FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	ETCS - Dynamic speed monitoring module for EVC
Student:	Alina Krasnenkova
Supervisor:	Ing. Jan Matoušek
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2022/2023

Instructions

ETCS (European Train Control System) is a train protection system and is the signaling and control component of the European Rail Traffic Management System (ERTMS). EVC (European Vital Computer) is the part of the ETCS. The main goal of this project is to make a dynamic speed monitoring module for EVC.

Instructions:
1. Analyze existing ETCS documentation (ERTMS/ETCS System Requirements Specification subsets 026, issue: 3.6.0).
2. Specify functional and non-functional requirements for dynamic speed monitoring module.
3. Design and implement MA (Movement Authority) module prototype for receiving necessary MA messages and packets.
4. Design and implement algorithm to calculate MRSP (most restrictive speed profile) based on received MA messages and packets and known railway restrictions.
5. Design and implement a module prototype for dynamic speed monitoring based on MRSP.
6. Test all of the implemented modules.
7. Collect experience and propose future development of the project.

Electronically approved by Ing. Michal Valenta, Ph.D. on 24 January 2022 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

ETCS - Dynamic speed monitoring module for EVC

Alina Krasnenkova

Department of Software Engineering
Supervisor: Ing. Jan Matoušek

May 10, 2022

Acknowledgements

I would like to thank my supervisor Ing. Jan Matoušek for his advice and guidance throughout writing of this thesis. He was always ready to help. I would also like to thank Ing. Jiří Chludil and doc. Ing. Martin Leso, Ph.D. for their readiness to share professional knowledge.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 10, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Krasnenkova, Alina. *ETCS - Dynamic speed monitoring module for EVC.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

První část této práce udává přehled o ETCS, popisuje EVC a jeho hlavní funkce potřebné pro řízení rychlosti a vzdálenosti vlaku. Druhá část popisuje návrh a implementaci modulu vytvořeného pro tuto bakalářskou práci. Tento modul je rozdělen do tří submodulů: přijímač zpráv a paketů, kalkulačka nejpřísnějšího rychlostního profilu a dynamické monitorování rychlosti.

Klíčová slova ERTMS, ETCS, EVC, Vlakový zabezpečovací systém, Dynamické monitorování rychlosti

Abstract

The first part of this thesis gives an overview of the ETCS, gives the description of the EVC and its main functionalities, that are necessary for the speed and distance monitoring of the train. The second part describes the design and implementation of the module, that is created for this thesis. This module is parted into three submodules: messages and packets receiver, MRSP calculator and dynamic speed monitoring.

Keywords ERTMS, ETCS, EVC, Train control system, Dynamic speed monitoring

Contents

List of Figures

List of Listings

Introduction

Railway transportation is one of the main kinds of transportation in almost every country. Train transport is very commonly used, inexpensive and can be used for both short and long distances. As it is convenient, it should also be safe to travel on, so a lot of different safety measures are used to minimize the risk of accidents. One of this measures is to present an international automatic train control system, which will ensure safe movements of trains and compatibility throughout different countries. The European Train Control System (ETCS) gives the specific technical information necessary for that.

ETCS system defines two subsystems, the on-board and the trackside. The ETCS on-board equipment is a computer-based system, that supervises the train movement. The heart of this equipment is the European Vital Computer (EVC). Dynamic speed monitoring is one of the main functions of EVC, because it controls the speed and the distance of the train and triggers train brakes whenever the situation calls for it. It helps to avoid dangerous situations and ensure safe movement of the train.

There is no such system in Czech Republic yet, so presenting modules for the ETCS can improve transport system inside the country in the future. That were the main reasons why I chose to proceed with this topic. I wanted to attract attention to the safety transport issues, which are not addressed yet.

This thesis is more or less a continuation of the implementation I have done during the software project course. The main focus of this thesis lays on designing and creating a dynamic speed monitoring module for the on-board system, so that the safe train movement can be possible for trains inside ETCS.

Theoretical part of this work describes the actual ETCS system and it's components, the ETCS simulator, that was developed at FIT CTU and FTS CTU during software project course, and main EVC functions, that are used for the dynamic speed monitoring.

Practical part consists of designing and implementing the whole dynamic speed monitoring module for EVC and essential submodules, that will help

that happen. The first part will describe different submodules and their functionality. Then the implementation details will be listed.

The main goal of this thesis is to create a functional dynamic speed monitoring module for EVC.

Goals for the theoretical part of this work are to analyze the existing ETCS documentation, that is described in ERTMS/ETCS System Requirements Specification subsets 026, issue: 3.6.0. The next goal is to describe ETCS, it's components, especially EVC, because the main work will be done for that exact component. Previously done work will be analyzed and described. As the communication between ETCS components is done by sending messages and packets to each other, they must be described and analyzed. The next main feature of this work is to take look at speed profiles, that are used for calculating most restrictive speed profile, that is later used for dynamic speed monitoring. At last, the dynamic speed monitoring function will be analyzed.

Goals for the practical part are to design submodule for receiving messages and packets, and implement it. Then process received speed profiles inside those packets. After that, to design an algorithm for calculating the most restrictive speed profile from the previously received speed profiles. The last goal is to design and implement the dynamic speed monitoring module, using the calculated most restrictive speed profile.

To sum it all, the created module must receive and process messages and packets from EVC. It should calculate MRSP from the data gained from those messages and packets. And it should monitor speed and distance of the train and trigger and release brakes commands, when necessary.

Chapter 1

Analysis

1.1 European Train Control System

Train control plays a very important role in a railway operations system. Different numbers of Automatic Train Control systems were created throughout the years in different countries. The main problem of these systems is incompatibility with each other. Introducing an international standardisation of Automatic Train Control system will not only solve this problem, but will also ensure the safety of the international traffic.[6]

European Rail Traffic Management System/European Train Control System (ERTMS/ETCS) gives the specific technical information necessary for the unambiguous understanding of the operating rules.

ETCS system defines two subsystems, the on-board and the trackside. The trackside consists of balise, lineside electronic unit, the radio communication network, the Radio Block Centre (RBC), Euroloop, Radio infill unit, Key Management Centre and Public Key Infrastructure. For its part, the on-board consists of the ERTMS/ETCS on-board equipment and the on-board part of the GSM-R radio system.[6]

The ETCS on-board equipment is a computer-based system, that supervises the train movement.

ETCS operates on different levels. These levels are used to determine the communication between trackside and on-board units and the functions to be used on these levels.

Levels 1 to 3 provide a continuous speed supervision system, which also protects against overrun of the authority.[16]

Level 0

This level is used when the track does not have operating train control system. Or the use of this system currently not possible.

Therefore different signalling are used to give movement authorities to the driver.

The following functions are present:

1. Supervision of maximum train speed.
2. Supervision of maximum speed permitted in an unfitted area.
3. Reading of Eurobalises to detect level transitions and certain special commands. All other messages are rejected.
4. No cab signalling. [7]

Level 1

This level is used when the track is equipped with Eurobalises and optionally Euroloop or Radio Infill.

[7]In this level movement authorities generates trackside and transmit them to the train using Eurobalises.

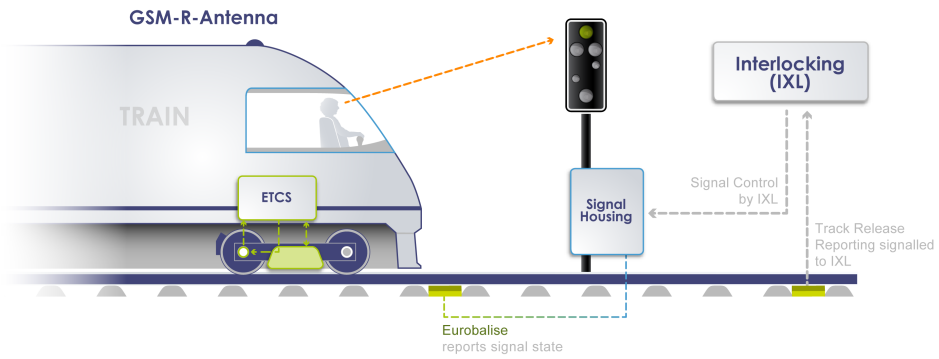

Figure 1.1: ERTMS/ETCS Level 1 [16]

Level 2

This level is used when the track line is controlled by RBC and equipped with Eurobalises and Euroradio. In this level train position and train integrity proving is performed by the trackside.

In this level movement authorities are generated by the trackside, that transmits them to the train using Euroradio.

This level is based on Euroradio communication between track and train. And uses Eurobalises for location references.[7]

Level 3

Is the same as level 2 with the exception of train position and integrity being supervised based on information received from the train.

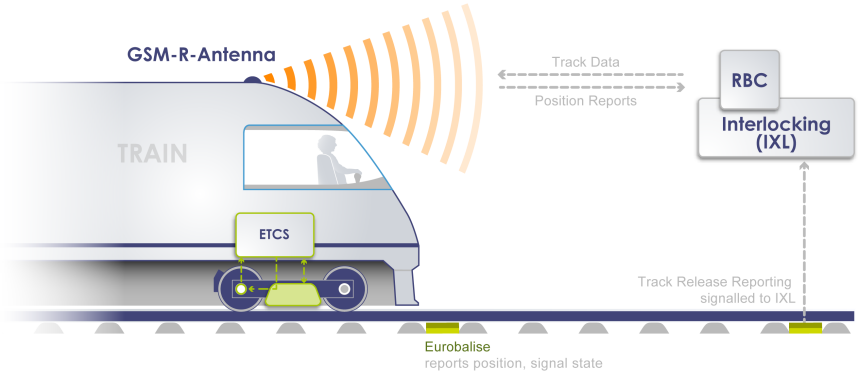

Figure 1.2: ERTMS/ETCS Level 2 [16]

This level is based on Euroradio communication between track and train. And uses Eurobalises for location references.[7]

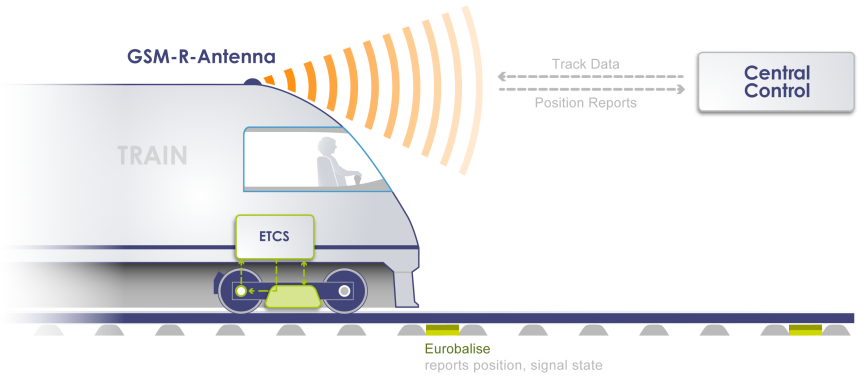

Figure 1.3: ERTMS/ETCS Level 3 [16]

Level NTC
Describes a line equipped with national system.

[9]ETCS system runs not only on levels, but uses different modes for different purposes as well. It is used to assign some functions to a particular mode. The following modes are present in the ETCS system:

Full Supervision (FS) is when the train runs with the full supervision of ETCS.

Limited Supervision (LS) is when the train runs with the ETCS supervision, but it is limited. This mode cannot be chosen by the driver, so it switches to this mode, when the train enters selected piece of track.

On Sight (OS) is when the train enters the piece of track, which is occupied by something else, e.g. another train.

Staff Responsible (SR) is when the driver takes full supervision of the train on a ETCS equipped piece of track.

Shunting (SH) is used for shunting movements.

