Software Design Description
for
Pittsburgh Train Automation System

Prepared by Alexander Mcmoil, Cameron Kirsch, Kyle Kessler, Justin Pacella, Yuheng Lin, and Yun Dong
Aurora, Inc.

10/27/2023

Current Version
0.1.0

Name	Date	Reason for Changes	Version
Aurora Team	10/27/2023	Initial Version	0.1.0

Table 1: Version History

Abstract:

The required information content and organization for software design descriptions (SDDs) are described. An SDD is a representation of a software design to be used for communicating design information to its stakeholders. The requirements for the design languages (notations and other representational schemes) to be used for conformant SDDs are specified. This standard is applicable to automated databases and design description languages but can be used for paper documents and other means of descriptions.

Keywords:

design concern, design subject, design view, design viewpoint, diagram, software design, software design description

Contents

Standard for Information Technology – Systems Design – Software Design Description for Pittsburgh Train Automation System

IMPORTANT NOTICE: *This standard is not intended to ensure safety, security, health or environmental protection in all circumstances. The authors of the standard are responsible for determining appropriate safety, security, environmental, and health practices or regulatory requirements*

1 Overview

1.1 Scope

This standard describes software designs and establishes the information content of our Aurora software systems. This SDD is a representation of our software design to be used for recording design information and communicating that design information to key stakeholders. This standard is intended for aiding in the use and understanding of Aurora Train Systems. This standard does not prescribe specific methodologies for design, configuration management, or quality assurance.

1.2 Purpose

This standard specifies requirements on the information contend and organization of Aurora Systems Software. The standard specifies the Use Cases, Class Diagrams, Sequence Diagrams, Software Layout, and Deployment Diagram.

1.3 Intended Audience

This document is intended for users and stakeholders involved in Aurora's Pittsburgh Train Automation System project.

1.4 Definitions

- **design attribute:** An element of a design view that names a characteristic or property of a design entity, design relationship, or design constraint.
- **design element:** An element of a design view that may be of any of the following: design entity, design attribute, or design constraint
- **design rationale:** Information capturing the reasoning of the designer that led to the system s designed, including design options, trade-offs considered, decisions made, and the justifications of the decisions.
- **design attribute:** An element of a design view that names a characteristic or property of a design entity, design relationship, or design constraint.

1.5 Document Conventions

- The terms "we" and "us" refer to the Aurora team's technical stakeholders
- *SDD* is an acronym for Software Design Descriptions

1.6 References

- *Aurora Team's SRS Document*

2 System Architecture & Design

2.1 Identified Stakeholders

Listed below are all of Aurora's technical stakeholders:

- Alexander McMoil - CTC Office module developer
- Justin Pacella - Wayside Controller Software module developer
- Cameron Kirsch - Wayside Controller Hardware module developer
- Kyle Kessler - Track Model module developer
- Yuheng Lin - Train Model module developer
- Yun Dong - Train Controller module developer

2.2 System Architecture

2.2.1 Architecture Diagram & IO Signal Dictionary

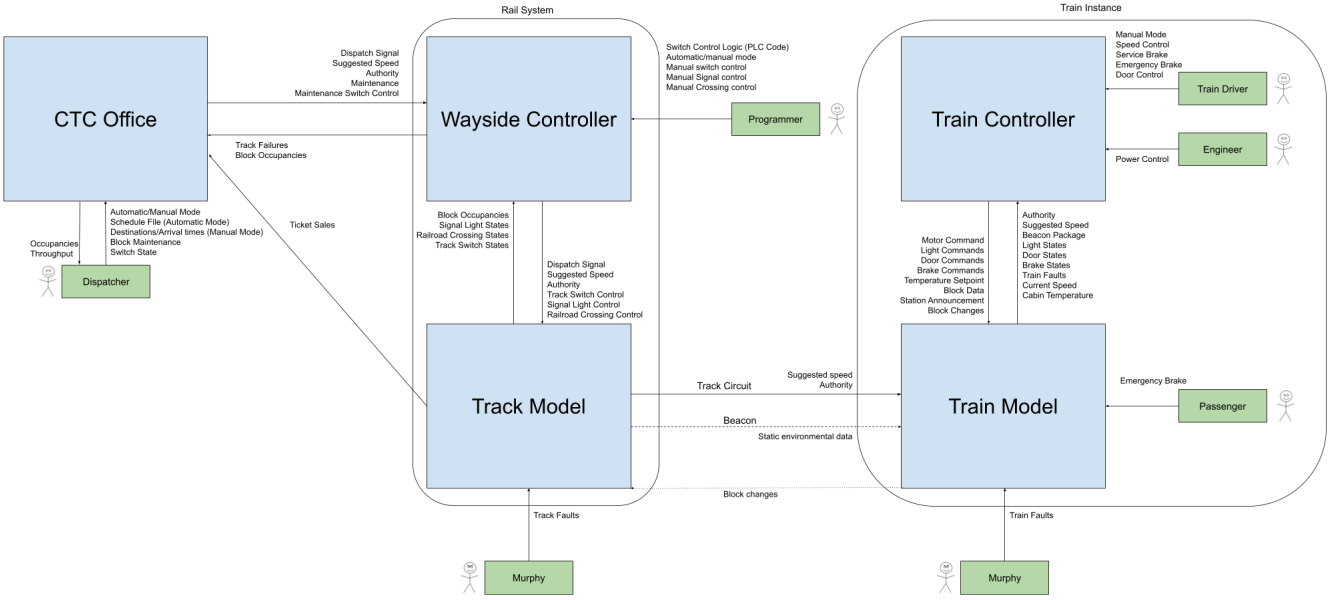

Figure 1: System Architecture Diagram

System IO Dictionary			
Sender	**Receiver**	**Signal**	**Signal Data Items**
CTC Office	Wayside Controller	Dispatch Train	Train ID, Authority, Speed
CTC Office	Wayside Controller	Update Authority	Block ID, Authority
CTC Office	Wayside Controller	Update Speed	Block ID, Suggested Speed
CTC Office	Wayside Controller	Maintenance	Block ID, Maintenance
CTC Office	Wayside Controller	Maintenance Switch	Block ID, State
Wayside Controller	CTC Office	Occupancy	Block ID, Occupied
Wayside Controller	CTC Office	Track Failure	Block ID, Failure Message
Wayside Controller	Track Model	Dispatch Train	Train ID, Authority, Speed
Wayside Controller	Track Model	Send Authority	Block ID, Authority
Wayside Controller	Track Model	Send Suggested Speed	Block ID, Speed
Wayside Controller	Track Model	Set Switch	Block ID, State
Wayside Controller	Track Model	Set Signal	Block ID, State
Wayside Controller	Track Model	Set Crossing	Block ID, State
Track Model	CTC Office	Ticket Sales	Station Name, Ticket Sales
Track Model	Wayside Controller	Occupancy	Block ID, Occupied
Track Model	Wayside Controller	Switch Handshake	Block ID, State
Track Model	Wayside Controller	Signal Handshake	Block ID, State
Track Model	Wayside Controller	Crossing Handshake	Block ID, State
Track Model	Train Model	Track Circuit	Authority, Suggested Speed
Track Model	Train Model	Beacon Package	Station ID, Station Side Exit, Block Lengths, Speed Limits, Block Grades, Underground
Track Model	Train Model	Passenger Info	Ticket Sales
Train Model	Track Model	Block Occupancy	Block Change
Train Model	Train Controller	Authority	Authority
Train Model	Train Controller	Speed	Suggested Speed
Train Model	Train Controller	Beacon Package	Station ID, Station Side, Block Lengths, Speed Limits, Block Grades, Underground/Tunnel
Train Model	Train Controller	Train Light	Exterior, Interior Lights
Train Model	Train Controller	Train Doors	Right, Left Doors
Train Model	Train Controller	Train Brakes	Service Brake, Emergency Brake
Train Model	Train Controller	Train Failure	Engine, Brake, Signal Pickup Failure
Train Model	Train Controller	Current Speed	Current Speed, Power Usage
Train Model	Train Controller	Temperature	Current Cabin Temperature
Train Controller	Train Model	Motor Command	Commanded Speed, Commanded Power
Train Controller	Train Model	Light Command	Exterior, Interior
Train Controller	Train Model	Door Command	Left, Right
Train Controller	Train Model	Brake Command	Service Brake, Emergency Brake
Train Controller	Train Model	Temperature	Cabin Temperature Set-point
Train Controller	Train Model	Environment Data	Block Grade, Underground/Tunnel
Train Controller	Train Model	Announcement	Station Name
Train Controller	Train Model	Block Occupancy	Block Change

2.2.2 Train Routing

The CTC has 2 methods of sending out a train: automatic mode through a schedule file and manual mode through the manual interface. While both modes dispatch trains similarly, there are notable differences.

In automatic mode, the dispatcher will upload a schedule file in the form of a *xlsx file format. The file will be formated as such: first row is labels for line, destination, and time to station with dwell time; for every row after, line name, a station name, and number for time to station w/ dwell each column, respectively, respectively. Notably,

the schedule file cannot send a train to a block without a station. Departure time will be calculated based off the first time to station.

In manual mode, the dispatcher will: 1. select a line through the drop down box, then 2. select either a station or a block as the destination and an arrival time, and 3. click the add destination button. The dispatcher will repeat the steps 2 and 3 for as many destinations as they need. Finally, they will click the dispatch train button. The CTC will then convert each arrival time to a time to station to match the format of auto.

When the information for the train to be dispatch is gathered, the CTC will add a destination at the end for the yard calculate an inital departure time based on the first time to station. The CTC will calculate the block by block path of the train. In addition, it will also calculate the authority, suggested speed, and expected time to each block along the path under ideal circumstances, ie. no other occupancies or blocks in maintenance. Using these paths, the CTC will compare all active train authority and suggested speed paths for each block based on the expected time to the block for the train. The train with the earliest expected time to block puts the authority and suggested speed to that block. The authority is then updated for various other cases such as block maintenance.

For most blocks, the suggested speed will be mostly static: set lower than the speed limit of both the block and the blocks within 200m since a train can brake from 70km/hr to 0 km/hr in about 150m. When the train wants to stop at a destination, the suggested speed for blocks before the destination in the path gradually decreases until the destination block's suggested is set to 0. The suggested speed and authority is sent to the Wayside Controller every time it changes for a block.

2.2.3 Authority Propagation

The Aurora team defines authority as a positive integer distance in meters for each train. Upon dispatching a train, the CTC office will assign that train an initial authority value. This value corresponds to the distance the train must travel before the next station on it's schedule. In addition to dispatching, CTC may also modify authorities corresponding to route changes. CTC sends this value to each wayside controller node on the corresponding line. When a train arrives at a station, the CTC Office will assign it a new authority.

The wayside controller may readjust the authority value for each train depending on that train's position and the positions of other trains. If a train needs to stop before crossing a switch, wayside shall modify the train's authority such that it will stop before crossing the switch. When wayside determines that train's path is clear and switch is in the correct position, wayside will reassign the train's authority so it may continue to it's next station. The wayside controller will send updated authority values for each train to track model.

When the track model receives updated authorities from wayside, they will be immediately pushed to their respective train models via the track circuit. Each train model then sends this value to its controller. When a train controller receives a non-zero authority, it will begin accelerating until it reaches its suggested speed. As the train travels along its route, it must update its own authority according to its speed. Its authority will steadily decrease as the distance between the train and its destination closes. The train will apply it's service brake when the authority becomes sufficiently small. Each train must have come to a complete stop when or before its authority reaches zero.

2.2.4 Wayside Node Distribution

Given the small scale of the track network, our team decided it best to place our wayside controller nodes intuitively. There are three nodes on the red line (1-3) and four on the green line (4-7) for a total of seven wayside controller nodes. Table 3 details which sections are owned by each node. For a visual representation, see Figures 2 and 3.

The rationale for our wayside node placements is based on the constraint preventing wayside nodes from communicating with each other and the limitations on the inputs to the PLC. We decided to have each node be responsible for roughly seven track sections and two switches. The reason we chose not to use more or less nodes is because we wanted to balance the cost of installing wayside nodes against the computational work performed by each individual node's PLC. Physically, the controllers are located at a station block for their respective regions.

Node ID	Line	Sections	Location
1	Red	*A-G*	Block 16
2	Red	*H, O-T*	Block 35
3	Red	*I-M*	Block 60
4	Green	*A-E*	Block 9
5	Green	*F-I, W-Z*	Block 31
6	Green	*J-L, T-V*	Block 73
7	Green	*M-S*	Block 88

Table 2: Wayside Controller Placements

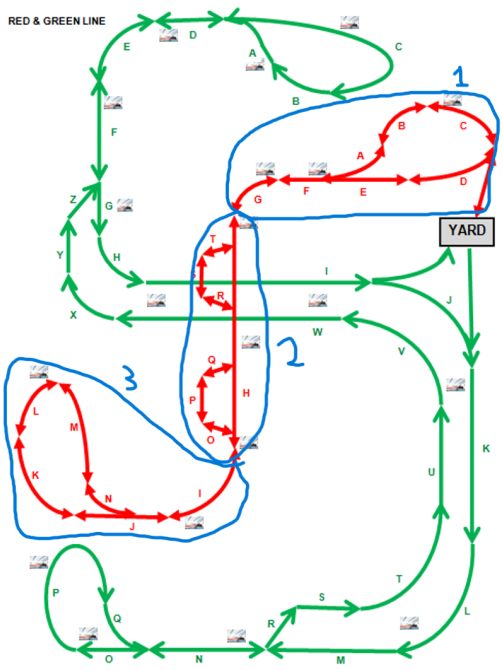

Figure 2: Red line wayside nodes

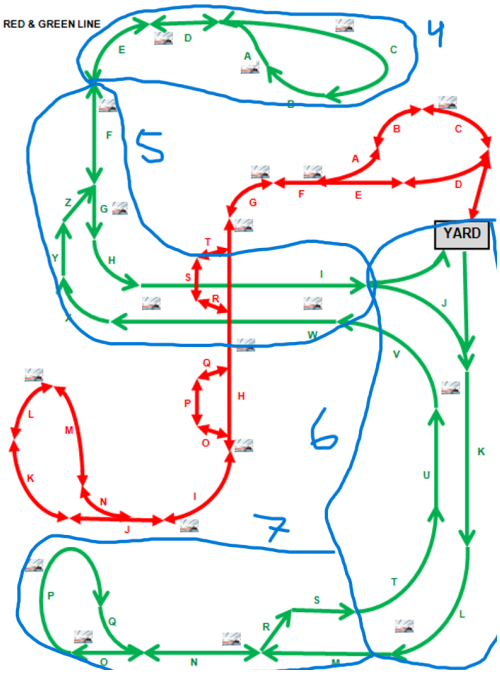

Figure 3: Green line wayside nodes

2.2.5 System Timing

To ensure the all the modules are synchronized, they will share a common simulation clock. This clock is implemented as a class titled *SimulationClock*. It stores its internal time using the python datetime module. The class has an internal threading-based repating timer which times out every tick. When the timeout occurs, the internal time is incremented by a timedelta based on the tick rate and simulation speed factor. In order for time-dependent modules to get an actively updated time, each top-level class will poll the simulation clock using an internal timer. For the non-time-dependent modules, the clock will be polled in the UI class so it may be displayed to the user. Each module's UI has a slider and pause/unpause button linked to the simulation clock.

2.2.6 System Vitality

The Wayside Controller implements vitality features through the use of handshakes. Whenever the Wayside Controller sends a switch, signal, or crossing command to the Track Model will reply with the udpated state. The Wayside Controller will compare the Track Model's state to its own state and assert that they are the same. If they are not the same, the Wayside COntroller will relay a failure a to the CTC Office.

For the train controller, the redundancy is for the commanded speed and power output to the train model. There shall be a function that activates the check every 10 seconds three times to ensure the information is correct. If there is an error within the checking, the action would not be enacted.

2.3 System Design

2.3.1 System Deployment

The system deployment diagram shows layout of the system if each module had its own dedicated device. Communication protocols are specified between each device, and each device contains its top-level software component and its UI component. In the system deployment of our railway control system, we have organized various modules to ensure seamless communication and control. At the heart of this deployment is the Central Traffic Control (CTC) module, which acts as the central nervous system of the entire network. CTC is responsible for connecting to Wayside Controller Hardware through a PubSub protocol and Wayside Controller Software via a wireless protocol. This bidirectional communication channel allows CTC to receive real-time information from Wayside Controllers and dispatch control commands to them, ensuring efficient train management and traffic coordination. The asynchronous Simulation Clock serves as the heartbeat of the entire system, providing a synchronized time reference for all modules, enabling precise and coordinated decision-making.

Moving further down the system hierarchy, Wayside Controllers establish a crucial link between CTC and the Track Model. Wayside Controllers communicate with the Track Model via track circuits, relaying track occupancy and condition information to CTC. In parallel, Track Model connects to the Train Model using a combination of beacons and the track circuit. This dual connection ensures the Train Model remains updated about its location and surrounding track conditions, facilitating safe and efficient train operation. Finally, the Train Model connects to the Train Controller through a wired connection, enabling real-time control of train operations based on instructions received from the Wayside Controllers. Together, these modules form a robust and interconnected system that guarantees the safe and efficient operation of our railway network, underpinned by the asynchronous Simulation Clock for synchronized time management.

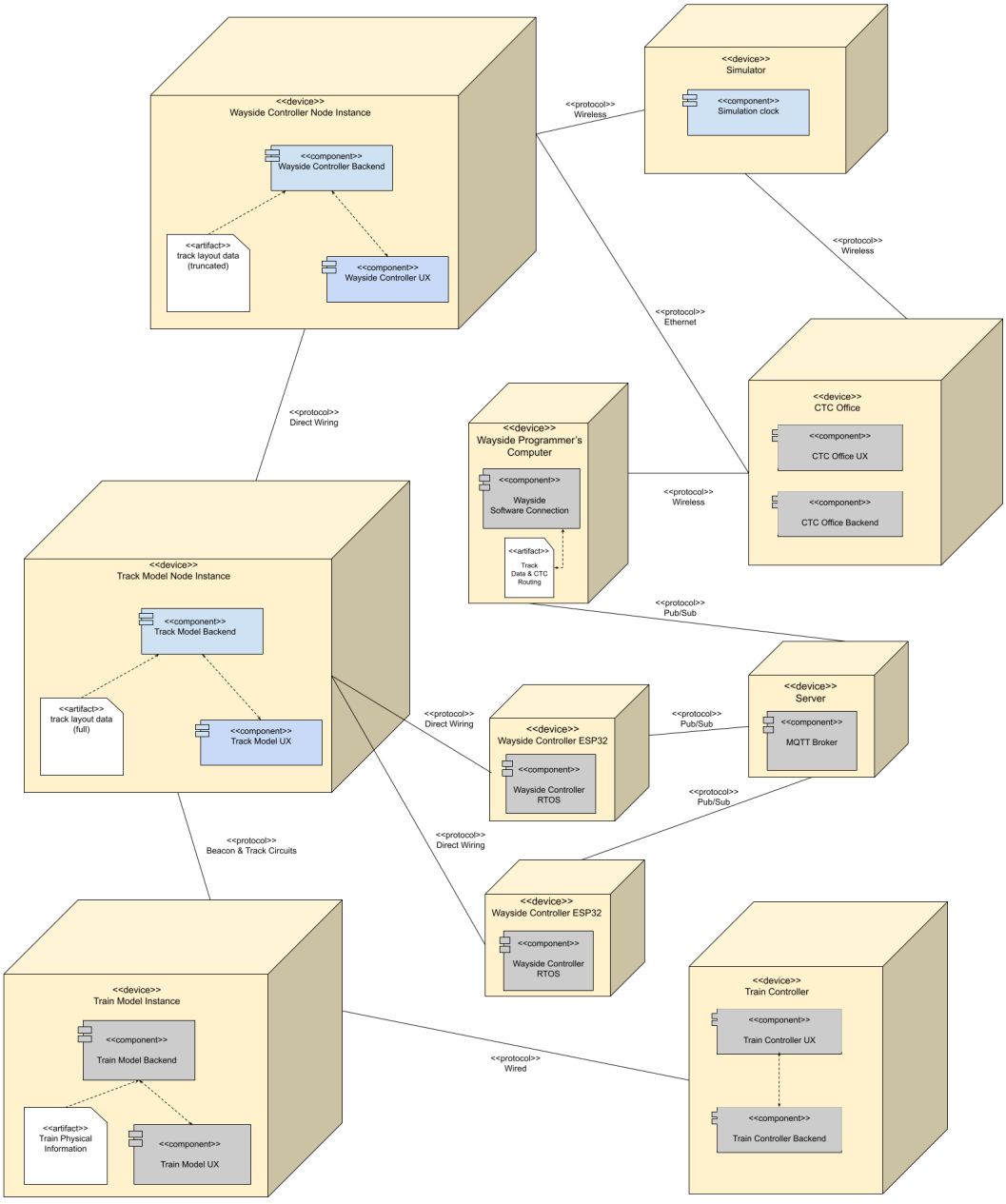

Figure 4: System Deployment Diagram

2.3.2 System Classes

Our system's overall class hierarchy consists of the five main modules, plus a launcher GUI and a universal simulation clock. The class diagram in Figure 6 shows classes for the launcher GUI and simulation clock, as well as five groups (gray boxes) for each module. Each group contains the top-level class and module UI as interfaces, plus one or more APIs for intermodular communication. Below the class diagram is a class-responsibility-collaborator card for each class not covered elsewhere in the document.

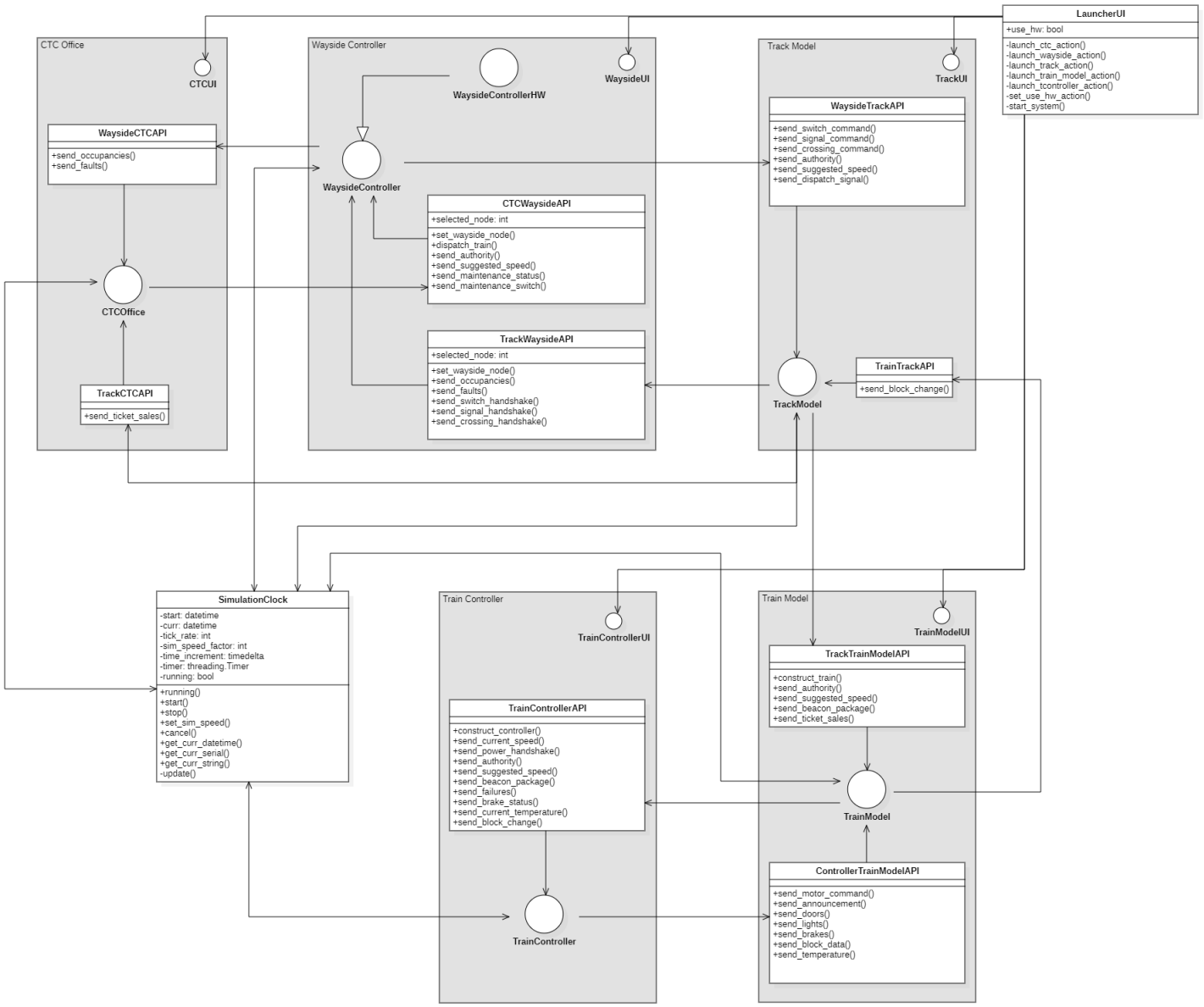

Figure 5: System Class Diagram

LauncherUI	
Description: GUI class for overall system launcher. Responsible for launching the whole application and opening individual module UI windows	
Attributes	
Name	Description
Hardware Use	Indicates whether to launch system using hardware Wayside Controller
Responsibilities	
Name	Collaborator
Launch CTC	User
Launch Wayside	User
Launch Track Model	User
Launch Train Model	User
Launch Train Controller	User
Select Hardware	User
Start System	User

SimulationClock	
Description: Timing class for the overall system. Responsible for keeping a frequently (∼50Hz) updated simulation time and allowing modules to adjust speed and pause/unpause	
Attributes	
Name	Description
Start Time	Store time to start clock
Current Time	Stores current simulation time
Tick Rate	Stores how often to update the clock in milliseconds
Simulation Speed	Stores multiplier for simulation clock speed
Time Increment	Stores how much to increment the current time by each tick
Timer	Threading-based repeating timer object
Running	Stores whether simulation is paused
Responsibilities	
Name	Collaborator
Running	All Modules
Start	All Modules
Stop	All Modules
Set Simulation Speed	All Modules
Cancel Timer	All Modules
Get Current Time as Datetime	All Modules
Get Current Time as Serial	All Modules
Get Current Time as String	All Modules

WaysideCTCAPI	
Description: API class responsible for dictating communications from Wayside Controller to CTC Office.	
Attributes	
Name	Description
Responsibilities	
Name	Collaborator
Send Occupancies	WaysideController
Send Faults	WaysideController

TrackCTCAPI	
Description: API class responsible for dictating communications from Track Model to CTC Office.	
Attributes	
Name	Description
Responsibilities	
Name	Collaborator
Send Ticket Sales	TrackModel

CTCWaysideAPI	
Description: API class responsible for dictating communications from CTC Office to Wayside Controller.	
Attributes	
Name	Description
Selected Wayside Node	Stores which Wayside Controller instance to communicate with
Responsibilities	
Name	Collaborator
Set Wayside Node	CTCOffice
Dispatch Train	CTCOffice
Send Authority	CTCOffice
Send Suggested Speed	CTCOffice
Send Maintenance	CTCOffice
Send Switch Command	CTCOffice

TrackWaysideAPI	
Description: API class responsible for dictating communications from Track Model to Wayside Controller.	
Attributes	
Name	Description
Selected Wayside Node	Stores which Wayside Controller instance to communicate with
Responsibilities	
Name	Collaborator
Set Wayside Node	Track Model
Send Occupancy	Track Model
Send Failure	Track Model
Send Switch Handshake	Track Model
Send Signal Handshake	Track Model
Send Crossing Handshake	Track Model

WaysideTrackAPI	
Description: API class responsible for dictating communications from Wayside Controller to Track Model.	
Attributes	
Name	Description
Responsibilities	
Name	Collaborator
Send Switch Command	WaysideController
Send Signal Command	WaysideController
Send Crossing Command	WaysideController
Send Dispatch Signal	WaysideController
Send Authority	WaysideController
Send Suggested Speed	WaysideController

TrainTrackAPI	
Description: API class responsible for dictating communications from Train Model to Track Model.	
Attributes	
Name	Description
Responsibilities	
Name	Collaborator
Send Block Change	TrainModel

TrackTrainModelAPI	
Description: API class responsible for dictating communications from Track Model to Train Model.	
Attributes	
Name	Description
Responsibilities	
Name	Collaborator
Construct Train	TrackModel
Send Authority	TrackModel
Send Suggested Speed	TrackModel
Send Beacon Package	TrackModel
Send Ticket Sales	TrackModel

ControllerTrainModelAPI	
Description: API class responsible for dictating communications from Train Controller to Train Model.	
Attributes	
Name	Description
Responsibilities	
Name	Collaborator
Send Motor Command	TrainController
Send Station Announcement	TrainController
Send Doors Command	TrainController
Send Lights Command	TrainController
Send Brakes Command	TrainController
Send Block Data	TrainController
Send Temperature Setpoint	TrainController
Send Block Change	TrainController

TrainControllerAPI	
Description: API class responsible for dictating communications from Train Model to Train Controller.	
Attributes	
Name	Description
Responsibilities	
Name	Collaborator
Construct Controller	TrainModel
Send Current Speed	TrainModel
Send Power Handshake	TrainModel
Send Authority	TrainModel
Send Suggested Speed	TrainModel
Send Beacon Package	TrainModel
Send Failures	TrainModel
Send Brakes Status	TrainModel
Send Current Cabin Temperature	TrainModel

2.3.3 System Use Cases

The system use case diagram (Figure 7) is a simplification of all the actions each user is capable of performing. The user stories, use case descriptions, and sequence diagrams for the use cases not covered elsewhere in the document are below the diagram.

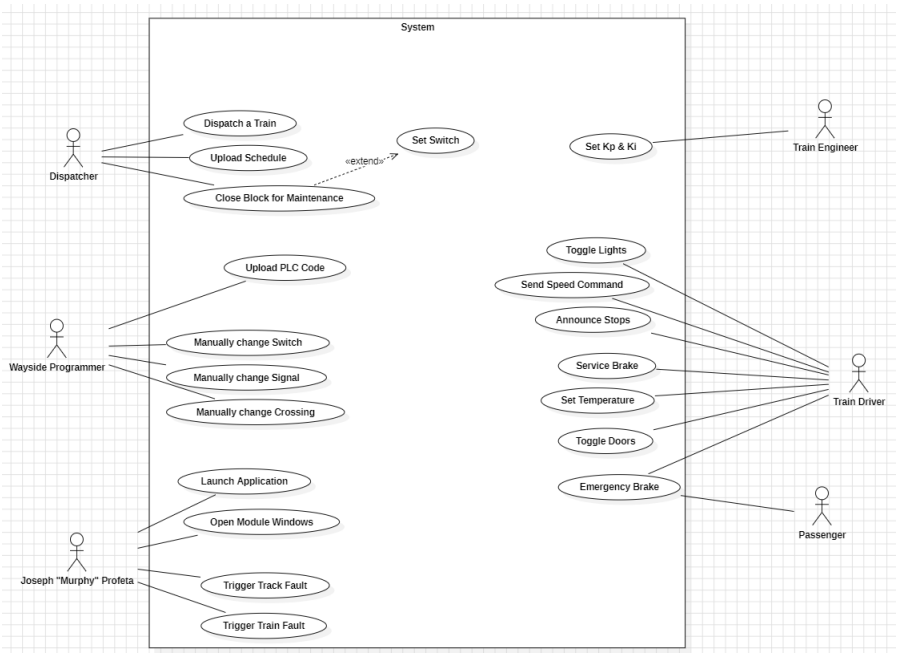

Figure 6: System Use Case Diagram

User launches the Application:

User launches the Application
User Story: As Joseph Profeta, I want to launch Aurora's application so that I may utilize the train system and give the team an outstanding grade.
Acceptance Criteria: Given that Joseph Profeta wants to launch the application, when he double-clicks the executable file, then the launcher window will open.

User launches the Application	
Actors	Joseph Profeta
Preconditions	Joseph Profeta needs to launch the application
Postconditions	The launcher window is opened and the top-level objects for the CTC Office, Wayside Controllers SW & HW, and Track Model are instantiated.
Normal Flow	1. Instantiate CTC, Wayside, and Track Model 2. Open Launcher Window
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

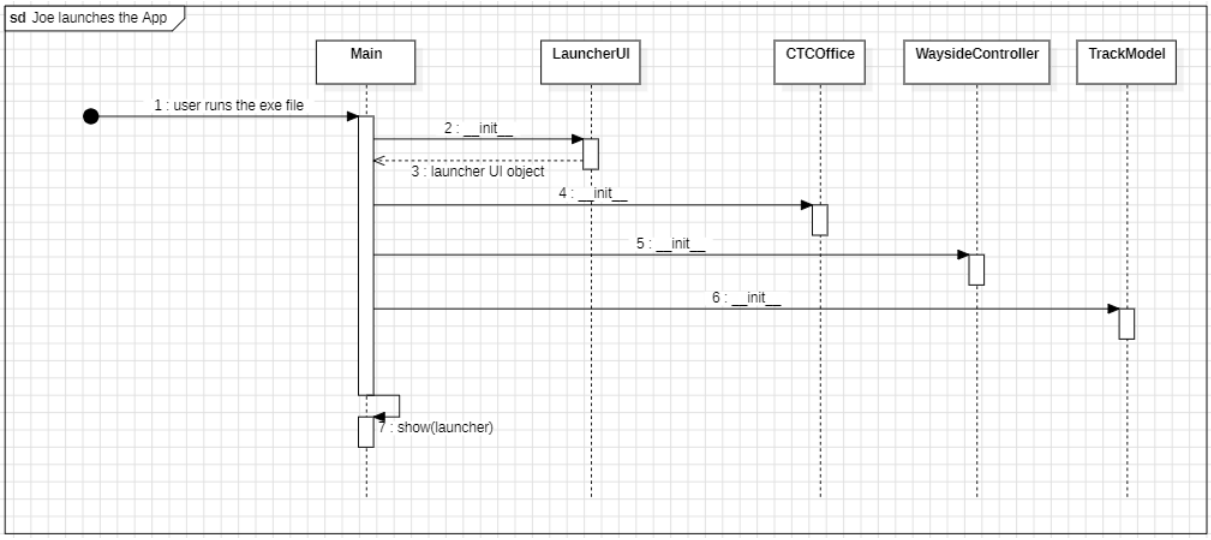

Figure 7: Sequence Diagram for ”User launches the Application”

User starts the System:

User starts the System
User Story: As Joseph Profeta, I want to start the system so that I may view and assess the results of the simulation.
Acceptance Criteria: Given that Joseph Profeta wants to start the system, when he clicks the ”start system” button in the launcher, then the simulation will begin.

User starts the System	
Actors	Joseph Profeta
Preconditions	1. Joseph Profeta would like to start the system 2. The launcher window is open
Postconditions	1. The APIs for CTC, Wayside, and Track Model are linked 2. The simulation clock begins running at 1x speed
Normal Flow	1. Launcher connects the APIs 2. Launcher starts the simulation by constructing the SimulationClock object
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

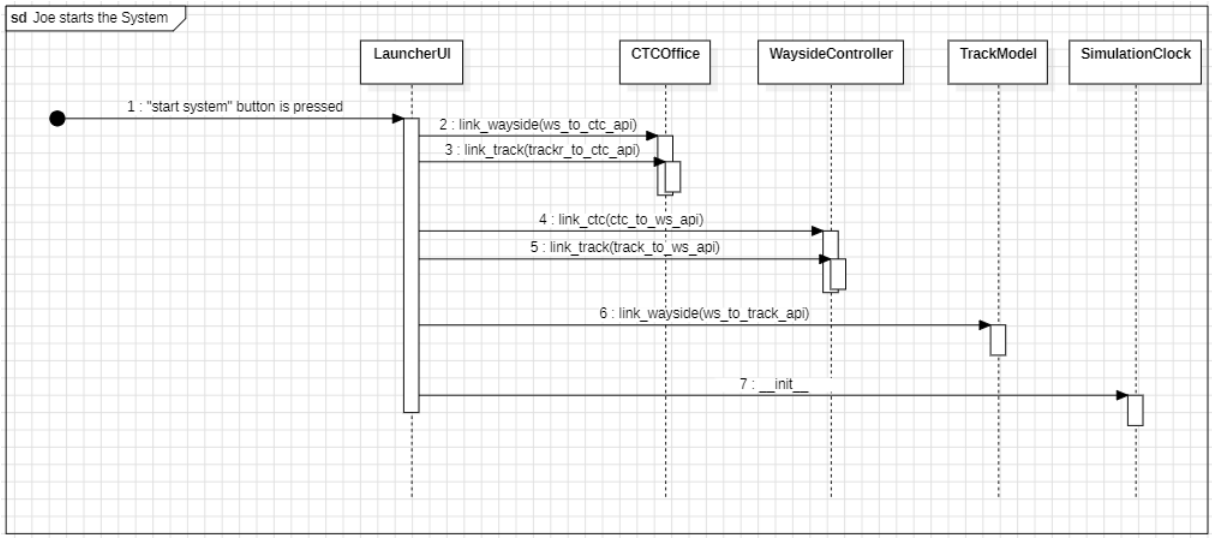

Figure 8: Sequence Diagram for ”User starts the System”

3 Sub-System Design

3.1 CTC Office

3.1.1 Design Overview

The purpose of the CTC Office is to let the dispatcher interact with the rail system. The dispatcher can dispatch and schedule trains through the CTC graphical user interface which calculate all data necessary to safely send out the train including departure time, path, authority, and suggested speed. The dispatcher will also be able to view the state of the system through block occupancies and throughput through ticket sales from the entire system. The dispatcher will also be able to perform maintenance on blocks.

3.1.2 Class Diagram

The below figure 9 shows objects, classes, and external functions that interact with the CTC.

