

BACHELORTHESIS
Mykhailo Svyrydovych

RPC-based cross-platform GUI application for configuring sensor modules over PROFIBUS and storing data

FACULTY OF COMPUTER SCIENCE AND ENGINEERING
Department of Information and Electrical Engineering

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

HAMBURG UNIVERSITY
OF APPLIED SCIENCES
Hochschule für Angewandte
Wissenschaften Hamburg

Mykhailo Svyrydovych

**RPC-based cross-platform GUI application for configuring
sensor modules over PROFIBUS and storing data**

Bachelor Thesis based on the examination and study regulations
for the Bachelor of Engineering degree program
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner: Prof. Dr. Pawel Buczek
Second examiner: Prof. Dr. Marc Hensel

Day of delivery: 5. October 2021

Mykhailo Svyrydovych

Thema der Arbeit

RPC-basierte plattformübergreifende GUI-Anwendung zum Konfigurieren von Sensormodulen über PROFIBUS und Speichern von Daten

Stichworte

Electron.js, JavaScript, Python, PROFIBUS, gRPC, Sensor, XML

Kurzzusammenfassung

Dieses Dokument beschreibt den Prozess der Implementierung einer komplexen Softwareanwendung zur Kommunikation mit Sensormodulen über das PROFIBUS-Protokoll. Die Anwendung besteht aus mehreren Teilen: Python-basiertes Backend zum Senden von Befehlen und Abrufen von Daten von Sensormodulen, Graphical User Interface zum Konfigurieren der Sensormodule und Graphical User Interface zum Konfigurieren und Steuern der Datenspeicherung. Diese Komponenten kommunizieren über den Remote Procedure Call-Mechanismus unter Verwendung der gRPC-Bibliothek von Google und können auf separate Computer verteilt werden. Die GUI wurde mit JavaScript unter Verwendung des Electron-Frameworks erstellt, das eine Desktop-ähnliche Anwendungserfahrung bietet. Das Programm wurde für die Ansteuerung von Sensormodulen im Solar House an der TU Lübeck entwickelt. Mit den Sensormodulen werden Wetterdaten (Windgeschwindigkeit, Lichtstärke etc.) gesammelt, gespeichert und für weitere Forschungen verwendet.

Mykhailo Svyrydovych

Title of the paper

RPC-based cross-platform GUI application for configuring sensor modules over PROFIBUS and storing data

Keywords

Electron.js, JavaScript, Python, PROFIBUS, gRPC, Sensor, XML

Abstract

This document describes the process of implementing a complex software application for communicating with sensor modules via PROFIBUS protocol. The application consists of several parts: Python-based backend for sending commands and retrieving data from sensor modules, Graphical User Interface for configuring the sensor modules and Graphical User Interface for configuring and controlling data storage. These components communicating via

Remote Procedure Call mechanism using gRPC library from google and can be distributed over separate machines. The GUI has been built with JavaScript using Electron framework that provides desktop-like application experience. The program was developed to be used at TU Lübeck for controlling sensor modules at Solar House. The sensor modules are used to collect weather data(wind speed, light intensity etc.), the data is stored and used in further researches.

Contents

1	Introduction	1
2	Project requirements	4
3	Planning and preparation	6
3.1	Lab hardware setup overview	6
3.1.1	Components	6
3.1.2	Connections	7
3.1.3	Serial Transmission	8
3.2	Lab pre-existing software overview	9
3.2.1	Module configuration software	9
3.2.2	Module readings	9
3.3	PROFIBUS	10
3.3.1	OSI model	10
3.3.2	PROFIBUS Layer 1	11
3.3.3	PROFIBUS Layer 2	11
3.3.4	PROFIBUS Reverse Engineering	13
4	Technologies	14
4.1	Graphical User Interface	14
4.1.1	GUI vs CLI	14
4.1.2	Modern GUI	15
4.2	Node.js and npm	19
4.3	Communication via COM port(RS232)	21
4.4	Python Pipenv environment	21
4.5	gRPC	22
4.5.1	RPC	22
4.5.2	gRPC overview	22
4.5.3	Protocol buffers	23
5	Concept	26
6	Implementation	27
6.1	Project structure	27
6.1.1	General Architecture diagram	27
6.1.2	Storage diagram	29
6.1.3	Directories structure	29

6.2	gRPC server API	30
6.3	Implemented Python modules	31
6.3.1	Config	31
6.3.2	Local Storage	33
6.3.3	Profibus	34
6.4	Telegrams reverse engineering results	34
6.4.1	Requests	34
6.4.2	Responses	41
6.4.3	Findings	42
6.5	UI overview	43
6.5.1	Sensor configuration app	43
6.5.2	Sensor readings app	47
6.6	JavaScript Design Patterns	49
6.6.1	Overview	49
6.6.2	Singleton	50
6.6.3	Callback	51
6.6.4	Bridge	52
6.7	Testing	53
6.7.1	Testing with ISM-111 test module	53
6.7.2	Testing with Mock objects	53
6.8	Setting up and running the project	53
6.8.1	Getting the project	54
6.8.2	Setting up directories	54
6.8.3	Installing Python packages	54
6.8.4	Installing Node packages	54
6.9	Challenges	55
6.10	Results	56
7	Appendices	57
7.1	Appendix A: Installation Guide	57
7.2	Appendix B: Configuration app User Guide	58
7.3	Appendix C: Storage app User Guide	59
7.4	Appendix D: Developer Notes	60

1 Introduction

The application has been developed for TU Lübeck to be used in Solar House lab. It is a software that sends and reads data from sensors' modules and has a GUI¹. The lab contains several sensor modules with connected analogue sensors which are collecting various weather data. Sensors are connected to each other into a cluster and the cluster can be connected to a computer via COM/USB port using RS232². PROFIBUS³ protocol is used for communication between a computer and sensor modules. Up to 126 sensor modules can be connected together, each one connected to 4 sensors. The data coming from sensors should be stored intermediately on the local machine and then transferred to a remote server which has a PostgreSQL⁴ database installed. Sensor modules has a small memory by themselves that is used to contain a configuration data: baud rate⁵, protocol, location, address, sensors names, sensors precision etc. Local computer also should store some configuration data. The application can be divided into 2 parts, each part has its own user interface: Module configuration app and Data saving app. The GUI parts are written in JavaScript, HTML and CSS using Electron.js library. And there is a back-end that sends and reads data from sensors' modules and it is written in Python.

¹"A graphical user interface (GUI) is an interface through which a user interacts with electronic devices such as computers and smartphones through the use of icons, menus and other visual indicators or representations (graphics)" (Graphical User Interface (GUI) by Justin Stoltzfus, <https://www.techopedia.com/definition/5435/graphical-user-interface-gui>)

²RS232 is a hardware interface that is developed for handling data transfer between two devices with a distance limits of 20-40 meters(that depends on the bit rate and cable type) These days it is used to connect PC to an embedded system Axelson (2007) p.43,44

³PROFIBUS (PROcessFIeldBUS) - an open fieldbus system that complies with EN(EuropaNorm) standard, and whose protocol has been specialized for decentralized peripherals (DP) Josef Weigmann (2004) preface

⁴PostgreSQL - open source object-relational database system that extends SQL. It implements many features that are required by SQL standard, but sometimes has different syntax or function. In addition, it has many new features related to data types, security, reliability, performance and extensibility (<https://www.postgresql.org/about/>)

⁵"A baud is defined as one pulse or bit per second, named after the French engineer Baudot." Schleicher M. (2001)

JavaScript and Python parts are communicating via RPC⁶ using gRPC⁷ library. The architecture will be explained later in more details.



Figure 1.1: ISM111 Sensor Module

ISM111⁸ module can measure up to 4 analog inputs and 4 digital I/Os. The module uses RS485⁹ fieldbus interface which can support communication via three protocols: PROFIBUS, ASCII and MODBUS. For the current application PROFIBUS and ASCII had been chosen.

List of sensors currently available at the Solar House:

- 6 pyranometers(light)
- barometer(pressure)
- thermometer(temperature)
- 2 anemometers(wind)

Main subject of this Thesis is a software design and a desktop program development. In addition, an introduction to a hardware and bus communication protocols will be given. The

⁶Remote Procedure Calls(RPC) - is a protocol for inter-process communication that provides the high level communication paradigm used in the operating system. The "RPC protocol enables users to work with remote procedures as if the procedures were local". (<https://www.ibm.com/docs/en/aix/7.1?topic=concepts-remote-procedure-call>)

⁷gRPC is a modern open source high performance Remote Procedure Call (RPC) framework developed by Google that can run in any environment(<https://grpc.io/about/>)

⁸Intelligent Sensor Module produced by Gantner Instruments

⁹RS485 - is an interface that allows communication over longer distances and higher speeds than RS232. Axelson (2007) p.79

1 Introduction

software is using PROFIBUS telegrams for communication with Sensor Modules, so some preparation and research has been done on that side.

2 Project requirements

- Business requirements
 - Separate program with GUI for reading/writing data into a sensor module
 - Separate program with GUI for controlling storage of the sensors' readings obtained from a sensor module(Intermediate local storage and remote database)
 - Common configuration files that contain module settings
- Architectural and Design requirements
 - A user can see a window with navigation buttons, module address and connection status
 - A user can obtain current module configuration from a module
 - A user can obtain current sensor readings from a module
 - A user can change module configuration
 - A user can connect/disconnect module on serial port and specify port settings
 - A user can store configuration in xml file
 - A user is notified in case there is a difference between xml file and current module actual settings
 - A user can initiate storing data to a local storage or remote database
 - A user can add new sensors to a database
 - A user gets a notification when any error occurs
 - A user can set readings interval
 - A user can start/stop recording data to local/remote storage
- System and Integration requirements
 - Configuration application

- * Navbar contains following parts: Module address (1-126), Port status with connect/disconnect buttons, navigation buttons for Module Overview, Measurements, Variable Configuration, Device Configuration.
- * Module address can be set from 1 to 126 to connect to corresponding device
- * When user clicks 'Connect' a dialog should be shown where the user can specify Port Name and Baud rate. Previous data should be stored and auto filled.
- * Notifications and errors should be displayed as toast messages.
- * Overview tab displays module info: Vendor, Module type, HW and SW versions, Location, S/N number, Number of channels.
- * Measurements tab displays a table with sensors' names and readings. It should be refreshed with an interval set by user.
- * Variable configuration displays a table with all sensors for the current module with all possible settings. User can change these settings and send to the module and save in a local xml configuration file. The difference between these 2 should be notified to the user.
- * Device configuration displays following: Location, User, Data and time of last change, address, baud rate, char format.
- * Initial setup in case config folder is empty
- Storage application
 - * Reading/saving interval can be configured
 - * Readings stored first locally in txt files
 - * Txt files are saved in folders corresponding to year and month (i.e. 2021/07/2021-07-01.txt)
 - * Readings can be set to remote database
 - * User Interface to configure connection to remote database

3 Planning and preparation

3.1 Lab hardware setup overview

3.1.1 Components

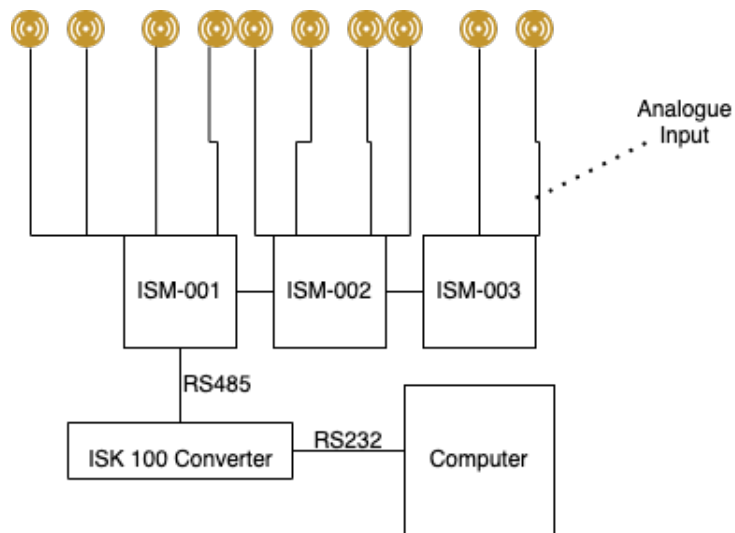
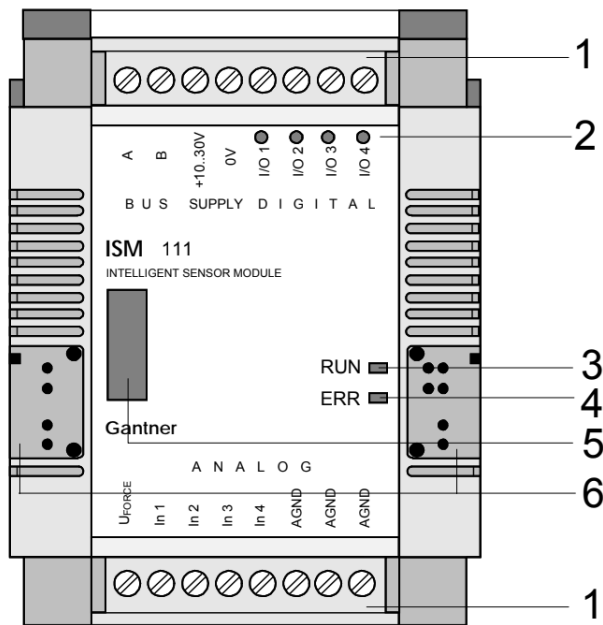


Figure 3.1: Lab hardware setup

There are 3 Sensors' Modules: ISM-001,002,003. Each of them has 4 analogue inputs and is currently connected to 10 analogue sensors.



