FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	A software solution for user-friendly reading and visualisations of machine messages from CAN Trace data logs
Student:	Ondřej Luks
Supervisor:	doc. Ing. Robert Pergl, Ph.D.
Study program:	Informatics
Branch / specialization:	Information Systems and Management
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2024/2025

Instructions

1) Perform review of the necessary theoretical foundations and possible similar applications.
2) Analyze the current functioning of company processes and propose improvements supported by the application.
3) Design and implement a desktop application for converting data from MF4 log files into relational database. The application should consist of separate backend and GUI application for configuration and control, communicating together. Implement the solution in Python.
4) Create a dashboard in Grafana for viewing the stored logs from the relational database.
5) Test the solution properly.
6) Demonstrate the solution on a case study.
7) Discuss the benefits and limitations of the application in the context of the company.

Electronically approved by Ing. David Buchtela, Ph.D. on 19 June 2023 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

A software solution for user-friendly reading and visualisations of machine messages from CAN Trace data logs

Ondřej Luks

Department of Software Engineering
Supervisor: doc. Ing. Robert Pergl, Ph.D.

May 16, 2024

Acknowledgements

I would like to extend my sincere gratitude to my supervisor, doc. Ing. Robert Pergl, Ph.D., for his invaluable guidance, assistance, and patience throughout the development of this thesis. Additionally, I wish to express my appreciation to my family, girlfriend, friends, and colleagues for their enduring support and encouragement. I am also grateful to Ing. Michal Karas for providing me with the opportunity to undertake this work. Furthermore, I extend my thanks to Ing. Peter Abraham for his valuable insights and knowledge that he shared with me.

Declaration

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

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

In Prague on May 16, 2024

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Luks, Ondřej. *A software solution for user-friendly reading and visualisations of machine messages from CAN Trace data logs.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Abstract

This bachelor's thesis delves into the process of handling and inspecting data from construction machines utilizing the CAN protocol at Doosan Bobcat EMEA, s.r.o. company. The current state of the process is thoroughly analysed, identifying associated problems and deficiencies. Based on the analysis of prospective solutions, followed by its subsequent design and implementation, a new Python desktop application with a graphical user interface is developed, coupled with dynamic data inspection dashboards created in Grafana. This novel solution saves the company up to 2 days and 5.75 hours of time with each execution of this process.

Keywords desktop app, GUI, MF4, DBC, data vizualization, Grafana, SAE J1939-TP, PostgreSQL, Python

Abstrakt

Tato bakalářská práce se zabývá procesem zpracování a inspekce dat ze stavebních strojů využívajících protokol CAN ve společnosti Doosan Bobcat EMEA, s.r.o. Současný stav procesu je důkladně analyzován s cílem identifikovat související problémy a nedostatky. Na základě analýzy možných řešení, následované jejich návrhem a implementací, je v Pythonu vyvinuta nová desktopová aplikace s grafickým uživatelským rozhraním, doplněná o dynamické nástěnky pro inspekci dat vytvořené v Grafaně. Toto nové řešení společnosti ušetří až 2 dny a 5,75 hodiny času při každém provedení tohoto procesu.

Klíčová slova desktopová aplikace, GUI, MF4, DBC, vizualizace dat, Grafana, SAE J1939-TP, PostgreSQL, Python

Contents

Introduction **1**

Goals and methodology **3**

1 Theoretical foundation **5**
1.1 CAN bus . . . 5
1.1.1 General information . . . 5
1.1.2 Messaging . . . 5
1.1.3 SAE J1939 Transport Protocol . . . 6
1.1.4 DBC file . . . 6
1.2 ASAM MDF . . . 6
1.2.1 General information . . . 7
1.2.2 Key features . . . 7
1.3 CANedge2 . . . 7
1.4 Grafana . . . 7
1.4.1 General information . . . 8
1.4.2 Grafana Dashboard . . . 8
1.4.3 Deployment and usage . . . 8

2 Analysis of the current state **9**
2.1 Basic information . . . 9
2.2 Detailed description of the process . . . 9
2.2.1 From the machine to Grafana . . . 10
2.2.2 Current activity diagrams . . . 10
2.2.3 Resources . . . 13
2.3 Used technologies . . . 14
2.4 Currently implemented requirements analysis . . . 14
2.4.1 Current functional requirements . . . 14
2.4.2 Current nonfunctional requirements . . . 15
2.5 Currently implemented Use case model . . . 15
2.5.1 Actors . . . 15
2.5.2 Conversion and upload of MF4 files into the database . . . 16
2.5.3 Data inspection . . . 17
2.6 Summary . . . 19

3 Analysis of the future solution **23**
3.1 To-Be requirements analysis . . . 23
3.1.1 Functional requirements . . . 23
3.1.2 Nonfunctional requirements . . . 24
3.2 To-Be Activity diagrams . . . 25
3.2.1 Data processing . . . 25
3.2.2 Data downloading . . . 25
3.2.3 Data inspection . . . 25
3.3 To-Be Use case model . . . 25
3.3.1 Actors . . . 25
3.3.2 Basic application operations . . . 26
3.3.3 Converting and uploading MF4 data into the database . 27
3.3.4 Database configuration . . . 30
3.3.5 Data downloading . . . 31
3.3.6 Creating Grafana dashboard JSON . . . 32
3.3.7 Data inspection . . . 33

4 Design **35**
4.1 Proposal of the new solution . . . 35
4.2 Architecture . . . 36
4.2.1 Domain model . . . 36
4.2.2 Architectural style . . . 37
4.2.3 Security architecture . . . 38
4.2.4 Technology stack . . . 38
4.2.5 Deployment . . . 41
4.2.6 Documentation . . . 42
4.3 User experience . . . 42
4.3.1 User roles . . . 42
4.3.2 Navigation and flow . . . 42
4.3.3 Performance . . . 43
4.3.4 Error handling . . . 43
4.3.5 Accessibility . . . 44
4.4 Graphical user interface . . . 44
4.4.1 Typography and colours . . . 44
4.4.2 Responsiveness . . . 44
4.4.3 Wireframe . . . 45

5 Implementation and testing **47**
5.1 Implementation . . . 47
5.2 Unit testing . . . 48
5.3 User testing . . . 48
5.4 Stress testing . . . 48

6 Case study **51**
6.1 Performed case study . . . 51

7 Evaluation **53**
7.1 Improvements and benefits . . . 53
7.2 Drawbacks . . . 54
7.3 Plans for future development . . . 54

Conclusion **55**

Bibliography **57**

A Abbreviations **59**

B To-Be activity diagrams **61**

C To-Be use case diagrams **65**

D Contents of attachments **71**

List of Figures

2.1 Current activity diagram for data retrieval subprocess 11
2.2 Current activity diagram for data processing subprocess 12
2.3 Current activity diagram for data inspection subprocess 13
2.4 Current actors 16
2.5 Current diagram of UC1, UC2 and UC3 19
2.6 Current diagram of UC4 19
2.7 Current diagram of UC5 20
2.8 Current diagram of UC6, UC7 and UC8 20

3.1 To-Be actors 26

4.1 System domain model design 37
4.2 Wireframe of the main window 45

B.1 To-Be activity diagram for data processing subprocess 62
B.2 To-Be activity diagram for data downloading subprocess 63
B.3 To-Be activity diagram for data inspection subprocess 64

C.1 To-Be use case diagram for Basic application operations 66
C.2 To-Be use case diagram for Converting and uploading of MF4 data 67
C.3 To-Be use case diagram for Database configuration 68
C.4 To-Be use case diagram for Data downloading 69
C.5 To-Be use case diagram for Creating Grafana dashboard JSON . . 69
C.6 To-Be use case diagram for Data inspection 70

Introduction

In today's dynamic business environment, employees often find themselves confronted with problems that fall outside their job scope, yet hinder their task completion. Consequently, unqualified workers resort to implementing new *temporary* solutions to meet deadlines. However, as one might predict, nothing is as enduring as a temporary fix. It comes as no surprise that these makeshift remedies are not very effective; they prove to be wasting valuable company time due to their poor design, complexity and lack of functionality.

That being said, companies – especially larger ones, where supervision may be insufficient – tend to accumulate these formerly temporary processes, further adding to the complexity of the colossal clockwork. But inefficiency and time waste are not the only consequences of this detrimental practice.

In businesses operating within the digital realms, hastily stitching together software without proper documentation or adherence to organizational culture can lead to significant repercussions. In a few years, employees may find themselves in the dark about how the program operates. Therefore, while these temporary measures may offer short-term relief, they ultimately contribute to long-term problems and undermine the company's ability to thrive in a competitive marketplace.

This thesis will take a closer look at a specific process established as a temporary measure, that has been utilized for over three years, and endeavour to optimize this process with the aim of improving efficiency and effectiveness. It concerns the handling and viewing of data collected from construction machines by Test engineers and Design engineers in Doosan Bobcat EMEA, s.r.o. company.

Goals and methodology

The primary goal of this thesis is to enhance the company's process for handling and viewing collected data from construction machines that utilize the CAN bus. These data originate from real-world tests and evaluate the machine's behaviour during these tests.

The theoretical segment will acquaint the reader with the foundational technological aspects of the CAN bus protocol and MDF file format DBC file format, CANedge2 device, as well as with an open-source data inspection tool named Grafana.

Furthermore, the practical part will concentrate on analysing and describing the current state of the process.

Drawing from the gathered information, this thesis will propose a novel solution to enhance the aforementioned process. Building upon this proposal, a new GUI desktop application will be designed, implemented, and tested. The outcomes will be showcased through a case study, demonstrating the extent of improvement achieved by the new solution.

The development of the application will utilize agile methodologies, ensuring flexibility and adaptability throughout the work. Additionally, Git will be employed for version control, enabling seamless management of code changes. This approach ensures that the development process remains organized, transparent, and conducive to rapid iteration and improvements.

Firstly in the scope of this thesis, Chapter 1 will briefly describe CAN protocol, as being one of the technological foundations of this process. All machines in the company utilize CAN for communication between controllers, hence the data collected from the machines is in the format of CAN messages. Next, ASAM MDF files will be introduced – as being the vessel for storing CAN messages – together with CANedge2 device recording them. Lastly in the theoretical part, this thesis will describe the basics of an open-source data visualization tool named Grafana.

Analysis of the current process will open up the practical part with Chapter 2, providing a detailed description of the problem, alongside with currently used technologies, requirements and use cases. Subsequent to this chapter, an analysis of the future solution will be conducted in Chapter 3. Next, Chapter 4 will introduce a design for a novel solution, revealing the conceptualization of a new desktop application and delineating its architectural framework. This will be followed by Chapter 5 with an implementation phase, wherein the proposed solution will be developed and tested to ensure its stability and robustness.

Followed by Chapter 6, a case study will be conducted, showcasing real-use application scenarios and evaluating the performance of the new solution. Finally, Chapter 7 will carry out an in-depth evaluation, comparing the new solution against existing benchmarks and assessing its impact on efficiency, productivity, and user experience. Concluding remarks will summarize key findings, highlight strengths and limitations, and propose avenues for future research and improvement.

Chapter 1

Theoretical foundation

1.1 CAN bus

All machines produced by Doosan Bobcat EMEA s.r.o. are equipped with a standardized automotive communication system known as the *CAN bus*, ensuring seamless integration between various controllers.

1.1.1 General information

The Controller Area Network (CAN) is a message-based protocol designed to allow the Electronic Control Units (ECUs) found in today's automobiles, as well as other devices, to communicate with each other in a reliable, priority-driven fashion. Messages, or "frames", are received by all devices in the network, which does not require a host computer. CAN is supported by a rich set of international standards under ISO 11898 [1].

1.1.2 Messaging

Devices on a CAN bus are called "nodes". Each node consists of a CPU, CAN controller, and a transceiver, which adapts the signal levels of both data sent and received by the node. All nodes can send and receive data, but not at the same time [1].

Nodes cannot send data directly to each other. Instead, they send their data onto the network, where it is available to any node to which it has been addressed. The CAN protocol is lossless, employing a bitwise arbitration method to resolve contentions on the bus [1].

All of the nodes are synchronized so that they all sample data on the network simultaneously. However, data is not transmitted with clock (time) data, so CAN is not truly a synchronous bus, such as EtherCAT, for example [1].

With CAN, all data is sent in frames, and there are four types:

- Data frames transfer data to either a single recipient or multiple receiver nodes, accommodating a maximum size of 8 bytes per frame.
- Remote frames ask for data from other nodes.
- Error frames report errors.
- Overload frames report overload conditions [1].

1.1.3 SAE J1939 Transport Protocol

Messages greater than 8 bytes in length are too large to fit into a single CAN Data Frame. Therefore they must be divided by the sender into individual packets, which can then be sent with a CAN message each. The receiver has to recombine the individual fragments in their original order. A set of rules is defined for this in the J1939 standard: a so-called Transport protocol [2].

Two types of the J1939 TP exist:

1. The CM (Connection Mode) intended for a specific device
2. The BAM (Broadcast Announce Message) intended for the entire network [3]

For example, a transmitting ECU may send an initial BAM packet to set up a data transfer. The BAM specifies the identifier for the multi-packet message as well as the number of databytes and packets to be sent. It is then followed by up to 255 packets/frames of data. Each of the 255 packets uses the first databyte to specify the sequence number (1–255), followed by 7 bytes of data. The max number of bytes per multi-packet message is therefore $8 \cdot 255 = 1,785$ bytes. In post processing, a conversion software tool can reassemble the multiple entries of 7 databytes into a single payload and handle it according to the multi-packet specifications as found in e.g. a DBC file [3].

1.1.4 DBC file

A DBC (CAN Database) file is a standardized format to define the communication parameters and message structures of a CAN bus. It serves as a *dictionary or mapping file* that translates raw CAN bus data into human-readable signals and parameters. DBC files contain information such as message identifiers, signal names, scaling factors, signal offsets, and signal data types, enabling software tools to interpret and process the data transmitted over the CAN bus [3].

1.2 ASAM MDF

Another important technological aspect of the company process is dealing with ASAM MDF files.

1.2.1 General information

MDF (Measurement Data Format) is a *binary file* format to store recorded or calculated data for post-measurement processing, off-line evaluation or long-term storage. The format has become a de-facto standard for measurement and calibration systems, but is also used in many other application areas [4].

As a compact binary format, ASAM MDF offers efficient and high performance storage of huge amounts of measurement data. MDF is organized in loosely coupled binary blocks for flexible and high performance writing and reading. Fast index-based access to each sample can be achieved by loss-free re-organization (i.e. sorting) of the data. Distributed data blocks even make it possible to directly write sorted MDF files. The file format allows storage of raw measurement values and corresponding conversion formulas; therefore raw data can still be interpreted correctly and evaluated by post-processing tools [4].

1.2.2 Key features

ASAM MDF offers efficient storage for vast amounts of measurement data, especially with compression options since ASAM MDF 4.1.0. (MF4). Utilizing a 64-bit data type, it overcomes the limitations of previous versions, promising virtually unlimited file and measurement data size [5].

High data rates for reading and writing make it suitable for real-time applications, facilitating direct writing of raw values from an ECU and efficient storage of fixed-formatted data chunks. Tools vary in complexity, with simpler loggers supporting unsorted writing, while advanced tools enable direct sorted writing, minimizing the need for sorting before reading [5].

