Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Software pro vizualizaci dat a konfiguraci zařízení s využitím sériové komunikace

Bc. Petr Jiruše

Diplomová práce

2025

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2024/2025

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Petr Jiruše**
Osobní číslo:	**I23274**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Software pro vizualizaci dat a konfiguraci zařízení s využitím sériové komunikace**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Cílem práce je vytvořit software pro vizualizaci dat ze zařízení společnosti Steinel s možností jejich konfigurace. Aplikace bude komunikovat se zařízeními pomocí asynchronního sériového přenosu dat a bude připravena na nové verze komunikačního protokolu. Data budou zobrazována pomocí grafických komponent, které si uživatel může přizpůsobit, a nastavení bude možné sdílet mezi různými počítači. Uživatelské rozhraní bude podporovat světlý a tmavý režim a bude kompletně v angličtině s možností přidání dalších jazykových verzí. Aplikace bude napsána v C# s využitím frameworku WPF a bude určena pro Windows. Student v teoretické části rozebere příslušné používané technologie.

Rozsah pracovní zprávy: **50-60 stran**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

GRIFFITHS, Ian. Programming C# 12: Build Cloud, Web, and Desktop Applications. O'Reilly Media, 2024. ISBN 978-1098158361.
SKEET, Jon. C# in depth. Fourth edition. Shelter Island, NY: Manning Publications Co., [2019]. ISBN 978-1-61729-453-2.
BURNETTE, Jan Axelson. Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems. 2nd ed. Lakeview Research LLC, 2018. ISBN 978-1931448284.
NATHAN, Adam. WPF 4.5 Unleashed. Sams Publishing, 2021. ISBN 978-0672336973

Vedoucí diplomové práce: **Ing. Jan Merta, Ph.D.**
Katedra softwarových technologií

Datum zadání diplomové práce: **31. října 2024**
Termín odevzdání diplomové práce: **23. května 2025**

L.S.

prof. Ing. Petr Doležel, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 29. listopadu 2024

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 12. 5. 2025

Petr Jiruše v.r.

PODĚKOVÁNÍ

Tímto bych rád poděkoval panu Ing. Michalu Lauterbachovi a panu Ing. Janu Mertovi, Ph.D. za jejich odborné připomínky a rady, které byly velkým přínosem při zpracování této práce.

ANOTACE

Diplomová práce se zabývá návrhem a realizací desktopové aplikace pro vizualizaci a konfiguraci zařízení společnosti Steinel. Cílem je vytvořit moderní a uživatelsky přívětivý nástroj pro flexibilní zobrazování dat v reálném čase za použití různých protokolů. Data budou zobrazována pomocí grafických komponent, které si uživatel může nakonfigurovat a tuto konfiguraci sdílet mezi další počítače. Aplikace je napsána v jazyce C# s použitím technologie WPF a knihovny MahApps.Metro pro dodatečné stylování. Dále podporuje přepínání mezi světlým a tmavým režimem, lokalizaci do více jazyků a uložení nastavení. V teoretické části jsou rozebrány použité technologie a praktická část popisuje do detailu vyvinuté moduly.

KLÍČOVÁ SLOVA

Windows aplikace, sériová komunikace, asynchronní přenos, vizualizace dat, C#, WPF, konfigurace

TITLE

Software for data visualization and device configuration using serial communication

ANNOTATION

This thesis focuses on the design and implementation of a desktop application for the visualization and configuration of devices manufactured by Steinel. The goal is to create a modern and user-friendly tool for flexible real-time data visualization using various communication protocols. The data is displayed using graphical components that users can configure and share across multiple computers. The application is developed in C# using the WPF framework and the MahApps.Metro library for enhanced styling. It also supports switching between light and dark modes, multilingual localization, and settings persistence. The theoretical part analyzes the technologies used, while the practical part provides a detailed description of the developed modules.

KEYWORDS

Windows application, serial communication, asynchronous transmission, data visualization, C#, WPF, configuration

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM KÓDŮ

SEZNAM ZKRATEK

ADC	Analog-to-Digital Converter
BPS	Bits Per Second
CRC	Cyclic Redundancy Check
CS	Chip Select
DAC	Digital-to-Analog Converter
HTML	Hypertext Markup Language
I2C	Inter-Integrated Circuit
IP	Internet Protocol
JSON	JavaScript Object Notation
LSB	Least Significant Bit
MISO	Master In Slave Out
MIT	Massachusetts Institute of Technology
MOSI	Master Out Slave In
OOP	Object-Oriented Programming
RAM	Random-Access Memory
RX	Receive
SCLK	Serial Clock
SDI	Serial Data In
SDO	Serial Data Out
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
SYN	Synchronize
TCP	Transmission Control Protocol
TX	Transmit
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

ÚVOD

Postupem času roste důraz na efektivitu a automatizaci v průmyslové sféře a s tím i potřeba efektivní správy a vizualizace dat ze zařízení. Moderní svítidla a senzory generují nepřetržité toky dat, které slouží nejen k monitorování jejich činnosti, ale i k jejich správnému nastavení nebo optimalizaci provozu. Na trhu existuje velké množství softwarových nástrojů pro práci se sériovou komunikací, ale většina z nich se zaměřuje na velkou škálu různých zařízeních a nejsou vhodná pro konfigurování konkrétních produktů společnosti Steinel. Důvodem je nutnost autorizace před jakýmkoliv zásahem do nastavení zařízení, tím pádem je nepřípustné posílat tyto citlivé údaje přes softwarové nástroje třetích stran. V minulosti již byly nějaké pokusy o vytvoření vizualizačních nástrojů pro specifické potřeby firmy, ale ani jedna z nich nenaplnila očekávání a požadavky na flexibilitu. Nehledě na fakt, že stávající vizualizace jsou nejednotné a běží na starších technologiích jako je Windows Forms. Z těchto důvodu vznikla potřeba vytvořit nový specializovaný nástroj, který bude vyhovovat požadavkům na vizualizaci dat ze zařízení a bude dostatečně flexibilní pro pokrytí různých protokolů pro komunikaci. Zároveň také bude splňovat moderní standardy pro styly aplikace a bude nabízet přívětivé uživatelské rozhraní s možností výběru lokalizace a dalších přizpůsobení.

Tato diplomová práce si bere za cíl realizovat softwarovou aplikaci pro vizualizaci dat a konfiguraci zařízení společnosti Steinel, které ve většině případů používají ke své komunikaci sériové rozhraní UART. Výsledná aplikace je určená pro desktopové počítače s operačním systémem Windows. Aplikace používá platformu WPF a doplňkový framework MahApps.Metro pro docílení moderního vzhledu. Hlavním požadavkem na aplikaci je schopnost zpracovávat a zobrazovat velké množství dat v reálném čase prostřednictvím uživatelsky volených komponent, které jsou dynamicky generované na základě externích konfiguračních souborů ve formátu JSON.

Oproti existujícím řešením aplikace přináší velice flexibilní rozhraní, které dovoluje uživatelům si definovat vlastní podobu části uživatelského rozhraní. Uživatel není pouhým pasivním příjemcem dat, ale dokáže přes aplikaci konfigurovat jakékoliv parametry zařízení, které daný protokol povoluje. Celkově je aplikace zaměřená velkou mírou na přizpůsobení ze strany uživatele, napomáhá tomu i jazykový systém umožňující lokalizaci uživatelského rozhraní, přičemž výchozí jazyk je angličtina. Další jazyky lze do aplikace přidávat bez nutnosti zásahu do zdrojového kódu. Vzhled aplikace musí také podporovat přepínání mezi světlým a tmavým režimem, což je mnohem důležitější vlastnost, než se může zdát. Důvodem jsou testy různých světel a senzorů, kde v některých případech se testují v temných místnostech a je naprosto

nežádoucí, aby jiný zdroj světla ovlivňoval výsledky měření. Jedním z dalších klíčových požadavků na aplikaci je modul pro grafickou vizualizaci hodnot přijímaných ze zařízení. Tím uživatel získá přehled o tom, jak se vybrané veličiny mění v čase. Okno s grafem a s ním i všechna další dialogová okna podporují změnu stylů na základě vybraného režimu a drží si jednotný vzhled s barvami charakterizující firmu Steinel. Samozřejmostí je responzivní design ve všech částech aplikace a definice minimálních rozměrů okna pro ideální rozložení komponent.

Teoretická část nejprve popisuje v první kapitole důležité technologie použité při vývoji této aplikace od platformy WPF, která slouží jako základ pro vzhled aplikace až po knihovnu LiveCharts2, která má za úkol vytvářet grafy pro vizualizaci. Ve druhé kapitole jsou popsány různé protokoly a rozhraní pro realizaci sériové komunikace, včetně rozhraní UART, které se nejčastěji používá v kontextu zařízení používajících vizualizace. Ve třetí kapitole se porovnává vyvíjená aplikace s volně dostupnými řešeními pro vizualizaci dat ze sériové komunikace. V poslední čtvrté kapitole teoretické části dojde k podobnému porovnání jako ve třetí kapitole, ale zde se text zaměří na aplikace, které se používají nebo používali v rámci firmy.

Praktická část počínaje pátou kapitolou se věnuje definici samotné architektury systému a jejího datového modelu. Postupně popisuje jednotlivé třídy od základu a snaží se vysvětlit, jak jednotlivé části aplikace mezi sebou interagují. Následuje šestá kapitola, která navazuje na kapitolu předešlou a popisuje, jak na pozadí aplikace řeší některé stěžejní operace. Sedmá kapitola se snaží popsat, jak funguje aplikace na frontendu, neboli to co uživatel aplikace může sám vidět. V této kapitole je zahrnut větší počet obrázků, aby čtenář tohoto textu si mohl udělat představu o fungování aplikace i bez jejího používání.

1 POUŽITÉ TECHNOLOGIE

První kapitola se věnuje technologiím, které byli využity k tvorbě aplikace, jenž je předmětem této práce. Jednotlivé technologie jsou rozebrány od základu, o co se vlastně jedná až do detailních vlastností, co daná technologie umožňuje. Nejprve se text podrobně zaměřuje na alternativu k Windows Forms WPF, na kterou navazuje popis jazyka XAML. Dále se text věnuje frameworku MahApps.Metro, jehož nástroje rozšiřují možnosti WPF. Následně je zmíněna knihovna LiveCharts2, která realizuje tvorbu grafů v aplikaci. Na konec popisuje známý formát JSON, který hraje klíčovou roli v definici uživatelského rozhraní a také sdílení konfigurace mezi různými počítači.

1.1 WPF

Windows Presentation Foundation je framework pro tvorbu komplexních formulářových aplikací, který do jisté míry nahrazuje starší Windows Forms. Vznikl jako řešení pro vývojáře a grafické návrháře, kteří chtějí vytvářet moderní uživatelské aplikace bez nutnosti se učit různé složité technologie. Reaguje na popularitu HTML a JavaScriptu, přičemž přináší podobnou jednoduchost na platformu Windows. [1]

Základ WPF lze rozdělit na 4 bloky.

1. **Model aplikace** – skládá se z velké části z ovládacích prvků a jeho úlohou je hostovat a zobrazovat obsah aplikace.
2. **Systém rozložení** – definuje, jak jednotlivé komponenty mají být v aplikaci uspořádány.
3. **Datová vazba** – usnadňuje integraci uživatelského rozhraní a dat zpracovávaných na pozadí.
4. **Média, grafika a animace** – obsahuje všechny možné vizuální efekty, kterými lze v rámci WPF vylepšit vzhled aplikace. [2]

1.1.1 Historie WPF

První vydání WPF mělo označení WPF 3.0, jelikož bylo součástí .NET Frameworku 3.0 a vyšlo v listopadu 2006. Druhé vydání nesoucí název WPF 3.5 přišlo o rok později. Třetí vydání bylo součástí Service Pack 1 pro .NET 3.5, které obsahovalo mnoho vylepšení a dostavilo se po roce od vydání druhé verze 3.5. První verze měla velice limitovanou paletu nástrojů. Postupně byla

představena rozšíření pro Visual Studio, zajišťující plnohodnotnou podporu pro vývoj WPF. [1]

1.1.2 Vlastnosti WPF

WPF nabízí mnoho různých vlastností a funkcí, jednou z těch hlavních je široká integrace. Před příchodem WPF se musel vývojář pro použití 3D videa, řeči a pokročilých zobrazování dokumentů spolu s 2D grafikou naučit několik na sobě nezávislých technologií s řadou nekonzistencí. Navíc systémy dříve neměli tak dobrou podporu technologií pro vývoj různých typů grafiky, což komplikovalo jejich kombinaci a použití. Příchod WPF zajistil pokrytí všech těchto oblastí a zajistil konzistentní programovatelný model. Stejné efekty jsou aplikovatelné konzistentně napříč různými typy médií a mnoho technik, které se uživatel naučí lze aplikovat napříč různými oblastmi. [1]

Jednou z dalších vlastností je nezávislost na rozlišení. WPF umožňuje přecházet mezi rozlišeními bez ovlivnění kvality komponent uživatelského rozhraní. Kupříkladu grafické rozhraní na malém notebooku bude vypadat stejně dobře jako na širokoúhlé televizi. Z velké části je toto možné díky velkému důrazu WPF na vektorovou grafiku. WPF obsahuje paletu standardních vektorově vykreslovaných 2D tvarů. K těm základním si uživatelé mohou vytvářet i vlastní tvary pro specifické úpravy uživatelského rozhraní. Toto je možné díky vlastní geometrii, kterou WPF poskytuje. Mimo jiné vlastní definované tvary lze dále použít jako štětce nebo jako šablonu pro oříznutí jiných tvarů. Kromě 2D grafických možností je možné také tvořit vizuální efekty. Například vytvářet barevné přechody, použití videa jako malířského nástroje, rotace, škálování a další. Samozřejmostí je i 3D vykreslování, které umožňuje vytvářet zajímavá uživatelská rozhraní s neobvyklými efekty. Většina tříd ve WPF je animovatelných a to i včetně vlastních tříd. S třídami je možné provádět efekty jako je například otáčení, zmizení, třesení, zvětšení nebo přemisťování se na ose. Někdy prvky, které WPF nabízí, nemusí stačit požadavkům aplikace a proto WPF umožňuje využití dalších mechanismů pro vytvoření jedinečného uživatelského rozhraní. [1]

WPF je postaveno na technologii Direct3D, takže elementy aplikace ať už jde o 2D, nebo 3D jsou převáděny na 3D trojúhelníky, textury a jiné Direct3D objekty, které jsou poté vykreslovány pomocí hardwaru. To znamená, že WPF aplikace těží z hardwarové akcelerace pro výkonnější grafické zpracování rozhraní. Zaručuje také benefity získané z nového hardwaru a ovladačů pro všechny aplikace. Nicméně WPF nevyžaduje výkonný hardware pro

vykreslování, obsahuje totiž také softwarové vykreslování. To umožňuje funkce nepodporované hardwarem jako kvalitní vykreslení jakéhokoli obsahu na obrazovce nebo převzetí vykreslování při nedostatečných hardwarových prostředcích. [1] [2]

