Group 6

Amraiza Naz
Aliya Iqbal
Adrian Gomes
Matthew Cunningham

Hug The Rails

IoT Project Overview

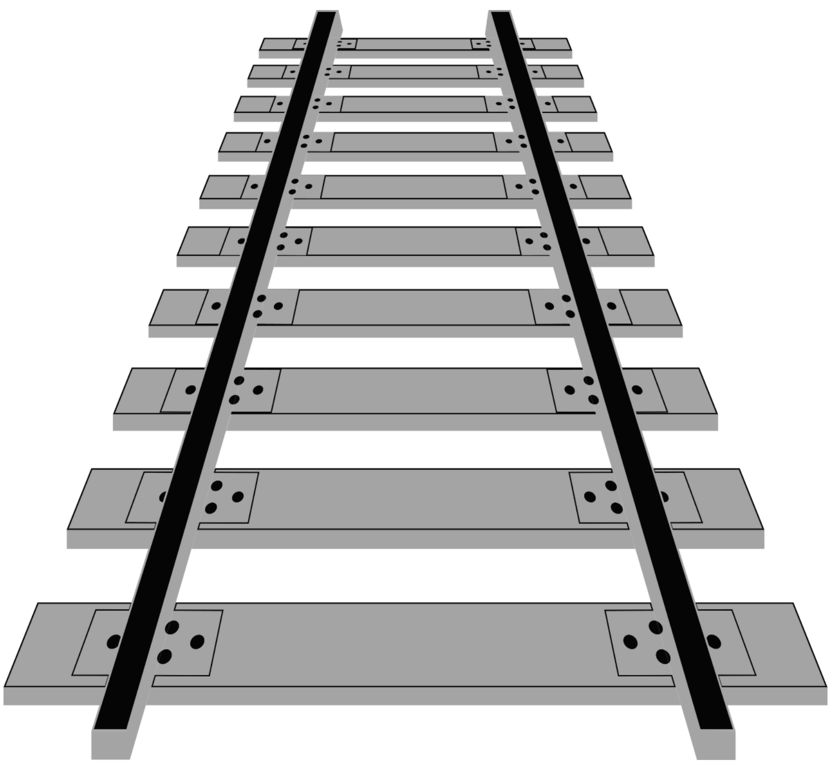

Table of Contents

1: Introduction

1.1 Problem Statement

Hug The Rails (HTR) uses the Internet of Things (IoT) to make HTR safer, less costly, and more efficient. By using IoT to make decisions locally in absence of cellular and wifi connectivity to Back Offices, Hug The Rails can capture data from locomotives and the environment. Unlike other train software, Hug The Rails uses an analytic engine, called an IoT Engine, to process information and make decisions passed on to the Locomotive Control System. We provide capability for the Locomotive operator to enter commands and receive status. Using IoT will allow the operator to maintain operations of the train in the event of a loss of wifi or cellular data, in order to ensure the safety of themselves, the passengers, and the cargo. To maintain code on the trains, operators can always download the latest rules for operation from the Fog/Cloud into the IoT Engine.

1.2 Our Stakeholders

Our stakeholders in this project include the train operators, who will operate the software; the owners, who will manage and oversee the Locomotive Control System; and the surrounding environment impacted by decisions made by the software developers, including riders on the train and customers relying on trains.

1.3 Our Audience (Users)

This software will be sold to train companies. The users are the owners of the train companies and the train operators who will use the software.

1.4 Importance and Value to Users

IoT is a decentralized train system that is safer for users and more efficient for train operators. Current train systems need a central authority to make decisions, such as a master control room, and this costs money, and wastes time and resources. It also means that users need constant access and connectivity with the central authority, and if trains lose this connection, trains are vulnerable to fatal consequences.

By contrast, IoT makes decisions locally in absence of cellular and wifi connectivity. This will allow for the operators to be aware of their surroundings and efficiently handle emergency situations.

1.5 Approach To Solution

The IoT approach is different from other train software. Sensors are at the foundation of all IoT design, allowing devices to collect data and interpret the environment. Therefore, our software will use sensors to measure distance between it and other objects to decide whether to accelerate or decelerate. It also has a touch-screen display that accepts and displays for an operator. The software can also send and receive between a central authority and other locomotives.

IoT can handle emergency situations. It's weather, speed, and infrared sensors are among the few that will be able to operate in the absence of wifi and cellular data to ensure all train operations can still be handled at all times.

We will be implementing the unified process model in order to handle this project. This will allow us to continuously update and review our requirements, and make any changes if necessary. Additionally we will be able to maintain communication and involvement as a team throughout the project.

2: Overview

2.1 Summary Of The Problem

Many existing softwares for train operations don't include a safety mechanism when there is a loss of wifi or cellular data which could endanger both the operator and consumer. Our software will specifically address this issue, and we will ensure our software will continue to run even when there is no wifi or cellular data by installing IoT features.

