DESIGN

Group 3

22 JUNE, 2023

PROJECT

T3

Short description of the proposed solution for the problem

The problem we want to solve with this project, is the congestions at certain points on a train station during peak hours. Due to this, walking through, getting on and of the trains will take longer. Reducing this could lead to increased efficiency of train station and fewer trains getting missed by passengers.

We want to solve this by directing the passengers to go in the right direction of less crowded wagons which should evenly spread passengers across all wagons and across the whole platform instead of having them condensed at a specific point. This will be achieved by using coloured LEDs on the platforms, where the trains and each wagon is expected to stop. The number of LEDs that light up, and their colour will depend on the amount of people inside the wagons, the length of the wagons and their position in the train.

GREEN LEDs will indicate that the wagon has few people inside, BLUE means that the wagon is roughly half full and RED indicates that the wagon is already full. The actual percentage values are shown below.

Colour	**Percentage occupation inside wagon**
GREEN	0-50%
BLUE	51-85%
RED	86-100%

Technical description of solution

3 wagons will be implemented and the wagon with the lowest hardcoded ID will become the head wagon. This ensures that when the head wagon disconnects, the remaining wagons can decide which one of them become the head wagon based on their hardcoded ID. The head wagon communicates with the other wagons through a wired connection and will give the other wagons their dynamic ID which will be used for communication. The head wagon will ask and receive the occupation data from the other wagons and itself. Once the information is gathered, it will be sent to the station module wirelessly. The station then depending on the information will light up the LEDs according to the previously explained method.

Class diagram

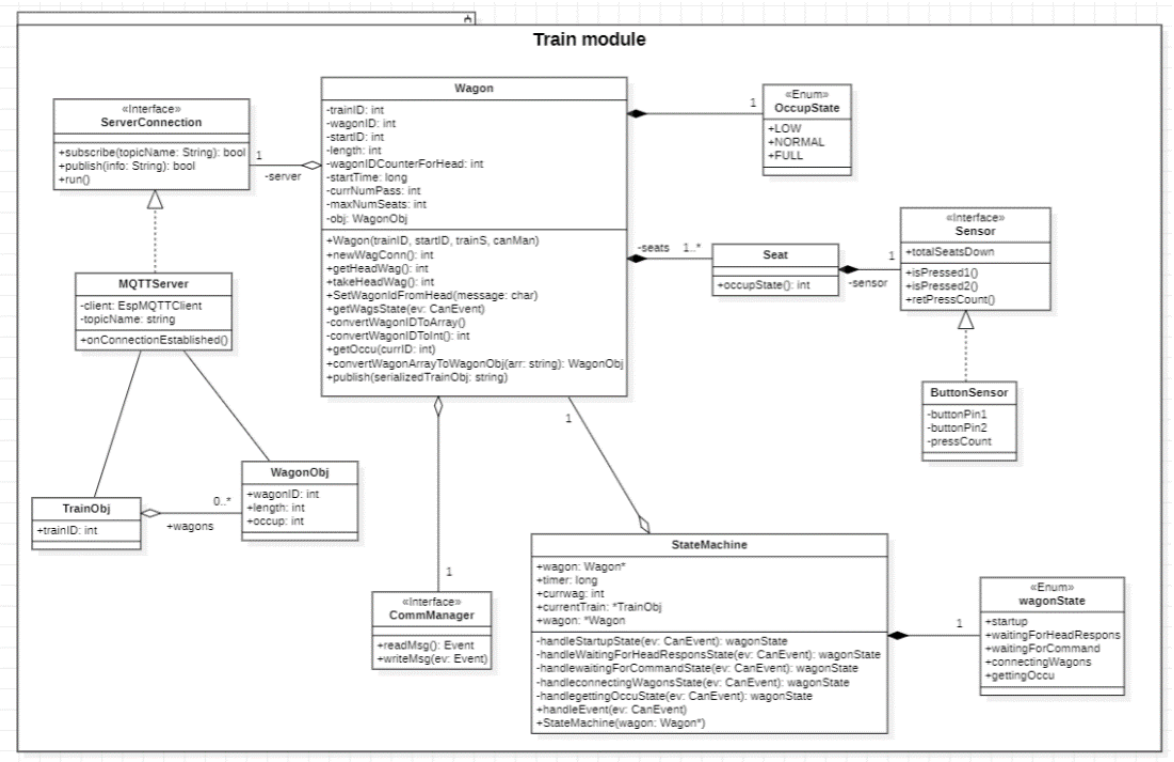
The Smart trains system consists of 3 submodules, which are the Train and the Train Station and the Can communication submodules.

- **Train module:**
    1. Wagons, their states and behaviors
    2. Seats, where passengers are detected
    3. Wireless connection, to communicate with the train station
    4. State machine, for taking care of incoming events for a wagon

- **Train station module:**
    1. Train station, their states and behaviors
    2. Wireless connection, to communicate with the head wagon
    3. Platforms, where the arriving wagons occupation is indicated
    4. Leds, signal the occupancy of a wagon

- **Can communication:**
    1. Communication manager, to handle change of information between wagons
    2. Event, to describe the events happening during communication

Train module:



Wagon	Class, that represents a train wagon. The trains are constructed of multiple wagons. They communicate information between themselves in a distributed fashion but elect a Head Wagon which is responsible for communication with the train station. They store the information of the number of passengers inside and the fullness of the given wagon. Has a CommManager, through with it communicates via CAN.

<<Enum>> OccupState	State that represents the occuppancy state of the wagon.
States	
LOW	LEDs turn green when wagon is below 50% occupancy
NORMAL	LEDs turn orange when wagon is above 50% and below 85%
FULL	When a wagon reaches 86% occupancy, led lights on the platform in front of this wagon will turn red.

