CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Transportation Sciences
Dean's office
Konviktská 20, 110 00 Prague 1, Czech Republic

K620.....Department of Transport Telematics

MASTER'S THESIS ASSIGNMENT

(PROJECT, WORK OF ART)

Student's name and surname (including degrees):

Bc. Jan Červenka

Code of study programme code and study field of the student

N 3710 – IS – Intelligent Transport Systems

Theme title (in Czech): **Implementace displeje mobilní části ETCS**

Theme title (in English): Implementation of the DMI Display for the ETCS On-Board Sub-System

Guides for elaboration

During the elaboration of the master's thesis follow the outline below:

- Introduce the general European Train Control System (ETCS) features.
- Describe the main elements and functions of the On-Board ETCS Driver Machine Interface (DMI).
- Research possible software programming tools and frameworks suitable for the display implementation.
- Implement the DMI display using the selected tools.
- Document the implementation for the future use as a part of an ETCS simulator.

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Graphical work range: standard

Accompanying report length: At least 55 pages

Bibliography:
ETCS System Requirements Specifications (SUBSET-026) - http://www.era.europa.eu/Document-Register/Pages/Set-2-ETCS-Driver-Machine-Interface.aspx

ETCS Driver Machine Interface Specifications - http://www.era.europa.eu/

Master's thesis supervisor: **Ing. Zuzana Bělinová, Ph.D.**
doc. Ing. Martin Leso, Ph.D.
Dr. Cristina Olaverri Monreal

Date of master's thesis assignment: **July 31, 2015**
(date of the first assignment of this work, that has be minimum of 10 months before the deadline of the theses submission based on the standard duration of the study)

Date of master's thesis submission: **May 30, 2017**
a) date of first anticipated submission of the thesis based on the standard study duration and the recommended study time schedule
b) in case of postponing the submission of the thesis, next submission date results from the recommended time schedule

L. S.

doc. Ing. Pavel Hrubeš, Ph.D.
head of the Department of Transport Telematics

prof. Dr. Ing. Miroslav Svítek, dr. h. c.
dean of the faculty

I confirm assumption of master's thesis assignment.

Bc. Jan Červenka-
Student's name and signature

Prague .....November 30, 2016-

Czech Technical University in Prague
Faculty of Transportation Sciences
Department of Transport Telematics

Master's thesis

Implementation of the DMI Display for the ETCS On-Board Sub-System

Bc. Jan Červenka

Supervisor: doc. Ing. Martin Leso, Ph.D.

30th May 2017

Acknowledgements

I would like to thank doc. Ing. Martin Leso, Ph.D. for his professional guidance and advice.

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 school work under the provisions of Article 60(1) of the Act.

In Prague on 30th May 2017 .....

Czech Technical University in Prague
Faculty of Transportation Sciences

Citation of this thesis

Červenka, Jan. *Implementation of the DMI Display for the ETCS On-Board Sub-System.* Master's thesis. Czech Technical University in Prague, Faculty of Transportation Sciences, 2017.

Abstrakt

První část této diplomové práce slouží jako úvod do zabezpečovače ETCS a také popisuje funkce jeho DMI jednotky. Druhá část představuje hardware a software platformu, vysvětluje proces implementace DMI aplikace a slouží jako dokumentace API.

Klíčová slova Vozidlový simulátor, rozhraní člověk stroj, ERTMS, ETCS, DMI

Abstract

The first part of the thesis provides an overview of the ETCS and describes the functionality of the DMI unit. The second part introduces the hardware and software platform, covers the implementation process of the DMI application, and documents the API.

Keywords Vehicle simulator, human machine interface, ERTMS, ETCS, DMI

Contents

List of Figures

List of Tables

Introduction

This thesis is a part of an ongoing effort at the Transportation Laboratory of the Czech Technical University in Prague.

The mission of the Transportation Laboratory is to study railway signaling and interlocking. For this purpose, the laboratory maintains and operates a physical model of a railway network with various Czech national interlocking systems.

The current focus of the laboratory is the ETCS (European Train Control System). The idea is to integrate this system into the railway model and to build a new train cab simulator with the ETCS on-board sub-system.

The Staff of the Transportation Laboratory has decided to build the train simulator and develop the necessary software in-house instead of buying a complete solution from an external contractor. Besides lower cost, the main advantage is the ability to customize any part as needed and perform changes or updates in the future. Furthermore, building the simulator on our own provides us with a deeper understanding of the system.

As of May 2017, the project of the simulator is still at an early stage. Functional specifications have been finalized in a thesis written by Lukáš Petřík [1]. Additionally, some hardware, such as the driver's desk or on-board computers, have been acquired.

Integral part of the proposed train simulator is the ETCS driver machine interface. The goal of this thesis is to develop the DMI application for future use in the simulator.

Background

Vehicle simulators are popular across all modes of transport. From space flight to road traffic. They can be used either for training purposes or as a platform for experiments. The capability to easily design scenarios with desired conditions cannot be replicated in real life environment. Moreover,

the cost of running a simulation is usually much lower than using real life vehicles and equipment.

In the last several years, railway industry has been influenced by gradual deployment of the ETCS. This system introduces new challenges requiring further research. One of the area of interest is the driver's interaction with the system.

When a train is under ETCS supervision, the driver is required to regulate the speed in such a way that it stays below a computed braking curve. This type of operation is different from the current state which requires the driver to rely mostly on trackside speed limits and signals. Such new conditions lead to different driver behavior, which can be studied using simulators.

A simulator can also be very useful for evaluation of the DMI design. Scenarios can be made to test whether the drivers can interpret information from the DMI easily. Furthermore, access to the source code allows us to make changes to the features of the DMI and compare them with the original solution.

The Simulator

The specification [1] states that the simulator shall be implemented as a full cab. The cab will consist of a dashboard and a windscreen. The cab will be surrounded by a virtual reality projection with tracks, trains and other objects displayed.

The simulator is static. This means it will not use actuators to simulate effects of dynamic forces on the vehicle. However, it shall be built in such a way it will be possible to add this feature later.

The dashboard itself comes from a real vehicle and is compliant with the UIC document [2].

Scope of the Thesis

- The first chapter provides an overview of the ERTMS/ETCS signaling.
- The second chapter introduces the features and functions of the DMI.
- The third and fourth chapter researches and analyzes possible frameworks, operating systems or other tools that could be implemented.
- The fifth and sixth chapter describes the design, implementation, and testing of the DMI application.
- The seventh chapter documents the software so it can be integrated in the simulator.

Chapter 1

ERTMS/ETCS Overview

ERTMS (European Railway Traffic Management System) is a European initiative aiming to achieve signaling interoperability between various European railway systems. It is a part of a broader standardization process coordinated by the European Railway Agency (ERA) known as the TSI (technical specification for interoperability). TSI defines the following interoperability areas:

- Power
- Rolling stocks
- Infrastructure
- Comand and control

ERTMS refers to the entire command and control TSI program and includes the ETCS signaling and the GSM-R wireless standard.

This chapter presents an overview of the ETCS development history, its components, and service levels.

1.1 Development History

The idea of standardization of European railway signaling dates back to 1990 when European Railway Research Institute (ERRI) assembled A200 expert group to work on the specifications. The group focused on developing an on-board computer architecture (EUROCAB) and data transmission systems (EUROBALISE and EURORADIO). [3]

In 1993, another expert group was established as a result of an interoperability directive issued by the EU. The group was named ERTMS and its goal was to define the TSI.

Companies involved in the railway signaling industry such as Siemens, Alcatel, or Bombardier joined the effort and created the UNISIG consortium to work on the TSI.

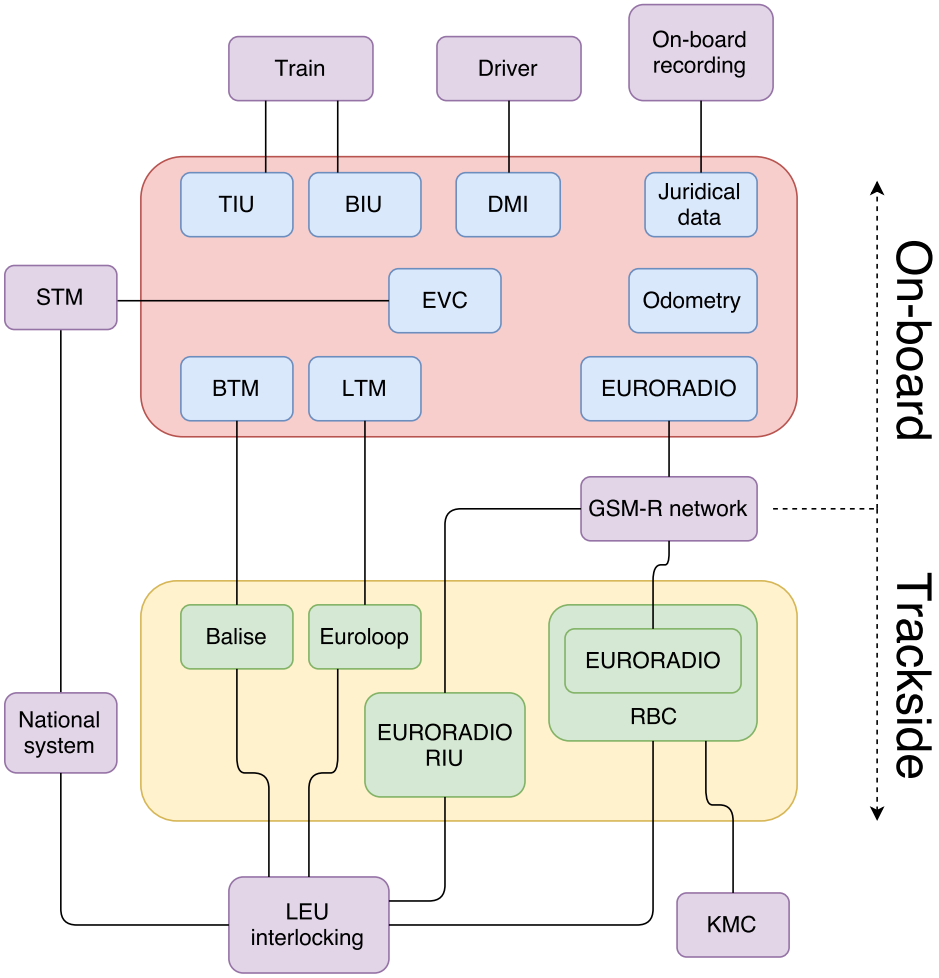
The first ERTMS specification was officially commissioned in 2000. Around this time, the system was deployed on several tracks across Europe to test the technology.

The year 2009 marks the adoption of the European ERTMS Deployment Plan. This documents defines six main European railway corridors where the use of ETCS is mandatory.

1.2 ETCS Architecture

ETCS consists of an on-board and trackside sub-system. Each sub-system includes a number of components described below.

Figure 1.1: ETCS Reference Architecture



1.2.1 Balise

Balise is an up-link wireless communication device that functions as an interface between the on-board computer and the trackside sub-system. Data transmitted by balises are called telegrams. Balise transmission module (BTM) is used by the on-board equipment to receive the telegrams. Balises are organized into groups, and telegrams sent by one group create a message. [4]

Balise message can be either fixed or dynamically changed. Variable messaging requires a connection to an LEU (Lineside Electronic Unit).

1.2.2 Lineside Electronic Unit

LEU serves as an interface between the balises and the external interlocking system. It receives data from the trackside infrastructure, assembles telegrams and sends them to the balises. [5]

1.2.3 Radio Block Center

RBC is a computer that gathers data from the on-board units and the external trackside systems, and generates messages to be sent back to the trains. The message provides the movement authorities required for train separation. [5]

Each RBC has its area of responsibility. Transfer from one area to another is governed by a handover process.

1.2.4 Euroloop

ETCS Level 1 track to train communication is based on spot data transmission via balises. When a state of the system changes, the on-board sub-system will receive the information only after the train reaches the next balise group. To mitigate this delay, Euroloops are used on ETCS level 1 track as an additional mean of data communication. The loops consist of a leaky cable placed in the track serving as an antenna transmitting data to on-board loop transmission module (LTM). [6]

1.2.5 Radio In-fill

Radio in-fill serves the same purpose as the Euroloop. The in-fill information is provided via the GSM-R network.

1.2.6 GSM-R Network

GSM-R provides wireless bi-directional connectivity for data and voice transmission between the train and the trackside systems. The technology is based on the GSM (Global System for Mobile Communication) cell technology. GSM-R is an essential component of the ETCS level 2 and level 3. [5]

EURORADIO is a component used by both on-board and trackside sub-systems as an interface to the GSM-R network.

1.2.7 Key Management Center

Communication over GSM-R between the trackside and on-board equipment is secure. The network identifies and authenticates all entities involved in the communication and then protects the transmitted data. The key management center is responsible for generation, allocation and distribution of the required cryptographic keys. [7]

1.2.8 European Vital Computer

EVC is the core element of the ETCS on-board sub-system. It is connected to other modules and interfaces as seen in the Figure 1.1. The computer provides the ETCS on-board functionality such as braking curve calculation and speed supervision.

1.2.9 Driver Machine Interface

Via the DMI, the driver interacts with the ETCS on-board sub-system. DMI displays information such as the current train speed, permitted speed, or target distance. The driver also uses the DMI to input data or to acknowledge certain situations.

1.2.10 Juridical Data

ETCS on-board sub-system is connected to a train data recorder. The sub-system sends data to the recorder when triggered by specific events. The data consists of a message associated to the event and supplementary information such as the current time, speed, position, level, mode, and system version. [8]

Examples of events triggering the recorder include Eurobalise telegram reception or a 5-second tick. [8]

1.2.11 Odometry

Odometry component provides an inertial positioning. Train position is estimated as a distance traveled from an initial location. Eurobalise data are used to derive the initial fix.

1.2.12 Train Interface Unit

TIU provides the ETCS on-board sub-system with access to train functions such as cab activation, power switch, or pantograph operation. TIU also connects to any external train integrity system. [9]

1.2.13 Brake Interface Unit

The main purpose of the BIU is to control service and emergency brakes. Via the BIU, the ETCS on-board equipment can trigger brake commands when an intervention is necessary. [9]

1.2.14 National System

This denotes any national train protection system that is deployed alongside the ETCS. French Le Crocodile, German LZB or Indusi are examples of national systems. They have similar purpose as the ETCS, ensuring safe movement and separation of trains.

1.2.15 Specific Transmission Module

Specific transmission module is a component that allows the ETCS on-board sub-system to interact with a legacy national control system. The module allows an ETCS equipped train to operate on a track without ETCS where a national system is present.

1.3 ETCS Levels

ETCS defines several levels with varying functionality. The level differs in the required trackside equipment and the means of data transmission.

Levels 1, 2, 3 are downward compatible. It is possible to operate a track on multiple levels at the same time [5]. However, the compatibility requires all necessary equipment from the lower levels. For example, level 3 track cannot support level 1 operation if no underlying interlocking system is present.

1.3.1 ETCS Level 0

Level 0 refers to operation of trains equipped with the ETCS on-board subsystem on tracks where the ETCS trackside sub-system is not present.

