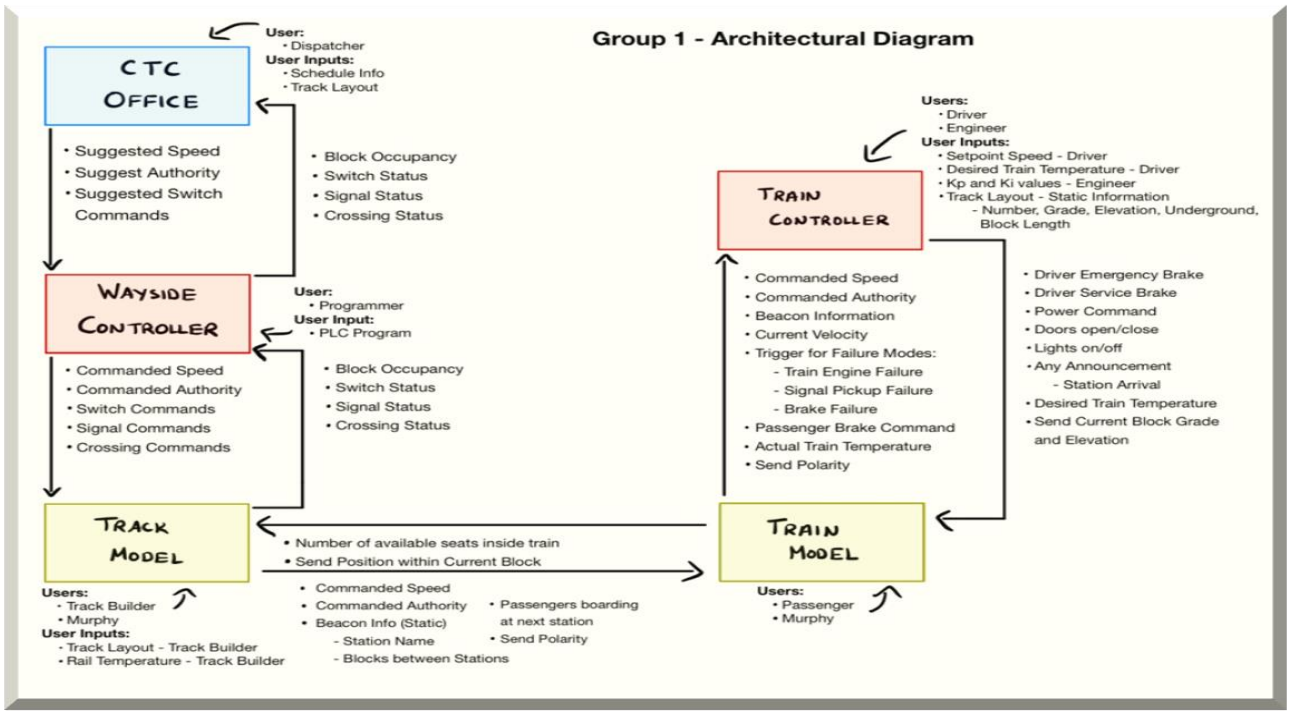
3.1.4 Train Model

The Train Model acts as the software implementation of a physical train. Hence, the train shall monitor the number of passengers as well as display relevant information regarding the train. The train shall accurately reflect the operation of the train according to the commands given by the train controller. This includes receiving a power command and displaying a speed corresponding to the command given. Moreover, information from the Track Model is relayed through the Train Model to the Train Controller as it acts as a medium of communication between the modules. Furthermore, Murphy also acts as a user of this module who commands the individual failure modes within the train.

3.1.5 Train Controller

The Train Controller is a vital module responsible for managing the operation of the train. Its primary function is to regulate train speed to the specified setpoint speed in accordance with the system's speed limits or authority constraints. Utilizing track signals, the module decodes information to determine speed limits and authority values. Moreover, the module manages door operations, lighting, station announcements, and fault monitoring. The engineer sets Kp and Ki values, while the driver adjusts the setpoint speed and desired temperature. The Train Controller ensures safe and efficient train operation by continuously monitoring and responding to various inputs.

3.1.6 System Architectural Diagram



Explanation: Architectural Diagram of the Train Control System, showing all modules, including the Centralized Traffic Controller (CTC), Wayside Controller, Track Model, Train Model, and Train Controller with the interactions, data flow, and inputs/outputs between these subsystems

3.2 Software Design Descriptions within the Life Cycle

In this SDD, the design of the system will be described within the Software Life Cycle. This life cycle is based on IEEE Std 12207-2008 [B21], which describes the various design situations in which an SDD can be created and used.

3.2.1 Influences on SDD Preparation

The main document that influences this document is the Software Requirements Specification (SRS) document. An SRS captures the software requirements that will drive the design and design constraints to be considered or observed. This ensures that the stakeholders and the software developers understand all of the requirements of the system before designing the system.

3.2.2 Influences on Software Life Cycle Products

This document plays an important role for other aspects of the Software Life Cycle:

- **Software Requirements Specifications**: Design decisions, or design constraints discovered during the preparation of the SDD, can lead to requirements changes. Maintaining traceability between requirements and design is one way to record the relationship between the two.
- **Test Documentation**: Test plans can be influenced by the SDD. The development of test cases and test procedures can take into consideration the contents of the SDD.

3.2.3 Design Verification and Design Role in Validation

The design verifications ensure that the software work meets the functional and non-functional requirements of the system. This document will ensure that the following categories are explained:

3.2.3.1 Stakeholders' Design Concerns

The design concerns of the stakeholders have been thoroughly addressed and elaborated upon in Section 4.3, providing a comprehensive and detailed analysis.

3.2.3.2 Follows Functional and Non-Functional Requirements

Refer to the Software Requirements Specifications (SRS) document to clearly understand the functional and non-function requirements of the system provided by the stakeholders. While working on the design decisions, the SRS document was used as a guide to help the group design the system.

- **Functional Requirements**: The design of this system is directly associated with the functional requirements specified in the SRS. The functional requirements explain the main functionalities of the system in which the different modules and the users of those modules interact with each other.
- **Non-Functional Requirements**: Furthermore, the design of this system is directly associated with the non-functional requirements detailed in the SRS, which include performance, security, reliability, and usability.

3.2.3.3 Design Decisions Implementations

The system-wide architecture and design of the train system heavily depends on key aspects such as Routing, Authority, Timing, Wayside Distribution, and the Vitality of the system.

3.2.3.3.1 Routing

In the system, the dispatcher in the CTC routes trains to their destinations. If there is a switch direction that needs to be changed, then the CTC has the ability to request a switch direction change in the track. After receiving the request, the wayside makes the determination if the switch change is safe to do and allows the system to function in a safe manner. In any case of failures, the CTC updates the train routes for the system. As defined in the modification of the track layout, the CTC would need to know if the block is unidirectional or bidirectional in order to route the trains properly.

3.2.3.3.2 Authority

The authority of the train is calculated and passed on from the CTC to the wayside controller. There are two scenarios where the CTC sends a new authority to the wayside controller:

- **When the train route is changed**: The two scenarios where the route of the train would be changed is if there is a failure within the current train route and the CTC has to route around that failure or if there is a change to the schedule of the current route.
- **When the train arrives at its station**: The CTC knows the block occupancies and is able to check if the station block is currently occupied or not. If it detects an occupancy for a long period of time, it can assume that the train has reached its station and send the updated authority to the wayside controller. The wayside controller would then send the commanded authority along the modules to the train controller, which then updates the authority to the commanded authority it received.

- **Wayside decision-making:** If at any point the wayside controller determines it is not safe to go any further, it can update the commanded authority accordingly. For example, if the wayside detects an failure in the block in front of the train, then the wayside will command the train to stop.

- **Train Controller decision-making:** The train controller would change the authority by decreasing it as the train is moving along the track. It can also accept any authority change it receives along the 10-baud communication medium if it is safe to do so (if there are no failure modes active and the emergency brake is not triggered). Whenever, the train controller does receive a new authority and it is in between a block, it should calculate the authority for it to make it to the end of its current block and add that value to the authority that it received and make the result the total commanded authority.

3.2.3.3.3 Timing

The timing in the system is kept by a separate clock timekeeper class outside of all of the modules of the system. The class ensures that all modules in the system are synchronized to the same time reference by emitting regular time steps (ticks).

Key Features:

- **Tick Emissiom**: The Clock emits a tick signal to each module which represents one second passing in the simulation.
- **Pause/Resume Functionality**: The Clock can pause and resume the entire system. When paused, the tick signal will no longer be sent. When resumed the tick signal will continue to be sent out.

- **Simulation Speed Control**: Users can adjust the simulation speed using a slider. This will either increase or decrease the rate at which ticks are sent out.

3.2.3.3.4 Vitality

In the system, the track controller and the train controller are considered to be vital modules, which means that those modules have the ability to make safety concerning decisions for the system to operate in a safe manner. In the railroad and transit industry “vital” means safety critical.

3.2.3.3.4.1 Train Controller

As a vital module, the train controller continuously monitors and makes real-time safety critical decisions that ensures the proper functionality of the train system. Its responsibilities include:

- **Setting upper bound for setpoint speed**: The train driver is not allowed to set a setpoint speed that is higher than the minimum of the commanded speed and the speed limit of the train. If the driver does input a velocity that is higher, then the train controller automatically changes the value that represents the minimum of the commanded speed and the speed limit.

- **Double checking power command**: When the train controller all of the needed inputs to calculate the power command, the train controller find the power command in both imperial and metric units and double checks both results to make sure that the correct result is found.

- **Setting bounds on desired temperature**: The train controller sets lower and upper bounds on the train driver's desired train temperature. The lower bound is set to 60 degrees Fahrenheit and the upper bound is set to 75 degrees Fahrenheit. If the train driver sets a desired temperature outside of this range, then the temperature will not change.
- **Only open doors when train is not moving**: The train driver is only allowed to open the train doors when the current velocity is 0, which means that the train is not moving right now.
- **Having two brake distance values for each brake type**: The train controller is going to calculate two different braking distances: one for the service brake and one for the emergency brake. If the service brakes, do not work and the train cannot start stopping at the correct distance, then we automatically trigger the emergency brake when the authority reaches the braking distance for the emergency brake deceleration value.
- **Reading correct velocity values**: If current velocity is read as negative from the train model, then discard that value and check for next velocity value. If the next velocity is still negative, then hit the emergency brake automatically and stop the train.
- **Making sure data type input is correct**: The input for both the setpoint speed and the desired temperature should be a float value. If there are any characters entered in the input box, then the input will be invalid, and nothing will have as a result of the input.

3.2.3.3.4.2 Track Controller

As a vital module, the track controller continuously monitors and makes real-time safety critical decisions that ensures the proper functioning of the train system. Its responsibilities include:

- **De Morgan's Law in PLC program**: Each PLC program will check all of the inputs and outputs of the module by using De Morgan's law to make sure that negating the input or output two times results in the same value.
- **Managing the Proximity of Trains to One Another:** The wayside will only allow one train to occupy a section of track at a time. An exception to this rule will be when two trains are heading in the same direction. If the train that is farther along the track is situated near the end of the section and about to cross onto the next section, the wayside will allow another train to enter the already occupied section. Furthermore, each section will be divided into thirds. If two adjacent thirds are registered as occupied, the wayside will stop all trains within the section except for the train that is furthest along in the section. Eventually this will safely space out all the trains that were stopped.

- **Managing the Priority of Trains Entering and Exiting a Loop:** The wayside will monitor the current direction of trains within loops to determine which side of the loop a train should enter from. Furthermore, the wayside will monitor the current direction of trains within the sections leading up to a loop. If the sections leading up to the loop have no direction, meaning no trains are moving through

them, or their direction matches the current direction of the loop, then trains will be allowed to exit the loop. Priority will always be given to trains wanting to enter the loop since they are dictating the direction of movement within the sections leading up to the loop.

- **Managing the Switching of Sections:** The wayside will monitor the section occupancy immediately after each switch. Following the rules established in the bullet “Managing the Proximity of Trains to One Another”, the wayside will decide whether or not switching to a particular section is allowed.

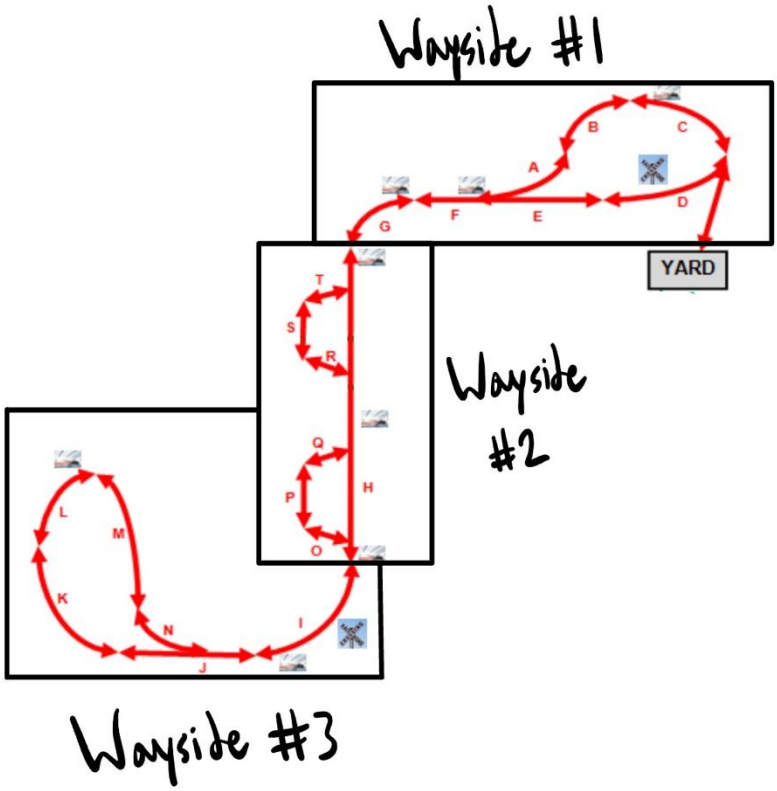
- **Managing the Signaling of Traffic Lights:** The wayside will monitor the occupancy of sections that contain traffic lights. Following the rules established in the bullet “Managing the Proximity of Trains to One Another”, the wayside will decide on whether to send a green, yellow, or red signal. If no trains are occupying the section, then the signal will be set to green. If a train is occupying the section, but is about to exit the section, then the signal will be set to yellow which tells the

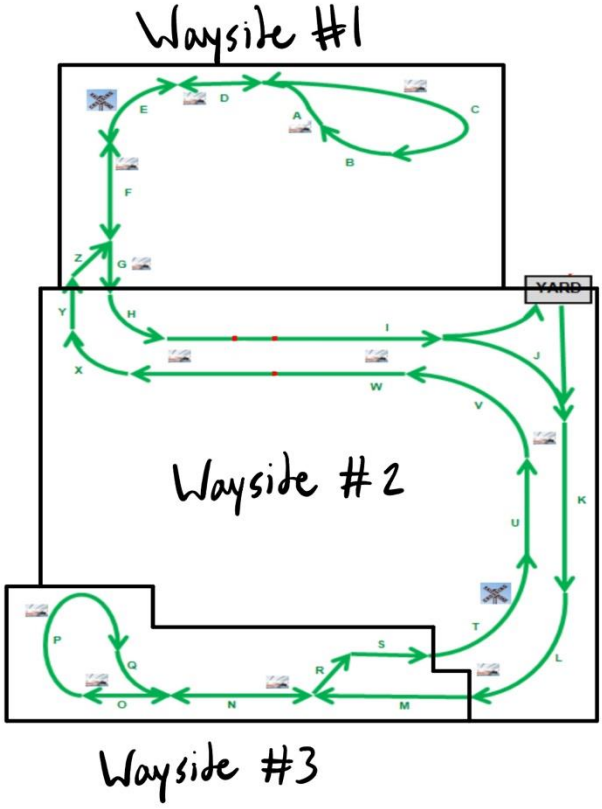
driver of the train attempting to enter the section to slow down to half speed of the commanded speed. The train receiving the yellow signal will return to normal speed once it either receives a new commanded speed or encounters a green signal. If a train is occupying the section but is not about to exit, then the signal will be set to red which tells the driver of the train attempting to enter the section to come to a full stop.

- **Managing a Failure Status Received from the CTC Office:** Once the CTC Office identifies a track failure, it will report which section and the block in that section that failed to the wayside. Using this information, the wayside will force trains within the specified block and any adjacent block to come to a full stop. Furthermore, any trains moving through the specified section and any adjacent section will be forced to a speed of 20 km per hour. This will ensure trains will be able to come to a full stop before they come into contact with the failed block while also allowing movement within the section.

3.2.3.3.5 Wayside Distribution

As seen in the diagrams below, there are going to be three wayside distributions for each line of the system. There will be separate PLC (Programmable Logic Controller) programs for each wayside controller in the system. The modification in the track layout shows that section H in the red line has been split into 5 sub-sections (H1-H5), section J in the red line into 2 sub-sections (J1-J2), and section N in the green line has been split into 2 sub-sections (N1-N2) in order to make the PLC program logic simpler.





Explanation: Diagrams of Each Train Line and Wayside Distributions. The first image shows the layout of Train Red Line, including its corresponding wayside controller distribution. The second image depicts Train Green Line with its own wayside controller distribution. These diagrams illustrate how wayside controllers are distributed along each train line for effective monitoring and control.

3.2.3.3.6 Wayside Language

The Wayside Controller uses python as the programming language of its PLC programs. The highest form of data structure allowed in a PLC program is the array (lists in python). The allowed data types include Boolean for main logic and integers for array indexing. The PLC program cannot process strings or floats. The general hierarchy of a PLC program is as follows: a PLC program has a Wayside object, that Wayside object has an array of Section objects, those Section objects have arrays of Block, Switch, Signal, and Crossing objects.

3.2.3.3.6.1 Protocol Overview

Every communication between the train system and the Wayside Controller is done through arrays of Boolean values (some of which are 2D), standalone Boolean values, and integers that are only used for indexing purposes. Once received, the Wayside Controller will parse the incoming values and update its own arrays of Booleans accordingly.

3.2.3.3.6.2 Signal States

Signal states will be represented by an array of Boolean values representing three possible signal states. The three states are as follows: green (00), yellow (01), and red (10). These values will be stored in a Signal object. If more signal states are required in the future then the size of the 2D array can be expanded depending on the number of required signal states.

3.2.3.3.6.3 Crossing States

Crossing states will be represented by a standalone Boolean value representing two possible crossing states. The two states are as follows: open (0) or closed (1). This value is stored in a Crossing object.

3.2.3.3.6.4 Switch States

Switch states will be represented by a standalone Boolean value representing two possible switch positions. The two positions are as follows: switched left (0) or switched right (1). This value is stored in a Switch object. If future track layouts require more than two switch states then the switch object can hold an array of Boolean, the size of which depends on the number of possible switch positions.

3.2.3.3.6.5 Block Occupancy

Block occupancies will be represented by a standalone Boolean value representing two possible crossing states. The two states are as follows: unoccupied (0) or occupied (1). This value is stored in a Block object.

3.2.3.3.6.6 PLC Code

The PLC code will primarily use if-elif-else statements to perform its main functionality. The PLC code can also use for-loops to parse through any internal arrays or arrays acting as an input or output for other modules.

3.2.3.3.7 Track Layout Modifications

3.2.3.3.7.1 Column for the List of Blocks that the current block is connected to

A column for the list of blocks that the given block is connected to, arranged such that the block immediately preceding the given block appears last, while the remaining blocks are listed in increasing order before that.

3.2.3.3.8 10-Baud Communication Medium

The communication between the track model and the train model is limited to 10-baud. The design being made by the developers is as follows: the first seven bits are used to communicate the commanded speed in binary. In the remaining three bits, there is a switch bit to communicate the status of the switch, a check-status bit that tells the train controller whether or not to check the switch bit, and the check speed bit that determines whether or not the train controller should accept the commanded speed.

4. Design Description Information Content

4.1 Introduction

The following sections shall discuss the required contents of the SDD which include the identification of the SDD, identification of design stakeholders, identification of design concerns, selected design viewpoints that each have type definitions of their allowed design elements/languages, design views, design overlays, and design rationale. The goal is to ensure the system meets the specified requirements while remaining within the bounds of the design constraints.

4.2 SDD Identification

4.2.1 Date of Issue and Status

This SDD was released on Thursday, October 24, 2024, for review by the design stakeholders. Consequently, the status of this document is under review until approval of furtherance. The design stakeholders shall evaluate the design descriptions and give feedback in order to update the status of this document.

4.2.2 Context and Scope

The SDD shall present a detailed description of the architecture, system, and key functionalities of the railway transit system that will facilitate the movement of passenger trains around Allegheny County. Inherently, this document will serve as a reference for both software developers and stakeholders involved in this system as it will be proposed to the Port Authority of Allegheny County (PAAC) for approval upon its completion. Furthermore, the document shall describe the various modules, interactions, and vital design decisions that are orchestrated within the system.

4.2.3 Issuing Organization

As a group, Polar Express presents the documentation, design, and implementation for the railway transit system.

4.2.4 Authorship

The authors of this document are Karthik Raja, Zachary McPherson, Manelin John Rajan, Thomas Eckrich, Jiahao Li, and Ricky Zhao who are all upperclassmen in the Electrical and Computer Engineering (ECE) department at the University of Pittsburgh's Swanson School of Engineering.

4.2.5 References

[1] “Flexity 2 Tram Datasheet,” Part datasheet, November 2009

[2] IEEE. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

[3] J. Profeta, “System and Project Engineering,” ECE1140, University of Pittsburgh, [Document], September 2024. Available: https://canvas.pitt.edu/courses/272320/files/folder/Grading%20Rubrics

[4] J. Profeta, “System and Project Engineering,” ECE1140, University of Pittsburgh, [PowerPoint slides], September 2024. Available: https://canvas.pitt.edu/courses/272320/files/folder/Lectures

[5] J. Profeta, “System and Project Engineering,” ECE1140, University of Pittsburgh, [PowerPoint slides], September 2024. Available: https://canvas.pitt.edu/courses/272320/files/folder/Labs

4.2.6 Design Language

Unified Modeling Language, or UML for short, shall be used throughout the design of the project. As a standardized visualization tool, this design language will depict the structure and behavior of the railway transit system. Through the use of various types of diagrams including sequence, class, and use case diagrams, stakeholders will be able to view and understand the overall architecture of the project. UML enables the project engineers of Polar Express to clearly communicate complex topics in a straightforward manner to stakeholders which allows for the effective facilitation of iterations and project maintenance.

4.2.7 Revision History

As this is Version 1.0 of the SDD document, revision history is not available. Revisions shall be made after design reviews with the stakeholders and shall be documented accordingly in this section.

4.2.8 Summary

- The current version of this document is Version 1.0.
- The document shall be issued on Thursday, October 24th, 2024, at 18:00 EST.
- The status of the document on the date of issuance is under review and waiting for approval.
- The official issuing organization of this document is the group Polar Express.
- The authors of this document are the members of Polar Express: Karthik Raja, Zachary McPherson, Manelin John Rajan, Thomas Eckrich, Jiahao Li, and Ricky Zhao.
- The document shall be distributed to the stakeholders of the transit system for review.
- The revisions mentioned above are based on feedback from stakeholders in previous stages of the project.

4.3 Design Stakeholders and Their Concerns

The Design Stakeholders for this project are the Port Authority of Allegheny County, Passengers of the Transit System, and the Employees who will interact directly with the project software.

4.3.1 Port Authority of Allegheny County

The design concerns of the PAAC include:

- **Cost Efficiency**: The system shall comply with the company's budget while still holding full functionality and still meeting the operational needs.
- **Safety and Compliance**: The system shall comply with the safety regulation laws and the PAAC rules.
- **System Reliability**: The system shall ensure consistent and dependable train operations, minimizing downtime and service interruptions.

4.3.2 Passengers

The design concerns of the passengers of the include:

- **Safety**: The system shall ensure the passenger's safety while riding including emergency stops and quick response mechanisms.

4.3.3 Dispatcher

The design concerns of the train dispatcher include:

- **Real-Time Control**: The driver shall be able to dispatch trains and monitor the track via the block occupancies, switch statuses, signal statuses, and crossing statuses.
- **Conflict Resolution**: The CTC shall be able to put any blocks in maintenance mode and route trains around any blocks in maintenance mode.

4.3.4 PLC Programmer

The design concerns of the PLC Programmer include:

- **Integration of Control Systems**: The PLC programmer shall ensure that the PLC program is integrated with the switch, signal, and crossing statuses and allow for the train to travel safely throughout the track.

4.3.5 Track Builder

The design concerns of the Track Builder include:

- **System Coordination**: The track builder ensures their physical track designs align with the software's digital control systems.
- **Track Availability**: The track builder shall receive timely alerts from the system for when a block failure has occurred.

4.3.6 Train Driver

The design concerns of the Train Driver include:

- **User-Friendly Interfaces**: The train driver shall have a simple and intuitive UI that minimizes the possibility of human error.
- **Safety Mechanisms**: The train driver shall have access to emergency override options in case of signal failures or system issues.

4.3.7 Train Engineer

The design concerns of the Train Engineer include:

- **System Stability**: The train engineer shall ensure that the selected Kp and Ki values provide smooth and stable control of the train
- **Ease of Adjustment**: The train engineer shall have user-friendly interfaces that allow for easy adjustment of these control parameters without deep system reconfiguration.

4.4 Design Views

The stakeholders are responsible for providing requirements for the system; however, these requirements can only highlight the details and influence the perspective of the project. Each identified design concern acts as the topic of at least one design view. The design views are comprised of various facets of the project in which design elements are categorized. The following sections shall detail the different design views employed in this project through the language of design viewpoints, elements, and rationale.

4.5 Design Viewpoints

4.5.1 Logical View

The key abstractions being implemented in this project are the CTC Office, Wayside Controller, Track Model, Train Model, and Train Controller. These major system components are comprised of various design elements that allow the end-users to perform a variety of tasks – essentially giving functionality to the whole system. In a logical sense, overlapping design elements shall have a place to be stored or communicated within each adjacent module. Consequently, variable types and names shall be maintained throughout the modules to keep the system concise and comprehensive.

4.5.2 Physical View

The system is comprised of multiple software modules and a single hardware module that are tasked to different members of the group; however, they shall be designed with integration guidelines. In other words, the modules shall be deployed into a single program that runs on a Windows 10 device. This suggests that the system components must work in tandem with one another to simultaneously calculate data and display information, accordingly.

4.5.3 Development View

The architecture of the system was determined by the stakeholders in a manner that requires communication among various system components in order to function as intended. The system is fronted by the CTC Office which directly communicates to the Wayside Controller. Then, the Wayside Controller is directly responsible for controlling the operation of the Track Model. Next, the Track Model communicates to the Train Model via signals sent and received through antennas. Finally, the Train Model is directly controlled by the Train Controller — thus, completing the architecture. Evidently, the design bounds the developers by restricting the mediums of communication via specific modules and by limiting the throughput of such forms of communication. Changes to information handled in each module may be encountered via additional information provided throughout the life cycle of the project.

4.5.4 Process View

In the beginning phases of the project, each member is tasked with a system module: Thomas Eckrich to the CTC, Zachary McPherson to the Wayside Controller, Ricky Zhao to the Track Model, Jiahao Li to the Train Model, Karthik Raja to the software Train Controller, and Manelin John Rajan to the hardware Train Controller. Each module is provided with different information that is either handled internally or externally as part of the overall system functionality.

4.6 Design Elements

The design elements refer to any component or aspects that are in the system. These elements include entities, relationships, and attributes of the system:

- **Design Entities**: Key components in the system, such as the trains, the track, the user, and the module interfaces.
- **Design Relationships**: Relationships include the communication and interactions between sub-system components, such as between the CTC and the wayside controller, the wayside controller and the track model, the track model and the train model, and the train model and the train controller.
- **Design Attributes**: Characteristics of the entities, such as commanded speed, commanded authority, switch statues, signal statuses, and beacon information.

All design elements are detailed in the corresponding sections under the viewpoints described in clause 5 of this document.

4.7 Design Overlays

The design overlays are usually used to add information to the existing views. This concept will be visualized clearly, when necessary, in the design viewpoints section of this document (Clause 5).

4.8 Design Rationale

The structure of this project requires intermodular communication amongst each other. However, the architecture introduces a design constraint such that data is scattered across modules and is required to be handled in accordance with the flow of the system. Therefore, the design and data structures employed by the developers were chosen in such a way that allows information to be relayed across modules, stored, or calculated to carry out various system functionalities.

4.9 Design Languages

Unified Modeling Language, or UML for short, shall be used throughout the design of the project. As a standardized visualization tool, this design language will depict the structure and behavior of the railway transit system. Through the use of various types of diagrams including sequence, class, and use case diagrams, stakeholders will be able to view and understand the overall architecture of the project. UML enables the project engineers of Polar Express to clearly communicate complex topics in a straightforward manner to stakeholders which allows for the effective facilitation of iterations and project maintenance.

5. Design Viewpoints

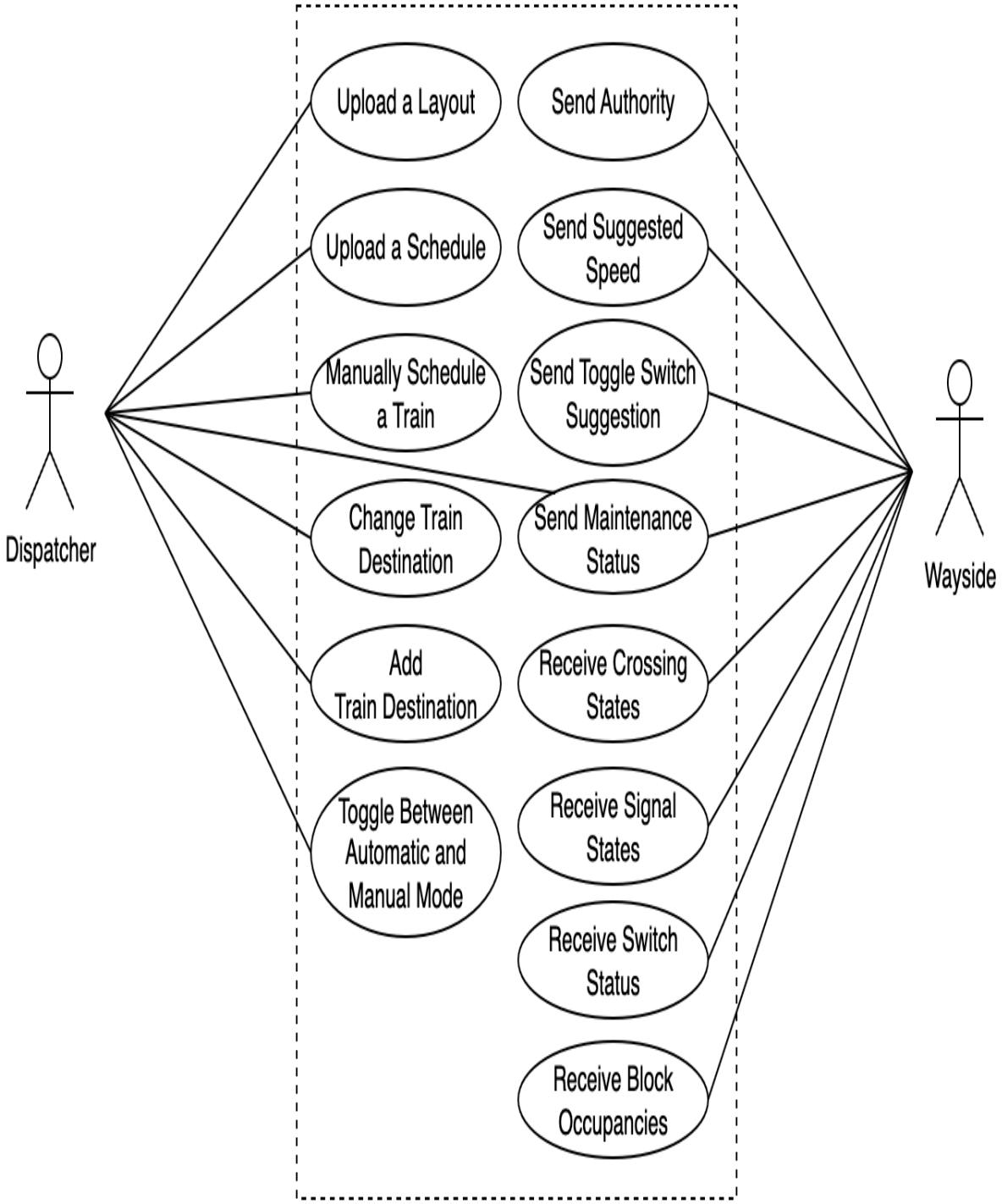
5.1 Introduction

This section outlines the primary design viewpoints used in the Software Design Description (SDD) to address the system's design concerns. It will include essential diagrams such as the use case diagrams with detailed use case descriptions, class diagrams, sequence diagrams, and deployment diagrams.

5.2 Context Viewpoint

The Context Viewpoint provides a high-level view of the system, depicting the interactions between the different sub-systems and external actors, such as users and other sub-systems, within its environment. This viewpoint adopts a "black box" approach, focusing on the services and functionalities offered by the system without diving into implementation details.

5.2.1 CTC Use Case Diagram



5.2.1.1 CTC Use Case Descriptions

Use Case 1: Upload a Schedule

Actor	Dispatcher
Description	The Dispatcher has the ability to upload a schedule in automatic mode.
Precondition	The CTC has to be in automatic mode.
Basic Sequence	1. In automatic mode, the Dispatcher can upload a schedule that they would like for the trains to follow.
Exceptions	The format of the input file is incorrect.
Data	Line – String Destination – Int Station Name – String Departure Time - Float Arrival Time - Float
Stimulus	The Dispatcher selects the operational mode to be automatic and selects the “Upload Schedule” button in the interface and selects the schedule they would like to upload.
Post Conditions	The CTC stores the schedule of the train without the Dispatcher needing to dispatch each and every train manually.

Use Case 2: Upload Layout

Actor	Dispatcher
Description	The Dispatcher uploads the layout of a line to the system.
Precondition	The system state is valid.
Basic Sequence	1. The Dispatcher Uploads an excel file representing the layout for a given line.
Exception	The format of the input file is incorrect.
Data	Block Number – Int Block Length – Int Section – String Station – String Switch – Bool Signal – Bool Crossing – Bool
Stimulus	The Dispatcher clicks the ‘Upload Layout’ button.
Post Conditions	The track layout information for a given layout is stored.

Use Case 3: Manually Schedule a Train

Actor	Dispatcher
Description	The Dispatcher manually schedules a single train at a time.
Precondition	The CTC has to be in manual mode.
Basic Sequence	1. In manual mode, the Dispatcher has the ability to manually route trains to their destination. 2. The Dispatcher can pick what train is going to what station.
Exceptions	If the CTC is in automatic mode, then the Dispatcher is unable to manually dispatch a train, since the CTC would be following the uploaded schedule instead.
Data	Line - String Destination – Int Station Name – String Departure Time – Float Arrival Time - Float
Stimulus	The Dispatcher clicks the ‘Schedule Train’ button on the UI.
Post Conditions	A new train is created with the given parameters.

Use Case 4: Change Train Destination

Actor	Dispatcher
Description	The Dispatcher has the ability to change a train's destination while the train is moving.
Precondition	The train must be dispatched already for the Dispatcher to enact changes.
Basic Sequence	1. The Dispatcher would select the train they would like to change the destination of. 2. Then, the Dispatcher would change the destination of the corresponding train and update it for the train.
Exception	If the station is inaccessible due to maintenance, then the change of destination is not valid.
Data	Line – String Destination – Int Station Name – String Arrival Time - Float
Stimulus	Dispatcher clicks the 'Change Destination Button'
Post Conditions	The train will go to the updated destination after the Dispatcher has confirmed the changes.

Use Case 5: Add Train Destination

Actor	Dispatcher
Description	Dispatcher adds an additional stop for a train.
Precondition	The train must already exist and have at least 1 stop already.
Basic Sequence	1. The Dispatcher clicks the button to add a new stop. 2. The given train has an extra stop added to its list.
Exception	Destination must be valid, Arrival Time to the new station must be after other existing stops.
Data	Destination – Int Station Name – String Arrival Time - Float
Stimulus	Dispatcher clicks the ‘Add Stop’ button
Post Conditions	The selected train will include the given stop in its destination list.

Use Case 6: Toggle Between Automatic and Manual Mode

Actor	Dispatcher
Description	The Dispatcher can switch between automatic and manual operational modes.
Precondition	The current operational state must be defined.
Basic Sequence	1. The Dispatcher wants to manually or automatically schedule a train. 2. The Dispatcher clicks the toggle button to switch to the correct mode.
Exception	If no schedule has been uploaded, automatic mode cannot be activated.
Data	Mode - Bool
Stimulus	Dispatcher clicks the ‘Automatic Mode’ toggle button
Post Conditions	The CTC will operate in either manual or automatic mode, based on the selection of the Dispatcher.

Use Case 7: Send Suggested Authority

Actor	Wayside Controller
Description	CTC calculates the authority for trains and sends updates to the Wayside.
Precondition	A train must be dispatched from the yard.
Basic Sequence	1. The CTC receives block occupancies. 2. The CTC calculates the suggested authority for each train dispatched. 3. If any authorities have been updated then the new values will be sent to the Wayside.
Exceptions	No trains have been dispatched or no block occupancies are reported.
Data	Block Occupancies – Array of Booleans Suggested Authority– Int
Stimulus	The Wayside sends block occupancies.
Post Conditions	The authority is passed through the modules to the Train Controller.

Use Case 8: Send Suggested Speed

Actor	Wayside Controller
Description	CTC calculates the authority for trains and sends updates to the Wayside
Precondition	A train must be dispatched from the yard.
Basic Sequence	1. The CTC receives block occupancies. 2. The CTC calculates the suggested speed for each train dispatched. 3. If and speeds have been updated then the new values will be sent to the Wayside.
Exceptions	No trains have been dispatched or no block occupancies have been reported.
Data	Blook Occupancies – Array of Booleans Suggested Speed – Int
Stimulus	The Wayside sends block occupancies.
Post Conditions	The suggested speed is passed through the modules to the Train Controller.

Use Case 9: Send toggle switch suggestions

Actor	Wayside Controller
Description	CTC will check whether a switch change should be made to keep the trains on course.
Precondition	A train must be dispatched from the yard and has a destination.
Basic Sequence	1. Receive block occupancies from the wayside. 2. Check the locations of the trains, the paths of the trains, and the status of the switches. Calculate if switch changes are necessary. 3. If switch changes are necessary send the request to the wayside.
Exceptions	No trains have been dispatched, or no block occupancies have been reported, or the switch states are unknown.
Data	Block Occupancies – Array of Booleans Switch States – Array of Booleans Suggested Switch Command – String, Int, Bool
Stimulus	The Wayside sends block occupancies.
Post Conditions	The Wayside Controller receives the suggested switch command and decides to see if it is safe to change the switch status.

Use Case 10: Send Maintenance Status

Actor	Wayside Controller, Dispatcher
Description	The Dispatcher puts a block in maintenance mode and that information is sent to the Wayside.
Precondition	The layout must be uploaded.
Basic Sequence	1. If the Dispatcher recognizes a change in a block failure state, then they toggle the maintenance status of that block. 2. The maintenance value of the block is updated, and the maintenance value is sent to the wayside.
Exceptions	The dispatcher attempts to send a maintenance status signal, but there are no blocks currently in maintenance mode.
Data	Maintenance Status - Bool
Stimulus	Dispatcher clicks the maintenance mode toggle button.
Post Conditions	The Wayside uses the extra maintenance information to make vital decisions.

Use Case 11: Receive Crossing States

Actor	Wayside Controller
Description	The CTC receives crossing states from the Wayside Controller and stores them.
Precondition	The layout needs to be uploaded.
Basic Sequence	1. The Wayside Controller sends all crossing states to the CTC. 2. The CTC receives the updated crossing values and stores them locally.
Exceptions	The track layout has not been uploaded or is incorrect.
Data	Crossing State - Booleans
Stimulus	The Wayside sends an array of crossing states.
Post Conditions	The CTC would have the updated values of the transit system.

Use Case 12: Receive Signal States

Actor	Wayside Controller
Description	The CTC receives signal states from the Wayside Controller and stores them.
Precondition	The layout needs to be uploaded.
Basic Sequence	1. The Wayside Controller sends all signal states to the CTC. 2. The CTC receives the updated signal values and stores them locally.
Exceptions	The track layout has not been uploaded or is incorrect
Data	Signal State – Booleans
Stimulus	The Wayside sends an array of signal states.
Post Conditions	The CTC would have the updated values of the transit system.

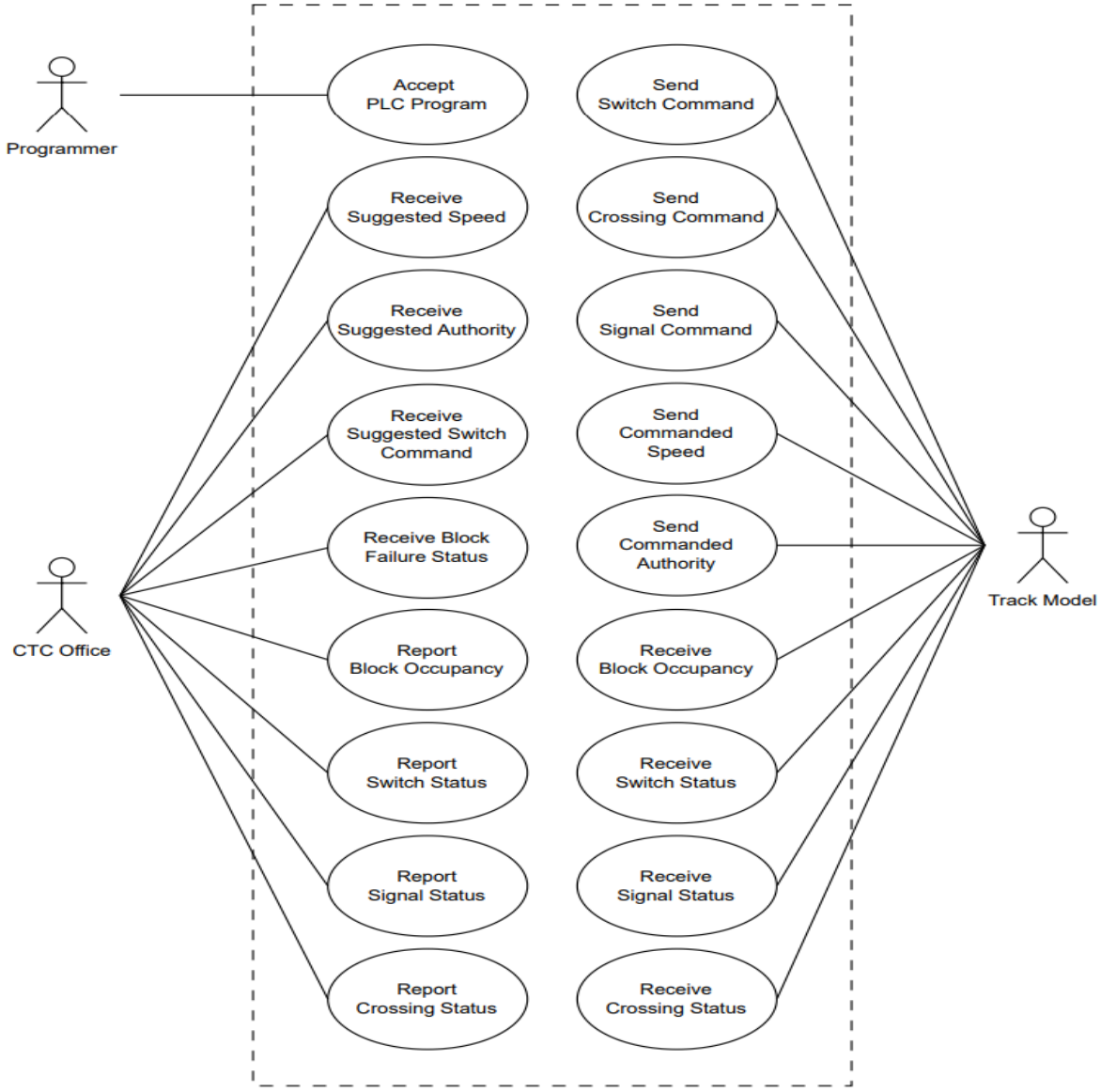
Use Case 13: Receive Switch Status

Actor	Wayside Controller
Description	The CTC receives switch statuses from the Wayside Controller and stores them.
Precondition	The layout needs to be uploaded.
Basic Sequence	1. The Wayside Controller sends all switch states to the CTC. 2. The CTC receives the updated switch values and stores them locally.
Exceptions	The layout has not been uploaded or is incorrect.
Data	Switch Status - Boolean
Stimulus	The Wayside sends an array of switch states.
Post Conditions	The CTC would have the updated values of the transit system.