Unfitted (UN) is used when the train enters the track, where there is no ETCS system, or it is not available for using.

Passive Shunting (PS) is for managing the ETCS on-board equipment of a slave engine, being part of a shunting consist.

Sleeping (SL) is the same as PS but with remote controlled slave engine.

Stand By (SB) is the basic mode when EVC first powers on.

Trip (TR) is when the emergency brakes are triggered.

Post Trip (PT) is when the mode leaves the trip mode.

System Failure (SF) is when some error occurs, affecting safety. The emergency brakes in this case are used.

Isolation (IS) is when the ETCS on-board equipment is not connected to other devices.

No Power (NP) is when EVC is not powered.

Non Leading (NL) is the same as PS but with a slave engine that is not electrically coupled with the leading engine, but has its own driver.

National System (SN) is when the National system through STM interface can access the resources using the on-board equipment.

Reversing (RV) is when the train can run in the opposite direction.

1.2 EVC and speed profiles

Existing ETCS Simulator:

ETCS SIMULATOR

The ETCS simulator, that is developed at FIT CTU and FTS CTU aims to simulate ETCS system. The simulator is based on the specifications which are shown in subsets version 2.3.0. The simulator currently implements functions, operating on levels 2 and 3 for the modes Full Supervision, Stand By and No Power. For the simulator, the following modules are implemented:

DMI module helps the driver to communicate with the whole ETCS system. It receives the data, neccesary for displaying various data on the screen for the driver to process. Meanwhile, it sends the data back to other modules, to ensure that the driver's inputs are heard.[1]

RBC module acts as a guidance system for the train. It calculates the rout and sends data to EVC module with movement permission. It ensures the safety of the route.[3]

JRU module is responsible for containment and preservation of the data in communication between EVC and other modules.[5]

Braking Curve module calculates the braking curve necessary for the safe movement of the train.[4]

EVC module acts like a communication module between every other module in ETCS system.[2]

For EVC the functions are the following: communication between other modules and EVC using messages and packets, determining the most restrictive speed profile based on the SSP, supervision of the train speed based on MRSP, MA.

The goals of this work are to implement determination of the MRSP based on SSP, ASP, TSR, Signalling related speed restriction, mode related speed restriction, train related speed restriction, STM max speed, STM system speed, LX speed, speed restriction to ensure a given permitted braking distance, override related speed restriction.

Previous implementation of EVC gives the speed monitoring based on MRSP. But MRSP in this context consists only of the static speed profile. SSP is the basic speed profile, which is given by the trackside to EVC via messages and packets. Usage of the one SSP alone does not fully represent the whole restrictions in the real world and ETCS. EVC was based on the subsets version 2.3.0, and this work will be based on the version 3.6.0. While there is no difference in some other things, even the basic SSP in this newer version is quite different. In version 3.6.0 there is a more complex view of the speed profiles. SSP has not got only basic speed restrictions, but also restrictions,

which are connected to the train categories. There are few arguments, which are passed through the packets, which help to determine the right speed limit, whereas in the previous version the train category is nowhere to be seen.

There are few things that must be taken into account when calculating MRSP, and one of them is a train length. Some of the speed restrictions ask for the whole train to pass the distance of the limit, before speed increase is taken place, while other speed restrictions do not.

MRSP will be calculated, depending on those speed profiles:

1.2.1 Static speed profile

SSP is a description of the fixed speed restrictions of a given piece of track. These restrictions are based on both train and track dependent factors, and are usually related to bridges, curves, tunnels and other track related things.

In subsets version 3.6.0 SSP provide two categories : Basic and specific. The basic one is to be used on every train, while the specific one specify the train category, on which the limit must be applied.[8]

There are two types of the specific categories:

1. Cant deficiency.
2. Other specific.

Trackside shall always send the basic SSP for the track route, while the specific SSPs are optional.

EVC shall select the SSP best suiting its Cant deficiency train category according to the following order:

1. Cant deficiency SSP matching its cant deficiency train category.
2. Cant deficiency SSP with the highest cant deficiency value below the value of its cant deficiency train category.
3. Basic SSP.

The corresponding train category must be send as a part of the train data.[8]

1.2.2 Axle load speed profile

Speed restriction due to axle load is given to different sections. For each of them are given speed values and for which minimum axle load category this speed value applies.

EVC must take the value which is the lowest due to axle load category, which is lower than or equal to that of the train.[8]

1.2.3 Temporary speed restrictions

Those types of restrictions are usually used for some working areas. It is convenient to use this type of restriction, to avoid temporary changes of the SSP.

Temporary Speed restrictions are independent of each other and can easily overlap. In those cases the minimum value of the overlapping restrictions is used.

Each TSR has its own identity. It is used for the purpose of revocation. Trackside can decide to revoke some TSR, so it sends a revocation message with the identity of the TSR to be revoked. When EVC receives this message, the restriction is revoked immediately.

It shall be noted, that TSR can be replaced by another value, if the new TSR for the existing identity is received.[8]

1.2.4 Signalling related speed restrictions

This type of restriction is used only in Level 1. When the value is received, it is applied immediately, for the exception of the restriction received from infill device. This restriction is valid until a new one is received. In case of the changing levels from 1 to 2 or 3, the restriction is valid until MA is received.[8]

1.2.5 Mode related speed restrictions

The values for this particular restriction are defined by default or national value. [8]There are some exceptions for the rule:

1. For some of the modes the restriction can be given from the trackside. For the On-sight, Limited Supervision and Shunting modes.
2. For the Reversing mode the value is always send by the trackside.
3. The driver can enter the speed for the Reversing mode.

1.2.6 Train related speed restriction

It represents the actual possible maximum train speed. It is send via message as part of the train data.

1.2.7 LX speed restriction

Speed restriction for when the train has to pass a non protected Level Crossing.

1.2.8 Override function related speed restriction

This restriction is used only if the override function is active.

1.2.9 Determination of Most Restrictive Speed Profile

MRSP is a description of the most restrictive speed restrictions the train shall obey on a given piece of track.

The MRSP must be computed from all the different speed restrictions given by the track. If on the same section of the track different restrictions are overlapping, then on that section the lowest value must be applied.

When the new restrictions are given to the EVC or the previous ones are deleted/changed, then MRSP must be immediately recalculated and applied.[8]

Some restrictions as axle load speed profile, international static speed profile and temporary speed restrictions are given to EVC by RBC via Euroradio. RBC sends message with movement authority and optional packets inside. In those packets there are descriptions of those speed profiles. Based on those profiles the MRSP is calculated.[10]

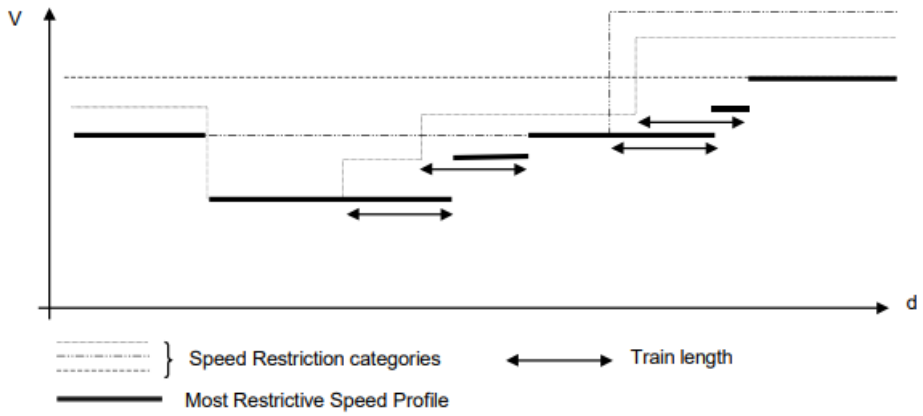

Figure 1.4: Most Restrictive Speed Profile [8]

1.3 Messages and packets

For communication between different modules in ETCS messages and packets are used. Messages and packets consist of different variables, values of which have to be assigned. Some of these variables must be unique, which is important for proper communication and unambiguous understanding of the received information. Therefore, those messages and packets must be handled due to common rules, given in the subsets 7 and 8.

Each variable has its own given size and listed values, so the handling of them is easier and well defined. Some of the variables have special values,

which are internationally used. Some of them have spare values, which can be used in later development. Names of variables are unique and the description is provided.[10]

Packets consist of number of variables with a defined internal structure. Packets are formed that way, as to create a complete group of variables to define some unit, e.g. Axle load speed profile.

Each packet always has packet identifier which is unique for each packet, and packet length, which represents number of bits in the packet.

For this work the main packets are:

Packet number 27: International Static Speed Profile represents the static speed profile to be processed by the EVC. This packet comes from RBC. Values D_STATIC represent incrementing distance to the next change of static speed profile, and V_STATIC represents the according speed, that the train shall not overcome. Q_DIFF shows if the train shall follow the Cant Deficiency train category speed limit, and whether or not the other specific category can override the Cant Deficiency speed limit value. NC_CDDIFF represents the Cant Deficiency category, NC_DIFF - other specific category. And V_DIFF represents the according speed.[10]

Packet number 15: Level 2/3 Movement Authority represents the movement authority on levels 2 and 3.

Packet number 12: Level 1 Movement Authority represents the movement authority on level 1.

Packet number 51: Axle load Speed Profile represents the axle load speed profile which is given to EVC by RBC. The value Q_TRACKINIT shows whether or not the initial states should be resumed or the speed profile follows and shall be applied. If the value of this variable equals to 1, then the variable D_TRACKINIT follows. It shows, that the empty speed profile will start at that distance. If this variable equals to 0, then the other variables follow. D_AXLELAD represents the distance to the axle load speed profile. L_AXLELOAD is the length of the according speed profile. Q_FRONT shows whether or not the length of the train shall be taken into consideration when the speed profile applies. M_AXLELOADCAT is the axle load category itself, and V_AXLELOAD is the speed corresponding to the according axle load speed profile.[10]

Packet number 65: Temporary Speed Restriction represents the temporary speed restrictions send to EVC by RBC. The variable NID_TSR is the id of the one temporary speed restriction. It is needed to possibly revoke this restriction by sendind packet number 66 to EVC. D_TSR represents the distance to the according temporary speed profile. L_TSR is the length of the temporary speed profile. Q_FRONT shows if the

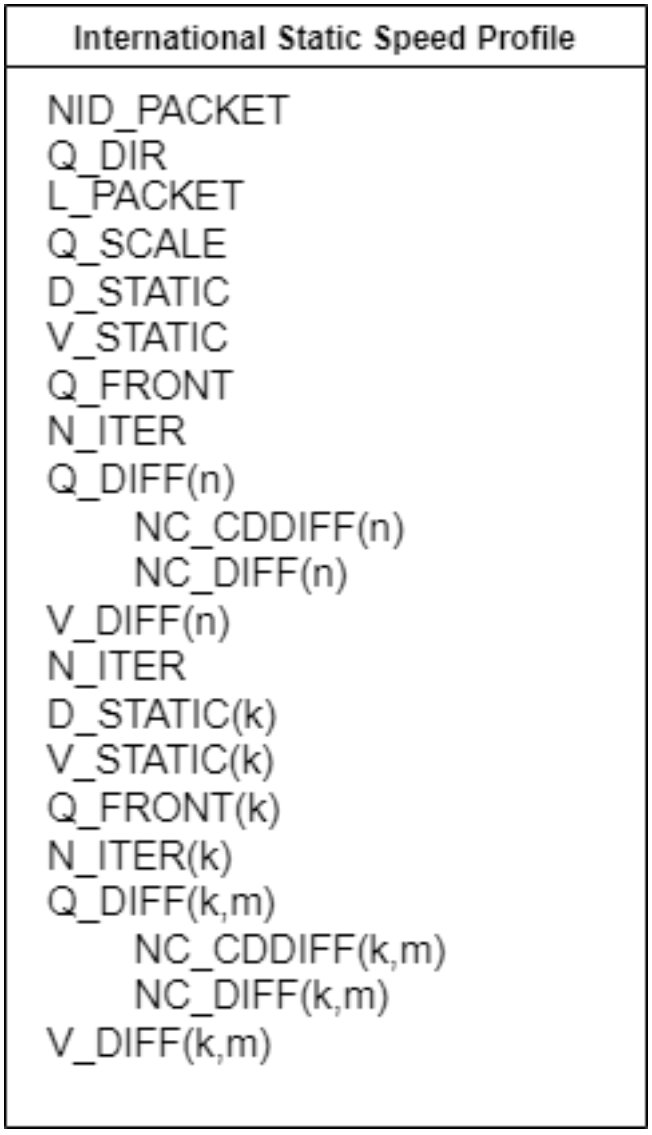

Figure 1.5: Packet Number 27

train length must be taken into account when applying temporary speed restriction. V_TSR is the speed for the temporary speed restriction. NID_TSR, when equals to 255, shows that this temporary speed restriction is not revocable.[10]