1. Terminal block
2. LEDs I/O 1-4
3. LEDs LED RUN
4. LEDs LED ERR
5. Infrared window
6. Module binder connection

Figure 3.2: ISM111(from GE-ISM111/5097d documentation p.22)

3.1.2 Connections

Three modules are connected together via Module binder connection(6). Each module must have a unique address from 001 to 127. Next, modules are connected to ICK100 converter that converts RS485 to RS232¹. ISK 100 Converter is connected to a computer with COM/USB port.

¹both RS-485 and RS-232 are industrial specifications that define the electrical interface and physical layer for point-to-point communication of electrical devices. The RS-485 standard allows for long cabling distances in

3.1.3 Serial Transmission

Serial transmission is used for longer routes and has reduced susceptibility to interference.

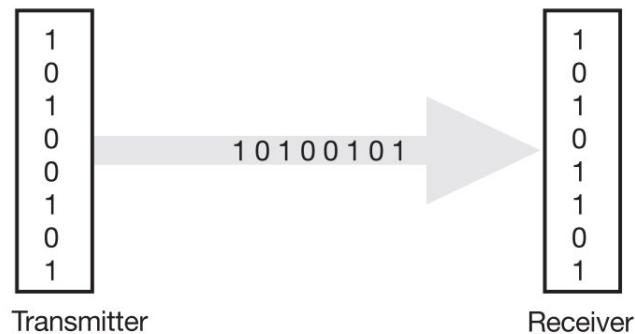


Figure 3.3: Serial transmission (Schleicher M.)

In this kind of transmission a data bits are sent one after the other, e.g. over a two-wire circuit. This method is much slower than parallel transmission, but it is used in automation engineering because of its suitability for bus systems.²

RS232

The RS232 standard is a serial interface that is a low-performance character interface. It converts digital data from parallel format to serial format for sending between devices. Originally, it was used to connect modems to computer terminals. It was established in 1960 by the Electronic Industries Association and published as 'Recommended Standard 232'. This standard is widely used, but there are some limitations with regard to the maximum bit rate (20 Kbits/sec), transmission distance and the use of the common signal ground. The RS232 specifies the maximum duration of the transition time from High to Low and vice versa. This limits the amount of stray capacitance allowed in a cable, because the transition time is a function of capacitance. Also the standard specifies the maximum cable capacitance - 2500 picofarads. This means that the maximum cable length can be 62m. In practice such long distances are not recommended.

In RS232 interface all the control and data signals are referenced to a common signal ground.

electrically noisy environments and can support multiple devices on the same bus. RS-232 is used for a shorter distances(up to 15 m) and connects 2 devices. <https://www.seeedstudio.com/blog/2019/12/06/what-is-rs485-and-its-difference-between-rs232/>

²Schleicher M. (2001) p.13, 14

It leads to a possibility of transmission errors in case of a significant difference in ground potential between two ends of the cable. This is another reason to keep the cables short.³

RS485

This standard offers longer distances and higher speeds than RS232. In addition, it is not limited to two devices. It can connect up to 256 computers with a single pair of wires. The interface is defined by *TIA-485-A: Electrical Characteristics of generators and receivers for use in balanced digital multi-point systems*. The advantages are: low cost, networking ability, long links, speed.⁴

3.2 Lab pre-existing software overview

3.2.1 Module configuration software

Lab computer had an old Configuration program written in Delphi and running on Windows XP. The code cannot be reverse engineered, but module commands could be obtained by listening on the COM port. The new software that is developed as part of the thesis has the main functionality of the old application. The old program contained following screens:

- "Info" - general module information
- "Measurements" - readings from connected sensors
- "Variables" - settings for each connected sensor(precision, type, units etc.)
- "Module Configuration" - username, physical location, bus address, baud rate etc.

The main problem of this old software is that it was written for an old Windows version. And it is not possible to use on Linux, Mac and newer Windows versions. The other problem is that it is not open source and cannot be adjusted and modified for future needs.

3.2.2 Module readings

There were some Python scripts that are reading data from modules and sending it to a Postgres database. Python is using "pyserial" library for communications over COM port with modules. The program obtains data from a sensor by sending a special ASCII command that contains module and variable addresses. There was no user interface: scripts could be run via command

³Richard W. D. Nickalls (1995)

⁴Axelson (2007)

line and could not be controlled once started

ASCII command format:

$\$ aa R kk < cr >$

where aa - module address,

R - command to read a value,

kk - sensor address

There is a very limited set of ASCII commands and to access more functions of the module PROFIBUS protocol should be used instead.

3.3 PROFIBUS

3.3.1 OSI model

PROFIBUS is a standard for a fieldbus⁵ communication developed in 1989 by German department of education and research and then used by Siemens. It makes use of OSI(Open Systems Interconnection) model.

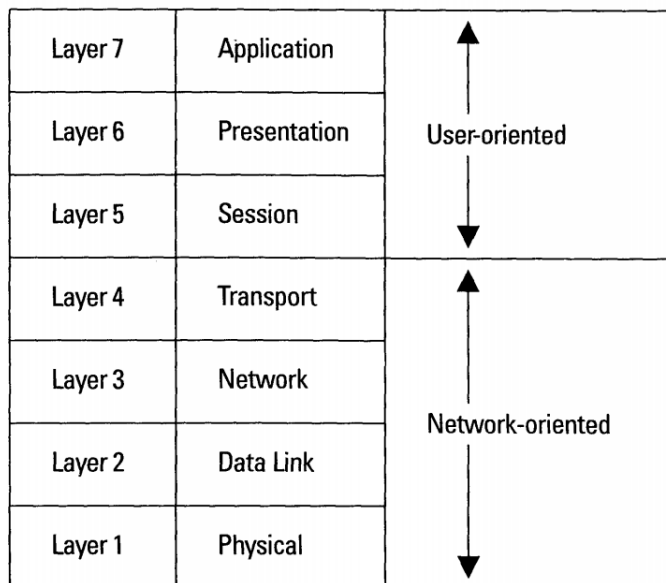


Figure 3.4: ISO/OSI model for communication standards Josef Weigmann (2004)p.13

⁵Fieldbus - a name for a family of industrial computer networks

Layers 1 and 2 and, if necessary, layer 7 are implemented for the PROFIBUS protocol. PROFIBUS can be divided into 3 versions: DP, FMS and PA. The Thesis equipment uses PROFIBUS-DP(Decentralised Peripherals). This version has 2 OSI Layers: 1 and 2 and User Interface. Other Layers are not implemented in DP-version. This type of the protocol provides a high-speed transmission. It is designed for communication between a programmable controller and distributed I/O devices at the field level.

3.3.2 PROFIBUS Layer 1

On Layer 1 PROFIBUS uses shielded twisted pair cables with transmission speeds of 9.6 kbit/s to 12 kbit/s. It uses RS 485 transmission procedure⁶. RS 485 is a standard that defines electrical characteristics of drivers and receiver devices in serial communication systems⁷. It is based on semi-duplex⁸, asynchronous, gap-free synchronization. Data transmission is done in 11-bit character frames in NRZ code⁹.

3.3.3 PROFIBUS Layer 2

In PROFIBUS Layer 2 is called FDL Layer(Fieldbus Data Link). Telegram¹⁰ formats are providing a high level of transmission security. The hamming distance¹¹ of call telegrams is 4. It means that 3 bit errors can be detected. DP version supports following transmission: SRD(Send and Request Data with Acknowledge) and SDN(Send Data without Acknowledge).¹²

Telegram formats



Figure 3.5: Telegram with fixed information field without data field Manual p.130

⁶Josef Weigmann (2004)p.15,16

⁷<https://en.wikipedia.org/wiki/RS-485>

⁸a device can send and receive data but not in the same time

⁹a binary code where ones are represented by a positive voltage and zeros are represented by negative voltage

¹⁰Telegram is a defined format/framework of the transmitted message. Schleicher M. (2001)

¹¹number of positions at which the corresponding symbols are different

¹²Josef Weigmann (2004)p.23

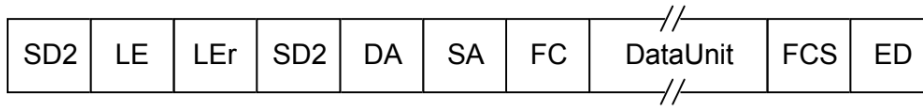


Figure 3.6: Telegram with variable length information field with data field Manual p.130

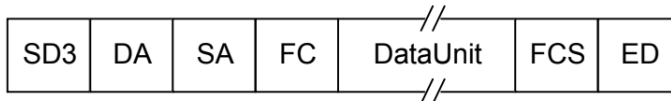


Figure 3.7: Telegram with fixed information field with data field Manual p.130

Telegram formats are distinguished by Start-Delimiter(SD): SD1, SD2 and SD3. Receive and transmit telegrams may have different formats and do not need to follow the same SD. In addition, there is a telegram that consists only of one symbol "SC" and is a response telegram that can have a positive or negative meaning.

SD(1 byte) - is a start delimiter in a telegram.

Format	Transmit	Receive	Data field length
SD1	Hex 10	Hex 10	0
SD2	Hex 68	Hex 68	1 ... 246 (32)
SD3	Hex A2	Hex A2	8

Figure 3.8: SD in PROFIBUS Manual p.131

LE(1 byte) - length, used for telegrams with variable data field. It is the length from DA to the end of Data Unit field.

LEr(1 byte) - length repeated, used for data safety control.

DA(1 byte) - destination address of the communication partner(from 0 to 127).

SA(1 byte) - source address(from 0 to 127).

FC(1 byte) - frame control. This byte contains a telegram type, a station type, a way of data transfer.

ReqDataUnit(0..n byte) - data field that is being sent to a communication partner with address DA.

ResDataUnit(0..n byte) - received data field from a communication partner.

FCS(1 byte) - Frame-Check-Sequence, checksum of the telegram. It is calculated as ASCII value of fields from DA to DataUnit modulo 256.

ED(1 byte) - End Delimiter, in PROFIBUS protocol it is hex 16.

3.3.4 PROFIBUS Reverse Engineering

As Module manual does not contain PROFIBUS commands for module configuration, they may to be reverse engineered. It can be done by using an old software for module configuration together with any COM monitoring tool. The process including setting some new data in the software, sending it to a module and monitoring which bytes are appearing on the COM port to get PROFIBUS telegrams from it."Reverse Engineering is a process of measuring, analyzing and testing to reconstruct the mirror image of an object or retrieve past event."¹³. A reverse engineer has to have an understanding of the original functionality and skills to reproduce its details. Nowadays, reverse engineering is one of the primary methodologies used in many industries, manufacturers all over the world are practising this technique in product development. The main question of the reverse engineering is how a part was made and not why this part is so designed.¹⁴

¹³Wang (2011)

¹⁴Wang (2011) p.1

4 Technologies

4.1 Graphical User Interface

4.1.1 GUI vs CLI

Graphical User Interface(GUI) is an important part of a desktop application. Generally modern applications can have two types of user interfaces: GUI and CLI(Command Line Interface). Each of them has own advantages and disadvantages¹.