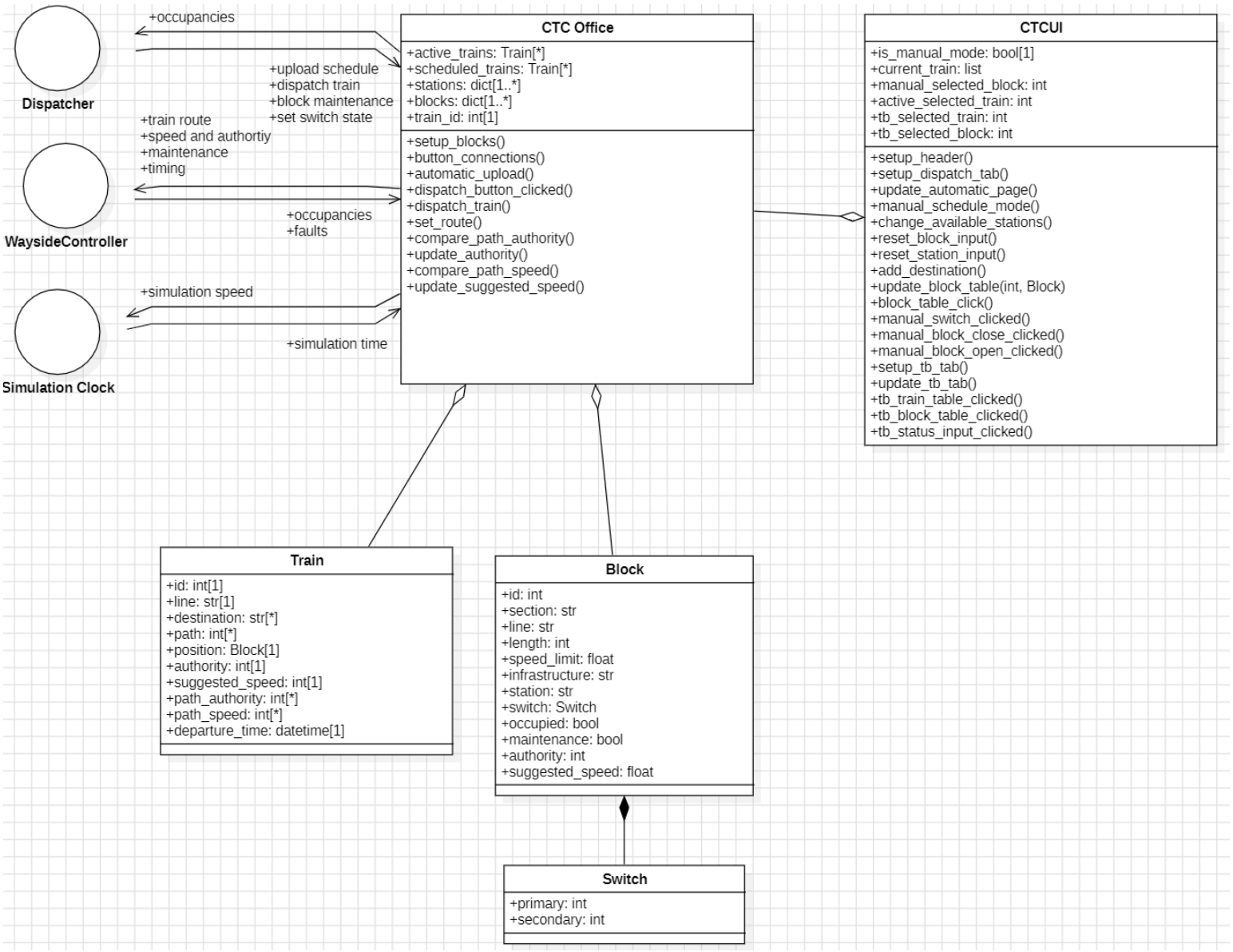

Figure 9: CTC Class Diagram

Class Cards

ctcOffice	
Description: Top-level class for the CTC software module. Responsible for interacting with CTC GUI, Wayside Controller, and Track Model.	
Attributes	
Name	Description
Active Trains	Keeps track of trains on the rail system
Scheduled Trains	Stores scheduled trains that have are still in yard
Blocks	Stores static & dynamic information about each block in the rail system
Train ID	Stores the ID of the next train to be dispatched
Responsibilities	
Name	Collaborator
Dispatch Train	CTCUI, Wayside Controller
Schedule Train	CTCUI, Wayside Controller
Set Switch	CTCUI, Wayside Controller
View Occupancies	CTCUI, Wayside Controller
View Throughput	CTCUI, Track Model
Set Maintenance	CTCUI, Wayside Controller
Set Authority	Self, CTC Office
Set Suggested Speed	Self, CTC Office

CTCUI	
Description: Graphical user interface for the CTC module. Responsible for directly interacting with the dispatcher.	
Attributes	
Name	Description
Is manual mode	Keeps track on whether the system is in manual or automatic mode
Current Train	Stores the train information of temporary train created in manual mode
Manual Selected Block	Stores which block the dispatcher has selected
Responsibilities	
Name	Collaborator
Dispatch Train	Dispatcher, CTC Office
Schedule Train	Dispatcher, CTC Office
Set Switch	Dispatcher, ctcOffice
View Occupancies	Dispatcher, Wayside Controller
View Throughput	Dispatcher, Track Model
Set Maintenance	CTCUI, Wayside Controller
Set Authority	Self, CTC Office
Set Suggested Speed	Self, CTC Office

Block	
Description: Storage class responsible for storing all relevant information about a block. This includes static and dynamic information.	
Attributes	
Name	Description
ID	Stores block ID number
Section	Stores section that this block is in
Line	Stores line that this block is on
Length	Stores the length of the block
Speed Limit	Stores the speed limit of the block
Station	Stores the name of the station on the block if applicable
Infrastructure	Stores a displayable list of infrastructure of the block
Occupied	Stores whether the block is occupied
Switch	Stores block switch data if applicable
Occupancy	Stores whether block is occupied
Maintenance	Stores whether block is under maintenance
Responsibilities	
Name	Collaborator

Switch	
Description: Storage class responsible for storing a switch's primary position and secondary position.	
Attributes	
Name	Description
Primary	Stores block ID of primary position
Secondary	Stores block ID of secondary position
Responsibilities	
Name	Collaborator

Train	
Description: Storage class responsible for storing all relevant information about a train. This includes static and dynamic information.	
Attributes	
Name	Description
Train ID	Stores train's ID number
Line	Stores the train's line
Destinations	Stores the trains destinations
Departure Time	Stores the train's departure time
Position	Stores the train's position
Authority	Stores train's current assigned authority
Suggested Speed	Stores train's current suggested speed
Path	Stores the train's block by block path
Path Authority	Stores the authority along the train's path assuming no interruptions
Path Speed	Stores the suggested speed along the train's path assuming no interruptions
Responsibilities	
Name	Collaborator

3.1.3 Use Cases

The below figure 10 shows use cases for the CTC.

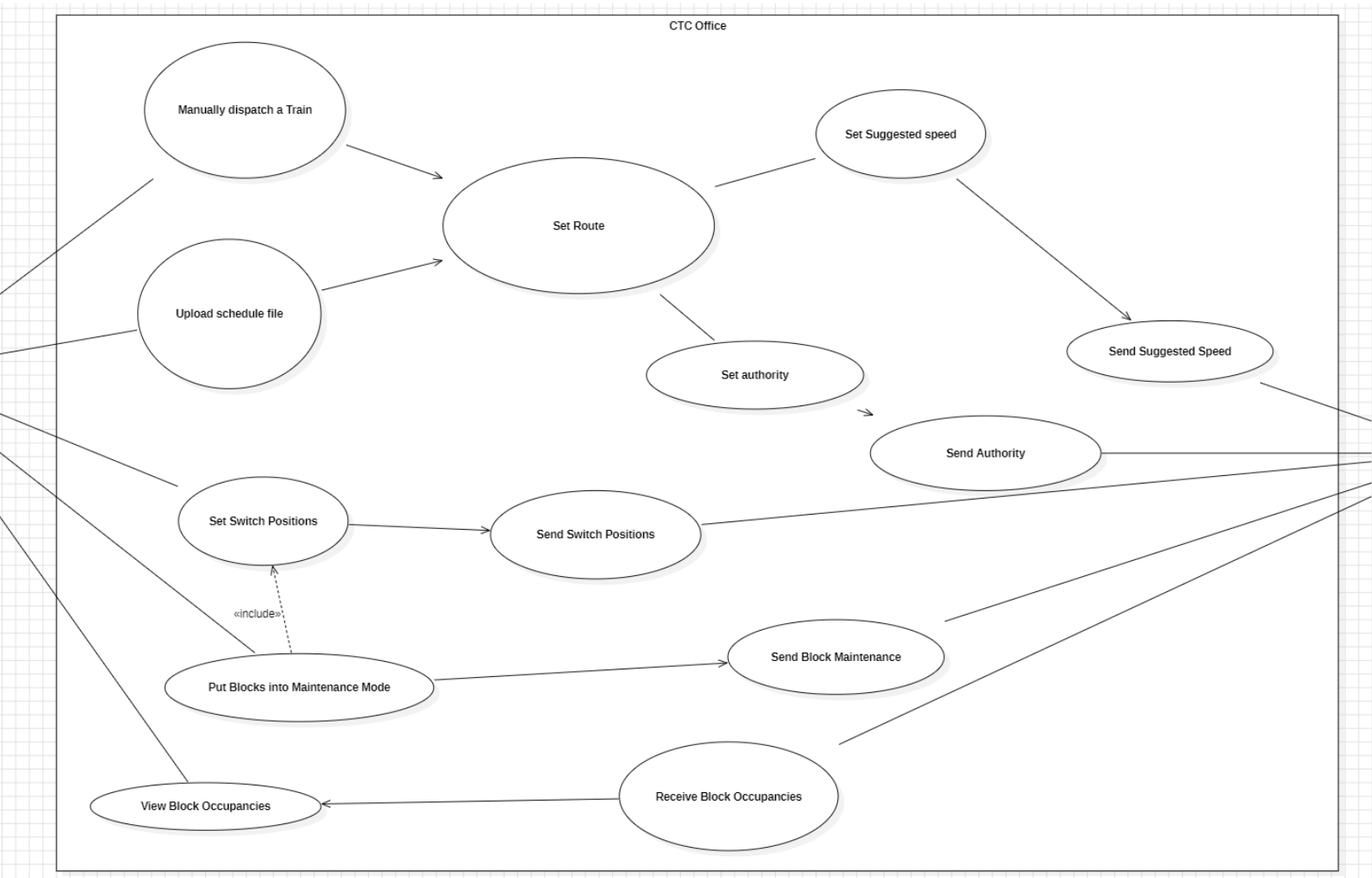

Figure 10: CTC Use Case Diagram

The dispatcher uploads a schedule in automatic mode
User Story: As a dispatcher, I want to upload a file with today's daily train routes so that the system automatically dispatches trains throughout the day..
Acceptance Criteria: .

The dispatcher views occupancies of the system
User Story: As a dispatcher, I want to view the occupancies of the entire train route so that I can notice when a train derails. (CHANGE)
Acceptance Criteria: .

The dispatcher switches between automatic mode and manual mode
User Story: As a dispatcher, I want to switch between manual and automatic mode so that I can decide how to run the system at a given time.
Acceptance Criteria: ·

In Manual Mode, the dispatcher can do the following:
User Story: As a dispatcher, I want to dispatch a train to the White station at 4:30pm so that the system will have enough capacity for rush hour.
Acceptance Criteria: ·

Dispatcher can dispatch trains to certain blocks, in addition to stations:
User Story: As a dispatcher, I want to dispatch a train to a specific block for sight seeing purposes.
Acceptance Criteria: ·

Dispatcher can put blocks into maintenance mode, effectively blocking all trains from entering the block:
User Story: As a dispatcher, I want to disable a block so that trains do not run over repair crews on the track.
Acceptance Criteria: ·

Disable a block in maintenance mode:
User Story: As a dispatcher, I want to disable a block so that trains do not run over repair crews on the track.
Acceptance Criteria: ·

Change switch connections in maintenance mode
User Story: As a dispatcher, I want to test the switch of a block in maintenance mode so that I know it works.
Acceptance Criteria: ·

Dispatcher receives ticket sales
User Story: As a dispatcher, I want to see how many ticket sales have occurred so that I can send out more trains if needed.
Acceptance Criteria: ·

Dispatcher dispatches a train manually:

Dispatcher dispatches a train manually
User Story: As a dispatcher, I want to dispatch a train to White Station at 4:00pm in order to accommodate the rush hour.
Acceptance Criteria: Given that the dispatcher wants to manually dispatch a train, when the dispatcher enters a set of destinations and arrival times and dispatches the train, then the CTC dispatches a train at an appropriate departure time with an appropriate authority and suggested speed.

Dispatcher dispatches a train manually	
Actors	Dispatcher, Wayside Controller
Preconditions	The dispatcher wants to manually dispatch a train
Postconditions	A train is dispatched with the destinations specified by the dispatcher with appropriate authority and suggested speed at an appropriate time.
Normal Flow	1. The dispatcher inputs a set of destinations and arrival times 2. The CTC calculates departure time for the train 3. The CTC calculates the block by block path of the train 4. At departure time, the CTC sends a dispatch train signal to the Wayside Controller
Exceptional Flow	1. The dispatcher inputs a time impossible to reach: The CTC will calculate the nearest possible time to the inpu

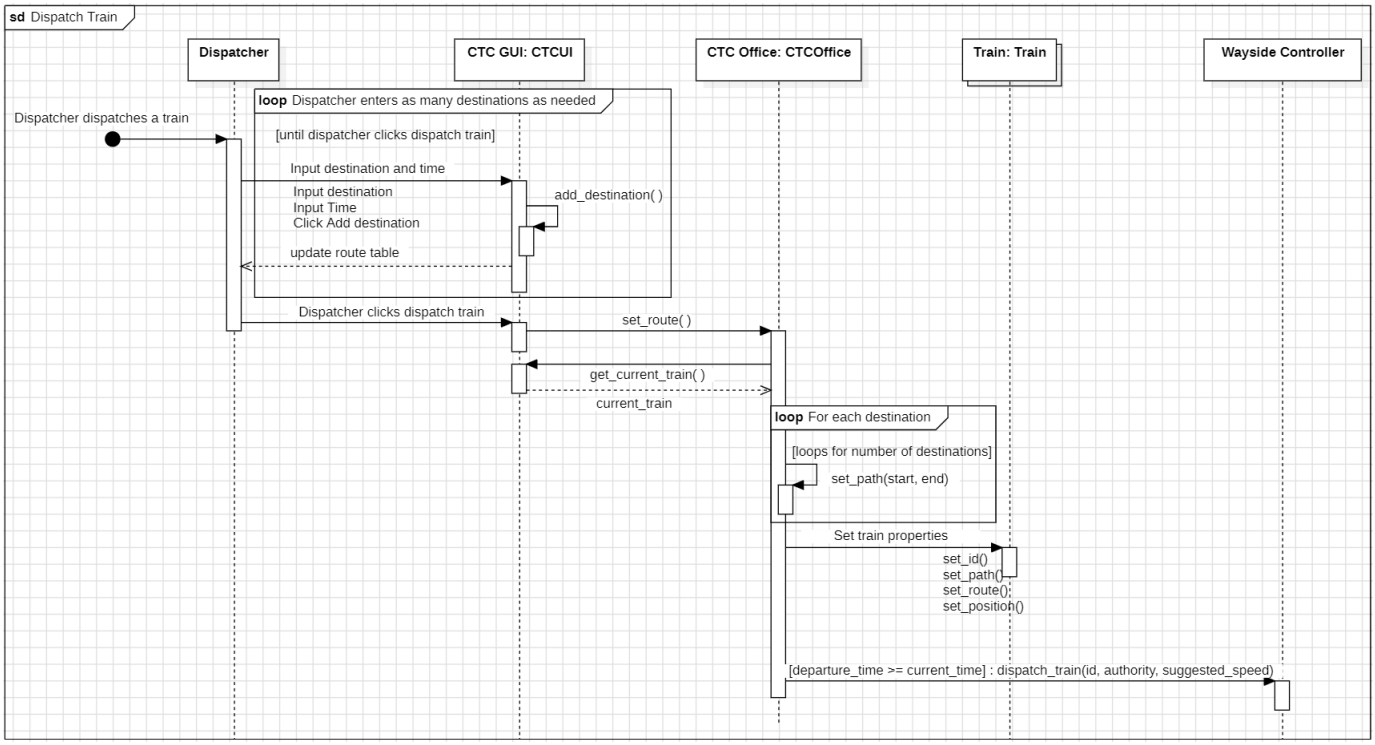

Figure 11: Sequence Diagram for ”Dispatcher dispatches a train manually”

Dispatcher uploads a schedule file:

Dispatcher uploads a schedule file
User Story: As a dispatcher, I want to upload a file with today's daily train routes so that the system automatically dispatches trains throughout the day.
Acceptance Criteria: Given that the dispatcher wants to schedule a train, when the dispatcher uploads a schedule file, then the CTC dispatches a train at an appropriate departure time with an appropriate authority and suggested speed.

Dispatcher uploads a schedule file	
Actors	Dispatcher, Wayside Controller
Preconditions	The dispatcher wants to schedule a train
Postconditions	A train is scheduled with the destinations specified by the schedule file with appropriate authority and suggested speed at an appropriate time.
Normal Flow	1. 2. 3. 4.
Exceptional Flow	XXXX

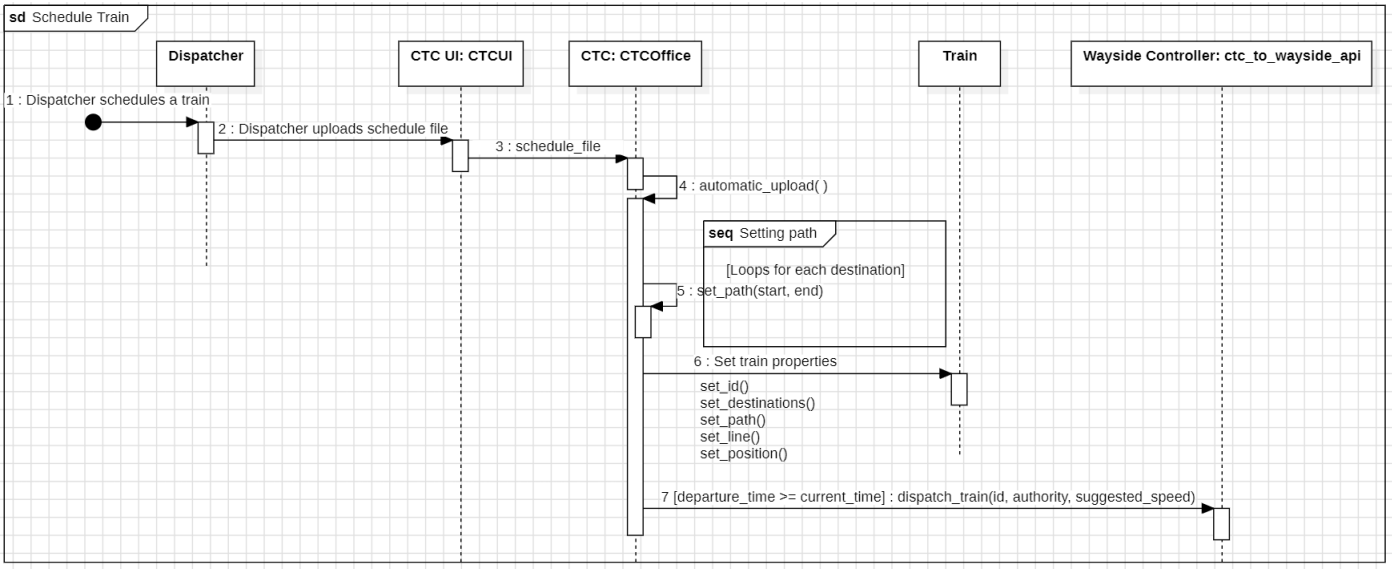

Figure 12: Sequence Diagram for "Dispatcher uploads a schedule file"

Dispatcher puts a block into maintenance:

Dispatcher puts a block into maintenance
User Story: As a dispatcher, I want to dispatch a train to White Station at 4:00pm in order to accommodate the rush hour.
Acceptance Criteria: Given that the dispatcher wants to dispatch a train, when the dispatcher does blank, do blank.

Dispatcher puts a block into maintenance	
Actors	Dispatcher, Wayside Controller
Preconditions	The dispatcher wants to put a block into maintenance
Postconditions	The track is put into maintenance mode.
Normal Flow	1. 2. 3. 4.
Exceptional Flow	XXXX

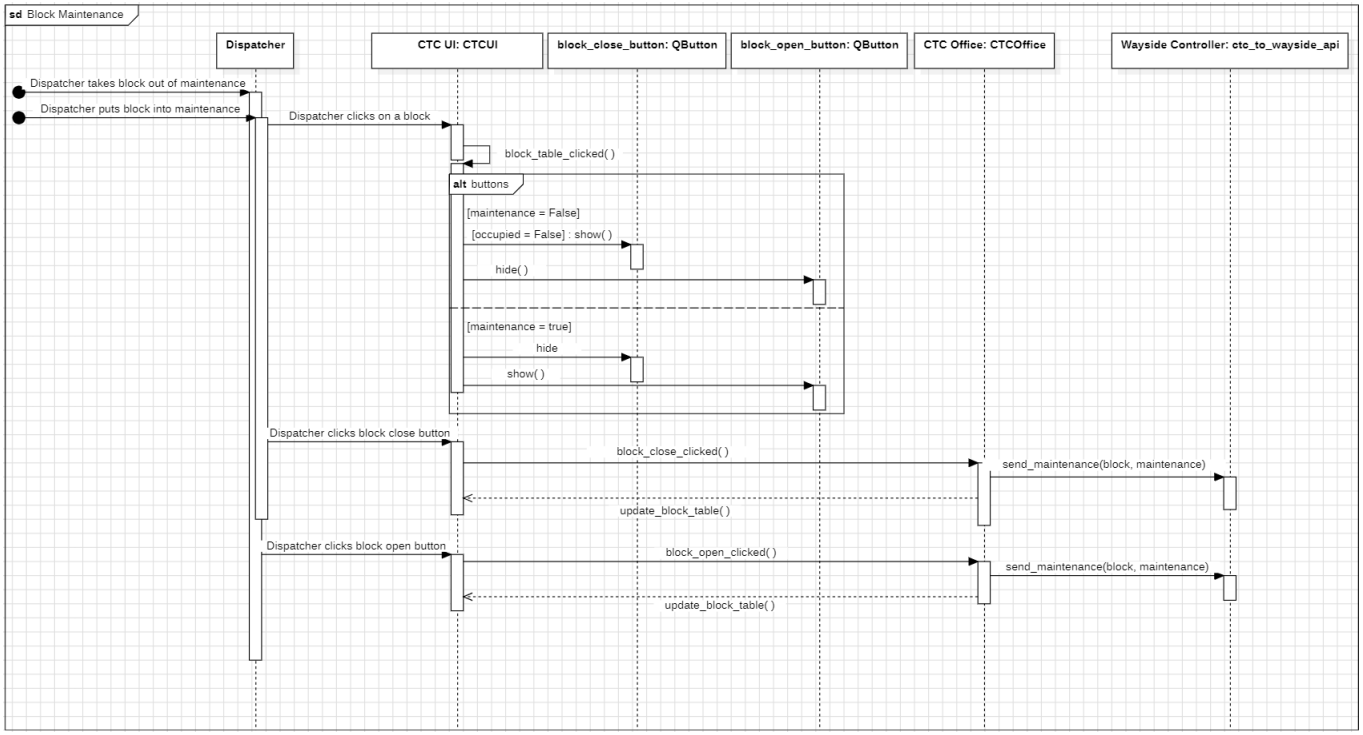

Figure 13: Sequence Diagram for ”Dispatcher puts a block into maintenance”

Dispatcher changes the switch state of a block:

Dispatcher changes the switch state of a block
User Story: As a dispatcher, I want to dispatch a train to White Station at 4:00pm in order to accommodate the rush hour.
Acceptance Criteria: Given that the dispatcher wants to dispatch a train, when the dispatcher does blank, do blank.

Dispatcher changes the switch state of a block	
Actors	Dispatcher, Wayside Controller
Preconditions	The dispatcher wants to change the state of a switch on a block
Postconditions	XXXX.
Normal Flow	1. 2. 3. 4.
Exceptional Flow	XXXX

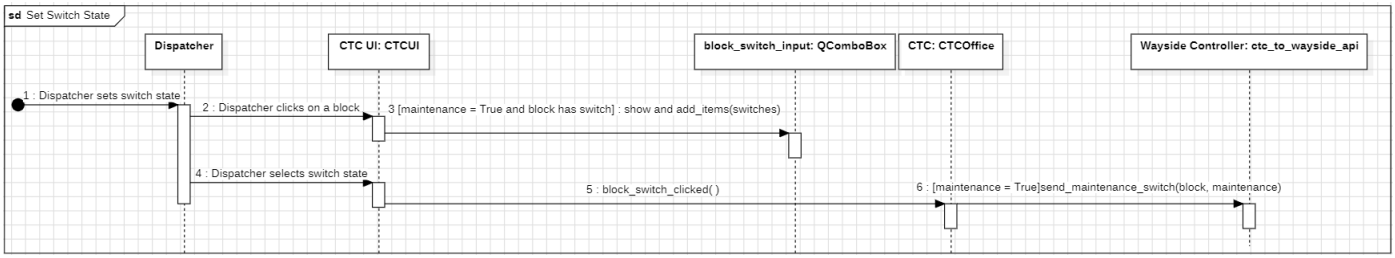

Figure 14: Sequence Diagram for ”Dispatcher changes the switch state of a block”

Dispatcher views the blocks' occupancies:

Dispatcher views the blocks' occupancies
User Story: As a dispatcher, I want to dispatch a train to White Station at 4:00pm in order to accommodate the rush hour.
Acceptance Criteria: Given that the dispatcher wants to dispatch a train, when the dispatcher does blank, do blank.

Dispatcher views the blocks' occupancies	
Actors	Dispatcher, Wayside Controller
Preconditions	XXXX
Postconditions	XXXX.
Normal Flow	1. 2. 3. 4.
Exceptional Flow	XXXX

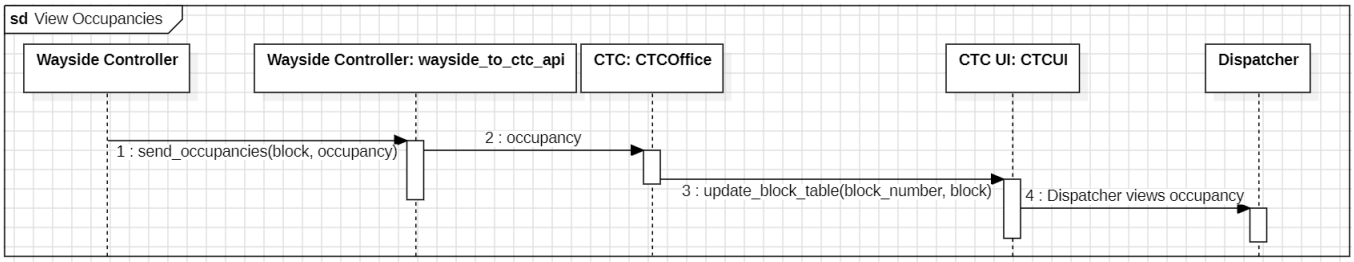

Figure 15: Sequence Diagram for ”Dispatcher views the blocks' occupancies”

Dispatcher views the system's throughput:

Dispatcher views the system's throughput
User Story: As a dispatcher, I want to dispatch a train to White Station at 4:00pm in order to accommodate the rush hour.
Acceptance Criteria: Given that the dispatcher wants to dispatch a train, when the dispatcher does blank, do blank.

Dispatcher views the system's throughput	
Actors	Dispatcher, Wayside Controller
Preconditions	The dispatcher wants to change the state of a switch on a block
Postconditions	XXXX.
Normal Flow	1. 2. 3. 4.
Exceptional Flow	XXXX

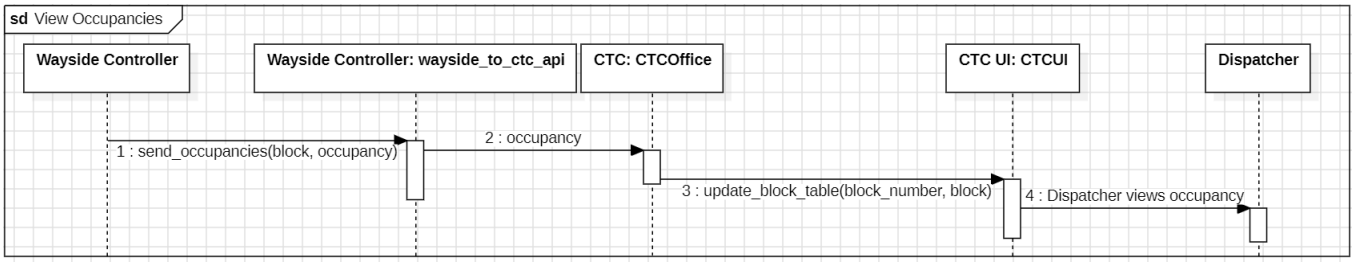

Figure 16: Sequence Diagram for "Dispatcher views the system's throughput"

CTC sends authority to Wayside Controller:

CTC sends authority to Wayside Controller
User Story: As the Wayside Controller, I want to get the authority .
Acceptance Criteria: Given that the dispatcher wants to dispatch a train, when the dispatcher does blank, do blank.

CTC sends authority to Wayside Controller	
Actors	Wayside Controller
Preconditions	The dispatcher wants to change the state of a switch on a block
Postconditions	XXXX.
Normal Flow	1. Simulation clock prompts CTC to update authority. 2. 3. 4.
Exceptional Flow	XXXX

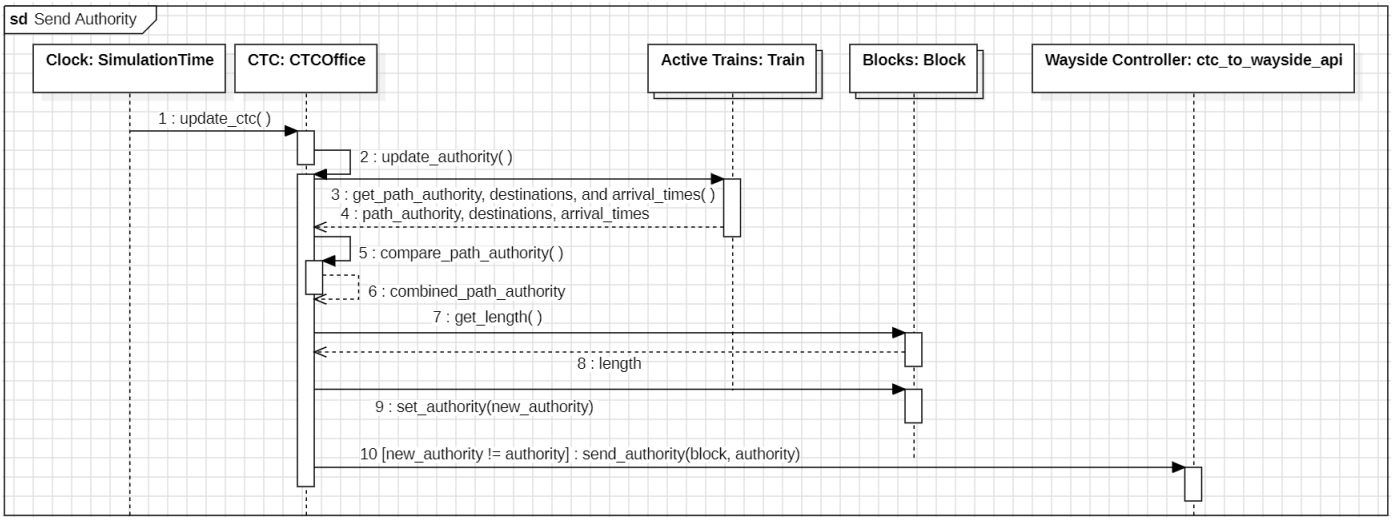

Figure 17: Sequence Diagram for ”CTC sends authority to Wayside Controller”

CTC sends suggested speed to Wayside Controller:

CTC sends suggested speed to Wayside Controller
User Story: As a dispatcher, I want to dispatch a train to White Station at 4:00pm in order to accommodate the rush hour.
Acceptance Criteria: Given that the dispatcher wants to dispatch a train, when the dispatcher does blank, do blank.

CTC sends suggested speed to Wayside Controller	
Actors	Wayside Controller
Preconditions	The dispatcher wants to change the state of a switch on a block
Postconditions	XXXX.
Normal Flow	1. 2. 3. 4.
Exceptional Flow	XXXX

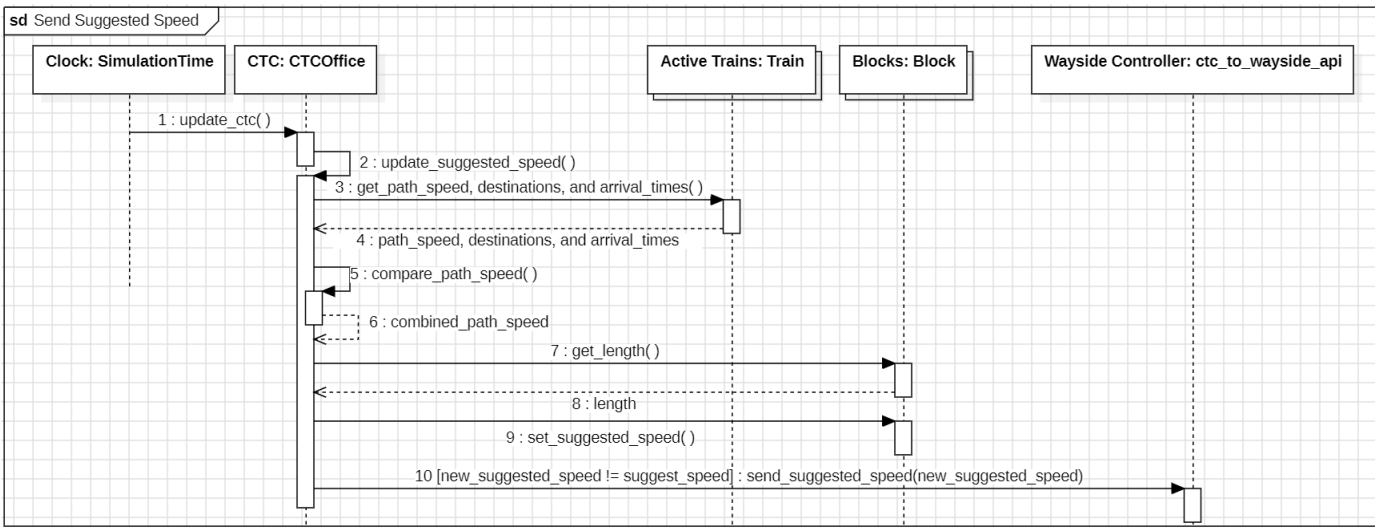

Figure 18: Sequence Diagram for ”CTC sends suggested speed to Wayside Controller”

3.2 Wayside Controller Software

3.2.1 Design Overview

The wayside controller is a small outdoor controller placed periodically throughout the rail system. It's primary purpose is to control track switches, signal lights, and railroad crossings. It is also responsible for reporting block occupancies and faults to the CTC office and relaying information from CTC to the track model. This information includes suggested speeds and authorities for each train. The wayside controller may also modify authority in situations involving intersecting train paths. Finally, the wayside controllers' programmable logic controllers (PLCs) that control the switches, signals, and crossings may be modified and appended to by the user. Details about this user are provided in the SRS document.

3.2.2 Class Diagram

The overall class diagram for the wayside controller software module is shown in Figure 19. The class titled *WaysideController* is the top-level class for this module. Precisely seven instances of this class will be constructed at runtime; one for each wayside controller node (see section 2.2.3 Wayside Node Distribution). Below the class diagram is a class-responsibility-collaborator card for each class. Note that the underlines indicate static class members.