<<Enum>> WagonState	State that represnets wether the wagon is head or just a normal wagon.
States	
NORMAL	Wagon is normal wagon.
HEAD	Wagon is the head wagon.

Seat	Represents a seat. Wagons are built up of available seats. It can determine that if the given seat is taken.

<<Interface>> Sensor	Interface for abstraction of the sensor

ButtonSensor	Specific sensor implementation. Can decide if wither the sensor activated or not.

<<Interface>> ServerConnection	Interface that abstracts the actual implementation of how the communication is done with the server. Because of this if a server implementation realizes this interface, it can be used in our system, without having to change any code at all.

MQTTServer	Implementaion of the specific server which handles communication.

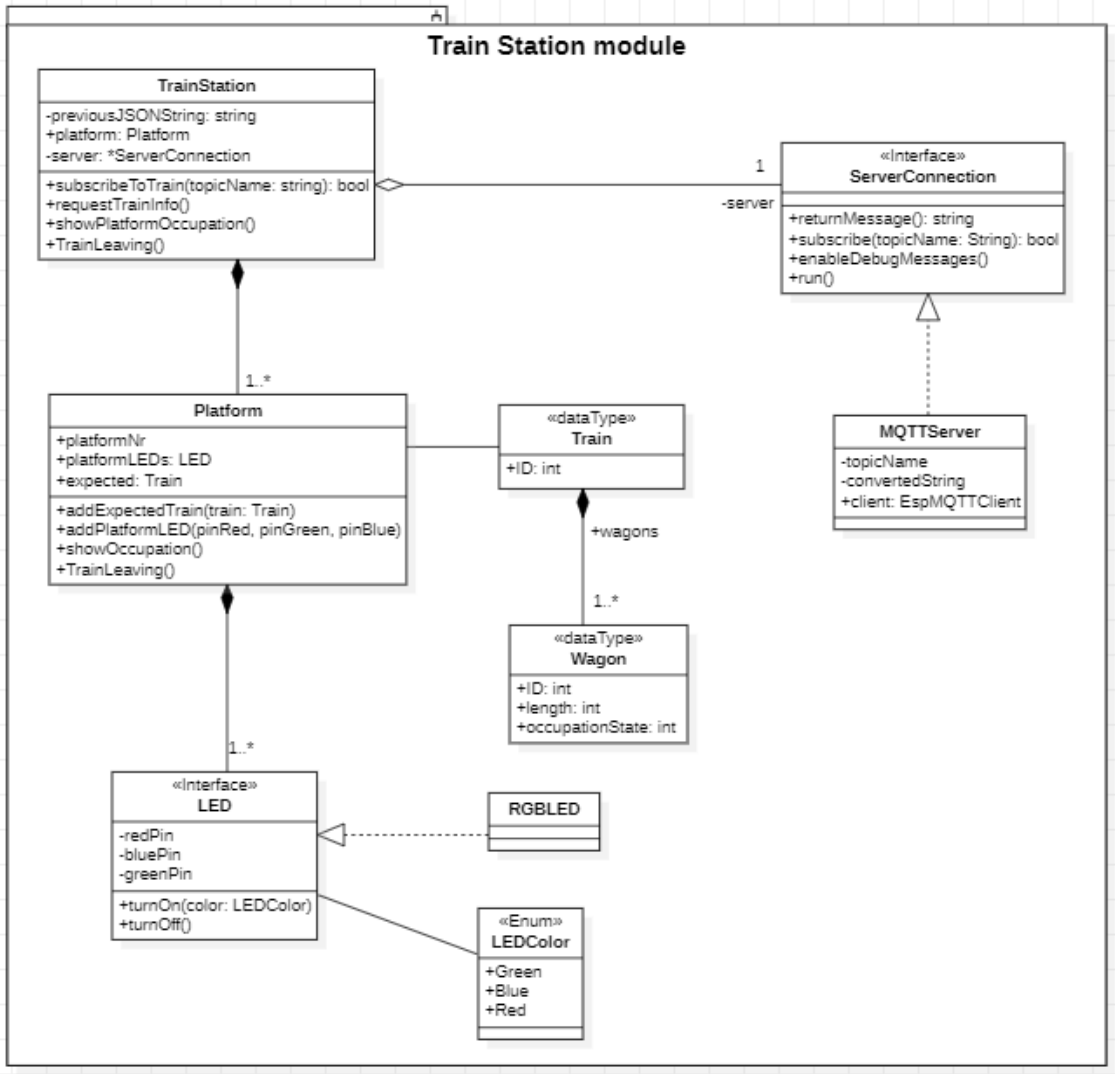
StateMachine	State machine that receives a CanEvent and based on that and the state machines current state it decides its behaviour. It controls a wagons behaviour as well.

wagonState	Statemachine enums
States	
startup	
waitingForHeadRespons	
waitingForCommand	
connectingWagons	
gettingOccu	

TrainObj	Contains the information of the train that gets sent through the server to the Train station. Consists of wagons. Will be serialized and deserialized.

WagonObj	Contains the information of a wagon for server communication.

Train Station module:



TrainStation	Class that represents the train station. It's duty is to accept connection from a train, communicate with it, and correctly light up the LEDs on the platform where the train arrives.

Platform	Class that represents the platform. It will light up the LEDs inside of it.

