UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Maria Eduarda Bastos Lübke

Conception and Development of a Device for Automatic Monitoring of Ethernet Communication Systems, Based on the Industrial Computer PR21

Florianópolis
2019

Maria Eduarda Bastos Lübke

Conception and Development of a Device for Automatic Monitoring of Ethernet Communication Systems, Based on the Industrial Computer PR21

Relatório submetido à Universidade Federal de Santa Catarina como requisito para a aprovação na disciplina DAS 5511: Projeto de Fim de Curso do curso de Graduação em Engenharia de Controle e Automação.
Orientador(a): Prof. Rômulo Silva de Olveira

Florianópolis
2019

Maria Eduarda Bastos Lübke

Conception and Development of a Device for Automatic Monitoring of Ethernet Communication Systems, Based on the Industrial Computer PR21

Esta monografia foi julgada no contexto da disciplina DAS5511: Projeto de Fim de Curso e aprovada na sua forma final pelo Curso de Engenharia de Controle e Automação.

Florianópolis, 9 de dezembro de 2019

Banca Examinadora:

Christian Kaufmann
Orientador na Empresa
Bosch Rexroth

Rômulo Silva de Oliveira
Orientador no Curso
Universidade Federal de Santa Catarina

Carlos Barros Montez
Avaliador
Universidade Federal de Santa Catarina

Gabriel Paiva de Oliveira
Debatedor
Universidade Federal de Santa Catarina

Thais Juliane Dall'Agnol
Debatedor
Universidade Federal de Santa Catarina

To my parents, whose immense support led me where I am.

Acknowledgements

It is a genuine pleasure to hereby express my gratitude to all people that have given me support during my internship period. It was only possible for me to develop myself and to achieve the results described in this document with the professional, technical and personal aid I have received during my time in Germany.

I would like to thank my supervisor, Christian Kaufmann, and the department for the trust placed in me to develop this project. During my time at Bosch Rexroth, I always felt respected, heard and welcome to the team. I appreciate all the patience and disposition to help and teach me. I am sure that I will be a better professional due to his counseling.

To my University, professors and friends, thank you for enabling this experience. Specially to Amadeu Plácido Neto and Rafael Vendramini Savi, who were my colleagues in University and were also my colleagues at Rexroth. You helped me to get in touch with the company and made this internship possible. Thank you for being my Brazilian family in Europe, those months would not have been the same without you. The best regards to my brazilian friends, who have supported me in so many ways, specially Lara Lucena Zacchi, Marina Bastos Cavallazzi. Special thanks to Alex Cani, Marcelo Menezes Morato, Gustavo Ardigó and Ramon Dettmer for reviewing this document and for the always welcome suggestions.

I am eternally grateful to my parents, Heloísa Helena Bastos Silva Lübke and Vitor Guilherme Lübke, as well as all members of my close family, who have always believed in me and measured no efforts to make my dreams and goals come true. To you, I owe everything.

To my intern colleagues, who have shared more than a department and lunch breaks with me. Specially Asli Yardim, David Riethmann, Eszter Siska, Thomas Kranzioch and Jannik Vogt, thank you for showing me that Germany is not so cold and making me feel like home, even if ten thousand kilometers away from Brazil.

After this year of discovery and learning, the best way I can express my feelings is with a Portuguese word, *obrigada*!

Abstract

Com o objetivo de capturar tráfego Ethernet nas aplicações de clientes da empresa Bosch Rexroth, este projeto busca disponibilizar um dispositivo de fácil instalação e operabilidade que seja capaz de registrar a comunicação em redes industriais. O hardware utilizado é o mini computador industrial IndraControl PR21, que utiliza um sistema operacional Linux. O PR21 é estendido com um módulo de Inputs/Outputs (I/O) com inputs de 24V, que são utilizados para engatilhar ações como iniciar e parar a captura de pacotes no dispositivo. A configuração do sistema é feita de forma automática quando iniciado o programa no dispositivo, de modo que ele interpreta arquivos em formato *.ini* de um USB stick. O software que coordena todas essas tarefas foi desenvolvido na linguagem de programação C++ seguindo uma arquitetura de software modular, abrindo a possibilidade de que diferentes ferramentas de captura e bibliotecas possam ser utilizadas, dependendo da necessidade do cliente. A meta do trabalho ficou definida em iniciar e parar a captura de tráfego Ethernet dependendo do status de certos inputs, que podem ser ativados manualmente (como no caso deste trabalho, através de botões), ou por sinais de saída de outros dispositivos, como CLPs. Os pacotes capturados são então gravados em uma localidade definida pelo arquivo de configuração, seja ela no USB stick ou mesmo dentro do PR21, para facilitar propósitos de análise.

Palavras-chave: CLP; Comunicação; Tempo real; Monitoramento; Ethernet.

Abstract

To capture Ethernet traffic in Rexroth costumer applications, an easy-to-install and operable recording hardware should be made available for use. The hardware used is the mini IPC IndraControl PR21, a compact industrial graded Linux target. The PR21 is extended with the aid of an Input/Output (I/O) module with 24V inputs. The configuring of the target is triggered automatically with an USB stick via configuration format files. The software which coordinates those tasks is developed in C++ with a modular architecture, making it possible to use different capture tools and other configurations to support distinct costumer needs. The aim of the work is, in short words, to start and stop the capture of Ethernet traffic depending on the status of certain inputs, which can be set manually (e.g. through buttons) or through outputs of other devices, like PLCs. Captured packets shall be stored in an USB stick in order to simplify analysis purposes.

Keywords: PLC; Real-time Ethernet communication; Monitoring; Sercos III.

List of Figures

List of Tables

List of abbreviations and acronyms

API	Application-Programming-Interface
AT	Acknowledge telegram
CPU	Central processing unit
GUI	Graphical user interface
IP	Internet protocol
IPC	Industrial PC
IT	Information technology
I/O	Input/Output
LED	Light-Emitting Diode
MAC	Media access control
MDT	Master data telegram
OS	Operational system
PC	Personal computer
PLC	Programmable logic controller
REST	Representation State Transfer
Sercos	Serial Real-time Communication System
TCP	Transmission control protocol
UC	Unified communication
UCC	Unified communication channel
UDP	User datagram protocol
XML	Extensible markup language

Contents

1 INTRODUCTION

In the context of industrial applications, this project develops and discusses both software and hardware work fronts for the development of a device for automatic monitoring of Ethernet communication systems. The construction of this device is based on the industrial computer PR21 (see Figure 1).

Figure 1 – Box-PC PR21

Source: Bosch Rexroth

The presented work is a Bachelor Monograph unfolded during the period of internship inside the company Bosch Rexroth AG, specifically in the Software Development department in Lohr am Main, Germany.

The project was developed looking at the company and the university perspectives, producing a relevant work that improves internal aspects of Bosch Rexroth and covers different areas of study inside the Control and Automation course at Universidade Federal de Santa Catarina (UFSC), such as embedded programming in PLCs, software design and network protocols.

1.1 Background

Programmable logic controllers (PLCs) are industrial computers adapted for the control of manufacturing processes, such as assembly lines, robotic devices, or any operation that requires high reliability, ease of programming and process fault diagnosis [1]. They are usually designed to operate accurately in harsh usage environments and conditions, such as strong vibrations, extreme temperatures and wet or dusty facilities.

Nowadays, PLCs are widely used in the most varied range of industrial areas and other applications. Rexroth's automation solutions are developed to be flexible and intelligent in order to ensure that the machines used are functional and future-proof.

Industrial Ethernet is often chosen to tie the parts together in manufacturing environments. Ethernet is a family of computer networking technologies that was first standardized in 1983 as IEEE 802.3 [2], but is still widely used due to its improving shared medium, the wide acceptance in devices, and the support to virtually all popular network protocols. Those attributes corroborate with the principles of Industry 4.0 [3], as the automation market is highly fragmented and there are plenty of PLC vendors, data bus and Ethernet protocols being supported. The interoperability brought enlarges the limits of PLC networks expansion and the number of possible applications.

The concept of Industry 4.0 puts a spotlight on data in a broad sense, being important to point out that only a very small percentage of industrial data is currently used in a way that makes sense or adds value. Therefore, it is clear to see why monitoring the network traffic of industrial applications can be crucial to the performance and efficiency improvement of the factories of the future.

More than knowing what is happening with a system through measured values, output messages and errors, it is necessary to understand what is going through every part of it, and packet analysis is an important tool to reach the goal of fully perceiving the internal aspects of a structure as complex as industrial ones can be. Through the packets, it is possible to see the signals sent from one device to the other and its respective answer. This brings awareness of where problems could be caused, in which parts of the system there has been an incorrect message, what kind of content each device emits and much more.

1.2 Motivation

Bosch Rexroth is nowadays one of the main suppliers of PLCs, Drives and related technology in the business, possessing an assortment of customers worldwide with multiple needs and requirements. The company also provides services and aid to the customers on the post-sale process, as any problem or questions can be brought up directly to the team responsible for the development. This can also mean meeting up the clients in their environment to help solve complications and adverse conditions. Thus, sending employees to directly support customers costs time and money for the company, especially knowing that the client plant can be basically anywhere in the world.

The study case of the work here described is about the costumers whose problems can be investigated and solved by adding a network tap to the system in order to monitor the packets travelling through the system. A network tap denotes a system that monitors

events (messages) on a local network with the goal of facilitating the analysis of the network traffic.

Sporadic errors can occur in the costumers' production machines, and since they are not monitored by humans the whole time, having a register of the packets with the time the errors happened, as well as other information, can be crucial on the understanding of the problem. This information can be acquired via a network tap, and the work here described proposes that this system and these functions be made automatic and versatile for easy integration with current functioning plants.

1.3 Contribution

The main goal of the project is to have a device, needing minimal configuration, that can be mounted in a running machine system without a big effort and that is safe for harsh environments, meaning that the device can be sent to the client's plant without being necessary to send an entire team to do the job of catching packets in a network.

The device should also be easily configured, since quite often the costumer does not have an employee who is a specialist on networks, and the tools used for the capture and analysis of packets data are not intuitive, requiring special knowledge. Thinking of that, the tap system should encapsulate all complicated configuration, leaving for the end-user a simple interface where data about the plant should be inserted.

Considering that most costumers work on harsh usage environments, the use of industrial computers as network tap systems are necessary replacements to regular laptops, so that the device is, besides functional and easy to configure, also robust.

Given the background, motivation and goal statements exposed in this Chapter, the continuation of the document is organized in the following way: Chapter 2 brings a quick briefing regarding Rexroth, the company where the project has been developed and its business.

Chapter 3 contains relevant information for the understanding of the problem and the proposed solution, providing the theoretical aspects used throughout the rest of the document.

Chapter 4 expands the problem and the case of study the project aims to solve, pointing the difficulties and complications that might exist. In Chapter 4 is also shown the planning for the solution and how it was modelled.

Chapter 5 presents all the steps taken to implement the solution as a whole. It is the documentation of what was developed and which parts already existed inside the company.

Chapter 6 showcases the results obtained with the development stage achieved

through this project. Moreover, it exposes the successes and failures of the work.

Lastly, Chapter 7 brings up the discussion on final findings and future prospects, as well as improvements that can be done and possibilities enhanced by this project.

2 THE COMPANY AND BUSINESS

The internship respective to this document was carried in the Connect X team from the department DC-AE/ESW2 (Drive and Control Technology - Automation and Electrification Solutions /Engineering Software) at Bosch Rexroth facilities in Lohr Am Main, Germany. An overview on the company and its main products is given on this Chapter. More information about Rexroth can be found at [4].