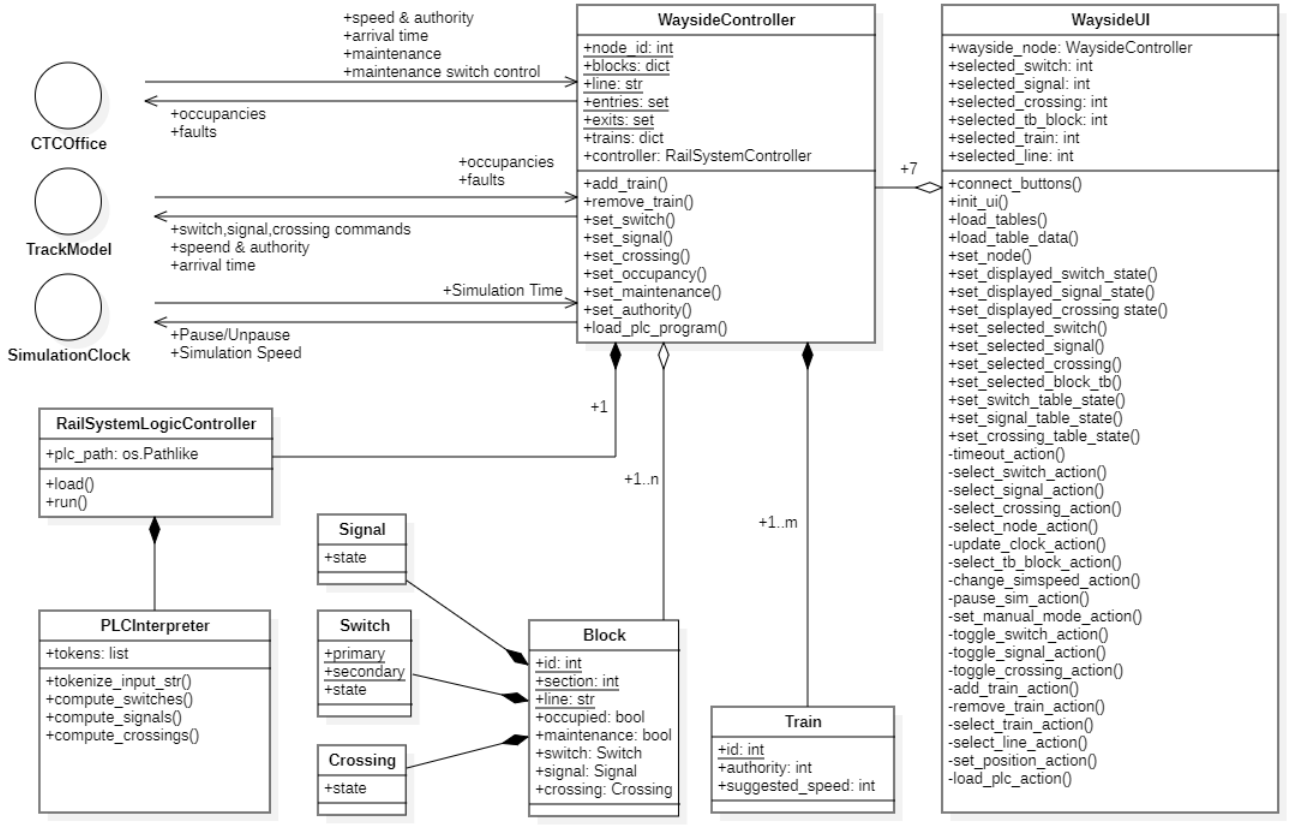

Figure 19: Wayside Controller Software Class Diagram

WaysideController	
Description: Top-level class for the wayside controller software module. Responsible for interacting with GUI, CTC Office, and Track Model.	
Attributes	
Name	Description
Node ID	Identifies which wayside controller node this instance corresponds to
Line	Identifies which line (red or green) the wayside controller is placed on
Entries & Exits	Identifies which blocks are entrances to or exits from the wayside node's region
Blocks	Stores static & dynamic information about each block in the wayside controller's region
Trains	Stores static & dynamic information about each train on the wayside controller's line
Responsibilities	
Name	Collaborator
Add Train	CTC Office
Remove Train	Self
Set Switch	WaysideUI, Self, Track Model
Set Signal	WaysideUI, Self, Track Model
Set Crossing	WaysideUI, Self, Track Model
Set Occupancy	Track Model
Set Maintenance	CTC Office
Load PLC Program	WaysideUI
Set Authority	Self, CTC Office
Set Suggested Speed	Self, CTC Office

WaysideUI	
Description: Graphical user interface class for wayside controller module. Responsible for allowing the user to interact with each wayside contoller instance.	
Attributes	
Name	Description
Wayside Controller Node	Stores current selected wayside node
Responsibilities	
Name	Collaborator
Select Switch	User, Self
Select Signal	User, Self
Select Crossing	User, Self
Select Node	User, Self
Set Simulation Speed	User, SimulationClock
Pause Simulation	User, SimulationClock
Set Manual Mode	User, Self
Toggle Switch	User, Self, WaysideController
Toggle Signal	User, Self, WaysideController
Toggle Crossing	User, Self, WaysideController
Toggle Switch	User, Self, WaysideController
Select Testbench Block	User, Self
Select Testbench Train	User, Self
Select Testbench Line	User, Self
Upload PLC Code	User, Self

RailSystemLogicController	
Description: Controller class responsible for controlling all rail system-related logic, including switches, signals, crossings, and occupancies.	
Attributes	
PLC Path	Store the file path to the PLC Program file
Responsibilities	
Name	Collaborator
Load PLC Program	WaysideController, PLCInterpreter
Run PLC Program	WaysideController, PLCInterpreter

PLCInterpreter	
Description: Interpreter class responsible for loading, verifying, and executing the PLC program.	
Attributes	
PLC Program Tokens	Stores all the parsed elements of the PLC Program
Responsibilities	
Name	Collaborator
Tokenize Input String	RailSystemLogicController
Get Switch States	RailSystemLogicController
Get Signal States	RailSystemLogicController
Get Crossing States	RailSystemLogicController

Block	
Description: Storage class responsible for storing all relevant information about a block. This includes static and dynamic information.	
Attributes	
Name	Description
ID	Stores block ID number
Section	Stores section that this block is in
Line	Stores line that this block is on
Occupancy	Stores whether block is occupied
Maintenance	Stores whether block is under maintenance
Switch	Stores block switch data if applicable
Signal	Stores block signal data if applicable
Crossing	Stores block crossing data if applicable
Responsibilities	
Name	Collaborator

Switch	
Description: Storage class responsible for storing a switch's primary position, secondary position, and current state.	
Attributes	
Name	Description
Primary	Stores block ID of primary position
Secondary	Stores block ID of secondary position
State	Stores current switch state: 1 for primary, 2 for secondary
Responsibilities	
Name	Collaborator

Signal	
Description: Storage class responsible for storing a signal state.	
Attributes	
Name	Description
State	Stores current signal state: 1 for Green, 2 for Red
Responsibilities	
Name	Collaborator

Crossing	
Description: Storage class responsible for storing a crossing state.	
Attributes	
Name	Description
State	Stores current state: 1 for Open, 2 for Closed
Responsibilities	
Name	Collaborator

Train	
Description: Storage class responsible for storing all relevant information about a train. This includes static and dynamic information.	
Attributes	
Name	Description
Train ID	Stores train's ID number
Authority	Stores train's current assigned authority
Suggested Speed	Stores train's current suggested speed
Responsibilities	
Name	Collaborator

3.2.3 Use Cases

The main three use case actors for the wayside controller module are the CTC office, track model, and traffic control programmer. I included a fourth actor called the quality assurance tester. In practice, this is a member of the Aurora team who will test the wayside controller as a standalone module via its GUI test bench. This actor will act as a substitute for the CTC office and track model. The wayside controller's use case diagram is seen in Figure 19. Below the use case diagram, there is a user story, use case description, and sequence diagram for each non-trivial use case.

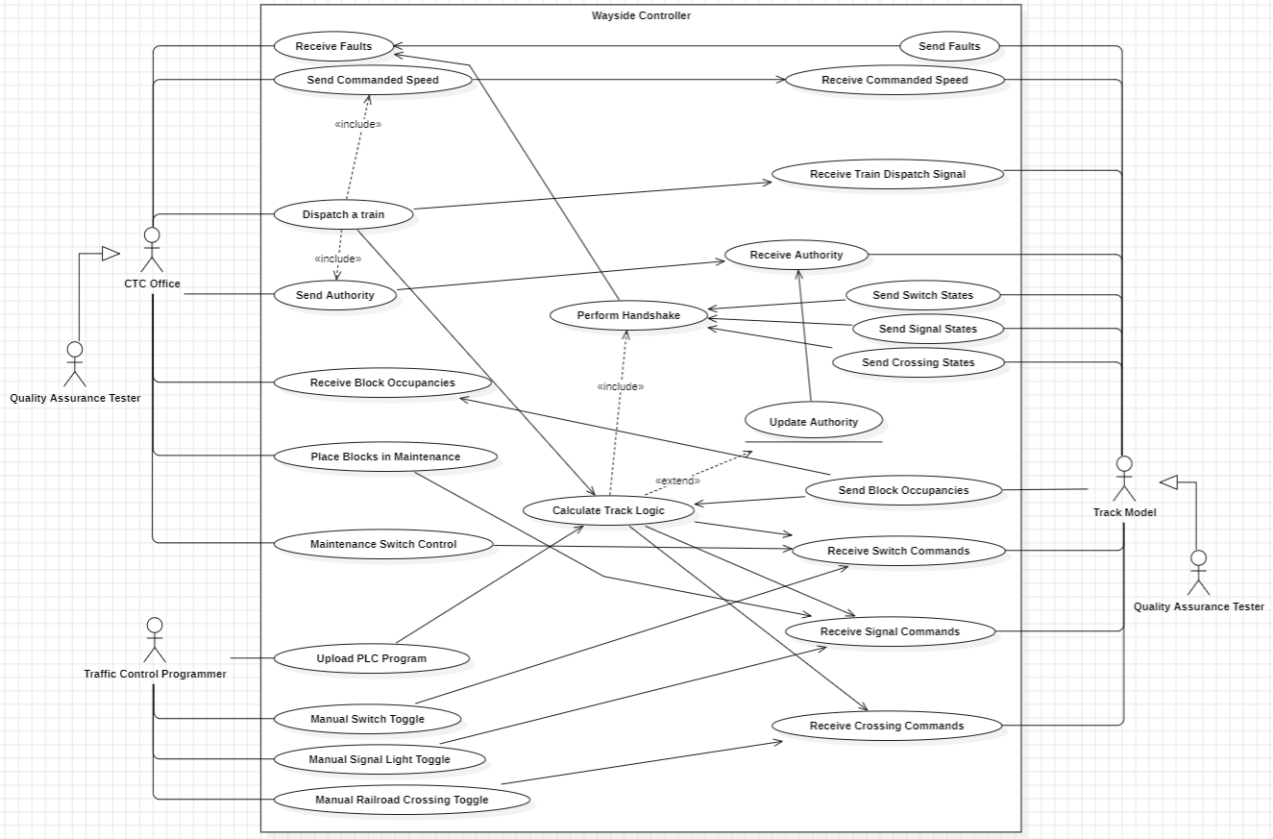

Figure 20: Wayside Controller Software Use Case Diagram

CTC dispatches a Train:

CTC dispatches a Train
User Story: As a CTC Office, I want to dispatch a train so that passengers may use it.
Acceptance Criteria: Given that CTC Office wants to dispatch a train, when the CTC Offices sends this information the Wayside Controller API, the Wayside Controller will update its internal data, update switch/signal/crossing values, update the trains authority if need be, and relay the dispatch signal to the track model.

CTC dispatches a Train	
Actors	CTC Office
Preconditions	CTC needs to dispatch a train
Postconditions	The Wayside Controller's internal train dict is updated, the switches/signals/crossings are updated, and a dispatch command is sent to the Track Model
Normal Flow	1. Wayside updates its internal train dict 2. Wayside calls its subcontroller to perform track logic 3. Wayside sends updated switches/signals/crossings to Track Model 4. Wayside sends dispatch command to Track Model
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

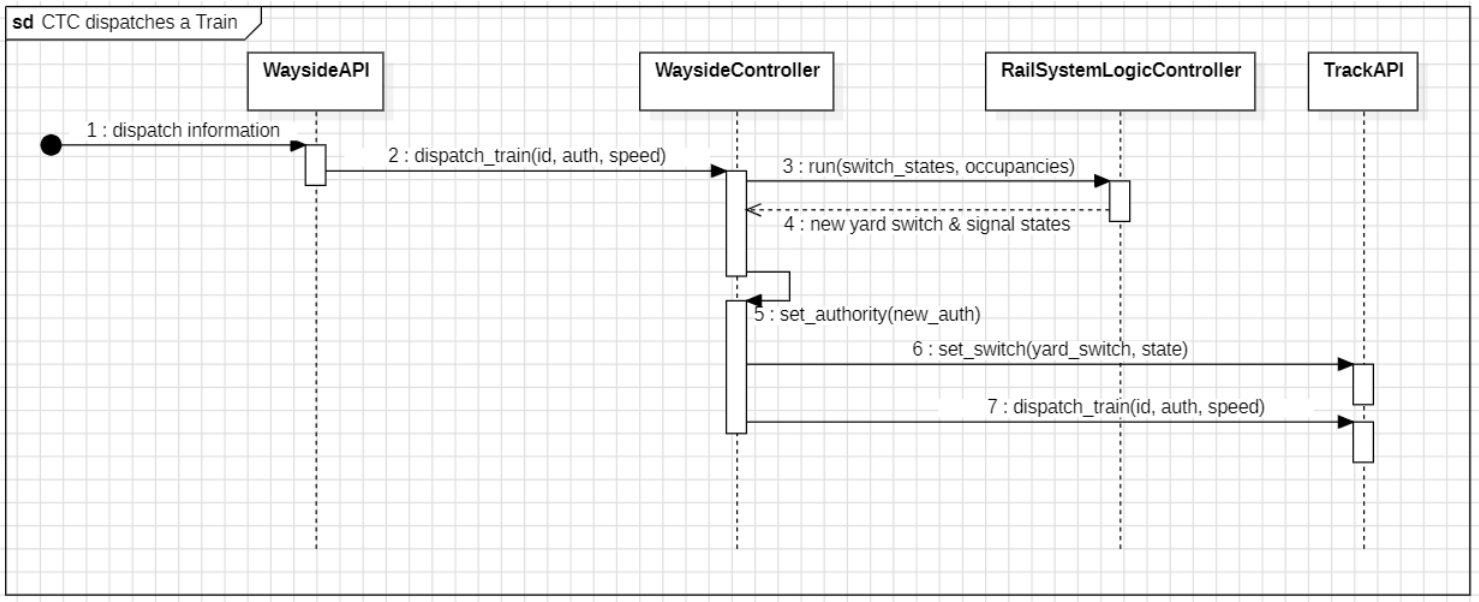

Figure 21: Sequence Diagram for "CTC dispatches a Train"

CTC sends a Suggested Speed:

CTC sends a Suggested Speed
User Story: As a CTC Office, I want to assign a suggested speed to a train so the train can know how fast it should move.
Acceptance Criteria: Given that CTC Office needs to change the suggested speed, when the CTC Office sends the speed value and block ID to the Wayside Controller API, the Wayside Controller relay these items to the Track Model.

CTC sends a Suggested Speed	
Actors	CTC Office
Preconditions	CTC has a need to set or change the suggested speed of a train
Postconditions	The speed value and associated train ID are sent to the Track Model
Normal Flow	1. Wayside Controller sends the train ID and speed to the Track Model
Exceptional Flow	The Track Model does not receive the suggested speed for the correct train or at all

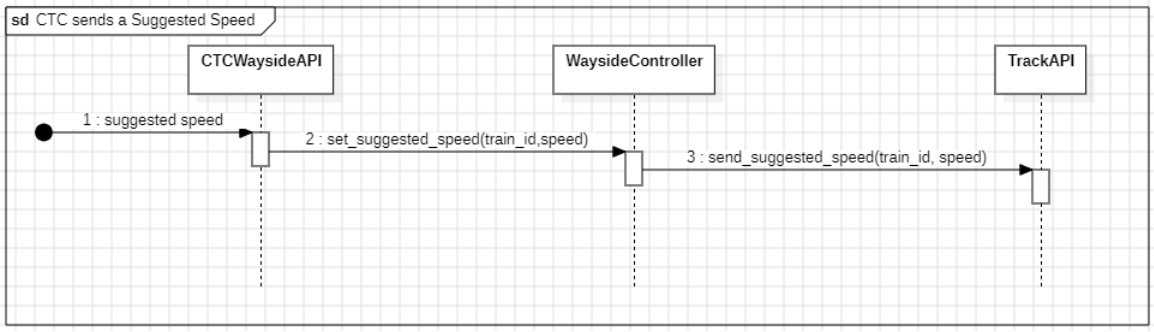

Figure 22: Sequence Diagram for ”CTC sends a Suggested Speed”

CTC sends an Authority:

CTC sends an Authority
User Story: As a CTC Office, I want to assign an authority to a train so the train can know how far it should go.
Acceptance Criteria: Given that CTC Office needs to change the authority, when the CTC Office sends the authority value and block ID to the Wayside Controller API, the Wayside Controller relay these items to the Track Model.

CTC sends a Suggested Speed	
Actors	CTC Office
Preconditions	CTC has a need to set or change the authority of a train at a block
Postconditions	The authority and associated train ID are sent to the Track Model
Normal Flow	1. Wayside Controller sends the train ID and authority to the Track Model
Exceptional Flow	The Track Model does not receive the authority for the correct block or at all

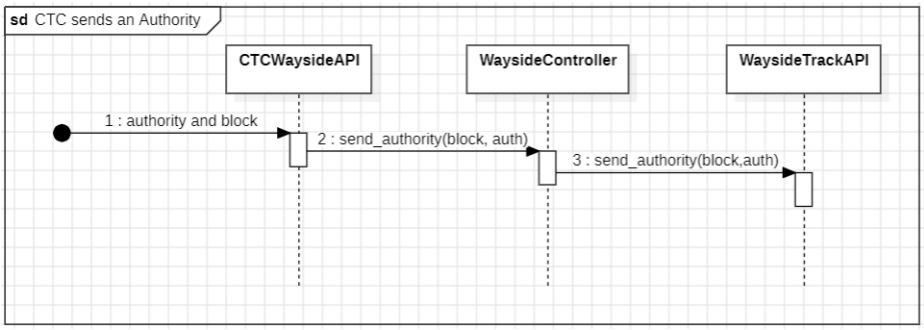

Figure 23: Sequence Diagram for ”CTC sends an Authority”

CTC places a Block in Maintenance:

CTC places a Block in Maintenance
User Story: As a CTC Office, I want to place a block under maintenance in order to repair faulty tracks and maintain the rail system.
Acceptance Criteria: Given that the CTC Office wants to place a block in maintenance, when the CTC Office sends the signal to the Wayside Controller API, then the Wayside Controller will relinquish switch control in that block (if applicable) and override the correct signals to prevent trains from entering the block(s) placed in maintenance.

CTC places a Block in Maintenance	
Actors	CTC Office
Preconditions	CTC wants to place a block in maintenance
Postconditions	Signals are set to prevent entry into maintenance blocks and switch control is given to CTC
Normal Flow	1. Wayside Controller sets the correct signals to red 2. Wayside Controller enables CTC to control the switch
Exceptional Flow	The correct signals are not set to red and/or the CTC Office cannot control the switch

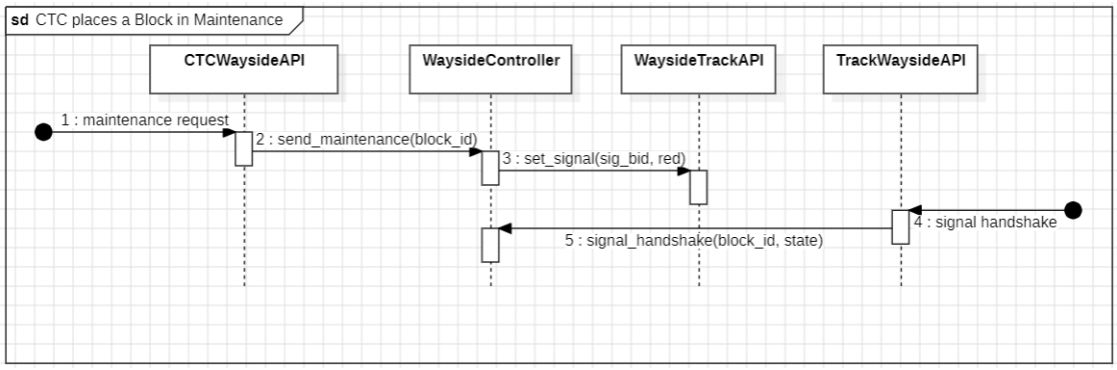

Figure 24: Sequence Diagram for ”CTC places a Block in Maintenance”

CTC controls a Switch:

CTC controls a Switch
User Story: As a CTC Office, I want to manually control a track switch to verify its functionality.
Acceptance Criteria: Given that the CTC Office want to set a switch, when the CTC Office sends the switch command to the Wayside Controller API, the Wayside Controller will set the corresponding switch.

CTC controls a Switch	
Actors	CTC Office
Preconditions	CTC wants to change a switch
Postconditions	Switch is set to CTC's desired state
Normal Flow	1. Wayside Controller sets the switch to the requested state
Exceptional Flow	The Wayside Controller fails to send the correct signal or the Track Model handshake fails

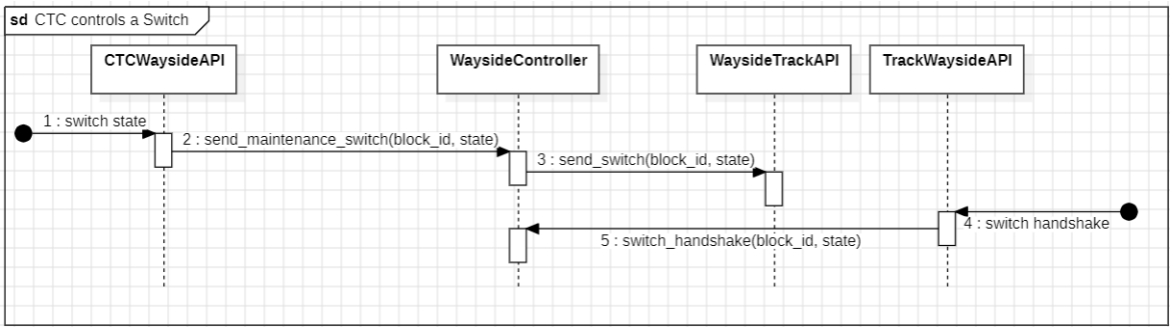

Figure 25: Sequence Diagram for "CTC controls a Switch"

Track Model sends an Occupancy:

Track Model sends an Occupancy	
User Story: As a Track Model, I want to send the Wayside Controller whether a block is occupied so it may perform track logic and relay the occupancy to CTC.	
	Acceptance Criteria: Given that an occupancy has changed in the Track Model, when the Track Model sends the occupancy to the Wayside Controller API, then the Wayside Controller will run its PLC program to compute new switch/signal/crossing states, send the new states to Track Model, and send the new occupancy to CTC.

Track Model sends an Occupancy	
Actors	Track Model
Preconditions	The Track Model detected a change in block occupancy
Postconditions	The switches, signals, and crossings are updated and the occupancy is sent to CTC
Normal Flow	1. Wayside Controller sends the new occupancy to CTC 2. Wayside Controller runs its PLC program with the new data 3. Wayside Controller sends the new states to Track Model
Exceptional Flow	CTC fails to receive occupancy or PLC program fails

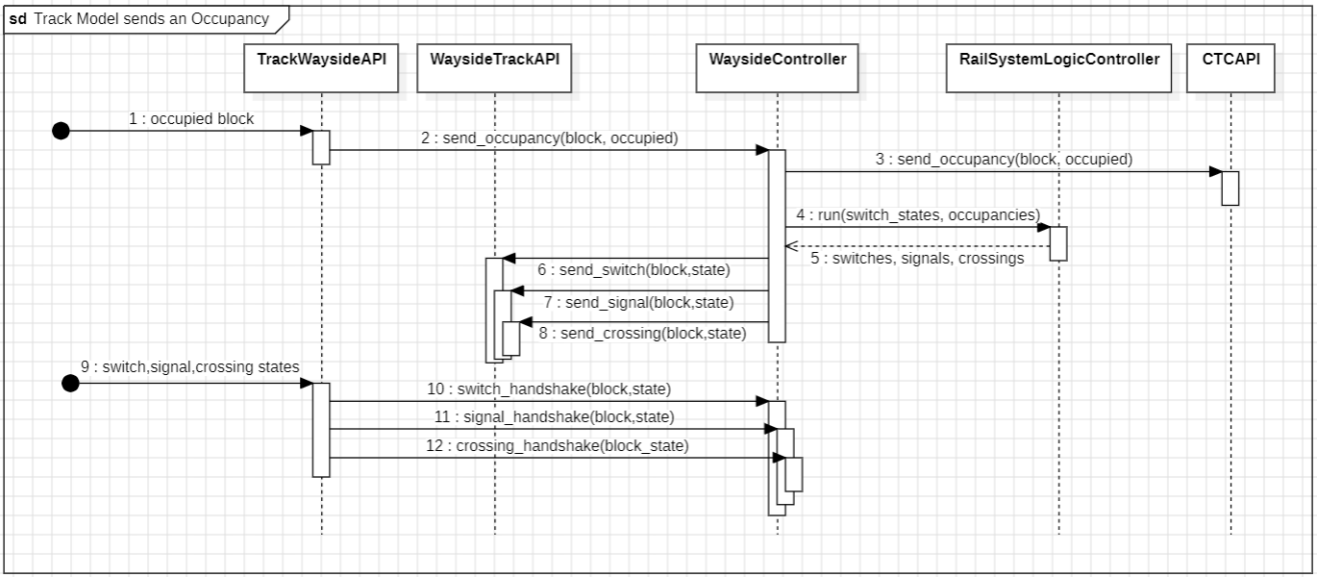

Figure 26: Sequence Diagram for ”Track Model sends an Occupancy”

Track Model sends a Fault:

Track Model sends a Fault
User Story: As a Track Model, I want to report a failure to the Wayside Controller and CTC Office so they may perform the proper actions to ensure safety.
Acceptance Criteria: Given that a failure condition is met within the Track Model, when the Track Model sends the fault information to the Wayside Controller API, the Wayside Controller will notify its user and relay the fault to the CTC Office.

Track Model sends a Fault	
Actors	Track Model
Preconditions	Failure condition is met within Track Model
Postconditions	Wayside user is notified and fault is relayed to CTC Office
Normal Flow	1. Wayside Controller informs its user of the fault 2. Wayside Controller sends the fault to CTC Office
Exceptional Flow	Either the fault is not displayed to the user, the CTC Office is not informed, or both occur

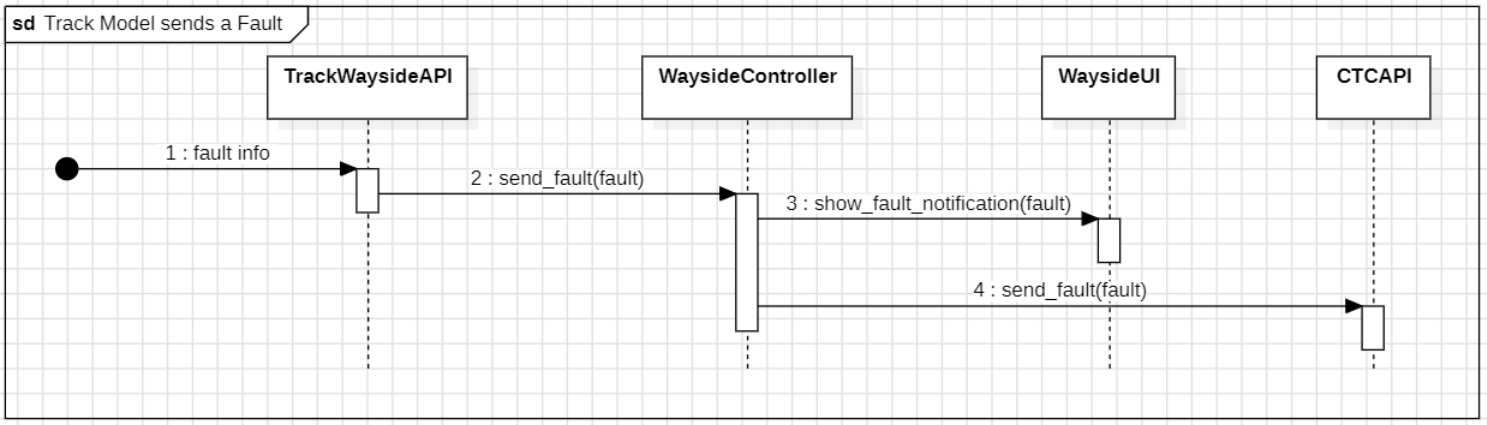

Figure 27: Sequence Diagram for ”Track Model sends a Fault”

Track Model sends Switch Handshake:

Track Model sends Switch Handshake
User Story: As a Track Model, I want to send my switch state back to the Wayside Controller so that it may verify that the switch flipped properly.
Acceptance Criteria: Given that the Track Model received a switch command from Wayside Controller, when the Track Model sends the new switch value back, the Wayside Controller will verify that received value is the same as the commanded state.

Track Model sends Switch Handshake	
Actors	Track Model
Preconditions	Track Model received a switch command
Postconditions	A fault is sent to the CTC Office if switch is not set to the correct value
Normal Flow	1. Wayside Controller compares received switch state to commanded switch state and comparison succeeds
Exceptional Flow	1a. Wayside Controller compares received switch state to commanded switch state and comparison fails 1b. Wayside Controller does not receive handshake signal 2. Wayside Controller sends a switch control fault to CTC Office

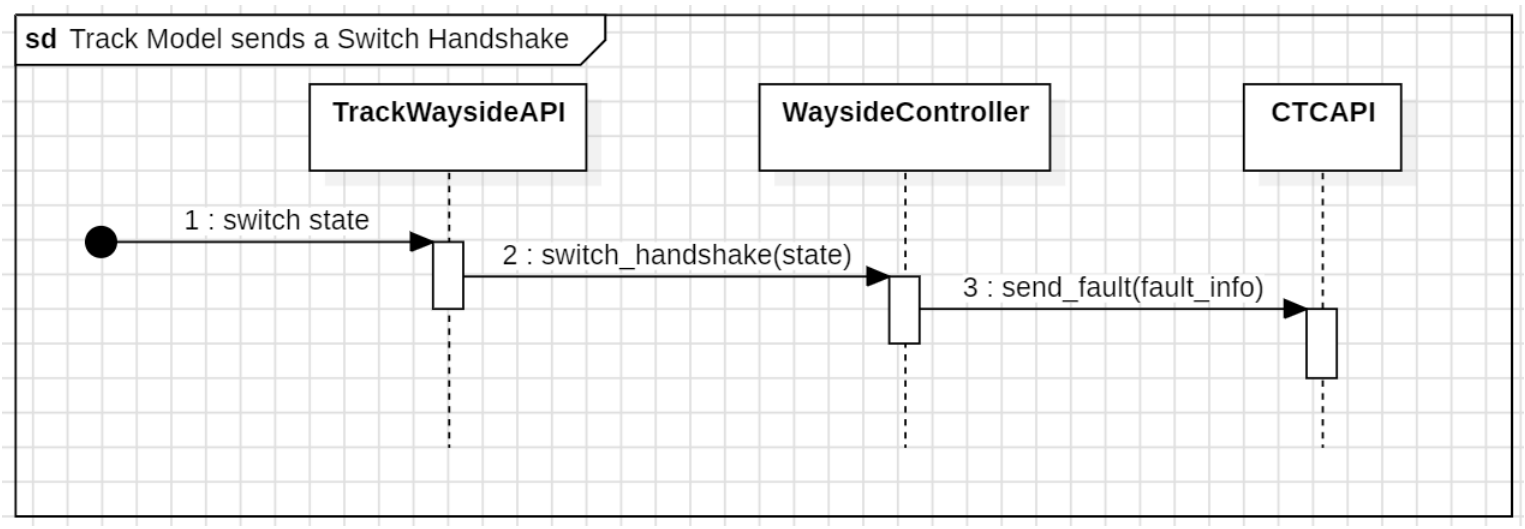

Figure 28: Sequence Diagram for ”Track Model sends a Switch Handshake”

Track Model sends Signal Handshake:

Track Model sends Signal Handshake
User Story: As a Track Model, I want to send my signal state back to the Wayside Controller so that it may verify that the signal toggled properly.
Acceptance Criteria: Given that the Track Model received a signal command from Wayside Controller, when the Track Model sends the new signal value back, the Wayside Controller will verify that received value is the same as the commanded state.

Track Model sends Signal Handshake	
Actors	Track Model
Preconditions	Track Model received a signal command
Postconditions	A fault is sent to the CTC Office if signal is not set to the correct value
Normal Flow	1. Wayside Controller compares received signal state to commanded signal state and comparison succeeds
Exceptional Flow	1a. Wayside Controller compares received signal state to commanded signal state and comparison fails 1b. Wayside Controller does not receive handshake signal 2. Wayside Controller sends a signal control fault to CTC Office

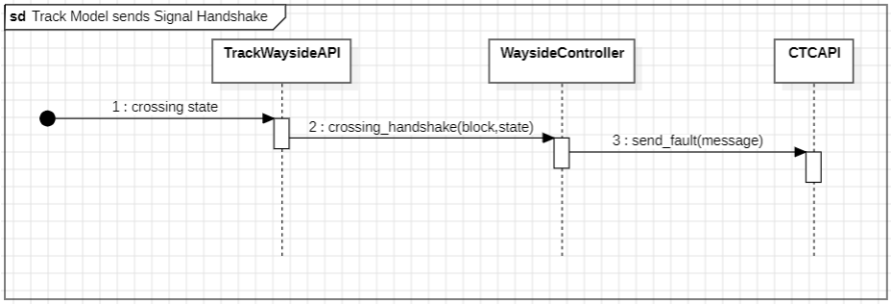

Figure 29: Sequence Diagram for ”Track Model sends a Signal Handshake”

Track Model sends Crossing Handshake:

Track Model sends Crossing Handshake
User Story: As a Track Model, I want to send my crossing state back to the Wayside Controller so that it may verify that the crossing toggled properly.
Acceptance Criteria: Given that the Track Model received a crossing command from Wayside Controller, when the Track Model sends the new crossing value back, the Wayside Controller will verify that received value is the same as the commanded state.

Track Model sends Crossing Handshake	
Actors	Track Model
Preconditions	Track Model received a crossing command
Postconditions	A fault is sent to the CTC Office if crossing is not set to the correct value
Normal Flow	1. Wayside Controller compares received crossing state to commanded crossing state and comparison succeeds
Exceptional Flow	1a. Wayside Controller compares received crossing state to commanded crossing state and comparison fails 1b. Wayside Controller does not receive handshake signal 2. Wayside Controller sends a crossing control fault to CTC Office

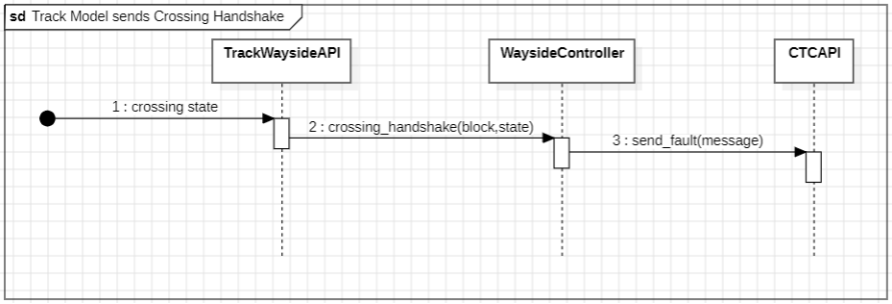

Figure 30: Sequence Diagram for ”Track Model sends a Crossing Handshake”

Wayside Programmer uploads PLC Code:

Wayside Programmer uploads PLC Code
User Story: As a Wayside Programmer, I would like to upload a new PLC program to a wayside controller so that I can modify how it performs track logic.
Acceptance Criteria: Given that the Wayside Programmer needs to change how track logic is controlled, when they input the file name and click upload, then the Wayside Controller will overwrite its current logic program.

Wayside Programmer uploads PLC Code	
Actors	Wayside Programmer
Preconditions	Wayside Programmer needs to upload new logic program
Postconditions	The Wayside Controller is prepared to run the new PLC Program.
Normal Flow	1. Wayside Controller receives path from UI 2. Wayside Controller sends path to logic controller 3. Logic controller sends program as string to interpreter 4. Interpreter tokenizes the file string
Exceptional Flow	The exceptional flow is the same as normal flow except if the file path is invalid or the PLC code is invalid.

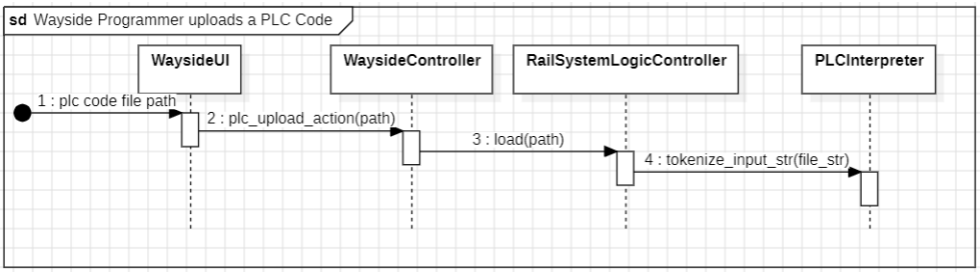

Figure 31: Sequence Diagram for "Wayside Programmer uploads PLC Code"

Wayside Programmer sets a Switch:

Wayside Programmer sets a Switch
User Story: As a Wayside Programmer, I want to set a switch to manually control trains' paths.
Acceptance Criteria: Given that the Wayside Programmer needs to manually set a switch and that the Wayside Controller is in manual mode, when the Wayside Programmer clicks the toggle switch button, then the Wayside Controller will set the requested switch to its alternative state.

Wayside Programmer sets a Switch	
Actors	Wayside Programmer
Preconditions	Wayside Programmer needs to set a switch and Wayside Controller is in manual mode
Postconditions	The Wayside Controller sets the switch and sends the new state to Track Model
Normal Flow	1. Wayside Controller sets switch to new state 2. Wayside Controller sends new state to Track Model 3. Wayside Controller receives handshake
Exceptional Flow	Wayside Controller handshake fails

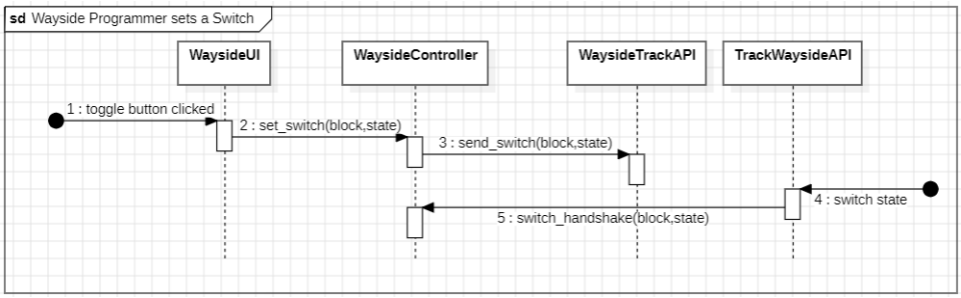

Figure 32: Sequence Diagram for ”Wayside Programmer sets a Switch”

Wayside Programmer sets a Signal:

Wayside Programmer sets a Signal
User Story: As a Wayside Programmer, I want to set a signal to manually stop/start trains.
Acceptance Criteria: Given that the Wayside Programmer needs to manually set a signal and that the Wayside Controller is in manual mode, when the Wayside Programmer clicks the toggle signal button, then the Wayside Controller will set the requested signal to its alternative state.

Wayside Programmer sets a Signal	
Actors	Wayside Programmer
Preconditions	Wayside Programmer needs to set a signal and Wayside Controller is in manual mode
Postconditions	The Wayside Controller sets the signal and sends the new state to Track Model
Normal Flow	1. Wayside Controller sets signal to new state 2. Wayside Controller sends new state to Track Model 3. Wayside Controller receives handshake
Exceptional Flow	Wayside Controller handshake fails

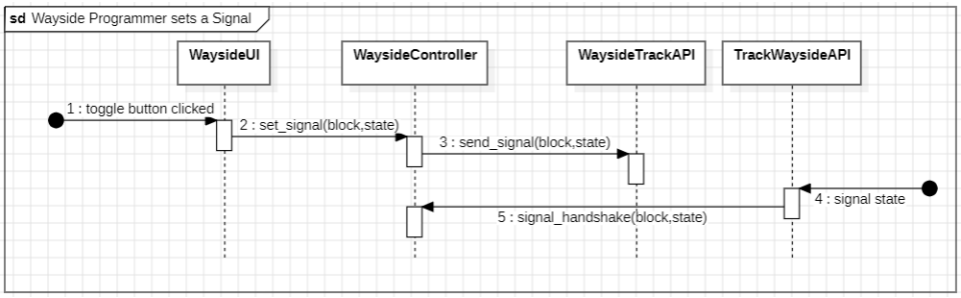

Figure 33: Sequence Diagram for "Wayside Programmer sets a Signal"

Wayside Programmer sets a Crossing:

Wayside Programmer sets a Crossing
User Story: As a Wayside Programmer, I want to set a crossing to manually control cars crossing the track.
Acceptance Criteria: Given that the Wayside Programmer needs to manually set a crossing and that the Wayside Controller is in manual mode, when the Wayside Programmer clicks the toggle crossing button, then the Wayside Controller will set the requested crossing to its alternative state.