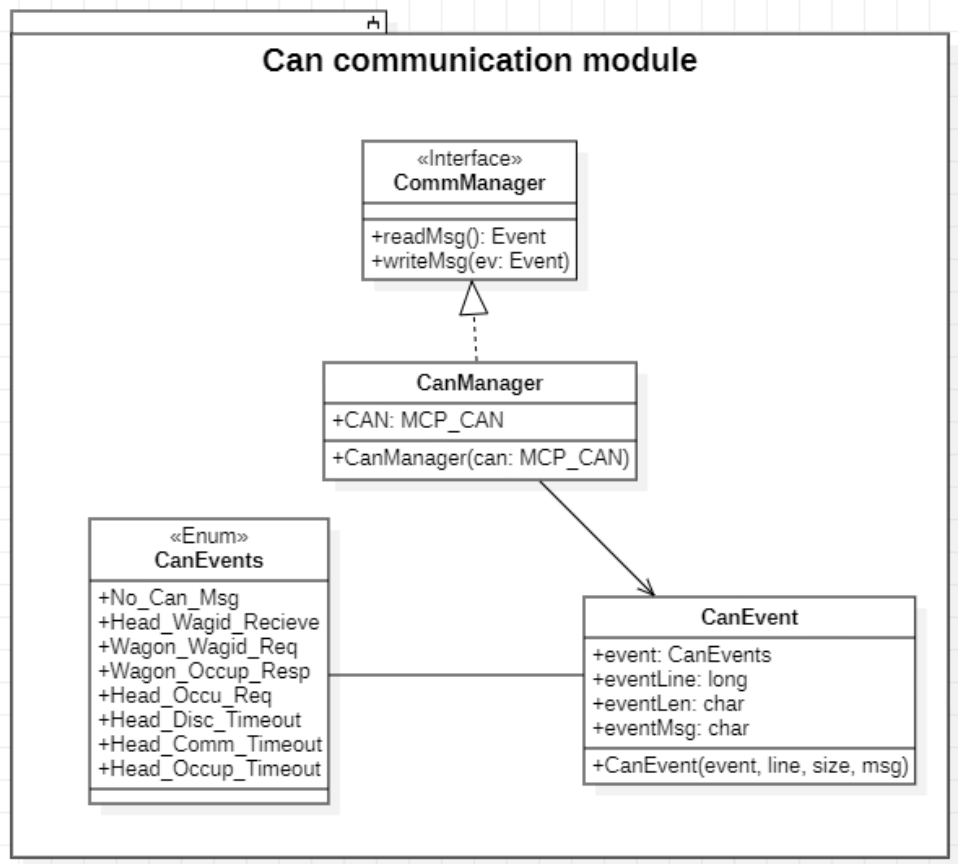
<<Interface>> LED	Interface that abstracts the LEDs different kind of implementation.

RGBLED	Implementaion of the specific LED.

<<Interface>> ServerConnection	Interface that abstracts the actual implementation of how the communication is done with the server. Because of this if a server implementation realizes this interface, it can be used in our system, without having to change any code at all.

MQTTServer	Implementaion of the specific server which handles communication.

Can communication module:



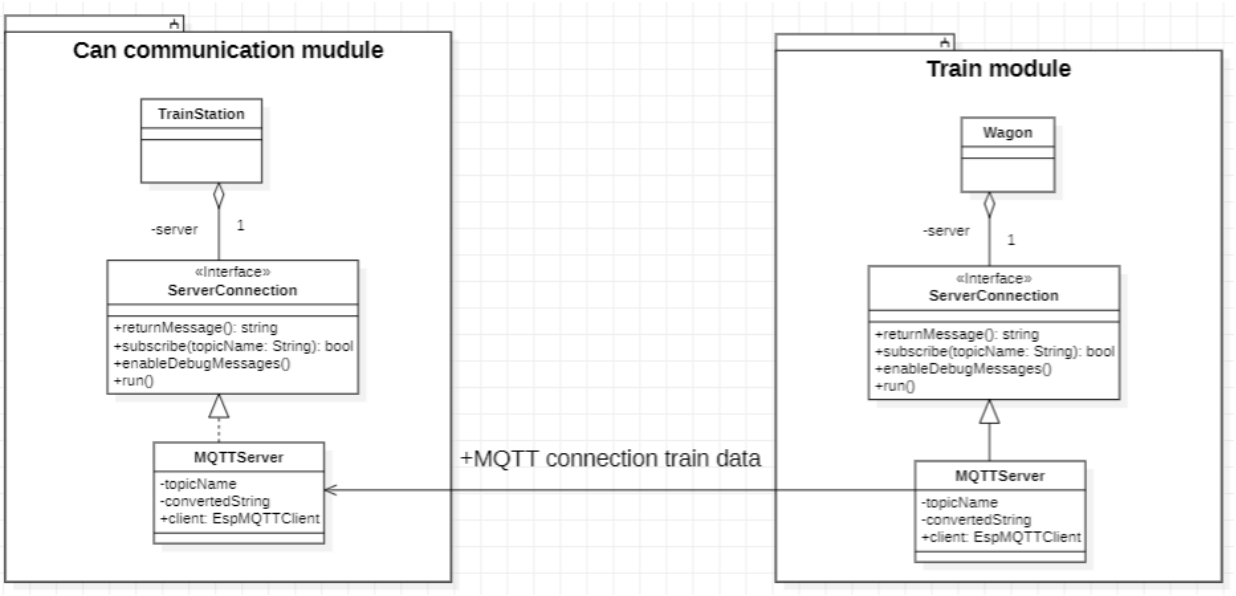
<<Interface>> CommManager	Interface for managing wagon communication

CanManger	Specific Can communication implementation, takes care of CAN communication, reads and writes CAN messages

CanEvent	Can communication event, contains the message and the id where the given message came from.