Figure 2 – Official logo of Bosch Rexroth AG

Source: Bosch Rexroth archive

2.1 Bosch Rexroth - A Bosch Company

Bosch Rexroth is one of the world's leading providers of drive and control technologies with more than 32,000 employees worldwide. The teams are set up to develop safe, efficient, intelligent and high-performance solutions for the areas of Factory Automation, Mobile Applications, as well as Machinery Applications and Engineering. With its cross-technology range, digital services and comprehensive service, Bosch Rexroth acts as a reliable partner for machine manufacturers and users. The company develops, produces and sells its components and system solutions in more than 80 countries.

The history of the company starts in 1795, when Charles Ludwig Rexroth puts a water-powered hammer mill into operation in Elsavatal (Spessart). The company Rexroth established itself in Lohr am Main, and during the years that followed, went through some reorganizations, like becoming a wholly owned subsidiary of Mannesmann AG in 1975, changing its name to Mannesmann Rexroth AG.

The technology provided by the company was already one of the leading in the market when in 2001, Bosch Automation Technology and Mannesmann Rexroth AG merged to form what we know now as Bosch Rexroth AG. About its products it is interesting to enlighten the 1994 revolution in the printing industry caused by Rexroth's Synax 200 automation system. It changed the old printing system that consisted in a "royal wave" which moved and synchronized the individual printing units with great mechanical effort.

This system also required that the whole production line was stopped when any minor changes or repairs had to be made.

The Synax 200 replaced these vulnerable mechanics and since that point, intelligent individual drives, that communicate with each other via the Sercos automation bus, are driving the rollers and rollers without shafts, see [5].

Thanks to the shaftless drive technology individual parts of the press can be turned on and off for repairs with no waste paper produced. In addition, the friction is lower due to the eliminated mechanical parts, reducing energy consumption.

Regarding other products, in 2009, Rexroth creates a hydraulic pump called Sytronix, that combined with a controlled electric motor only generates as much hydraulic power as is needed for a given task or machine cycle. Sytronix variable-speed pump drives reduce energy consumption by up to 80 percent.

Bosch Rexroth in 2014 launches the IndraControl XM control platform, which offers more intelligence for Industry 4.0 applications and links it with extremely fast processing of signals. Also in 2014, Rexroth becomes heading operator of Industry 4.0: on the networked multi-product line in the Homburg/Saar Bosch Rexroth plant, employees were installing 200 different variants of highly-efficient hydraulic valves in nine intelligent stations.

The company has become synonymous of tailored solutions. For Rexroth, every industry has specific needs and requires many technologies for one solution. The drive and control division develops solutions for Mobile Applications like construction machinery and agricultural and forestry machinery. For Machinery Applications and Engineering it is possible to mention the following applications: marine and offshore, motion simulation, energy technology, testing technology and presses. For Factory Automation, Bosch Rexroth develops technologies for electronics and manufacturing, printing and paper, automotive, machine tools, as well as many other industries.

Data from 2018 show that in this year, Bosch Rexroth had an amount of 6.2 billion euros on total sales and 327.6 million euros on total research and development. The company invests in its technologies and therefore customers profit from the resources and innovative strength of a global player, flexible structures and an understanding of local needs.

.

3 THEORY

In pursuit of a better understanding of the project executed inside the company, this Chapter goes into the details of the technologies adopted in this work, from hardware to software. All of the technologies and equipment used in this project were defined by the team guidelines, for being used in other works and for being developed inside the company, making it easy to adapt the equipment if needed and for easier integration with other teams. Also, the guidelines here used are chosen to make the project as versatile and general as possible to integrate with different applications in industry.

3.1 Industrial Ethernet

Industry 4.0 represents the current trend of automation and data exchange in manufacturing processes. It incorporates a variety of disruptive technologies, such as machine learning, predictive analytics, autonomous robotics and cloud computing. What they all have in common, is that they are entirely related to digitalization, blurring the lines between the spheres of virtual and physical world. It is expected that those technologies combined deliver higher operational efficiency and achieve unprecedented levels of scalability and mass customization, as discussed by [6] and [7].

In order to achieve those forecasts and to promote the desired interconnected systems, it is vital that there is a universal network technology that works across a vast number of devices and locations. Inserted in this context, it can be said that Ethernet has become the technology of choice to connect and tie the parts together in manufacturing environments.

Ethernet is a family of computer networking technologies that was first standardized in 1983 as IEEE 802.3 [2], but is still widely used due to its improving shared medium, the wide acceptance in devices, and the support to virtually all popular network protocols. Those attributes corroborate with the principles of Industry 4.0, as the automation market is highly fragmented and there are plenty of PLC vendors, data bus and Ethernet protocols being supported. The interoperability brought enlarges the limits of PLC networks' expansion.

Ethernet and, specifically, industrial Ethernet have recently become popular industry terms in the manufacturing world. While similar, they both offer different characteristics and benefits [8]. Regular Ethernet, suitable for office automation, does not meet the standards of industrial applications, that require more rigorous performance and durability [9].

Knowing that the factories environment can present harsh conditions, like being

densely packed with industrial machinery, facing extreme hot or cold conditions, it can be inferred that industrial Ethernet must follow more rigorous standards than regular Ethernet. It has to offer protection against shock, vibration and to be shielded in a way that electromagnetic frequencies cannot penetrate the cables. Also, facing intensive use, it should follow protocols developed specially for it, like EtherCat and Sercos III, in order to ensure specific manufacturing data is correctly sent and received.

3.2 PR21

The PR21 is a compact industrial graded Linux target. It is manufactured by Bosch Rexroth as a compact Box Pc from the PR series, being available in a variety of different housing versions. The description from the company says: "Various interfaces can be used to connect the spatially separated DR multi-touch displays. By using the latest processor generation with low power dissipation, the device does not require a fan. High performance, in addition to a wide range of expansion options and standard interfaces, including real-time Ethernet communication, make the PR Box PC the ideal platform solution for all HMI-based applications. The integrated TPM 2.0 chip provides necessary security in the IoT environment, making the robust devices ideal for Industry 4.0." [10]

Figure 3 shows the Box PCs from PR series. The technical details and features from the product are defined in [10] and include:

- CPU: Intel ATOM E3815
- RAM: 4 GB
- Operating system: Linux Ubuntu Core
- Extension slots: 1 x mPCIe interface module or mSATA mass memory
- Mass memory: 32 G eMMC onboard

Figure 3 – Box PCs - PR series

Source: Bosch Rexroth [10]

The PR21 is the choice of hardware for this project due to its simplicity and robustness, along with the possibility of mounting a 24V I/O module for receiving the inputs. It gives the necessary flexibility required by the application, and yet is complex enough to carry all the necessary functionalities.

3.3 Sercos

The PR21 is meant to be inserted in contexts where the Sercos (Serial Real-Time Communication System) bus system is used between the devices and read through its telegrams. Therefore, it is relevant to explain how it works and how the telegrams are constituted.

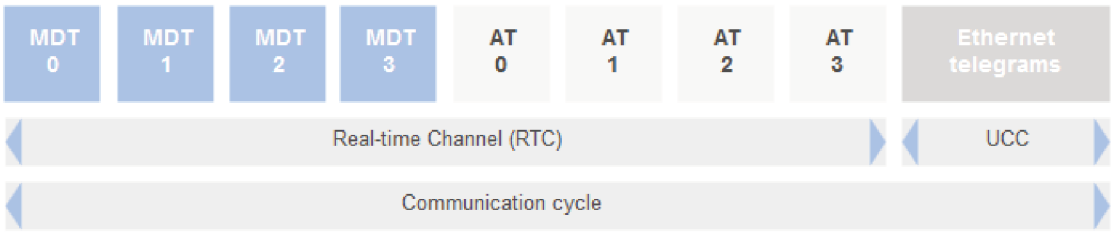
Sercos is one of the main communication protocols used in industrial applications, specially in mechanical systems, due to its efficient and deterministic protocol that is based on an optical transmission system for immunity to high noise. Sercos II, the predecessor system to Sercos III, was based only in optical transmission. Nowadays, the used version is Sercos III, which is based on Ethernet technology, therefore being useful for Ethernet based systems. It specifies over 700 standardized parameters meant to describe precisely the interaction between control systems, drives and other peripheral devices using universal semantics. This creates a manufacturer-independent communication solution, that today is being used on a daily basis in over 500,000 applications. The protocol combines standard Ethernet with the need for real-time accuracy in automation engineering.

3.3.1 Transmission principle

The communication is based on a time slot process with cyclical transmissions of telegrams based on a Master-Slave principle. It is divided in two time slots (channels) so the real-time demands are fulfilled despite using non-deterministic Ethernet.

A representation of the communication cycle, including both types of telegrams, can be seen in Figure 4. Real-time telegrams are transmitted through a collision-free real-time channel. Parallel to this real-time channel, a unified communication (UC) channel can be configured, in which all other Ethernet telegrams and IP-based protocols such as TCP/IP and UDP/IP can be transmitted, see [11]. Cycle times and the division of the bandwidth or bus cycle in the real-time and UC channels can be adjusted for each application. With Sercos, real-time data is sent according to the IEEE 802.3 standard in cyclical telegrams with Ethernet protocol type "0x88CD". The exchanged data is addressable via standardized functional groups, classes and profiles.

Figure 4 – Structure of Sercos communication cycle



Source: Personal archive based on Sercos archive

Sercos differentiates between the following kinds of telegrams:

- Master Data Telegram (MDT): The master sends schedule data to the slave devices.
- Acknowledge Telegram (AT): The slaves send their status data to the master and to other slave devices.

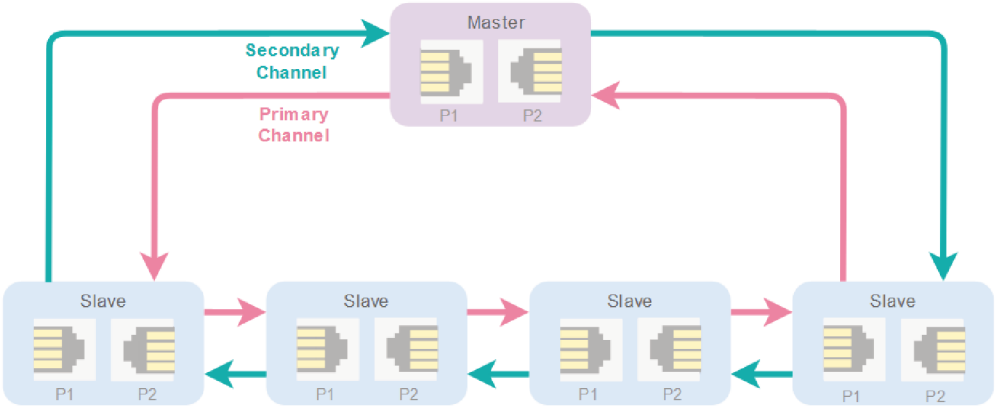
3.3.2 Topology

The connected devices are recognized in the initialization phase (phase start-up with communication phases CP0 – CP4), and they are addressed and configured for each application. Each slave is assigned a device channel in the MDT and the AT, which the slave either uses to read from or to write into. Depending on the amount of data, several MDT and several AT telegrams are sent by the master per communication cycle. The telegram transports data from device to device. The relevant schedule data is read at each device or the required status data is written in.

With the addition of the proposed network tapping device, the telegrams sent by the master to other devices will reach the PR21's adequate port and there it can be read and saved.

In the way used in the scope of this project, the protocol can be synthesized by Figure 5. P1 and P2 represent the ports in the devices, those devices divided between master and slaves.