At this level, the driver relies on external optical signals. The ETCS onboard equipment offers only the maximum design speed supervision.

Underlying external interlocking and signaling systems are responsible for train integrity supervision and train detection. [5]

1.3.2 ETCS Level STM

STM Level covers the case of a train equipped with the ETCS on-board sub-system running on a national system-only track. The STM level should provide similar supervision capability as the legacy system. The STM onboard interface can support multiple national systems.

1.3.3 ETCS Level 1

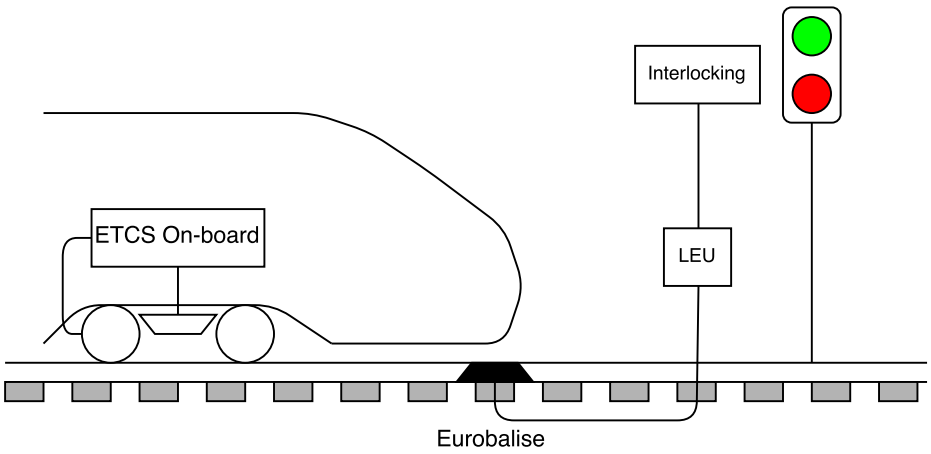
ETCS level 1 employs switchable balises for track to train data transmission. The communication system can be enhanced with the Euroloops and the radio in-fills. The trackside sub-system does not have any information about the train to which the data are sent.

Via continuous speed monitoring, the supervision functionality prevents exceeding the permitted speed and overrunning the movement authority.

The data transmission is not continuous, however; the driver is still required to watch the trackside signals as new information may not be immediately available to the onboard sub-system. [5]

Level 1 can be seen as an extension of the underlying interlocking and signaling system that is responsible for train integrity supervision and train detection. [5]

Figure 1.2: ETCS Level 1



1.3.4 ETCS Level 2

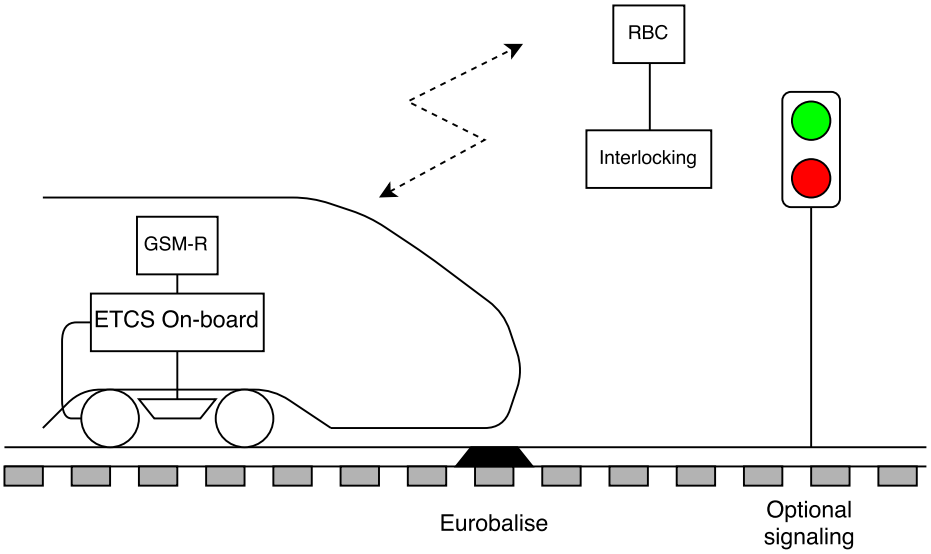
ETCS level 2 relies on the GSM-R network for data communication. Balises with fixed messaging are used as location reference.

The RBC knows the identity of all trains in its area of responsibility, and the data are sent to each train individually. Furthermore, the level 2 is not an up-link only. Trains are able to send messages over the GSM-R back to the ETCS trackside infrastructure. [10]

Level 2 ensures that the trains do not exceed the permitted speed or overrun assigned movement authority.

Thanks to the GSM-R continuous data transmission, the trackside signals are not required. L2 relies on the underlying interlocking and signaling system for train integrity supervision and train detection. [5]

Figure 1.3: ETCS Level 2

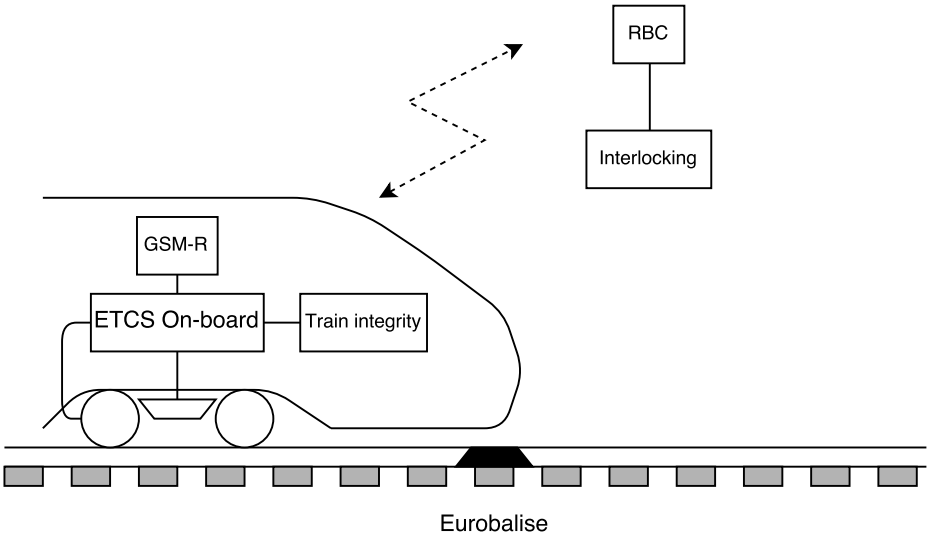


1.3.5 ETCS Level 3

Similar to the level 2, ETCS level 3 is also based around the GSM-R network continuous data transmission but offers extended capabilities.

Level 3 is a standalone system that does not require any underlying signaling. The train integrity supervision and train detection is done by the RBC and the on-board train integrity equipment. Trackside signals are not used at this level. [5]

Figure 1.4: ETCS Level 3



Chapter 2

DMI Description

This chapter serves as an overview of the DMI principles and functions, as defined in the ERA document [11]. Not all features and requirements defined in the specification are covered, only the key ones.

2.1 Glossary

The following terms are used in the rest of this chapter to describe the DMI functions.

- Movement authority (MA) – *"Permission for a train to run to a specific location within the constraints of the infrastructure."* [12]
- Most restrictive speed profile (MRSP) – *"The speed which a train must not exceed. It is the lowest speed taking into account all the various speed profiles."* [12]
- V_{train} – current train speed
- V_{perm} or V_{PSL} – permitted speed
- V_{target} – permitted speed at the end of a movement authority
- V_{WSL} – Warning supervision limit threshold
- V_{ISL} – Indication supervision limit threshold
- V_{SBI} – Threshold for triggering the service brake intervention
- V_{rel} – *"A speed value calculated within the ERTMS/ETCS to allow a train to approach the end of its movement authority in a safe way."* [12]

2.2 General Principles

These principles refer to rules and definitions valid in all the different parts and elements of the DMI.

2.2.1 Resolution and Colors

The entire display area has a ratio of 4:3 with the resolution of 640 px × 480 px at minimum. The width and height of the display must be at least 180 mm and 150 mm respectively. [11]

Colors used in the DMI are defined in the Table 2.1.

Table 2.1: DMI Colors

Color	RGB HEX	Color	RGB HEX
White	FFFFFF	Black	000000
Dark blue	031122	PASP dark	21314A
Gray	C3C3C3	Dark gray	555555
Red	BF0002	PASP light	294A6
Yellow	DFDF00	Orange	EA9100
Shadow	081839	Medium gray	96969

2.2.2 Text and Numbers

Alphanumeric characters are displayed with a font that does not use serifs. The recommended ones are Helvetica, Verdana, Swiss, and Chicago. The default text color is gray. [11]

2.2.3 Sounds

DMI has an audio output with four different sound signals in total. Two of the signals are related to the speed monitoring function and are played when the supervision status changes. The third one called “Sinfo” is a general notification sound. The fourth signal is used as a button click feedback.

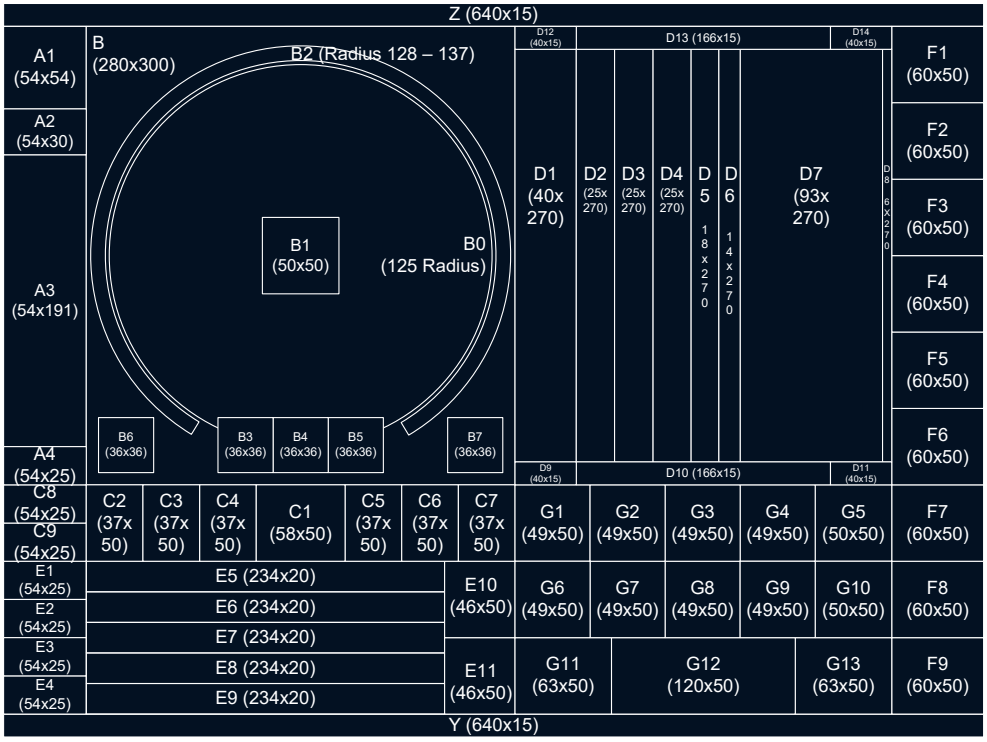
2.2.4 Display Area Layout

The display is defined as a two-dimensional grid array of cells. A cell is defined as one or multiple pixels. For example, when the basic 640 px × 480 px resolution is used, one cell equals one pixel. The cell grid array can be seen as a Cartesian coordinate system.

The grid array is divided into sub-areas serving specific functions. Each sub-area is positioned using an (X, Y) tuple representing the number of cells on the X and Y axis from the top-left corner of the display area. The position of various graphical elements located inside the sub-areas is defined relative

to the top-left corner of the particular sub-area. The width and height of the sub-areas and other graphical elements are also defined in terms of cells [11].

Figure 2.1: DMI Display Area Layout - Touchscreen



Source: [11]

2.2.5 Buttons

All buttons have three states called “enabled”, “disabled”, or “pressed”. Multiple buttons cannot be in the “pressed” state simultaneously. There are three types of buttons called “up-type”, “down-type”, and “delay-type”; each with different logical transition between the button states.

2.2.6 Acknowledgments

Certain situations require the driver to respond with an acknowledgment in the form of pressing an up-type button.

When the touch screen technology is used, the location of the acknowledgment button is situation-dependent. With soft keys, the button is located in the area H7. It is also possible to have an external button located on the driver’s desk. [11]

Only one acknowledgment can be displayed to the driver at one time. If two or more objects require acknowledgments simultaneously, they are put in

a FIFO (first in, first out) queue and presented to the driver one by one with a period of 1 s between each acknowledgment. [11]

2.2.7 User Input

The DMI specification allows the implementation to use either a touch screen or hardware keys around the DMI with associated software buttons.

The two variants differ in the object layout on the default screen and structure of the sub-level windows.

2.2.8 DMI and ETCS Modes

ETCS modes define the general behavior of the system. A complete description of all the modes is available in [5].

DMI functions differ with each mode. For example, some elements of the interface might not be available unless the system is in the FS mode.

2.2.9 DMI and STM

DMI supports custom objects and functions that interpret data coming from a national train control system when the train operates at the STM level.

For example, the Czech national control system is LS06. One of its functions is a signal repeater which displays trackside signals to the driver inside the train. This feature can be integrated into the DMI as a custom object; thus, no dedicated LS06 interface is required.

2.3 Speed Monitoring

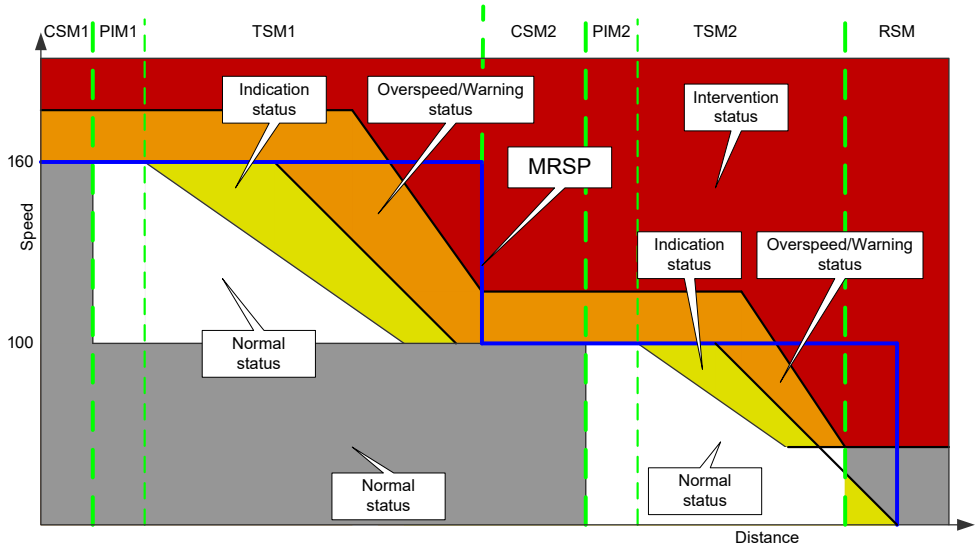
The core feature of the ETCS is its speed monitoring functionality. Based on the MRSP and other relevant data, the system computes the dynamic speed profile which is compared to the actual train speed to ensure that the train stays within the permitted limits.

