TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JANNE ILMONEN

EFFICIENT COMMUNICATION USING LIMITED BANDWIDTH IN REMOTE MONITORING SYSTEMS

Master of Science Thesis

Examiner: Professor Jose L. Martinez Lastra
Examiner and topic approved by the Faculty Council of the Faculty of Engineering Sciences on May 30th 2018

ABSTRACT

JANNE ILMONEN: Efficient Communication Using Limited Bandwidth in Remote Monitoring Systems
Master of Science Thesis, 88 pages
May 2018
Master's Degree Programme in Automation Engineering
Major: Factory Automation and Industrial Informatics
Examiner: Professor Jose L. Martinez Lastra

Keywords: Remote Monitoring, PLC Programming, Data Transmission

Serial radio modems are long-standing but usable technology and they are still widely used in data communication in many applications. Slow transmission speed, point-to-point connections and signal attenuation are the negative aspects of serial modems. Old and new radio modem models are still compatible with each other because they have not changed greatly over the years. Radio modems can transfer many transmission protocols therefore user can select the most suitable protocol for the application.

The goal of this thesis was to improve the usage of serial radio modems in communication between PLCs. Data transfer speed generally cannot be increased thus only necessary data should be transferred through radio modems. The protocol that was used in this thesis was Modbus RTU which is supported by numerous process control devices.

Radio network enables reading and writing data to remote automation systems using Modbus RTU. Every device in the network has to be polled individually and round times of large networks can increase to tens of seconds. The basic idea in improving the communication rate is that every station in the network was read sequentially but stations can jump the queue if needed.

This thesis studied how different data should be divided in the communication. It is not necessary to use cyclical data transfer method for all the data and therefore some data could only be transferred when needed. Performance of the communication system was tested using existing remote monitoring system. The new communication system is similar to existing one therefore performance of the new implementation can be verified. The performance of both systems were measured and results have been presented in the last chapter.

TIIVISTELMÄ

JANNE ILMONEN: Tehokas kommunikaatio rajoitetun kaistanleveyden kaukovalvonta sovelluksissa
Tampereen teknillinen yliopisto
Diplomityö, 88 sivua
Toukokuu 2018
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Factory Automation and Industrial Informatics
Tarkastaja: Professori Jose L. Martinez Lastra

Avainsanat: Kaukovalvonta, PLC ohjelmointi, tiedonsiirto

Etävalvottujen automaatiolaitteistoiden tiedonsiirtoon on vuosikymmeniä käytetty sarjaliikenneradioita. Radiot ovat vanhaa, mutta toimivaa tekniikkaa ja niitä käytetään edelleen yleisesti tiedonsiirrossa monissa sovelluksissa. Sarjaliikenne radioiden käyttöön liittyy haasteita kuten hidas tiedonsiirtonopeus, point-to-point yhteydet ja maastosta ja etäisyydestä johtuva signaalin vaimeneminen. Radioiden hyvinä puolina voidaan pitää muuttumattomuutta. Kymmeniä vuosia vanhat radiomodeemit ovat edelleen yhteensopivia uusien kanssa. Modeemi ei myöskään ota kantaa käytettyyn protokollaan vaan radiolaitteiston käyttäjä voi lähettää dataa monella eri protokollalla sarjaliikenneradioiden kautta.

Tämän työn tavoitteena oli tehostaa sarjaliikenneradioiden käyttöä ohjelmoitavien logiikoiden välisessä tiedonsiirrossa. Tiedonsiirtonopeutta ei tavalliseti voi kasvattaa, vaan lähettevää dataa pitää pystyä valikoimaan siten, että vain tarpeellinen data lähetetään sarjaliikenneradion kautta. Tässä työssä käytetty protokolla on Modbus RTU, koska se on yleisesti tuettu monissa prosessien ohjaukseen käytetyissä laitteissa. Radioverkkoon kytketyiltä laitteilta voidaan lukea tai niihin voidaan kirjoittaa dataa käyttäen Modbus RTU:ta. Jokaiselta verkon laitteelta data pitää lukea erillisellä kyselyllä. Tällaisten verkkojen kierrosajat voivat kasvaa kymmeniin sekunteihin, jos luettavaa dataa on paljon. Tiedonsiirron tehostamisen yksi perusajatus olikin, että dataa luetaan normaalitilanteessa jonotus periaatteella, eli jokaisella asemalla on oma vuoro jonossa, mutta tarvittaessa aseman dataa kuitenkin voidaan lukea ohittamalla muut asemat jonossa.

Työssä selvitettiin, miten erilainen data kannattaa jakaa tiedonsiirrossa. Kaikkea tietoa ei ole tarpeellista lukea asemalta syklisessä kierrossa, vaan osa datasta voidaan lukea vain kun on tarpeen. Samaan tapaan asemalle voidaan kirjoittaa dataa jokaisella kierroksella, mutta tavallisesti dataa tarvitsee kirjoittaa asemalle vain kun joitain arvoja tarvitsee muuttaa. Tietoliikennejärjestelmän toiminnan ja suorituskyvyn tutkimiseen käytettiin olemassa olevaa kaukovalvonta automaatiojärjestelmää, jossa on ennestään ollut käytössä periaatteeltaan samankaltainen järjestelmä. Mahdollisuudet ja kirjoittamisen määrittelyyn ovat olleet kuitenkin huomattavasti rajallisemmat. Molemmista järjestelmistä tehtiin suorituskykyyn liittyviä mittauksia ja tulokset on koottu työn viimeiseen kappaleeseen.

PREFACE

I would like to thank Insta Automation Oy for giving me the opportunity to make this thesis for them. Insta made this thesis possible by providing the necessary tools, software and premises for me. Thesis project all in all took over six months of all-day work consisting finding references, planning, coding and writing. I would like to thank my superior at Insta who arranged opportunity for me to test my communication implementation in Insta's customer's remote monitoring application. I would also like to thank my co-workers Jarkko and Jussi-Pekka for assistance in planning and development of the software. They gave me a lot of fresh ideas and feedback that helped me along these six months.

I would also thank my fellow students in TUT. I believe we can think back with nostalgia of our late nights with different projects and homework. Yet they were nothing compared to the amount of work that took to write this thesis. After all these months it was quite rewarding to hand over finalized thesis to Professor Lastra. In 2015 when I started my master studies at TUT, I could not imagine that this day would come when I can finally say that my thesis is finished.

In Tampere, Finland, on 22 May 2018

Janne Ilmonen

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CIP	Common Industrial Protocol
CRC	Cyclic Redundancy Check
DB	Data Block
ECC	Execution Control Chart
FB	Function Block
FUN	Function
HMI	Human Machine Interface
IL	Instruction List
IP	Internet Protocol
LD	Ladder Diagram
mA	Milliampere
MTU	Master Terminal Unit
PDU	Protocol Data Unit
PLC	Programmable Logic Controller
POU	Program Organization Unit
PROG	Program
RSSI	Received Signal Strength Indicator
RTU	Remote Terminal Unit
SCADA	Supervisory Control and Data Acquisition
SFC	Sequential Function Chart
ST	Structured Text
TDMA	Time Division Multiple Access
TIA	Totally Integrated Automation
UDT	User defined Data Type
UML	Unified Modeling Language

LIST OF SYMBOLS

AW	Acyclic writing state
B(x)	Polynomial presentation of bit string
C	Complexity of problem
Cl	Closed state
CR	Cyclic reading state
CS	Set of cyclic states
CW	Cyclic writing state
D	Amount of data in bytes
EC	Set of cyclic events
Ef	Effort in problem solving
Ev	Set of events
EW	Set of acyclic events
Fi	Set of final states
G(x)	Generator polynomial
In	Initial state
M	Monitoring state
Mo	Moving state
N	Logic operator NOT
O	Logic operator OR
Op	Open state
R	Report reading state
R(x)	Remainder polynomial
Rreq	Read request time in seconds
Rresp	Read response time in seconds
RT	Round time in seconds
S	Transmission speed in bits/s
St	Set of significant states
T(x)	Polynomial presentation of transmitted bit string
T'(x)	Polynomial presentation of received bit string
Tr	Set of transaction rules
Wreq	Write request time in seconds
Wresp	Write response time in seconds

1. INTRODUCTION

This chapter would give the reader a background to understand why this thesis was made and expose the problem and objectives that guided the implementation of the communication system.

1.1 Motivation

For users of automation systems it is important to obtain data from their facilities. Collected data is used to keep track of what is happening in automation devices and the process. Nowadays people will not be monitoring the process all the time. Automation system controls the process using set points from plant operators and then reports back the outcome. [1] Automation system's data is usually widely distributed over many controllers in the system. [2] Since one person can control very large systems from one place, the data from individual controllers, PLC's and measuring devices need to be centralized [3]. There are many ways and protocols to acquire data from remote controllers, measuring devices and motor drives [2]. Most common protocols nowadays are for example Profibus, Profinet, Modbus, Ethercat and CANOpen [4] [5].

In large industrial plants, control rooms with HMI's are usually placed in the same area with the process control [3]. Therefore data communications can be made with fiber optics, cables or high speed wireless technologies like Wi-Fi. In remote monitoring applications like power grid disconnectors, groundwater pumping stations and sewage pumping stations process control devices are distributed over large area [6]. Commonly it is not profitable to install communication cables to these kinds of distant plants [7]. Data communications need to be implemented using different methods.

For decades data communications between monitoring facilities and remote plants have been made using serial radio modems. These are usually easy to use and can transfer most of the commonly used data transfer protocols like Modbus [8]. Radio modems biggest drawback is transmission speed [3]. Simple process control relays on simple measurements like current signals and discreet input and outputs. The amount of data in these applications is low compared to modern control systems with intelligent measuring and control devices. [3] Increase of transferred data has led to situation where serial radio modems are not fast enough to transfer all the needed data in reasonable time. In these cases it is more sensible to use faster technologies like 3G or 4G networks. [9]

1.2 Justification

The way to communicate between SCADA systems and stations in the networks is changing towards IP based systems [3]. Nevertheless there are many solutions in use where

master-slave communication is based on serial communications. Serial radio communication is bit old fashioned technology but still widely used. Mostly because needed hardware has not changed dramatically in the past 30 years therefore old and new radio modem models are still compatible with each other [10]. Therefore end users are not willing to replace their old hardware with different solutions as long as the old devices are still functioning.

Other reason why radio networks are still widely used is that companies own their networks. Finnish Communications Regulatory Authority allocates frequencies for users [11] and therefore there should not be situations where two different users in the same area are using the same frequency and can disturb each other's communications. Also service area of remote monitoring radio networks is up to 50km [12] so chances of intentional or unintentional disturbance to the communications is low [9]. Normally radio communications is used when there are multiple small remote plants e.g. sewage pumping stations within service range. Therefore limited data transfer rate is adequate because amount of data from one plant is low and it is enough to get data from remote plant within round time of radio network [9].

1.3 Problem

The problem itself is quite simple. The devices that are used cannot be changed but the use of the devices could be improved. Developing includes identifying limitations of the devices and how to cope with them. The most important limitation is data transmission speed. In the past the speed has not been that much of an issue because the amount of transferred data was much smaller than nowadays.

Data transfer speed is not the only limitation of serial radios. Modems can only communicate point-to-point, meaning that master station can only read or write one station at the time. That leads to a problem with queue and timing. Preceding transfer must be finished before new transfer can be started and master needs to keep track of which stations it has read thus all stations in network are read or written within one cycle. Keeping in mind the statements of this and previous chapters of the introduction following questions can be made:

- How to make efficient and reliable solution for serial communications control using current features of PLC controller?
- How to organize acyclic data transfer with cyclic one?
- How to prevent and recognize faults and how to manage them?

1.4 Objectives

The goal of this thesis is to implement new communication software for Siemens PLC that will replace the old version of the communication system. Developing the new implementation of the software will start with literature review of present industry practices in industrial communications, data transfer and software methodology. Following part is proposition of system functionality and it is based on use case analyses of the system and its parts. Requirements for the software will be discussed with specialists in Insta Automation Oy to ensure that implementation would be beneficial to the company. Development of the system will be an iterative process. Finding the best way to achieve goal will demand a lot of testing with hardware and feedback from company's representatives. The following list of objectives was determined based on previous problem chapter.

- Study of serial point-to-point based communication system.
    - Study of serial communications fundamentals.
- Analysis of possible use cases
    - Where the solution might be used
- Selection of the methodology used in software development.
    - For understandability software should follow one or more known software development methods.
- Design and implementation of communication system
    - Ensuring that software is easy to understand and modify if needed
    - Implementation of Modbus based point-to-point communication system
- Design and implementation of commissioning software and manual
    - Easy on-site commissioning without knowledge about software itself
- Design guidelines for deployment of communication system.
    - How users can deploy communications system the most efficient way.
- Empirical study of communications systems performance
    - Comparing new solution with old implementation to verify performance.

1.5 Limitations

The performed work will be limited by several issues. The hardware, programming environment and transfer protocol cannot be changed. Limitations will have an effect on the decisions made in this thesis.

- Limitation 1: Siemens TIA environment programmable logic controllers
- Limitation 2: Serial communication over radio or wire
- Limitation 3: All the devices or RTUs in the network can communicate using Modbus
- Limitation 4: Communications in the network is point-to-point communication

1.6 Outline

The thesis is divided into 5 main chapters. Chapter 1 is an introduction to the thesis. In chapter 2 there is literature review of current industry practices of communications and theoretical description of used technologies and techniques. Chapter 3 demonstrates proposal that was made for this work including system principle and demonstration of selected methodology. Chapter 4 explains the steps of implementation based on chapter 3 proposal. Final chapter presents conclusions about the thesis and results of finished project.

2. LITERATURE REVIEW

This chapter presents literature review which describes the technologies and concepts used in this thesis. This section is divided into three main parts: Concepts of communication in process automation, System modeling process and PLC programming methodology.

2.1 ISA-95 Hierarchy Model

ISA-95 standard Part 1 describes the interface content between manufacturing operations, control functions and other enterprise functions. Standards goal is easy integration of inter-operating enterprise and control systems. [13]

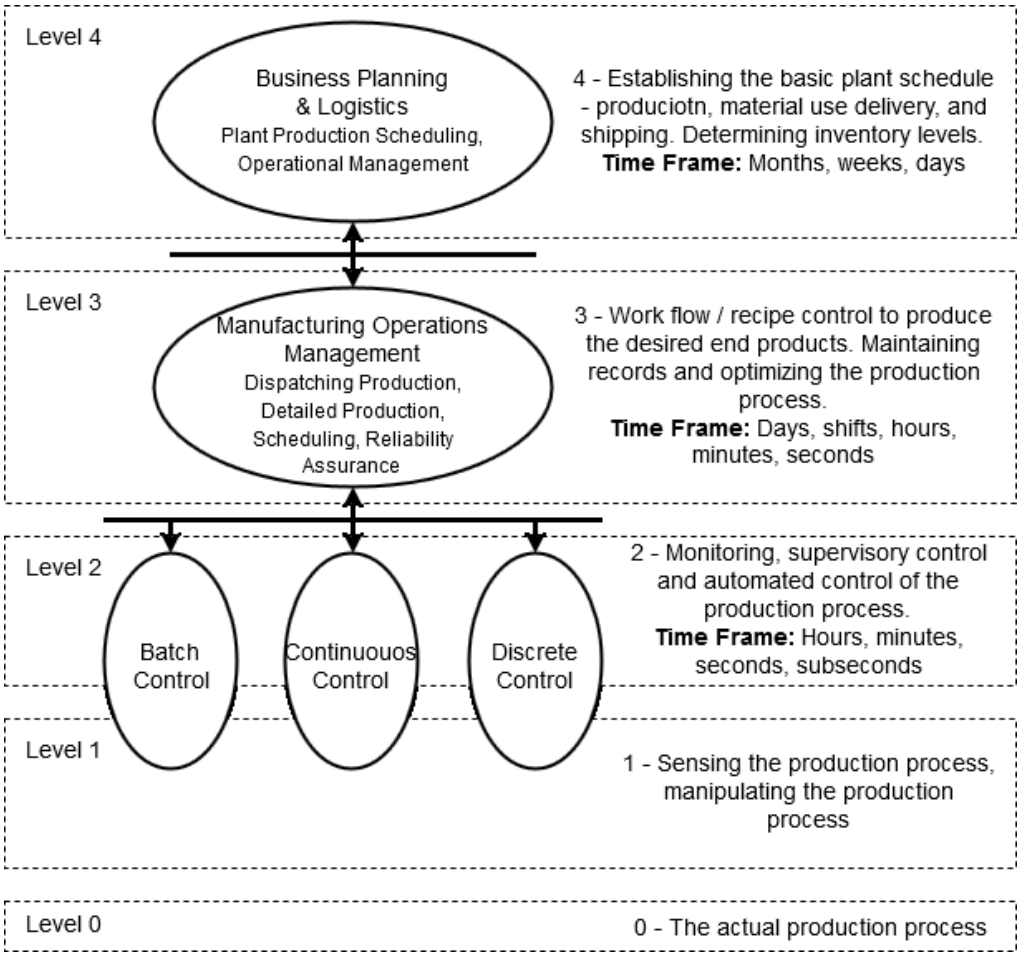

Figure 1 ISA-95 Functional Hierarchy (adapted Hashieman 2010)

Functional Hierarchy is one of the hierarchy models specified by ISA-95 standard (Figure 1). Hierarchy models are associated with manufacturing operations, controls systems and

other business systems. Functional Hierarchy model defines five hierarchy levels. Highest level (Level 4) defines business related activities that are needed to manage manufacturing organization. Level 3 defines the activities of the work flow to produce the desired end-products. Activities of monitoring and controlling physical processes are defined in level 2. Level 1 defines activities involved in sensing and manipulating the physical processes. Lowest level (Level 0) defines the actual physical process. [13]

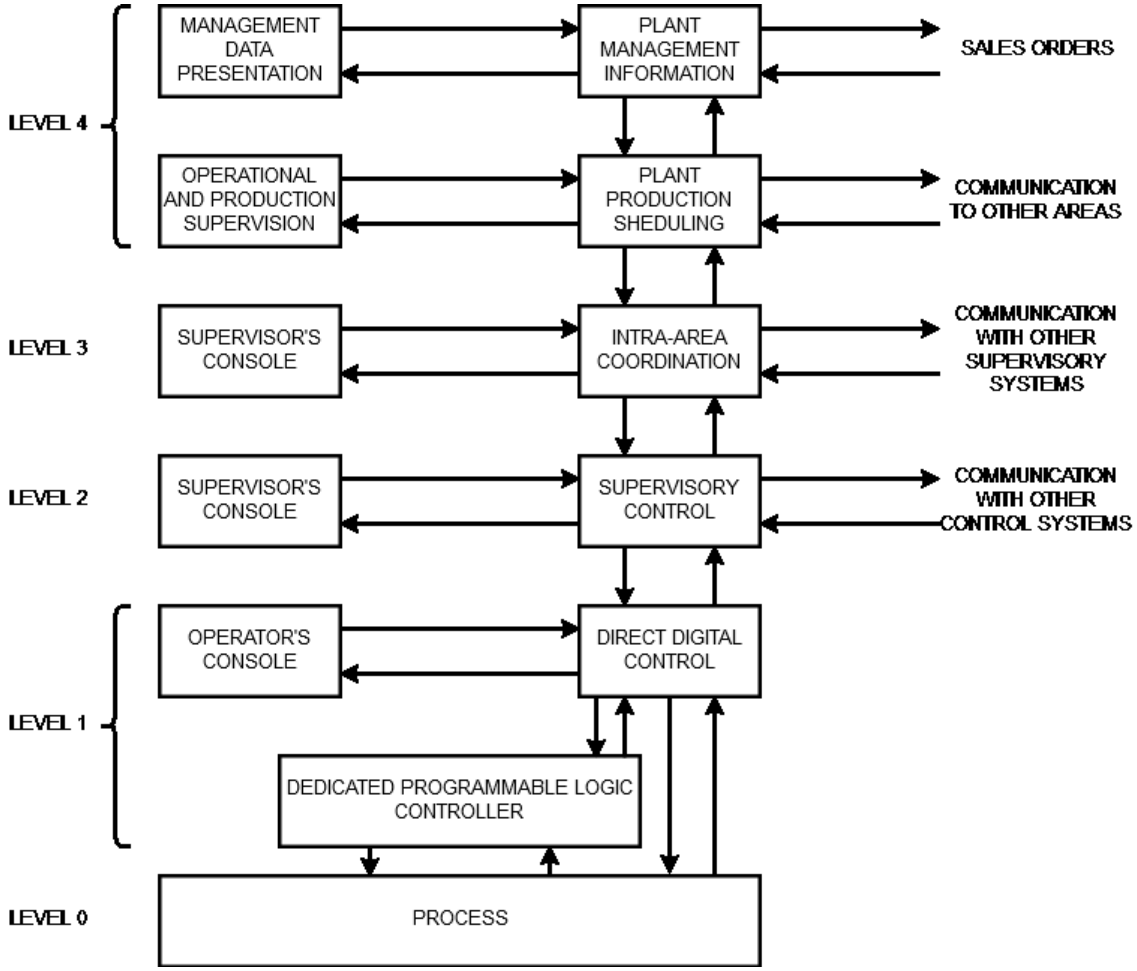

Figure 2 ISA-95 Hierarchical Computer Control Structure for Industrial Plant (adapted Hashieman 2010)

Figure 2 presents computer control structure based on ISA-95 Functional Hierarchy model. Level 2 indicates the control activities that keep the process under control and stable. These control activities can be either manual or automatic. Level 1 means measurements and actuators that are used to monitor and manipulate the process. Level 0 is the actual process, usually manufacturing, production or industrial process. [13]

2.2 Industrial Networks and Fieldbuses

In early days of automation signals between devices were mechanical. Actuating to process was done manually or by mechanical force like hydraulics. When discrete electronics

like relays and hardwired control circuits became more popular mechanical control circuits were replaced. [14] Still these systems were large and required a lot of wiring which made them expensive. When integrated circuits and microprocessors came to market the functionality of large control network could be replicated with a single digital controller. Analogue controls were replaced by digital controllers yet communication in the network was still analogue. Moving toward digital control systems required new communication protocols. [14]

Industrial networks differ from traditional enterprise networks because their operations demand specific requirements [14]. Cyclic data exchange based on predefined scheme is typical in industrial networks. Cyclical network activity is built on established network models like Master-Slave, Token passing, Time slicing or TDMA. All activities in network are executed within of repeated time windows. Given schema defines time-arrangement for cyclical and acyclical events. [1] Depending of network model each node in the network will be able to transfer data between one or more nodes in the network within its own time-window.

Fieldbus technologies were developed to replace signaling techniques like mA-signals. Originally fieldbuses were developed as a digital standard for transferring information between intelligent field devices. Major benefit of fieldbuses was that they replaced point-to-point connections between process control and field devices. [15] As mentioned before, networks for industrial purposes differ from e.g. basic TCP/IP based networks. In enterprise or office networks data is usually send in large packets and any real-time behavior is not required, whereas data packets in industrial networks are smaller and networks need to have real-time capabilities. [15]

Fieldbus is local area distributed control network. It is digital two-way multidrop communication link between intelligent field devices and process control devices such as PLC controllers and DCS systems. [16] Because fieldbuses are two-way communication it is possible to read data from field device as well as write data into it. Multidrop communication facility (Figure 3) can result cabling savings because every field device does not require independent wiring from control devices.

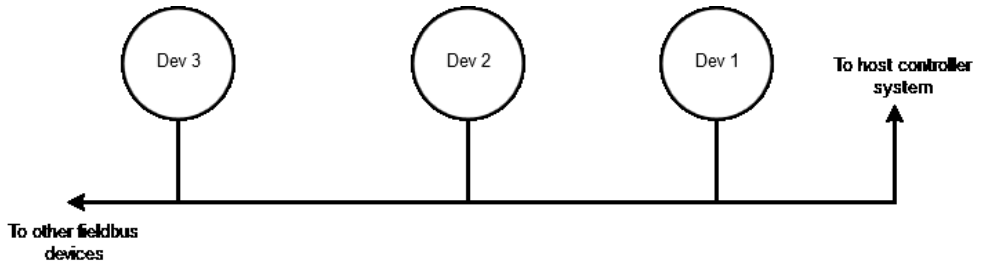

Figure 3 Multidrop fieldbus principle (adapted Sen 2015)

Currently there are multiple fieldbus technologies from multiple vendors for different purposes. Application fields of different fieldbus technologies varies and other fieldbuses

are more suitable in some applications than others. Following presents the most common fieldbus technologies and their general fields of application.

- Actuator Sensort Interface (AS-I) [17]
  - Simple I/O networking
  - Building, Process and Factory Automation
- CAN [18]
  - Originally developed for in-vehicle network
  - Nowadays used in many other industries
- CANopen [18]
  - Originally developed for motion-oriented machine control systems
  - Nowadays used for example medical equipment, maritime electronics and building automation
- ControlNet[19]
  - Serial communication system for time-critical applications
  - Member of Common Industrial Protocol (CIP) family
- DeviceNet [20]
  - Digital multidrop network for connecting industrial controllers and I/O devices.
  - Member of Common Industrial Protocol (CIP) family
- FoundationFieldbus [21]
  - Foundation H1 is intended primarily for process control, field-level interface and device integration
  - Foundation HSE is designed for device, subsystem and enterprise integration
- HART [22]
  - A global standard for sending and receiving digital information across the 4-20mA analog current loops
  - HART is designet to be simple, reliable and easy to use
- Interbus [23]
  - Sensor/actuator bus system for transmission of process data
- ModBus [24]
  - Provides client/server communication between devices connected on different types of buses or networks
- Profibus DP [25]
  - Supports variety of applications in factory and process automation, motion control and safety-related tasks
- Profibus PA [26]
  - Extended version of Profibus DP for process automation
- Seriplex [27]
  - Developed for industrial control applications by Automated Process Control Inc in 1980s.

- Transmits both digital and analog I/O signals for both control and data acquisition applications

2.2.1 Data Transmission in Serial Communication Media

This chapter presents principles of serial data transmission and methods to ensure data integrity.