WPF využívá pro deklarace Extensible Application Markup Language, který se zkráceně označuje jako XAML. Kombinace WPF a XAML je podobná použití HTML pro definování uživatelského rozhraní, s tím rozdílem, že je k dispozici mnohem více možností úprav. WPF používá XAML pro formát dokumentů, reprezentaci 3D modelu a dalších věcí. Pomocí tohoto jsou schopni vývojáři měnit vzhled nebo chování částí aplikací, aniž by museli psát kód navíc. [1] [2]

Ovládací prvky WPF lze sestavovat způsoby, které nebyly dosud možné. Například lze vytvořit ComboBox naplněný animovanými tlačítky nebo komponentu Menu naplněnou video klipy. Nejdůležitější je, že autor nemusí psát velké množství zdrojového kódu, v některých případech dokonce žádný, aby radikálně upravil vzhled aplikace. [1]

V souhrnu se WPF snaží kombinovat to nejlepší ze systémů jako je DirectX, Windows Forms, Adobe Flash a HTML. To vše kvalitně, flexibilně a jednoduše pro zaručení co největší produktivity. [1]

1.2 XAML

XAML je jednoduchý a univerzální programovací jazyk vhodný pro vytváření objektů. Jedná se vlastně o obohacený jazyk XML. Oproti XML jsou zde pravidla pro elementy a jejich atributy, které jsou mapovány na objekty. XAML sám o sobě nedefinuje žádné předdefinované elementy, jako jsou tlačítka, panely nebo okna. Pouze obsahuje pravidla pro to, jak by měly kompilátory zpracovávat deklarované objekty. Oproti většině značkovacích jazyků, které jsou většinou bez přímé vazby na typový systém, představuje instanci objektů ze specifických typů definovaných v knihovnách. Textová reprezentace XAML souborů je ve formátu XML s příponou .xaml. Běžně používaným kódováním pro tento typ souborů je UTF-8. [1] [3]

```
<!-- Window Definition -->
<Window x:Class="Visu_NG.Examples.XAML_syntax"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:Visu_NG.Examples"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Title="XAML_syntax">

    <!-- Container -->
    <StackPanel Margin="10">
        <!-- Elements  -->
        <TextBox x:Name="InputBox" Width="200" Margin="0,0,0,10" />
        <Button Content="Add To List" Width="200" Click="AddItem_Click" />
        <ListBox x:Name="ItemList" Margin="0,10,0,0" />
    </StackPanel>
</Window>
```

Kód 1 – Ukázka syntaxe XAML

Vztah mezi XAML a WPF bývá někdy nepochopen, tyto dvě technologie pracují nezávisle na sobě. XAML může být využíván jinými technologiemi, přestože byl původně navržen pro WPF. Díky svojí obecnosti může být integrován s jakoukoli objektově orientovanou technologií. Navíc XAML je volitelný způsob, jak definovat objekty a jejich vlastnosti. Vše, co lze udělat v XAML, lze udělat i v jiných procedurálních jazycích. [1]

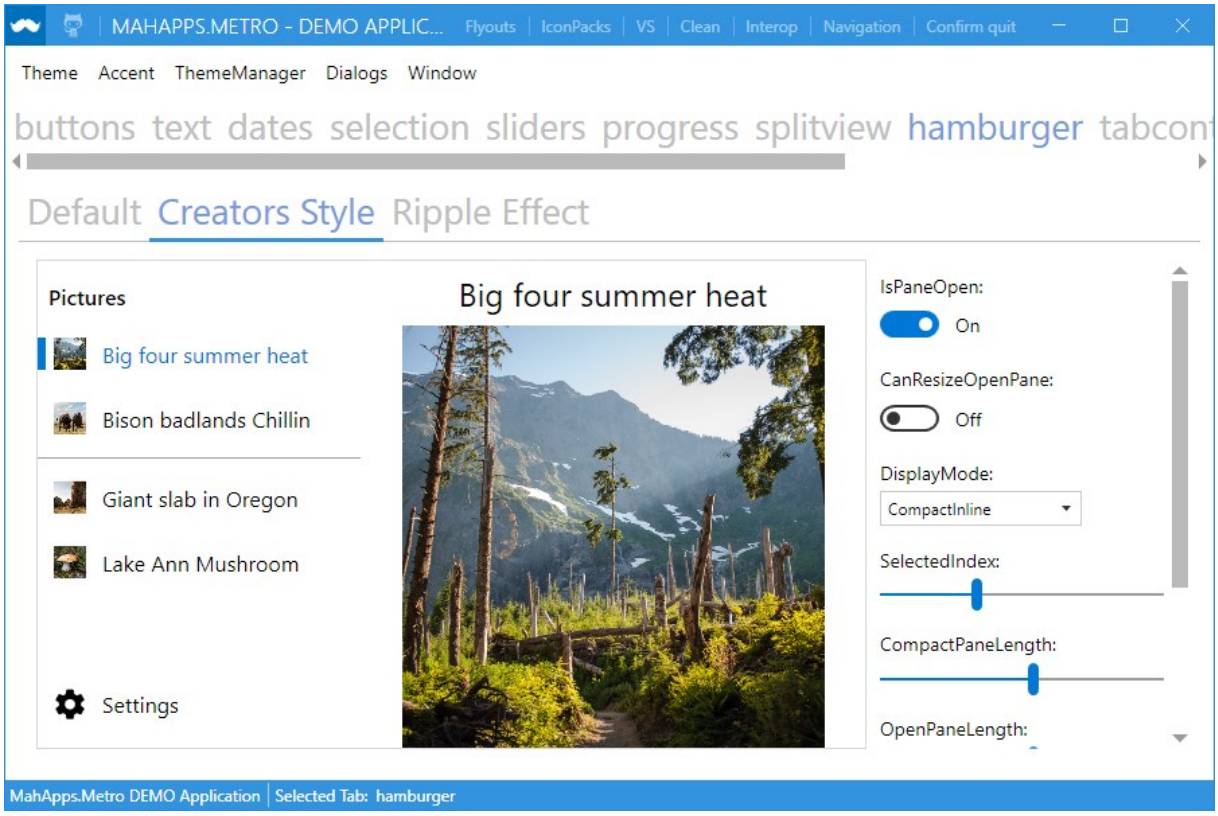
Cílem jazyka XAML je usnadnit spolupráci vývojářů a specialistů z ostatních oblastí. Schopnost jazyka být čitelný pro člověka usnadňuje lidem podílet se na vývojovém ekosystému bez nutnosti užití speciálních nástrojů. Většinou specialisté bývají grafičtí návrháři a XAML jim umožňuje jednoduše spolupracovat s vývojáři na backendu. Zatímco grafici mají na starost navrhnout přívětivé uživatelské rozhraní, vývojáři mezitím mohou psát nezávisle kód. To umožňuje navrhovat komplexní stylová rozhraní, jako jsou složité animace nebo stavové změny bez obavy z implementace procedurálního kódu. XAML podporuje separaci frontendové implantace od backendové logiky, což usnadňuje údržbu a nezáleží, zda na systému pracuje tým vývojářů nebo pouze jeden člověk. [4]

1.3 MahApps.Metro

Framework MahApps.Metro slouží jako rozšíření pro WPF a usnadňuje vývojářům tvořit moderní uživatelská rozhraní. MahApps.Metro je open-source a všechny jeho soubory jsou umístěné veřejně v GitHub repositáři. Obsahuje celou řadu nástrojů pro personalizaci aplikací

běžících na Windows Phone, Windows 8, Windows 10 nebo Windows 11. Mezi zajímavé nástroje a ovládací prvky, které stojí za zmínku patří například:

- Možnost definice vlastní tlačítek pro zavření, minimalizaci či maximalizaci okna.
- Flyouts, což jsou postranní panely, které se vysouvají při najetí myší k okraji okna.
- Stylizovaný přepínač ToggleSwitch, který nahrazuje standardní CheckBox a poskytuje moderní vzhled ve stylu známého z mobilních zařízení.
- ProgressRing představující načítaní nějaké operace, jedná se o moderní pojetí klasického načítacího kolečka.
- RangeSlider, který může být užitečný pro jednoduché zadávání číselných pomocí posuvníku na ose. [5]



Obrázek 1 – Ukázka okna MahApps.Metro, Zdroj: [6]

MahApps.Metro přepisuje výchozí styly definované za pomoci WPF a nahrazuje je vlastními atributy, což zvyšuje míru přizpůsobení jednotlivých komponent. Nejenom, že dokáže obohatit komponenty již existující ve WPF, ale jak bylo zmíněno výše sám přidává nové zajímavé elementy. Přitom integrace obou frameworků je jednoduchá a bez nutnosti jakékoliv složitější konfigurace. Výhodou je také dobře zpracovaná dokumentace popisující všechny komponenty a aktivní komunita vývojářů, která dále framework vylepšuje a poskytuje podporu. [5]

1.4 LiveCharts2

Představuje moderní knihovnu pro vizualizaci dat, která umožňuje vytvářet interaktivní grafy. Její návrh je kompatibilní s mnoha platformami, včetně WPF. Jedná se o rozšíření starší knihovny LiveCharts, oproti které vylepšuje flexibilitu a výkonnost v zobrazování dat. Tato knihovna byla zvolena hlavně z důvodu potřeby zpracovávat velké objemy dat přicházející v krátkých časových intervalech. Velkou výhodou je také licence MIT, která dovoluje používat, upravovat a distribuovat danou knihovnu bez nutnosti platit licenční poplatky. [7]

1.5 JSON

JavaScript Object Notation je textový formát s definovanou strukturou pro výměnu dat, který je univerzální a lze ho použít na více platformách. Navazuje na formát XML, který je standardizovaný a slouží pro výměnu dat mezi různými systémy. Systémy mají vlastní jazyky a způsoby zpracování dat, proto tyto standardizované formáty zajišťují, že systémy si mezi sebou porozumí. Tato schopnost je klíčová pro mnoho firem a organizací, jelikož bez společného standardu by každý systém potřeboval překladač pro všechny různé způsoby, jak definovat formát dat a to by způsobilo enormní spotřebu času a zdrojů. [8]

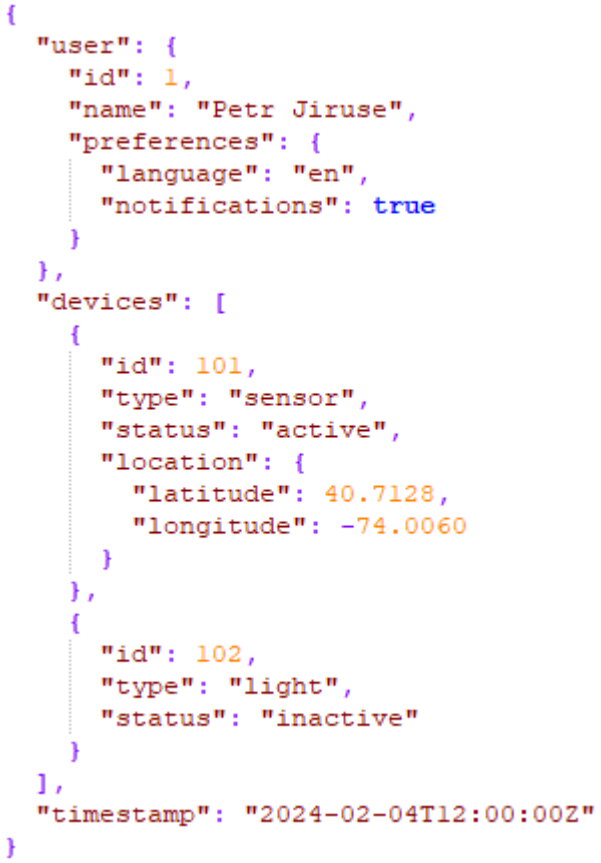
Formát JSON je založen na JavaScript literálech a jedná se o jistou podmnožinu JavaScriptu, ale pro používání JSONu není nutné se předem učit JavaScript. Přestože by měl být JSON univerzální a programově nezávislý, tak vychází z konkrétního programovacího jazyka. Důvod je ten, že i když se člověk učí programovat, začíná obvykle s jedním programovacím jazykem. Jakmile porozumí základním konceptům programování v daném jazyce, může přecházet mezi různými dalšími jazyky bez potřeby učit se základy znovu. Z toho důvodu se přebral již existující formát namísto toho, aby se vymýšlelo něco nového. Pokud by se odebral kompletně JavaScript, zůstal by Object a Notation, což je vlastně to nejdůležitější, protože Object je naprostý základ všech OOP jazyků a Notation reprezentuje systém znaků jakým jsou vyjadřovány čísla nebo slova. JSON vyjadřuje data stylem, kterému je schopno porozumět mnoho jazyků, a to i ty, které nejsou objektově orientované. [8]

Data ve formátu JSON jsou reprezentována jako klíč-hodnota. Nejprve je tedy uveden název a k němu nějaká hodnota. Hodnota může nabývat řetězce, čísla, boolean (true/false), null, pole nebo objektu. Pro oddělení dvojice klíč-hodnota se používá dvojtečka, kde klíč je vždy nalevo a hodnota napravo. Název hodnoty je vždy uveden ve dvojitých uvozovkách a pokud hodnota nabývá typu řetězce, tak je také ve dvojitých uvozovkách, ostatní typy hodnot se uvádí bez nich.

Vytvoření objektu z jednotlivých dvojic klíč-hodnota už je v celku jednoduché stačí přidat složené závorky, které zapouzdří atributy do objektu. Důležitou věcí je oddělovat jednotlivé páry klíč-hodnota čárkou a to samé platí i u jednotlivých objektů. Z pohledu systému se text JSONu čte následovně:

- **Levá složená závorka {** – začíná objekt
- **Pravá složená závorka }** – končí objekt
- **Levá hranatá závorka [** – začíná pole
- **Pravá hranatá závorka]** – končí pole
- **Dvojtečka :** – následuje hodnota klíče, oddělení hodnoty od klíče
- **Čárka ,** – následuje další položka, oddělení hodnot v poli nebo jednotlivých objektů [8]

Syntaxe musí odpovídat a každé závorky ať už hranaté nebo složené musí být párové, dále je potřeba začínat levou závorkou a končit pravou, jinak objekt nebo pole nebudou rozpoznány.



Obrázek 2 – Syntaxe JSON, Zdroj: vlastní

Na vzorovém obrázku lze pozorovat, jak vypadá standardní struktura souboru typu JSON. Soubor začíná hlavním objektem, tudíž je potřeba, aby soubor začínal levou složenou závorkou a končil pravou složenou závorkou. Následují poté vnořené objekty, které organizují data

do sekcí. Uvnitř vnořených objektů mohou být další vnořené objekty nebo pole, které v případě většího počtu jsou oddělovány čárkami. Takto definovaná struktura zajistí, že systém správně přečte všechen obsah a zároveň obsah bude jednoduše čitelný i pro lidi.