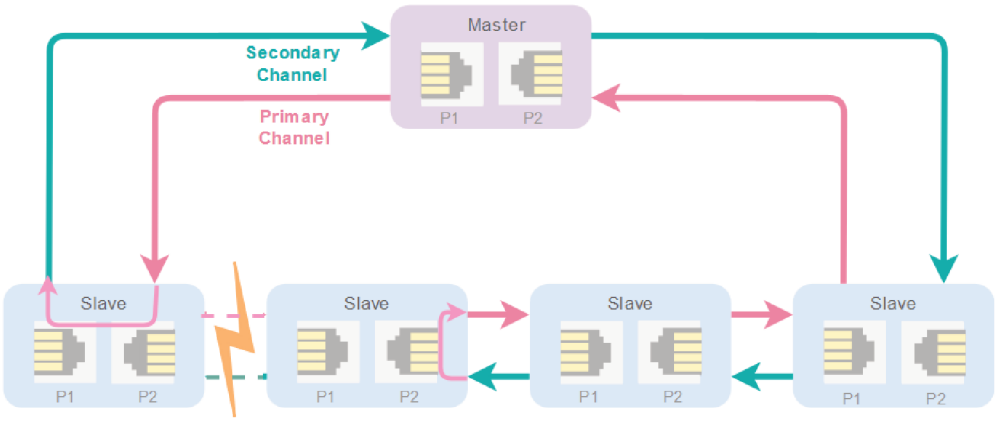
Figure 5 – Sercos ring topology



Source: Personal archive based on Sercos archive

Figure 5 shows the topology of this communication protocol, for which two channels are used. The Primary channel and the Secondary channel together in the way depicted represent hardware with redundancy. The Sercos network forms a ring, that in addition to having all the benefits of a line topology, it provides the redundant cabling. This means more security for the application, as the ring can be broken and the communication will not suffer a breakdown or loss of synchronization. It happens because the master feeds the ring in both directions, therefore, the data gets analyzed by both channels 1 and 2 [12]. Figure 6 depicts the scenario where the ring is broken (where the bolt is drawn) and yet communication remains happening, as described.

Figure 6 – Sercos broken ring



Source: Personal archive based on Sercos archive

3.4 Software architecture

Considering that this work is heavily based on software development, this Section introduces the strategy taken to find the most elegant and flexible programming solution. The book *Clean Architecture: A Craftsman's Guide to Software Structure and Design* includes relevant base knowledge on this topic and is thoroughly used here to justify the chosen development approaches.

In pursuance of turning this software project into a modular, easily extendable tool, object orientation comes as a useful resource. Object orientation provides simple and effective encapsulation of data and function. As a result, a line can be drawn around a cohesive set of data and functions. Using this paradigm allows to impose discipline on indirect transfer of control, as seen in [13].

When talking about a system's architecture, one can say that the architecture is the shape given to that system by those who build it. The form of that shape is in the division of that system into components, the arrangement of those components, and the ways in which those components communicate with each other. The purpose of that shape is to facilitate the development, deployment, operation, and maintenance of the software system contained within it. By separating the system into components, and isolating those components through stable interfaces, it is possible to illuminate the pathways for future features and greatly reduce the risk of inadvertent breakage.

The concept of modularity comes with the importance of making the software flexible enough to manage different tools and frameworks. Using a house metaphor, it can be said that the first concern of the house architect is to make sure that the house is usable. Not to ensure that the house is made of bricks. Indeed, the architect takes pains to ensure that the homeowner can make decisions about the exterior material (bricks, stone,

or cedar) later, after the plans ensure that the use cases are met [13]. Same applies to software modular architecture, where the most important aspects are functionality and requirements fulfillment, not the tools that can be used inside it.

Moreover, modularity permits abstractions to be reused, if modules are developed with sufficiently general interfaces. This reduces design, programming, testing, documentation, and maintenance costs. It permits code to go through all development steps more easily, since code is divided into manageable, comprehensible portions [14].

3.5 Network monitoring

The most notable trend in manufacturing over the past years is probably the move towards networks at all levels. At lower levels in the factory infrastructure, networks provide higher reliability, visibility, diagnosability, and enable capabilities such as distributed control, diagnostics, safety, and device interoperability. At higher levels, networks can leverage internet services to enable factory-wide automated scheduling, supervisory control, and diagnostics; improve data storage and visibility; and open the door to e-manufacturing [15]. With this type of knowledge in mind, it is almost implicit that network monitoring plays an important part in maintaining the quality standards of industrial applications.

Network monitoring is the process of constantly observing a chain of components in order to identify defective machinery, to collect statistics and to gather relevant data about system, which can be generalized as simply to map a network and its packets. The packets that go through it help on the comprehension of notable events, may those be desired or not. In the direction of this belief, there are several different packet analyzers available in the current market with varied features and goals, as described by [16].

3.5.1 Tools

With the plurality of nowadays networks, comes a multiple range of tools for packet analyzing and network monitoring. From those tools, *Wireshark* deserves a highlight, as it is the world's most popular network analyzer. It is a very powerful tool that provides network and upper layer protocols information about data captured in a network [16]. Although, due to its complexity, other tools took place in this project: *TCP dump* and *Sercos Monitor*.

3.5.1.1 TCP dump

TCP dump is a common packet analyzer that runs under the command line. It allows the user to display TCP/IP and other packets being transmitted or received over a network to which the computer is attached [17]. It works on most UNIX systems using a library called *libpcap*.

Even though it is a rather simple tool, TCP dump has multiple configurable parameters, which can personalize the tool in order to fulfill specific requirements, such as storing the captured packets in a file of *.pcap* type (*-w* flag), capturing only packets from a certain interface (*-i* flag) and many others, that can be found in [18].

In a general way, the TCP dump command with all its options is:

```
tcpdump [ -AbdDefhHIJKlLnNOpqStuUvxX#] [ -B buffer_size ]
```

3.5.1.2 Sercos Monitor

The Sercos Monitor is a powerful diagnosis tool for Sercos III networks which is available for free download from the Sercos website. It allows a comprehensive and detailed analysis of data traffic in Sercos III networks.

The tool allows a retrospective evaluation of network records saved in *.pcap* format, as well as real-time analysis of network traffic. The user-friendly interface and overview functions which are characteristic of Sercos III networks, such as topology, communication phases and service channel transfers, allow the analysis process to be started quickly and in a targeted manner. Various views and filters allow an evaluation of Sercos III real-time telegrams and other Ethernet telegrams which is tailored to meet the user's requirements. If required, the functionality of the Sercos Monitor can also be expanded with user-specific plug-ins [19].

Furthermore, Sercos Monitor provides a Web API, named Monitor Core, which is supported by the REST (Representation State Transfer) model in a way that is possible to integrate different types of applications through methods like GET and POST.

3.6 Chapter summary

In resume, this Chapter brings to light important concepts and technologies that will be approached in further Chapters, in order to make the understanding of other concepts easier and more fluid. It is presented here a discussion about industrial Ethernet and its importance to today's industry, as well as the context the protocols are inserted. Also, the PR21, the hardware device used in this project is introduced in its details and Sercos III protocol is explained with its principles and functionalities. The software architecture modelling is brought up to serve as explanation and justification of the program developed, which modelling will be approached in the next Chapter. The capture tools inserted in the scope are also mentioned along their specificities. With all this information in mind, it is now possible to discuss and introduce the modelling strategies used in this project, detailing the problem and determining how the solution should be carried.

4 MODELLING

The detailed description of the problem's scope, as well as the tasks that were listed in order to achieve the main goals of the project, are introduced in this Chapter. Also presented here is an overview of the proposed solution, which is of great importance to this work, as it aids on the understanding of the implementation and the results that can be seen on further Chapters.

The modelling of the work was defined along with the company supervisor, Mr. Christian Kaufmann, in order to fulfill both the company and the university requirements for this project.

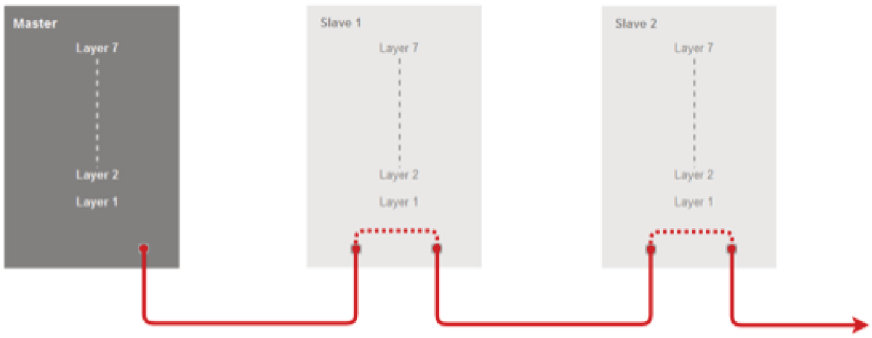
4.1 Problem overview

In order to fully clarify the work done, it is necessary to explain in details why this project is important. In other words, to describe the problem and build the logical path taken to the solution.

As mentioned previously, the need of monitoring the telegrams between two or more devices came directly from observing costumer maintenance and how long this procedure took when done manually. Errors can occur sporadically in the costumers' production machines, and for humans to observe those errors and define the exact cause of them is a task that requires a significant amount of work and time. This happens because those machines and controllers are inserted in complex systems with multiple layers where failures might occur. In a traditional Master/Slave topology, the controllers are set as in Figure 7. The image shows the Master and the Slaves with their 7 internal communication OSI layers [20]. The communication between them in Figure 7 is illustrated by the red continuous line, and it can be seen in this topology that only adjacent nodes are interconnected. The doted red line here represents the internal slave communication from the receiving port to the port that will transmit the master message to the next slave. Simply by describing Figure 7 and its characteristics, several error sources can already be brought up by the way the system is mounted:

- Communication errors between the 7 OSI layers [20];
- Physical errors related to the cables;
- Errors related to FPGA problems;
- Errors in the telegrams the Master sends to the Slaves.

Figure 7 – Master/Slave system representation



Source: Personal archive

Regarding the last form of error, it is known that to occur because first the master sends configuration telegrams to the slaves, and, since the Ethernet protocol here used is Sercos, the master does not send one telegram for each slave. Instead, it sends one telegram for all of them, and the slaves are aware of which part of the frame belongs to them and they are allowed to modify. In reality, this can also be an error source because a defective telegram sent from the master can wrongly configure the slaves, and even when the configuration was done correctly, a slave can write in a part of the frame that does not belong to it, causing trouble for other Slaves to reply to the master.

As introduced in Section 1.2, Rexroth has responsibility on aiding costumers with problems and adverse conditions, and when the error sources are so varied like the ones mentioned in this Section, solving those issues can demand a very high effort, as the team involved should examine which device, from which vendor (because applications often have devices from different vendors) is causing the problems. This is important for the company because when it is the case of applying financial penalties for the malfunctioning device, the company which caused the problem should be the one to pay for the damage. For costumers whose problems can be investigated with a network tap, instead of doing the process with a human team, it was proposed that it could be done automatically in order to save time and money for the company and its employees. That would mean facilitating the analysis of network traffic and consequently finding solutions faster and more efficiently.

The idea then is to have a device that is easily mounted in a running system. The device should be triggered externally or internally, and once the trigger is activated, it should capture the telegrams going through the network. The principle here is to store only the telegrams that are interesting for analyzing what is happening inside the system in a certain moment of the run-time. This scheme turns system inspections much faster, because the number of telegrams per minute increases along with the system complexity,

therefore increasing the size of the sample to be analyzed. For instance, when working with a 1ms cycle time, at least 2000 telegrams have to be capture and analyzed per second of capture time. When storing only the telegrams in between the triggered period of time, the problem decreases in difficulty.

The understanding of costumer plants is based on experience and Rexroth's expertise, and through the years developers have figured that taking regular PCs or laptops to those plants could not be the most adequate solution. Rexroth's costumers are divided in many different fields and areas of interest, like marine and offshore applications, printing and agricultural machinery and much more. What those environments have in common was taken in consideration when planning the project here described, and it was a consensus that the device should be able to work on harsh conditions, such as the ones with dust and high noises. Besides that, standard PCs need 110V or 230V, depending on which country they are used, and industrial environments present easy to find 24V, since almost all automation systems work with this voltage.