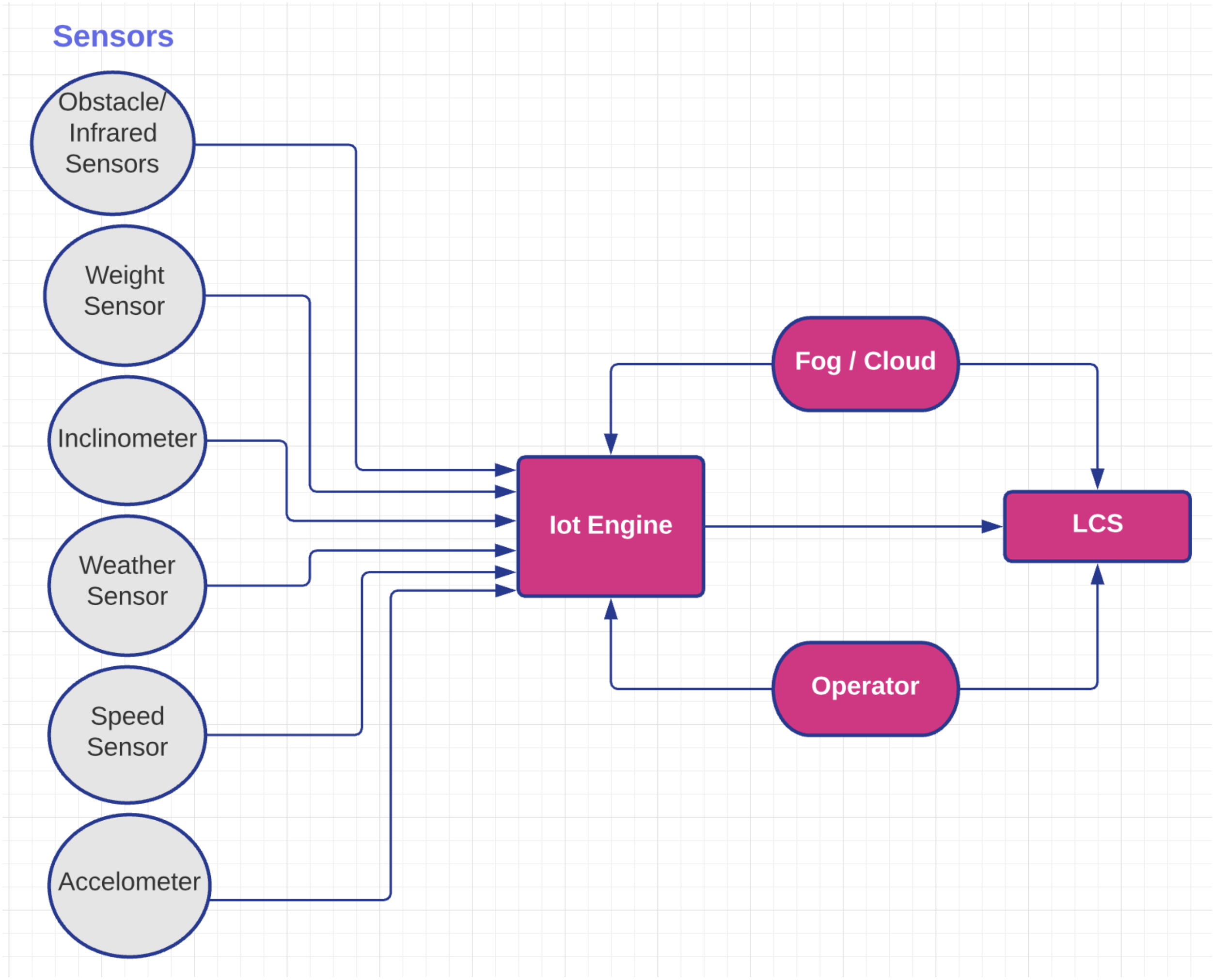

Figure 2.1.1: Conceptual architecture of IoT system.

2.2 IoT Features to be Implemented:

1. **Obstacle/Infrared Sensors:** These sensors will be placed on all sides of the train, and can emit/detect infrared radiation to measure objects up to 1000ft away. They will be used to inform the operator if there is something on the track that they need to slow down to avoid (such as animals, other trains, etc.).

2. **Weight Sensors:** These weight sensors are triggers that will be placed along the track at a fixed distance away from railroad crossings on both sides. When the train crosses each trigger, it will toggle the barriers, which will lower once the train is a certain distance away and raise when the train has passed.

3. **Inclinometer:** This sensor will be placed on the side of the train and be able to detect when the train is tilting due to curves in the track, obstacles, etc. If the sensor detects the train is at an angle beyond a certain threshold (with respect to gravity), it will alert the operator so its speed can be adjusted.

4. **Weather sensor:** This sensor will be placed on the windshield and will be able to detect air pressure, humidity, rainfall, snowfall, and wind speed. If the sensor detects extreme levels of weather, adjustments to the speed can be made as needed.

5. **Speed sensor:** These sensors will be placed on the wheels of the train and will detect how fast the train is going based on the rpm. This will be able to detect if the train is slipping due to snow, oil, etc. and allow adjustments to be made.

6. **Accelerometer:** Can detect if the train is speeding up or slowing down at an extreme rate so adjustments to the speed can be made as necessary

2.3 Phases Of The Process:

I. **Inception: 2/9 - 3/1**

   A. Establish project scope and boundaries

   B. Identify key features of project

   C. Define resources and technology needed for development

II. **Elaboration: 3/2 - 3/13**

   A. Specify programming language and overall design

   B. Define the methods that the application can communicate with other assets

   C. Expand model into a **Life-Cycle Architecture** that captures most of the software's functional requirements

III. **Construction: 3/14 - 3/31**

A. Continuously check for any errors and ensure target goals are being met
B. Careful documentation of any changes made to code/requirements
C. Establish **Initial Operational Capability** that ensures the software is fully operational in a beta environment

IV. **Transition: 4/1**

A. Make software available to consumers
B. Check for deployment issues

2.4 Plans for Future Development:

1. **Automatic Driving Software:** Sometimes people have a fear that an automatic driving software will not work in time or result in an accident. To ensure the customer feels safe while sitting in the train we have implemented a design where if the operator wishes to do so he can override the software by simply just starting to drive themselves. When they drive, the software will not control the driving temporarily.

2. **"Auto Pilot":** A train is always starting from one destination point and travelling to another. For the comfort of our customer we have installed a system where the software will be able to detect where the train is initially starting from, and by simply putting in a destination, the train will "auto pilot" and navigate itself to the location.

3: Requirements

3.1 Non-Functional Requirements

3.1.1 Reliability Requirements

R-1: IoT HTG shall be operable under extreme weather conditions in temperatures ranging from -50 to 150 F.

R-2: IoT HTG shall stand drops up to 5 feet.

R-3 IoT HTG system shall have reliability of 0.999.

3.1.2 Performance

R-4: IoT shall have a response time of 0.5 second, assuming IoT has been on.

R-5: IoT shall be able to support up to 1000 sensors.

3.1.3 Security

R-6: IoT shall be accessed only by User ID/Password.

R-7: IoT shall be temporarily disabled after 3 failed login attempts.

R-8: IoT shall require monthly updates to User ID/Password.

R-9: IoT shall be able to register up to 3 fingerprints in lieu of a User ID/Password.

3.1.4 Operating System

R-10: Reliable operating system shall be chosen for the IoT Engine.

R-11: Operating system executes IoT Engine indefinitely and locally.

R-12: Operating system shall be portable to allow for transfer between trains.

R-13: Operating system shall be efficient to ensure reliable processing of different sensory data.

3.1.5 Hardware

R-14: The train shall be equipped with weather sensors, infrared sensors, weight sensors, accelerometer, inclinometer, speed sensor, Time-Sensitive Networking router (TSNR), and display.

R-15: TSNR shall be connected to all sensors and IoT.

R-16: IoT shall be equipped with harddrive storage of 1TB for sensor data.

3.1.6 Network

R-17: IoT shall use TSNR to communicate with both the sensors and the display.

R-18: IoT shall process received data from the sensors.

R-19: IoT shall send the given data from the sensors to the display.

3.2 Functional Requirements

3.2.1 Display on or off

R-20: Operator shall be able to turn IoT on or off.

R-21: IoT shall initialize required sensors.

R-22: When turning software off the IoT shall deactivate the sensors.

R-23: Also when turning the software off the screen shall display a goodbye message and a short clip of a train driving off before the screen shuts off,

3.2.2 Display start up

R-24: IoT shall display a welcome message and logo of Hug the Rails and then start up the train.

R-25: IoT shall require user ID and password after start up.

3.2.3 Weather Conditions

R-26: Weather sensors shall process external temperatures.

R-27: Weather sensors shall detect rain, snow, etc. on the windshield and send conditions to IoT.

R-28: IoT shall display weather conditions to the operator.

3.2.4 Obstacle Detection

R-29: Infrared sensors shall process infrared light.

R-30: Infrared sensors shall determine distance of objects on track up to 1000ft.

R-31: Infrared sensors shall determine the speed of objects moving ahead/behind the train.

R-32: Infrared sensors shall send IoT data about any obstacles which may come in the way of the train, and alert the operator about said obstacle.

R-33: If obstacle is within 500ft of the train, IoT will signal warning to operator and suggest braking.

3.2.5 Railroad Crossing Trigger

R-34: Weight sensor shall identify triggers on the track before approaching a railroad crossing.

R-35: Weight sensor shall send a signal to railroad crossing barriers which shall send a message to the railroad crossing to raise/lower the barrier.

R-36: Railroad crossing barriers shall send a return signal to IoT and inform operator that barriers have gone down and it is safe to cross.