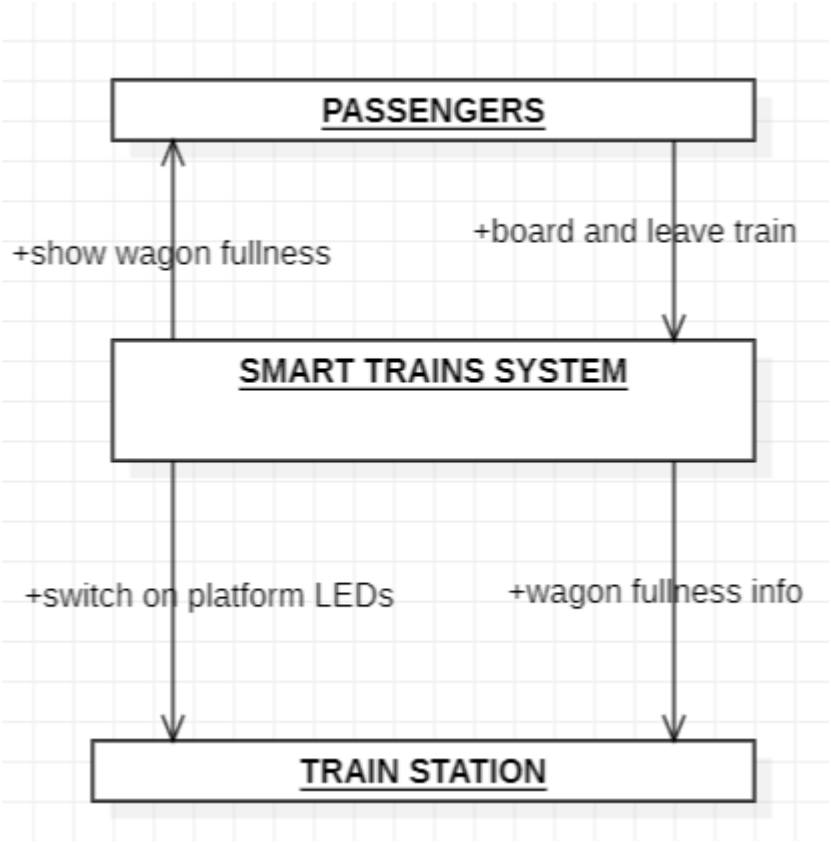
CanEvents	Event enums to differenciate
States	
No_Can_Msg	
Head_Wagid_Recieve	
Wagon_Wagid_Req	
Wagon_Occup_Resp	
Head_Occu_Req	

Whole system:



System design & communication

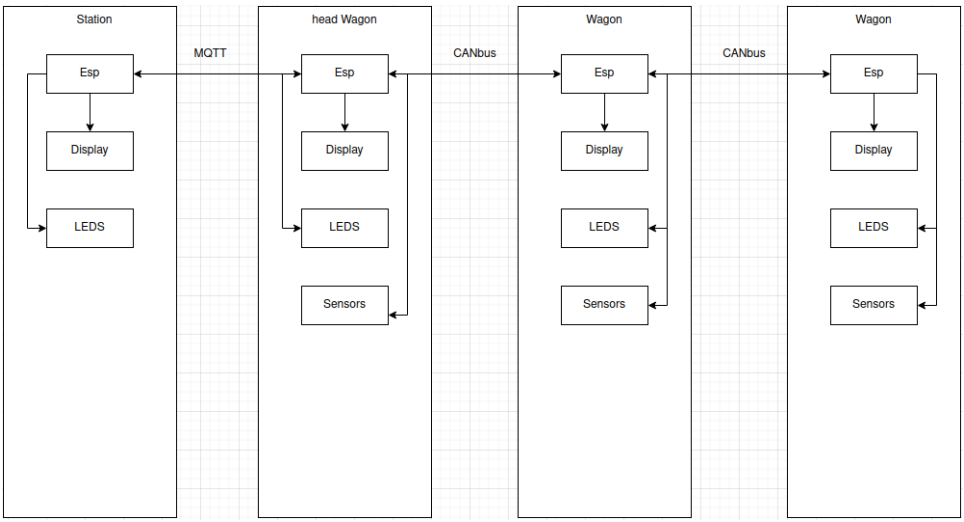
System interaction (context diagram)



In this diagram the system and its connection with external entities is detailed. There are only 2 of them according to our current design. Passenger and Train Station. Passengers can board and get off from our trains wagons. Communication with the Train station is needed so the information that we provide to passengers are all correct (direction, enough wagons, occupancy levels).

System structure (system architecture diagram)

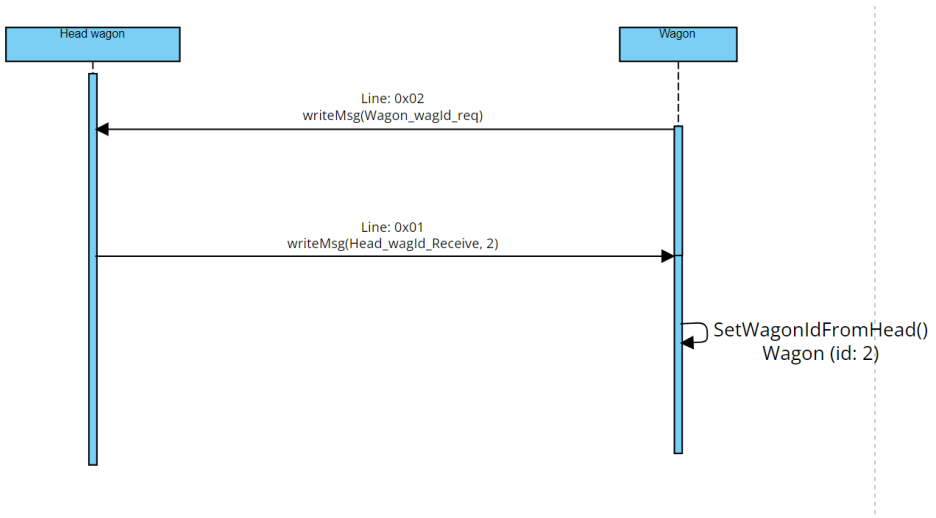
Components connection diagram



Component interaction (sequence diagram)