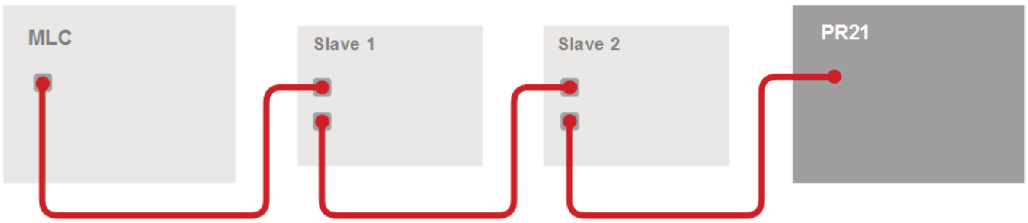
In the possession of all this information, there is the indication of three main pillars for the development of the device: Automation, Easiness of Usage and Robustness. Those are taken as guidelines and are of crucial importance to the implementation of the project, since they matter not only for the choice of technologies to be used, but also on the strategies of approach to the problem.

For instance, the choice of the PR21 as the computer to be used is based on the need of robustness and simplicity stipulated on this project. The alignment of it with a 24V input/output module allows this relatively small structure to have all the functionalities necessary to work as a network tap in harsh environments. Adding the structure to a simple on-board Linux OS complements the device and supports the needed softwares.

Looking at the mounting of the structure in the system, three possibilities were brought up at first:

1. For the mounting shown in Figure 8, the PR21 is added at the end of the network line, which is positive because it is easy to mount and it is possible to do it on a running machine. On the other hand, the con of this mounting is that the PR21 only receives one-sided messages, meaning that it does not get to know what the slaves are sending back to the master. This can limit in certain ways the analysis of the telegrams going through the network.

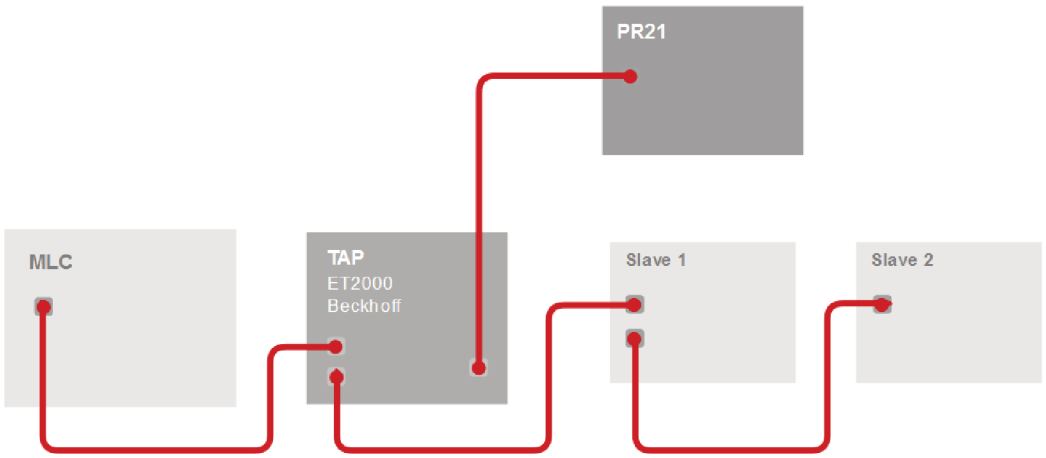
Figure 8 – User Case 1



Source: Personal archive

2. Next, Figure 9 shows another proposed topology. There is a network tap (here, ET2000 from Beckhoff) working between the master and the slaves. It is known from its datasheet that by inserting Beckhoff's network tap, a small delay is generated in the transmission of telegrams, approximately $1us$. The monitoring port of the device copies what goes through the network to the PR21 in order to have the packets stored there for analysis. This strategy inserts the ET2000 in between the master and the first slave, so it is possible to say that its mounting might not be the simplest one. In addition to it, instead of only having the PR21, this option includes a second device, which increases the costs of this mounting and requires some more configuring.

Figure 9 – User Case 2



Source: Personal archive

3. Figure 10 shows a case with a structure similar to the one seen in Figure 8. The PR21 is also connected in the end of the network line, but here, it has an extra part: an input module, that allows this mounting to receive external signals in pursuance of triggering the capture of telegrams. Those inputs can work like triggers for starting, stopping, or shutting down the process, and they can come as a signal

from the master when an error is detected, as the example in the Figure. This shows a limitation, because as in Figure 8, the PR21 does not receive what the slaves are sending back to the master. The situation is acceptable because of the pros that this mounting presents, like its simplicity, and because most problems can be solved by only having access to the packets sent by the master. For prototyping reasons, the inputs can also come from physical buttons. This approach facilitates testing of the individual functions of the device. The focus of the work presented on this document will be on this third case, due to its simplicity and fulfillment of the requirements that were firstly introduced.

Figure 10 – User Case 3

Source: Personal archive

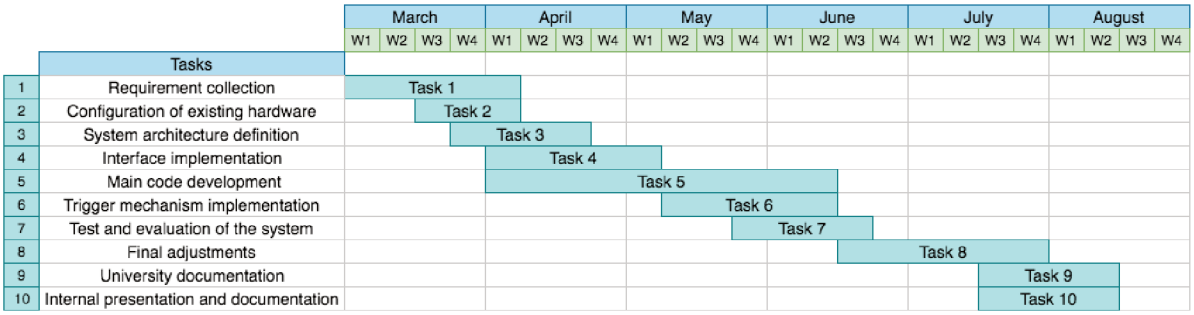
To achieve this project's goal, a definition of an initial list of activities was made based on the comprehension of the problem and the empiric knowledge of the colleagues inside the company. The list has a generic character, as the details will be shown in the implementation and development description. The main activities to be accomplished are:

- Requirement collection: Definition of Use Cases and functional requirements. Description of the work's scope.
- Configuration of existing hardware: In possession of the industrial computer PR21 where the project is to be developed, configure all necessary parts, choose an adequate operational system, download libraries and programs to be used in the implementation of the project.
- System architecture definition: Choosing the technologies that will be part of the project, the approach to be made and how the sub-systems will relate between each other. Definition on non-functional requirements and other small characteristics.
- Interface implementation: Inside what has been defined in the previous steps, develop components and code programs that create cooperation between the sub-systems.
- Main code development: Programming the central parts of the software developed for the application, as well as the details defined by functional requirements.

- Trigger mechanism implementation: Progressing on the full functioning of the desired trigger mechanisms, may they be through buttons, signals sent from PLCs or other sources.
- Test and evaluation of the system: Validation of what has been developed through testing, considering the proposed use cases.
- Final Adjustments: Adjustments and changes regarding the connection between the components of the project and other minor settings.
- University documentation: Writing of reports, documents and presentations meant to be shown in the University scope.
- Internal presentation and documentation: Writing of documents and manuals for internal use in order to make the continuation of the project possible. Presentation of the work done to supervisors and company co-workers.

The Gantt diagram of the planned activities is shown in Figure 11.

Figure 11 – Tasks schedule



Source: Personal archive

4.2 System general view

The first step to be taken after analyzing the problem is to propose a general view of the solution. According to 21, the general view of a system is a text which describes the main idea of a system without pointing a specific structure to it. For this project, a general view can be defined as the following:

The device for automatic monitoring of Ethernet traffic consists of a system focused on keeping track of important network telegrams in industrial applications, in pursuance of easily finding error causes, may those be from hardware or software. The device used is the industrial computer PR21, which can be placed inside industrial networks in different topologies to read the telegrams that go through it without interfering in the machinery

communication. Developed inside Bosch Rexroth, Lohr am Main site, it aims to make costumer problems solving faster and simpler. The device has a 24V input module connected to it, which has the purpose of receiving trigger signals for starting and stopping telegram capture.

In the scope of this project's development, the trigger inputs are manually generated, but for real scenarios can be expanded to other sources. The software generates log files in text format and stores the captured telegrams in the format specified by the capture configuration. There are two different *.ini* files to be read during the program's execution: the general configuration file and the capture configuration file. Therefore, the system has three main functionalities:

- Reading the two text configuration files (general configuration file and capture configuration file) to define the specificity of the workflow.
- Interpret the input triggers to start and stop telegram network captures.
- Store the necessary information (logs and capture files) in the adequate format and inside the correct folder locations.

This behavior is expected when combining several other smaller functionalities, such as: The selection of the capture tool to be used in the program through the configuration file; The possibility of choosing the storage format and location of the captured telegrams via capture configuration file; The possibility of choosing different type of log files; To configure the drive and the input pins to trigger different program settings, as well as a number of other small tasks.

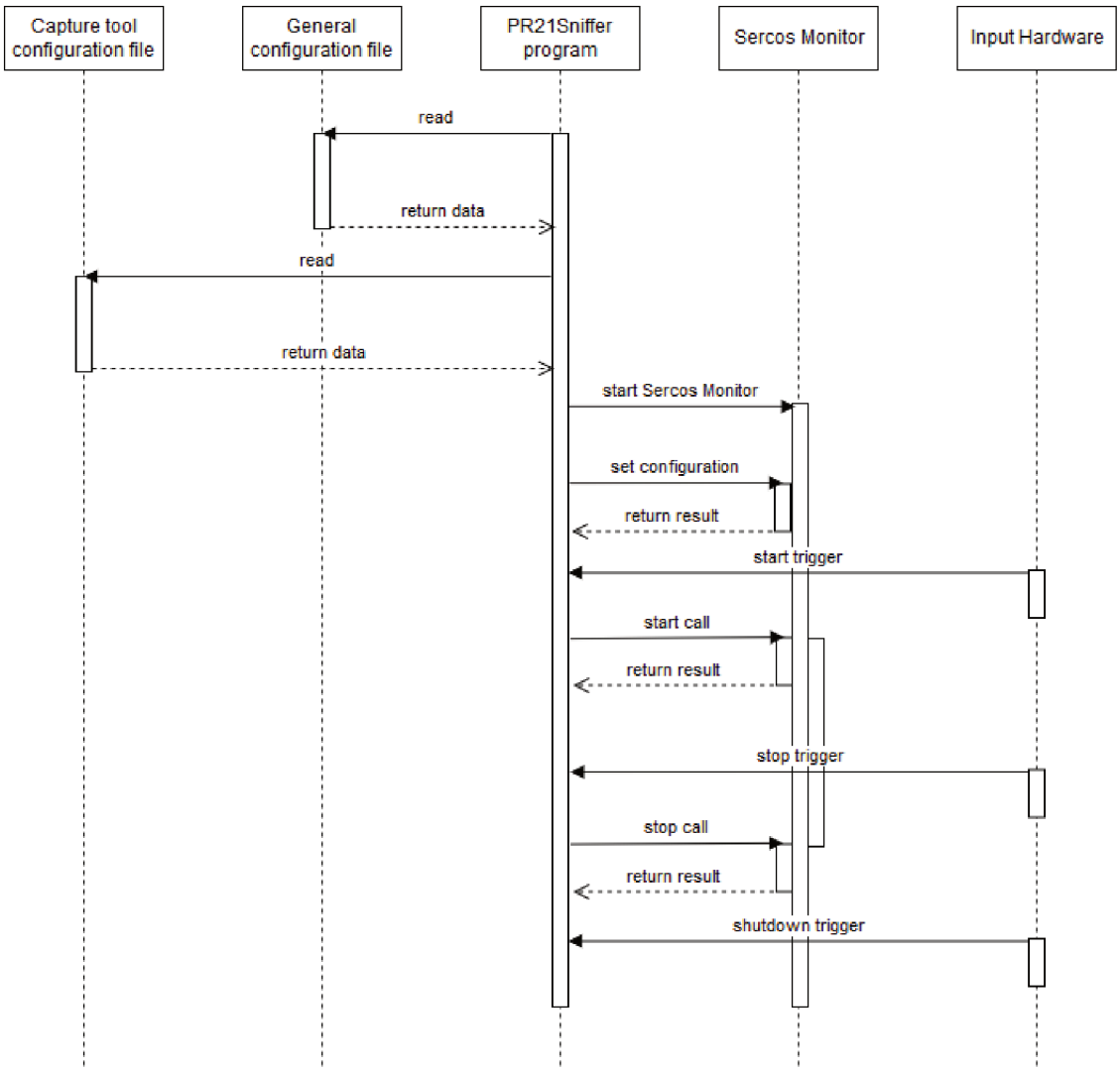
4.3 Workflow

Detailing the general view of the system, an UML sequence diagram can be brought up in order to better explain the system's workflow and justify some implementation choices.