2 ÚVOD DO KOMUNIKAČNÍCH ROZHRANÍ

Komunikace v souvislosti s integrovanými obvody může být poněkud komplexní téma a je zde hodně pojmů, které si člověk může zaměnit a špatně si vyložit principy. Proto se tato kapitola věnuje vysvětlení základních pojmů a také rozdíly mezi různými typy komunikace. Existuje celá řada protokolů, sběrnic a rozhraní, které v této kapitole budou rozebrány.

Základní stavební kameny komunikace jsou: rozhraní, sběrnice a protokol. Rozhraní je elektronický obvod, který přenáší signály mezi dvěma systémy. Takové rozhraní většinou může signály přijímat nebo vysílat a patří na fyzickou vrstvu, kde má vývody nebo konektory, ale zároveň patří i na funkční vrstvu, kde kóduje a dekóduje zprávy dle určitého standardu. Sběrnice představuje fyzické cesty nebo vodiče, díky nim se propojují jednotlivé komponenty. Jsou zde linky seskupené podle funkce, jako jsou adresové linky nebo datové sběrnice. Sběrnice je podobná rozhraní, ale oba prvky se vztahují k jiné části komunikačního systému. Rozhraní připojuje konkrétní obvod ke sběrnici. Protokol definuje soubor pravidel, dle nich se tvoří způsob komunikace. Popisuje, jakým způsobem si jednotlivé části systému mají vyměňovat informace a v jakém formátu jsou informace přenášena. Nad protokolem je norma, která většinou popisuje celek fungování trojice rozhraní, sběrnice a protokol. [9]

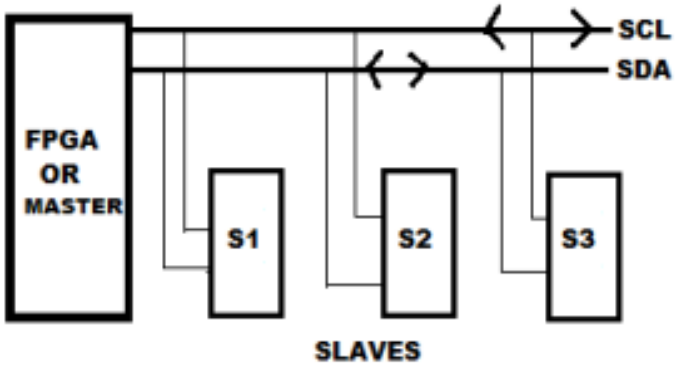
Při komunikaci se rozlišují dva druhy přenosů, sériový a paralelní přenos. Sériový přenos posílá data postupně po jednom bitu, kde se začíná nejméně významným bitem. Přenášená jednotka je označovaná jako znak v rozsahu 7 nebo 8 bitů. Sériový přenos se dá dále specifikovat na synchronní nebo asynchronní. Synchronní přenos pracuje tak, že se vezmou celé bloky znaků, kde datové bity jednotlivých znaků následující těsně za sebou a není mezi nimi časový odstup. Začátek bloku je označen synchronizačním znakem SYN. Cílem tohoto znaku je definovat časové okamžiky, ve kterých se vyhodnocují datové bity a synchronizují tak příjemce i odesílatele. Blok znaků může být zakončen dalšími synchronizačními znaky, ty jsou posílány až do momentu zpracování nového bloku. Pro asynchronní přenos platí, že znaky jsou naopak přenášeny s proměnlivými časovými odstupy mezi sebou. Je nutné určit kdy začíná další znak, proto je zde příznak tzv. start-bit, kterým začíná každý přenášený znak. Při jeho příchodu si příjemce musí správně nastavit měřítko času, aby byl schopný dodržet časové okamžiky pro vyhodnocování datových bitů, které jsou posílány po start-bitu. Na konci přenosu za datovými bity se nachází parita pro detekci chyb při přenosu a nakonec stop-bit, který neobsahuje žádnou informaci a zajišťuje tak minimální odstup mezi jednotlivými znaky, jelikož vysílání dalšího znaku může začít až po odvysílání stop-bitu. U paralelního přenosu jsou data

posílána po více bitech najednou. Pro přenos je proto potřeba odpovídající počet vodičů pracujících souběžně. To se hodí na krátké vzdálenosti, typicky do 20 metrů. [10]

Při vytváření elektronických rozhraní je kladen důraz na jednoduchost použití a funkčnost. Mezi nejpoužívanější sériové protokoly patří I2C, SPI, USB, UART nebo RS-485, které jsou popsané v dalších kapitolách. [9]

2.1 I2C

Protokol vyvinutý firmou Philips jako univerzální řešení pro obousměrnou komunikaci mezi integrovanými obvody v elektronických systémech. Využívá dva vodiče pro komunikaci. Konkrétně se jedná o Serial Clock (SCL) a Serial Data (SDA), kde oba vodiče jsou poloduplexní a přenášejí data mezi zařízeními připojenými na sběrnici. Každé zařízení je identifikováno jedinečnou adresou, ať už se jedná třeba o mikrokontrolér, paměť nebo rozhraní klávesnice.



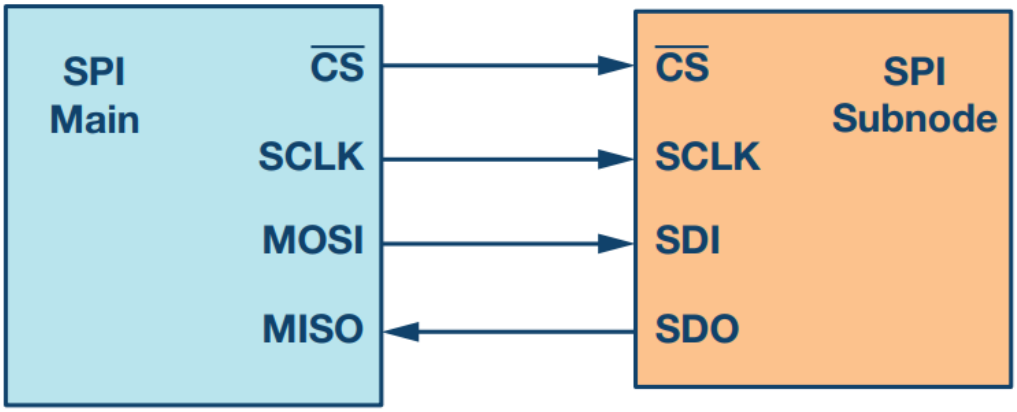
Obrázek 3 – Nákres sběrnice I2C, Zdroj: [11]

Cílem protokolu je umožnit vysokorychlostní komunikaci a zajistit efektivní správu registrů uvnitř zařízení, včetně manipulace dat, které jsou v nich uložena. Pomocí této správy lze nastavovat různé parametry. Komunikační protokoly typu UART, SPI nebo USB používají způsob přenosu point-to-point (komunikace probíhá mezi dvěma body), na rozdíl od nich I2C používá softwarové adresování, což zajišťuje jednoduchou implementaci a údržbu. Mezi hlavní výhody I2C patří jednoduchost, podpora plug-and-play, cenová dostupnost a široká podpora zařízení. I2C používá vylepšený bezpečností systém, proto se hodí pro připojení senzorů nebo biometrických zařízení. Nevýhodou může být počet připojených zařízení omezený kapacitou sběrnice. [11]

2.2 SPI

Sériové komunikační rozhraní SPI patří mezi nejpoužívanější rozhraní pro digitální obvody. Používá se jako rozhraní mezi mikrokontrolerem a periferními integrovanými obvody. Do těchto obvodů lze zahrnout například senzory, ADC a DAC převodníky, paměti SRAM a jiné. Komunikace na SPI je synchronní a plně duplexní. Základ představuje vztah hlavního a podřízeného uzlu, kde data z jednotlivých uzlů se synchronizují v okamžiku, kdy se komunikace nachází na vzestupné nebo sestupné hraně hodinového signálu. Díky plně duplexní komunikaci hlavní i podřízený uzel mohou posílat data současně. Rozhraní SPI může obsahovat 3 nebo 4 vodiče. V případě 4 vodičů, SPI může vyvolat 4 různé signály:

1. Signál SCLK (Serial Clock) pro hodiny
2. Signál CS (Chip Select) pro výběr zařízení
3. Signál MOSI (Master Out Slave In) pro přenos dat z hlavního uzlu podřízenému
4. Signál MISO (Master In Slave Out) pro přenos dat z podřízeného uzlu hlavnímu



Obrázek 4 – Nákres zapojení hlavního a podřízeného uzlu, Zdroj: [12]

Hlavní uzel generuje hodinový signál, ten synchronizuje přenos dat mezi hlavním a podřízeným uzlem. Hodinový signál může být vysílán na mnohem vyšší frekvenci, než je tomu třeba u rozhraní I2C. Hlavní uzel může být vždy pouze jeden, ale podřízených uzlů může být větší množství. Začátek komunikace na SPI probíhá nejprve spuštěním generování hodinového signálu a je třeba vybrat, který podřízený uzel bude aktivní. Pomocí signálu CS směrem z hlavního do podřízeného uzlu se vybírá, který to bude. Oba uzly, jak hlavní, tak podřízený, mohou současně posílat data pomocí signálů MOSI a MISO. Data jsou posouvána na sběrnici MOSI/SDO a zároveň vzorkována a čtena ze sběrnice MISO/SDI. Posun a vzorkování je synchronizováno na hraně hodinového signálu. U rozhraní SPI lze nastavit, zda proběhne posun a vzorkování na vzestupné nebo sestupné hraně hodinového signálu. [12]

2.3 USB

Pravděpodobně nejrozšířenější rozhraní pro osobní počítače a jejich periferie. Úspěch rozhraní spočívá hlavně v neustálém zlepšování, spolehlivosti, energetické úspornosti, podpoře operačními systémy nebo cenové dostupnosti. Klíčové je také, že USB dokáže uspokojit potřeby jak běžných uživatelů, tak i vývojářů, kteří píšou kód pro komunikaci mezi zařízeními. Pro uživatele je hlavní, že USB je jednoduché na použití. Tudíž jim stačí zapojit libovolné zařízení do počítače a operační systém detekuje, o jaké zařízení se jedná. Ve většině případů se automaticky načte ovladač příslušného zařízení. Pokud toto selže je uživatel vyzván manuálně vybrat ovladač. Také se nemusí bát zařízení kdykoliv odpojit bez toho, aniž by způsobilo jakékoliv škody na operačním systému. Pro vývojáře je hlavní univerzálnost rozhraní, kde díky čtyřem typům přenosů a více než pěti rychlostem v závislosti na verzi je USB vhodné pro mnoho různých zařízení. Podpora operačních systému včetně základních ovladačů je také důležitá, jelikož vývojář nemusí vytvářet veškerou logiku od začátku. Standardy USB obsahují kontrolu chyb na úrovni hardwaru, tudíž vývojáři nemusí vytvářet vlastní protokoly pro kontrolu chyb. [13]

Pro realizaci komunikace přes USB je zapotřebí počítač s podporou USB a zařízení, které disponuje portem USB. Hostitelský systém nemusí být jen počítač, ale jakékoliv embedded nebo přenosné zařízení, které má USB hardware hostitelského řadiče a kořenový hub. Hostitelský řadič má na starosti přenos, formátování a převod dat do formátu, kterému porozumí operační systém. Hostitelský řadič spolupracuje s kořenovým hubem pro detekci připojených a odpojených zařízení. Zařízeních může být v jeden okamžik připojeno více, jelikož kořenový hub obvykle disponuje více konektory pro připojení zařízení. [13]

Při započetí komunikace na USB je potřeba, aby hostitelský počítač inicioval přenos s určitým předdefinovaným formátem pro odesílání dat, adresy, bitů detekujících chyby a informací o stavu zařízení. Rozlišují se různé formáty pro odesílání nebo přijímání dat a to samé platí i o různých typech přenosů. Provoz na sběrnici má na starosti hostitel a zařízení reaguje na jeho komunikaci. Důležitou komponentou je koncový bod neboli endpoint, který je vyrovnávací pamětí a má za úkol uchovávat všechny data, která jsou určena k přijetí nebo k odeslání. Parametry endpointu jsou adresa, číslo, směr přenosu a počet bajtů udávající maximální velikost dat, které jsou možné v jednom přenosu odeslat nebo přijmout. V případě USB verze 2.0 začínají transakce pomocí tokenového paketu, který vyšle hostitel. Paket určuje, jakému endpointu se data budou posílat a jaký bude směr přenosu. Po tokenovém paketu se začínají

posílat datové pakety obsahující identifikátor paketu, sekvenci dat a kontrolní bity. Potvrzení přijetí dat se provádí potvrzovacím paketem tzv. handshake packet. [13]

V USB komunikaci se lze setkat se čtyřmi typy přenosů:

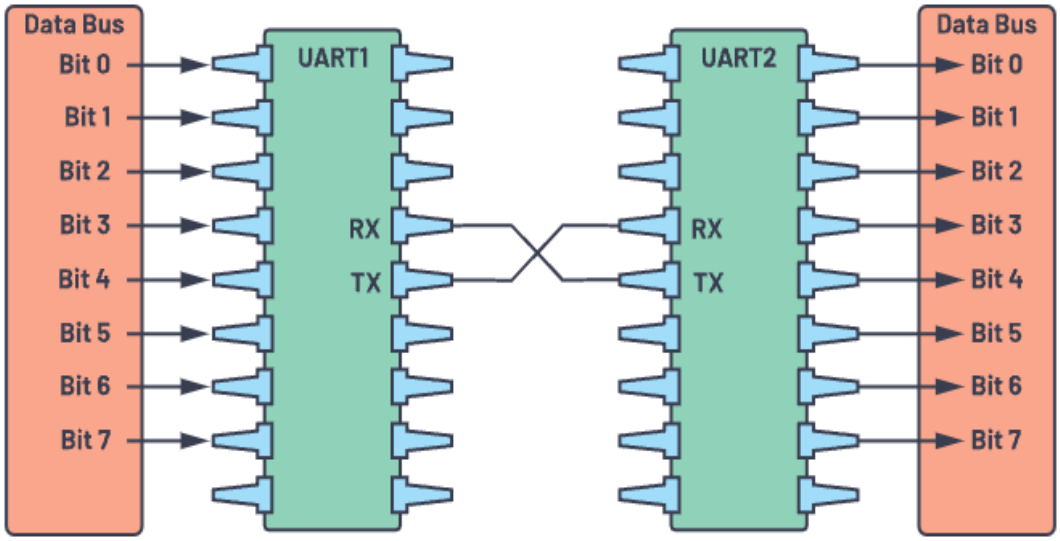
1. **Řídící přenosy** – používají se pro odesílání požadavků od hostitele k zařízení. Pomocí řídících přenosů si hostitel může zobrazit sérii datových struktur, jinak nazývaných deskriptory. Deskriptor udává, co všechno je zařízení schopno dělat. Hostitel na základě deskriptorů dokáže určit jaký ovladač pro zařízení použít.
2. **Blokové přenosy** – jedná se o rychlé přenosy na volné sběrnici, ale nemají jasně definované časy provedení. Jsou využívány hlavně tiskárnami a disky.
3. **Přerušovací přenosy** – mají definovaný maximální čas, do kterého musí být doručeny. Využívají se pro vstupní periferie jako je například myš nebo klávesnice.
4. **Izochronní přenosy** – jsou přesně načasované na okamžik provedení, ale nejsou u nich kontrolovány chyby. Tyto přenosy se používají pro přenos zvuku a videa. [13]