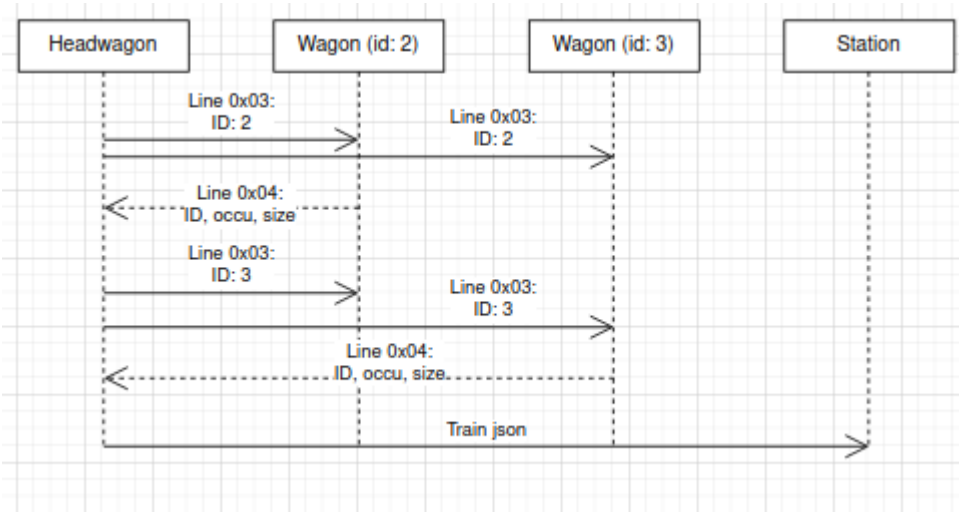
1. New wagon connects and gets a new wagonID from the head wagon

The sequence diagram below shows what happens when a new wagon is connected to the system as the new wagon will ask the head wagon for an ID to differentiate itself from existing/future wagons and to know its place in the wagon order. The new wagon sends a request for an ID on line 0x02 to the head and the head will send an ID to the new wagon on line 0x01. In this case, the new wagon is only the second wagon of the train so the ID that was sent to it was 2. If another new wagon is connected, its ID will be 3.



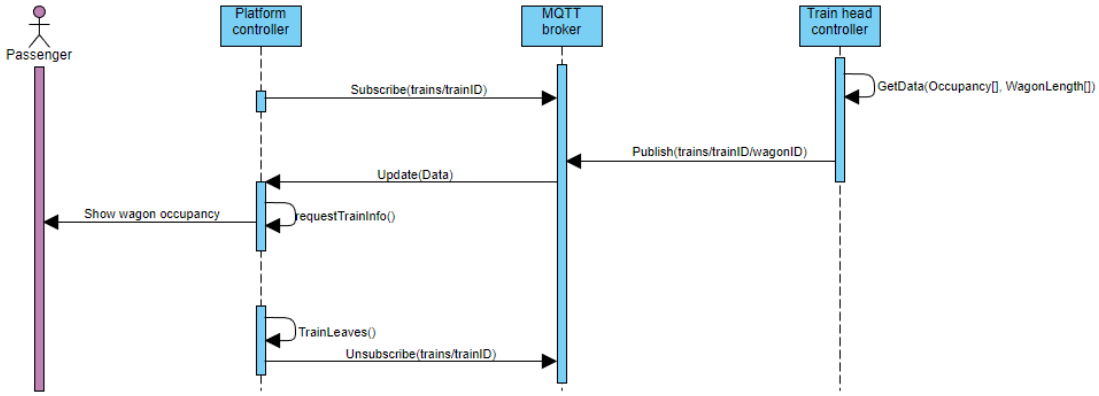
You can see below how the head wagons ask the other wagons its occupation and their size. After it has got all the information it sends it to the station with a Json.

2. Train station receiving data from the head wagon



3. Train station receives data from wagons, based on that lights up the appropriate platform LEDs

Happy flow 2 – Station shows occupation of the wagons.