Use Case 14: Receive Block Occupancies

Actor	Wayside Controller
Description	The CTC receives block occupancies from the Wayside Controller and stores them.
Precondition	The layout needs to be uploaded.
Basic Sequence	1. The Wayside Controller sends all block occupancies to the CTC. 2. The CTC receives the updated occupancies and stores them locally.
Exceptions	The layout has not been uploaded or is incorrect.
Data	Block Occupancies - Boolean
Stimulus	The Wayside sends an array of block occupancies.
Post Conditions	The CTC would have the updated values of the transit.

5.2.2 Wayside Controller Use Case Diagram



Explanation: Use Case Diagram for the Wayside Controller, illustrating the interactions between the Wayside Controller and its actors, such as the CTC, Track Model, and the PLC Programmer. This diagram highlights key use cases, including uploading PLC programs, track management, and reporting the status of the track.

5.2.2.1 Wayside Controller Use Case Descriptions

Use Case 1: Accept PLC Program

Actor	Programmer
Description	The Programmer shall be able to upload a PLC script to the system to carry out logical algorithms.
Precondition	PLC Program is written and uploaded to the system.
Basic Sequence	1. The Programmer presses the “Upload PLC” button. 2. Then, they are prompted to upload a file containing the PLC program.
Exceptions	Throws an error if the file type is not valid.
Data	- Uploaded_file: python file
Stimulus	The file upload button is pressed.
Post Conditions	The PLC Program may now handle the logical processes of the railway track operation.

Use Case 2: Receive Suggested Speed

Actor	CTC Office
Description	The CTC Office communicates a suggested speed of the train based on the schedule for that train.
Precondition	A train is dispatched onto the track.
Basic Sequence	1. The CTC determines the suggested speed of the train and encodes it into binary for the Wayside Controller. 2. The Wayside Controller stores and processes the suggested speed.
Exceptions	Throws an error if the data type is invalid.
Data	- sugg_speed: pyqtSignal
Stimulus	The CTC determines a suggested speed for a train based on the schedule information.
Post Conditions	The suggested speed is communicated with the Wayside and processed within the PLC Program.

Use Case 3: Receive Suggested Authority

Actor	CTC Office
Description	The CTC Office communicates a suggested authority of the train based on the schedule for that train.
Precondition	A train is dispatched onto the track.
Basic Sequence	1. The CTC determines the distance the train may travel before it should stop according to the schedule. 2. This value is encoded and sent to the Wayside Controller.
Exceptions	Throws an error if the data type is invalid.
Data	- sugg_authority: pyqtSignal
Stimulus	The CTC determines a suggested authority for a train based on the schedule information.
Post Conditions	The suggested authority is communicated with the Wayside and processed within the PLC Program.

Use Case 4: Receive Suggested Switch Command

Actor	CTC Office
Description	The CTC Office communicates a suggested switch orientation based on the routing of trains.
Precondition	A train has been dispatched onto the track.
Basic Sequence	1. The CTC determines how the switch should be oriented for the train to follow the route on the schedule. 2. The value is encoded and sent to the Wayside Controller.
Exception	Throws an error if data type is invalid.
Data	- sugg_switch_cmd: pyqtSignal
Stimulus	The CTC wants to route a train to a different section of track.
Post Conditions	The suggested switch orientations are communicated to the Wayside Controller to be processed by the PLC.

Use Case 5: Receive Block Failure Status

Actor	CTC Office
Description	The CTC processes the list of bock occupancies and determines which blocks are in failure.
Precondition	The CTC has retrieved the list of block occupancies.
Basic Sequence	1. The CTC checks the block occupancies and determines which occupancies are the result of failures on the blocks. 2. The CTC reports the blocks in failure to the Wayside Controller.
Exception	The Wayside fails to send the block occupancies received from the Track Model.
Data	- section_failure_index: pyqtSignal - block_failure_index: pyqtSignal
Stimulus	The needs to know the failures on the track in order to put blocks into failure.
Post Conditions	Boolean signals shall be relayed to the Wayside Controller to be handled by the PLC algorithm.

Use Case 6: Report Block Occupancies

Actor	CTC Office
Description	The Wayside Controller shall send a list of current block occupancies to the CTC Office.
Precondition	The Wayside Controller retrieves the list of block occupancies from the Track Model.
Basic Sequence	1. The Wayside Controller relays the occupancies to the CTC Office.
Exception	
Data	- block_occ: pyqtSignal
Stimulus	The CTC needs to know the block occupancies in order to route trains and toggle maintenance mode.
Post Conditions	The CTC receives the list of block occupancies.

Use Case 7: Report Switch Status

Actor	CTC Office
Description	The Wayside Controller shall report the current state of the switches along the track.
Precondition	The Track Model reports the switch status to the Wayside Controller.
Basic Sequence	1. The Wayside Controller sends the switch statuses to the CTC in the form of Boolean signals.
Exceptions	
Data	- switch_status: pyqtSignal
Stimulus	The CTC needs to know how the switches are oriented along the track to make the necessary changes when routing trains.
Post Conditions	The CTC receives the current switch states to compare with the suggested switch state according to the schedule.

Use Case 8: Report Signal Status

Actor	CTC Office
Description	The Wayside Controller shall report the current states of the traffic lights along the track.
Precondition	The Track Model reports the traffic light statuses to the Wayside Controller.
Basic Sequence	1. The Wayside Controller sends the traffic light statuses to the CTC in the form of Boolean signals.
Exceptions	
Data	- signal_status: pyqtSignal
Stimulus	The CTC needs to know the states of the traffic lights along the track to make the necessary changes when routing trains.
Post Conditions	The CTC receives the current traffic light states.

Use Case 9: Report Crossing Status

Actor	CTC Office
Description	The Wayside Controller shall report the current state of the crossings along the track.
Precondition	The Track Model reports the railway crossing status to the Wayside Controller.
Basic Sequence	1. The Wayside Controller sends the railway crossing statuses to the CTC in the form of Boolean signals.
Exceptions	
Data	- crossing_status: pyqtSignal
Stimulus	The CTC needs to know the operational state of the railway crossings along the track to make the necessary changes when routing trains.
Post Conditions	The CTC receives the current crossing states.

Use Case 10: Send Switch Command

Actor	Track Model
Description	The Track Controller shall determine a switch command given the suggested switch orientation from the CTC while considering the safety limitations of the track.
Precondition	The PLC program must be uploaded to the system.
Basic Sequence	1. The CTC communicates the orientation in which the switch should face given the schedule. 2. The Wayside Controller receives this suggestion and does calculations based on the current state of the track to make sure the suggestion does not interfere with the safety of the system. 3. The Wayside Controller sends result of the calculation to the Track Model.
Exceptions	
Data	- cmd_switch_cmd: pyqtSignal
Stimulus	The CTC wants to route the train to another section of track.
Post Conditions	The Track Model makes necessary changes to reflect the new switch status.

Use Case 11: Send Crossing Command

Actor	Track Model
Description	The Track Controller shall determine a crossing command for the railway crossings along the track.
Precondition	The PLC program must be uploaded to the system.
Basic Sequence	1. The Wayside Controller determines if a train is within the section and will occupy a crossing. 2. If there is a train within the section, the crossing gates will lower. 3. If there isn't a train, the crossing gates are open. 4. The Wayside Controller sends this commanded state of the crossing to the Track Model
Exceptions	
Data	- crossing_cmd: pyqtSignal
Stimulus	The Wayside Controller makes a vital decision to prevent collisions at the crossings.
Post Conditions	The Track Model makes necessary changes to reflect the new crossing status.

Use Case 12: Send Signal Command

Actor	Track Model
Description	The Track Controller shall determine the proper light command for the traffic lights along the track.
Precondition	The PLC program must be uploaded to the system.
Basic Sequence	1. The Wayside Controller determines the states of the lights according to the operational states of the switches. 2. Green means the train can safely proceed. 3. Yellow suggests that it is safe to proceed; however, should slow down to maintain a safe distance from another train. 4. Red means that the train can no longer proceed for whatever reason. 5. The Wayside Controller sends these commanded signals to the Track Model
Exceptions	
Data	- signal_cmd: pyqtSignal
Stimulus	The Wayside Controller makes a vital decision to prevent collisions.
Post Conditions	The Track Model makes necessary changes to reflect the new light statuses.

Use Case 13: Send Commanded Speed

Actor	Track Model
Description	The Track Controller shall determine a commanded speed according to the safety limitations of the track.
Precondition	The current trajectory of the train requires recalculation of a commanded speed.
Basic Sequence	1. The Wayside Controller determines a commanded speed by calculating a speed that adheres to the safety limitations of the track. 2. This value is encoded into binary and sent to the Track Model.
Exceptions	
Data	- cmd_speed: pyqtSignal
Stimulus	The train needs to know the safe speed to travel at regarding the safety limitations of the track.
Post Conditions	The Track Model receives and stores the commanded speed.

Use Case 14: Send Commanded Authority

Actor	Track Model
Description	The Track Controller shall determine a commanded authority according to the condition of the track.
Precondition	A current state of the track requires a new commanded authority to be calculated to prevent accident.
Basic Sequence	1. The Wayside Controller determines a commanded authority by calculating the distance the train shall stop at to avoid collision or successfully reach a station. 2. This value is encoded into binary and sent to the Track Model.
Exceptions	
Data	- cmd_authority: pyqtSignal
Stimulus	An event occurs on the track that requires the train to stop at a new position.
Post Conditions	The Track Model receives and stores the commanded authority.

Use Case 15: Receive Block Occupancy

Actor	Track Model
Description	The Track Controller shall receive a list of block occupancies from the Track Model.
Precondition	A current state of the track requires a new commanded authority to be calculated to prevent accident.
Basic Sequence	1. The Track Model reports all the current block occupancies to the Wayside Controller.
Exceptions	
Data	- block_occ: pyqtSignal
Stimulus	The Wayside Controller needs receive the block occupancies to perform logical operations on them.
Post Conditions	The Wayside Controller knows which blocks are occupied along the track.

Use Case 16: Receive Switch Status

Actor	Track Model
Description	The Track Controller shall obtain the current switch orientations from the Track Model.
Precondition	The track is operational.
Basic Sequence	1. The Track Model sends a list of all the switch states along the track in the form of Boolean signals.
Exceptions	
Data	- switch_status: pyqtSignal
Stimulus	The CTC needs to know the current switch states along the track.
Post Conditions	The Wayside Controller can now send a report of the switch statuses to the CTC.

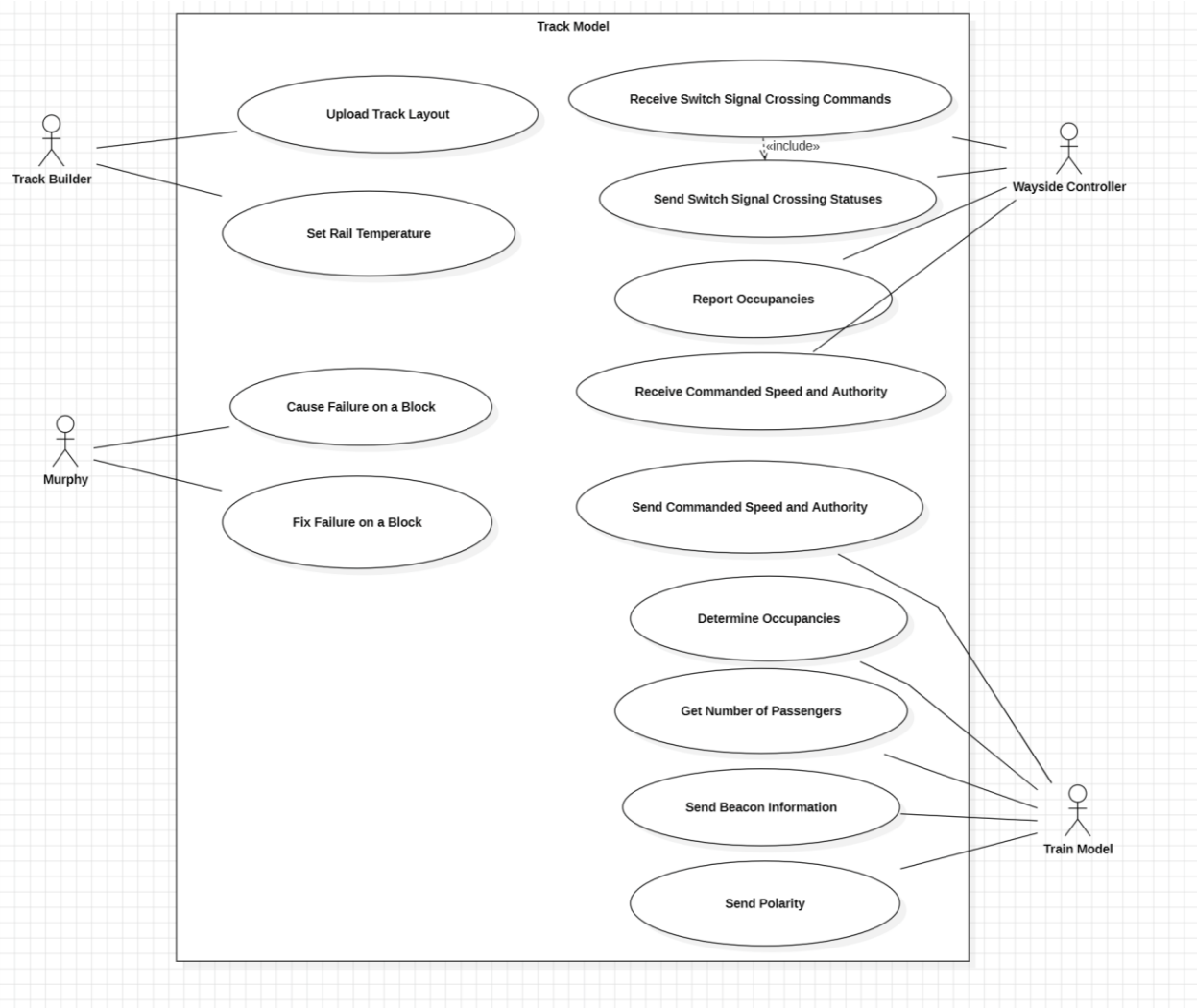
Use Case 17: Receive Signal Status

Actor	Track Model
Description	The Track Controller shall obtain the current signal statuses from the Track Model.
Precondition	The track is operational.
Basic Sequence	1. The Track Model sends a list of all the traffic signal states along the track in the form of Boolean signals.
Exceptions	
Data	- signal_status: pyqtSignal
Stimulus	The CTC needs to know the current signal states along the track.
Post Conditions	The Wayside Controller can now send a report of the signal statuses to the CTC.

Use Case 18: Receive Crossing Status

Actor	Track Model
Description	The Track Controller shall obtain the current crossing states from the Track Model.
Precondition	The track is operational.
Basic Sequence	1. The Track Model sends a list of all the crossing states along the track in the form of Boolean signals.
Exceptions	
Data	- crossing_status: pyqtSignal
Stimulus	The CTC needs to know the current crossing states along the track.
Post Conditions	The Wayside Controller can now send a report of the crossing statuses to the CTC.

5.2.3 Track Model Use Case Diagram



Explanation: Use Case Diagram for the Track Model, illustrating the interactions between the Track Model and its actors – the Wayside Controller, Train Model, Track Builder, and Murphy. This diagram highlights key use cases including track occupancy detection, operational status, communication of information.

5.2.3.1 Track Model Use Case Descriptions

Use Case 1: Upload Track Layout

Actor	Track Builder
Description	The Track Builder shall upload a track layout to the system.
Precondition	The track layout must be in a valid database format.
Basic Sequence	1. The Track Builder shall press the “Upload New Track Layout” button in the Track Model user interface. 2. The Track Builder is directed to the File Explorer where they can select the track layout file. 3. The UI will be populated by the information from the track layout.
Exceptions	Throws an error if the file type is not valid.
Data	- line: str - section: str - number: int - grade: float - speedLimit: float - infrastructure: str - elevation: float - cumulativeElevation: float - right: bool - left: bool
Stimulus	The file upload button is pressed.
Post Conditions	The track layout is represented in the interactive UI.

Use Case 2: Toggle Failure Modes

Actor	Murphy
Description	Murphy shall be able to toggle individual blocks in and out of various failure modes.
Precondition	The blocks have been created from information provided by the excel file.
Basic Sequence	3. Murphy may select a block from three separate combo boxes in the Track Model UI which correspond to the three failure types. 4. Each failure type has a respective check box next to it which allows Murphy to toggle the failure mode of the block. 5. Blocks in failure will have a visual indication in the user interface.
Exception	Throws an error if the blocks have not been initialized.
Data	- failureTypes: str[] - functional: bool - occupied: bool
Stimulus	A block is selected, and the checkbox is checked or unchecked.
Post Conditions	The blocks in failure will be visually represented in the user interface, the blocks' functional attribute will be set to False, and the occupied attribute will be set to True.

Use Case 3: Receiving Switch, Crossing, and Light Commands and Sending Respective Statuses

Actor	Wayside Controller
Description	The Wayside Controller shall be able to command the operational states of the switches and crossings along the track.
Precondition	A track layout has been successfully uploaded to the system with valid infrastructure formatting.
Basic Sequence	3. The Wayside Controller may send individual Boolean signals to the Track Model to control the switches and crossings. 4. The switches and crossings in the Track Model shall be represented according to the received commands. 5. The light status shall change according to the new switch and crossing states. 6. These new statuses will be reported back to the Wayside.
Exception	The switches, crossing, lights will not be initialized if the track layout has invalid formatting.
Data	- switches: List[int] - switchStatus: bool - crossings: List[int] - crossingStatus: bool - switchLightColor: str - crossingLight: str - CmdSwitchCmd: pyqtSignal - CmdCrossingCmd: pyqtSignal
Stimulus	The CTC dispatches a train on a route with switches and crossings.
Post Conditions	The trains movement shall be reflected in accordance with the switch and crossing states.

Use Case 4: Report Occupancies

Actor	Wayside Controller
Description	The Track Model shall be able to report the block occupancies to the Wayside Controller for processing.
Precondition	There is a track layout uploaded to the system.
Basic Sequence	3. The Track Layout sends a list of Boolean signals to the Wayside Controller.
Exception	The service brake fails to stop the train in a safe manner.
Data	- occupancies: List[bool]
Stimulus	The system requires information about the current track state.
Post Conditions	Boolean signals shall be relayed to the Wayside Controller to be handled by the PLC algorithm.

Use Case 6: Receive Commanded Speed and Authority

Actor	Wayside Controller
Description	The Wayside Controller must pass along vital information including commanded speed and authority to the Train Controller.
Precondition	The Wayside Controller determines a commanded speed and authority for the train.
Basic Sequence	2. The Wayside encodes the commanded speed and authority into binary and sends it to the Track Model. 3. The Track Model shall temporarily store this data.
Exception	Throws an error if data is the wrong data type.
Data	cmdSpeed: pyqtSignal authority: pyqtSignal
Stimulus	The Wayside Controller determines a commanded speed and authority for the train based on safety limitations of the track.
Post Conditions	The Track Model shall discretely store the commanded speed and authority.

Use Case 7: Send Commanded Speed and Authority

Actor	Train Model
Description	The Track Model shall take the commanded speed and authority received from the Wayside controller and pass it along to the Train Model.
Precondition	The Train Model must be dispatched on the Track Model and receiving signal.
Basic Sequence	2. The Track Model sends the commanded speed and authority that it previously stored to the Train Model via its antennas.
Exceptions	The data transfer is restricted to 10 baud.
Data	- cmdSpeed: pyqtSignal - authority: pyqtSignal
Stimulus	The Wayside Controller determines a commanded speed and authority for the train based on safety limitations of the track.
Post Conditions	The Train Model shall receive and temporarily store the data it receives from the Track Model.

Use Case 8: Set Rail Temperature

Actor	Track Builder
Description	The Track Builder can set the temperature of the rails and control the operation of the track heater.
Precondition	The track must be operational.
Basic Sequence	2. The Track Builder can numerically set the desired temperature of the rails. 3. The Track Heater can raise the temperature of the track when necessary.
Exceptions	If the temperature of the rails falls below 32 degrees Fahrenheit, the Track Heater will turn on automatically.
Data	- temp: float
Stimulus	The Track Builder determines the desired temperature for the track.
Post Conditions	The rail temperature is set to the desired value and may be adjusted using the track heater.

Use Case 9: Determine Occupancies

Actor	Train Model
Description	The Train Model occupies a block by shorting the track circuit within the block.
Precondition	The track is operational and at least one train is running on the track.
Basic Sequence	2. The Train Model sends its position to the Track Model. 3. The Track Model compares this position value to the blocks in the track and sets the intersecting block's occupied attribute to True.
Exceptions	The train's position must be within the range of the track.
Data	- position: float - occupied: bool - lengthArray: float[] - blockNum: int - blockStart: float - blockEnd: float
Stimulus	The train is operating on the track.
Post Conditions	The Track Model determines the blocks occupied by trains and updates the list of occupancies.

Use Case 10: Get Number of Passengers

Actor	Train Model
Description	The Track Controller generates a random number of people boarding at a station.
Precondition	The train must arrive at a station.
Basic Sequence	4. The Train Model sends an integer representing the number of open seats on the train when it arrives at a station. 5. The Track Model uses a random number algorithm to generate a number of people boarding at the station.
Exceptions	The number generated must be less than or equal to the number of vacant seats on the train.
Data	- vacancy: int - passengers: int
Stimulus	The train reaches a station where people load and unload.
Post Conditions	A new number of passengers are on the train when it leaves the station.

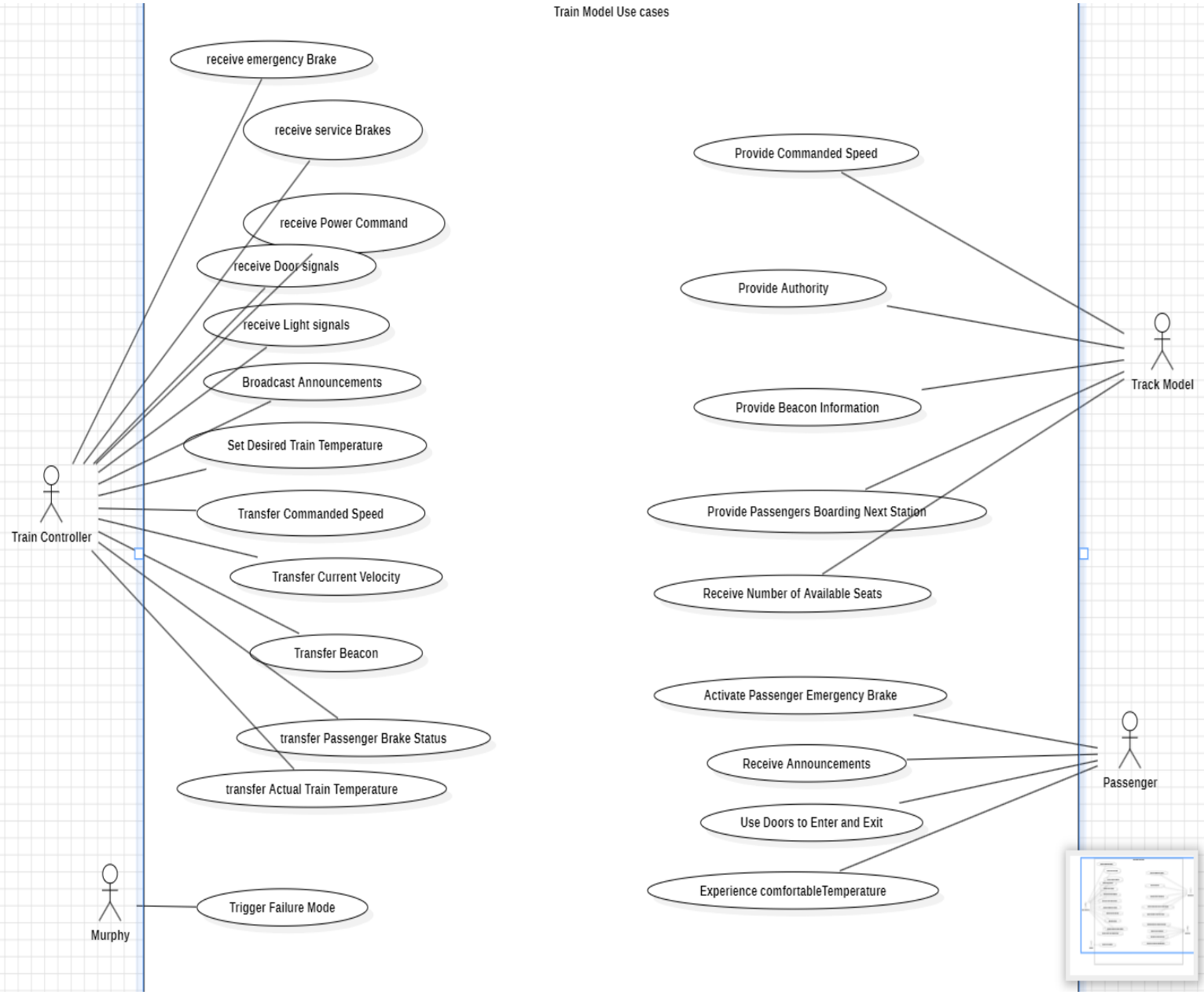
Use Case 11: Send Beacon Information

Actor	Train Model
Description	There are beacons at each end of a station that store static information for the train.
Precondition	The train must pass the beacon as it is entering and leaving the station.
Basic Sequence	1. Static information is stored inside the beacon when the track layout is uploaded. 2. The train picks up the information via its antennas as it passes the transponders.
Exceptions	The data size is limited to 128-bits.
Data	- stationName: str - right: bool - left: bool
Stimulus	The train needs to know some static information about the proceeding blocks.
Post Conditions	The train picks up information about the station and the following blocks.

Use Case 12: Send Polarity

Actor	Train Model
Description	The Track Models have individual track circuits within each block that alternate in polarity for the entire length of the track.
Precondition	The train must be operating on the track model.
Basic Sequence	1. The Train Model has a sensor that can read the polarity of the block that it is currently on.
Exceptions	The track may be in power failure, or the train is in signal pickup failure.
Data	- polarity: pyqtSignal
Stimulus	The Train Controller needs to know if and when the train has moved onto another block.
Post Conditions	The train receives the polarity of the block via its sensor.

5.2.4 Train Model Use Case Diagram



Explanation: Use Case Diagram for the Train Model, illustrating the interactions between the Train Model and its actors, such as the Track Model, Train Controller, Passengers, and Murphy. This diagram highlights key use cases including train movement control, speed regulation, controlling train doors, turning on/off lights, and other onboard systems.

5.2.4.1 Train Model Use Case Descriptions

Use Case 1: Receive Service Brake Command

Actor	Train Controller
Description	The Train Model receives a command from the Train Controller to apply the service brake for normal deceleration.
Precondition	The Train Controller determines the need to slow down the train under normal conditions.
Basic Sequence	1. Train Controller sends a Service Brake command to the Train Model. 2. Train Model applies the service brake. 3. Train Model updates the current speed.
Exceptions	Service brake system malfunction prevents braking.
Data	Driver Service Brake (Boolean)
Stimulus	Brake command signal from the Train Controller.
Post Conditions	1. Train decelerates using the service brake. 2. Train Model updates speed status.

Use Case 2: Receive Emergency Brake Command

Actor	Train Controller
Description	The Train Model receives a command from the Train Controller to apply the emergency brake for immediate stopping.
Precondition	The Train Controller determines the need to stop the train immediately due to an emergency.
Basic Sequence	1. Train Controller sends an Emergency Brake command to the Train Model. 2. Train Model applies the emergency brake. 3. Train Model updates the current speed.
Exceptions	Emergency brake system malfunction prevents braking.
Data	Driver Emergency Brake (Boolean)
Stimulus	Emergency brake command signal from the Train Controller.
Post Conditions	1. Train decelerates rapidly using the emergency brake. 2. Train Model updates speed status.

Use Case 3: Receive Power Command

Actor	Train Controller
Description	The Train Model adjusts its power output based on the power command received from the Train Controller to achieve the desired speed.
Precondition	The Train Controller calculates the required power to reach or maintain the desired speed.
Basic Sequence	1. Train Controller sends the Power Command to the Train Model. 2. Train Model adjusts motor power output according to the command. 3. Train Model updates current speed and acceleration.
Exceptions	• Engine failure prevents power adjustment. - Power command exceeds allowable limits and is capped.
Data	Power Command (Watts)
Stimulus	Power command signal from the Train Controller.
Post Conditions	1. Train adjusts speed based on the new power output. 2. Train Model updates speed and acceleration status.

Use Case 4: Receive Door Signals

Actor	Train Controller
Description	The Train Model receives commands from the Train Controller to open or close the train's left and right doors.
Precondition	The train is at a station or approaching one where door operation is required.
Basic Sequence	1. Train Controller sends door open/close commands to the Train Model. 2. Train Model operates doors as per the commands. 3. Train Model updates door status.
Exceptions	• Door mechanism failure prevents doors from operating. - Safety interlocks prevent doors from opening when unsafe.
Data	• Left Doors open/close (Boolean) - Right Doors open/close (Boolean)
Stimulus	Door operation commands from the Train Controller.
Post Conditions	1. Doors are opened or closed as commanded. 2. Passengers can board or alight safely.

Use Case 5: Receive Light Signals

Actor	Train Controller
Description	The Train Model adjusts the interior and exterior lights based on signals received from the Train Controller.
Precondition	A need arises to adjust the train's lighting conditions.
Basic Sequence	1. Train Controller sends light on/off commands to the Train Model. 2. Train Model adjusts the interior and exterior lights accordingly. 3. Train Model confirms light status.
Exceptions	Lighting system malfunction prevents lights from adjusting.
Data	• Outside lights on/off (Boolean) - Interior lights on/off (Boolean)
Stimulus	Light adjustment commands from the Train Controller.
Post Conditions	1. Train's lighting is adjusted as commanded. 2. Passenger comfort and safety are maintained.

Use Case 6: Broadcast Announcements

Actor	Train Controller
Description	The Train Model broadcasts announcements to passengers as received from the Train Controller.
Precondition	There is information that needs to be communicated to passengers.
Basic Sequence	1. Train Controller sends an announcement to the Train Model. 2. Train Model broadcasts the announcement over the PA system. 3. Train Model may display the announcement on screens if available.
Exceptions	Public address system failure prevents announcements from being broadcast.
Data	Any Announcement (Text)
Stimulus	Announcement message from the Train Controller.
Post Conditions	1. Passengers receive the announcement. 2. Train Model confirms the announcement was broadcast.

Use Case 7: Set Desired Train Temperature

Actor	Train Controller
Description	The Train Model adjusts the HVAC system to achieve the desired temperature set by the Train Controller.
Precondition	A temperature adjustment is needed for passenger comfort.
Basic Sequence	1. Train Controller sends the Desired Train Temperature to the Train Model. 2. Train Model adjusts HVAC settings to reach the desired temperature. 3. Train Model monitors interior temperature.
Exceptions	HVAC system malfunction prevents temperature adjustment.
Data	Desired Train Temperature (Fahrenheit)
Stimulus	Temperature setting command from the Train Controller.
Post Conditions	1. Train's interior temperature adjusts towards the desired setting. 2. Train Model reports actual temperature back to the Train Controller.

Use Case 8: Transfer Commanded Speed

Actor	Train Controller
Description	The Train Model provides the current commanded speed to the Train Controller for monitoring purposes.
Precondition	The Train Model has updated the commanded speed.
Basic Sequence	1. Train Model sends the Commanded Speed to the Train Controller. 2. Train Controller receives and records the commanded speed.
Exceptions	Communication failure prevents the speed data from being received.
Data	Commanded Speed (km/hr)
Stimulus	Automatic data transmission from the Train Model.
Post Conditions	1. Train Controller has up-to-date commanded speed information. 2. Operational monitoring is accurate.

Use Case 9: Transfer Current Velocity

Actor	Train Controller
Description	The Train Model sends the current velocity of the train to the Train Controller for real-time monitoring.
Precondition	The train is in operation and current speed data is available.
Basic Sequence	1. Train Model measures current velocity. 2. Train Model sends Current Velocity to the Train Controller. 3. Train Controller receives and displays the current velocity.
Exceptions	• Sensor failure prevents speed measurement. - Communication failure prevents data transmission.
Data	Current Velocity (km/hr)
Stimulus	Regular updates from the Train Model.
Post Conditions	1. Train Controller has accurate, real-time speed data. 2. Speed-related decisions can be made appropriately.

Use Case 10: Transfer Beacon

Actor	Train Controller
Description	The Train Model provides beacon information to the Train Controller, such as station announcements or track conditions.
Precondition	Beacon data has been received by the Train Model.
Basic Sequence	1. Train Model processes beacon information. 2. Train Model sends Beacon data to the Train Controller. 3. Train Controller receives and uses the information.
Exceptions	• Communication failure prevents data transmission. - Data corruption renders beacon information unusable.
Data	Beacon (Characters)
Stimulus	Beacon data transmission from the Train Model.

Post Conditions	1. Train Controller is informed of upcoming stations or track conditions. 2. Operational adjustments can be made if necessary.

Use Case 11: Transfer Passenger Brake Status

Actor	Train Controller
Description	The Train Model notifies the Train Controller when a passenger activates the emergency brake.
Precondition	A passenger has pulled the emergency brake handle.
Basic Sequence	1. Train Model detects passenger emergency brake activation. 2. Train Model applies the emergency brake. 3. Train Model sends Passenger Brake status to the Train Controller.
Exceptions	Communication failure prevents the alert from reaching the Train Controller.
Data	Passenger Brake status (Boolean)
Stimulus	Activation of passenger emergency brake system.
Post Conditions	1. Train comes to an emergency stop. 2. Train Controller initiates emergency response procedures.

Use Case 12: Transfer Actual Train Temperature

Actor	Train Controller
Description	The Train Model sends the actual interior temperature to the Train Controller for monitoring.

Precondition	Temperature data is available from sensors.
Basic Sequence	1. Train Model reads actual temperature. 2. Train Model sends Actual Train Temperature to the Train Controller. 3. Train Controller receives and may adjust desired temperature if necessary.
Exceptions	• Sensor failure prevents temperature reading. - Communication failure prevents data transmission.
Data	Actual Train Temperature (Fahrenheit)
Stimulus	Temperature data update from the Train Model.
Post Conditions	1. Train Controller has current temperature data. 2. HVAC settings can be adjusted for passenger comfort.

Use Case 13: Trigger Failure Mode

Actor	Murphy System
Description	The Murphy System triggers a failure mode in the Train Model to simulate a fault for testing purposes.
Precondition	System is in a state where failure simulation is permissible.
Basic Sequence	1. Murphy System selects a failure mode to simulate. 2. Murphy System sends Failure Mode Trigger to the Train Model. 3. Train Model simulates the failure. 4. Train Model activates safety protocols as appropriate.
Exceptions	Safety constraints prevent the failure mode from being triggered.
Data	Failure Mode Trigger (Specific failure to simulate)

Stimulus	Failure simulation command from the Murphy System.
Post Conditions	1. Train Model operates under simulated failure conditions. 2. System responses are tested and validated.

Use Case 14: Provide Commanded Speed

Actor	Track Model
Description	The Track Model provides the commanded speed to the Train Model to control the train's speed.
Precondition	The Track Model has determined the appropriate commanded speed.
Basic Sequence	1. Track Model sends Commanded Speed to the Train Model. 2. Train Model receives and acknowledges the commanded speed.
Exceptions	• Communication failure prevents data transmission. - Invalid data is received.
Data	Commanded Speed (km/hr)
Stimulus	Commanded speed data from the Track Model.
Post Conditions	1. Train Model adjusts speed according to the commanded speed. 2. Speed control maintains safe and efficient operation.

Use Case 15: Provide Authority

Actor	Track Model
Description	The Track Model provides the authority distance to the Train Model, indicating how far the train is allowed to travel before stopping.

Precondition	The Track Model has calculated the safe authority.
Basic Sequence	1. Track Model sends Authority to the Train Model. 2. Train Model receives and uses the authority information.
Exceptions	Communication failure prevents authority data from being received.
Data	Authority (Meters)
Stimulus	Authority data from the Track Model.
Post Conditions	1. Train operates within the given authority. 2. Safety protocols prevent exceeding authority limits.

Use Case 16: Provide Beacon Information

Actor	Track Model
Description	The Track Model sends beacon information to the Train Model, such as station announcements or track conditions.
Precondition	Beacon information is available and relevant to the train's current location.
Basic Sequence	1. Track Model sends Beacon info to the Train Model. 2. Train Model receives and processes the beacon information.
Exceptions	• Communication failure prevents data transmission. - Data corruption makes beacon info unusable.
Data	Beacon info (Characters)
Stimulus	Beacon information from the Track Model.

Post Conditions	1. Train Model has updated information about upcoming stations or track conditions. 2. Passengers may receive relevant announcements.

Use Case 17: Provide Passengers Boarding Next Station

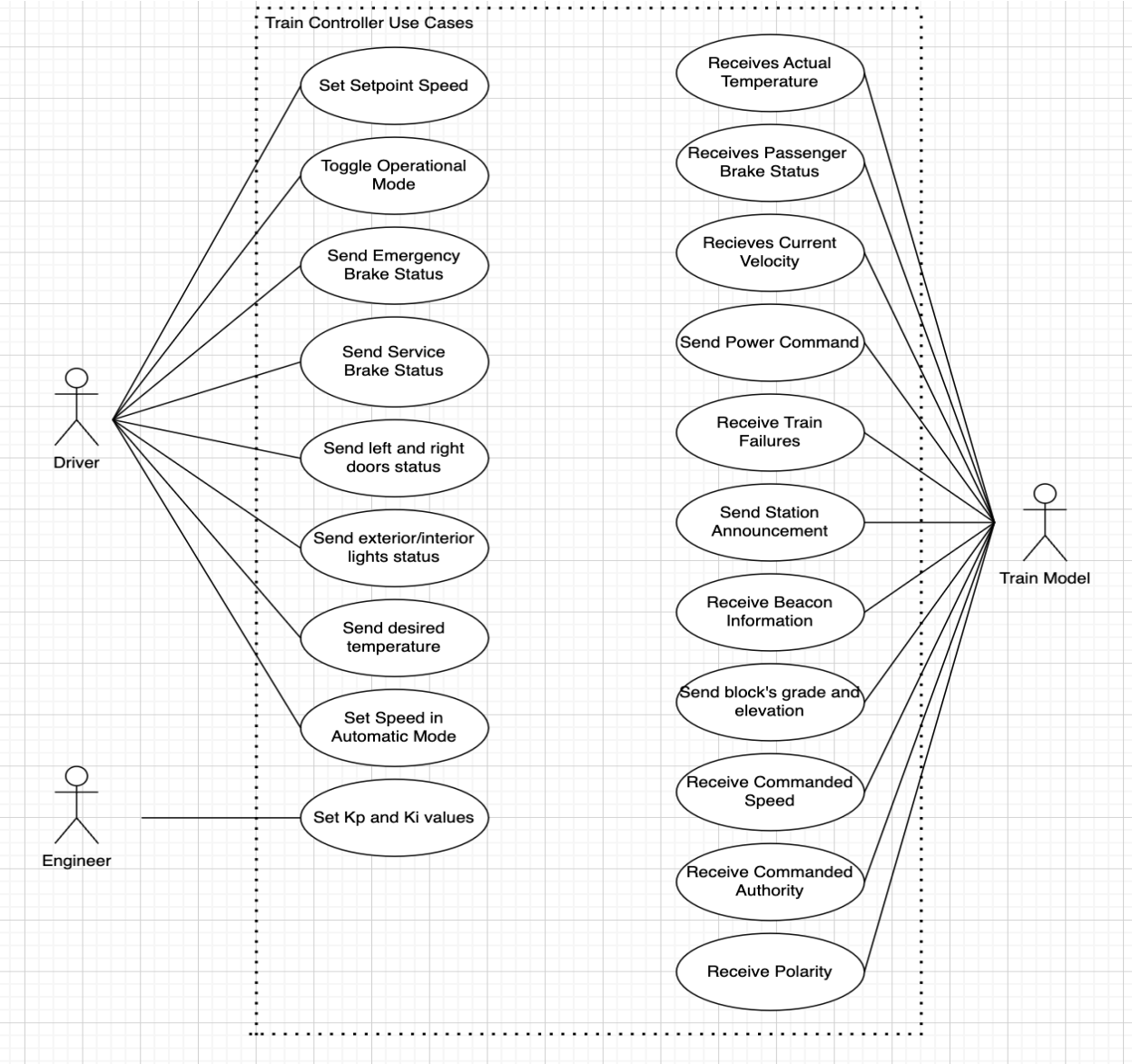
Actor	Track Model
Description	The Track Model informs the Train Model of the expected number of passengers boarding at the next station.
Precondition	The Track Model has estimates of passenger boarding numbers.
Basic Sequence	1. Track Model sends Passengers boarding at next station to the Train Model. 2. Train Model receives the data and prepares for boarding.
Exceptions	Communication failure prevents boarding data from being received.
Data	Passengers boarding at next station (Integer)
Stimulus	Boarding data from the Track Model.
Post Conditions	1. Train Model updates the passenger count. 2. 2. Boarding processes are managed efficiently.