2.4 UART

UART neboli Universal Asynchronous Receiver-Transmitter je komunikační protokol spolupracující s různými typy sériových protokolů, které podporují přenos a příjem sériových dat. Používá se ve velkém množství hlavně v embedded systémech, mikrokontrolerech a počítačích, kde definuje, jakým způsobem se data přenášejí mezi zařízeními. Vyžaduje pro své fungování pouze dva vodiče, jeden na odesílání dat (TX) a druhý na přijímání dat (RX). Komunikace probíhá asynchronně a sériově s nastavitelnou přenosovou rychlostí, tudíž zde neexistuje žádný hodinový signál a jednotlivé bity proudící z vysílacího zařízení k přijímacímu nejsou synchronizovány. [14]

Každé zařízení UART obsahuje dvojici Transmitter (TX) a Receiver (RX) neboli vysílač a přijímač. Účelem signálů je zpracování sériových dat pro sériovou komunikaci. Data jsou odesílána v paralelní podobě z datové sběrnice, ke které je vysílající UART připojen. Data se posílají posléze po transmisní lince sériově, jednotlivě bit po bitu směrem k přijímajícímu zařízení UART. Po přijetí sériových dat jsou data převedena zpět do paralelního formátu a cílové zařízení je může přečíst a dále zpracovat. Důležité je nastavit stejnou přenosovou rychlost tzv. baud rate na obou zařízeních, jak pro vysílání, tak pro příjem. Baud rate udává, jak rychle jsou data přenášena prostřednictvím komunikačního kanálu. V kontextu baud rate se zavádí jednotka bps (bits per second), která určuje maximální počet bitů za sekundu, které mohou být přeneseny. Na následujícím obrázku je možné vidět schéma zapojení mezi dvěma

zařízeními UART s jejich datovými sběrnicemi. Komunikace probíhá zleva doprava, kde jedno zařízení převede paralelní data na sériový signál a druhé jej rekonstruuje zpět do paralelní podoby. Za povšimnutí stojí, že piny TX a RX jsou propojené do kříže, což zařízením umožňuje vzájemnou komunikaci. [14]



Obrázek 5 – UART s datavou sběrnicí, Zdroj: [15]

Rozhraní UART nedisponuje hodinovým signálem pro synchronizaci mezi vysílačem a přijímačem, namísto toho vysílač generuje datový proud tzv. bitstream na základě svého vlastního hodinového signálu. Na druhé straně přijímač používá pro vzorkování příchozích dat svůj vlastní interní hodinový signál. Pro správnou synchronizaci mezi vysílačem a přijímačem je potřeba nastavit na obou uzlech stejnou hodnotu baud rate, pokud na jednom uzlu rychlost výrazně nesouhlasí, dochází k chybám v časování a data nejsou doručována. Na UART je možné tolerovat maximálně 10 % rozdílu v baud rate obou uzlů, poté je již odchylka moc velká a časování bitů chybné. [14]

Komunikace mezi zařízeními probíhá ve formě paketů. Při vysílaní je potřeba vytvořit sériový paket a řídit fyzické hardwarové linky. Struktura paketu je následující:

Obrázek 6 – Struktura paketu UART, Zdroj: [16]

Paket se skládá ze start bitu, rámce obsahující datové bity, parity a stop bitu. Linka UART je standardně udržována v logické úrovni 1. Při zahájení přenosu pomocí start bitu se UART stáhne na úroveň logické 0 na dobu jednoho hodinového cyklu. V ten moment přijímací uzel

detekuje změnu logické úrovně a začne číst datové bity, které přicházejí v rámcích, rychlostí, jakou definuje hodnota baud rate. V datovém rámci se nacházejí skutečná data, která mohou mít délku od 5 do 8 bitů. Zaleží také zda je použit paritní bit. Pokud se paritní bit nepoužívá délka rámce může být až 9 bitů. Obvykle se přenášejí datové bity počínaje nejméně významným bitem (LSB). Parita slouží k určení sudosti nebo lichosti logických jedniček, což slouží k identifikaci změn v datech během přenosu. Při přenosu mohou být data negativně ovlivněna například špatně nastavenou hodnotou rychlosti baud rate, vysokou vzdáleností přenosu nebo elektromagnetickým rušením. Vždy po přečtení celého datového rámce příjemce spočítá počet bitů s hodnotou 1 a určí, zda je počet sudý nebo lichý. Tento součet porovná s paritním bitem a pokud odpovídá přeneseným datům příjemce ví, že přenos proběhl bez chyb. Naopak pokud parita nesedí, došlo ke změně bitů při přenosu. Po skončení přenosu datového rámce odesílatel vyšle stop-bit, který nastaví hodnotu logické 1 po dobu jednoho až dvou bitů. [14]

U UART se lze setkat s funkcí rámcového protokolu, který se ne vždy naplno využívá, ale jeho použití zvyšuje bezpečnost a ochranu zařízení. Koncepce rámcového protokolu je navržená tak, aby byl unikátní a bezpečný. Pokud je protokol správně implementovaný, bezpečnost se zvýší, jelikož data musí být parsovaná v souladu s koncepcí protokolu. Může však nastat i situace, kdy je protokol použit a nastanou chyby nebo poruchy v systému. Například pokud dvě zařízení používají stejný rámcový protokol UART a dojde k připojení ke stejnému rozhraní UART bez kontroly konfigurace. To má za následek, že zařízení se připojí ke špatným pinům. Při vývoji rámcového protokolu lze jeho záhlaví, zakončení a kontrolní součet nastavit pro různá zařízení. Struktura rámcového protokolu je následující: [14]

Obrázek 7 – Struktura rámcového protokolu UART, Zdroj: [17]

Jednotlivé komponenty protokolu jsou:

- **Header 1 a Header 2** – dohromady tvoří jedinečný identifikátor, který určí, zda komunikace probíhá se správným zařízením.
- **Command** – vybírá příkaz na základě seznamu dostupných příkazů, které umožňují komunikaci mezi zařízeními.
- **Data Length** – udává délku přenášených dat podle vybraného příkazu, tudíž se tato délka mění v závislosti na výběru příkazu.

- **Data n** – užitečná data o proměnlivé délce, které je potřeba přenést.
- **Trailer 1 a Trailer 2** – část dat přidaná na konec, podobně jako u záhlaví může být tato část unikátní, ale nemusí.
- **CRC** – kontrolní součet, který slouží k detekci chyb v přenosu. Hodnota vysílaného CRC musí být vždy shodná s hodnotou CRC na straně příjemce. [14]

Hodnoty záhlaví, zakončení a CRC musí být unikátně nastaveny pro používané zařízení a tato konfigurace musí být identická jak na vysílacím, tak i přijímacím zařízení. [14]

UART je hojně používaný při vývoji produktů, kde pomáhá zajistit jejich spolehlivost a kvalitu. Disponuje mnoha funkcemi klíčových pro podporu vývoje. Přidáním UART je možné zachytávat diagnostické zprávy od systému a odhalovat tak včas chyby. Uchovávají se také logy nezbytné pro monitoring funkčnosti a případné zasílání upozornění při nestandardních stavech. Při průběžných aktualizacích lze použít rozhraní UART pro dynamické aktualizace a zajistit tak udržovaný a flexibilní systém. To vše napomáhá při vývoji a testování produktů před dokončením výroby a zajistit tak kvalitní produkty pro koncové zákazníky. [14]

2.5 RS-485

Preferovaný standard pro průmyslovou automatizaci, který vznikl již v roce 1983. Realizuje sériovou komunikaci, která je spolehlivá při středních rychlostech, podporuje vícebodovou komunikaci a dokáže realizovat přenosy na dlouhé vzdálenosti. Samotný standard RS-485 nedefinuje komunikační protokol, ale pouze obsahuje elektrické charakteristiky pro vysílání a přijímaní. Vyšší komunikační standardy ho často využívají pro svůj základ, jako fyzickou vrstvu, na které dále stavějí. Mezi takové protokoly patří například ModBus, ProfiBus nebo DMX512. [18]

Oproti svému předchůdci RS-232 rozšiřuje RS-485 dosah signálu a zlepšuje odolnost proti šumu, jelikož používá diferenciální způsob přenosu signálu namísto jednovodičového přenosu. Na rozdíl od RS-422 umožňuje obousměrnou komunikaci. Dále také RS-485 používá vyšší úrovně signálů a širší rozsah, což zajišťuje komunikaci na delší vzdálenosti. Nevýhodou je vyšší spotřeba energie při přenosových rychlostech nad 50 Mbps ve srovnání s konkurenčními řešeními jako je například M-LVDS. Topologie RS-485 se skládá z vícero uzlů, které jsou zapojeny paralelně k jedné sběrnici. RS-485 podporuje full-duplexní komunikace, ale je zapotřebí další pár vodičů, tudíž pro realizaci takového spojení je nutné vytvořit zapojení se čtyřmi vodiči. Odměnou za to je větší propustnost. [18]

3 PŘEHLED VOLNĚ DOSTUPNÝCH ŘEŠENÍCH

Tato kapitola stručně popisuje existující řešení pro vizualizaci a konfiguraci zařízení s využitím sériové komunikace. Na trhu existuje celá řada nástrojů pro správu sériové komunikace, pro porovnání byli vybrány aplikace ze dvou kategorií. První kategorie slouží pro monitoring a ladění přenosu dat, do této kategorie patří nástroje Putty a RealTerm. Druhou kategorii zastupují aplikace Serial Studio a SerialTool, které poskytují více funkcí včetně pokročilejší vizualizace. Zmíněné nástroje a aplikace jsou v této kapitole porovnávány s budovanou aplikací. Nejprve je krátce vysvětleno, co daný software umí, následováno srovnáním, co jaká aplikace dělá lépe nebo hůře.

3.1 Serial Studio

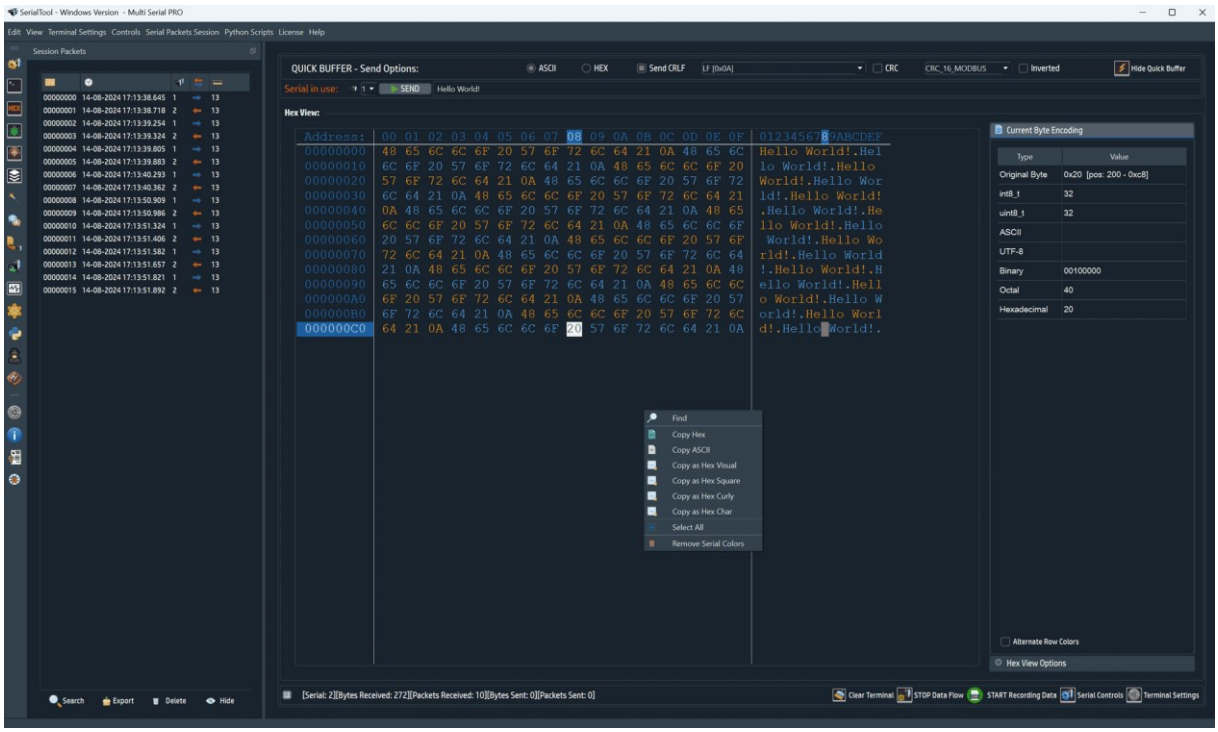
Jedná se o open-source projekt, na kterém pracuje komunita lidí. Umožňuje vizualizovat a analyzovat data bez žádných dalších specializovaných nástrojů. Snaží se pokrýt širokou škálu zařízení a experimentů s nimi. Serial Studio se tudíž zaměřuje na širokou skupinu lidí, kteří pracují s embedded systémy, kdežto budovaná aplikace slouží jako interní aplikace pro zaměstnance firmy. Obě aplikace nabídnou vizualizaci dat ze sériové komunikace, ale u Serial Studio není jisté, zda bude umět komunikovat se zařízeními firmy Steinel. Navíc budovaná aplikace má exkluzivní schopnost konfigurovat daná zařízení firmy, kdežto Serial Studio tuto schopnost nemá. Obrázek 8 zobrazuje uživatelské rozhraní softwaru Serial Studio.

Grafické komponenty v nástroji Serial Studio jsou předdefinované, zatímco v budované aplikaci má uživatel možnost si přizpůsobit část rozhraní podle sebe. Serial Studio podporuje nastavení pouze pro anglický jazyk, naproti tomu budovaná aplikace poskytuje výběr mezi jazyky – angličtina, čeština a němčina. Dále umožňuje přidání dalších jazykových verzí, což jí přidává flexibilitu a zlepšuje uživatelský zážitek. Aplikace Serial Studio je napsaná v jazyce C++ a Qt, což jí umožňuje běžet na různých operačních systémech. Budovaná aplikace je vázaná výhradně na operační systém Windows. Závěrem lze říci, že Serial Studio je obecná aplikace, která se snaží zajistit vizualizaci pro co nejvíce embedded zařízení, zatímco budovaná aplikace se specializuje na zařízení firmy Steinel. [19]

Obrázek 8 – Vizuální podoba aplikace Serial Studio, Zdroj: [19]

3.2 SerialTool

Představuje multiplatformní software pro správu sériové komunikace dostupný na platformách Windows, macOS a Linux. Aplikace má bezplatnou verzi, která má omezenou funkcionalitu. Verze premium dokáže nabídnout řadu pokročilých nástrojů, jako je například Python skriptování, alarm na obdržení určité hodnoty, funkce auto-answer pro vytváření automatické zpětné vazby na příjem sekvence bajtů koncovým zařízením a další. Grafická podoba tohoto nástroje je na obrázku níže (Obrázek 9).