2.2.1.1 Data transmission modes

Data transmission between devices can be divided into different modes. The two basic modes are serial and parallel data transmission. In serial data transmission bits are sent from one device to another consecutively trough one wire, where in parallel data transmission usually eight bits are transferred simultaneously trough separate wires (Figure 4). [28]

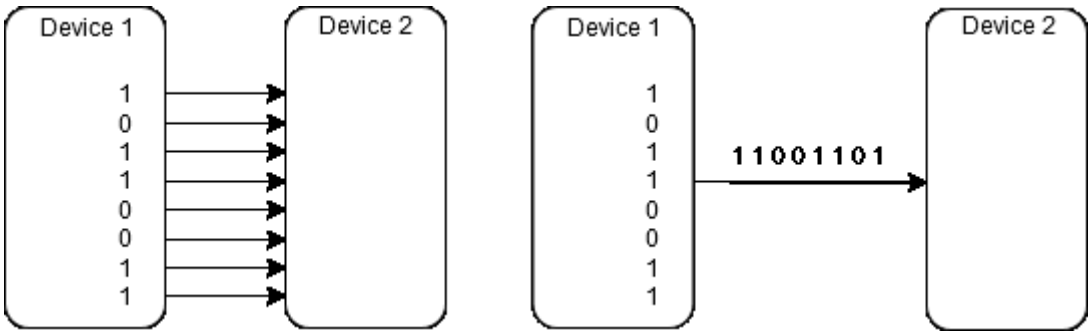

Figure 4 Parallel and serial data transfer (adapted Shay 1995)

Serial data transfer can be provided in two different ways, asynchronous and synchronous transfer. In asynchronous transfer bits are collected into small groups, usually bytes (8 bits). Bytes are sent independently from one device to another, meaning that sending device can send bytes at any time and receiver never knows when data will arrive. In synchronous data transfer larger data packets are sent per one transmission. Organization of data packet or frame depends on protocol that is used. Figure 5 presents differences between asynchronous and synchronous data transfer. First sent bit is the start bit that "wakes" receiving device for data transfer. Then data bits are sent consecutively from bit 0 to bit 7. The last bit is a stop bit that tells receiver that data transmission is finished. In synchronous data transfer first are sent synchronization and control data, followed by payload and error correction data. End of frame indication is sent last to tell receiving end that the data transmission is complete. [28]

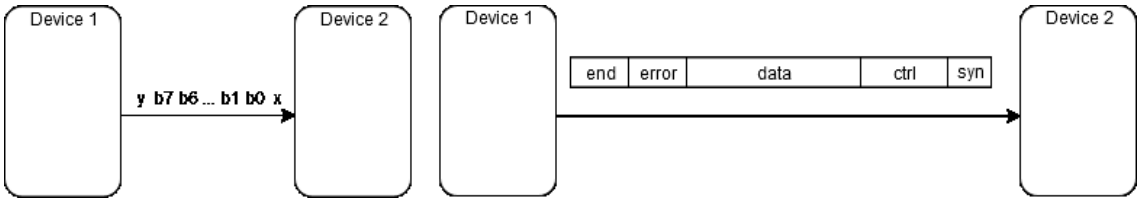

Figure 5 Asynchronous and synchronous data transfer (adapted Shay 1995)

Previous examples are considered as simplex communication (one-way), because there is no communication from device 2 to Device 1. Simplex communication is useful if reply from receiving device is not required. Many applications require more flexibility in which both devices can send and receive data. Basic bi-directional communications can be divided into two main methods, half-duplex and full-duplex communication. In half-duplex method both ends can send and receive but they must alternate sending and receiving. In full-duplex method both ends can send and receive simultaneously. In Figure 6 is presented difference between two methods. In half-duplex communication denoted as solid line cannot occur same time with communication denoted with dashed line whereas with full-duplex communication both Device 1 and Device 2 can send at the same time. [28]

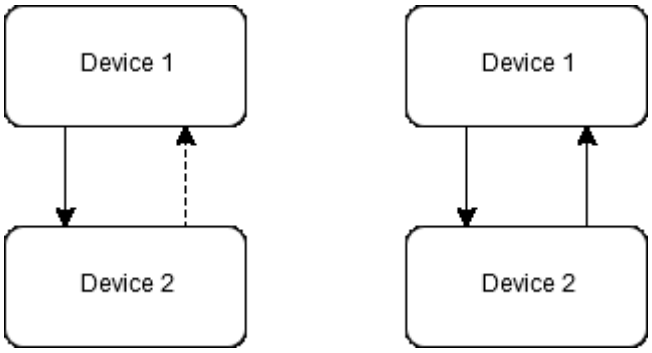

Figure 6 Half-duplex and full-duplex communication (adapted Shay 1995)

2.2.1.2 Data Integrity

Previous chapter dealt with mechanisms that are necessary to transmit data between devices. Integrity of data cannot be guaranteed without additional error detection and correction techniques. Simple and common technique for error detection is parity checking. It is a simple mechanism that can detect if one bit is erroneous in transmitted data. [28] For example ASCII character “C” presented in binary code is 01000011. When it is transmitted in asynchronous data format additional start and stop bits are added in the beginning and to the end of transmission. Parity checking adds one more bit to message then the total length of one transmission is 11 bits. [12] Order of bits in transmission is presented in Figure 7.

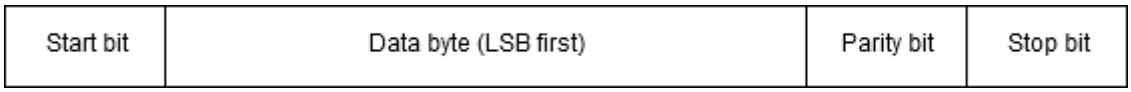

Figure 7 Bit order in transmission

Even parity checking calculates number of 1 bits from data byte. If number of 1's in data byte is equal number, parity bit is set to 0. With odd parity checking if number of 1's in data byte is equal parity bit is set to 1 [28]. In example binary format of ASCII character "C" contains three bits of value 1. Therefore with even parity checking parity bit is set to 1 to create character with even number of bits with value 1. Odd parity checking in this case is set to 0 because number of bit 1's is already even number (Table 1). Stop bit will not be considered as a part of character and will not be calculated as value 1 bit.

Table 1 Parity bit comparison

Data format	Character	Character length
8 bit, even parity	01100001011	11 bit
8 bit, odd parity	01100001001	11 bit

Parity checking will detect any single bit errors in message. But if two bits in message are erroneous parity checking will not detect it. For example two bits both value 0 are read as 1. Parity calculation will not detect error because two extra 1 bits produces same parity when calculation is made in receiving end. Double bit errors can be recognized when double bit error detection is used. For example parity can be calculated from even and odd bits independently. In previous example parity could be calculated from even bits of binary **0**1**0**0**0**1**1** and from odd bits 0**1**0**0**0**1**1 independently. Amount of 1's in even bits (0001) is odd and even in odd bits (1001). If two consecutive bits in binary for example 010**11**011 changes from 0 to 1, double bit error detection will recognize error, because amount of 1's in even bits (0011) is now even, and 1's in odd bits (1101) is odd. [28]

Parity checking itself is not very reliable error detection method because multiple erroneous bits cannot be detected. More elaborate way of error detection is to detect errors in frames. With Cyclic Redundancy Check or CRC it is possible to detect erroneous bits in received frame whether they occur in single or consecutive bits. In CRC every bit string is interpreted as polynomial. In general bit string is interpreted as (equation 1)

$$b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + b_{n-3}x^{n-3} + \cdots + b_2x^2 + b_1x + b_0 \quad (1)$$

Polynomial presentation of bit string is marked as B(x). G(x) is a generator polynomial for calculating remainder polynomial R(x) equation 2.

$$R(x) = \frac{B(x)}{G(x)} \quad (2)$$

T is transmitted bit string corresponding to T(x) (equation 3) and T' is received bit string corresponding polynomial T'(x)

$$T(x) = B(x) - R(x) \quad (3)$$

Receiving end will calculate division (equation 4) and if remainder is 0 receiver can determine that T'(x) = T(x).

$$\frac{T'(x)}{G(x)} \quad (4)$$

Following example presents CRC calculations of bit string B = 1101011 and generator bit string G = 11001. First is needed to append additional zeros to bit string. Number of zeros is same as degree of generator polynomial G(x) which is four. Then bit string B is 11010110000. Then R(x) can be calculated by dividing B(x) by G(x). The result R is bit string 1010. Then transmitted polynomial T(x) can be calculated by subtracting R(x) from B(x). The result is bit string T = 11010111010. T is actually same as bit string B with added zeros replaced by bit string R. [28]

If received bit string T' is same as sent bit string T division on T'(x) by G(x) yields zero remainder. Whereas errors occurred while transmission remainder of calculation T'(x)/G(x) yields non-zero remainder receiving end can conclude that there were errors in transmission and it can reject received data. It is possible that received T'(x) is not same as T(x) but still yields zero remainder when divided by G(x). But when G(x) is chosen properly this kind of errors occurs rarely. [28]

2.2.2 MODBUS

One widely used protocol in industrial networks is Modbus, originally developed by Modicon in 1979. Modbus is messaging protocol positioned at layer 7 of the OSI model. Client/Server communication can be established between devices that are connected using buses or networks. Modbus communication can be implemented using TCP/IP over Ethernet or as serial communication using mediums like wire, fiber optics and radio. [24] Modbus TCP/IP is not covered in this thesis.

Modbus over serial line or Modbus RTU is Master-Slave protocol where only one Master at a time can be connected to network. Communication is always initiated by Modbus Master. Slave cannot communicate with other slave directly but trough Master. Only one transaction can be initiated at the time, meaning that Master can only communicate with one Slave at the same time [24].

2.2.2.1 Modbus Master

Figure 8 explains Modbus master behavior. Modbus Master can issue a request to a slave using two different modes, unicast and broadcast. In unicast mode master sets an individual address to the request and waits for reply. After reply reception, reply from slave is processed and master goes to idle state, which means there is no pending request. If slave does not reply the request within response timeout or reply from slave has errors, master process error(s) and goes to idle state. Other request type is broadcast request. In this mode Modbus master will set address 0 to request and therefore every Slave in the network will receive the request. Slave devices will not send response to the master with broadcast requests. Master goes back to idle state after delay expiration timeout. New request can only be sent when Master is in idle state. When request is sent, master leaves idle state and it cannot process another request at the same time [29].

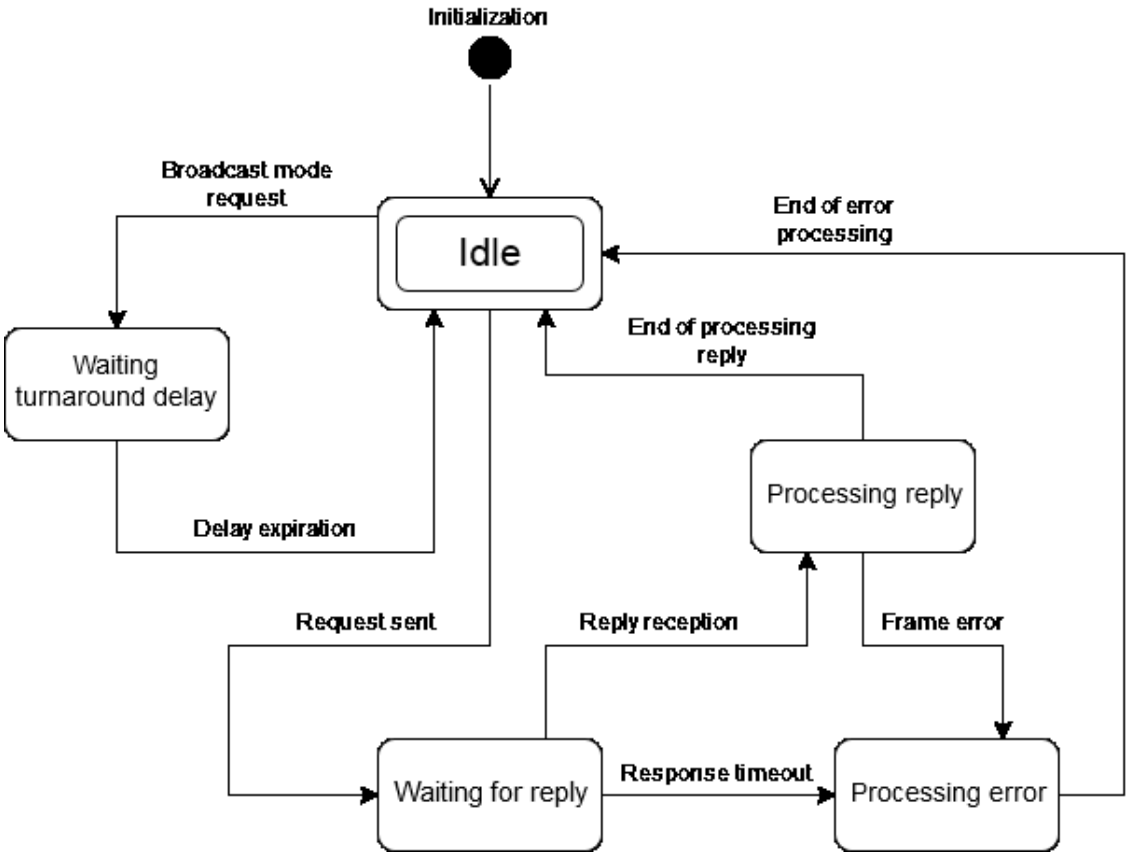

Figure 8 Modbus Master state diagram (adapted Modbus 2002)

2.2.2.2 Modbus Slave

Figure 9 presents Modbus Slave state diagram. Idle or no pending request state is Modbus Slave's initial state after power up. When Slave receives request from Master, Slave changes from idle to checking request state and checks the packet before continuing processing state. If there is an error in the frame or slave address does not match slave goes

back to idle state. If there is no errors in checking, slave processes desired action and formats reply to Master. Processing the errors will set Slave to error reply state. Modbus implementation guide requires that error reply must be sent to the Master.

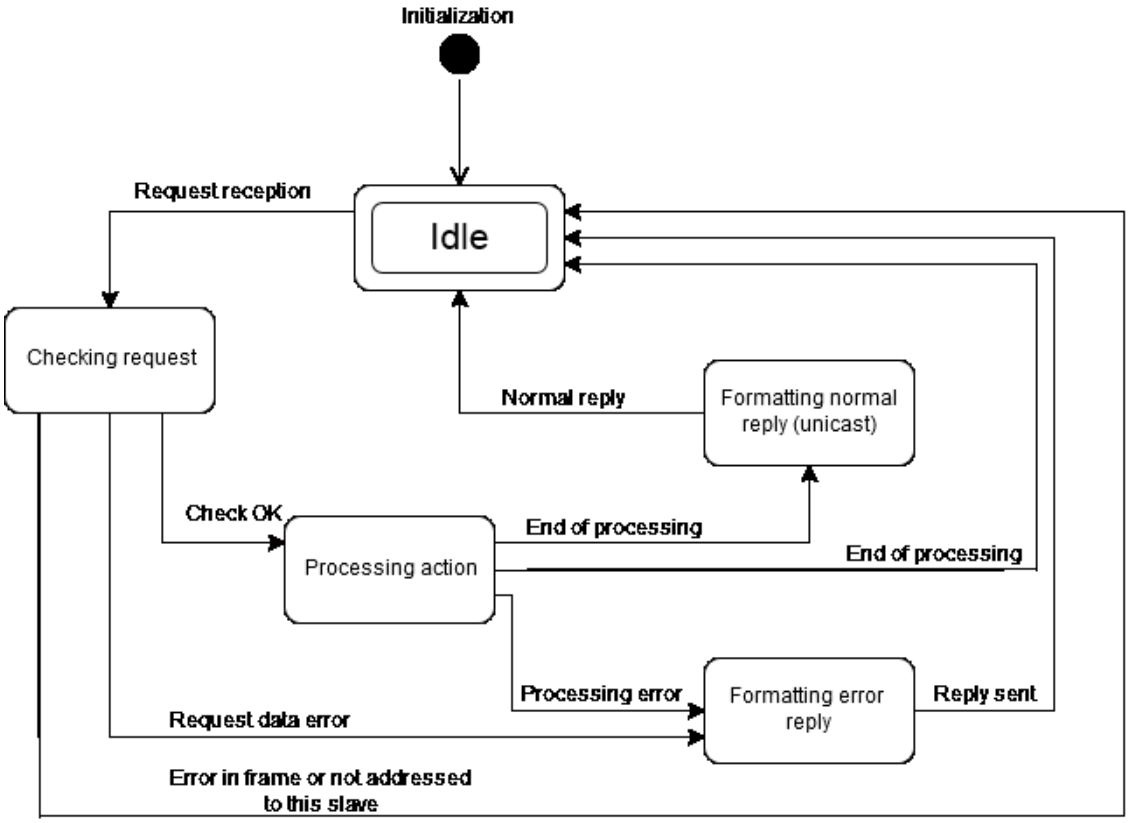

Figure 9 Modbus Slave state diagram (adapted Modbus 2002)

2.2.2.3 Modbus Addressing and Data Unit

Modbus over serial line implementation guide defines addressing for Modbus Slaves [Imp guide]. Each Slave in the network must have unique address between 1 and 247. Address range also defines the maximum amount of slaves in one network to 246 (247 – 1 = 246). As mentioned before, address 0 is reserved for broadcast messaging. If broadcast sending is used in the network, all the Slave nodes must be able to recognize the broadcast address [30].

Simple Protocol Data Unit (PDU) (Figure 10) is defined in Modbus protocol. PDU is independent of OSI model's layers 1 and 2, meaning that PDU will be unchanged in both Modbus over serial line and Modbus over TCP/IP.

Function Code	Data

Figure 10 Modbus PDU frame

Protocol Data Unit in Modbus serial line has two additional fields (Figure 11). In serial mode addressing and error detection information is added to simple PDU.

Address Field	Function Code	Data	CRC or LRC

Figure 11 Modbus Serial Line PDU

Modbus features two serial line transmission modes RTU (Remote Terminal Unit) and ASCII. Modes define the bit content of message fields and how information is packed when transmitted over serial line. In Modbus RTU transmission mode each 8-bit message contains two 4-bit hexadecimal characters. One Modbus RTU frame consist four different parts (Table 2). First part is Slave Address that can be number from 1 to 247, function code defines to receiver (server) what kind of action to perform. Third part of Modbus Message Frame is payload and the last one is for CRC checksum. [30]

Table 2 Modbus RTU Message Frame

Slave Address	Function Code	Data	CRC
1 byte	1 byte	0…252 bytes	2 bytes

In Modbus ASCII transmission mode each 8-bit message contains two ASCII characters. Message frame has two additional fields compared to RTU Message Frame. Start-of-frame and End-of-frame characters are used to delimit Message Frames. Start character ':' and End characters 'D' and 'A' are defined by Modbus specification. LRC is Longitudinal Redundancy Check. LRC is calculated by sender. Receiver calculates LRC again and compares results and non-identical values results an error. [30]

Table 3 Modbus ASCII Message Frame

Start	Address	Function	Data	LRC	End
1 char	2 chars	2 chars	0…2x252 chars	2 chars	2 chars

2.3 Unified Modeling Language (UML)

This chapter presents basics of Unified Modeling Language. Most of the modeling methods consist at least modeling language and modeling process. Modeling language is mainly graphical notation which is used to describe plans. Modeling language is possibly the most important part of modelling method. Language is major part of communication between people. If plan is needed to describe to someone else, both needs to understand modeling language, but not the whole design process behind plans. [31] UML 2.5 standard defines 14 different diagrams [32]. This chapter presents the four different UML diagrams that are used in this thesis.

2.3.1 Use Case Diagram

Use Case Diagram defines what systems are supposed to do. In Figure 12 is presented simple Use Case diagram of ATM System. Use Case diagram consists of Use Cases, Actors and subjects. In this case there are five different Use Cases (oval shapes), three actors (stick figures) and one subject (ATM System rectangle). [32]

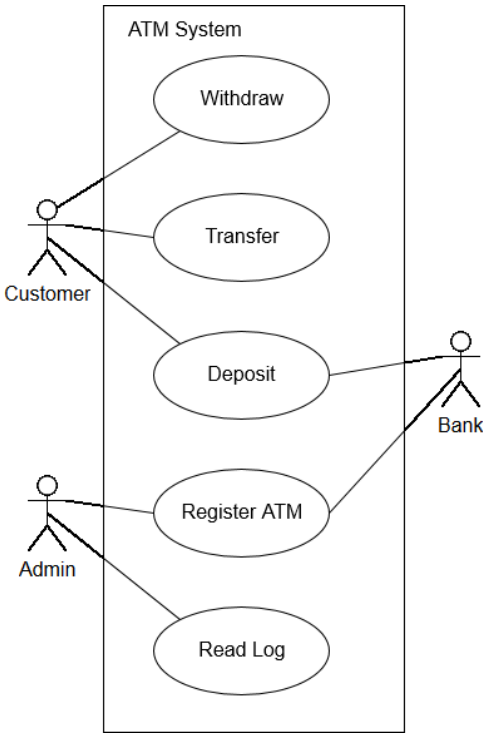

Figure 12 UML Use Case diagram of ATM System (adapted OMG 2017)

2.3.2 State Diagram

State Diagram describes systems behavior over time. The behavior of the system is modeled in terms of what states system is in at various times. [33] In Figure 13 is presented simple State diagram of ATM machine. State diagram (or State Machine) consists of initial state (black circle), states (rounded rectangles) and transitions (arrows). [34] In this case machine changes from initial state to Idle state without triggering event. Transition from Idle state to Active state occurs when card is inserted into a machine. Idle state is activated again if events cancel or done occurs. Third state of the machine is Out of Service state. As can be seen from diagram Active state cannot be triggered from Out of Service state before machine is fixed and it is in the Idle state. [31]

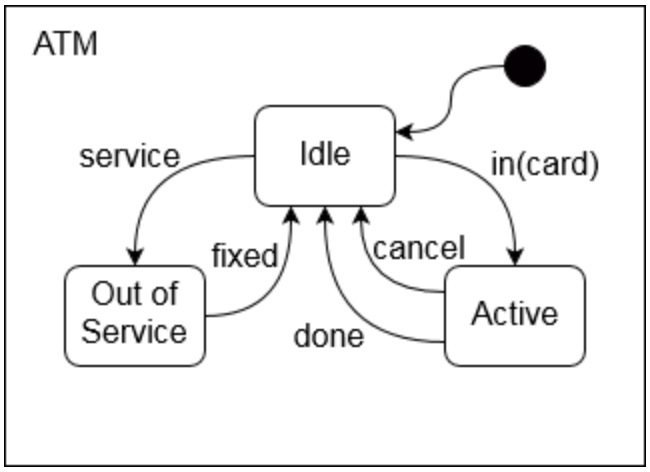

Figure 13 UML State diagram of ATM machine (adapted State Machine Diagrams)

2.3.3 Sequence Diagram

UML Sequence diagram is the most common kind of interaction diagrams. It represent the details of a UML use cases and how components interacts with each other. [35] Sequence diagram focuses on interchanging messages between lifelines. [32] Figure 14 presents sequence diagram of ATM machine. Entities interacting with the system are described as stick figures. In this case user is interacting with ATM machine to check balance of bank account. Activation box depicts the time needed for to complete the task. Objects within the system are presented as rectangles. Object symbols demonstrate how objects behave in the context of the system. Every message is presented as an arrow between lifelines of objects. Messages can be either synchronous (sender waits for reply to message) or asynchronous (reply to message is not required) [35]

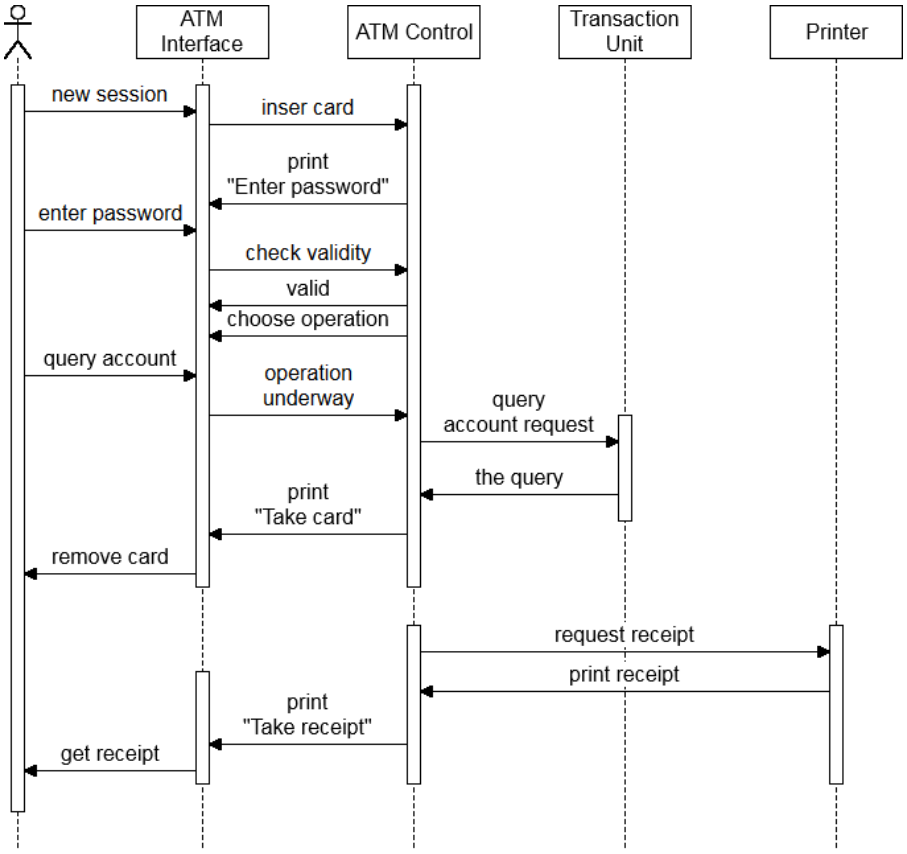

Figure 14 UML Sequence diagram of ATM machine (adapted UML Sequence Diagram Tutorial)

2.3.4 Class Diagram

UML Class diagram depicts static relations between objects of system. Classes of system are presented as rectangles. Every class in the system has a name (first bolded row inside class), Attributes (second row inside class) and Methods (last row) [36]. Also there is interaction between classes marked as lines, arrows or lines with diamond shapes at the end. [37]

Interaction with white diamond shape is basic aggregation. It indicates that one class is part of another class. In this case ATM, Customer and Account classes are part of Bank class. Another interaction is uni-directional association. It is marked with black solid arrow from class to another. This interaction means that two classes are related, but only one class (where arrow is pointing) knows that the relation exists. Bi-directional association is line between classes. In this case Customer class has two accounts and one Account class can have multiple ATM Transactions. Last interaction shown in Figure 15 is inheritance marked as white hollow arrow head. Inheritance means that one class (child or subclass) has an ability to inherit the identical functionality of another class (Super class). [37]

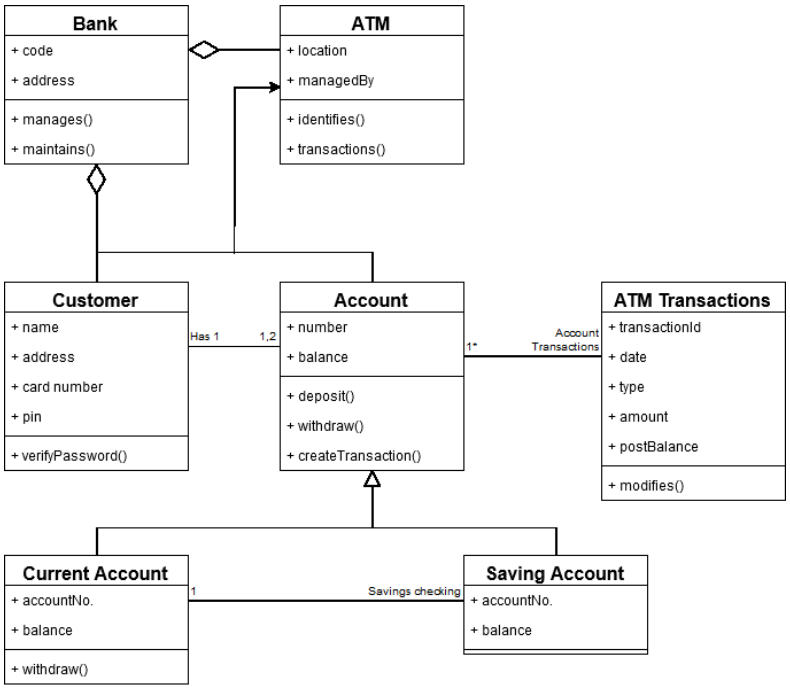

Figure 15 UML Class diagram of Bank system (adapted UML Class Diagram Tutorial)

2.4 PLC Programming Methodology

This chapter provides background for understanding Programmable Logic Controller's software and communication models, program organization units and programming languages based on IEC 61131-3 standard. Also data storages, software modularity and encapsulation and diagrams for program design are covered in this chapter.