The diagram shown in Figure 12 depicts the sequence of the tasks made and the interaction between the modules of the project. As it was already mentioned in Section 4.2, there are two types of *.ini* files to be read. It was decided that only one configuration file would not be enough for describing all the settings possibilities because different capture tools have different parameters to be set. Therefore, only having a general file is too generic for this application. So they were separated in two types: General configuration file and capture tool configuration file. The first one is meant to be more generic and should contain the location of the second one. The second one, about the capture tools, is more specific in the matter of which parameters are defined according to the application.

Analyzing the sequence diagram in Figure 12, it can be seen that the main program (PR21 Sniffer program) is initiated (via command line parameters that set the location of the general configuration file) then reads the general configuration file as its first task. From there, it reads the capture tool configuration file. When it acquires all this information, it configures itself and starts the capture tool determined by the *.ini* files. In this sequence diagram, the chosen capture tool is Sercos Monitor. The program sets the configuration to Sercos Monitor and keeps running in a loop that waits for inputs until a trigger is detected. When a start trigger is detected by the program, it sends a start call to Sercos Monitor, which means that now all the telegrams running through the PR21 are stored in a determined location. Once a stop trigger is detected, the program stops the capture and finishes storing the telegrams. When a shutdown trigger is detected, the PR21 Sniffer program is finished and the application is ended.

Figure 12 – UML system's sequence diagram - Sercos Monitor



Source: Personal archive

4.4 Requirements

To fulfill the desired workflow, functional requirements are gathered in the conception phase of the project. This step in the process serves as guideline for the implementation and comprehends only the extension of what the system is supposed to do, without detailing how it is done [21].

Following Wazlawick's model, the functional and non-functional requirements of the system can be read in Tables 1, 2, 3, 4, 5, 6 and 7. From those requirements, it is interesting to highlight the determined need to have several configurations set by the user, like *NF5.1* and *NF5.2*, that it is the user's task (through the configuration files) to specify where and how to store the capture files.

NF1.2 also specifies an interesting aspect of the application: it should be able to handle wrongly inserted information from the user, either with error or warning messages.

By taking a closer look to the requirements, the path for the implementation begins to look clearer, as software requirements should, in general, be related to specific methods or classes of the code.

Code	**Functional Requirements**
FR1	Automatically read the configuration files and set the necessary parameters.
FR2	Receive signals from the I/O module connected to the hardware.
FR3	Interpret signals to trigger telegram capture start and stop.
FR4	Communicate with different capture tools.
FR5	Automatically save the captured files.
FR6	Incorporate diagnose LEDs.

Table 1 – System's Functional Requirements

FR1 - Configuration files			
Description: The system must automatically read and interpret both general configuration file and capture tool configuration file.			
Non-Functional Requirements			
Name	**Restriction**	**Category**	**Desirable**
NF1.1 - Variable reading	The program should be set to read and interpret only expected variables, for example *LogFilePath*.	Specification	Yes
NF1.2 - Error and warning handling	When incorrect values are read from the configuration files, the program should show error or warning messages.	Security	Yes

Table 2 – Non-Functional Requirements FR1

FR2 - I/O module			
Description: The system must be able to receive signals from the I/O module connected to the hardware			
Non-Functional Requirements			
Name	**Restriction**	**Category**	**Desirable**
NF2.1 - Cycle time	The program should be able to receive the signals in a cycle time, found empirically, that is fast enough to not miss any signals, but also not too fast to face the switch debouncing problem.	Performance	Yes
NF2.2 - Edge detection	When receiving a signal, the program should know whether it is a rising edge signal or a falling edge signal.	Specification	Yes

Table 3 – Non-Functional Requirements FR2

FR3 - Triggers			
Description: The system must trigger determined actions according to the inputs received.			
Non-Functional Requirements			
Name	**Restriction**	**Category**	**Desirable**
NF3.1 - Trigger interpretation	The actions to be taken after a specific signal being triggered can only be defined in the configuration file.	Security	Yes
NF3.2 - Capture tool uniqueness	A signal can only trigger starting/stopping capture for one capture tool at a time. Meaning, the only capture tool used in the run-time is the one specified in the configuration file.	Specification	Yes

Table 4 – Non-Functional Requirements FR3

FR4 - Capture Tool			
Description: The system must be able to communicate with determined capture tools in order to use their functionalities.			
Non-Functional Requirements			
Name	**Restriction**	**Category**	**Desirable**
NF4.1 - Communication adaptation	The program must send different commands to start and stop the capture, according to the capture tool chosen on the configuration file.	Specification	Yes

Table 5 – Non-Functional Requirements FR4

FR5 - Capture files			
Description: The system must store all the captured telegrams files automatically.			
Non-Functional Requirements			
Name	**Restriction**	**Category**	**Desirable**
NF5.1 - Storage location	The telegrams should be stored in the path specified in the configuration files.	Specification	Yes
NF5.2 - Storage format	The telegrams should be stored in the format specified in the configuration files.	Specification	Yes

Table 6 – Non-Functional Requirements FR5

FR6 - Diagnose LEDs			
Description: The device should have LEDs to point out the status of the application.			
Non-Functional Requirements			
Name	**Restriction**	**Category**	**Desirable**
NF6.1 - Status	The LEDs should be used in the way to indicate two different status: green for running capture and red for error.	Specification	Yes

Table 7 – Non-Functional Requirements FR6

4.5 Chapter summary

This Chapter approaches the identification of the problem in details and then proposes the modelling for the solution, following the UML standards for diagrams and requirements. It takes in consideration the technologies previously presented to develop the workflow of the project, as well as the patterns for the sequence scenario. Also displayed here is the schedule for the activities of development and documentation. The use cases are analyzed and the option is made to adopt the one that is simple enough to be implemented, yet has enough complexity to fulfill its task. By bringing up those topics and decisions, it foments the next Chapter, which consists on the description of the implementation and the options made during this phase. Chapter 5 points out the different tools and strategies taken to develop each part modelled in this current Chapter.

5 IMPLEMENTATION

Chapter 5 consists on the detailed description of the implementation and development of all software and hardware aspects that were brought up during the modelling phase, as seen on Chapter 4. Sections 4.2 and 4.3 go through the workflow and general view of the system, giving a good overview on the main aspects of the implementation of this project, as the requirements are used as guidelines for software development.

The implementation of this work comes based on a software developed inside Bosch Rexroth by another student, Simon Müller, who started to write the software and had the idea to mount the input module on the PR21. After his initial work, he switched projects and the continuation of the project became the Bachelor thesis here described.

As previously stated, the PR21 is an industrial computer, which means in practical scenarios that its operational system does not have a graphical user interface (GUI) or supports complex tools. Although the development PC is Windows OS based, the chosen OS for the target is a Linux one, therefore, all programs developed follow the specificity of this operational system. This is possible because tools like Visual Studio [22] allow cross-compiling, meaning that the execution of codes can happen in an outside target, enabling all the work to be developed in a regular PC.

Following the workflow of the project structure, this Chapter covers through its Sections the different components that together form the implementation of this work. Firstly, the configuration files are exposed and explained with their details. Then, the C++ program *PR21Sniffer* is brought up, as well as the interaction with the capture tools, log files and other aspects.

5.1 Configuration files

Configuration files in this scope are archives used to configure parameters and initial settings in applications. They are read during initialization phase in order to personalize the type of capture wanted and further information, as the configuration files for this project are meant to be the interface between the final user and the program. The chosen format for those files is the *.ini* format, which are simple text files with a basic structure composed of "sections" and "keys", and comments are defined by ";", like the example:

```
[section]
name=value ; this is a key
```

For this project, it was defined that two configuration files must be read in the startup step. First, the general configuration file and second, the capture tool configuration file. As mentioned in Section 4.3, this decision comes from the need of flexibility in the system. The general configuration file contains all basic information to be defined about the run-time of the program, and the capture tool configuration file regards the specificity of different capture tools. So, they are separated in a way that it is possible to have different capture tools working without having to change the entire structure of the general document.

5.1.1 General configuration file

The general configuration file, as the name points out, serves to set the most basic information to the program. It is used as an argument to start the C++ program, in a way that it is the first contact to the user needs.

The file is divided in the following sections and their respective keys:

- *general_information*
  - *CaptureTool*: Identifies which capture tool will be used.
  - *CycleTimeDelay*: Delay in mili seconds, to avoid switch debouncing.
  - *LogFilePath*: Path and name of the log file
  - *TraceTypeLOG*: TRUE/FALSE to log output. Case sensitive.
  - *TraceTypePRINTF*: TRUE/FALSE to printf output. Case sensitive.
  - *TraceTypeEXTERN*: TRUE/FALSE to external option output. Case sensitive.
- *capture_tool*
  - *CapToolIniPath*: Path to the *.ini* file related to the capture tool.
- *drive_id*
  - *DevInfo*: String that defines the hardware to be used.
- *pin_number*
  - *PinNr*: Pin identifier.
  - *EdgeToStart*: Edge type that triggers the start state.
  - *EdgeToStop*: Edge type that triggers the stop state.
  - *EdgeToShutdown*: Edge type that triggers the shutdown state.

Figure 13 – Example of a general configuration *.ini* file

```
;;;;;;;;;  Configuration example file for the PR21 as a monitoring device   ;;;;;;;;;

[general_information]
CaptureTool = TCP_DUMP
CycleTimeDelay = 1
LogFilePath = ../../../../../test.log
TraceTypeLOG = TRUE
TraceTypePRINTF = TRUE
TraceTypeEXTERN = FALSE

;;  Information relevant for the capture configuring
; CaptureTool          : Identifies which capture tool should be used
; CycleTimeDelay       : Delay in milli seconds for avoiding the debounce of inputs, 1ms usually sufficient
; LogFilePath          : Path and name to the log file
; TraceTypeLOG         : TRUE/FALSE to log output
; TraceTypePRINTF      : TRUE/FALSE to printf output
; TraceTypeEXTERN      : TRUE/FALSE to external function output

[tcp_dump]
CapToolIniPath = /home/rexroth/capTool.ini

;;  Information specific to the capture tool
; CapToolIniPath       : Path to the ini file related to the capture tool


[driver_id]
DevInfo = PCM-27D24DI,BID#0

;;  Information about the hardware of the driver
; DevInfo              : String that defines the hardware, found at "cat /sys/class/daq/daq0/desc" on Linux target

[pin_0]
PinNr = K_IDI_0
EdgeToStart = E_RISING
EdgeToStop = E_NONE
EdgeToShutdown = E_NONE


;;  Description of all pins and their respective data
; PinNumber            : Identifier of the pin
; EdgeToStart          : Edge type that triggers the START state
; EdgeToStop           : Edge type that triggers the STOP state
; EdgeToShutdown       : Edge type that triggers the SHUTDOWN state
```

Source: Personal archive

Figure 13 shows an example of how a general configuration file looks like when it is filled with correct key names and values.