For the purpose of speed monitoring, movement authority area is divided into different supervision sections:

- Ceiling speed monitoring section (CSM)
- Pre-indication monitoring section (PIM)
- Target speed monitoring section (TSM)
- Release speed monitoring section (RSM)

In each section, the braking curves define multiple supervision statuses. When a train moves through an authority area, it always has an associated supervision status depending on the train speed and the current supervision section (assuming the full supervision mode is available).

Figure 2.2: ETCS Speed Supervision



Source: [11]

2.3.1 Ceiling Speed Monitoring

CSM refers to the section of a movement authority area with no nearby speed restriction, which would require decreasing the train speed to a target speed.

- Normal status (NoS): If $V_{train} \leq V_{PSL}$
- Over-speed status (OvS): If $V_{train} > V_{PSL}$, deactivated if $V_{train} < V_{PSL}$
- Warning status (WaS): If $V_{train} > V_{WSL}$, deactivated if $V_{train} < V_{PSL}$
- Service brake intervention status (IntS): if: $V_{train} > V_{SBI}$, deactivated with the on-board sub-system SBI command

Table 2.2: CSM Order of Precedence

↓ supersedes →	IntS	Was	OvS	NoS
IntS	-	Yes	Yes	Yes
WaS	No	-	Yes	Yes
OvS	No	No	-	Yes
NoS	No	No	No	-

2.3.2 Pre-indication Speed Monitoring

PIM prepares the driver for an incoming speed restriction. PIM defines the supervision in the same way as the CSM [11].

2.3.3 Target Speed Monitoring

TSM supervises gradual decrease of the train speed to a new target speed.

- Normal status (NoS): If $V_{train} \leq V_{ISL}$
- Indication status (IndS): If $V_{train} > V_{ISL}$, deactivated when $V_{train} < V_{target}$
- Over-speed status (OvS): If $V_{train} > V_{PSL}$, deactivated if $V_{train} < V_{PSL}$
- Warning status (WaS): If $V_{train} > V_{WSL}$, deactivated if $V_{train} < V_{PSL}$
- Service brake intervention status (IntS): If $V_{train} > V_{SBI}$, deactivated with the on-board sub-system SBI command

Table 2.3: TSM Order of Precedence

↓ supersedes →	IntS	Was	OvS	IndS	NoS
IntS	-	Yes	Yes	Yes	Yes
WaS	No	-	Yes	Yes	Yes
OvS	No	No	-	Yes	Yes
IndS	No	No	No	-	Yes
NoS	No	No	No	No	-

2.3.4 Release Speed Monitoring

RSM is active in the vicinity of the end of a movement authority.

- Indication status (IndS): If $V_{train} \leq V_{release}$
- Service brake intervention status (IntS): If $V_{train} > V_{rel}$, deactivated with the on-board sub-system SBI command

Table 2.4: RSM Order of Precedence

↓ supersedes →	IndS	IntS
IntS	-	Yes
IndS	No	-

2.4 Current Train Speed Information

DMI displays the current speed of the train in a digital form and using circular speed dial with a pointer.

Figure 2.3: Current Train Speed Presentation

Source: [11]

2.4.1 Train Speed Digital

Train speed is displayed digitally in the center of the circular dial. To display the speed correctly, the current supervision status is required because the color of the digits depends on this information.

Table 2.5: Train Speed Digital Colors

Supervision	Color
NoS/IntS/OvS/WaS/undefined	Black
IndS	Red

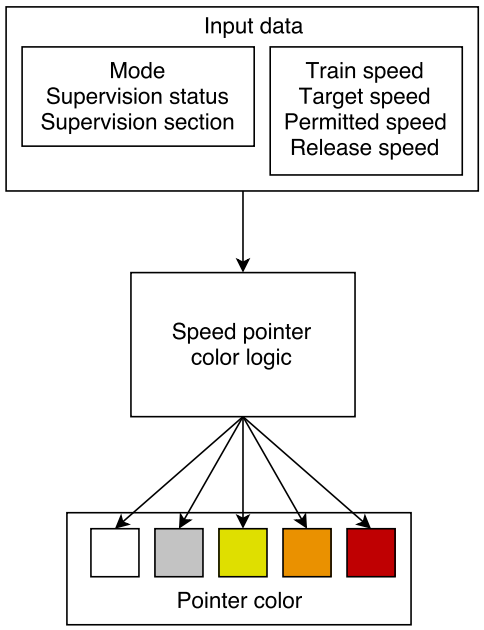
2.4.2 Circular Speed Dial

Four possible dials are allowed. DMI is set to display one selected range, and the configuration cannot be changed by the driver. The four ranges are following:

- 0 km/h – 140 km/h
- 0 km/h – 180 km/h
- 0 km/h – 250 km/h
- 0 km/h – 400 km/h

The dial also visualizes the current supervision status by changing the color of the pointer. To resolve the color, the DMI needs input information as shown in the Figure 2.4. The logic itself is described by the Table 8 in the ERA document [11].

Figure 2.4: Pointer Color Overview



2.5 Speed Supervision Information

The driver is informed about the current speed supervision either via the circular speed gauge or basic speed hooks.

2.5.1 Circular Speed Gauge

CSG (Circular speed gauge) is an area around the speed dial displaying the permitted speed, target speed, release speed, and service brake intervention limit. The CSG is operational only in the FS (full supervision) mode and it consists of four segments.

The main segment points to the current value of the permitted speed and is always visible.

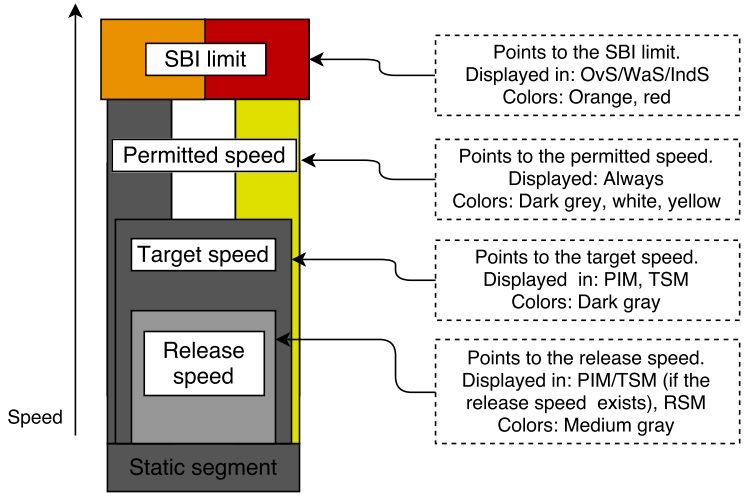
The second segment points to the SBI limit. This segment is visible when the train exceeds the permitted or release speed and the supervision is in the warning, over-speed, or intervention status.

The third segment displays the target speed during the ceiling speed and pre-indication monitoring.

The last segment functions as the release speed visualization.

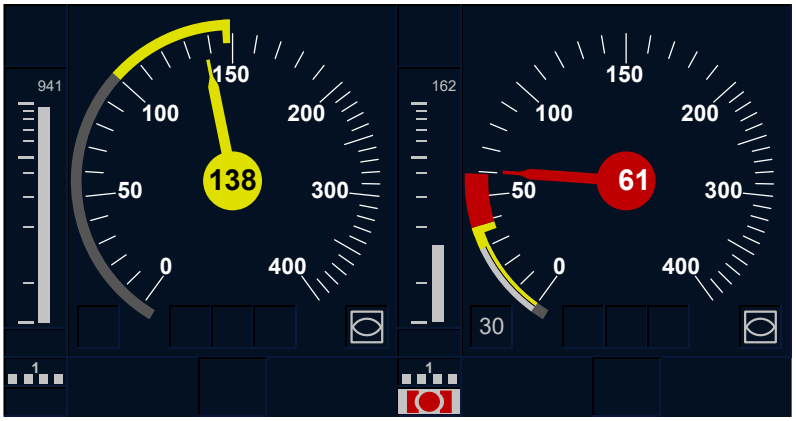
Complete logic controlling the CSG is defined by the Table 9 in the ERA document [11].

Figure 2.5: CSG Overview



In the Figure 2.6, the left CSG displays the target speed and permitted speed. The right CSG shows the release speed, permitted speed and the SBI limit.

Figure 2.6: CSG in TSM/IndS and TSM/IntS



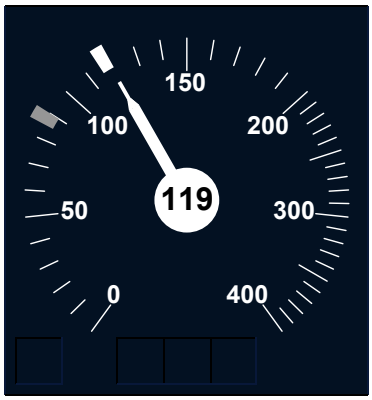
Source: [11]

2.5.2 Basic Speed Hooks

In modes RV (reversing), OS (on sight), SR (staff responsible), SH (shunting), when the CSG is not available, the DMI offers the basic speed hooks to visualize the permitted speed and target speed [11].

The basic hook representing the permitted speed is always available. The target speed basic hook is visible during the PIM, TSM, and RSM. [11]

Figure 2.7: Speed Dial with the Basic Hooks



Source: [11]

2.6 Braking Information

Brake information consists of the distance to target display and service/emergency brake intervention indication.

2.6.1 Distance to Target Information

Distance to target is presented digitally and by a bar in the sub-area A3 that scales analogically with the target distance. An example of the distance to target information display can be seen in the Figure 2.6.

The distance bar is active when the target distance is less than 1000 m. Logarithmic mapping is used between 100 m and 1000 m. [11]

The relation between the distance to target value D_{target} and the scale S of the distance bar is

$$S = \begin{cases} \log_{17}(D_{target}(58.8236\,\mathrm{m})^{-1}), & \text{if } 100\,\mathrm{m} < D_{target} \leq 1000\,\mathrm{m} \\ 1.772 \times 10^{-3}\,\mathrm{m}^{-1} D_{target}, & \text{if } 0\,\mathrm{m} < D_{target} \leq 100\,\mathrm{m} \end{cases} \tag{2.1}$$

where $S = 1$ represents the bar at its full size.

2.6.2 Intervention Indication

The sub-area C9 displays information regarding the service/emergency brake intervention and might also serve as an acknowledgment button.

2.7 Supplementary Information

Supplementary information refers to various supervision and track conditions that are presented to the driver via the DMI.

Depending on the exact situation, some of the information might require an acknowledgment or another defined response from the driver.

Level, mode, and track conditions are presented using graphical symbols defined in the Chapter 13 of [11].

Section 8.2.3 of [11] covers the entire specification of the supplementary driving information.

2.7.1 Level and Mode

DMI displays the current supervision mode and ERTMS/ETCS level to the driver.

2.7.2 Track Conditions

Various types of track conditions are listed below.

- Track ahead free
- Tunnel stopping area
- Level crossing
- Orders and announcements
  - Air condition intake
  - Pantograph
  - Neutral section
  - Radio hole
  - Non-stopping area
  - Brake inhibition
  - Traction system
  - Horn

Specific subset of track conditions is called orders and announcements. Orders require a response from the driver that is not executed via the DMI. For example, an order to lower the pantograph or open the air condition intake. Announcements require no response.

2.7.3 Text Messages

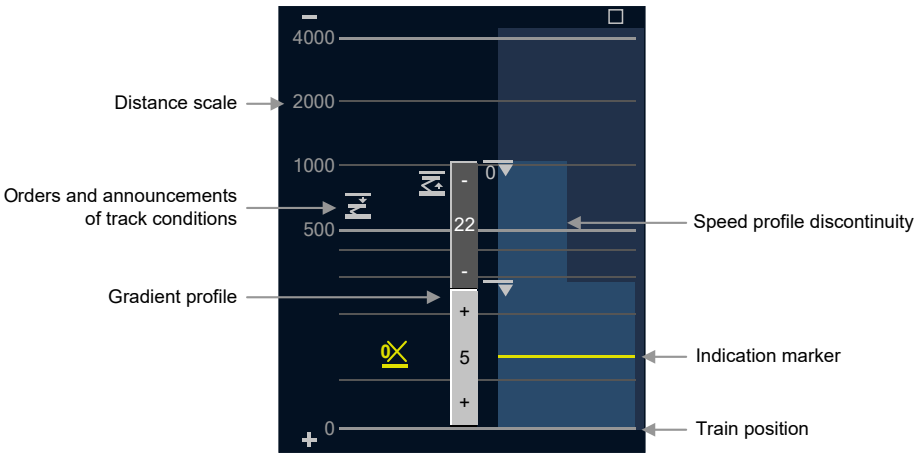
Via the DMI, the driver can be presented with information in the form of text. The content of these messages is specified in the Chapter 15 of [11].

2.8 Planning Area

Planning area is an object displayed in the D sub-area of the DMI default screen. Its purpose is to inform the driver of various conditions ahead of the train. The planning area displays the following:

- Gradient profile
- Indication marker
- Speed profile
- Orders and announcements of track conditions

Figure 2.8: Planning Area



Source: [11]

2.8.1 Distance Axis

The distance axis is vertical and always starts at the front of the train. As the train moves, the entire planning area scrolls up and all the objects move towards the beginning of the axis.

The planning area offers a zoom function, which may change the scale of the distance axis. There are six scales available with maximum distance ranging from 1000 m to 32000 m [11].

The relation between the distance D_O to an object and its position x_O in the area in pixels from the bottom is

$$x_O = \begin{cases} 50 + 2.32 \times 10^{-2} \log_2(\frac{D_O}{z_L}), & \text{if } z_L < D_O \leq 40 z_L \\ 17 + 33 \frac{D_O}{z_L}, & \text{if } 0 < D_O \leq z_L \end{cases} \tag{2.2}$$

where $z_L = \frac{D_{max}}{40}$ is the zoom level used.

2.8.2 Gradient Profile

Gradient profile is presented using rectangles with different colors as seen in the Figure 2.8. Uphill gradient is displayed with gray color, downhill gradient uses dark gray.

The numeric value of the gradient is shown as well.

2.8.3 Indication Marker

Indication marker represents the location on the track where the TSM to the next target begins.