Wayside Programmer sets a Crossing	
Actors	Wayside Programmer
Preconditions	Wayside Programmer needs to set a crossing and Wayside Controller is in manual mode
Postconditions	The Wayside Controller sets the crossing and sends the new state to Track Model
Normal Flow	1. Wayside Controller sets crossing to new state 2. Wayside Controller sends new state to Track Model 3. Wayside Controller receives handshake
Exceptional Flow	Wayside Controller handshake fails

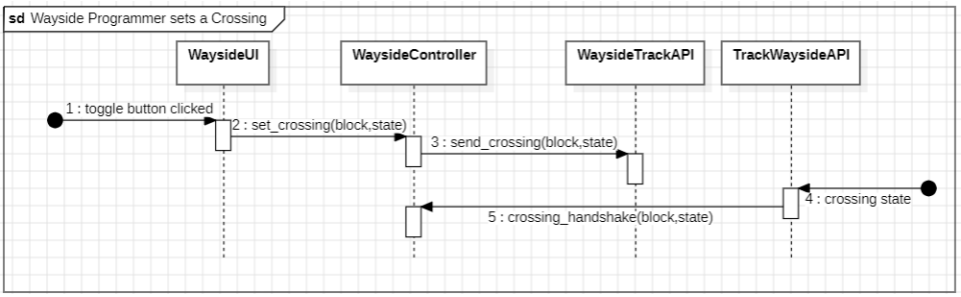

Figure 34: Sequence Diagram for "Wayside Programmer sets a Crossing"

3.3 Wayside Controller Hardware

3.3.1 Design Overview

wayside_software_wrapper	
Description: Software wrapper class to link system's python source code to ESP-32 Microcontroller.	
Attributes	
Name	Description
node_id	Identifies which wayside controller node this instance corresponds to
signals	Identifies which signals shall be publishedsubscribed via the MQTT Broker
switches	Identifies which switches shall be publishedsubscribed via the MQTT Broker
blocks	Identifies which blocks shall be publishedsubscribed via the MQTT Broker
trains	Identifies which trains shall be publishedsubscribed via the MQTT Broker
mqtt_broker	Identifies the MQTT broker object
mqtt_port	Identifies the port for which the MQTT Broker will listening
client	Identifies the client to be used to publishsubscribe to MQTT Broker
suggested_speed	Identifies the suggested speed to be published to the MQTT Broker
authority	Identifies the authority to be published to the MQTT Broker
Responsibilities	
Name	Collaborator
switches	waysideControllerOne, waysideControllerTwo, Track Model
signals	waysideControllerOne, waysideControllerTwo, Track Model
set_blocks	waysideControllerOne, waysideControllerTwo, Track Model
publish_blocks	Self
publish_trains	Self
publish_switches	Self
publish_signals	Self
connect_to_mqtt	Self
on_message	Self
on_connect	Self
add_train	CTC Office
Set Maintenance	CTC Office
Set Authority	Self, CTC Office
send_suggested_speed	CTC Office
send_authority	CTC Office

waysideControllerOne	
Description: Script to be uploaded to ESP32 Microprocessor for user to view track states and control them.	
Attributes	
Name	Description
waysideNodeID	Identifies the wayside node that will be adressed
railLine	Identifies the which line is being adressed
blocksData	Identifies the block data subscribed to via the MQTT Broker
switchesData	Identifies the switch data subscribed to via the MQTT Broker
signalsData	Identifies the signal data subscribed to via the MQTT Broker
crossingsData	Identifies the crossing data subscribed to via the MQTT Broker
trainsData	Identifies the train data subscribed to via the MQTT Broker
mqttClient	Identifies the MQTT Client to subscribe/publish track data
attribute	discription
Responsibilities	
Name	Collaborator
sendDataToWrapper	Self
receiveDataFromWrapper	Self
setup	Self

waysideControllerTwo	
Description: Script to be uploaded to ESP32 Microprocessor for user to view track states and control them.	
Attributes	
Name	Description
waysideNodeID	Identifies the wayside node that will be adressed
railLine	Identifies the which line is being adressed
blocksData	Identifies the block data subscribed to via the MQTT Broker
switchesData	Identifies the switch data subscribed to via the MQTT Broker
signalsData	Identifies the signal data subscribed to via the MQTT Broker
crossingsData	Identifies the crossing data subscribed to via the MQTT Broker
trainsData	Identifies the train data subscribed to via the MQTT Broker
mqttClient	Identifies the MQTT Client to subscribe/publish track data
attribute	discription
Responsibilities	
Name	Collaborator
sendDataToWrapper	Self
receiveDataFromWrapper	Self
setup	Self

BlueLine	
Description: BlueLine class responsible for keeping track of, and organizing track data on the blue line for testing purposes only	
Attributes	
section	Identifies sections of the blue line
blockCount	Identifies number of blocks in each section
blueSwitchID	Identifies IDs of switches on the line
blockIdList	List of all block IDs on the line
switchOwnerBlock	Identifies common block of switch connection
switchPrimaryBlock	Identifies primary block of switch connection
switchSecondaryBlock	Identifies secondary block of switch connection
blueSections	Accessor for connected controllers
blueLineSwitch	Accessor for connected controllers
Responsibilities	
Name	Collaborator
BlueLine	TestController

GreenLine	
Description: GreenLine class responsible for keeping track of, and organizing track data on the green line	
Attributes	
section	Identifies sections of the green line
blockCount	Identifies number of blocks in each section
greenSwitchID	Identifies IDs of switches on the line
blockIdList	List of all block IDs on the line
switchOwnerBlock	Identifies common block of switch connection
switchPrimaryBlock	Identifies primary block of switch connection
switchSecondaryBlock	Identifies secondary block of switch connection
greenSections	Accessor for connected controllers
greenLineSwitches	Accessor for connected controllers
greenNodes	Identifies node IDs on the line
greenCrossing	Identifies crossing on the line
Responsibilities	
Name	Collaborator
GreenLine	waysideControllerTwo

RedLine	
Description: RedLine class responsible for keeping track of, and organizing track data on the red line	
Attributes	
section	Identifies sections of the red line
blockCount	Identifies number of blocks in each section
redSwitchID	Identifies IDs of switches on the line
blockIdList	List of all block IDs on the line
switchOwnerBlock	Identifies common block of switch connection
switchPrimaryBlock	Identifies primary block of switch connection
switchSecondaryBlock	Identifies secondary block of switch connection
redSections	Accessor for connected controllers
redLineSwitches	Accessor for connected controllers
redNodes	Identifies node IDs on the line
redCrossing	Identifies crossing on the line
Responsibilities	
Name	Collaborator
RedLine	waysideControllerOne

PLC	
Description: Interpreter class responsible for loading, verifying, and executing the PLC program.	
Attributes	
nodeID	Identifies Wayside Node to upload PLC code.
Responsibilities	
Name	Collaborator

Block	
Description: Storage class responsible for storing all relevant information about a block. This includes static and dynamic information.	
Attributes	
Name	Description
sectionID	Identifies what section the block is in
blockID	Idendifies the ID of a block
isOccupied	Identifies current state of a block
Responsibilities	
Name	Collaborator
getBlockState	BlueLine, RedLine, GreenLine
setBlockState	BlueLine, RedLine, GreenLine

Switch	
Description: Switch class responsible for storing a switch's primary block, secondary block, and current state.	
Attributes	
Name	Description
switchID	Identifies ID of corresponding switch
ownerBlock	Identifies owner block of switch
priamryBlock	Identifies primary block of switch
secondaryBlock	Identifies secondary block of switch
switchState	Identifies current state of switch
Responsibilities	
Name	Collaborator
setSwitchLocation	BlueLine, RedLine, GreenLine
getOwnerBlock	BlueLine, RedLine, GreenLine
getPrimaryBlock	BlueLine, RedLine, GreenLine
getSecondaryBlock	BlueLine, RedLine, GreenLine
setSwitchState	BlueLine, RedLine, GreenLine
getSwitchState	BlueLine, RedLine, GreenLine

Signal	
Description: Signal class responsible for storing a signal state.	
Attributes	
Name	Description
signalID	Idendifies signal, ID should match BlockID
signalState	Idendifies signal state, state should match block state
Responsibilities	
Name	Collaborator
getSignalState	Block
setSignalState	Block

Crossing	
Description: Storage class responsible for storing a crossing state.	
Attributes	
Name	Description
crossingID	Identifies ID of crossing
crossingState	Identifies current state of crossing
crossingBufferBlocks	Identifies list of blocks surrounding crossing
Responsibilities	
Name	Collaborator
setCrossingState	RedLine, BlueLine, GreenLine
getCrossingState	RedLine, BlueLine, GreenLine
getCrossingLocation	RedLine, BlueLine, GreenLine

Train	
Description: Storage class responsible for storing all relevant information about a train. This includes static and dynamic information.	
Attributes	
Name	Description
Train ID	Stores train's ID number
Responsibilities	
Name	Collaborator

3.3.2 Class Diagram

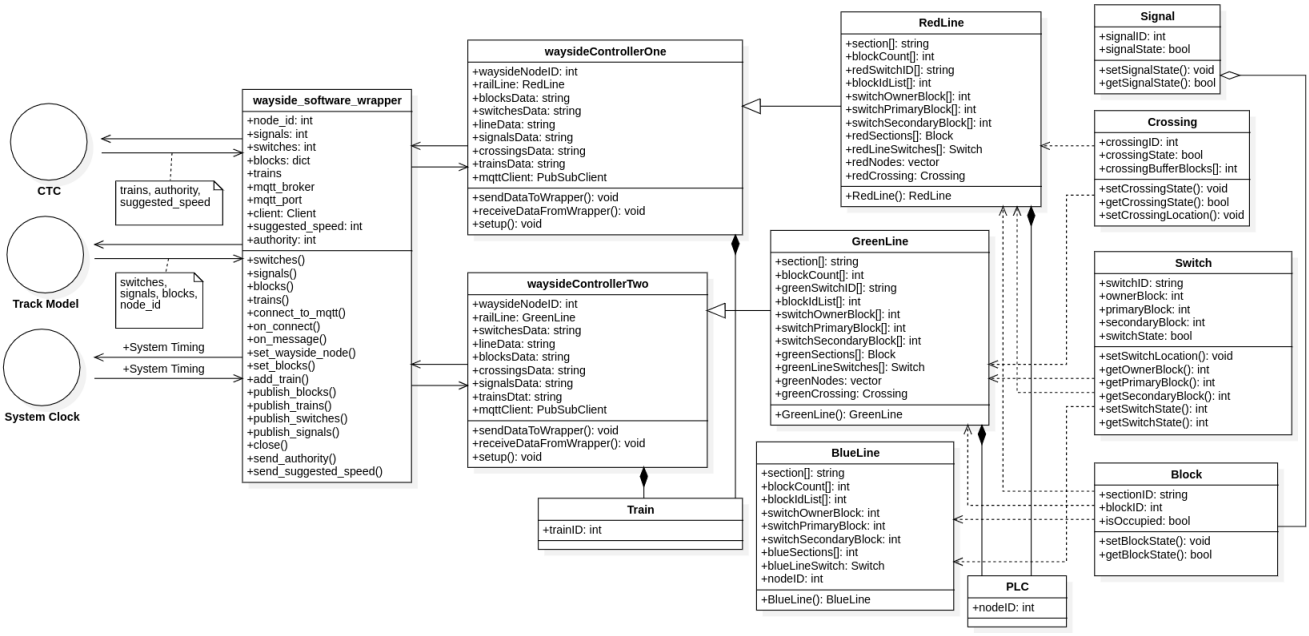

Figure 35: Wayside Controller Hardware Class Diagram

3.3.3 Use Cases

The primary actors involved in the wayside controller module's use cases are the CTC office, track model, and traffic control programmer. Additionally, a fourth actor, referred to as the quality assurance tester, has been included. In practice, the quality assurance tester is typically a member of the Aurora team responsible for independently testing the wayside controller hardware using its physical UI interface. This actor serves as a substitute for the CTC office and track model during testing. The use case diagram for the wayside controller is presented in Figure 6, followed by a user story, use case description, and sequence diagram for each significant use case.

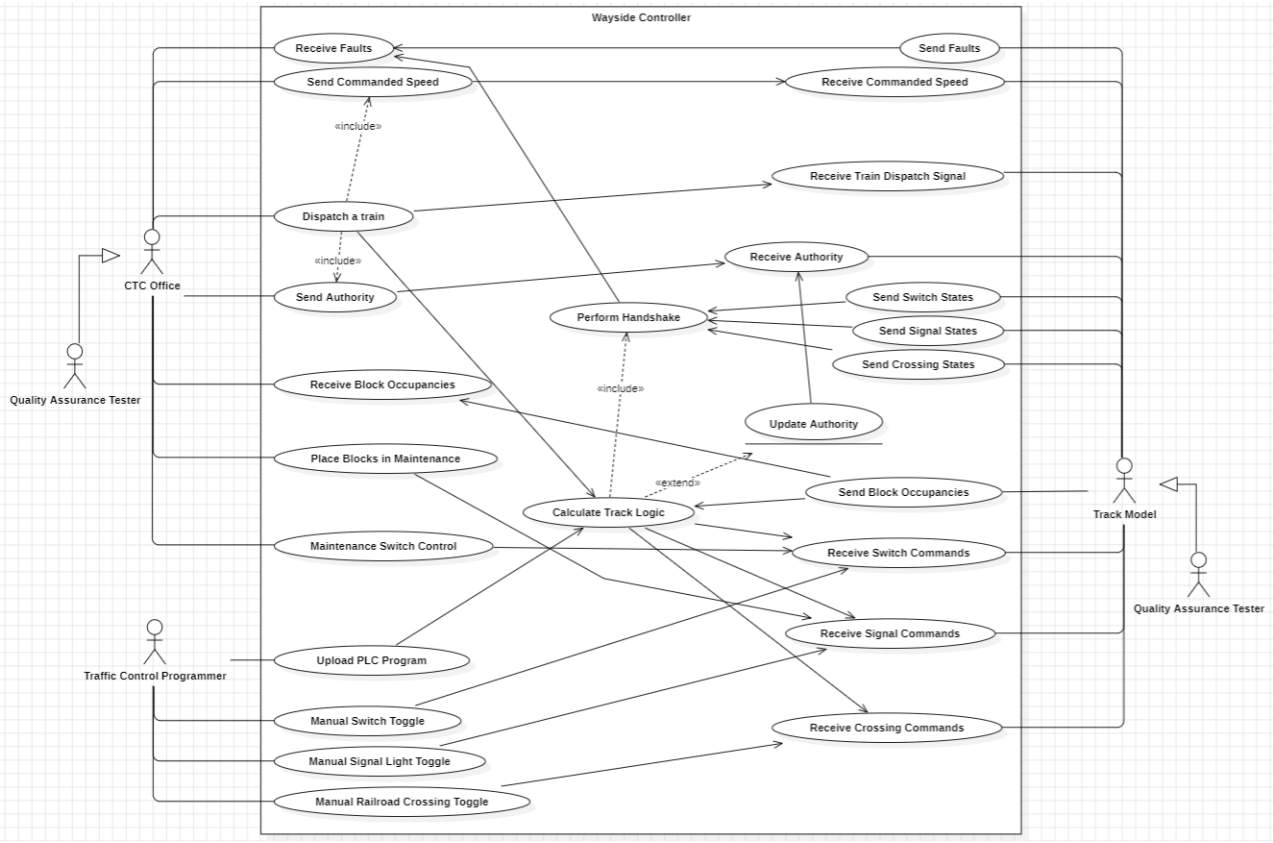

Figure 36: Wayside Controller Software Use Case Diagram

CTC dispatches a Train:

CTC dispatches a Train
User Story: As a CTC Office, I want to dispatch a train so that passengers may use it.
Acceptance Criteria: Given that CTC Office wants to dispatch a train, when the CTC Offices sends this information the Wayside Controller API, the Wayside Controller will update its internal data, update switch/signal/crossing values, update the trains authority if need be, and relay the dispatch signal to the track model.

CTC dispatches a Train	
Actors	CTC Office
Preconditions	CTC needs to dispatch a train, and connection to MQTT Broker is established.
Postconditions	Wayside Controller's internal train vector is updated, the switches/signals/crossings are updated, and a dispatch command is sent to the Track Model
Normal Flow	1. Software Wrapper publishes to MQTT Broker dispatch train topic 2. Wayside Controller subscribes to dispatch train topic 3. Wayside calls its PLC Logic Program to perform track logic and publishes to Logic topic 4. Wayside publishes to MQTT Broker dispatch train topic 5. Software Wrapper subscribes to Logic topic and sends updated switches/signals/crossings to Track Model 6. Software Wrapper sends dispatch command to Track Model
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

CTC Dispatches a Train

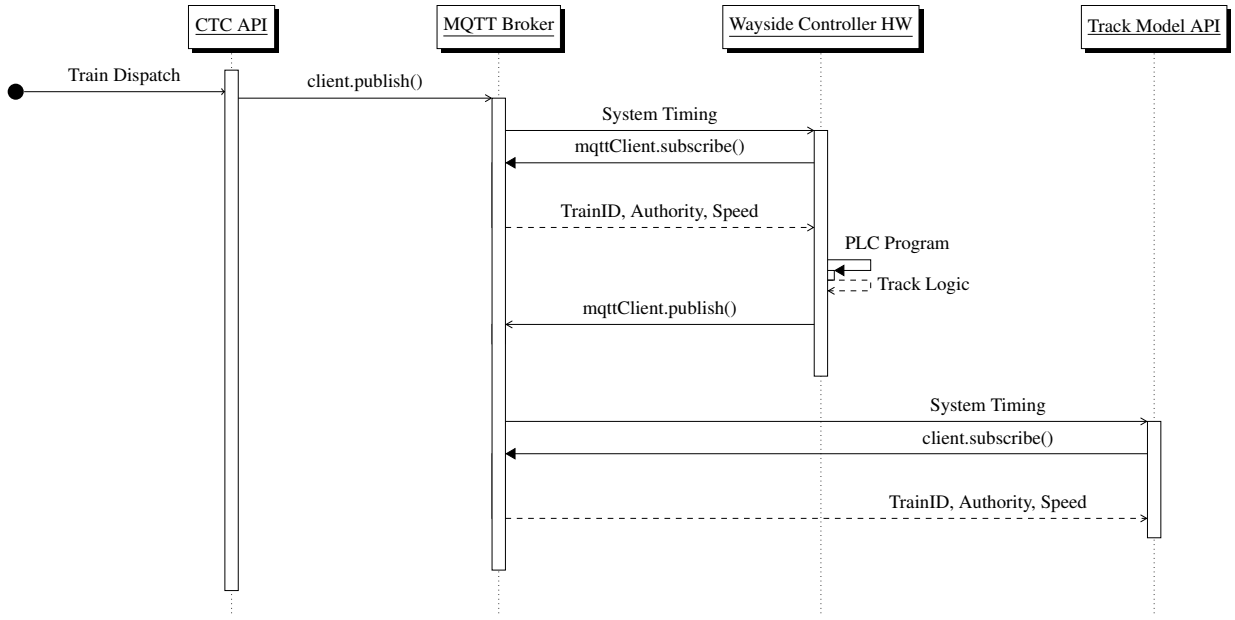

Figure 37: Sequence Diagram for "CTC dispatches a Train"

CTC sends a Suggested Speed:

CTC sends a Suggested Speed
User Story: In my role as the CTC Office, my objective is to allocate a recommended speed to a train, enabling the train to determine its optimal speed for operation.
Acceptance Criteria: In the context of the CTC Office requiring a speed adjustment, the process unfolds as follows: when the CTC Office transmits the speed value and block ID to the Wayside Controller API, the Wayside Controller subsequently conveys this information to the Track Model.

CTC sends a Suggested Speed	
Actors	CTC Office
Preconditions	CTC has a need to set or change the suggested speed of a train
Postconditions	The speed value and associated train ID are published and sent to the Track Model
Normal Flow	Wayside Software Wrapper sends the train ID and speed to the Track Model
Exceptional Flow	The Track Model does not receive the suggested speed for the correct train or at all

CTC Sends Suggested Speed

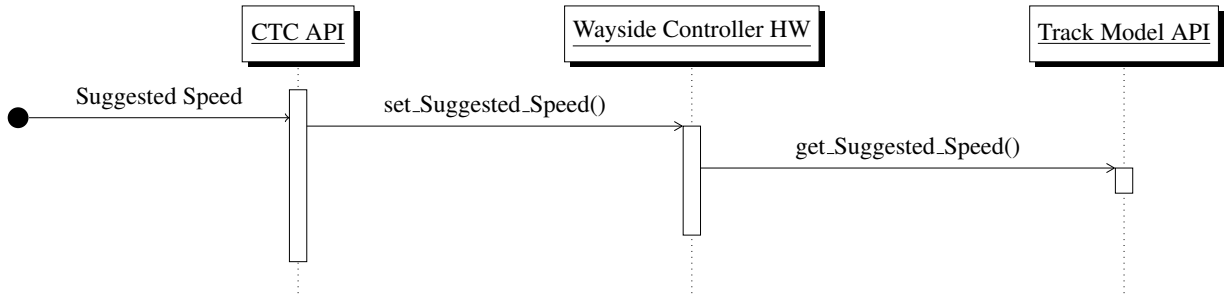

Figure 38: Sequence Diagram for ”CTC sends a Suggested Speed”

CTC sends an Authority:

CTC sends an Authority
User Story: In my capacity as the CTC Office, my goal is to designate an authorization for a train, providing the train with the necessary information regarding the distance it should travel.
Acceptance Criteria: In the event that the CTC Office requires an alteration in authority, the process transpires as follows: when the CTC Office transmits both the authority value and the block ID to the Wayside Controller API, the Wayside Controller proceeds to forward these specific elements to the Track Model.

CTC sends a Suggested Speed	
Actors	CTC Office
Preconditions	CTC requires the capability to establish or modify a train's authorization within a specific block.
Postconditions	The authority, along with the corresponding train ID, is forwarded to the Track Model.
Normal Flow	1. Wayside Software Wrapper sends the train ID and authority to the Track Model
Exceptional Flow	However, the Track Model may not receive the authorization for the intended block, or it may not receive it at all.

CTC Sends Authority

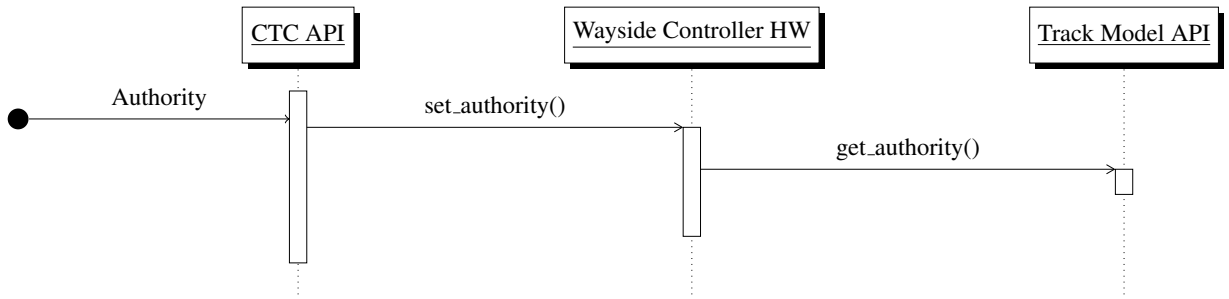

Figure 39: Sequence Diagram for ”CTC sends an Authority”

CTC places a Block in Maintenance:

CTC places a Block in Maintenance
User Story: In my role as the CTC Office, my objective is to designate a block for maintenance. This is done to address issues with the tracks and ensure the proper upkeep of the railway system.
Acceptance Criteria: When the CTC Office intends to designate a block for maintenance the process unfolds as follows: upon the CTC Office transmitting the signal to the Wayside Controller API, the Wayside Controller will take appropriate action. It may relinquish switch control in the designated block, if applicable, and override the relevant signals to ensure that trains are prevented from entering the block(s) marked for maintenance.

CTC places a Block in Maintenance	
Actors	CTC Office
Preconditions	CTC wants to place a block in maintenance
Postconditions	Signals are set to prevent entry into maintenance blocks and switch control is given to CTC
Normal Flow	1. The Wayside Controller adjusts the signals to the appropriate red status. 2. The Wayside Controller facilitates CTC in taking control of the switch.
Exceptional Flow	The signals are not configured correctly to display red, and it is possible that the CTC Office is unable to assume control over the switch.

CTC Puts Block into Maintenance

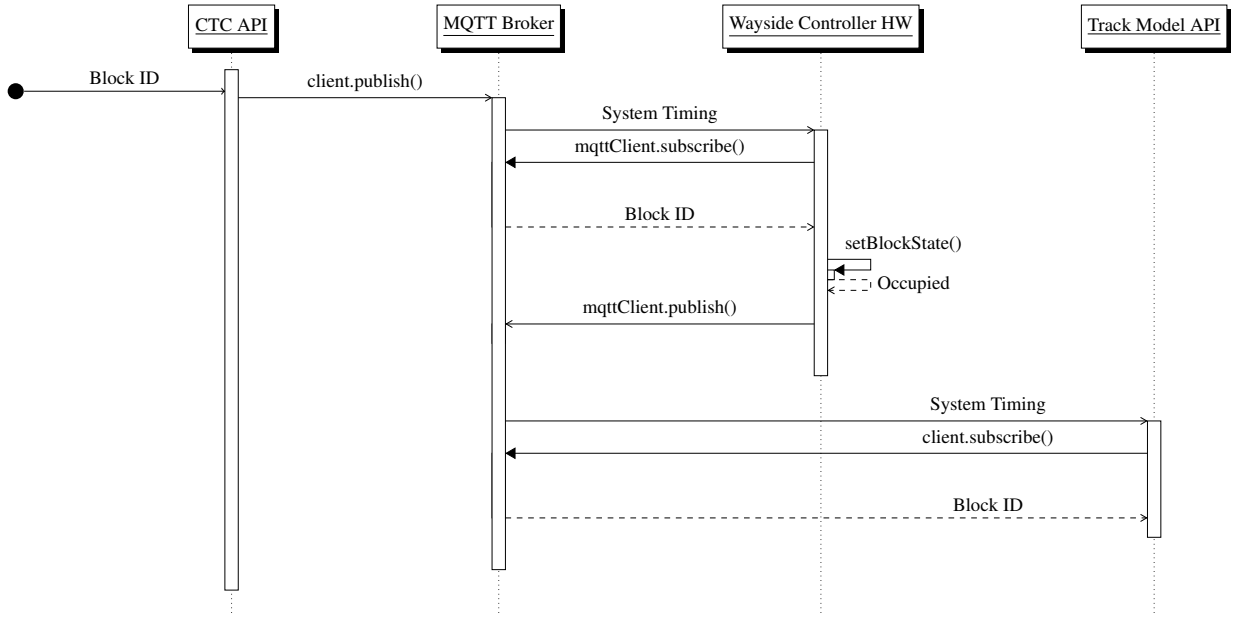

Figure 40: Sequence Diagram for ”CTC places a Block in Maintenance”

CTC controls a Switch:

CTC controls a Switch	
	User Story: As a CTC Office, I want to manually control a track switch to verify its functionality.
Acceptance Criteria: In the scenario where the CTC Office needs to adjust a switch, the process proceeds as follows: upon the CTC Office transmitting the switch command to the Wayside Controller API, the Wayside Controller will execute the necessary action to configure the relevant switch.	

CTC controls a Switch	
Actors	CTC Office
Preconditions	CTC wants to change a switch
Postconditions	Switch is set to CTC’s desired state
Normal Flow	1. Wayside Software Wrapper publishes CTC request to MQTT Broker 1. Wayside Controller subscribes to Switch topic 1. Wayside Controller sets the switch to the requested state
Exceptional Flow	The Wayside Controller fails to send the correct signal or the Track Model handshake fails

Figure 41: Sequence Diagram for ”CTC controls a Switch”

Track Model sends an Occupancy:

Track Model sends an Occupancy
User Story: As a Track Model, I want to send the Wayside Controller whether a block is occupied so it may perform track logic and relay the occupancy to CTC.
Acceptance Criteria: In the event of a modification in occupancy within the Track Model, the process unfolds as follows: when the Track Model transmits the updated occupancy to the Wayside Controller API, the Wayside Controller will initiate its PLC program to calculate new switch, signal, and crossing states. Subsequently, it will relay these updated states to the Track Model and provide the new occupancy information to the CTC.

Track Model sends an Occupancy	
Actors	Track Model
Preconditions	The Track Model detected a change in block occupancy
Postconditions	The switches, signals, and crossings are updated and the occupancy is sent to CTC
Normal Flow	1. Wayside Wrapper subscribes to ocuppancies topic 2. Wayside Software Wrapper publishes the new occupancy to CTC 3. Wayside Controller runs its PLC program with the new data 4. Wayside Controller sends the new states to Track Model
Exceptional Flow	CTC fails to receive occupancy or PLC program fails

Track Model Sends Occupancies

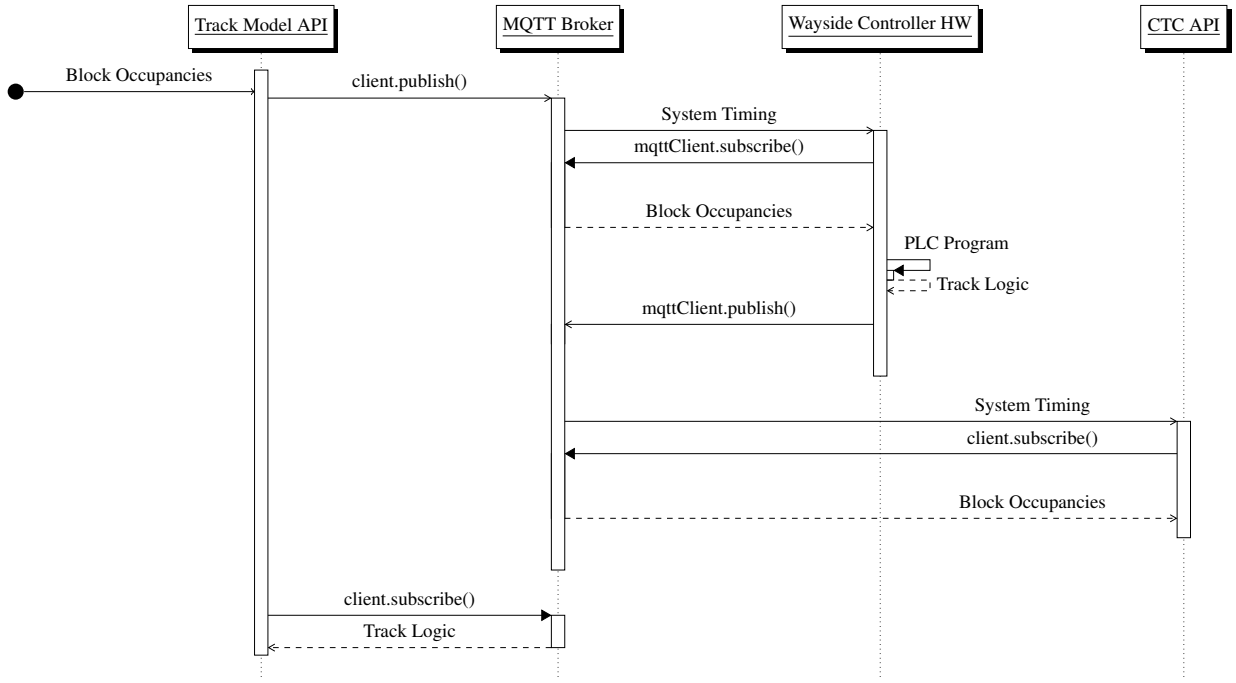

Figure 42: Sequence Diagram for ”Track Model sends an Occupancy”

Track Model sends a Fault:

Track Model sends a Fault
User Story: In my capacity as the Track Model, my aim is to notify both the Wayside Controller and the CTC Office about a failure. This is to ensure that the necessary measures are taken to maintain safety.
Acceptance Criteria: In the event that a failure condition is encountered in the Track Model, the process proceeds as follows: when the Track Model transmits the details of the fault to the Wayside Controller API, the Wayside Controller will promptly inform its user and subsequently communicate the fault information to the CTC Office.

Track Model sends a Fault	
Actors	Track Model
Preconditions	Failure condition is met within Track Model
Postconditions	Wayside user is notified and fault is relayed to CTC Office
Normal Flow	1. Wayside Controller informs its user of the fault 2. Wayside Software Wrapper sends the fault to CTC Office
Exceptional Flow	Either the fault is not displayed to the user, the CTC Office is not informed, or both occur

Figure 43: Sequence Diagram for "Track Model sends a Fault"

Track Model sends Switch Handshake:

Track Model sends Switch Handshake
User Story: As a Track Model, I want to send my switch state back to the Wayside Controller so that it may verify that the switch flipped properly.
Acceptance Criteria: In the scenario where the Track Model has received a switch command from the Wayside Controller, the process unfolds as follows: when the Track Model transmits the updated switch value in response, the Wayside Controller will validate that the received value matches the originally commanded state.

Track Model sends Switch Handshake	
Actors	Track Model
Preconditions	Track Model received a switch command
Postconditions	A fault is sent to the CTC Office that the switch is not set to the correct value
Normal Flow	1. The Wayside Controller conducts a comparison between the received switch state and the commanded switch state, and when this comparison proves successful
Exceptional Flow	1a. The Wayside Controller scrutinizes the received switch state in relation to the commanded switch state, and if the comparison results in a mismatch 1b. The Wayside Controller does not receive the expected handshake signal. 1. The Wayside Controller proceeds to transmit a fault signal regarding switch control to the CTC Office.

Track Model Sends Switch States

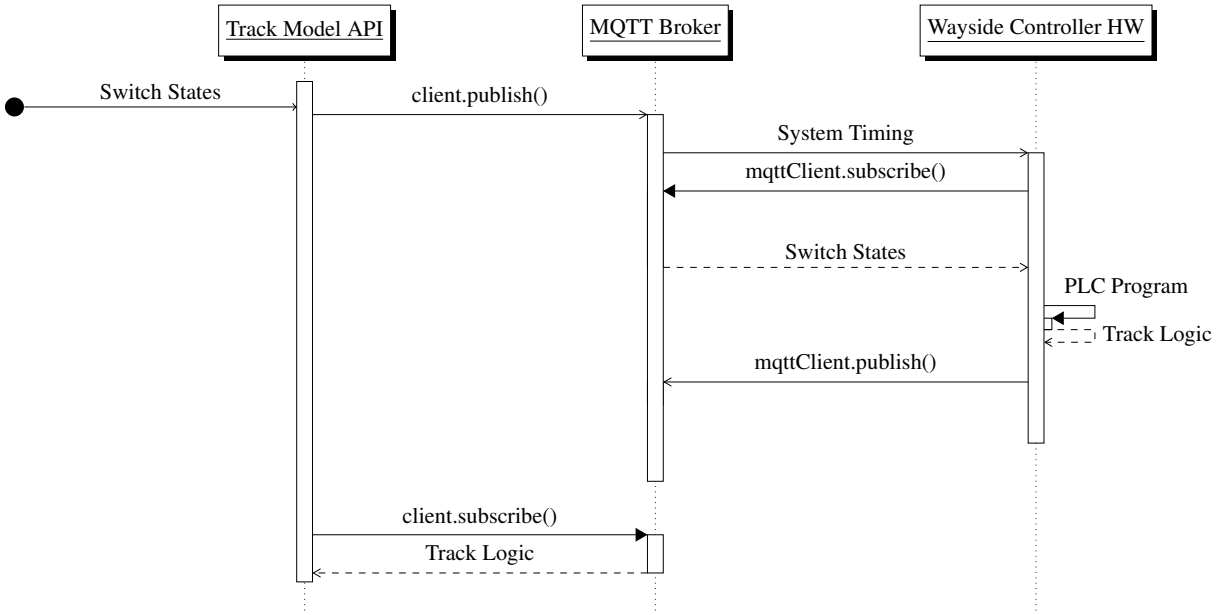

Figure 44: Sequence Diagram for "Track Model sends a Switch Handshake"

Track Model sends Signal Handshake:

Track Model sends Signal Handshake
User Story: As a Track Model, I want to send my signal state back to the Wayside Controller so that it may verify that the signal toggled properly.
Acceptance Criteria: In the scenario where the Track Model has received a signal command from the Wayside Controller, the process unfolds as follows: when the Track Model transmits the updated signal value in response, the Wayside Controller will validate that the received value matches the originally commanded state.