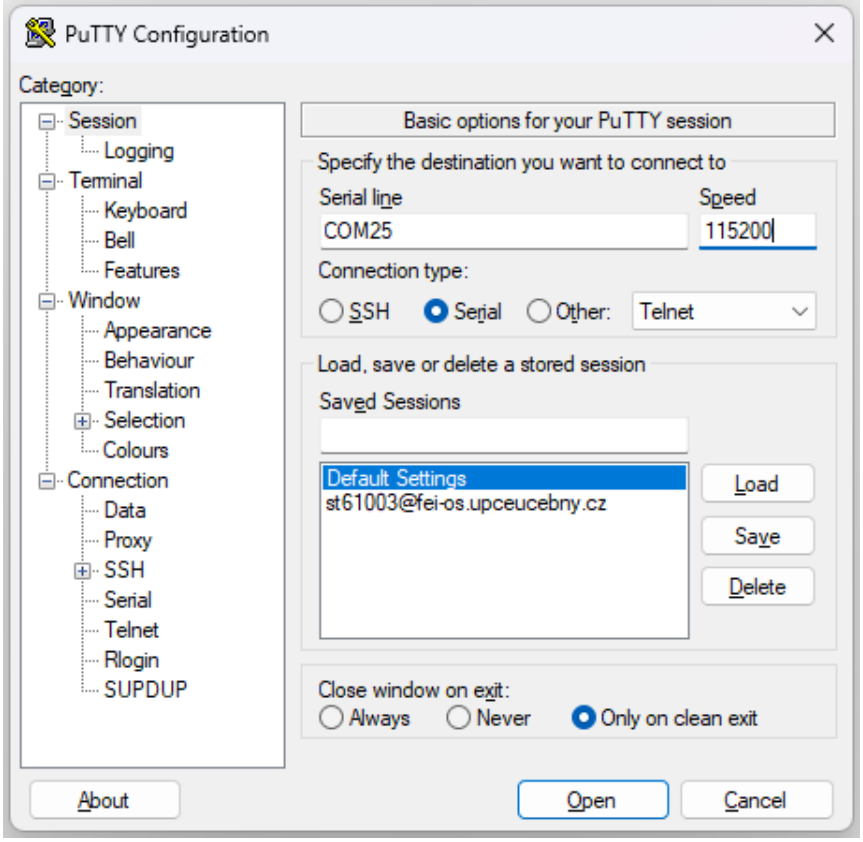
Obrázek 9 – Vizuální podoba aplikace SerialTool, Zdroj: [20]

SerialTool se zaměřuje na vývojáře desek Arduino všech úrovní. Snaží se být co nejjednodušším nástrojem pro správu a testování při vývoji těchto desek. Ve srovnáním s budovanou aplikací je na tom dost podobně jako v případě Serial Studio. SerialTool se snaží zaujmout širokou škálu uživatelů, zatímco budovaná aplikace je specializovaným softwarem. [21]

3.3 Monitorování komunikace za pomoci nástroje Putty

Pro rychlé zkontrolování sériové komunikace, lze využít také nástroj Putty i když to spolu přináší jisté omezení. Putty nabízí univerzální terminálový emulátor, kde jedna z jeho funkcí je sledování sériové komunikace ze specifikovaného portu. Jeho hlavní síla spočívá v jednoduchosti, široké kompatibilitě s různými systémy a účelovém rozhraním, jak je možné vidět na obrázku níže (Obrázek 10). Oproti budované aplikaci nenabízí žádnou možnost vizualizace dat. Sledovat je možné pouze nezpracovaná příchozí data, která se v reálném čase vypisují do konzole. Z toho je patrné, že Putty nenabízí žádné interaktivní uživatelské rozhraní a bez pokročilé vizualizace je pro uživatele těžké analyzovat nebo interpretovat získaná data. Budovaná aplikace nabízí v tomto směru pokročilé rozpoznání a zpracování příchozích paketů do snadno pochopitelné formy. Data jsou automaticky parsována při přijetí, kdežto u Putty

se musí příchozí data ručně zkopírovat a zpracovat v jiném softwaru, jako například Excel, Python nebo Matlab.

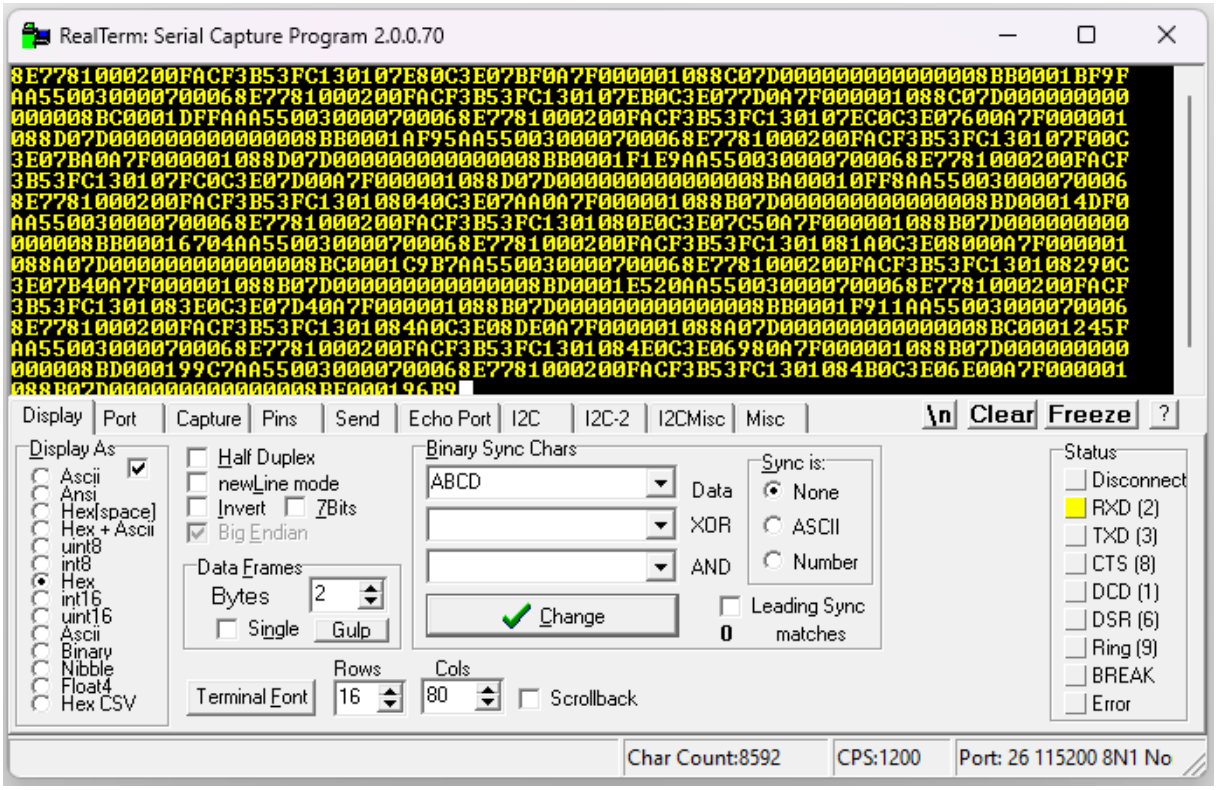


Obrázek 10 – Vizuální podoba nástroje Putty, Zdroj: vlastní

Velkou výhodou budované aplikace je několik automatizovaných funkcí, kde první je automatická detekce portů. Uživatel hned vidí v seznamu názvy dostupných portů včetně jejich popisu pro lepší představu, k čemu se vlastně chystá připojit. Při opětovném spuštění aplikace je zde automatické připojení k poslednímu používanému portu. Dále aplikace umožňuje nastavení přenosové rychlosti (Baudrate), kde rychlosti jsou předem definované. V porovnání Putty spoléhá na ruční nastavení všech výše zmíněných funkcí. Porty nejsou detekované automaticky, uživatel musí napsat číslo portu, o kterém musí předem vědět informace co je na něm zapojeno. Dále rychlost se musí nastavit ručně, kde může dojít k nastavení nekompatibilní hodnotě a přenos nebude fungovat nebo bude ztrácet rámce. Putty se hodí pro ověření, že data opravdu přichází na vybraném portu, tudíž je vhodným nástrojem na prvotní diagnostiku. V ostatních oblastech má navrch budovaná aplikace díky inteligentnímu zpracování dat, vizualizaci a možnosti konfigurace.

3.4 RealTerm

Tento nástroj je navržený pro práci s binárními daty a dalšími typově obtížnými datovými proudy. Na rozdíl od terminálového emulátoru Putty umožňuje inspekci přijatých dat a také nabízí přímé odesílání hexadecimálních nebo binárních zpráv. Kromě schopnosti přijímat komunikaci z klasických sériových portů podporuje i zpracování TCP/IP komunikace. RealTerm obsahuje zajímavé funkce, jako je skriptování pomocí příkazové řádky nebo automatické posílání dat. [22]



Obrázek 11 – Vizuální podoba nástroje RealTerm, Zdroj: vlastní

Oproti RealTerm budovaná aplikace dělá více v oblasti vizualizace naměřených hodnot a interaktivní konfiguraci zařízení. Uživatelské rozhraní budované aplikace má lepší rozvržení a celkově je přehlednější pro uživatele. RealTerm má velice účelové a technické rozhraní, které v dnešní době neodpovídá žádným standardům, hlavně co se týče vzhledu. Rozhraní je rozděleno do několika záložek, kde na každé z nich je mnoho polí a tlačítek pro nastavení komunikace a dalších věcí. Pro uživatele může být takové rozhraní matoucí a celkově vzhled jednotlivých komponent vypadá velice zastarale. Naproti tomu budovaná aplikace je založená na moderním WPF rozhraní, které se snaží být uživatelsky přívětivé, nabízí volbu světlého nebo tmavého režimu a definuje komponenty s moderním vzhledem. Dále jsou zde rozdíly jako u nástroje Putty, kde budovaná aplikace má navrch ve funkcích jako je automatická detekce

portů, použití posledně používaného portu a předdefinované přenosové rychlosti. Z výše uvedeného porovnání lze dojít k závěru, že RealTerm se soustředí na nízko úrovňové zpracování a analýzu komunikace. Hodí se pro ladění, testování nebo periodickému monitorování komunikace, díky možnosti skriptování. Zároveň zaostává v grafické reprezentaci dat a hlavně v rámci celé aplikace. [22]

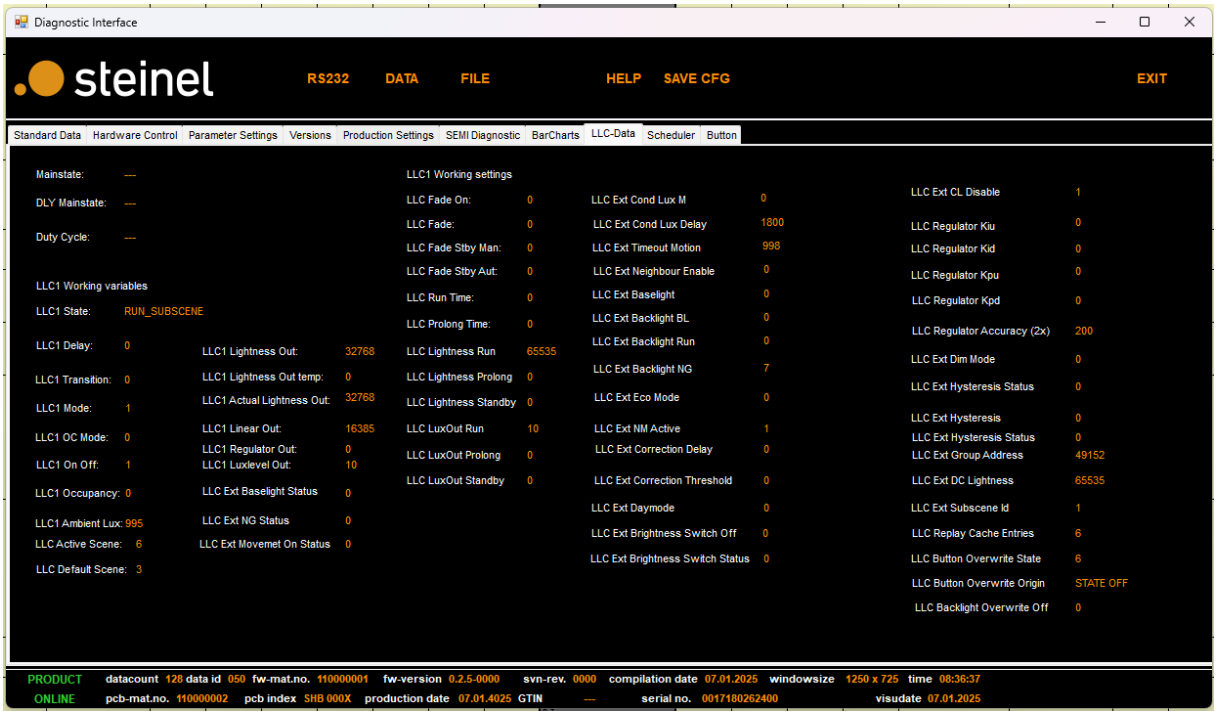
4 PŘEHLED POUŽÍVANÝCH ŘEŠENÍ

Ve firmě Steinel se používají různé vizualizace pro různá zařízení. Tato kapitola popisuje tyto nástroje pro vizualizaci a konfiguraci, které daly důvod k vzniku vyvíjené aplikaci. Cílem této části je představit existující přístupy a hlavně zhodnotit jejich silné a slabé stránky s ohledem na údržbu, přenositelnost a uživatelskou přívětivost. Analyzovaná jsou dvě základní řešení, kde z jednoho řešení vychází více modifikací pro různá zařízení a vzniká tak velká množina aplikací. Prvním analyzovaným řešením je aplikace komunikující přes protokol UART a poté následuje druhé řešení realizované pomocí protokolu RTT.

4.1 Vizualizace založená na UART

Dlouhodobě využívané řešení, které je realizováno jako šablonová aplikace, která komunikuje se zařízeními přes protokol UART. Aplikace je postavená na technologii Windows Forms s využitím .NET Frameworku verze 4.8. Komunikace s hardwarem je zajištěno pomocí knihovny NI VISA, která však v poslední době způsobuje komplikace, kde například v prostředí Visual Studio 2022 není správně rozpoznaná licence. Vývojáři tak mají problém projekt zkompilovat nebo rozvíjet stávající funkcionality.