2.4.1 IEC 61131-3

IEC 61131-3 is part of International Standard IEC 61131 and it specifies syntax and semantics of programming languages for programmable controllers. [38] IEC 61131-3 is the first real endeavor to standardize industrial automation programming languages [39] because it is vendor independent and it is worldwide supported [40].

2.4.2 IEC 61131-3 Software Model

Configuration is the highest level in the software model of IEC 61131-3 (Figure 16). IEC 61131-1 standard defines configuration as a programmable controller system [38]. Configuration includes hardware arrangement, system capabilities, processing resources and memory addresses. Formulated configuration is capable of solving particular control problem and one or more resources can be defined within one configuration. IEC61131-1 defines resource as signal processing function and its machine, sensor and actuator interface functions [38]. A single resource consists of tasks and programs. Tasks controls execution of programs, function blocks or both and execution of tasks can be periodic or

by trigger like change of a variable. Programs typically consists of number of functions (FUN) and function blocks (FB) written in any of IEC 61131-3 languages. [41]

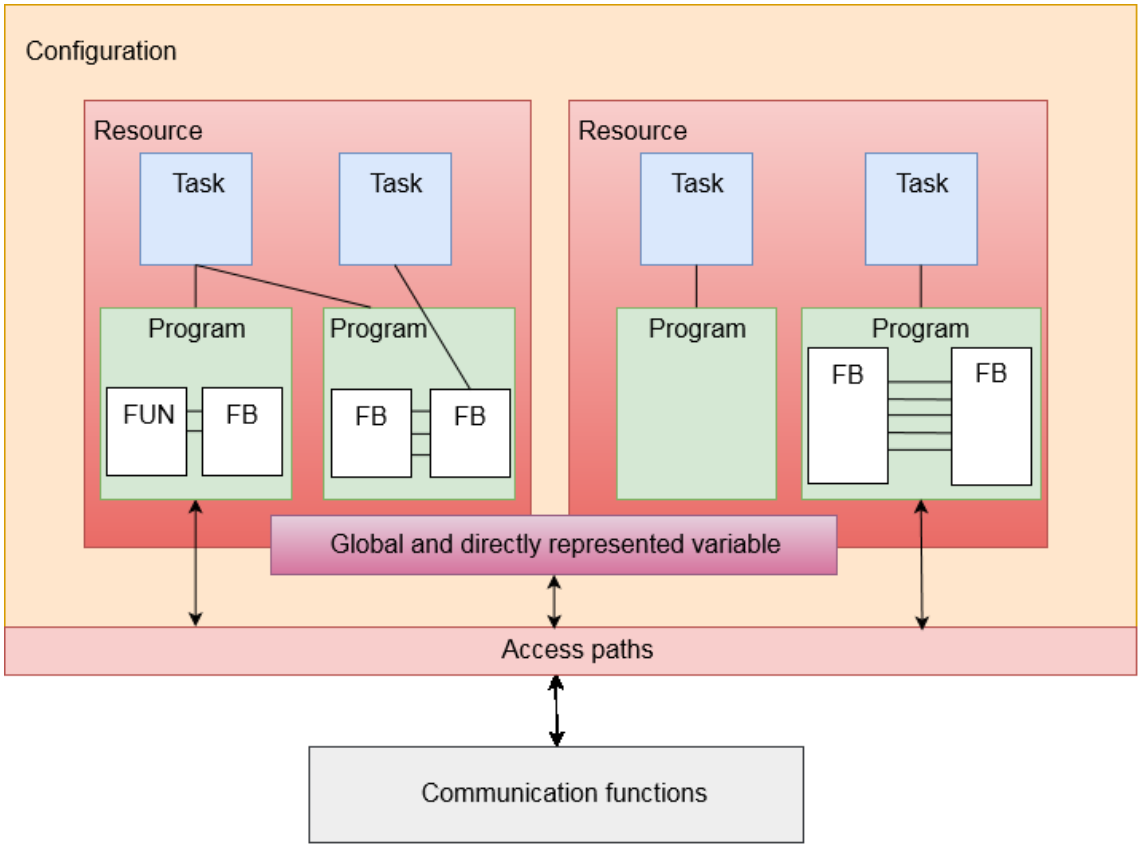

Figure 16 IEC 61131-3 software model (adapted Suomen standardoimisliitto 2006)

2.4.3 IEC 61131-3 Communication Model

Communication between software elements is illustrated in Figure 16. Variables within one program can be transferred between elements from output of one FB or FUN directly to input of another. In graphical, e.g. Function Block diagram, languages connection between elements is shown explicitly and implicitly in textual languages e.g. statement list.

Communication between programs within one configuration can be executed using global variables (Figure 17). In this example Boolean variable x is been passed from output “a” of Function Block in program A to input “b” of Function Block in program B.

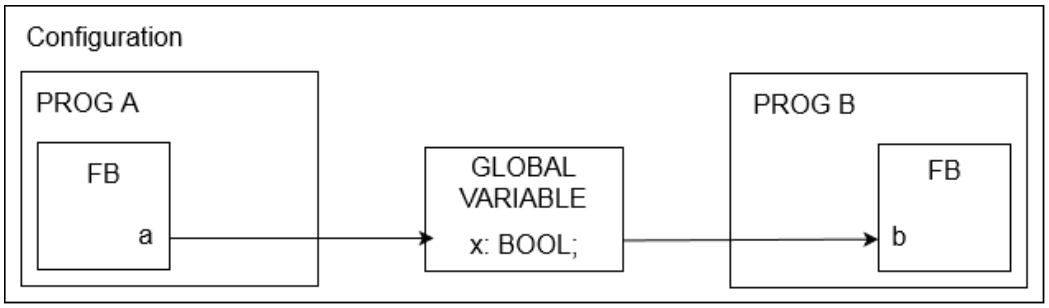

Figure 17 Variable communication between programs (adapted Suomen standardoimisliitto 2006)

Variables can be communicated not only between two programs in the same configuration but programs between different configurations using communication functions defined in IEC 61131-5 standard. In example Figure 18, configuration C could be programmable logic controller and configuration D non-programmable device like radio modem. [38]

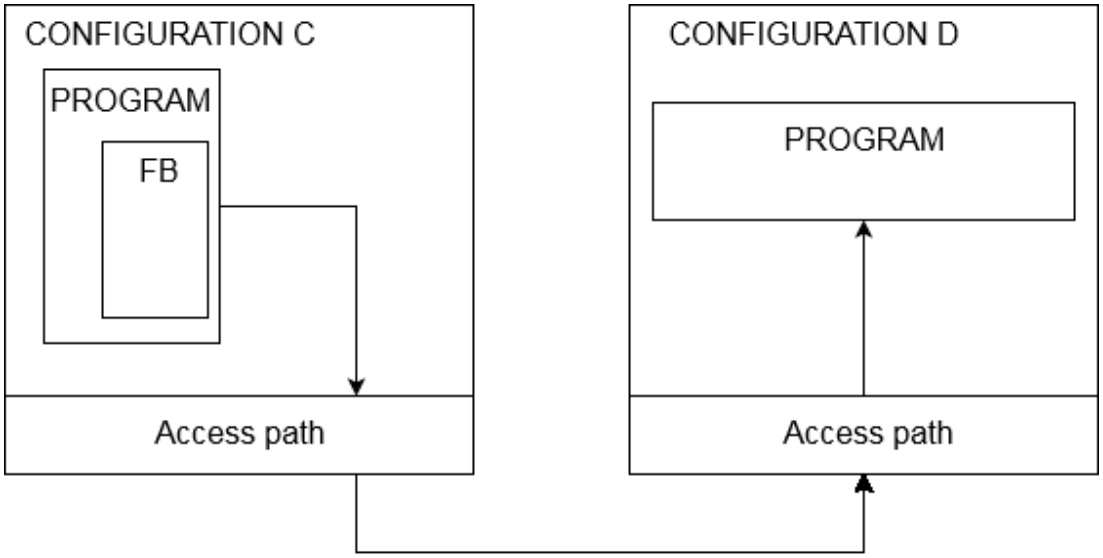

Figure 18 Communication (simplified) between two different configurations (adapted Suomen standardoimisliitto 2006)

2.4.4 Program Organization Units

Program Organization Unit or POU is concept of IEC61131-3 standard and it defines three different POU types, Program, Function Block and Function [38]. Declaration of type defines the behavior and the structure of POU. In following three chapters different POU types are explained.

Important concept in IEC 61131-3 is calling hierarchy of POUs. If POUs are considered as levels where Program is in highest and Function in lowest level calling can be made from higher to lower level or between POUs of same level (Figure 19).

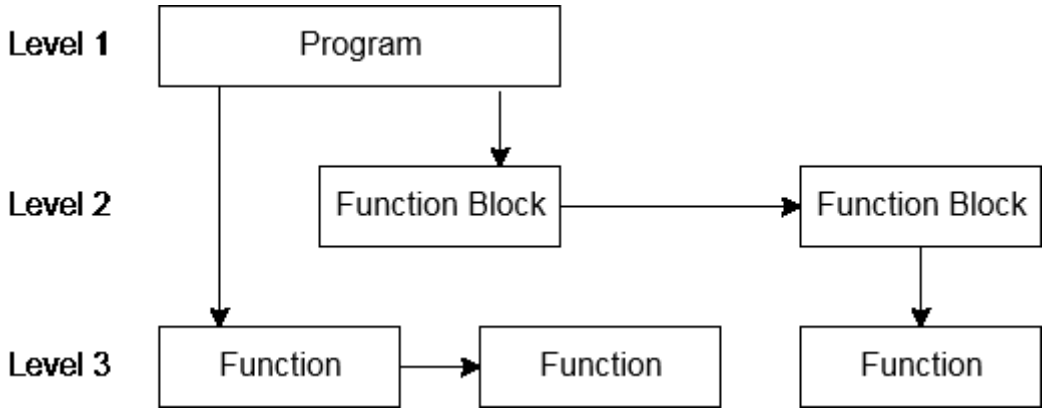

Figure 19 Calling hierarchy of IEC 61131-3 (adapted John, Tiegelkamp 2010)

Another important principle is that recursive calling is not permitted. Invocating of POU shall not cause invocating POU of same type, e.g. function A cannot be called by itself. [38] Recursive calling is problematic, because real-time behavior of recursive software is not predictable [42].

2.4.4.1 Program (PROG)

Program is defined in IEC 61131-1 as a "Logical assembly of all the programming language elements and constructs necessary for the intended signal processing required for the control of a machine or process by a programmable controller system" [38]. Program (PROG) type of POU represents the main program in PLC. All variables that are accessed in whole program must be assigned in this POU, meaning that Functions or Function Blocks cannot access for example physical addresses of PLC if they are not declared in PROG POU. [43] Programs are instantiated within resource as shown in IEC 61131-3 software model in Figure 16 [42]. Controllers with multitasking capabilities can have several PROGs running parallel [43].

2.4.4.2 Function Block (FB)

Function Blocks (FB) are the main building blocks of PLC programs. Elementary parts of FB's are variables and algorithms. One Function Block can have unlimited number of input, output and internal variables. Input variables are passed to FB from for example physical PLC inputs, other Function Blocks or Data Blocks. Output variables are the results of FB's internal algorithms and they can be passed to PLC physical outputs, other FB's or Data Blocks. Declaration of FB is called instantiation. Every time FB is used in the program new instance must be created to it. Instance is FB's memory and it contains the values of input, output and internal variables. Therefore individual instance must be created every time FB is used in the program. [43] When Program can be instantiated only within resources, Function Blocks can be instantiated within Programs or other Function Blocks [42]. It means that instance of FB can be declared in instance of other FB.

One Function Block can be used as many times as needed in one Program. Every time FB is used in the Program new copy or instance is created. Every instance has unique identifier or name. When FB is called by program the input variables are loaded to its internal memory (instance memory). FB's internal algorithm is then executed and output variables are written. Because FB has internal memory, invocating the same input variables do not always yield the same output. FB's internal variables are hidden from user of Function Block, meaning that only input and output parameters are accessible outside of the instance of the FB. [42]

2.4.4.3 Function (FUN)

Function is defined in IEC61131-3 as a POU which yields exactly one data element when executed. Output of function is considered its result. Data element can be multi valued, meaning that function can have several output parameters. [38] Different from Function Block, Functions don't have internal memory. When FUN is called in the Program with same input values, Function yields exactly the same output every time. [42]

Declared function can be used (called) multiple times by other POUs [42] . Function cannot call Programs or Function Blocks. So it is not permitted to FUN to call standard FB's like timers and counters as explained in chapter 2.4.4.

2.4.5 Programming Languages of IEC 61131-3 standard

IEC 61131-3 standard defines four different programming languages. Syntax and semantics of languages have been defined so there is no room for dialects [41]. These four languages can be divided into two main groups, textual and graphical. Textual languages consists instruction list (IL) and structured text (ST). Graphical languages are ladder diagram (LD) and function block diagram (FBD). [38] Sequential Function Chart (SFC) is also defined in IEC61131-3 standard but it is not considered as full programming language.

In following chapters a simple mathematical formula (5) is converted into each of four languages defined in IEC 61131-3. Sequential Function Chart is not considered as a complete programming language, see chapter 2.4.5.5.

$$I0.0 \wedge I0.1 \vee I0.2 \wedge I0.3 = Q0.0, \tag{5}$$

2.4.5.1 Instruction List (IL)

Instruction List (IL) is assembler like programming language. It is low level language and effective for small applications or for optimizing parts of larger application [42]. Other

textual and graphical languages are commonly translated into instruction list for PLC execution. Instruction list is line-oriented language. One instruction is an executable PLC command and it is described exactly in one line. [43]

Expression in formula (1) can be converted into Instruction List code using operator A(AND), O(OR) and =(EQUALS).

A	“I0.0”	Input channel 0
AN	“I0.2”	Input channel 2
O	“I0.1”	Input channel 1
A	“I0.3”	Input channel 3
=	“Q0.0”	Output channel 0

2.4.5.2 Structured Text (ST)

Structured text is another textual PLC programming language of IEC 61131-3. It is comparable with PASCAL and C languages in PC world. ST is very powerful language to describe complex functionality in compressed way. It does not use low level machine orientated operators like IL, but abstract statements. Basic categories of ST statements are assignment, selection, iteration, function and function block and control statements [42]. High level of abstraction can lead to loss of efficiency, thus compiled programs could be slower and longer. [43]

Example equation is converted into ST code (Figure 20) using Boolean operators “AND” and “OR”, selection statement IF…THEN and assignment statement :=.

```
1 IF (#"I0.0" AND #"I0.1" OR NOT #"I0.2" AND #"I0.3") THEN
2     #"Q0.0" := true;
3 ELSE
4     #"Q0.0" := false;
5 END_IF;
6
```

Figure 20 Structured text of example equation

2.4.5.3 Ladder Diagram (LD)

Ladder diagram (LD) is primarily designed for processing Boolean signals. The root of LD comes from electromechanical relay systems. Ladder diagram uses networks to describe “power flow”. It is processed from top to bottom and from left to right, but user can specify otherwise if needed. Boundaries of network are “power rails”. Power reaches components in the network, depending on their logic state. Component either allows or blocks the power flow to the next component in the network. [43]

Example equation written in LD (Figure 21) is lot easier to understand than textual version. Connections between variables are presented explicitly thus it is easy to understand the “power flow” trough components in the network.

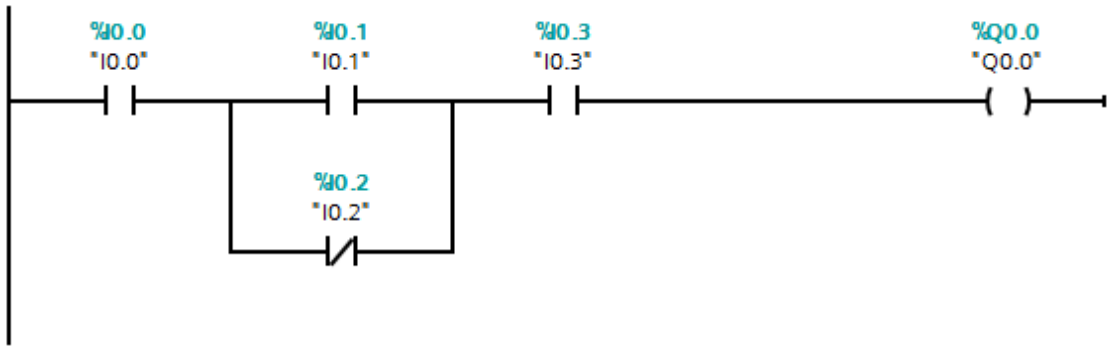

Figure 21 Ladder diagram of example equation

2.4.5.4 Function Block Diagram (FBD)

Function block diagram (FBD) is commonly used in process control. Like ladder diagram FBD also uses networks to present interconnections between function blocks. Connections between blocks are not considered as “power” like with LD, but signals. [43] Signals in case of example equation are Boolean (true and false) signals. Function Block Diagram is not necessarily as easy to understand as Ladder Diagram, but FBD has advantage when there is need to make calculations with different data types like real or integer numbers. [43]

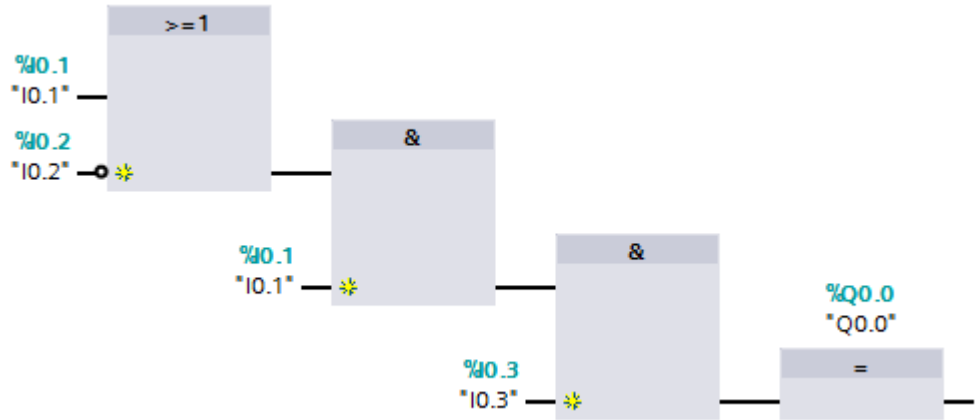

Figure 22 Function block diagram of example equation

2.4.5.5 Sequential Function Chart (SFC)

Complex programs with multiple steps can be divided into smaller units. These units would be more manageable and therefore easier to understand [43]. SFC uses step-tran-

sition sequence. Linking elements of same type, step or transition, is not permitted, meaning that after every step there must be transition and vice versa. Steps are either active or inactive. When step is active the associated actions are executed repeatedly until step becomes inactive. Activating or inactivating steps is controlled by transitions (Figure 23). Transition from step “ST” to “SB” occurs when Boolean expression “Trans” becomes true. [44]

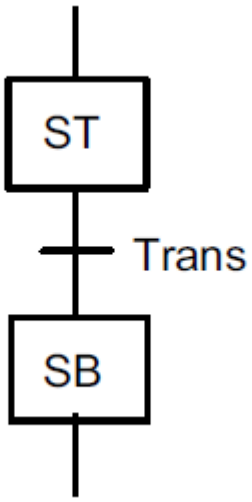

Figure 23 SFC steps and transition

Though SFC is very expressive, it is not considered as a full programming language in IEC61131-3. That’s because parts of program like actions and transitions require implementation using other languages, like FBD. [45]

2.4.6 Software Modularity and Re-usability

Re-use rate of the software is important aspect in all software engineering. Without leverage of previous experiences with similar systems requirement, specification and analysis is time consuming operation. Designing of control software will require less time if designs from previous cases are properly executed and documented. Small variations to existing software modules are easy to carry out and could reduce system cost and development time. [42]

It is common in engineering culture to decompose large and complex problems into simpler sub problems. Particularly in software engineering proper division of problem is crucial to ensure code understandability and readability. Modularity is a key element for software internal quality. Decomposition of problem reduces overall complexity. Problem complexity x is represented with function C. As can be seen from equation (6), whole problem is always as complex as or more complex than sum of complexity of sub problems. [42]

$$C(x_1 + x_2) \geq C(x_1) + C(x_2), \quad (6)$$

Effort (Ef) equation (7) is basically same as complexity equation. Most of cases effort equals cost. When effort of solving problem is decreased also cost of problem solving is reduced. [42]

$$Ef(x_1 + x_2) \geq Ef(x_1) + C(x_2), \quad (7)$$

Module coupling is key factor of re-usability. No coupling means that all modules of software are independent. Without coupling of some degree software modules are unable to communicate or change data between each other. It is obvious that some degree of coupling is needed. Coupling has effect on internal quality of the code. Number of links (couplings) between modules is inversely proportional to the code quality. System with N module, range of interactions is between N-1 and $(N^2\text{-}N)/2$. The best case scenario, N-1 is possible when one module interacts with just one other module. Worst case scenario $(N^2\text{-}N)/2$ occurs when all modules interacts with all other modules. [42]

Weak coupling in the system is achieved when interactions between modules is limited to data change only. Stronger coupling between modules is found when modules exchange data structures. Shared data memory is one of the strongest forms of coupling. When one or more module uses the same data memory it is difficult to separate the contribution of each module. [42]

2.4.7 Software Element Encapsulation

Nested encapsulation is a design methodology where unit is an independent and encapsulated element with defined algorithm [43]. Unit shall only manage its own operation and relations of its components. It means that encapsulated units can be universally used and any unwelcome side effects will not arise. Control organization is explained in Figure 24. System is composed of four units (U1-U4) and total of six modules. Unit's U1 and U2 behavior is strongly related i.e. they are strongly coupled. Module U12 is designed to control interactions between U12. UX then manages interactions between U3, U4 and U12. Nested encapsulation or organizational approach to control organization increases software re-use possibilities, because for example U3 can be used somewhere else in the program since its behavior is not directly dependent from other units. [42]

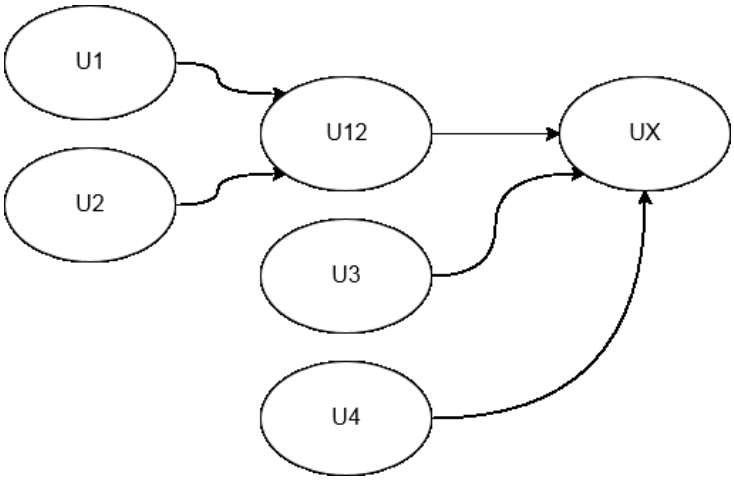

Figure 24 Control organization (adapted Bonfatti 1997)

2.4.8 Diagrams for Program Design

Programming methodology is a framework that is used to structure and plan the system. Selecting methodology is dependent of requisite operation, specifics and overall nature of the system. Different methodologies are suitable for different kinds of systems. In following three chapters are presented the industry standard methodologies for PLC programming. [42]

2.4.8.1 State Diagram

State diagram is graphic representation of an algorithm. So called *finite state automaton* is easy to handle since it can be described formally in non-procedure way. A finite state automaton is entirely defined by the finite set of states, the set of events, the initial state, the set of final states and the set of transition rules. Example in this chapter is adapted from IEC 1131-3 Programming methodology [42]

Notation “St” presents the infinite set of states. Each state is significant situation in the system that has been described with state diagram. Identification of meaningful states is crucial for reasonable partition of the system control. For example on/off valve could have three meaningful states, open, closed or moving. Events are the triggers, which cause system to change its state. The set events are represented as “Ev”. Every state of system is caused by one or more events. The initial state “In” is part of set “St”. It is the point where automaton execution starts. The set of final states is a subset within the set “St”. The set of transaction rules ”Tr” contains as many transaction rules that there are events in the system.