Track Model sends Signal Handshake	
Actors	Track Model
Preconditions	Track Model received a signal command
Postconditions	A fault is sent to the CTC Office if signal is not set to the correct value
Normal Flow	1. Wayside Controller compares received signal state to commanded signal state and comparison succeeds
Exceptional Flow	1a. Wayside Controller compares received signal state to commanded signal state and comparison fails 1b. Wayside Controller does not receive handshake signal 2. Wayside Controller sends a signal control fault to CTC Office

Track Model Sends Switch States

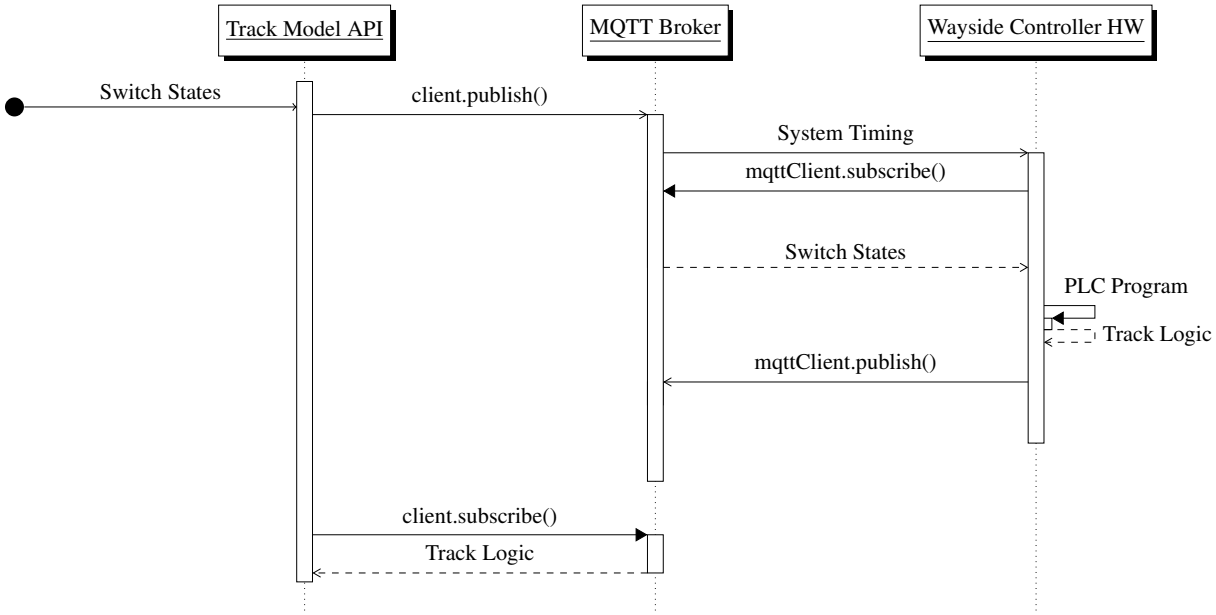

Figure 45: Sequence Diagram for ”Track Model sends a Signal Handshake”

Track Model sends Crossing Handshake:

Track Model sends Crossing Handshake
User Story: As a Track Model, I want to send my crossing state back to the Wayside Controller so that it may verify that the crossing toggled properly.
Acceptance Criteria: In a situation where the Track Model has received a crossing command from the Wayside Controller, the following process takes place: when the Track Model sends back the updated crossing value, the Wayside Controller will confirm that the received value matches the initially commanded state.

Track Model sends Crossing Handshake	
Actors	Track Model
Preconditions	Track Model received a crossing command
Postconditions	A fault is sent to the CTC Office if crossing is not set to the correct value
Normal Flow	1. Wayside Controller compares received crossing state to commanded crossing state and comparison succeeds
Exceptional Flow	1a. The Wayside Controller examines the received crossing state in relation to the commanded crossing state, and if the comparison results in a mismatch 1b. The Wayside Controller does not receive the expected handshake signal. 1. The Wayside Controller proceeds to transmit a fault signal regarding crossing control to the CTC Office.

Track Model Sends Crossing States

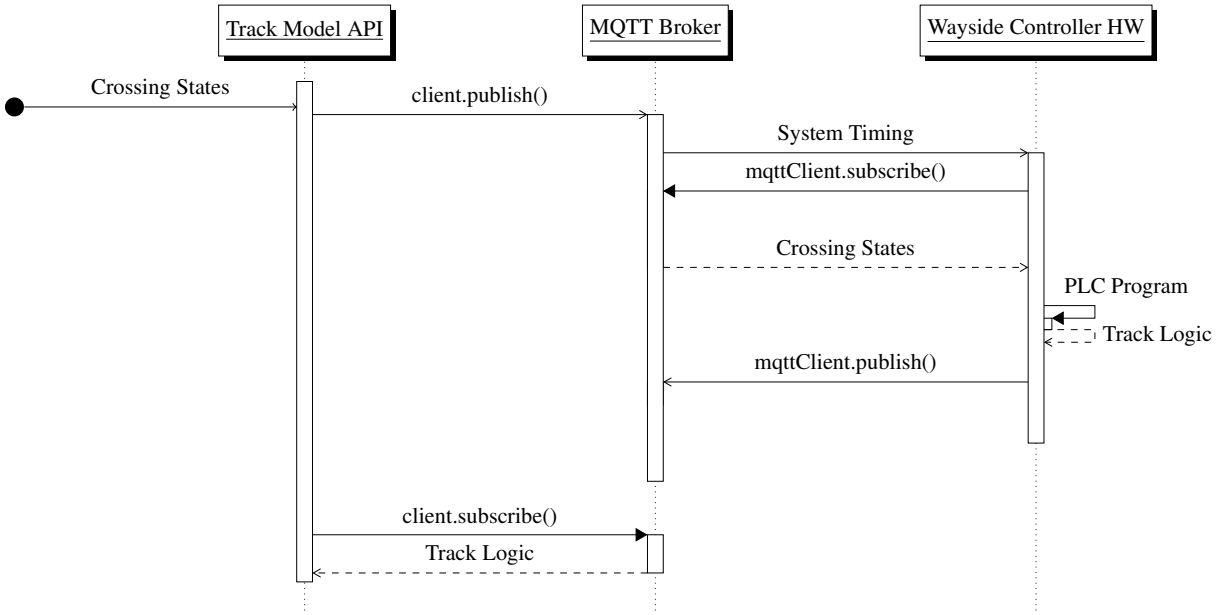

Figure 46: Sequence Diagram for ”Track Model sends a Crossing Handshake”

Wayside Programmer uploads PLC Code:

Wayside Programmer uploads PLC Code
User Story: As a Wayside Programmer, I would like to upload a new PLC program to a wayside controller so that I can modify how it performs track logic.
Acceptance Criteria: In the scenario where the Wayside Programmer requires a modification in the control of track logic, the process unfolds as follows: when the Wayside Programmer provides the file name and initiates the upload process, the Wayside Controller will replace its existing logic program with the newly uploaded one.

Wayside Programmer uploads PLC Code	
Actors	Wayside Programmer
Preconditions	Wayside Programmer needs to upload new logic program
Postconditions	The Wayside Programmer has uploaded programmer.
Normal Flow	1. Wayside Controller subscribes to PLC Uploaded file 2. Wayside Controller PLC interprets file contents 3. PLC sets track outputs.
Exceptional Flow	PLC program upload fails if the file path is invalid or the PLC code is invalid.

Wayside Programmer Uploads PLC

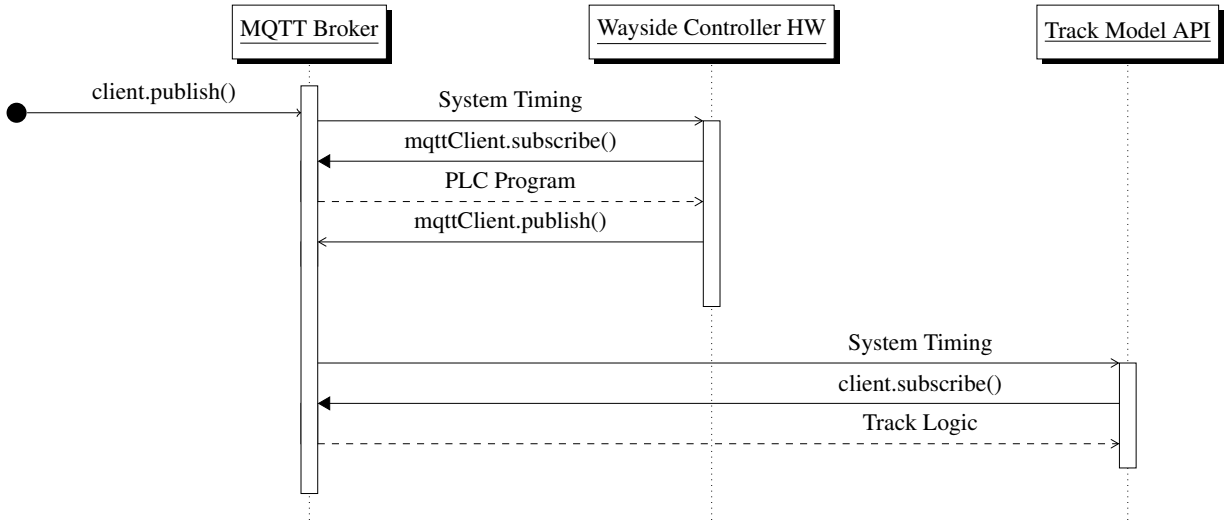

Figure 47: Sequence Diagram for "Wayside Programmer uploads PLC Code"

Wayside Programmer sets a Switch:

Wayside Programmer toggles a Switch
User Story: As a Wayside Programmer, I want to set a switch to manually control trains' paths.
Acceptance Criteria: In a situation where the Wayside Programmer must manually configure a switch, and the Wayside Controller is currently in manual mode, the process proceeds as follows: when the Wayside Programmer selects the toggle switch button, the Wayside Controller will adjust the requested switch to its alternate state.

Wayside Programmer toggles a Switch	
Actors	Wayside Programmer
Preconditions	Wayside Programmer needs to set a switch and Wayside Controller is in manual mode
Postconditions	The Wayside Controller sets the switch and sends the new state to Track Model
Normal Flow	1. Wayside Controller sets switch to new state and publishes change 2. Wayside Software wrapper sends new state to Track Model 3. Wayside Controller receives handshake
Exceptional Flow	Wayside Controller handshake fails

Wayside Programmer Manually Toggles Switch

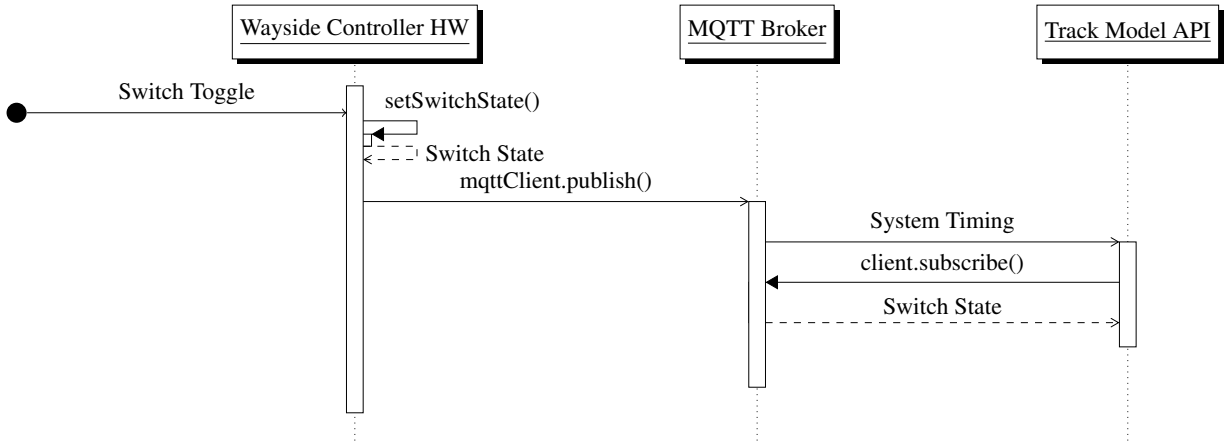

Figure 48: Sequence Diagram for "Wayside Programmer sets a Switch"

Wayside Programmer sets a Signal:

Wayside Programmer sets a Signal
User Story: As a Wayside Programmer, I want to set a signal to manually stop/start trains.
Acceptance Criteria: In a scenario where the Wayside Programmer must manually configure a signal, and the Wayside Controller is operating in manual mode, the process unfolds as follows: when the Wayside Programmer activates the toggle signal button, the Wayside Controller will adjust the requested signal to its alternate state.

Wayside Programmer toggles a Signal	
Actors	Wayside Programmer
Preconditions	Wayside Programmer needs to set a signal and Wayside Controller is in manual mode
Postconditions	The Wayside Controller sets the signal and sends the new state to Track Model
Normal Flow	1. Wayside Controller toggles signal to new state 2. Wayside Wrapper subscribes to new signal and sends new state to Track Model 3. Wayside Controller receives handshake
Exceptional Flow	Wayside Controller handshake fails

Wayside Programmer Manually Toggles Signal

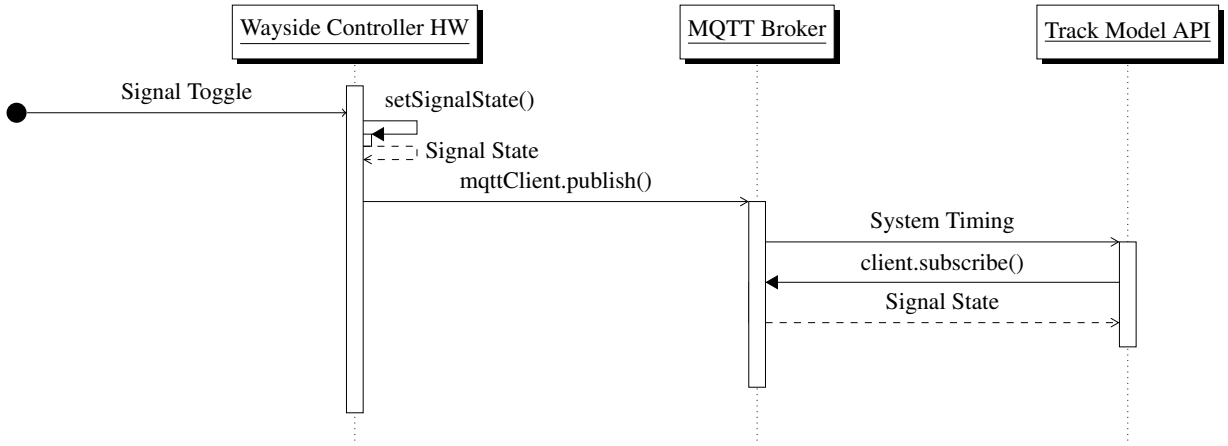

Figure 49: Sequence Diagram for "Wayside Programmer sets a Signal"

Wayside Programmer sets a Crossing:

Wayside Programmer sets a Crossing
User Story: As a Wayside Programmer, I want to set a crossing to manually control cars crossing the track.
Acceptance Criteria: In a scenario where the Wayside Programmer is required to manually configure a crossing, and the Wayside Controller is currently operating in manual mode, the process unfolds as follows: when the Wayside Programmer activates the toggle crossing button, the Wayside Controller will adjust the requested crossing to its alternative state.

Wayside Programmer sets a Crossing	
Actors	Wayside Programmer
Preconditions	Wayside Programmer needs to set a crossing and Wayside Controller is in manual mode
Postconditions	The Wayside Controller sets the crossing and sends the new state to Track Model
Normal Flow	1. The Wayside Controller adjusts the crossing to a new state. 2. The Wayside Controller transmits the updated state to the Track Model. 3. The Wayside Controller awaits and receives a confirmation signal (handshake).
Exceptional Flow	Wayside Controller handshake fails

Wayside Programmer Manually Toggles Crossing

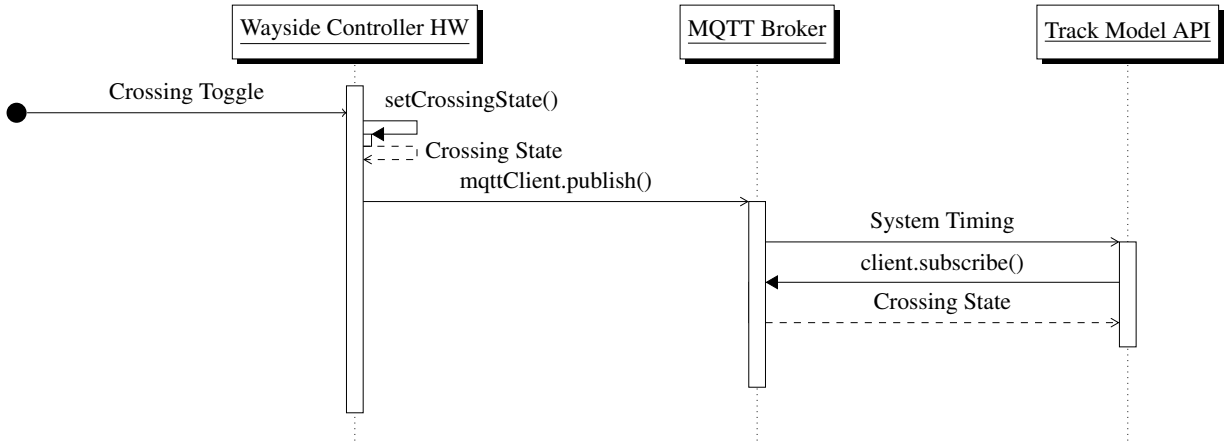

Figure 50: Sequence Diagram for ”Wayside Programmer sets a Crossing”

3.4 Track Model

3.4.1 Design Overview

The purpose of the Track Model is to act as a railroad and the simulated physical objects on a railroad. It is made up of rail lines, rail blocks, switches, crossings, stations, and signal lights. This module provides critical information to both the Wayside Controller and Train model. It also provides throughput to the CTC. The below sections describe further processes.

User Interface: Below are the Graphical User Interface Descriptions

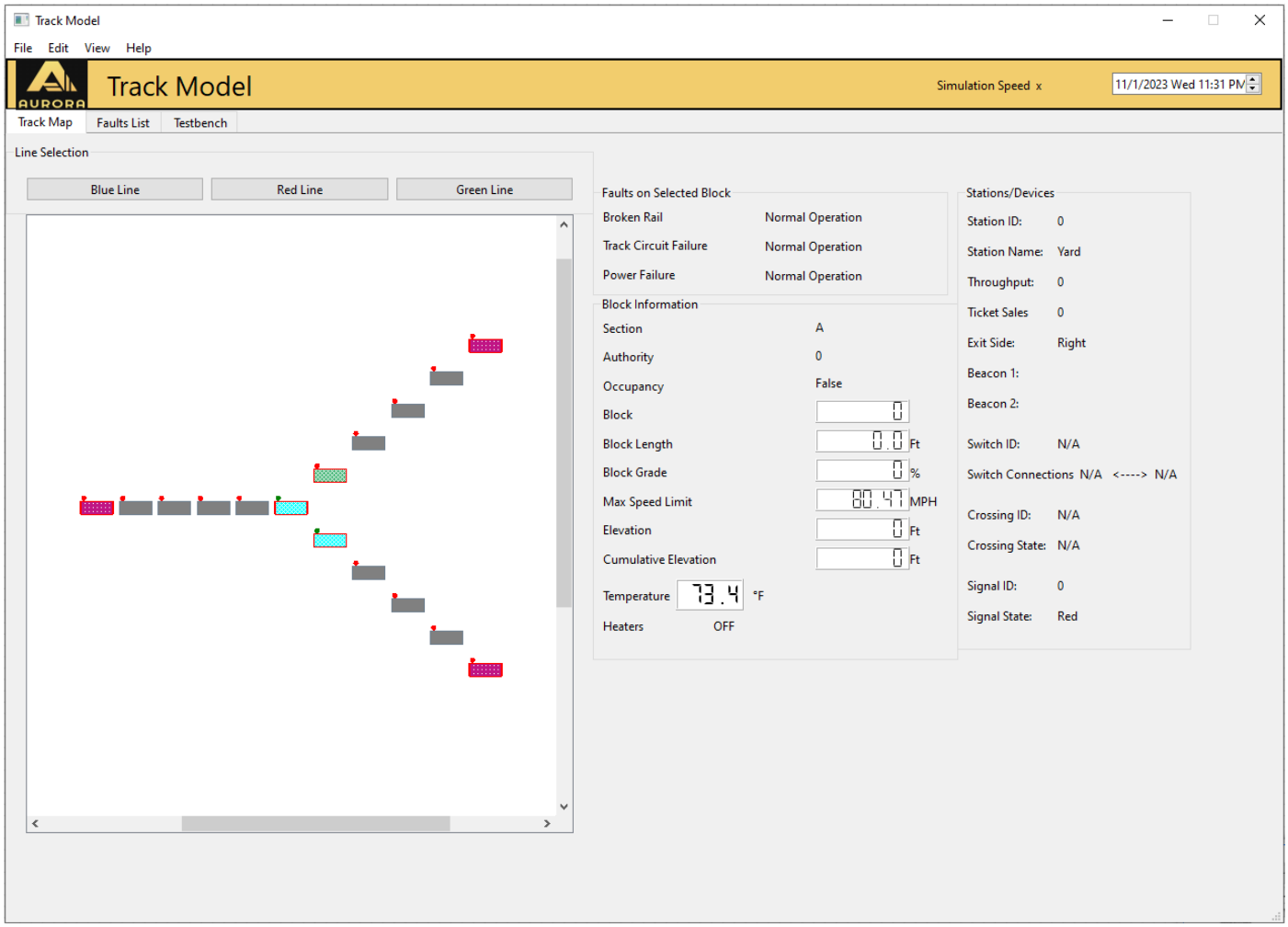

Figure 51: User interface for the Track Map

The track Map (Figure 51) is the graphical display the user will open up when they open the module from the loader. The Track Model has a selection buttons for all three rail lines blue, red, and green. Once selected a map will display in the box on the left. The map provides the user the ability to select a rail block and when selected, it will display all the track block's information on the right.

- **Faults on Selected Block:** displays the faults of the selected block if a fault exists.
- **Block Information:** Displays the following information

  - Section: The section each rail block belongs to (alphabetical)
  - Authority: Whether the train is permitted to continue and how far
  - Occupancy: Whether the block is occupied by a train or not.
  - Block length: the length of a block
  - Block Grade: The longitudinal slope across the track block
  - Max Speed Limit: Safe speed limit of the track block
  - Elevation: Elevation of the block
  - Cumulative Elevation: Total cumulative elevation of previous track blocks
  - Temperature: Displays ambient temperature outside
  - Heaters: Displays state of heaters ”ON” for 35°F and below, ”OFF” for any temperature above 35°F
- **Stations/Devices:** This block displays information for all stations or powered track devices on the track
  - Station ID: Displays station’s Identification Number
  - Station Name: Name of the station on the selected track block
  - Throughput: Total number of passengers that have been in the system
  - Passengers Waiting: Number of passengers that bought tickets and are waiting at the station for a train.
  - Exit Side: The side of the train the passengers exit on at that block.
  - Beacon 1: Displays beacon information of 128 bytes
  - Beacon 2: Displays the second beacon’s information of 128 bytes
  - Switch ID: Displays the Identification number for the switch on the block
  - Switch Direction: Displays the two block ID’s that the switch is connected to.
  - Crossing State: Displays the current state of the crossing
  - Signal ID: Displays the signal light’s Identification Number
  - Signal State: Displays the current signal’s light color

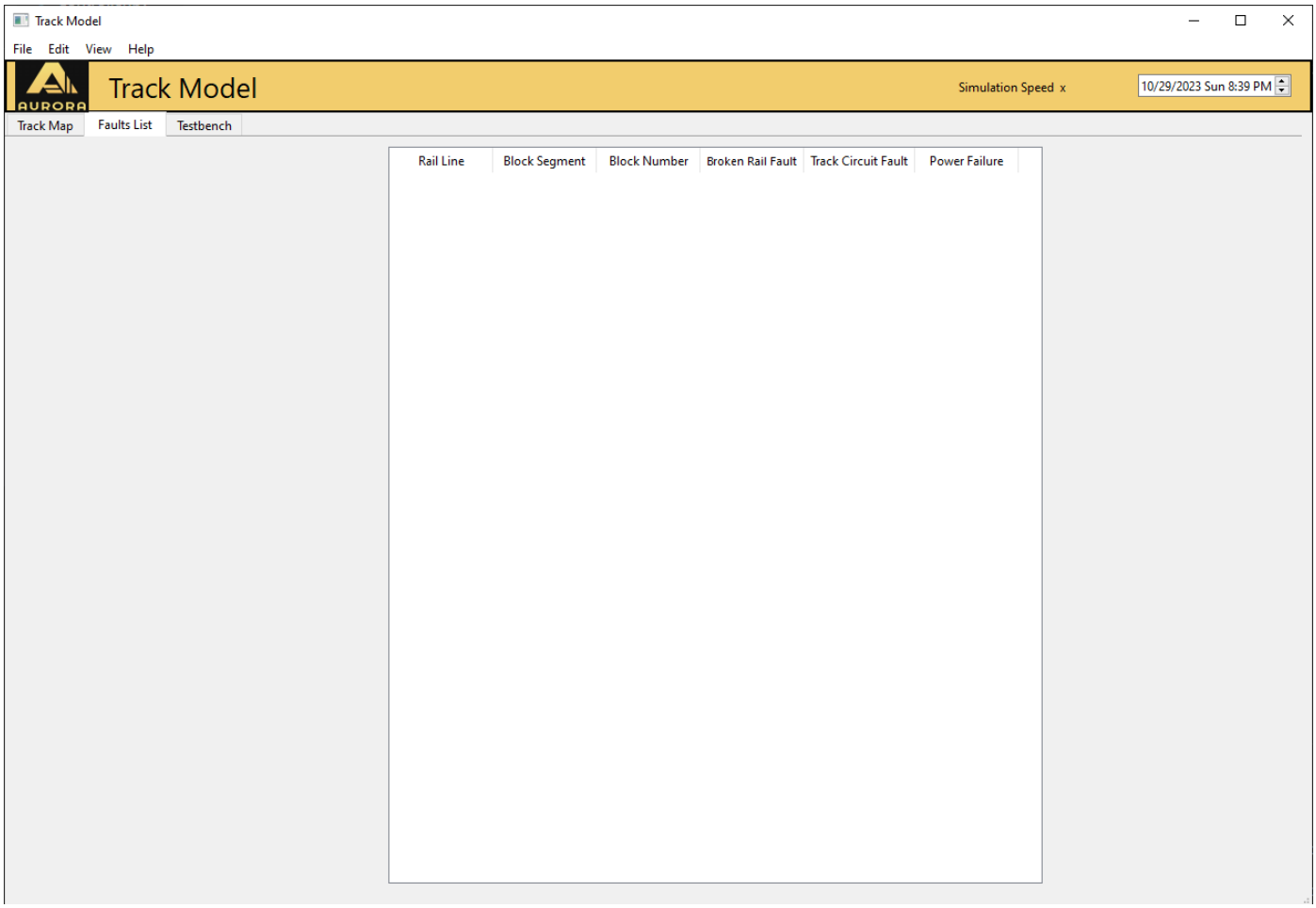

Figure 52: User Interface displaying fault list for all rail blocks

The Fault List (Figure 52) page displays every rail line, track block number, the segment each block is on, and whether that block has either a Broken Rail, Track Circuit Failure, or Power Failure.

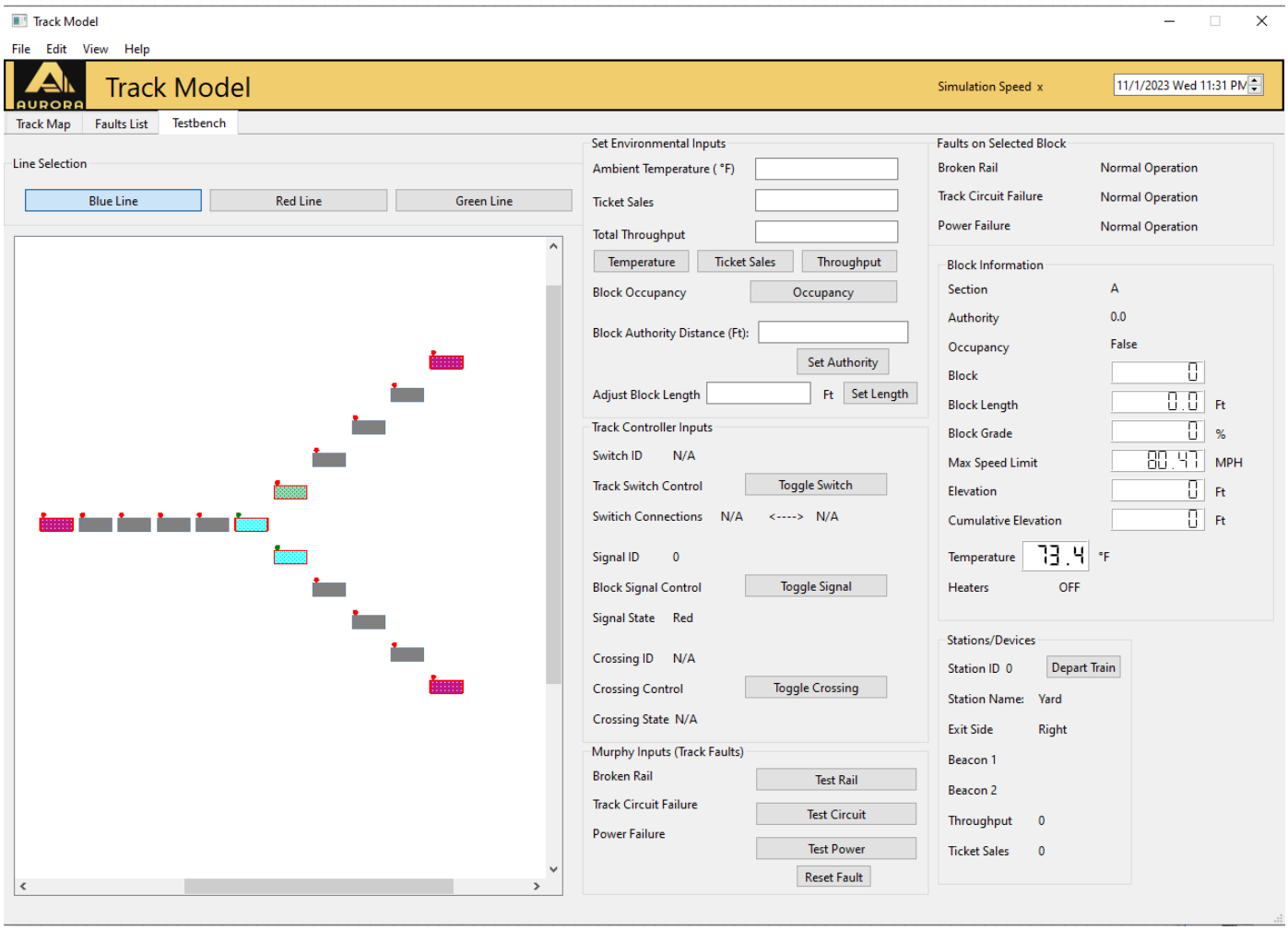

Figure 53: Testbench to test any problems in the system

The Testbench (Figure 53 page of the track model has a selection buttons for all three rail lines blue, red, and green. Once selected a map will display in the box on the left. The map provides the user the ability to select a rail block and when selected, it will display all the track block's information on the right.

- **Set Environmental Inputs:** displays the faults of the selected block if a fault exists.
  - Ambient Temperature text box input: Type the temperature in Fahrenheit, and click the "Temperature" button to set the ambient temperature to test heaters.
  - Passenger Boarding Count: Type number of passengers boarding the next train, and click "Boarding Count" button to set. NOTE: A station MUST be selected, else no changes will be made.
  - Total Throughput text box input: Type in the total throughput of the system and click the "Throughput" button to set.
  - Block Inputs: Occupancy button: Sets the selected track block to occupied
  - Radio Button: True(if selected): If selected and distance is input in the textbox to the right. Pressing the "Authority button" will set the distance a train is permitted to drive and whether that train should continue receiving authority on the block. If the controller wishes the train to stop early, set the distance you wish

it to stop in and de-select the radio button.

  - Adjust Block Length: Typing the new distance of the selected block and pressing set length will change the block length.

- **Murphy Inputs(Track Faults):** Set faults on the track blocks, and send the signals to the Wayside Controller

  - Broken Rail: Test Rail Button: Pressing this button it will trigger a 'Broken Rail Fault' on the selected block.

  - Track Circuit Failure: Test Circuit Button: Pressing this button will trigger a 'Track Circuit Failure' on the selected block.

  - Power Failure: Test Power Button: Pressing this button will trigger a power failure on the selected block.

- **Faults on Selected Block:** displays the faults of the selected block if a fault exists.

  - Broken Rail

  - Track Circuit Failure

  - Power Failure

- **Block Information:** Displays the following information

  - Section: The section each rail block belongs to (alphabetical)

  - Authority: Whether the train is permitted to continue and how far

  - Occupancy: Whether the block is occupied by a train or not.

  - Block length: the length of a block

  - Block Grade: The longitudinal slope across the track block

  - Max Speed Limit: Safe speed limit of the track block

  - Elevation: Elevation of the block

  - Cumulative Elevation: Total cumulative elevation of previous track blocks

  - Temperature: Displays ambient temperature outside

  - Heaters: Displays state of heaters "ON" for 35°F and below, "OFF" for any temperature above 35°F

- **Stations/Devices:** This block displays information for all stations or powered track devices on the track

  - Station ID: Displays station's Identification Number

  - Depart Train Button: Departs a train, reduces passengers waiting to 0 and increases throughput by "passengers waiting".

  - Station Name: Name of the station on the selected track block

  - Throughput: Total number of passengers that have been in the system

  - Passengers Waiting: Number of passengers that bought tickets and are waiting at the station for a train.

- Exit Side: The side of the train the passengers exit on at that block.
- Beacon 1: Displays beacon information of 128 bytes.
- Beacon 2: Displays the second beacon's information of 128 bytes.
- Switch ID: Displays the Identification number for the switch on the block.
- Switch Direction: Displays the track blocks that the switch is connected to.
- Crossing State: Displays the current state of the crossing.
- Signal ID: Displays the signal light's Identification Number.
- Signal State: Displays the current signal's light color.

3.4.2 Class Diagram

The below figure 54 shows objects, classes, and external functions that interact with the Track Model User Interface. The track_model_ui interacts with the rail_lines class object.

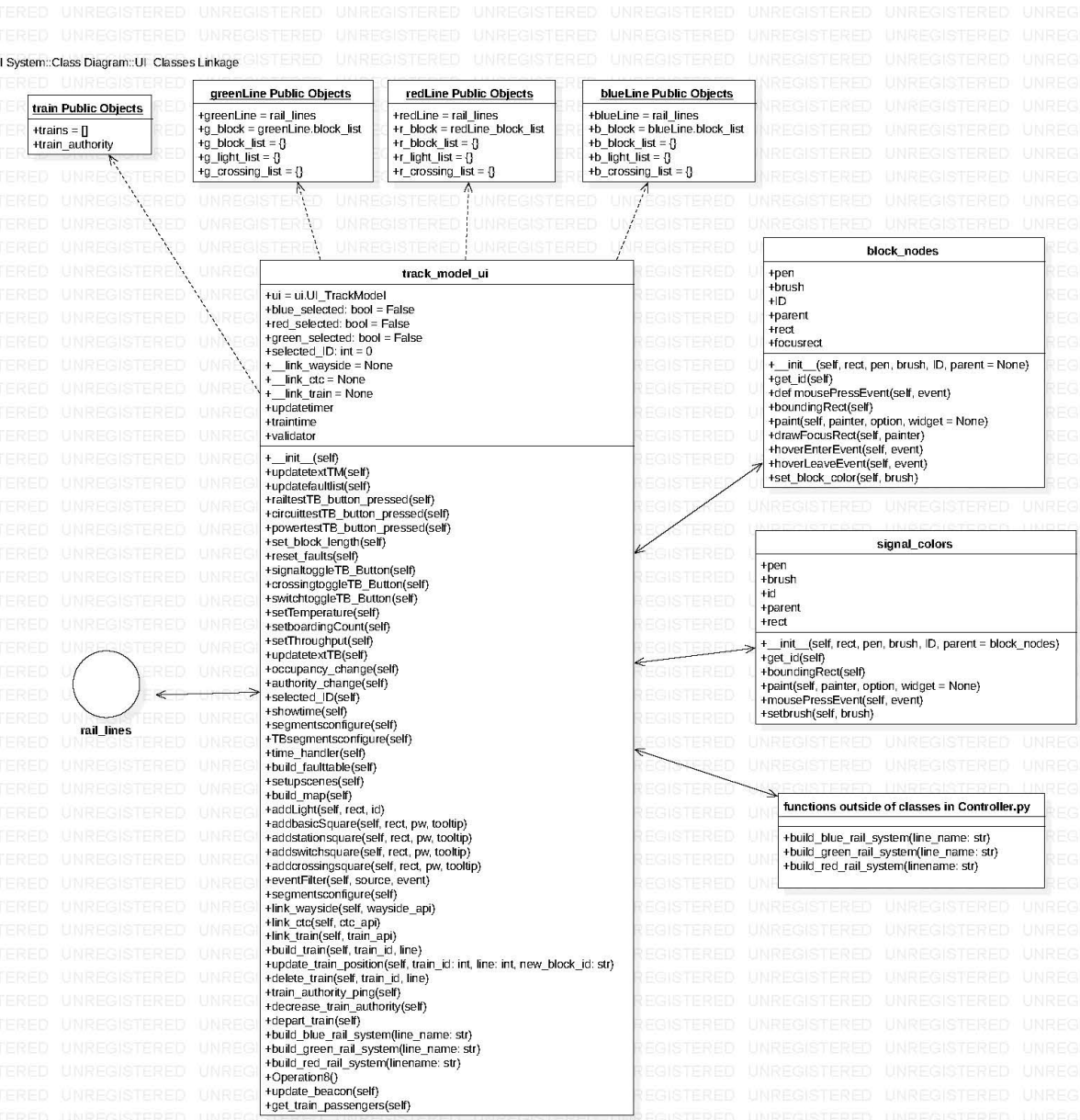

Figure 54: Track Model Software Use Case Diagram

track mode ui	
Description: Graphical user interface class for wayside controller module. Responsible for allowing the user to interact with the Track Model.	
Attributes	
Name	Description
ui	creates the UI object
blue_selected	A bool that stores if the blue line is selected by the user
green_selected	A bool that stores if the green line is selected by the user
red_selected	A bool that stores if the red line is selected by the user
selected_ID	An int that stores which track block is selected by the user
__link_wayside	An empty variable for linking an API class object that is used to transmit information to the wayside
__link_ctc	An empty variable for linking an API class object that is used to transfer ticket sales to the CTC
__link_train	An empty variable for linking an API class object that can talk to the track model
updatetimer	A timer object to update the User Interface Display
traintime	A timer to update trains and positions on the User Interface
validator	An object used to prevent any non-integer inputs into text input of the Testbench
Responsibilities	
Name	Collaborator
Blue Line	User, Self
Red Line	User, Self
Green Line	User, Self
Set Temperature	User, Self
Set Ticket Sales	User, Self
Set Total Throughput	User, Self
Set Occupancy	User, Self
Set Authority	User, Self, Wayside Controller
Adjust Block Length	User, Self
Toggle Switch	User, Self, Wayside Controller
Toggle Signal	User, Self, Wayside Controller
Toggle Crossing	User, Self, Wayside Controller
Test Broken Rail	Self, Murphy
Test Track Circuit Failure	Self, Murphy
Test Power Failure	Self, Murphy
Reset Faults	User Self, Murphy
Depart Train	User, Self, Wayside Controller

block_nodes	
Description: A class that is used to create track block graphic objects in the viewer used to indicate track blocks, switches, stations, and crossings.	
Attributes	
Name	Description
pen	Stores the pen information to color the block outline.
brush	Stores the brush used to paint the block's color.
ID	Stores the graphical objects Identification Number
parent	Stores the parent of the block
rect	Stores the position, look, and size of the block
focusrect	Stores the position, look, and size of the graphical object that displays that the block is selected
Responsibilities	
Name	Collaborator

signal_colors	
Description: A class that is used to create track block graphic objects in the viewer used to indicate track lights.	
Attributes	
Name	Description
pen	Stores the pen information to color the block outline.
brush	Stores the brush used to paint the block's color.
ID	Stores the graphical objects Identification Number
parent	Stores the parent of the block
rect	Stores the position, look, and size of the block
focusrect	Stores the position, look, and size of the graphical object that displays that the block is selected
Responsibilities	
Name	Collaborator

Train Related Public Objects	
Description: Simulated Train public objects that the track_model_ui uses.	
Attributes	
Name	Description
trains	An array for storing trains
train_authority	A value used to send a new authority to the train
Responsibilities	
Name	Collaborator

Blue Line Related Public Objects	
Description: Public objects the Green Line uses.	
Attributes	
Name	Description
breenLine	A variable to store a rail_lines class object
b_block	A variable that shortens the "BlueLine.block_list" class name and objects.
b_block_list	A dictionary to store all visual graphic track blocks, stations, and crossings objects on the blue line
b_light_list	A dictionary to store all visual signal light objects on the blue line.
Responsibilities	
Name	Collaborator

Green Line Related Public Objects	
Description: Public objects the Green Line uses.	
Attributes	
Name	Description
greenLine	A variable to store a rail_lines class object
g_block	A variable that shortens the "greenLine.block_list" class name and objects.
g_block_list	A dictionary to store all visual graphic track blocks, stations, and crossings objects
g_light_list	A dictionary to store all visual signal light objects.
Responsibilities	
Name	Collaborator

Red Line Related Public Objects	
Description: Public objects the Red Line uses.	
Attributes	
Name	Description
redLine	A variable to store a rail_lines class object
r_block	A variable that shortens the ”redLine.block_list” class name and objects.
r_block_list	A dictionary to store all visual graphic track blocks, stations, and crossings objects
r_light_list	A dictionary to store all visual signal light objects.
Responsibilities	
Name	Collaborator

Red Line Related Public Objects	
Description: Functions that is used to load the backend and create class object information that the track_model_ui utilizes.	
Attributes	
Name	Description
build_blue_rail_system(line_name:str)	A function used to call the Database and create the Blue Line’s class objects
build_green_rail_system(line_name:str)	A function used to call the Database and create the Green Line’s class objects
build_red_rail_system(line_name:str)	A function used to call the Database and create the Red Line’s class objects
Responsibilities	
Name	Collaborator

The below figure 55 displays the class connections between the track objects and the UI. The rail_lines class object holds information that is disseminated to the track_module_ui, the Wayside Station, CTC, and Train Model.