Advantages of CLI:

- Speed. Using keyboard only can speed up the input performance
- Computer resources. CLI requires much less PC resources than a modern GUI
- Stability. It does not change so much as GUI, new commands can be introduces, but the original commands often remain the same.

Disadvantages of CLI:

- Not user friendly. It is much harder to familiarize and memorize commands.
- Harder to do multitasking. CLI does not offer the same ease and ability as GUI to view multiple things.
- Higher risk of strain. CLI can have more strain on a user's vision. In addition, risk of Carpal Tunnel Syndrome is higher as user has to type commands on a keyboard all the time.

Advantages of GUI:

- User friendly. Learning goes much faster than with CLI

¹Command line vs. GUI <https://www.computerhope.com/issues/ch000619.htm>

- Multitasking. Desktop GUI provides windows that allow user to view and control several programs
- Less strain. Using mouse in addition to a keyboard reduces strain. Also GUI has more colors and is more visually appealing, leading to a reduction in visual strain

Disadvantages of GUI:

- Speed. Mouse usage is reducing operating speed compared to CLI
- Computer resources. GUI requires much more computer resources than CLI.
- Diversity. Different programs can have much different GUIs. So user have to deal with different patterns to perform tasks.

Considering all of these and the fact that the program may be used in future by personal not familiar with CLI interfaces, it was decided to use a GUI.

4.1.2 Modern GUI

Ten Principles for Good GUI Design

Leslie Cortes in his work called "Designing a Graphical User Interface" stated 10 principles of good GUI design². It was written in 1997 but remains undeniable also today. Here is a short summary of his findings:

1. User must be able to understand a widget's behavior from its visual properties: every part of UI should behave in a consistent way, i.e. if one button responds to a single click than all such buttons should behave the same.
2. User must be able to understand the behavior of the program with help of knowledge gained from other programs: it means consistency in abstractions, menu placements, icons, toolbars etc.
3. Every warning or error dialog is an opportunity to improve the GUI: "prevent, don't complain!"
4. Adequate user feedback should be provided: user should see that his action did something, i.e. click button animation.

²Leslie (1997)

5. Safe environment for exploration: encourage user to safely explore the UI, e.g. with undo/redo option.
6. Self-evidence: self-evident interface relieves a user from reading huge manuals, can be achieved by widget arrangement, labels etc.
7. Use sound, color, animation and multimedia sparingly.
8. User should be able to customise the working environment.
9. Modal behaviors should be avoided: user should not be forced to perform tasks in a specific order. Example of accepted modal behavior: in Paint program when a user picks Brush, he can paint, when a user picks Text, he can type text. So selecting a widget alters the subsequent function of the program and therefore results in the modal behavior. It is accepted, because it is based on a real world analogy.
10. Interface should be designed in a such way that users can accomplish tasks while being minimally aware of the interface itself: principle of transparency.

GUI libraries

GUI programs became really popular in the past years. In 90s and 2000s the main tools commonly used for GUI application development were: Visual Basic, Delphi and C++³. Nowadays, web interfaces are becoming the most popular. They are written in HTML, CSS and JavaScript and require a modern web browser to run. On the other hand, there are some libraries and frameworks apart from web interfaces that is still used for desktop programs. It was decided not to use the web GUI in the project, because it is assumed that the application will have only one user at a time and does not require to have GUI opened on several machines. Moreover, no need to support several browsers and support the application code for the future browser versions, and no need to have a browser installed at all. The modern cross-platform GUI libraries and frameworks are: Qt, Java FX, GTK and others.

Advantages of Web-GUI

- Maximum platform independence: can run on any device/OS that has a browser installed and JavaScript support enabled
- Development speed: HTML, CSS and JavaScript allow to implement UI elements fast

³Leslie (1997)

- Knowledge base: lots of documentation, tutorials and discussions available online
- Future perspective: web-GUI are at the leading place and will stay for many years, so there will be people available to support/update the code

Disadvantages of Web-GUI

- Performance: less than classical compiled programs.
- Must be compatible with different browser versions, also in future: in this aspect it is harder to develop and maintain a web-GUI
- Design differs from native desktop applications: i.e web-apps do not have same toolbars and dropdown menus as desktop window applications
- Security limitations: due to a security policy of web browsers, there are limits to what hardware/software features of the computer the app can access

Advantages of Desktop-GUI

- Performance: compiled programs give the best performance then scripts running in a browser and require less PC resources.
- Native design: App that is developed for a particular OS uses standard navbars and menus, so styling looks familiar to a user.
- There is no dependency on 3rd party web browsers: less issues to deal with.

Disadvantages of Desktop-GUI

- Less popular: less developers working with desktop-GUI today, so web-GUI are growing faster. It means the chance to find someone to support the code for classical GUI in future is less. Not so many libraries to choose from(that is still being developed and not abandoned).
- Development speed: usually less as it is done in C++. And it usually takes years to develop a program.
- Licensing: Popular frameworks, like Qt are free only for open-source projects, otherwise a license must be purchased.

Electron.js

A hybrid solution has been chosen for the project - Electron.js

This framework appeared in 2013 when Node.js⁴ was becoming popular. It is used to develop desktop programs with JavaScript. Electron allows to create desktop apps for Windows, Linux and Mac with the same code used for web apps. It means a lot for code and skills reusability and use huge Node.js ecosystem. Electron was created by GitHub company, they used it for their text editor called Atom. After official release in November, 2013 it became very popular and a huge number of startups and large businesses started to use it for desktop apps development. Electron is combined with Node.js through Chromium's⁵ content API and uses Node.js's node_bindings. Electron use separate JavaScript contexts - one is for a back-end process that starts the app window(main process), one is for each app window(renderer process).⁶

Electron features⁷:

- Create multiple windows, each with own JavaScript context
- Create tray apps
- Control and tracking PC power management - prevent PC from going into power saving mode etc.
- Integrating with OS features
- Creating OS specific menus and menu items
- Global keyboard shortcuts
- App updates
- Reporting crashes
- Customizing dock menu icons
- OS notifications
- Creating setup installers

⁴Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.(wikipedia)

⁵Chromium - is a free and open-source codebase for a web browser, principally developed and maintained by Google(Wikipedia)

⁶Jensen (2017) p.4, 17. 18

⁷Jensen (2017) p.23,24

Apps created with Electron:

- Visual Studio Code (VS Code)
- Slack
- Tusk
- Mailspring
- Skype
- Discord
- Streamlabs OBS
- WordPress Desktop
- WhatsApp Desktop
- Atom

4.2 Node.js and npm

Electron framework uses Node.js⁸. It is an open source JavaScript runtime environment that inherits event-driven model. Node.js uses the same engine like Chrome browser - V8, this allows it to be very performant. The important part that comes with Node is npm⁹ - the world's largest software registry. It consists of three parts: the website, the Command Line Interface(CLI), the registry. Npm registry contains packages. A package is a file or a directory that is described by package.json file.

Package formats:

- A folder containing a program described by a package.json file
- A gzipped tarball¹⁰
- A URL that resolves to a gzipped tarball

⁸<https://nodejs.org>

⁹<https://www.npmjs.com>

¹⁰On Unix systems files can be packed into a tar file that can be compressed with gzip

- A `<name>@<version>` that is published on the registry with a URL
- A `<name>@<tag>` that points to `<name>@<version>`
- A `<name>` that has a latest tag satisfying `<name>@<tag>` pair
- A git url that, when cloned, results in a folder that contains a program code.

A module is a separate file(JavaScript) or directory(containing package.json with 'main' field) located in `node_modules` folder and can be loaded into JS code using `require()` function.

Npm packages can be searched on <https://npmjs.io> website. Required package can be installed locally or globally. By default all packages are installed locally(scoped to one local project). Downloading and installing a package globally allows to use this package in any project on the computer.

List of used npm packages and modules:

- `electron` - framework for developing cross-platform desktop applications using JavaScript, HTML and CSS
- `@electron/remote` - electron module that bridges JavaScript objects from the main process to the render process(used for calling dialog windows)
- `@fortawesome/fontawesome-free` - free css icons used for navbar buttons
- `@grpc/grpc-js` - JavaScript gRPC Client
- `@grpc/proto-loader` - utility package for loading .proto files for use with gRPC
- `google-protobuf` - Google protocol buffers for gRPC
- `async` - provides a set of functions for working with asynchronous JavaScript
- `lodash` - JavaScript utility library
- `minimist` - argument parser
- `tcp-port-used` - to check if a TCP port is currently in use

4.3 Communication via COM port(RS232)

As there were some existing Python script for communication with the ISM device, it was decided to continue using them and to wright an additional code to extend their functionality. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics¹¹. Built-in data structures, dynamic binding and dynamic typing makes it popular for rapid application development. It has a simple syntax and it makes it easy to learn. One of the key library used in the project is pySerial. This module encapsulates the access for the serial port.

Features of pySerial module¹²:

- Same class based interface on all supported platforms
- Access to the port settings through Python properties
- Support for different byte sizes, stop bits, parity and flow control with RTS/CTS and/or Xon/Xoff
- Working with or without receive timeout
- File like API with “read” and “write” (“readline” etc. also supported)
- The files in the package are 100% pure Python
- The port is set up for binary transmission. No NULL byte stripping, CR-LF translation etc. (which are many times enabled for POSIX.) This makes this module universally useful
- Compatible with io library

4.4 Python Pipenv environment

Pipenv is a tool that provides a virtual enviroment for a Python project. It helps to manage project dependencies on 3rd party modules. Installing and uninstalling is done through 'Pipfile' file.

Pipenv can be installed with a following command:

¹¹<https://www.python.org/doc/essays/blurb/>

¹²<https://pyserial.readthedocs.io/en/latest/pyserial.html>

```
1 sudo -H pip install pipenv
```

It will install Pipenv with Home variable set to home directory of current user (root by default).

Packages can be installed with:

```
1 pipenv install <name>
```

The main packages are:

- grpcio - for Remote Procedure Calls
- grpcio-tools
- pyserial - for communication over serial port

Others can be found in the 'Pipfile'.

4.5 gRPC

4.5.1 RPC

The Remote Procedure Call (RPC) is a mechanism that allows to implement client-server applications in a simple way. It keeps details of network communications out of the application code. Each side behaves as much as possible if it would be normal traditional application. To keep this illusion there is some hidden code that handles all the networking.¹³

RPC technology is much older than the Web. A credit for creating a generic formal mechanism used to call procedures and return results over a network belongs to Sun Microsystems company. It was fitting very well with the procedure approach that dominated in 90s.¹⁴

4.5.2 gRPC overview

RPC is used to connect Python back-end scripts with Electron.js front-end GUI. With this technique these two parts can even be physically located on two different machines. To achieve RPC communication gRPC¹⁵ library developed by Google is used. gRPC is used by such companies: Netflix, CISCO, Jupiter Networks, Square, Core OS. Originally gRPC was developed for internal usage. It was called Stubby and was used to connect the large number of microservices that were running across various Google data centers. In March 2015, google

¹³Ward Rosenberry (1995) p.1

¹⁴Simon St. Laurent (2001) p.10

¹⁵A high performance, open source universal RPC framework (<https://grpc.io/>)

released the next version and made it open source. From that time many organizations are using gRPC. gRPC operates a client-server architecture. A client program can call a method/function on a server program that can be on a different machine as if it were a local object. gRPC is based around idea of specifying a service that contains all possible methods to be called, their parameters and their return values. It is located on the server side. And on the client side there is a stub that provides the same methods as the server.