Use Case 18: Receive Number of Available Seats

Actor	Track Model
Description	The Train Model provides the number of available seats to the Track Model to assist in managing passenger boarding.

Precondition	Passenger counts are updated after boarding or alighting.
Basic Sequence	1. Train Model calculates Number of available seats inside train. 2. Train Model sends this information to the Track Model.
Exceptions	Communication failure prevents seating data from being transmitted.
Data	Number of available seats inside train (Integer)
Stimulus	Seating availability data from the Train Model.
Post Conditions	1. Track Model has accurate seating availability information. 2. Passenger boarding is managed to prevent overcrowding.

5.2.5 SW Train Controller Use Case Diagram



Explanation: Use Case Diagram for the Train Controller, illustrating the interactions between the Train Controller and its actors, such as the Train Model, Train Driver, and the Train Engineer. This diagram highlights key use cases including monitoring and controlling train speed and authority, initiating emergency stops, managing onboard systems (such as door control and lighting), and sending the current power command of the engine to the Train Model.

5.2.5.1 SW Train Controller Use Case Descriptions

Use Case 1: Set Setpoint Speed

Actor	Train Driver
Description	Train Driver sets desired speed of the train.
Precondition	The train has to be in manual mode.
Basic Sequence	1. In manual mode, the train driver can enter the speed that they would like the train to go. 2. The speed input by the driver is not allowed to exceed the minimum between the commanded speed and the speed limit.
Exceptions	The driver enters in a value that is higher than the minimum of the commanded speed and the speed limit. But, this is handled with an upper bound to the setpoint speed input.
Data	Commanded Speed – Float Setpoint Speed (Desired Speed) - Float
Stimulus	The train driver inputs the desired setpoint speed in the corresponding input box in the train controller.
Post Conditions	The desired speed of the train is then set to the setpoint speed input by the train driver.

Use Case 2: Set Speed in Automatic Mode

Actor	Train Driver

Description	Train Driver has entered the train in manual mode.
Precondition	The train has to be in automatic mode.
Basic Sequence	1. In automatic mode, the train controller would automatically set the desired speed to the minimum of the commanded speed and the speed limit.
Exceptions	The train controller accidentally chooses the higher of the commanded speed and speed limit of the block. The software checks the math of the train controller two times to make sure that the desired speed is indeed the minimum of the commanded speed and the speed limit. That is how this exception is handled.
Data	Commanded Speed – Float Desired Speed - Float Speed Limit - Float
Stimulus	The train driver selects the current train operational mode to be automatic.
Post Conditions	The desired speed of the train is then set to the minimum of the commanded speed and the speed limit of the block.

Use Case 3: Toggle Operational Mode

Actor	Train Driver
Description	The train driver selects if the train is operating in manual or automatic mode.
Precondition	The train has to be powered on and in an operational state.

Basic Sequence	1. The Train Driver has two options to choose the train's operational mode: Manual and Automatic. The selected mode will appear green, while the unselected one will appear white. 2. In manual mode, the driver can set the setpoint speed, toggle the exterior/interior lights, and open/close doors when the train is stopped. 3. In automatic mode, the train is able to run safely without the driver having to press anything. The desired velocity of the train is picked from the minimum of the commanded speed and the speed limit.
Exception	The train does not successfully transition over to the selected operational mode. This is handled by letting the driver know that the mode has not been changed successfully.
Data	Commanded Speed – Float Operational Mode - Boolean
Stimulus	The train driver selects which operational mode they would like the train to operate in.
Post Conditions	The train will operate in whichever mode is currently selected by the Train Driver.

Use Case 4: Send Emergency Brake Status

Actor	Train Driver
Description	The train driver and the passengers can trigger the emergency brake whenever it is necessary.
Precondition	A critical safety event has occurred.

Basic Sequence	1. The Train Driver has the ability to trigger the train's emergency brake if they see something in front of the tracks that prevents the train from proceeding forward. 2. The Passengers have the ability to trigger the train's emergency brake if they see something in front of the tracks that prevents the train from proceeding forward.
Exception	The emergency brake fails to stop the train in a safe manner. This is handled by constantly applying the emergency brake until the signal is successfully sent to the Train Model.
Data	Emergency Brake Status - Boolean
Stimulus	The train driver or passenger triggers the emergency brake.
Post Conditions	The signal that the emergency brake has been pressed is sent to the train model, where the train model decreases the speed of the train with the corresponding deceleration rate.

Use Case 5: Send Service Brake Status

Actor	Train Driver
Description	The train driver can trigger the service brake whenever it is necessary.
Precondition	The train is approaching a station, or the train must stop for some other reason.
Basic Sequence	1. The Train Driver has the ability to trigger the train's service brake if they see something in front of the tracks that prevents the train from proceeding forward.
Exception	The service brake fails to stop the train in a safe manner. This is handled by constantly applying the service brake until the signal is successfully sent to the Train Model.
Data	Service Brake Status - Boolean

Stimulus	The train driver triggers the service brake.
Post Conditions	The signal that the service brake has been pressed is sent to the train model, where the train model decreases the speed of the train with the corresponding deceleration rate.

Use Case 6: Send Door Status

Actor	Train Driver
Description	The train driver can open and close the train doors if the train is not moving.
Precondition	The train must be in manual mode.
Basic Sequence	1. In manual mode, the train driver has the ability to open and close both doors on the train. 2. The driver can only change the status of the doors when the train's current speed is 0.
Exception	The trains doors are jammed, and they are not opening. This is handling by letting the driver know that they are jammed and cannot open.
Data	Train Doors Status - Boolean
Stimulus	The train driver toggles both train doors.
Post Conditions	The signal for which door to open and close is sent from the train controller to the train model, which then physically opens or closes to corresponding door.

Use Case 7: Send Exterior/Interior Lights Status

Actor	Train Driver

Description	The train driver can toggle the exterior and interior lights of the train
Precondition	The train must be in manual mode.
Basic Sequence	1. In manual mode, the train driver has the ability to toggle the exterior and interior lights.
Exceptions	The lights on the train are burned out and do not turn on. This is handling by letting the driver know that they are burnt out.
Data	Train Light Status - Boolean
Stimulus	The train driver toggles both exterior and interior lights.
Post Conditions	The signal for which door to open and close is sent from the train controller to the train model, which then physically opens or closes to corresponding door.

Use Case 8: Send Desired Temperature

Actor	Train Driver
Description	The train driver can set the temperature of the train.
Precondition	The train has to be powered on and in an operational state.
Basic Sequence	1. The train driver has an input where he/she can enter the desired train temperature. 2. The temperature entered has to be between 60-75 degrees Fahrenheit.
Exceptions	If the temperature entered is lower than 60 or higher than 75, then the temperature will not change.
Data	Actual Train Temperature – Float Desired Train Temperature - Float

Stimulus	The train driver enters in the desired train temperature.
Post Conditions	The desired temperature is given to the Train Model, which then uses a first-order equation that gradually increases the actual train temperature to the desired train temperature.

Use Case 9: Set Kp and Ki values

Actor	Train Engineer
Description	The train engineer can set the Kp and Ki value of the train.
Precondition	The train has to be in the yard and has not departed yet.
Basic Sequence	1. The train engineer tunes the system to find stable Kp and Ki values for the train. 2. The train engineer inputs the Kp and Ki values they found into the train before it leaves the yard.
Exceptions	The Kp and Ki values change while the train is moving. This is handled by not allowing the Kp and Ki values to change.
Data	Kp – Float Ki - Float
Stimulus	The train engineer enters in the Kp and Ki values of the train.
Post Conditions	The Kp and Ki are used for the calculation of the power command of the train.

Use Case 10: Send Power Command

Actor	Train Model
Description	The train controller calculates the current power command and sends it to the train model.
Precondition	The train has to be powered on and in an operational state.
Basic Sequence	1. The train controller obtains the Kp and Ki value from the engineer 2. The train controller calculates the current power command with the current velocity given by the train model and the Kp and Ki values.
Exceptions	If the current velocity is given to be negative. The train controller ignores the current velocity given and then waits for the next current velocity input.
Data	Power Command - Float
Stimulus	The train is moving and has a current velocity of more than 0.
Post Conditions	The power command that is found is given to the train model, which uses it to find the current velocity and gives that value to the train controller. This loop will go on if the train is moving.

Use Case 11: Receive Train Failures

Actor	Train Model
Description	The train controller displays the status of all failure modes in the train.
Precondition	The train has to be powered on and in an operational state.

Basic Sequence	1. The train controller gets a signal from the train model that triggers the failure 2. The driver will see that the indicator of the failure is red, and action needs to be taken.
Exceptions	If the sensor for the failure indicators is broken, then the driver would not be able to see what had failed in the train. The driver would be told that the indicators are broken.
Data	Engine Failure – Boolean Brake Failure – Boolean Signal Pickup Failure - Boolean
Stimulus	The train model sends a signal to the train controller that triggers one of the failures in the train.
Post Conditions	The train driver sees that the failure has been triggered and must act on that failure in a safe manner.

Use Case 12: Receive Commanded Speed

Actor	Train Model
Description	The train controller receives the commanded speed from the train model and uses those values to make vital decisions.
Precondition	The train has to be powered on and in an operational state.
Basic Sequence	1. The train controller gets a signal from the train model with the value of the commanded speed. 2. The train controller changes the commanded speed to the minimum of that value and the speed limit.

Exceptions	If the signal pickup failure is triggered, the train would not be able to receive the commanded speed from the train model.
Data	Commanded Speed – Float
Stimulus	The train model sends a signal to the train controller with the value of the commanded speed.
Post Conditions	The train controller would display the current commanded speed. The train controller would then be able to make vital decisions based on those values.

Use Case 13: Receive Commanded Authority

Actor	Train Model
Description	The train controller receives the commanded authority from the train model and uses those values to make vital decisions.
Precondition	The train has to be powered on and in an operational state.
Basic Sequence	1. The train controller gets a signal from the train model with the value of the commanded speed and commanded authority. 2. The train controller then decreases the authority as it moves along.
Exceptions	If the signal pickup failure is triggered, the train would not be able to receive the commanded authority from the train model.
Data	Commanded Authority - Float

Stimulus	The train model sends a signal to the train controller with the value of the commanded authority.
Post Conditions	The train controller would display the current commanded authority. The train controller would then be able to make vital decisions based on those values.

Use Case 14: Send Station Announcement

Actor	Train Model
Description	The train controller sends a message to the train model for an announcement for when the train arrives at the station.
Precondition	The train has to be approaching a station.
Basic Sequence	1. When the train is 300 feet away from the destination, the train controller makes an announcement message for the passengers in the train and passes the message onto the train model to display.
Exceptions	If the message is not passed on successfully during the first attempt, the message will be passed on again. Even if the message was passed on correctly, it would be the same message being displayed again.
Data	Announcement – String Station Name - String
Stimulus	The train has to be 300 feet away from its destination.
Post Conditions	The passengers in the train would be able to see what station they are approaching.

Use Case 15: Receive Beacon Information

Actor	Train Model
Description	The train controller receives all of the static information in the beacon from the train model.
Precondition	The train has to have passed a beacon.
Basic Sequence	1. When the train passes by a beacon, the antenna on the train picks up any information that was stored in the beacon and uses that information for any calculations.
Exceptions	If there is a delayed beacon reception, revert to the last known valid beacon information and try to run the train with that information. Also, the driver would need to be alerted that the current beacon is not correct.
Data	Blocks between current station to next station– List of ints
Stimulus	The train passes by a beacon, which transmits static data to the train.
Post Conditions	The train controller would have more information about the track between the current station and the next station.

Use Case 16: Receive Current Velocity

Actor	Train Model
Description	The train controller receives a signal from the train model for the current velocity of the train.
Precondition	The train has to be powered on and in an operational state.

Basic Sequence	1. When the train is moving, the current velocity is given to the train controller from the train model, and the train controller finds a power command with that velocity. 2. The train model then finds a new current velocity from that power command and that loop goes until the train stops moving.
Exceptions	If the current velocity is given as negative, the train controller would ignore that value and then wait for the next velocity. If it is still negative, then hit the emergency brake automatically.
Data	Current Velocity – Float Power Command - Float
Stimulus	The train is moving along the track.
Post Conditions	The train controller is able to calculate the power command of the train with the current velocity given by the train model.

Use Case 17: Send Current Block's Grade and Elevation

Actor	Train Model
Description	The train controller has access to the track layout at all times and passes the current block's grade and elevation to the train model for calculations.
Precondition	The train controller needs to have access to the track layout.
Basic Sequence	1. During the beginning of the simulation, the train controller has access to the entire track layout.

	2. As the train moves from block to block (indicated by polarity), the train controller would pass the current block's grade and elevation to the train model for physics calculations.
Exceptions	Double check the position of the train by adding up all of the block lengths and comparing it with the current position number, and also with the polarity switches. These double check the polarity of the tracks is being transmitted correctly.
Data	Grade – Float Elevation - Float
Stimulus	The switching of polarity in the track would indicate that new grade and elevation values would need to be found.
Post Conditions	The train model would have the correct grade and elevation values to successfully complete the physics calculations.

Use Case 18: Receives Actual Temperature

Actor	Train Model
Description	The train controller receives the actual train temperature from the Train Model.
Precondition	The train model would need to have received a desired train temperature from the train controller.
Basic Sequence	1. The train controller would give the train model the desired train temperature from the driver's input.

	2. When the train model gets the desired speed, it uses a first-order equation to get the current train temperature gradually up to the desired temperature.
Exceptions	If the desired temperature that the train model receives is negative, then it ignores that desired temperature for the train. Then when the driver sees the temperature does not move in the driver UI, then the driver would input the desired temperature again.
Data	Desired Temperature – Float Actual Temperature - Float
Stimulus	The train driver inputs a desired temperature in the driver UI and that value gets passed onto the train model.
Post Conditions	The train model would use a first-order equation to gradually increase the train temperature and pass it onto the train temperature.

Use Case 19: Receives Passenger Brake Status

Actor	Train Model
Description	The train model would pass a signal to the train controller whenever the passenger brake has been pressed.
Precondition	The passenger brake would have to be pressed by a passenger.
Basic Sequence	1. When the passenger hits the passenger brake command, that is passed onto the train controller. 2. When the train controller gets that signal, then it automatically triggers the emergency brakes until the train has stopped moving.

Exceptions	The train model would have to emit the passenger brake signal two times back-to-back, in case the train controller for some reason did not receive the first signal.
Data	Passenger Brake Command - Boolean
Stimulus	The passenger brake command has been pressed.
Post Conditions	The train controller receives the train controller signal and automatically applies that emergency brake until the train stops moving.

Use Case 20: Receives Polarity

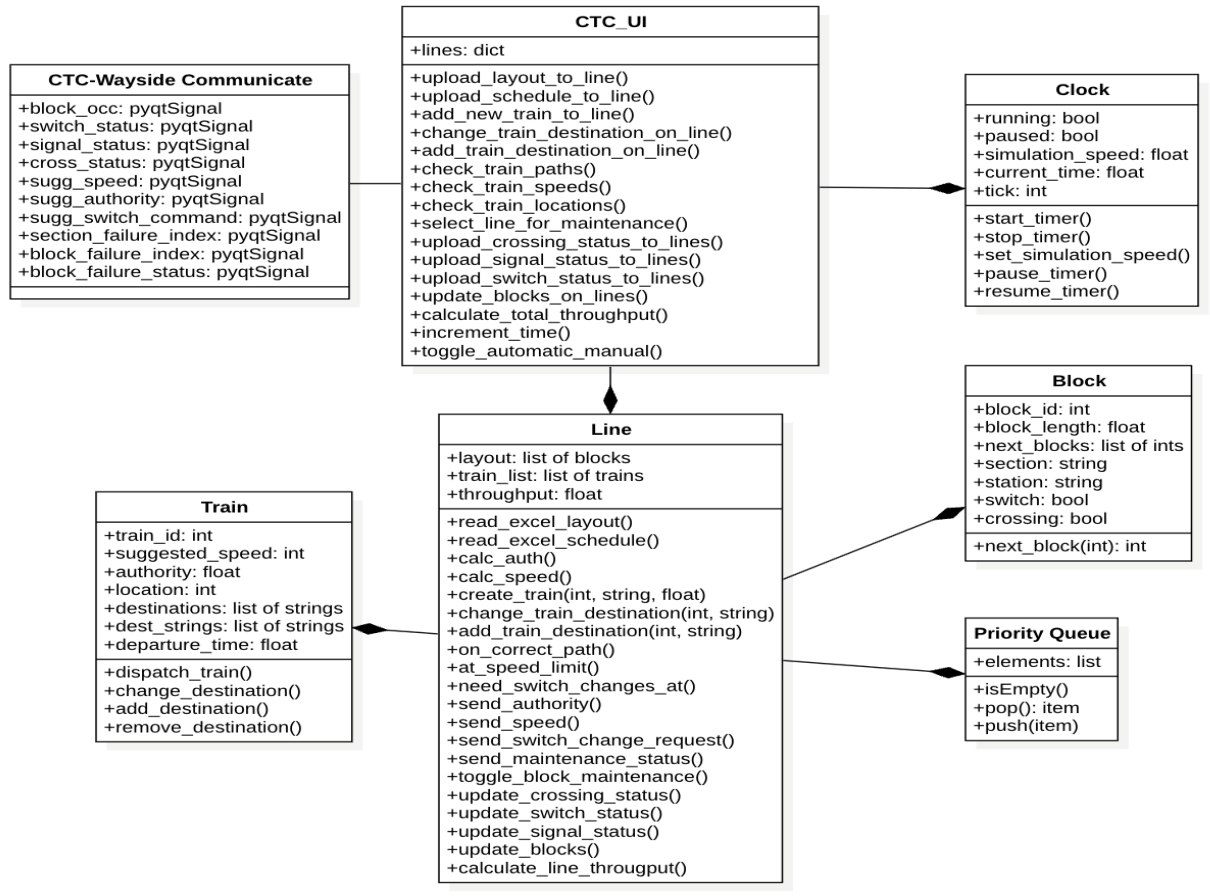
Actor	Train Model
Description	The train controller receives a polarity signal every time the train moves from one block to another.
Precondition	The train has to be moving along the track.
Basic Sequence	1. As the train moves along the track from block to block, the train model sends the train controller a polarity signal saying that the train has moved to a new block. 2. Every time the train controller receives a new signal for the polarity, the train controller would send the train model the current block's grade and elevation for the train model's calculations.
Exceptions	The train model does not pick up the polarity signal and is not able to communicate with the train controller saying that the train has moved on to a new block.
Data	Polarity - Boolean
Stimulus	The train has to be moving along the track.

Post Conditions	The switching of polarity in the track would indicate that new grade and elevation values would need to be found.

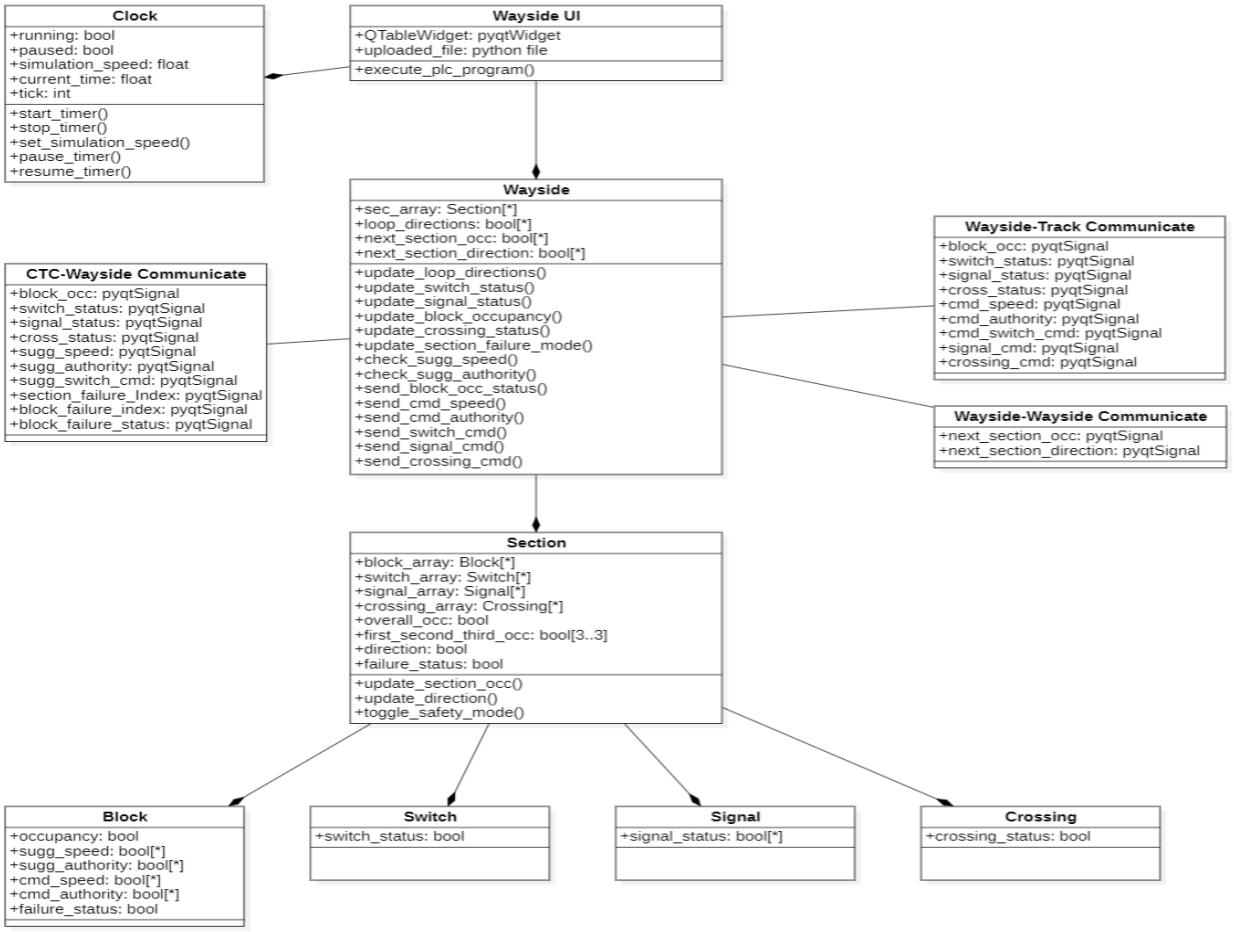
5.3 Logical Viewpoint

The Logical Viewpoint focuses on the internal structure of the system by detailing the classes and interfaces that make up the design. This viewpoint helps us understand how these elements interact and relate to each other. The following class diagrams all have Communicate classes, which allows for values to pass from one sub-system to another sub-system. These Communicate class shall be the exact same for both of the sub-systems that use the class in order to ensure consistency in how values and data are passed between subsystems.

5.3.1 CTC Class Diagram

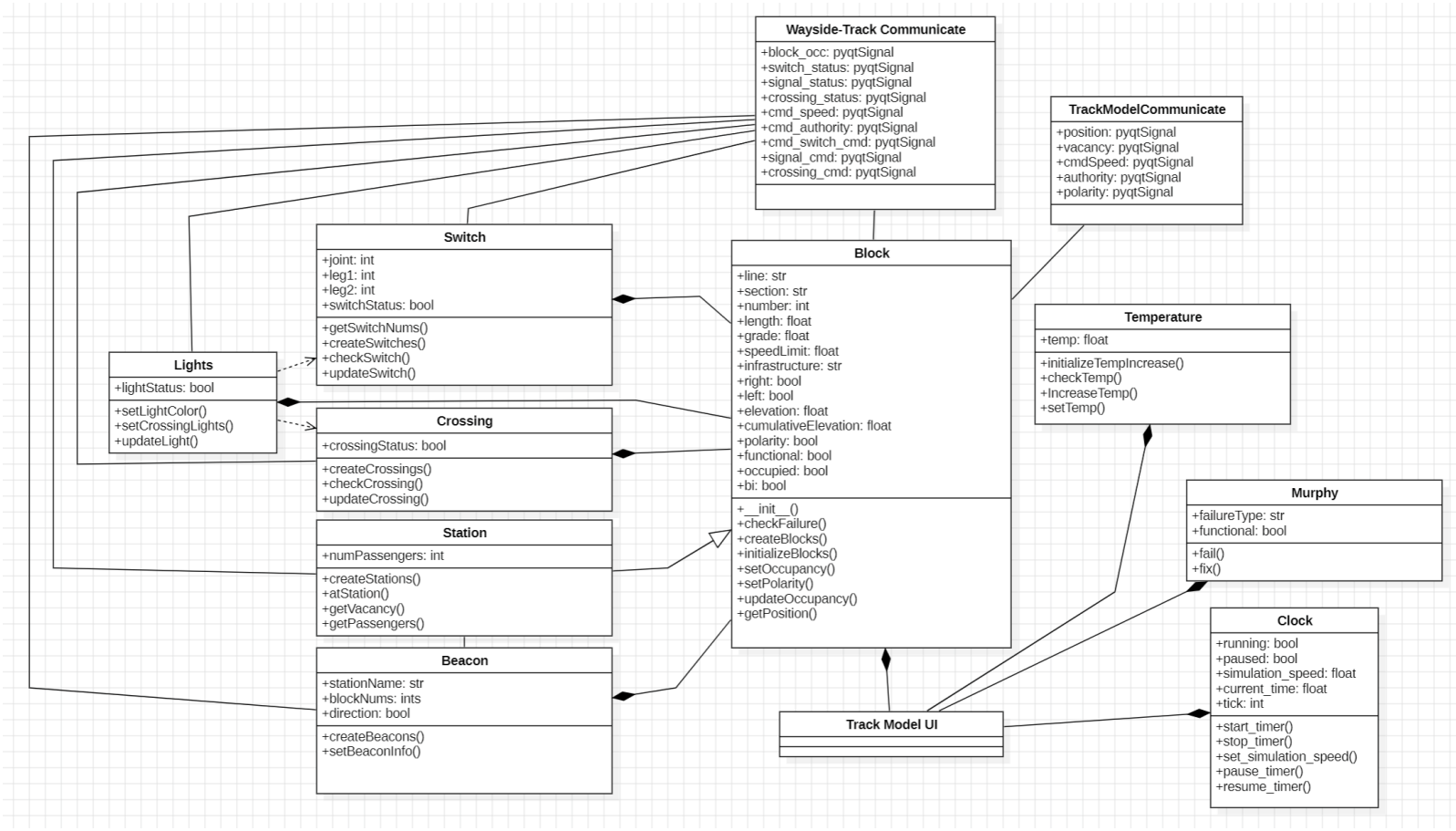


5.3.2 Wayside Controller Class Diagram



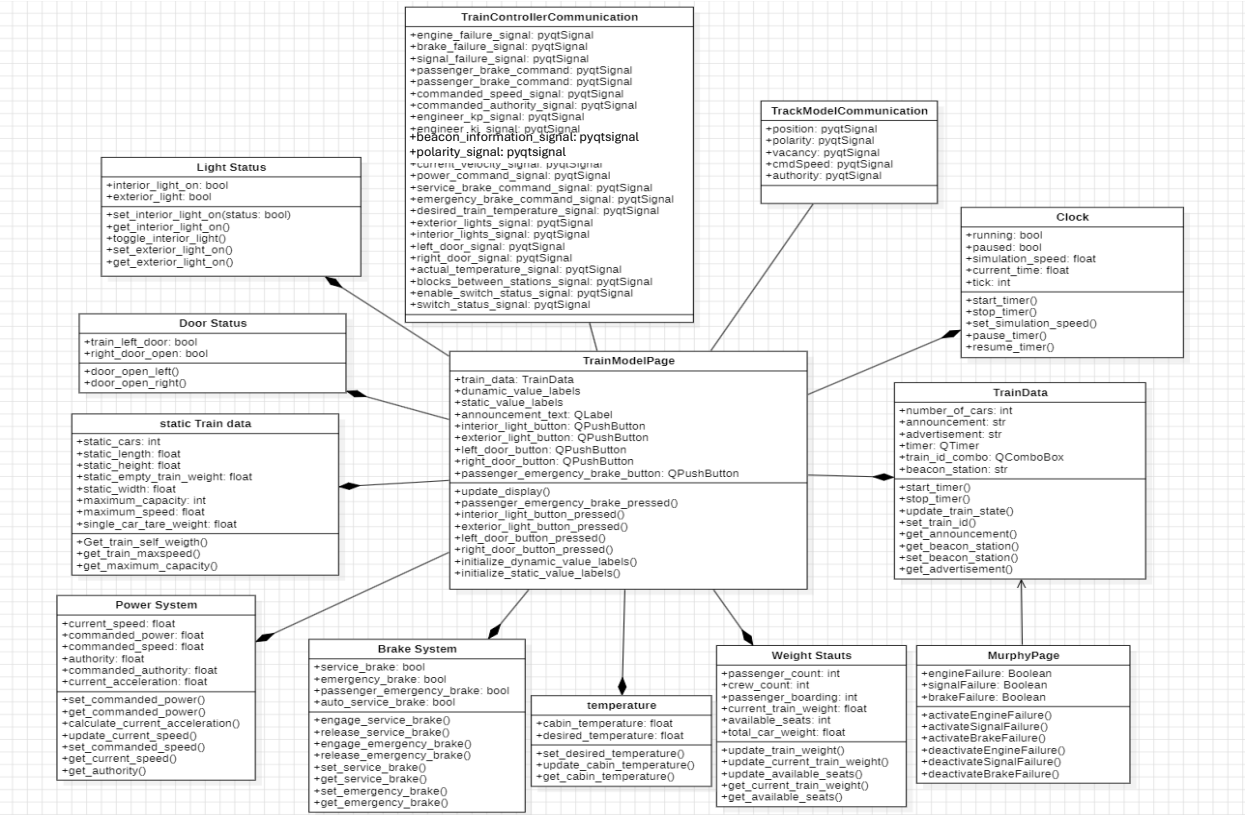
Explanation: Class Diagram for the Wayside Controller, illustrating the core classes and their relationships, including Wayside, Section, Block, Switch, Crossing, and Signal. There are also classes with help explain the inter-system communication between the Wayside Controller and its adjacent modules, which are the CTC and the Track Model, such as the CTC-Wayside Communicate, Wayside-Track Communicate, and Wayside-Wayside Communicate classes. This diagram shows how the Wayside Controller manages signals and tracks by interacting with the Track Model and communicating with the Centralized Traffic Controller (CTC). Finally, there is also a Clock class that all the modules have in order ensure that all modules have their timing synchronized to the same exact second, so that communication occurs smoothly and efficiently.

5.3.3 Track Model Class Diagram



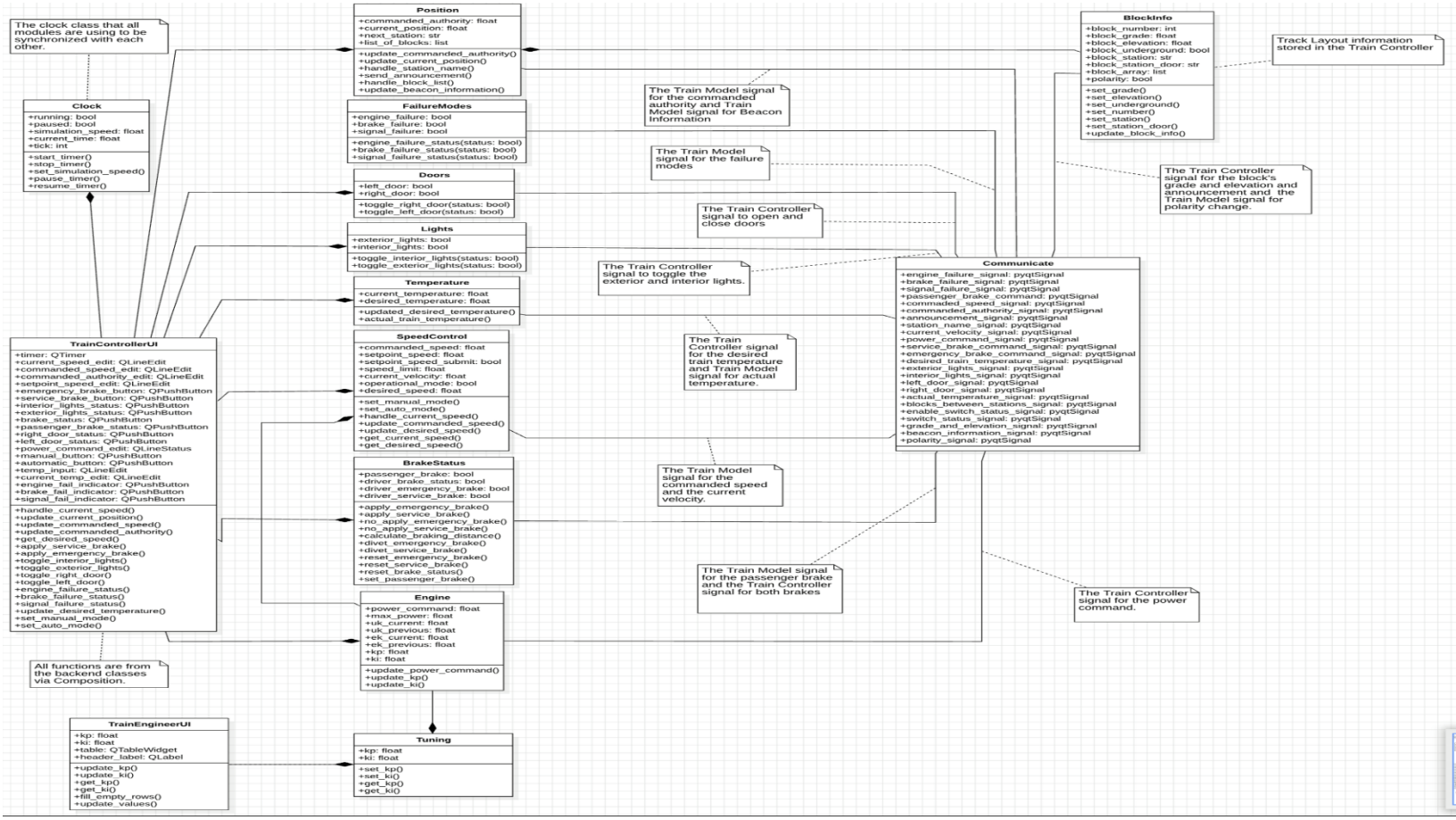
Explanation: Class Diagram for the Track Model, illustrating the core classes and their relationships between each other. There also exist classes that enable inter-system communication between the Track Model and its adjacent modules, namely, the Wayside-Track Communicate and the TrackModel Communicate classes. The diagram above shows how the Track Model acts a presentation of a physical track layout by monitoring track occupancies, representing switches, interfacing with other subsystems such as the Wayside Controller and Train Model. Furthermore, a Clock class is an essential component that is included in each module in order ensure that all modules have synchronized timing which allows inter-communicaion to occur smoothly and efficiently.

5.3.4 Train Model Class Diagram



Explanation: Class Diagram for the Train Model, illustrating the core classes and their relationships, including Power System, Brake System, Door/Light Status, TrainData, and Weight Status. There are also classes with help explain the inter-system communication between the Train Model and its adjacent modules, which are the Track Model and the Train Controller, such as the TrackComms and the TrainComms classes. This diagram illustrates how the Train Model represents the train's internal systems, including controlling speed and authority, managing doors, and operating lights, while interacting with external systems like the Train Controller and Track Model. Finally, there is also a Clock class that all of the modules have in order ensure that all modules have their timing synchronized to the same exact second, so that communication to occur smoothly and efficiently.

5.3.5 SW Train Controller Class Diagram



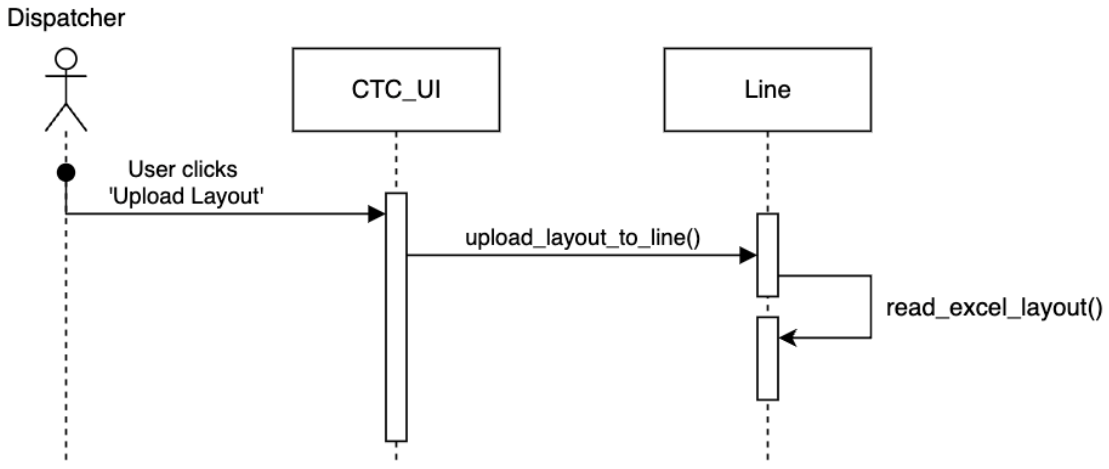
Explanation: Class Diagram for the Train Controller, illustrating the core classes and their relationships, including Engineer, Engine, Position, Lights, Doors, Temperature, SpeedControl, FailureModes, and BrakeStatus. There is also a class that helps explain the inter-system communication between the Train Communication and its adjacent module, which is the Train such as the Communicate class. This diagram shows how the Train Controller interacts exclusively with the Train Model to find power commands, monitor failure modes, trigger emergency and service brakes, control door operations, turn lights on and off, allow the driver to input speed and changing the train's authority, while interacting with the Train Model. Finally, there is also a Clock class that all of the modules have in order ensure that all modules have their timing synchronized to the same exact second, so that communication occurs smoothly and efficiently.

5.4 Interaction Viewpoint

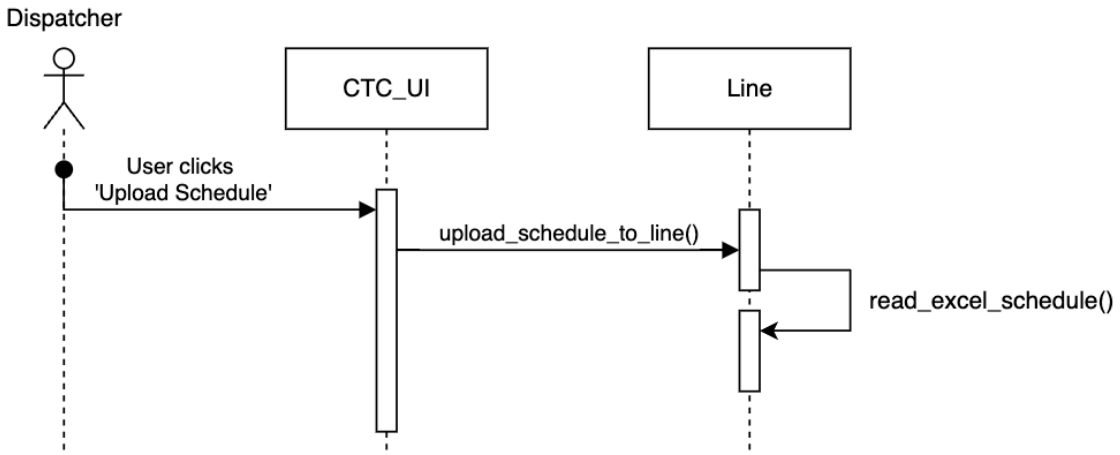
The **Interaction Viewpoint** focuses on the strategies for interactions among system entities, including the reasons, locations, methods, and levels at which these interactions occur. In this section, sequence diagrams are used to represent the dynamic behavior of the system, illustrating the flow of messages and actions between different components over time. These diagrams provide a detailed view of how entities interact with one another to fulfill specific use cases, helping to ensure that the timing, sequence, and coordination of actions are well-defined and align with the system's design goals.