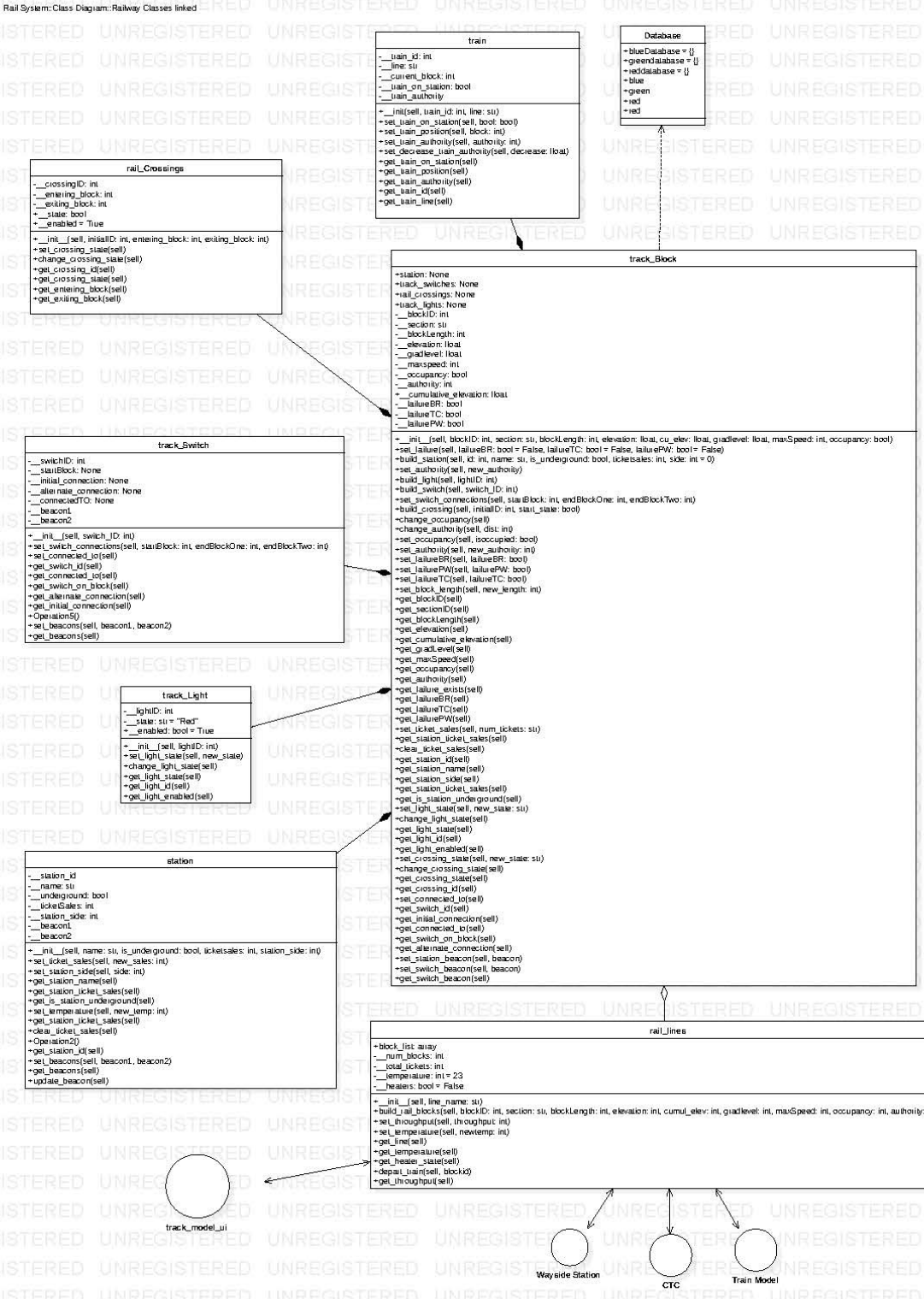

Figure 55: Track Model Software Use Case Diagram

rail_lines	
Description: This class is used to store all class objects and information for each line.	
Attributes	
Name	Description
block_list	An array that holds all track block class objects.
__num_blocks	Integer that stores the number of blocks in the line.
__total_tickets	Integer that stores the total number of tickets sold.
__temperature	Integer that holds the ambient air temperature.
__heaters	A boolean that contains the state of the rail heaters on the line
Responsibilities	
Name	Collaborator

track_Block	
Description: This class is used to store all class objects and information for the track_Block class. It also holds functions to access all the class objects associated with it.	
Attributes	
Name	Description
station	A variable to store a station class object.
track_switches	A variable to store a track switch object.
rail_crossings	A variable to store a rail crossing class object.
track_lights	A variable to store a track light class object.
__blockID	An integer variable to store that blocks Identification Number.
__section	A string variable to store the section the block is on.
__blockLength	An integer variable to store the length of the block.
__elevation	A float variable to store the elevation of the block.
__gradlevel	A float variable to store the gradient of the block.
__maxspeed	An integer variable to store the max speed of the block.
__occupancy	A boolean variable to store if the block is occupied.
__authority	An int variable to store the authority of the block.
__cumulative_elevation	A float variable to store the cumulative elevation of the block.
__failureBR	A boolean variable to store if the block has a Broken Rail Fault.
__failureTC	A boolean variable to store if the block has a Track Circuit Failure.
__failurePW	A boolean variable to store if the block has a Power Failure.
Responsibilities	
Name	Collaborator

station	
Description: The station class holds all the station variables and contains set and get functions to modify the class.	
Attributes	
Name	Description
__station_id	A variable to store the stations identification number.
__name	A variable to store the name of the station.
__underground	A variable to store if the station is underground.
__ticketSales	A variable to store the ticket sales at that station.
__station_side	An integer variable to store which side of the train the passengers can exit.
__beacon1	A variable to store the beacon information.
__beacon2	An variable to store the beacon 2's information.
Responsibilities	
Name	Collaborator

track_light	
Description: The station class holds all the station variables and contains set and get functions to modify the class.	
Attributes	
Name	Description
__lightID	An integer variable to store the signal's identification number.
__state	A string variable to store the state of the signal.
__enabled	A boolean variable to store if the light is off.
Responsibilities	
Name	Collaborator

track_switch	
Description: The station class holds all the station variables and contains set and get functions to modify the class.	
Attributes	
Name	Description
__switchID	An integer variable to store the switch's identification number.
__startBlock	A string variable to store the block the switch is on.
__initial_connection	An integer variable to store the block ID the switch is connected to initially.
__alternate_connection	An integer to store the block ID of the second connection the switch can connect to.
__connectedTO	An integer to store the ID of the block the switch is currently connected to
__beacon1	A variable to store beacon information
__beacon2	A variable to store beacon 2's information
Responsibilities	
Name	Collaborator

rail_Crossings	
Description: The rail crossings class holds all the crossing variables and contains set and get functions to modify the class.	
Attributes	
Name	Description
__crossingID	An integer variable to store the crossing's identification number.
__entering_block	A integer variable to store the block ID of the block before the crossing.
__exiting_block	A integer variable to store the block ID of the block after the crossing.
__state	An boolean variable to store the crossing's current state.
__enabled	An boolean variable to store if the data can be accessed or not during a power outage.
Responsibilities	
Name	Collaborator

rail_Crossings	
Description: The rail crossings class holds all the crossing variables and contains set and get functions to modify the class.	
Attributes	
Name	Description
__train_id	An integer variable to store the train's ID
__line	A string variable to store what line the train is on
__current_block	An integer variable to store the current position of the train
__train_on_station	A boolean to dictate if the train is stopped at a station and "dwelling".
__train_authority	A variable that holds the trains authority.
Responsibilities	
Name	Collaborator

Database	
Description: The rail crossings class holds all the crossing variables and contains set and get functions to modify the class.	
Attributes	
Name	Description
blueDatabase	A dictionary that holds most blue line block information from an xlsx file.
greenDatabase	A dictionary that holds most green line block information from an xlsx file.
redDatabase	A dictionary that holds most red line block information from an xlsx file.
blue	A variable to pull in the xlsx database information and transfer it to blueDatabase variable.
green	A variable to pull in the xlsx database information and transfer it to greenDatabase variable.
red	A variable to pull in the xlsx database information and transfer it to redDatabase variable.
Responsibilities	
Name	Collaborator

3.4.3 Use Cases

The main case actors for the Track Model are the Testbench, Train Model, CTC, Wayside Controller, and Murphy. Murphy is random faults that occur, and the Testbench is a program used to interact with the system on the back-end to test proper functionality.

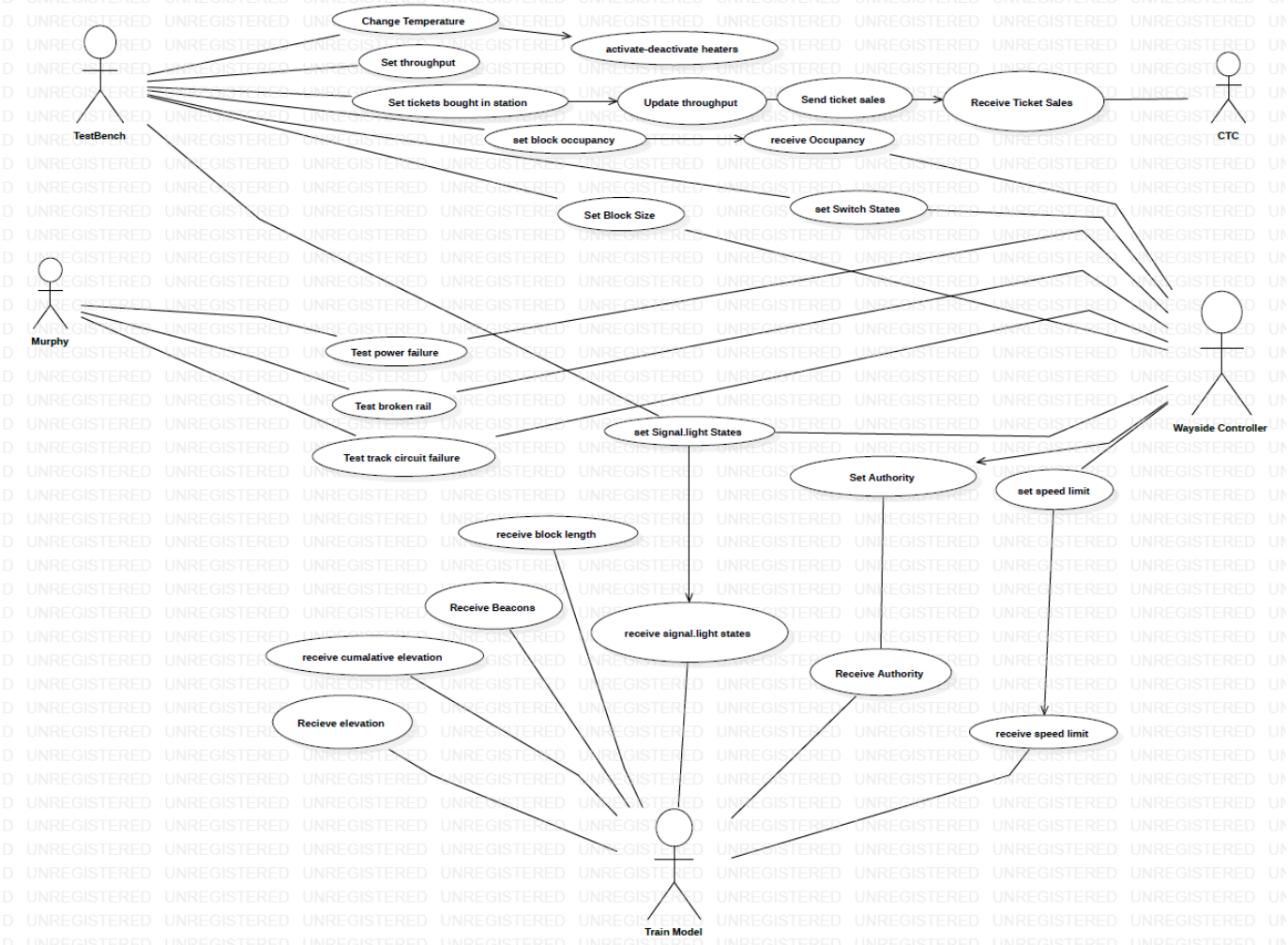

Figure 56: Track Model Software Use Case Diagram

The Track Model Use Case Diagram in Figure 56 displays the connections between all the actors that may interact with our system at some point throughout the design process and commercial use. Various cases mentioned in this diagram are mentioned below.

Authority Handling

Authority Handling
User Story: As the train model, I want to receive authority and know whether I am allowed to continue moving along the track from the Track Model.
Acceptance Criteria: When the Wayside Controller updates a blocks authority the authority for that block will be sent through the Track Model to the Train.

Authority Handling	
Actors	Wayside Controller and Train Model
Preconditions	Wayside Controller receives new authority for blocks from the ctc
Postconditions	Once authority is received the Train Model sends a signal to let the Track Model know the transmission was received
Normal Flow	1. Receive authority from Wayside Controller for the selected block 2. Send the updated authority to the selected block for the train to read.
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

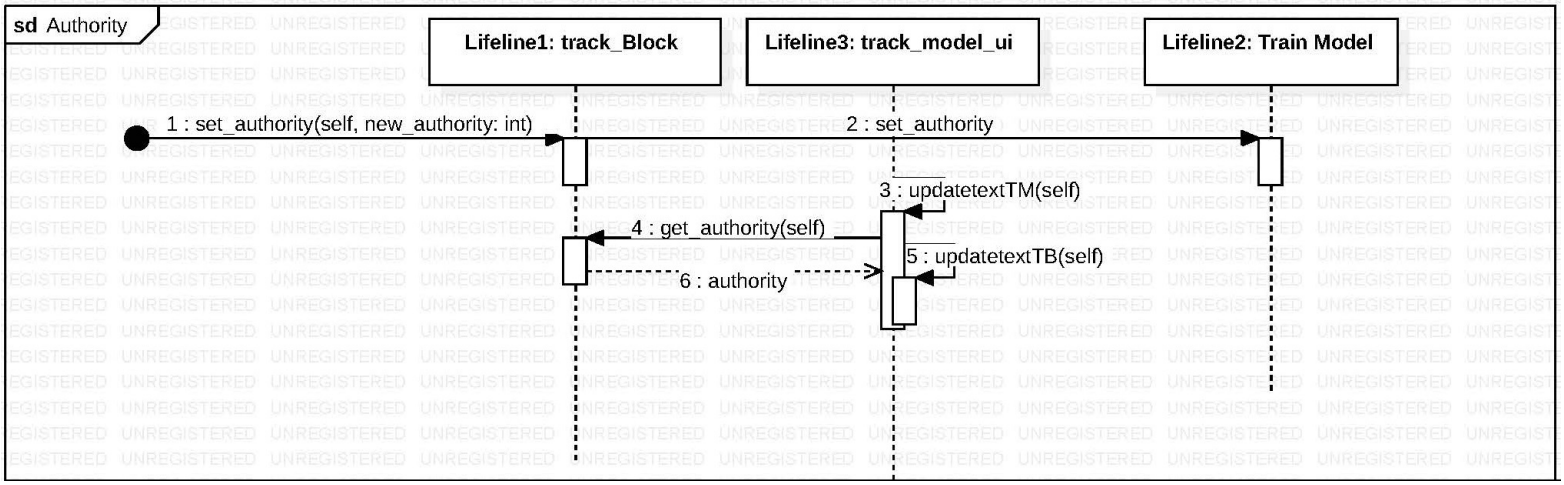

Figure 57: Track Model Authority Sequence Diagrams

Beacon Handling

Beacon Handling
User Story: As the train model, I want to receive two beacons from track switches and stations of 128 bytes each. The Beacons will contain: - track length - track grade - track speed limit - track length - track grade - if track blocks are underground - station ID (if beacon exists on a station
Acceptance Criteria: The Train model gets the data and sends it to the Train Controller.

Beacon Handling	
Actors	Wayside Controller and Train Model
Preconditions	Track Model gathers information to send and sets them to be static on switches or stations
Postconditions	Once authority is received the Train Model sends a signal to let the Track Model know the transmission was received,
Normal Flow	1. Receive authority from Wayside Controller for the selected block 2. Send the updated authority to the selected block for the train to read.
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

Switch Handling

Switch Handling
User Story: As the train Model I want to travel across switches connections that are in the correct positions.
Acceptance Criteria: The train travels across switches to their destination without issues.

Switch Handling	
Actors	Wayside Controller and Train Model
Preconditions	The Wayside Controller should know occupancy positions
Postconditions	As a train crosses a switch, send it beacon information
Normal Flow	1. Wayside Control receives track block occupancy. 2. Wayside Control sends a signal to Track Model to flip switch. 3. Track Model changes switch connection 4. Train travels across switch connection.
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

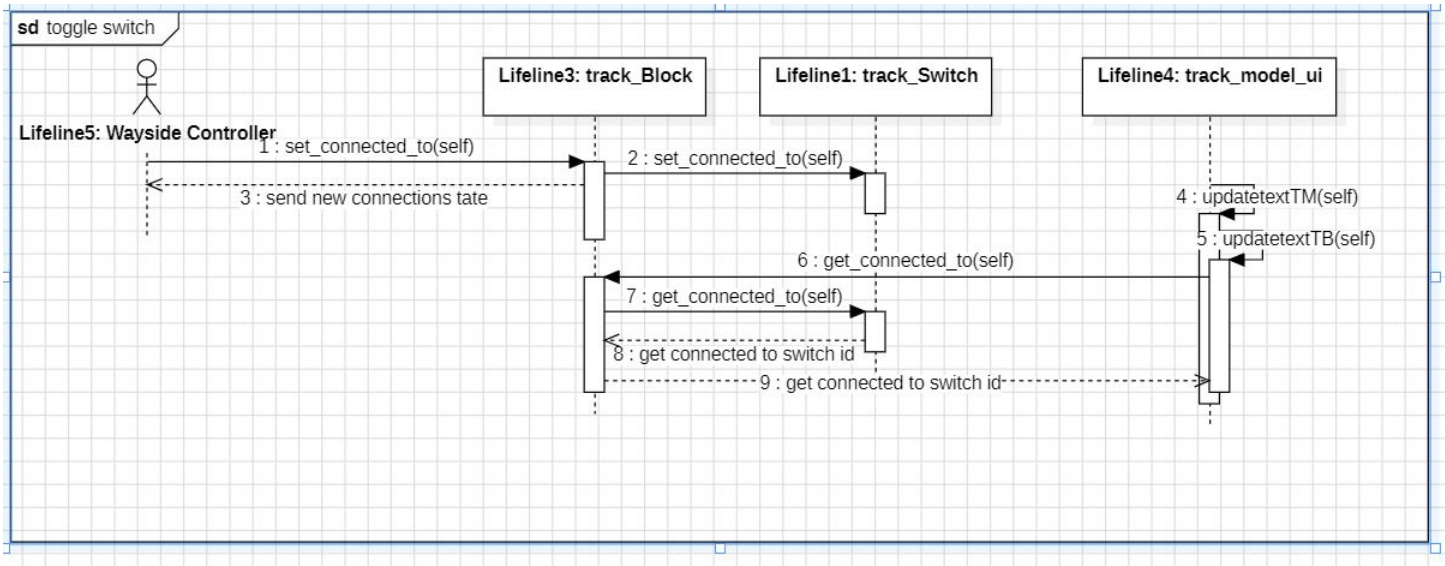

Figure 58: Track Model Switch Handling Sequence Diagrams

Signal Handling

Signal Handling
User Story: As the train Model I want to see light signals and stop when they change to red or go when they turn green.
Acceptance Criteria: The signals must display properly without losing power.

Signal Handling	
Actors	Wayside Controller and Train Model
Preconditions	The Wayside Controller should know occupancy positions
Postconditions	As a train enters a block it sees that blocks signal light
Normal Flow	1. Wayside Control receives track block occupancy. 2. Wayside Control sends a signal to Track Model to change the signal. 3. Track Model changes signal colors 4. Train travels past the signal if it is green, else stops.
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

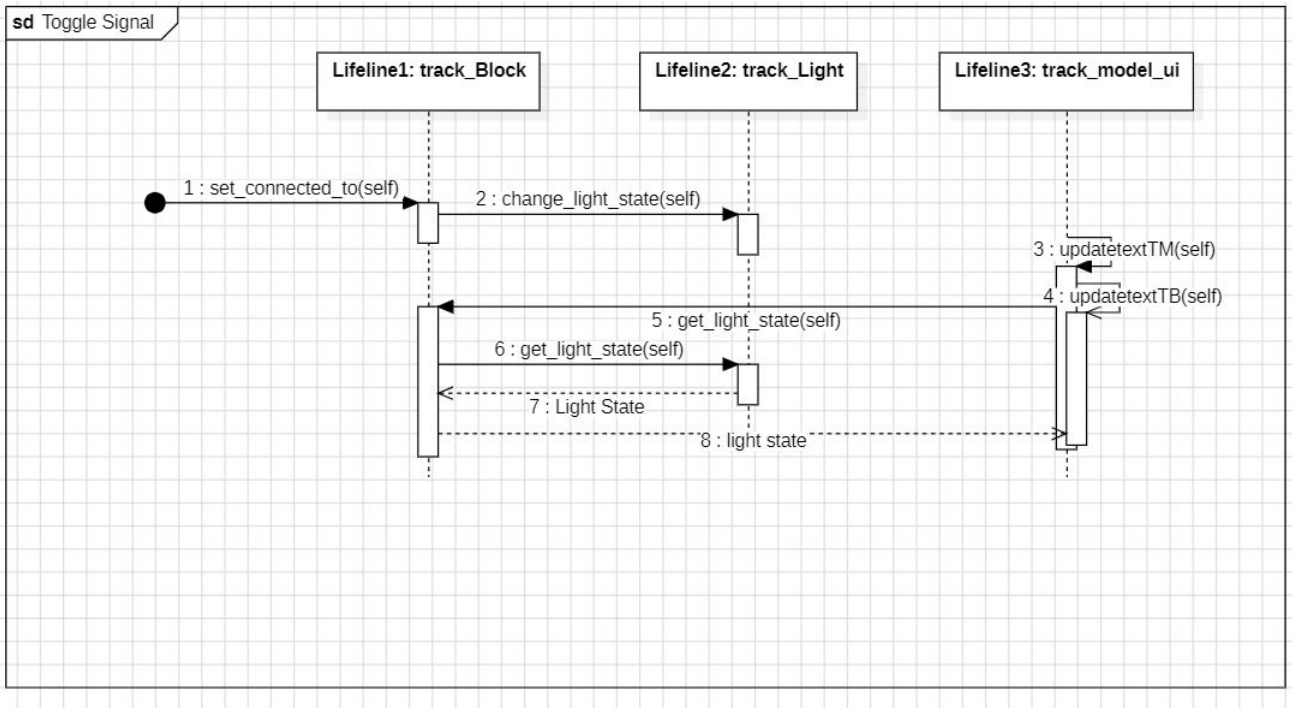

Figure 59: Track Model Signal Handling Sequence Diagram

Temperature Handling

Temperature Handling
User Story: As the Track Model I want to turn on rail heaters if the temperature gets too low.
Acceptance Criteria: Heaters Activate when temperature is below 35.06 degrees Fahrenheit. Conversely, they turn off when above 35.06 degrees Fahrenheit

Temperature Handling	
Actors	Track Model
Preconditions	None
Postconditions	Heater state must turn on when below 35.06 degrees Fahrenheit. Heater state must turn off when above 35.06 degrees Fahrenheit.
Normal Flow	1. Track Model sensors read temperature. 2. If temperature read is below 35.06 degrees Fahrenheit turn on heaters 3. If temperature read is above 35.06 degrees Fahrenheit turn off heaters 4. Update track model visuals display.
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

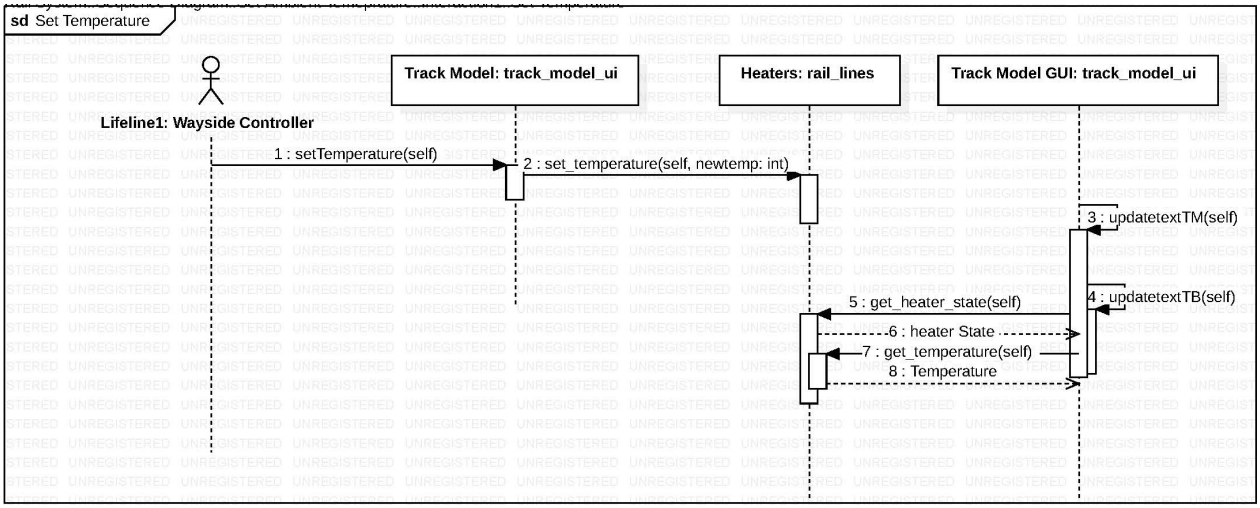

Figure 60: Track Model Temperature Handling Sequence Diagrams

Block Length Handling

Block Length Handling
User Story: As the Track Model Controller user, I want to change the length of blocks manually.
Acceptance Criteria: Change the block information in the track block class object, and display it on the block length text output block.

Block Length Handling	
Actors	Track Model
Preconditions	User must select a block
Postconditions	Beacon information must update with new information input.
Normal Flow	1. User selects a track block 2. User types in new block length 3. User presses set block length button 4. Block object and changes block length 5. User Interface displays the new information.
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

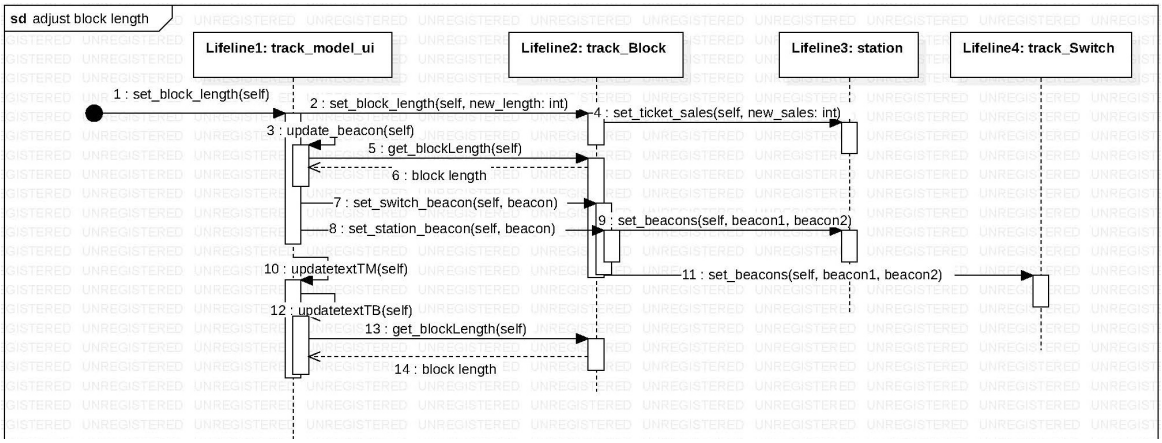

Figure 61: Track Model block length Handling Sequence Diagrams

Broken Rail Handling

Broken Rail Handling
User Story: As Murphy I want to cause a Broken Rail failure. As the Track Model, If there is a broken rail I want to stop the train model from getting to that block, before a catastrophic event occurs.
Acceptance Criteria: The Track Model receives a new track block authority to send to the train.

Broken Rail Handling	
Actors Murphy	Wayside Controller and Train Model
Preconditions	Occupancy flag gets sent to Wayside Controller and it gets recognized as erroneous
Postconditions	Beacon information must update with new information input.
Normal Flow	1. Murphy presses the broken rail test button. 2. The track block recognizes a Broken Rail Failure 3. The track block sends occupancy status of ”occupied” to the Wayside Controller. 4. The Wayside Controller sends a new authority to the track block. 5. The track block sends the authority to the block(s) incoming trains are on 6. The Train Model receives the new authority and slows down. 7. The Track Model updates it’s display with the fault.
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

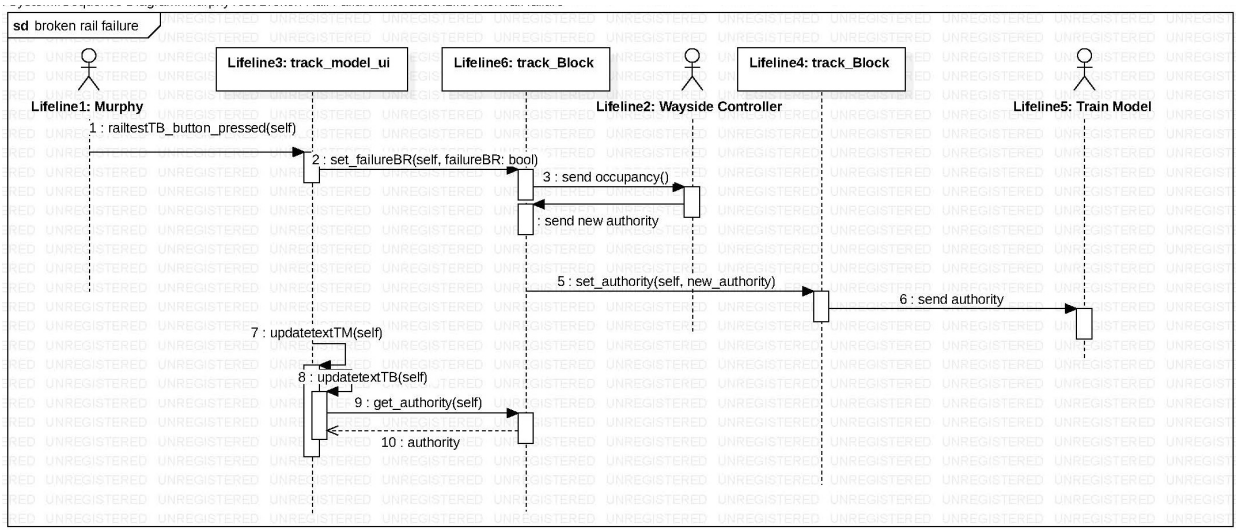

Figure 62: Track Model broken rail Handling Sequence Diagrams

Track Circuit Failure Handling

Track Circuit Failure Handling
User Story: As Murphy I want to cause a track circuit failure and have the Wayside Controller be notified using occupancy something occurred.
Acceptance Criteria: No function calls work to set power devices statuses if there is a power failure.

Track Circuit Failure Handling	
Actors	Murphy and Wayside Controller
Preconditions	None
Postconditions	None.
Normal Flow	1. Murphy presses the test track circuit failure button. 2. The UI sets the fault on the track block. 3. The track block sends occupancy status of ”occupied” to the Wayside Controller. 4. The Wayside Controller sends a new authority to the block(s) to send to the train model. 5. The track block sends the new authority to the Train Model.
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

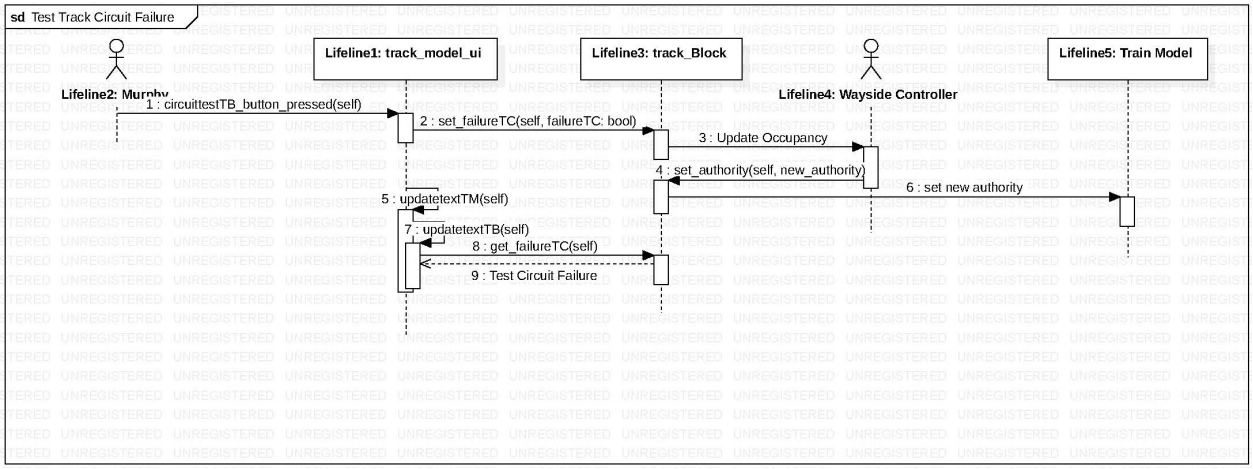

Figure 63: Track Model Track Circuit Failure Handling Sequence Diagrams

Power Failure Handling

Power Failure Handling
User Story: As the track model if there is a power failure, none of my values will get changed when told to change by the Wayside Controller.
Acceptance Criteria: No function calls work to set power devices statuses if there is a power failure.

Power Failure Handling	
Actors	Murphy and Wayside Controller
Preconditions	None
Postconditions	None.
Normal Flow	1. A power failure occurs. 2. The track block sends occupancy status of ”occupied” to the Wayside Controller. 3. If any signals get sent from the Wayside controller they will be lost till the fault is resolved.
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

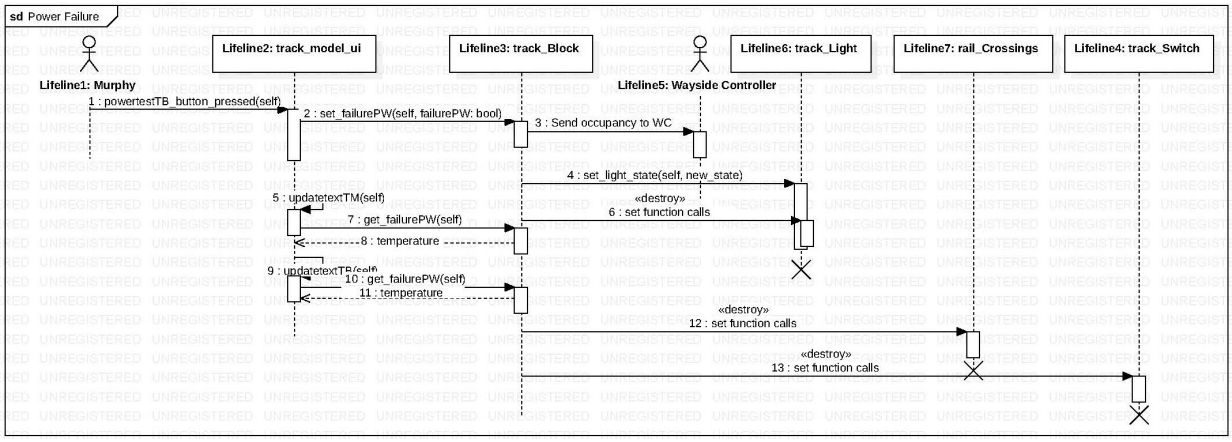

Figure 64: Track Model Power Failure Handling Sequence Diagrams

Train Departure Handling

Train Departure Handling
User Story: As the Train Model, I want to receive the number of passengers that are on the train. As the CTC, I want to receive the number of tickets bought for each train.
Acceptance Criteria: The Train Model and CTC receive tickets bought for the associated train.