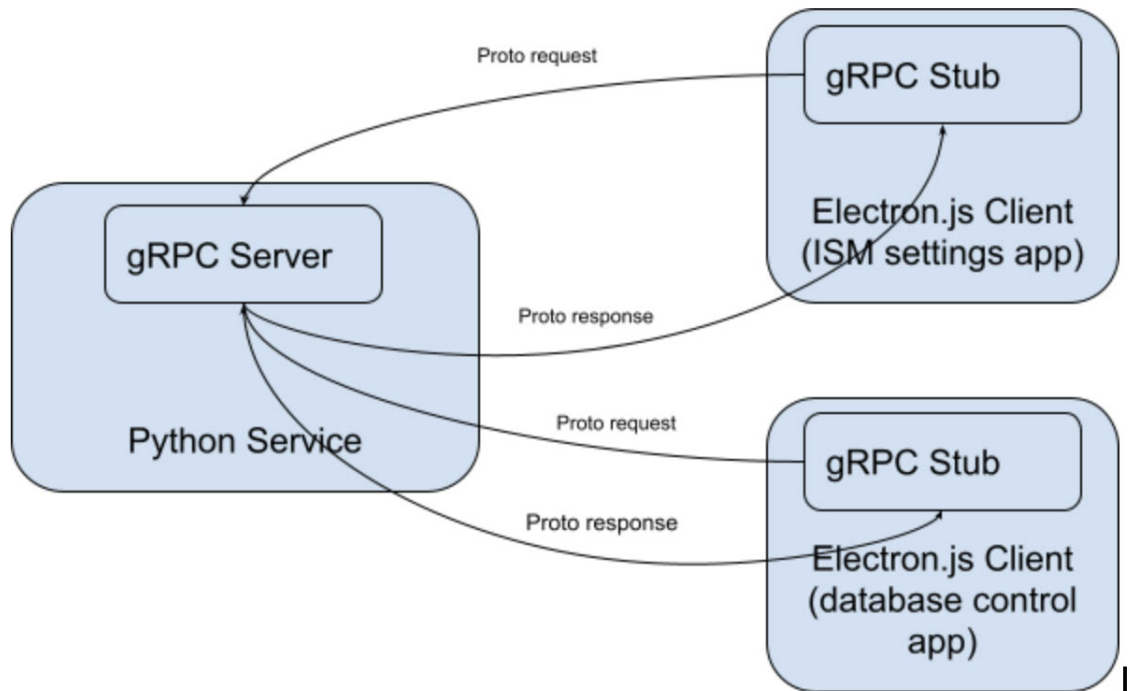


Figure 4.1: gRPC overview

4.5.3 Protocol buffers

Protocol buffers is a mechanism used by gRPC for serializing data. They are defined in files with .proto extension. The data is structured in messages, each message contains fields in key-value pairs.

Here is an example of a protocol buffer message form a project's proto file:

```
1 message DeviceConfig {
```

```
2 int32 deviceAddress = 1;
3 string location = 2;
4 string user = 3;
5 string date = 4;
6 string time = 5;
7 optional int32 newAddress = 6;
8 }
```

Besides that proto files contain services. A service should contain methods that will be remotely called by the client. Here is an example of a service from the project:

```
1 service DeviceCommunication {
2   rpc ConnectToSerialPort (PortConfig) returns (Answer) {}
3   rpc DisconnectPort (PortMsg) returns (Answer) {}
4
5   rpc GetDeviceInformation (DeviceAddress) returns (DeviceInfo) {}
6   rpc GetDeviceConfig (DeviceAddress) returns (DeviceConfig) {}
7   rpc GetPortInformation (PortMsg) returns (PortInfo) {}
8   rpc GetMeasurements(MeasurementsMsg) returns (Measurements) {}
9
10  rpc SetDeviceConfig(DeviceConfig) returns (Answer) {}
11  rpc SetVariableConfig(VariableConfigRequest) returns (Answer) {}
12
13  rpc StoreVariablesConfig(VariablesConfigXml) returns (Answer) {}
14  rpc ReadVariablesConfig(DeviceAddress)
15  returns (VariablesConfigXml) {}
16  rpc ReadGlobalConfig(GlobalConfigXml) returns (GlobalConfigXml) {}
17  rpc WriteGlobalConfig(GlobalConfigXml) returns (GlobalConfigXml) {}
18
19  rpc StoreOverviewConfig(OverviewConfig) returns (Answer) {}
20 }
```

There are 4 kinds of service methods:

- Unary RPC - client sends a single request to the server and gets back a single response.

```
1   rpc SayHello(HelloRequest) returns (HelloResponse);
```

- Server streaming RPC - client sends a request to the server and gets a stream to read back. It reads from the stream until there are no more messages.

```
1   rpc LotsOfReplies(HelloRequest)
2   returns (stream HelloResponse);
```

- Client streaming RPC - client writes a message sequence and sends it as a stream to the server. Then it waits until the server finishes reading.

```
1   rpc LotsOfGreetings(stream HelloRequest)
2     returns (HelloResponse);
```

- Bidirectional RPC streaming - both streams operate independently

```
1   rpc BidiHello(stream HelloRequest)
2     returns (stream HelloResponse);
```

Once proto file is filled with services and messages it can be used to generate actual code for client and server. In our case, 2 python files are generated.

5 Concept

Background

Solar House at TU Lübeck has a set of ISM-111 modules with connected meteo sensors. These are connected to a Windows PC via Serial port. There is an old software used to configure the modules called "Combilog" and some Python scripts that communicate with the modules over ASCII protocol. ISM modules can communicate with a computer via ASCII, PROFIBUS or MODBUS protocols. ASCII commands are very limited, PROFIBUS and MODBUS provide more control over the module.

Purpose

The project addresses a data collection automation problem. There is a need of a system that can communicate with meteo sensors. It should be written in a modern language, using modern technologies and be extendable and ready to be used on different Operating Systems. Possible extensions can include various data processing and sending the data to other systems. It should have modern and clear Graphical User Interface, so even a person with non-technical background can use it. The old software consisted of a closed non-expandable program for configuration the modules and some Python script without GUI. The aim of the project is to have a centralized system (but non-monolithic in the same time) with opened code so it can be extended in future.

Main activities

- Researching such subjects as: PROFIBUS, Electron.js, gRPC, Python, Serial Port reverse engineering
- Writing code for gRPC communication, GUI, communication with ISM modules
- COM Port Reverse engineering to find out PROFIBUS telegrams structure for particular commands
- Running, adjusting and testing the developed system on a real set up at Solar House

6 Implementation

6.1 Project structure

6.1.1 General Architecture diagram

The application can be roughly divided into a front-end and back-end parts. On the front-end there are two separate Electron.js apps with some shared code. These apps provide a User Interface for the whole application - one is for configuring Sensor Modules and the other one is for obtaining and storing the sensors' reading. Front-end communicates with Python back-end via gRPC (and gRPC server itself is a Python process). There are three main back-end parts: Communication with sensor modules over PROFIBUS, Reading/Writing Xml config files and Communication with Database.

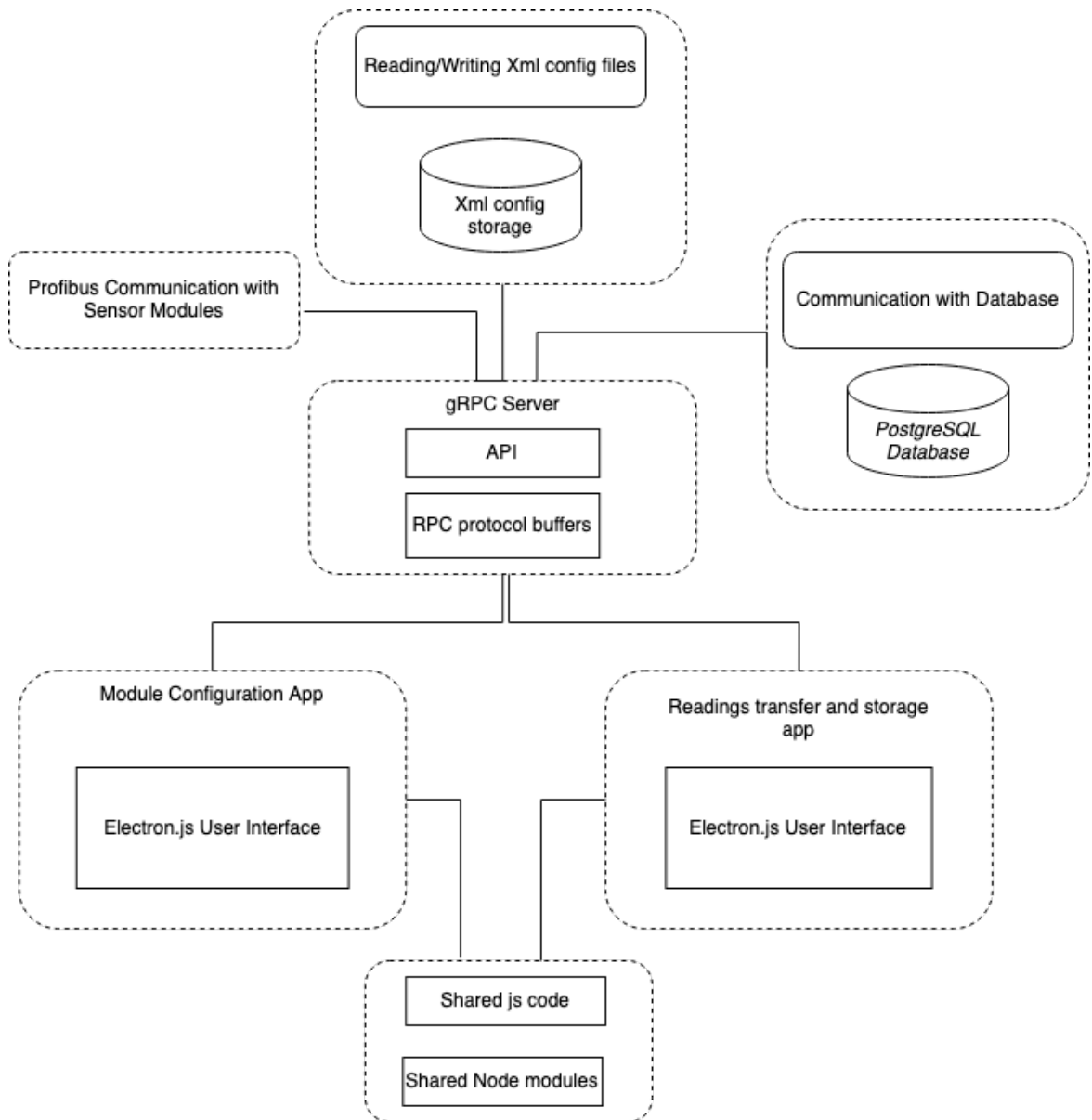


Figure 6.1: General Architecture diagram

6.1.2 Storage diagram

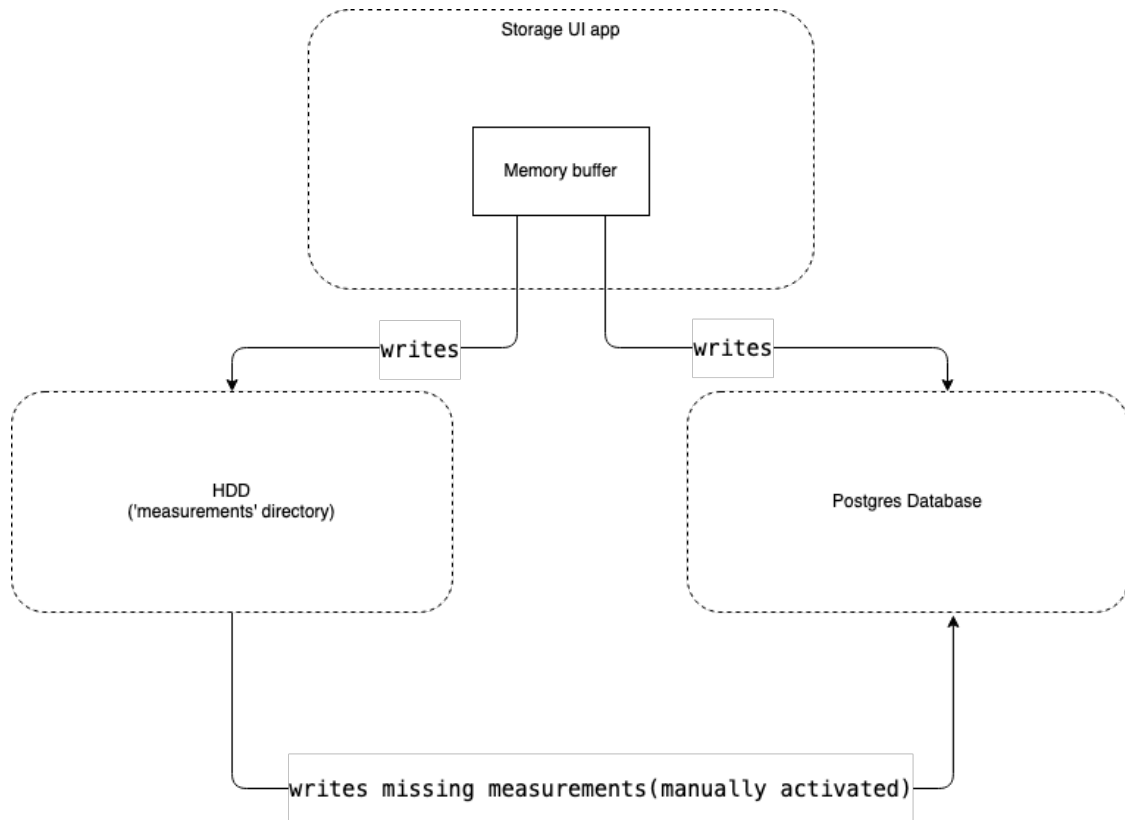


Figure 6.2: Storage diagram

Measurement reading process is triggered from UI application at some configurable time interval. This interval can be set by User in UI or in xml config file. First, readings are stored into a memory buffer. When the buffer is full, the readings are transferred into a txt file and into a database. Both of these options can be enabled/disabled.