5.4.1 CTC Sequence Diagrams

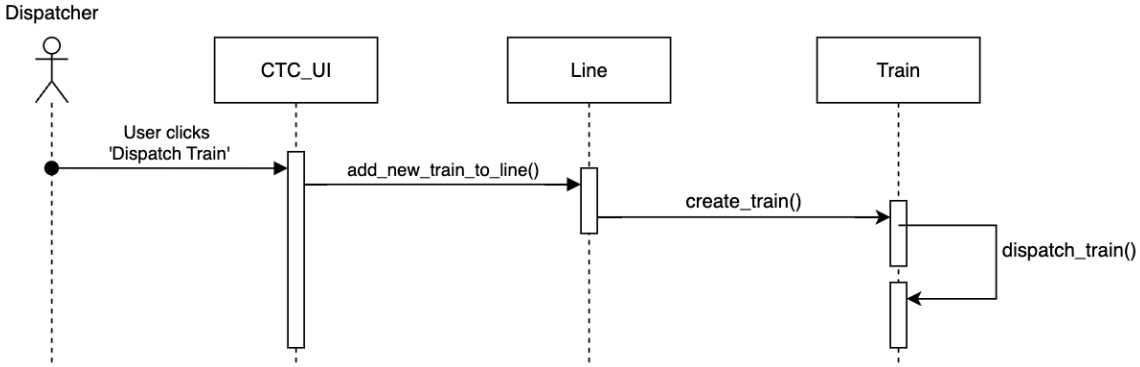
5.4.1.1 Upload Layout



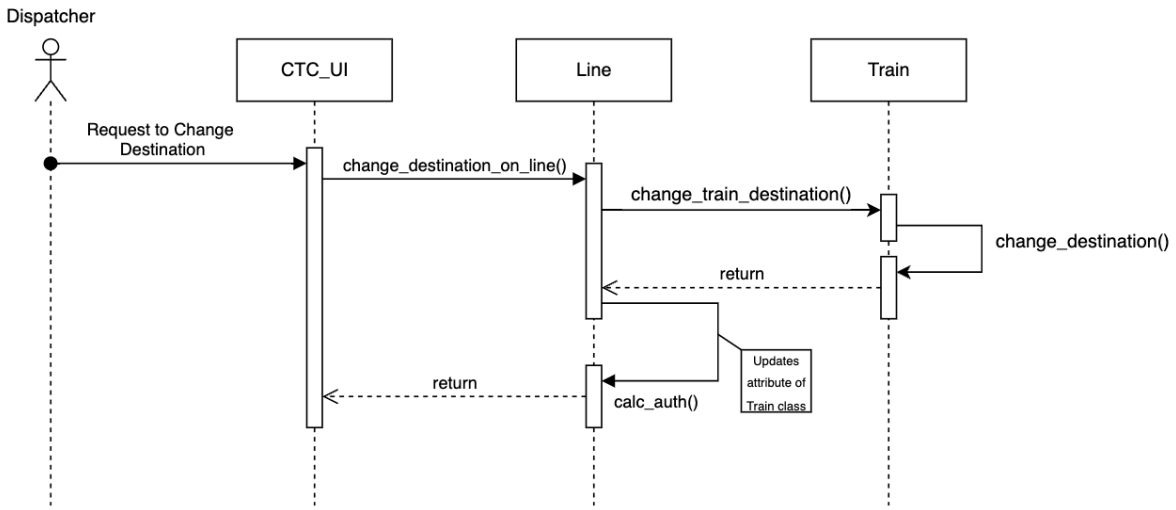
5.4.1.2 Upload Schedule



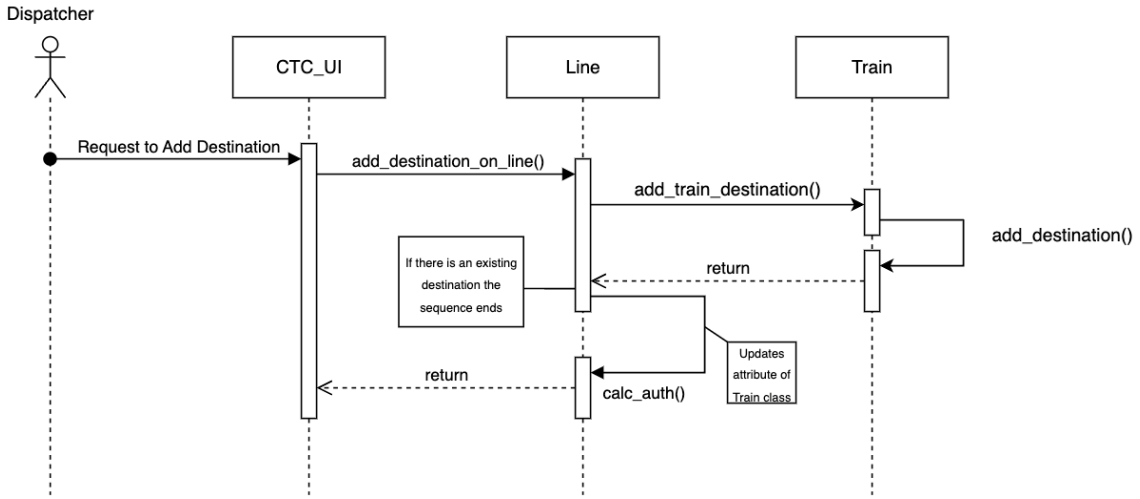
5.4.1.3 Manually Schedule a Train



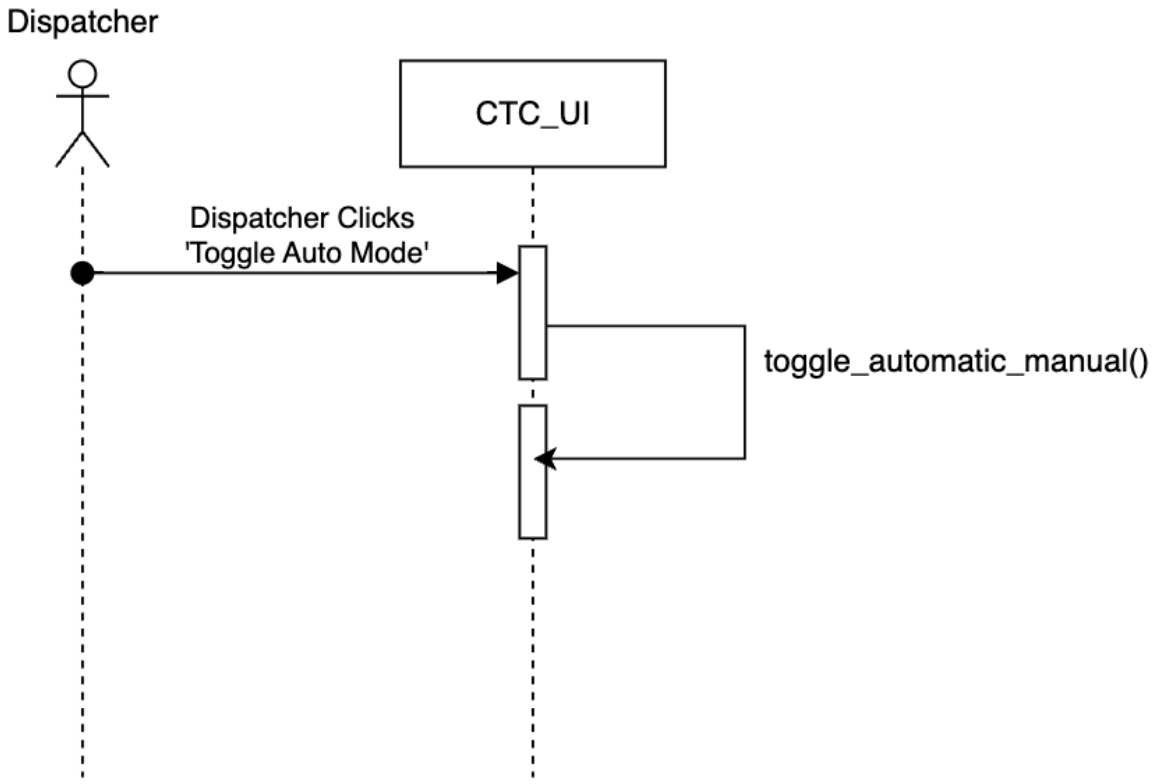
5.4.1.4 Change Train Destination



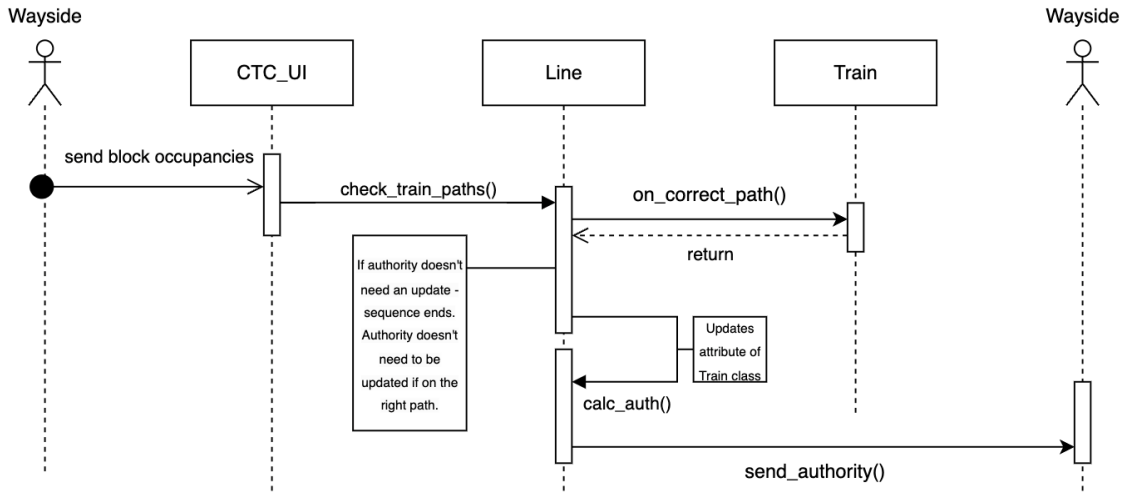
5.4.1.5 Add Train Destination



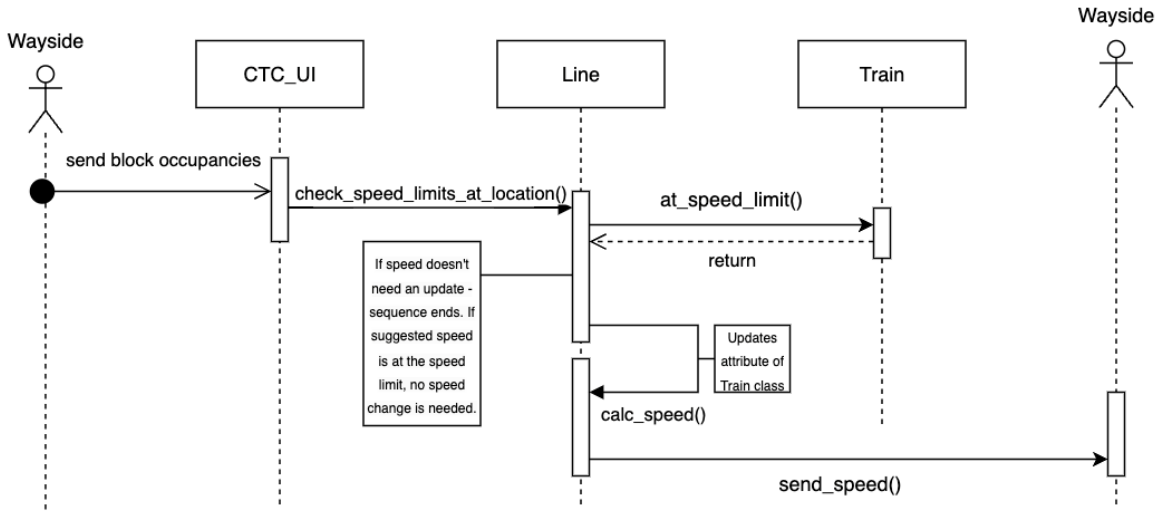
5.4.1.6 Toggle Between Automatic and Manual Mode



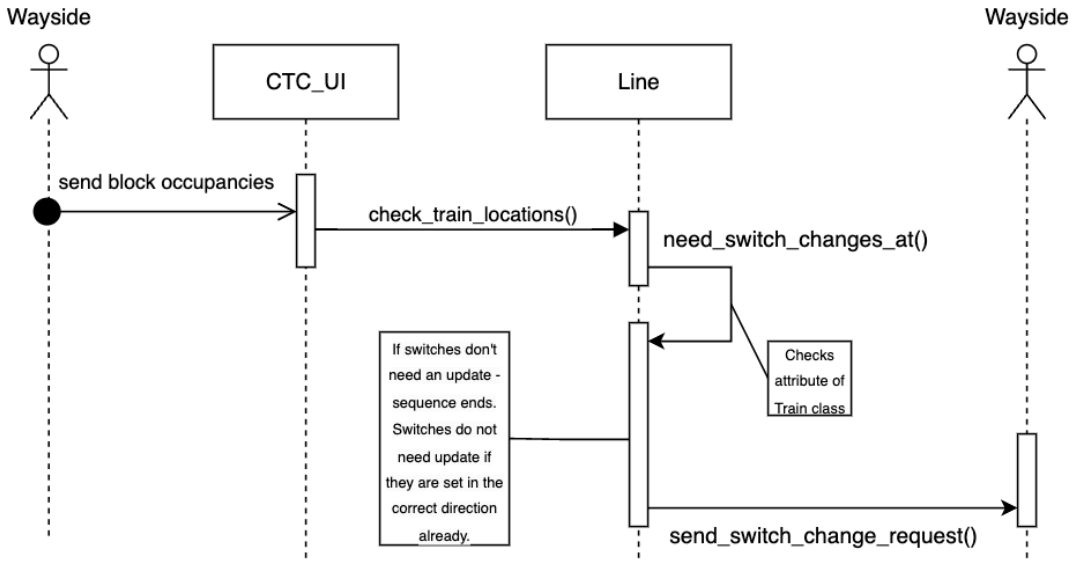
5.4.1.7 Send Suggested Authority



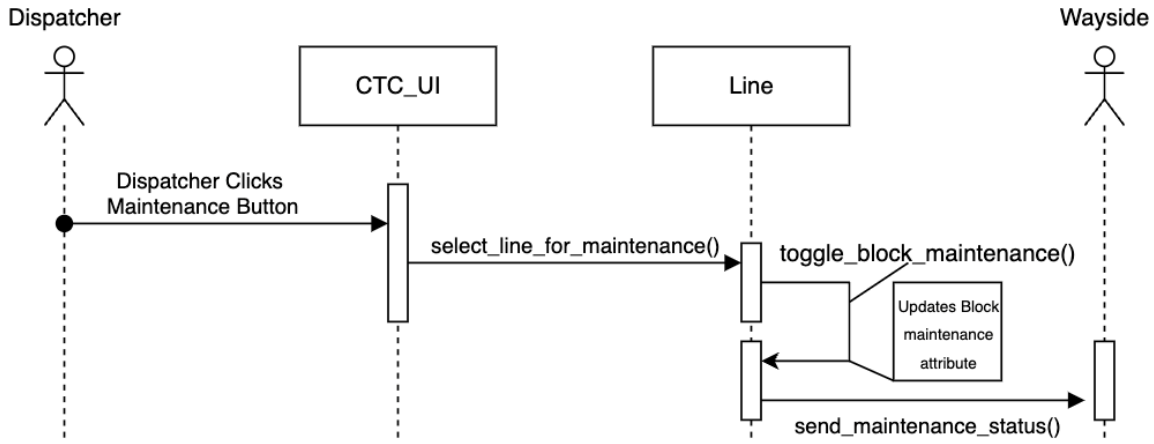
5.4.1.8 Send Suggested Speed



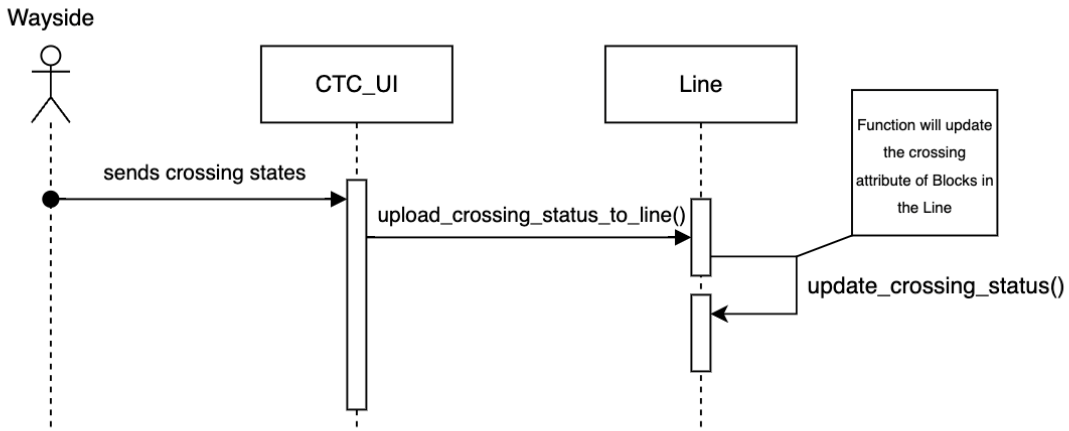
5.4.1.9 Send Suggested Switch Command



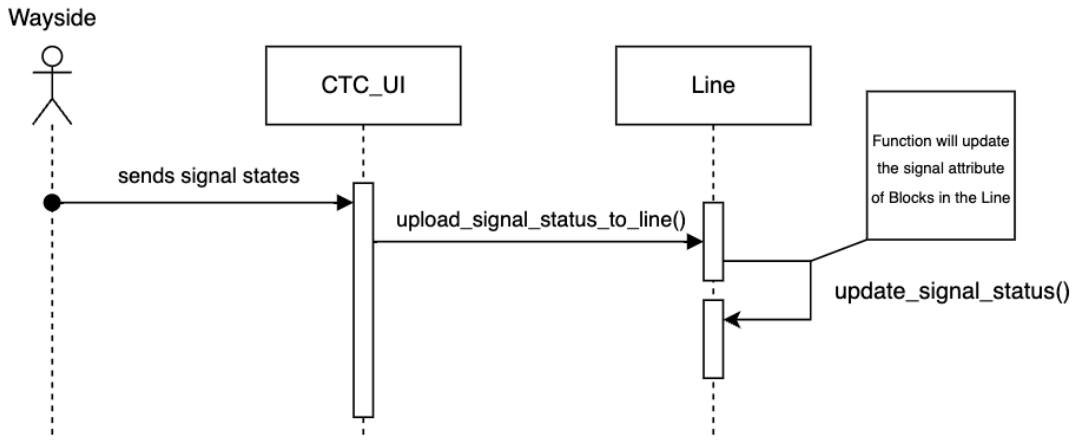
5.4.1.10 Send Maintenance States



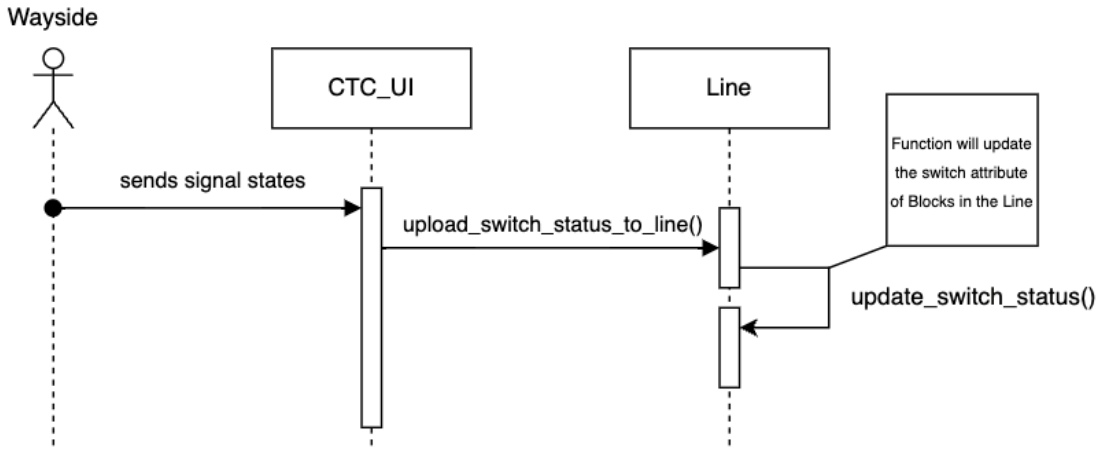
5.4.1.11 Receive Crossing States



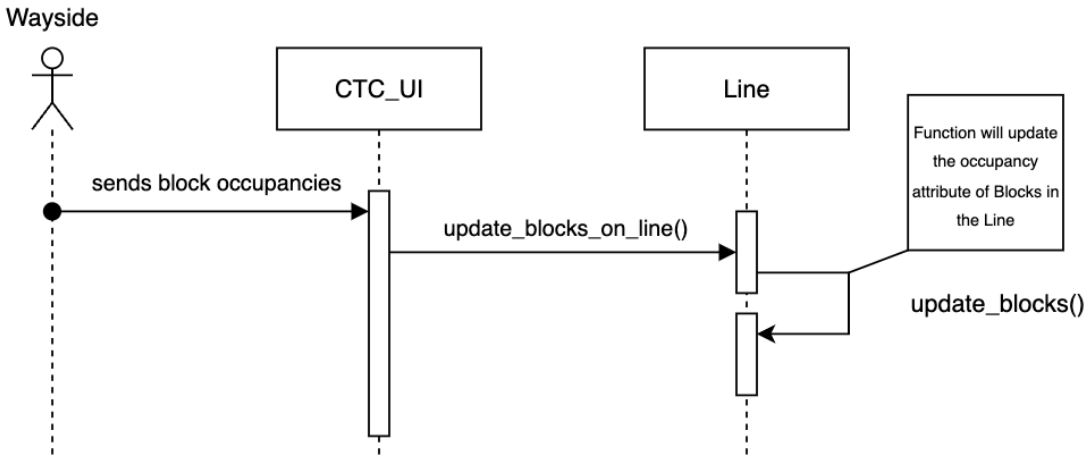
5.4.1.12 Receive Signal States



5.4.1.13 Receive Switch States

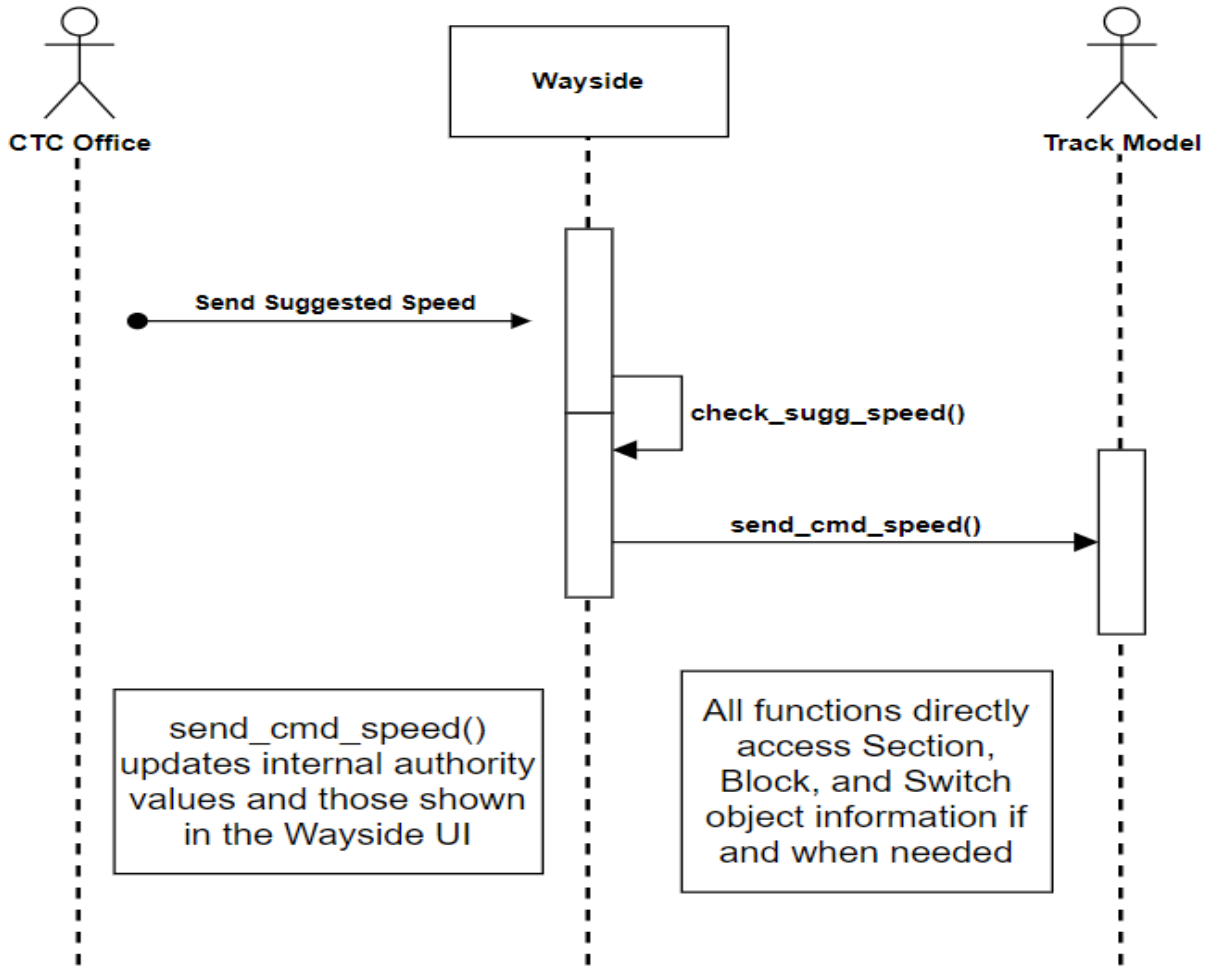


5.4.1.14 Receive Block Occupancies



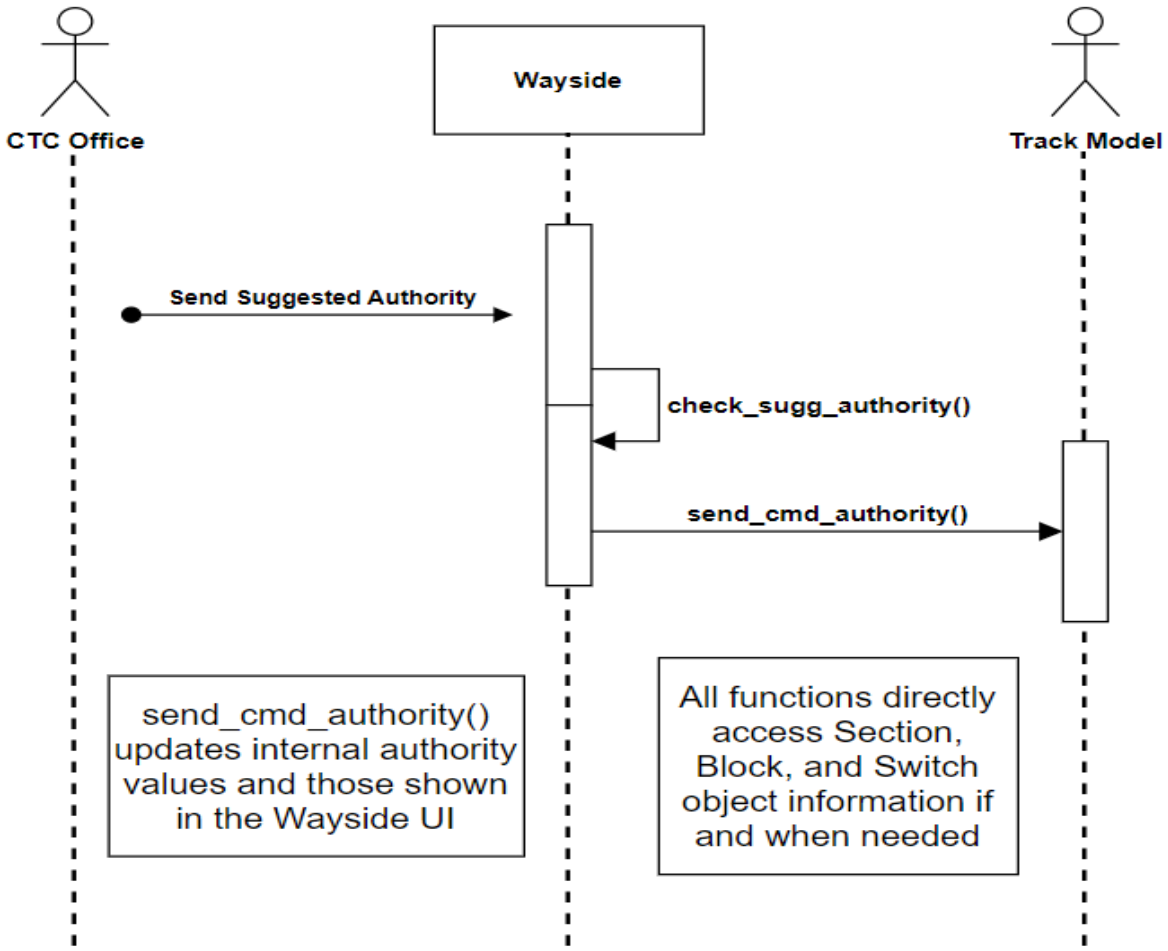
5.4.2 Wayside Controller Sequence Diagrams

5.4.2.1 Receive Suggested Speed and Send Commanded Speed



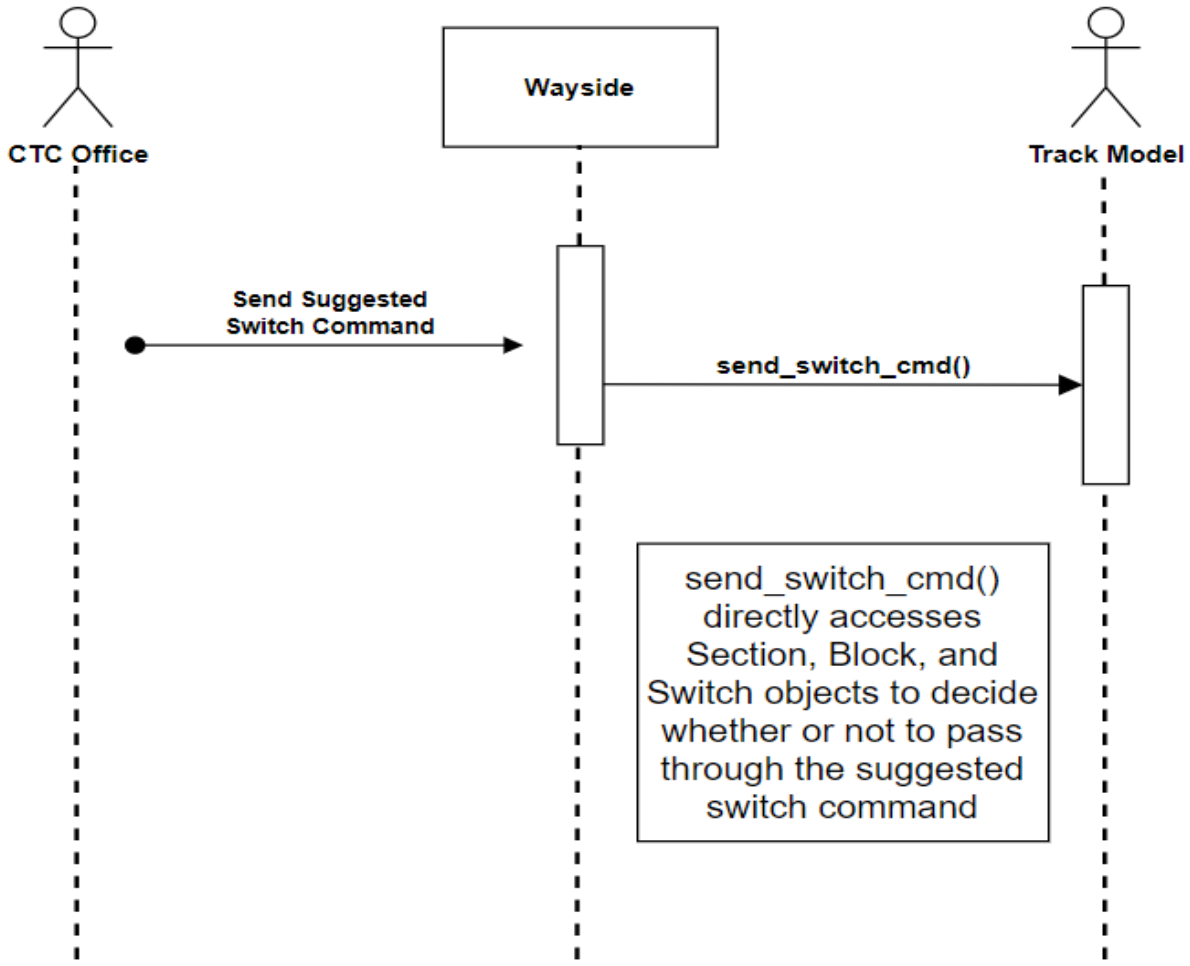
Explanation: Sequence diagram for the Wayside Controller, illustrating the process of receiving a suggested speed from the CTC Office and deciding whether or not to send the value to the Track Model. This diagram shows interactions between the CTC Office, the Wayside class, and the Track Model. First the suggested speed is sent from the CTC Office. Once received by the Wayside class, the suggested speed is processed and deemed safe or not safe. Finally, if the speed value is deemed safe, then it is sent to the Track Model; otherwise, the speed value is not sent to the Track Model.

5.4.2.2 Receive Suggested Authority and Send Commanded Authority



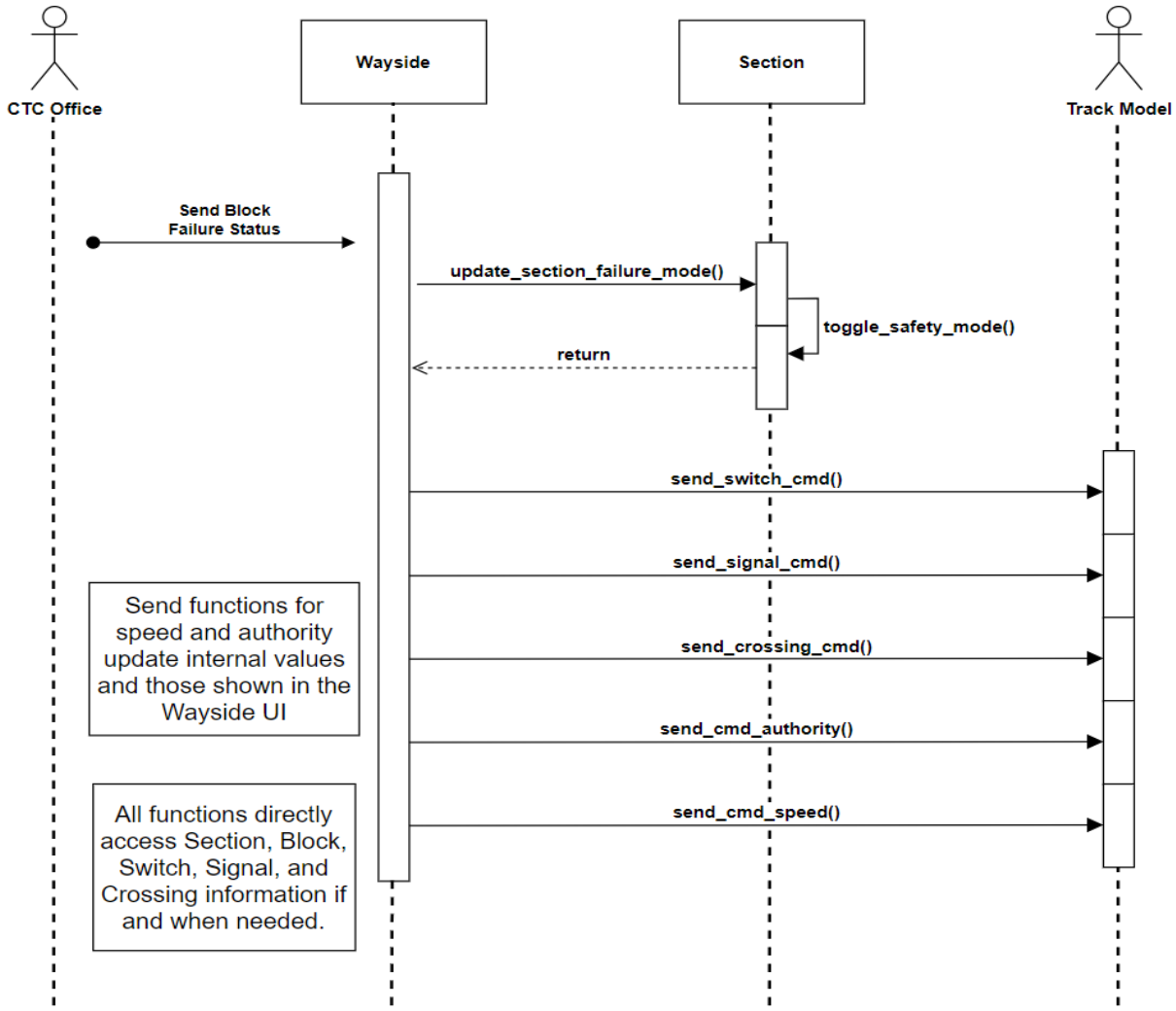
Explanation: Sequence diagram for the Wayside Controller, illustrating the process of receiving a suggested authority from the CTC Office and deciding whether or not to send the value to the Track Model. This diagram shows interactions between the CTC Office, the Wayside class, and the Track Model. First a suggested authority is sent from the CTC Office. Once received by the Wayside class, the suggested authority is processed and deemed safe or not safe. Finally, if the authority value is deemed safe, then it is sent to the Track Model; otherwise, the authority value is not sent to the Track Model.

5.4.2.3 Receive Suggested Switch Command and Send Commanded Switch Command



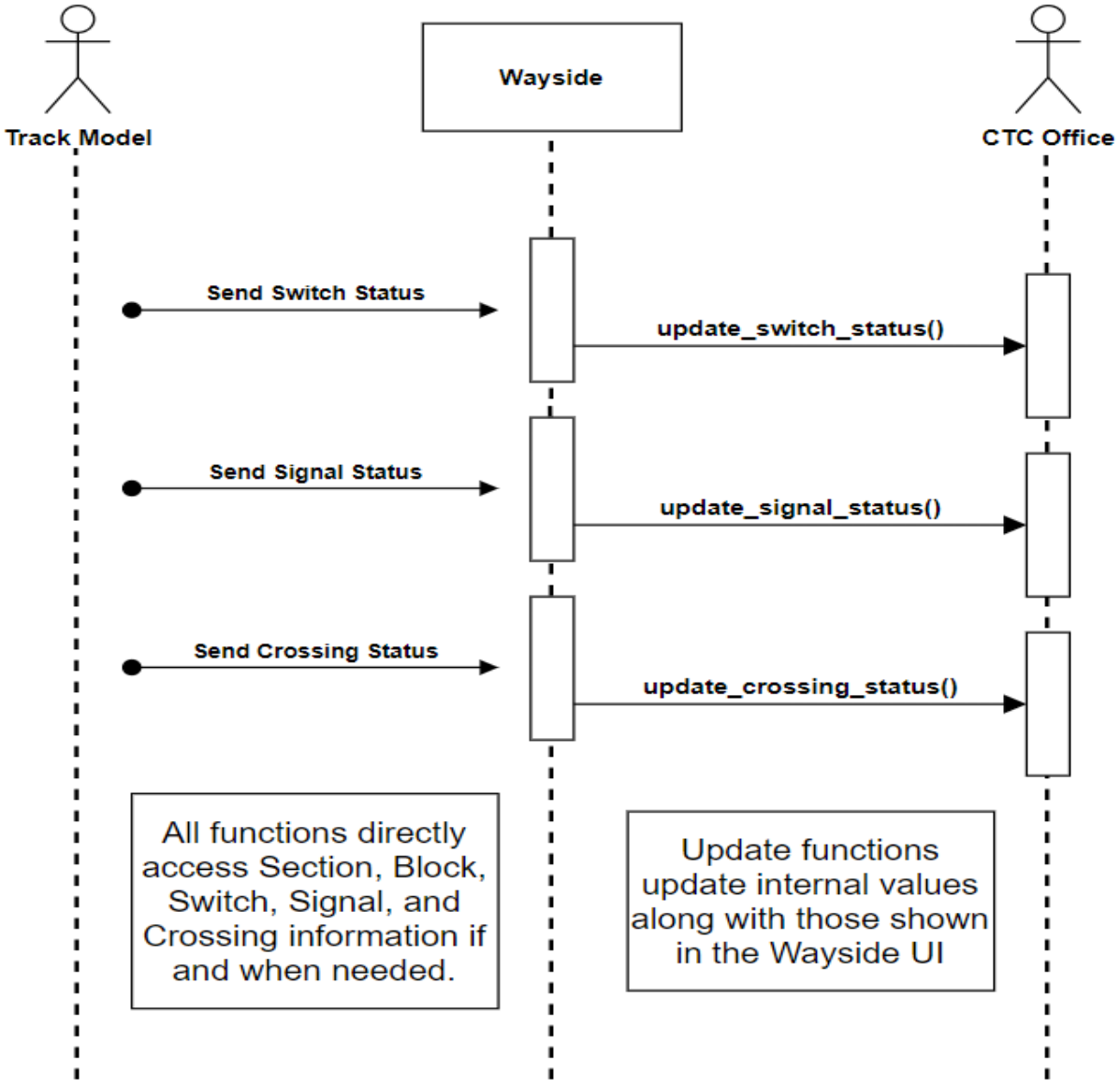
Explanation: Sequence diagram for the Wayside Controller, illustrating the process of receiving a suggested switch command from the CTC Office and deciding whether or not to send the value to the Track Model. This diagram shows interactions between the CTC Office, the Wayside class, and the Track Model. First a suggested command is sent from the CTC Office. Once received by the Wayside class, the suggested command is processed and deemed safe or not safe. Finally, if the command is safe it is sent to the Track Model; otherwise, the command is not sent to the Track Model.

5.4.2.4 Receive Failure Status



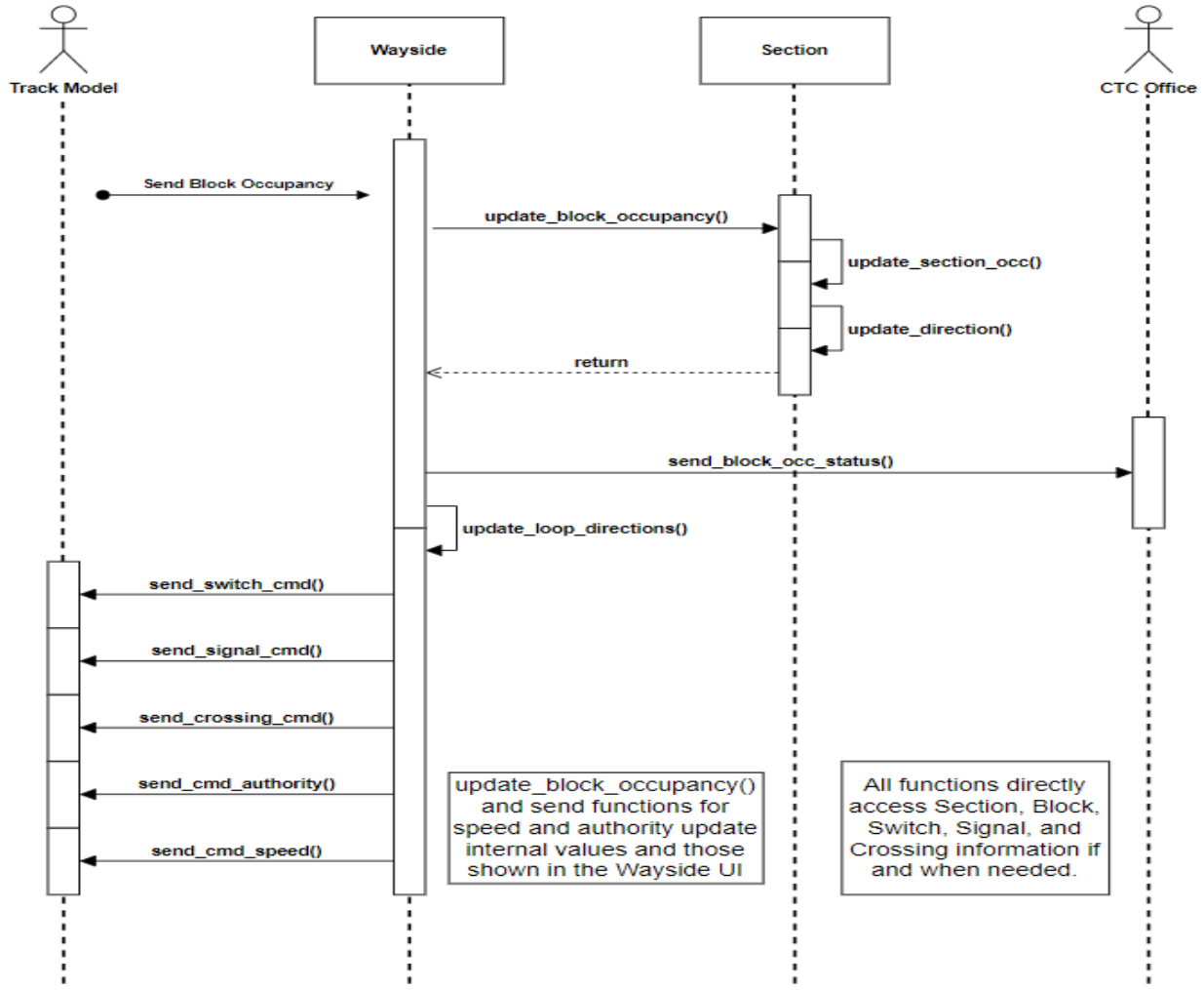
Explanation: Sequence diagram for the Wayside Controller, illustrating the process of receiving a block failure status from the CTC Office and updating outgoing commands to the Track Model according to which block has failed. This diagram shows interactions between the CTC Office, the Wayside class, the Section class, and the Track Model. The CTC tells the wayside which block and section a failure has occurred. Then the wayside will update the failure status of the specific section where the failure occurred. Then the specific failed section will update its Block objects' attributes according to which specific block failed. Finally, any updated commands are sent out to the Track Model.