Packet number 66: Temporary Speed Restriction Revocation is the packet send by RBC to EVC. It is used to revoke temporary speed restriction which was previously applied. This packet has variable NID_TSR with the identity of the temporary speed profile to be revoked.[10]

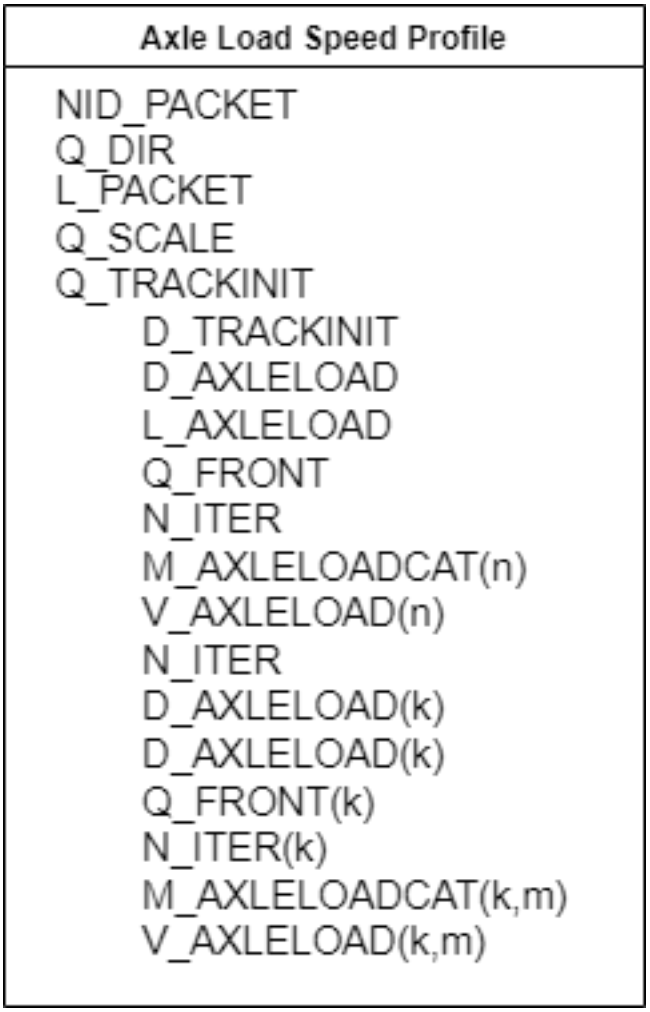

Figure 1.6: Packet Number 51

Messages are the main use of communication between modules in ETCS. They consists of different variables and optional packets inside. So communication goes on via messages, and packets are the part of them.

[11]There are three different types of messages used in ETCS communication:

Eurobalise message is the telegram which is transmitted by balises.

Euroradio message is the message which is transmitted from RBC to EVC or vise versa.

Euroloop message is the message which is transmitted by the loop.

A message consists of:

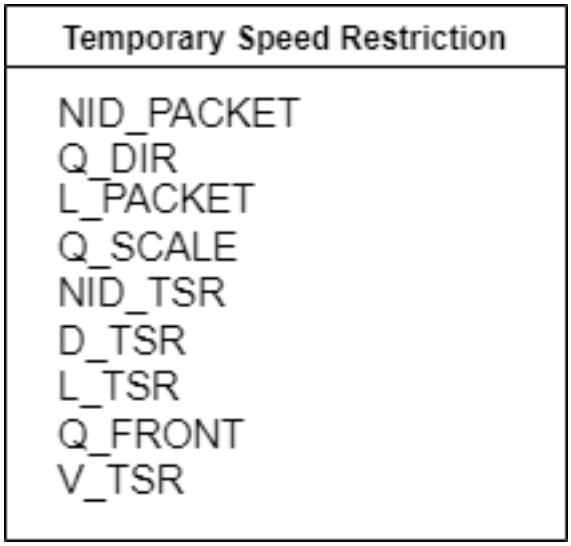

Figure 1.7: Packet Number 65

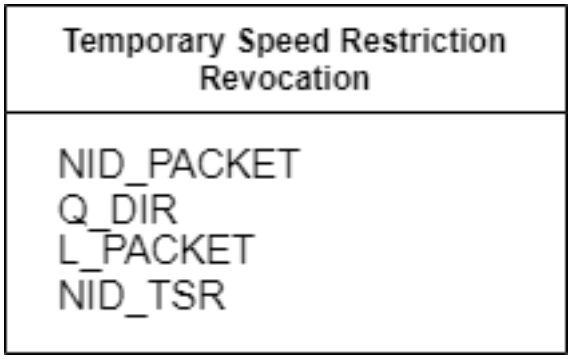

Figure 1.8: Packet Number 66

1. One header.
2. Predefined set of variables (only for Radio).
3. Predefined set of packets (only for Radio).
4. Optional packets.

Message 3: Movement Authority is send by RBC to EVC. It represents the Movement Authority and has additional packets to it with different speed profiles to be processed by EVC.

Message 37: Infill MA is sent to EVC and represents the Movement Authority but only in Level 1.

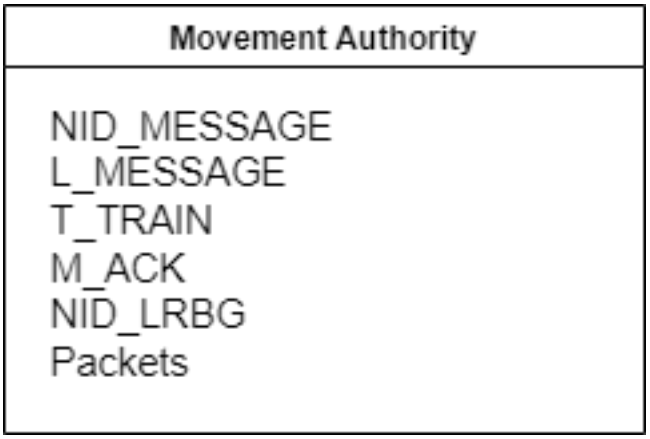

Figure 1.9: Message Number 3 and 37

1.4 Speed and distance monitoring

Speed and distance monitoring is one of the main features of EVC for assuring safety. It helps to monitor speed of the train and its position, so that the train remains within the given by the trackside limits. As when the movement authority is given to EVC by RBC, the rules and limits must be followed. Those speed and distance limits are send via Euroradio messages as discussed earlier.

In order to fulfill those rules, the brakes of the train must be functioning and the whole train related data must be entered to the system correctly.

The on-board system may trigger brakes when necessary, e.g. when train exceeds speed limit on the piece of track or when it exceeds the end of movement authority, or revoke brakes, when the speed of the train goes under the speed limit.[8]

Many different things are taken into account when speed and distance monitoring. For example, train length, maximum train speed and nominal rotating mass. As for the trackside, gradients, track related speed restrictions and track condition.

To make it easier to follow the movement of the train and to monitor its position, the distance is measured from the last balise group the train passed. It gives the best point from which all the coming speed profiles, gradients and movement authorities are count. So EVC must follow the balise groups and make calculations based on that.

For the handling of the speed and distance monitoring, some supervision limits are used. Those limits are helpful for the driver to maintain the speed of the train within the appropriate limits, so that the emergency brakes will not be triggered.[8]

Ceiling supervision limits are derived from the MRSP elements. Based on

it the permitted speed, warning, service brake intervention and emergency brake intervention are calculated.[8]

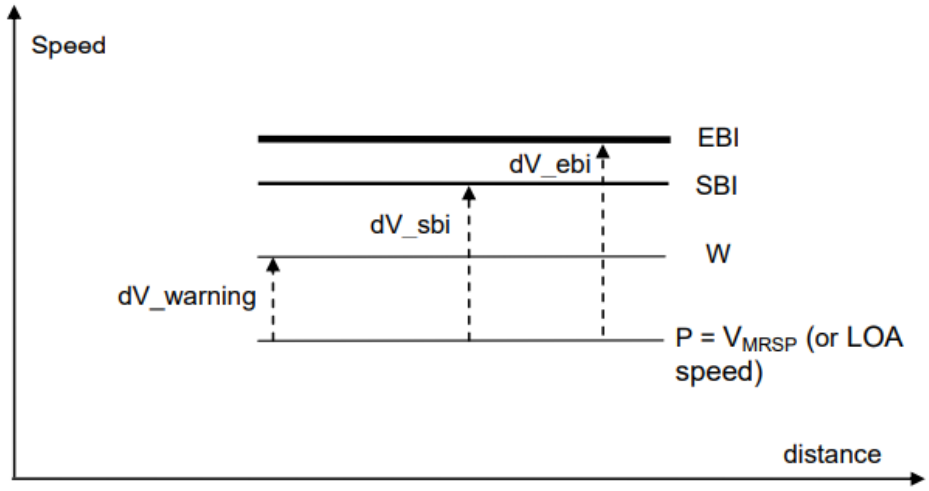

Figure 1.10: Ceiling Supervision Limits [8]

[8]The following speed supervision limits exist:

1. Emergency Brake Intervention (EBI)
2. Service Brake Intervention (SBI)
3. Warning (W)
4. Permitted speed (P)
5. Indication (I)
6. Release speed monitoring start location

[8]There are few different speed and distance monitoring types:

Ceiling speed monitoring (CSM) is used throughout the area where train runs without the need to bake to a target.

Target speed monitoring (TSM) is the type of supervision in the area, where train brakes to the target, and related information is displayed to the driver.

Release speed monitoring (RSM) is used in the area close to the end of authority. There the train can run within the release speed.