4. People leaving at station

Unhappy flow 1 – Sensor fails

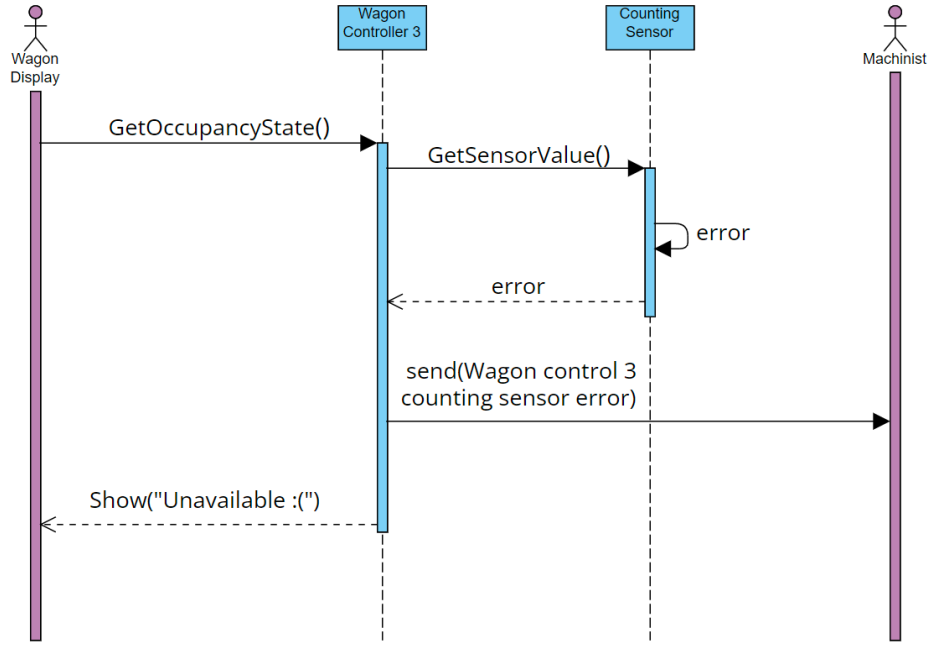
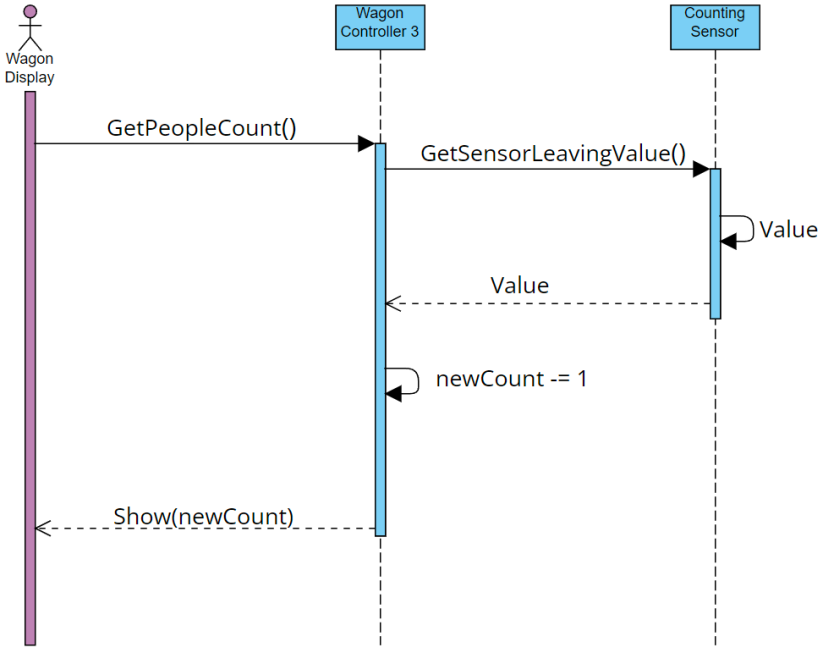


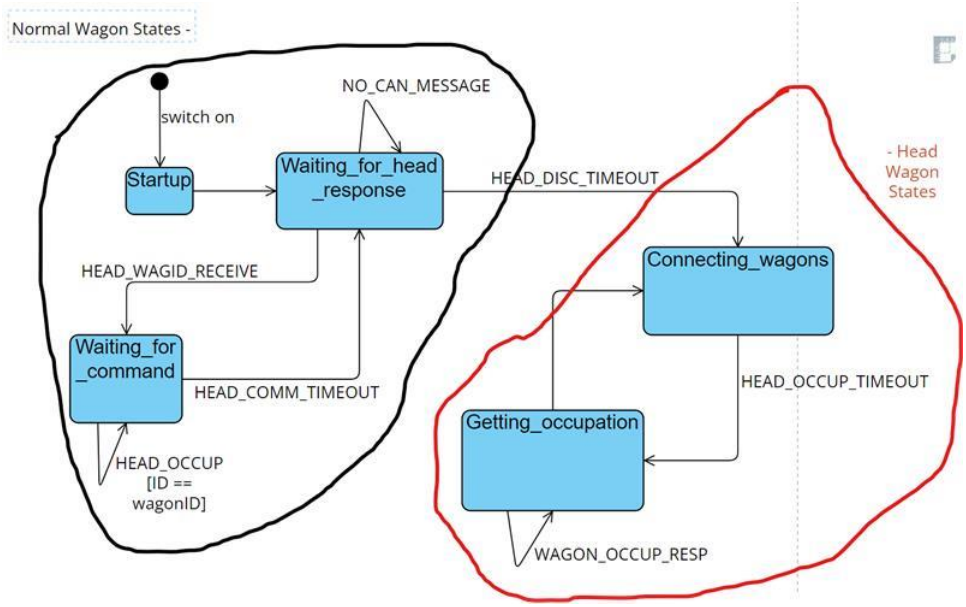
Figure 5 - Scenario 4- Unhappy flow 1

Happy flow 2 – Count goes down by 1



State-Machine Diagram

The diagram below shows the state machine. The states inside the red circle are for only the head wagon while the ones inside the black circle are for the normal wagon.