5.4.2.5 Receive and Report Switch, Signal, and Crossing Statuses



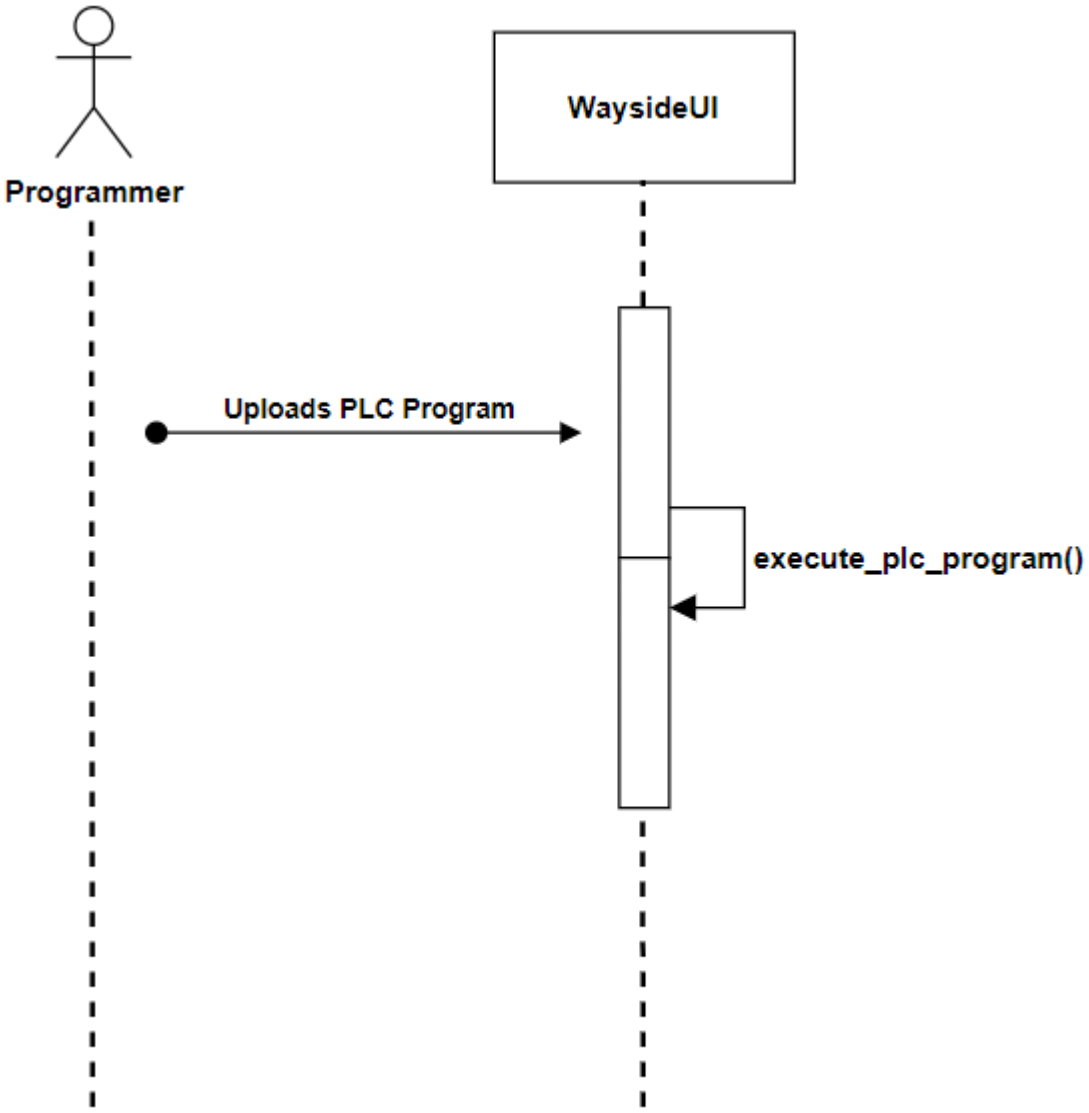
Explanation: Sequence diagram for the Wayside Controller, illustrating the process of receiving a switch, signal, and crossing status from the Track Model and reporting those statuses to the CTC Office. First the Tack Model sends the switch, signal, and crossing statuses to the wayside class. Then the wayside class updates its own internal values and reports those updated values to the CTC Office.

5.4.2.6 Receive and Report Block Occupancies



Explanation: Sequence diagram for the Wayside Controller, illustrating the process of receiving a block occupancy status from the Track Model and reporting the new occupancy information to the CTC Office. First the Tack Model sends block occupancy status to the wayside class. Then the wayside class updates its own internal occupancy values. Then individual sections will update their own overall occupancy status and determine the direction of movement within the section. Then block occupancies are reported to the CTC Office. Then the wayside will update the direction of any loops within its jurisdiction. Finally, any updated commands are sent out to the Track Model based on the new block occupancy information.

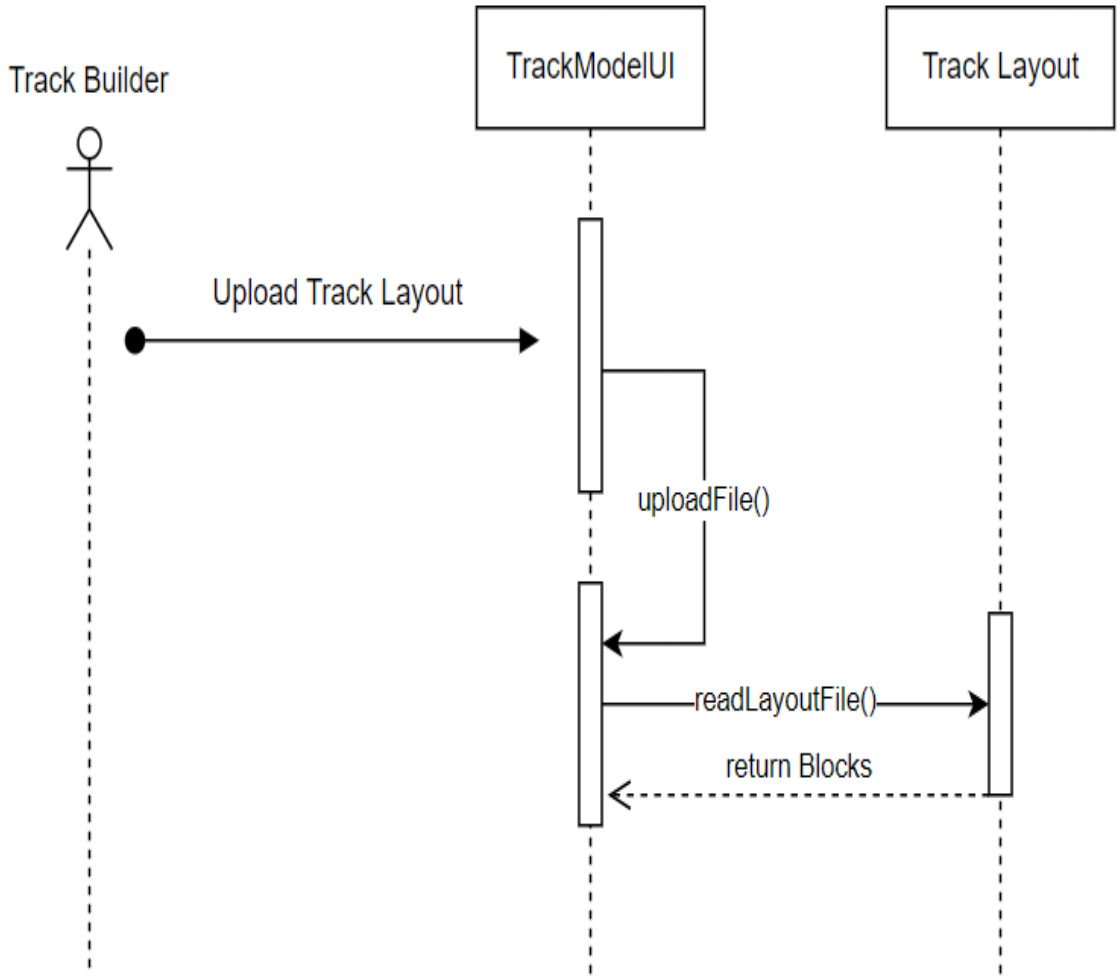
5.4.2.7 Upload PLC Program



Explanation: Sequence diagram for the Wayside Controller, illustrating the process of the programmer uploading a PLC program to the wayside. First, the programmer uploads a folder to the Wayside UI containing all the PLC programs meant for each of wayside on the track. Then each PLC program will be distributed to its corresponding wayside on the track.

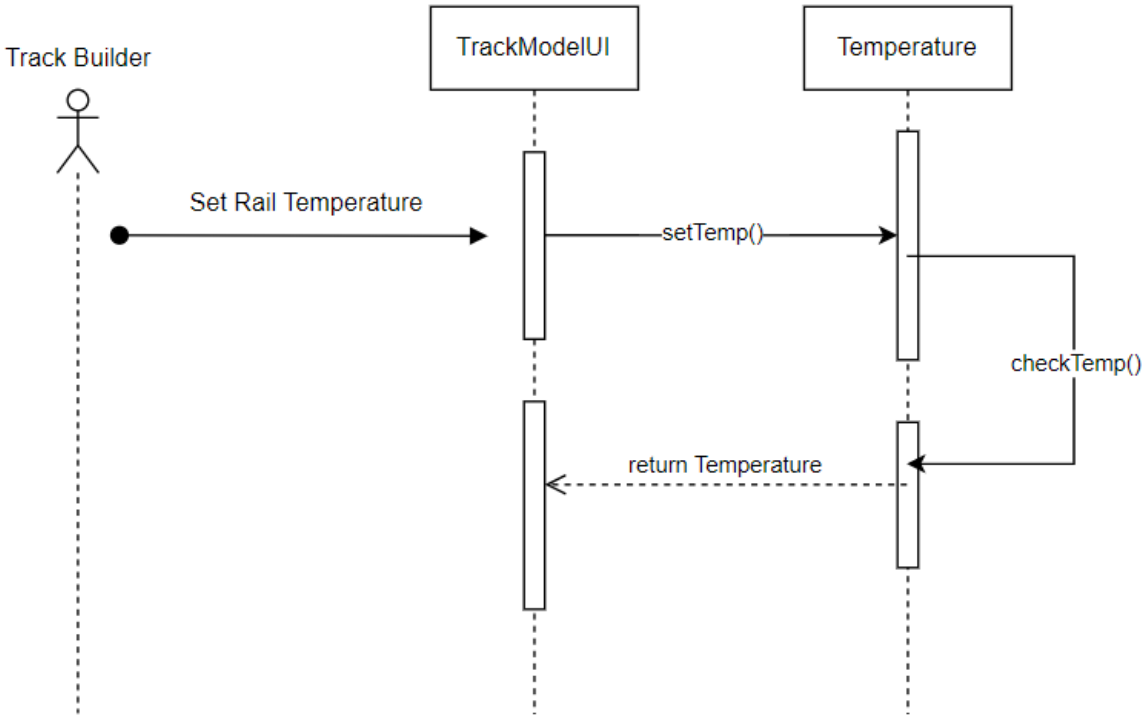
5.4.3 Track Model Sequence Diagrams

5.4.3.1 Upload Track Layout



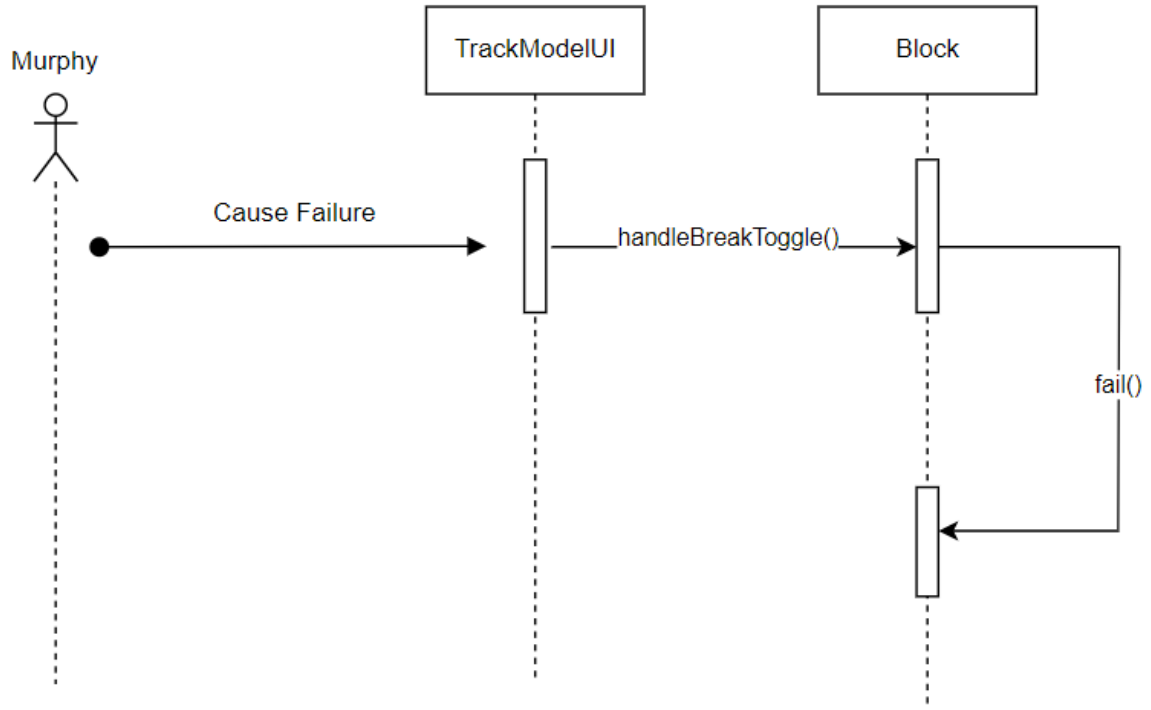
Explanation: Sequence Diagram for uploading a new track layout to the Track Model UI, illustrating the process of uploading information for all the components of the railway track. This diagram depicts how the Track Builder interacts with the Track Model UI to upload a track layout from a database file. The Track Model then accesses the data in the Track Layout that is necessary to instantiate the railway blocks.

5.4.3.2 Set Rail Temperature



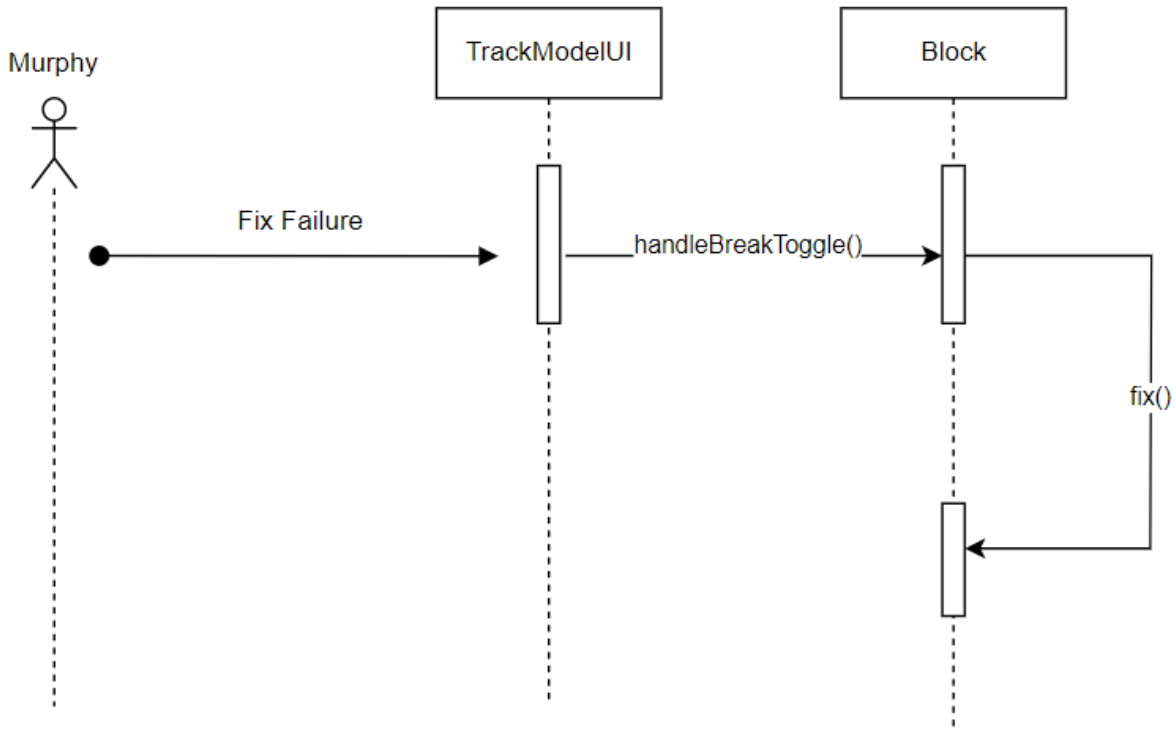
Explanation: Sequence Diagram for setting the temperature of the rails, illustrating the process of determining an appropriate temperature for railway operation. This diagram depicts the Track Builder inputting a set rail temperature within the user interface. Then, the Track Model UI interacts with the Temperature class to make the appropriate adjustments to the temperature. Finally, the changes are reflected in the Track Model UI.

5.4.3.3 Cause Failure on a Block



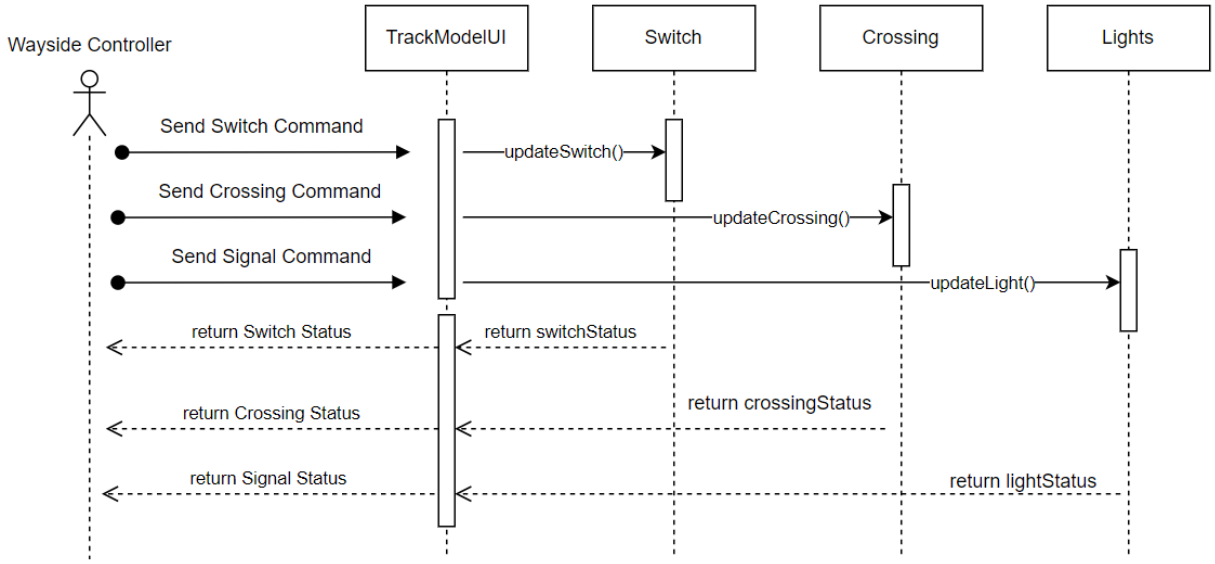
Explanation: Sequence Diagram for initiating a failure on a block, illustrating the process of Murphy putting a block into failure. This diagram depicts Murphy selecting a failure mode and a block to put into failure within the Track Model UI. Then, the Track Model UI handles this request and makes the necessary changes to the block's attributes.

5.4.3.4 Fix Failures on a Block



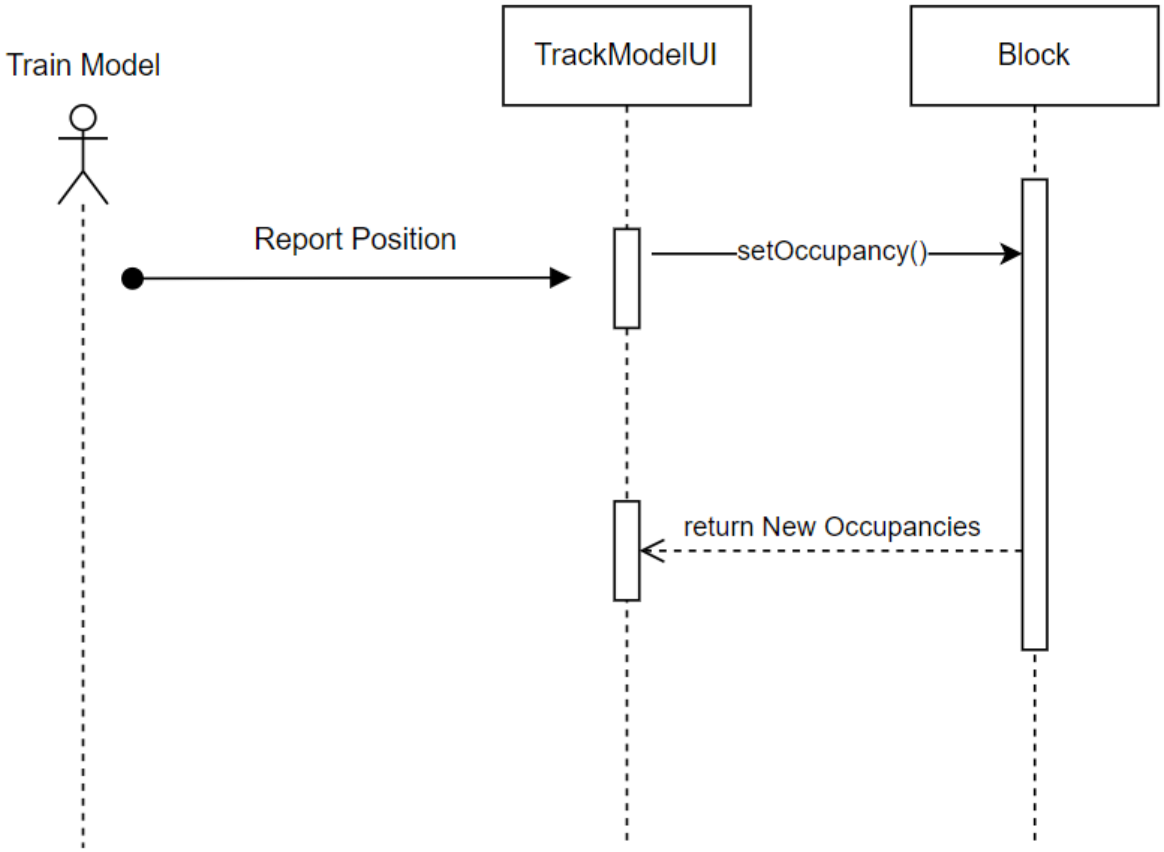
Explanation: Sequence Diagram for fixing the failure on a block, illustrating the process of returning the block to its proper functional state. This diagram depicts Murphy toggling the failure status of a failed block within the user interface. Then, the Track Model UI handles this request and makes the necessary changes to the block's attributes.

5.4.3.5 Receive Switch, Crossing, Light Commands and Send Respective Statuses



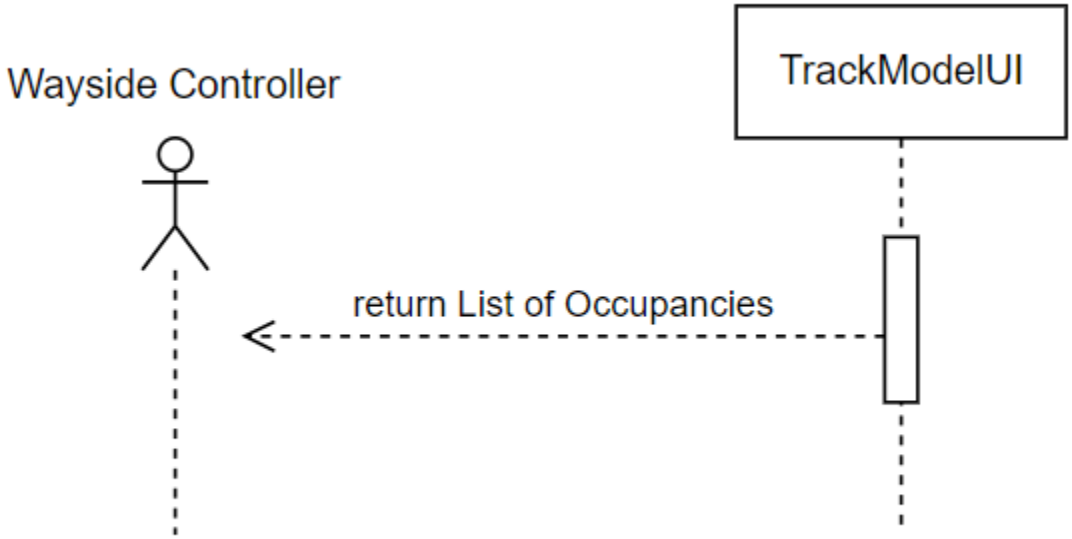
Explanation: Sequence Diagram for receiving switch, signal, and crossing commands and sending the respective statuses. The diagram above illustrates the flow of commands and signals from the Wayside Controller and the Track Model. As shown, the Wayside Controller sends switch, crossing, and light statuses to the Track Model in accordance with routing and safety guidelines. Then, the Track Model updates the statuses of the switches, crossings, and lights. Finally, the statuses are reflected in the Track Model and sent to the Wayside Controller to be processed.

5.4.3.6 Determine Occupancies



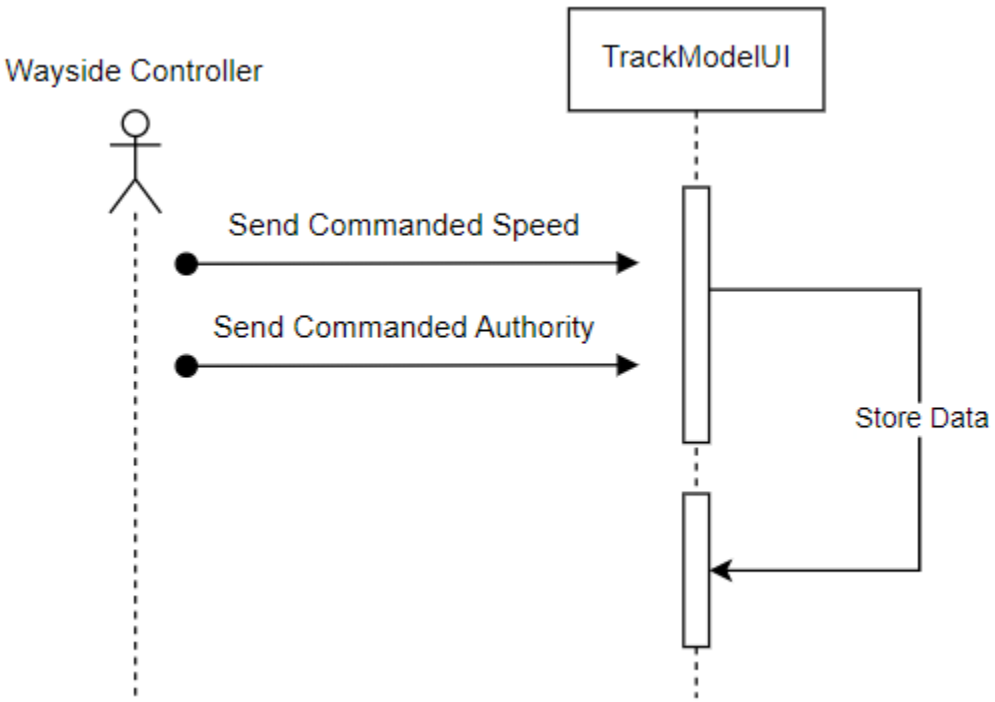
Explanation: Sequence Diagram for determining occupancies on the track. As shown, the Train Model reports its position to the Track Model. Then, the Track Model updates the block occupancies, and the changes are reflected in the Track Model UI.

5.4.3.7 Report Occupancies



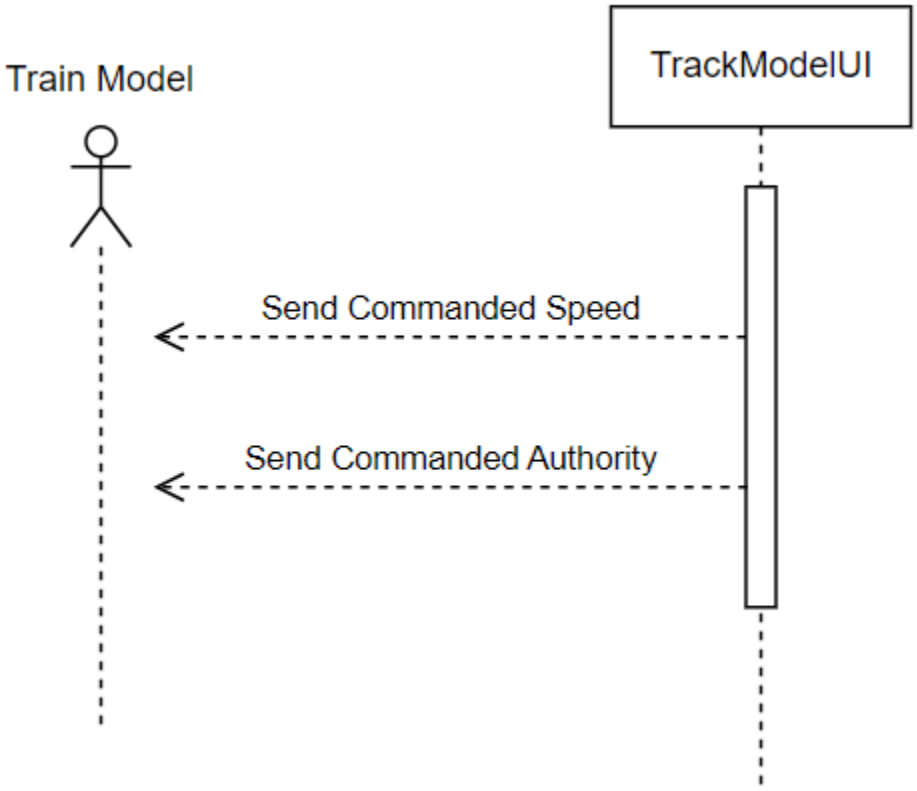
Explanation: Sequence Diagram for sending occupancies to the Wayside Controller. As depicted, Track Model takes the list of block occupancies on the track and sends the report to the Wayside Controller.

5.4.3.8 Receive Commanded Speed and Authority



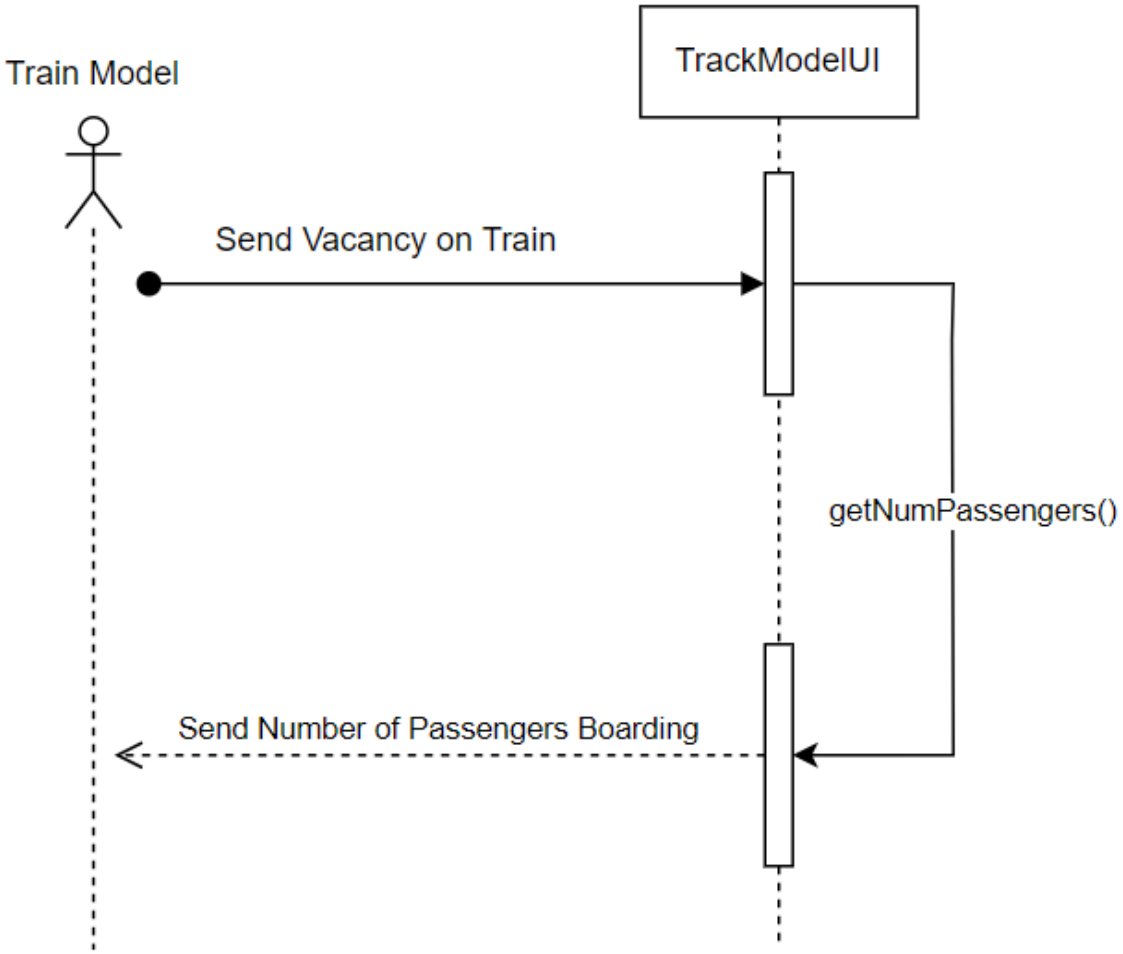
Explanation: Sequence Diagram for receiving commanded speed and authority. The diagram illustrates the Wayside Controller sending commanded speed and authority to the Track Model where is then stored until further notice.

5.4.3.9 Send Commanded Speed and Authority



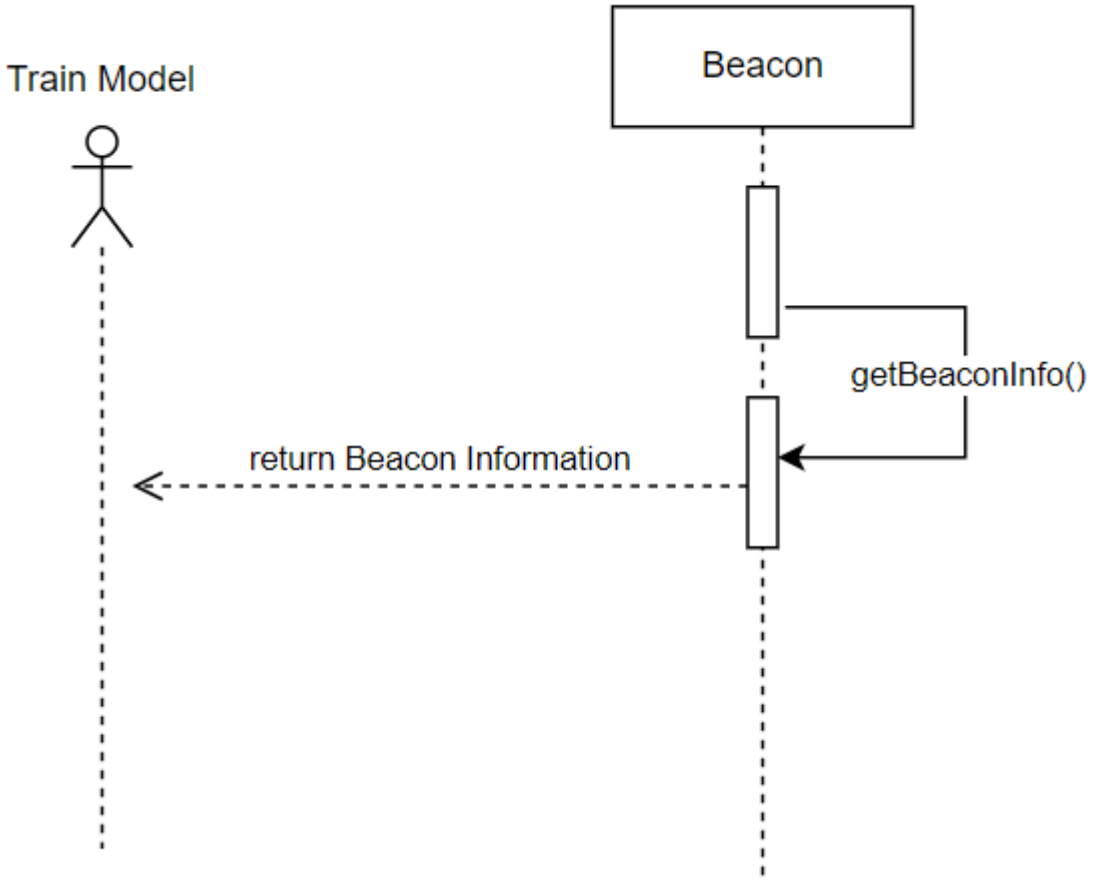
Explanation: Sequence Diagram for sending commanded speed and authority. The diagram illustrates the Track Model retrieving the stored commanded speed and authority and sending the values to the Train Model.

5.4.3.10 Get Number of Passengers



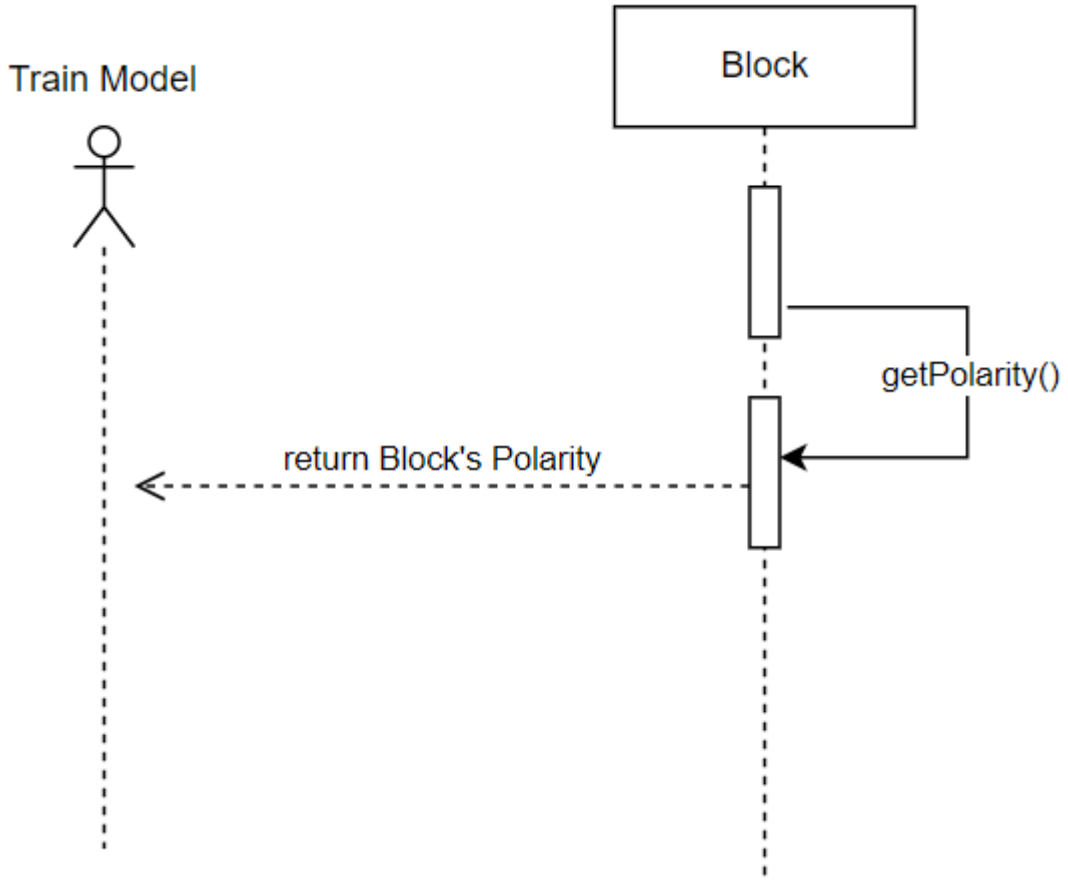
Explanation: Sequence Diagram for getting number of passengers, illustrating how the number of passengers on the train as it leaves the station is obtained. As shown, the Train Model sends an amount of open seats on the train at a given station. The number is used within the Track Model to generate an amount of passengers boarding at said station. This result is then sent back to the Train Model

5.4.3.11 Send Beacon Information



Explanation: Sequence Diagram for sending beacon information, illustrating how the beacon sends static information to the Train Model as it passes. As shown, the Track Model retrieves the static information stored in the Beacon and sends that data to the Train Model.