It is interesting to note that the section regarding *capture_tool* must contain the name of the capture tool to be used, like showed in the example with *tcp_dump*. Also, the sections regarding the pins are implemented with the number of the pin on the section name. The example only shows one pin section, but in the development of this work it is used an input module with fifteen pins, and all are configured in the *.ini* file, including the ones that do not trigger anything. Explaining the *EdgeToStart* and related keys, they represent the change of the input state that triggers some action on the program, in a way that the pins are the abstraction of the trigger inputs. For example, on Figure 13, *pin_0* has an edge that is different than none only for the start key, therefore, when this input signal changes from 0 to 1, it triggers the start of a capture.

About the *CycleTimeDelay*, as the description above exposed, it is used to avoid switch debouncing in the hardware. Switch debouncing happens when a button is pushed and its internal switch goes from making contact to not making contact a few times until

stabilizing. The value of 1 millisecond in the example is defined empirically as a good value for the delay time, small but still safe.

Other sections and keys of this file are intuitive. *CapToolIniPath* points out the path where the capture tool configuration file is located, so the program can find it and read it.

5.1.2 Capture tool configuration file

The capture tool configuration file is meant to be the interface between the user and the program that contains the specific information about the packet capture needed. Knowing that different tools have different parameters and settings, the capture tool configuration file changes accordingly. In the scope of this project two distinct capture tools are approached: Sercos Monitor and TCP dump. Therefore, it is shown in this Sub-Section the files for both tools.

As exposed in Section 3.5, TCP dump is, between the presented tools, the simplest one. The configuration file for TCP dump is divided in the following sections and keys:

- *tcp_dump*
  - *StoragePath*: Relative path and file name to store the capture files.
  - *Interface*: Name of the interface where network traffic is recorded.
  - *AddTimeToFileName*: TRUE/FALSE to add date and time to the capture file name. Case sensitive.
  - *FileSize*: Option for ring buffering, defines the maximum size of each capture file, in Mega Bytes.
  - *FileNumber*: Option for ring buffering, defines the number of files the ring buffer should work.

Figure 14 explores an example of a TCP dump configuration file. In this example, there are two files in the ring buffer to store the captured packets, each one with 1MB of size. Since there is no date and time added to the name of the files, they are differentiated in the program by adding the numbers "0" and "1", for this example, at the end of the file name. With these settings, what happens to the storage files when the capture lasts long enough to occupy more than 2MB (1MB for each file) is that the *.pcap* files start to get overwritten. Hence, the number of files does not increase, it always stays in 2.

Figure 14 – Example of a TCP dump configuration *.ini* file

```
;;;;;;;;;  Configuration file for the Capture Tool - TCP dump  ;;;;;;;;;

[tcp_dump]
StoragePath = ../../../../../ethCapture.pcap
Interface = enp2s0
AddTimeToFileName = FALSE
FileSize = 1
FileNumber = 2


;;  Information specific to the capture tool
; StoragePath         : Path to save the capture, relative to the .out folder, example saves in the rexroth folder
; Interface           : Name of the interface where network traffic will be recorded
; AddTimeToFileName   : TRUE/FALSE to have in the capture file name, the date and time of the capture start
; FileSize            : Options for ring buffering. Maximum size of each capture file, in Mega Bytes
; FileNumber          : Number of files which the ring buffer should work with
```

Source: Personal archive

For Sercos Monitor, there are several different parameters to be set, but due to the tool's complexity and the need of using a REST client to communicate with Monitor Core API, the possibilities for personalizing are slightly reduced. For the implementation, the choice is to use Restbed as the framework used in C++ to send the requests to the Monitor Core (the server, in this scope). The configuration file for Sercos Monitor is divided in the following sections and keys:

- *sercos_monitor*
  - *Interface*: Name of the interface where network traffic is recorded, as well as the IP address of the target and the MAC address.
  - *PathToGeneralConfig*: Path where Sercos Monitor *.xml* configuration file is stored.
  - *Host*: Specific information for requests to Sercos Monitor API.
  - *IpPort*: Specific information for requests to Sercos Monitor API.
  - *StoragePath*: Absolute path to store the captured files.
  - *AddTimeToFileName*: TRUE/FALSE to add date and time to the capture file name. Case sensitive.

Figure 15 shows an example of Sercos Monitor configuration file. Some parameters are shared between the two tools, like *StoragePath* and *AddTimeToFileName*, but others are used exclusively because of the REST client needed, like *Host* and *IpPort*.

Another aspect that deserves a highlight from the file is the *PathToGeneralConfig* key. Sercos Monitor application, when run in a regular PC, has manifold options and filters that can be applied in the capture of packets. All those options configure a *.xml* file generated by the tool itself. Knowing that through the REST API it is not easy to set

many different configurations, the opportunity of using this *.xml* file for the composition of the capture tool enhances the number of possible settings. Therefore, the option is explored in this project in order to maintain the complexity of Sercos Monitor tool, which is of great value to some applications.

Figure 15 – Example of a Sercos Monitor configuration *.ini* file

```
;;;;;;;;;  Configuration file for the Capture Tool - Sercos Monitor  ;;;;;;;;;

[sercos_monitor]
Interface = "\"enp2s0 / enp2s0 / 192.168.7.1 / 74-FE-48-28-B6-48"\"
PathToGeneralConfig = /home/rexroth/test.xml
Host = 192.168.7.1:8888
IpPort = http://192.168.7.1:8888
StoragePath = /home/rexroth/SMCapture.pcap
AddTimeToFileName = FALSE


;;  Information specific to the capture tool
; Interface                : Name of the interface where network traffic will be recorded
; PathToGeneralConfig      : Path to find the Configuration file for Sercos Monitor (complete file)
; Host                     : Information necessary to do the requests to Sercos Monitor api
; IpPort                   : Information necessary to do the requests to Sercos Monitor api
; StoragePath              : Absolut path to save the capture
; AddTimeToFileName        : TRUE/FALSE to have in the capture file name, the date/time of the capture start
```

Source: Personal archive

5.2 C++ program

The project presented in this document has its level of complexity increased due to the different components that must be integrated somehow in order to meet the goals established in the early development phases. This requires a fluid integration system that must be flexible enough to be updated as demanded, in a way that the system itself is not attached to specific tools, libraries or methods. The choice of having the main part of the software developed in C++ comes from the already existing expertise in the language inside the team where the project is inserted. Although the target has a Linux OS, the development was carried in the Visual Studio 17 tool in a Windows PC, which is possible due to VS features, as mentioned previously.

Besides the task of integrating all other parts of the system, like reading the configuration files and receiving the input signals, the main program (called PR21Sniffer) is also responsible for interpreting and abstracting all data involved in the entire process. The implementation stage demands an extra attention to the software, because for this project not only the functionality of the project is important, but also the possibility of future expansion and improvement of it. On account of this information, the concept of modularity comes once again as a general guideline to be followed, justifying the use of strategies such as interface classes, standard maps and macro definitions in the design of the system structure.

5.2.1 Class diagram

The main classes of the program can be seen in Figure 16. The classes are divided by functionality, and a brief explanation of those main parts and their role in the system is given as:

- Application: The most general of classes, it is responsible for creating objects of basically all the other classes and for the main loop. Also responsible for the settings of the different types of logs.

- LightController: This class refers to the control of the LEDs that indicate error or success of the program.

- StatefulPin: It abstracts the inputs and their functions in objects that contain data such as pin number, edge to start, edge to stop, edge to shutdown and latest state.

- InputController: Class that abstracts the grouping of all pins and other information read from the configuration file, like delay time for input reading and the drive's identifier string.

- IfaceIniFileHandler: Interface class created to abstract the functions of reading the configuration file not being too specific about the method for doing it, since there are lots of models available.

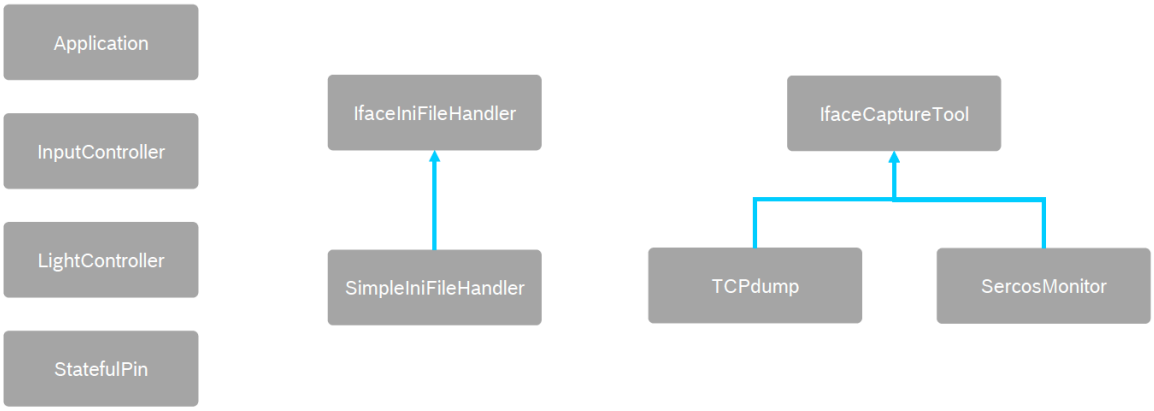
- SimpleIniFileHandler: Class for dealing with the *.ini* file, currently the only one, but with the aid of the interface class, more can be added later without having to change the entire code.

- IfaceCaptureTool: Interface class to abstract the different possibilities of capture tools to be used, containing functions that are common to all of those tools, such as start capture, stop capture and configure tool. As all interface classes, it is not supposed to instantiate any objects.

- TCPdump: This class refers to TCP dump and its specific methods and variables, inheriting the ones from the interface class IfaceCaptureTool.

- SercosMonitor: Refers to the Sercos Monitor capture tool and its specific methods and variables, inheriting the ones from the interface class IfaceCaptureTool.

Figure 16 – C++ class diagram



Source: Personal archive

In order to maintain the confidentiality standards of the company, Figure 16 shows only a high level representation of the implemented classes. For further content on the C++ classes and their methods and variables, see Appendix A.

5.2.2 Error handling

Error and warning handling refers to the response procedures from error conditions presented in the software application. It is the process of maintaining the normal workflow of the program, or shutting it down when necessary.

When it comes to interacting with humans for configuring and setting parameters that will go through the entire process, the possibility of having wrong information as inputs has to be considered in the development phase. With this in mind, the project's code is structured in a way that methods that are prone to error always return a status variable, in hexadecimal notation, which contains the code for either successful processes (`0x00000000` status code) or processes returning warnings or errors.

This measure is taken with special attention, as parameters wrongly set can cause troubles that are not so easily identified. The status codes are divided between success, warnings and errors. Warnings are triggered when a information is wrongly set, but there are ways of fixing it using default information without stopping the run-time. On the other hand, errors trigger the complete stop of the program, as there are no solutions that can be taken without compromising the user needs. The status codes, as well as the error and warning sources already identified and predicted can be seen in Table 8.