6.1.3 Directories structure

- config (storing xml configuration files)
- measurements (storing measurements locally, in txt format)
- src
 - config (code for reading/writing into xml config files)

- db (database communication related code)
- localStorage (code for storing measurements)
- postgresql (database communication related code)
- profibus (module communication)
- rpc (protocol buffers)
- test (various python tests and mock objects)
- UI (Electron.js)
 - database-ui
 - sensors-ui
 - shared
 - node_modules (generated with "npm install")
- utility

6.2 gRPC server API

The entry point is implemented with `api.py` file. This program is started when one of the UI apps is been opened. The gRPC server is started on localhost port 50051(default gRPC port). The `api.py` implements a number of functions that may be called over RPC:

- `ConnectToSerialPort` - establishes connection that can be reused by other calls
- `DisconnectPort`
- `GetDeviceInformation` - returns ISM module information such as id, status etc.
- `GetPortInformation` - if port is currently opened or no(checks if port is busy)
- `GetMeasurements` - gets readings from all sensors of the module, adds them to measurements buffer. In case if buffer is full - writes data to local txt file, send to remote database and clears the buffer
- `GetDeviceConfig` - gets location, user, date and time from the module
- `SetDeviceConfig`
- `GetVariableConfig` - gets configuration for sensors: type, name, format, length, precision and unit

- SetVariableConfig - store in the module memory
- StoreVariableConfig - store in xml config file, on computer.
- ReadVariablesConfig - from xml file
- WriteGlobalConfig - to xml file
- StoreOverviewConfig - stores ISM module info into xml file
- CheckConfig - checks if xml config file is exist for selected ISM module

Also api.py can be switched from normal to test mode. Test mode uses mocked devices with fake data instead of real serial port communication. This function can be called from Electron.js UI apps like:

```
1 client.ConnectToSerialPort(...)
```

6.3 Implemented Python modules

6.3.1 Config

This Python module is responsible for storing ISM module configuration locally, in xml files. It consists of two classes: GlobalXmlConfig and XmlConfig. First operates with one config file and the second one assumes a separate file for each module that contains more specific configuration. To create and parse xml files it uses minidom package that is a part of the Python standard library.

Here is an example of global config xml file:

```
1 <?xml version="1.0" ?>
2 <config>
3     <readingIntervalMs>1000</readingIntervalMs>
4     <database>
5         <name>zeus-db</name>
6         <host>localhost</host>
7         <port>5433</port>
8         <username>postgres</username>
9         <password></password>
10    </database>
11    <modules>
12        <module>
```

```
13     <address>7</address>
14   </module>
15 </modules>
16 </config>
```

It is used by Database UI app and contains reading interval in milliseconds, an array of modules with addresses and database credentials.

Password is stored in encrypted format, so it is not exposed to xml config viewers. A Python library named cryptography is used for the encryption and decryption.

In addition, a separate config file is created for each module that is connected to a computer. Here is an example of such file:

```
1 <?xml version="1.0" ?>
2 <module>
3   <address>6</address>
4   <vendor>Gantner</vendor>
5   <module-type>ISM-111</module-type>
6   <hw-version>M3.20</hw-version>
7   <sw-version>U5.41</sw-version>
8   <location>Hamburg</location>
9   <sn>077379</sn>
10  <channels>4</channels>
11  <variables>
12    <variable>
13      <address>1</address>
14      <name>Pyranometer1</name>
15      <globalIndex>21</globalIndex>
16      <type>AI</type>
17      <format>Real</format>
18      <length>8</length>
19      <unit>W/m2</unit>
20      <precision>5</precision>
21    </variable>
22    <variable>
23      <address>2</address>
24      <name>Pyranometer2</name>
25      <globalIndex>22</globalIndex>
26      <type>AI</type>
27      <format>Real</format>
```

```
28     <length>8</length>
29     <unit>W/m2</unit>
30     <precision>6</precision>
31 </variable>
32 <variable>
33     <address>3</address>
34     <name>Virtual1</name>
35     <globalIndex>23</globalIndex>
36     <type>AI</type>
37     <format>Real</format>
38     <length>8</length>
39     <unit> V</unit>
40     <precision>4</precision>
41 </variable>
42 <variable>
43     <address>4</address>
44     <name>Virtual2</name>
45     <globalIndex>24</globalIndex>
46     <type>AI</type>
47     <format>Real</format>
48     <length>8</length>
49     <unit> V</unit>
50     <precision>4</precision>
51 </variable>
52 </variables>
53 </module>
```

6.3.2 Local Storage

This module is responsible for storing readings from connected sensors to a local computer. It is an intermediate step before sending the data to a remote database. It allows to be sure that data is stored even if the database is not available. There are 2 formats supported: xml and txt. Text format is the simplest one and takes less space. MeasurementsTxt class stores the data in a text files. Each file has a name that contains a date, so each day a new file is created. First, readings are written into a Python memory buffer every time GetMeasurements api call is done. When buffer is full, it is written into the txt file.

6.3.3 Profibus

This module was extended with some classes required for communication with the ISM via PROFIBUS. It contains various types of telegrams that can be send and received. The telegrams are explained in "Reverse Engineering" section in more detail.

6.4 Telegrams reverse engineering results

6.4.1 Requests

Module configuration edit

Implemented in profibus/config_edit_request.py

This telegram allows to set the following parameters:

- ISM module location - e.g. room number or name (not more that 20 bytes)
- Username - not more that 20 bytes
- Date and time of last editing

Byte number	Description	Byte value
1	Start Delimiter, const	0x68
2	Length, const	0x46
3	Length, const	0x46
4	Start Delimiter, const	0x68
5	Module Address	0x80 + actual address(1-127)
6	Source Address, const	0x80
7	Frame Control, const	0x4C
8	DSAP, const	0x28
9	SSAP, const	0x00
10	Command, const	0x64
[11:30]	Module Location string	not more than 20 chars
[31:38]	Unknown bytes, const	0x00 0x32 0x00 0x04 0x00 0x6F 0x17 0x70
[39:58]	Username	not more than 20 chars
[59:68]	Date and time	yymmddHHMM
[69:74]	Unknown bytes	0x00 0x01 0x00 0x01 0x00 0x00
75	Checksum	[DA+SA+FC+DataUnit] mod 256
76	End byte	0x16

Notes:

- Bytes marked "const" should not be modified
- DSAP - Destination Service Access Point
- SSAP - Source Service Access Point
- Fields should be always zero padded if their length is less than maximum

Sensor configuration edit

Implemented in profibus/variable_edit_request.py

This telegram allows to set the following parameters:

- Type

- Name
- Additional Name
- Format
- Length
- Precision(number of decimals)
- Unit

Byte number	Description	Byte value
1	Start Delimiter, const	0x68
2	Length, const	0x46
3	Length, const	0x46
4	Start Delimiter, const	0x68
5	Module Address	0x80 + actual address(1-127)
6	Source Address, const	0x80
7	Frame Control, const	0x4C
8	DSAP, const	0x28
9	SSAP, const	0x00
10	Sensor address	1-4
11	Sensor Type	See sensor types subsection
12	Padding, const	0x00
[13:32]	Sensor name	up to 20 chars(zero padded)
[33:52]	Additional Name	up to 20 chars(zero padded)
53	Variable format. Not tested properly.	0x03 (real number format)
54	Length	up to 8
55	Precision	For real numbers: not more than Length - 2
56	Padding	0x00
[57:60]	Unit	max 4 chars
[61:74]	Padding	0x00
75	Checksum	[DA+SA+FC+DataUnit] mod 256
76	End byte	0x16

Device address change

Implemented in profibus/address_edit_request.py

It allows to set new ISM module address(1-127). It is important not to connect two modules with the same address to prevent conflicts.

Byte number	Description	Byte value
1	Start Delimiter, const	0x68
2	Length, const	0x46
3	Length, const	0x46
4	Start Delimiter, const	0x68
5	Module Address	0x80 + actual address(1-127)
6	Source Address, const	0x80
7	Frame Control, const	0x4C
8	DSAP, const	0x28
9	SSAP, const	0x00
10	Command	0x6E
11	New address	from 0x01 to 0x7F
12	Unknown byte, const	0x01
13	Unknown byte, const	0x4B
[14:74]	Padding	0x00
75	Checksum	[DA+SA+FC+DataUnit] mod 256
76	End byte	0x16

Get Module configuration

Implemented in profibus/module_config_read_request.py

Requests a telegram with ISM module configuration.

Byte number	Description	Byte value
1	Start Delimiter, const	0x68
2	Length, const	0x06
3	Length, const	0x06
4	Start Delimiter, const	0x68
5	Module Address	0x80 + actual address(1-127)
6	Source Address, const	0x80
7	Frame Control, const	0x4C
8	DSAP, const	0x29
9	SSAP, const	0x00
10	Command	0x64
11	Checksum	$[DA+SA+FC+DataUnit] \bmod 256$
12	End byte	0x16

Get Sensor configuration

Implemented in profibus/variable_config_read_request.py

Requests a telegram with variable(sensor) configuration.

Byte number	Description	Byte value
1	Start Delimiter, const	0x68
2	Length, const	0x06
3	Length, const	0x06
4	Start Delimiter, const	0x68
5	Module Address	0x80 + actual address(1-127)
6	Source Address, const	0x80
7	Frame Control, const	0x4C
8	DSAP, const	0x0C
9	SSAP, const	0x00
10	Sensor address	1-4
11	Checksum	[DA+SA+FC+DataUnit] mod 256
12	End byte	0x16

Get Sensors Readings

Implemented in profibus/measurement_request.py

Requests a telegram with variable(sensor) value for a selected module.