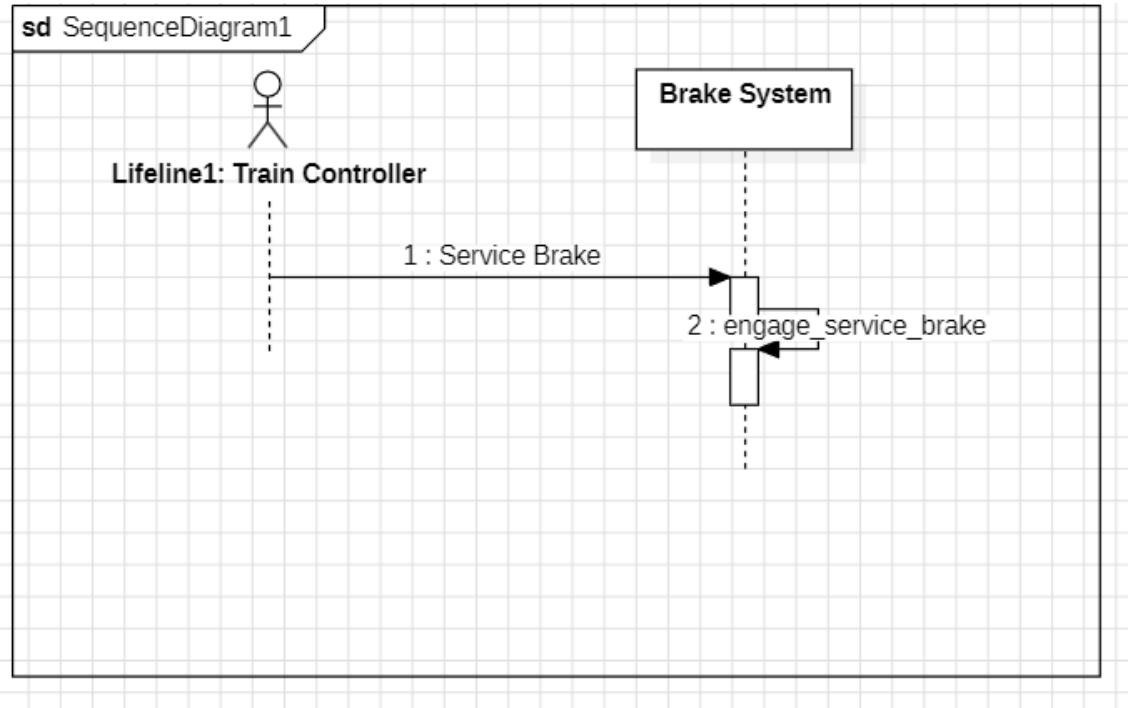
5.4.3.12 Send Polarity



Explanation: Sequence Diagram for sending polarity. The diagram above illustrates how Track Model retrieves an individual block's polarity and sends it to the Train Model. As shown, the Block class has a polarity attribute which is accessed by a getter function. The Boolean value is then sent to the Train Model in order for the train to determine when it moves from block to block.

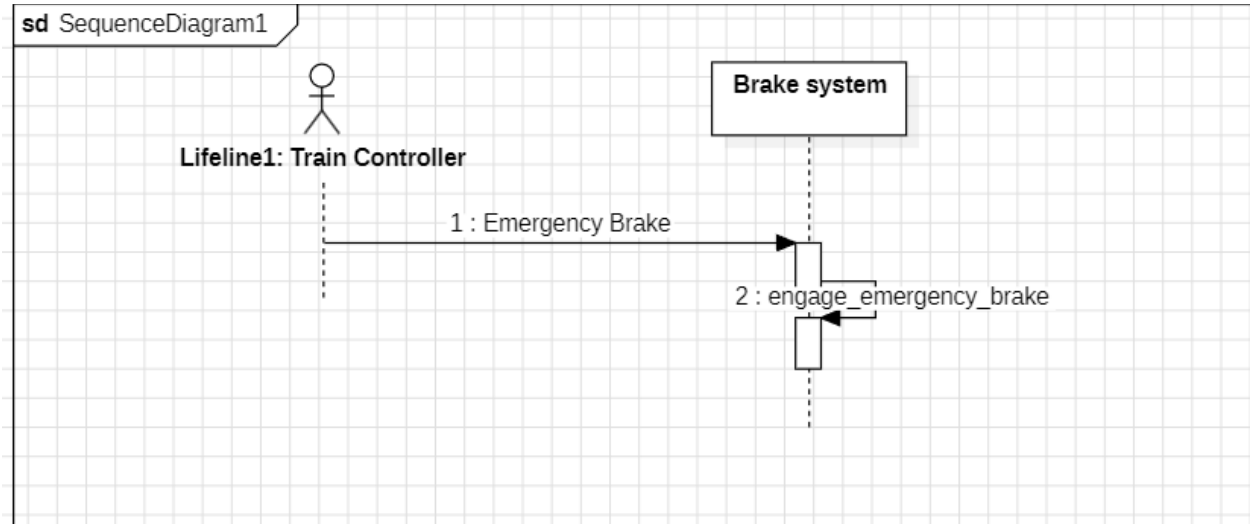
5.4.4 Train Model Sequence Diagrams

5.4.4.1 Receive Service Brake Command



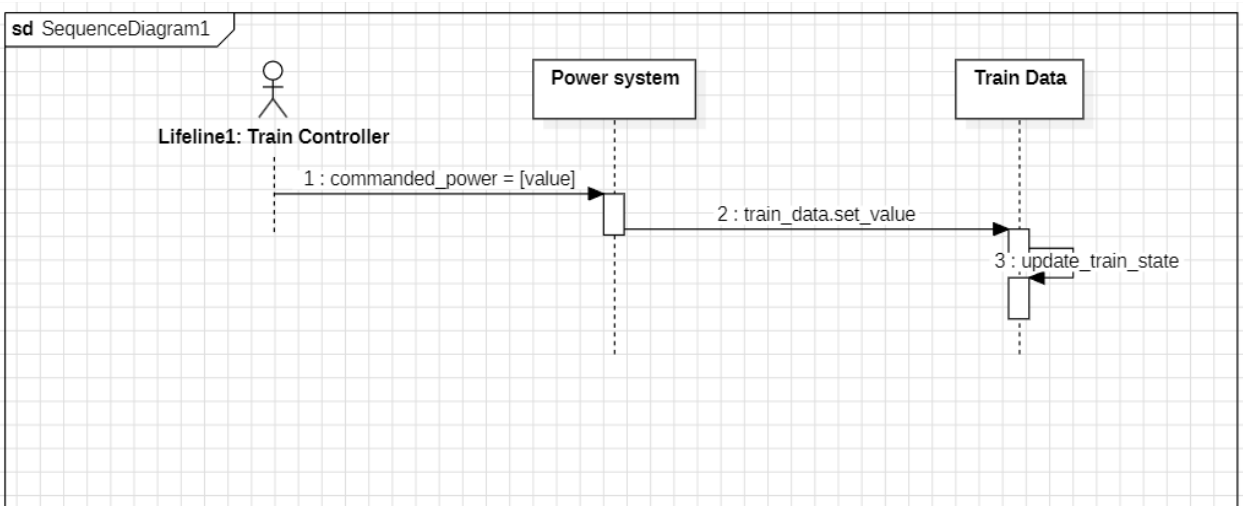
Explanation: Sequence Diagram for the Train Controller, illustrating the process of the train driver setting the setpoint speed of the train and how the current train engine power is calculated from that. This diagram shows interactions between the train driver, TrainControllerUI, SpeedControl, Tuning, Engine, and the TrainModel users and classes as the system takes in the setpoint speed input and uses the desired and current speed value, and the Kp and Ki values to find the current power command of the train.

5.4.4.2 Receive Emergency Brake Command



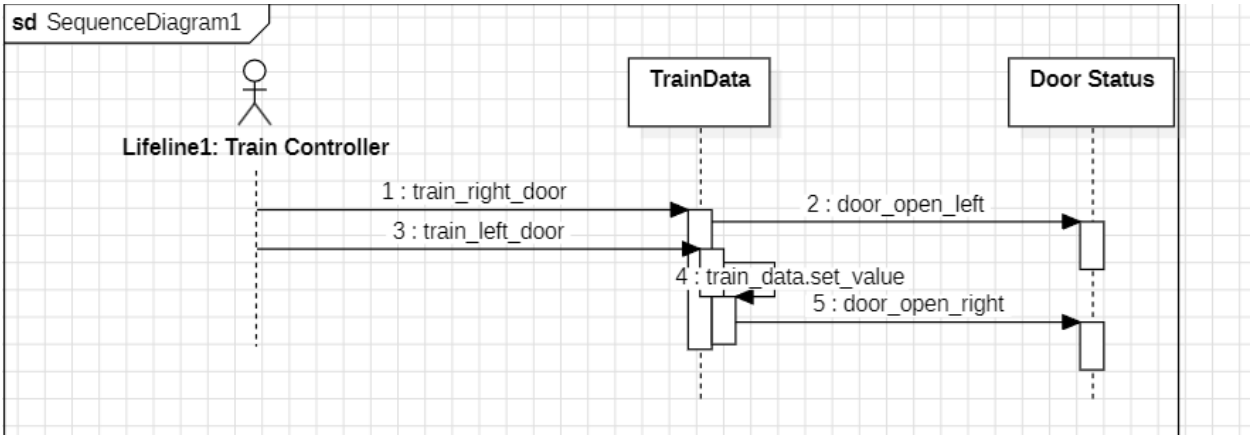
Explanation: Sequence Diagram for receiving the emergency brake command. As shown, the Train Controller triggers the emergency brake by sending a command to the Train Model. The brake system within the train model calls a function the engage the emergency brake.

5.4.4.3 Receive Power Command



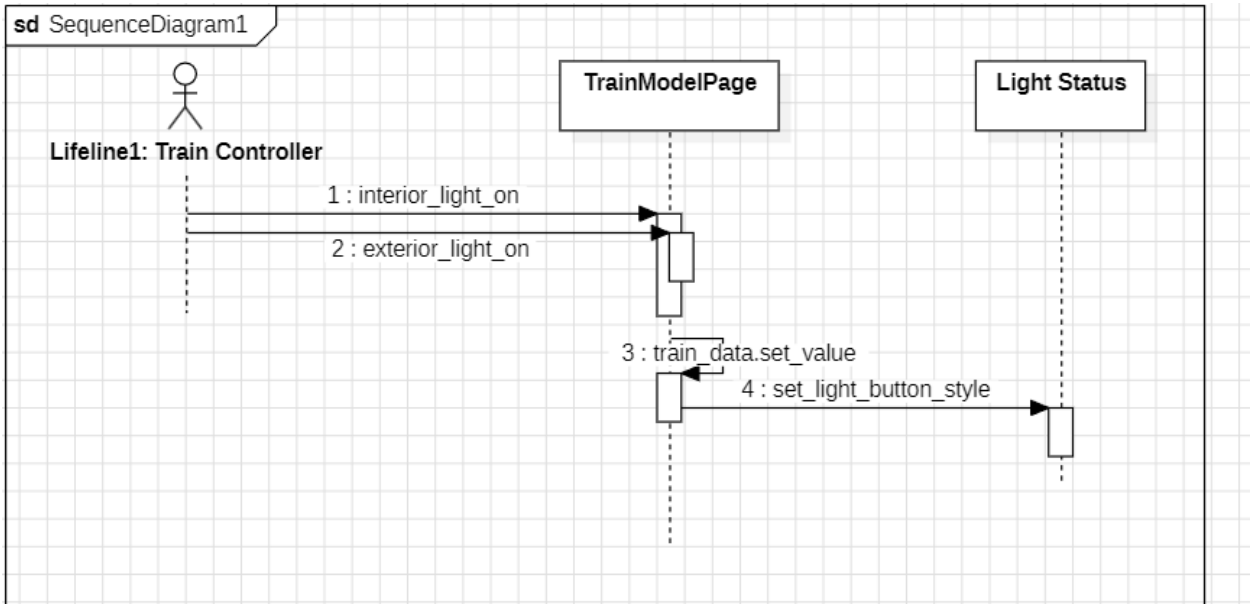
Explanation: Sequence Diagram for receiving power command. As shown, the Train Controller sends a value for the commanded power of the train which is fed into the Train Model's power system. The Power System handles the value by accessing the train data and updating the train's state.

5.4.4.4 Receive Door Signals



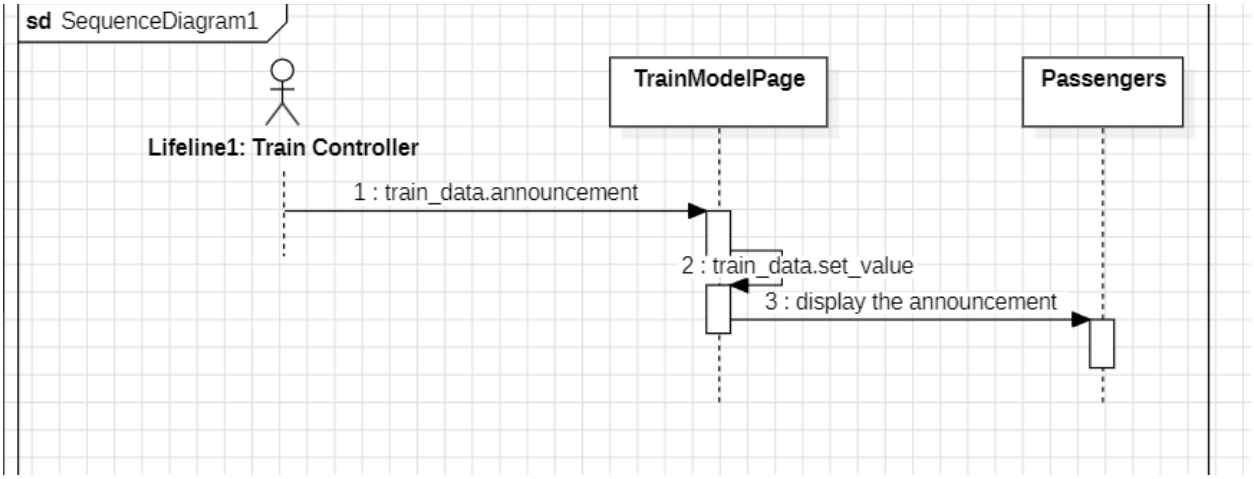
Explanation: Sequence Diagram for receiving door signals. As depicted in the diagram, the Train Controller sends Boolean signals for the train's right and left door, individually. The train data class handles and sets the values to then command the door statuses.

5.4.4.5 Receive Light Signals



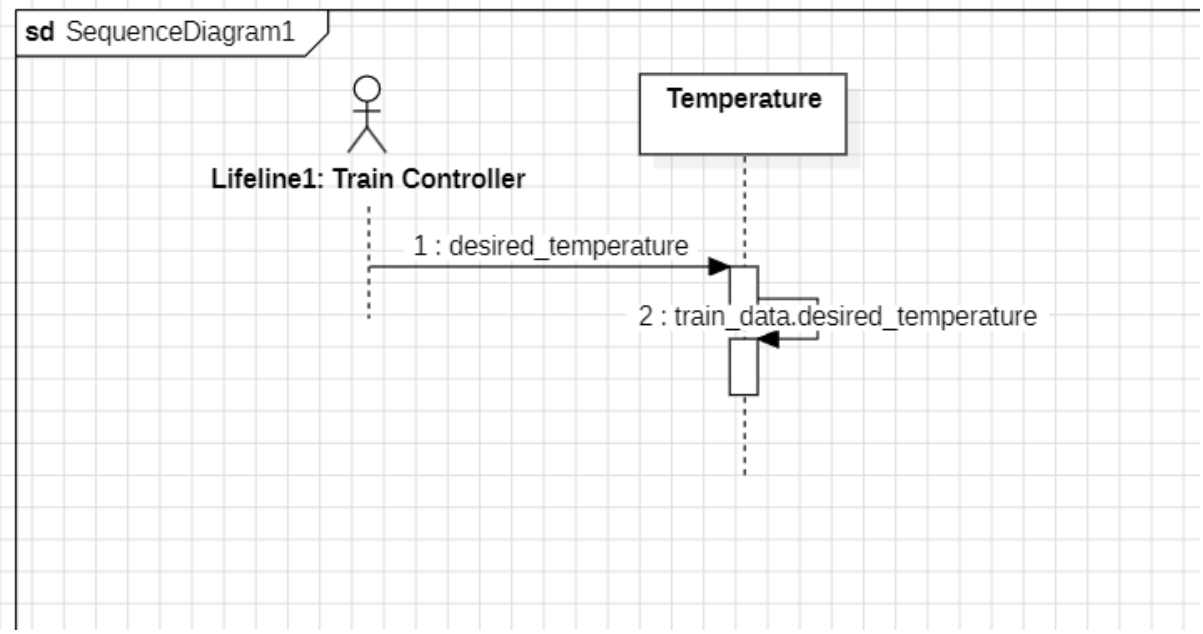
Explanation: Sequence Diagram for receiving light signals. As depicted in the diagram, the Train Controller sends Boolean signals for the train's internal and external lights, individually. The train model then sets the value and changes the light status via the light button color.

5.4.4.6 Broadcast Announcements



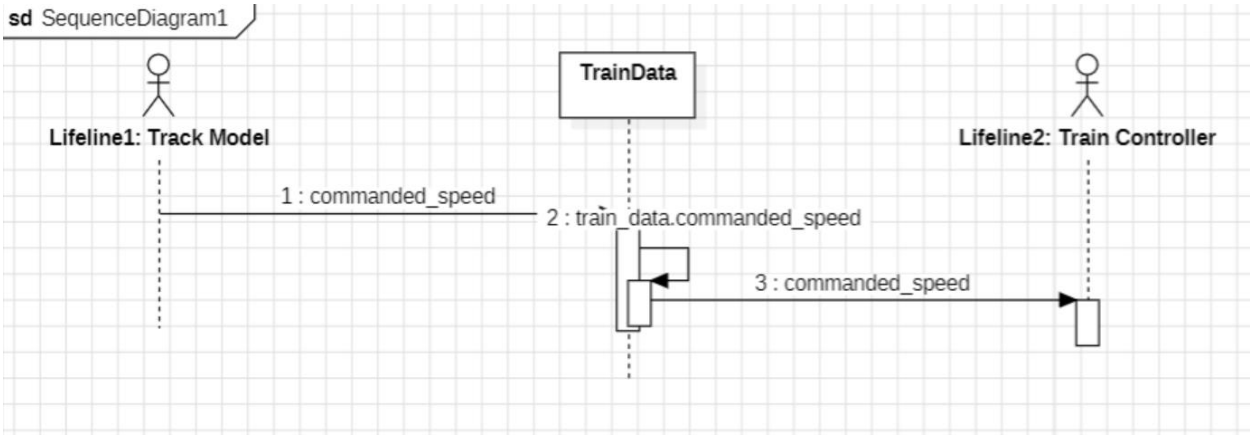
Explanation: Sequence Diagram for broadcasting announcements. The train controller determines the announcement type. This data is given to the train model where is sets the announcement to the specified type and displays it to the passengers.

5.4.4.7 Set Desired Train Temperature



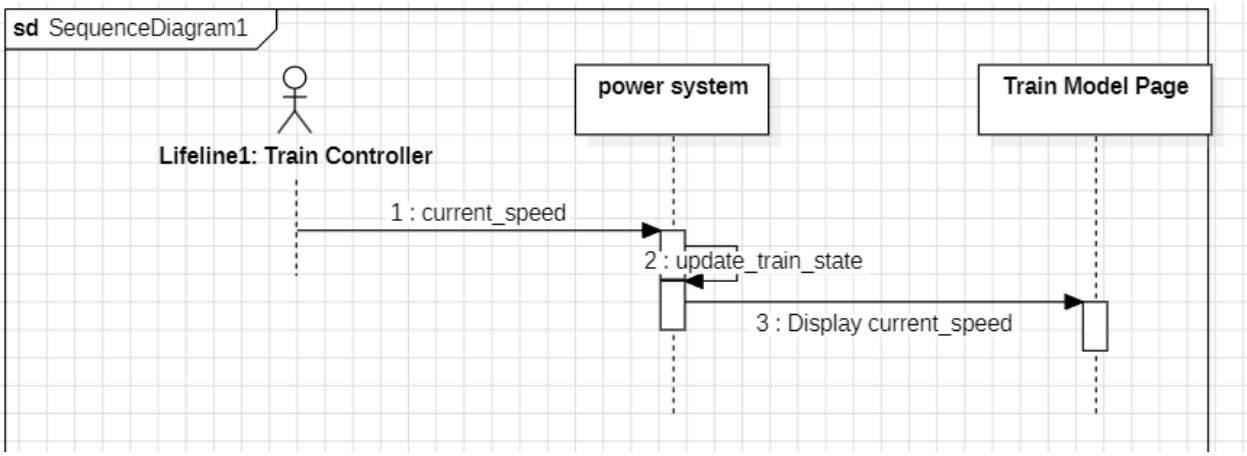
Explanation: Sequence Diagram for setting the desired temperature. As depicted in the diagram, the Train Controller sends a desired temperature for the train's cabin. The temperature class then takes this value and calls a function to set the desired temperature.

5.4.4.8 Transfer Commanded Speed



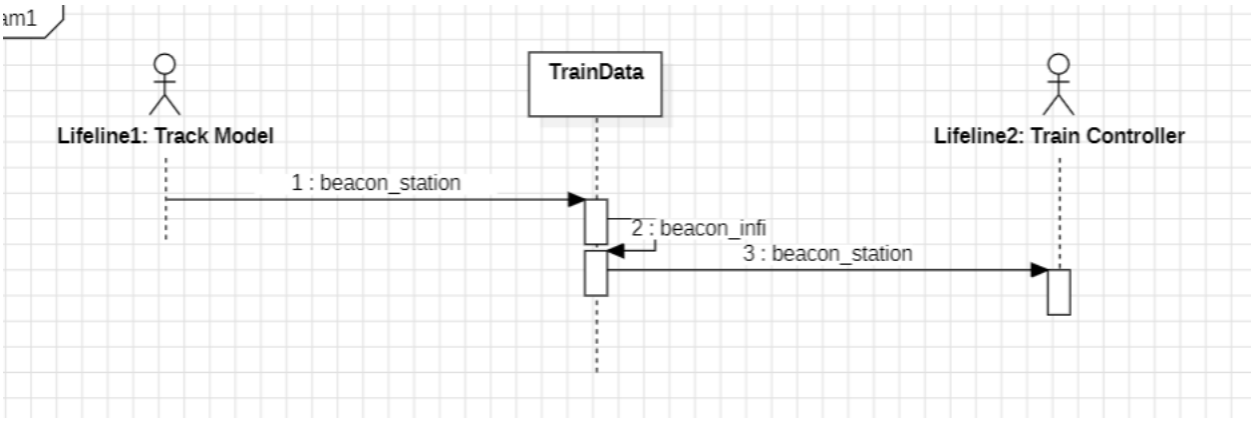
Explanation: Sequence Diagram for transferring commanded speed to the Train Controller. As depicted in the diagram, the Train Controller requests the commanded speed from the Track Model class which then is temporarily stored within the train's data and then relayed to the Train Controller.

5.4.4.9 Transfer Current Velocity



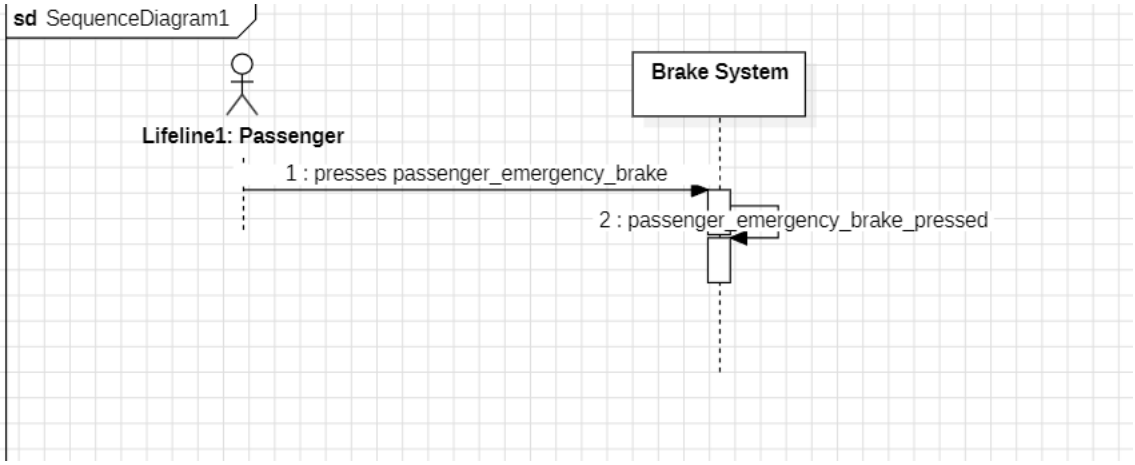
Explanation: Sequence Diagram for transferring current velocity. As depicted in the diagram, the Train Controller sends the current speed to the PowerSystem where a function is called to update the train's state. Then, the updates are reflected in the Train Model UI.

5.4.4.10 Transfer Beacon



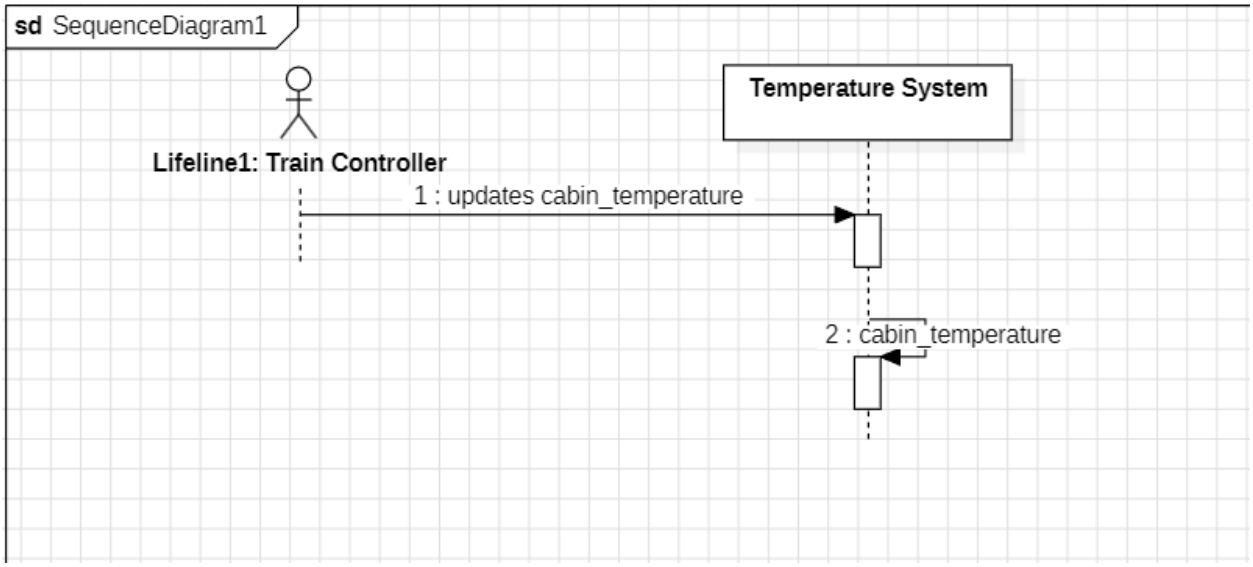
Explanation: Sequence Diagram for transferring beacon information. The Track Model sends the beacon information to the train's data to be temporarily stored and then sent on to the Train Controller.

5.4.4.11 Transfer Passenger Brake Status



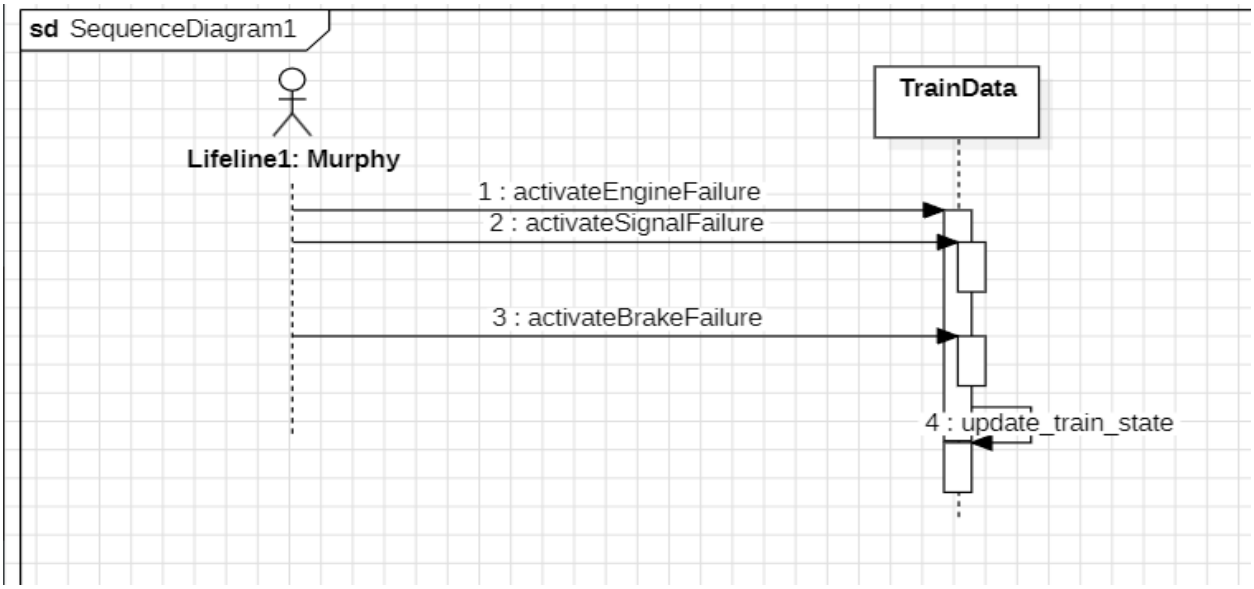
Explanation: Sequence Diagram for sending passenger brake status. As shown in figure above, the Passenger presses the passenger emergency brake. This is handled by the Brake System where a Boolean signal indicates that the Passenger has triggered the emergency brake system.

5.4.4.12 Transfer Actual Train Temperature



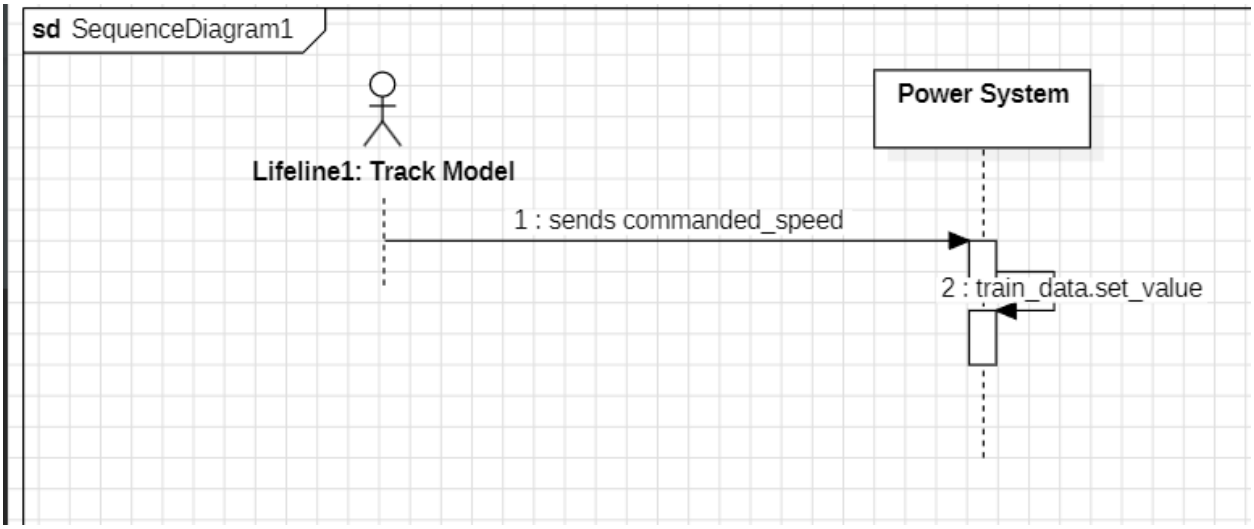
Explanation: Sequence Diagram for transferring the actual train temperature. As depicted the Train Controller sends an updates cabin temperature that the Temperature System then uses to update the temperature in the cabin.

5.4.4.13 Trigger Failure Modes



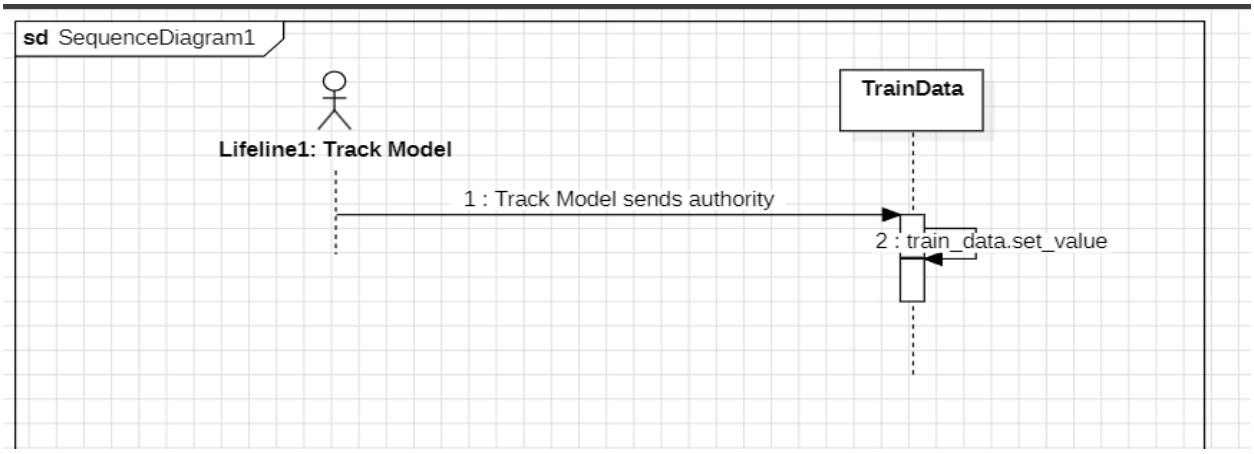
Explanation: Sequence Diagram for triggering failure modes in the Train Model. As shown in the figure above, Murphy may send a signal for each failure mode. These signals are sent to the Train Data class which then calls a function to update the train' status.

5.4.4.14 Receive Commanded Speed



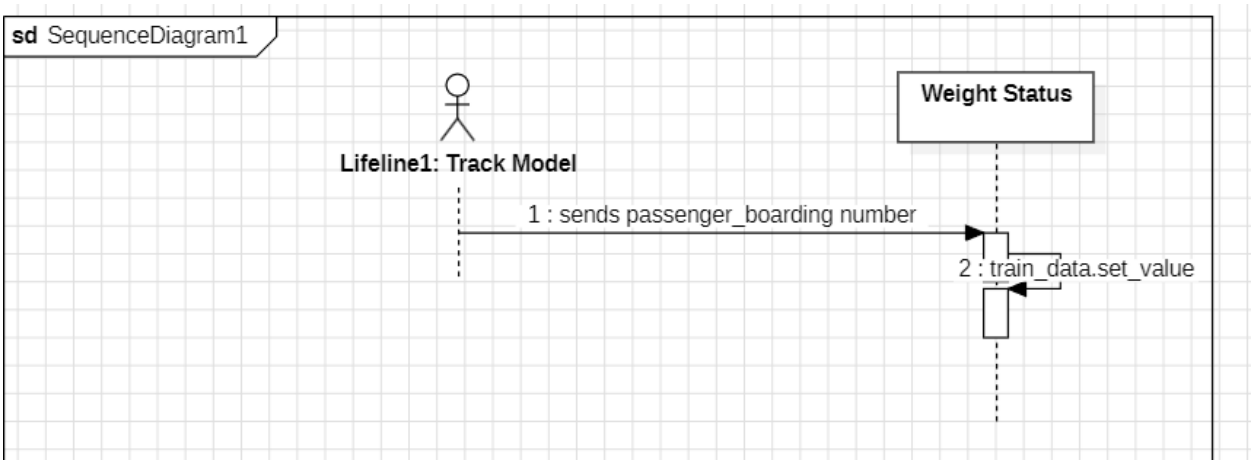
Explanation: Sequence Diagram for receiving the commanded speed from the Track Model. As depicted in the figure the Track Model sends the commanded speed to the Power System where the data value is stored and set.

5.4.4.15 Receive Authority



Explanation: Sequence Diagram for receiving the commanded authority from the Track Model. As depicted in the figure the Track Model sends the commanded authority to the TrainData class where the data value is stored and set.

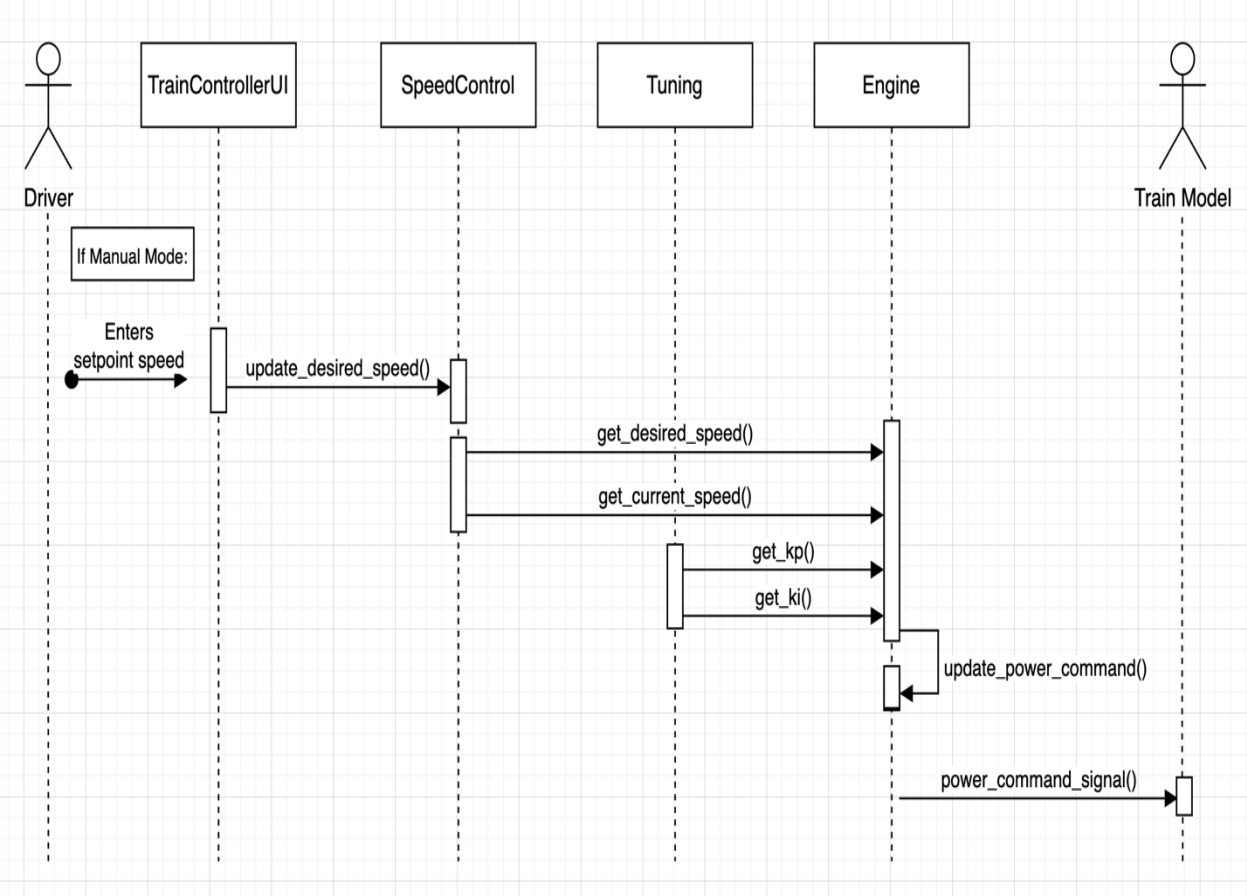
5.4.4.16 Send a Number of Available Seats on the Train



Explanation: Sequence Diagram for sending the number of available seats on the train to the Track Model. As shown in the figure above the Train Model sends a number of open seats after passengers unload the train at a station to the Track Model.