System can only be in one of its states. For example on/off valve cannot be open and closed at the same time. Before drawing state diagram it is necessary to recognize all the meaningful states in the system. For the valve there are four possible states, initial state,

open state, closed state and moving state. Members of the set "S" (equation 8) are all the possible states of the system.

$$St = \{In, Op, Cl, Mo\}, \tag{8}$$

Defining final states depends on the system. In case of on/off valve final states are open state and closed state. "Fi" is defined as subset of "St" (equation 9).

$$Fi \subseteq St, \ = \{Op, Cl\} \subseteq \{In, Op, Cl, Mo\}, \tag{9}$$

Events are causing the transitions between states. There will be six transitions and therefore six events. From initial state there are two transitions, to open state (a) and to close state (b). Other four transitions are from open to moving state (c), moving to close state (d), close to moving state (e) and moving to open state (f). Set of events "Ev" (equation 10) is then.

$$Ev = \{a, b, c, d, e, f\}, \tag{10}$$

Transition rules (functions) are defined as <current_state,event,reached_state>. The set of events can be converted into set of transition rules (equation 11).

$$Tr = \begin{Bmatrix} < I, a, O >, < I, b, C >, < O, c, M >, \\ < M, d, C >, < C, e, M >, < M, f, O > \end{Bmatrix}, \tag{11}$$

After definition of all related sets of the system, a state diagram can be drawn (Figure 25). Automaton is deterministic because one event can only cause one state to become active e.g. transition "c" causes only state "Mo" to become active.

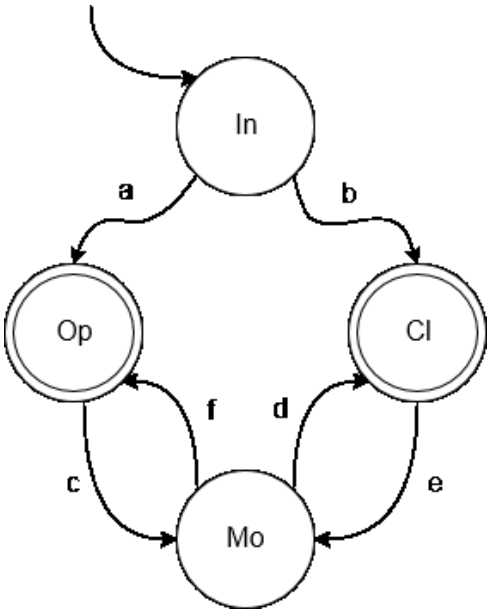

Figure 25 State diagram of on/off valve

2.4.8.2 Sequential control diagram

Many manufacturing processes are sequential, meaning that they are composed of series of steps, which are performed consecutively. Automated assembly line, metal molding and packaging are examples of sequential processes. [44] Sequential control has characteristics of a state diagram. Main difference between state and sequential control diagram is presence of actions in sequential control diagram [42]

Sequential control starts with initial step and it is activated by the turn on of the control system. Sequential control can feature as many steps as is necessary to control the process. Triggering conditions controls transition between two steps. In Figure 26 is presented linear sequence of steps. Start step is deactivated and Step 1 is activated when condition associated to T1 becomes true. Final transition T3 stops the sequence and system cannot leave the final state without external event. This is called “one-shot” behavior. Sequence is started by an explicit command whenever needed, meaning that behavior of the system is kept under operator control. [42] For example washing machines works like that. Sequence is started by user and new sequence cannot be started without new command.

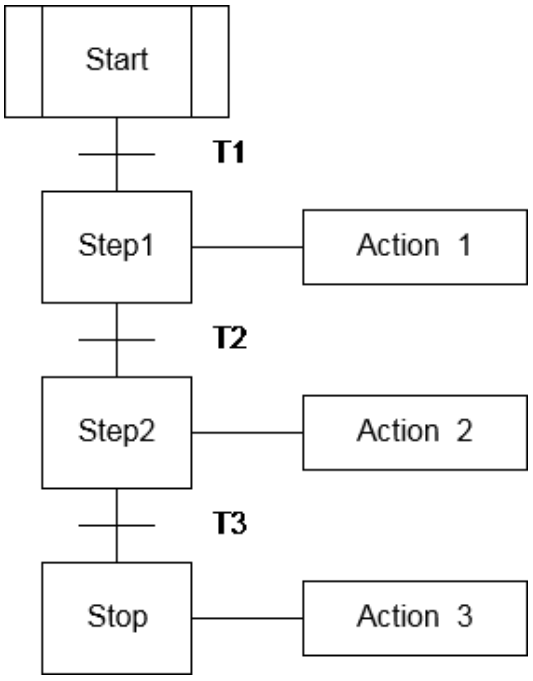

Figure 26 Linear sequence

More common situation than “one-shot” is cyclic sequence of steps (Figure 27). In this case external event starts the sequence but it will never stop until another external event. This is desired behavior in continuous processes or when operation of sequence is part of a higher level step in other sequence. The action is stopped when upper level step is inactivated. [42]

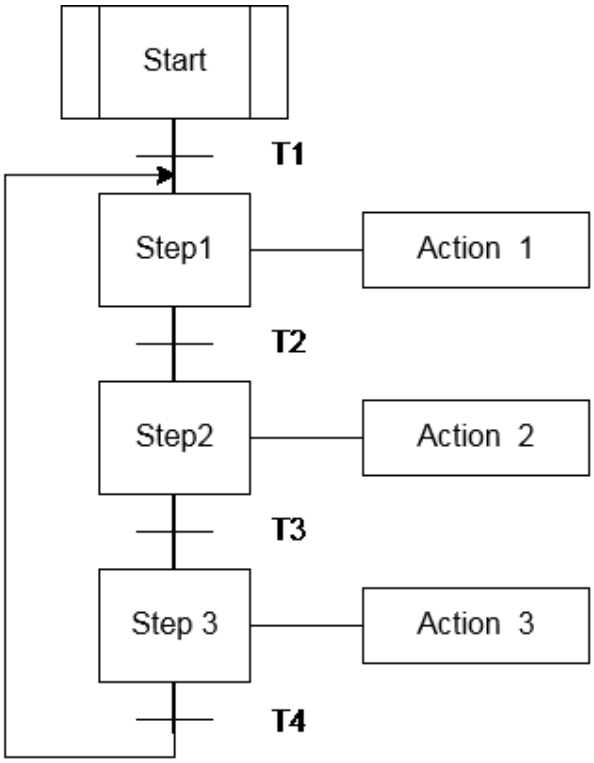

Figure 27 Cyclic sequence

In Figure 28 is presented the same cyclic sequence as in Figure 28 but with addition of controlled exit condition. Placing divergence in various points of diagram it is possible to take different paths of controlling the process. Sequence is endless loop until transition T5 (or external event) terminates the execution. [42]

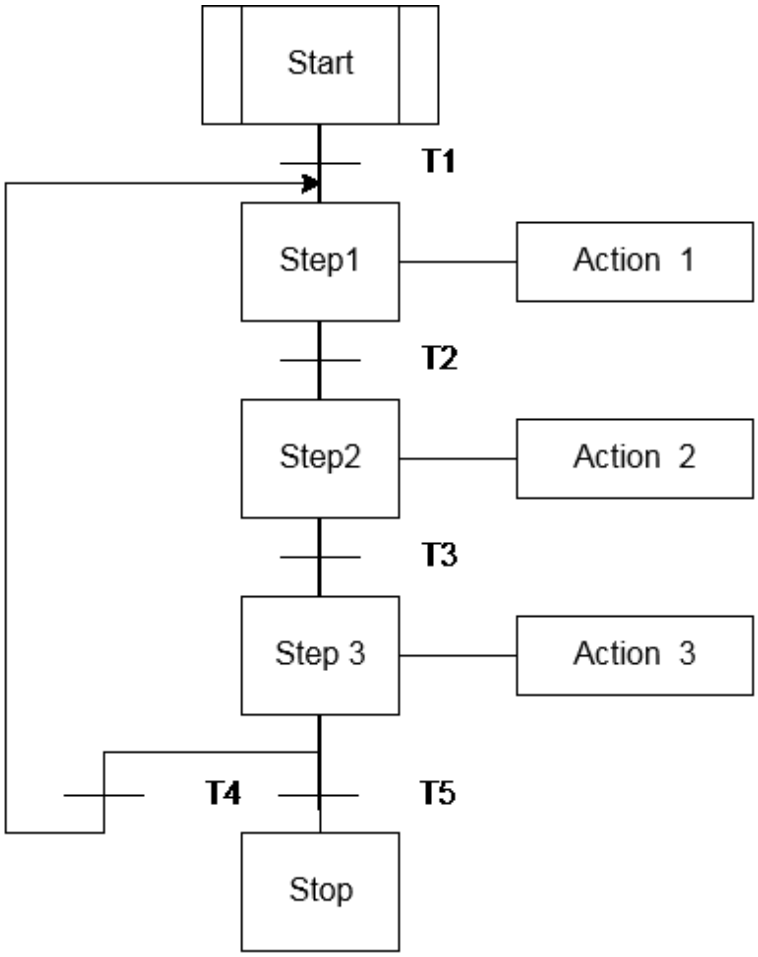

Figure 28 Cyclic sequence with divergence

2.4.8.3 Flowchart

Flowchart is a graphical representation of the operations involved in a process or system. Flowchart consists of symbols and lines connecting the symbols together. Symbols represents particular operations in a process or a system, and lines indicates sequence of operations. [46] Basic flowchart symbols are presented in Figure 29. Different shapes indicate the various kinds of activity described by the diagram. Usage of shapes can be highly formalized where every shape has a specific meaning. But it is acceptable to use simple boxes with text if process or system can be clearly represented. [46]

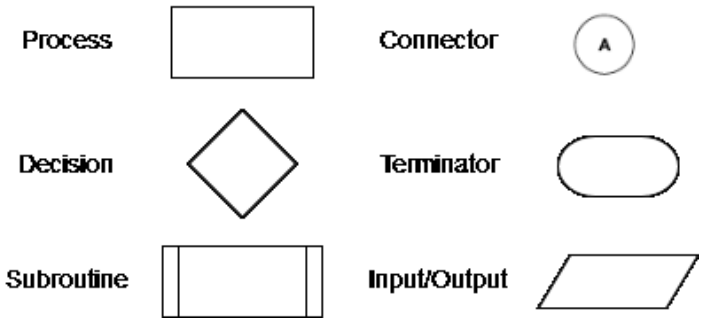

Figure 29 Basic flowchart symbols

Figure 30 presents simple flowchart of money withdrawal from ATM machine. Process starts from Terminator "Start". Then user inputs amount of cash to withdraw. Machine then checks if user has enough money and whether displays error message or deducts money from users account. Process is terminated after money has been dispensed.

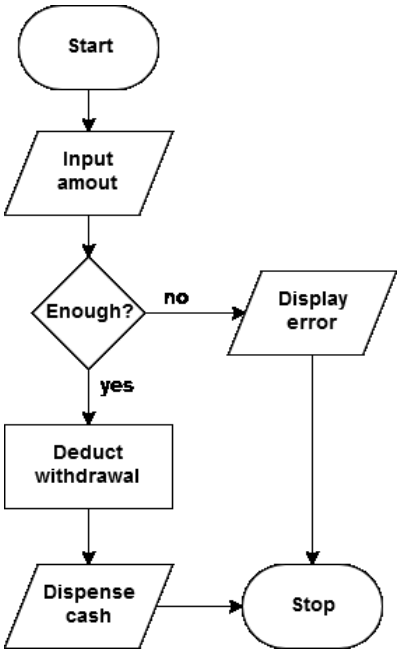

Figure 30 Flowchart of money withdrawal from ATM (adapted BCS Glossary of Computing 2013)

2.5 Event Driven Systems

This chapter provides a brief review of IEC 61499 standard which was published in 2005 by IEC as an international standard in three parts. Part 1 - Architecture covers the main elements of the reference model, Part 2 - Requirements for Software tool defines software tool requirements and needs and Part 3 - Rules for Compliance Profiles is for defining rules and guidelines for the specification of compliance profiles used in different domains and applications. [47]

The IEC 61499 defines two different input/output types for Function Blocks (Figure 31): events and data. FB execution is triggered by events after the input data is available. After event has occurred the FB's algorithms are executed and the output data is updated and an output event is generated. The Output event usually triggers execution of another FB. [47]

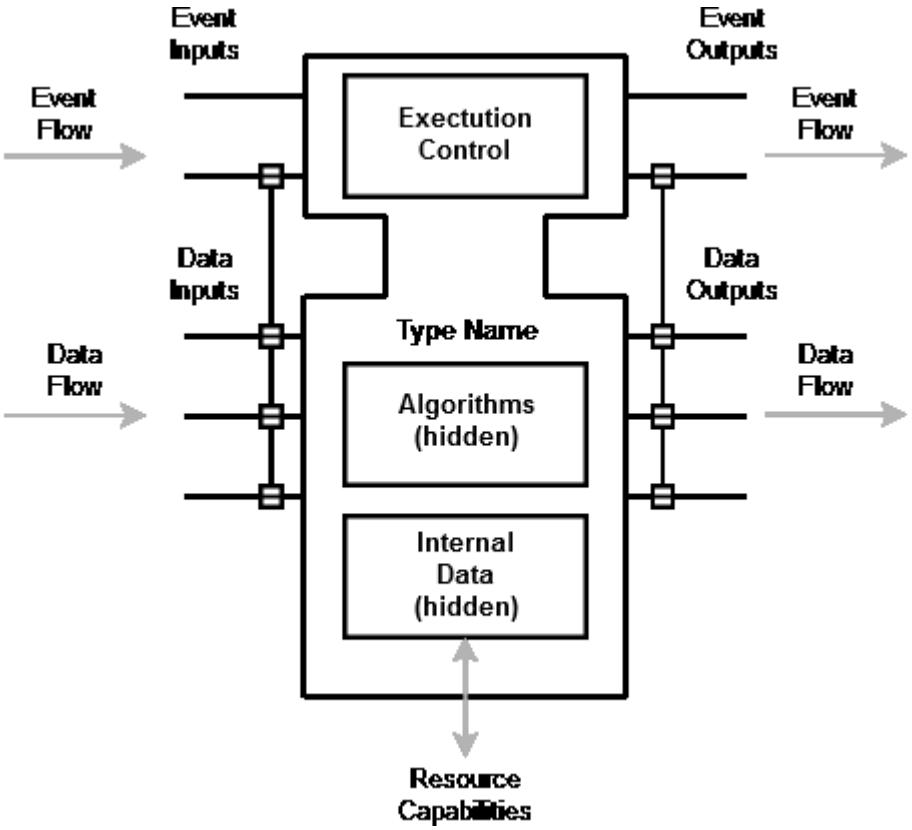

Figure 31 IEC 61499 FB Model (adapted Distributed Control Applications 2016)

Function Block definition of IEC 61499 is based on IEC 61131-3 but it is extended with an event interface [47]. The standard defines three FB types: The basic FB for algorithm encapsulation, Service Interface FB as interface between non-IEC 61499 elements and composite FB for functional aggregation. The basic FB type can be used to construct higher level FBs. Algorithms written in IEC 61131-3 or higher level (Java, C, C++) languages can be encapsulated and they are executed by so called Execution Control Charts (ECC) which are event-driven state machines (Figure 32). [47]

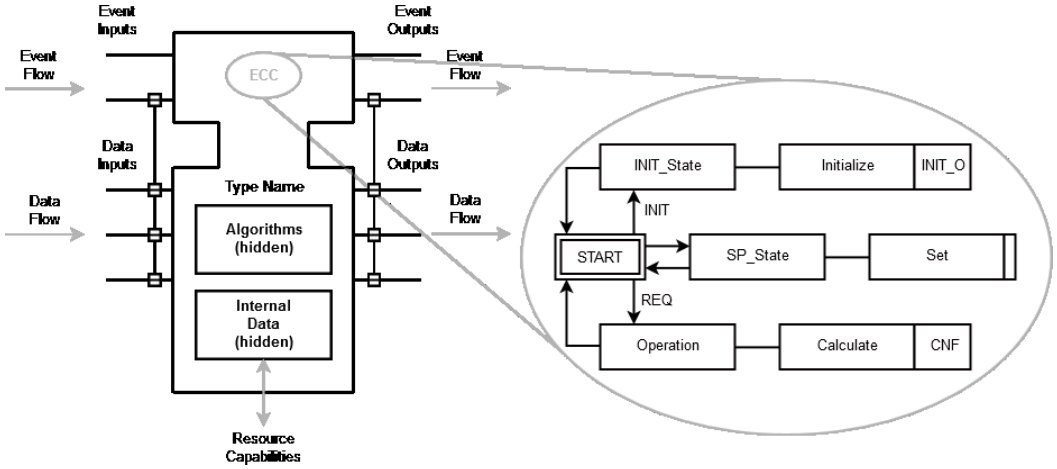

Figure 32 FB Model and Execution Control Chart (adapted Distributed Control Applications 2016)

3. COMMUNICATION MECHANISM

This chapter presents the design process and methodology selection of implementation for this work based on chapter 2. Design process is driven by end user needs. Best case scenario is that the end user will not even notice the communication solution running in the back. Any lag or transmission delays affect the user's experience. Properly thought out software implementation should reduce and minimize any delays or lag that are affected by hardware limitations.

In the highest level of the scope are the input and the output of the system. Recognizing the user needs is the first driver of the design process. By simplifying enough, user needs are capability to give input parameters and receive output parameters from the system. Therefore it is crucial to ensure sufficient and systematic data transfer between all stations connected in the network.

Fulfilling the end user needs is not the only requirement of functional communication system. It should be easy to use for developer of the automation system. If it is not possible to understand the operation of the communication system in reasonable time, it will be hard to implement any remote monitoring for automation system.

The following design parameters and drivers are derived from problem and objectives in chapter 1

- Maximum data transmission rate without affecting reliability
- Data acquiring where and when its needed
- Understandability of program execution
- Easy commissioning and configuration

3.1 System Principle

The system itself is quite simple. There will be one master station that initiates communication to slave stations. Transferring data between master and slave station is more complex as some data will be treated differently from another. Cyclic data like alarms and measurement values need to be transferred to Master station and from there to SCADA frequently. For acyclic data like control signals it is adequate to transfer only when they are requested by operator of the system or Master station. Also there will be need to organize cyclic data transfer. In large networks round times can be up to one minute. If operator needs to have more frequent updates from selected Slave station, Master station must be able to read one or more stations in the network more often than others. That will lead to new problem with those other stations. If too many stations jump the queue, Master station will not be able to read other Slaves often enough.

3.2 Design Process

The designing process of the system follows the V-model. It is a variation of waterfall model presented by Paul Rook in 1980s. [48] The general idea of V-model is that it has layers of system development. It means that every step has planning and testing phase. Requirements is the highest level of the model in the planning side. Its corresponding testing layer is Acceptance Testing. It means that before accepting the system it must fulfill requirements defined in the planning stage. Every lower level planning phase has its own testing phase where only the corresponding features of the system are tested (Figure 33).

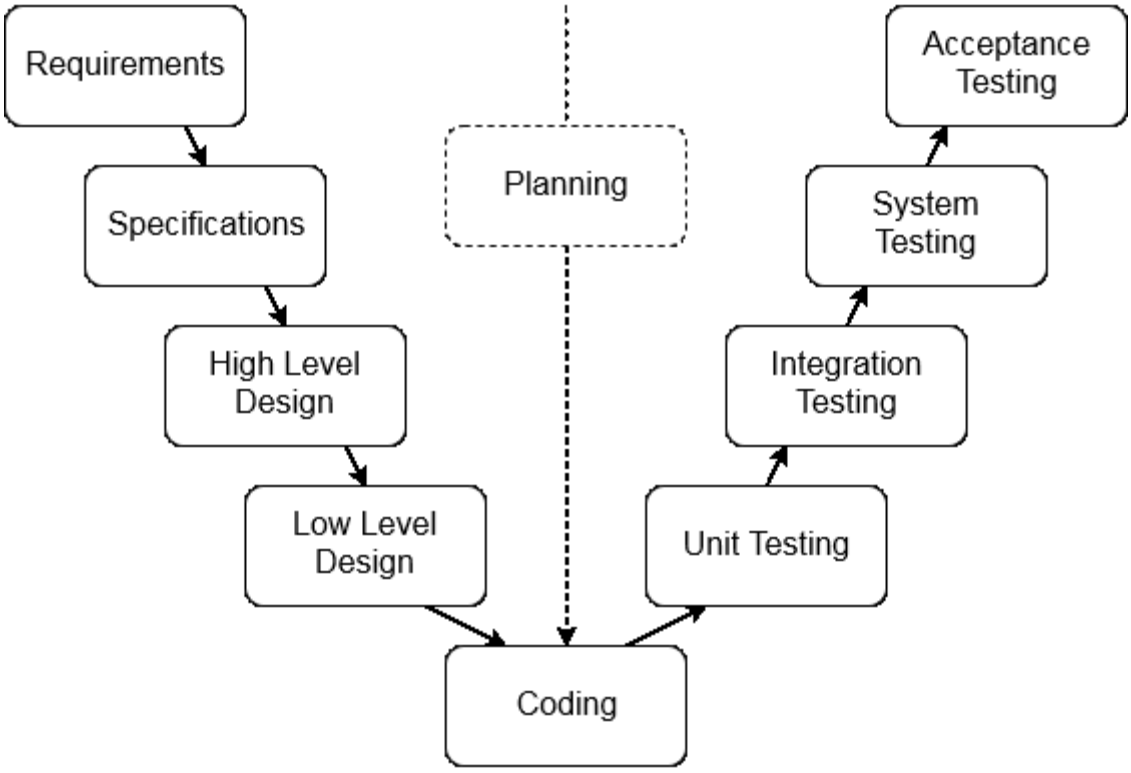

Figure 33 V-Model for System Designing adapted from Isaias & Issa 2015

3.2.1 Requirements and Analysis of the Use Cases

Previous chapter presented the V-Model for system designing. First step of system design is requirements definition. Use case diagram was created to illustrate actors affecting the system (Figure 34).

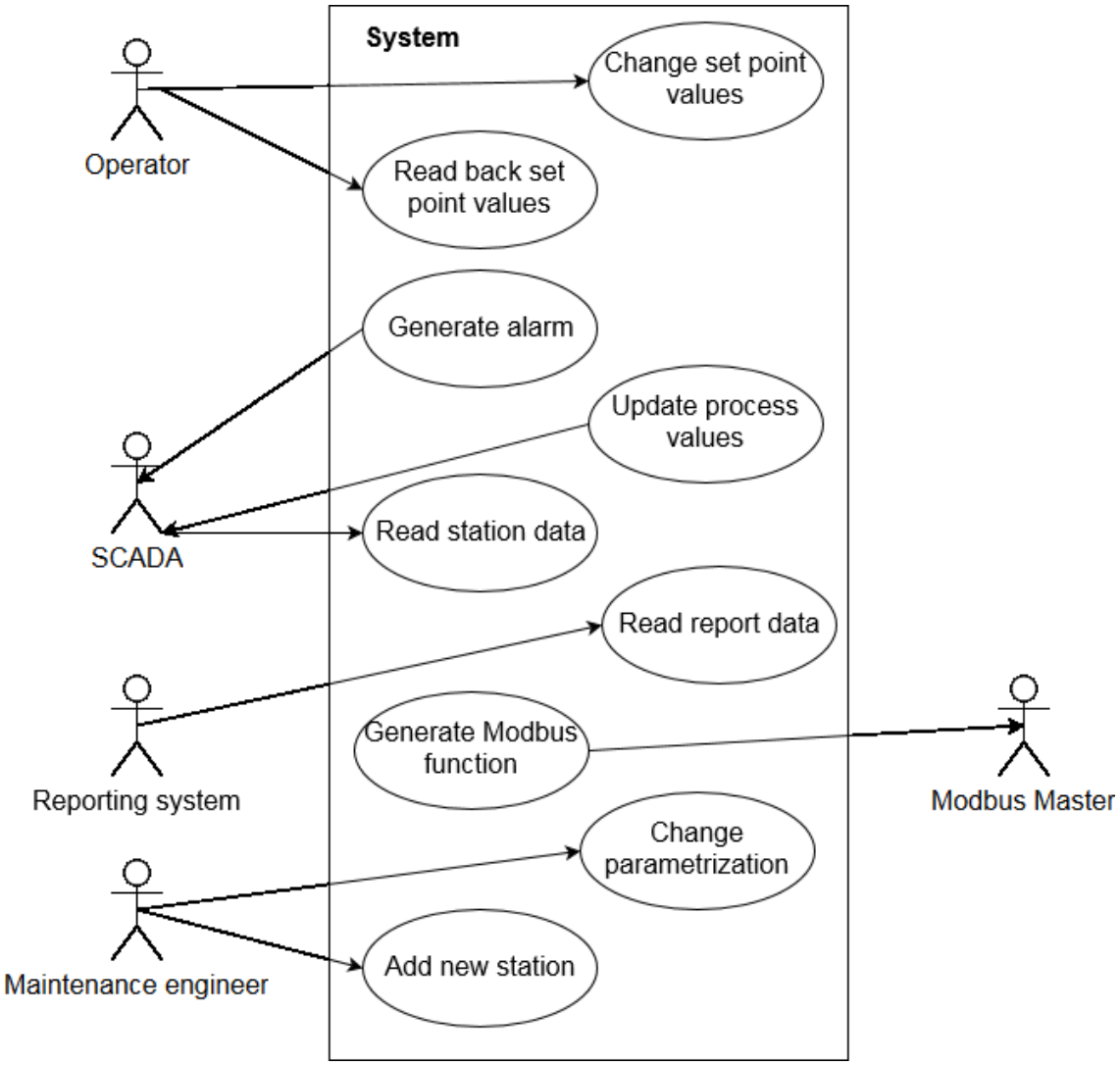

Figure 34 UML Use Case Diagram of the Communication System

Every actor in context of the systems has their own requirements for the system. Operator is the end user of the system. As mentioned in introduction to chapter 3 the operator of the automation system has to be able to change the set point values of remote station. It is also necessary to read back the values which were entered to ensure sure that transmitted values were received by remote station. Operator is not the only actor of the communication system. One important actor is SCADA. Operator will not be affecting the system directly but trough SCADA. Higher level "Supervisory Control and Data Acquisition" system collects data from e.g. PLCs and shows desired data trough HMI to end user. Third actor in use case diagram is higher level reporting system. Reporting data is being generated in remote plants. The system's task is to collect the report data for higher level reporting system. Last actor in left side is Maintenance engineer who can parametrize the system and add new Slave station to network. Modbus Master is the actor between the communication system and remote plants. The system's task is converting the parametrization data to suitable format for Modbus Master. Behavior of Modbus data transmission is explained in chapter 2.2.2.