Byte number	Description	Byte value
1	Start Delimiter, const	0x68
2	Length, const	0x06
3	Length, const	0x06
4	Start Delimiter, const	0x68
5	Module Address	0x80 + actual address(1-127)
6	Source Address, const	0x80
7	Frame Control, const	0x4C
8	DSAP, const	0x0D
9	SSAP, const	0x00
10	Sensor address	
11	Checksum	[DA+SA+FC+DataUnit] mod 256
12	End byte	0x16

6.4.2 Responses

Module configuration

Implemented in profibus/config_answer.py

The Datafield of the response telegram contains following data:

- Location - datafield bytes [2:21]
- Username - [30:49]
- Date - [50:55]
- Time - [56:59]

Sensor configuration

Implemented in profibus/variable_config_answer.py

The Datafield of the response telegram contains following data:

- Type - datafield byte 2
- Name - [3:22]

- Additional Name - [24:43]
- Format - 44
- Length - 45
- Precision - 46
- Unit - [48:51]

Sensor readings

Implemented in profibus/measurements_response.py

The Datafield of the response telegram contains following data:

- DSAP - byte 0
- SSAP - byte 1
- Sensor data [2:]

6.4.3 Findings

- All request telegrams that are writing some data into the ISM module always have constant length - 76 bytes. All of this telegrams can be distinguished to each other by looking at first three bytes in Data Field. Data Field starts with DSAP byte followed by SSAP byte. And the third byte is unique for each type of the telegram.
- Frame Control byte is always 0x4C. According to PROFIBUS specification document, Frame Control(FC or Control Octet) indicates the frame type. And, in addition, it has the station type information and function and control information, which prevents loss and multiplication of messages. 0x4C is 0100 1100 in binary format. From right to left, first 4 bits(1100) represent function. In current case the function is Send and Request Data Low. Bit 5 and 6 can represent Station Type or Frame Count depending on bit 7 value(flag), in this case - frame count(00 indicates request with no acknowledgement). Bit 7 - flag. Bit 8 - reserved.¹

¹PROFIBUS specification 1.0

6.5 UI overview

6.5.1 Sensor configuration app

Connection dialog

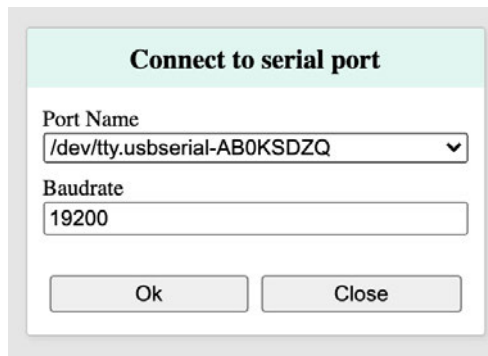


Figure 6.3: Connection dialog

This dialog is shown if the app is opened first time. It asks User to select a port name(starts with "COM..." in Windows, and "/dev/..." in Unix. It must be a Serial Port that is used for a connection to the device. Next field is a baud rate and it is 19200 by default.

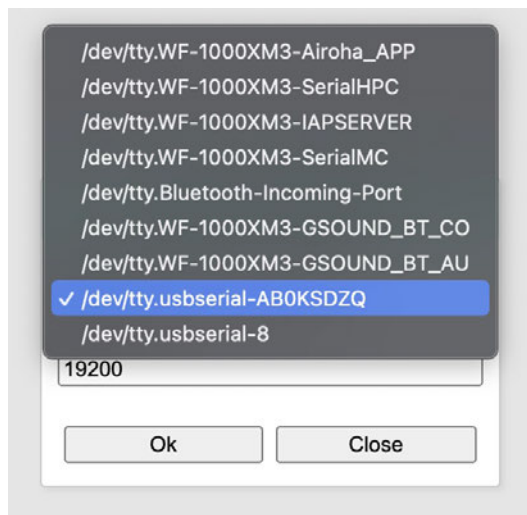


Figure 6.4: Serial port selection dropdown (Mac)

Configuration

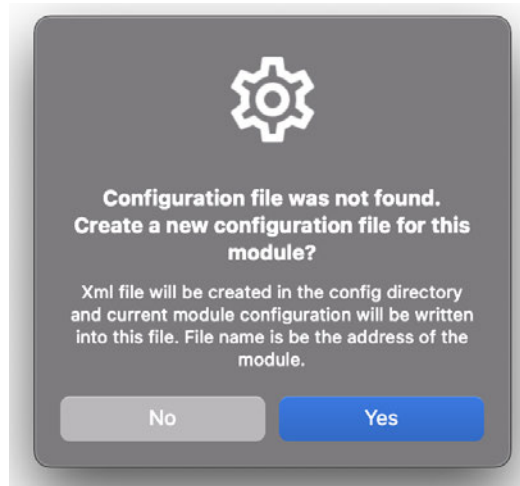


Figure 6.5: Configuration dialog

This dialog is shown when there is no xml file in "config" folder detected. In this case the file can be generated and filled with module data automatically.

Navigation/Status Bar

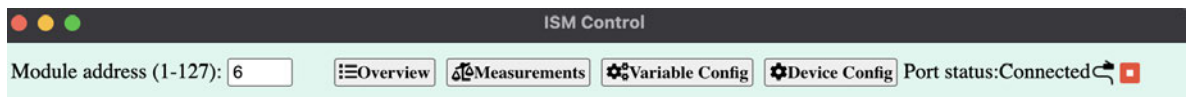


Figure 6.6: Navigation/Status Bar

The bar is shown on all app screens. It allows User to set an address of ISM Module that he wants to configure. It contains navigation buttons leading to different app screens and, finally, it contains Serial Port status and control buttons. Elements have tooltips that are shown on hovering.

Welcome

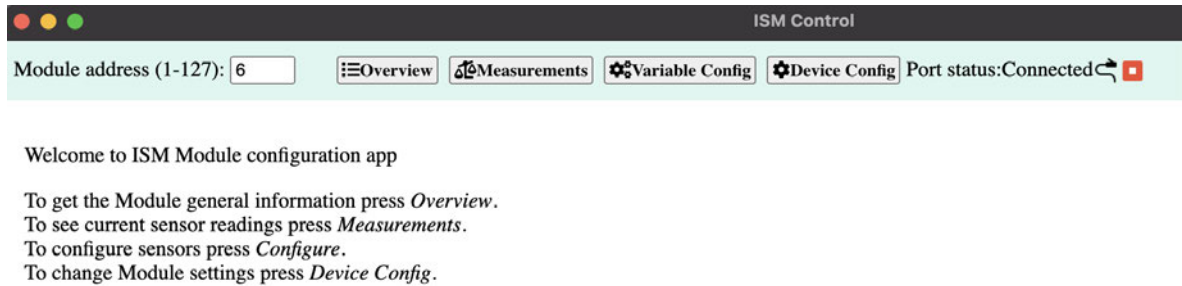


Figure 6.7: Welcome screen

This screen User sees after he opens the app, it appears when Serial connection with ISM Module is successfully established. It contains some introductory information, so User can get familiar with what he can do next.

Overview

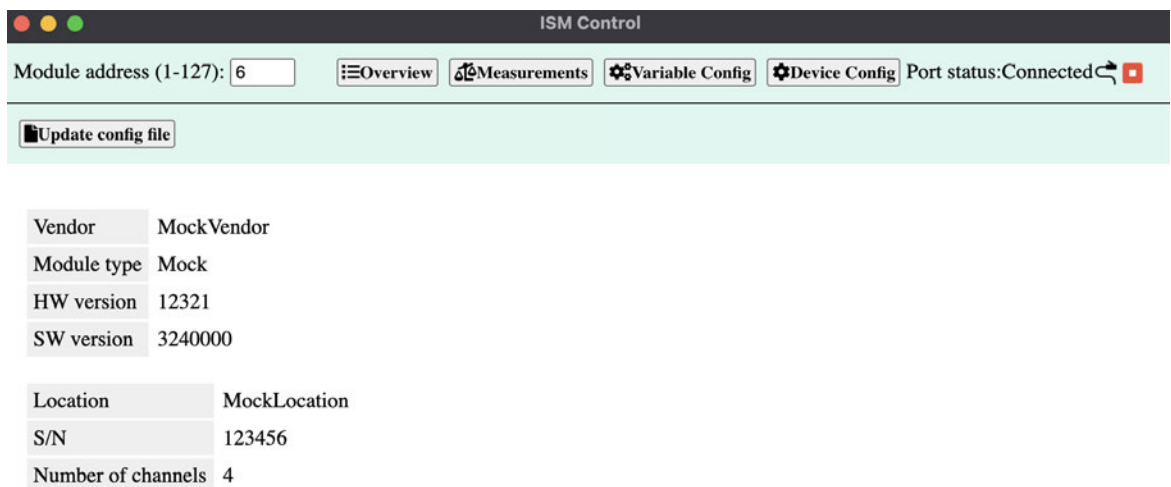


Figure 6.8: Overview

The overview screen shows data requested from the ISM module memory. It displays module's characteristics and allows to write these into xml config file("Update config file").

Measurements

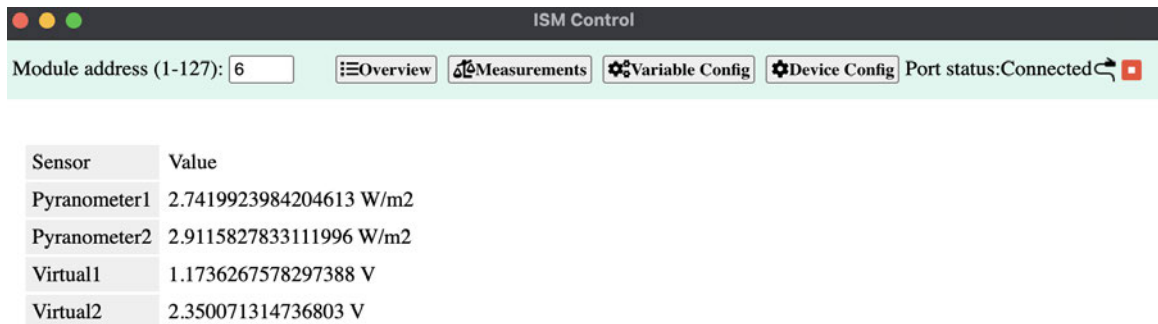


Figure 6.9: Measurements

In Sensors Configuration app Measurements sections carries only informational function. It just displays current readings and does not store it anywhere, as it is done in the Database app. Sensor names are taken from local xml config file to avoid querrying the ISM module.

Sensor Configuration

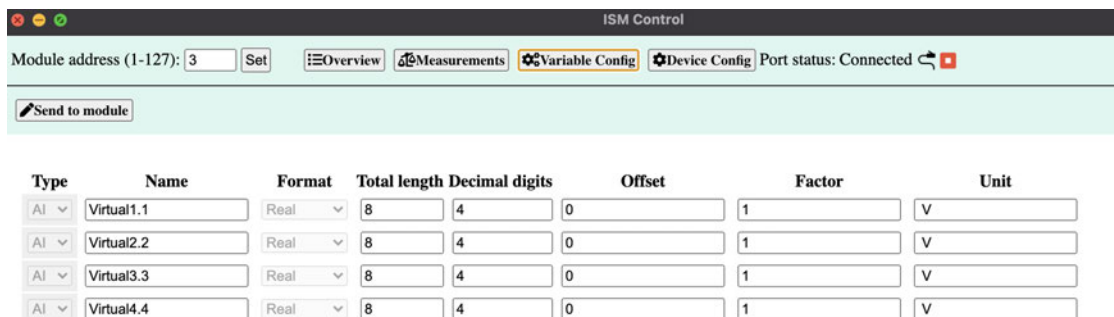


Figure 6.10: Sensor Configuration

One of the most important screens. It allows to configure each sensor's name, output length and precision and unit. Also it displays sensor type and output format. The maximum length of real numbers is 8, and decimal digits should take not more that length - 2. Clicking "Send to module" sends data to ISM module and writes into local xml config file as well. "Offset" and "Factor" are local settings that are not stored in the module. They are used to map voltage

levels to actual values. If User does not want to overwrite data in ISM module memory, he can just update xml config file manually.