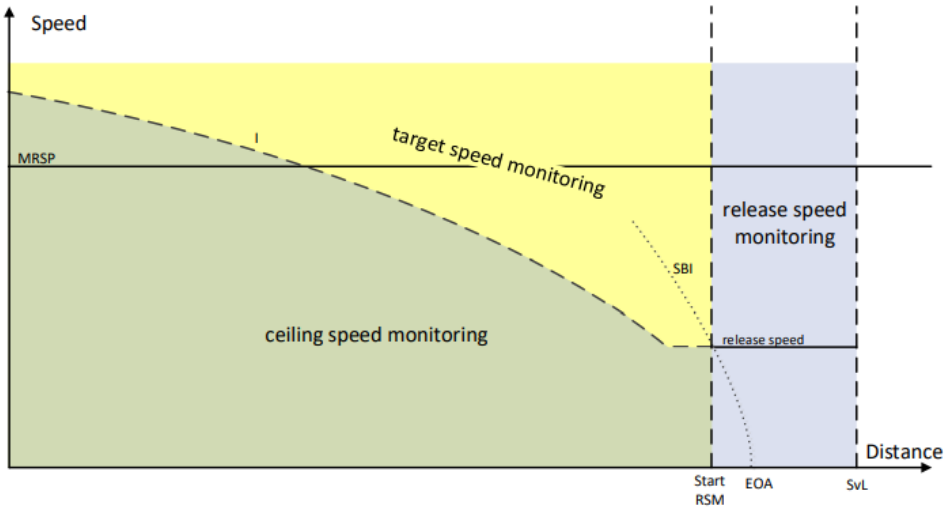

Figure 1.11: Types of Speed Monitoring [8]

Chapter 2

Requirements

2.1 Functional requirements

FE1 - Handle received MA Movement Authority shall be handled according to the subsets version 3.6.0.

FE2 - Check correctness of messages All received messages shall be checked based on the SUBSET-036-8. If the message is not correct, the message shall be omitted.

FE3 - Check correctness of packets All received packets shall be checked based on the SUBSET-036-7. If the packet is not correct, the packet shall be omitted.

FE4 - Calculate MRSP based on SSP The Most Restrictive Speed Profile shall be calculated based on static speed profile. Speed of the train is limited by SSP, which is based on the combination of the exact train and given piece of track.

FE5 - Calculate MRSP based on ASP Axle Load speed restrictions are based on the axle load train category and the state of the track.

FE6 - Calculate MRSP based on TSR Temporary peed restrictions are based on the condition of the track ahead of the train.

FE7 - Calculate MRSP based on Signalling related speed restriction when evaluated as a speed limit The track is able to send some speed limits to the train, based on the signaling.

FE8 - Calculate MRSP based on Mode related speed restriction Each ETCS mode has its own speed limit for the train to follow.

FE9 - Calculate MRSP based on Train related speed restriction It is possible to define maximum train speed based on the configuration and state of the train.

FE10 - Calculate MRSP based on STM max speed When ETCS operates through STM mudule, it can be possible to calculate MRSP based on STM max speed.

FE11 - Calculate MRSP based on STM system speed When ETCS operates through STM mudule, it can be possible to calculate MRSP based on STM system speed.

FE12 - Calculate MRSP based on LX speed When the train passes a non protected Level Crossing, it can be possible to use LX speed restriction to calculate MRSP.

FE13 - Calculate MRSP based on Override related speed restriction The override speed restriction is taken into account, when the override function is active.

FE14 - Delete Revoked TSR It shall be possible to delete previously received TSR using the id in the TSR Revocation packet.

FE15 - Dynamic Speed Monitoring based on MRSP, MA, release speed, gradient, mode profile It is possible to monitor speed by comparing the train speed and position, using various supervision limits. In that case, the speed monitoring is based on the defined features. The according information is shown to the driver, when the speed monitoring is active.

FE16 - Dynamic Speed Monitoring based on MRSP This speed monitoring is the same as the previous one, only it is based only on MRSP.

FE17 - Dynamic Speed Monitoring based on MRSP, allowed distance to run in Staff Responsible mode This speed monitoring is the same as the previous one, only it is based only on MRSP and the distance allowed in a specific mode Staff Responsible.

FE18 - Dynamic Speed Monitoring based on Ceiling Speed Monitoring only (no braking curve) based on MRSP In that case the dynamic speed monitoring is based only on MRSP, but the braking curve is not calculated, and the supervision status is only CSM.

2.2 Non-Functional requirements

N1 - C++ language - modules shall be implemented in C++ programming language, because all of the existing ETCS simulator modules are implemented in C++ language.

N2 - Subset Version - modules shall correspond to the Subsets version 3.6.0.

Chapter 3

Design

One module is created for this work. This module contains three other sub-modules inside. The whole module communicates with the existing EVC module via MQTT broker. Messages are sent using JSON structures. These structures will be represented further in this work.

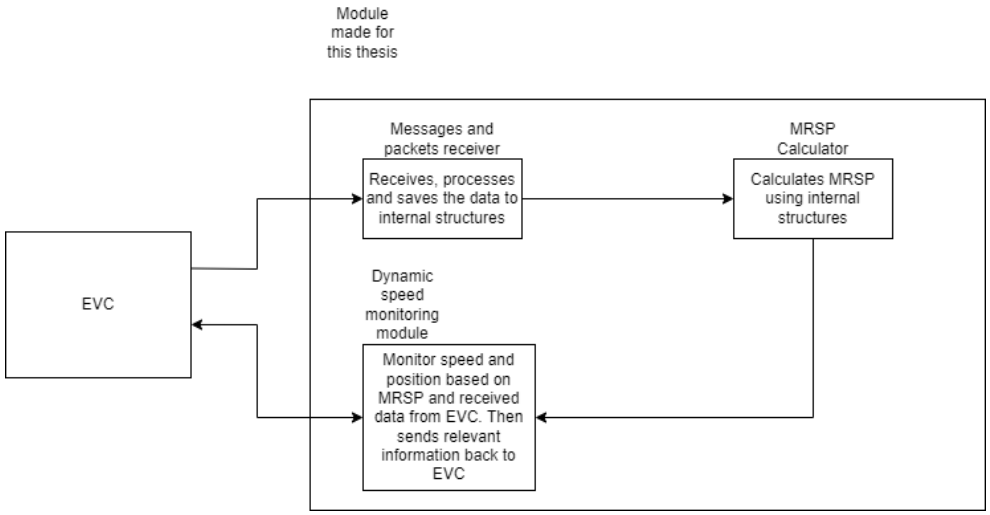

Figure 3.1: Modules

The main module first receives train data and EVC data. Train data consists of essential information about the train. Then train data is saved to the internal structure, which represents train category, train deficiency category, length of the train, train maximum speed, loading gauge, axle load category and axle load number. EVC data consists of the current level and mode, in which EVC operates right now. This data also is saved to the internal structure. After that the MA can be received and dynamic speed monitoring

enables.

When the new MA is received, previously received data from the MA is deleted. This feature corresponds to one of the EVC functions.

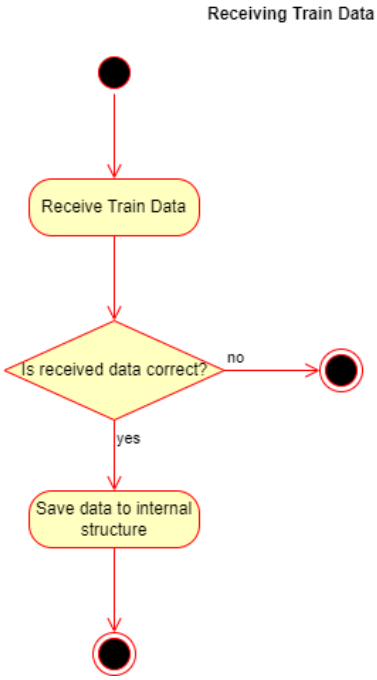

Figure 3.2: Receiving of Train Data

3.1 Messages and packets receiver module

This module receives messages and packets described earlier in this work. Then processes the information and evaluates it. If the evaluation fails, then the error will be thrown and the process will be stopped. If the evaluation succeeds, then the information inside packets and messages will be saved to the internal structures, corresponding to the profiles inside.

The evaluation of the messages follows the diagram Receiving MA messages, which describes possibilities. But, before message Movement Authority will be received, Train Data and data from the EVC must be received, because it has essential information for MA evaluation. EVC must send current level and mode. When receiving MA message, the current mode and level will be checked and MA message will be processed accordingly. If Infill MA is received, then EVC must be running in the Level 1, so it can be processed by the module, and only in modes FS and LS. If the modes does not correspond to that, then Infill MA is rejected. MA messages are parted to messages for Level 1, 2 and 3, so current level of EVC must correspond to that. If all the

conditions are met, then messages and packets inside are processed and the data is saved to the internal structures.

Axle load profile can represent empty profile and in that case it will be saved to the structure as an empty profile and the distance to the beginning of the empty profile. If the profile is not empty, then the structure will consist of the distance to the change of the speed, the according speed and the length of the profile.

Static speed profile is saved as a structure of speed, distance and the identifier, which says if the front end of the train is going to be considered when evaluating MRSP.

Temporary speed restriction is saved as a structure of id of the TSR, distance to the change of the speed, the according speed and the length of the profile. The id is further used to identify the TSR to be deleted in certain situations.

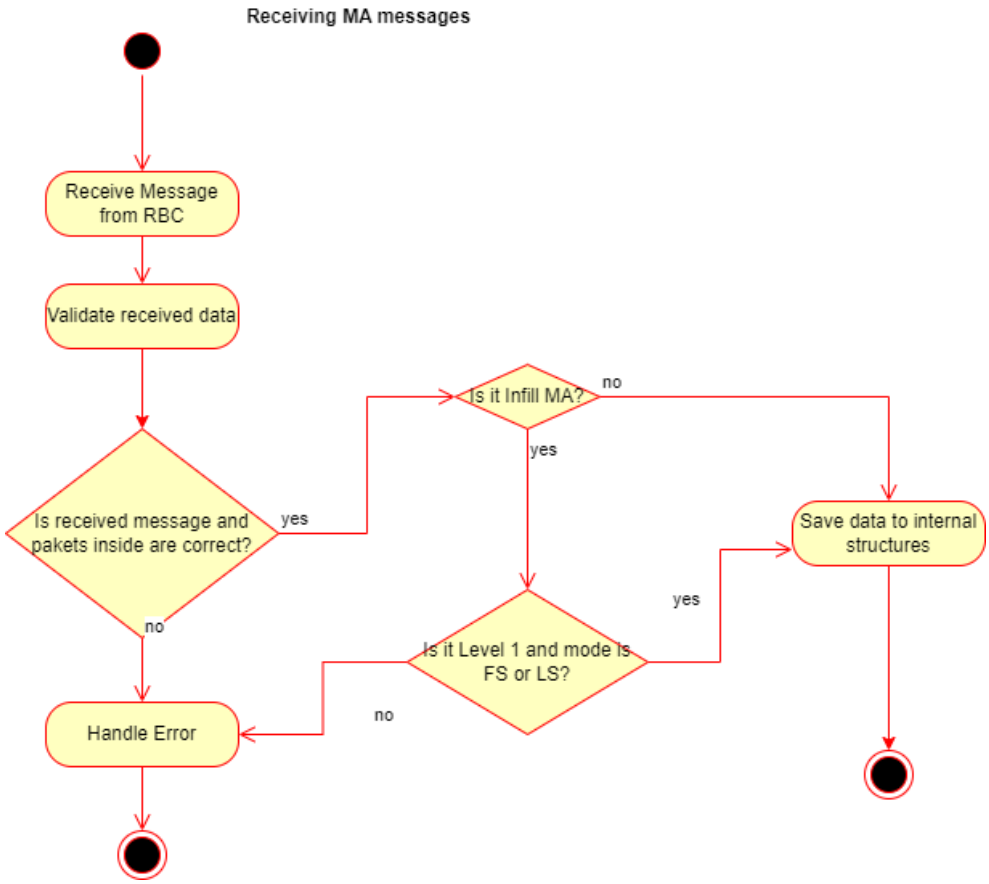

Figure 3.3: Receiving MA messages

Temporary speed restriction revocation allows to delete previously received TSR, using it's identity. If one of the previous TSR is deleted, then new MRSP will be calculated, so it can be updated due to the new rules.