3.2.6 Speed Control

R-37: Speed sensors shall detect the speed at which the wheels are rotating.

R-38: Speed sensors shall send the train speed to IoT.

R-39: IoT shall display speed of train to operator.

3.2.7 Acceleration Control

R-40: Accelerometer shall detect the rate at which the train is speeding up/slowing down.

R-41: Accelerometer shall send the data to IoT.

R-42: IoT shall display the data to the operator.

R-43: If the train is accelerating a rate that exceeds 12 miles per hour, IoT will send a warning signal to the operator and suggest braking.

3.2.8 Curve Detection

R-44: Inclinometer shall detect when the train is at a sharp curve when its angle (with respect to the direction of gravity) exceeds 8˚.

R-45: Inclinometer shall send an alert to the operator that the train is at a tilt so the speed can be adjusted.

3.2.9 Wheel Slippage

R-46: If there is potential for wheel slippage due to unfavorable weather conditions, IoT will signal warning to the operator and suggest braking.

R-47: If IoT detects difference between train speed and wheel speed, the operator will be warned of potential wheel slippage and suggested to release the throttle.

R-48: If there is potential for wheel slippage due to sharp curves on the track, IoT will send a warning signal to the operator and suggest braking.

4: Requirements Modeling

4.1 Use Cases

Use Case: IoT startup

No. : 4.1.1

Primary Actor: Operator

Secondary Actor(s): IoT, Sensors

Goal: Activate IoT and begin processing data gathered from sensors. Display request for User ID/Password

Preconditions: Train has power

Trigger: Train ignition is on

Scenario:

1) IoT system is powered by train
2) IoT starts up and turns on sensor and Time-Sensitive Networking router (TSNR)

Use Case: Validate User ID/Password

No. : 4.1.2

Primary Actor: Operator

Secondary Actor(s): IoT

Goal: To authenticate user credentials

Preconditions: IoT has power and is operating, operator has valid credentials

Trigger: IoT startup

Scenario:

1) Display requests User ID/Password
2) Operator enters credentials
3) IoT processes credentials and determine validity

Use Case: Display data

No. : 4.1.3

Primary Actor: IoT

Secondary Actor(s): Sensors, TSNR

Goal: To provide regular information about the train for the operator

Preconditions: Both train and IoT have power and are operating

Trigger: Train departs from station

Scenario:

1) Sensors process data from surrounding area
2) Sensors send data to TSNR
3) TSNR sends data to IoT
4) IoT processes and displays data to the operator

Use Case: Display warning

No. : 4.1.4

Primary Actor: IoT

Secondary Actor(s): Sensors, TSNR, Log File

Goal: To alert the train operator about a possible emergency situation

Preconditions: Both train and IoT have power and are operating; sensors are operating and connected to IoT.

Trigger: Data from sensors meet conditions that would require the operator to slow down the train

Scenario:

1) Sensors detect extreme conditions
    i) Weather sensors detects snow or ice on track
    ii) Infrared sensors detect moving object on track
    iii) Infrared sensors detect stationary object on track
    iv) Inclinometer detects train inclination that exceeds 8˚
    v) Speed sensors detect discrepancy between train and wheel speed
    vi) Accelerometer detects acceleration that exceeds 12 mph
2) Sensors send data to TSNR
3) TSNR sends data to IoT
4) IoT processes data and displays a warning message to the operator
5) Log file records incident with timestamp

Use Case: Access log data

No. : 4.1.5

Primary Actor: Technician

Secondary Actor(s): IoT, Log file

Goal: To view all recorded data from sensors

Preconditions: Train is on, Logged in as technician

Trigger: Technician requests log file

Scenario:

1) Display log file data

4.2 Use Case Diagram

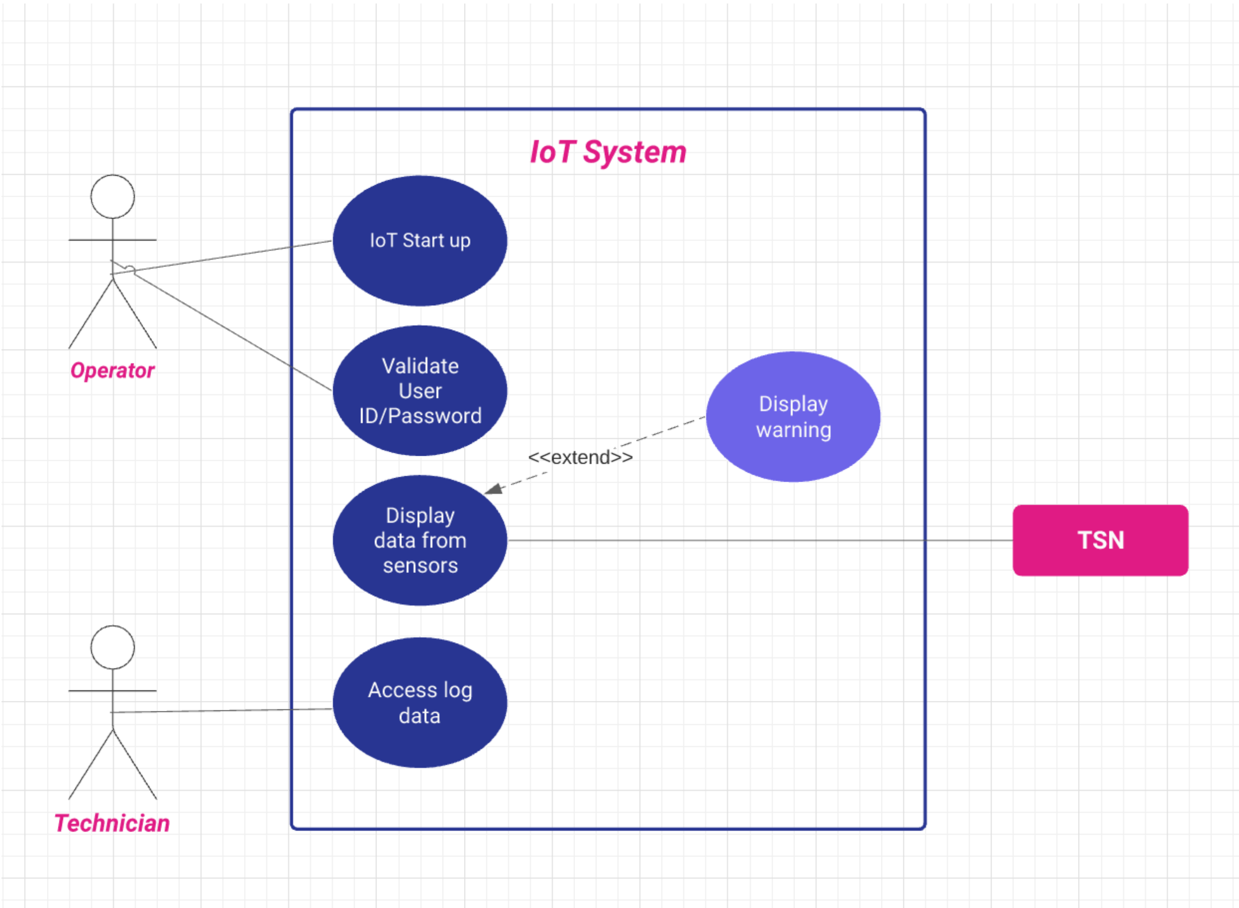

Figure 4.2.1: UML diagram for IoT system use cases.