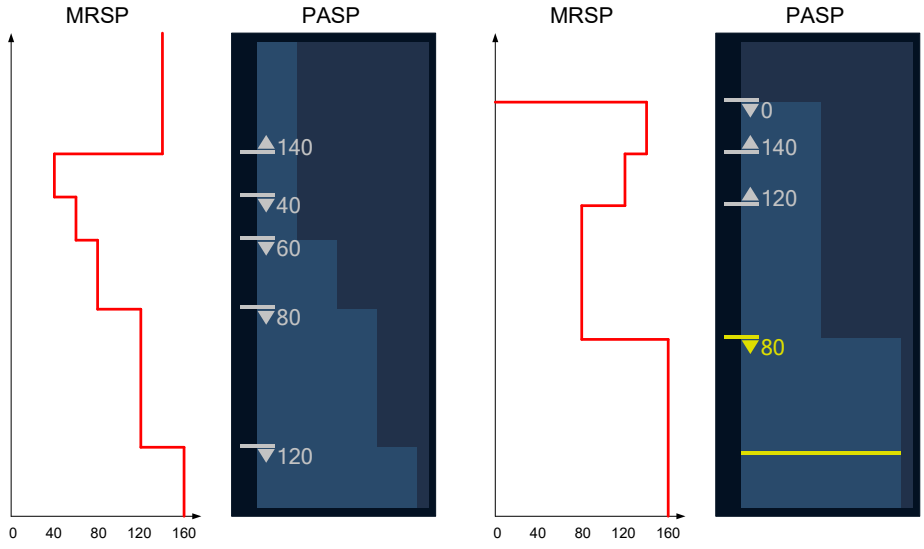
2.8.4 Speed Profile

Speed profile is a diagram displaying MRSP-based speed discontinuities ahead of the train.

Horizontal axis, where the MRSP values are presented, is discrete. The axis is divided into four quarters. Permitted speed at the current train position always takes all four quarters of the axis.

The speed profile displays up to three restrictions as 3/4, 2/4, 1/4 of the axis respectively. The last quarter is reserved for $V_{target} = 0$. Upward MRSP discontinuities are not presented in the diagram. The three quarters represents 99% to 75%, 74% to 50%, and 49% to 1% decrease in the permitted speed at the current position. [11]

Figure 2.9: Planning Area Speed Profile



Source: [11]

Additionally, the speed profile is presented by graphical symbols with numeric values as seen in the Figure 2.9. These symbols show both decreasing and increasing speed discontinuities of the MRSP.

2.8.5 Orders and Announcements

The planning area displays the same orders and announcements of track conditions as previously described in the section 2.7.2.

2.9 Monitoring Information

Monitoring information is similar to the supplementary information previously mentioned in the section 2.7. However, it does not require any response from the driver.

Monitoring information includes:

- Radio connection indication
- Reversing permitted indication
- Local time
- Geographical position

2.10 Sub-level Windows

Sub-level windows can be summoned on demand and serve for tasks such as accessing settings, entering required data, or navigating various dialogues.

This section serves as a brief overview of each sub-window. For the complete specification, please refer to chapters 10 and 11 of [11].

2.10.1 Menu Windows

Menu windows serve as entry points offering access to other sub-level windows described in this section.

2.10.2 Data Entry Windows

Data entry windows are interfaces used to enter the data listed in the Table 2.6. Each window has an associated keyboard with allowed input values.

The values entered by the driver are subjected to a data check. The values need to have a valid resolution, be in a valid range, and be consistent with other inputs. There are two kinds of data checks. Failed operational data check can be overruled by the driver, failed technical data check requires a new data entry. [11]

Data entry windows might be accompanied with a “Data entry complete?” question and a “yes” answer button confirming the input.

Table 2.6: Data Entry Windows

Data window	Input
Train running number	Numeric
ETCS level	Selected from predefined values
Driver ID	Numeric
Radio network ID	Selected from predefined values
RBC ID and phone number	Numeric
Language	Selected from predefined values
Volume	Implementation specific
Brightness	Implementation specific
SR speed and distance	Numeric
Adhesion	Selected from predefined values
Set VBC	Numeric
Remove VBC	Numeric

There is one additional window for train data entry described in the Table 2.7. This window allows two possible ways of entering the data.

Flexible train data entry means that each input field can be modified separately by the driver. However, some of them might be preconfigured or received from external sources. [11]

When fixed train data entry is used, the driver only chooses one of the train types that contains a predefined combination of all train data values.

Table 2.7: Train Data Entry Window

Data	Input
Train category	Selected from predefined values
Length	Numeric
Brake percentage	Numeric
Maximum speed	Numeric
Axle load category	Selected from predefined values
Airtight	yes/no
Loading gauge	Selected from predefined values

2.10.3 Data Validation Windows

Validation windows are used to validate or invalidate data entries using a “yes” or “no” answer. Train data, set VBC, and remove VBC windows have associated validation windows. [11]

2.10.4 Data View Windows

All the items that can be displayed via the data view windows are listed in the Table 2.8.

Table 2.8: Data View Windows

Data item	Note
Driver ID	
Train running number	
Train type	Fixed train data entry only
Train category	Only when modifiable
Length	Only when modifiable
Brake percentage	Only when modifiable
Maximum speed	Only when modifiable
Axle load category	Only when modifiable
Airtight	Only when modifiable
Loading gauge	Only when modifiable
Tadio network ID	
RBC ID	
RBC phone number	
VBC code set	Multiple codes possible
System version	

2.10.5 Dialogue Sequences

Dialogue sequences connect multiple sub-level windows and navigate the driver to perform certain actions. Chapter 11 of [11] specify the sequences.

- **Start up sequence** guides the driver through all the windows and data entries required for the start of a mission.
- **Main window sequence** defines the relationship between the main window and other sub-level windows that are accessed via the main window.
- **Shunting sequence** serves for authorization of a shunting action.
- **Settings window sequence** connects the settings window with sub-level windows such as Volume, Brightness, or Language.
- **Special window sequence** defines access to the adhesion and SR speed/distance windows via the special window.
- **Override sequence** is used to override an EOA.

Chapter 3

Hardware and Operating System

The first part of this chapter introduces the device which the DMI application will run on. The second part goes over possible operating systems that could be deployed.

3.1 Hardware

A panel computer from the AP9 series made by the Czech company AMiT is used as the target hardware. It was selected because AMiT computers are used in real locomotives and EMUs such as ŠKODA 109E, ŠKODA 10Ev, or STADLER FLIRT 3.

AP9 is a series of embedded industrial computers that combine the internals with all the peripherals into one self-contained unit.

A total of four AP9s will be used in the train simulator. Aside from the DMI, they will serve as the diagnostic display, electronic timetable, and train radio.

Table 3.1: HW Specifications

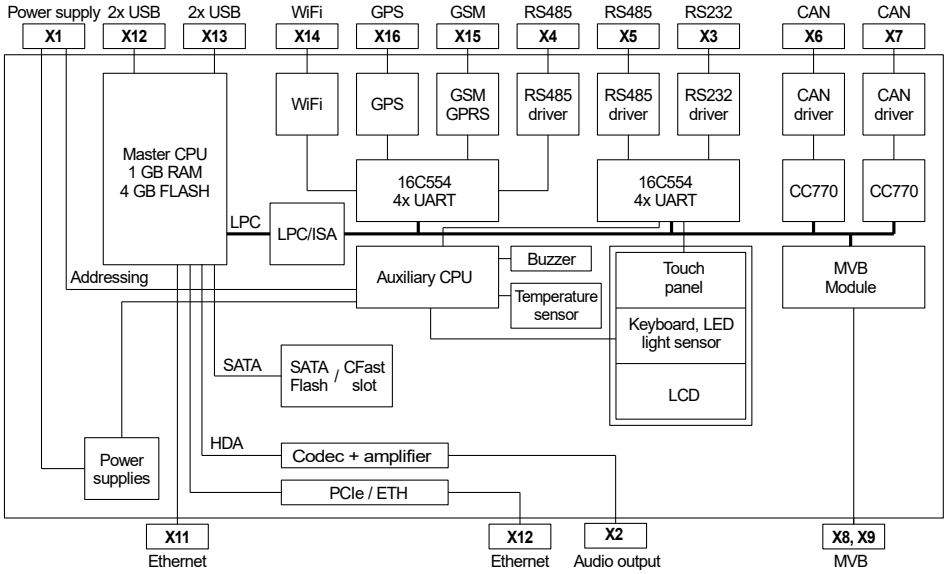
CPU	Intel Atom E680T 1.6 GHz
RAM	DDR2 1 GB
Flash	2 GB
GPU	Intel GMA 600
Display resolution	800 px × 600 px
Touch screen	Capacitive
Peripherals	USB 2.0
Networking	Ethernet, CAN, RS232
OS	Linux, Windows Embedded

AP9 is built around a kontron COM board containing an Intel Atom CPU, RAM, and flash memory. The main CPU is connected to an auxiliary Cor-

tex CPU that handles additional functions. The CPUs communicate using a UART serial link.

Figure 3.1: AMiT AP9

Figure 3.2: AMiT AP9 High Level Schematics



Source: [13]

3.1.1 Peripherals

The main peripherals are the 10.4 inch TFT display with a capacitive touch screen and a keyboard with 24 backlit keys placed around the display. The computer also has a light sensor that can be used to automatically adjust the screen brightness and the keyboard backlight.

3.1.2 Auxiliary CPU

Auxiliary CPU services the keyboard, controls the screen brightness, or turns on/off the computer. The main CPU sends commands over the UART link and receives responses from the auxiliary CPU. The communication uses ACK/N-ACK messages comprising a variable number of fields.

Table 3.2: Auxiliary CPU Messages

Command		Response	
Field	Value	Field	Value
Message start	0x1B	Message start	0x1B
#	0x23	#	0x23
Command		Command	
Parameter P_1		Return value V_1	
Delimiter	0x2C	Delimiter	0x2C
Parameter P_n		Return value V_n	
Message end	0x0D	Message end	0x0D

The structure of the messages is described in the Table 3.2. Complete lists of commands, parameters, and return values are available in [13].

3.1.3 Write Filters

The computer uses NAND flash instead of a conventional hard drive. This type of memory is limited by a finite amount of erase cycles it can perform. AMiT recommend using write filters to protect the memory and prolong its service life [14].

Write filter redirects data to RAM, lowering the amount of flash drive erase cycles. The disadvantage is that the data will be lost when the computer is shut down.

Windows XP Embedded and Windows 7 Embedded systems offer two write filters: EWF and FBWF. On Linux, write filters can be emulated [15].

When EWF is active, the entire drive partition is being protected. Using EWF causes continuous allocation of free RAM up to the point when no memory is available and the system crashes. To prevent this situation, periodic system restarts are required. The rate of memory allocation depends on the number of data write cycles and can reach up to 24 MB/day. This can be lowered down to 1 MB/day by optimizing the OS configuration.

FBWF works on a per-file basis. Files can be configured individually to either allow data write or redirect it to RAM. FBWF can allocate only a predefined amount of RAM so using it cannot lead to a system crash.

3.2 Operating System

AMiT distribute the computer both with or without an OS. Officially supported operating systems are Windows XP Embedded, Windows 7 Embedded Standard, and Linux. However, AP9 should be compatible with any x86 platform OS.

3.2.1 Linux

Linux refers to a family of UNIX-like operating systems that are built around the Linux kernel.

Using Linux requires no software licensing and allows the user to access the source code and customize any part of it. Many of the existing Linux distributions are already designed specifically for use in embedded devices. OpenWrt, Raspbian, or Embedded Debian are examples of embedded distributions.

3.2.2 Windows XP Embedded

Microsoft released the embedded version of Windows XP in 2001. The support for the latest version of the OS, the Service Pack 3, was ended in early 2016 [16]. However, the system is still being used.

Windows XP Embedded allows the user to build a custom-tailored system using Windows XP components the user wants. This approach helps to reduce resources required to run the OS while increasing security and stability. For example, if a device does not need an internet connection, networking capabilities can be removed from the OS. [17]

The system also includes advanced functions, such as booting over network or the already mentioned write filters. On the other hand, some desktop features are not available.

3.2.3 Windows 7 Embedded

Windows 7 Embedded, was released in 2010, and as of now it is still being supported by Microsoft.

The system uses the same component-based design as the XP Embedded to create a highly customized OS. Additionally, Windows 7 Embedded has an improved performance and is also compatible with recent versions of the .NET framework. [18]

3.3 Application Requirements

The DMI application is being developed for the x86 platform using Microsoft .NET framework. The reasons why this framework was chosen are covered in the following chapter.

The framework choice dictates the use of a Windows operating system; the computer runs Windows 7 Embedded system. Originally, the application used .NET version 4.5, which is officially supported by Windows 7. However, for an unknown reason, AMiT were unable to install this version on the computer forcing a downgrade to version 4.0.

Hardware-wise, the computer should be powerful enough to run the application without problems. The program employs relatively simple 2D graphic objects and most of the more computationally heavy tasks, such as breaking curve calculation, are handled by the EVC. Calculations done by the DMI are mostly related to transforming physical quantities such as, speed or distance, to some pixel representation on the screen.

Chapter 4

Application Framework

Frameworks offer mainly a level of abstraction simplifying the development. They also encourage use of a design pattern leading to a clear code structure with separation of user interface and logic.

In this chapter, various frameworks supporting GUI development are explored and analyzed in order to select a suitable one for the DMI implementation.

4.1 Qt

Qt is a cross-platform application framework. It is developed by The Qt Company with numerous other companies and individuals contributing to the project.

Qt can be used for free under GNU licenses if the final product is open source. Otherwise, a commercial license is required.

The framework supports all the major operating systems and can be used to develop desktop, mobile, and embedded applications.

C++ is the primary language used with the Qt, but native binding for Python is available as well. Qt offers two main options how to develop the UI. One possibility is Qt Quick and a language called QML which uses JSON-like syntax to define graphical objects and JavaScript to implement the logic. The second option is the QtWidgets module with more traditional approach of creating the UI elements directly from the C++ code. [19]

Qt Quick employs a pattern with model/view/delegate. Models contain data, views display the data and delegates control the appearance of the data.

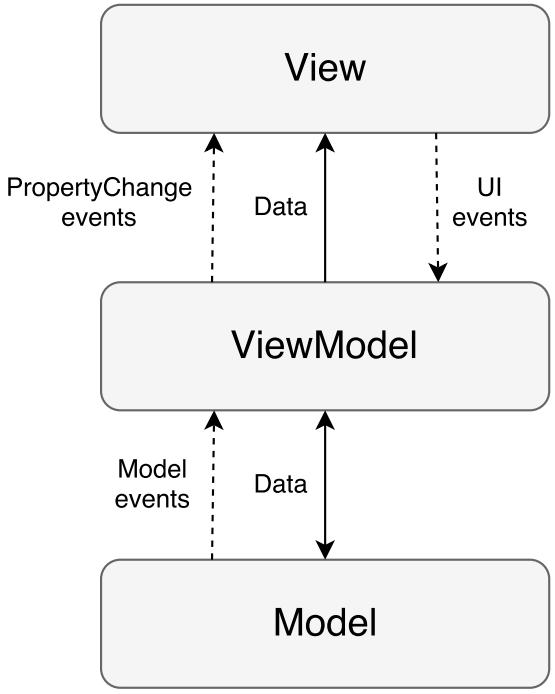
The Qt framework is shipped with its own IDE (integrated development environment) called Qt Creator.