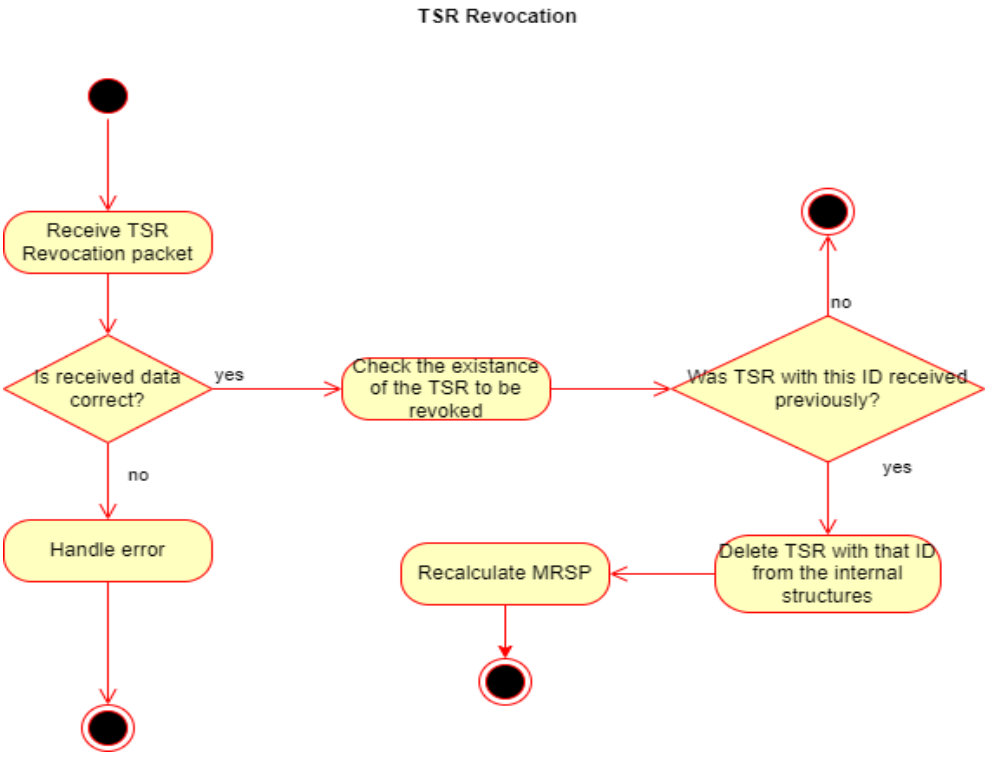

Figure 3.4: TSR Revocation

3.2 MRSP calculator module

To calculate MRSP, Train data and data from EVC must be received beforehand, because it is used in calculations inside the MRSP calculator submodule. Different speed limits are used for the calculations, which is shown on the diagram "Calculation of MRSP". Due to current modes and levels, different limits are used. For example, when the Level is 2 and mode is FS, ASP, SSP and TSR speed limits and train speed restrictions are used. To calculate MRSP, ASP, SSP and TSR limits are saved to the first container in ascending order due to the distance from LRBG, while all other restrictions are saved to the second container. Then calculation will take place, using both those containers.

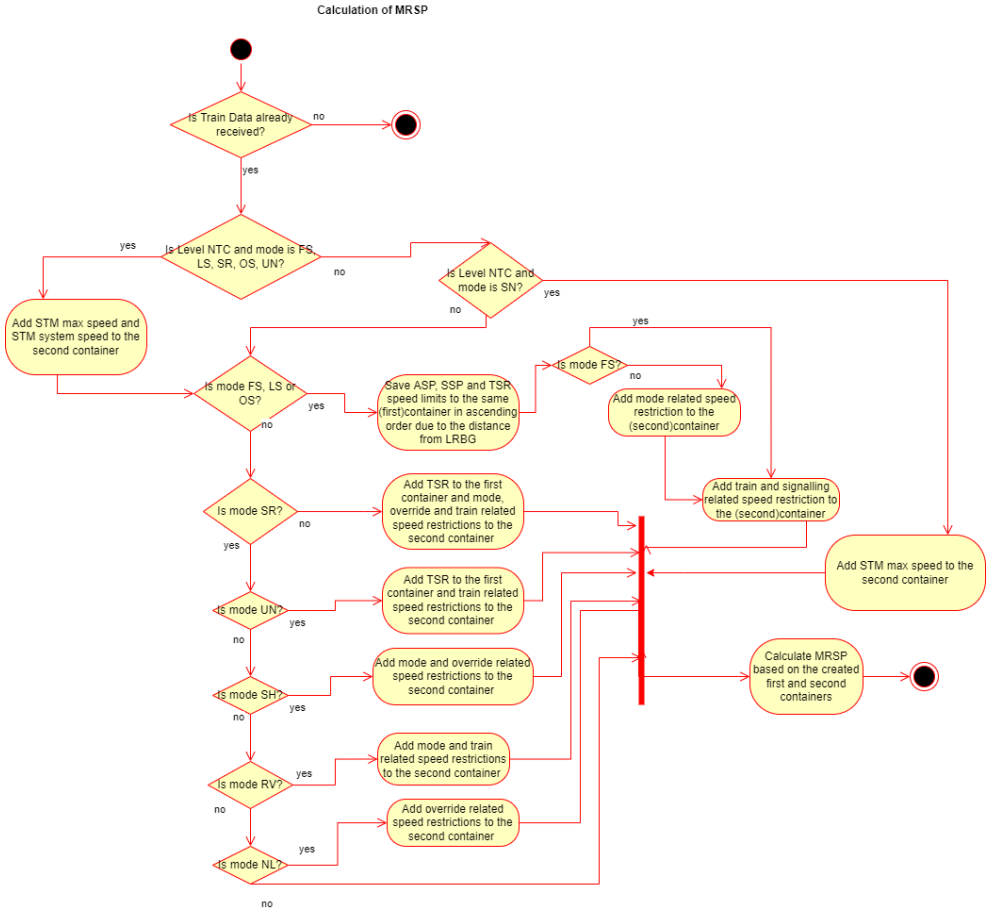

Figure 3.5: Calculation of MRSP

3.2.1 Algorithm for MRSP calculation

As was already discussed earlier in this chapter, MRSP is calculated after receiving and saving data to the internal structures. After doing that, speed profiles are just kept with their own kinds of profiles. It can be perfectly seen on the diagram ”Speed Profiles”, where same kinds of speed limits are coloured with the same color. For example, static speed restrictions, which are coloured red, are kept in one container within MRSP calculator class. Before evaluating MRSP, all of those speed profiles are saved to the same container, which is represented on the diagram ”MRSP algorithm”. There can be seen all of the previous speed profiles, but combined to the one line. This line represents distance of speed limits from the LRBG. As can be seen, speed limits are saved in ascending order, where each dot represents either start of the speed limit, end of the speed limit or both start and end of speed limit.

Start and end of speed limit can be placed on the same distance, because one speed limit can start at distance 300, while the other speed limit can end at the same distance.

The whole algorithm of the calculation of MRSP works like that:

1. Takes a look at the first dot.
2. Saves all ”starting” at that dot speeds to the separate container.
3. Deletes all ”ending” speeds from the container.
4. Takes look at the next dot.
5. Takes the distance between the first and next dot and minimum speed between them. And saves that to the output container.
6. Next dot becomes first dot.
7. Repeat, until all of the dots are processed.

The output container is the result of the algorithm. And this output container consists of the distance to the change of the speed and according speed. As a result, MRSP is calculated and is shown on the diagram ”MRSP calculated”. MRSP is next used in dynamic speed monitoring.

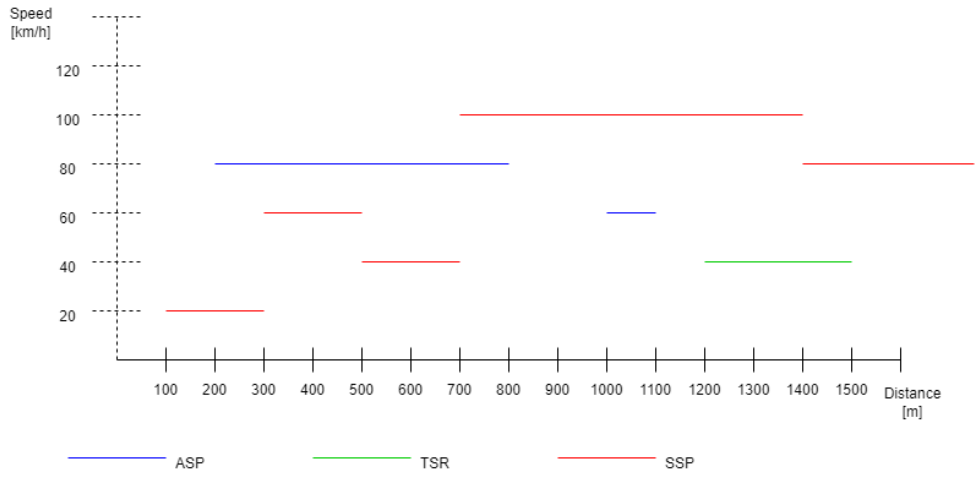

Figure 3.6: Speed Profiles

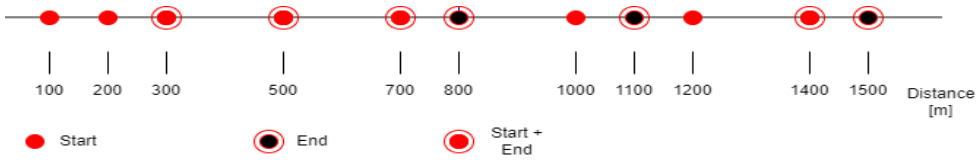

Figure 3.7: MRSP algorithm

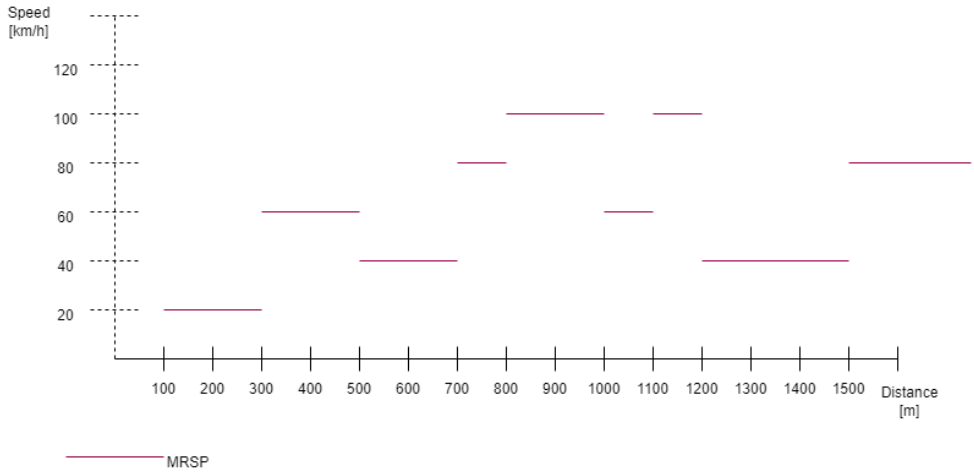

Figure 3.8: MRSP calculated

3.3 Dynamic speed monitoring module

Dynamic speed monitoring module operates on the received data like Train data, EVC data and MA. If this data is not received yet, then the error is thrown. Dynamic speed monitoring operates on modes SH, SV, RV, UN, SR, FS, LS and OS. Different kinds of speed monitoring applies to different modes. The main goal of this module is to monitor speed, using distance of the train, to ensure safe movement. It means, that this module will check the train speed and, if necessary, will send service or emergency brake command to EVC. Or, contrary, will release those brake commands, if the train speed comes below the required speed limit.