Furthermore, it supports partial file reading, allowing access to descriptive information without loading signal data entirely, which is crucial for large datasets. It accommodates multiple and non-periodic sampling rates, synchronized via a master channel concept, and offers flexibility in recording based on time, angle, or distance. Besides storing data and descriptions within a single file. Primarily designed for automotive use, it includes specialized data types and features tailored for automotive applications, and enjoys wide industry adoption with support from leading tools [5].

1.3 CANedge2

For inspection and testing purposes, transmitted CAN messages between nodes need to be recorded and stored.

The CANedge2 data logger, developed by CSS Electronics, is a plug-and-play device designed for the *simultaneous recording* of timestamped CAN data onto an SD card. It seamlessly saves the recorded data into MF4 files for efficient storage and retrieval [6].

1.4 Grafana

Lastly, it is needed to introduce an open-source *data inspection tool* named Grafana by Grafana Labs, which is also being used in the company process.

1.4.1 General information

Grafana is a powerful data visualization and monitoring tool that enables users to create insightful dashboards and graphs for analysing and understanding metrics from diverse data sources. Renowned for its flexibility and ease of use, Grafana supports a wide range of data types and integrations, including databases, cloud services, and even IoT devices [7].

With its intuitive interface and customizable features, users can effortlessly build dynamic visualizations, set up alerts for anomalies, and collaborate with team members to derive valuable insights from their data. Whether used for performance monitoring, infrastructure analysis, or business intelligence, Grafana provides organizations with the tools to effectively utilize their data for informed decision-making [7].

1.4.2 Grafana Dashboard

Grafana dashboards serve as dynamic interfaces for visualizing and analysing data collected from a variety of sources. At their core, these dashboards are composed of *panels*, each representing a different aspect of the data. Users configure queries to fetch specific information from connected data sources, such as databases or monitoring systems. Once the data is retrieved, Grafana offers a range of visualization options, including graphs, gauges, tables, and heat maps. Users can customize these visualizations to suit their needs, adjusting parameters such as time ranges, dynamic filters, and thresholds [8].

The arrangement of panels on the dashboard is entirely customizable, allowing users to create layouts that best present their data. Panels can be grouped together to highlight relationships between different metrics or organized to provide a comprehensive overview of a particular system or process. Grafana also supports interactivity, enabling users to zoom in on specific time periods, toggle between different metrics, or drill down into underlying data for deeper analysis. This flexibility empowers users to explore their data in real-time and derive meaningful insights [9].

A dashboard in Grafana is represented by a JSON object, which stores its metadata. Dashboard metadata includes dashboard properties, metadata from panels, template variables, panel queries, etc. Consequently, users possess the capability to export or import these JSON objects, facilitating the publication or creation of various dashboards [10].

1.4.3 Deployment and usage

Installing Grafana on a local server is a straightforward process when using a supported operating system and compatible database. The executable necessary for installation can be easily obtained from the Grafana Labs website. Following successful installation and configuration, the Grafana server can be deployed, offering a user-friendly interface accessible via web browsers [11].

Grafana's flexible user hierarchy further enhances its utility, allowing administrators to define user roles and permissions to regulate access levels. That ensures users to efficiently collaborate on dashboard creation and analysis while maintaining data security and integrity [12][13].

Chapter 2

Analysis of the current state

Efficiency is critical for organizational success. This chapter conducts a comprehensive analysis of the existing company process to identify areas for improvement. Through process mapping, stakeholder interviews, and observation, it aims to uncover inefficiencies and bottlenecks within the work flow.

This analysis serves as a foundation for identifying opportunities for optimization, enhancing organizational performance.

2.1 Basic information

At the beginning of the year 2019, an initiative was launched within the Embedded Software team of Doosan Bobcat EMEA s.r.o. company to create a new tool for visualizing diagnostic data downloaded from machines, as the tools being used at that time were inadequate.

A former employee took on the solution, seeing the project as a good opportunity to learn *Python* programming language. His initial functional prototype only worked within Python and Jupyter notebooks, where machine data (MF4 files) was converted with the help of DBC files from binary to text format and plotted using the Python library Pyplot. The script only worked for one specific machine and any modifications required code intervention.

Later, it was found that such an approach was insufficient for larger data volumes, so the program was enhanced with *another script* that uploaded the converted data to a local PostgreSQL database. In addition to the database, a Grafana server was also running on the computer for simple visualization.

This brings us closer to the current state of the solution. The scripts underwent several modifications to store more detailed data, but the core remained the same – two Python scripts executed sequentially, a PostgreSQL database, and Grafana. There is also no support for SAE J1939-TP type of messages.

Everything is set up at a virtual Windows desktop running within the company intranet, accessible to all employees.

2.2 Detailed description of the process

This section delivers a detailed examination of the current process, offering an in-depth understanding of its intricacies and dynamics.

2.2.1 From the machine to Grafana

The process begins with *Worker 1* approaching the machine and removing the SD card from the CANedge2 data logger. This typically occurs once or twice a week for all necessary operating machines, ranging from 5 to 10 in number. *Worker 1* then returns to the office with the SD card and downloads the data, which he then packages into a ZIP archive and sends to *Worker 2*, as only he is proficient in Python scripting.

Worker 2 downloads, moves, and extracts the ZIP archive onto the company's virtual desktop computer where the scripts are located. He either places the files into a specific folder from which the script can automatically retrieve them, or he must rewrite the file paths in the code. Additionally, he must provide the scripts with properly configured DBC files using the same method as the files from the SD card. Once everything is prepared, *Worker 2* can execute the first script, which converts and saves the binary MF4 files into Excel spreadsheets using the DBC files. Each MF4 file corresponds to one .xlsx file, with each signal occupying one column in the respective file. The data is always sorted by timestamp. Upon completion of the first script, *Worker 2* must configure and execute the second script, responsible for uploading all generated Excel files into the PostgreSQL database. The database connection configuration needs to be entered into a text file that the script reads. *Worker 2* is responsible for setting up the database with a specific schema for each machine and properly adjusting everything in the text file.

After the second script finishes running, all the converted data is inside the database. Several static dashboards have been created in Grafana, which automatically pull and display data through queries. Thanks to Grafana's web user interface, all company employees have access to the data if they are connected to the intranet and have the correct login credentials. If someone wishes to display different signals or expand the CAN signal bus, it is needed to manually modify or create a new dashboard. That responsibility would most likely fall on *Worker 2* again, as there are very few employees proficient in administration of Grafana in the company.

2.2.2 Current activity diagrams

Following figures show the forementioned process transformed into activity diagrams. Figures are also included as attachments.

2.2.2.1 Data retrieval

Figure 2.1 illustrates the activity diagram of the subprocess, which consists of downloading data from the machine to the computer.

2.2.2.2 Data processing

Figure 2.2 illustrates the activity diagram of the subprocess, which consists of handling, processing and uploading the data to the database.

Figure 2.1: Current activity diagram for data retrieval subprocess

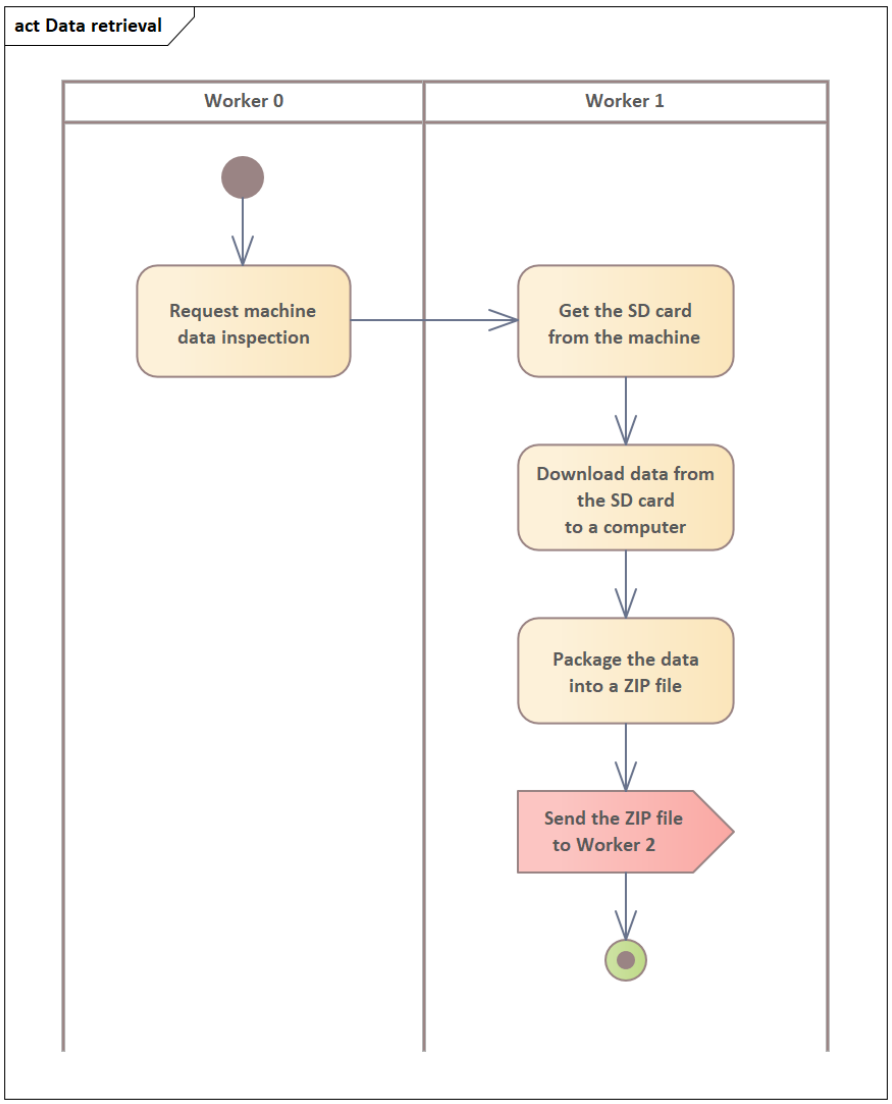


Figure 2.2: Current activity diagram for data processing subprocess

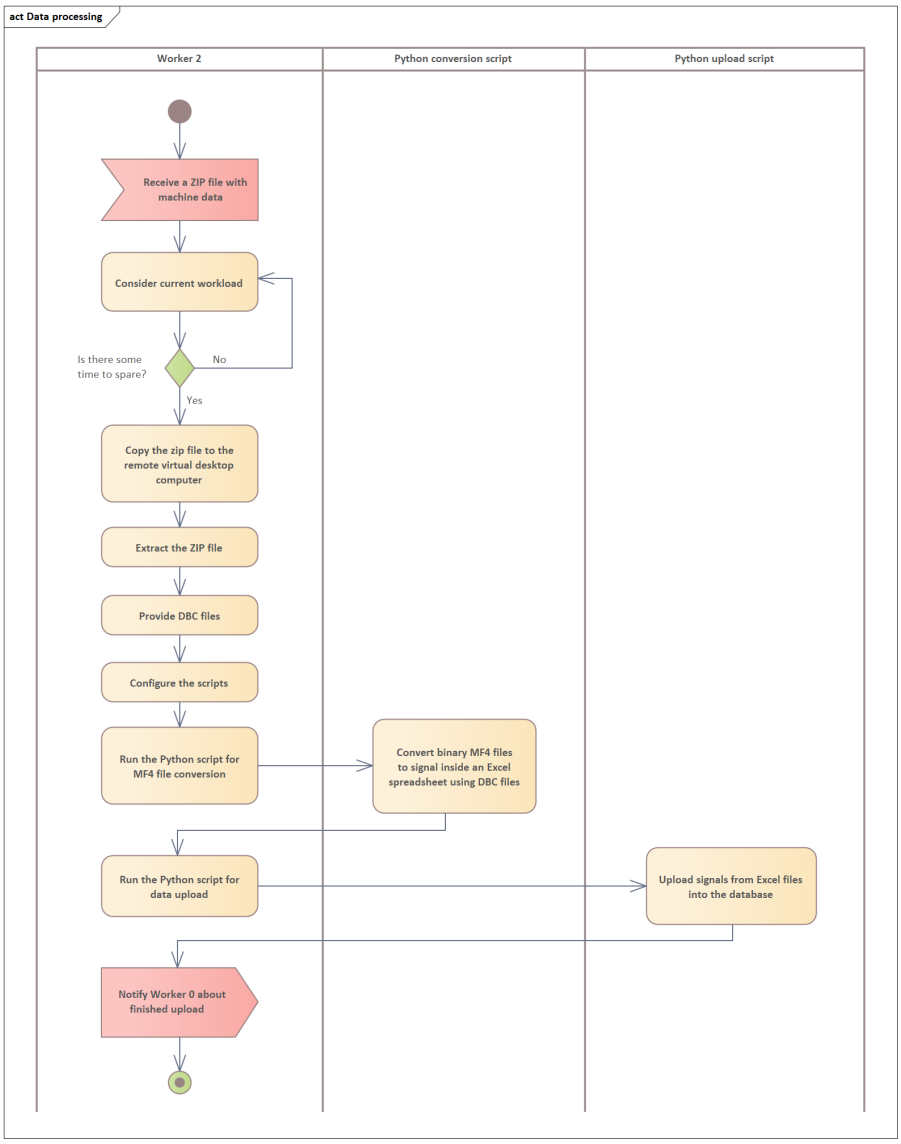
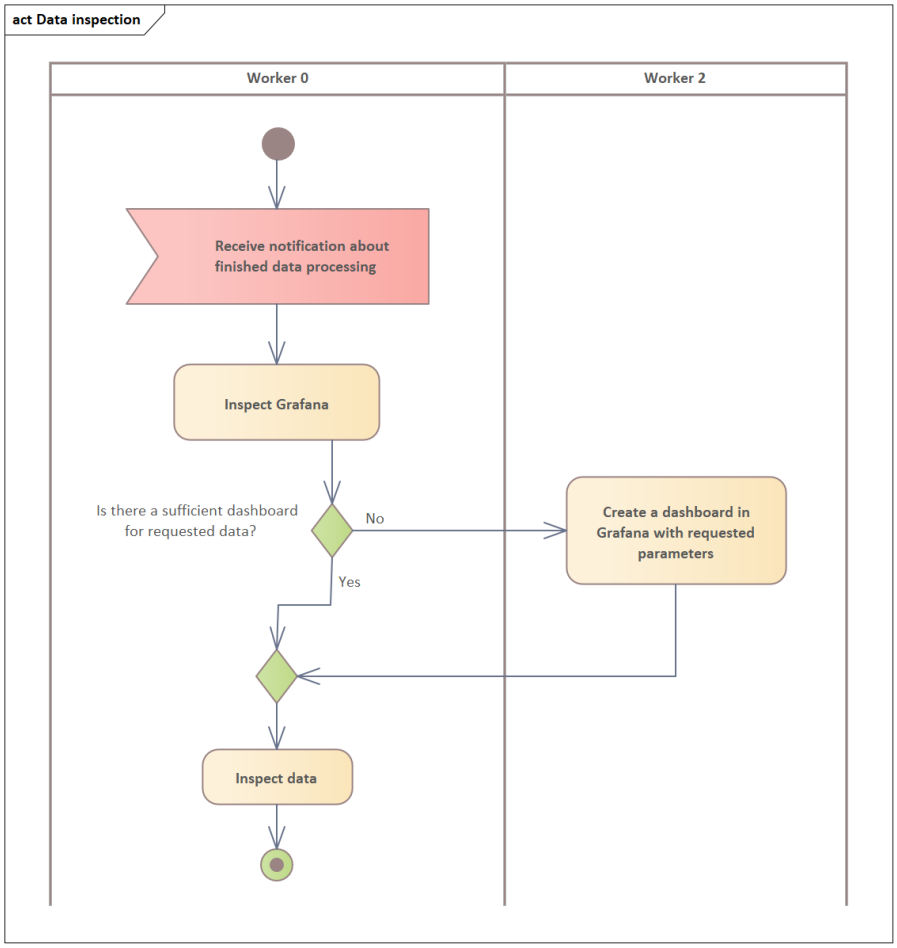


Figure 2.3: Current activity diagram for data inspection subprocess



2.2.2.3 Data inspection

Figure 2.3 illustrates the activity diagram of the subprocess, which consists of final data inspection.