4.2 .NET

.NET is a framework developed by Microsoft mainly for the Windows platform. Parts of the framework are open source. The framework itself consists of a class library and the Common Language Runtime, a virtual machine executing the code. The framework supports programming languages such as C#, F#, Visual Basic, or Visual C++.

In .NET, User interfaces are built using the MVVM pattern as seen in the Figure 4.1. View is the UI; it presents the data to the user. Model represents the data. For example, Models can refer to database entries. View model connects the model and the view; it controls how the data are transferred and displayed. When a UI event occurs, the view model changes the model accordingly. Conversely, model updates are propagated through the view model to the view. [20]

Figure 4.1: Model-View-ViewModel



WPF application uses XAML (Extensible Application Markup Language) to define the GUI. XAML contains basic UI objects such as geometric shapes, paths, or buttons. More complex XAML examples are ListView or GridView, which can display collections of data. DataTemplate then customizes the view presentation of the collections. XAML can also be used to create custom controls. XAML objects employ construct of data binding to connect either to other UI elements or view models.

Microsoft produces its own IDE called Visual Studio, which is heavily focused on .NET development. A community version of the IDE can be installed for free.

4.3 wxWidgets

wxWidgets is a popular open source GUI library. The development started in 1992; the latest version 3.1.0 was released in 2016. Windows, Linux, and Mac desktop platforms are supported.

The library is distributed for free under a GNU based license, which does not require publishing your own code.

wxWidgets uses the C++ language. However, bindings for numerous other programming languages, such as Python, Ruby, Lua, Java, or PHP, have been developed as well.

The library is the most suitable one for creating traditional desktop applications using native control elements.

4.4 Unity Engine

Unity Engine is a video game framework developed by Unity Technologies. Originally designed for OS X, it now supports all the major desktop and mobile platforms as well as video game consoles.

The framework is available for free. However, some advanced features, such as access to the source code, are paid.

Unity Engine has dedicated tools for 2D graphics development and UI design. Game objects are controlled by "components" that implement the logic. Unity has a selection of in-built "components", but also allows the developer to write custom scripts using either C# or UnityScript. [21]

The engine includes a high level networking API which implements communication between remote hosts and servers. A network manager, built using the API, can be used to manage the state of an application. [21]

4.5 Framework Analysis Conclusion

I have concluded that the wxWidgets and Unity Engine are not very suitable for the task of implementing the DMI.

wxWidgets is designed for basic window-based UI applications and is not entirely ideal for more graphics-oriented, touch-screen interface.

Unity Engine is focused primarily on video game development and the framework might have limitations that would require some modifications.

Between the Qt and .NET framework, I have decided to choose the .NET. The main reason is that most of the software in the CTU Transportation Laboratory was built using this framework.

Chapter 5

Implementation

This chapter documents the process of designing and developing the DMI application.

5.1 Environment

The DMI was built using my laptop with Intel Core i7 64-bit CPU, which runs Windows 10 OS. The application was periodically tested on the AP9 hardware.

I used GitHub and Visual Studio 2015 IDE to manage and write the source code of the DMI application.

5.1.1 GitHub

GitHub is a hosting service providing Git and additional features such as issue tracking, or wikis. A student account and a private repository were used to host the DMI code.

Git itself is an open source version control system that stores a snapshot of the code each time it is pushed into the repository. The project can also be divided into multiple branches for easier development.

5.1.2 Visual Studio

Visual studio organizes the code into projects. One project usually results in an `.exe` program and a `.dll` file. Multiple dependent projects are combined into solutions. The DMI code is represented by one project.

Multiple built profiles were used to build the release and debug versions of the program for either my 64-bit laptop or the 32-bit target computer.

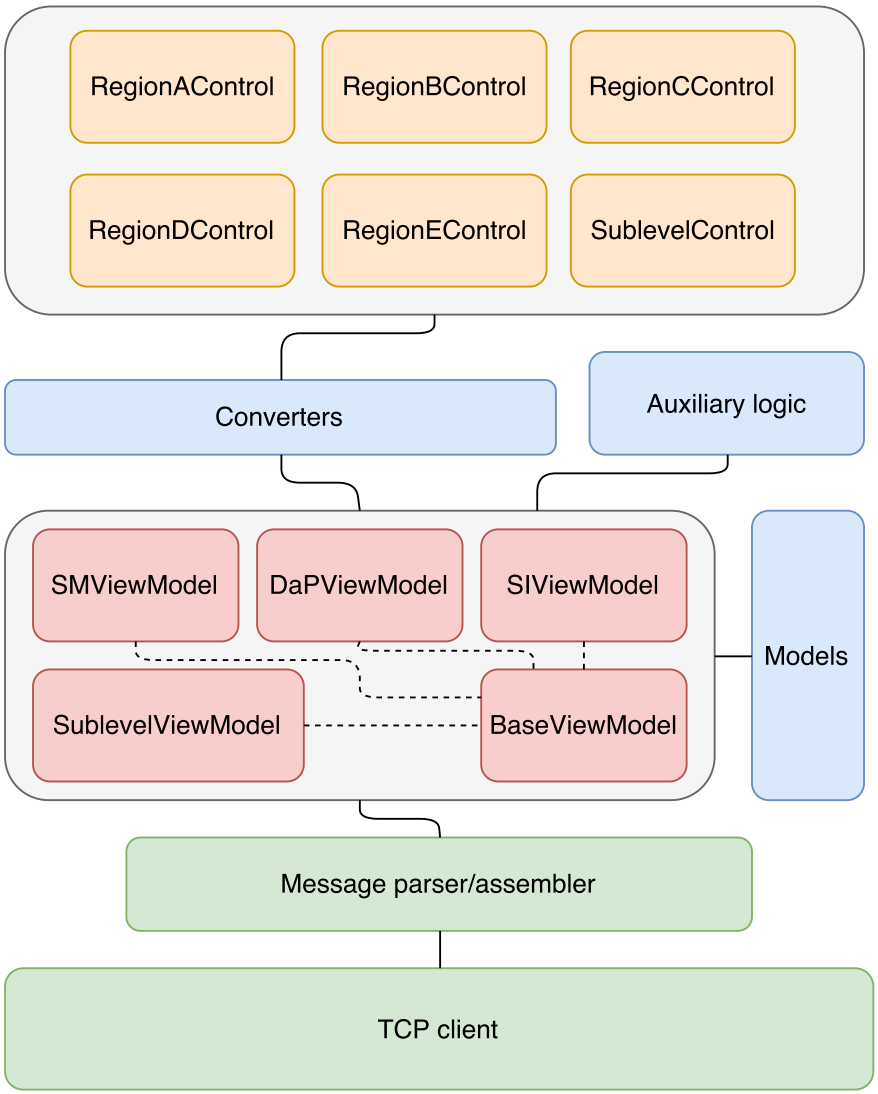
5.2 Project Setup

The project is set up as a WPF application with the target framework version 4.0. External assets, such as bitmap images or audio files, are loaded as resources directly into the solution, which means they are included directly in the executable file.

5.3 Design

The architecture of the application is based on the MVVM pattern; the main components are shown in the figure below.

Figure 5.1: Application Architecture



5.4 View

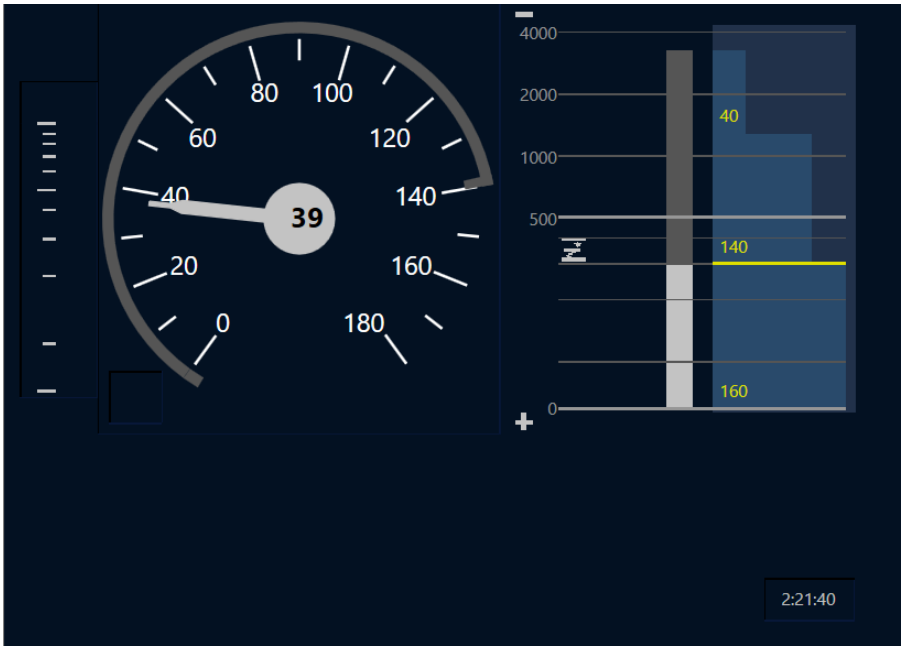
The View represents the UI itself. It contains `.xaml` files, with graphical objects defined using XAML, and associated `.xaml.cs` files with C# code, which implements interaction with the objects.

In the DMI application, the view consists of user control objects that implement the DMI functions. There are six main user controls (as seen in the Figure 5.1), five associated to the display regions of the DMI default screen, and one control for the sub-level windows.

The UI elements inside the user controls are connected via data binding to the view models which store the data sources.

The width, height, and position of all the UI elements is defined in absolute terms using $1\,\text{px} = 1\,\text{cell}$ relation. This means that the default resolution of the application is $640\,\text{px} \times 480\,\text{px}$, however; the view employs `<ViewBox>` that allows the application to be scaled to whichever resolution is desired.

Figure 5.2: DMI View



5.4.1 RegionBControl

`RegionBControl` includes the speed monitoring functions such as the current train speed pointer, CSG, and basic speed hooks. The data comes mainly from the `SMViewModel`.

5.4.1.1 Train Speed Pointer

The speed pointer is represented by a `<Polyline>` and `<Elipse>` shapes. The polyline is rotated around its origin, using `RenderTransform`, to an angle calculated from the current train speed.

To smooth the movement of the pointer, an animation is employed. The duration of the animation is 1 s which is the same interval as the refresh rate of the display. This leads to a lag; when the pointer finishes the transition to the old data point, new data comes in. However, the difference is not substantial for usual train speeds.

The color of the pointer is controlled by `SMColorEngine` object described in another section.

5.4.1.2 Circular Speed Gauge

The CSG is implemented using a total of five `<ArcSegment>` objects. One for the permitted, target, and SBI speeds; the variable width of the release speed is achieved with two overlaying segments. An additional `<Rectangle>` object represents the hook at the end of the permitted speed segment.

Listing 5.1: CSG Permitted Speed Arc Segment

```
<Path Stroke="{Binding Path=VpermColor,
                NotifyOnTargetUpdated=True}"
      Visibility="{Binding Path=VpermColor,
                   Converter={StaticResource
                              VisibilityConverter},
                   NotifyOnTargetUpdated=True}"
       StrokeThickness="8"
       DataContext="{DynamicResource SMViewModel}">
  <Path.Data>
    <PathGeometry>
      <PathFigure StartPoint="61.8,257.6">
        <ArcSegment IsLargeArc="{Binding Path=Vperm,
                                 Converter=
                                 {StaticResource
                                  IsLargeConverter}"
                    Size="133,133"
                    Point="{Binding Path=Vperm,
                            Converter=
                            {StaticResource
                             SpeedToPointConverter},
                            NotifyOnTargetUpdated=True,
                            ConverterParameter=133}"
                    SweepDirection="Clockwise"/>
      </PathFigure>
    </PathGeometry>
  </Path.Data>
</Path>
```

The Listing 5.1 shows an XAML definition of the permitted speed segment. The starting point and radius of the segment are fixed; the endpoint is bounded to the `Vperm` property. The `DataContext` assignment tells the application to look inside the `SMViewModel` for the property. Because a set of two endpoints and a radius leads to two possible arc segments, the `IsLargeArc` has to be bounded to the permitted speed to resolve which arc should be rendered.

The CSG is available only under certain conditions; the `Visibility` property binding resolves whether the CSG should be displayed or not.

5.4.1.3 Basic Speed Hooks

Basic speed hooks can be seen as a simplified version of the CSG. They comprise two `<Rectangle>` objects bounded to the target speed and permitted speed.

5.4.2 RegionDControl

`RegionDControl` is the planning area displaying the speed discontinuity profile, gradient and track conditions.

5.4.2.1 Speed Profile

Listing 5.2: Planning Area Speed Items Control

```
<ItemsControl DataContext="{DynamicResource DaPViewModel}"
              ItemsSource="{Binding SpeedProfile}"
              Background="Transparent">
  <ItemsControl.ItemsPanel>
    <ItemsPanelTemplate>
      <Canvas Width="246" Height="300"
              Background="Transparent"/>
    </ItemsPanelTemplate>
  </ItemsControl.ItemsPanel>
    <ItemsControl.ItemContainerStyle>
      <Style TargetType="ContentPresenter">
        <Setter Property="Canvas.Bottom"
                Value="{Binding Pos}"/>
        <Setter Property="Canvas.Left" Value="147"/>
      </Style>
    </ItemsControl.ItemContainerStyle>
  <ItemsControl.ItemTemplate>
    <DataTemplate>
      <Rectangle Width="{Binding Q}" Fill="#294A6B"
                 Height="{Binding Height}"/>
    </DataTemplate>
  </ItemsControl.ItemTemplate>
</ItemsControl>
```

The speed profile view is not bound to a single variable, as in the previous cases, but to a collection of objects each representing the $(X, f(X))$ data points, where X is the position and $f(X)$ is the value of the discontinuity.

The `<ItemsControl>` element contains the collection and presents it in the view (seen in the Listing 5.2). The `<DataTemplate>` defines that members of the collection are displayed as `<Rectangle>` objects with `Width` and `Height` bound to the speed limit value and to the distance where the limit is valid. The style setter is then used to bind the overall position of the UI objects to the actual position of the data points.

5.4.2.2 Gradient Profile

The gradient profile works in the same way as the speed profile. It is rather simplified because the width of the `<Rectangle>` shapes is fixed. The color of the shapes depends on the gradient values and is controlled by a converter.

5.4.2.3 Track Conditions Profile

Track condition profile also employs the `<ItemsControl>`; however, the members are displayed as `<Image>` objects with image source bound to URIs (uniform resource identifier) associated to track condition bitmaps.

5.4.3 RegionAControl

The distance to the target indicator is the main element of the control. It is implemented as a `<Line>` shape. The length of the indicator is then changed via `ScaleTransform` where `ScaleY` property is bound to the target distance value.