Dalším nedostatkem tohoto řešení je způsob správy verzí a celková jednotnost vizualizační aplikace. Aplikace je navržena jako univerzální šablona, která se dále klonuje a upravuje se na míru požadavkům pro jednotlivé typy zařízení. Z toho vznikne stav, kde každé zařízení má vlastní nezávislou modifikaci aplikace, často ve více verzích, tak jak se přidávali nové funkcionality nebo opravovali chyby. Důsledkem takového přístupu je značná fragmentace kódu, kde opravné změny v šabloně nejsou ve většině případů zpětně aplikovány do již existujících modifikací aplikace. Tím pádem se rapidně zvýší nároky na údržbu, testování a zajištění konzistence mezi jednotlivými modifikacemi.

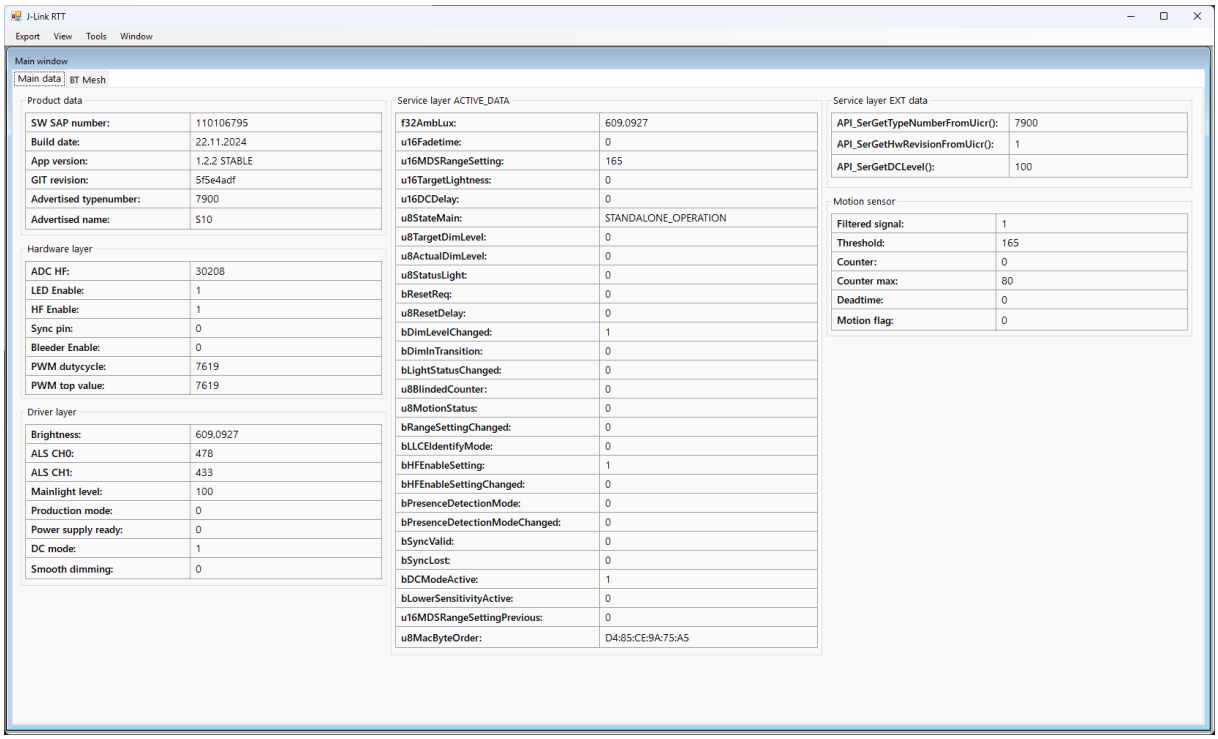


Obrázek 12 – Vizuální podoba UART vizualizace, Zdroj: vlastní

I co se týče uživatelského rozhraní aplikace zaostává za moderními zvyklostmi, jak je možné pozorovat na obrázku 12. Rozvržení stránky není ideální a data jsou neuspořádaně naházena na jedné hromadě, bez většího zarovnání. Chybí také responzivita při změně velikosti okna a možnost přepínání mezi světelnými tématy. Zároveň má aplikace pevnou definici toho, jak se jednotlivé parametry mají zobrazovat. Dále absence centralizace značně komplikuje sdílení nastavení mezi zařízeními a snižuje efektivní použití v praxi. Proto je potřeba sjednotit přístup vizualizace a konfigurace zařízení bez nutnosti vytvářet duplicity. Takové řešení je právě předmětem vyvíjené aplikace.

4.2 Vizualizace založená na RTT

Druhé používané řešení pro vizualizaci ve vývoji elektronických zařízení je aplikace používající protokol RTT pro komunikaci se zařízeními. Představuje alternativu ke standardní komunikaci přes UART a je založen na nástroji J-Link RTT Viewer. Ten umožňuje výměnu dat mezi počítačem a mikrokontrolérem prostřednictvím programovacího rozhraní. Narozdíl od prvního řešení není tato aplikace závislá na licencovaných modulech a celá je napsaná v čistém prostředí .NET s využitím Windows Forms a dalších volně dostupných open-source knihoven, například pro grafické výstupy nebo tabulkové zobrazení. Tím se vyhýbá problému, se kterým se potýká první vizualizace a vývojáři mohou bez problému upravovat aplikace bez omezení třetími stranami. Obrázek 13 zobrazuje podobu tohoto řešení.



Obrázek 13 – Vizuální podoba RTT vizualizace, Zdroj: vlastní

Uživatelské rozhraní aplikace je koncipováno jako jednoduchý přehled parametrů v tabulkách. Nachází se zde pás záložek, kde na každé se zobrazuje určitá kategorie hodnot od výrobních dat až po stavy z různých vrstev firmwaru. Jako u prvního řešení přes UART se i u této aplikace používá přístup, kde každé zařízení má svoji vlastní variantu vizualizačního nástroje, která se přizpůsobuje specifickým potřebám daného zařízení. Naopak rozdíl je v tom, že u RTT vizualizace neexistuje žádná šablona, ze které by se vycházelo. Různé verze vizualizace se vytvářejí kopírováním nebo modifikacemi předchozích verzí, které se nejvíce blíží cílovému zařízení z hlediska požadovaných funkcionalit. Dočasně tento přístup může zrychlit vývoj, ale zároveň limituje dlouhodobou udržitelnost a komplikuje správu verzí. V případě přidávání nových funkcionalit nebo oprav do starších aplikací se změny neprovádí systematicky a pouze v případě, když to okolnosti vyžadují.

Opakovaným problémem je absence jednotné správy, centrální konfigurace nebo sdílení konfigurace. Z dlouhodobého hlediska to znamená náročnější správu a riziko duplicitní práce, obdobně jako je tomu u vizualizace UART. Z porovnání obou současně používaných řešení je patrné, že vývoj nového jednotného a snadno rozšiřitelného řešení pro vizualizaci by výrazně ulehčil práci vývojářům, zjednodušil by údržbu a podpořil dlouhodobou udržitelnost aplikace.

5 ÚVOD DO PRAKTICKÉ ČÁSTI

V této části práce jsou rozebrány požadavky na práci, jakým způsobem je práce zpracovaná a také jak konkrétně vypadá komunikace mezi aplikací a fyzickým zařízením.

5.1 Funkční požadavky

FR01: Aplikace bude schopná komunikovat se zařízeními firmy Steinel pomocí protokolu UART.

FR02: Aplikace bude umět interpretovat přijatá data a vypisovat je na domovské stránce v kategorizovaných záložkách dle typu přijímaných dat.

FR03: Aplikace bude schopná ovládat připojené zařízení a nastavovat jeho příslušné parametry.

FR04: Aplikace bude umožňovat posílat pakety ve dvou módech, první pro odesílání paketů s jednotlivými parametry a druhý, kde se v paketu budou posílat všechny parametry najednou.

FR05: Zpracování dat bude probíhat v reálném čase a hodnoty vypisované na domovské stránce budou reflektovat aktuální hodnoty nastavené na připojeném zařízení.

FR06: Zobrazení dat bude probíhat za pomoci grafických komponent, které se vytvoří dynamicky na základě definic v souboru formátu JSON. Různá zařízení budou mít různě definované komponenty.

FR07: Aplikace bude schopná přijímat různé rámce ze zařízení a číst jeho konfiguraci.

FR08: Aplikace bude moct měnit konfiguraci a posílat příkazy do zařízení.

FR09: Uživatel si bude moct přizpůsobit grafické komponenty na domovské stránce v přehledném editoru JSON.

FR10: Vytvořenou konfiguraci bude moci uživatel exportovat na vybrané místo v lokálním úložišti.

FR11: Aplikace bude kontrolovat, že do vstupních polí bude moct vkládat uživatel pouze hodnoty ve validním formátu.

FR12: Soubory JSON budou mít pevně definovanou strukturu, která zajistí snadné sdílení mezi různými počítači.

FR13: Konfigurací se bude udržovat vícero a jejich načtení bude záviset na právě použitém zařízení.

FR14: Při startu aplikace budou automaticky načteny všechny dostupné konfigurace rozhraní.

FR15: Aplikace bude umožňovat přepínání mezi tmavým a světlým režimem.

FR16: Grafické komponenty budou reagovat na volbu barevného režimu přizpůsobením své barevné reprezentace.

FR17: Aplikace bude zajišťovat přehledné a moderní rozhraní, kde grafické komponenty budou responzivní a budou se přizpůsobovat velikosti okna.

FR18: Primární jazyk aplikace bude angličtina, ale bude zde možnost přepnout do dalších jazykových verzí.

FR19: Aplikace bude vypisovat svojí aktuální verzi včetně data poslední kompilace.

FR20: Aplikace bude moci nastavovat hodnoty přenosové rychlosti mezi aplikací a připojeným zařízením.

FR21: Aplikace bude schopná měnit právě aktivní zařízení ze seznamu dostupných zařízení.

FR22: Aplikace bude na všech stránkách zobrazovat název připojeného zařízení a jeho stav pomocí stavové lišty.

FR23: Aplikace bude logovat příchozí nezpracované pakety.

FR24: Seznam s nezpracovanými pakety se bude automaticky rolovat na poslední přidaný záznam.

FR25: Záznamy o nezpracovaných paketech půjde kopírovat nebo mazat.

FR26: Aplikace bude umožňovat zobrazit pro jakýkoliv parametr graf zobrazující průběh hodnot.

FR27: Aplikace bude umožňovat konfigurovat osy grafu, počet bodů nebo přiblížení.

FR28: Aplikace bude umět interpretovat číselné hodnoty na text v části uživatelském rozhraní.

5.2 Nefunkční požadavky

NR01: Aplikace bude napsaná v jazyce C#.

NR02: Rozhraní bude realizováno za pomoci frameworku WPF s doplňkovou knihovnou MahApps pro dodatečné stylování.

NR03: Aplikace bude schopná zpracovávat velké množství dat přicházející sériovou komunikací.

NR04: Data se budou aktualizovat v uživatelském rozhraním bez znatelných prodlev.

NR05: Operace ukládání a načítaní souborů typu JSON budou efektivně prováděny i při větším počtu souborů.

NR06: Aplikace bude plně kompatibilní s operačními systémy Windows 10 a Windows 11.

NR07: Konfigurační soubory budou standardizované ve formátu JSON a budou zajišťovat jednoduché sdílení mezi počítači.

NR08: Konfigurace budou uchovávány lokálně a nebudou vyžadovat připojení k internetu.

NR09: Aplikace bude pracovat pouze s autorizovanými zařízeními, které jsou připojené.

NR10: Změny v konfiguraci budou probíhat pouze na podnět uživatele, aplikace sama nebude provádět žádné změny v nastavení zařízení.

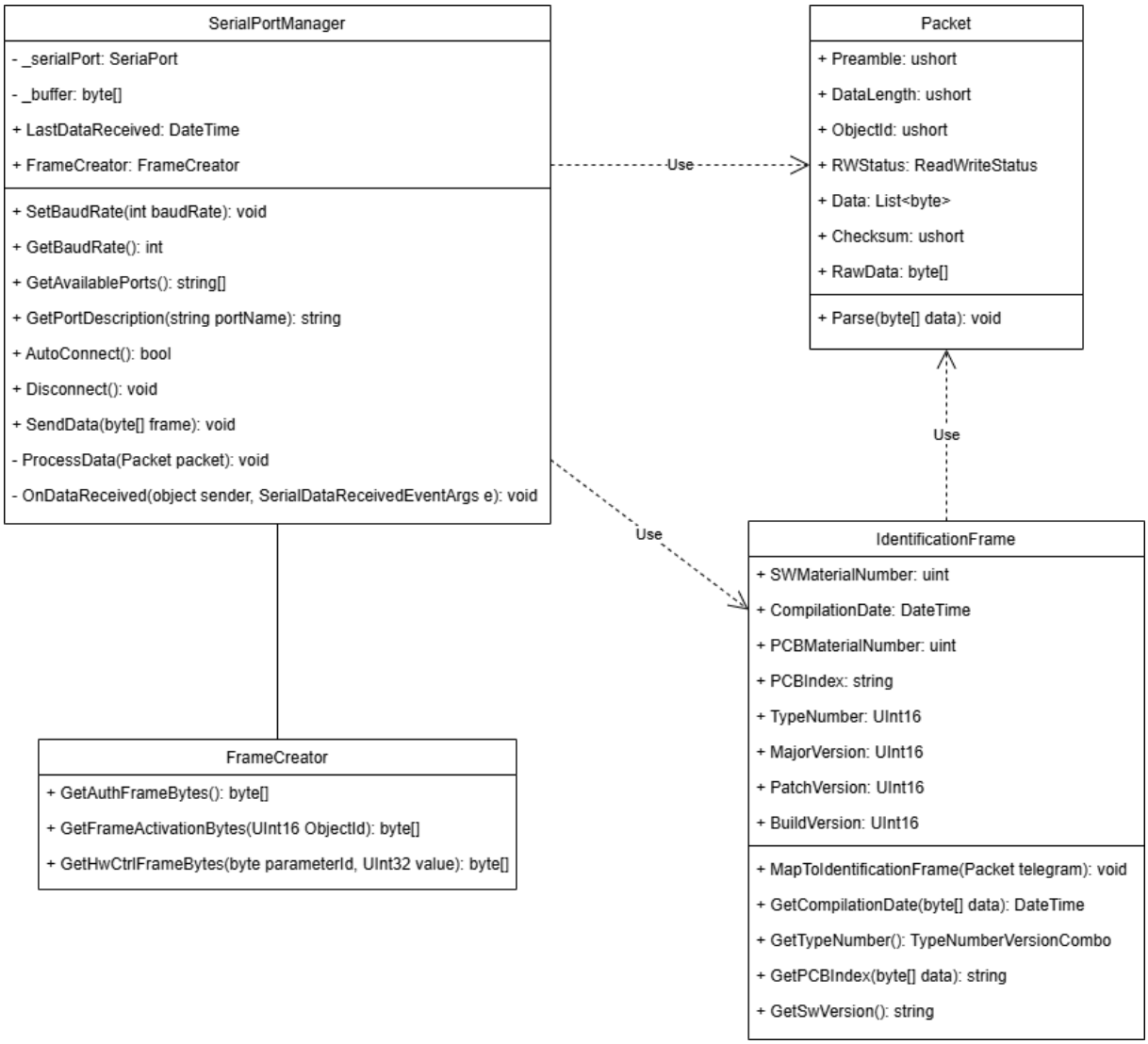
NR11: Aplikace bude navržena tak, aby do budoucna bylo možné přidávat snadno další funkce a rozšíření bez většího zásahu do konstrukce aplikace.