2.2.3 Resources

For the purpose of improvement and subsequent objective assessment, it is necessary to mention the current time and personnel burden of the entire process. The measured tests were conducted with a package of MF4 files totalling 100 MB and a DBC file defining 21 signals without any SAE J1939-TP messages to extract. The following runtime numbers are the average of five measurements on the same system under consistent conditions.

From chapter 2.2.1, it is evident that at least two employees are required – Worker 1 and Worker 2 – not considering a third person, who requested the data and needs to inspect it. Worker 1 is responsible only for collecting data from the machine, which takes an average of 40 minutes. However, after

downloading and packaging the data, a problem arises – it needs to be sent to Worker 2, who is usually busy with other tasks. This leads to a *dead period*, during which nothing is done with the data, on average up to 2.5 days.

Downloading, extracting, and moving the data to the server takes Worker 2 an average of 25 minutes. Another 5 to 10 minutes are spent setting up and running both Python scripts, which run in the background for 27.5 minutes. Only then is it possible to see the result in Grafana, provided that a dashboard has already been defined. Creating one, if needed, takes an average of 1.3 hours.

Overall, the process starting with data retrieval from the machine to visualization in Grafana can take up to *2 days and 7 hours*, from which 27.5 minutes represent the runtime of the scripts, considering only a 100MB sample of 21 signals. Furthermore, it requires two extra personnel. However, the actual size of downloaded data from the machine fluctuates around tens of gigabytes every week, which substantially increases the runtime – a data package with a scale of 1 gigabyte defining 21 signals extends to 3 hours and 7.3 minutes.

2.3 Used technologies

Every employee of the company utilizes a computer running the Windows 10 Professional operating system, forming the foundation of the infrastructure. As previously mentioned, the files downloaded from the machines are in the MF4 format, and processing is conducted through Python, particularly the version 3.9.5, with external libraries including asammdf, cantools, numpy, openpyxl, pandas, psycopg2 and sqlalchemy.

Converted MF4 files are uploaded into the PostgreSQL 13 database, upon which Grafana 10.1.0 executes queries to display desired data. In order to see those displays, the employee must utilize a web browser for connecting to the Grafana server user interface.

2.4 Currently implemented requirements analysis

A necessary prerequisite to designing software that meets user needs is to understand what the users intend to do with it. Some teams take a product-centric approach, they focus on defining the features to implement in the software, with the hope that those features will appeal to prospective users. In most cases, though, it is better to take a user-centric and usage-centric approach to requirements elicitation. Focusing on users and their anticipated usage helps reveal the necessary functionality, avoids implementing features that no one will use, and assists with prioritization [14].

The following requirements have been formulated based on analysis of the current company process.

2.4.1 Current functional requirements

Functional requirements specify the behaviours the process will exhibit under specific conditions. They describe what the developers must implement to enable users to accomplish their tasks [15].

2.4.1.1 F1 – Conversion and upload of MF4 files into the database

The system shall allow the selection of MF4 files from a directory on the computer, conversion of these files into a text format by extracting CAN and SAE J1939-TP messages with the help of DBC files and uploading the resulting data to a remote database. All of this with graphical feedback to inform the user about the progress of the process.

2.4.1.2 F2 – Data inspection

2.4.2 Current nonfunctional requirements

Nonfunctional requirements describe the product's characteristics in various dimensions that are important either to users or to developers and maintainers, such as performance, safety, availability, and portability. Additionally nonfunctional requirements describe external interfaces between the system and the outside world. These include connections to other software systems, hardware components, and users, as well as communication interfaces [15].

2.4.2.1 N1 – Microsoft Windows platform

Despite the possibility of multi-platform development, the system will exclusively run on the Microsoft Windows 10 or Microsoft Windows 11 operating systems.

2.5 Currently implemented Use case model

A *use case* describes a sequence of interactions between a system and an external actor that results in the actor being able to achieve some outcome of value [14].

The following use cases represent the current state of the process. They have been formulated based on analysis conducted among Test engineers and Design engineers within the company. The subprocess of collecting the SD card from the machine is left out, as it does not require any interaction with information system and can't be optimized in the scope of this thesis.

2.5.1 Actors

An actor is a person (or sometimes another software system or a hardware device) that interacts with the system to perform a use case [14]. Refers to diagram on figure 2.4.

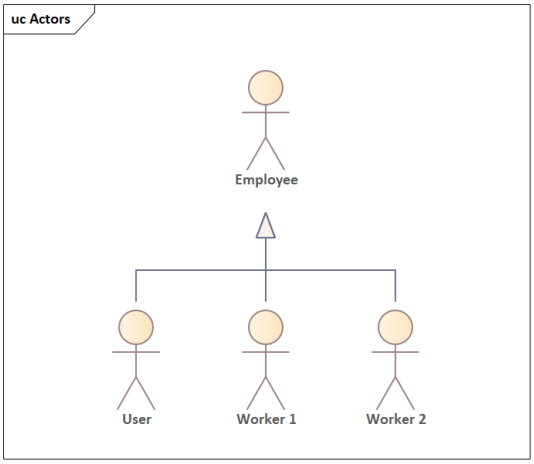
2.5.1.1 Worker 1

The person responsible for downloading data from the machine and subsequently sending it to further processing.

2.5.1.2 Worker 2

The person responsible for processing data from the machine and uploading it to the database. Additionally, this actor manages the operation of the Grafana tool.

Figure 2.4: Current actors



2.5.1.3 User

The person who requested the inspection of data from the machine – the end user.

2.5.1.4 Employee

Generalization of all forementioned actors.

2.5.2 Conversion and upload of MF4 files into the database

This use case package contains the description of functionalities ensuring the conversion and upload of MF4 files into the database.

2.5.2.1 UC1 – Packaging the machine data

Allows packaging the MF4 files downloaded from the machine into a ZIP file for easy transmission. Refers to diagram on figure 2.5.

1. The use case begins when new data from the machines are present on the Worker 1's computer.
2. Worker 1 packages the MF4 files into ZIP files.

2.5.2.2 UC2 – Sending the packaged data

Allows sending the packaged machine data to Worker 2. Refers to diagram on figure 2.5.

1. The use case begins when new data from the machine have been packaged into a ZIP file.
2. Worker 1 transfers the ZIP file to Worker 2.

2.5.2.3 UC3 – Transfering the data onto a remote computer

Allows transferring the data to the remote desktop computer within the company, where all necessary scripts are located. Refers to diagram on figure 2.5.

1. The use case begins with Worker 2 receiving a ZIP file containing collected machine data.
2. Worker 2 downloads the ZIP file.
3. Worker 2 moves the zip file from his computer to the remote desktop computer.
4. Worker 2 extracts the ZIP file.

2.5.2.4 UC4 – Converting MF4 files into Excel spreadsheets

Allows converting the binary MF4 files into a text-based Excel files with the help of DBC files. Refers to diagram on figure 2.6.

1. The use case begins with extracted machine data on the remote desktop computer.
2. Worker 2 configures all necessary changes in the Python script for MF4 file conversion.
3. Worker 2 runs the Python script for MF4 file conversion.

2.5.2.5 UC5 – Uploading the Excel spreadsheets into the database

Allows uploading the signals from converted Excel spreadsheets to the database. Refers to diagram on figure 2.7.

1. The use case begins when the Python script for MF4 file conversion finishes running.
2. Worker 2 configures all necessary changes in the Python script for data upload.
3. Worker 2 runs the Python script for data upload.

2.5.3 Data inspection

This use case package contains functionalities that facilitate inspecting the machine data uploaded to the database.

2.5.3.1 UC6 – User login

Allows any user to log into Grafana server website under given role. Refers to diagram on figure 2.8.

1. This use case begins when any employee needs to log into the Grafana server website.
2. Employee opens Grafana login website.

3. Employee fills out his login credentials.
4. Employee clicks on the *Log in* button.

2.5.3.2 UC7 – Dashboard creation

Enables the creation of a new Grafana dashboard if a suitable one does not exist. Refers to diagram on figure 2.8.

1. This use case begins if there is a lack of Grafana dashboard suitable for inspection of the freshly processed data and Worker 2 is logged into Grafana server website as an administrator.
2. Worker 2 opens the Dashboards menu on the Grafana server website.
3. Worker 2 clicks on the *New* button.
4. Worker 2 clicks on the *New dashboard* button.
5. Worker 2 configures the new dashboard as needed.

2.5.3.3 UC8 – Data inspection

Enables the User to inspect freshly processed data via the Grafana server website. Refers to diagram on figure 2.8.

1. This use case begins if User needs to inspect machine data and is already logged into the Grafana server website.
2. User opens the Dashboards menu on the Grafana server website.
3. User clicks on the desired Dashboard from the listing.
4. User inspects displayed data.

Figure 2.5: Current diagram of UC1, UC2 and UC3

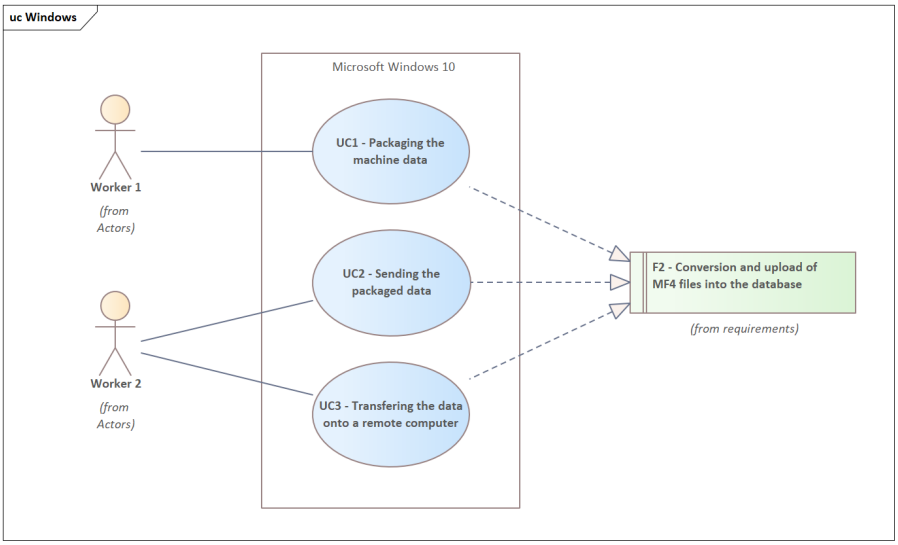
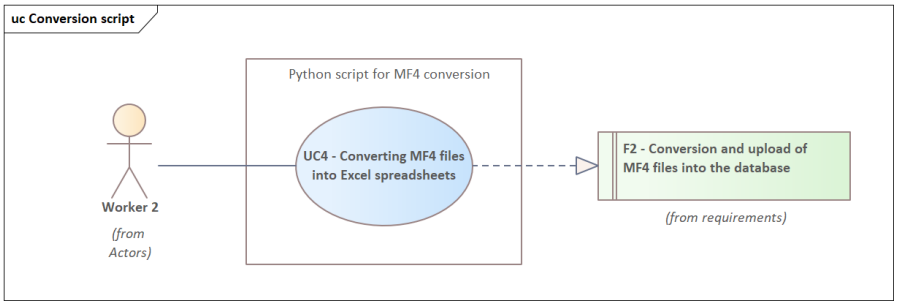


Figure 2.6: Current diagram of UC4



2.6 Summary

In this chapter, a relatively complex workflow process of data retrieval from the machine and its visualization was described. At least two employees are needed to carry it out, not counting the third one who originally requested the data inspection. The data must be sent to the correct employee, who must upload them to a dedicated remote desktop computer, as only on it the scripts for processing this data function. Therefore, the current solution is not portable. The process is also not intuitive; it requires code intervention and manual execution of Python scripts. Even the visualization itself is not automated; only statically created dashboards in Grafana are available. If a new signal needs to be read from the machine, the dashboard must be manually adjusted or a new one must be created.

All of this can mean up to 2 days and 7 hours of time for a small dataset, of which 27.5 minutes is the pure script runtime. In a real-world environment, however, larger data sets reaching tens of gigabytes will be processed, and with just 1 GB of data, the scripts now run for 3 hours and 7.3 minutes.

Figure 2.7: Current diagram of UC5

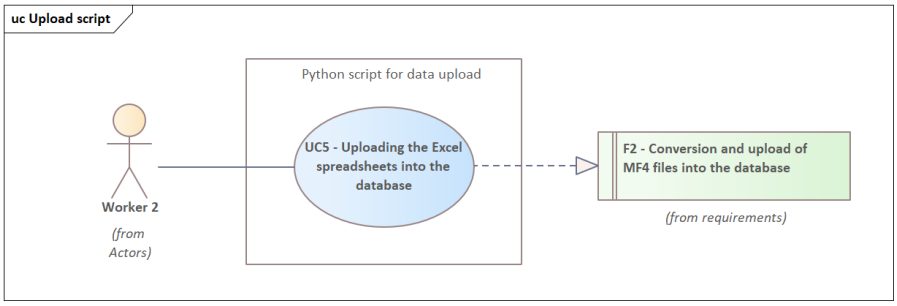
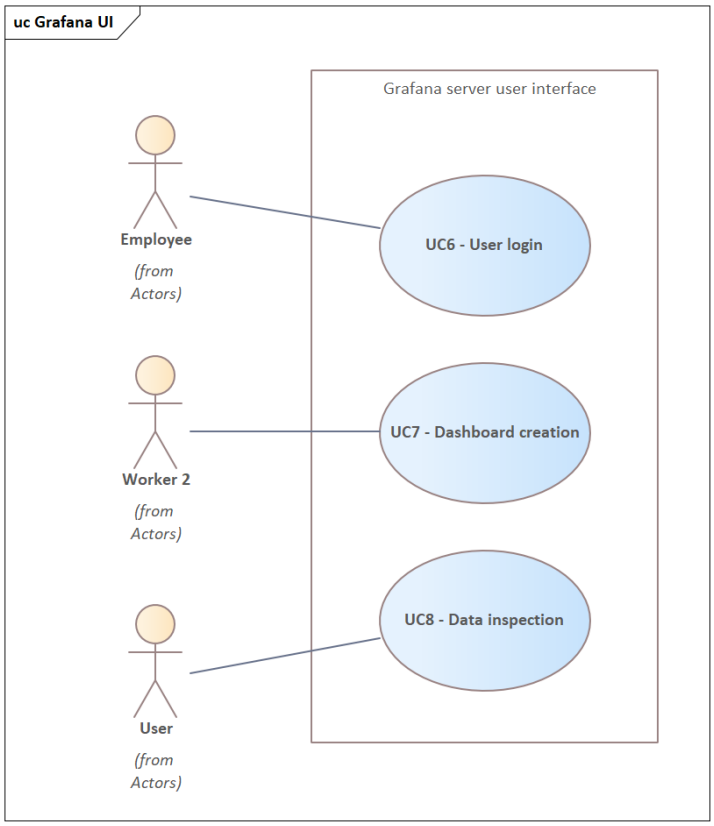


Figure 2.8: Current diagram of UC6, UC7 and UC8



Furthermore, based on the analysis, functional and non-functional requirements were defined, and the end of this chapter described existing use cases. It was discovered that only one functional requirement is currently implemented (F1 – Conversion and upload of MF4 files into the database, described in the Section 2.4.1.1). Regarding non-functional requirements, it is similar; currently, only one is implemented (N1 – Microsoft Windows platform, described in the Section 2.4.2.1).