Module Configuration

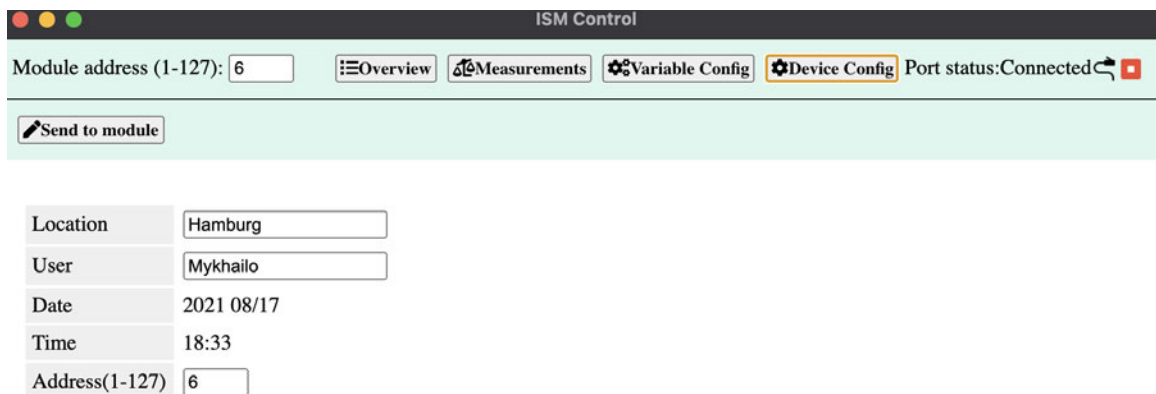


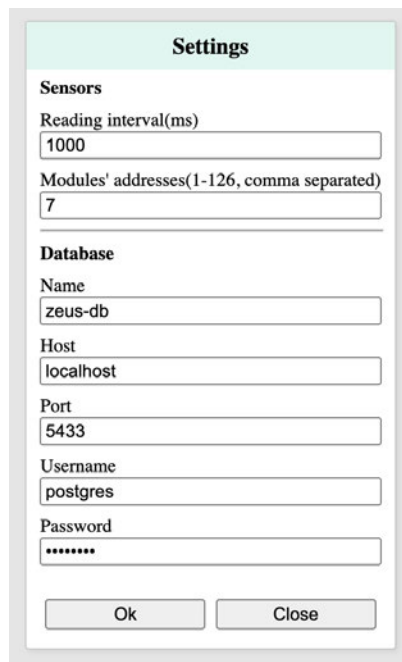
Figure 6.11: Module Configuration

Here User can set location of the ISM Module(room number etc.), username and set a new module address. "Send to module" button sends the data into ISM memory as well as storing it locally in xml config file.

6.5.2 Sensor readings app

Settings dialog

When the Application started for the first time, a settings dialog is shown. An xml config file will be generated and these settings will be stored there.



Settings

Sensors

Reading interval(ms)
1000

Modules' addresses(1-126, comma separated)
7

Database

Name
zeus-db

Host
localhost

Port
5433

Username
postgres

Password

Ok Close

Figure 6.12: Application Configuration

Later on the Settings can be accessed via navigation toolbar. They contain the **interval** in milliseconds with which module get requested for the sensors' readings. These readings can be then stored locally and sent to a remote database. **Modules' addresses** allow to specify all connected modules that can provide the readings. The important part is the **Database**. It contains remote Postgres database credentials needed to connect to the database. Password is stored locally in xml file in an encoded and encrypted format.

Control bar

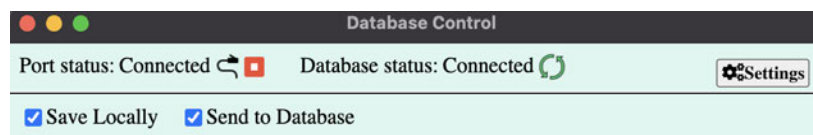


Figure 6.13: Control bar

Control bar shows the connection status of the serial port. It allows to disconnect or connect to another serial port. Also a database connection status is shown. It is possible to try to

reconnect in case of an error. In addition User can disable/enable storing sensors' readings locally to txt file and remotely to the database.

Readings screen

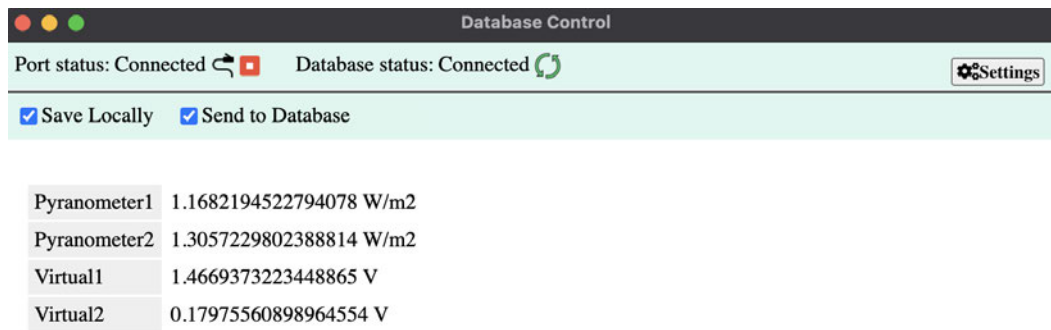


Figure 6.14: Readings screen

The Application makes requests to ISM sensor modules in intervals that were set, the obtained readings are showed to the User and added to a buffer. When buffer is full, the readings are recorded both locally and in the database.

6.6 JavaScript Design Patterns

6.6.1 Overview

Originally object oriented design patterns were documented by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides(the Gang of Four) and have been used in many object-oriented languages. JavaScript differs from such languages as C++ or Java and it makes it hard to straightforward apply the patterns of the Gang of Four. First, JavaScript is a dynamically typed language, it means that the type of variables is determined at the runtime. So, to apply some specific patterns, a range of special tricks and techniques needs to be used that makes it questionable if the effort worth it.² The patterns used in the project are rather simple and do not require these. Patterns are needed to make the code more modular, efficient and more readable and understandable.

²Ross Harmes (2008) Introduction

6.6.2 Singleton

The Singleton is one of the most basic designed patterns and it is used in JavaScript a lot. It groups a piece of code into a single logical unit that can be assigned to a single variable. There are several use cases for the Singleton. It can be used for namespacing, to prevent function names clash. It can encapsulate browser differences with a technique called branching(not relevant for the current project as it does not use browsers). And it can be used to keep the code organized³. In JavaScript, the Singleton can be created in a manner that differs from other languages:

```
1 const Singleton = {
2   attribute1: true,
3   attribute2: 5,
4
5   method1: () => {
6     \\method code ...
7   }
8 }
9
10 \\this can be accessed as:
11 Singleton.attribute1;
12 Singleton.method1();
```

Here is an example of one of the Singletons from the project:

```
1 const XmlConfig = {
2   client: null,
3   _send: (action, message, deadline, callback) => {
4     //Code removed for simplicity. Method is sending request to ISM
5     },
6   checkConfig: (deadline, params, callback) => {
7     XmlConfig._send('CheckConfig', params,
8       { deadline }, callback);
9   },
10  storeDeviceConfig: (deadline, params, callback) => {
11    XmlConfig._send('StoreOverviewConfig', params,
12      { deadline }, callback);
13  }
14 }
```

Other example of a Singleton is SerialPort.js.

³Ross Harmes (2008) p. 65

Benefits of the Singleton

One of the advantages of using the Singleton is that it keeps the code organized. Grouping related methods together in a single location, which cannot be instantiated several times, it is easier to debug and maintain the code. Namespacing protects the methods from accidental overwriting and prevents the global namespace pollution.

Drawbacks of the Singleton

The Singleton patterns has a potential of tightly coupling modules together by providing a single access point. It can be better to create an instantiated class even it will be only instantiated once in a program lifetime. The Singleton also makes unit testing harder because of classes coupling.⁴

6.6.3 Callback

A Callback is a function that is invoked for result propagation in case of an asynchronous operation. It is based on a fact that JavaScript supports passing any functions as a parameter of another function. It makes the Callback easy implementable. Also JavaScript closures allows the callback to access the context in which asynchronous operation was requested, and not to depend on when or where the Callback was invoked. In JavaScript, the Callback pattern is achieved by creating a function that has another(callback) function as a parameter(by convention - last parameter).⁵ Let's have a look into one of the previous examples:

```
1 const XmlConfig = {
2   client: null,
3   _send: (action, message, deadline, callback) => {
4     try {
5       XmlConfig.client[action](message, deadline, function (err,
6         response) {
7         if (err) {
8           callback.onError(err);
9           return;
10        }
11        if (!response) {
12          callback.onError('Got undefined response');
13        }
14      });
15    }
16  }
```

⁴Ross Harmes (2008) p.82

⁵Casciaro (2014) p.18,19

```
14     }
15     callback.onSuccess(response);
16   });
17   } catch (error) {
18     callback.onError('Got unexpected error: ' + error);
19   }
20 },
21 checkConfig: (deadline, params, callback) => {
22   XmlConfig._send('CheckConfig', params,
23     { deadline }, callback);
24 },
25 storeDeviceConfig: (deadline, params, callback) => {
26   XmlConfig._send('StoreOverviewConfig', params,
27     { deadline }, callback);
28 }
29 }
```

Here a callback is passed to `checkConfig()` or `storeDeviceConfig()` and then to `_send()`. Some asynchronous operation is done inside `_send()` and, when it is finished, the callback is executed. The callback is an object that has two member functions: `onError()` and `onSuccess()`, that are executed in case of an error or success respectively.

6.6.4 Bridge

Bridges can be useful when implementing an API. It is one of the simplest patterns and it is easy to implement and put into practice. The purpose of this pattern is to decouple an abstraction from its implementation.⁶

If you look at the example from the previous Callback pattern description you will see a Bridge there as well. It has an API that is represented by two functions: `checkConfig()` and `storeDeviceConfig()`. They are abstractions over rpc client calls which are done via `XmlConfig.client` object. The implementation is the `_send()` function.

Benefits of the Bridge

Using the Bridge can increase the code maintainability. Parts of the code can be managed independently when abstractions are separated and decoupled from implementations. It makes it easier to find bugs, and the code is less likely to be seriously broken.

⁶Ross Harmes (2008) p.109

Drawbacks of the Bridge

This pattern has a few disadvantages. Performance of the program can be affected by the fact that the Bridge creates another function call. Bridges also add complexity, which can make the code harder to debug and read.⁷

6.7 Testing

6.7.1 Testing with ISM-111 test module

A separate ISM-111 module with two connected pyranometers has been provided by TU Lübeck. It was used during the development process of the application. It differs from the real set up, because it does not have several interconnected modules and the number of sensors is much less. Other than that the test set up has all the functionality as the main set up and can be used to develop and test some required features. To improve the testing some mocking had to be done.

6.7.2 Testing with Mock objects

With a mock object the serial communication can be simulated. Instead of sending and getting data from COM port, object functions are called. There are 3 classes in "src/test/mocks" directory.

- "ProfibusMock" is used instead of "ProfibusAsciiMaster". It mocks 127 ISM modules with 4 sensors in each module and it has the functions from "ProfibusAsciiMaster".
- "ModuleMock" represents ISM module. It contains sensor configurations as byte arrays in PROFIBUS telegram format and provide return values for different requests that are coming from User.
- "SensorMock" contains sensor name and measurement.

6.8 Setting up and running the project

Currently it is supposed that all project components are running and hosted on a single machine(except the Database). In future it can be easily distributed after some small adjustments if needed. The project can be started on Windows, Linux or Mac. After downloading the folder (e.g. from git), node and Python packages should be installed.

⁷Ross Harmes (2008) p.123

6.8.1 Getting the project

The project can be cloned from GitLab instance. Current url is <https://gitlab.tphys.jku.at/haw-hamburg/solarhaus/meteosystem/meteo-control-software.git>

6.8.2 Setting up directories

Root directory of the project contains following directories: *docker* and *src*. The first one is related to Database docker image and the latter contains the source code. In addition two directories should be manually created in the root directory of the project: *config* and *measurements*. The first one will be used to store xml config files and the latter is for storing measurements in txt files.

6.8.3 Installing Python packages

Python is using *Pipenv*(see section 4.4) which makes it easy to install or required packages. Make sure that *Python 3* as well as *pipenv* is installed and is available in the PATH (can be called with 'python3' or 'python' and 'pipenv' from a command line). To install all packages from the Pipfile user must navigate to the project directory(where Pipfile is located) and execute the following command:

```
1 pipenv install
```

To test the pipenv the following command can be executed:

```
1 pipenv shell
```

It creates a new pipenv shell process. It can be terminated by typing *exit* command.

Missing packages can be installed with:

```
1 pipenv install <package-name>
```

6.8.4 Installing Node packages

Once Node.js is installed(make sure that it was added to PATH) all required packages can be installed by navigating to *src/UI* directory and running 'npm install' command(see section 4.2). A new directory called 'node_modules' will be created - it contains all the required modules to run both ISM Configuration app and Database app as they share the code.