3.4 JSON

Format JSON was chosen for transferring data, because it is easy to use and read. Second, most programming languages can easily work with JSON, so generating and parsing data will not be a problem.

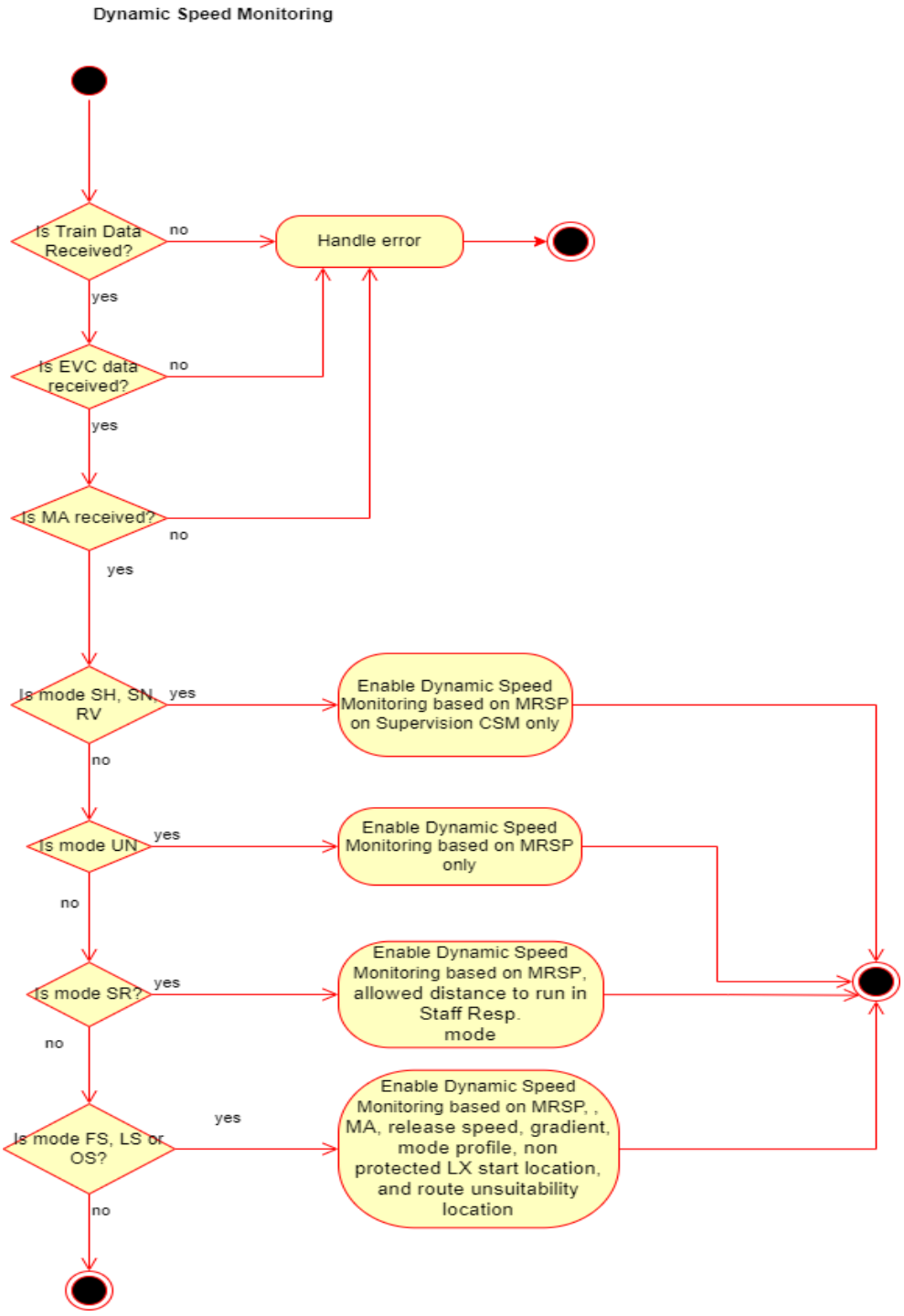

Figure 3.9: Dynamic Speed Monitoring

Chapter 4

Implementation

All modules for this work were implemented in C++ language. The whole implementation corresponds to the subsets version 3.6.0.

4.1 Main Module

For this thesis one main module and three submodules were implemented. The main module operates on other submodules. It has instances of the three submodules and receives necessary data, that is redirected to the necessary submodule.

Method `receiveTrainData` gets JSON object, which has all necessary details about the train, that then will be taken into account, when processing movement authorities and speed monitoring. So this data must be received first along with data from EVC. This JSON object is then passed to the message and packets receiver module, which processes all information and saves necessary data into internal structures.

When the movement authority is received, the main module redirects JSON object to the messages and packets receiver, that then processes the data. It saves necessary data and, as an output, has processed speed profiles inside, that then will be used for MRSP calculations. Inside method `receiveMA`, after movement authority is processed, the main module gives necessary information, as speed profiles and train data, to the MRSP calculator module, that calculates the data. The output of the calculations is the most restrictive speed profile, that then is passed to the dynamic speed monitoring module.

Method `receiveODO` gets the data from the odometer of the train and passes the data to the dynamic speed monitoring module, which processes necessary data. As for the output, it has all the necessary information, that then will be send to EVC. This information can be seen on the listing "Data to send to EVC".

```
struct StructureToSendEVC {
    std::uint8_t supervision_section;
    std::uint8_t supervision_status;
    std::uint16_t pre_indication_location;
    std::uint16_t indication_location;
    bool service_brake = false;
    bool emergency_brake = false;
};
```

Listing 4.1: Data to send to EVC

The main module is more or less acts like a router, that receives and redirects information between different submodules and sends the output information to EVC.

4.2 Messages and packets receiver

Messages and packets receiver receives message Movement Authority and passes packets as arguments to according methods, like `receiveTSR`, that can be seen on the listing "Method Receive TSR". As an output of this module, it saves all speed profiles to according containers, that later will be used for calculating the most restrictive speed profile. Mainly, this modules takes packets, searches for speed and distance inside those packets, and saves these data to the containers.

Message Movement Authority can be received as a JSON object. It has one of the packets with movement authority inside and additional packets with speed profiles inside them. `receiveSSP` method, for example, checks the category of the train and the given categories inside the packet, compares them, and then saves the according speed and distance to the internal container `std::vector<SSP>`. For all of the speed profiles, the following procedure is taken place. So that all of the received speed profiles are processed and are ready to be used later, when calculating the most restrictive speed profile of them all.

On the listing "Method Receive TSR" can be seen the processing of the packet and saving necessary information about speed and distance to the according structure. Temporary speed restrictions are given for the all train types and categories, so the process is simple enough, but when the axle load speed profiles or static speed profiles are given, it is important to compare categories and their speed limits, to choose the right speed restriction for the given piece of track. All of the resolutions and formulas for that are given by the subsets and were described in earlier chapters. As for the formulas, that are necessary for calculating the length of the speed restrictions, as well, as their speed at that areas.

```
void MessagesAndPacketsReceiver::ReceiveTSR(TemporarySR * tem){
//Gets the distance to the TSR with according scale measures.
double x = Scale(tem->getQScale(), tem->getDTsr());
//Gets the length of the TSR with according scale measures.
double y = Scale(tem->getQScale(), tem->getLTsr());
//Checks if the train length must be taken into account when
//  calculating the length of the restriction.
if(tem->getQFront() == 0)
    y += trainData.length;
//Formula for receiving speed: received_speed * 5.
//  According to subset-7.
std::uint16_t speed = tem->getVTsr() * 5;
TSR tmp{tem->getNidTsr(), x, y, speed};
tsr.emplace(tem->getNidTsr(), tmp);
}
```

Listing 4.2: Method Receive TSR

4.3 MRSP calculator

MRSP calculator module calculates the most restrictive speed profile. The method `MRSPCalculator` takes as an arguments the axle load speed profile, all temporary speed restrictions, the static speed profile and train data.

This algorithm was described earlier in the chapter ”Design”. Basically, it saves all of the speed profiles to the same `std::map`, which represents the line of the distance from LRBG and has different speeds on it. It also has the information inside, if the speed profile starts or ends at that exact distance. This container is called `speed_profiles`, so the saving process can be seen on the listings ”Saving speed profiles to the Speed Profiles container” and ”Implementation of the ToSpeedProfiles method”. It represents the distance line and dots on it, that tell the speed of profiles and whether or not it starts there or ends. This was shown earlier in the ”Design” chapter.

Method `ToSpeedProfiles` takes distance of the speed profile, it’s speed and tag, that represents if the profile starts here or ends. Values: true - it starts here, false - it ends here. For the speed profiles, there is always defined the starting and ending points, unless it is the last value of the static speed profile, because it can have only starting point and be valid up to the end of the movement authority or beyond.

After all speed profiles are processed, then it goes to the calculating of the most restrictive speed profile based on that container and other train data. It was also shown at the ”Design” chapter earlier. This can be seen at the picture ”MRSP calculation”. It iterates through the line of distances, which is represented by the container `speed_profiles`. It looks at each distance from the beginning, then saves all of the ”starting point” speeds to the

```
double dist = 0;
std::uint16_t speed = 0;
//Saves speed profiles, their starting and ending distance,
//  and their speed to the common container.
for(auto &i: ssp){
    dist += i.distance;
    ToSpeedProfiles(dist, i.speed, true);
    ToSpeedProfiles(dist, speed, false);
    speed = i.speed;
}

for(auto &i: tsr){
    ToSpeedProfiles(i.second.distance, i.second.speed, true);
    ToSpeedProfiles(i.second.distance + i.second.length,
                    i.second.speed, false);
}

for(auto &i: asp){
    ToSpeedProfiles(i.distance, i.speed, true);
    ToSpeedProfiles(i.distance + i.length, i.speed, false);
}
```

Listing 4.3: Saving speed profiles to the Speed Profiles container

```
void MRSPCalculator::ToSpeedProfiles(double distance,
    std::uint16_t speed, bool tag)
{
    auto it = speed_profiles.find(distance);
    if(it != speed_profiles.end()){
        it->second.emplace(speed, tag);
    } else {
        std::multimap<uint16_t, bool> tmp;
        tmp.emplace(speed, tag);
        speed_profiles.emplace(distance, tmp);
    }
}
```

Listing 4.4: Implementation of the ToSpeedProfiles method

`std::multiset speeds`. Then it looks to the next distance and searches for the ”ending point” speeds. Then, for the section between these two distances, the minimum speed is taken as a MRSP value, so it is saved to the output container `calculated_mrsp`. To avoid speed repetition, before saving data to the output container, it looks at the previous value and compares speed

values. So if they are the same, then the restriction just prolongs to that distance.

MRSP is represented as two different ways for the convenience. They both are represented as `std::vector<MRSP>`, where MRSP structure has speed and distance inside. The first container has all distances as a distance from LRBG to the change of speed, while the second container has each distance as an incremental distance from the previous one, so that only the first value represents the distance from LRBG. It is implemented like that, so it can be used later for different purposes. The second one is represented the same way as an international static speed profile, which was received earlier as a packet, that is why it was decided to make two different MRSP containers. It is possible just to convert MRSP incremental container to differential one and vise verse. So two methods `MRSPToIncremental` and `MRSPToDifferential` were implemented inside this module for that reason, so that MRSP can be easily converted to whichever type is needed.

```
struct MRSP{
    double distance;
    std::uint16_t speed;
};
```

Listing 4.5: Structure MRSP

4.4 DSM

Dynamic speed monitoring module operates on the calculated previously MRSP, which is set by the Main Module, EVC data and other necessary Train Data. After receiving new movement authority, data is updated and new braking curve is calculated. Module for the calculation of braking curves was taken from previous EVC implementation.[4] After braking curve is calculated, the module can operate and receive data from the odometer of the train. This module was more or less implemented during the course BI-SP2. It then was changed to better suite the purpose of this bachelor's thesis, so that it can function according to the given task.