4.3 Class-Based Modeling

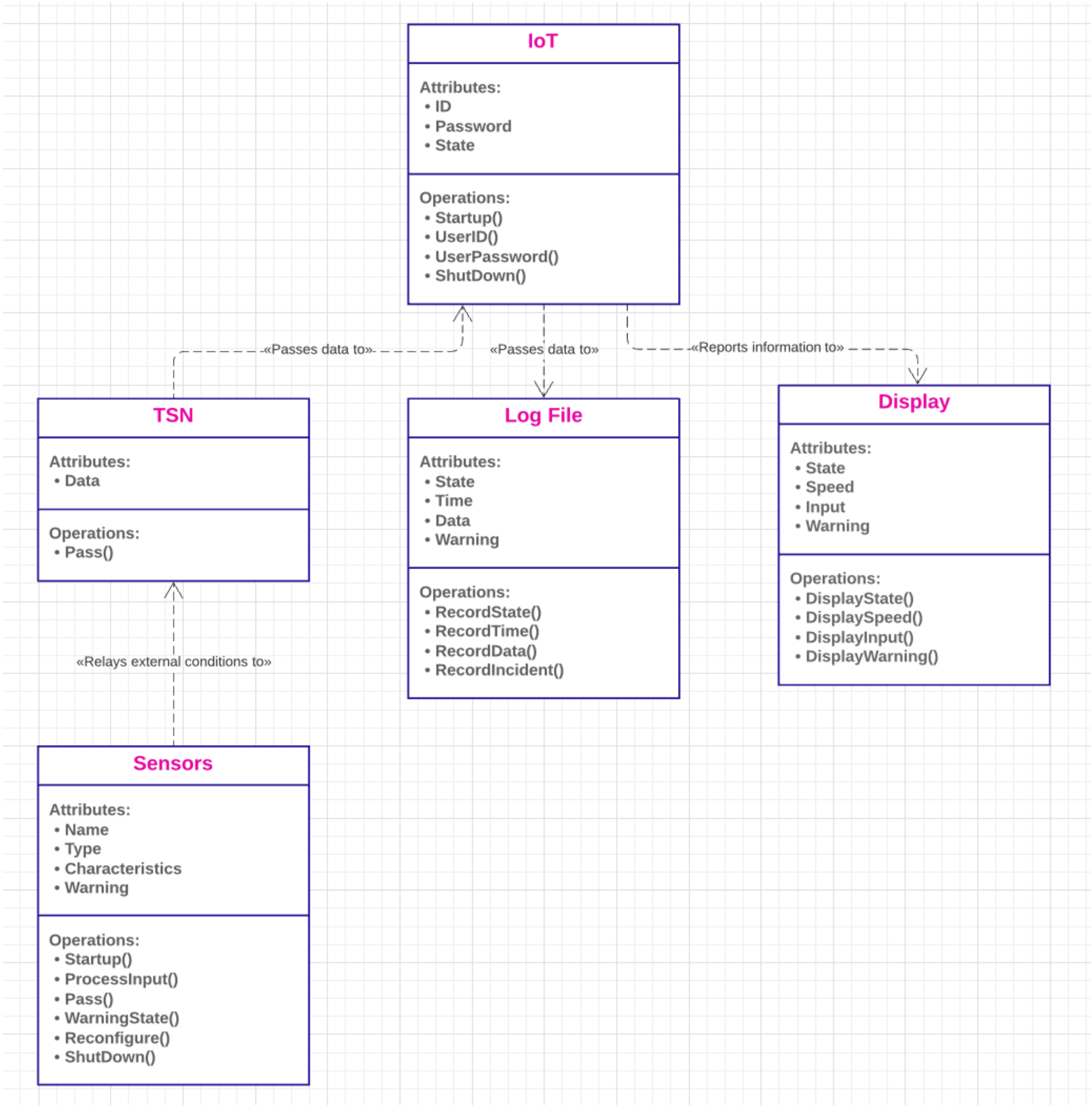

Figure 4.3.1: UML Class-Based Model of IoT system.

4.4 CRC Modeling/Cards

Class: IoT
Validate user and process sensor data retrieved from TSNR
Responsibility: Request User ID/Password from operator **Collaborators:** display
Responsibility: Gather sensor data from TSNR **Collaborators:** TSNR, weather sensors, infrared sensors, weight sensors, accelerometer, inclinometer, speed sensor
Responsibility: Display sensor data to operator **Collaborators:** display
Responsibility: Display warnings/suggest speed changes based on sensor data **Collaborators:** display, weather sensors, infrared sensors, weight sensors, accelerometer, inclinometer, speed sensor

Class: Sensors
Handles functions and attributes for every sensor
Responsibility: Detect conditions from surrounding area **Collaborators:**
Responsibility: Add sensor data to the technician log **Collaborators:** Log File
Responsibility: Send data to TSNR **Collaborators:** TSNR

Class: Display
Display sensor data to the operator
Responsibility: Report speed of train **Collaborators:** IoT, TSNR
Responsibility: Report weather conditions (rain, snow, etc.) **Collaborators:** IoT, TSNR

Responsibility: Report any moving obstacles on track **Collaborators:** IoT, TSNR

Class: TSNR
Processes and sends data from sensors
Responsibility: Receives data from all the sensors and sends it to IoT **Collaborators:** IoT, weather sensors, infrared sensors, weight sensors, accelerometer, inclinometer, speed sensor

Class: Log File
Keeps track of speed changes, general sensor data, and warnings
Responsibility: Record speed change **Collaborators:** weather sensors, infrared sensors, weight sensors, accelerometer, inclinometer, speed sensor
Responsibility: Record sensor data every 10 seconds **Collaborators:** weather sensors, infrared sensors, weight sensors, accelerometer, inclinometer, speed sensor
Responsibility: Record warnings sent by sensors to IoT **Collaborators:** weather sensors, infrared sensors, weight sensors, accelerometer, inclinometer, speed sensor