Type	Code name	Value	Description
Success	PR21_S_OK	0x00000000	Status for successful process.
Warning	PR21_W_MISSING_LOG_PATH	0x00000001	Warning for lack of path to store the log.
Warning	PR21_W_MISSING_INTERFACE	0x00000002	Warning for lack of interface specification for capture.
Warning	PR21_W_MISSING_CAPTURE_PATH	0x00000003	Warning for lack of path to store the capture.
Warning	PR21_W_MISSING_LOG_OPTION	0x00000004	Warning for not choosing which log options should be active.
Warning	PR21_W_CAPTOOL_NOT_SPECIFIED	0x00000005	Warning for not selecting the capture tool.
Error	PR21_E_MISSING_DRIVE	0xF0000001	Error for lack of drive information in the *.ini* file.
Error	PR21_E_MISSING_TRIGGER	0xF0000002	Error for lack inputs set to trigger the capture.
Error	PR21_E_MISSING_CAPTOOL_INI	0xF0000003	Error for lack of path to the capture tool configuration file.

Table 8 – Status codes of the C++ program

About the errors seen in Table 8, it is noticeable that stopping the program is the only alternative to follow, as for example in error *PR21_E_MISSING_DRIVE*, without the hardware information it is not possible to get the inputs from the device and capture packets. When error *PR21_E_MISSING_TRIGGER* is set, no input is configured in the device to trigger the capture, therefore it would never happen, so there is no reason to continue the program. Finally, error *PR21_E_MISSING_CAPTOOL_INI* depicts a situation where there is no path to the capture tool *.ini* file, and the capture can not be configured.

The warning handling deserves a special note on it, as it does not stop the program, only modifies the configuration parameters that were not set to default ones. This information, along with the warning code and the warning explanation, is printed out in the log so the user can keep track of it. The default settings are defined by the developer as the most basic ones. When there is the case of multiple warnings being triggered, all of them are printed out in the log option. Table 9 shows for the following warnings the respective default options to be taken.

Warning	Response
PR21_W_MISSING_LOG_PATH	No path inserted for log. No log file writing allowed. Default PRINTF is selected.
PR21_W_MISSING_INTERFACE	No interface for capture was specified. All interfaces shall be captured.
PR21_W_MISSING_CAPTURE_PATH	No capture storage path was specified. Capture will be stored on default path "~/defaultCapture.pcap".
PR21_W_MISSING_LOG_OPTION	No log option was chosen. Default PRINTF is selected.
PR21_W_CAPTOOL_NOT_SPECIFIED	No capture tool was specified. Default TCP dump is selected.

Table 9 – Status codes of the C++ program

5.2.3 Logs

A log file, in the scope of this project is a text file used to automatically document the produced and time-stamped information of events, behaviors and conditions relevant to a particular system. By reviewing the data contained in a log file, the developer and the user can see where problems are occurring, what is causing them, and in the case of success, to check that everything is happening as planned.

The log type in this project is set through the *.ini* general configuration file, as seen in Section 5.1. There can be log messages directly printed in the Linux console, in a text file or in an external output printing. This option comprehends other alternatives that may be implemented according to the user needs, like logging into Linux journal, for instance.

The stored log files structure can be divided in two separate parts: configuration phase and execution phase. Firstly, when the project is started and all the configuration files are read, the selected settings are printed in the log, as well as the time-stamp for every event. The second phase of the log file printing is where all configurations have already been set and the program enters the loop of waiting for inputs to trigger the capture of packets.

Those files are the detailed outcome of what happens inside the system during the run-time. It is the evidence that reveals whether the task is successful or not, showing errors or warnings. Therefore, the detailing of it will be given in the Chapter to come, as the results are proven.

5.3 Chapter summary

In resume, Chapter 5 presents all the topics related to the implementation, which consists of different tools combined together in the way that comprehend all the specified requirements and fulfill the initial goals, in order to actually do an automatic capture of

packets in an industrial network. First, the configuration files are approached and the reason for them to be separate is explained. They are the interface with the user and, sequentially, the first phase of the system's run-time. After, the C++ program is presented in a high level, depicting the classes and the proposed functionalities. Inside this part, the error handling is treated and explained, as well as the possibilities of logs allowed by the program. Next Chapter brings more light into this matter, as log files are presented and explained as successful results of the work. The results are then analyzed and the goals stated initially are fetched back in order to explain the outcomes of the project.

6 RESULT ANALYSIS

Before anything else, to analyze the outcome of the monograph means to evaluate if the primary goals were achieved in the best possible ways. In other words, to describe if the development has followed best practice guidelines to keep easy maintainability, good user experience and optimized resource utilization. This Chapter goes through those aspects in a mostly qualitative measure, as the results are evaluated in a research facility and cannot be compared to real costumers work conditions.

Recapitulating, the program starts by reading the *.ini* configuration files, setting the necessary parameters and then starting the chosen capture tool. After this configuration stage, the program enters the execution loop, where it keeps waiting for triggers in order to start or stop a capture of packets.

It is possible to begin with the analysis and evaluation of the project described in this document by observing the log files, briefly described in Sub-Section 5.2.3. As stated, they can be logically divided into two parts: configuration phase and execution phase. Figure 17 depicts the configuration phase of the log file, and from it, to see that all configurations described in the configuration files are printed out, in order to keep track of what is happening to the system and to consult it afterwards for further analysis and other costumer needs. Moreover, the timestamp shows that the configuration phase happens in less than one second, which corroborates to the idea that the software developed in the scope of this project has good performance measures. Once all setting information is read and printed out, as there are no errors, the configuration phase is finished.

Figure 17 – Configuration part printed in the log file

```
[08-08-2019 11:01:29]         Path to.ini file: /home/rexroth/configFile.ini
[08-08-2019 11:01:29]
[08-08-2019 11:01:29]         ------------------------------------------------------------
[08-08-2019 11:01:29]         Configuration started
[08-08-2019 11:01:29]         ------------------------------------------------------------
[08-08-2019 11:01:29]
[08-08-2019 11:01:29]         Reading data from configuration file
[08-08-2019 11:01:29]
[08-08-2019 11:01:29]         Driver hardware: PCM-27D24DI,BID#0
[08-08-2019 11:01:29]         Capture Tool: TCP dump
[08-08-2019 11:01:29]         Delay for input reading: 1ms
[08-08-2019 11:01:29]         Path to store log : ../../../../../captureInfo.log
[08-08-2019 11:01:29]         Extern output: FALSE
[08-08-2019 11:01:29]         Log output: TRUE
[08-08-2019 11:01:29]         Printf output: TRUE
[08-08-2019 11:01:29]         Storage file has date/time: FALSE
[08-08-2019 11:01:29]         Number of capture files: 2
[08-08-2019 11:01:29]         Capture file size: 1
[08-08-2019 11:01:29]         Interface for capture: enp2s0
[08-08-2019 11:01:29]         Path for capture storage: ../../../../../ethCapture.pcap
[08-08-2019 11:01:29]         Ring buffer option: Active
[08-08-2019 11:01:29]         Number of active pins 2
[08-08-2019 11:01:29]         List of pins:
[08-08-2019 11:01:29]
[08-08-2019 11:01:29]         PIN 0
[08-08-2019 11:01:29]         PIN 1
[08-08-2019 11:01:29]         PIN 10
[08-08-2019 11:01:29]         PIN 11
[08-08-2019 11:01:29]         PIN 12
[08-08-2019 11:01:29]         PIN 13
[08-08-2019 11:01:29]         PIN 14
[08-08-2019 11:01:29]         PIN 15
[08-08-2019 11:01:29]         PIN 2
[08-08-2019 11:01:29]         PIN 3
[08-08-2019 11:01:29]         PIN 4
[08-08-2019 11:01:29]         PIN 5
[08-08-2019 11:01:29]         PIN 6
[08-08-2019 11:01:29]         PIN 7
[08-08-2019 11:01:29]         PIN 8
[08-08-2019 11:01:29]         PIN 9
[08-08-2019 11:01:29]
[08-08-2019 11:01:29]         Port count: 2
[08-08-2019 11:01:29]
[08-08-2019 11:01:29]         ------------------------------------------------------------
[08-08-2019 11:01:29]         Configuration finished
[08-08-2019 11:01:29]         ------------------------------------------------------------
```

Source: Personal archive

Next, the second logical phase of the log file starts, where the application keeps constantly checking for new inputs in order to trigger an action from the capture tool. Figure 18 shows that by the time *[08-08-2019 11:01:29]* the program had already entered the execution loop. On *[08-08-2019 11:01:44]* there is the message of the detection that pin 0 has changed its Boolean value from 0 to 1, therefore a rising edge. As this is the pin whose rising edge triggers the start of the capture of packets, TCP dump command is called right after the trigger, and the capture of telegrams begins. It can be noted that in the log file all edge changes appear, even the ones that do not trigger anything (such as "Falling edge from pin 0"). On *[08-08-2019 11:01:48]* a rising edge from pin 3 is detected, the stop of TPC dump is triggered and its respective process is killed.

Figure 18 – Execution part printed in the log file

```
[08-08-2019 11:01:29]
[08-08-2019 11:01:29]         Waiting for inputs...
[08-08-2019 11:01:29]
[08-08-2019 11:01:44]         Rising edge from pin 0
[08-08-2019 11:01:44]         Start capture triggered
[08-08-2019 11:01:44]
[08-08-2019 11:01:44]         sudo tcpdump -n -q -t -w ../../../../../ethCapture.pcap -i enp2s0 -C 1 -W 2 &
[08-08-2019 11:01:44]         Starting tcpdump capture
[08-08-2019 11:01:44]         Falling edge from pin 0
[08-08-2019 11:01:44]
[08-08-2019 11:01:48]         Rising edge from pin 3
[08-08-2019 11:01:48]         Stop capture triggered
[08-08-2019 11:01:48]
[08-08-2019 11:01:48]         TCP dump process: 3380

[08-08-2019 11:01:49]         Falling edge from pin 3
```

Source: Personal archive

Bringing back the goals enlightened in Section 1.3, it is interesting to cite once again the three backbones of the project's objectives:

- Robustness
- Easiness of Usage
- Automation

For Robustness, one can say that this pillar is comprehended in the development of this work, as it is fully developed to run in a Linux simple environment inside an industrial graded computer, therefore being ready for harsh environments. As for the Easiness of Usage part, many components of the implementation corroborate to this goal. The configuration files, for example, are thought in a way that they abstract complicated information, such as the ones from the capture tools, and are made simple enough to be filled by an user that does not necessarily have specific knowledge on network monitoring. Nevertheless, the configuration *.ini* files embrace several different configurations, in a way that solutions can be tailored according to costumer needs, as desired. The Easiness of Usage can be expanded further than only the user experience, since the software is designed following the best code practices and can be easily maintained, and grow in complexity standards. More capture tools can be added and therefore more scenarios implemented.

When it comes to Automation, it is easy to see where it applies. Besides the interface with the user through the *.ini* files, there is not supposed to be basically any human intervention in the entire process. For the development process cited in this document, buttons are used in order to trigger packet captures, but the system is ready to receive outside signals as inputs.

Considering all this information and the content previously presented, it can be said that the work here presented can potentially have a high impact on productiveness

and optimization of time and human resources for the company when implemented in real applications. Being able to monitor industrial networks brings the necessary awareness to the data inside those systems, following the current trend of looking to a better understanding of data in a way that adds knowledgeable value to the costumers plants.

It is important to note, though, that yet the application itself is fully functional, during the development phase the capture tool Sercos Monitor has presented unpredictable behavior while trying to interconnect it with the application. Their costumer service was contacted, but by the end of this internship there has been no solution to this situation. So, Sercos Monitor is only implemented virtually as a capture tool prone to be used, with all its methods and particularities. Although the project has fulfilled the expectations and is already available for use, there are some limitations on the implementation, that can mainly be used only as specified with the use cases.