NR12: Restartování sériové komunikace bude možné bez nutnosti restartu celé aplikace.

5.3 Modely UML

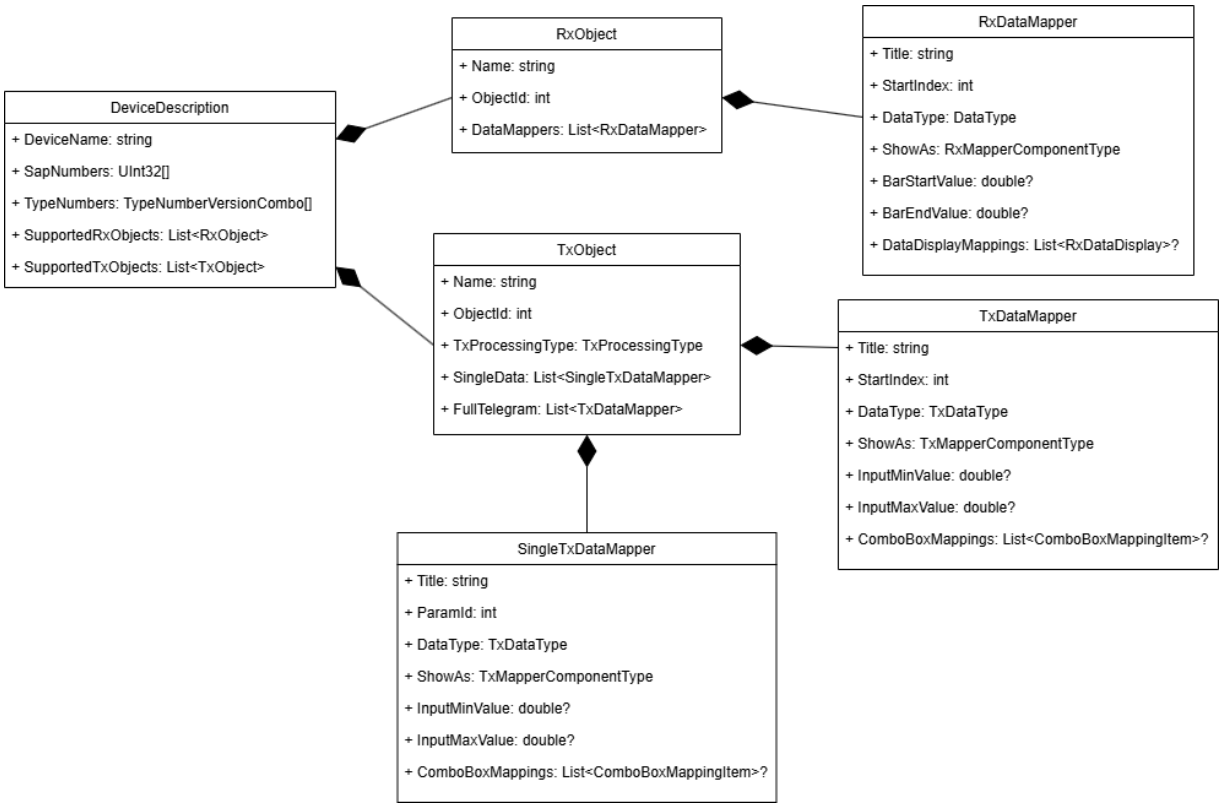
Tato kapitola popisuje návrh stěžejních částí aplikace a snaží se zvýšit srozumitelnost fungování aplikace na pozadí. Probíhá zde vizualizace klíčových tříd a jejich vztahů, což může pomoct v případě budoucího rozšiřování a údržby systému.

Na obrázku níže (Obrázek 14) lze vidět zachycení logiky komunikace s fyzickým zařízením prostřednictvím sériového portu. Tato část aplikace je odpovědná za příjem a odesílání, včetně zpracovávání přijatých paketů a jejich distribuci do vyšší vrstev aplikace. Centrálním bodem této logiky je třída „SerialPortManager“, která spravuje přístup k sériovému portu. Dále zajišťuje zpracování příchozích paketů a také odesílání paketů do zařízení za pomoci třídy „FrameCreator“, která slouží k vytváření odesílaných paketů. Zpracované rámce se vytvářejí pomocí třídy „Packet“ a „SerialPortManager“ je může dále interpretovat jako „IdentificationFrame“. „IdentificationFrame“ zajišťuje, že bude instance třídy „Paket“ správně převedena na typ „IdentificationFrame“. Vzhledem k rozsáhlosti třídy „SerialPortManager“, byly některé pomocné proměnné a metody vynechány za účelem zachování přehlednosti modelu.



Obrázek 14 – Ukázka UML – realizace komunikace, Zdroj: vlastní

V dalším diagramu tříd (Obrázek 15), je vyobrazen datový model konfigurace zařízení. Tento model popisuje strukturu konfiguračních dat, která určují, jakým způsobem má být zařízení interpretováno v rámci uživatelského rozhraní aplikace. Definuje se jaké datové objekty bude možno přijímat (RX) a jaké datové objekty bude možno konfigurovat (TX). U jednotlivých datových objektů se dále definuje, jak mají být interpretovány a jaké grafické komponenty se mají pro ně použít.



Obrázek 15 – Ukázka UML – realizace konfigurace, Zdroj: vlastní

5.4 Datový model

V této části práce jsou stručně popsány jednotlivé stěžejní třídy a jejich vztahy. Popsány jsou úlohy, které tyto třídy mají na starosti. Dále je zmíněno, jaké obsahují atributy včetně jejich datových typů. Aplikace využívá objektově orientovaný model pro organizaci dat a je založená na třídách s kolekcemi pro uchovávání dat.

5.4.1 DeviceDescription

Klíčová komponenta, která reprezentuje konkrétní zařízení a uchovává důležité informace. Tyto informace poté slouží, ke správnému vyhodnocení připojeného zařízení a funkci dynamického mapování komponent.

```
public string DeviceName { get; set; }
public UInt32[] SapNumbers { get; set; }
public TypeNumberVersionCombo[] TypeNumbers { get; set; }
public List<RxObject> SupportedRxObjects { get; set; }
public List<TxObject> SupportedTxObjects { get; set; }
```

Kód 2 – Struktura třídy „DeviceDesctiption“

Třída obsahuje 5 atributů, kde první 3 se snaží jednoznačně identifikovat připojené zařízení a poslední dva atributy definují jaké objekty lze na zařízení zpracovávat nebo konfigurovat. „DeviceName“ reprezentuje formální název zařízení. Pole „SapNumbers“ uchovává identifikátory SAP, které se vážou k zařízení. Dodatečnými identifikátory jsou „TypeNumbers“, které jsou typu „TypeNumberVersionCombo“. Tento typ obsahuje další dva identifikátory podle verze softwaru a uživatelsky zadaného identifikátoru. Předposledním atributem je seznam podporovaných typů objektů, které zařízení je schopno posílat. V praxi to znamená, jaké záložky se budou zobrazovat na domovské stránce při připojeném zařízení vysílající aktivované rámce. Při ukládání do konfiguračního souboru, je použit právě tento objekt pro serializaci. Poslední seznam podobně definuje, jaké záložky budou dostupné na stránce s ovládáním zařízení a v nich konkrétní parametry pro konfiguraci.

5.4.2 RxObject

Reprezentuje jeden typ objektu, které připojené zařízení vysílá. Jedná se o klíčový objekt, kterým uživatel dokáže definovat podobu rozhraní domovské stránky. Domovská stránka má v horní liště tolik záložek, kolik je definovaných objektů „RxObject“ v konfiguračním souboru typu JSON.

```
public string Name { get; set; }
public int ObjectId { get; set; }
public List<RxDataMapper> DataMappers { get; set; }
```

Kód 3 – Struktura třídy „RxObject“

Struktura třídy definuje název typu objektu a jeho identifikační číslo. Důležitým prvkem je list objektů „RxDataMapper“. Tyto objekty definují, jaké informace budou zobrazovány v konkrétní záložce na domovské stránce. Mimo jiné také stanovují, v jakém formátu se budou data zobrazovat. Uživatel může sám tyto objekty vytvářet na stránce s editorem JSON.

5.4.3 TxObject

Tato třída představuje prostředek pro komunikaci od aplikace k zařízení. Podobně jako „RxObject“ je „TxObject“ klíčová komponenta aplikace a díky ní je možné konfigurovat připojené zařízení. Prostřednictvím této třídy uživatel dokáže definovat co a jakým způsobem půjde na zařízení nastavit. Může také definovat konkrétní hodnoty, které půjde vybrat nebo rozmezí v jakém se budou pohybovat hodnoty odeslané do zařízení. V případě načtené

konfigurace ze souboru JSON se vytvoří tolik záložek na stránce s ovládáním zařízení, kolik je existujících objektů „TxObject“ v daném souboru JSON.

```
public string Name { get; set; }
public int ObjectId { get; set; }
public TxProcessingType TxProcessingType { get; set; }
public List<SingleTxDataMapper> SingleData { get; set; }
public List<TxDataMapper> FullTelegram {  get; set; }
```

Kód 4 – Struktura třídy „TxObject“

Atributy třídy dokážou definovat vlastnosti jakým způsobem bude realizována záložka v uživatelském rozhraní. Nejprve definuje, jak se záložka bude jmenovat. Dále specifikuje unikátní číslo objektu pro jednoznačnou identifikaci záložek. Důležitým atributem je „TxProcessingType“, ten udává, jakým způsobem bude vykonávána následná komunikace na dané záložce. Nabývá dvou hodnot „SingleData“ pro přímé odesílání jednotlivých atributů do zařízení a „FullTelegram“, který posílá kompletní paket se všemi atributy na dané záložce. Poslední dvě kolekce hodnot se váží k těmto dvou módům odesílání paketů. Seznam „SingleData“ obsahuje mapování komponent, které se zobrazí na stránce se zpracováním jednotlivých atributů a naopak pro seznam „FullTelegram“.

5.4.4 RxDataMapper

„RxDataMapper“ je třída umožňující dynamické mapování komponent, která umožňuje definovat, jak extrahovat přijaté rámce do čitelných hodnot a hlavně převést korektní hodnoty k příslušnému označení. Jinými slovy určuje, jaká data se nacházejí, na jakém místě v přijatém paketu. Podobně jako u „RxObject“ opět uživatel může definovat svoje objekty „RxDataMapper“, což je možné až po definici „RxObject“.

```
public string Title { get; set; }
public int StartIndex { get; set; }
public DataType DataType { get; set; }
public RxMapperComponentType ShowAs {  get; set; }
public double? BarStartValue {  get; set; }
public double? BarEndValue { get; set; }
public List<RxDataDisplay>? DataDisplayMappings { get; set; }
```

Kód 5 – Struktura třídy „RxDataMapper“

Třída popisuje, jak se daný údaj bude jmenovat pomocí atributu „Title“. Atribut „StartIndex“ udává, na jakém bajtu se má začít načítat hodnota. „DataType“ je výčtový typ, který říká, o jaký

typ hodnoty se jedná. Typy hodnot jsou klasické Uint16, Uint32 a další, ale jsou zde i speciální hodnoty jako například „Mac_Address“. Dále třída uchovává nepovinné atributy „BarStartValue“ a „BarEndValue“, které se používají pouze při nastavení zobrazení jako průběhový ukazatel. Tyto nepovinné atributy definující rozsahy hodnot, jež v uživatelském rozhraní budou ukazovat, jak moc je daná hodnota nahoře nebo dole. Poslední atributem je seznam pro mapování vstupních enum indexů na jejich formátované textové hodnoty, které se následně zobrazují v uživatelském rozhraní.

5.4.5 RxDataDisplay

„RxDataDisplay“ je pomocná třída, která uchovává hodnoty potřebné k mapování číselných hodnot, které slouží poté jako indexy ke konkrétním enum hodnotám. Pro lepší pochopení je zde příklad – zařízení posílá v daném parametru číslo, parametr je označený v konfiguraci jako datový typ enum, zároveň je také v konfiguraci seznam, kde jsou možné indexy a k nim textové hodnoty. Takže tam může být třeba index s hodnotou 2 a k němu textová hodnota „Red“. Tím pádem aplikace interpretuje hodnoty 2 ze zařízení u daného parametru na hodnoty „Red“ v uživatelském rozhraní. Aplikace zpracuje takový typ a podle čísla indexu dosadí textovou hodnotu do příslušného datového boxu. Tudíž zařízení posílá soustavně hodnotu 1, ale aplikace ho díky mapování interpretuje jako 50 Hz například.

```
public string DisplayValue { get; set; }
public int MappedValue { get; set; }
```

Kód 6 – Struktura třídy „RxDataDisplay“

Struktura třídy je velice jednoduchá a obsahuje pouze dva atributy. „DisplayValue“ je hodnota, která se má zobrazit v uživatelském rozhraní a „MappedValue“ je přijatá hodnota ze zařízení, která má být mapována.

5.4.6 TxDataMapper

Za pomoci této třídy se mohou vytvářet mapovací objekty pro jednotlivé položky, které jsou součástí objektu „TxObject“. Mapovací objekty této třídy definují, jak bude vypadat uživatelské rozhraní na dané záložce v módu „FullTelegram“. Každá instance třídy specifikuje jednu položku v přenášeném paketu ve směru k připojenému zařízení. Instance generuje příslušný ovládací prvek, který zajistí odeslání pouze validních hodnot.

```
public string Title { get; set; }
public int StartIndex { get; set; }
public TxDataType DataType { get; set; }
public TxMapperComponentType ShowAs { get; set; }
public double? InputMinValue { get; set; }
public double? InputMaxValue { get; set; }
public List<ComboBoxMappingItem>? ComboBoxMappings { get; set; }
```

Kód 7 – Struktura třídy „TxDataMapper“

Atributy obdobně jako u „RxDataMapper“ popisují mapovací položku v uživatelském rozhraním. V prvé řadě je název mapovací položky, počáteční index v přenášeném rámci, kde je hodnota umístěna. Následují dva výčtové typy „DataType“ a „ShowAs“. „DataType“ určuje datový typ hodnoty, jenž nabývá různých hodnot jako je Int8, Uint16, Uint32, Float a další. Druhý typ „ShowAs“ určuje, jaká komponenta bude použita jako vstup u dané mapovací položky. Dostupné komponenty jsou „Input“ realizovaný za pomoci „NumericUpDown“, „ComboBox“ nebo „CheckBox“. V poslední řadě tato třída uchovává další hodnoty pro ochranu vstupu jako je definice minimální a maximální hodnot nebo seznam předdefinovaných hodnot pro „ComboBox“.