These insights stemming from the analysis set the stage for subsequent chapters focused on optimizing the process and developing a new system to address identified inefficiencies and meet user needs effectively.

Chapter 3

Analysis of the future solution

3.1 To-Be requirements analysis

The following requirements have been formulated based on the analysis conducted among Test engineers and Design engineers within the company.

3.1.1 Functional requirements

3.1.1.1 F1 – Graphical user interface

The system shall feature a graphical user interface (GUI) that allows users to interact with system functionalities seamlessly and intuitively. This includes but is not limited to navigation menus, input forms, buttons, and graphical elements.

3.1.1.2 F2 – Conversion and upload of MF4 files into the database

The system shall allow the selection of MF4 files from a directory on the computer, conversion of these files into a text format by extracting CAN and SAE J1939-TP messages with the help of DBC files and uploading the resulting data to a remote database. All of this with graphical feedback to inform the user about the progress of the process.

3.1.1.3 F3 – Simple data aggregation

The system shall allow the user to configure data aggregation, which will occur before uploading to the database. In this case, aggregation involves skipping records that occur consecutively within a maximum specified time interval if these records have the same value.

3.1.1.4 F4 – Universal data inspection

The system shall offer the user a user-friendly and versatile tool for inspecting signals from the remote database. The data will be automatically updated based on the content of the database. The user will be able to display multiple signals in one graph simultaneously.

3.1.1.5 F5 – Data download

The system shall allow the user to download selected data losslessly directly from the database without any aggregation and save it to their computer. The selection of data for download will be done by selecting the desired signals and a specific time range.

3.1.1.6 F6 – Configuration files

The system shall save all configuration changes into configuration JSON files if the user chooses to save the changes. Upon each system start-up, the current configuration will be loaded from these files.

3.1.1.7 F7 – Manual page

Users shall have the ability to effortlessly locate a manual within the system, providing them with all necessary information for understanding and navigating the system.

3.1.1.8 F8 – Packaged application

The system shall be packaged as a self-contained application, requiring no installation of additional components or dependencies. Users will be able to launch the system simply by opening the designated application or accessing a web page.

3.1.1.9 F9 – Informational file

The system shall allow the user to log information about the currently processed MF4 file during conversion. It will record its name, path, first timestamp, and last timestamp.

3.1.1.10 F10 – Administrator mode

Certain configuration elements of the system will only be accessible to the user after entering a password for the administrator mode. These elements primarily include database connection settings.

3.1.1.11 F11 – Downloaded data formats

Downloaded data from the database shall be savable in CSV or XLSX format.

3.1.2 Nonfunctional requirements

3.1.2.1 N1 – Portability

The system shall be easily transferable between computers via various transfer media, such as cloud storage or portable storage devices. This will enable users to effortlessly run the system on any computer within the company.

3.1.2.2 N2 – Microsoft Windows platform

Despite the possibility of multi-platform development, the system will exclusively run on the Microsoft Windows 10 or Microsoft Windows 11 operating systems.

3.1.2.3 N3 – Responsive design

The system's user interface shall be responsive to the size of the window set by the user, including full-screen display capability.

3.1.2.4 N6 – No unnecessary temporal files

During its operation, the system will not store any temporary files on the computer, such as converted signals before uploading them to the database.

3.2 To-Be Activity diagrams

Following figures show an improved company process with a new system transformed into activity diagrams. Figures are also included in the attachments.

3.2.1 Data processing

Figure B.1 illustrates the activity diagram of the new subprocess, which consists of processing and uploading the source MF4 files into the database.

3.2.2 Data downloading

Figure B.2 illustrates the activity diagram of the new subprocess, which consists of downloading selected data from the database.

3.2.3 Data inspection

Figure B.3 illustrates the activity diagram of the new subprocess, which consists of final data inspection.

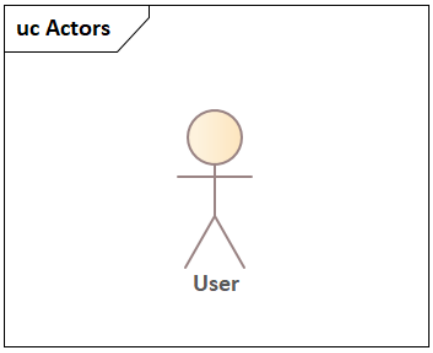
3.3 To-Be Use case model

The following use cases represent a new, improved state of the process. They have been formulated based on To-Be requirements (ref. 3.1) and analysis conducted among Test engineers and Design engineers within the company. The subprocess of collecting the SD card from the machine is left out, as it does not require any interaction with the information system and can't be optimized in the scope of this thesis.

3.3.1 Actors

In the new system, the end user stands as the sole actor. There is no need for additional participants in the improved process.

Figure 3.1: To-Be actors



3.3.1.1 User

Any employee of the company that wishes to process or inspect new machine data.

3.3.2 Basic application operations

This use case package contains the description of functionalities ensuring basic operations in the new application. Refers to diagram on figure C.1.

3.3.2.1 UC1 – Saving changes

Allows saving all changes of configuration made within the application to the configuration file.

1. The use case begins with some unsaved changes in the application, which User wants to save.
2. User clicks on the *Save changes* button in the main window.
3. If there are no problems, the application responds with "Settings updated." as a feedback information.

3.3.2.2 UC2 – Discarding changes

Allows discarding all changes of configuration made within the application.

1. The use case begins with some unsaved changes in the application, which User wants to scratch.
2. User clicks on the *Discard changes* button in the main window.
3. Application prompts a pop-up asking, if the User is sure about this action.
4. User clicks on the *Yes* button.

3.3.2.3 UC3 – Entering Admin mode

Allows User to gain administrator rights within the application.

1. The use case begins when User wants to gain administrative privileges in the application.
2. User clicks on the *Admin mode* button in the main window.
3. Application prompts a pop-up asking for the password.
4. User fills the password into the entry field in the pop-up window.
5. User click on the *Ok* button in the pop-up window.

3.3.2.4 UC4 – Viewing the manual

Allows User to view the usage manual of the application.

1. This use case begins when User wants to view the usage manual of the application.
2. User clicks on the *Manual* button in the main window.

3.3.2.5 UC5 – Changing the appearance theme

Allows changing the visual appearance of the application between *light* and *dark* modes.

1. The use case begins when User wants to change the visual appearance mode of the application.
2. User clicks on the expandable menu next to the *Theme* label in the main window.
3. The application shows a list of the following options: *Light*, *Dark*, *System*.
4. User clicks on the desired option.

3.3.3 Converting and uploading MF4 data into the database

This use case package contains the description of functionalities ensuring processing and uploading of source MF4 files. Refers to diagram on figure C.2.

3.3.3.1 UC6 – Selecting MF4 files directory

Allows User to select a directory with source MF4 files.

1. This use case begins when it is needed to specify the source directory of MF4 files.
2. User clicks on the *Before start* button in the main navigation within the main window of the application.
3. User clicks on the *Select MF4 dir* button in the newly displayed content frame within the main window.

4. Application prompts User with a standard directory selection tool default to the operating system.

5. User navigates to the desired directory and confirms the selection.

3.3.3.2 UC7 – Selecting DBC files directory

Allows User to select a directory with DBC files.

1. This use case begins when it is needed to specify the source directory of DBC files.

2. User clicks on the *Before start* button in the main navigation within the main window of the application.

3. User clicks on the *Select DBC dir* button in the newly displayed content frame within the main window.

4. Application prompts User with a standard directory selection tool default to the operating system.

5. User navigates to the desired directory and confirms the selection.

3.3.3.3 UC8 – Turning on/off data aggregation

Allows enabling simple aggregation upon data from MF4 files during the conversion.

1. This use case begins when User wants to enable simple aggregation upon data from MF4 files.

2. User clicks on the *Conversion & Upload* button in the main navigation within the main window of the application.

3. User checks (on) or unchecks (off) the switch next to the *Aggregate raw data* label.

3.3.3.4 UC9 – Configuring data aggregation

Allows configuring the simple aggregation upon data from MF4 files, if the *Aggregate raw data* option is enabled.

1. This use case begins when the *Aggregate raw data* option in the *Conversion & Upload* content frame is enabled and User wants to further configure this setting.

2. User types a new value of seconds into the entry next to the *Seconds to skip when value is consistent* label.

3.3.3.5 UC10 – Turning on/off moving done files

Allows enabling movement of already processed MF4 files into another directory.

1. This use case begins when User wants to enable the movement of processed MF4 files.
2. User clicks on the *Conversion & Upload* button in the main navigation within the main window of the application.
3. User checks (on) or unchecks (off) the switch next to the *Move done files* label.

3.3.3.6 UC11 – Configuring moving done files

Allows selection of destination directory for the moved MF4 files.

1. This use case begins when the *Move done files* option in the *Conversion & Upload* content frame is enabled and User wants to further configure this setting.
2. User clicks on the *Select destination* button.
3. Application prompts User with a standard directory selection tool default to the operating system.
4. User navigates to the desired directory and confirms the selection.

3.3.3.7 UC12 – Starting conversion and upload of MF4 files

Allows running of the main conversion and upload of the source MF4 files with the help of DBC files.

1. This use case begins when User wants to start the main conversion and upload process.
2. User clicks on the *Conversion & Upload* button in the main navigation within the main window of the application.
3. User clicks on the *Start conversion & upload* button.
4. If there are no problems, the application displays a progress bar and starts displaying feedback messages.

3.3.3.8 UC13 – Inspecting informational file

Allows inspection of the informational file, when the application has converted and uploaded at least 1 MF4 file.

1. The use case begins when User wants to view the contents of the informational file and the application has already converted and uploaded at least 1 MF4 file.
2. User navigates to the directory where the application executable is located.
3. User opens and inspects the *MF4-info.csv* file in the same directory.

3.3.4 Database configuration

This use case package contains the description of functionalities ensuring all database configurations within the new application. Refers to diagram on figure C.3.

3.3.4.1 UC14 – Configuring database host

Enables configuring the host of the database connection. Requires administrator privileges.

1. This use case begins when User has administrator privileges and wants to configure the host of the database connection.
2. User clicks on the *Database config* button in the main navigation within the main window of the application.
3. User types a new value into the entry field next to the *Host* label.

3.3.4.2 UC15 – Configuring database port

Enables configuring the port of the database connection. Requires administrator privileges.

1. This use case begins when User has administrator privileges and wants to configure the port of the database connection.
2. User clicks on the *Database config* button in the main navigation within the main window of the application.
3. User types a new value into the entry field next to the *Port* label.

3.3.4.3 UC16 – Configuring database name

Enables configuring the database name of the database connection. Requires administrator privileges.

1. This use case begins when User has administrator privileges and wants to configure the database name of the database connection.
2. User clicks on the *Database config* button in the main navigation within the main window of the application.
3. User types a new value into the entry field next to the *Database* label.

3.3.4.4 UC17 – Configuring database user

Enables configuring the user of the database connection. Requires administrator privileges.

1. This use case begins when User has administrator privileges and wants to configure the user of the database connection.
2. User clicks on the *Database config* button in the main navigation within the main window of the application.
3. User types a new value into the entry field next to the *User* label.

3.3.4.5 UC18 – Configuring database password

Enables configuring the password of the database connection. Requires administrator privileges.

1. This use case begins when User has administrator privileges and wants to configure the password of the database connection.
2. User clicks on the *Database config* button in the main navigation within the main window of the application.
3. User types a new value into the entry field next to the *Password* label.

3.3.4.6 UC19 – Configuring database schema name

Enables configuring the used schema name within the database.

1. This use case begins when User wants to configure the used schema name within the database.
2. User clicks on the *Database config* button in the main navigation within the main window of the application.
3. User types a new value into the entry field next to the *Schema* label.

3.3.5 Data downloading

This use case package contains the description of functionalities ensuring downloading of signal data from the database. Refers to diagram on figure C.4.

3.3.5.1 UC20 – Updating signal names

Allows dynamical updating of the signal names currently stored in the database.

1. The use case begins when User wants to download data from the database.
2. User clicks on the *Download data* button in the main navigation within the main window of the application.
3. User clicks on the *Update signal names* button on the newly displayed content frame.

3.3.5.2 UC21 – Selecting signals for download

Allows selecting signals to be downloaded from the database. Requires having the update of signals already completed (see 3.3.5.1).

1. The use case begins when User wants to download data from the database and has already updated the signal names.
2. User clicks on the *Select signals* button on the *Data download* content frame.
3. Application will display a new widows containing a list of all the available signals with check marks next to every one of them.

4. User selects the *check marks* of all the signals intended for download.

5. User clicks on the *Ok* button.

3.3.5.3 UC22 – Selecting time range for download

Allows the selection of the time frame delineating the range of downloaded signals. Requires having the update of signals already completed (see 3.3.5.1).

1. The use case begins when User wants to select the time range for downloaded data and has already updated the signal names.

2. User clicks on the *Set time filter* button on the *Data download* content frame.

3. Application will display a new widows containing a tool for selecting opening and closing timestamps.

4. User selects the proper date in the interactive calendar for both opening and closing timestamp.

5. User fills out desired hours, minutes and seconds for both opening and closing timestamp.

6. User clicks on the *Ok* button.

3.3.5.4 UC23 – Downloading selected data

Allows downloading requested signals of a specific time range from the database. Requires having the update of signals, selection of signals and selection of time range already completed (see 3.3.5.1 to 3.3.5.3).

1. The use case begins when User wants to download signals from a specific time range and has already updated the signal names, selected the signals and selected the time range.

2. User clicks either on the *Download as CSV* or *Download as Excel* button on the *Data download* content frame.

3. Application prompts User with a standard file selection tool default to the operating system.

4. User navigates to the desired directory and confirms the selection.

5. If there are no problems, the application responds with a feedback information confirming the download in the selected format.

3.3.6 Creating Grafana dashboard JSON

This use case package contains the description of functionalities ensuring the creation of a new Grafana dashboard JSON to be imported into Grafana. Refers to diagram on figure C.5.

3.3.6.1 UC24 – Entering dashboard name

Allows entering the name of the JSON Grafana dashboard.

1. This use case begins when User wants to enter a name for the generated Grafana dashboard.
2. User clicks on the *Get Grafana template* button in the main navigation within the main window of the application.
3. User types a new value into the entry field next to the *1) Enter Dashboard name:* label.