3.2.2 Specification Planning

The specifications for the system are presented in Figure 34 inside the “System” box. Every oval shape is an individual requirement for the system. Importance of the actors is not equal. End users or operator is the most important actor of the system. Therefore the most important specification of the system is end user experience. Operator must be able to change remote plants operation when needed. Also changing values to remote plant should take place as soon as possible. Operators will not want to wait long for e.g. transmission of control command to remote control system. Next in descending order of importance is SCADA system. SCADA uses the data from remote plants to show operator the current state of stations connected to network. As mentioned in chapter 3.1 data acquisition from distant plants should be systematic. If data from station is not acquired frequently enough it will affect the end user experience thus e.g. alarm signals are delayed and trends that SCADA generates becomes stepped and will not give right picture what has happened in remote plants processes.

Reporting data is not acquired all the time. It is adequate to collect reporting data for example every half an hour. However reporting data is very important to end user, so gathering the data is crucial and the communication system should be able to collect the data when higher level reporting system requires. Maintenance engineer is substantial actor of the system and when maintenance engineer needs to change the systems functionality e.g. change some stations parametrization it should not have affect to other actors of the system.

Following specifications were derived from use case and requirement analysis.

- Priority to end user experience
- Sufficient data acquiring for SCADA and higher level reporting system
- Maintenance procedures will not affect first two specifications.

3.2.3 High Level Design

High level design phase is driven by specifications presented in last chapter. Recognition the nature of the system is base for programming methodology selection. In chapter 3.2.3.1 there is analysis of qualities of communication system. Decisions made in this phase also affects to code internal quality discussed in chapter 2.4. The software should be designed to be modular thus units of software are loosely coupled and coupling is made explicitly.

3.2.3.1 Methodology Selection

Start point for methodology selection was to divide data to be transferred into different types (Table 4). Remote station is usually independent control system. Measurements,

motor controls and alarm triggering are managed by standalone control system like PLC. One of communication system's tasks is to gather data produced by remote control systems. This kind of data needs to be acquired frequently i.e. cyclically. First of the defined data types is cyclic read data. Although remote control systems are independent they can require data from higher level systems. This data will also need to be transferred to remote stations frequently. After cyclic read the next defined data type is cyclic write data. Third data type with cyclic nature is report data, generated by remote plants. Report data is not changing continuously like measuring values so it is sufficient to read report data only when remote plants have generated new data or by the request from the higher level reporting system.

It is necessary for operators to sometimes change the behavior of remote plants. Operator might need to decrease or increase the output of e.g. remote pumping station. Then it is needed to send control commands and set point values to remote stations. Therefore acyclic data types need to be defined. First acyclic data type is acyclic write. Set point values defined by operators should be transferred to remote plants immediately and only when needed. Acyclic read is defined for reading back set point values of the system. Remote plants can be equipped with local HMI. Therefore it is substantial to read set point values to SCADA system, because operators might not know if the values have been changed locally.

Table 4 Data type comparison

Datatype	Cyclic	Acyclic	Continuous
Cyclic read	X		X
Cyclic write	X		X
Report	X		
Acyclic write		X	
Acyclic read		X	

All five different data type transfer modes can be seen as the significant states of system (Table 5). More states could be defined by dividing these five defined states into smaller sub states. It is noteworthy that too many defined states could lead into very complex looking system and huge amount of transitions and therefore transition rules. On the other hand fewer states would not describe the system properly e.g. one state "communication" where all different modes are described to belong into the same state of the system. Different from example in chapter 2.4.8.1 the system will not have a final state because once communication is started it should never stop without external event. The following states were defined for system's state diagram.

Table 5 Significant states of communication system

Significant states
Init
Cyclic Reading
Cyclic Writing
Monitoring
Report Data Reading
Acyclic Writing
Acyclic Reading

Monitoring state is not part of communications between Master station and Slave stations. Its task is to keep track of communication performance of each station in the network. It is defined as state because monitoring can be done after communication system has executed its tasks for one cycle.

Different states of the system can be described as sets. All the states of the system are members of set "St" (equation 12).

$$St = \{In, CR, CW, M, R, AW, AR\}, \tag{12}$$

Sub sets of set "St" are the set of initial states "In", the set of cyclic states "CS" and the set of acyclic states "AS". The set of I has only one member because system has only one initial state. Members of cyclic state (equation 13) are cyclic reading (CR), cyclic writing (CW), monitoring (M) and report reading (R) and members of acyclic states (equation 14) are acyclic writing (AW) and acyclic writing (AW).

$$CS \subseteq St, \ = \ \{CR, CW, M, R\} \subseteq \{In, CR, CW, M, R, AW, AR\}, \tag{13}$$

$$AS \subseteq St, \ = \ \{AW, AR, \} \subseteq \{In, CR, CW, M, R, AW, AR\}, \tag{14}$$

Division of the states into sub sets facilitates the definition of transitions between states. First can be defined transitions between cyclic states. The system is a loop without exit condition, meaning that every cyclic state should have event that activates it and event that deactivates it. Order of states is arbitrary. Cyclic reading can be done after cyclic writing and vice versa. Order definition of the states is based on the communication mode definition in Table 4. To create loop the four transitions between states must be defined. Transitions (equation 15) are described as set "EC".

$$\boldsymbol{EC} = \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}\}, \tag{15}$$

After definition of cyclic states and transition between states, first state diagram can be created (Figure 35). Initialization of the system occurs when system is powered up and new initialization should not take place without external event such as reboot.

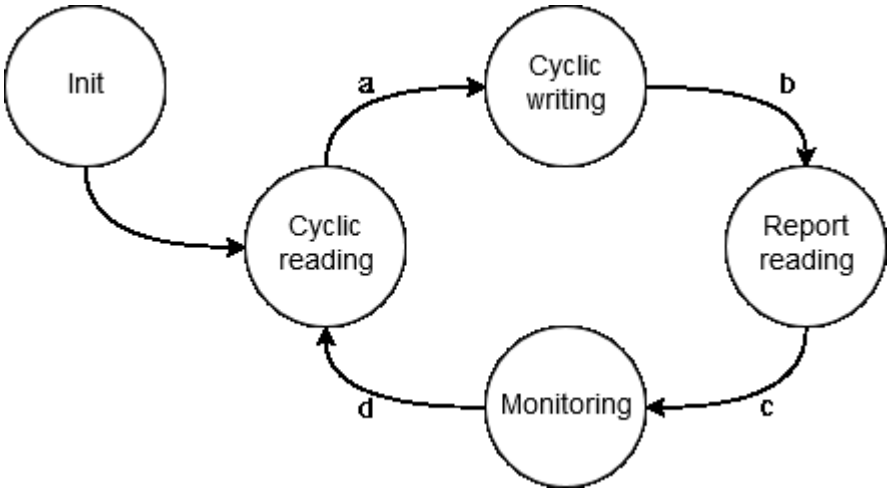

Figure 35 UML State Diagram of the Cyclic States of the System

Without Acyclic states, the diagram represents systems behavior when user is not affecting to the system. System is looping from one state to another, unless external higher level event causes it to stop. Transition to acyclic states is always consequence of end user actions. Addition of acyclic states to diagram clarifies the problem of systems overall complexity. Amount of transitions increases from four to 20. In Figure 36 system state diagram is divided into two different figures to make it more readable.

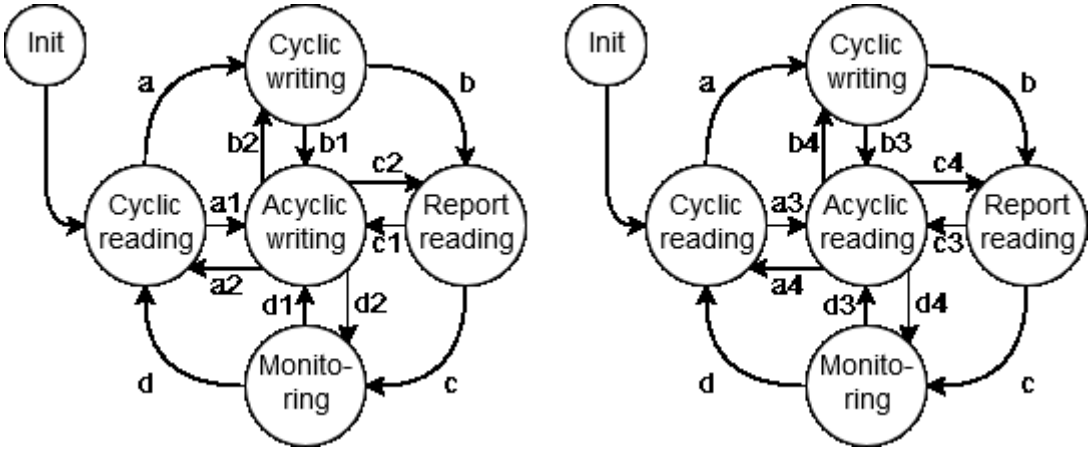

Figure 36 UML State Diagram of all States of the System

$$EW = \{a1 \dots a4, b1 \dots b4, c1 \dots c4, d1 \dots d4\}, \quad (15)$$

The set of acyclic event notated as “EW” contains 16 transitions (equation 15). System implemented with events from state to another leads to high degree of coupling. Acyclic states are related to all cyclic states of the system. Acyclic state needs to know which state activated it in order to continue cyclic loop where it stopped, after acyclic execution has ended. Cyclic execution can be distorted if acyclic state is triggered consequently and wrong state is activated after acyclic execution.

Figure 37 is other visual representation of systems behavior using sequential control chart. It features the same 20 transitions of the system with additional “c24” for skipping Report reading (Step 3) when needed. High degree of coupling cannot be resolved using only steps and triggering conditions between them. Higher level module for state or step triggering is needed to keep software re-usable and easy to understand.

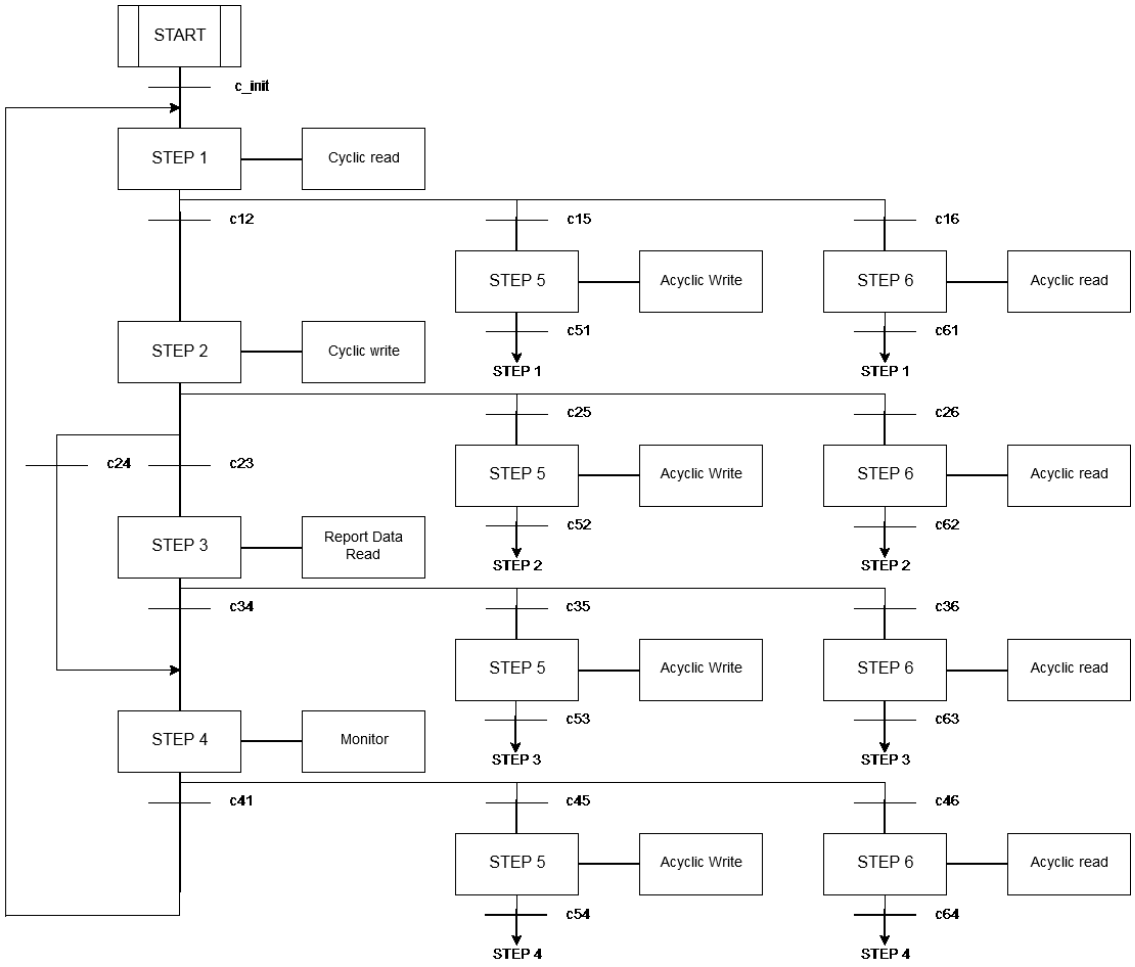

Figure 37 Sequence diagram of the communication system

3.2.3.2 Software Modularity and Coupling

Principle for software modularization, coupling and encapsulation is presented in chapter 2.4. Complex systems should be divided into smaller units responsible for small entities of the whole program. For this instance at least seven modules are needed, one module for each state of the system and additional module for event triggering. Basic principle of the system is that every state module is either allowed or forbidden to execute its task. When user triggers acyclic state others states are asked to stop their execution. Event triggering module is responsible for allowing and preventing states to become active. Then adding or removing steps from program can be done just modifying the event triggering module, because it is only module in the system that “knows” every other module.

Control organization of the system is presented in Figure 38. Usage of event trigger module decreases connections or couplings between modules from twenty to only six.

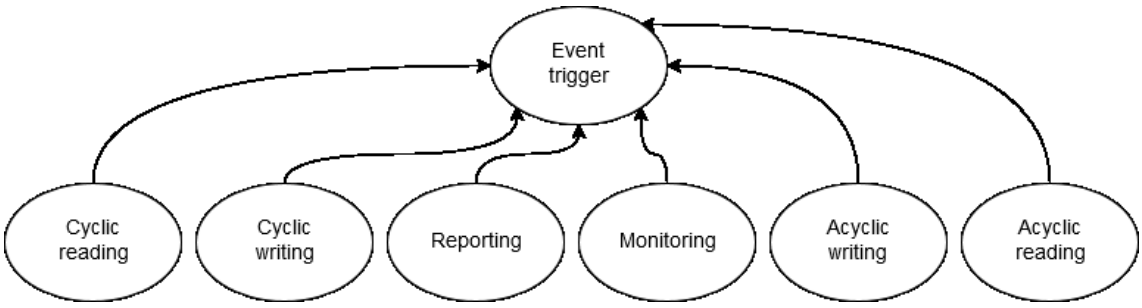

Figure 38 Diagram of Control Organization of the Communication System

Event trigger module is not responsible for stepping trough loop of cyclic states. Every module has its own unique index and consequent step parameter which defines the order of execution. Current step of execution is shared read/write memory in the system. This increases degree of coupling, but another module for step control is not needed. Step control between modules is presented in chapter 3.2.4

3.2.4 Low Level Design

First part of low level design is to determine step control. Value of the step variable in the system is stored in memory where all modules have read and write access. IEC 1131-3 describes shared data memory as one of the strongest and most dangerous form of coupling. When different modules access the same memory area it is difficult to distinguish contribution of each module. [42] In this case every module is allowed to write Step variable only when they are active. Step number 1 activates module with assigned name 1. After module number 1 has executed its tasks it writes new Step number based on input parameter (Figure 39).

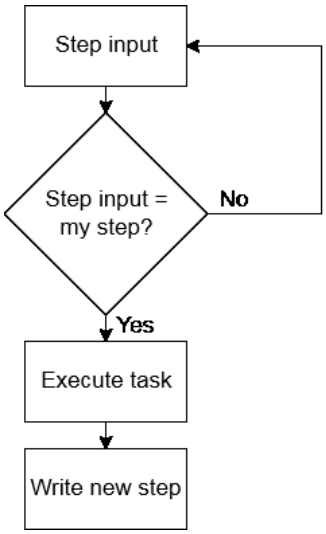

Figure 39 UML Activity Diagram of Step Control

Step control can also be done with external Step module that monitors which of the modules in the system is active and when it becomes inactive and writes Step number based on that. Usage of Step module leads to problem with Event trigger module, meaning that if Event trigger disables e.g. Cyclic modules it also has an effect on Step module. Event trigger would need to notify Step module of which state is needed to become active and Step module then writes a new step value. After that the activated step can execute its task. Figure 40 and Figure 41 illustrates event flow with and without external Step module.

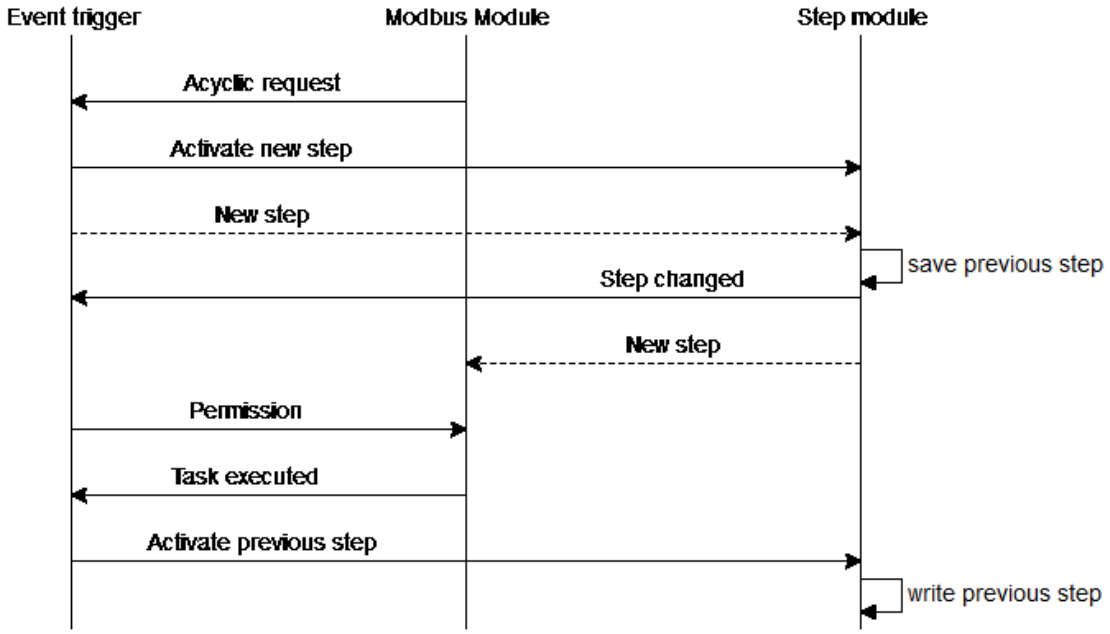

Figure 40 UML Sequence Diagram of Event Trace With Step Module

When acyclic request occurs, Modbus module e.g. Acyclic write notifies Event trigger, which then stops execution of other modules. Then Event trigger requests Step module to change step. After new step is activated, Event trigger gives Modbus module a permission to execute the task. After execution Modbus module notifies Event trigger which requests Step module to write previous step back. In this case Step module does not “know” i.e. is not connected to any other module than Event trigger. One possibility is that Step module does not “know” Event trigger and events to Step module comes from Modbus module. This will require more deduction from Modbus module to know when it is allowed to execute its task. This will increase amount of events even more because there will be signals also between Modbus and Step module.

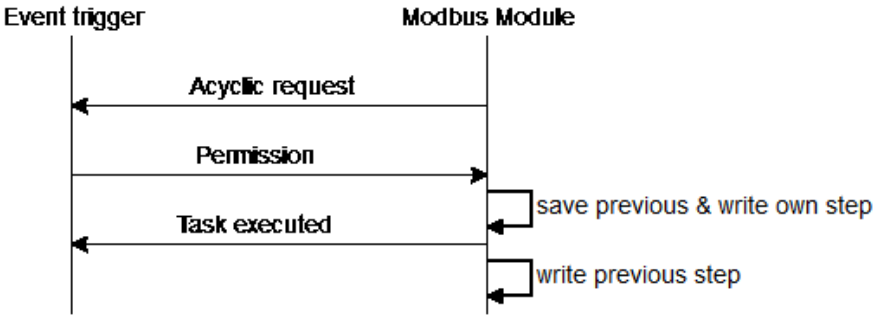

Figure 41 UML Sequence Diagram of Event Trace without Step Module

Without Step module the event flow is reduced. Just three signals between modules are enough to execute e.g. Acyclic request. Even though shared data memory between modules is strong coupling it can reduce complexity of program execution. Without Step module programmer should only ensure that all modules controlling step variable are parametrized correctly.

Step variable is also used to access data in arrays. All modules apart from Monitoring and Event triggering are for generating Modbus functions based on parametrization data. Generated Modbus function is stored in multiple arrays. One array for Slave address, one for Modbus register address and so on. Arrays are data types that can be accessed with an index. Every Modbus module has its own memory area where it stores generated function. Depending on step number, the data from the array is transferred to input of Modbus Master Function Block (Figure 42).

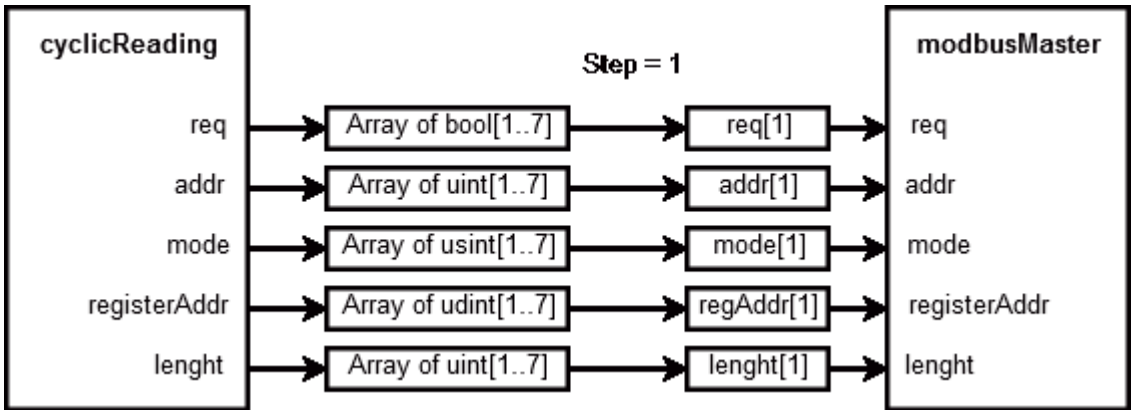

Figure 42 Data array access method

3.2.4.1 Data Blocks

Data blocks are used to access data in PLC's memory. Data blocks can be divided into two main groups: Global and Instance data blocks. Global data blocks store data that can be used by all Functions and Function Blocks in the program. The call of a function is referred to as an Instance Data Block. Instance Data Block is always assigned to Function

Block and it contains declarations of local variables and constants that are used within the block. (Siemens help: Global DB and Instance DB)

Communication solution uses several data blocks for reading and writing data. For one module point of view there are three types of data blocks: read, read/write and write. Read type DB is for storing parameters that module is not allowed to change such as Network Parameters (Figure 44). Read/write type contains data that module can either read or write depending on task it is executing. For example module reads previous history data and modifies data based on last transmission success.

Previous example Figure 42 uses data block for storing functions generated by Cyclic Reading module. DB is write type for cyclic read module and read type for Modbus Master Function Block. Depending on value of Step variable different data is transferred from array to MB Master Function Block's input.

3.2.4.2 Cyclic Slave Reading and Writing Function Blocks

As mentioned in chapter 3.2.3.2 program is divided into different modules. Cyclic Slave Reading module is responsible for reading cyclically all Slave devices in the network. It is independent from other modules of the program, meaning that the whole communication solution could be only Cyclic Reading module, if other features like cyclic writing are not needed. UML Use case diagram (Figure 44) presents actors and use cases of cyclic slave reading. Diagram defines five significant use cases: Read Process Values, Update Process Values, Read RSSI, Read Station On Display and Generate Modbus Function.