Train Departure Handling	
Actors	CTC and Train Controller
Preconditions	None
Postconditions	None.
Normal Flow	1. Tickets are bought at the station 2. The train is sent the number of tickets for that train. 3. The CTC is sent the number of tickets bought. 4. Ticket Sales are updated on the Track Model User Interface.
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

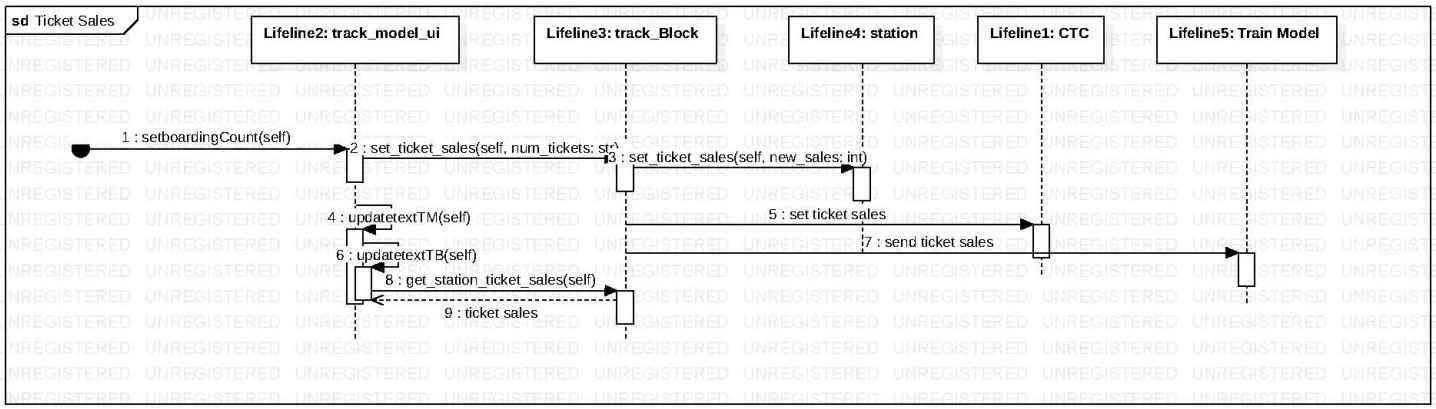

Figure 65: Track Model Ticket Handling Sequence Diagrams

Train Departure Handling

Train Departure Handling
User Story: As the Train Model, I want to receive the number of passengers that are on the train. As the CTC, I want to receive the number of tickets bought for each train.
Acceptance Criteria: The Train Model and CTC receive tickets bought for the associated train.

Train Departure Handling	
Actors	CTC and Train Controller
Preconditions	None
Postconditions	None.
Normal Flow	1. Tickets are bought at the station 2. The train is sent the number of tickets for that train. 3. The CTC is sent the number of tickets bought. 4. Ticket Sales are updated on the Track Model User Interface.
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

Train model user Stories

Authority
User Story: As the Train Model, I want to receive authority and know whether I am allowed to continue moving along the track from the Track Model.
Acceptance Criteria: The Train Model receives authority and adjusts their continued distance accordingly.

Authority and Track Lights
User Story: As the train model, I want to come to a stop before I run into a train, broken rail, or faulty line..
Acceptance Criteria: Track Model Authority is changed to stop the train, or a track light stops the train.

Beacons
User Story: As the Train Model I want to receive beacon information on where I am heading next to display to the passengers from the Track Model.
Acceptance Criteria: Track Model beacon is read and information is displayed to the passengers.

Beacons
User Story: As the train model I want to receive rail information from this beacon covering block information to the next beacon. Provided by the Track Model.
Acceptance Criteria: Track Model beacon is read and information is used to adjust power or ping for next authority.

Beacons
User Story: As the train model I want to receive information on which side of the train the passengers should exit on.
Acceptance Criteria: Track Model beacon is read and information is used to allow passengers off on the correct side of the train.

Murphy user Stories

Broken Rail Fault
User Story: As Murphy, I want to break a rail.
Acceptance Criteria: The track model displays a Broken Rail Failure, and sends block occupancy to the Wayside Control.

Track Circuit Failure
User Story: As Murphy, I want to cause a Track Circuit Failure in the system.
Acceptance Criteria: The track model displays a Track Circuit Failure, and sends block occupancy to the Wayside Control.

Power Failure
User Story: As Murphy, I want to cause a power failure in the system.
Acceptance Criteria: The track model displays a Power Failure, and sends block occupancy to the Wayside Control.

Track Model User Stories

Blue Button Pressed
User Story: As the Track Model User, I want to click on the blue track button and have the blue track show up in the Graphics Viewer.
Acceptance Criteria: The track model user interface displays the interactive map of the blue line when the button is pressed.

Red Button Pressed
User Story: As the Track Model User, I want to click on the red track button and have the red track show up in the Graphics Viewer.
Acceptance Criteria: The track model user interface displays the interactive map of the red line when the button is pressed.

Green Button Pressed
User Story: As the Track Model User, I want to click on the green button and have the green track show up in the Graphics Viewer.
Acceptance Criteria: The track model user interface displays the interactive map of the green line when the button is pressed.

Block Object on interactive-map is pressed
User Story: As the Track Model User, I want to click on a track block on the interactive map and have the blocks information show up. Including information from stations, crossings, switches, and signals that exist on that block.
Acceptance Criteria: The track model user interface displays the above information.

Display Signal Colors
User Story: As the Track Model User, I visually want to see the most current signal status(colors).
Acceptance Criteria: The track model user interface changes colors of lights as they are changed, visually. Also updates the information in text.

Display Switch States
User Story: As the Track Model User, I visually want to see the most current switch connections.
Acceptance Criteria: The track model user interface changes colors of switches as they are changed, visually. Also, updates the information in text.

Display crossing States
User Story: As the Track Model User, I visually want to see the most current crossing state.
Acceptance Criteria: The track model user interface changes colors of crossings as they are changed, visually. Also, updates the information in text.

Faults
User Story: As the Track Model User, I want to see a list of all blocks with faults. I want to see each block that has a fault change be displayed visually, and if I select the block I want to see what the fault is.
Acceptance Criteria: The track model UI updates a list of all blocks and displays their fault state. While updating each block visually, and if selected display the fault in text.

Imperial
User Story: As the Track Model User, I want to see all the information displayed in Imperial units.
Acceptance Criteria: The track model UI displays information in imperial, and properly does conversions on the backend.

Wayside Controller Actor user stories

Occupancy
User Story: As the Wayside Control, I want to receive occupancy of each block as it updates by the Track Model.
Acceptance Criteria: The track model sends the proper occupancy to the Wayside Control.

Switch Connections Change
User Story: As the Wayside Control, I want to send a signal to change a switch connection.
Acceptance Criteria: The track model sends response confirming switch changed.

Light Color Change
User Story: As the Wayside Control, I want to send the track Model a signal to change a light state.
Acceptance Criteria: The track model sends response confirming switch changed.

Crossing State Change
User Story: As the Wayside Control, I want to send the track Model a crossing to change the state.
Acceptance Criteria: The track model sends response confirming crossing changed.

Authority
User Story: As the Wayside Control, I want to send the track Model authority to send to trains at any time.
Acceptance Criteria: The track model sends response confirming changed.

3.5 Train Model

3.5.1 Design Overview

The purpose of the Train Model is to act like a train in a train system and simulate a moving train on a railroad. It is made up of train lights, train doors, train brakes. This module communicates back and forth with the Train Controller to ensure accurate train metrics. The Train Controller sends suggested speed and power command to the Train Model. The Train Controller also sets the light, door and brake states of the Train Model. The Train Controller decides what the temperature is set to according to the current temperature that is sent to the Train Controller by the Train Model. The Train Controller also determines when and what is being announced to passengers aboard the Train Model. The Train Controller receives information such as current speed, and actual power consumption to ensure Train Controller is sending the correct power command. The Train Model also receives information like coded beacon while passing beacons from the Track Model. The Track Model also sends authority and suggested speed from the CTC to Train Model via the track circuits. The tickets sales that the Track Model receives from station sales machines is also sent to the Train Model.

3.5.2 Class Diagram

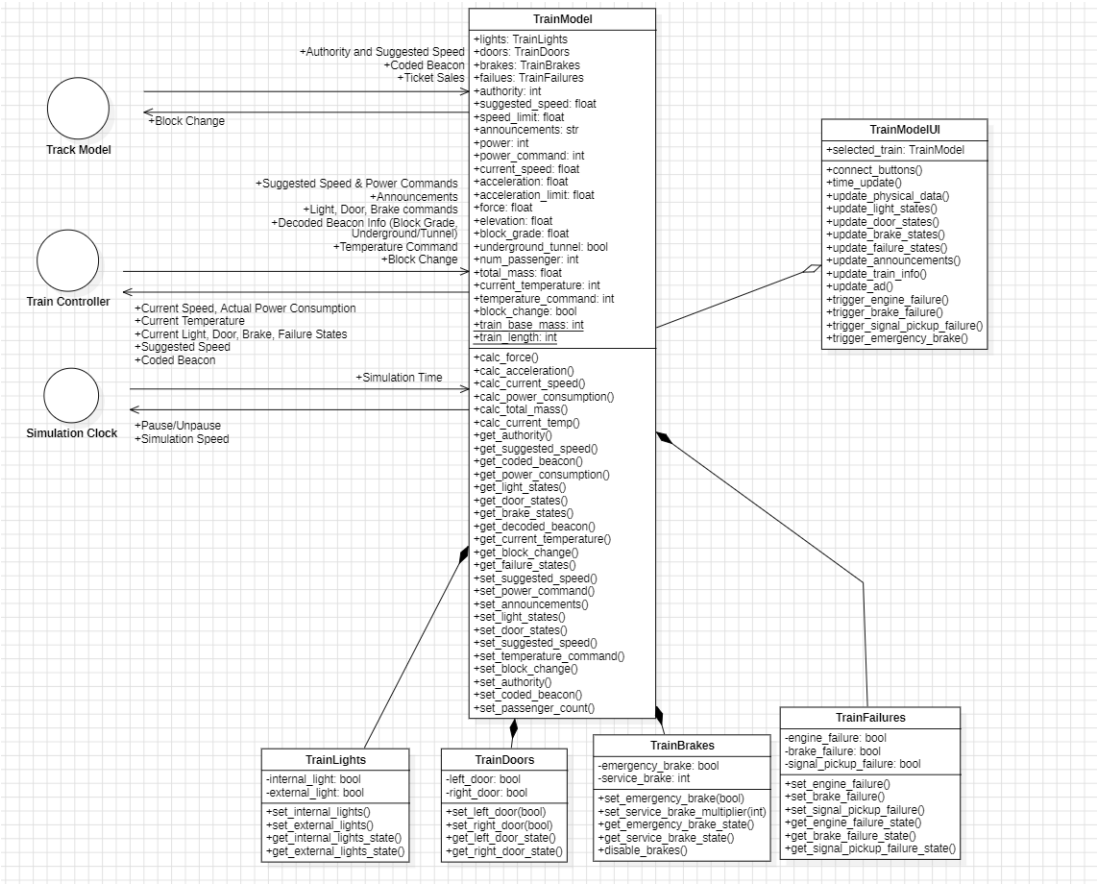

Figure 66: Train Model Class Diagram

TrainModel	
Description: Controller class for the temperature, speed, force, and acceleration for the train model module. Responsible for interacting with Track Model, Train Controller and GUI.	
Attributes	
Name	Description
Force	Stores train's current force
Total Mass	Stores train's total mass
Acceleration	Stores the train's current acceleration
Speed Limit	Store the current block's speed limit
Current Speed	Store the train's current speed
Suggested Speed	Store the train controller's last suggested speed
Block Grade	Store the current block's block grade
Power	Store the train's current power
Power Command	Store the train controller's last power command
Passenger Count	Store the train's current passenger count
Temperature	Store the train's current temperature
Temperature Command	Store the train controller's last temperature command
Block Change	Store if the train is on the next block
Responsibilities	
Name	Collaborator
Calculate Force	Self, Train Failures, Train Controller
Calculate Acceleration	Self
Calculate Current Speed	Self
Calculate Power Consumption	Self, Train Controller
Calculate Total Mass	Self, Track Controller
Calculate Current Temperature	Self, Train Controller

TrainModelUI	
Description: Graphical user interface class for train model module. Responsible for allowing the user to interact with the train emergency button and display of advertisements.	
Attributes	
Name	Description
Train Model	Stores current selected train instance
Responsibilities	
Name	Collaborator
Connect Buttons	User, Self
Time Update	Sim Clock, Self
Trigger Engine Failure	User, Self, TrainModel
Trigger Brake Failure	User, Self, TrainModel
Trigger Signal Pickup Failure	User, Self, TrainModel
Trigger Emergency Brake	User, Self, TrainModel

TrainLights	
Description: Storage class responsible for storing all relevant information about train lights	
Attributes	
Name	Description
Internal Lights	Stores the train's internal light state as a bool
External Lights	Stores the train's external light state as a bool
Responsibilities	
Name	Collaborator
Set and get Internal Lights State	Train Model, Self
Set and get External Lights State	Train Model, Self

TrainDoors	
Description: Storage class responsible for storing all relevant information about train doors	
Attributes	
Name	Description
Left Door	Stores the train's left door state as a bool
Right Door	Stores the train's right door state as a bool
Responsibilities	
Name	Collaborator
Set and get Left Door State	Train Model, Self
Set and get Right Door State	Train Model, Self

TrainBrakes	
Description: Storage class responsible for storing all relevant information about train brakes	
Attributes	
Name	Description
Emergency Brake	Stores the train's emergency brake state as a bool
Service Brake	Stores the train's service brake as a percent of the service brake activated
Responsibilities	
Name	Collaborator
Set and get Emergency Brake State	Train Model, Self
Set and get Service Brake State	Train Model, Self
Disable Brakes	Train Model, Train Failures, Self

TrainFailures	
Description: Storage class responsible for storing all relevant information about train failures	
Attributes	
Name	Description
Engine Failure	Stores the train's engine failure states as a bool
Brake Failure	Stores the train's brake failure states as a bool
Signal Pickup Failure	Stores the train's signal pickup failure states as a bool
Responsibilities	
Name	Collaborator
Set and get Engine Failure State	Train Model, Self
Set and get Brake Failure State	Train Model, Self
Set and get Signal Pickup State	Train Model, Self

3.5.3 Use Cases

The main three use case actors for the Train Model module are the Track Model, Train Controller and Murphy.

A fourth actor called the quality assurance tester was added. This actor will act as a substitute for the Train Controller for checking correct reception from Train Controller. The Train Model's use case diagram is seen in Figure 65. Below the use case diagram, there is a user story, use case description, and sequence diagram for each non-trivial use case.

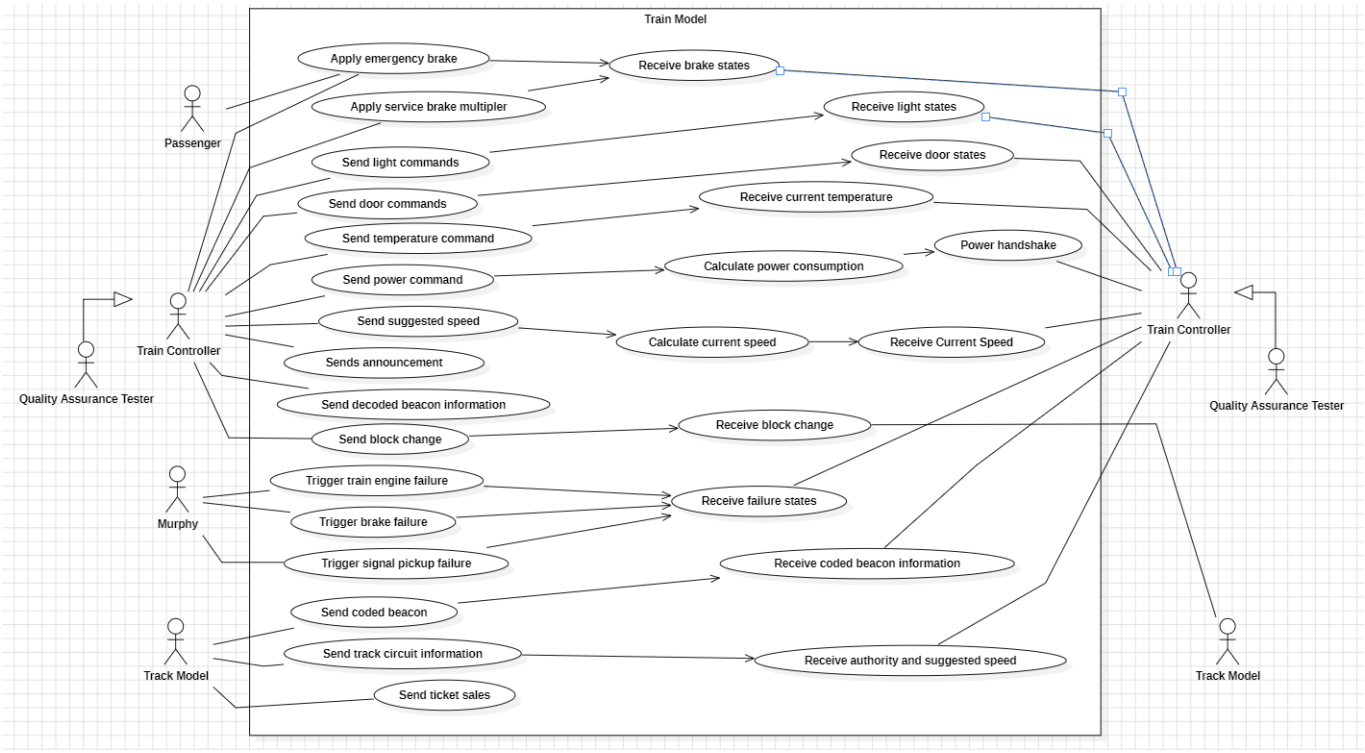

Figure 67: Train Model Use Case Diagram

Passenger or Train Controller Applies Emergency Brake:

Passenger or Train Controller Applies Emergency Brake
User Story: As a Passenger or Train Controller, I want to apply emergency brake when an emergency situation arises.
Acceptance Criteria: Given that the Passenger or Train Controller needs to manually apply the emergency brake in case of an emergency, when one of the two clicks the emergency brake button, the Train Model emergency brakes will be activated.

Passenger or Train Controller Applies Emergency Brake	
Actors	Passenger, Train Model, Train Controller
Preconditions	Passenger or Train Controller needs to apply emergency brake and emergency brake is not already activated and brake failure is not activated
Postconditions	The Train Model sets the emergency brake and sends the new brake state to Train Controller
Normal Flow	1. Passenger hits the emergency brake button on the UI 2. Train Model sets emergency brake 3. Train Model updates acceleration due to new brake status 4. Update brake state on Train Model UI 5. Notify Train Controller of new brake status
Alternative Flow	1. Train Controller hits the emergency brake 2. Train Controller notify Train Model that the emergency brake should be on 3. Update brake state on Train Model UI 4.Train Model sets emergency brake 5. Train Model updates acceleration due to new brake status
Exceptional Flow	Emergency brake is already activated

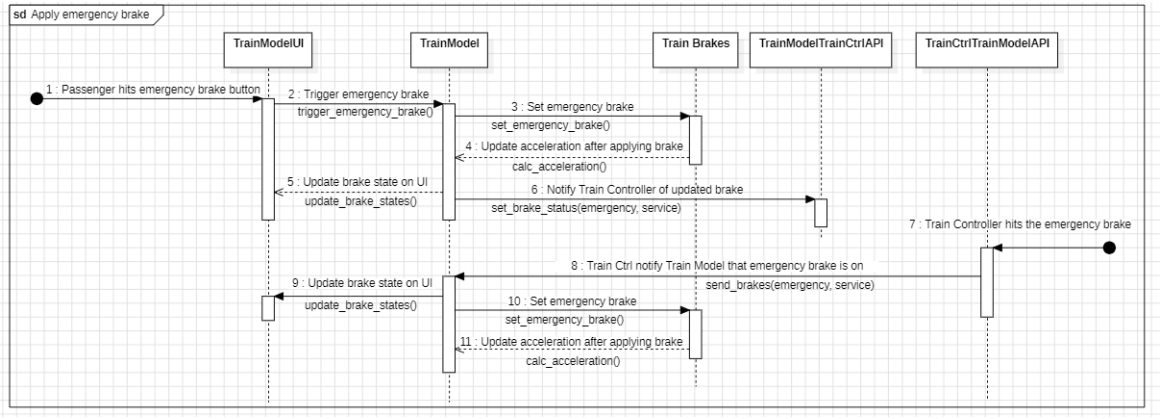

Figure 68: Sequence Diagram for ”Applying Emergency Brake”

Train Controller Applies Service Brake Multiplier:

Train Controller Applies Service Brake Multiplier
User Story: As a Train Controller, I want to apply service brake multiplier to slow the Train Model down as it approaches stations and as authority approaches zero.
Acceptance Criteria: Given that the Train Controller needs to manually apply the service brake multiplier when the Train should slow down, when Train Controller sets the service brake multiplier, the Train Model service brakes will be set to the multiplier.

Train Controller Applies Service Brake Multiplier	
Actors	Train Model, Train Controller
Preconditions	Train Controller needs to apply service brake multiplier and brake failure is not activated
Postconditions	The Train Model sets the service brake multiplier and sends the new brake state to Train Controller
Normal Flow	1. Train Controller applies the service brake multiplier 2. Train Controller sends the service brake multiplier to Train Model 3. Train Model sets service brake multiplier 4. Update brake state on Train Model UI 5. Train Model updates acceleration due to new brake status
Exceptional Flow	Brake failure is activated

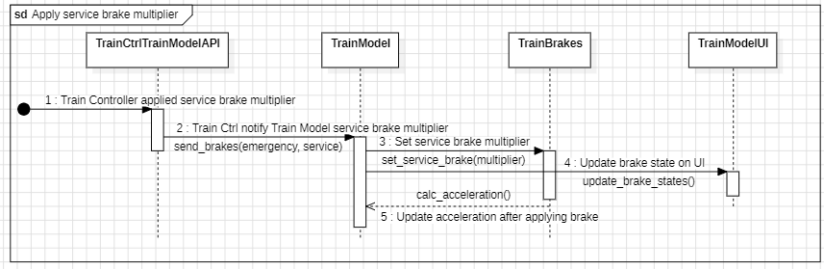

Figure 69: Sequence Diagram for ”Applying Service Brake”

Train Controller Receives Brake States:

Train Controller Receive Brake States
User Story: As a Train Controller, I want to receive brake states from Train Model to check if the passenger has applied emergency brake already.
Acceptance Criteria: Given that the Train Controller wants to check brake states if brake states was changed by the Passenger, the Train Model will send the brake states to the Train Controller.

Train Controller Receives Brake States	
Actors	Train Model, Train Controller
Preconditions	Train Controller needs to see if the brake states have been changed by the Passenger
Postconditions	The Train Model sends the current brake state to Train Controller
Normal Flow	1. Train Controller calls for brake state update 2. Train Model gets brake states 3. Train Model sends brake states to Train Controller
Exceptional Flow	None

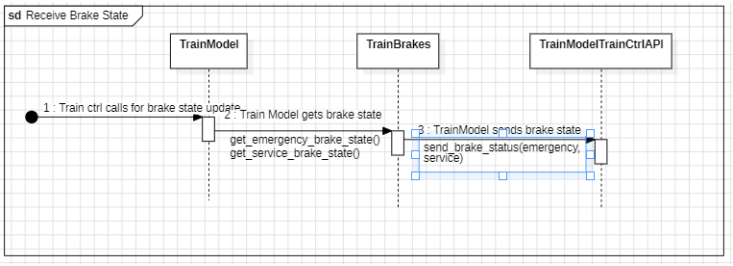

Figure 70: Sequence Diagram for ”Receive Brake States”

Train Controller Send Door Commands:

Train Controller Send Door Commands
User Story: As a Train Controller, I want to open doors of the train to allow passengers to exit when at a station.
Acceptance Criteria: Given that the Train Controller wants to send door commands to the Train Model, the Train Model will receive and set the door states according to the commands.

Train Controller Send Door Commands	
Actors	Train Model, Train Controller
Preconditions	Train Controller needs to send door commands
Postconditions	The Train Model set the door states to Train Controller door commands
Normal Flow	1. Train Controller sets and sends door commands 2. Train Model sets door states 3. Update door states on UI
Exceptional Flow	None

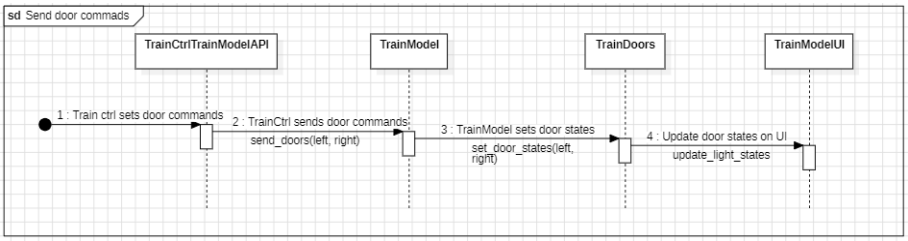

Figure 71: Sequence Diagram for ”Send Door Commands”

Train Controller Send Light Commands:

Train Controller Send Light Commands
User Story: As a Train Controller, I want to open lights when the train is in a tunnel or underground.
Acceptance Criteria: Given that the Train Controller wants to send light commands to the Train Model, the Train Model will receive and set the light states according to the commands.

Train Controller Send Door Commands	
Actors	Train Model, Train Controller
Preconditions	Train Controller needs to send light commands if the train is underground or in a tunnel
Postconditions	The Train Model set the door states to Train Controller light commands
Normal Flow	1. Train Controller sets and sends light commands 2. Train Model sets light states 3. Update light states on UI
Exceptional Flow	None

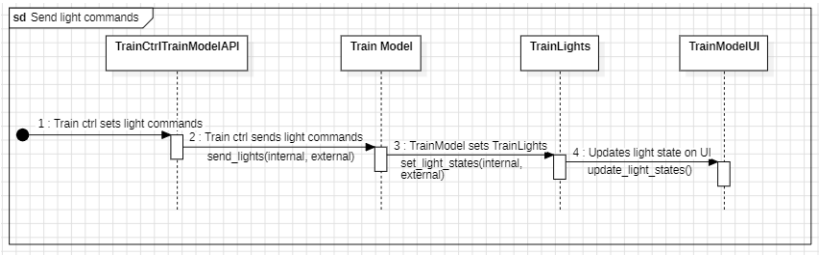

Figure 72: Sequence Diagram for ”Send Light Commands”

Train Controller Send Temperature Commands:

Train Controller Send Temperature Commands
User Story: As a Train Controller, I want to send temperature command depending on the train’s current temperature.
Acceptance Criteria: Given that the Train Controller wants to send temperature command to the Train Model, the Train Model will receive and set the power to the temperature command.

Train Controller Send Temperature Command	
Actors	Train Model, Train Controller
Preconditions	Train Controller needs to send temperature command if the train’s current temperature is not adequate for passenger comfort
Postconditions	The Train Model set the temperature command to Train Controller’s temperature command
Normal Flow	1. Train Controller sets and sends temperature command 2. Train Model sets temperature command 3. Calculate current temperature 4. Update current temperature on UI 5. Train Model sends current temperature to Train Controller
Exceptional Flow	Power command is greater than maximum power of train

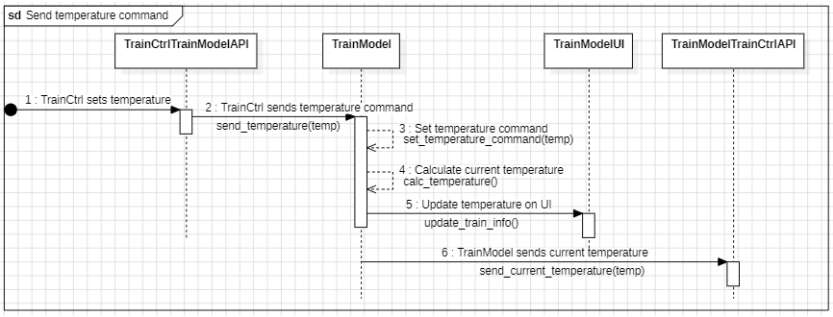

Figure 73: Sequence Diagram for ”Send Temperature Commands”

Train Controller Send Power Commands:

Train Controller Send Power Commands
User Story: As a Train Controller, I want to send power depending on the train's current speed and CTC's suggested speed.
Acceptance Criteria: Given that the Train Controller wants to send power command to the Train Model, the Train Model will receive and set the power to the power command.

Train Controller Send Power Command	
Actors	Train Model, Train Controller
Preconditions	Train Controller needs to send power command if the train's current speed is not equal to the suggested speed
Postconditions	The Train Model set the power to Train Controller's power command
Normal Flow	1. Train Controller sets and sends power command 2. Train Model sets power command 3. Update power on UI 4. Calculate power consumption 4. Power handshake to confirm correct power command
Exceptional Flow	Power command is greater than maximum power of train

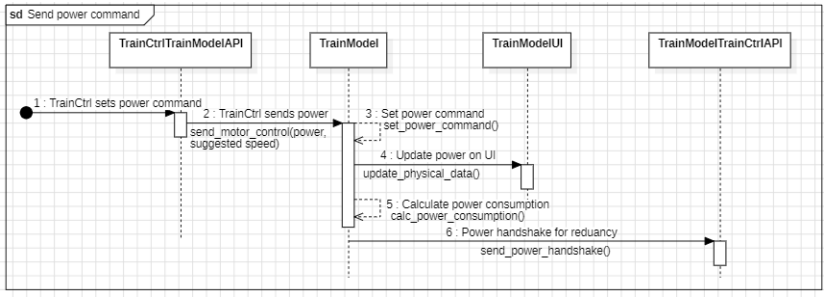

Figure 74: Sequence Diagram for ”Send Power Commands”

Train Controller Send Suggested Speed:

Train Controller Send Suggested Speed
User Story: As a Train Controller, I want to send suggested speed to control the speed of the train.
Acceptance Criteria: Given that the Train Controller wants to send suggested speed to the Train Model, the Train Model will receive set its suggested speed.

Train Controller Send Suggested Speed	
Actors	Train Model, Train Controller
Preconditions	Train Controller needs to send suggested speed if the train's current speed is too fast or too slow for the block's speed limit or if train isn't up to schedule.
Postconditions	The Train Model set the suggested speed to Train Controller's suggested speed
Normal Flow	1. Train Controller sets and sends suggested speed 2. Train Model sets suggested speed 4. Calculate current speed 5. Update suggested speed and current speed on UI 6. Send current speed to Train Controller
Exceptional Flow	Suggested speed is greater than the maximum speed of the train

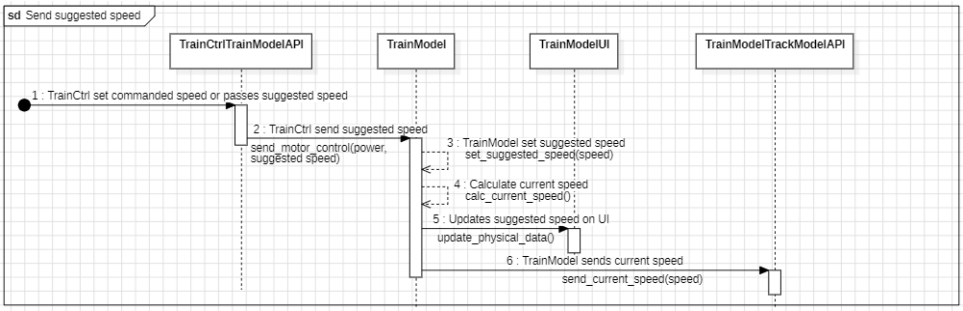

Figure 75: Sequence Diagram for "Send Suggested Speed"

Train Controller Send Announcement:

Train Controller Send Announcement
User Story: As a Train Controller, I want to send announcement to display to the passengers on the train to let them know what the next stop is.
Acceptance Criteria: Given that the Train Controller wants to send announcement to the Train Model, the Train Model will receive and display the announcement.

Train Controller Send Announcement	
Actors	Train Model, Train Controller
Preconditions	Train Controller needs to send announcement if the train is approaching the next stop.
Postconditions	The Train Model set and display the announcement
Normal Flow	1. Train Controller sets and sends announcement 2. Train Model sets announcement 3. Update announcement on UI
Exceptional Flow	None

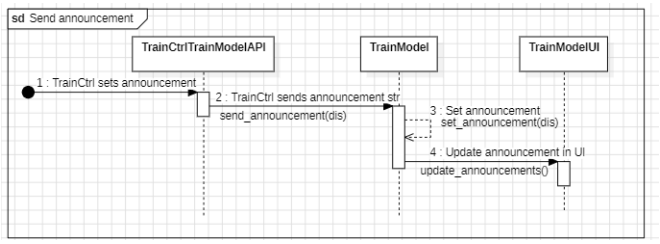

Figure 76: Sequence Diagram for ”Send Announcement”

Train Controller Send Decoded Beacon Information:

Train Controller Send Decoded Beacon Information
User Story: As a Train Controller, I want to send decoded beacon information to let Train Model determine speed accurately based on the block.
Acceptance Criteria: Given that the Train Controller wants to send decoded beacon information to the Train Model, the Train Model will receive the information.

Train Controller Send Decoded Beacon Information	
Actors	Train Model, Train Controller
Preconditions	Train Controller needs to send decoded beacon information if Train Model does not have block data.
Postconditions	The Train Model set block data information.
Normal Flow	1. Train Controller decodes the beacon and sends block data 2. Train Model sets block grade and underground and tunnel status 3. Update block information on UI
Exceptional Flow	None

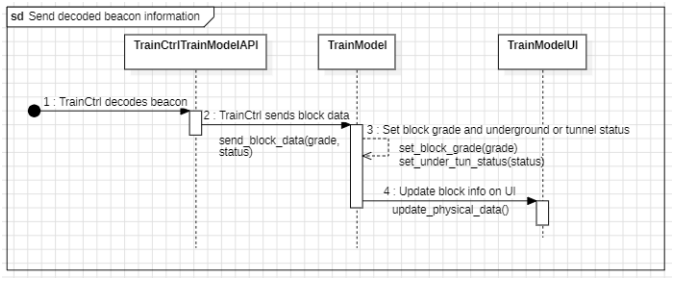

Figure 77: Sequence Diagram for ”Send Decoded Beacon Information”

Train Controller Send Block Change:

Train Controller Send Block Change
User Story: As a Train Controller, I want to send block change if the train on the next block to let Track Model determine block occupancy.
Acceptance Criteria: Given that the Train Controller wants to send block change to the Train Model, the Train Model will set the block change and send block change to Track Model.

Train Controller Send Block Change	
Actors	Train Model, Train Controller, Track Model
Preconditions	Train Controller needs to send block change if train is on the next block.
Postconditions	The Train Model set block change and send to Track Model.
Normal Flow	1. Train Controller calculates if train has passed the current block and sends block change 2. Train Model sets block change 3. Train Model sends block change to Track Model
Exceptional Flow	None

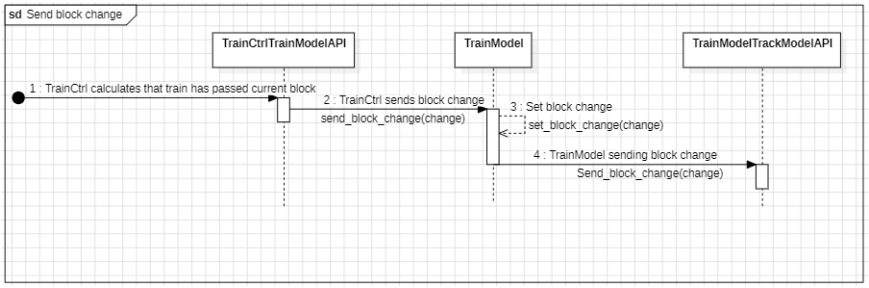

Figure 78: Sequence Diagram for ”Send Block Change”

Murphy Trigger Engine Failure:

Murphy Trigger Engine Failure
User Story: As a Murphy, I want to trigger engine failure to break train engine and cause chaos and laugh at the consequences.
Acceptance Criteria: Given that Murphy needs to trigger engine failure, the Train Model will set the engine failure and send engine failure to Train Controller.

Murphy Triggers Engine Failure	
Actors	Train Model, Train Controller, Murphy
Preconditions	Murphy needs to trigger engine failure.
Postconditions	The Train Model set engine failure and send to Train Controller.
Normal Flow	1. Murphy triggers Engine Failure 2. Train Model sets engine failure and updates UI 3. Train Model sends engine failure to Train Controller 4. Train Controller sends power command to Train Model 5. Train Model recalculate current speed
Exceptional Flow	Engine failure is already activated

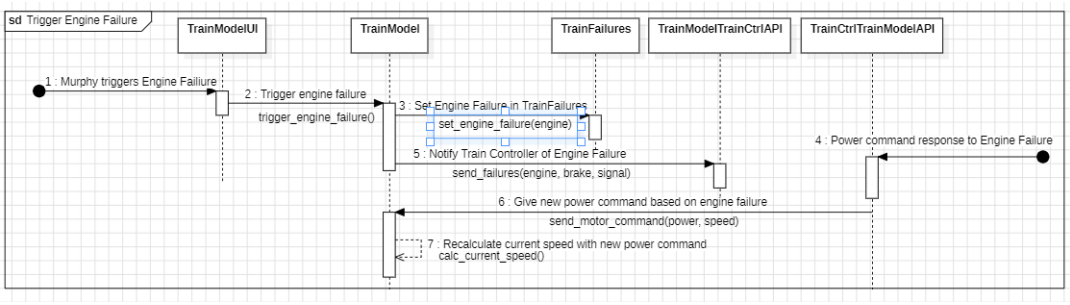

Figure 79: Sequence Diagram for ”Trigger Engine Failure”

Murphy Trigger Brake Failure:

Murphy Trigger Brake Failure
User Story: As a Murphy, I want to trigger brake failure to break train brake and cause chaos and laugh at the consequences.
Acceptance Criteria: Given that Murphy needs to trigger brake failure, the Train Model will set the brake failure and send brake failure to Train Controller.