4.5 Activity Diagram

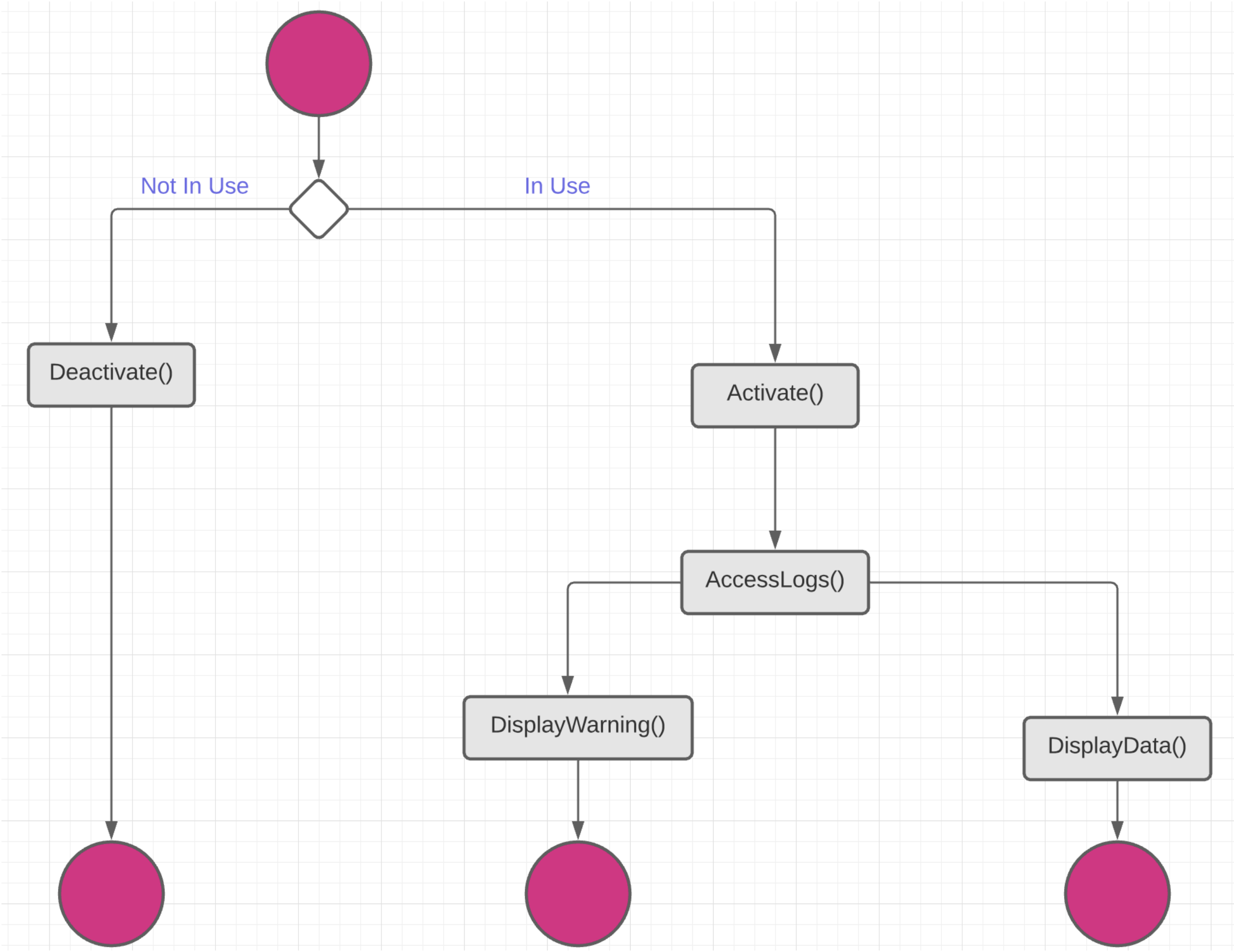

Figure 4.5.1: UML activity diagram for IoT system.

4.6 Sequence Diagram

Use Case: IoT startup (4.1.1)

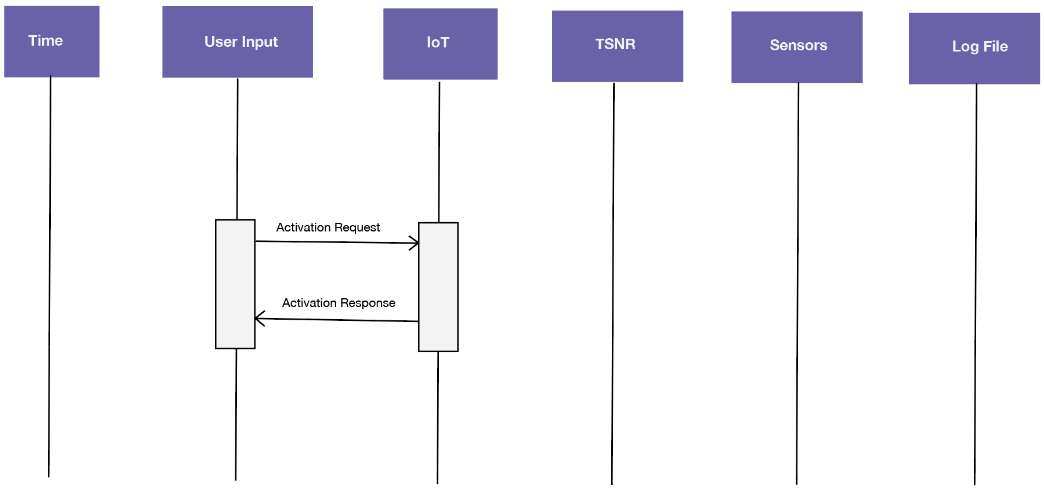

Figure 4.6.1: UML sequence diagram for IoT system use case IoT startup (4.1.1).

Use Case: Validate User ID/Password (4.1.2)

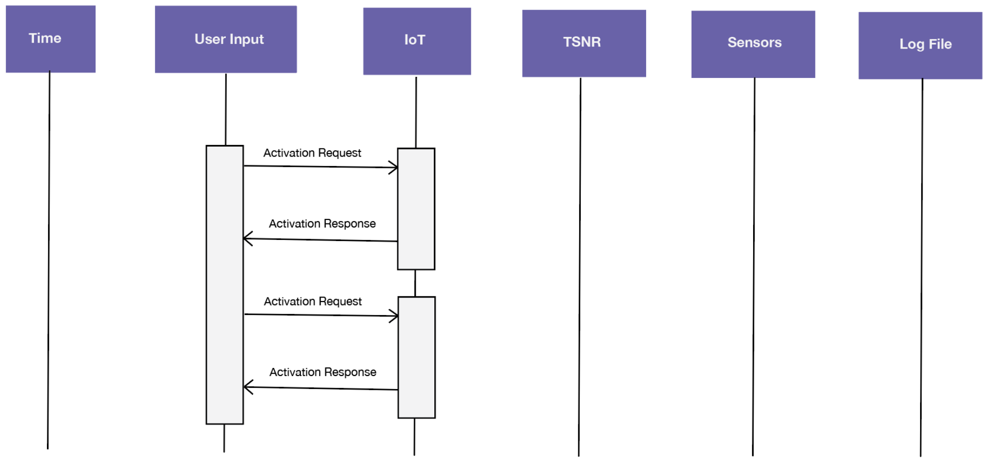

Figure 4.6.2: UML sequence diagram for IoT system use case validate user id/password (4.1.2).

Use Case: Display data (4.1.3)

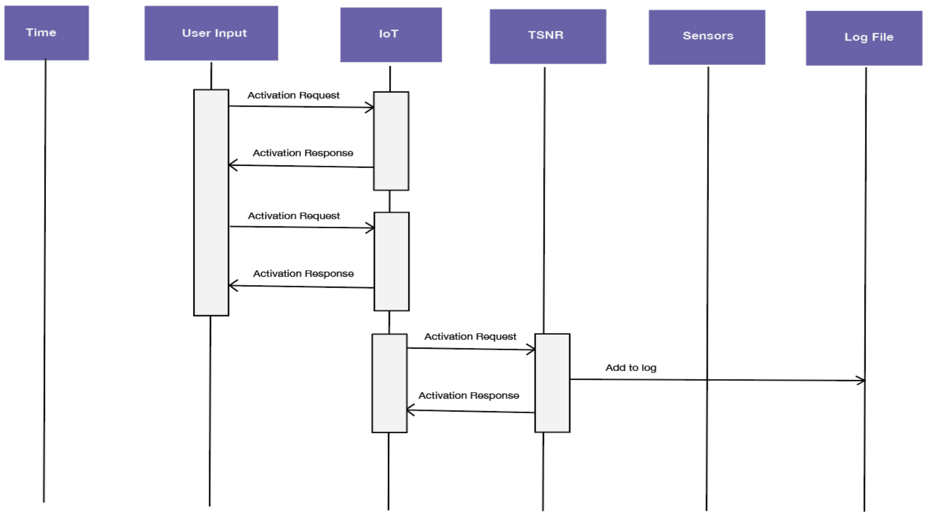

Figure 4.6.3: UML sequence diagram for IoT system use case display data (4.1.3).