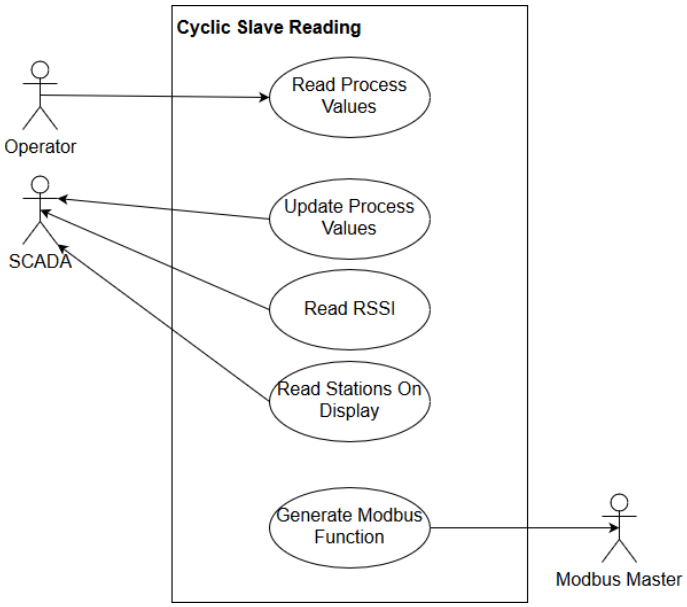

Figure 43 UML Use Case Diagram of Cyclic Slave Reading

Read Process Values Use case fulfills operator's needs for the system. Update Process Values, Read RSSI (Received Signal Strength Indicator) and Read Stations On Display

Use cases are determined for SCADA system. Modbus Master Function Block inside PLC is actor involved in Generate Modbus Function of the module.

UML Class diagram (Figure 44) defines interactions between different modules, Function Blocks and Data Blocks in the program. Cyclic read has internal attributes such as internal step of execution and methods such as “readSlave” and “writeData”. “ExecutionParameters”, “networkParameters” and “slaveData” are data blocks which Cyclic Read module uses to generate Modbus function and reading values from Modbus Buffer DB. Modbus Master interacts with one or more Modbus Slave FBs based on parameters generated by Cyclic Read module. Event trigger module can stop execution of cyclic read module if interruption is being requested.

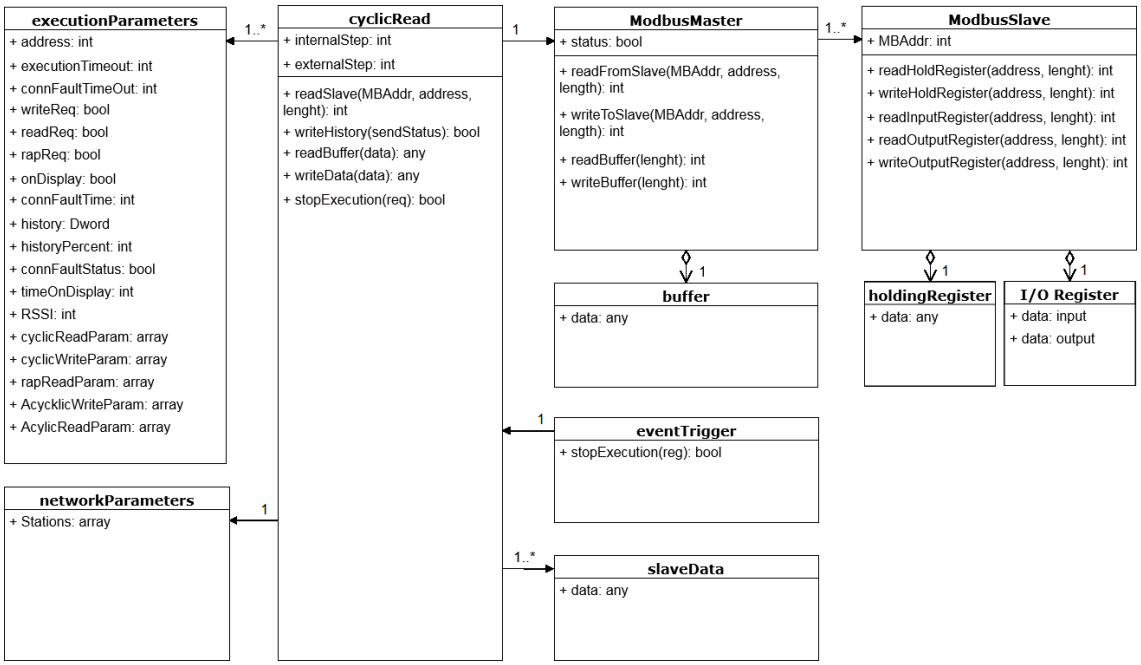

Figure 44 UML Class diagram of Cyclic Slave Reading Module

UML sequence diagram (Figure 45) presents internal execution of one cycle of the module. After initialization the network parameters are read to modules internal memory (instance). The Network parameters contain a list of all slave stations in the network. Module is executed cyclically until all slave stations in the network have been read. After reading Cyclic Read module writes new Step variable and next module starts its cycle.

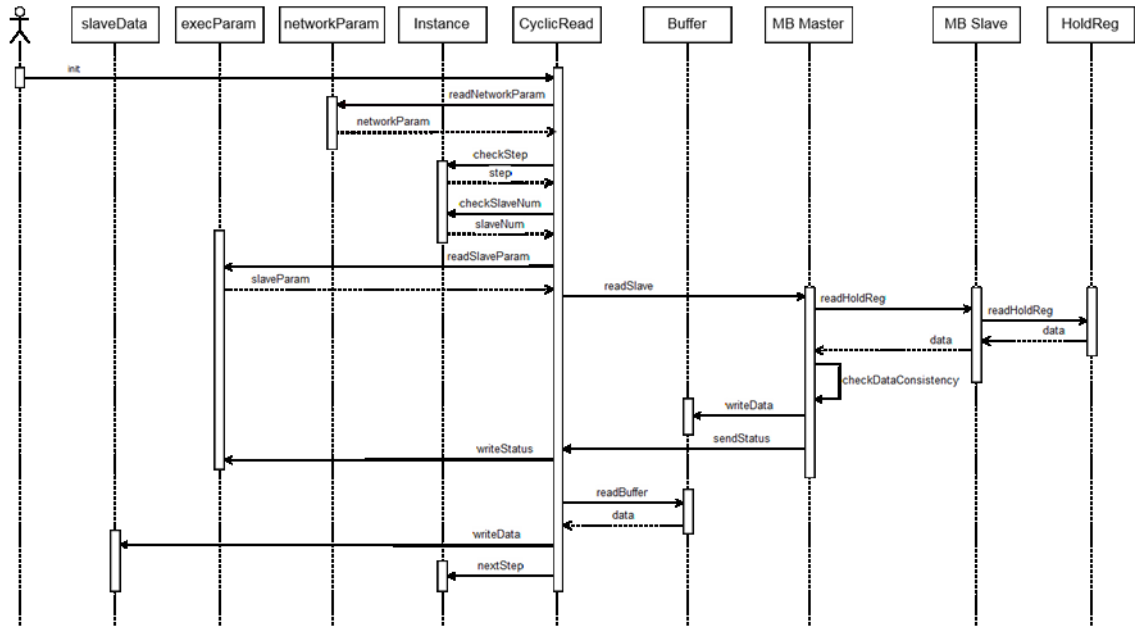

Figure 45 UML Sequence Diagram of Cyclic Slave Reading Module

Functionality of the Acyclic Read module is similar to Cyclic Read module except the acyclic module generates function for Modbus Master that initiates writing of slave device. Use Case diagram of cyclic write has only two significant use cases and two actors (Figure 46). SCADA updates values such as time of day to Slave device. Generate Modbus Function is for collecting right data from Execution Parameter DBs and creating right function for Modbus Master.

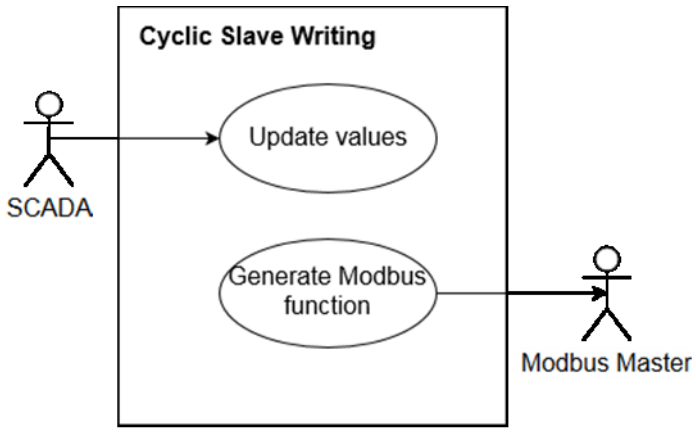

Figure 46 UML Use Case diagram of Cyclic Slave Writing

3.2.4.3 Acyclic Slave Reading and Writing

It is not necessary to read all the data from stations cyclically. Amount of data read cyclically should be minimized because reading large amount of data from remote stations slows down round time of communication system. Acyclic reading module is activated only when needed. For example operator selects certain station to HMI display and the data can be read from station to be shown to operator. When station is no longer on display acyclic reading can be stopped. Acyclic reading of the remote station is required when operator sends new set point values to the station. Data transmission to remote stations is not 100% sure and sometimes sent data is not received by remote unit. Therefore data that is sent to station should be read back to make sure that sending of data was successful. Again reading back sent data is not necessary to be continuous. Read back is initiated only after the sending to Slave device was successful.

UML Use case diagram (Figure 47) shows actors and use cases of acyclic slave reading. First use case is reading back transmitted values because operator of the system requires confirmation of successful transmission. Another use case is updating the data to SCADA system. Values that were read back are stored into memory which SCADA system can read. Generate Modbus function use case is for creating suitable function for Modbus Master so it can read desired data from Slave device.

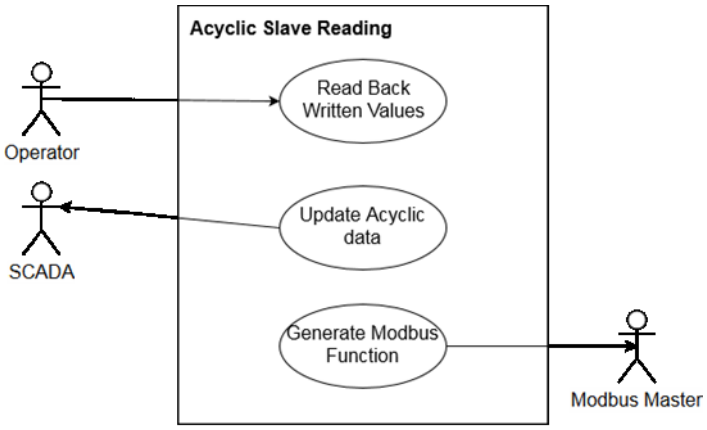

Figure 47 UML Use Case diagram of Acyclic Slave Reading

UML class diagram of acyclic slave reading (Figure 48) shows connections between Acyclic Read module, Data Blocks and Function Blocks. Connections to other blocks are basically same as in Cyclic Slave reading but in this case direction to event trigger module is reversed. Acyclic data transmission is initiated by operator of the automation system. When acyclic reading is requested the module notifies event trigger module, which then stops execution of cyclic modules. When acyclic reading is finished the trigger module is notified again that transmission is over and it allows cyclic modules to continue their normal operation.

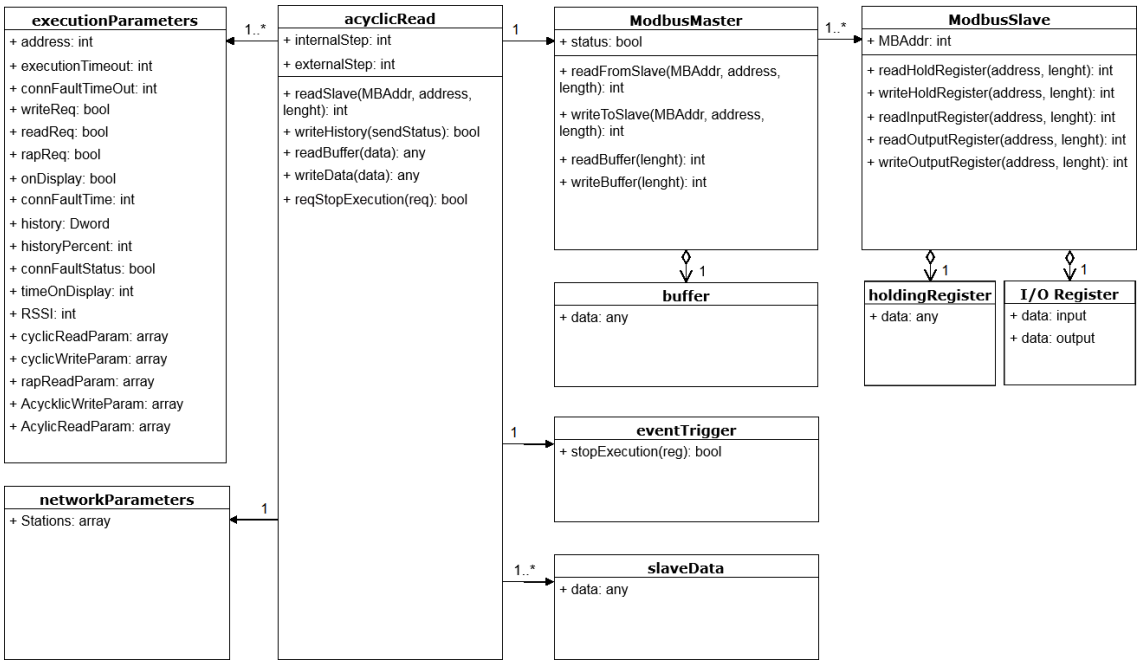

Figure 48 UML Class diagram of Acyclic Slave Reading Module

UML use case diagram (Figure 49) defines use cases and actors of acyclic slave reading. SCADA system will not transmit data to remote stations without operator request so it is not an actor in this use case diagram. Acyclic data is usually set point values to remote station. Operator changes desired values and requests writing to Slave station. Modbus function is then generated based on operator's action.

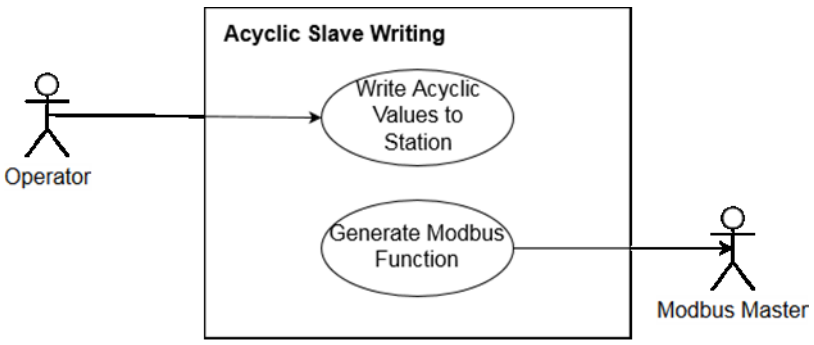

Figure 49 UML Use Case diagram of Acyclic Slave Writing

3.2.4.4 Communications Monitoring

Monitoring module is responsible for calculating round times of the communications system. Three different round times are calculated in the program. First calculated round time is cycle time of all cyclic states of the system presented in Figure 35. Round time calculation is started when module with index 1 is activated and stopped when module with index 4 is deactivated. Every time the acyclic state is activated it will affect the

calculated round time because execution of current cyclic state is interrupted and continued after acyclic module has executed its task. Round times are also calculated individually for cyclic reading and cyclic writing so it is easier to monitor how round time of whole system is divided by individual modules.

Monitoring module is also responsible for calculating communication success percent. Every time slave station is read or written, status of transmission is stored in slaves Execution Parameter Data Block. Transmission status is a Boolean variable indicating that if one transmission to station was successful or not. 32 consecutive transmission status values are stored in one double word. Monitoring module counts bits with value one from 32 bit double word and calculates percentage value. Calculated value is then stored in Execution Parameter Data Block. For example last 32 transmissions to one slave is bit string 1111 0110 1111 0011 1111 1111 0011 1011. This bit string contains 25 bits with value one. Calculated percentage is then 25/32 * 100% = 78%.

Monitoring module also keeps track of Slave stations that are not communicating with master. Disconnection time calculation is based on communication success bits. If least significant (LSB) bit 1111…101**0** of transmission status double word (DWord) has value of 0 monitoring module adds calculated time of last cycle to Slaves Disconnection time variable. For example one station in the network was read or written four times per cycle. If every transmission was successful last four bits of transmission DWord are xxxx…**1111**. Because LSB has value of 1 Slave is communicating with master and Disconnection time variable has value of 0. But if last of four transmissions was failed (LSB=0) monitoring module adds last cycle time value to Disconnection time. Maximum value of disconnection time is 65535 seconds or approximately 18 hours. This calculated value is then used to determine when stations connection fault alarm is generated.

Figure 50 is UML Sequence Diagram that presents internal execution of monitoring module. After initialization module reads network parameters to its internal memory. Then based on internal step, module reads slave data, calculates values and writes values back.

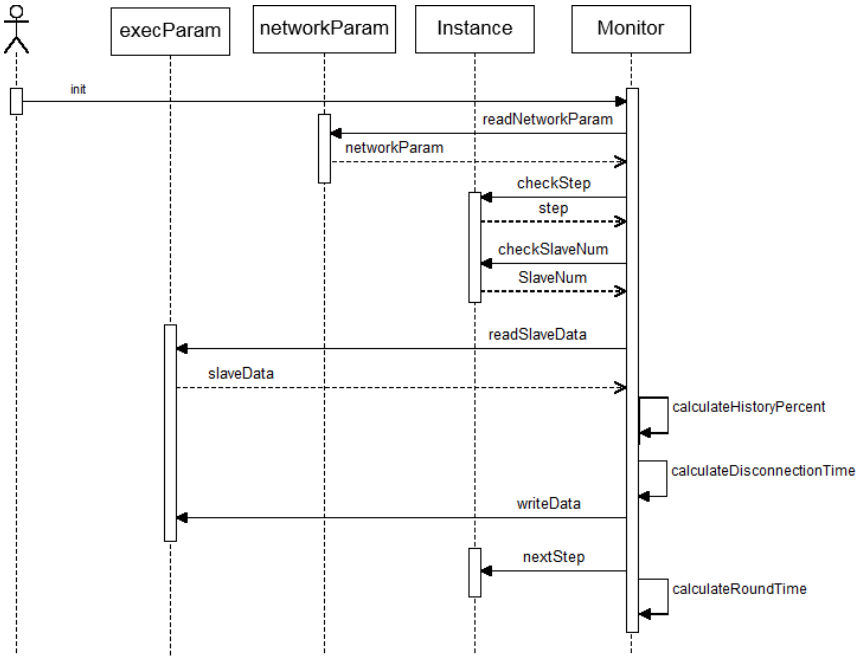

Figure 50 UML sequence Diagram of Monitoring Module

3.2.4.5 Data integrity

The data integrity methods presented in 2.2.1.2 are built in features of Modbus Function Block provided by Siemens. Parity checking is used in serial communication between devices. As explained in chapter 2.2.2.3 one Modbus RTU message frame consists up to 252 bytes of data. Parity checking is performed for every byte that has been transmitted between PLC and radio modem thus transmission of one frame requires up to 252 parity checks. CRC is calculated for every frame that has been transmitted. As noted in chapter 2.2.1.2 parity check will only detect single bit errors in transmitted bytes but multiple erroneous bits cannot be noticed. CRC check is required to ensure that there were no errors in transmitted frame. Figure 51 illustrates performed data integrity checks when two bytes are transmitted from one device to another using radio modems.

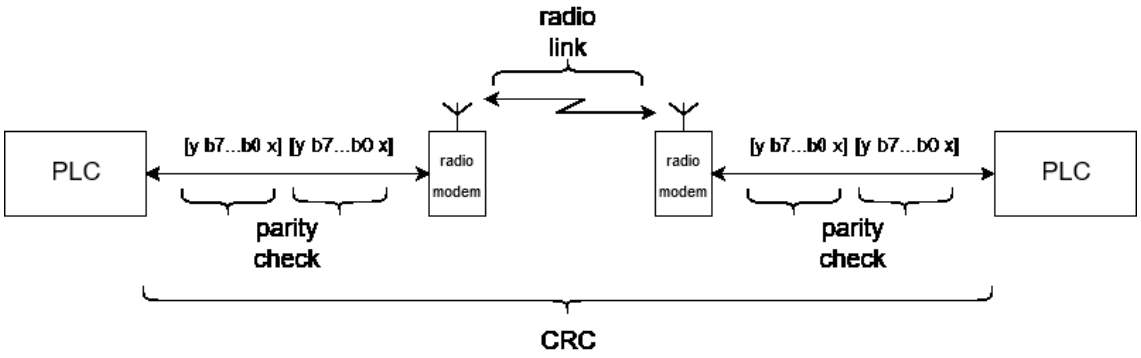

Figure 51 Data integrity checks in data transmission

3.3 SCADA Interface

UML Class diagram (Figure 52) of SCADA interface presents interactions between data blocks and SCADA Interface Function Block. Interface DB is used to communicate between SCADA system and PLC. The Function Block has read/write permission to all data blocks that are connected to it. Other modules of the system are not allowed to write Interface or Execution Parameters Data Blocks. In this way it is clear to system engineer which FB processes for example the request made by operator of the automation system.

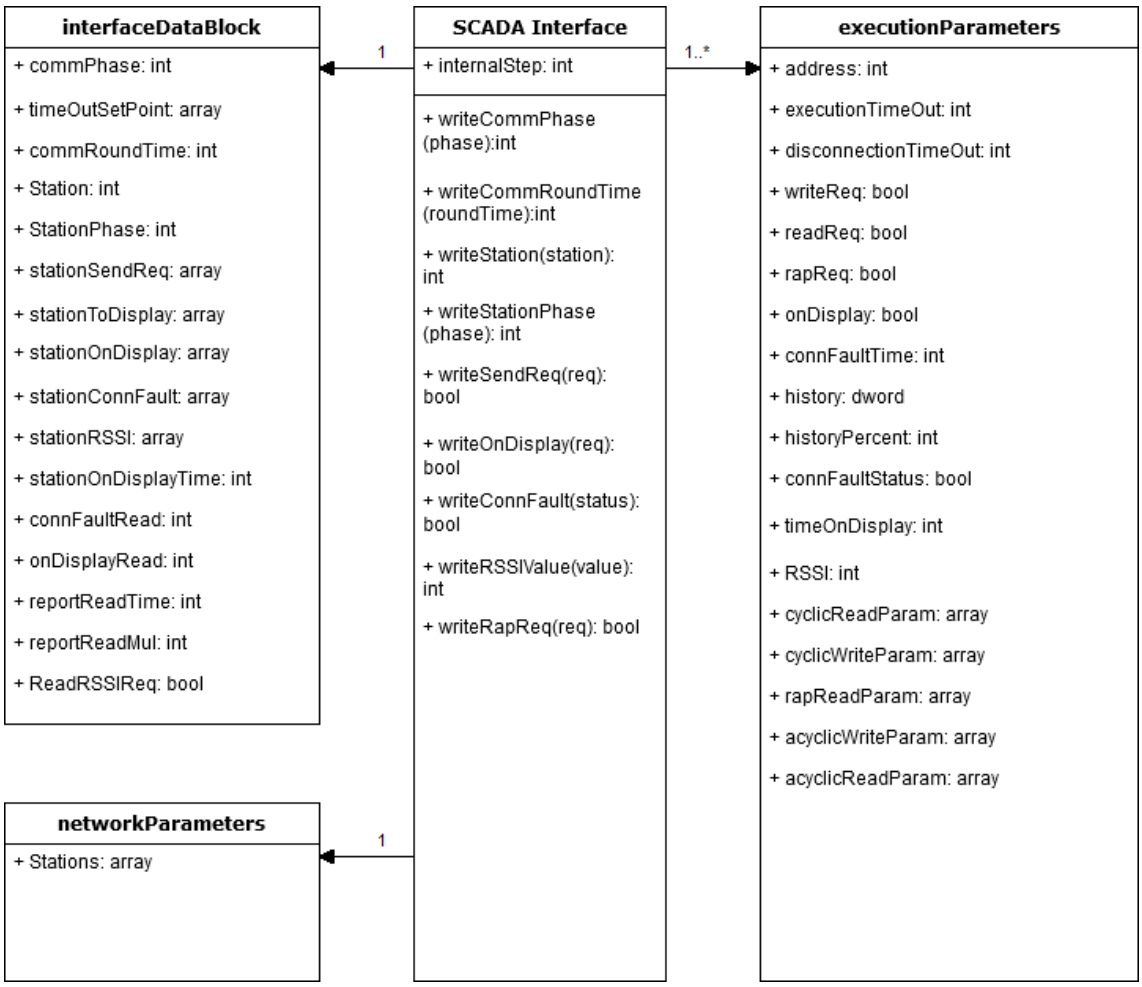

Figure 52 UML Class Diagram of SCADA interface

UML Sequence Diagram (Figure 53) of SCADA Interface presents the internal execution of interface FB. Function Block is executed in every PLC scan cycle (between 5-15ms). Function Block reads SCADA Interface DB to its internal memory in the beginning of the internal loop. Then request parameters are checked for every station in the network and written into station Execution Parameter DB. This means that SCADA system is not allowed to write directly to station's Execution Parameter DB. Linking the Master PLC to SCADA is easier when communication is done to only one DB. Programmer of PLC and SCADA are not always same person so it is easier for SCADA programmer to work with only one DB than link the system with several DBs in the PLC.