Murphy Trigger Brake Failure	
Actors	Train Model, Train Controller, Murphy
Preconditions	Murphy needs to trigger brake failure.
Postconditions	The Train Model set brake failure and set brake states and send to Train Controller.
Normal Flow	1. Murphy triggers Brake Failure 2. Train Model sets Brake failure 3. Train Model disables all brakes 4. Update brake state and failure state on UI 5. Train Model sends brake failure to Train Controller 6. Train Model sends brake states to Train Controller
Exceptional Flow	Brake failure is already activated

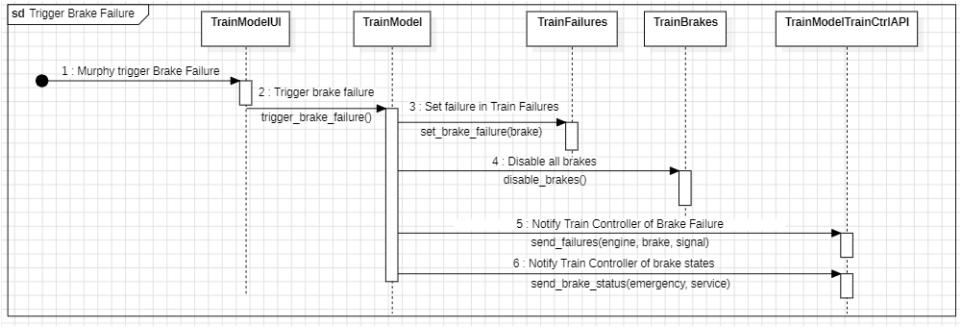

Figure 80: Sequence Diagram for ”Trigger Brake Failure”

Murphy Trigger Signal Pickup Failure:

Murphy Trigger Signal Pickup Failure
User Story: As a Murphy, I want to trigger signal pickup failure to break track circuit inputs and cause chaos and laugh at the consequences.
Acceptance Criteria: Given that Murphy needs to trigger signal pickup failure, the Train Model will set the signal pickup failure and send signal pickup failure to Train Controller.

Murphy Trigger Brake Failure	
Actors	Train Model, Train Controller, Murphy
Preconditions	Murphy needs to trigger brake failure.
Postconditions	The Train Model set signal pickup failure and send to Train Controller. The authority and suggested speed is no longer being updated.
Normal Flow	1. Murphy triggers Signal pickup Failure 2. Train Model sets Signal Pickup Failure and update UI 3. Train Model sends Signal Pickup Failure to Train Controller
Exceptional Flow	Signal pickup failure is already activated

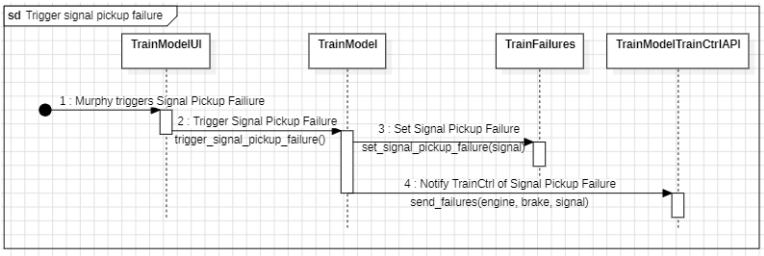

Figure 81: Sequence Diagram for ”Trigger Signal Pickup Failure”

Track Model Send Coded Beacon:

Track Model Send Coded Beacon	
	User Story: As a Track Model, I want to send beacon package so the Train Model can receive it and the Train Controller can decode the package and have block data.
Acceptance Criteria: Given that Track Model needs to send beacon package, the Train Model will receive the beacon package and send beacon package to Train Controller.	

Track Model Sends Beacon Package	
Actors	Train Model, Train Controller, Track Model
Preconditions	Track Model sends the beacon package.
Postconditions	The Train Model receives and sends beacon package failure Train Controller.
Normal Flow	1. Train Model passes by a beacon 2. Track Model sends beacon 3. Train Model sends beacon to Train Controller
Exceptional Flow	Signal pickup failure is activated

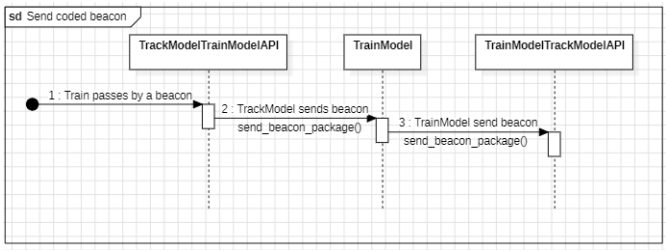

Figure 82: Sequence Diagram for ”Coded Beacon”

Track Model Send Track Circuit Information:

Track Model Send Track Circuit Information
User Story: As a Track Model, I want to send track circuit information so the Train Model can have authority and suggested speed to operate correctly.
Acceptance Criteria: Given that Track Model needs to send track circuit information, the Train Model will receive the authority and suggested speed and send both to Train Controller.

Track Model Sends Track Circuit Information	
Actors	Train Model, Train Controller, Track Model
Preconditions	Track Model sends the track circuit information.
Postconditions	The Train Model receives and sends track circuit information to Train Controller.
Normal Flow	1. CTC sends authority and suggested to Track Model 2. Track Model sends authority and suggested speed to Train Model 2. Track Model sets authority and updates UI 3. Train Model sends authority and suggested speed to Train Controller
Exceptional Flow	Signal pickup failure is activated

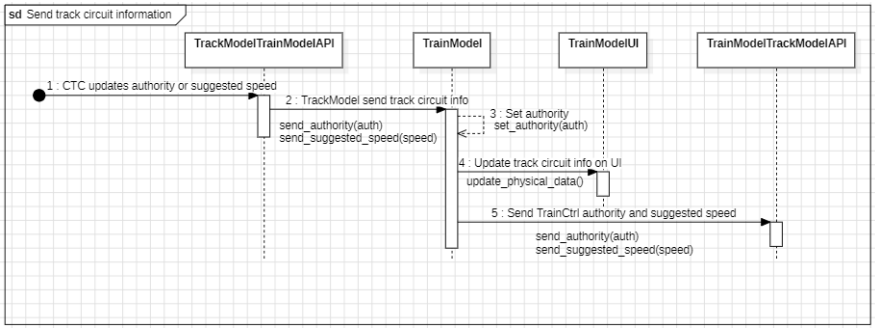

Figure 83: Sequence Diagram for ”Send Track Circuit Information”

Track Model Send Ticket Sales:

Track Model Send Ticket Sales
User Story: As a Track Model, I want to send ticket sales so the Train Model can have passenger count and calculate train mass and speed accurately.
Acceptance Criteria: Given that Track Model needs to send ticket sales, the Train Model will receive the ticket sales and update train mass.

Track Model Sends Track Circuit Information	
Actors	Train Model, Track Model
Preconditions	Track Model sends the ticket sales.
Postconditions	The Train Model receives ticket sales and updates passenger count.
Normal Flow	1. Track Model is sent ticket sales 2. Track Model sends ticket sales if train enter station 3. Train Model increases passenger count and calculates train mass 4. Train Model updates UI
Exceptional Flow	Signal pickup failure is activated

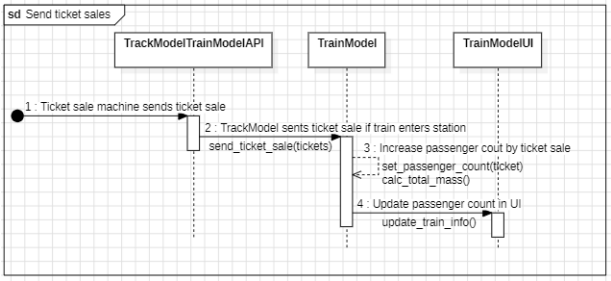

Figure 84: Sequence Diagram for ”Send Ticket Sales”

3.6 Train Controller

3.6.1 Design Overview

- The purpose of the Train Controller is to control the train and ensure the active train is running in perfect condition with the actors: train driver and train engineer. The train driver has control over the commanded speed of the train, setting the cabin temperature, toggling the doors and lights, announcing station stops, and applying brakes. The train engineer has control over the amount of power the train is outputting.

- organized list of design choices and rationales for each

3.6.2 Class Diagram

The overall class diagram for the train controller is shown in Figure . The class titled *TrainController* is the top-level class for this module. Whenever a new train is being created the train model will create an instance of the train controller.

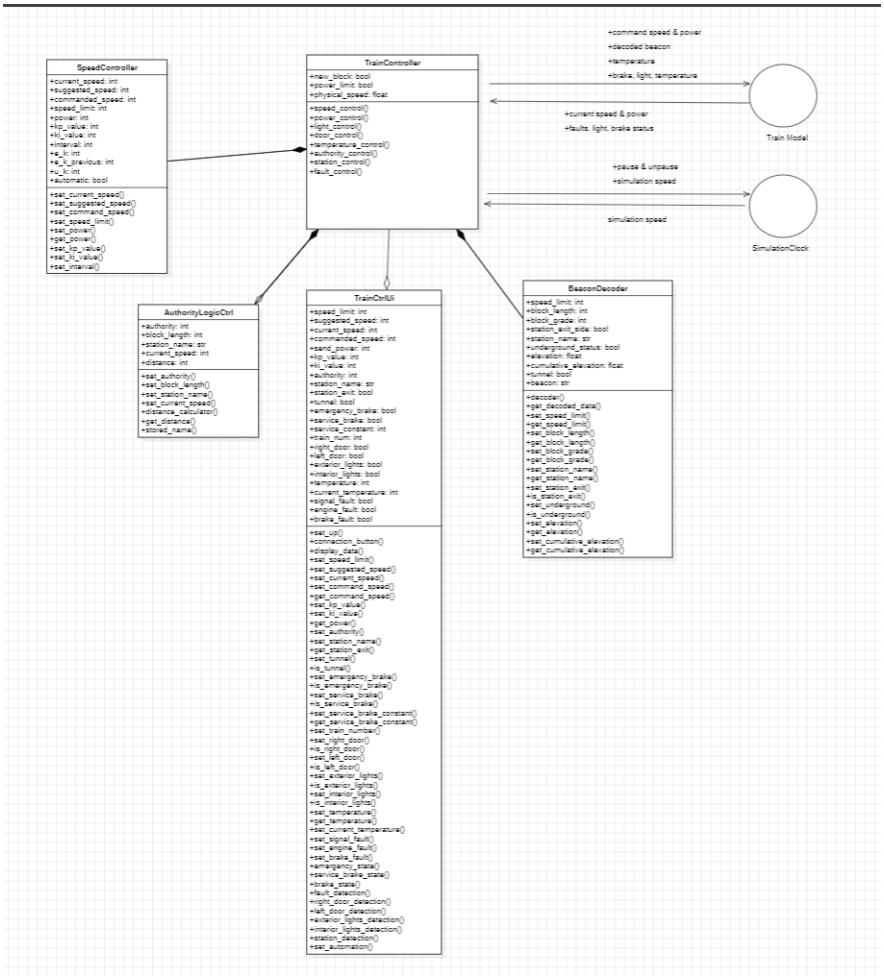

Figure 85: Train Controller Class Diagram

TrainController	
Description: Top-level class for the train controller module. Responsible for interacting with GUI and Train Model.	
Attributes	
Name	Description
newblock	Identifies if the train is going into a new block
powerlimit	Identifies the limit of the total power that the train can produce
physicallimit	Identifies the maximum speed that the train can go
Responsibilities	
Name	Collaborator
Set Commanded Speed	TrainCtrlUi, Self, Train Model
Set Suggested Speed	TrainCtrlUi, Self, Train Model
Set Speed Limit	TrainCtrlUi, Self, Train Model
Set Current Speed	TrainCtrlUi, Self, Train Model
Set Power	TrainCtrlUi, Self, Train Model
Set Authority	TrainCtrlUi, Self, Train Model
Set Temperature	TrainCtrlUi, Self, Train Model
Set Lights	TrainCtrlUi, Self, Train Model
Set Doors	TrainCtrlUi, Self, Train Model
Set Station Name	TrainCtrlUi, Self, Train Model
Set Faults	TrainCtrlUi, Self, Train Model

TrainCtrlUi	
Description: Graphical user interface class for train controller module. Responsible for allowing the user to interact with each train controller instance.	
Attributes	
Name	Description
Varibles	Look at Figure 84
Responsibilities	
Name	Collaborator
Set Up	User, Self
Set Speed Limit	User, Self
Set Suggested Speed	User, Self
Set Current Speed	User, Self
Set Command Speed	User, Self
Get Command Speed	User, Self
Set Kp Value	User, Self
Set Ki Value	User, Self
Set Power	User, Self, SimulationClock
Set Authority	User, Self
Set Station Name	User, Self
Set Tunnel	User, Self
Is Tunnel	User, Self
Set Emergency Brake	User, Self
Is Emergency Brake	User, Self
Set Service Brake	User, Self
Is Service Brake	User, Self
Set Service Brake Constant	User, Self
Get Service Brake Constant	User, Self
Set Train Number	User, Self
Set Right Door	User, Self
Is Right Door	User, Self
Set Left Door	User, Self
Is Left Door	User, Self
Set Exterior Lights	User, Self
Set Interior Lights	User, Self
Is Exterior Lights	User, Self
Is Interior Lights	User, Self
Set Temperature	User, Self
Get Temperature	User, Self
Set Current Temperature	User, Self
Set Signal Fault	User, Self
Set Brake Fault	User, Self
Set Engine Fault	User, Self
Emergency State	User, Self
Service Brake State	User, Self
Brake State	User, Self
Fault Detection	User, Self
Right Door Detection	User, Self
Left Door Detection	User, Self
Exterior Lights Detection	User, Self
Interior Lights Detection	User, Self
Station Detection	User, Self
Set Automation	User, Self
Display Data	User, Self
Connection Button	User, Self

SpeedController	
Description: Controller class responsible for calculating the power and ensure the speed is at a safe speed.	
Attributes	
Variable	Look at Figure 84
Responsibilities	
Name	Collaborator
Set Current Speed	TrainController
Set Suggested Speed	TrainController
Set Commanded Speed	TrainController
Set Speed Limit	TrainController
Get Power	TrainController
Set Kp Value	TrainController
Set Ki Value	TrainController
Set Interval	TrainController

AuthorityLogicCtrl	
Description: Controller class responsible for calculating the distance to stop and ensure the train is able to stop at station.	
Attributes	
Variable	Look at Figure
Responsibilities	
Name	Collaborator
Calc Stop	TrainController

BeaconDecoder	
Description: Controller class responsible for decoding the beacon from the train model.	
Attributes	
Variable	Look at Figure 84
Responsibilities	
Name	Collaborator
Decoder	TrainController
Set Speed Limit	TrainController
Get Speed Limit	TrainController
Set Block Length	TrainController
Get Block Length	TrainController
Set Block Grade	TrainController
Get Block Grade	TrainController
Set Station ID	TrainController
Get Station ID	TrainController
Set Station Exit	TrainController
Get Station Exit	TrainController
Set Underground	TrainController
Get Underground	TrainController
Set Elevation	TrainController
Get Elevation	TrainController
Set Cumulative Elevation	TrainController
Get Cumulative Elevation	TrainController

3.6.3 Use Cases

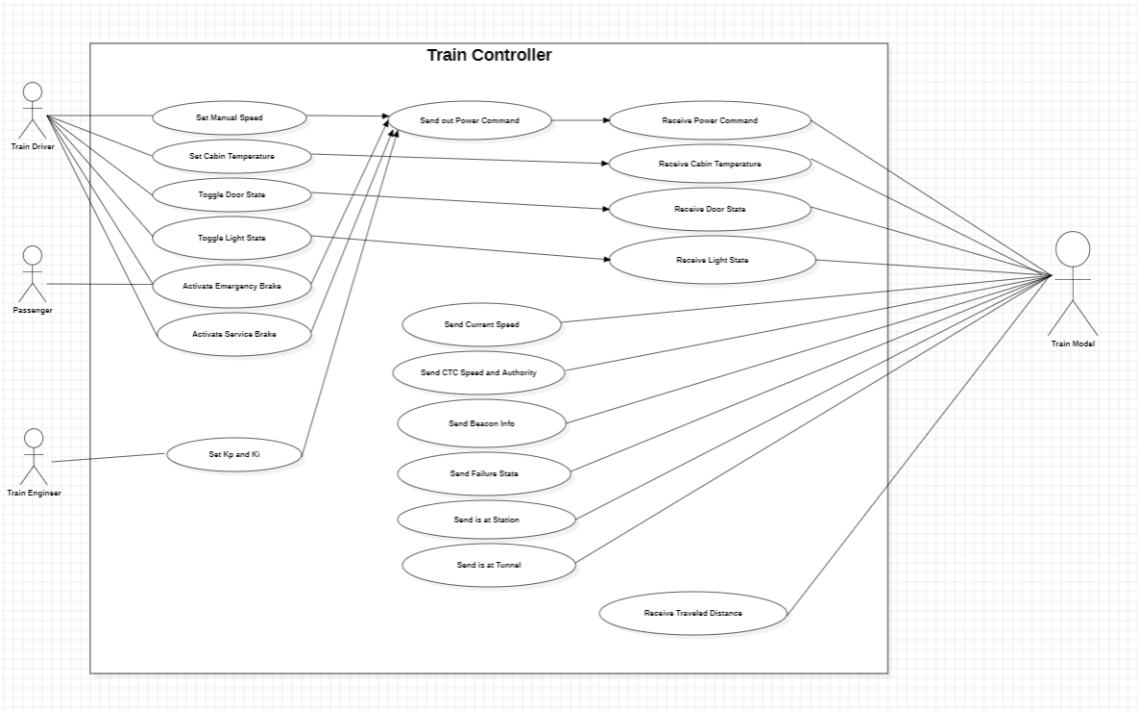

Figure 86: Train Controller Use Case Diagram

CTC send Authority:

CTC send Authority
User Story: As a CTC Office, I want to tell the train when to stop.
Acceptance Criteria: Given that CTC Office wants to the train stop at a specific stop this information is sent from track model via the track circuit to the train model.

CTC send Authority	
Actors	CTC Office
Preconditions	CTC needs the Train to Stop
Postconditions	The TrainController will take that value and do necessary calculations for the stop distance and notify the driver to slow down.
Normal Flow	1. train controller update authority 2. train controller calculates the distance 3. train controller send the distance to the ui 4. train controller display the authority
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

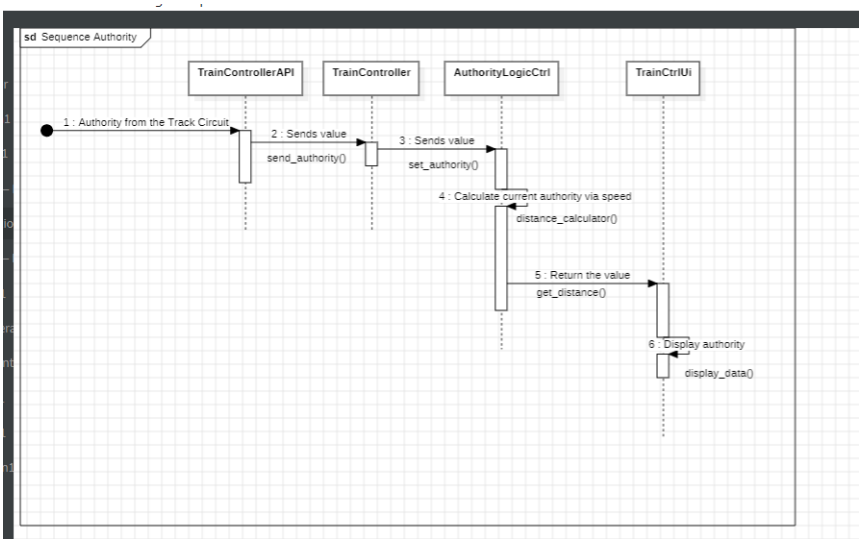

Figure 87: Train Controller Authority Sequence Diagram

Train Model Send Beacon:

Train Model Send Beacon	
User Story: As a Train Model, I would like the beacon to be decoded and provide me the necessary information about the block.	
	Acceptance Criteria: Given that the beacon has all the information about the track this information is sent from the track model via the station beacon.

Train Model Send Beacon	
Actors	Train Model
Preconditions	Train Model needs the Block Information
Postconditions	The TrainController will take the beacon and decodes the beacon and send information to the Train Model.
Normal Flow	1. train controller gets the beacon 2. train controller decodes the beacon 3. train controller sets the values 4. train controller sends the values to the Train Model
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

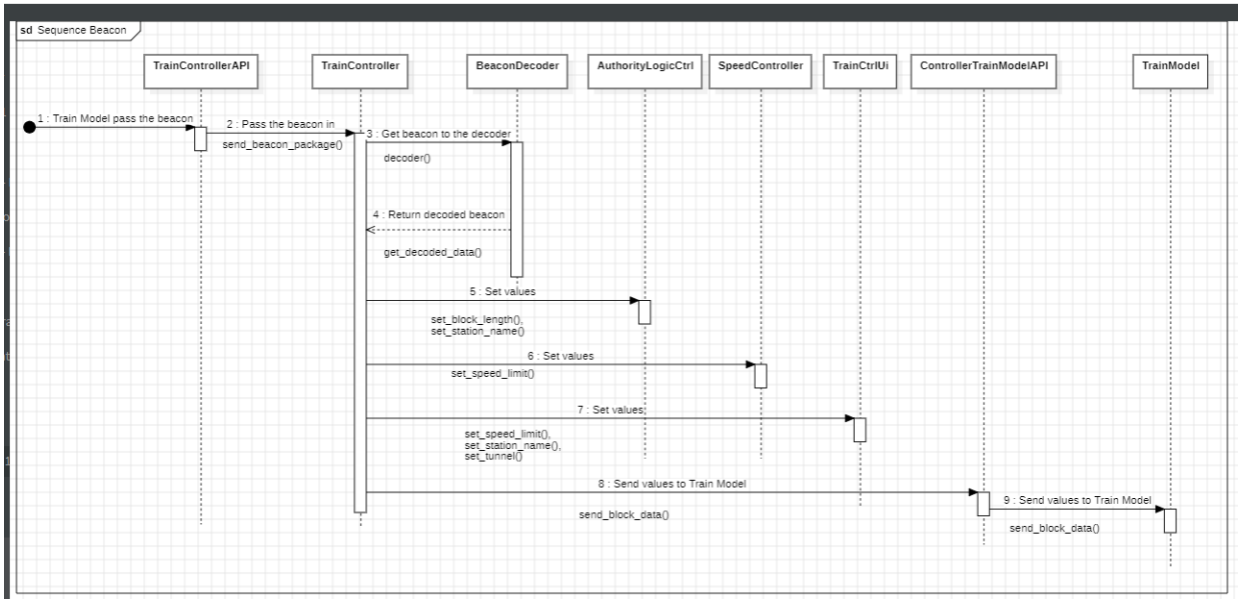

Figure 88: Train Controller beacon Diagram

Train Driver want to Slow Down or Stop:

Train Driver want to Slow Down or Stop
User Story: As a Train Driver, I would like to slow down the train or put it into an emergency stop.
Acceptance Criteria: Given that the driver sends the service or emergency brake command the train would slow down.

Train Driver want to Slow Down or Stop	
Actors	Train Driver
Preconditions	Train Driver needs to see current speed be down
Postconditions	The TrainController will take the brake command and send information to the Train Model, where it returns the current speed.
Normal Flow	1. train controller gets the brake command 2. train controller sends to the train model 3. train controller gets the current speed 4. train controller display the current speed
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

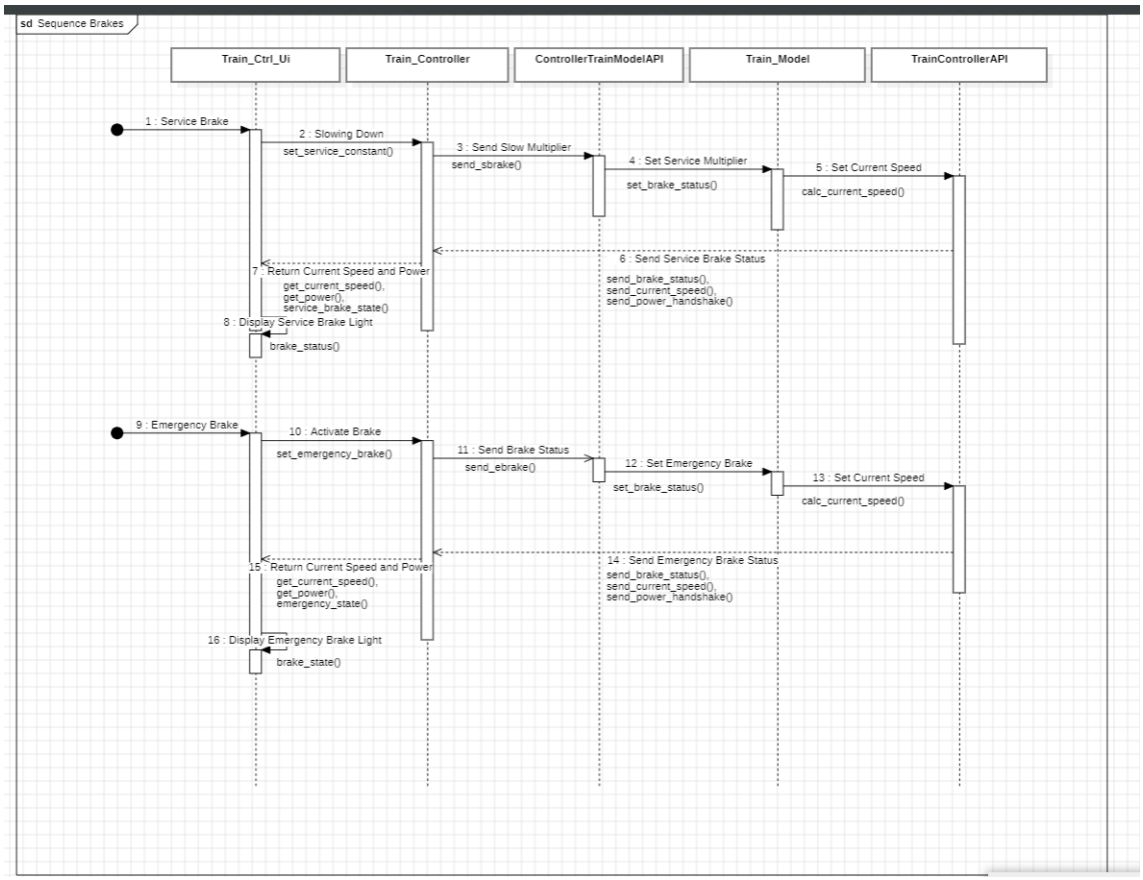

Figure 89: Train Controller Braking Sequence Diagram

Train Driver to set Command Speed:

Train Driver to set Command Speed
User Story: As a Train Driver, I want to set the command speed and the train model would speed up.
Acceptance Criteria: Given that the driver sends command speed, the train would speed up.

Train Driver to set Command Speed	
Actors	Train Driver
Preconditions	Train Driver needs to see current speed be up
Postconditions	The TrainController will take the command speed and send information to the Train Model, which returns the current speed.
Normal Flow	1. train controller gets the command speed 2. train controller sends to the train model 3. train controller gets the current speed 4. train controller display the current speed
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

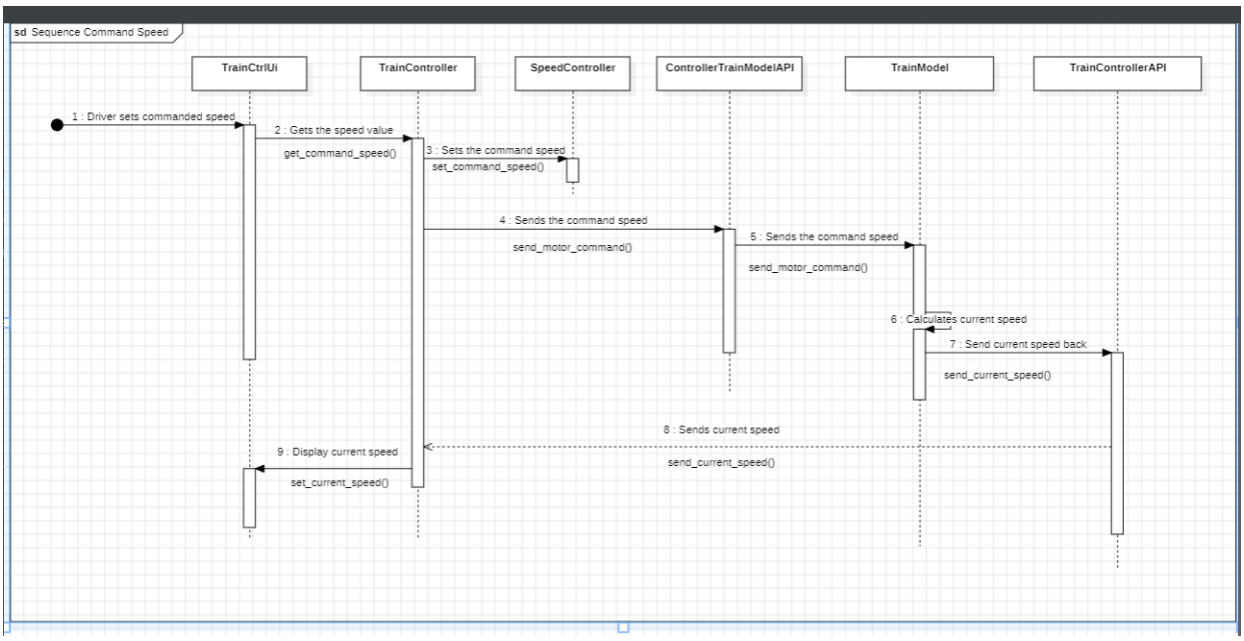

Figure 90: Train Controller Command Speed Sequence Diagram

Train Driver set Door Command:

Train Driver set Door Command
User Story: As a Train Driver, I want to set the command door and the train model would open doors.
Acceptance Criteria: Given that the driver sends command speed, the train would speed up.

Train Driver set Door Command	
Actors	Train Driver
Preconditions	Train Driver needs to set the door to be open or closed
Postconditions	The TrainController will take the door command and send information to the Train Model.
Normal Flow	1. train controller gets the door command 2. train controller sends to the train model
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

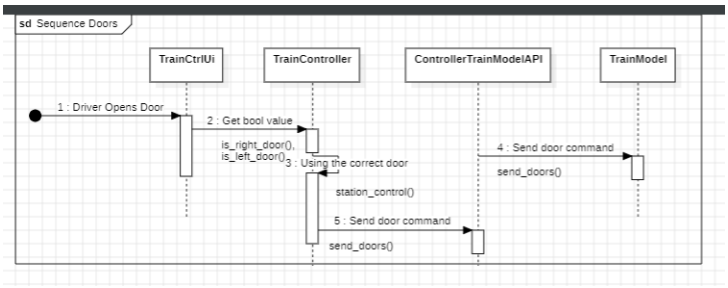

Figure 91: Train Controller Doors Sequence Diagram

Train Engineer set Power Command:

Train Driver set Power Command
User Story: As a Train Driver, I want to set the power command and the train model would speed up.
Acceptance Criteria: Given that the driver sends power command, the train would speed up.

Train Engineer set Power Command	
Actors	Train Engineer
Preconditions	Train Driver needs to set the power
Postconditions	The TrainController will take the power command and send information to the Train Model.
Normal Flow	1. train controller gets the power command 2. train controller sends to the train model 3. train controller get the current speed
Exceptional Flow	One or more of the steps in Normal Flow are incorrect or fail

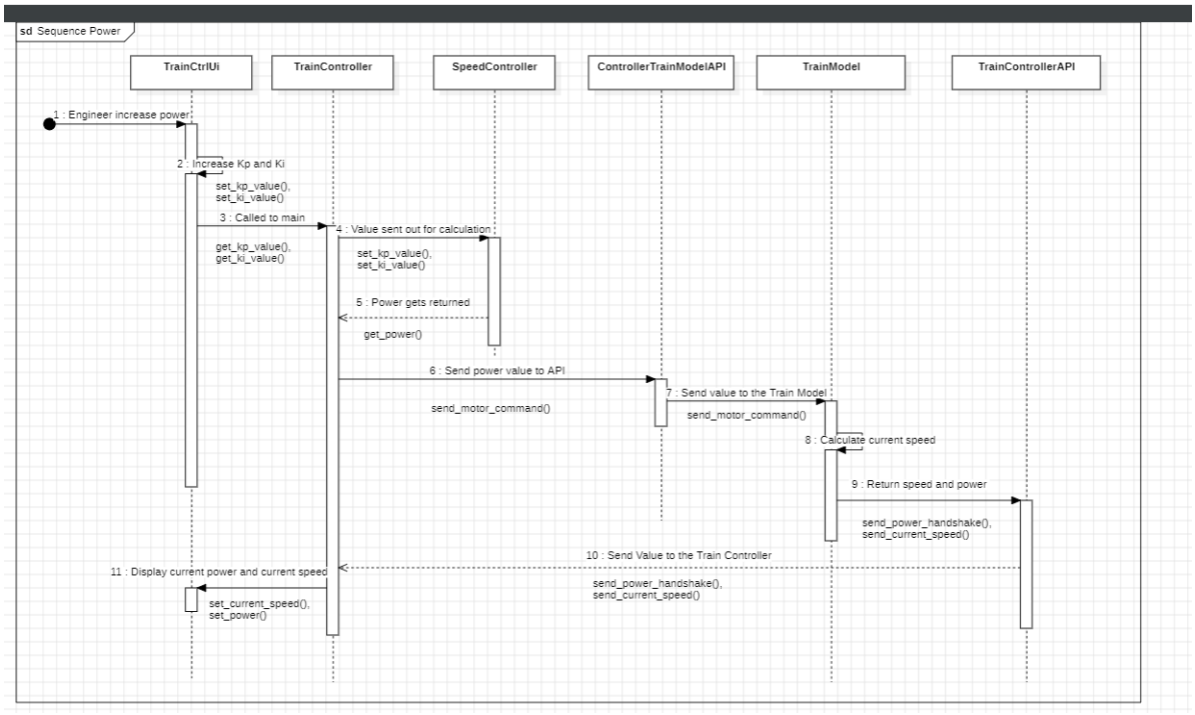

Figure 92: Train Controller Power Sequence Diagram

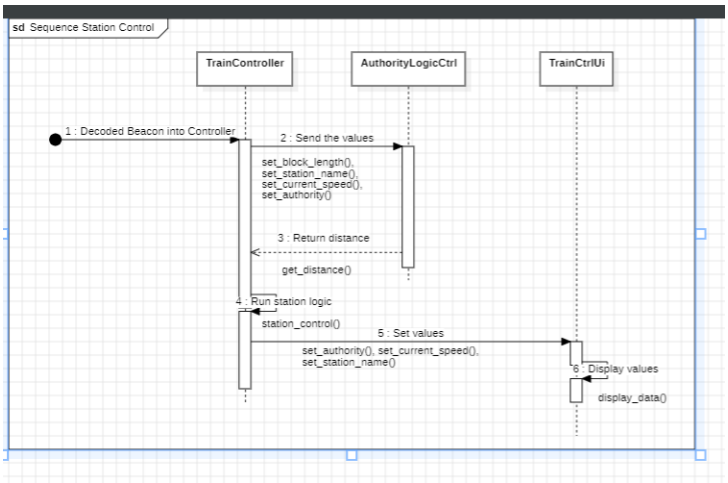

Figure 93: Train Controller Station Control Sequence Diagram

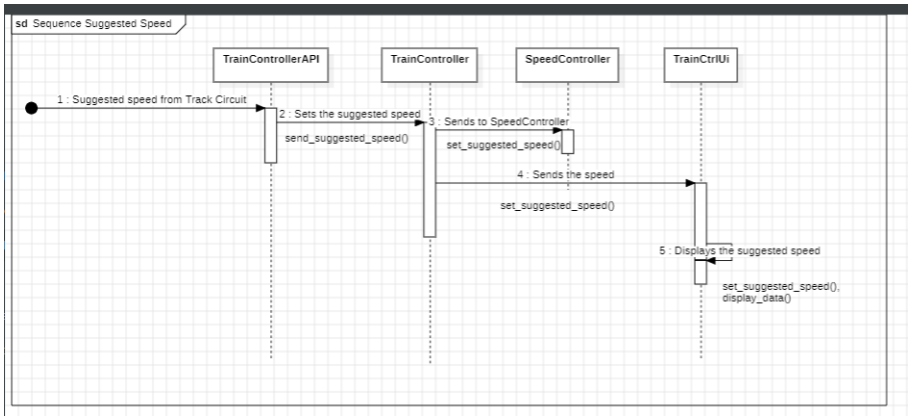

Figure 94: Train Controller Suggested Speed Sequence Diagram

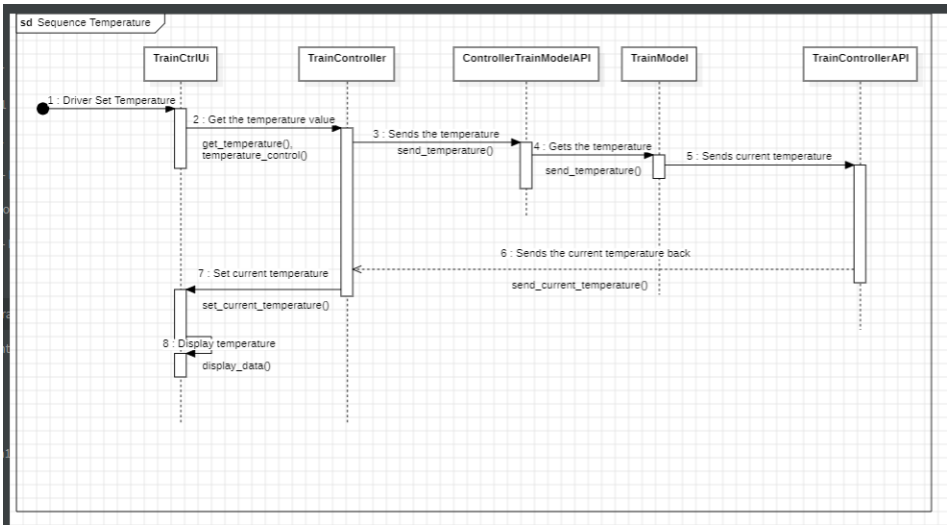

Figure 95: Train Controller Temperature Sequence Diagram