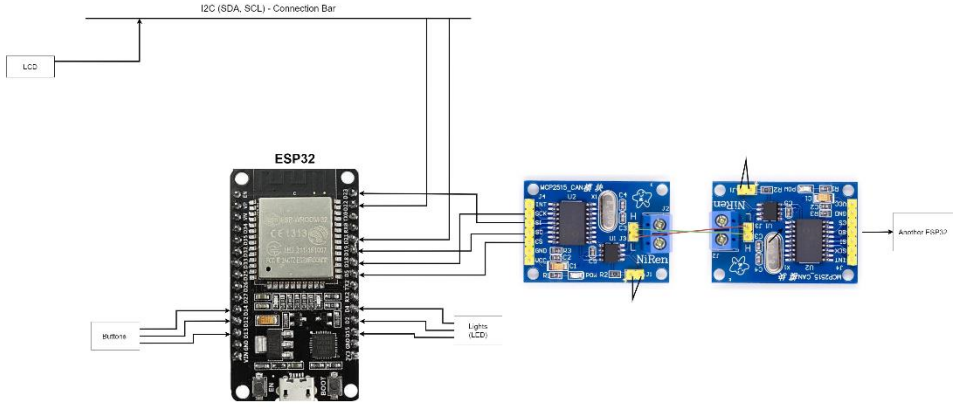


Events	Description
NO_CAN_MESSAGE	When no message available through CAN, remains in the same state and continues to check and wait for a CAN message.
HEAD_WAGID_RECEIVE	Normal wagon receives its ID from head wagon and moves to the next state.
HEAD_DISC_TIMEOUT	Searches for a head wagon and if no head available in the allotted time, a timeout will occur, and it will become the head and move to the next state.
HEAD_COMM_TIMEOUT	Waits for a command from the head wagon for a certain time and if no command given, timeout will occur, and it will move to the waiting for head response state and could potentially become the head or start a head selection due to there not being a head.
HEAD_OCCUP [ID == wagonID]	If no timeout, then it has received a command from the head wagon which can only be the head asking it it's occupation through specifying its specific wagon ID. This occupation is then sent to the head.
HEAD_OCCUP_TIMEOUT	After a certain timeout, the head wagon asks for its own occupation by moving to the next state.
WAGON_OCCUP_RESP	The head wagon's response to itself asking the occupation, head wagon gets its own occupation.

Message protocols

Message	Line ID	Meaning	Response
New id wagon	0x01	A new wagon connected to the train and the head sends the ID for the wagon.	No response
Requesting id from head	0x02	A new connected wagon will send this message to get a new id from the head wagon. If a message is written on line 0x01 or 0x03 without its id it resets the timer.	1) A new ide from head on line 0x01 2) No response this wagon will take head
x	0x03	Head wagon want to get the occupation of the train to send to the station	1) Wagon response with its id, size and occupation on line 0x04 2) No response after certain amount of time it goes to next id
Give occupation	0x04	Wagon got asked for its occupation. It will send its id, size and occupation to the head wagon.	No response

System module diagram



Sensor/Actuator	Pin protocol	Pin
3 LED (only station module)	(PWM)	GPIO 15, 2, 0, 4, 16, 17, 5, 18, 19, 21, 22, 23, 25, 33, 32, GND
3 Buttons (only wagon modules)	(Digital) - Input	GPIO 12, 13, GND, 3.3V
LCD (station & wagons)	(SDA), (SCL) - (I2C)	GPIO 21, 22, GND, 5V (VIN)

CAN Bus (only wagon modules)	(MOSI), (MISO), (CLK), (CS)	GPIO 23, 5, 18, 19, GND, 3.3V

The ESP32 has connections to LEDs, buttons, LCD and a CAN bus as shown in the diagram above. The CAN bus module connected to the ESP is connected to another CAN bus module which will be connected to another ESP.

Justifications

Component choices

The ESPs represent a wagon, and one ESP will represent a station and due to making the system decentralized, any wagon should have the ability to wirelessly communicate with the station which is why ESPs were chosen. The buttons were chosen to manually change the occupation of a wagon where pressing it increases the occupation count. The CAN bus was used to have a wired communication between the wagons to send and receive information. The LEDs were used to visually show how busy a wagon is. If the LED turns green then the wagon is empty or mostly empty, blue means it is slightly busy, but seats are still available while red means almost/completely full wagons. The LCD was used to display the direction in which to go to an empty wagon if the current wagon is full.

Task distributions

Project design and Can communication - János Gorondi:

- *Design of project:*
  - I was the main architect in this project, I laid out most of the fundamental concepts, and ideas. Although our team always discussed each decision with each other, to go for the best solution, that everyone agrees with and understands.
- *Design of classes and their connection:*
  - I designed the class diagram with as many software development principles in mind as I could. Sometimes I even overcomplicated parts, but my team was always able to help me with reducing complexity.
- *Implementing the structure of the code:*
  - I created the folder structure, files and the base of the classes and committed them to git, so we could start working together.
- *Solving git related problems:*
  - I am not an expert with git but have worked with it previously. For my teammates it was a new concept, so I helped to solve merge conflicts, branch related problems and rebases, but I still have a long way to go.
- *Can communication module and its integration:*
  - My individual part in the project was the design, implementation and testing of the communication module. After integration it was working well relatively fast.

- *Organizing, motivating and helping other teammates:*
    - I tried to always look out for my teammates to ensure that everyone passes this project. If I saw that they needed help to understand concepts or lost motivation I was able to make them move forward. I tried to assign tasks to everyone so that we would all have individual parts to work on.
- *Writing documents:*
    - My role was to contribute to the introduction part and the writing of all the class diagram related text as well as managing, reviewing and correcting what other team members wrote.

Normal wagon - Chaitanya Schoka:

- Normal wagon functions

The functions related to the normal wagon where functions and constructors of different classes needed to be incorporated were implemented by me with the help of Janos guiding me on how to improve it and Owen helping me with debugging it due to me accidentally pushing to git without building the code. I faced challenges in using git as I have never fully used it as much as in this project which is why I made some mistakes but learnt from it and I better understand how to use git now than before.

- Data format conversion, sending and receiving through MQTT

I researched and implemented the conversion of the train object containing all the wagon information into a Json document to be sent through MQTT to the station and receiving it through the station and converting it back to a train object for use.

- Helped with implementing the state machine

I helped Owen with the initial implementation of the state machine and understood how to properly incorporate events with state machines.

- Implemented the seats

I implemented buttons in an OOP manner to manually increase/decrease seats occupied and implemented the sending of this information to the head wagon and to the station.