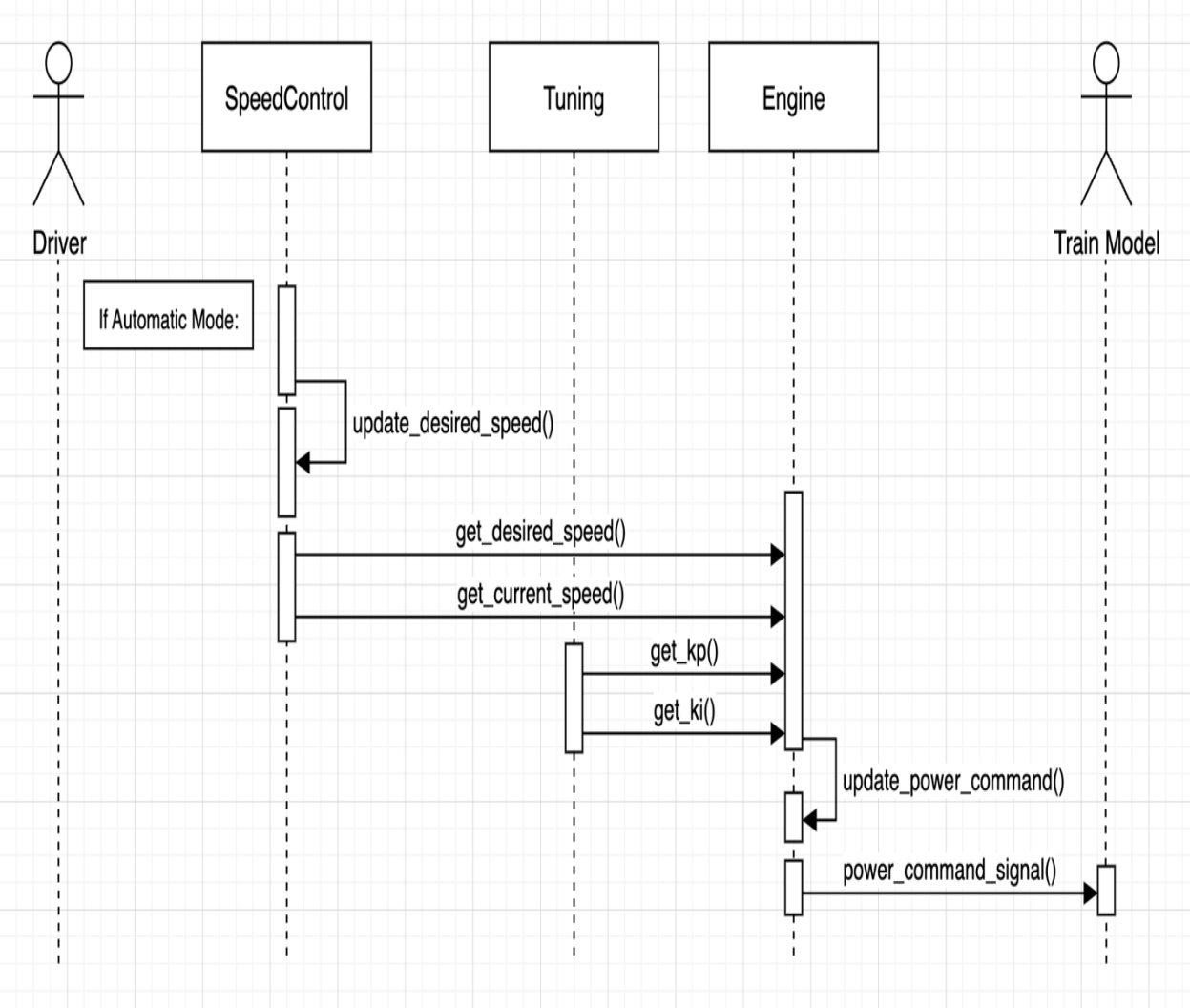
5.4.5 SW Train Controller Sequence Diagrams

5.4.5.1 Set Setpoint Speed & Send Power Command



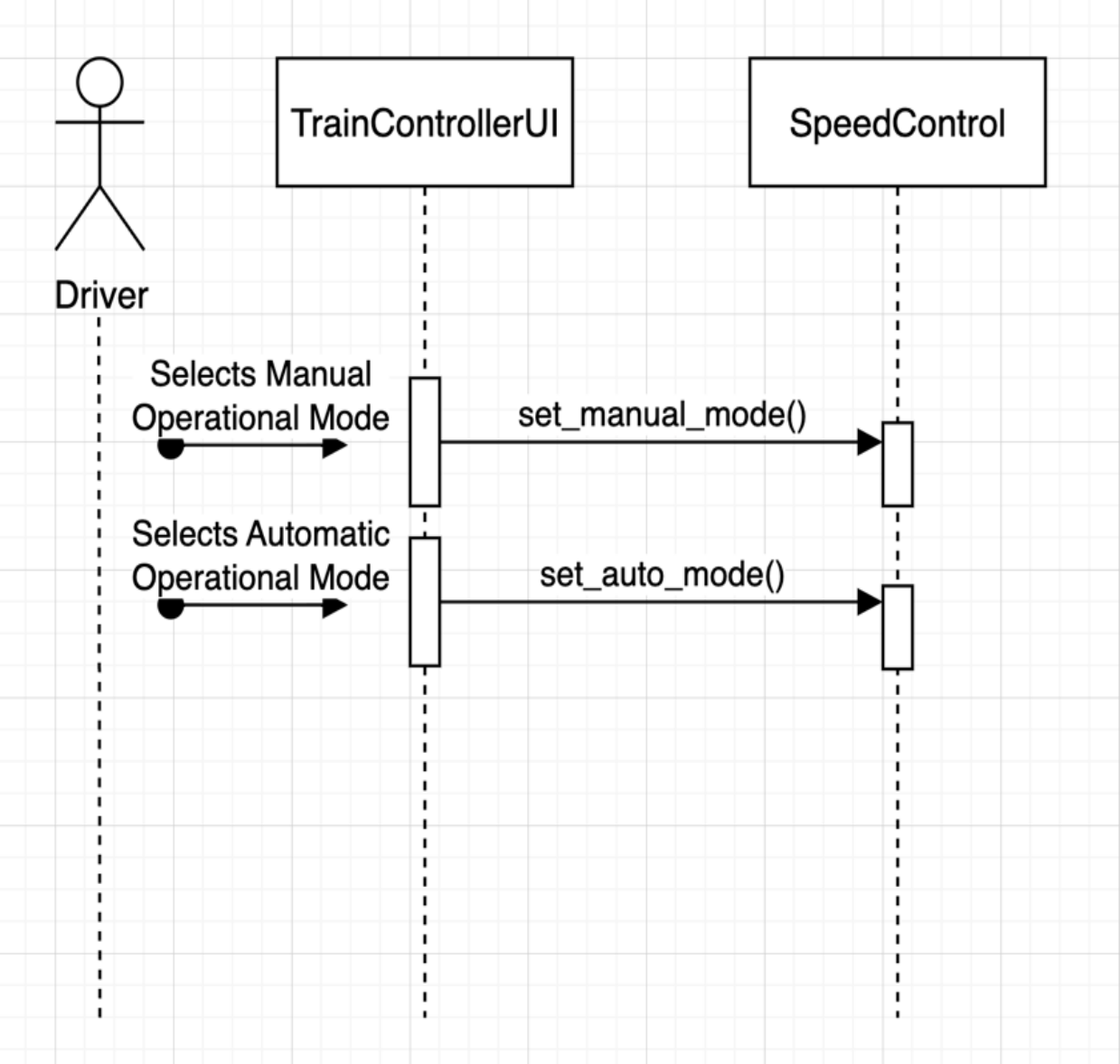
Explanation: Sequence Diagram for the Train Controller, illustrating the process of the train driver setting the setpoint speed of the train and how the current train engine power is calculated from that. This diagram shows interactions between the train driver, TrainControllerUI, SpeedControl, Tuning, Engine, and the Train Model actors and classes as the system takes in the setpoint speed input and uses the desired and current speed value, and the Kp and Ki values to find the current power command of the train and sends it to the Train Model.

5.4.5.2 Automatic Mode Speed



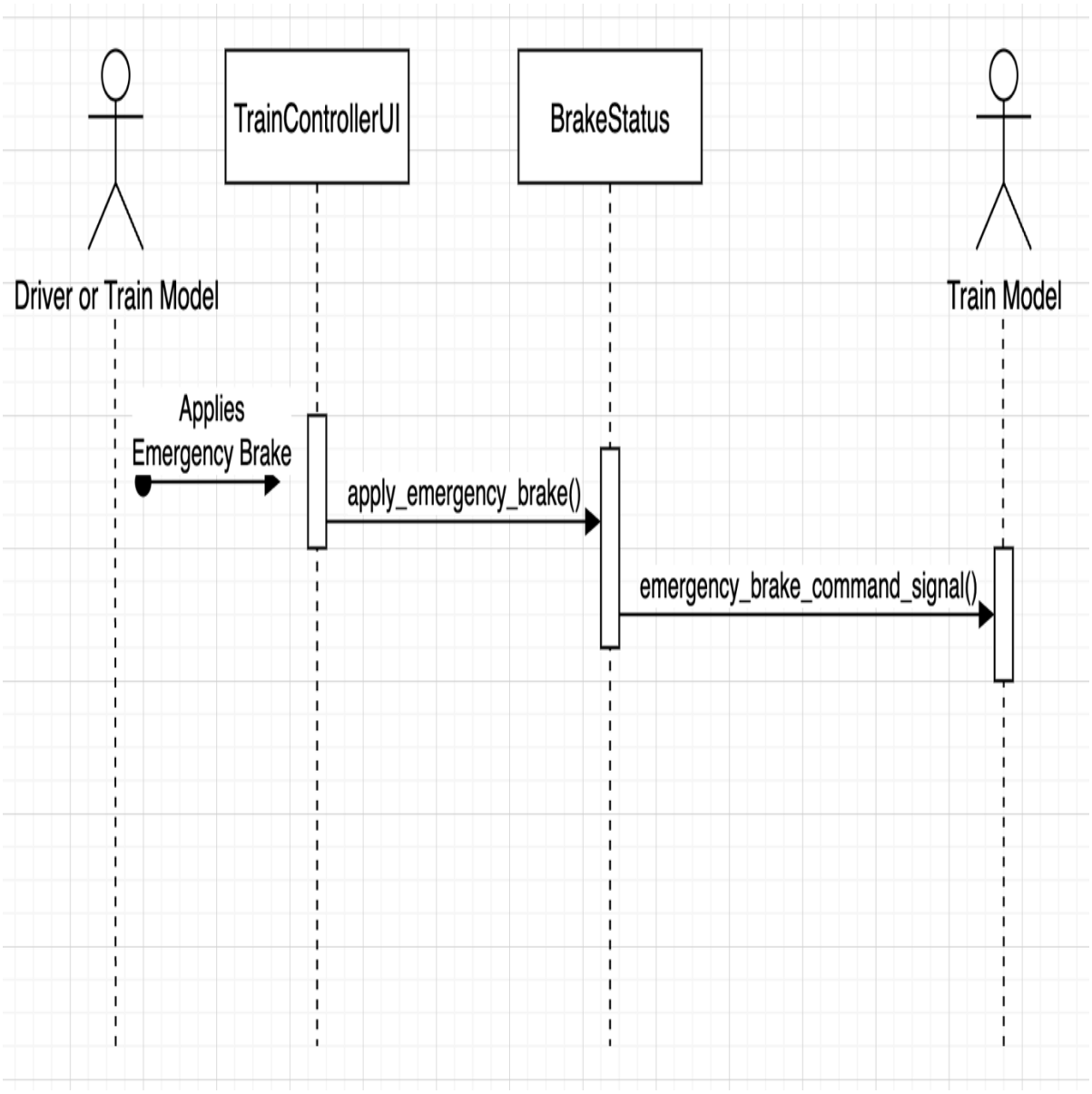
Explanation: Sequence Diagram for the Train Controller, illustrating the process of the train driver setting the setpoint speed of the train and how the current train engine power is calculated from that. This diagram shows interactions between the train driver, TrainControllerUI, SpeedControl, Tuning, Engine, and the Train Model actors and classes as the system takes in the setpoint speed input and uses the desired and current speed value, and the Kp and Ki values to find the current power command of the train and send it to the Train Model.

5.4.5.3 Toggle Operational Mode



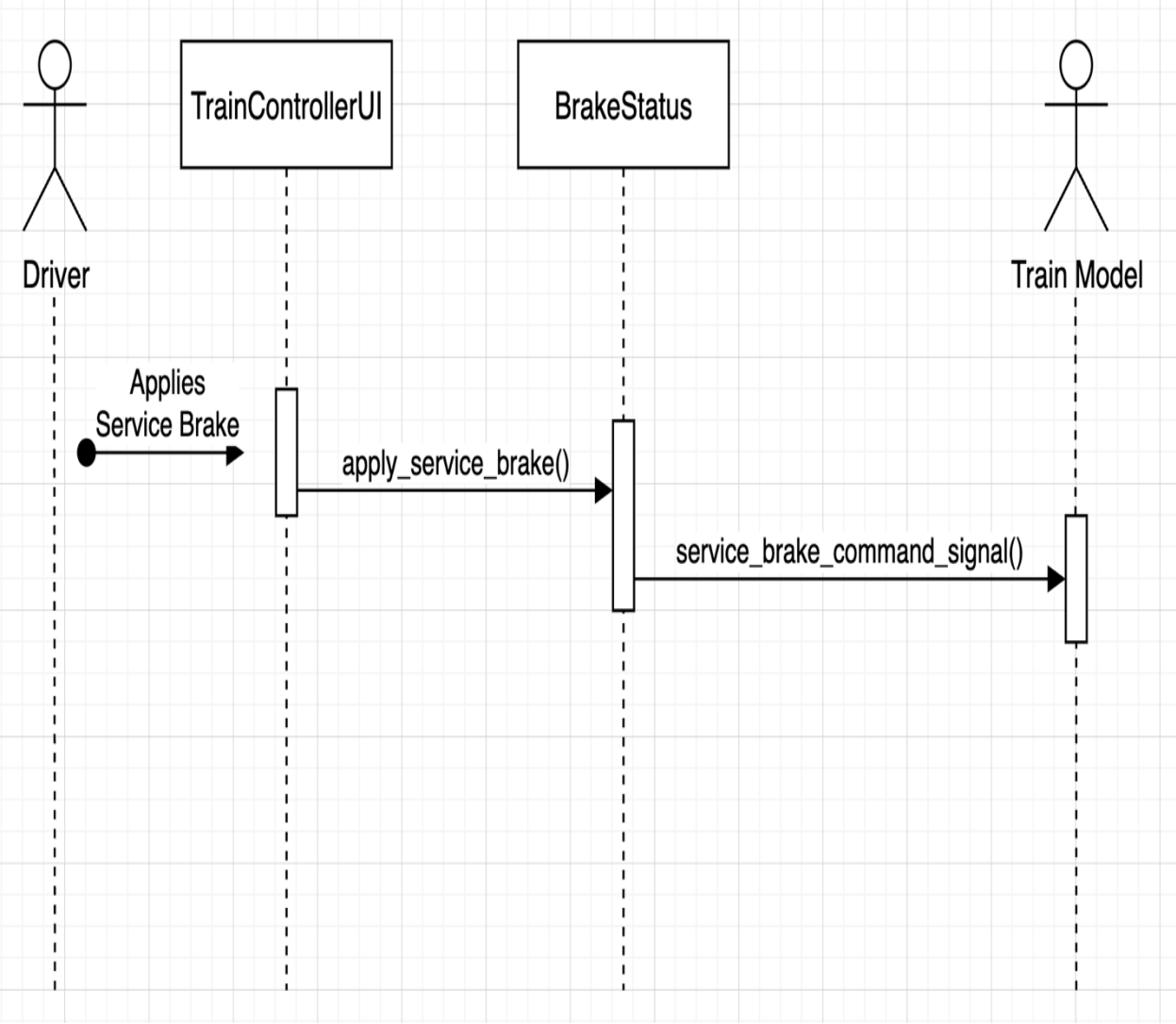
Explanation: Sequence Diagram for the Train Controller, illustrating the process of the train driver toggling the operational mode of the train. This diagram shows interactions between the train driver, TrainControllerUI, and the SpeedControl user and classes as the system recognizes which operational mode to driver wants the train to operate in.

5.4.5.4 Send Emergency Brake Status & Receives Passenger Brake Status



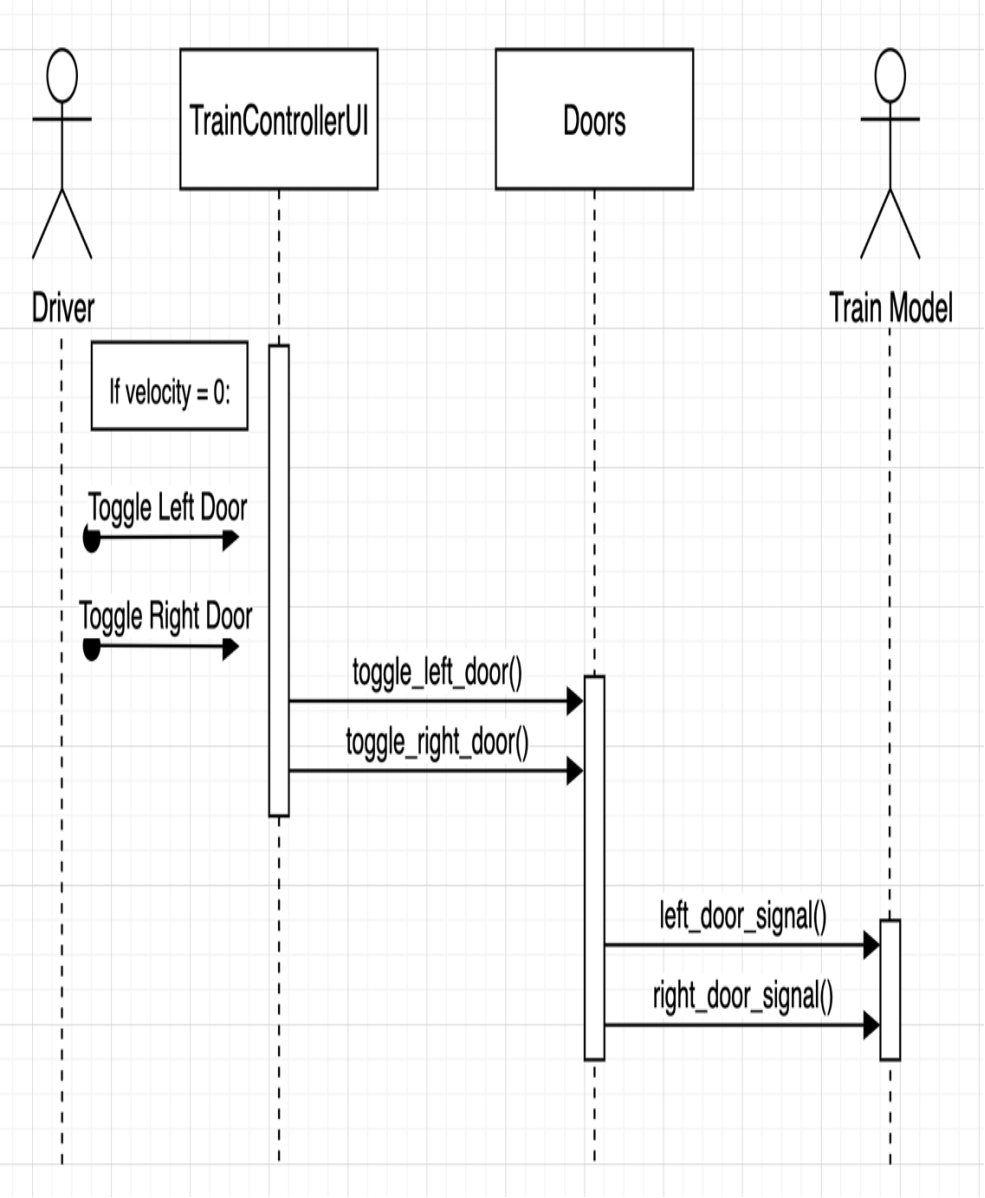
Explanation: Sequence Diagram for the Train Controller, illustrating the process of the train driver and the Train Model for the passenger brake command signal for the emergency brake command to be activated. This diagram shows interactions between the train driver, Train Model, TrainControllerUI, and BrakeStatus actors and classes as the system recognizes which brake has been applied.

5.4.5.5 Send Service Brake Status



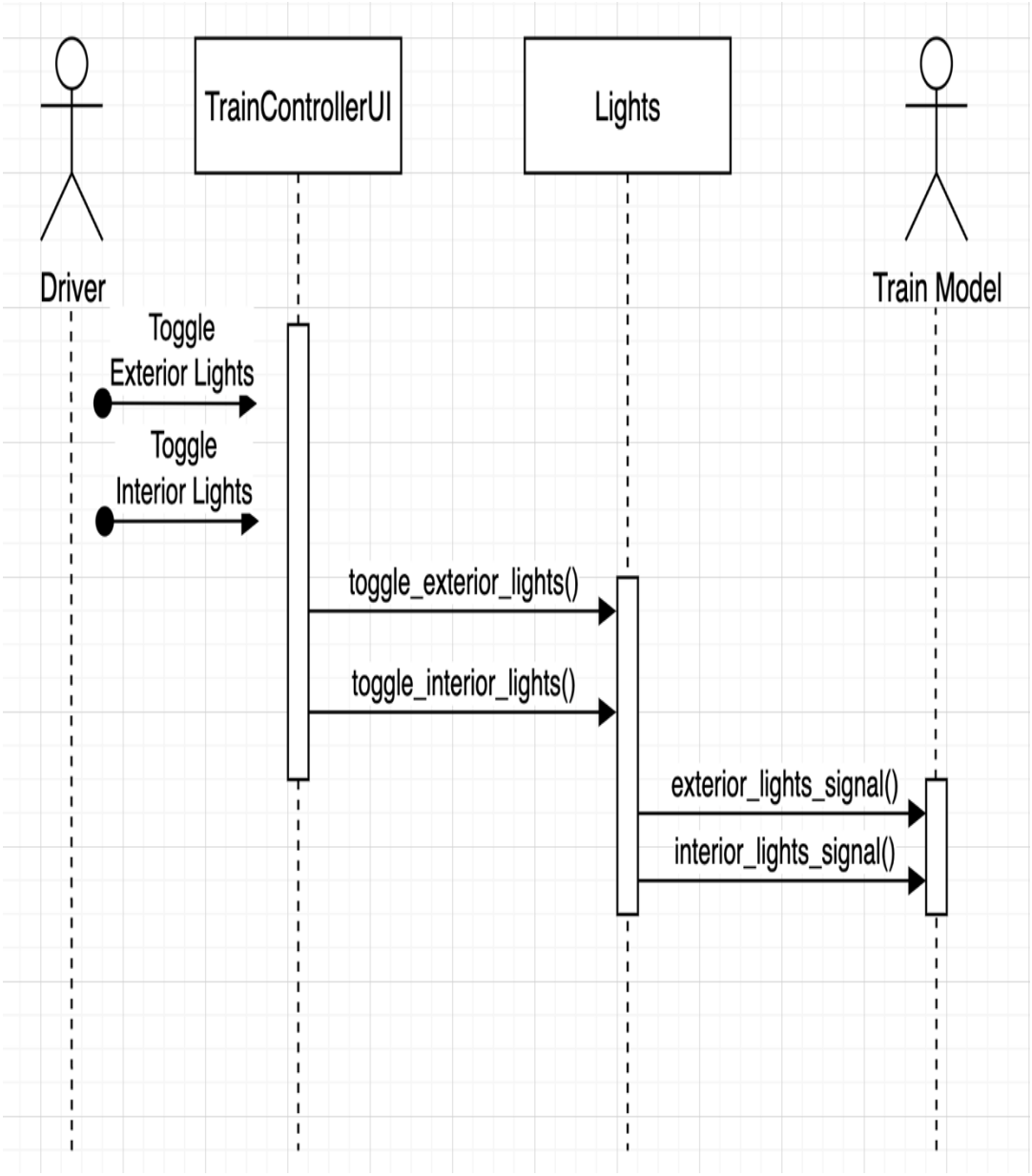
Explanation: Sequence Diagram for the Train Controller, illustrating the process of the train driver applying the service brake command in order for it to be activated. This diagram shows interactions between the train driver, TrainControllerUI, BrakeStatus, and the TrainModel actors and classes as the system recognizes which brake has been applied.

5.4.5.6 Send Door Status



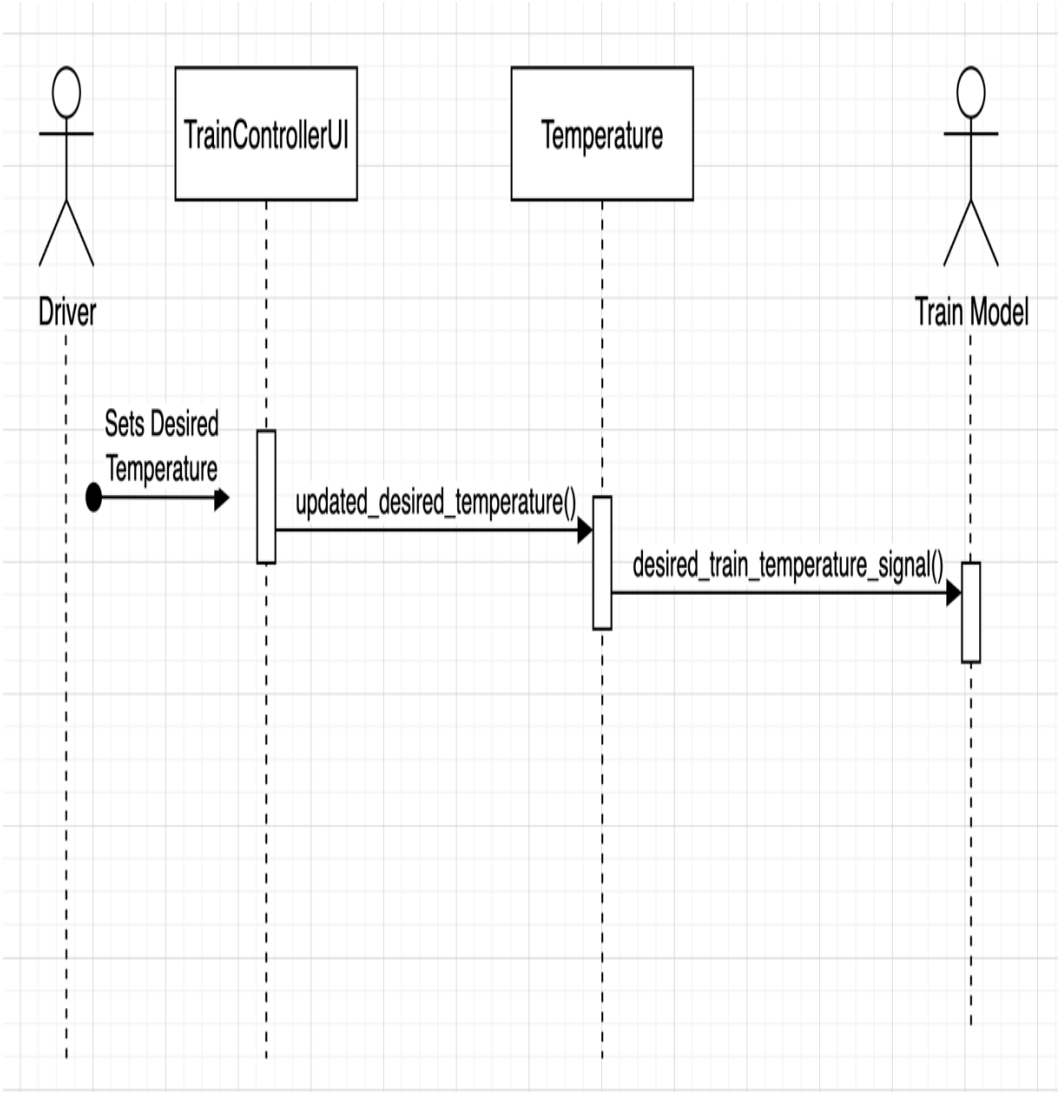
Explanation: Sequence Diagram for the Train Controller, illustrating the process of the train driver changing the status of both train doors. This diagram shows interactions between the train driver, TrainControllerUI, Doors, and the TrainModel actors and classes as the system recognizes which door has been toggled and if it has been opened or closed.

5.4.5.7 Send Exterior/Interior Lights Status



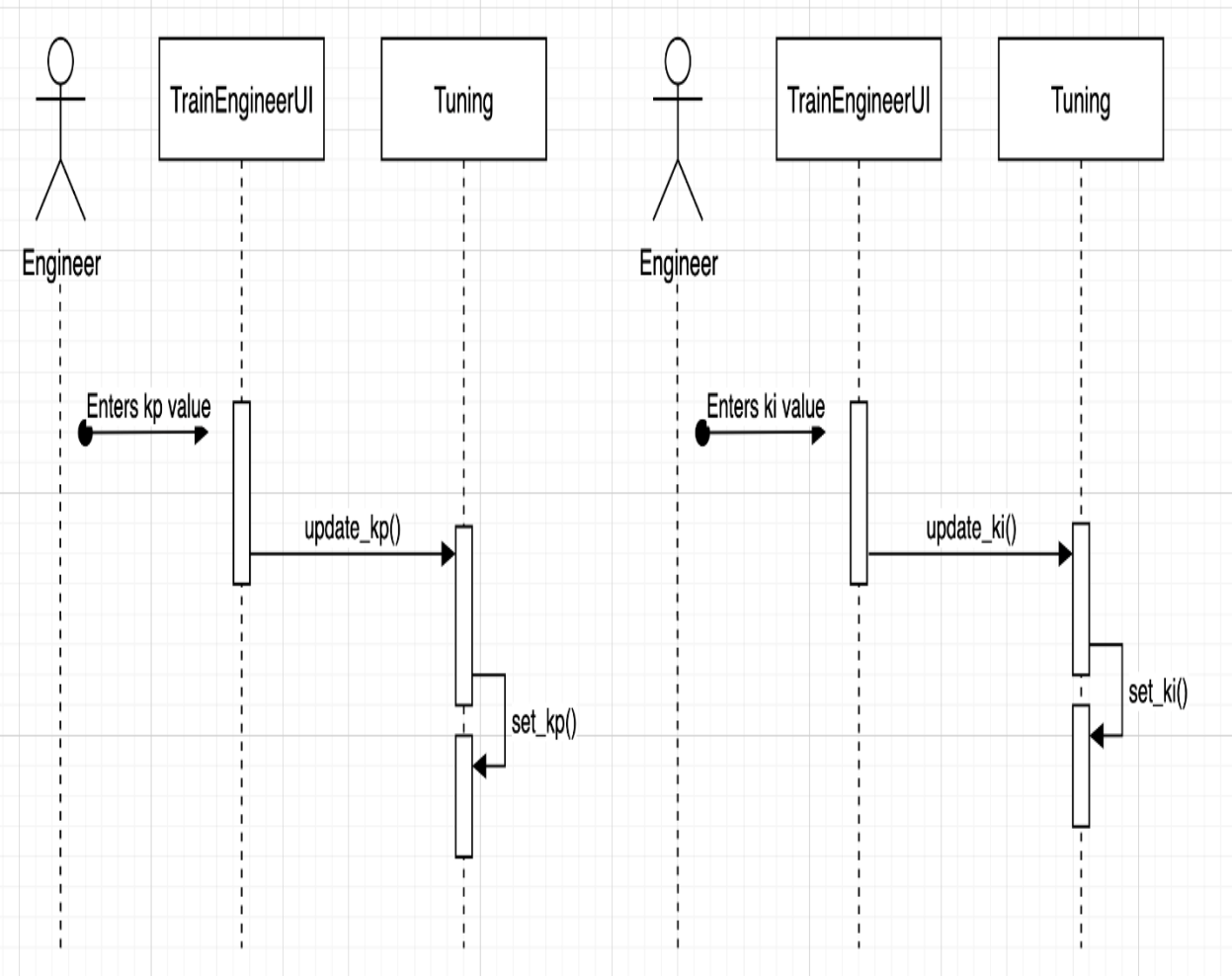
Explanation: Sequence Diagram for the Train Controller, illustrating the process of the train driver changing the status of both train lights. This diagram shows interactions between the train driver, TrainControllerUI, Lights, and the TrainModel actors and classes as the system recognizes which lights have been toggles and if it has been turned on or off.

5.4.5.8 Send Desired Temperature



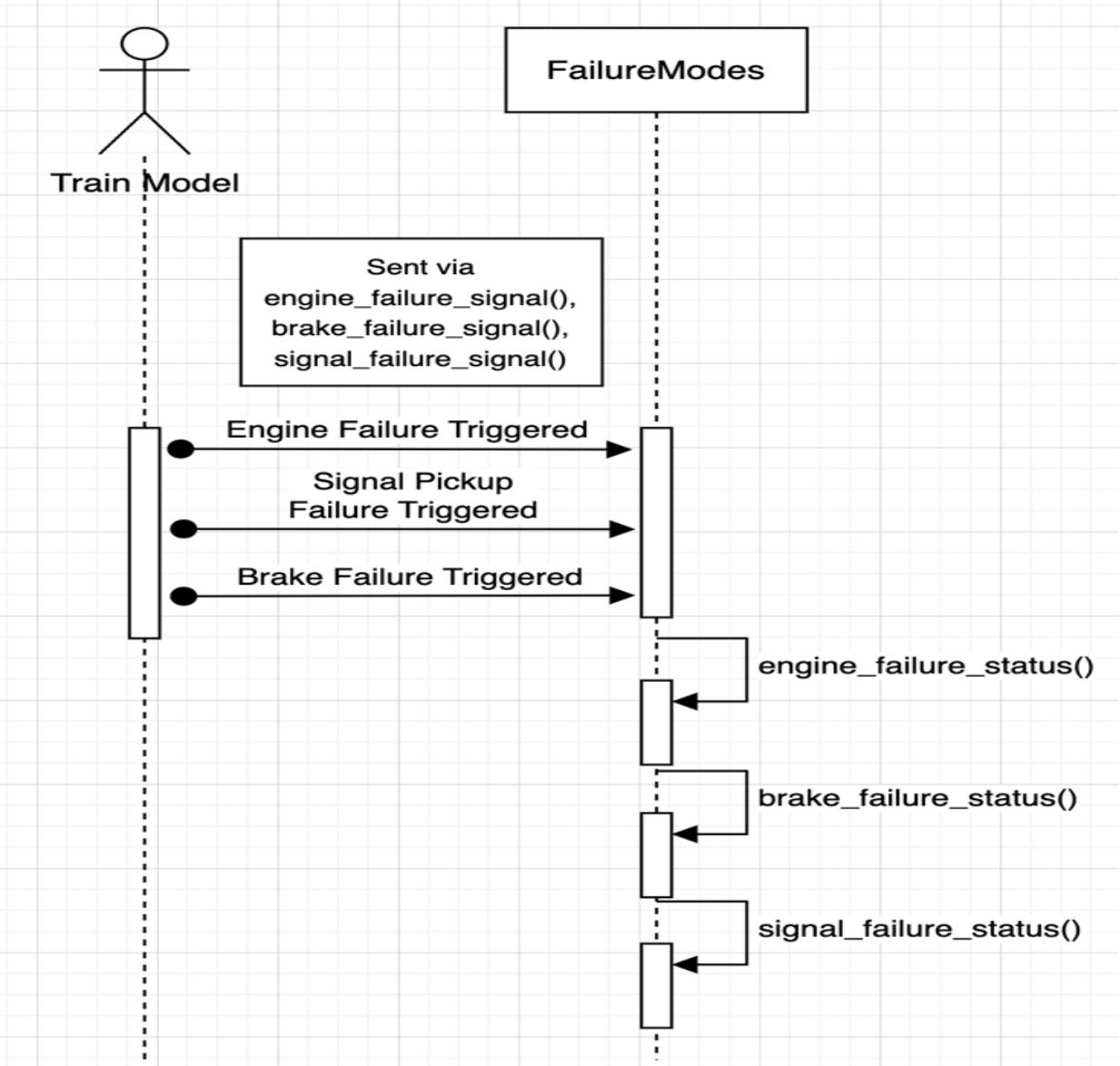
Explanation: Sequence Diagram for the Train Controller, illustrating the process of the train driver setting a desired temperature input in the Driver's user interface. This diagram shows interactions between the train driver, TrainControllerUI, Temperature, and the TrainModel actors and classes as the system recognizes what was the value of the desired temperature that the driver inputted and pass it onto the Train Model sub-system.

5.4.5.9 Set Kp and Ki values



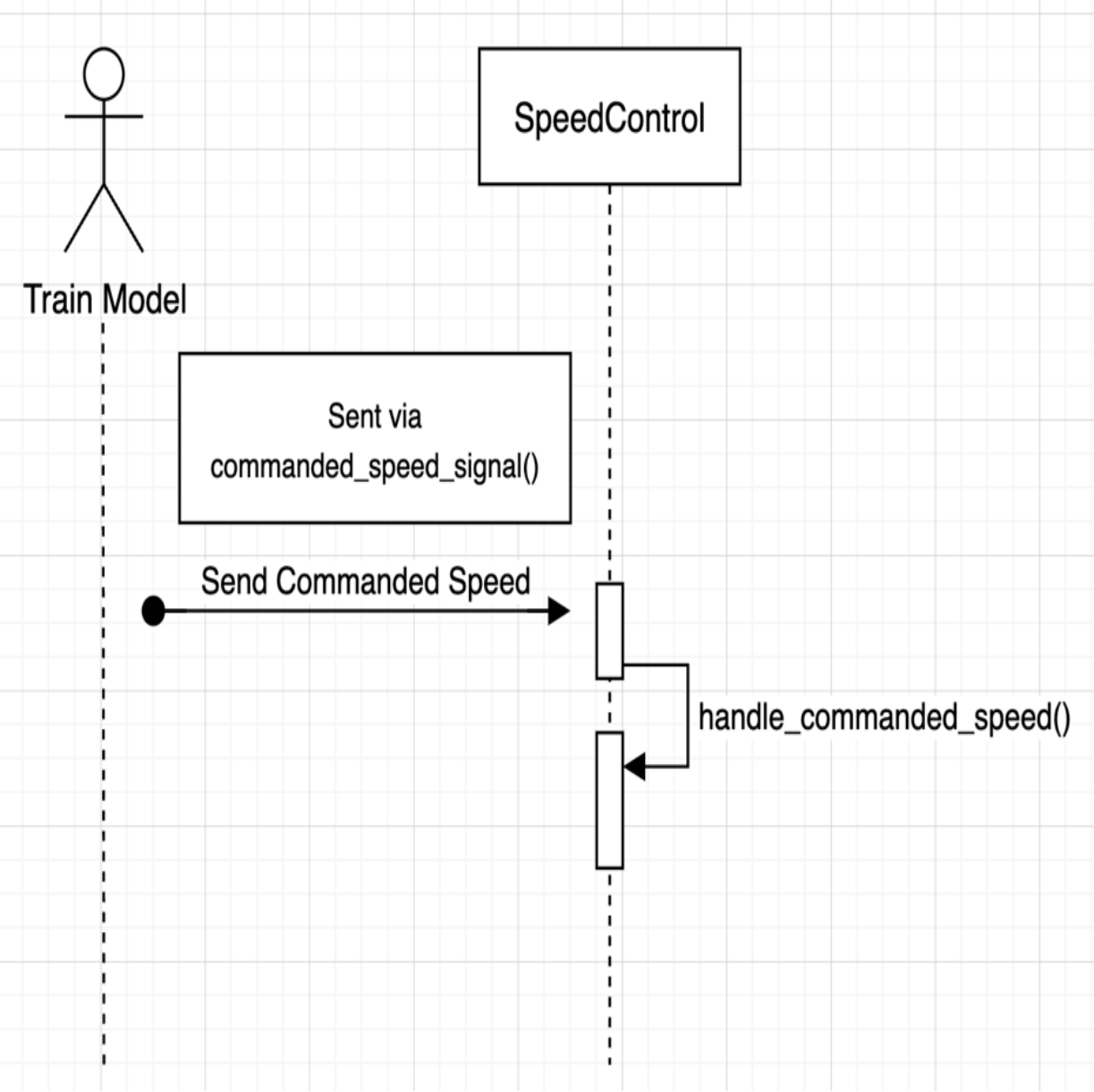
Explanation: Sequence Diagram for the Train Controller, illustrating the process of the train engineer tuning the train system to find stable Kp and Ki values making the train as stable as possible. This diagram shows interactions between the train engineer, Tuning, and TrainControllerUI users and classes as the system recognizes what the train-specific Kp and Ki values are.

5.4.5.10 Receive Train Failures



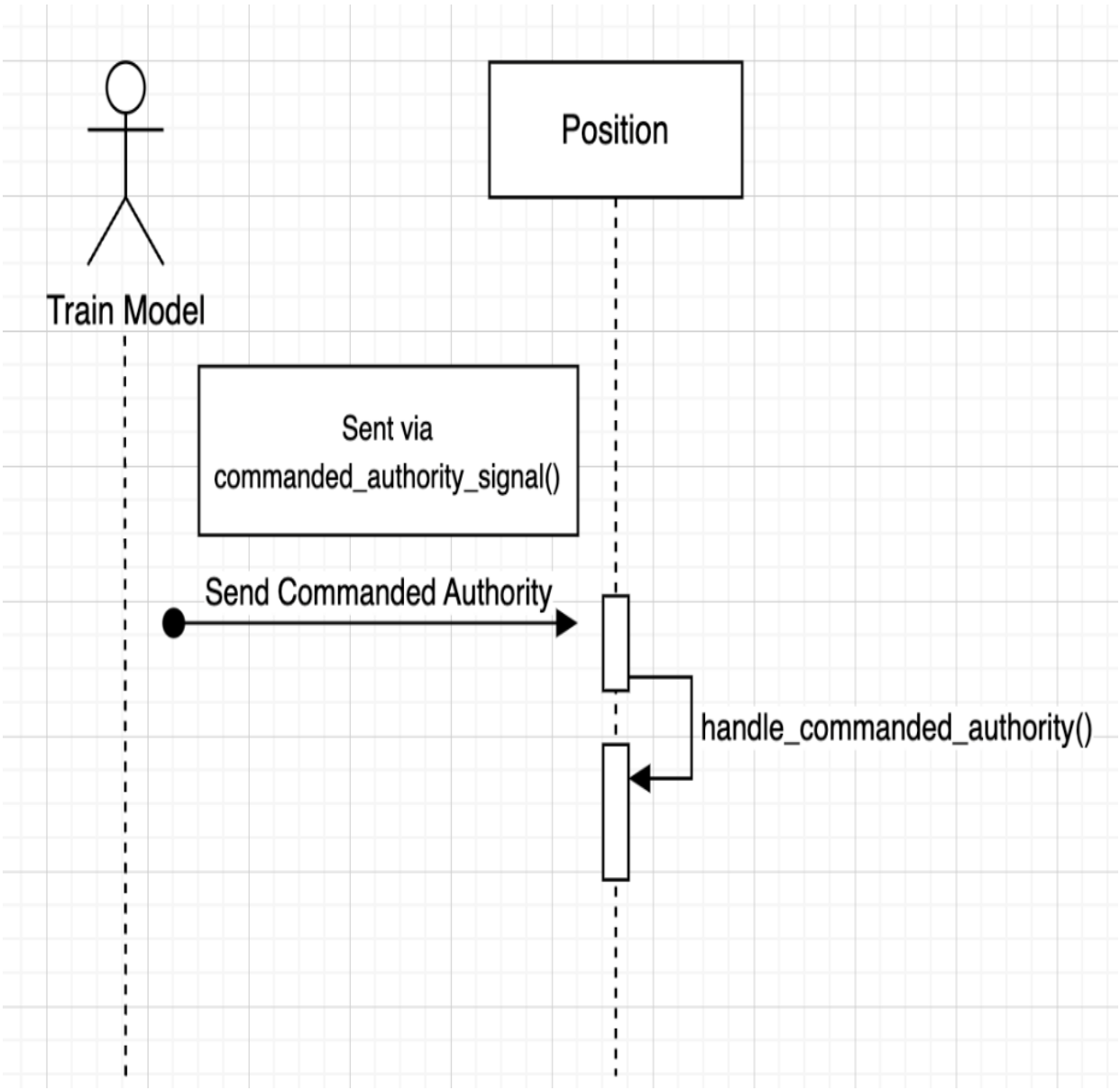
Explanation: Sequence Diagram for the Train Controller, illustrating the process of how the Train Controller recognizes if a failure mode has been triggered and if so which failure. This diagram shows interactions between the Train Model and the FailureModes actor and class as the system recognizes which failure mode has been triggered, if any.