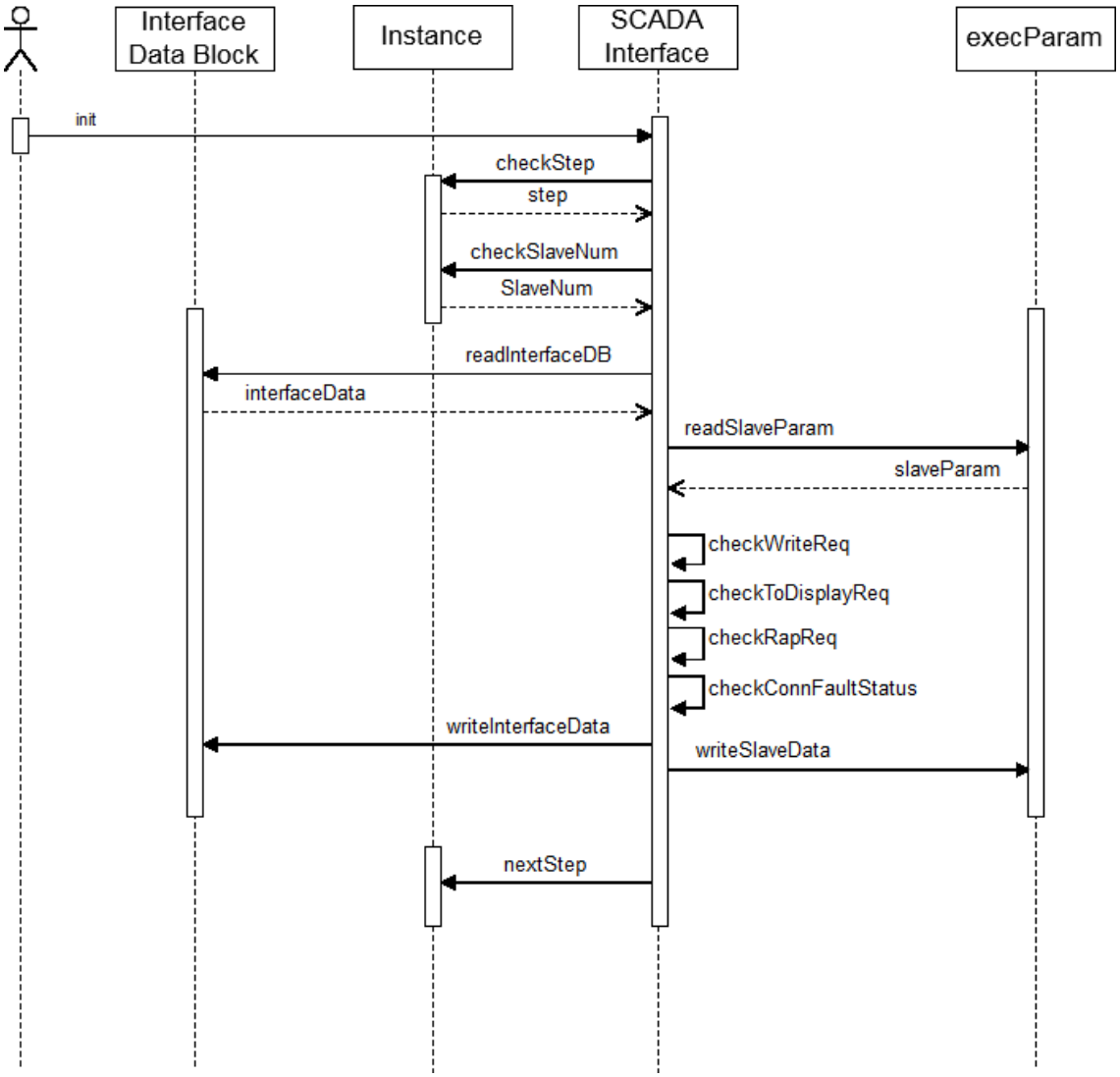

Figure 53 UML Sequence Diagram of SCADA Interface

3.4 Parametrization

Parametrization tool is simple excel based tool for creating Data Block source files. Parametrization tool will reduce time that is used to set up new remote monitoring system. The tool enables system engineer to easily create source files of all the data blocks that are needed for communication solution to work. Process of parametrization is presented in Figure 54. Parametrization of system starts with defining amount of stations in the network. Then is needed to determine DB numbers (address in PLC) for Execution Parameter DBs. Definition of Modbus parameters is the last step of parametrization process. Modbus address, register address and data length of every required read or write is stored in stations Execution parameters DB. Tool will generate source files for SCADA Interface, Network Parameters and Execution Parameters Data Blocks for every station on the network. Generated source files then can be imported into TIA environment for Data Block generation.

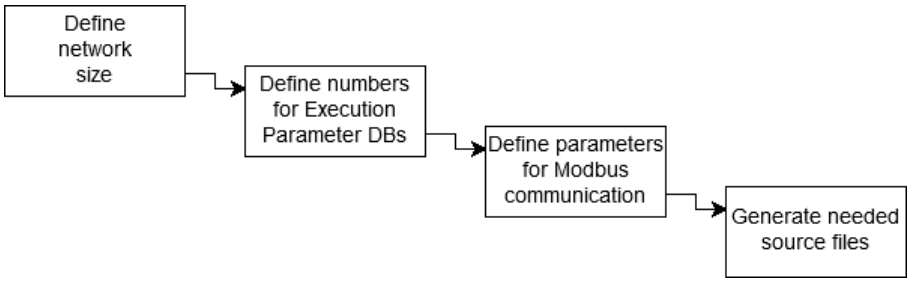

Figure 54 Parametrization process

4. COMMUNICATION SOLUTION IMPLEMENTATION

Previous chapter presented planning phase of communication system based on V-model (Figure 33). In this chapter is presented the implementation phase of the communication system. In V-Model planning flows from top to bottom, meaning that in every phase, plans become more elaborated. Implementation phase flows from bottom to top. First step is coding of lowest level software components. The Further the implementation phase goes program parts are connected together to form complete system. This chapter is divided into five parts. Every part considers different implementation phase based on V-Model.

4.1 Coding

In this phase abstract functionality of every module is translated into code that can be executed by PLC controller. Used Siemens 1200 series PLC can be programmed in four of IEC 61131-3 programming languages. Most of code was written in SCL (ST) and ladder (LD) languages.

4.1.1 Step 1 Basic Testing

The coding phase was started by defining the data blocks that are used by Cyclic Slave Reading module. In first phase of coding Data Blocks consisted only data that is needed for unit testing such as Modbus Address, Holding Register Address and Data Length. After definition of the needed variables was done the first test version of Cyclic Slave Reading module was created. It only consisted few inputs and outputs and the whole program was only constructed from two modules or Function Blocks. Figure 55 shows basic concept of usage of Cyclic Read and Modbus Master Function Blocks. This program was tested with actual hardware, Two PLC's were connected together using RS-232 bus which enabled the Master PLC to read and write the Slave PLC. The first test of the system was successful with only few lines of code and without any proper functionality.

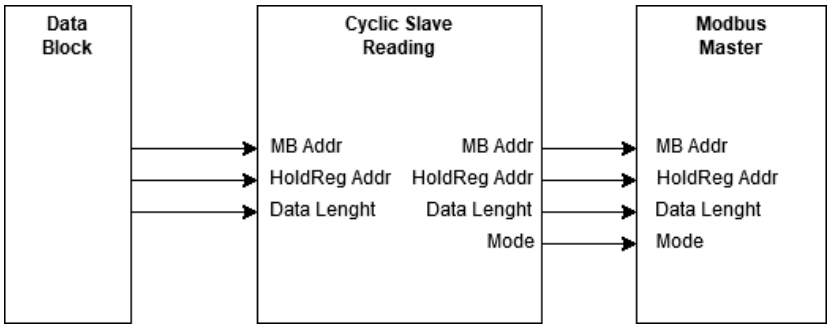

Figure 55 First testing version of Cyclic Read Module

4.1.2 Step 2 Adding Automatic Functionality

Next step of development was to create some automatic functionality to Cyclic Read module. The First test version required manual reading commands from user. For this step another Data Block was created. As chapter 3.2.4.3 presented Cyclic Slave Reading module is “connected” to one Network Parameters DB and several Execution and slave data DB’s. Addresses of Execution Parameters Data Blocks are stored in Network Parameters DB. Siemens TIA Portal programming environment offers some clever functions for testing Data Blocks in program. In this case before Network parameter DB is stored in the internal memory of Cyclic Slave Reading module it is tested that it consists an array of DB numbers. If the function returns false, Cyclic Slave Reading module is not allowed to continue execution. If testing of Network Parameter DB is successful, every Execution Parameters DB is also tested. Similar function can be used to test that if DB is derived from known data type. In this case every time Execution Parameters DB is created, it uses UDT Type (User Defined Data Type) in derivation of data block. It means that contents of Data Block cannot be changed afterwards without changing the UDT Type. It is important to test Data Blocks that are used indirectly before use, because loading a wrong type of data block can cause PLC to enter stop state. Figure 56 shows simplified procedure of DB checking in modules of the program.

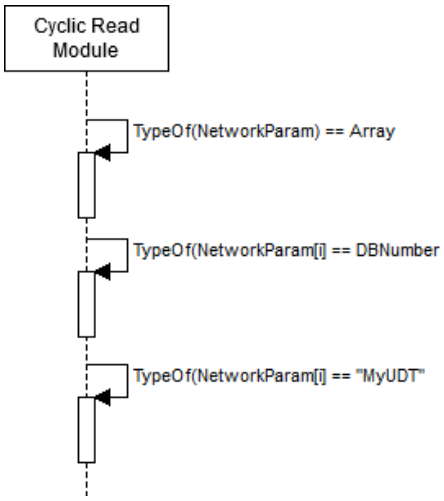

Figure 56 Procedure of DB checking in Module

At this point Cyclic Read module already has the information about Data Blocks that are "connected" to it. Saying that DB's are connected to module is not strictly true. The Only DB that is actually connected to module is Network Parameters DB and the other DB's that module uses are addressed indirectly. It means that module only knows address of DB's and it cannot access them through its input interface. In this case module needs to have internal variable that is derived from UDT Type. Data from actual Execution Parameters DB can be copied to internal variable and used in program without explicit connection to module. Figure 57 shows simplified process of copying DB to modules internal memory.

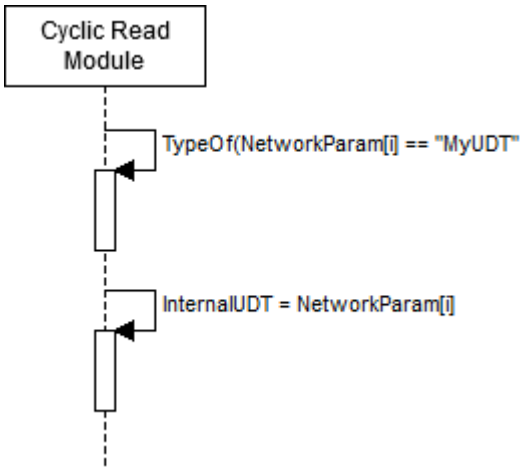

Figure 57 Copy DB to internal memory

Now it is possible to create loop trough Network Parameters DB. Array in the DB can be for example from 0 to 99. In most real life cases adequate size for array is between 20 and 40 DB numbers. Reserve in the DB is justified because changing the size of the array afterwards will cause the PLC to enter stop state which is always risky when PLC is in use. Loop size in the program can be defined to be dynamic by using function that checks

how many of the indexes are actually used. In this case PLC will not need to loop through all the indexes but only the ones that have other value than zero.

4.1.3 Step 3 Storing Data and Error Handling

In previous step the module was able to generate function for Modbus Master but it did not actually read the data from Modbus Buffer DB. For this step some modifications was made to modules interface. Modbus Master indicates whether reading was done or some errors occurred while reading or writing of Slave device. Two additional inputs were defined to module. One for reading done and one for error in reading.

After Modbus Master indicates that reading was carried out successfully and it has written the read data to Modbus Buffer Data Block. After that Cyclic Read module can read the data from Buffer DB and write it to Slave Data DB. Transmission is not 100% sure and sometimes packets are lost or corrupted. When transmission was not successful Modbus Master indicates that error has occurred. After error, Cyclic Read module will not read the Modbus Buffer because the data is not valid. There is basically two options what module can do after error has occurred. It can request re-transmit or just pass station for this round and try to read it in the next round. In this case modules will not request re-transmit because one lost transmission will not have major effect, if station can be read in the next round. Figure 58 shows simplified process of storing data from Modbus Buffer after transmission and handling of erroneous transmission.

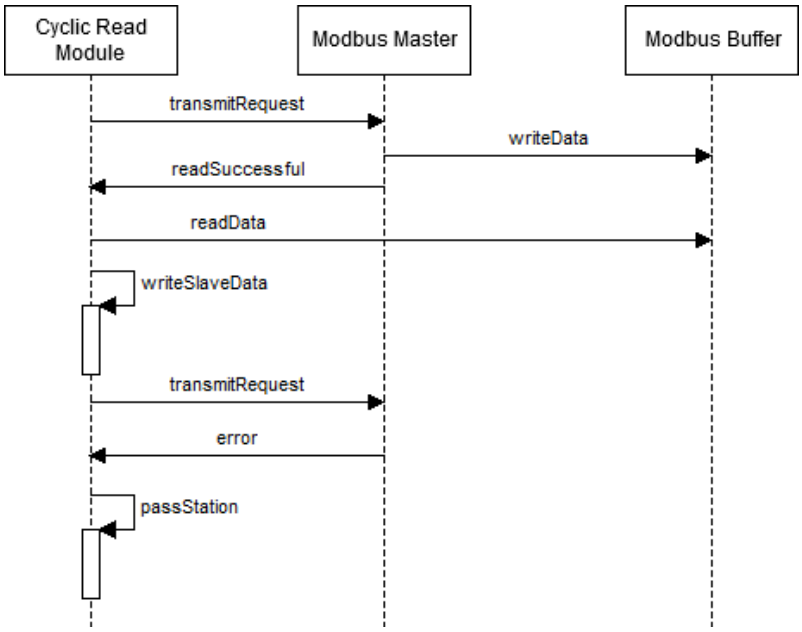

Figure 58 Storing data and error handling

4.1.4 Step 4 Additional Features

Until this step the module was able to read cyclically all stations in the network. Next step was to implement own loop for stations that are currently viewed in SCADA. In large systems the round times can be up to 30-40 seconds. It means that values are updated to SCADA or HMI every 30 seconds. When operator of the automation system is viewing specific station, it would be useful if communication system is able to read one or more stations in the network more frequently. SCADA Interface module writes on display indication to Execution Parameter DB. Cyclic Reading module uses that information for reading specific stations more often than other stations in the network. Read on display-parameter is used to determine how often viewed stations are read. Figure 59 shows reading sequence of stations when Read on display-parameter is set to three. In the example there are two station on display in the network. In Figure 59 green boxes represents currently viewed stations which are read more often than stations that are not on display.

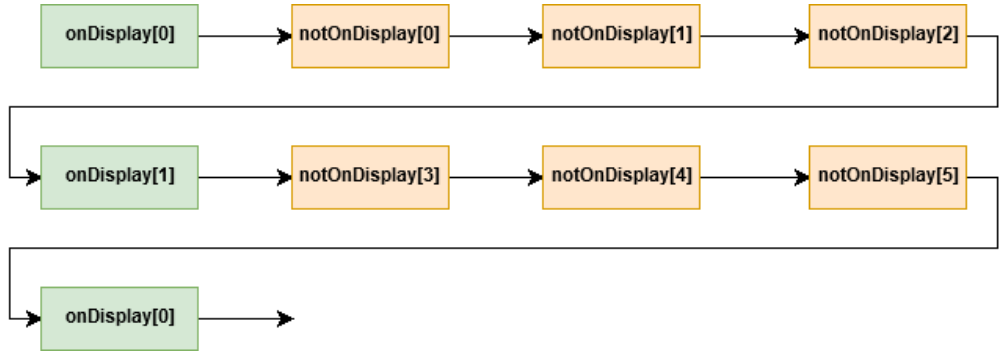

Figure 59 Read on display sequence

There is sometimes a situations where remote stations cannot be reached at all. It is not needed to try read or write not responding stations cyclically because it will slow down the round time of the communication system. Monitor module checks every transmission to remote station. If station's transmissions are failed consecutively for example one minute, station is will be set to communication fault state. Cyclic modules then can pass stations which are not communicating with Master station. End user can define how often Master station tries to read remote stations that are in communication fault state. Figure 60 shows example of reading communication fault stations every five rounds. In this case if there are stations that are not responding round time is not increased for every round, but only rounds when faulty stations are tried to read or write.

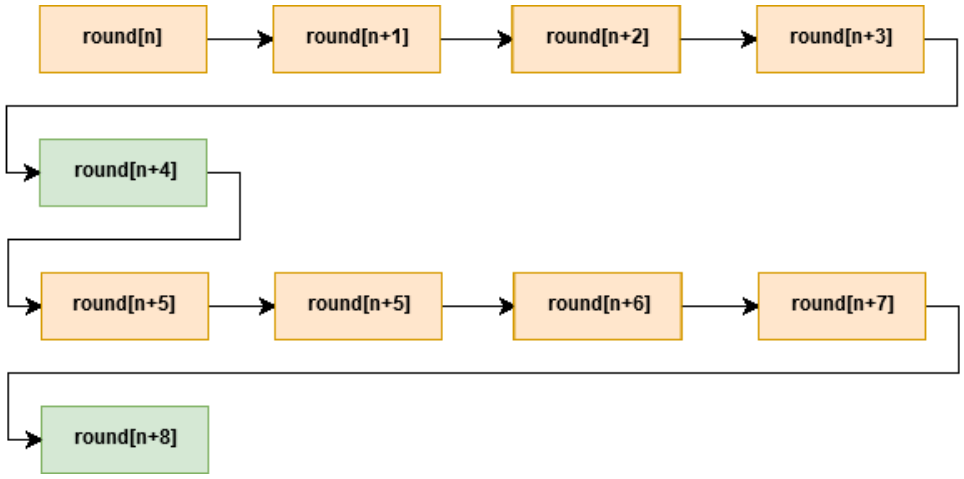

Figure 60 Reading stations with communication fault

Chapter 3.2.4.2 presented Read RSSI Use case of Cyclic Read module. This functionality is not part of communication between Master and Slave station. SCADA or user of automation system can read RSSI (Received Signal Strength Indicator) values from radio modem if needed. Satel Radio modems which were used in this thesis stores value of previous successful transmission. PLC can be used to read that value from radio modem and show it to user. There could be situation where signal strength of transmission to one or more stations is slowly deteriorating. If RSSI values are read for example once a day the data can be used to draw trends or to generate alarms if signal strength is lower what is should or used to be.

Addition of RSSI read feature needed modifications only to Event Trigger module. RSSI values can only be read when no other data transmission is active between PLC and radio modem. In this case if RSSI read is requested, Event Trigger prevents any of the other modules to operate and gives permission to RSSI read Function Block to read values. After reading is done RSSI read notifies Event Trigger that reading is finished and other modules then can continue their normal execution. Figure 61 illustrates sequence of RSSI value reading.

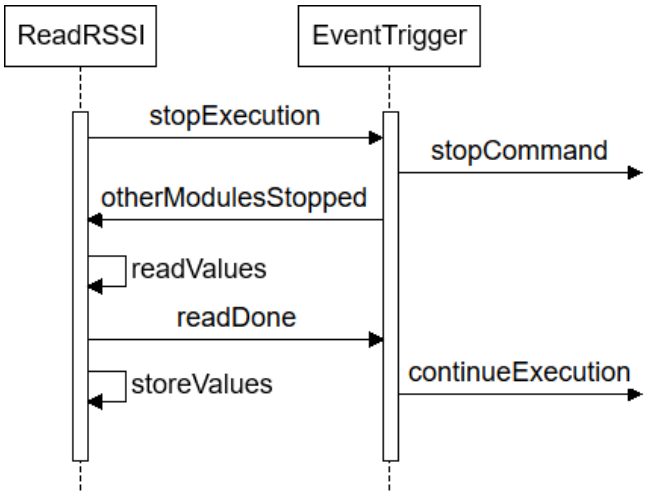

Figure 61 Sequence of RSSI reading

4.2 Unit Testing

Unit testing was done parallel with coding. Every time small new functionality was added to module it was tested. Previous chapter presented coding steps of Cyclic Read module. Unit testing of the module was done in simulated environment with Master station and Slave station connected by RS-232 bus. Functionality of bus cable and radio modem is the same, only data transmission speed is faster with cable. Simulation environment consist only two PLCs so testing the module with multiple slave stations demanded bit of reasoning. The Master PLC was set to read 10 "different" stations. It had Network Parameter Data Block with 10 different DB numbers stored into it. Also 10 Execution Parameter DBs and Slave Data DBs were created. Modbus Slave device can only have one Modbus address so every Execution Parameter DB had the same Modbus address but data was read from different areas of Slave device's Holding register. Figure 62 shows principle of testing environment

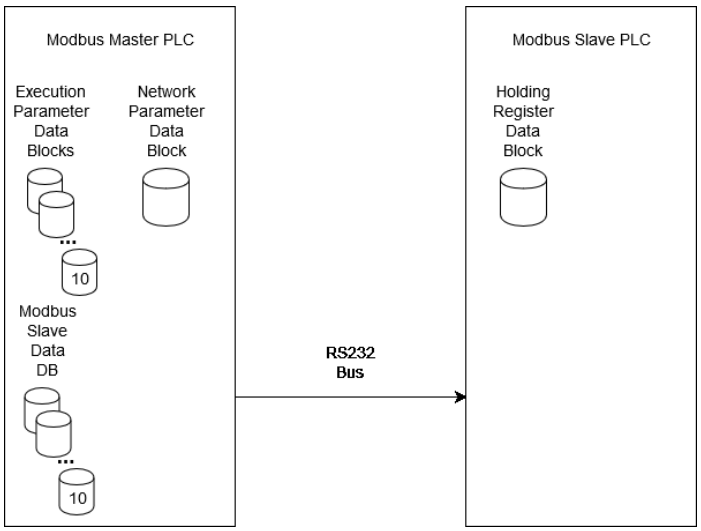

Figure 62 Testing environment

This test revealed some problems with Cyclic Read module. Testing with only one station in the network was successful but addition of several stations lead to problem with writing the Slave Data DBs. Cyclic Read module wrote data to wrong Slave Data DB because the index of the loop in the module was changed after triggering the transmission not after the data was actually read from Modbus Buffer. The problem was not seen when only one station was read in the loop.

As presented in 3.2.4 every module's functionality is fundamentally the same. When first module was tested it was a base point for other modules. Cyclic Write module's functionality is similar to Cyclic Read modules but it will not write to Slave Data DB but read data from there and write to Modbus Buffer DB. Difference between reading and writing modules is presented in Figure 63.

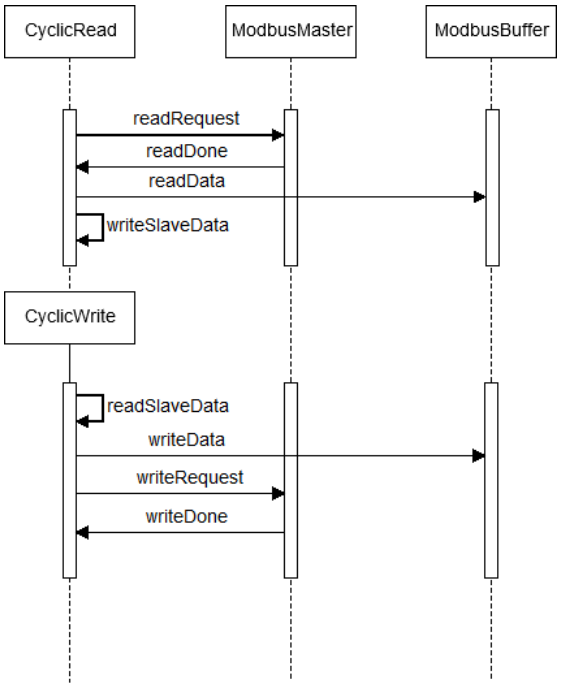

Figure 63 Reading and writing Modbus Buffer

This change is Modbus Buffer handling did not had an effect on execution of internal loop of module. When the first implemented module was properly tested, testing on Cyclic Write module was faster and there were no similar errors or bugs as there were with Cyclic Read module. These two modules are the base points for other cyclic and acyclic modules. For example Report Reading module has exactly the same functionality as Cyclic Read module, it only checks before reading if the report request is active in the Execution Parameter DB. One module could have done both cyclic and report reading, but it was determined that it is easier to understand the program execution if there were independent modules for both.

4.3 Integration Testing

When every unit of the program was created the integration and cooperation of the modules were tested. There are five modules in the program that generates functions for Modbus Master and three other modules: Monitor, Event Trigger and SCADA are also affecting to functionality of the program. Integration phase was started with just two modules. Figure 64 shows the principle of the first part of the integration testing. Cyclic Read and Write modules were connected together using Step variable. Cyclic Read was assigned to execute when Step = 1 and Cyclic Write when Step = 2. Every time Cyclic Read modules internal execution was finished Step variable's value was changed from 1 to 2, and after Cyclic Write was executed Step was set from 2 to 1.

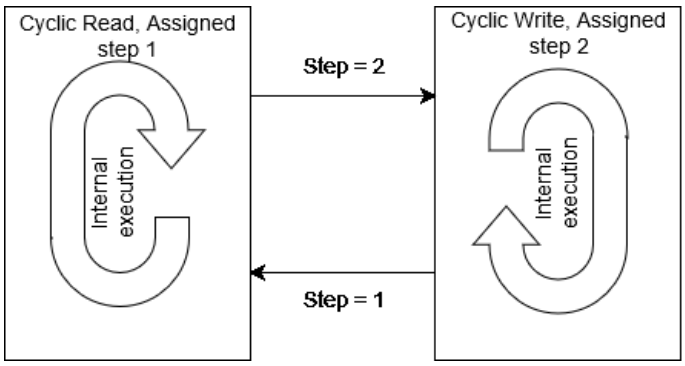

Figure 64 First step of integration testing

Integration testing consisted six iterations where each time one more module was added to program. After third iteration the program featured all four cyclic modules (Figure 65). At this point no new problems or bugs were found in the program.