Listing 5.3: Distance to Target Control

```
<Line X1="0" X2="0" Y1="0" Y2="-186"
      StrokeThickness="10" Stroke="#C3C3C3"
      Canvas.Left="39" Canvas.Top="186"
      DataContext="{DynamicResource DaPViewModel}"
      Visibility ="{Binding Path=Supervision1,
                    Converter={StaticResource
                    AnalogDistanceVisibilityConverter},
                    NotifyOnTargetUpdated=True}"
      RenderOptions.EdgeMode="Aliased">
  <Line.RenderTransform>
    <ScaleTransform ScaleY="{Binding Path=Dtarget,
                             Converter={StaticResource
                             AnalogDistanceConverter},
                             NotifyOnTargetUpdated=True}">
    </ScaleTransform>
  </Line.RenderTransform>
</Line>
```

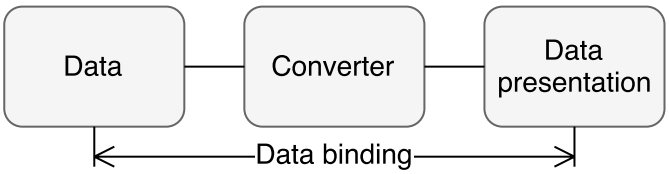
5.4.4 Not Implemented Controls

The `RegionCControl`, `RegionEControl`, and `SublevelControl` are not yet implemented in the application. However, they will use the same features of the .NET framework as the other controls. `SublevelControl` will be slightly different because it functions mainly as an input.

5.5 Converters

Converters form an interface between the view and the view model. They inherits properties from `IValueConverter` or `IMultiValueConverter`, and implement the methods `Convert` and `ConvertBack`.

Figure 5.3: Converter Function



The Listing 5.4 shows an example of a converter which takes a speed value and computes the endpoint of an arc segment as an (X, Y) amount of pixels from the top left corner.

Listing 5.4: Speed to Point Conversion Method

```
1  public object Convert(object value, Type targetType,
       object parameter, CultureInfo culture)
2  {
3
4      double V = System.Convert.ToDouble(value);
5
6      if (V > 180)
7          V = 180;
8      else if (V <= 0)
9          V = 0;
10
11     double angle = (234 - 1.6 * V) / 180 * Math.PI;
12
13     double x = 140 + System.Convert.ToInt32(parameter) *
           Math.Cos(angle);
14     double y = 150 - System.Convert.ToInt32(parameter) *
           Math.Sin(angle);
15
16     return new Point(x, y);
17 }
```

Converters are also used to control the visibility of various objects. For example, the distance to the target indicator uses a visibility converter to ensure that the bar is visible only under certain supervision condition.

5.6 ViewModels

View models contain the data that the elements of the view are bound to. Furthermore, the auxiliary-logic objects and functions are called inside the view models.

5.6.1 BaseViewModel

The `BaseViewModel` implements the `INotifyPropertyChange` interface. With this interface employed, changes to view models properties are instantly propagated to the view when the `RaisePropertyChanged` is called.

All the other view models inherits the interface from the `BaseViewModel` along with some properties that are used across multiple controls.

Listing 5.5: BaseViewModel

```
1  namespace DMI
2  {
3      public class BaseViewModel : INotifyPropertyChanged
4      {
5          public event PropertyChangedEventHandler
               PropertyChanged;
6
7          protected void RaisePropertyChanged(string
               propertyName)
8          {
9              var handler = PropertyChanged;
10             if (handler != null)
11             {
12                 handler(this, new PropertyChangedEventArgs
                       (propertyName));
13             }
14         }
15
16         // some code excluded
17
18         public BaseViewModel()
19         {
20             // some code excluded
21         }
22     }
23 }
```

5.6.2 SMViewModel

`SMViewModel` is connected mainly to the speed monitoring functions of the `RegionBControl`. It incorporates data such as the train speed and all the various speed limits.

5.6.3 DaPViewModel

`DaPViewModel` handles the planning area and the distance to the target information. It uses `ObservableCollection` objects to store the planning area profile data points.

The Listing 5.6 shows an `ObservableCollection` property containing the speed profile data. It is also shows how the `RaisedPropertyChanged` is called when the data changes.

Listing 5.6: Speed Profile Property

```
1  private ObservableCollection<SpeedPresentationModel>
       _speedProfile { get; set; }
2  public ObservableCollection<SpeedPresentationModel>
       SpeedProfile
3  {
4      get { return _speedProfile; }
5      set
6      {
7          if (_speedProfile != value)
8          {
9              _speedProfile = value;
10             RaisePropertyChanged("SpeedProfile");
11         }
12     }
13 }
```

5.6.4 Not Implemented View Models

Not yet implemented `SIViewModel` and `SublevelViewModel` will hold the data related to the supplementary information and sub-level windows.

5.7 Models

Models represents the data. In case of simple concepts, such as the train speed or the target distance, the models are the built-in data types such as `int`, `float`, or `string`. On the other hand, planning-area profile data employs more complex custom classes as the models.

Properties of the view models are implemented using the model classes. A property can hold a single model; for example, `Vtrain` holds one `float`, or multiple models using `ObservableCollection` as shown in the Listing 5.6.

5.7.1 Speed Profile Models

Speed profile models are used to represent the speed discontinuity profile data. The `SpeedModel` in the Listing 5.7 contains the value of the speed limit and the position X where the limit ends. For example, $X = 1000\,\text{m}$ means that the speed limit is valid between the end of the previous limit and 1000 m from the initial train location.

Listing 5.7: SpeedModel

```
public class SpeedModel
{
    public int X { get; set; }
    public float Speed { get; set; }
    public SpeedModel(int x, float speed)
    {
        X = x;
        Speed = speed;
    }
}
```

`SpeedPresentationModel` objects contain a subset of speed profile data which is currently being displayed in the planning area of the DMI. The `Pos`, `Height`, and `Q` specify the position and proportions of the `<Rectangle>` objects in the planning area in pixels, so no converters are required.

The process of creating a `SpeedPresentationModel` instance based on a `SpeedModel` is covered in the next section.

Listing 5.8: SpeedPresentationModel

```
public class SpeedPresentationModel
{
    public int Pos { get; set; }
    public int Height { get; set; }
    public float Speed { get; set; }
    public int Q { get; set; }
    public SpeedPresentationModel(int pos, int height,
        float speed)
    {
        Speed = speed;
        Height = height;
        Pos = pos;
        Q = 0;
    }
}
```

5.7.2 Gradient Profile Models

`GradientModel` and `GradientPresentationModel` work similarly to the speed profile models. The presentation model is simplified because of the fixed width.

5.7.3 Track Conditions Profile Models

`TrackCondModel` is almost identical to the `SpeedModel`. The difference is that it carries the track condition information instead of the speed limit. Moreover, the track conditions are related to a single position, not a track segment.

5.8 Auxiliary Logic

Auxiliary logic refers to objects that implements some functions of the DMI that were too complex to be handled by converters.

5.8.1 SMColorEngine

`SMColorEngine` includes `ResolveCSGColors` and `ResolvePointerColor` methods that are used to determine the colors of the pointer and the CSG.

5.8.1.1 ResolveCSGColors

The Listing 5.9 shows a part of the `ResolveCSGColors`. The method implements the Table 9 in the [11].

Listing 5.9: ResolveCSGColors

```
1  public static string[] ResolveCSGColors(string Mode,
       string Supervision1, string Supervision2)
2  {
3      if (Mode == "FS")
4      {
5          if (Supervision1 == "CSM")
6          {
7              if (Supervision2 == "NoS")
8              {
9                  return new string[] {"#555555", null,
10                                      null, null};
11             }
12             else if (Supervision2 == "OvS")
13             {
14                 return new string[] {"#555555", null,
15                                      null, "#EA9100"};
16             }
17             else if (Supervision2 == "WaS")
18             {
19                 return new string[] {"#555555", null,
20                                      null, "#EA9100"};
21             }
22             // some code exlcuded
23         }
24     }
25 }
```

The method uses the mode and the supervision data as input and returns an array of four elements representing the colors of each segment of the CSG. Returning `null` means that the segment is not visible.

5.8.1.2 ResolvePointerColor

This method controls the color of the train speed pointer. It implements the Table 8 in the [11]. It is also an if/else decision tree, as in the previous case. The required input is the mode, supervision status, permitted speed, target speed, release speed, and the SBI speed limit.

5.8.2 PAPresentationEngine

Planning area profile data loaded into the DMI covers the entire length of the movement authority. However, only a segment of the profile is visible in the planning area. The `PAPresentationEngine` selects which part of the data should be visible and computes where in the planning area the UI objects should be displayed. The results are then stored into instances of presentation model classes covered in the previous section. The collections of the created presentation models are the binding sources for the `<ItemsControl>` elements in the view part of the application.

Listing 5.10: DistanceToPoint

```
1  public static int DistanceToPoint(int x, int scale, int
       dfromStart)
2  {
3
4      float dist = x - dfromStart;
5
6      if (dist < 0)
7      {
8          return -1; // out of PA bounds
9      }
10     else if (dist > scale * 40)
11     {
12         return -2; // out of PA bounds
13     }
14     else if (dist <= scale) // linear part
15     {
16         return (int) Math.Round(17 + dist / scale * 33);
17     }
18     else // logarithmic part
19     {
20         return (int) Math.Round(50 + Math.Log(dist / scale
               , 2) / 5.3219 * 229);
21     }
22 }
```

The `PAPresentationEngine` uses the function in the Listing 5.10 to compute the position of an object on the distance axis of the planning area. The function implements the Formula 2.2. The `x` (the distance of the object from the initial train position) and `dfromStart` (current position of the train as the distance from the start of the movement authority) inputs are used to calculate how far ahead of the train the object is. The `scale` is the selected range of the planning area, which can be changed by the zoom in/out buttons. The function returns -1 or -2 when an object should not be displayed in the planning area because it either lies beyond the selected range or the train has already passed its position. When an object should be visible, the function returns its position in pixels from the bottom of the area.

The methods `ResolveTrackCond`, `ResolveGradient`, and `ResolveSpeed` are then used to display the correct profile data. They all start by going through the collection of associated profile models, and if conditions are met, they add the data point to the related presentation model collection which is displayed in the profile area. The presentation collections are created every time new positional data arrives to the DMI.

5.8.2.1 ResolveGradient

Algorithm 1 ResolveGradient Algorithm

$Data \leftarrow$ entire collection of gradient profile data points
$Pres \leftarrow$ empty collection of presentation models
$S \leftarrow$ selected scale
$D \leftarrow$ current distance from the start of the MA
$L \leftarrow$ position of the bottom border of the planning area
$U \leftarrow$ position of the top border of the planning area
for $i = 0$ to $count(Data)$ **do**
    $end \leftarrow DTP(x(Data[i]), S, D)$
    if $i = 0$ **then**
        $start \leftarrow DTP(0, S, D)$
    else
        $start \leftarrow DTP(x(Data[i-1]), S, D)$

    if $start = -1 \wedge end = -2$ **then**
        $Pres \leftarrow push(model(pos \leftarrow L, height \leftarrow L - U))$
    else if $start = -1 \wedge end > 0$ **then**
        $Pres \leftarrow push(model(pos \leftarrow L, height \leftarrow L - end))$
    else if $start > 0 \wedge end = -2$ **then**
        $Pres \leftarrow push(model(pos \leftarrow start, height \leftarrow U - start))$
    else if $start > 0 \wedge end > 0$ **then**
        $Pres \leftarrow push(model(pos \leftarrow start, height \leftarrow start - end))$

The Algorithm 1 describes the logic of `ResolveGradient`. The function creates a new collection of data representing `<Rectangle>` objects with calculated `Height` and `Pos` specifying their height and position on the distance axis respectively. The method pushes the instance to the presentation collection, making the data visible, if one of the following conditions is true:

1. The data point is a segment that starts beyond the bounds of the planning area.
2. The data point is a segment that intersects the lower bound of the planning area, but ends within the planning area.
3. The data point is a segment that intersects the upper bound, but starts within the planning area.
4. The data point is a segment that starts and ends within the planning area.

5.8.2.2 ResolveSpeed

The first part of the `ResolveSpeed` is identical to the `ResolveGradient`. However, there is a second step which computes the width as previously specified in the section 2.8.4.

Full `width` is assigned to the `<Rectangle>` shape representing the current speed limit. If the next limit is lower than the previous one, the `width` of the shape depends on the speed limit ratio. Speed increase has the same `width` as the previous limit.

5.8.2.3 ResolveTrackCond

The `ResolveTrackCond` is less complex than the previous cases. It adds the data to the presentation collection if the track condition is within the bounds of the planning area.

5.9 Message Parser

The purpose of the message parser is to extract the data from the massages which receives as byte arrays from the TCP client. The extracted data are then distributed to the view models and propagated into the view.

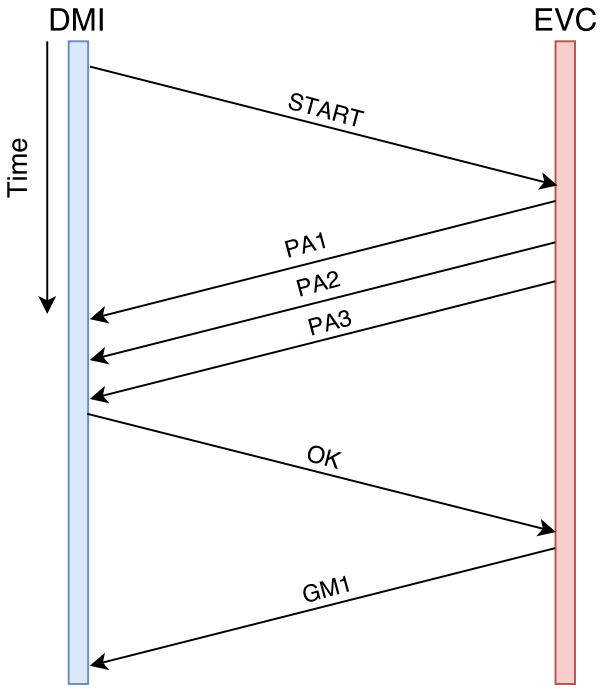
The messages received from the server may contain various amount of different data items. Each data item is accompanied with an ID and length; when the parser goes through the data it uses this information to extract the correct number of bytes, convert them to the correct data type, and assign the data to the correct view model property. The structure of the messages is further defined in the Chapter 7.

As of now, the DMI application does not yet implement features that sent data to the EVC which would require a message assembler.

5.10 Networking

The application uses TCP to send and receive data. The communication is based on the client-server architecture with the DMI being the client and the EVC being the server.

Figure 5.4: Communication Process



The Figure 5.4 shows an example of a data exchange between the DMI and an EVC emulator which have been built as a part of this thesis because the EVC for the train simulator has not yet been developed. In the future, the communication with the actual EVC software might be different. The emulator supports test scenarios that involve one movement authority. The emulator program is introduced in the Chapter 6.

The EVC emulator initiates a TCP server that listens and waits for a connection from DMI. When the connection is accepted, the DMI sends START message (byte 0xFF), commanding the emulator to start transmitting the data. The server starts with three planning area messages (PA), which cover the movement authority. The DMI responds with OK (byte 0xFE) and the emulator proceeds to send general data (GM) according to the scenario. The server always waits for the OK from the DMI before transferring next message.