Use Case: Display warning (4.1.4)

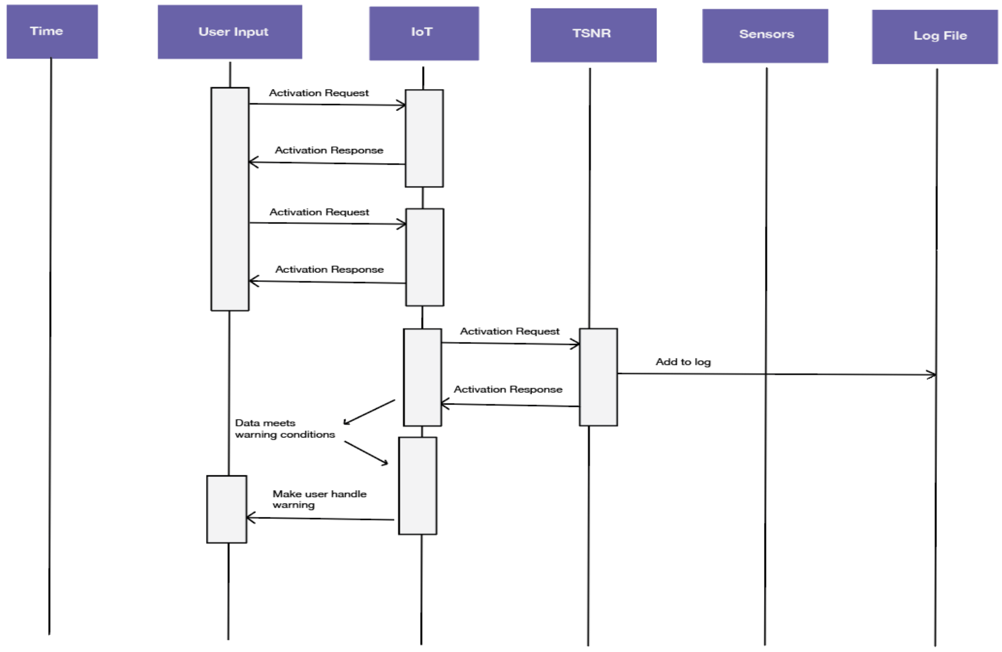

Figure 4.6.4: UML sequence diagram for IoT system use case display warning (4.1.4).

Use Case: Access log data (4.1.5)

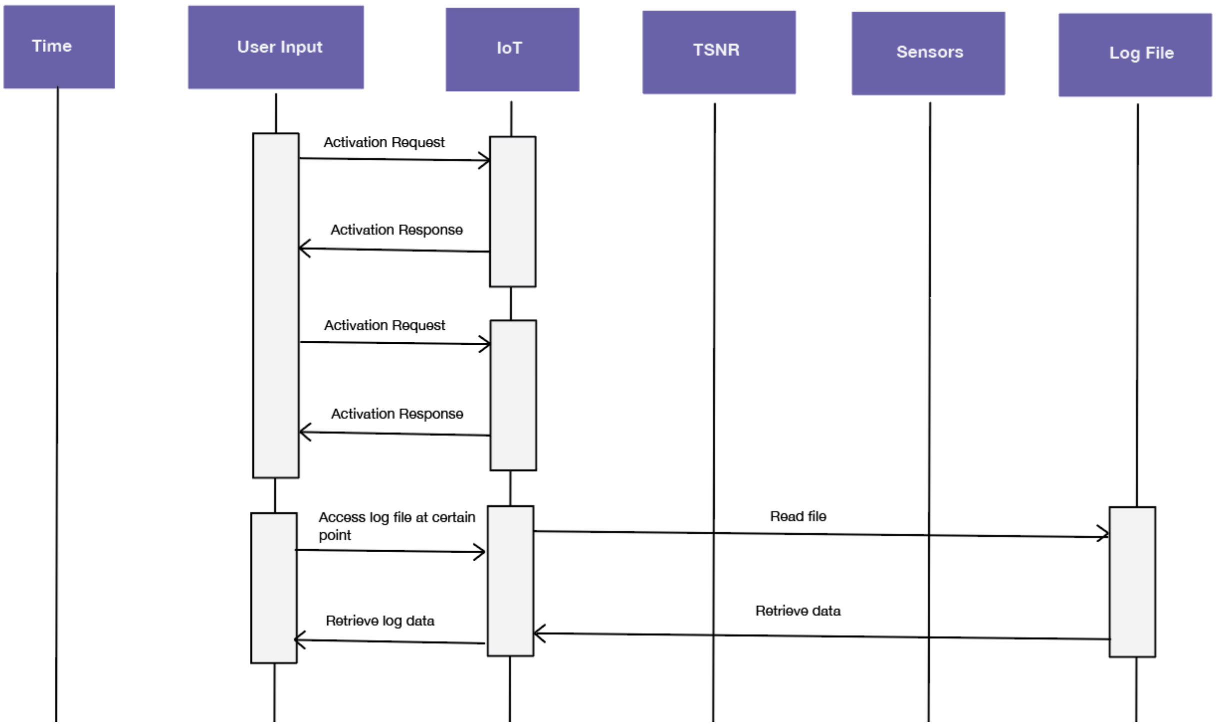

Figure 4.6.5: UML sequence diagram for IoT system use case access log data (4.1.5).