3.3.6.2 UC25 – Getting dashboard JSON

Allows downloading of the correct JSON Grafana dashboard.

1. This use case begins when User wants to generate and save the Grafana dashboard.
2. User clicks on the *Get Grafana template* button in the main navigation within the main window of the application.
3. User clicks on the *Get template* button.
4. Application prompts User with a standard file selection tool default to the operating system.
5. User navigates to the desired directory and confirms the selection.
6. If there are no problems, the application responds with a feedback information confirming the successful save of the JSON file.

3.3.7 Data inspection

This use case package contains the description of functionalities ensuring inspection of the processed data. Refers to diagram on figure C.6.

3.3.7.1 UC26 – Logging into Grafana

Allows User to log into Grafana server website under given role.

1. This use case begins when User needs to log into the Grafana server website.
2. User opens Grafana login website.
3. User fills out the login credentials.
4. User clicks on the *Log in* button.

3.3.7.2 UC27 – Dashboard creation

Enables the creation of a new Grafana dashboard if a suitable one does not exist.

1. This use case begins if there is a lack of Grafana dashboard suitable for inspection of the freshly processed data and User is logged into Grafana server website as an administrator.
2. User opens the *Dashboards* menu on the Grafana server website.
3. User clicks on the *New* button.
4. User clicks on the *Import* button.
5. User clicks on the *Upload dashboard JSON file* section
6. Grafana prompts User with a standard file selection tool default to the operating system.
7. User navigates to the desired JSON file and confirms the selection.
8. User clicks on the *Import* button.

3.3.7.3 UC28 – Data inspection

Enables User to inspect freshly processed data via the Grafana server website.

1. This use case begins if User needs to inspect machine data and is already logged into the Grafana server website.
2. User opens the Dashboards menu on the Grafana server website.
3. User clicks on the desired Dashboard from the listing.
4. User inspects displayed data.

Chapter 4

Design

Having dissected the intricacies of the current workflow and identified its limitations in the Chapter 2, it's evident that a significant overhaul – featuring content from the analysis of the future solution in Chapter 3 – is necessary. The preceding analysis unveiled a complex process fraught with inefficiencies and a lack of automation, leading to substantial time investments and operational constraints.

With the Design chapter, the focus shifts towards crafting a novel system that addresses these shortcomings while adhering to the new requirements outlined earlier. Through planning and innovative ideation, this chapter aims to engineer a solution that streamlines machine data processing and visualization, fostering agility, simplicity, and user-centric functionality.

4.1 Proposal of the new solution

The entire current process can be almost fully automated. Instead of several required workers, only one person will be responsible for its operation. Specialized workers will no longer be required, as the intuitive nature of the new solution will empower anyone to handle the task effectively.

The main enhancement will result in a new desktop application, replacing the current disjointed scripts. The user-friendly graphical interface will guide each user through the entire process of handling machine data from the comfort of their computer with minimal intervention required. Primarily, the application will feature loading of MF4 files, decoding them with extended CAN databases (with the support of SAE J1939-TP messages) and uploading the resulting data to a remote server. Additionally, users will be able to choose whether they want to perform operations on the data, such as aggregation.

The secondary functionality of the new application will involve the capability to download data directly from the remote database using signal names and timestamps as references. This data will be storable in both CSV and Excel formats.

A web service accessible from anywhere within the company's intranet will facilitate the reading of all signals stored in the database. Data from this database will be dynamically pulled into interactive plots, allowing users to define which signals they want to view within a specified time range. The framework of the solution to this part is already provided by Grafana tools and the

system will utilize its capabilities. Grafana dashboard JSON files will be generated within the desktop application, allowing simple set-up of new views for data inspection.

With this approach, the program will implement all of the requirements defined by the analysis in the chapter 3.1, while simultaneously reducing overall complexity and time consumption of this process.

4.2 Architecture

Qhe software architecture of a system is the set of structures needed to reason about the system. These structures comprise software elements, relations among them, and properties of both [16].

Since architecture consists of structures, and structures consist of elements and relations, it follows that an architecture comprises software elements and how those elements relate to each other. This means that architecture specifically and intentionally omits certain information about elements that is not useful for reasoning about the system. Thus an architecture is foremost an abstraction of a system that selects certain details and suppresses others. In all modern systems, elements interact with each other by means of interfaces that partition details about an element into public and private parts. This abstraction is essential to taming the complexity of an architecture: one simply cannot, and do not want to, deal with all of the complexity all of the time. The understanding of a system's architecture is needed to be many orders of magnitude easier than understanding every detail about that system [16].

Several following sections of this chapter will be devoted specifically to the architecture of the designed application.

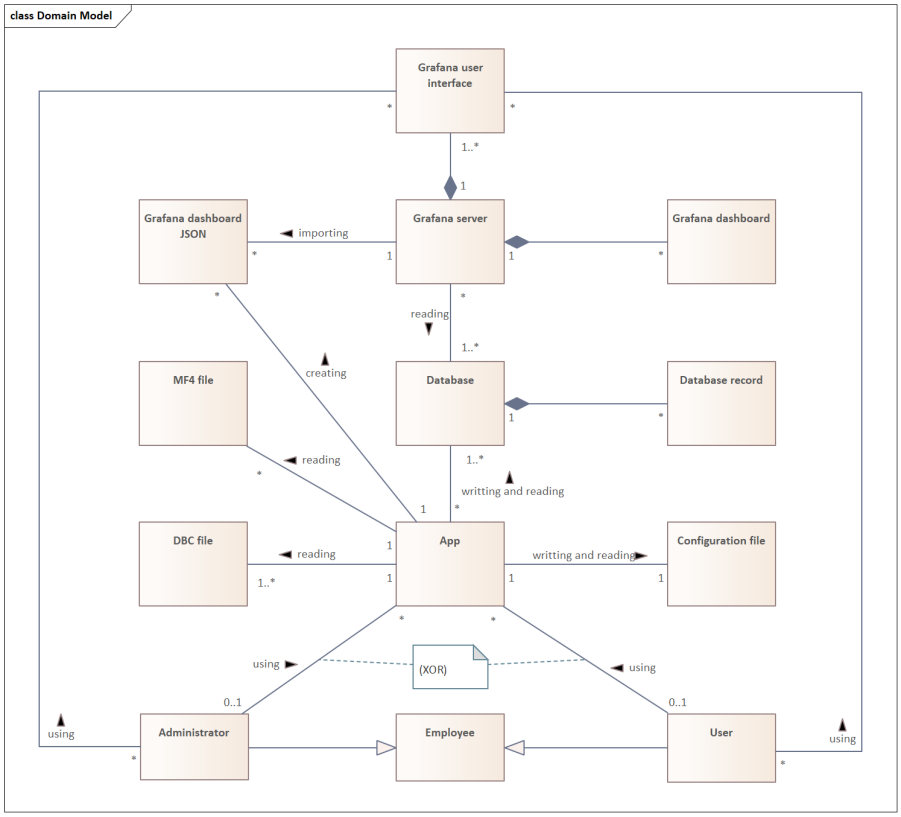
4.2.1 Domain model

A domain model serves as a conceptual representation of the fundamental entities and their relationships within a specific problem domain. It encapsulates the essential concepts and interactions, providing stakeholders with a common understanding of the problem space that the information system addresses. By identifying the relevant entities and relationships, the domain model helps define the scope and requirements of the system, guiding the design, development, and implementation processes.

Figure 4.1 shows the domain model of this system design. It consists of a core composed of three objects – *App*, *Database*, and *Grafana server*. The *App* represents the main desktop application with which users will primarily interact, when it comes to data processing or downloading. The *App* has access to the *Database*, where processed data is stored and from where data is also retrieved. Additionally, it has a *Configuration file* for storing all settings, *MF4 files* with machine data, and *DBC files* used for their translation. The *App* can be used by either a regular *User* or an *Administrator*, but not both simultaneously.

The second important node is the *Grafana server* entity, which can display *Grafana dashboards* through the *Grafana user interface*. In this way, users can dynamically view data pulled from the *Database*. The *Grafana server* is capable of creating an unlimited number of *Grafana user interface* instances, so the total of simultaneous users is unlimited.

Figure 4.1: System domain model design



4.2.2 Architectural style

The proposed system comprises two distinct components: a desktop application responsible for data processing, upload, and download functionalities, and a server dedicated to hosting the visualization tool Grafana, featuring a web-based user interface.

The desktop application will adhere to a *two-layer architecture* model, strategically partitioning the visual and logical components. This architectural approach ensures optimized performance of the graphical user interface (GUI) while efficiently distributing computationally intensive tasks across available CPU cores. Embracing conventional frontend-backend development paradigm, the application segments will exclusively communicate via a text-based pipeline, ensuring seamless interoperability between processes.

In contrast, Grafana adopts a *microservices-based architecture*, aptly named "Mimir". This architectural model embodies multiple horizontally scalable microservices, each capable of autonomous operation and concurrent execution. Leveraging this framework, the system attains enhanced scalability and flexibility. Notably, the visualization tool capitalizes on web technologies to furnish users with an intuitive interface, facilitating seamless navigation and interaction.

Both components are mutually independent; they are connected solely through a shared database containing records from converted MF4 files.

4.2.3 Security architecture

The Security Architecture (SA) practice focuses on the security linked to components and technology that developers deal with during the architectural design of the software. Secure Architecture Design looks at the selection and composition of components that form the foundation of the solution, focusing on its security properties [17].

The security architecture of the new desktop application is meticulously designed to ensure robust protection and controlled access. The application operates locally within its directory on any computer within the company, ensuring it does not interfere with any files except for its configuration file. This isolation helps maintain the integrity of the system and minimizes the risk of accidental or malicious file modifications. The only external connection established by the application is to a remote database within the company. This connection is tightly controlled and requires the entry of correct access credentials, which are safeguarded and can only be altered through administrative rights. Users can obtain these rights only by entering the correct password within the application, ensuring that only authorized personnel can make critical changes.

The security of the server hosting the database is governed by the company's security policies. Access to the server is restricted to authorized employees only, ensuring that sensitive data is protected from unauthorized access and potential breaches. The Grafana dashboard, an integral part of this architecture, is configured to execute read-only queries, thus preventing data manipulation risks. Any changes to the dashboard or its queries require administrative rights, which can be obtained only by logging in with the correct credentials. This read-only configuration ensures that the integrity of the data is maintained, and unauthorized modifications are prevented.

By adhering to these security measures, the application ensures that both the local environment and the remote database are protected against unauthorized access and potential security threats. The architecture supports a secure, controlled, and reliable operation, safeguarding sensitive company data and maintaining compliance with internal security standards.

4.2.4 Technology stack

A technology stack is a comprehensive set of tools and technologies that work together to build, deploy, and maintain an application. It includes everything from front-end and back-end development to infrastructure, security, and monitoring, providing a cohesive framework for delivering a functional and efficient software solution.

This chapter describes the architecture of the technological solution for the proposed system. Given that Grafana is not the subject of design and implementation, its technological parameters will not be discussed in detail. However, its presence in this system will be described and justified.

4.2.4.1 Python

The entire desktop application will be written in the Python programming language. The primary reason for this choice is the ability to utilize the *asammdf* and *can_decoder* libraries, which handle the translation of MF4 files into signal-based text format, supporting the SAE J1939-TP standard.

Other than that, Python was chosen due to its versatility, ease of use, and strong community support. One of the key advantages of Python is its simplicity and readability, which make it accessible to developers of all skill levels. This is particularly important for ensuring that the codebase is maintainable and that new developers can quickly become productive.

Furthermore, Python boasts a rich ecosystem of libraries and frameworks that can significantly speed up the development process. For data processing tasks, libraries such as *pandas* and *NumPy* provide powerful tools for handling large datasets efficiently. These libraries are essential for the project, given the need to process and analyze machine data. Additionally, Python's robust support for integration with databases through libraries like *SQLAlchemy* ensures that data can be managed effectively.

In addition to its strengths in data processing and integration, Python also offers excellent frameworks for building GUIs, such as the *customtkinter* library. It allows developers to create modern, responsive, and user-friendly interfaces with ease. This framework extends the capabilities of the standard *Tkinter* library, providing more flexibility and a wider range of design options.

Lastly, Python's extensive community support and comprehensive documentation make it easier to find solutions to potential problems and stay updated with the latest developments in technology. The availability of numerous tutorials, forums, and third-party modules ensure that developers have access to a wealth of resources, which can facilitate the development process and enhance the overall robustness of the application. Therefore, Python's combination of simplicity, powerful libraries, and strong community support makes it the optimal choice for this project's programming language.

4.2.4.2 JSON file

The JSON (JavaScript Object Notation) file format is ideal for storing the application configuration due to its simplicity, readability, and compatibility. It is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate. Its straightforward, key-value pair structure allows configuration settings to be clearly defined and organized, making it easy to understand and modify. Additionally, JSON is language-independent but has parsers available in almost every programming language, including Python. This ensures seamless integration and manipulation within the application.

4.2.4.3 PostgreSQL

The database system for storing the converted data and subsequently reading it for visualization will be implemented using PostgreSQL 16. This open-source relational database management system offers a high level of reliability and stability, which is essential for storing and managing the application's data. Its

strong support for ACID (Atomicity, Consistency, Isolation, Durability) properties ensures data integrity and transaction reliability, making it a trustworthy choice for applications dealing with large datasets.

Another key reason for selecting PostgreSQL is its advanced support for SQL standards and extensive functionality. It supports complex queries, foreign keys, joins, views, triggers, and stored procedures, providing a useful toolkit for data management. PostgreSQL's powerful indexing and full-text search capabilities enable efficient querying and data retrieval, which is crucial for the performance of the application.

PostgreSQL also boasts strong community support and extensive documentation, ensuring that developers have access to a wealth of resources and expertise. This active community contributes to regular updates and improvements, keeping the database system secure and up-to-date with the latest features and best practices.

4.2.4.4 Grafana

There is no need for designing a new visualization tool because Grafana offers a robust solution that meets all the requirements for data visualization. As a leading open-source platform, Grafana is renowned for its ability to visualize time-series data effectively and interactively. Its extensive features and proven reliability make it an ideal choice for this project, eliminating the need to develop a custom tool from scratch.

Grafana can seamlessly integrate with PostgreSQL, the selected database system, as well as numerous other databases and data sources. This integration capability allows for the efficient querying and display of the converted data, enabling users to interact with and analyze the information dynamically. Importantly, Grafana can be deployed on a server within the company, and employees can simply access it via a web browser from their computer, making it highly accessible and convenient.

Its user-friendly interface and powerful visualization capabilities are another significant advantage. It provides a wide range of visualization options, including graphs, tables, and heatmaps, allowing stitching together the most appropriate format for the data.

In addition, Grafana allows administrators to set up interactive dashboards, ensuring that only authorized personnel can create and modify visualizations. The ability to create and share dashboards quickly enhances collaboration and ensures that stakeholders have access to the information they need in an easily digestible format. Regular users are then enabled to customize their views by selecting desired signals, applying filters, and drilling down into specific data points. Moreover, administrators have the option to import dashboards from JSON files, providing a convenient way to share and replicate dashboards across different environments.

4.2.5 Deployment

This chapter will outline the deployment strategy and procedures for the system, detailing the environment setup, portability, rollout plan and installation process. It aims to provide an understanding of how the developed solution will be deployed and managed in a real-world environment, ensuring its successful operation.