The main functions start, when the module starts to receive odometer data. This module controls the speed and the distance of the train. It takes the speed of the train, calculates travelled distance and, based on that, it supervises speed and location of the train. It looks at the speed and distance limits and, if train exceeds those limits, it triggers train brakes. This information is also helps to change supervision sections and statuses, so that is also sent to EVC later. Those supervision parameters are important for the correct work of the EVC, so that it can show them to the driver and trigger necessary functions, so that the train can move according to them.

```
void MRSPCalculator::CalculateMRSP(const std::vector<SSP>& ssp,
const std::map<std::uint8_t, TSR>& tsr, const std::vector<ASP>& asp,
TrainSR trainSr, std::uint16_t signallingSr) {
    //...The code is reduced for the better view purposes.
    // The reduced part was shown earlier on the listing:
    // "Saving speed profiles to the Speed Profiles container"
    std::multiset<std::uint16_t> speeds;
    for(auto itr = speed_profiles.begin(); itr != speed_profiles.end();)
    {
        MRSP mrsp;
        auto itr_first = itr;
        for(auto &i: itr->second) {
            if(i.second)
                speeds.emplace(i.first);
        }
        itr++;
        if(itr == speed_profiles.end()) {
            if(!speeds.empty()) {
                itr--;
                mrsp.distance = itr->first;
                mrsp.speed = *speeds.begin();
                auto m = calculated_mrsp_differential.end() - 1;
                if(m->speed != mrsp.speed)
                    calculated_mrsp_differential.emplace_back(mrsp);
            }
            break;
        }
        mrsp.speed = *speeds.begin();
        for(auto &i: itr->second){
            if(!i.second)
                speeds.erase(speeds.find(i.first));
        }
        mrsp.distance = itr_first->first;
        if(mrsp.speed > signallingSr && signallingSr != 0)
            mrsp.speed = signallingSr;
        if(mrsp.speed > trainSr.speed)
            mrsp.speed = trainSr.speed;
        auto m = calculated_mrsp_differential.end() - 1;
        if(calculated_mrsp_differential.empty() || m->speed != mrsp.speed)
            calculated_mrsp_differential.emplace_back(mrsp);
    }
    calculated_mrsp_incremental =
        MRSPToIncremental(calculated_mrsp_differential);
}
```

Listing 4.6: MRSP calculation

Chapter 5

Testing

Testing is one of the most important parts of the implementation. It helps to ensure correct work of the program and prevent any unwanted behaviour. Undetected earlier on errors can have fatal influence further on. The best practice is to test code during the implementation, because it will help to find and fix errors sooner.

Automated testing helps to reduce time by providing tests, that can be applied multiple times. Therefore, when new functionalities are added, those automated tests can be run, to ensure, that those new functionalities did not break the older ones. This kind of testing is usually good for small, easy tests like unit tests, because writing automated tests for a big, complicated application can be very time consuming and costly.

Unit tests are used for testing some small pieces of the code. Those pieces can be isolated from the whole system and tested separately from the other parts of the code. It is useful for checking the correctness of functions inside an application. Sometimes some small tests can be written beforehand, so that the end result can be understood better.

During this work, unit tests were written. Those tests are fully automated and isolate different modules, so it can be possible to test them separately.

5.1 Messages and Packets Receiver testing

Unit tests for messages and packets receiver module were created. Those tests create an instance of the receiver class and, using JSON structures, sends them to receiver. Receiver gets this JSON object and processes data inside, then saving necessary data to internal structures. The picture "JSON Train Data" shows relevant train data, that can be received by the messages and packets receiver. "JSON TSR" shows the example of the speed profile, there temporary speed profile, that is received and processed. "JSON MA" shows the example of the movement authority with one temporary speed restriction and one temporary speed restriction revocation. Similar JSON objects were

created for testing messages and packets receiver, along with the incorrect data to test the work of the receiver.

```
{
    "NC_CDTRAIN" : 7,
    "NC_TRAIN" : 15,
    "L_TRAIN" : 200,
    "V_MAXTRAIN" : 25,
    "M_LOADINGGAUGE" : 16,
    "M_AXLELOADCAT" : 3,
    "N_AXLE" : 20
}
```

Listing 5.1: JSON Train Data

```
{
    "NID_PACKET": 65,
    "Q_DIR": 1,
    "L_PACKET": 71,
    "Q_SCALE": 1,
    "NID_TSR": 0,
    "D_TSR": 100,
    "L_TSR": 150,
    "Q_FRONT": 1,
    "V_TSR": 4
}
```

Listing 5.2: JSON TSR

To check the correctness of the saved data, different tests were implemented. For example, for each individual speed profile receiver, tests were created during the implementation, without using the JSON objects. The example of the test can be seen on the listing "Receive SSP test". Then tests, that use JSON objects, were implemented to check the correctness of the parsing. Some of the results of the tests with the same values were compared with each other for that purpose. One using the JSON receiver, and other using ordinary constructor. The test example with JSON object can be seen on the listing "Receive ASP test". These tests check the correctness for the requirements FE1 - FE3 and FE14.

It is understandable, that when testing new implemented code, the errors are almost always found. During the testing of this modules, there were few details, that needed to be changed. Those errors were primary connected with the processing international static speed profiles and axle load speed profiles. It was because the receivers were saving data to internal structures, but were

```
{
  "NID_MESSAGE": 3,
  "L_MESSAGE": 123,
  "T_TRAIN": 0,
  "M_ACK": 0,
  "NID_LRBG": 0,
  "PACKETS": [
    {
      "NID_PACKET": 65,
      "Q_DIR": 1,
      "L_PACKET": 71,
      "Q_SCALE": 1,
      "NID_TSR": 3,
      "D_TSR": 1300,
      "L_TSR": 500,
      "Q_FRONT": 0,
      "V_TSR": 6
    },
    {
      "NID_PACKET": 66,
      "Q_DIR": 1,
      "L_PACKET": 31,
      "NID_TSR": 0
    }
  ]
}
```


Listing 5.3: JSON MA

using wrong train categories for evaluating speed restrictions. So, that was fixed, and then tests were run again, and new tests for that were created. As for now, those tests run correctly, and speed restrictions due to specific speed category are saved.

5.2 MRSP calculator testing

Unit tests for MRSP calculator module were also created. Those tests created an instance of the class calculator, then were given different speed profiles for processing, and then the results of the calculations were compared to the estimated results. The estimated results were the same as the calculated results. Example of the created tests are shown on the picture "MRSP Calculator test". These tests check the correctness for the requirements FE4 - FE13.

Tests for this module helped to detect some errors during the implementa-

```
void testReceiveSSP(){
    MessagesAndPacketsReceiver odo;
    std::vector<std::uint8_t> tmp1{0, 1};
    std::vector<std::uint8_t> tmp2{0};
    std::vector<std::uint8_t> tmp3{1};
    std::vector<std::uint8_t> tmp4{5, 6};
    std::vector<std::uint16_t> tmp5{100, 200};
    std::vector<std::uint8_t> tmp6{7, 8};
    std::vector<std::uint8_t> tmp7{1, 1};
    std::vector<std::uint8_t> tmp8{1, 1};
    std::vector<std::vector<std::uint8_t>> tmpt1{{0}, {1}};
    std::vector<std::vector<std::uint8_t>> tmpt2{{0}};
    std::vector<std::vector<std::uint8_t>> tmpt3{{1}};
    std::vector<std::vector<std::uint8_t>> tmpt4{{5}, {6}};
    InternationalSSP ssp(27, 1, 177, 1, 100, 7, 1, 2, tmp1,
        tmp2, tmp3, tmp4, 2, tmp5, tmp6, tmp7, tmp8, tmpt1,
        tmpt2, tmpt3, tmpt4);

    ifstream ifs("../Tests/TrainData1.json");
    nlohmann::json j = nlohmann::json::parse(ifs);
    odo.ReceiveTrainData(j);
    odo.ReceiveSSP(&ssp);
    assert(odo.ssp.size() == 3);
    // First speed restriction
    assert(odo.ssp[0].distance == 100);
    assert(odo.ssp[0].front_end == false);
    assert(odo.ssp[0].speed == 25);
    // Second speed restriction
    assert(odo.ssp[1].distance == 100);
    assert(odo.ssp[1].front_end == false);
    assert(odo.ssp[1].speed == 25);
    // Third speed restriction
    assert(odo.ssp[2].distance == 200);
    assert(odo.ssp[2].front_end == false);
    assert(odo.ssp[2].speed == 40);
}
```

Listing 5.4: Receive SSP test

tion. At first, the calculation of the MRSP was created in different way, so the tests were created to help to see, if the module works as it supposed to. Then the implementation of the module changed and those tests were run to check the new implementation. It was almost right, but one error was detected, that

```
void testReceiveASP3(){
    MessagesAndPacketsReceiver odo;
    ifstream ifs("../Tests/ASP2.json");
    nlohmann::json j = nlohmann::json::parse(ifs);
    AxleLoadSP as(j);
    odo.trainData.axle_category = 4;
    odo.trainData.length = 100;
    odo.ReceiveASP(&as);
    assert(odo.asp.size() == 3);
    // First speed restriction
    assert(odo.asp[0].distance == 700);
    assert(odo.asp[0].length == 200);
    assert(odo.asp[0].speed == 20);
    // Second speed restriction
    assert(odo.asp[1].distance == 1000);
    assert(odo.asp[1].length == 150);
    assert(odo.asp[1].speed == 25);
    // Third speed restriction
    assert(odo.asp[2].distance == 1500);
    assert(odo.asp[2].length == 350);
    assert(odo.asp[2].speed == 15);

}
```

Listing 5.5: Receive ASP test

was about the last value of the most restrictive speed profile. At first, that last value was not saved to the calculated MRSP container, but it was fixed later on.

5.3 DSM testing

Dynamic speed monitoring module was tested, using different types of MRSP and MA. The results of the tests were the same as the estimated results. This means, that the brakes commands where triggered at the right time, when the current speed of the train was above the permitted one. These tests check the correctness for the requirements FE15 - FE18.