4.7 State Diagram

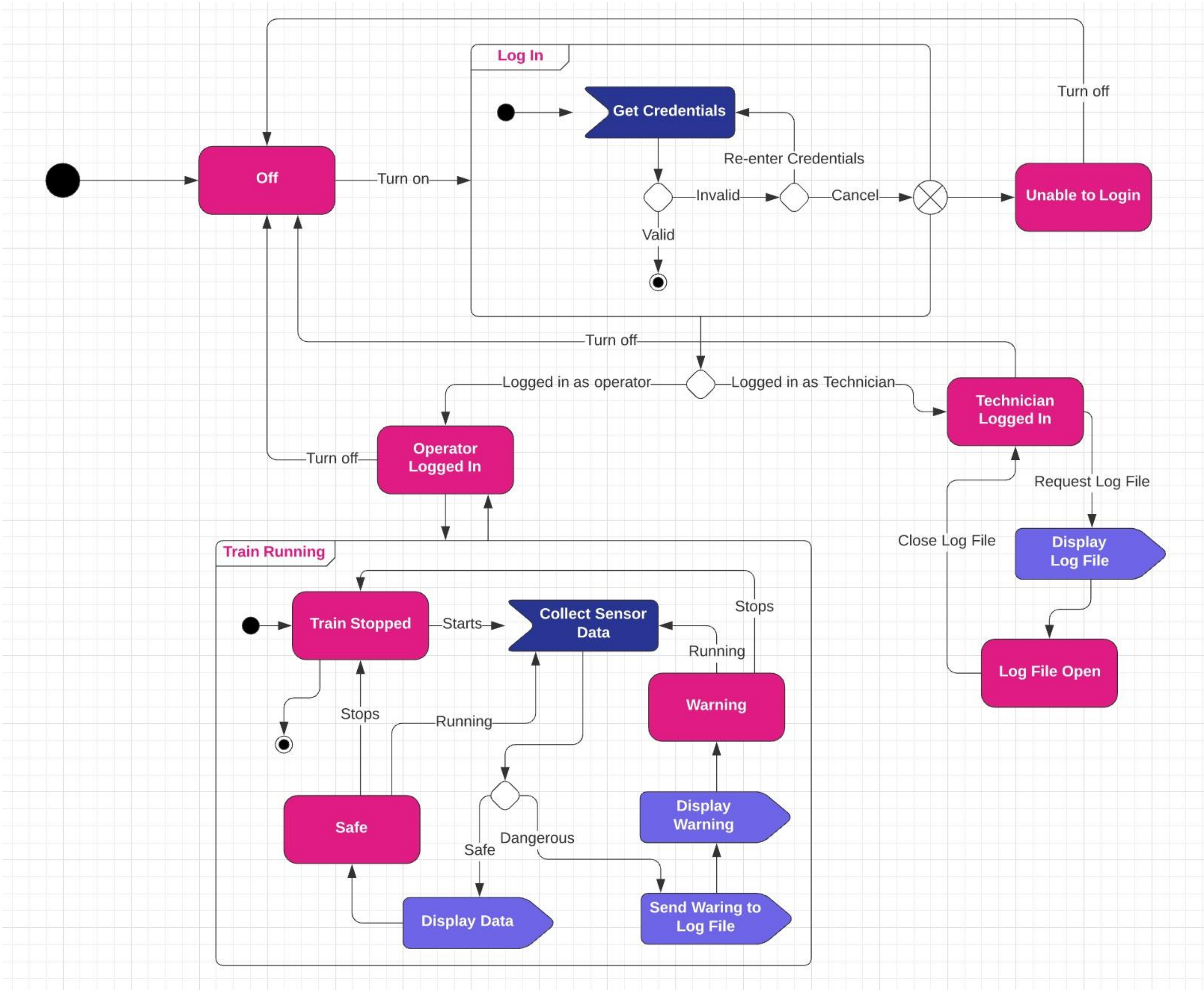

Figure 4.7.1: UML state diagram for IoT system.

5: Software Architecture

5.1 Architecture Style

Considering the pros and cons of each architectural style, we decided that our IoT system would best utilize an Object Oriented Architecture (OOA). One of the most significant benefits of an OOA is code usability and modularity, which would allow us to maintain an agile development process while reducing risk management. Additionally, an OOA most closely models real world architecture, facilitating visualization of how to carry out tasks while maintaining a relatively user-friendly model. Not only would the code be easier to read and understand, it is also easier to test, manage, and debug, while maintaining the integrity of the code.

There are still a few cons to consider when using OOA. For example, it is difficult to determine all the necessary classes and objects required for a system. Additionally, the strong coupling between superclasses and subclasses could cause difficulties in terms of modifications, as swapping out superclasses can often “break” subclasses. Software that is built using this style may also require extensive middleware to provide access to remote objects.

In terms of our software, the IoT system is manipulated by operations of the objects encapsulated within it. Data from the sensors is accessed through the TSNR, which it then passes to the Display class. We would represent this data as objects, with attributes to closely resemble the real world. The warning object will contain a boolean that will be set to true if any of the sensors meet the “warning conditions”. For example, the weather and infrared sensor objects will contain a boolean that determines extreme weather conditions (such as excessive snow, rain, etc.) or moving/stationary objects on the track, and they will pass their value to the Display class. If either of these objects are true, the warning object will be set to true. Similarly, acceleration, speed, and inclinometer sensor objects will contain an integer value that represents the acceleration, speed, and tilt of the train and pass their value to the Display class. If any of these values exceed the maximum values defined in Section 3, the warning object will be set to true. If the warning object is true, IoT will also send that data to the Display class, which will then display a braking suggestion to the operator. In addition, the weight sensor object will also contain a boolean that determines if the train has crossed it, and send its value to the Display class to alert the operator that the railroad crossing barrier has been triggered. The log file object will contain a tuple that includes a timestamp and string to log the conditions of the train. Because coordination and communication between these components is done primarily through message passing, an OOA would be the most optimal architecture for the IoT system.

While we settled on an OOA for our software, there were still other styles we considered:

a) Call-and-Return Architecture: With this style, we would be able to split the main program into subprograms, such as modules. We would then be able to set the hierarchy for the different programs, which can increase performance and improve efficiency by splitting computations between them. Additionally, it would be easier to modify and build on because of the ability to add more subprograms. However, this could cause the main program to become more difficult to manage, and it usually fails to scale. Another major downside to this style is the inadequate attention to data structures, which is crucial to our software as efficient data organization and modification is integral to ensuring safe train operations, as well as the difficulty created by attempting to have programs run in parallel

b) Layered Architecture: This architectural style supports rapid and parallel development (which allows for a faster development process), as well as asynchronous technique (which helps minimize the loading time of the application). Additionally, it is easy to modify software built using this style because any changes made will not affect the whole model. However, this style is not suitable for small applications, and could have a negative effect on performance. Additionally, failure to separate variables, functions, etc. between layers can lead to leaky abstraction. It would also be difficult to design our software using this architecture because we would need to define a very specific, unchanging hierarchy, and the “lower layers” cannot make calls to any “upper layers”.

c) Data-Centered Architecture: This style emphasizes integrability of information, and comprises various components that communicate through shared information “vaults”. It provides adaptability and reusability of operators since they don’t have coordinate correspondence with each other, as well as reduces overhead of transient information between the various software parts. However, it is difficult and costly to modify the stored information, as the focal point of this style is the data store. Because external conditions are always changing, this would be inconvenient for our system. Additionally, since the data store is simply a “stockpile” of information, it could introduce performance problems every time there is an attempt to access it

d) Data-Flow Architecture: With this architectural style, the whole software system is seen as a series of transformations on consecutive pieces of input data, where data and operations are independent of each other. Similar to Data-Centered Architectures, the independent nature of the subsystems allow for reusability, as well as flexibility in terms of both sequential and parallel execution. It is also easy to modify the connection between filters by offering a simple pipe connection between them, However, considering the different subsystems have no interaction at all, this would introduce problems in our software as the objects do interact with each other, such as the sensor and warning objects. Maintenance of this architecture is also complex

5.2 Components, Connectors, and Constraints

Components:

A software component is a part of the software that contains a subset of the system's functionality and/or data. A set of these components will be able to perform a function required by the system. Examples of some components of a software system may include network services, graphics engine, user interface, etc. The components of our IoT system would include the various classes that both collect input data and store user data. They would also include the computational modules that are able to perform any methods required by the IoT system, for example determining whether a warning message or recommended speed adjustments need to be displayed.

Connectors:

Connectors work in conjunction with the software components in order to accomplish the system's goal; they allow the components to communicate and coordinate with each other. This process of sending data through message passing can be seen as passing parameters to various methods in the system, and invoking methods from each of the classes. In terms of the IoT system, the connectors would include the various sensors that pass data concerning the external conditions of the train to the TSNR. It would also include the TSNR itself, which then passes the given data to the IoT system.