4.2.5.1 Environment

All components will be designed for the Microsoft Windows 10 operating system running on computers with processors featuring 4 or more physical cores. These computers will be connected to the corporate intranet, ensuring communication with company servers where the database and Grafana backend will be running. The application and Grafana frontend will be executable from individual employee personal computers.

4.2.5.2 Portability

The Python application will be encapsulated within an executable file, complete with its own interpreter, thereby facilitating seamless execution for users without the need to install all the requisite dependencies individually. The executable file will be located in the application directory, which users can freely move between computers using any means of information transport. Subsequently, each employee can access the Grafana user interface with ease by simply inputting the correct URL into a web browser.

4.2.5.3 Rollout plan

The entire development process will be monitored through the company's GitHub repository. On its page, release versions will be available for download, accompanied by comments detailing the changes implemented. Each release will be versioned according to the schema $x.y.z$, where x signifies the major version of the application, y denotes additions or enhancements to functionality, and z indicates improvements that do not affect functionality, such as bug fixes. Each new version will evolve based on any new system requirements that may arise – without any strict schedule.

4.2.5.4 Installation

The application will not require any installation; it will suffice to download the release from GitHub and *run the executable*. However, before the initial launch, it will be necessary to set up the PostgreSQL database and Grafana backend on the company server. Within the database, two users will need to be created: one with editing and schema/table creation rights, and another with read-only privileges. Within the application, logging in with administrative rights will be necessary for correctly configuring the database connection with the authorized user for modifications. In Grafana, it will be necessary to define an administrator and users. Subsequently, the administrator can configure the database connection using the read-only user and import the JSON dashboard generated from the application.

Users will be also able to build the executable of the application locally by following the instructions provided in the *README* file, which defines all the necessary technologies and guides users through the entire building process.

4.2.6 Documentation

Documentation and sustainability are crucial aspects of ensuring the long-term success and viability of the project. Comprehensive code documentation will be provided through *GitHub Pages* of the repository to guide future developers through the application code, ensuring a smooth onboarding process and minimizing potential issues.

All requirements for the building process and technological requirements will be written in the *README* file. The user guide on how to use the application will be included as part of the application itself, ensuring accessibility and convenience for users. Additionally, users will find instructions on how to use Grafana dashboards on the *Wiki pages* of the GitHub repository, providing comprehensive support and guidance for utilizing the system's visualization features.

4.3 User experience

User experience (UX) is made of all the interactions a user has with a product or service. It is the personal, internal experience customers go through when using a product's interface [18].

The aim of this section is to design such UX, that will allow the users to work with the system with ease. The UX of Grafana has already been designed and implemented by Grafana Labs, so the following text will only pertain to the Python desktop application.

4.3.1 User roles

By default, the application will operate without requiring any login credentials. Any user can simply open it and use it. However, there will be an option to enter a password for administrator login. With administrator rights, additional options will become available on various screens of the application.

4.3.2 Navigation and flow

The application will consist of two main frames – a *navigation menu* and a display area (referred to as the *content frame*). Through the navigation menu, users will switch between various application functionalities, which will be displayed in the content frame. The menu items will exclusively include

- selection of MF4 and DBC files,
- database connection settings,
- conversion and upload of MF4 files,
- data downloading and
- creation of a JSON dashboard for Grafana.

The content frame for selecting MF4 and DBC files will feature an intuitive selection of paths to directories where these files are located on the computer. Users will also receive feedback displaying the currently selected directory path.

The content frame for database connection settings will depend on whether the user is logged in as an administrator or not. If so, all necessary configurations for connecting to the remote database will be displayed. This may include fields such as IP address, port, username and password. Without administrative rights, users will only be able to edit the schema name where the application will store converted data.

The content frame for converting and uploading MF4 files will consist of configurations designed for data processing, primarily options related to aggregation. The main element will be a button that initiates the entire process of converting and uploading MF4 files.

The content frame for downloading data from the database will offer users a selection of signals currently available in the database, from which they can choose any number. Subsequently, users will use an intuitive tool to select the start and end data timestamp to specify the range of downloaded data. Another button will then start the download of selected data to the location chosen by the user on their computer.

The content frame for creating a JSON dashboard for Grafana will only require an option for defining its name. The application will handle the rest, and users will only need to click the download button.

Additionally, the content frame will be divided into two sections, with the smaller one containing a screen where the application will display feedback information for the user. This screen will always be in the same location and will be present in all instances. Below the content frame, buttons will be placed for saving the changes made in the application.

4.3.3 Performance

The application's response time will be instantaneous; users do not want to wait after each button click. However, the speed of the MF4 file conversion and upload process will depend on the hardware resources of the computer where the application will be running. The faster the clock speed of the CPU, the quicker it will be done.

4.3.4 Error handling

It is possible for errors to occur within the application, whether on the user's side or within the program itself. In both cases, the application will respond by creating a new dialogue window to inform the user of the encountered issue. Depending on the nature of the error, the application will provide the user with certain options on how to proceed further.

User errors may, for example, include incorrectly filled or omitted fields, unsaved settings, or attempts to close the application during the conversion process. In all these cases, the application will alert the user with a new dialogue window and guide them on how to resolve the issue.

4.3.5 Accessibility

The application will scale relative to the display scale settings of the operating system. To enhance readability and reduce eye strain, the application will support both light and dark modes for appearance. The color scheme of the application will accommodate all possible types of color blindness affected users.

4.4 Graphical user interface

The user interface (UI) is the point of human-computer interaction and communication in a device. This can include display screens, keyboards, a mouse and the appearance of a desktop. It is also how a user interacts with an application or a website, using visual and audio elements, such as type fonts, icons, buttons, animations and sounds [19].

Most modern devices and applications leverage a graphical user interface in their builds. A graphical user interface (GUI) is a digital interface in which a user interacts with graphical components such as icons, buttons, and menus. In a GUI, the visuals displayed in the user interface convey information relevant to the user, as well as actions that they can take [20].

In accordance with the initial functional requirements outlined in this system (refer to 3.1.1.1), the application will showcase a GUI constructed upon the UX framework delineated in Section 4.3. Subsequent paragraphs will delve into detail, describing and showcasing the visual aspects of the application.

4.4.1 Typography and colours

The text font utilized throughout the entire application will belong to the Roboto font family and will be set to the normal weight at a size of 13 points. The application's colors will vary based on the display mode – *light* or *dark*.

In the *light* mode, the color of all texts except buttons will be black (hex #000000), and the background will be filled with light gray (hex #ebebeb). The color of the navigation menu will be slightly darker (hex #dbdbdb). All buttons will be a bold blue (hex #3b8ed0), with white text (hex #ffffff). Any input text elements will also be white.

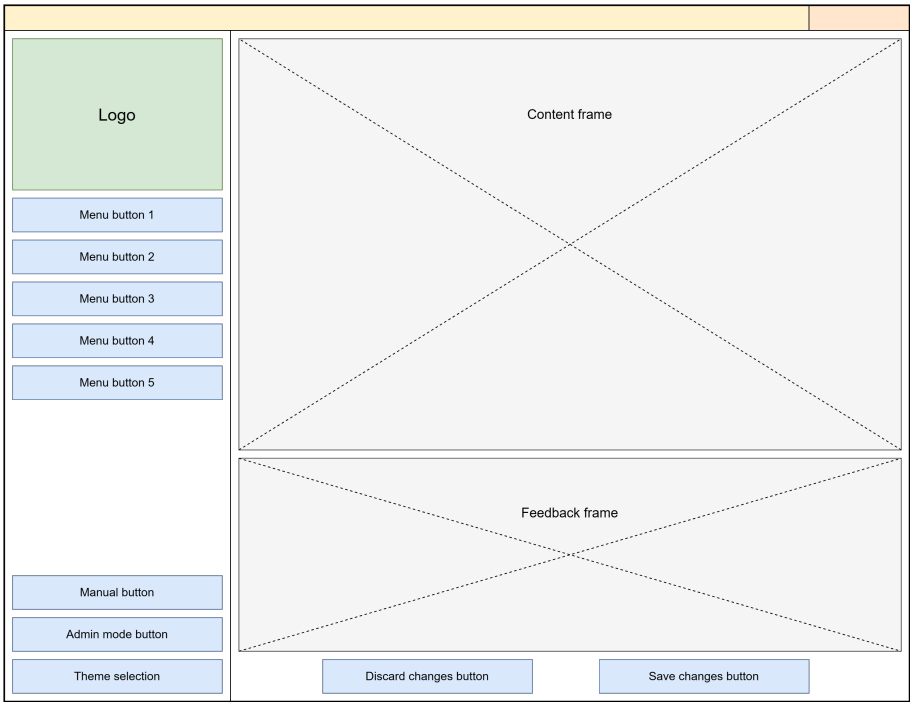
The *dark* mode will function as a complement to the light mode, except for the buttons, which will remain the same color, including their text. Text outside the buttons will be white, the default background will be dark gray (hex#242424), and the navigation menu will be slightly lighter (hex #2b2b2b). Any input text elements will also have the same lighter color.

4.4.2 Responsiveness

The main window of the application will be resizeable, and its content will dynamically adjust to the current size. By content, everything except the navigation panel is meant. Navigation will remain fixed in size at all times.

The application exclusively utilizes English as its language, reflecting the formal language standard of the company. Given this, there is no requirement for additional language support.

Figure 4.2: Wireframe of the main window



4.4.3 Wireframe

The figure 4.2 shows the basic wireframe of the GUI of the new application. On the left side, there will be a column with the main navigation menu, the content frame will be located at the top right, and below it there will be the program feedback field. At the bottom of the left side, there will be buttons for displaying the manual and switching to the Admin mode, while on the right, there will be buttons for saving or discarding changes in the configuration.

Chapter 5

Implementation and testing

This chapter transitions from the design phase to the implementation and testing of the application. Here, the steps taken to turn the design specifications into a working product will be described. Following the implementation, testing strategies will be discussed.

5.1 Implementation

The application was implemented based on the specifications defined in the analysis conducted in chapter 3 and the design defined in chapter 4. Using Python, a two-tier desktop application with a fully responsive GUI was developed in an object-oriented manner. The application meets all functional and non-functional requirements outlined in Section 3.1, except for the functional requirement F4 (see 3.1.1.4), which is addressed by the Grafana dashboard.

The separation of the frontend and backend was achieved using the *multiprocessing* library, enabling the creation of new Python processes within a single program. These processes communicate textually through a *pipeline*. Both layers feature an interface class to interpret received messages correctly, running on separate threads. Message transmission between processes is encapsulated in a custom *PipeCommunication* class, ensuring simpler operations.

The frontend was developed using the *customtkinter* library, an extension of the widely spread *tkinter* library. This library facilitates a simple grid-like window design with customizable widgets. Logical operations were moved to the backend layer to avoid overloading the rendering process.

The backend process primarily involves functions for processing MF4 files and uploading them to a database. Libraries such as *can_decoder*, *mdf_iter*, *canedge_browser*, *asammdf*, and *pandas* translate the original binary files into dataframes containing decoded signals and their data. If the user desires, aggregation is performed on the final dataframes. Once this is completed, the final data is uploaded to the database, managed by a class that utilizes the *sqlalchemy* and *psycopg2* libraries.

All Python modules were eventually packaged into an executable file and placed in a folder along with other necessary components. This was accomplished using the *cx_Freeze* library, which bundles all source files and the Python interpreter into an *.exe* file, ensuring portability and eliminating the need for installations. This process was automated using a batch script.

To implement the creation of a dashboard in Grafana, it was first necessary to create a JSON template, which the application would edit. This was achieved by creating a real dashboard template in Grafana, from which the JSON configuration was exported. The dashboard has been revamped to dynamically retrieve all signals from the database, empowering users to selectively compare desired values in time within single plots.

5.2 Unit testing

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually scrutinized for proper operation. Software developers complete unit tests during the development process. The main objective of unit testing is to isolate written code to test and determine if it works as intended [21].

During the development of the application, the basic functions were tested, primarily focusing on independent logical operations and file handling. The graphical user interface was mainly reviewed through user tests – as described in the following Section 5.3.

For implementing unit tests, the *unittest* library was used. It is a built-in Python module, making it easily accessible without requiring additional installations, which simplifies the setup process. Additionally, *unittest* supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of tests from the reporting framework, which streamlines the testing workflow. Its wide adoption and extensive documentation also provide ample resources and community support, making it an ideal choice for systematic and effective unit testing.

5.3 User testing

User testing is the process through which the interface and functions of an app are tested by real users who perform specific tasks in realistic conditions. The purpose of this process is to evaluate the usability of that website or app and to decide whether the product is ready to be launched for real users [22].

In the application, the effectiveness and intuitiveness of the designed UX, the functionality of the GUI, and the stability of functionalities were mainly verified through user tests. These tests engaged multiple independent employees, each following specific scenarios that delineated the steps required to accomplish various use cases. This approach significantly fostered the agile development methodology of the application. It ensured that based on the insights gained from these tests, continuous enhancements, fixes, or additions were made to specific functionalities.

5.4 Stress testing

In a system dealing with large volumes of data, testing its durability under extreme conditions with a maximal performance load becomes imperative. This is precisely where stress tests come into play, as conducted on this application.

The stress tests involved running up to 20 instances of the application simultaneously, with each instance executing the conversion process of source MF4

files with DBC files defining over 300 signals. The operating system efficiently distributed the load across processors and RAM, ensuring smooth operation. Every one of the instances did not end in a fault. However, the overall runtime for a single application instance was slightly longer compared to tests conducted with only one application running, as the operating system faced hardware resource constraints and had to allocate them among the processes.

Chapter 6

Case study

Case study is a detailed description and assessment of a specific situation in the real world created for the purpose of deriving generalizations and other insights from it. A case study can be about an individual, a group of people, an organization, or an event, among other subjects. Unlike experiments, where researchers control and manipulate situations, case studies are considered to be "naturalistic" because subjects are studied in their natural context [23].

In the scope of this thesis, the following case study will delve into the process described in Section 2.2.1, but applied to the new, improved solution utilizing the application and Grafana dashboards. Same parameters as in Section 2.2.3 will be used, ensuring a valid comparison of the new solution to the old one.

6.1 Performed case study

The whole process begins similarly to how it was in the previous state – a person is still needed to physically retrieve data from the machine. However, after this action, things start to differ. The same individual is now capable of performing all other necessary tasks comfortably using the new application on their computer. Thus, the average of 40 minutes for data retrieval from the machine remains, but the *dead period* is completely *eliminated*, resulting in savings of up to 2.5 days on average.

Once the data is downloaded onto the computer, there is no further need for manual operations; only setting the correct paths within the new application is required, taking an average of 2 minutes. Another 30 minutes are therefore saved when subtracting the 5 minutes from the original script setup. The conversion and upload process into the database in the new system runs in the background for only 2 minutes and 15 seconds without aggregation. With aggregation, the time increases to 7 minutes and 19 seconds. Additionally, the application is now capable of translating SAE J1939-TP messages, if they are defined within the DBC files. Grafana is then once again used for displaying the converted data, however with the creation and import of missing dashboards taking only 4 minutes on average.