- Implemented the LCD screens for wagons

I implemented the LCD screens to show the state of the wagon (empty, normal, full), the number of available seats and the wagon ID. I also implemented an LCD screen to show in which direction to go to an empty wagon for the station, however there were bugs, and it did not work properly. I had to properly think about how to implement this functionality in an OOP manner and to integrate it with the already existing functionalities and code without breaking the overall system.

- Writing documents

I wrote the research report on the reason why we chose to use the CAN protocol for communication between the wagons. I also helped with the intro of the design document explaining the technical aspects. I helped in creating the sequence diagrams and the system module diagram that shows the hardware connections so that the wiring is done correctly.

Head wagon – Owen van Uden:

- Head wagon selection

I created a big part of the head wagon selection. For this I created the part that checks if there is a head wagon already.

- Debugging code

  I also debugged a big part of the code for the wagons because there was some code pushed to the main branch that was not built yet

- Making hardware setup for the train/ wagons.

I made hardware connection between the different wagons and the can modules.

- Implementing state machine

I implemented the state machines of the wagons that were made János.

- Adding security CAN messages

After seeing that the can messages worked, I added a small security feature to the wagons so that wagons cannot get the same id at the same time.

- Helped with getting the occupation of the train and sending it to the station

I helped chai with implementing the occupation getting and sending it to the station.

- Writing documents

I helped with writing the document for the project and helped with some of the design that was needed for the project.

Train station - Joep van Dijk:

- Train station

My main task for this project was to make the demonstration and implementation for the train station. I made some extensions and implemented all the necessary functionalities of the train station. I did this by using Object Orienting Programming(OOP). The goal was to make a visual representation of a platform and to show for each of the wagons of the train that is going to arrive at the station its occupancy by changing the color of the LEDs.

- Documentation and design

I helped with the design steps of how the wagons communicate with each other and how the station should respond. And created some of the diagrams in the design document for it and I also helped with the documentation of the analysis document for the use cases, happy and unhappy flows. And upon these flows I created some sequence diagrams inside the design document.

- Code
  - Implementation MQTT

The implementation of the MQTT class and functionalities within our project is done by me.

- Implemented the functionality at the train station and extended the classes.

I implemented the functionalities of the structure of the code for the train station. I tried to make the train station code as extendible as possible. But on further review of Sioux this wasn't completely necessary. I also extended the population of certain classes.

- Hardware setup for the train station

How the train stations LEDs are connected and light up to show the occupancy and the length of each of the wagons is made by me.

Reflections

Owen van Uden

Looking back on the project it started a bit slow with the design phase where the big lines of the project were not understood by most of the group. But after having the first couple of sessions with the team mates the project was better understood by everyone in the group. After this the pace started to pick up and people could start with the coding part of the project. Here there were a couple of problems. One big problem was that the CAN communication between the wagons were sending and receiving correctly. This problem was fixed with some wiring fixing. After this, we finished the remaining parts at a good pace. And looking back I am happy with the final delivery of the project.

Joep van Dijk

When I reflect on this project from the beginning, we had some difficulties with the overall design of the system and how we were going to get it to work. This is also why we were a little behind on the sprints. We eventually got the design of the system right but underestimated how much work everything would be. So, some of the implementations we had to drop because of time restrictions. But I think in the end we have a very nice system overall.

We as a team worked well together and we're very understanding and helpful when things didn't really go as planned or when people had some difficulties. Everybody took their part very serious, and we helped each other out when necessary.

For myself I noticed that in the beginning I had a more active role within the group, but the longer we where in the project I noticed I had more difficulties with that, but fortunately other people within the group also were taking that role. I think I made some good progress in my confidence with working in groups again because of this project and the way we communicated with each other. On the technical side of the project, I was a little slow with getting to know how to use git and visual studio code and I had some difficulties with getting the understanding how the structure and OOP part of the code should be working. And in the end Sioux also told me that I was making it too difficult because the extendibility of the code wasn't going to be shown with the demo so some of the code wasn't necessary. But overall I am very satisfied with my progress this semester.

Chaitanya Schoka

Initially the project started slowing with people getting sick and most of the time was spent on making sequence diagrams without having a clear picture of the solution. Due to this, when we started coding, our initial code had to be completely revamped according to Sioux. However, once we started remaking the code according to a clear picture and state diagram, the code was looking much better, and the project picked up speed. Thus, our first sprint goals were not fully completed but from sprint 2 onwards, most of the goals were completed and we were back on track.

There were certain problems we faced such as me personally not knowing how to use git, but I learnt this with the help of my teammates. I also merged code accidentally that was not built and checked for errors which is something I learnt from and improved. I also learnt a lot on how to code in an OOP manner on such a large scale for a group project. I had help and was taught efficient and OOP centric methods while creating code from my teammates. Thus, I definitely progressed and learnt a lot during this project.

As a team, we faced many bugs and issues in relation to the CAN communication and the choosing of a head wagon. These bugs and issues took 2 sprints to fix and might have been longer if not for the help of Sioux. Other than these issues, the rest of the tasks and project went smoothly and the way we as a group tackled, solved and completed this project was impressive. We were all actively participating and were present in all meetings, which is always a good sign.

János Gorondi

I really liked the idea of the project. It was not a standard idea, but it seemed fun, so we just went with it. After completing the full project, I can confidently say that programming distributed systems is a hard challenge. Thanks to the project I got a little insight into how to work with them.

Because my team members reactive behavior in the early stages of the project, I had to step up as a leader. I organized the meetings and motivated others while trying to make them understand the whole big picture. This was a new role for me, and I feel like I grew with the responsibility and managed to guide the team, to finish the project on time. Last week I could not attend meetings because of family matters, but the whole team worked together and managed to make a lot of progress without me as well. I was really happy that I could count on my team.

This whole project has made me see working with groups in a new way, I learn a lot about working with people from different cultures with different ideas.