5.4.5.11 Receive Commanded Speed



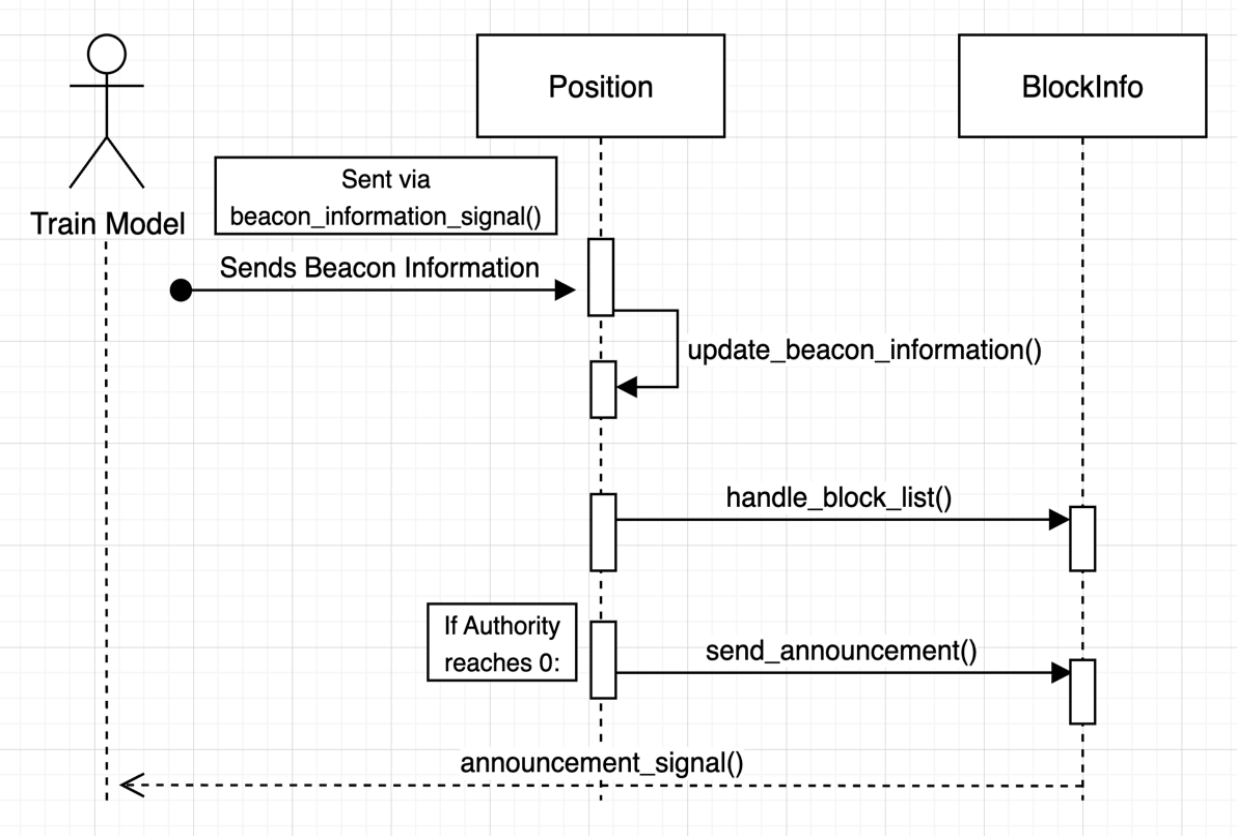
Explanation: Sequence Diagram for the Train Controller, illustrating the process of the train controller receiving the commanded speed from the train model. This diagram shows interactions between the Train Model and the SpeedControl actor and class as the system recognizes the new commanded speed value.

5.4.5.12 Receive Commanded Authority



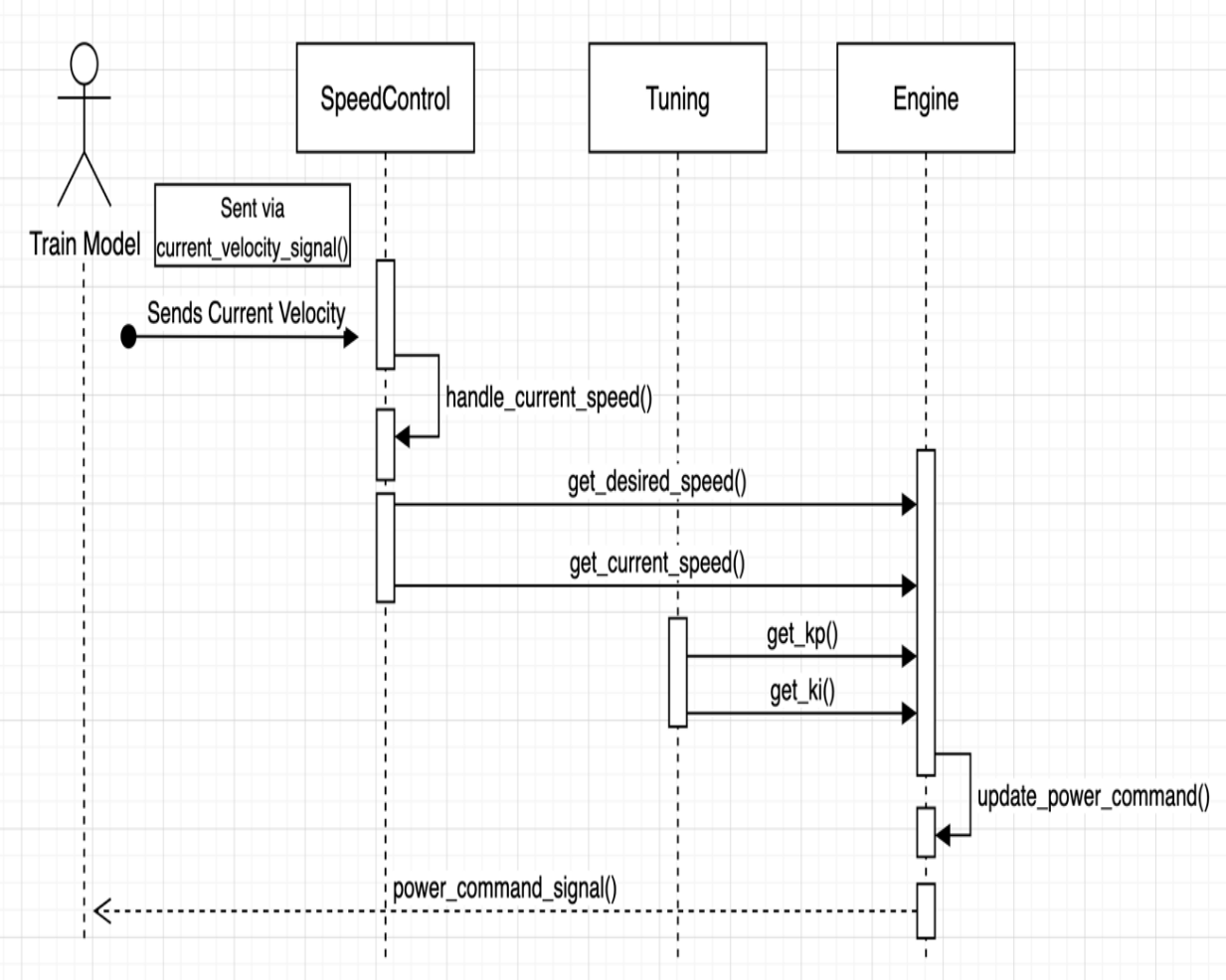
Explanation: Sequence Diagram for the Train Controller, illustrating the process of the train controller receiving the commanded authority from the train model. This diagram shows interactions between the Train Model and the Position actor and class as the system recognizes the new commanded authority value.

5.4.5.13 Send Station Announcement & Receive Beacon Information



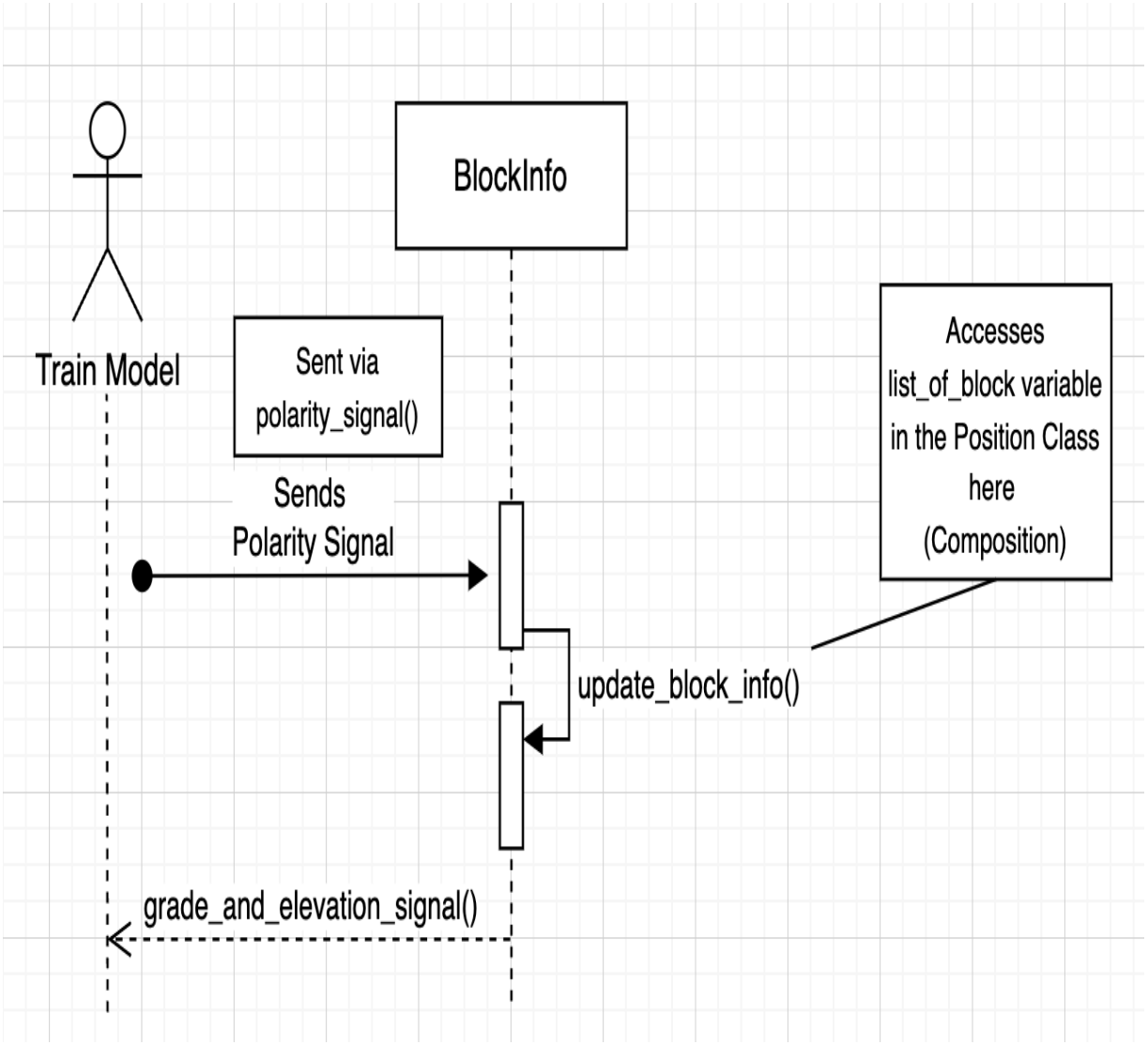
Explanation: Sequence Diagram for the Train Controller, illustrating the process of the train controller receiving the beacon information and handling each of the inputs. The Position class of the Train Controller takes in the beacon information (update_beacon_information()) and handles the two things inside of the beacon, which are the list of blocks between the current station and the next station and the current station name. With the current station name, the BlockInfo class makes an announcement for the passengers and passes it on to the Train Model to display. This diagram shows interactions between the Train Model, Position, and the BlockInfo actor and classes as the system recognizes all of the Beacon Information and make sure that the Train Controller has access to all of that information.

5.4.5.14 Receive Current Velocity & Send Power Command



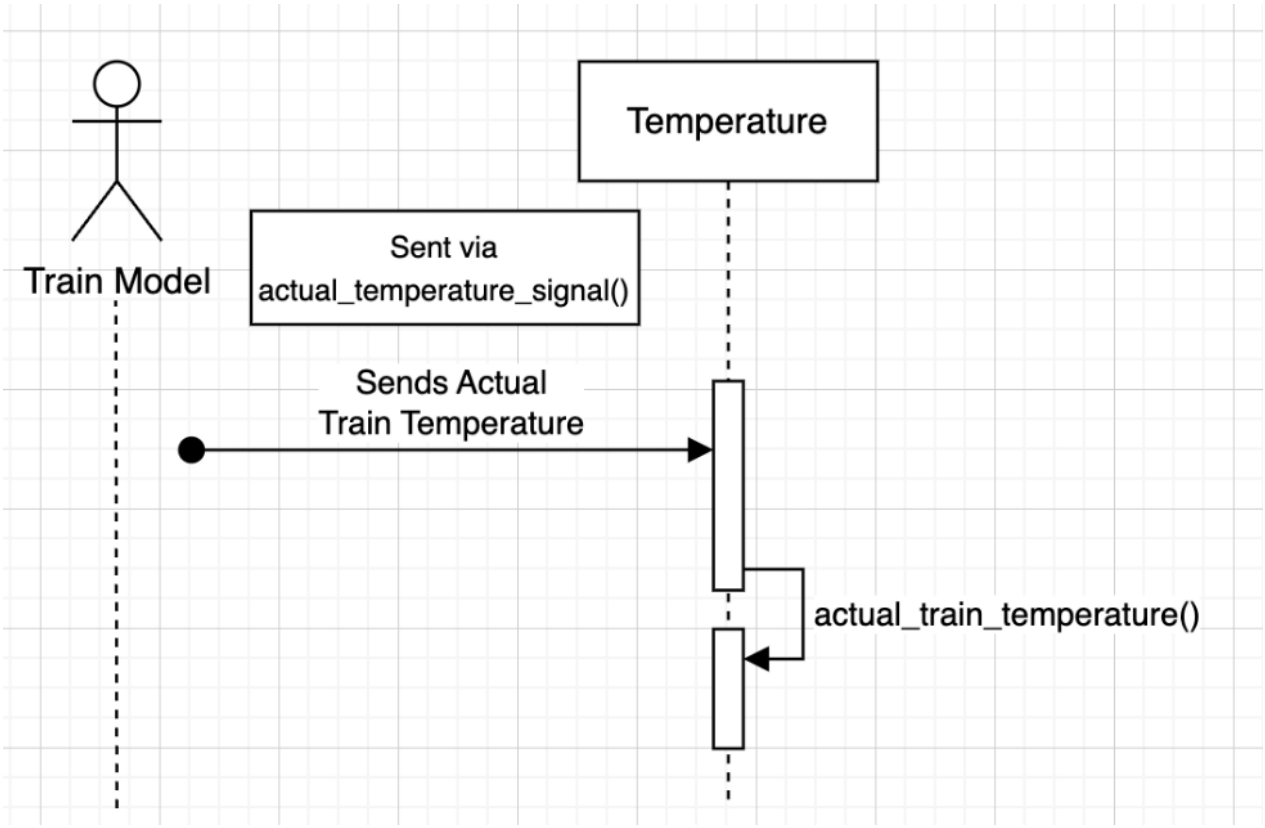
Explanation: Sequence Diagram for the Train Controller, illustrating the process of the Train Controller receiving the current velocity from the Train Model, and outputting the current power command to the Train Model. The Engine class would need the desired speed, current speed, Kp, and Ki values in order to successfully calculate the power command. This diagram shows interactions between the Train Model, SpeedControl, and the Tuning, Engine actor and classes as the system recognizes the value of the current velocity and outputting the corresponding power command value of the train.

5.4.5.15 Send Current Block's Grade and Elevation & Receives Polarity



Explanation: Sequence Diagram for the Train Controller, illustrating the process of the Train Model sending the Train Controller a polarity signal, which indicates that the train has moved onto a new block and the Train Model would need a new grade and elevation value for the current block. This diagram shows interactions between the Train Model and the BlockInfo actor and class as the system recognizes which grade and elevation the train model needs for the current block it is on depending on the polarity signal.

5.4.5.16 Receives Actual Temperature

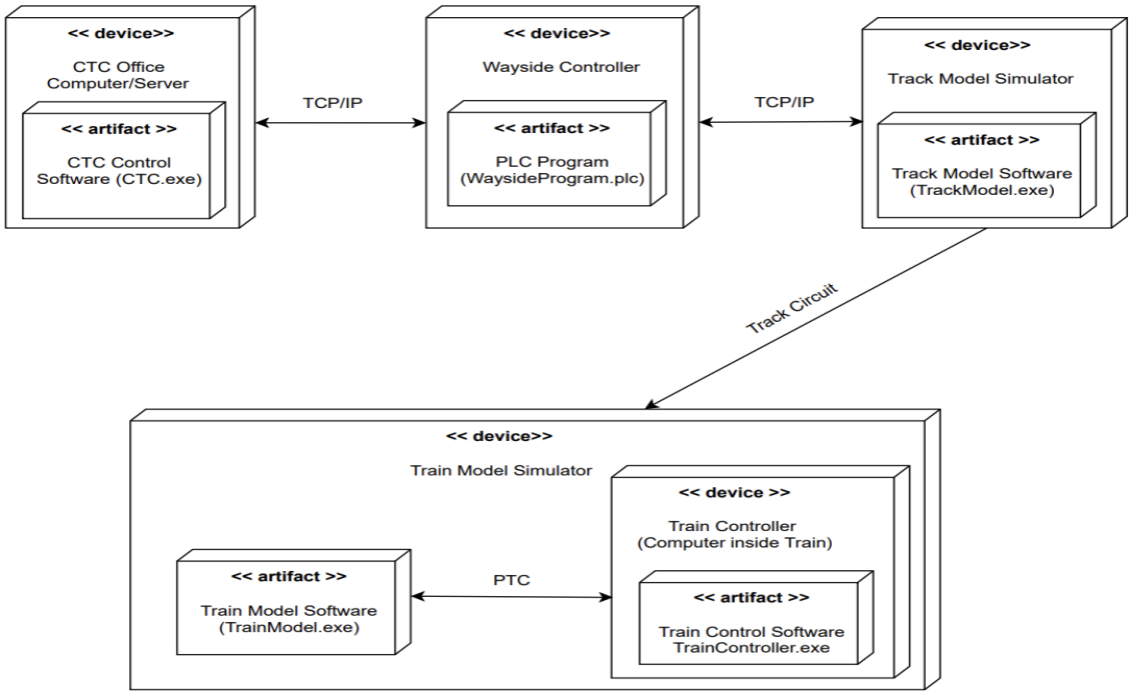


Explanation: Sequence Diagram for the Train Controller, illustrating the process of the Train Model sending the actual train temperature to the Train Controller, which displays it on the Driver's user interface. The Train Model is able to get the actual train temperature with the first-order equation it uses with the desired train temperature to calculate the actual train temperature. This diagram shows interactions between the Train Model, TrainControllerUI, and the Temperature users and classes as the system recognizes the current actual temperature of the train and displays it in the Driver's user interface.

5.5 Composition Viewpoint

The Composition Viewpoint shows how the system is made up of different parts and how these parts work together. This section breaks down the system into its components and explains the role of each part in achieving the overall goals of the system.

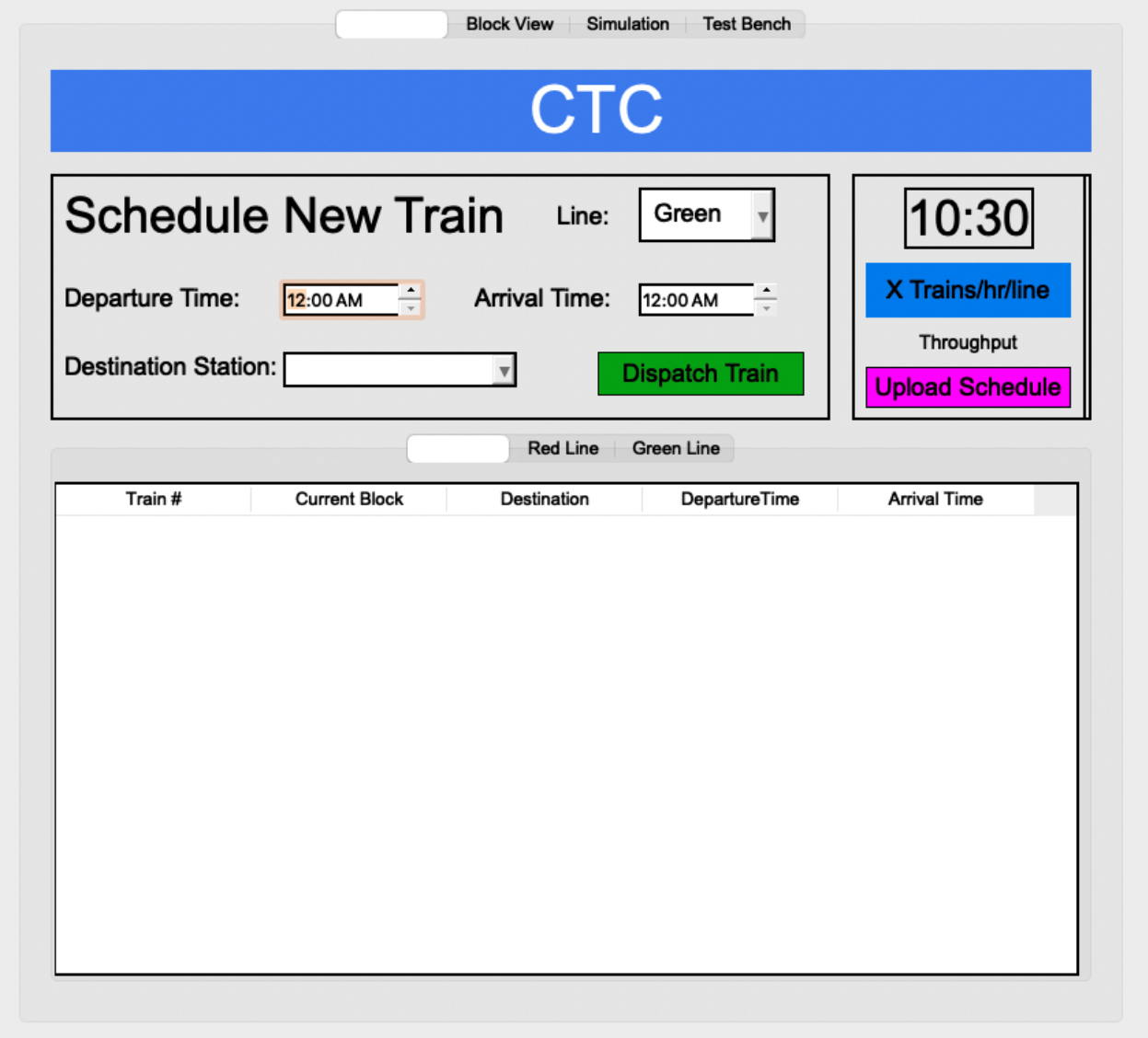
5.5.1 System Deployment Diagram

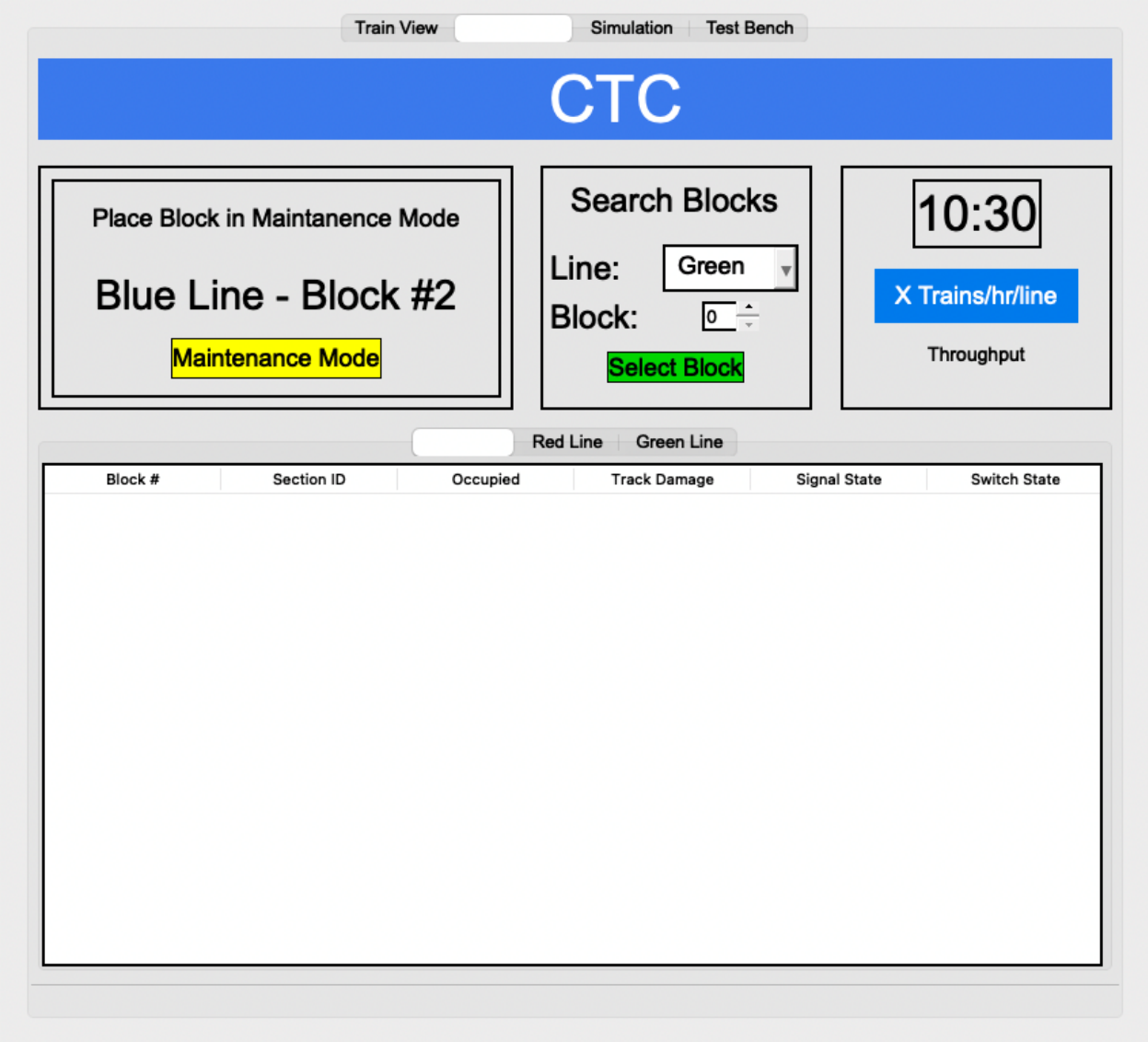


Explanation: Deployment Diagram of the Train Control System, depicting the physical architecture of all modules, including the Centralized Traffic Controller (CTC), Wayside Controller, Track Model, Train Model, and Train Controller. This diagram illustrates how each module is deployed in a real-world environment and shows the communication paths between them, highlighting network connections and data exchange mechanisms necessary for system operation.

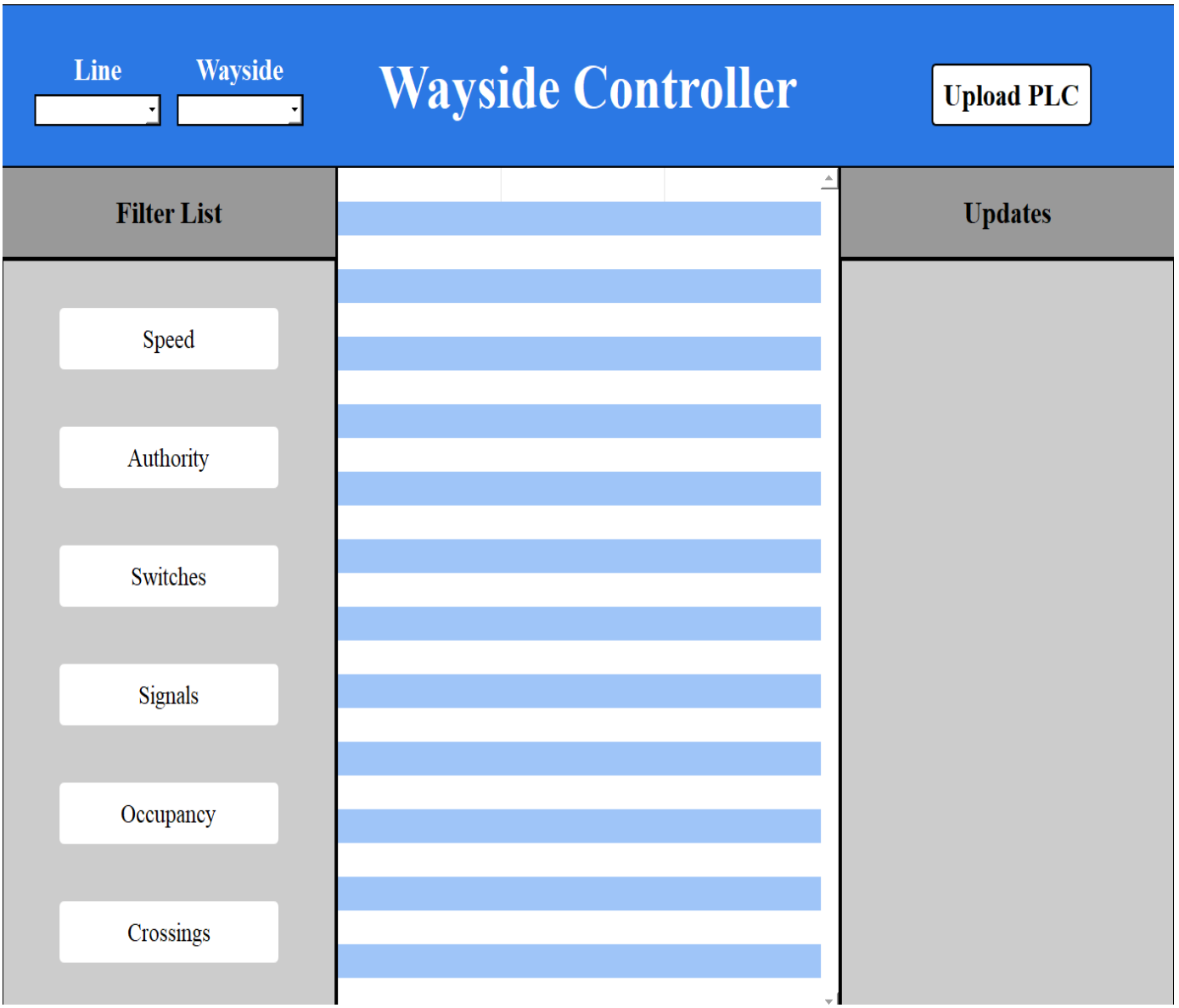
Appendix A: Software User Interfaces

A.1 CTC User Interface

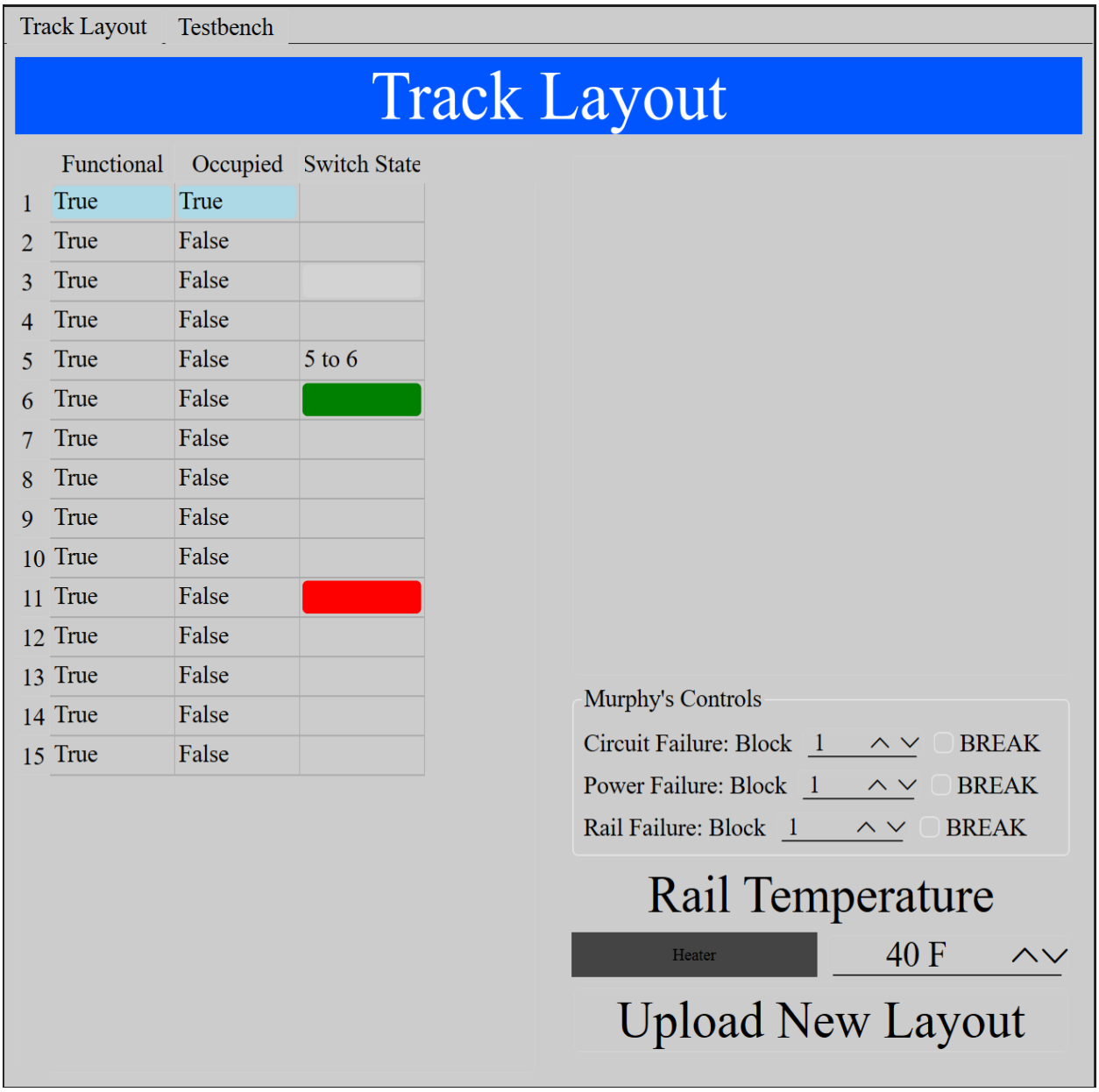



Train View
Simulation
Test Bench
CTC
Place Block in Maintanence Mode
Blue Line - Block #2
Maintenance Mode
Search Blocks
Line:
Green
Block:
0
Select Block
10:30
X Trains/hr/line
Throughput
Red Line
Green Line
Block #
Section ID
Occupied
Track Damage
Signal State
Switch State

A.2 Wayside Controller User Interface

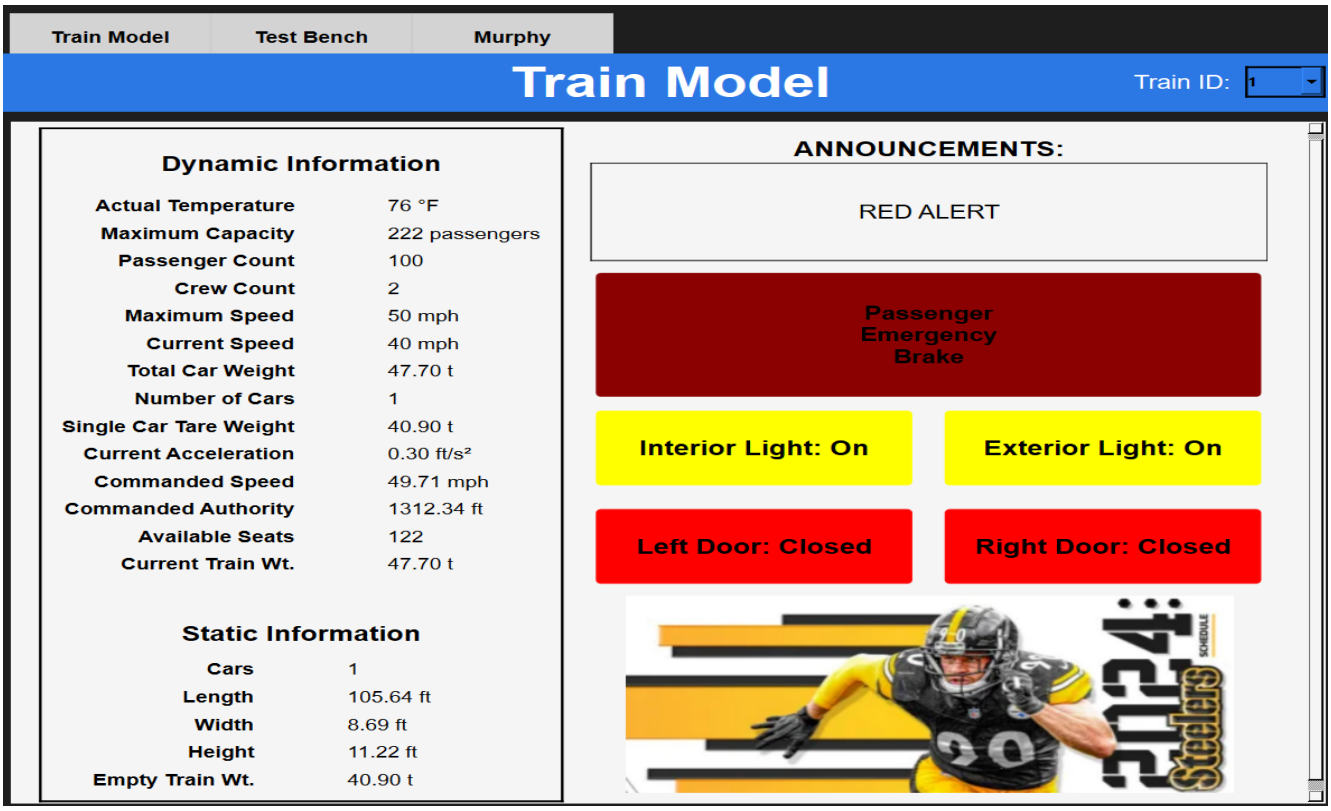


A.3 Track Model User Interface

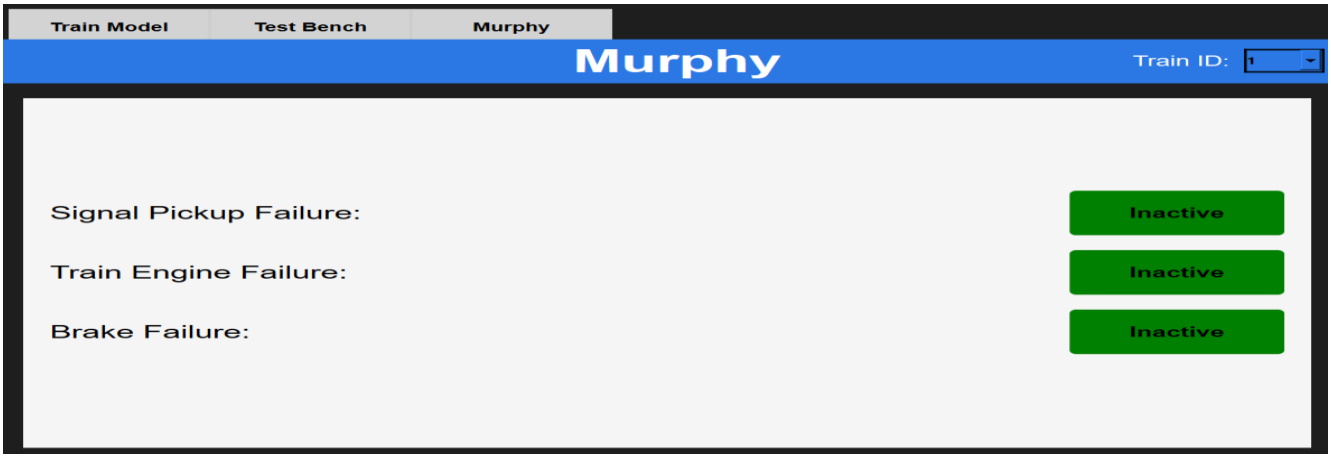


A.4 Train Model User Interface

A.4.1 Main User Interface

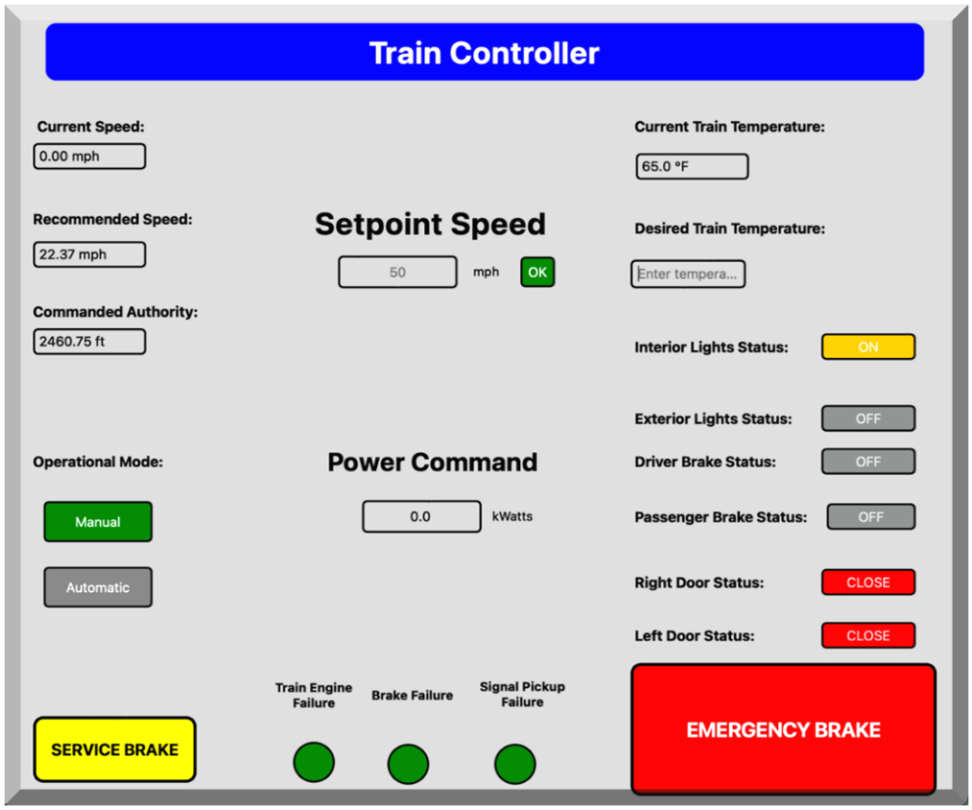


A.4.2 Murphy User Interface



A.5 Train Controller User Interface

A.5.1 Driver User Interface



A.5.2 Engineer User Interface