6.1 Chapter summary

This Chapter presented the results and outcomes of the project developed throughout the internship period. By presenting the log files, an association of what is planned to happen can be made with the events depicted, showing a successful scenario. The Chapter also approaches the initially stated goals as metrics for success, comparing the backbones of the objectives with the implementation elements to begin the discussion on what is still open for improvement, or what has shown limitations. The next Chapter brings a conclusion comprehending the results and analysis of all other Chapters.

7 CONCLUSION

This document aimed to identify and develop an effective strategy for industrial network monitoring, based on actual needs gathered from an insider company, Bosch Rexroth. Developing the monograph's project inside the company brings awareness of several important aspects that in a theoretical environment would not be so clearly seen, such as the difficulties of actually implementing an idea in an established corporation and putting together the needs for Rexroth, the University and the costumers as a whole.

As stated, the use cases here have limitations and are better applied when following the allegations seen on this document, but as the system was entirely developed thinking about modularity and possible expansions, it is very fortunate to observe that the project itself is successful and prone to growth.

The results presented in Chapter 6 indicate that the area still has potentially space to grow, as the industrial sector tends to be more receptive and show more interest in data collection and analysis for the future years. Different features can be added and existing ones can be improved, comprehending a vast room for further development, which is brought up by Section 7.1. Section 7.2 portrays a brief personal synthesis of the work and the internship.

7.1 Possible future work

Knowing that there is still room for improvement in this project, and the continuation of it would bring interesting results for the company, this Section puts a highlight on possible future work that were thought during the development stage. Although the ideas for enhancement are many, due to the short period that this internship took place, it is not possible to cover and develop all strategies. The work presented here satisfies the initial goals stated along with the company, but future work can confirm, build on and enrich the conclusions taken.

For instance, some of the ideas that can come up as projects in the future include the implementation for different capture tools, as the only one completely usable at the moment is TCP dump. By making it possible to work with Sercos Monitor, for example, the complexity of the project is increased, as the tool provides a broad range of options and customization that might be interesting for some applications.

When it comes to user experience, it is possible to implement an actual GUI interface instead of the configuration files used currently, in order to have a better interaction with the user and still be able to make the necessary settings. However, the idea does not

implicate on extinguishing the *.ini* files, since it is a common and practical configuration format. It implies on the creation of a new layer, to stand between the configuration files and the user as a front-end. With this approach, the main program would only have to go through some minor changes, and all the parts developed to interpret the *.ini* files could still remain useful.

Perhaps one of the nicest future implementation idea was given during the presentation of this work to the company, by the end of the internship. Rexroth is launching in SPS Nürnberg 2019 (from November 26 to November 29) a new line of controllers and devices, called *cntrlX AUTOMATION*, described in [23], which possess a highly modern processing capacity. They have several cores inside the device, which implicates that operational systems can be installed in one or more cores, in the pursuance of having different features directly inside the controller. Those features come as optional *snap*, that the costumer can choose to buy together with the system in order to fulfill new and special requirements for the application.

Knowing the use cases for this project and the limitations presented by having an external hardware for monitoring the network, the idea given was that this project could be adapted to become one of those snaps, in a way that controllers can be bought with this embedded extra already and facilitate monitoring for problem solving reasons or any other that the application requires. It is interesting to notice that the acquiring of packets data is filling a gap that is being more and more recognized as industrial networks develop towards intelligent factories.

It can be concluded that, although the work already presented here aims for its objectives successfully, there is still a lot to be perceived as new necessities for industries, and therefore has a heavy relevance and projection of expansion.

7.2 Personal synthesis

Developing this project inside an internationally recognized corporation and being able to use a vast broad of knowledge acquired during all those years in University is a truly gratifying experience. I could not think of a best way to end my graduation, if not producing such an interesting and relevant work that is indeed related to the current needs of engineering field.

It is interesting to notice that, although the project's basis is mainly software development, knowledge from many different areas had to be used in order to pursue the best solution possible for the scope of the problem. It demanded knowledge on network protocols, industrial computers, PLCs, different operational systems and engineering tools and several soft skills that are so important for future engineers nowadays. Those contents were studied during University courses, but are actually more retained now that I had to

use them in a practical subject.

Besides learning so many technical topics with this internship, I have gained experience not only on my field of studies, but also personally. Being abroad for so long taught me how to be resilient, understanding and humble, and I will always carry this period fondly with me. The wish now is that the future brings opportunities as enriching as the one I had in Germany.

Bibliography

1 BOLTON, W. *Programmable Logic Controllers*. 4th. ed. [S.l.]: Elsevier, 2006. Cited on page 19.

2 IEEE. *IEEE 802.3 ETHERNET WORKING GROUP*. 1983. Cited 2 times on pages 20 and 25.

3 COHEN, Y. et al. Assembly system configuration through industry 4.0 principles: the expected change in the actual paradigms. *IFAC (International Federation of Automatic Control)*, 2017. Cited on page 20.

4 BOSCH REXROTH. *Rexroth - A Bosch Company*. 2019. Disponível em: <https://www.boschrexroth.com/de/de/>. Cited on page 23.

5 BOSCH REXROTH. *Revolution mit Druck*. 2012. Disponível em: <https://www.boschrexroth.com/de/de/trends-und-themen/directions/revolution-in-printing>. Acesso em: 11 feb. 2019. Cited on page 24.

6 SCHWAB, K. The fourth industrial revolution: What it means and how to respond. *Foreign Affairs*, 2015. Cited on page 25.

7 LIN, Z.; PEARSON, S. *An inside look at industrial Ethernet communication protocols*. 2018. Disponível em: <http://www.ti.com/lit/wp/spry254b/spry254b.pdf>. Acesso em: 25 nov. 2019. Cited on page 25.

8 ANALOG DEVICES. *What Is the Difference Between Ethernet and Industrial Ethernet?* Disponível em: <https://www.analog.com/en/technical-articles/what-is-the-difference-between-ethernet-and-industrial-ethernet.html#>. Acesso em: 27 nov. 2019. Cited on page 25.

9 C ENTERPRISES. *What's the Difference between Ethernet and Industrial Ethernet?* Disponível em: <https://blog.centerprises.com/whats-the-difference-between-ethernet-and-industrial-ethernet>. Acesso em: 27 nov. 2019. Cited on page 25.

10 BOSCH REXROTH. *Box PC - PR Series*. 2019. Disponível em: <https://www.boschrexroth.com/en/xc/products/product-groups/electric-drives-and-controls/industrial-pc-and-operator-panels/box-pc/pr>. Cited 2 times on pages 26 and 27.

11 SERCOS. *Transmission Principle*. 2019. Disponível em: <https://www.sercos.org/technology/functions-and-features/transmission-principle/>. Acesso em: 24 jan. 2019. Cited on page 28.

12 SERCOS. *Topology*. 2019. Disponível em: <https://www.sercos.org/technology/functions-and-features/topology/>. Acesso em: 20 nov. 2019. Cited on page 29.

13 MARTIN, R. C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. 1st. ed. [S.l.]: Prentice Hall, 2017. Cited 2 times on pages 30 and 31.

14 SPECTOR, A. Z. Modular architectures for distributed and database systems. *Association for Computing Machinery.*, 1989. Cited on page 31.

15 MOYNE, J. R.; TILBURY, D. M. The emergence of industrial control networks for manufacturing control, diagnostics, and safety data. *Proceedings of the IEEE*, IEEE, v. 95, 2007. Cited on page 31.

16 SURI, S.; BATRA, V. Comparative study of network monitoring tools. *International Journal of Innovative Technology and Exploring Engineering*, v. 1, 2012. Cited on page 31.

17 AMOEDO, D. *Tcpdump, conoce el tráfico de una interfaz de red desde la terminal*. 2018. Disponível em: <https://ubunlog.com/tcpdump-descripcion-general-herramienta/>. Acesso em: 20 nov. 2019. Cited on page 31.

18 TCPDUMP. *Manpage of TCPDUMP*. [S.l.], 2019. Disponível em: <https://www.tcpdump.org/manpages/tcpdump.1.html>. Acesso em: 19 nov. 2019. Cited on page 32.

19 SERCOS. *Tools - Sercos Monitor*. 2019. Disponível em: <https://www.sercos.org/technology/implementation/tools/>. Acesso em: 11 feb. 2019. Cited on page 32.

20 KUROSE, J. F.; ROSS, K. W. *Computer Networking: A Top-Down Approach*. 6th. ed. [S.l.]: Pearson, 2014. Cited on page 33.

21 WAZLAWICK, R. S. *Análises e Projetos de Sistemas de Informação Orientados a Objetos*. 3rd. ed. [S.l.]: Elsevier, 2010. Cited 2 times on pages 38 and 41.

22 MICROSOFT. *Microsoft Visual Studio*. Disponível em: <https://visualstudio.microsoft.com/>. Acesso em: 20 nov. 2019. Cited on page 45.

23 NüRNBERG, S. *SPS compact: The official e-paper for the SPS 2019*. 2019. Cited on page 62.

Appendix

APPENDIX A – C++ Classes

This Appendix is meant to give some more detailing on the implementation of the C++ classes mentioned in Chapter 5. The classes are divided here in Tables to exemplify some of their methods in a simple way.

StatefulPin	
Method	**Description**
bool getLatestState()	Returns the Boolean last state of the respective pin.
PinNumber getPinNumber()	Returns the identifier of the respective pin.
EdgeType getEdgetoStart()	Returns the edge to start referent to the pin.
EdgeType getEdgetoStop()	Returns the edge to stop referent to the pin.
EdgeType getEdgetoShutdown()	Returns the edge to shutdown referent to the pin.
InputTriggeredSignal getSignalOnPinChange()	Returns the assigned signal that shall be emitted on pin-change.

Table 10 – C++ methods for StatefulPin class

IfaceCaptureTool	
Method	**Description**
void init(SimpleIniFileHandler ini)	Initializes the capture tool and gets the *.ini* file.
void insertInfo()	Configures the capture tool according to the *.ini* file.
void startCapture()	Triggers the start of packet capture.
void stopCapture()	Triggers the stop of packet capture.

Table 11 – C++ methods for IfaceCaptureTool class

IfaceIniFileHandler	
Method	**Description**
void loadIniFile()	Loads the *.ini* file.
string vector getAllSections()	Gets all sections from the *.ini* file.
string vector getAllKeys(char section)	Gets all keys from a section in the *.ini* file.
string getValuefromKey(char section, char key)	Gets the value from a specific key inside a section.

Table 12 – C++ methods for IfaceIniFileHandler class

LightController	
Method	**Description**
void update()	Updates outputs to enable blinking of the LEDs.
LightType getLightType()	Returns the currently emitted LED color pattern.
void setLightType(LightType type)	Sets the new LED color pattern that will be emitted.
setOutputs(LightType type)	Sets LEDs outputs immediately.

Table 13 – C++ methods for LightController class

Application	
Method	**Description**
PR21_STATUS divideSections()	From the sections found by the *.ini* file handler, map them into specific types.
void configureLog()	Initial log configuration.
PR21_STATUS setGeneral(char section)	Sets the configurations from the *.ini* file regarding the *general* section.
PR21_STATUS configurePins(char section)	Configures the pin abstractions according to the *.ini* file.
PR21_STATUS configureDrive(char section)	Configures the hardware part.
setCapToolIni()	Initiates the chosen capture tool and passes the path to the capture tool *.ini* file to it.
PR21_STATUS checkVectorSize()	Checks if there are enough configured triggers.
PR21_STATUS execute()	Enter the execution loop of waiting for inputs.

Table 14 – C++ methods for Application class