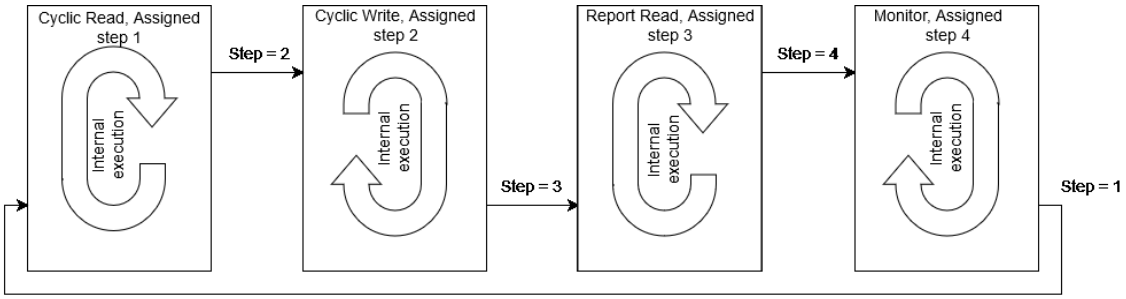

Figure 65 Third step of integration testing

Final stage of integration testing consisted all seven modules. Addition of Event Trigger and acyclic modules had some undesired effect on cyclic modules. Problems were encountered when execution of cyclic modules were stopped for acyclic execution. After the pause cyclic module was unable to continue its execution. Handling of stop commands were modified for every cyclic module so they operated correctly after acyclic operation was finished. This caused some extra work with all cyclic modules because stop command was not tested properly in unit testing phase.

4.4 System Testing

System testing was carried out using simulation environment which consisted three PLCs, three radio modems and PC with Simatic WinCC SCADA system (Figure 66). Previous testing phase was carried out with just two PLCs and RS-232 bus but this kind of system is not common in real life applications. System covers Master PLC, one PLC which radio modem was a link modem and Slave PLC. So basically this testing environment consisted Master and two Slaves and the data transmission between PLCs was done using serial radio modems.

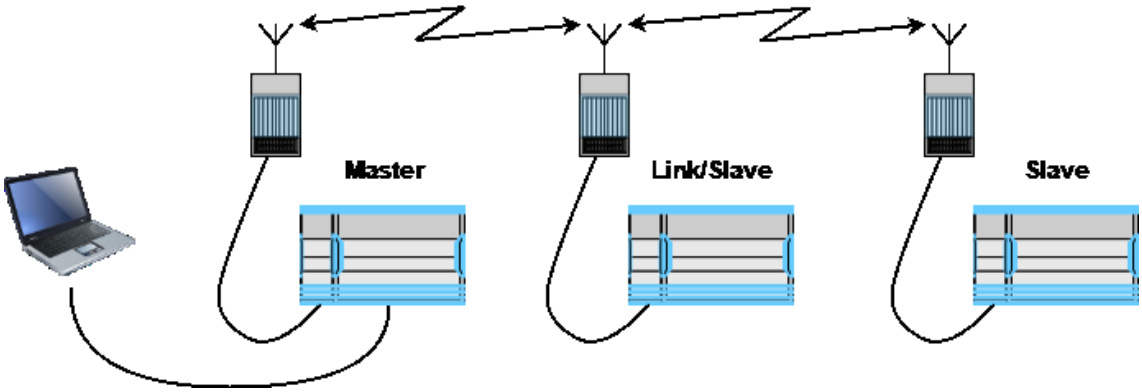

Figure 66 System testing environment

SCADA system was copy of Insta Automation's customer's SCADA system. Also Master and Slave PLCs were copies of existing stations from the same automation system, only Master station's communication system was replaced with new implementation. This arrangement provided testing environment which was as close to a real system as possible.

System testing phase did not revealed major bugs or errors in code. In this stage of testing it was a good point also to test more the parametrization tool. Slave station was "changed" couple of times and every time different program was downloaded to Slave PLC new Data Blocks were created to Master PLC with parametrization tool. System testing took two days in which five different Slave stations were tested with SCADA and all the features of new implementation were tested and found working in this kind of small system.

4.5 Acceptance Testing

Proper testing of the system is difficult when only simulation environment is used. Communication implementation was installed in Insta Automation's customer's remote monitoring system. The whole system comprises of SCADA, seven Master PLCs and about hundred Slave stations (Figure 67). New communication system was installed in Master station 2 which communicates to 20 Slave stations.

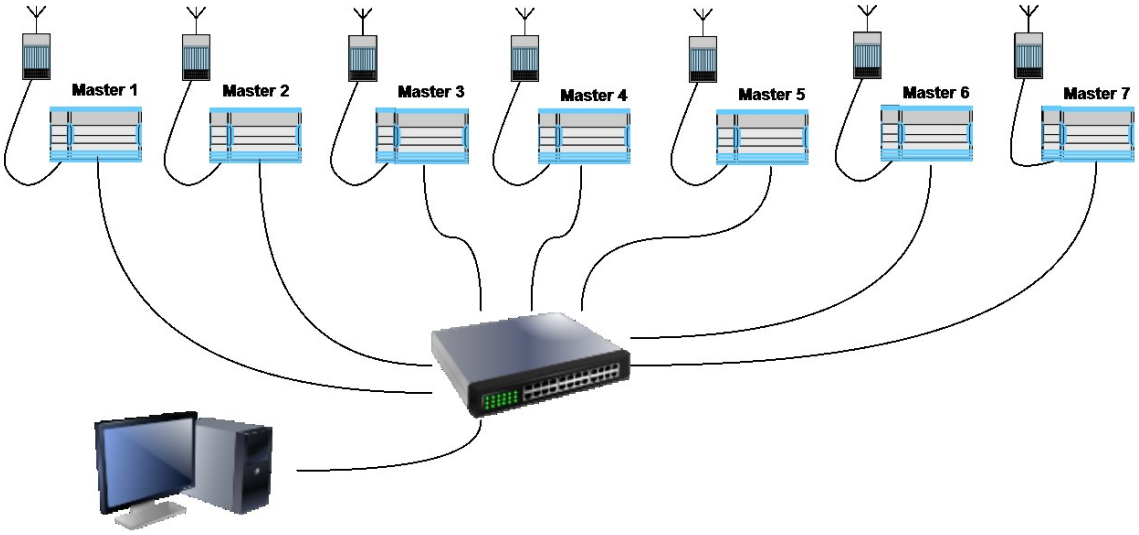

Figure 67 System structure

5. RESULTS AND CONCLUSIONS

This chapter sums up the results of the performed work. The results are compared to theoretical values and measurements form the old system. Also future prospects are considered in the last chapter.

5.1 Performance Validation

Communication system based on serial radio modems has three indicators that reflects the performance of the system. First measurable indicator is round time of the system. As it has been presented in chapter 3.2.4.4 system calculates three different round times. The calculated round time signifies how often data from one station is transferred to the SCADA system. Individual round time calculations are also executed for cyclic read and write sequences. On the basis of cyclic sequence round time calculations, user of the system can determine how the whole round time has been divided between the two different cyclic executions.

Second indicator of the system performance is how much time it takes to transfer acyclic data from remote station when it is selected onto HMI display. As explained in chapter 3.2.4.3 acyclic data is only read from the station when it is viewed in HMI. Since the Modbus Master can only communicate to one station at the time the cyclic execution has to be paused in order to read acyclic data from another station. Pausing the cyclic execution cannot be done immediately but after current transmission is over. Third performance indicator is how long it takes to send acyclic data to station. As with acyclic read the cyclic execution has to be interrupted before transmission is initiated.

5.1.1 Cyclic Execution Comparison

The round time of the system were measured from both old and improved systems. The measured round time is average from twenty individual measurements. There was only minor difference in measured times. Calculated average and standard deviation of values is presented in Figure 68. Standard deviation value depicts how far the values are from arithmetic average of all the values. Bigger deviation value means that individual measurements have been varying widely. The actual round time measurements are compared to theoretical calculations.

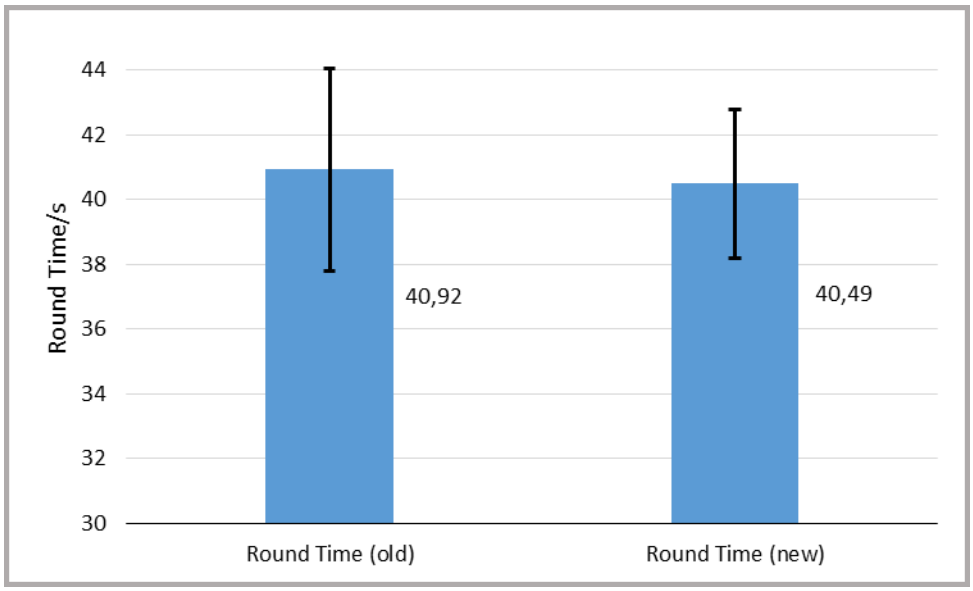

Figure 68 Communication system round time comparison

Theoretical round time of the system without repeater station can be calculated by using equation 16 where RT is round time, D is amount of data to be transmitted and S equals transmission speed in bits per second. Calculated values are presented in Table 6.

$$RT = \frac{D}{S}, \tag{16}$$

Table 6 Theoretical transmission and round times

Station	Read resp (byte)	Write resp (byte)	Transmission time
1	110	10	0,14
2	110	10	0,14
3	212	10	0,25
4	124	10	0,15
5	94	10	0,12
6	94	10	0,12
7	94	10	0,12
8	142	10	0,17
9	212	10	0,25
10	212	10	0,25
11	94	10	0,12
12	212	10	0,25
13	142	10	0,17
14	142	10	0,17
15	212	10	0,25
16	274	10	0,33
17	124	10	0,15
18	124	10	0,15
19	212	10	0,25
20	124	10	0,15
SUM	3064	200,00	3,74

This round time calculation does not represent real life situation because this includes only the payload but not the requests or responses. In addition the data transmission in this system is not direct but there is relay station thus all data is first transmitted to relay

radio and from there to remote station. Figure 69 shows the principle of data transmission with link station. When the link station has received the data it cannot send it to Slave station before the transmission has ended. As presented in chapter 2.2.1.1 the transmission type is half-duplex therefore devices cannot receive and send data simultaneously.

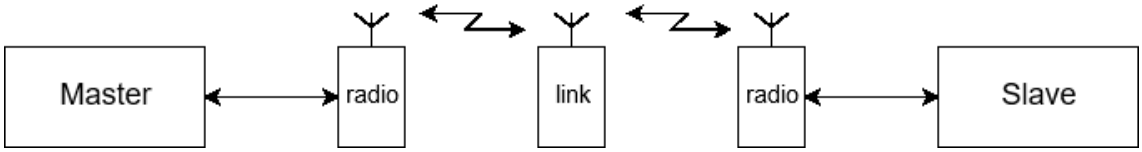

Figure 69 Radio network with link station

When request and response data and the delays affected by the link station have been taken into account theoretical round time can be calculated using equation 17. Terms used in the equation are Read request (Rreq), Read response (Rresp), Write request (Wreq) and Write response (Wresp).

$$RT = \frac{(Rreq+Rresp+Wreq+Wresp)*2}{S}, \quad (17)$$

Table 7 presents calculations of round times for each of network's station. Read and write requests increased transmitted data by 10 bytes and delay of the link station doubles the transmission time. The actual round time is still much longer than calculated value. These calculations do not take account the execution time of PLC when read or write request is processed. PLC's internal execution time of the code is between 5-15 ms and therefore more delay to communication is affected by request processing. Also this system's radio modems feature FEC (Forward Error Correction) which according to manufacturer increases data output by 30% [8]. Therefore round time estimation of communication is difficult.

Table 7 Theoretical round times with delays

Station	Read req (byte)	Read resp (byte)	Write req (byte)	Write resp (byte)	Transmission time
1	5	110	5	10	0,30
2	5	110	5	10	0,30
3	5	212	5	10	0,53
4	5	124	5	10	0,33
5	5	94	5	10	0,26
6	5	94	5	10	0,26
7	5	94	5	10	0,26
8	5	142	5	10	0,37
9	5	212	5	10	0,53
10	5	212	5	10	0,53
11	5	94	5	10	0,26
12	5	212	5	10	0,53
13	5	142	5	10	0,37
14	5	142	5	10	0,37
15	5	212	5	10	0,53
16	5	274	5	10	0,67
17	5	124	5	10	0,33
18	5	124	5	10	0,33
19	5	212	5	10	0,53
20	5	124	5	10	0,33
SUM	100	3064	100	200,00	7,94

5.1.2 Acyclic Execution Comparison

The following presents comparison of the other two indicators of system performance. Acyclic read (To Display) and acyclic write (Transmit) times were each measured sixty times and the averages can be seen from Figure 70. Difference between measured times in old and new system is considerable. The old communication system's execution was cyclical and therefore when station was selected to be shown in HMI display the system could not read acyclic data immediately. The acyclic data was read from the station when it was selected on HMI and when it was the station's turn in the cyclic execution. Acyclic writing to station was implemented similarly. Transmission to station only took place when sending was requested and the cyclic execution reached the station. This meant that delays in acyclic read and write were varying because station were read in loop and there were no possibility to read the station outside the loop.

The measured times and deviation of measurements are lower in the new implementation because it is possible to read or write station independent from the cyclic loop. Lower deviation in measurements mean that variation on to display and transmission times is smaller. This meant that the end user can expect transmission to station takes approximately the same time in every transmission.

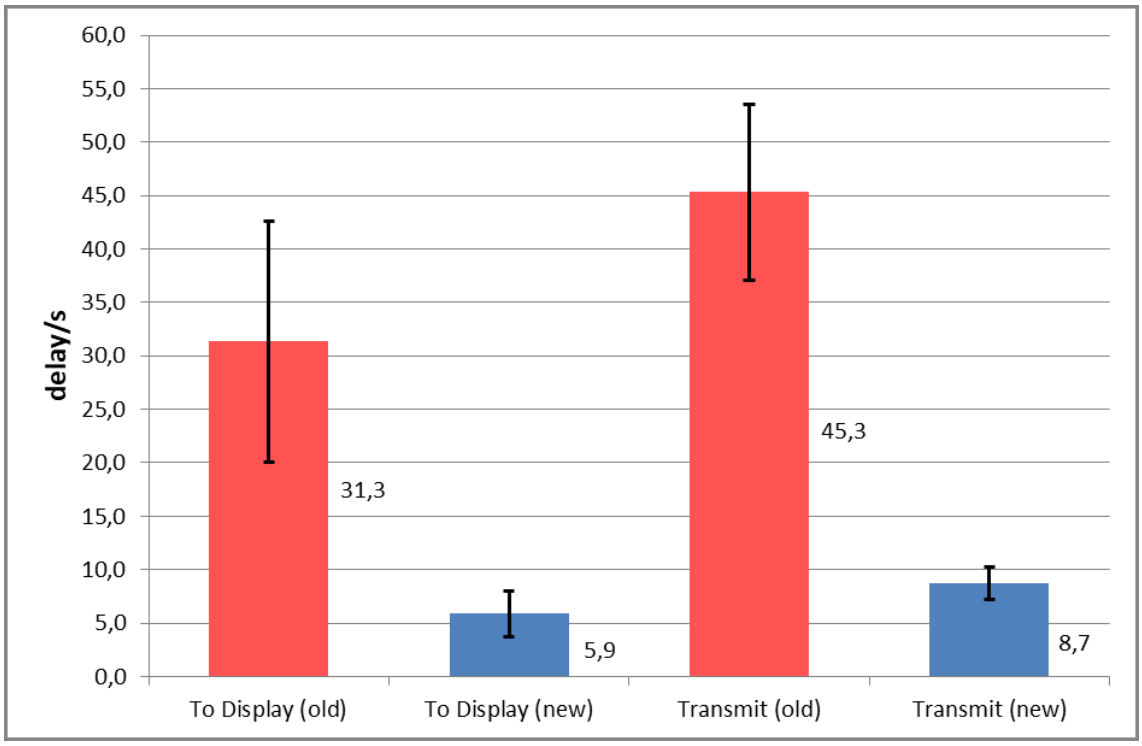

Figure 70 Communication system delay comparison

5.1.3 Summary

Based on the results in previous chapters it can be stated that there was notable improvement in system performance. One of the key design parameters was to improve end user experience. Round time of the system could not be decreased because data transfer speed remained the same. The round time has less effect on user experience since remote plants are usually stand-alone control devices and their operation is not dependent on the round time of the communication system. End user's requirements are that all important data is transferred from remote plants systematically. Improvements in acyclic transfer is more important because it is related to end user experience. For example the operator needs to change one set point value of the station. With the new system the waiting time is reduced by 80% (equation 18)

$$wait\ difference = 100\% - \frac{(5{,}9+8{,}7)}{(31{,}3+45{,}3)} * 100\% = 80{,}1\%, \qquad (18)$$

The new system was developed also to be easy to set up and configure. Parametrization tool was developed to reduce time that it takes to set up new remote monitoring system. Also it is not needed for system engineer to completely understand how the communication system works, but only follow the steps in parametrization process. Engineer can only use template project and generate needed Data Blocks for the system.

5.2 Further Work

This implementation of communication system will be used in future remote monitoring projects in Insta Automation Oy. Testing of the system will continue after the completion of the thesis. Further modifications to software might be made when the end users have gathered enough experiences and are able to give feedback of the system functioning. Reliability of the system is still unproven because it has not been in use for long enough. This implementation is currently in use only in one of Insta Automation's customer's remote monitoring system but there are other projects in progress where results of this work can be utilized.

Even though the system was developed to be easy to use other employees of Insta need to be instructed. It will require writing a manual for the system and training sessions with other engineers. This will also ensure that Insta's engineers are capable of serving the customers and manage possible problems that are faced.

REFERENCES

[1] P. Gaj, J. Jasperneite, M. Felser, Computer Communication Within Industrial Distributed Environment-a Survey, IEEE Transactions on Industrial Informatics, Vol. 9, Iss. 1, 2013, pp. 182-189.

[2] T. Hadlich, C. Diedrich, K. Eckert, T. Frank, A. Fay, B. Vogel-Heuser, Common communication model for distributed automation systems, 2011 9th IEEE International Conference on Industrial Informatics, pp. 131-136.

[3] B.R. Mehta, Y.J. Reddy, Industrial Process Automation Systems: Design and Implementation, Butterworth Heinemann, US, 2014.

[4] F. Pethig, B. Kroll, O. Niggemann, A. Maier, T. Tack, M. Maag, A generic synchronized data acquisition solution for distributed automation systems, Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012), IEEE, pp. 1-8.

[5] Vlad Cristian Georgescu, Operations Management in Water and Wastewater Treatment Plants, Applied Mechanics and Materials, Vol. 245, 2012, pp. 179. https://search.proquest.com/docview/1442702416.

[6] X. Zhou, P. Xiang, Y. Ma, Z. Gao, Y. Wu, J. Yin, X. Xu, An overview on distribution automation system, 2016 Chinese Control and Decision Conference (CCDC), IEEE, pp. 3667-3671.

[7] Evaluation of Wireless Technologies for Power Delivery Automation, Schweitzer Engineering Laboratories Inc., 2012.

[8] SATTELINE-3AS NMS User Guide, Version 6.0, 2014, pp. 94, Available: https://www.satel.com/wp-content/uploads/2018/04/SATELLINE-3AS_NMS_V7.pdf

[9] B. Flerchinger, R. Ferraro, C. Steeprow, M. Mills-Price, J.W. Knapek, Field Testing of 3G Cellular and Wireless Serial Radio Communications for Smart Grid Applications, 2016 IEEE Rural Electric Power Conference (REPC), IEEE, pp. 42-49.

[10] Radio technology ensures reliable communications, Available (accessed January 2018): https://www.satel.com/products/.

[11] FICORA grants radio licences, Available (accessed January 2018): https://www.viestintavirasto.fi/en/spectrum/radiolicences.html.

[12] SATELLINE-EASy Pro User Guide v.1.7, 2018, pp. 104, Available: https://www.satel.com/wp-content/uploads/2018/03/SATELLINE_EASy_Pro_V_1_7.pdf.

[13] H.M. Hashieman, ANSI/ISA-95.00.01-2010 (IEC 62264-1 Mod) Enterprise-Control System Integration - Part 1: Models and Terminology, 2010.

[14] B. Galloway, G.P. Hancke, Introduction to Industrial Control Networks, IEEE Communications Surveys & Tutorials, Vol. 15, Iss. 2, 2013, pp. 860-880.

[15] T. Sauter, The Three Generations of Field-Level Networks-Evolution and Compatibility Issues, IEEE Transactions on Industrial Electronics, Vol. 57, Iss. 11, 2010, pp. 3585-3595.

[16] S.K. Sen, Fieldbus and Networking in Process Automation, 1st ed. CRC Press, Bosa Roca, 2015.

[17] ANY STATE OF MATTER - THE AUTOMATION SOLUTION IS ALWAYS AS-INTERFACE, Available (accessed January 2018): http://www.as-interface.net/applications/process-automation.

[18] CANopen – The standardized embedded network, Available (accessed January 2018): https://www.can-cia.org/canopen/.

[19] ControlNet™ -
CIP on CTDMA Technology, in: PUB00200R1, 2016.

[20] ODVA DeviceNet, Available (accessed January 2018): https://www.odva.org/Technology-Standards/DeviceNet-Technology/Overview.

[21] Fieldbus Foundation - Our Technologies, Available (accessed January 2018): http://www.fieldbus.org/index.php?option=com_content&task=view&id=23&Itemid=308.

[22] HART - Digital Transformation for Analog Instruments, Available (accessed January 2018): https://www.fieldcommgroup.org/technologies/hart.

[23] INTERBUS, Available (accessed: January 2018): https://www.profibus.com/technology/interbus/.

[24] MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3, Available: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf.

[25] PROFIBUS System Description, PROFIBUS & PROFINET International, 2018, Available: https://www.profibus.com/download/profinet-technology-and-application-system-description/.

[26] Profibus, Available (accessed Januray 2018): http://www.siemens.fi/fi/industry/teollisuuden_tuotteet_ja_ratkaisut/tuotesivut/automaatiotekniikka/teollinen_tiedonsiirto_esim_profinet/profibus.htm.

[27] White Paper Distributed, Intelligent I/O for Industrial Control and Data Acquisition … The SERIPLEX® Control Bus, SERIPLEX Technology Organization Inc, 1997, Available: https://www.idc-online.com/technical_references/pdfs/data_communications/seriplexwhitepaper.pdf.

[28] W.A. Shay, Understanding data communications and networks, PWS Publishing Co, Boston (MA), 1995.

[29] MODBUS over Serial Line Specification & Implementation guide V1.0, MODBUS.ORG, 2002, Available: http://www.modbus.org/docs/Modbus_over_serial_line_V1.pdf.

[30] MODBUS MESSAGING ON TCP/IP IMPLEMENTATION GUIDE V1.0b, Available: http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf.

[31] M. Fowler, K. Scott, E. Sarkkinen, UML, Docendo, Jyväskylä, 2002.

[32] An OMG® Unified Modeling Language® Publication, Object Management Group, 2017.

[33] State Machine Diagrams, Available (accessed February 2018): https://www.uml-diagrams.org/state-machine-diagrams.html.

[34] IBM Knowledge Center - UML State Diagram, Available (accessed February 2018): https://www.ibm.com/support/knowledgecenter/en/SS6RBX_11.4.2/com.ibm.sa.oomethod.doc/topics/c_UML_State_diag.html.

[35] UML Sequence Diagram Tutorial, Available (accessed February 2018): https://www.lucidchart.com/pages/uml-sequence-diagram.

[36] UML Class Diagram Tutorial, Available (accessed 2018): https://www.lucidchart.com/pages/uml-class-diagram.

[37] Bell Donald UML Basics: The class diagram, Available (accessed: February 2018): http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/index.html.

[38] Suomen standardisoimisliitto, Automaatio: Osa 2 = Part 2, Ohjelmointi ja teollisuusprosessien valvonta = Programming and industrial-process control. Automation, Suomen standardisoimisliitto, Helsinki, 2006.

[39] Overview of the IEC 61131 Standard, ABB, Available: https://library.e.abb.com/public/81478a314e1386d1c1257b1a005b0fc0/2101127.pdf.

[40] IEC 61131-3 Protocol Overview, RTA, Available (accessed February 2018): https://www.rtaautomation.com/technologies/control-iec-61131-3/.

[41] Introduction into IEC 61131-3 Programming Languages, Available (accessed February 2018): http://www.plcopen.org/pages/tc1_standards/iec61131-3/index.htm.

[42] F. Bonfatti, P.D. Monari, U. Sampieri, IEC 1131-3 programming methodology, CJ International, Seyssins, 1997.

[43] K. John, M. Tiegelkamp, IEC 61131-3: Programming Industrial Automation Systems, Second; 2nd ed. Springer Verlag, DE, 2010.

[44] J. Stenerson, Fundamentals of programmable logic controllers, sensors and communications, Regents/Prentice Hall, Englewood Cliffs (NJ), 1994.

[45] M. Bonfe', C. Fantuzzi, L. Poretti, PLC object-oriented programming using IEC61131-3 norm languages: An application to manufacture machinery, 2001 European Control Conference, ECC 2001, pp. 3235-3240.

[46] DESCRIBING PROGRAMS, in: BCS Glossary of Computing and ICT, 2013.

[47] Distributed control applications: guidelines, design patterns, and application examples with the IEC 61499, CRC Press Taylor & Francis Group, Boca Raton, FL, 2016.

[48] P. Isaias, T. Issa, High Level Models and Methodologies for Information Systems, 1st ed. Springer-Verlag, New York, 2015, 145 p.