Altogether, the new process starting with data collection from the machine to visualization in Grafana takes a maximum of *48.25 minutes*, of which 2.25 minutes represent the runtime of the new system for converting and uploading MF4 files with a 100MB dataset containing 21 signals. With aggrega-

tion enabled, the process runtime in the application extends by 5 minutes and 4 seconds to 7.32 minutes. Moreover, all of this can be handled by just one person. Considering a larger volume of data, the new application is naturally more efficient. Processing a 1GB dataset takes the new application 27 minutes and 49 seconds without aggregation, or 1 hour, 36 minutes and 24 seconds with aggregation enabled.

Lastly, the application also features new functionalities, such as the ability to download selected data from the database or generate a JSON model of the dashboard for Grafana. Thanks to their graphical interface and simple design, users can utilize them in just a matter of a few minutes.

Chapter 7

Evaluation

The forementioned case study has provided data that allows for a measurable comparison between the previous system and the new one. This data enables to draw tangible conclusions regarding the performance and efficacy of the two systems.

7.1 Improvements and benefits

The old system relied on multiple personnel and manual processes, resulting in significant time delays and inefficiencies. It required at least two employees to handle data retrieval, processing, and visualization. The process was marred by a "dead period" where no progress was made, leading to substantial delays of up to 2.5 days on average.

In contrast, the new system revolutionizes the workflow by minimizing manual intervention and streamlining operations, eliminating the "dead period" entirely. The new application empowers a single individual to perform all necessary tasks efficiently, eliminating the need for additional personnel and the associated coordination challenges.

Moreover, the new system significantly reduces processing times. Data conversion and upload processes, which previously took considerable time, are now completed swiftly in the background, with minimal user input. This optimization results in a dramatic reduction in the overall processing time, with the entire process of the same source MF4 file dataset taking a maximum of 2.25 minutes, which is *12.25 times faster*, to be exact. With aggregation, the processing speed is accelerated by a factor of 3.8.

Additionally, the new system introduces advanced functionalities that enhance user experience and flexibility. Users can now selectively download data from the database and generate JSON models of dashboards for Grafana effortlessly. These features, coupled with the application's intuitive graphical interface, enable users to accomplish tasks more efficiently in just a matter of minutes.

In conclusion, the new system represents a remarkable improvement over its predecessor. It not only reduces processing times and eliminates inefficiencies but also introduces new functionalities that enhance usability and productivity. By automating manual processes and streamlining operations, the new system optimizes resource utilization and enhances overall efficiency, **saving the company up to 2 days and 5.75 hours per one process that happens several times a month**, not counting the improvements in runtime of the application.

7.2 Drawbacks

The new solution has not introduced any negatively impacting elements into the process. The only remaining drawback is the persistent need for manual data retrieval from the machines.

7.3 Plans for future development

Building upon the mentioned drawback, it is advisable to propose and implement a solution for the future that ensures automatic, wireless transfer of data directly from the machines to the server, where they would be automatically processed.

Moreover, the creation of Grafana dashboards could be completely automated in the future, utilizing its API communicating with Python.

Conclusion

The primary aim of this thesis was to enhance and optimize the process of handling, converting and inspecting data collected from construction machines utilizing the CAN bus at Doosan Bobcat EMEA, s.r.o. company.

In Chapter 2, the work meticulously analyzes the current process, thus providing a solid foundation for evaluating its deficiencies. Based on further analysis of prospective solutions, in Chapter 3 the thesis delineated new requirements, activity diagrams, and use cases. Subsequently, leveraging the data gathered, the Chapter 4 designs a new information system resembling a desktop application with a graphical user interface and data inspection dashboards using the open-source tool Grafana. Emphasis was placed on enhancing efficiency, simplicity, and portability during the design phase.

The resulting application serves as the focal point of the entire process, consolidating all functionalities into one accessible platform for all users. The entire process of collecting MF4 data from the machines to their upload or download from the database is implemented within this tool, made also easily executable from an *.exe* file. Its implementation is discussed in the Chapter 5.

Lastly in this thesis, the Chapter 6 describes a conducted case study utilizing the new information system in real-world operations. The results of this study are subsequently summarized and evaluated in the Chapter 7, contrasting with the previous solution, which was found to be significantly inferior in all possible aspects. The new solution saves the company a substantial amount of time and logistical difficulties, ultimately validating the main goals of this document.

Nevertheless, the evolution of this new system does not conclude here; there is still room for future enhancements, such as the automation of data collection from machines or the complete automation of dashboard creation in Grafana.

Bibliography

1. SMITH, Grant. *What Is CAN Bus* [online]. 2024. [visited on 2024-02-15]. Available from: https://dewesoft.com/blog/what-is-can-bus.
2. VECTOR INFORMATIK GMBH. *SAE J1939* [online]. 2024. [visited on 2024-04-21]. Available from: https://www.vector.com/at/en/know-how/protocols/sae-j1939/.
3. CSS ELECTRONICS. *CAN Bus - The Ultimate Guide*. 8230 Aabyhoej, Denmark, 2023.
4. ASAM E. V. *ASAM MDF* [online]. 2019. [visited on 2024-02-15]. Available from: https://www.asam.net/standards/detail/mdf/.
5. ASAM E. V. *ASAM MDF* [online]. 2019. [visited on 2024-02-15]. Available from: https://www.asam.net/standards/detail/mdf/wiki/.
6. CSS ELECTRONICS. *CANedge2 Docs* [online]. 2024. [visited on 2024-03-02]. Available from: https://canlogger.csselectronics.com/canedge-docs/ce2/introduction.html.
7. GRAFANA LABS. *Grafana Labs Documentation - About Grafana* [online]. 2024. [visited on 2024-03-08]. Available from: https://grafana.com/docs/grafana/latest/introduction/.
8. GRAFANA LABS. *Grafana Labs Documentation - Panels and visualizations* [online]. 2024. [visited on 2024-03-08]. Available from: https://grafana.com/docs/grafana/latest/panels-visualizations/.
9. GRAFANA LABS. *Grafana Labs Documentation - Use dashboards* [online]. 2024. [visited on 2024-03-08]. Available from: https://grafana.com/docs/grafana/latest/dashboards/use-dashboards/.
10. GRAFANA LABS. *Grafana Labs Documentation - Dashboard JSON model* [online]. 2024. [visited on 2024-03-08]. Available from: https://grafana.com/docs/grafana/latest/dashboards/build-dashboards/view-dashboard-json-model/.
11. GRAFANA LABS. *Grafana Labs Documentation - Install Grafana on Windows* [online]. 2024. [visited on 2024-03-08]. Available from: https://grafana.com/docs/grafana/latest/setup-grafana/installation/windows/.

12. GRAFANA LABS. *Grafana Labs Documentation - Manage dashboard permissions* [online]. 2024. [visited on 2024-03-08]. Available from: https://grafana.com/docs/grafana/latest/administration/user-management/manage-dashboard-permissions/.

13. GRAFANA LABS. *Grafana Labs Documentation - Roles and permissions* [online]. 2024. [visited on 2024-03-08]. Available from: https://grafana.com/docs/grafana/latest/administration/roles-and-permissions/.

14. WIEGERS, Karl E.; BEATTY, Joy. *Software Requirements*. Vol. 3, Chapter 8. Understanding user requirements. Redmond, Washington: Microsoft Press, 2013. ISBN 978-0735679665.

15. WIEGERS, Karl E.; BEATTY, Joy. *Software Requirements*. Vol. 3, Chapter 1. The essential software requirement. Redmond, Washington: Microsoft Press, 2013. ISBN 978-0735679665.

16. BASS, Len; DR. CLEMENTS, Paul; KAZMAN, Rick. *Software Architecture in Practise*. Vol. 4, Chapter 1. What is software architecture? Boston, Massachusetts: Addison-Wesley Professional, 2021. ISBN 978-0136886099.

17. OWASP. *Security architecture* [online]. 2024. [visited on 2024-04-22]. Available from: https://owaspsamm.org/model/design/security-architecture/.

18. ZAPPA, Letizia. *What is User Experience? Overview and examples* [online]. 2024. [visited on 2024-05-01]. Available from: https://userreport.com/blog/user-experience/.

19. HASHEMI-POUR, Cameron. *User interface (UI)* [online]. 2024. [visited on 2024-05-01]. Available from: https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI.

20. JUVILER, Jamie. *What Is GUI? Graphical User Interfaces, Explained* [online]. 2024. [visited on 2024-05-01]. Available from: https://blog.hubspot.com/website/what-is-gui.

21. TECHTARGET. *Unit testing* [online]. 2023. [visited on 2024-05-03]. Available from: https://www.techtarget.com/searchsoftwarequality/definition/unit-testing.

22. OMNICONVERT. *User Testing* [online]. 2024. [visited on 2024-05-03]. Available from: https://www.omniconvert.com/what-is/user-testing/.

23. RAIKAR, Sanat Pai. *Case study research* [online]. 2024. [visited on 2024-05-05]. Available from: https://www.britannica.com/science/case-study.

Appendix A

Abbreviations

API Application Programming Interface

BAM Broadcast announce message

CAN Controller area network

CM Connection mode

CSV Comma-separated values

DBC CAN database

ECU Electronic control unit

GB Gigabyte

GUI Graphical user interface

JSON JavaScript Object Notation

MDF Measurement data format

MB Megabyte

MF4 MDF4

SQL Structured query language

TP Transport protocol

UI User interface

UX User experience

XLSX Microsoft Excel Spreadsheet

Appendix B

To-Be activity diagrams

Figure B.1: To-Be activity diagram for data processing subprocess

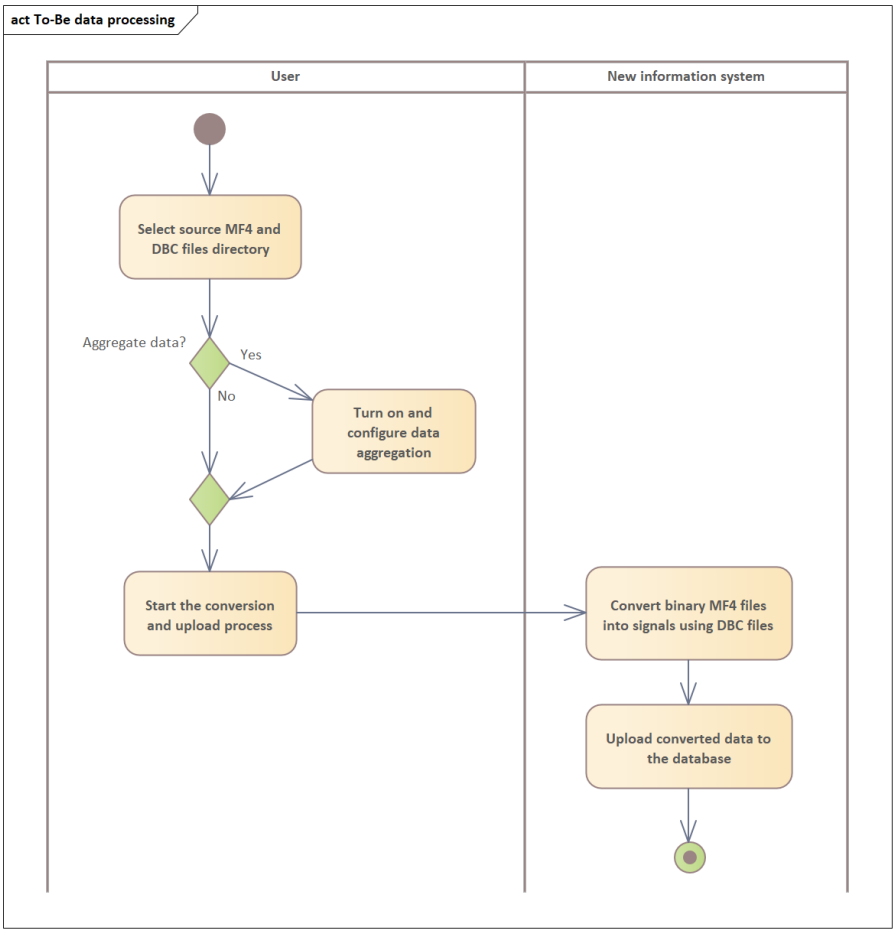


Figure B.2: To-Be activity diagram for data downloading subprocess

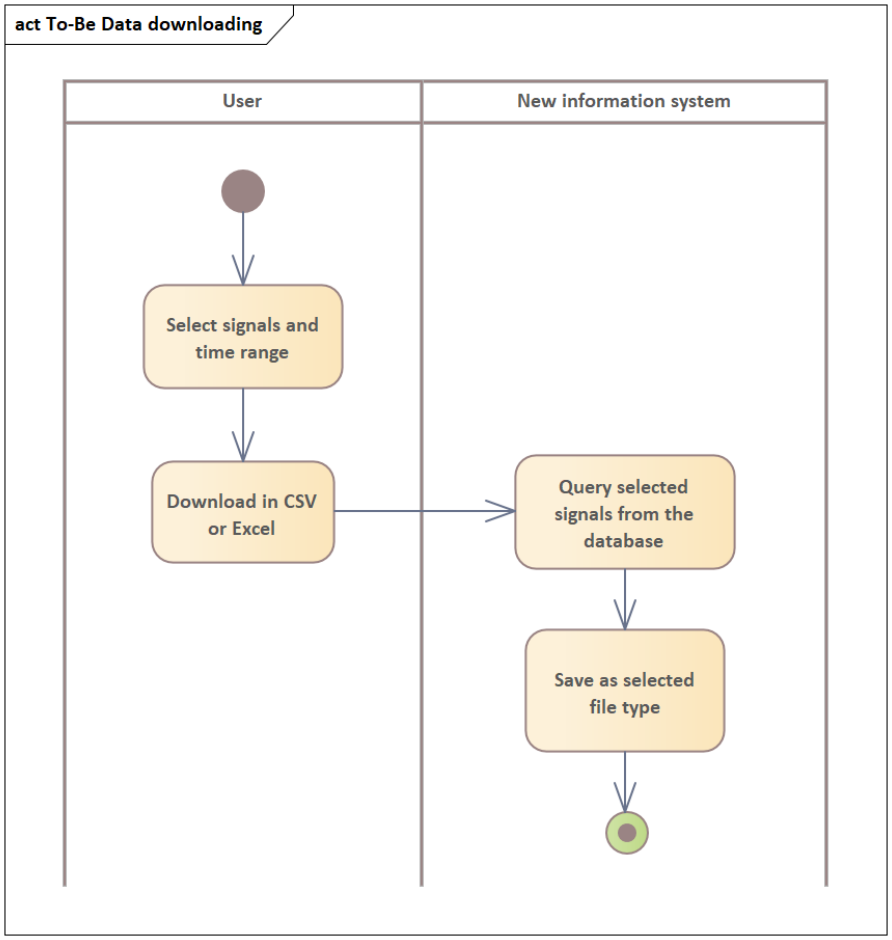
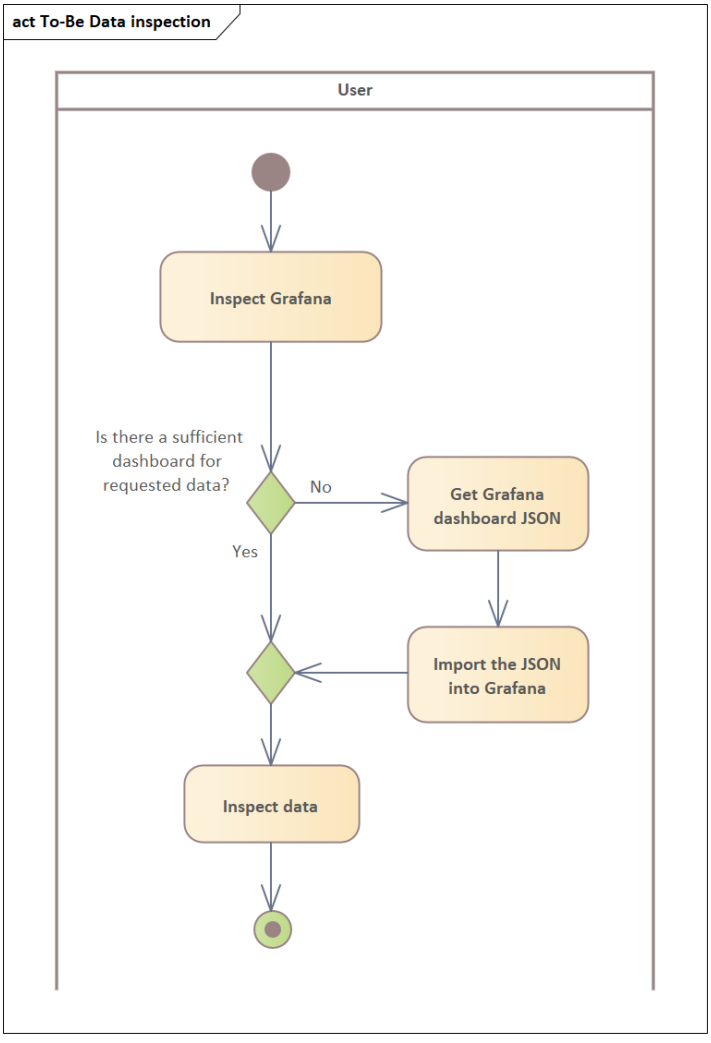