Chapter 6

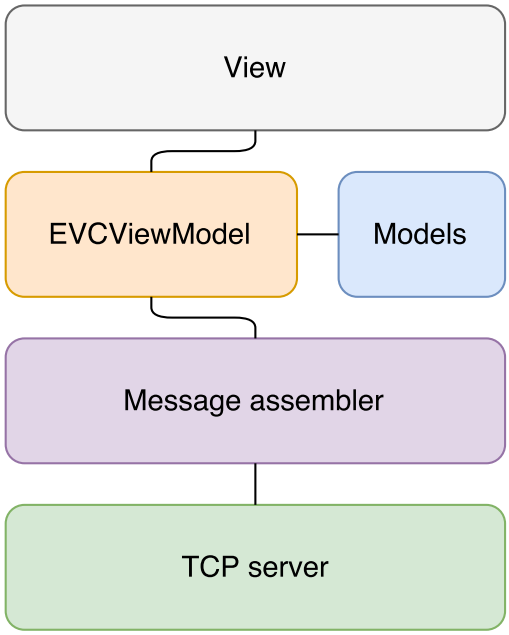
Testing

The chapter outlines the DMI backend software and simulation scenarios which are used to test the functions of the DMI application.

6.1 EVC Emulator

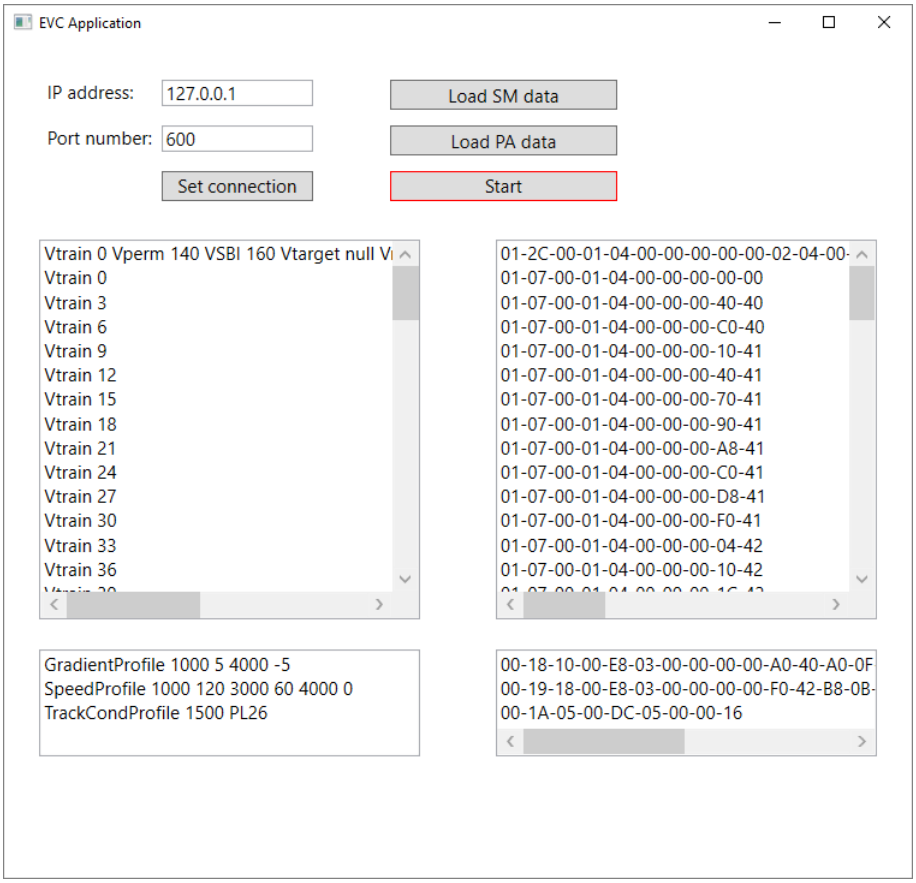
The EVC emulator serves as a backend to the DMI. It generates the messages carrying the data and sends them to the DMI. The emulator does not implement any actual logic of the EVC; its main purpose is to allow the user to input the scenario in a human-readable form.

Figure 6.1: EVC Emulator Architecture



The architecture of the program, defined in the Figure 6.1, follows the same pattern as the DMI application.

Figure 6.2: EVC Emulator UI



The user interface of the emulator is depicted in the Figure 6.2. The scenario is loaded into the application as two text files. One file for the general data, and another for the planning area data.

Each line of the text files represents one set of data sent to the DMI. The left area of the emulator UI shows the scenario as defined by the user. The right area shows the scenario data encoded into messages according to the API (the API is documented in the next chapter) which are sent to the DMI. The TCP client asks for a new message every single second, which means that the lines in the scenario files are interpreted as 1 Hz samples of the environment.

Each line of the data in both scenario files shall be formatted using the following pattern:

```
name1  value1  name2  value2  ...  nameN  valueN
```

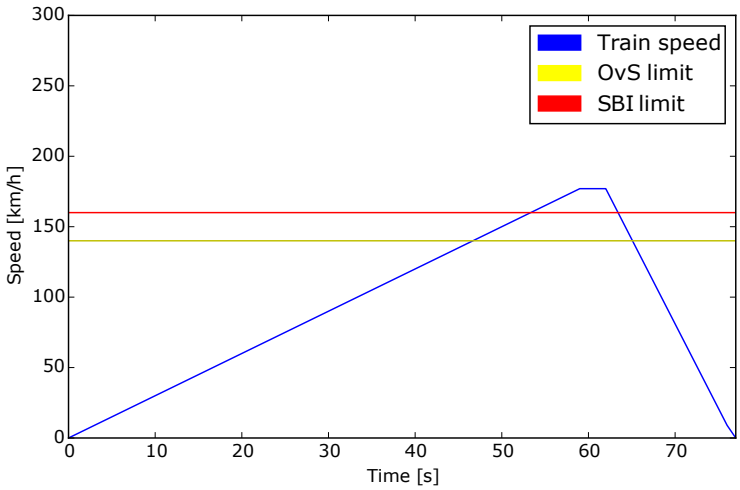
For example, a line `Vperm 100` would set the V_{perm} to 100 km/h.

6.2 Test Scenarios

Two scenarios have been created to verify the implemented functions of the DMI software.

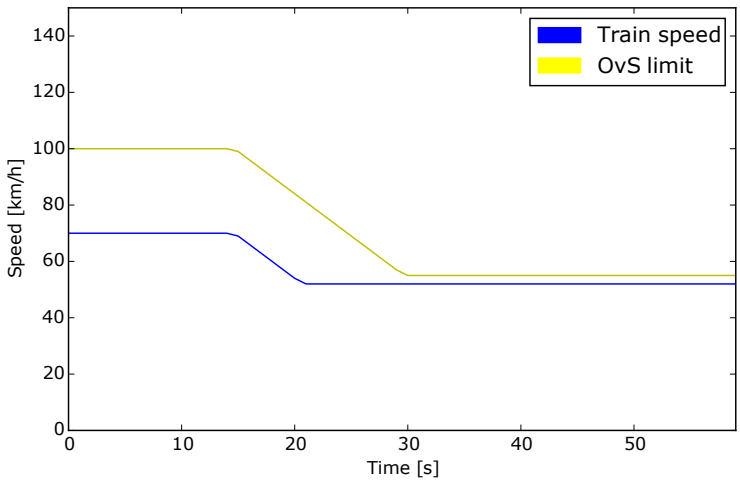
The first scenario, seen in the Figure 6.3, emulates a train during the CSM. The train accelerates and exceeds both the OvS and SBI limits, which triggers the service brake intervention resulting in deceleration of the train.

Figure 6.3: Scenario 1



The second scenario, displayed in the Figure 6.4, shows a train slowing down to a new target speed during the TSM. All the time, the train stays under the permitted speed limit and thus within the normal status supervision boundaries, so no warning or intervention is activated.

Figure 6.4: Scenario 2



Chapter 7

API Documentation

The purpose of the API is to establish the communication between the DMI and EVC. The DMI specification does not define how the communication should be implemented and thus the API was defined as part of the thesis after studying and analyzing the documents [11] and [5].

As covered in the previous chapters, the DMI and the EVC communicate with each other via TCP. The data sent over this connection are byte arrays with a defined structure. This documentation specify what data can be sent over the TCP connection and how to create messages carrying the data.

7.1 Data

The data contained inside the messages are generally `float`, `int`, or `string` data types carrying values associated to DMI objects or functions. The data can be grouped into the following categories:

1. Data carried by general messages
    - Data sent to the DMI
    - Data sent to the EVC
    - Bidirectional data
2. Data carried by planning area messages
    - Data sent to the DMI

The difference between the general and the planning area messages is described in the next section of the documentation. All the data items supported by the API are documented in the Appendix B.

The DMI application assumes that the data sent from the EVC are mutually consistent according to the documents [5] and [11]. If the consistency is not satisfied, the result will be a system failure in a form of an undefined behavior or a complete application crash.

Permitted speed being higher than the service brake intervention speed is an example of inconsistent data.

7.2 Messages

The data sent from or to the DMI are contained inside messages. The messages consists of a header and the data itself. Multiple data fields can be included in the data part of a message.

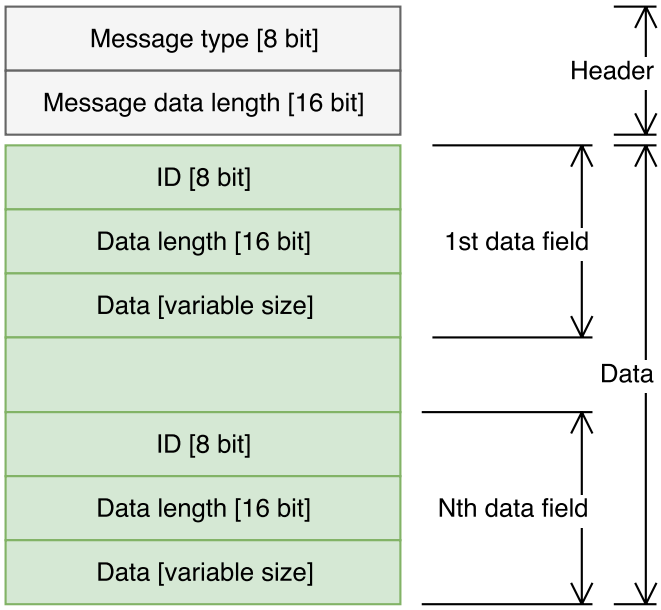
The API defines two types of messages: a general message and a planing area message. The following principles are valid for both message types:

1. Data field can carry `null` value. This is done by setting the length of the field to zero.
2. Any `float` or `int` are stored using little-endian format (least significant byte first).
3. Multiple data fields with the same data item ID can be stored inside one message.

7.2.1 General Message

General messages can be sent both from or to the DMI. They consist of a header and one or multiple data fields. General messages carry the data listed in the Appendix B.1.

Figure 7.1: General Message Structure

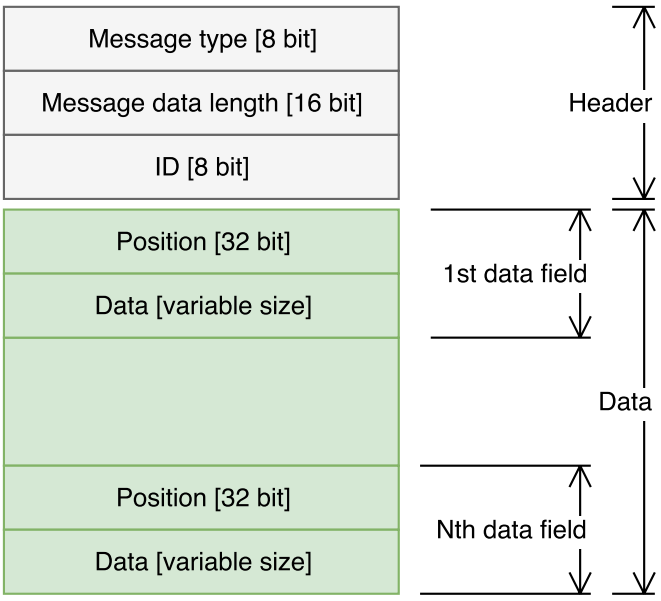


- **Header** identifies the type of the message and contains the information about the length of the data part. General messages use 0x01 as the message type value.
- **Data field** is composed of a unique ID (associating the data to a DMI object), length information and the data itself.

7.2.2 Planning Area Message

Planning area messages transport track profile data from the EVC to the DMI. They consist of a header identifying the type of profile data and one or multiple data points. Panning area messages can contain the data listed in the Appendix B.2.

Figure 7.2: Planning Area Message Structure



- **Header** identifies the type of the message and the type of the profile data. Planning area messages use the 0x00 type value. ID refers to either the speed discontinuity profile data, gradient profile, or track conditions. Furthermore, the header carries information about the length of the data part.
- **Data field** represents an $(X, f(X))$ data point. X is the distance from the initial train position, $f(X)$ is the data at that position. For example, $f(X)$ can represent the value of a MRSP speed discontinuity. The position value is defined in meters and stored as 32-bit integer. For the gradient and the speed discontinuity profile, the position refers to the end point of either a gradient segment or a speed limit.

7.2.3 Examples

First data field of the message bellow is sent from the EVC to the DMI and states that the current train speed is 120 km/h. The second data field sets the value of the release speed to `null`, which means that the release speed does not exist under current conditions.

Header		Data				
0x01	0x0038	0x01	0x0020	0x42F00000	0x05	0x0000
1	2	3	4	5	6	7

- 0x01 identifies the message as the general message.
- 0x0038 states that the data part is 56 bit long.
- 0x01 associates the data to the current train speed.
- 0x0020 means the data value is 32 bit.
- 0x42F00000 is an IEEE-754 float value equal to 120.
- 0x05 is an ID of the release speed.
- 0x0000 sets the release speed to `null`.

The next example shows a planning area message. This message has two data points. The first one orders 5000 m from the initial train position that the driver shall sound a horn. This order is represented by PL24 symbol. The second one announces change of traction to AC 25 kV at 7000 m. PL27 symbol is used for the announcement.

Header			Data			
0x00	0x0028	0x1A	0x00001388	0x14	0x00001B58	0x14
1	2	3	4	5	6	7

- 0x00 represents the planning area message.
- 0x0008 informs that the data part is 40 bit long.
- 0x01A associates the data to track condition profile.
- 0x00001388 is an integer equal to 5000.
- 0x14 identifies the PL24 symbol.
- 00001B58 is an integer equal to 7000.
- 0x17 stands for the PL27 symbol.

Conclusion

The thesis is focused on the DMI display of the ETCS on-board sub-system and the process of implementing it as a Microsoft .NET application. This task was selected because the Transportation Laboratory at the Czech Technical University in Prague is currently building a train vehicle simulator, which the DMI application will be part of.

The theoretical part of this text outlines the general features and characteristics of the ETCS in order to establish the environment in which the DMI operates. It then proceeds to describe the functions of the DMI itself.