6.9 Challenges

Time frame

Bachelor project has a fixed time frame. And the most important thing here is to have a fixed plan and to choose the right technologies. JavaScript and Python showed to be suitable languages for rapid development.

Project size

It turned to be quite large project. It contains several components that are communicating over gRPC: GUI apps for sensor configuration and for readings storage, backend python service that communicates with ISM modules, GUI apps, database and filesystem. Some prior experience and knowledge is crucial here, as it is impossible to learn from scratch in this case.

Lack of ISM-111 module documentation

Not all PROFIBUS commands are officially documented. I.e. commands for setting sensors configuration. So they had to be reverse engineered by using old software(Combilog) and listening on a COM port. Both request and response telegrams had to be deciphered. It requires deep understanding of serial communication and PROFIBUS Telegram formats.

Small prior experience with Python

Python differs from C-style languages, so it takes some time to get used to it. OOP also differs from other languages(like Java or C++). An example of interesting feature: class member variables are all static by default(it took quite a lot of time to understand what is going on).

Lack of good books on PROFIBUS

Seems to be a niche technology. Not so many people involved nowadays, that is why there are not many authors. But, at the end, it turned out that official PROFIBUS specification and ISM-111 user manual was enough.

Cross-platform issues

The Application was developed and tested using Mac computer. Installing and running it on Windows 10 took some time. The main difficulty was to deal with security errors that were coming from Windows platform, so the code had to be adjusted. The other trick was a

difference in COM port naming. Also processes spawning and killing differs between platforms. Other than that, Electron.js and Python fit perfectly for the cross-platform development.

Test System vs Real System

Testing is an important phase in software development. The main challenge of the project was that real system had some differences from the testing system: different length of cables(more distortion), more interconnected modules, different number of sensors per a module. And the main system was located in Lübeck. Test System consisted of one ISM-111 module and 2 connected sensors. Real System(located at the Solar House) consisted of three ISM-111 modules connected between each other and single RS232 cable. First, the application did not run smooth with the real system. Some adjustments needed in handling corrupted telegrams with wrong checksums. Also some adjustments were made to handle different number of sensors that can be connected to a module.

6.10 Results

- GUI technologies have been researched. Electron.js has been used to create both sensor configuration and readings storage clients.
- Inter Process Communication technology has been researched and gRPC has been used to implement communication between Python back-end and Electron.js front-ends.
- The Application can be installed on Windows, Linux or Mac.
- User can obtain module information from the internal storage of the module. This data is presented in GUI.
- Use can write configuration data into a module
- Configuration data is also stored on PC in xml files.
- Readings are stored in txt files and also in a remote database

7 Appendices

7.1 Appendix A: Installation Guide

This guide describes the installation process on Windows as the main focus platform. Installation on Linux and Mac slightly differs. The installation is mainly happening using cmd.exe tool which is a command line terminal included in each Windows installation by default.

1. Download project files from <https://gitlab.tphys.jku.at/haw-hamburg/solarhaus/meteosystem/meteor-control-software.git>.
2. Set up directories. Root directory of the project contains following directories: docker and src. The first one is related to Database docker image and the latter contains the source code. In addition two directories should be manually created in the root directory of the project: config and measurements. The first one will be used to store xml config files and the latter is for storing measurements in txt files.
3. Check if Python 3 is installed (open cmd.exe and type `python -V`). Python 3.9 has been used for development, but other versions may work as well. It can be downloaded from <https://www.python.org/downloads/>
4. Open cmd.exe(command line tool). Go to Project's root directory. Type "pipenv install". If you see an error, then pypenv is not installed or is not part of the PATH system variable. To install pipenv type: "pip install -user pipenv". After installation it will display a path to pipenv. This location should be added to PATH enviromental variable(more info here: <https://www.architectryan.com/2018/03/17/add-to-the-path-on-windows-10/>). More information on pip installation: <https://www.geeksforgeeks.org/how-to-install-pip-on-windows/>. More information on pipenv installation: <https://www.pythontutorial.net/python-basics/install-pipenv-windows/>
5. If pipenv is installed correctly cmd command "pipenv shell" should create a new pipenv shell process. Type ".exit" to exit the shell.

6. Download Node.js(<https://nodejs.org/en/download/>) and add to PATH. The installation can be checked with "node -V" cmd command.
7. Navigate to src/UI directory and run "npm install" command. It will download and install all required Node packages.

7.2 Appendix B: Configuration app User Guide

Configuration App is used to read and write data to internal ISM module memory. Also it stores the configuration in '/config' directory. These files can be also created and modified manually without using the Configuration App.

Starting the App

It can be started by double clicking "start_config.bat" file located in the Project's root directory. In case of an error the App can be restarted manually by using CTRL+R. If the error related to Python background process, it should be killed from Task Manager. At the initial start, if there are no xml configuration files found, the App will ask to generate it. It may not work well if several modules are connected together. One solution is to disconnect them and use one by one. Easier and better solution - manually create and adjust xml files. Root directory contains 'config_template.xml'

Manually creating xml configuration

1. Copy 'config_template.xml' into 'config' directory.
2. Rename 'config_template.xml' into 'module_address.xml'. I.e. '1.xml', '2.xml' or '3.xml'.
3. Open the file and adjust the contents. You can use the Config App to get the data.

Variable Config Screen

It displays sensors' configuration data. If you are using it with several interconnected ISM modules, some of them can provide a corrupted data. "Send to module" button stores/updates local xml config file and also writes data into ISM module memory. Offset and Factor are only stored locally. Rest of the settings are stored both locally and in the module's memory. It is not recommended to send to a module if several modules are connected together. The workaround is to manually edit xml files in '/config' directory as ISM module memory data is not really used.

More Information

More description can be found in section 5.5.1

7.3 Appendix C: Storage app User Guide

Storage App is used to request readings from sensors in a regular intervals, store them in txt files in '/measurements' directory and send them to a remote database.

Starting the App

It can be started by double clicking "start_reading.bat" file located in the Project's root directory. In case of an error the App can be restarted manually by using CTRL+R. If the error related to Python background process, it should be killed from Task Manager. At the initial start, if there is no global xml configuration file found, the App will ask to generate it. Alternatively it can be manually created by copying 'config_global_template.xml' into 'config' directory and renaming the file into 'global.xml'.

Settings

The most important settings are: readings interval and modules' addresses. Reading interval should be adjusted not to be too short as requesting the data itself takes some time and can be around one second for each module. The recommended interval is from 5000 ms and above. In case if a lot of telegrams come corrupted, the interval should be increased as the program retries the request if it gets corrupted telegram. Modules' addresses are the addresses of connected ISM modules(i.e. 1, 2, 3). Other settings group is needed for a Postgres database access. In case the database is offline the application will display an error, but can continue to store the data locally.

Saving modes

1. Save Locally: stores the data in local txt file that is located in '/measurements' directory. The file has three columns: Sensor Global Index, Timestamp, Value(adjusted with offset and factor)
2. Send to Database: sends data to Postgres database in a regular intervals. Data is written into 'measurements' table.

Manual Transfer

In case if there is a difference between local storage and remote database, you can transfer missing readings from local txt files to the database. Uncheck modes checkboxes and click 'Manual Transfer'. It will show the difference in timestamps. Click 'Apply' to transfer missing data from txt files to the database.

More information

More description can be found in section 5.5.2

7.4 Appendix D: Developer Notes

Pipenv

The Application uses pipenv. If some new python package is required it should be added via executing following cmd command in project root directory: pipenv install <package-name>

Python backend process

Python process is running as a daemon process. The main thing to consider: if you update some python code, this process should be killed from the Task Manager(open Task Manager with CTRL-ALT-DEL search for Python and kill all Python processes). Otherwise changes to the code will be not visible as old process continues to run. Also in case of Python related error it may be required to kill the process and restart the app after that. When any of the UI apps is started, it spawns a python background process(if not yet started).

gRPC

The application uses gRPC for communication between Python back-end and Electron.js front-ends. Python server api functions are implemented in "api.py" file. These functions are using generated files from "/rpc" directory. In case some new function is needed it should be added into "/rpc/device.proto". Then new python files should be generated with the following cmd command(executed from rpc directory): python -m grpc_tools.protoc -proto_path=/Users/User/PATH-TO-PROGRAM/meteo-control-software/src/rpc -python_out=. -grpc_python_out=. device.proto --experimental_allow_proto3_optional (Linux. In case of Windows path separators are different!). Make sure that generated "device_pb2_grpc.py" in

line 5 has : "import rpc.device_pb2 as device__pb2". For more information see:
<https://grpc.io/docs/languages/python/basics/#generating-client-and-server-code>

Node modules

Both UI apps use shared node modules. "/UI" directory contains package.json file, so all required modules can be installed by running "npm install" in this directory. Do not do it inside "sensors-ui" or "database-ui"

Testing

The application can be started in Test Mode(without connecting to real device via COM port). To do this, open api.py and change line 54 to TEST_MODE = True. Close the app and kill Python processes from Task Manager. Testing Mode uses "/src/test/mocks" python code. It contains a mock classes for ISM sensor module and sensors. It supports some important requests. Both sensors-ui and database-ui can be started in Test Mode.

Bibliography

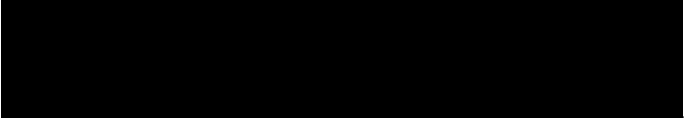
- [Axelson 2007] AXELSON, Jan: *Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems (Complete Guides series)*. Lakeview Research, 2007. – ISBN 9781931448062
- [Casciaro 2014] CASCIARO, Mario: *Node.js Design Patterns*. Packt Publishing, 2014. – ISBN 1783287314,9781783287314
- [Jensen 2017] JENSEN, Paul B.: *Cross-Platform Desktop Applications using Electron and NW.js*. Manning Publications Co., 2017. – ISBN 9781617292842
- [Josef Weigmann 2004] JOSEF WEIGMANN, Gerhard K.: *Decentralization with PROFIBUS DP DPV1: Architecture and Fundamentals, Configuration and Use with SIMATIC S7*. Publicis Corporate Publishing, Erlangen, 2004. – ISBN 3895782181,9783895782183
- [Leslie 1997] LESLIE, Cortes: *Designing a Graphical User Interface*. Clinical Information Engines, Austin Texas, 1997. – ISBN 0-201-60842-1,0-385-26774-6,0-442-01750-2,1-55615-439-9,1-55615-679-0
- [Manual] MANUAL, ISM M.: *Intelligentes Sensor Module ISM 111 Gerätebeschreibung V 2.21*.
- [Richard W. D. Nickalls 1995] RICHARD W. D. NICKALLS, R. R.: *Interfacing the IBM-PC to Medical Equipment: The Art of Serial Communication*. Cambridge University Press, 1995. – ISBN 0521462800,9780521462808
- [Ross Harmes 2008] ROSS HARMES, Dustin D.: *Pro JavaScript Design Patterns*. Apress, 2008. – ISBN 978-1-59059-908-2, 978-1-4302-0495-4
- [Schleicher M. 2001] SCHLEICHER M., Blasinger F.: *Digital interfaces and bus systems for communication*. M.K. JUCHHEIM & Co, Fulda, 2001. – ISBN 3935742037
- [Simon St. Laurent 2001] SIMON ST. LAURENT, Joe J.: *Programming Web Services with XML-RPC*. O'Reilly Internet Series, 2001. – ISBN 9780596001193,0596001193

Bibliography

[Wang 2011] WANG, Wego: *Reverse Engineering. Technology of reinvention*. CRC Press Taylor & Francis Group, 2011. – ISBN 9781439806319

[Ward Rosenberry 1995] WARD ROSENBERRY, John S.: *Microsoft RPC Programming Guide*. O'Reilly Media, 1995. – ISBN 1565920708,9781565920705

I declare that this Bachelor Thesis has been completed by myself independently without outside help and only the defined sources and study aids were used.

Hamburg, 5. October 2021  Mykhailo Svyrydovych