Figure B.3: To-Be activity diagram for data inspection subprocess



Appendix C

To-Be use case diagrams

Figure C.1: To-Be use case diagram for Basic application operations

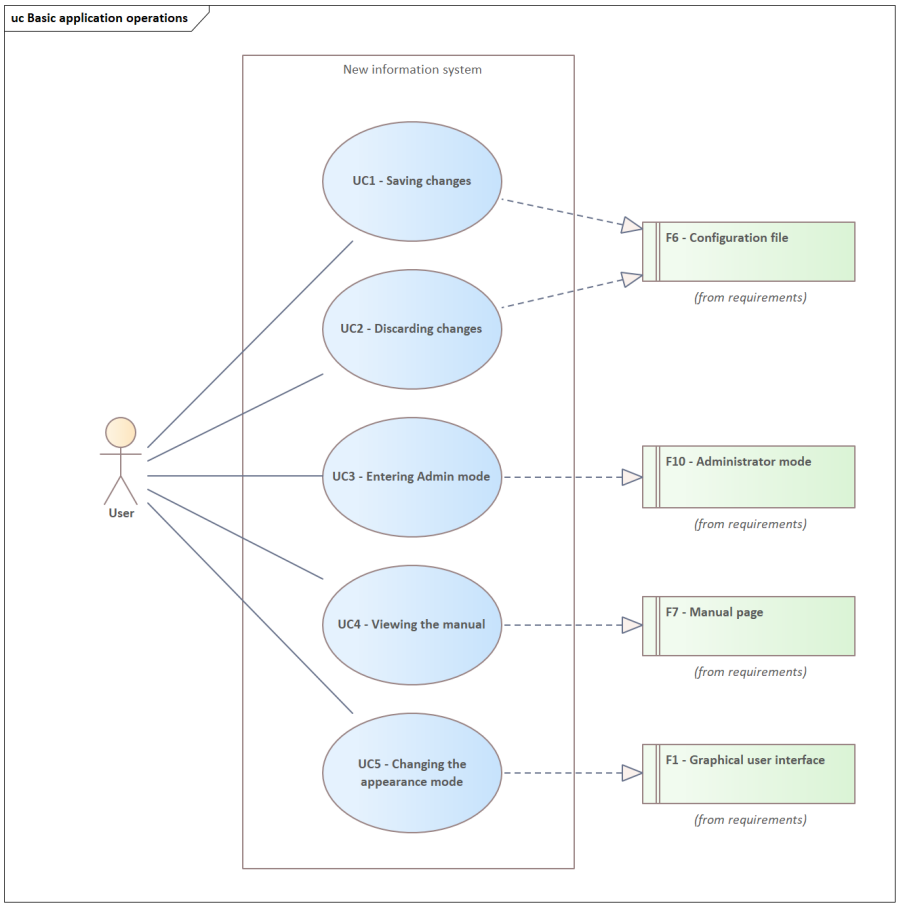


Figure C.2: To-Be use case diagram for Converting and uploading of MF4 data

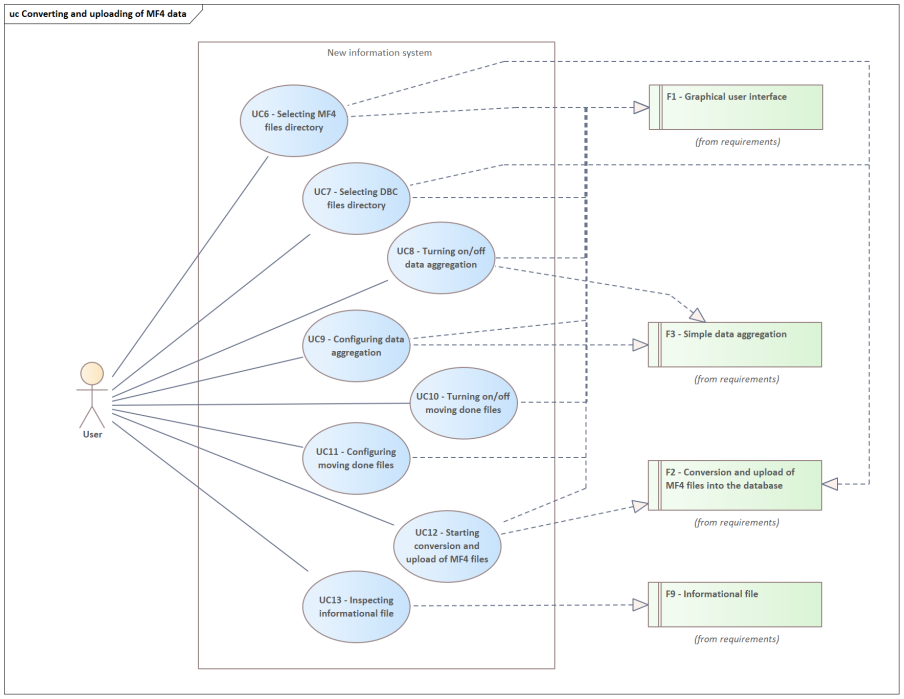


Figure C.3: To-Be use case diagram for Database configuration

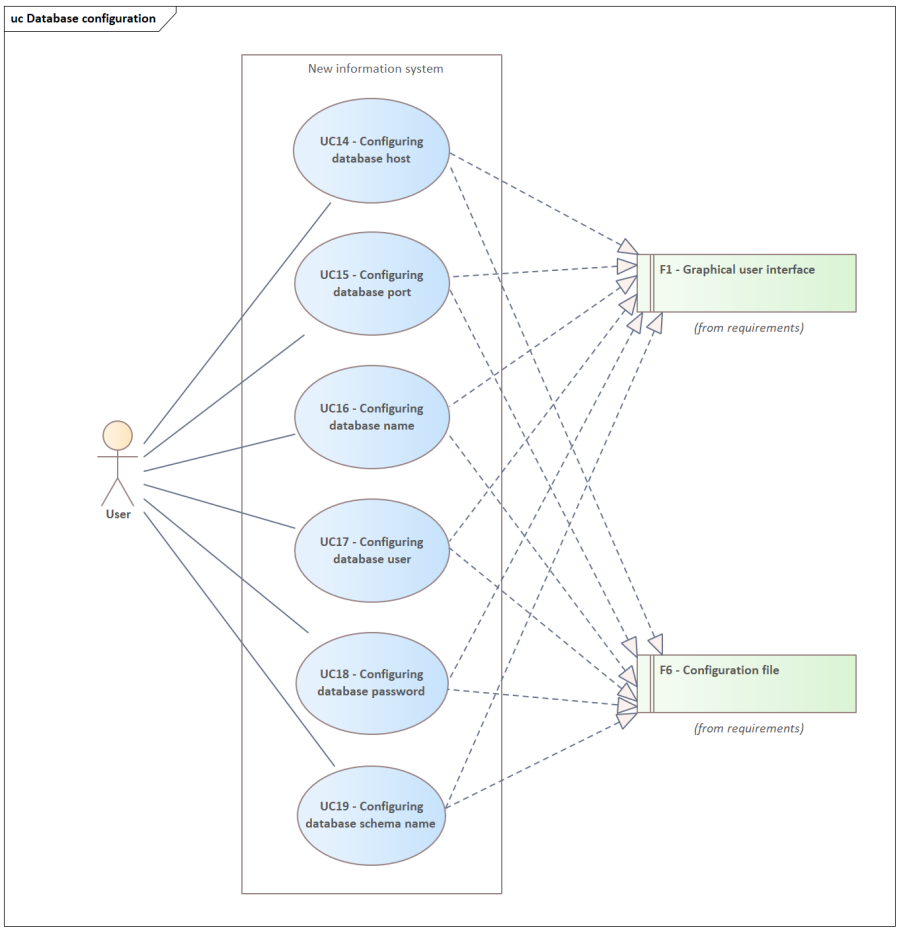


Figure C.4: To-Be use case diagram for Data downloading

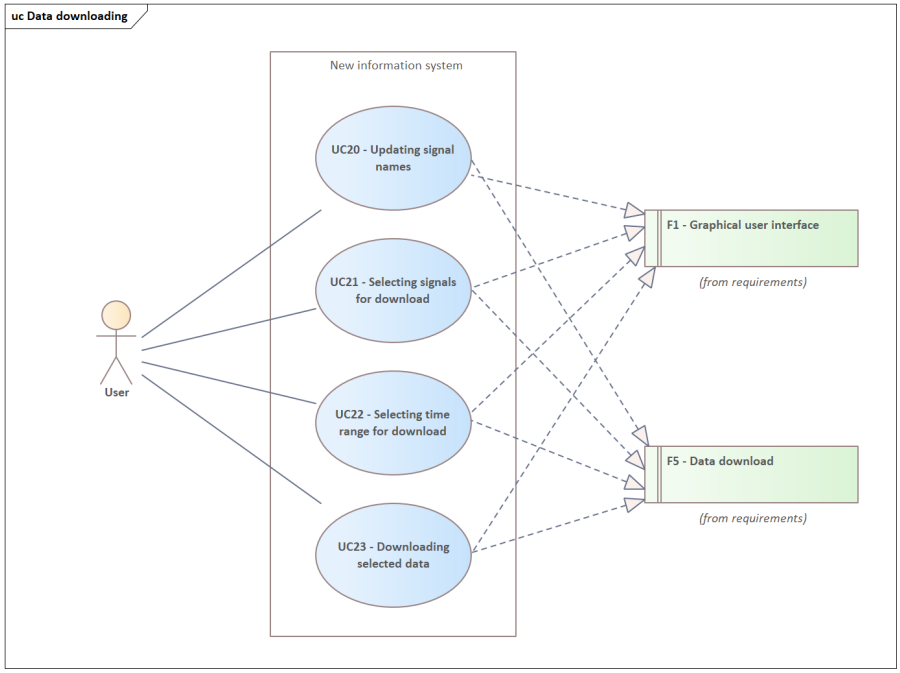


Figure C.5: To-Be use case diagram for Creating Grafana dashboard JSON

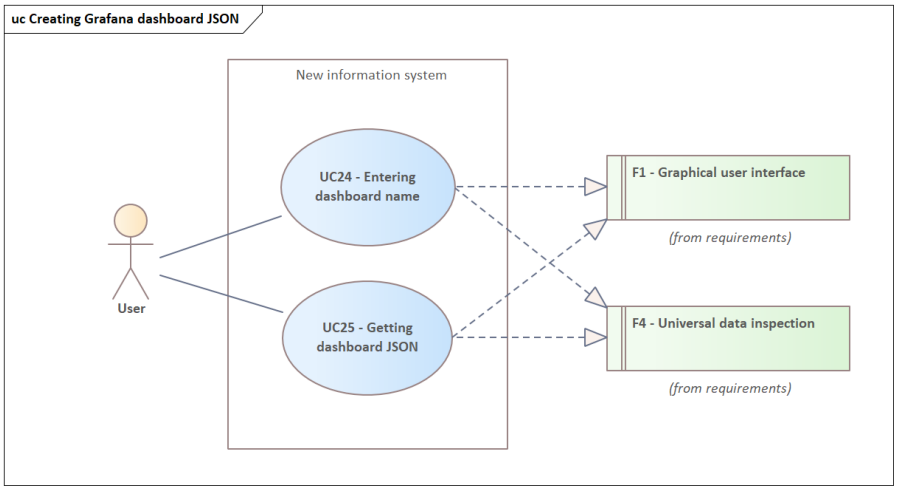
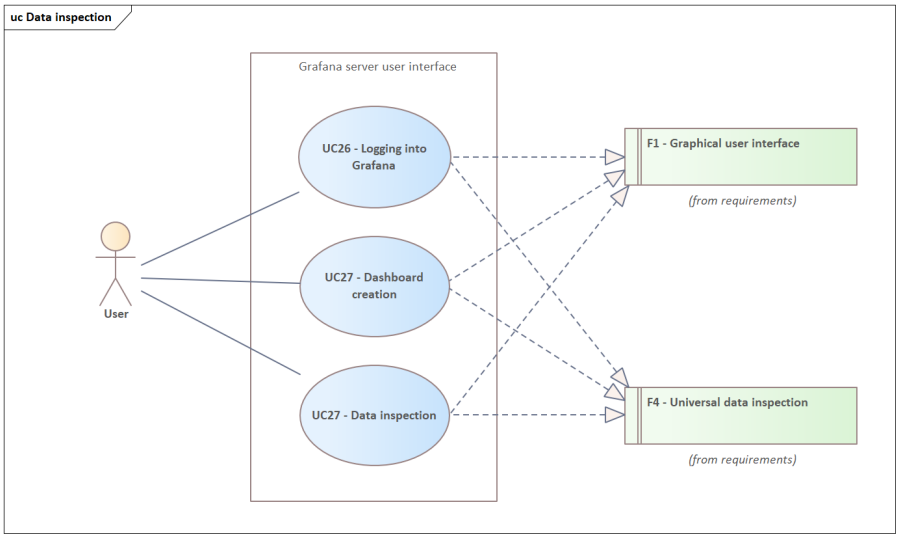


Figure C.6: To-Be use case diagram for Data inspection



Appendix D

Contents of attachments

- `readme.txt` ..... a brief description of the contents
- `exe` ..... directory with the executable application
- `src`
  - `impl` ..... the implementation source files
  - `thesis` ..... the thesis LaTeX source files
    - `fig` ..... used figures
  - `diagrams` ..... Enterprise Architect source files
- `text` ..... the thesis text
  - `thesis.pdf` ..... the thesis text in PDF format