5.4 Main Module testing

The main module was tested, using data from the previous tests, so that can be seen, if the Main Module correctly receives and redirects information. Later on, big tests were added to check the correctness of all modules, because it

```
void testMRSP(){
    MRSPCalculator odo;
    SSP ssp{0, 50, false};
    SSP ssp1{200, 200, false};
    SSP ssp2{250, 100, false};
    SSP ssp3{150, 120, false};
    odo.ssp.emplace_back(ssp);
    odo.ssp.emplace_back(ssp1);
    odo.ssp.emplace_back(ssp2);
    odo.ssp.emplace_back(ssp3);
    ASP asp{false, 0, 550, 100, 140};
    odo.asp.emplace_back(asp);
    TSR tsr{1, 300, 200, 40};
    odo.tsr.emplace(1, tsr);
    odo.trainSr.speed = 130;
    odo.CalculateMRSP();
    vector<MRSP> tmp {{0, 50}, {200, 130},
        {300, 40}, {500, 100}, {600, 120}};
    assert(tmp.size() == odo.mrsp.size());
    for(size_t i = 0; i < tmp.size(); i++){
        assert(tmp[i].speed == odo.mrsp[i].speed);
        assert(tmp[i].distance == odo.mrsp[i].distance);
    }
};
```

Listing 5.6: MRSP Calculator test

was easier to take information as JSON objects and redirects that information to different modules. So the main tests for the MRSPCalculator module and DSM module were created inside the main module tests. It was convenient to do so, when the previous small tests were shown to be correct. The short examples of the tests are shown on the listings ”Main Module test 1” and ”Main Module test 2”. These tests check the correctness for the presented earlier requirements.

```
void testReceiveMA(){
    MainModule odo;
    ifstream ifs11("../Tests/TrainData1.json");
    nlohmann::json j1 = nlohmann::json::parse(ifs11);
    odo.receiveTrainData(j1);

    ifstream ifs("../Tests/EVCData.json");
    nlohmann::json j = nlohmann::json::parse(ifs);
    odo.receiveEVCData(j);

    ifstream ifs2("../Tests/MA1.json");
    nlohmann::json j2 = nlohmann::json::parse(ifs2);
    odo.receiveMA(j2);
    //MRSP tests
    assert(odo.mrsp_calc.getMRSP().size() == 10);
    assert(odo.mrsp_calc.getMRSP()[0].distance == 100
            && odo.mrsp_calc.getMRSP()[0].speed == 25);
    assert(odo.mrsp_calc.getMRSP()[1].distance == 500
            && odo.mrsp_calc.getMRSP()[1].speed == 40);
    assert(odo.mrsp_calc.getMRSP()[2].distance == 700
            && odo.mrsp_calc.getMRSP()[2].speed == 20);
    assert(odo.mrsp_calc.getMRSP()[3].distance == 900
            && odo.mrsp_calc.getMRSP()[3].speed == 40);
    assert(odo.mrsp_calc.getMRSP()[4].distance == 1000
            && odo.mrsp_calc.getMRSP()[4].speed == 15);
    assert(odo.mrsp_calc.getMRSP()[5].distance == 1200
            && odo.mrsp_calc.getMRSP()[5].speed == 40);
    assert(odo.mrsp_calc.getMRSP()[6].distance == 1300
            && odo.mrsp_calc.getMRSP()[6].speed == 30);
    assert(odo.mrsp_calc.getMRSP()[7].distance == 1500
            && odo.mrsp_calc.getMRSP()[7].speed == 15);
    assert(odo.mrsp_calc.getMRSP()[8].distance == 1950
            && odo.mrsp_calc.getMRSP()[8].speed == 30);
    assert(odo.mrsp_calc.getMRSP()[9].distance == 2000
            && odo.mrsp_calc.getMRSP()[9].speed == 40);
}
```

Listing 5.7: Main Module test

```
void testReceiveODO1(){
    MainModule odo;
    ifstream ifs11("../Tests/TrainData1.json");
    nlohmann::json j1 = nlohmann::json::parse(ifs11);
    odo.receiveTrainData(j1);

    ifstream ifs("../Tests/EVCData.json");
    nlohmann::json j = nlohmann::json::parse(ifs);
    odo.receiveEVCData(j);

    ifstream ifs2("../Tests/MAL22.json");
    nlohmann::json j2 = nlohmann::json::parse(ifs2);
    odo.receiveMA(j2);

    ifstream ifs3("../Tests/ODO1.json");
    nlohmann::json j3 = nlohmann::json::parse(ifs3);
    odo.receiveODO(j3);

    ifstream ifs4("../Tests/ODO2.json");
    nlohmann::json j4 = nlohmann::json::parse(ifs4);
    odo.receiveODO(j4);

    ifstream ifs5("../Tests/ODO3.json");
    nlohmann::json j5 = nlohmann::json::parse(ifs5);
    odo.receiveODO(j5);

    //...
    //There are tests for the correctness of the speed profiles
    //  and MRSP but are not shown, because they are too long.

    // Tests for the DSM module
    // It is the data to be sent to EVC
    // Service brake is triggered here, because the train speed
    //  now is above the permitted train speed.
    assert(odo.getEvcSend().service_brake == true);
    assert(odo.getEvcSend().emergency_brake == false);
    assert(odo.getEvcSend().supervision_section == 0);
    assert(odo.getEvcSend().supervision_status == 0);
}
```

Listing 5.8: Main Module test 2

Conclusion

The main goal of this thesis was to create a functional dynamic speed monitoring module for EVC. This goal was achieved. Three different submodules were implemented. One submodule allows to receive messages and packets from EVC, then processes the information and saves the necessary data. The next submodule calculates MRSP, using the data, that was received previously. The last submodule is called Dynamic speed monitoring and it allows to monitor speed and distance of the train, using previously calculated MRSP and other necessary data, received from EVC. As a result, these module can trigger emergency and service brakes, when necessary, and can release them, so that the train can move safely on the selected route.

First, the analysis of the ETCS system was given, along with the analysis of the existing ETCS simulator. Then were described essential parts of the ETCS system, that later were used for the design and implementation, as it specified the rules and characteristics of the system.

Second, the design part was created and described for this thesis. It showed different modules and communication between them. It also showed the diagrams of each part of the modules and how they must work.

Third, the implementation details were described and shown in the Chapter "Implementation". It describes each implemented module and shows a few implemented parts of the code.

At last, testing details were described in the chapter "Testing". It shows the examples of the tests, that were used during the implementation, as well, as JSON structures, that were passed to the modules for the processing. The results of the tests and errors, that were found during the implementation, were also indicated inside said chapter.

Future Development

Although features described in section "Requirements" were implemented, some new features can be added in the future. For example, version com-

patibility can be added, so that the module will work with different other modules, that are implemented with different subset versions. Current implementation supports only subsets version 3.6.0. There can be added few adjustments, so it will work with versions 2.4.0 and 3.4.0. Those adjustments primary concerns Messages and Packets Receiver module, that later on can detect the version of the EVC and, therefore, can redirect to differently implemented speed profiles receivers. They can be implemented for different versions inside the module.

Bibliography

[1] KADLČEK, David; STEJSKAL, Jan; JAHODA, Petr; UDAVICHENKA, Yury; MACHÁČEK, Jiří; VEJVODA, Štěpán. *Analýza projektu DMI displej pro simulátor ETCS*. 2021. Tech. rep. Faculty of Information Technology, CTU in Prague. [cit. 2022-04-20].

[2] KRASNENKOVA, Alina; ROSHCHUPKINA, Daria. *ETCS simulátor – EVC, Dokumentace*. 2021. Tech. rep. Faculty of Information Technology, CTU in Prague. [cit. 2022-04-20].

[3] SKIPALA, Michal; BENK, Patrik; GORGOL, Matěj; KRASNENKOVA, Alina; ROSHCHUPKINA, Daria; STERNWALD, Jiří. *ETCS simulátor – RBC, Dokumentace*. 2021. Tech. rep. Faculty of Information Technology, CTU in Prague. [cit. 2022-04-20].

[4] GOLMGREN, Nikita; BÍLEK, Matouš; KRAVTSOV, Aleksei; LANCA, Matěj; SLANINOVÁ, Dominika; UMPRECHT, Jan; WICHTERLE, David. *Výpočet brzdné křivky pro ETCS simulátor, Analytická dokumentace*. 2021. Tech. rep. Faculty of Information Technology, CTU in Prague. [cit. 2022-04-20]

[5] GOLMGREN, Nikita; BÍLEK, Matouš; KRAVTSOV, Aleksei; LANCA, Matěj; SLANINOVÁ, Dominika; UMPRECHT, Jan; WICHTERLE, David. *JRU a JRU DL Tool pro ETCS simulátor, Analytická dokumentace*. 2021. Tech. rep. Faculty of Information Technology, CTU in Prague. [cit. 2022-04-20].

[6] *System Requirements Specification, Chapter 1, ERTMS/ETCS language*. 2016. Tech. rep. European Union Agency For Railways. Available also from: https://www.era.europa.eu/content/set-specifications3-etcs-b3-r2-gsm-r-b1_en. [cit. 2022-04-20].

[7] *System Requirements Specification, Chapter 2, ERTMS/ETCS language*. 2016. Tech. rep. European Union Agency For Railways. Available also from: https://www.era.europa.eu/content/set-specifications3-etcs-b3-r2-gsm-r-b1_en. [cit. 2022-04-20].

[8] *System Requirements Specification, Chapter 3, ERTMS/ETCS language*. 2016. Tech. rep. European Union Agency For Railways. Available also from: https://www.era.europa.eu/content/set-specifications3-etcs-b3-r2-gsm-r-b1_en. [cit. 2022-04-20].

[9] *System Requirements Specification, Chapter 4, ERTMS/ETCS language*. 2016. Tech. rep. European Union Agency For Railways. Available also from: https://www.era.europa.eu/content/set-specifications3-etcs-b3-r2-gsm-r-b1_en. [cit. 2022-04-20].

[10] *System Requirements Specification, Chapter 7, ERTMS/ETCS language*. 2016. Tech. rep. European Union Agency For Railways. Available also from: https://www.era.europa.eu/content/set-specifications3-etcs-b3-r2-gsm-r-b1_en. [cit. 2022-04-20].

[11] *System Requirements Specification, Chapter 8, ERTMS/ETCS language*. 2016. Tech. rep. European Union Agency For Railways. Available also from: https://www.era.europa.eu/content/set-specifications3-etcs-b3-r2-gsm-r-b1_en. [cit. 2022-04-20].

[12] *System Requirements Specification, Chapter 5, ERTMS/ETCS language*. 2016. Tech. rep. European Union Agency For Railways. Available also from: https://www.era.europa.eu/content/set-specifications3-etcs-b3-r2-gsm-r-b1_en. [cit. 2022-04-20].

[13] *ERTMS in brief*. [N.d.]. Available also from: https://www.ertms.net/about-ertms/ertms-signaling-levels/. [cit. 2022-04-20].

[14] *European Rail Traffic Management System (ERTMS)*. [N.d.]. Available also from: https://www.era.europa.eu/activities/european-rail-traffic-management-system-ertms_en. [cit. 2022-04-20].

[15] UIC *ETCS Implementation Handbook*. 2008.

[16] *European Train Control System (ETCS)*. [N.d.]. Available also from: https://www.thalesgroup.com/en/european-train-control-system-etcs. [cit. 2022-04-20].

[17] *The ERTMS/ETCS Signalling System*. 2014. Available also from: https://www.railwaysignalling.eu/ertmsetcs-manual-free-handbook-download-free-pdf-format.

Appendix A

Acronyms

ASP Axle load speed profile

CSM Ceiling speed monitoring

DMI Driver machine interface

DSM Dynamic speed monitoring

EBI Emergency Brake Intervention

EOA End of authority

ERTMS European Rail Traffic Management System

ETCS European Train Control System

EVC European Vital Computer

FS Full Supervision

GSM-R Global System for Mobile Communication - Railway

I Indication

IndS Indication status

IntS Intervention status

IS Isolation

JRU Juridical Recording Unit

JSON JavaScript Object Notation

LRBG Last relevant balise group

LS Limited Supervision

LX Level crossing

MA Movement authority

MQTT Message queuing telemetry transport

MRSP Most restrictive speed profile

NL Non Leading

NoS Normal status

NP No Power

OS On Sight

OvS Over-speed status

P Permitted speed

PS Passive Shunting

PT Post Trip

RBC Radio Block Centre

RSM Release speed monitoring

RV Reversing

SB Stand By

SBI Service Brake Intervention

SF System Failure

SH Shunting

SL Sleeping

SN National System

SR Staff Responsible

SSP Static speed profile

STM Specific Transmission Module

TR Trip

TSM Target speed monitoring

TSR Temporary speed restriction

UN Unfitted

W Warning

WaS Warning status

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `impl` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format