Constraints:

A software constraint is defined as the way in which components can be integrated to form the system, and is a restriction on the degree of freedom you have in providing a solution. Some constraints are applicable to any software such as limitations in resources, or decisions made by senior management, team members, or the consumer that restricts the way in which you are able develop your system.

Constraints specific to our IoT system would also include a strict time restriction, as data that is shared between the system, sensors, TSNR, and display must be delivered extremely quickly to give the conductor the most accurate information and recommendations possible. There is also a time constraint in terms of the way in which the log files keep track of the condition of the train and external conditions. The data passed needs to be accurate and securely stored, and the transfer process needs to be reliable. Additionally, the user and technician login information needs to be secure.

5.3 Control Management

Control will be managed in the architecture by testing throughout the system hierarchy against the use cases defined in the previous section. The results should match up with the expected outcomes of the use cases of the IoT system. Data will flow from the sensors to the TSNR to the IoT system, where it will be processed, and then delivered to the operator; if the train is put in the warning state, the data will be pushed to the log, aswell, which can then be accessed by the technician. Understanding this hierarchy and how external entities with the software is an important part of the architectural design.

> *Refer to Figure 5.5.1 for the architectural context diagram to understand the interactions between external entities and software.*

5.4 Data Architecture

Data will be structured in our IoT system in ways that align with the practices and procedures of Java-based object-oriented programming. This means that information related to the IoT system will exist within variables, methods, and other structural features of a program written in Java. This, however, does not mean that the IoT system will use methods and functions to place data in a separate file or exterior database. In other words, the data will almost always be in the objects created by the IoT system.

For this data architecture to be effective, it will need to account for several cases. Passing a piece of data from one object to another will require a unique function with parameters to carry the information. Some functions will be queried frequently because of high-value information needed to be presented regularly. This data will need to be stored locally in the object, usually in private variables that can be accessed, manipulated, and re-set with public functions and methods.

5.5 Architectural Designs

Our IoT system will have a few external entities which will help our system run such as our archetype, sensors, which is incredibly useful when the train needs to run locally when there is no wifi. As we stated in the previous sections, the sensors will be placed all around the trains as well as on the tracks when there is a railroad crossing to help the train recognize moving objects, curves, weather, and when it is nearing a railroad crossing. This is very essential for the IoT system to function properly.

Not only that but another archetype which the IoT to run is the operator. Once the operator receives messages from the sensors and other databases, they will have to control the train and drive the train by using the brakes and throttle. The operator will also be in charge of entering the password once they turn on the IoT system due to security issues and they will also be responsible for turning off the system once they are done.

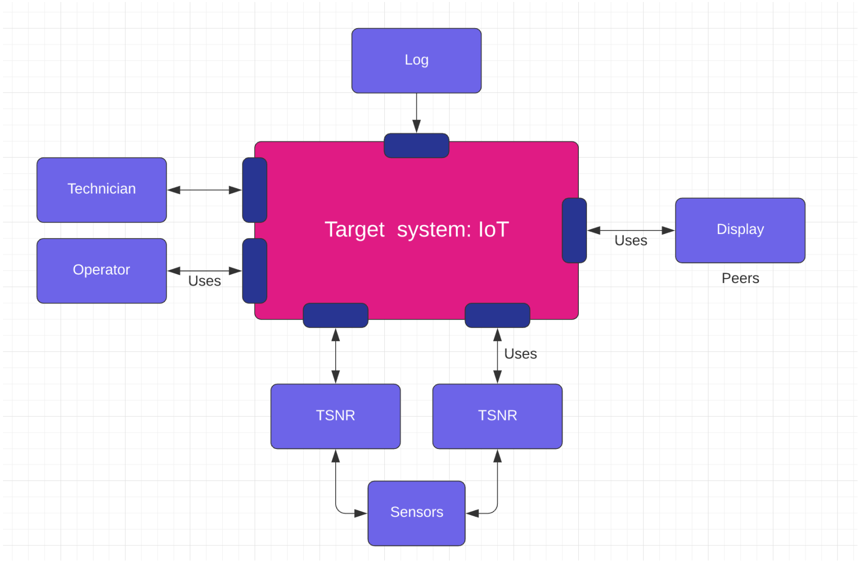

Figure 5.5.1: Architectural context diagram of the IoT system.

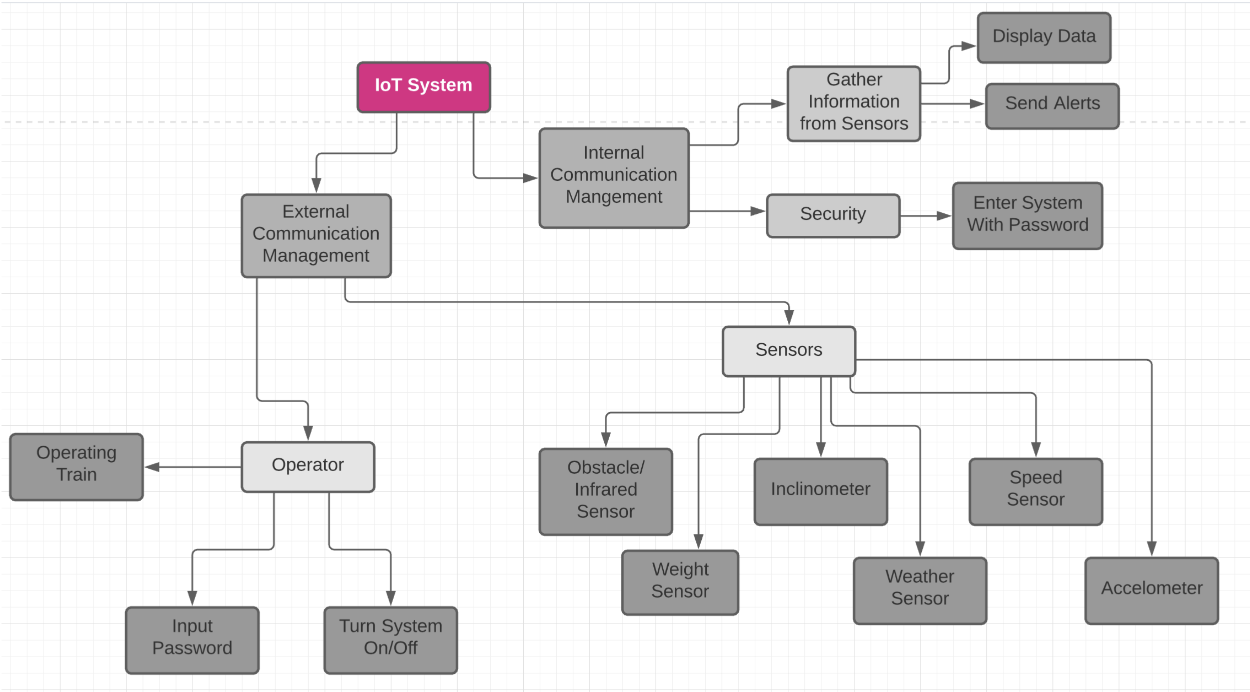

Figure 5.5.2: The instantiation of the IoT System which explains how each archetype plays a significant role in the system and how it plays its job and portrays the overall structure of the system and the major components which will be used in the IoT system.