The next part documents the research that had been done prior to the development of the DMI program. It introduces the target hardware, an AMiT AP9 panel computer. It also explores and analyzes the operating systems that could be deployed on the hardware, as well as frameworks suitable for the DMI development. This part concludes with the choice of Microsoft Windows 7 Embedded OS and Microsoft .NET framework. In retrospect, it is apparent that the .NET is not ideally suited for building real time applications, such as the DMI, which made the development sometime difficult.

The DMI implementation required a definition of an API for communication between the DMI and either the current EVC emulator or the future EVC unit of the simulator. The API defines the structure of the message and the data carried. The API covers all the functions of the DMI, even those which are not implemented in the current version of the application. It is also important to mention that the API might change in the future as the rest of the DMI will be completed.

The main goal of the thesis was to develop the DMI application. The speed monitoring function, comprising the train speed dial and the circular speed gauge, as well as the distance to the target and the planning area have been successfully implemented. Sub-level windows and supplementary information remains for future work. Alongside the DMI software, an EVC emulator has been built, and test scenarios have been designed to verify the implemented functions of the display.

Bibliography

[1] Petřík, L. *Functional Specification for a Driver's Cab Simulator with ETCS*. Master's thesis, Czech Technical University in Prague, 2016.

[2] UIC. *Driver Machine Interfaces for EMU/DMU, Locomotives and Driving Coaches*. 2009.

[3] Palumbo, M. The ERTMS/ETCS Signalling System. *railwaysignalling.eu*, 2015.

[4] UNISIG. *FFFIS for Eurobalise*. 2012.

[5] ERA UNISIG EEIG ERTMS Users Group. *ERTMS/ETCS System Requirements Specification*. 2016.

[6] UNISIG. *FFFIS for Euroloop*. 2012.

[7] UNISIG. *On-line Key Management FFFIS*. 2015.

[8] UNISIG. *FIS Juridical Recording*. 2014.

[9] UNISIG. *Train Interface*. 2014.

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

[11] European Union Agency for Railways. *ETCS Driver Machine Interface*. 2014.

[12] UNISIG. *Glossary of UNISIG Terms and Abbreviations*. 2000.

[13] AMiT. *AP9xxx programátorská příručka*. 2015.

[14] AMiT. *WINXPE – Filtry pro ochranu flash paměti*. 2012.

[15] Ghilardi, G. P. Emulate EWF Effects on Linux. [cited 2017-04-30]. Available from: `https://serverfault.com/questions/64035/emulate-ewf-effects-on-linux`

[16] Microsoft. Lifecycle FAQ – Windows Products. [cited 2017-03-02]. Available from: https://support.microsoft.com/en-us/help/18581/lifecycle-faq-windows-products

[17] Enos, K. Differences Between Windows XP Embedded and Windows XP Professional.

[18] Microsoft. *Windows Embedded Standard 7 Technical Overview.*

[19] The Qt Company. *Qt Examples and Tutorials.* [cited 2017-05-05]. Available from: https://doc.qt.io/qt-5/qtexamplesandtutorials

[20] Microsoft. *Overview of the .Net Framework.* [cited 2017-05-05]. Available from: https://msdn.microsoft.com/en-us/library/zw4w595w

[21] Unity Technologies. *Unity Manual.* [cited 2017-05-05]. Available from: https://docs.unity3d.com/Manual

Appendix A

Acronyms

A.1 ETCS Related

ATP Automatic train protection

BIU Brake interface unit

BTM Balise transmission module

DMI Driver machine interface

CSG Circular speed gauge

EOA End of authority

ERA European Union Agency for Railways

ERRI European Railway Research Institute

ERTMS European Railway Traffic Management System

ETCS European Train Control System

EVC European vital computer

FS Full supervision

GSM-R Global System for Mobile Communication - Railway

IndS Indication status

IntS Intervention status

ISL Indication supervision limit

KMC Key management center

LEU Lineside electronic unit

LTM Loop transmission module

MA Movement authority

MRSP Most restrictive speed profile

NoS Normal status

OvS Over-speed status

PASP Planning area speed profile

PSL Permitted speed limit

RBC Radio block center

SBI Service brake intervention

STM Specific transmission module

TSI Technical specification for interoperability

TIU Train interface unit

VBC Virtual balise cover

WaS Warning status

WSL Warning supervision limit

A.2 Other

API Application programming interface

COM Computer-on-module

CPU Central processing unit

EMU Electric multiple unit

FIFO First in, first out

GPU Graphics processing unit

GUI Graphical user interface

IDE Integrated development environment

OS Operating system

RAM Random access memory

TCP Transmission Control Protocol

UART Universal asynchronous receiver/transmitter

UI User interface

URI Uniform resource identifier

WPF Windows Presentation Foundation

XAML Extensible Application Markup Language

Appendix B

Message Data

Data that can be carried via the messages between the DMI and EVC are listed in this appendix.

B.1 General Data

B.1.1 Data Sent to the DMI

Table B.1: Train Speed

Name	Vtrain
ID	0x01
Length	32 bit
Nullable	No
Data	Any 32-bit float representing a valid speed

Table B.2: Permitted Speed

Name	Vperm
ID	0x02
Length	32 bit
Nullable	Yes
Data	Any 32-bit float representing a valid speed

Table B.3: Target Speed

Name	Vtarget
ID	0x03
Length	32 bit
Nullable	Yes
Data	Any 32-bit float representing a valid speed

Table B.4: SBI Speed

Name	VSBI
ID	0x04
Length	32 bit
Nullable	Yes
Data	Any 32-bit float representing a valid speed

Table B.5: Release Speed

Name	Vrel
ID	0x05
Length	32 bit
Nullable	Yes
Data	Any 32-bit float representing a valid speed

Table B.6: LSSMA

Name	VLSSMA
ID	0x06
Length	32 bit
Nullable	Yes
Data	Any 32-bit float representing a valid speed

Table B.7: Distance to Target

Name	Dtarget
ID	0x0A
Length	32 bit
Nullable	Yes
Data	Any 32-bit float representing a valid distance

Table B.8: Distance from Start

Name	Dstart
ID	0x0B
Length	32 bit
Nullable	Yes
Data	Any 32-bit float representing a valid distance

Table B.9: Supervision Section

Name	Supervision1	
ID	0x07	
Length	24 bit	
Nullable	Yes	
Value list	0x43534D	Ceiling speed monitoring
	0x50494D	Pre-indication monitoring
	0x54534D	Target speed monitoring
	0x52534D	Release speed monitoring

Table B.10: Supervision Status

Name	Supervision2	
ID	0x08	
Length	24 bit	
Nullable	Yes	
Value list	0x4E6F53	Normal status
	0x496E6453	Indication status
	0x4F7653	Over-speed status
	0x576153	Warning status
	0x496E6453	Intervention status

Table B.11: Mode

Variable	Mode	
ID	0x09	
Length	8 bit	
Nullable	Yes	
Value list	0x4653	Full supervision
	0x4C53	Limited supervision
	0x4F53	On sight
	0x5352	Staff responsible
	0x5348	Shunting
	0x554E	Unfitted
	0x5053	Passive shunting
	0x534C	Sleeping
	0x5342	Stand by
	0x5452	Trip
	0x5054	Post trip
	0x5346	System failure
	0x4E50	No power
	0x4E4C	Non leading
	0x534E	STM national
	0x5256	Reversing

Table B.12: ETCS Level

Name	ETCSLevel	
ID	0x0C	
Length	8 bit	
Nullable	No	
Value list	0x30	Level 0
	0x31	level 1
	0x32	Level 2
	0x33	Level 3
	0x34	Level STM

Table B.13: Acknowledgment

Name	Ack
ID	0x0D
Length	8 bit
Nullable	No
Data	Any value representing a valid data item ID

Table B.14: Track Ahead Free Indication

Name	TrackAheadFree
ID	0x0E
Length	8 bit
Nullable	No
Data	Boolean

Table B.15: Text Messages

Name	TextMsg
ID	0x0F
Length	Variable
Nullable	No
Data	Any character string repressing a valid message

Table B.16: Override Active

Name	OverrideActive
ID	0x10
Length	8 bit
Nullable	No
Data	Boolean

Table B.17: Adhesion Factor

Name	AdhFactor
ID	0x11
Length	8 bit
Nullable	No
Data	Boolean

Table B.18: Tunnel Stopping Area

Name	TunnelSA
ID	0x12
Length	32 bit
Nullable	Yes
Data	Any 32-bit float representing a valid distance

Table B.19: Track Conditions Symbols

Name	TrackCond			
ID	0x14			
Length	8 bit			
Nullable	No			
Value list	0x00	TC19	0x12	TC15
	0x01	TC21	0x13	TC16
	0x02	TC20	0x14	TC17
	0x03	TC22	0x15	TC18
	0x04	TC02	0x16	TC23
	0x05	TC03	0x17	TC24
	0x06	TC01	0x18	TC25
	0x07	TC04	0x19	TC26
	0x08	TC05	0x1A	TC27
	0x09	TC06	0x1B	TC28
	0x0A	TC07	0x1C	TC29
	0x0B	TC08	0x1D	TC30
	0x0C	TC09	0x1E	TC31
	0x0D	TC10	0x1F	TC32
	0x0E	TC11	0x20	TC33
	0x0F	TC12	0x21	TC34
	0x10	TC13	0x22	TC35
	0x11	TC14	0x23	LX01

Table B.20: Indication Marker

Name	IndicationMarker	
ID	0x13	
Length	32 bit	
Nullable	Yes	
Data	Any 32-bit float representing a valid distance	

Table B.21: Radio Connection

Name	TrackCond	
ID	0x15	
Length	8 bit	
Nullable	No	
Value list	0x00	No connection
	0x01	Connection up
	0x02	Connection lost/set-up failed

Table B.22: Reversing Permitted Indication

Name	ReversePerm
ID	0x16
Length	8 bit
Nullable	No
Data	Boolean

Table B.23: Geographical Position

Name	GeoPos
ID	0x17
Length	32 bit
Nullable	Yes
Data	Any 32-bit int representing a valid position

Table B.24: System Version

Name	SysVer
ID	0x2B
Length	Variable
Nullable	No
Data	Any character string representing a valid version

Table B.25: Time to Indication

Name	TTI
ID	0x2D
Length	32 bit
Nullable	Yes
Data	Any 32-bit int representing a valid time

Table B.26: Track Condition Deactivation

Name	TrackCondDe
ID	0x2C
Length	Variable
Nullable	No
Data	Same as B.19

B.1.2 Data Sent from the DMI

Table B.27: Driver ID

Name	DriverID
ID	0x1C
Length	32 bit
Nullable	No
Data	Any 32-bit int representing a valid driver ID

Table B.28: Train Running Number

Name	TrainNo
ID	0x1D
Length	32 bit
Nullable	No
Data	Any 32-bit int representing a valid running number

Table B.29: General Commands

Name	GenCommand	
ID	0x1B	
Length	8 bit	
Nullable	No	
Value list	0x00	Start
	0x01	Shunting
	0x02	Exit shunting
	0x03	Maintain shunting
	0x04	Override

Table B.30: SR Speed

Name	VSR
ID	0x1F
Length	32 bit
Nullable	No
Data	Any 32-bit float representing a valid speed

Table B.31: SR Distance

Name	DSR
ID	0x20
Length	32 bit
Nullable	No
Data	Any 32-bit float representing a valid distance

Table B.32: Set VBC

Name	VBC
ID	0x21
Length	32 bit
Nullable	No
Data	Any 32-bit int representing a valid VBC code

Table B.33: Remove VBC

Name	VBC
ID	0x22
Length	32 bit
Nullable	No
Data	Any 32-bit int representing a valid VBC code

Table B.34: Radio Network ID

Name	RadioID
ID	0x23
Length	*variable* bit
Nullable	No
Data	Any character string representing a valid GSM-R ID

Table B.35: RBC ID

Name	RBCID
ID	0x24
Length	32 bit
Nullable	No
Data	Any 32-bit int representing a valid RBC ID

Table B.36: RBC Phone Number

Name	RBC Phone Number
ID	0x24
Length	Variable
Nullable	No
Data	Any string representing a valid phone number

B.1.3 Bidirectional Data

Table B.37: Train Type

Name	TrainType
ID	0x25
Length	Variable
Nullable	No
Data	Any valid character string

Table B.38: Train Length

Name	TrainLength
ID	0x26
Length	32 bit
Nullable	No
Data	Any 32-bit integer representing a valid length

Table B.39: Brake Percentage

Name	BrakePrcnt
ID	0x28
Length	32 bit
Nullable	No
Data	Any 32-bit float representing a valid brake percentage

Table B.40: Maximum speed

Name	Vmax
ID	0x29
Length	32 bit
Nullable	No
Data	Any 32-bit float representing a valid speed

Table B.41: Airtight

Name	Airtight
ID	0x2A
Length	8 bit
Nullable	No
Data	Boolean

Table B.42: Loading Gauge

Name	LoadG	
ID	0x2A	
Length	8 bit	
Nullable	No	
Value list	0x00	Out of GC
	0x01	G1
	0x02	GA
	0x03	GB
	0x04	GC

Table B.43: Train Category

Name	TrainCat			
ID	0x2A			
Length	8 bit			
Nullable	No			
Value list	0x00	PASS1	0x12	TILT7
	0x01	PASS2	0x13	FP1
	0x02	PASS3	0x14	FP2
	0x03	TILT1	0x15	FP3
	0x04	TILT2	0x16	FP4
	0x05	TILT3	0x17	FG1
	0x06	TILT4	0x18	FG2
	0x07	TILT5	0x19	FG3
	0x08	TILT6	0x1A	FG4

Table B.44: Axle Load Category

Name	ALC
ID	0x2A
Length	8 bit
Nullable	No
Data	Undefined

B.2 Planning Area Data

Table B.45: Track Conditions

Name	TrackCondProfile			
ID	0x1A			
Length	8 bit			
Nullable	No			
Value list	0x00	PL17	0x10	PL13
	0x01	PL19	0x11	PL14
	0x02	PL18	0x12	PL15
	0x03	PL20	0x13	PL16
	0x04	PL01	0x14	PL24
	0x05	PL02	0x15	PL25
	0x06	PL03	0x16	PL26
	0x07	PL04	0x17	PL27
	0x08	PL05	0x18	PL28
	0x09	PL06	0x19	PL29
	0x0A	PL07	0x1A	PL30
	0x0B	PL08	0x1B	PL31
	0x0C	PL09	0x1C	PL32
	0x0D	PL10	0x1D	PL33
	0x0E	PL11	0x0F	PL12

Table B.46: Gradient Profile

Name	GradientProfile
ID	0x18
Length	32 bit
Nullable	No
Data	Any 32-bit float representing a valid gradient

Table B.47: Speed Profile

Name	SpeedProfile
ID	0x19
Length	32 bit
Nullable	No
Data	Any 32-bit float representing a speed

B.3 Notes

Table B.48: Units

Physical quantity	Unit
Length	meters
Time	seconds
Speed	km/h

Appendix C

Source Code

The DMI application and its source code is available at the Transportation Laboratory of the Faculty of Transportation Sciences.