5.4.7 SingleTxDataMapper

Varianta třídy „TxDataMapper“, která se chová úplně stejně, akorát s tím rozdílem, že reprezentuje pouze jednu konkrétní položku v přenášeném paketu. Tato položka se odesílá samostatně a jeho součástí nejsou žádné další konfigurační atributy jako je to v případě „TxDataMapper“.

```
public string Title { get; set; }
public int ParamId { get; set; }
public TxDataType DataType { get; set; }
public TxMapperComponentType ShowAs { get; set; }
public double? InputMinValue { get; set; }
public double? InputMaxValue { get; set; }
public List<ComboBoxMappingItem>? ComboBoxMappings { get; set; }
```

Kód 8 – Struktura třídy „SingleTxDataMapper“

Struktura atributů třídy je skoro stejná, akorát s rozdílem, že zde je atribut „StartIndex“ nahrazen atributem „ParamId“. Ten určuje, jaká konkrétní vlastnost zařízení se bude nastavovat. Pro potřebu nastavit pouze jeden parametr využije uživatel právě tento druh mapovače, aniž by musel odesílat všechny parametry najednou.

5.4.8 Packet

Reprezentuje rámec přijatý během sériové komunikace se zařízením. Třída umožňuje parsování surových dat na jednotlivé atributy. Slouží k analýze a následné vizualizaci přijatých hodnot v rozhraní.

```
public ushort Preamble { get; set; }
public ushort DataLength { get; set; }
public ushort ObjectID { get; set; }
public ReadWriteStatus RWStatus { get; set; }
public List<byte> Data { get; set; }
public ushort Checksum { get; set; }
public byte[] RawData { get; set; }
```

Kód 9 – Struktura třídy „Packet“

Po rozdělení všech dat do příslušných atributů vzniká objekt „Packet“, který umožňuje lepší manipulaci s přijatými daty, díky svojí struktuře. Například lze okamžitě zjistit, o jaký typ rámce se jedná přečtením atributu „ObjectID“, bez toho, aniž by se pokaždé musely odpočítávat bajty z nezpracovaných dat.

Na obrázku níže (Obrázek 16) je vyobrazen formát paketu, tak jak přichází od zařízení.

Obrázek 16 – Nákres struktury paketu, Zdroj: interní dokumentace

Preambule a hodnota kontrolního součtu jsou pevné části paketu, které obklopují flexibilní data. Teoretická délka takového paketu je interně omezena na 1000 bajtů. Přenosové možnosti závisí na každém jednotlivém produktu.[1]

5.4.9 IdentificationFrame

Jedná se o speciální typ rámce, který vychází z objektu „Packet“. V případě, že zařízení je nově připojené, odesílá identifikační rámec. Třída „IdentificationFrame“ mapuje data z objektu „Packet“ pomocí výčtového typu „IdentificationFramePositionBase“ na konkrétní hodnoty

[1] Zdrojem tohoto odstavce je interní dokumentace společnosti Steinel

typické pro identifikační rámec. Všechny atributy této třídy tak reprezentují datový obsah identifikačního rámce.

```
public uint SWMaterialNumber { get; set; }
public DateTime CompilationDate { get; set; }
public uint PCBMaterialNumber { get; set; }
public string PCBIndex { get; set; }
public UInt16 TypeNumber { get; set; }
public UInt16 MajorVersion { get; set; }
public UInt16 MinorVersion { get; set; }
public UInt16 PatchVersion { get; set; }
public UInt16 BuildVersion { get; set; }
```

Kód 10 – Struktura třídy „IdentificationFrame“

Pro identifikační rámec jsou charakteristické hodnoty určující přesně jaké verze softwaru se nacházejí na zařízení. Dále zde lze najít typová čísla pro rozlišení modelu zařízení nebo datum a čas kompilace softwaru.

5.4.10 TypeNumberVersionCombo

Jedná se o podpůrnou třídu, která reprezentuje unikátní kombinaci identifikátoru a jednotlivých verzí firmwaru. Pomocí této struktury lze jednoznačně určit, o jaké zařízení se jedná.

```
public int TypeNumber { get; set; }
public int MajorVersion { get; set; }
public int MinorVersion { get; set; }
public int PatchVersion { get; set; }
public int BuildVersion { get; set; }
```

Kód 11 – Struktura třídy „TypeNumberVersionCombo“

Unikátní identifikátor se skládá z typového čísla, hlavní verze firmwaru, vedlejší verze, opravné verze a verze konkrétního sestavení. Účel této třídy je tedy ukládat konfiguraci mapování komponent pro různé verze firmwaru, porovnání jednotlivých identifikátorů a zajistit načtení správného konfiguračního souboru pro mapování komponent ze souboru JSON.

5.4.11 SerialPortManager

Jak z názvu napovídá, třída „SerialPortManager“ má na starosti správu nezpracované komunikace mezi aplikací a zařízením. V této třídě se inicializuje sériový port, který může být po úspěšné inicializaci otevřen. Zodpovídá také za příjem bajtů ze sériového přenosu, která jsou následně ihned zpracována díky událostem. Při příjmu těchto bajtů se v této třídě rozhoduje,

jak budou surová data zpracována na základě získané hodnoty „ObjectID“. Třída „SerialPortManager“ volá pomocnou třídu „ChecksumCalculator“, která poskytuje služby pro výpočet kontrolních součtů (CRC), což umožňuje ověření správnosti přenášených dat mezi aplikací a zařízením. Pro uživatelský komfort dále třída implementuje logiku pro automatické připojení k poslednímu používanému zařízení, pokud je v ten moment dostupné. V praxi to znamená, že když je zařízení po sobě odpojeno a znovu připojeno, komunikace se automaticky obnoví bez potřeby restartu aplikace.

Celkově lze říct, že se jedná o základní komponentu pro fungování celé aplikace. Zajišťuje, že přenos bude spolehlivý a data budou správně interpretována. Díky svojí znovu použitelnosti a modularitě je připravená na zpracování různých komunikačních protokolů, což zajišťuje robustnost a připravenost na změny v hardware.

5.4.12 LanguageManager

Třída realizuje manažera pro správu lokalizace v aplikace. To zahrnuje načítaní textů z lokalizačních souborů a dynamickou změnu jazyka v uživatelském rozhraní. Tudíž aplikace je díky tomu schopná přepínat mezi jazykovými verzemi bez nutnosti restartování. Překlady do různých jazyků jsou získávány z lokalizačních souborů typu „resx“ uložené ve zdrojích aplikace. Třída za pomoci „ResourceManager“ dokáže tyto překlady načíst. Hlavní metodou třídy je metoda pro změnu aktuálního jazyka a nastavení nově zvoleného jazyka jako výchozího při příštím spuštění. Při aktualizaci jazyka aplikace projde přes všechna otevřená okna a jejich kontext nastaví na novou instanci „LanguageManager“, což způsobí okamžitou změnu jazyka.

5.4.13 FrameCreator

Třída „FrameCreator“ je důležitým prvkem pro stavbu datových rámců (paketů), které se následně posílají do zařízení. Hlavní úlohou je vytvářet binární reprezentace zpráv, počítat kontrolní součty a vracet pole obsahující hotový rámec. Dále také tato třída slouží jako prostředník mezi vyšší logikou aplikace a nízkoúrovňovou komunikací. Oddělení tvorby komunikačních rámců od ostatní logiky aplikace přináší svoje výhody ve formě větší modularity a snazší údržbě kódu. Tudíž pokud dojde k změnám v komunikačním protokolu nebo ve formátu zpráv, stačí provést požadované změny pouze v této třídě a není nutné provádět žádné změny ve vyšších vrstvách aplikace.

Každá metoda třídy poskytuje vytvoření nějakého typu rámce. Jedná se o důležité rámce, bez kterých nelze zařízení ovládat, jako je například autorizační rámec, rámec pro aktivaci hardwarové kontroly, aktivaci konkrétního objektu nebo rámec pro odeslání specifických dat. Všechny metody aplikují stejný formát zpracování. Ten spočívá nejprve v sestavení posloupnosti bajtů, která zahrnuje synchronizační bajty, data, případně i příznaky pro zpracování a úplně na konci je připojen kontrolní součet.

5.4.14 RelayCommand

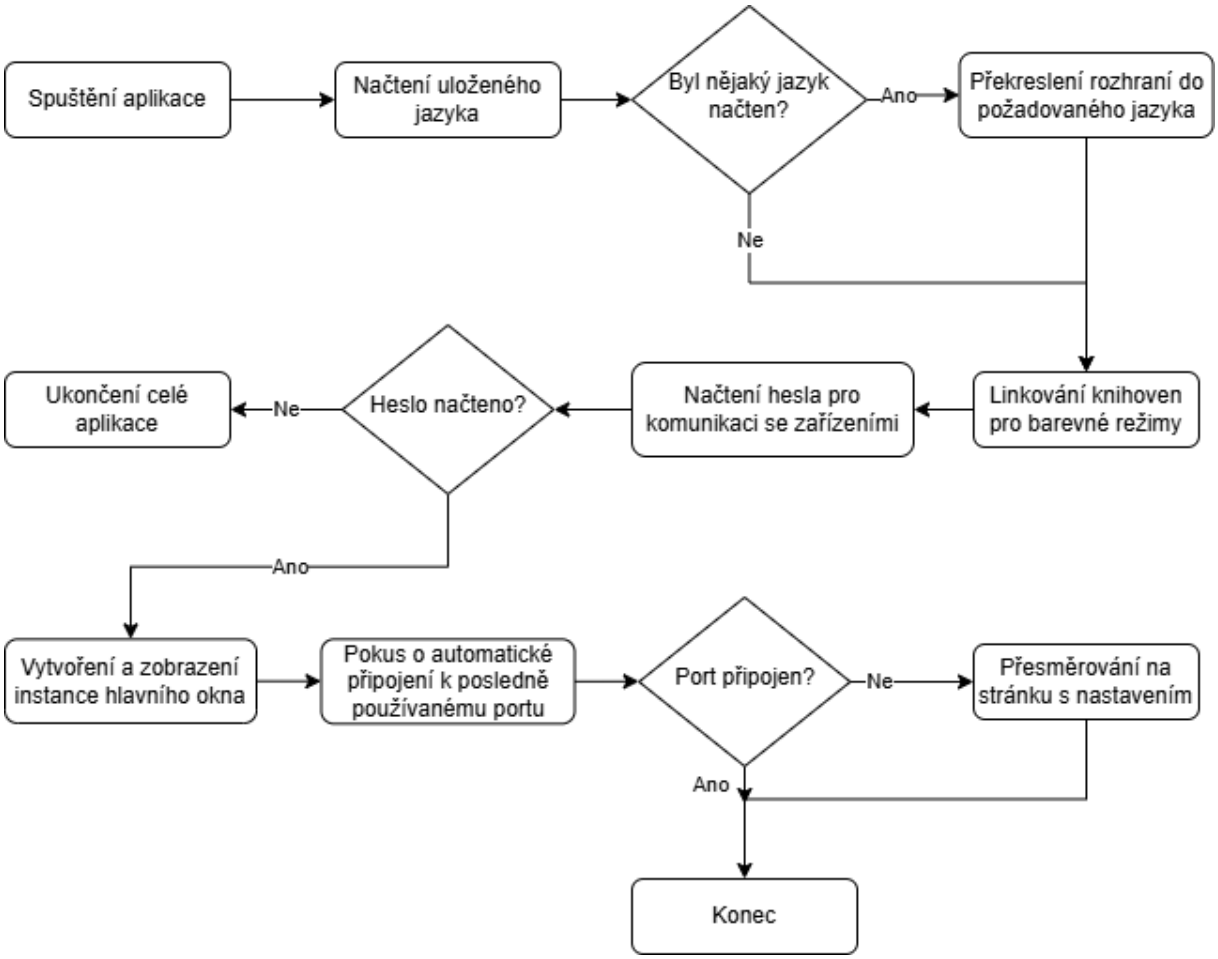
Jedná se implementaci rozhraní „ICommand“ a slouží jako univerzální prostředek pro vytváření příkazů v rámce architektury Model-View-ViewModel ve WPF aplikacích. Hlavní účel třídy je zprostředkovat předání logiky prováděného příkazu a logiky určující, zda lze daný příkaz vykonat, bez potřeby psát samostatnou třídu pro každý konkrétní příkaz. Na vstupu „RelayCommand“ přijímá metodu typu „Action“, která definuje logiku pro vykonání příkazu. Současně je zde volitelný parametr pro přijetí funkce, jenž rozhoduje, zda je příkaz momentálně vykonatelný. Díky této implementaci se výrazně usnadňuje správa uživatelských interakcí v aplikaci.

6 POPIS FUNGOVÁNÍ APLIKACE

Tato kapitola má za cíl detailní popis funkcionality aplikace a jakými způsoby vykonává jednotlivé operace. Snaží se čtenáři poskytnout ucelený pohled o tom, jak aplikace funguje a jakým způsobem mezi sebou spolupracují jednotlivé třídy nebo metody. Posloupnosti metod jsou v některých případech probrány do samotného detailu, aby bylo zajištěno úplné porozumění posloupnosti událostí pro realizaci dané operace. Probrány budou mechanismy od samotné inicializace aplikace až po správu specifických preferencí pro přizpůsobení uživatelského rozhraní.

6.1 Inicializace aplikace

Aplikace při spuštění začíná ve třídě „App“, která je odvozená od základní třídy „Application“. V této třídě se nachází vstupní metoda „OnStartup“, jenž je volána automaticky a zajišťuje počáteční inicializaci při spuštění aplikace před samotným zobrazením uživatelského rozhraní. Při této inicializaci se nejprve načte výchozí jazyk aplikace, který je uložený v konfiguračním souboru aplikace. Po načtení je jazyk nastaven pomocí třídy „LanguageManager“ a aplikován pro všechny textové komponenty uživatelského rozhraní. Následně se provede linkování souborů obsahujících definice, jak se mají chovat komponenty aplikace při nastavení světlého nebo tmavého režimu. Zároveň se podle konfiguračního souboru nastaví režim, který byl zvolen jako výchozí. Důležitou akcí, která se musí provést při startu je načtení autorizačního hesla ze souboru. Toto heslo slouží k přístupu do vývojářského režimu zařízení, tudíž zařízení pak posílá data a je možné ho konfigurovat. Bez autorizačního hesla nelze aplikaci ani spustit. Dalším krokem metody „OnStartup“ je vytvoření samotné instance hlavního okna aplikace se jménem „MainWindow“ a následné zobrazení. Posledním krokem je automatické připojení k posledně používanému sériovému portu, číslo portu je opět získáno za pomoci konfiguračního souboru. Tento postup zajistí, že po zapnutí aplikace je vše přizpůsobeno uživateli a ten nemusí po každém spuštění všechny věci nastavovat znovu. Celý tento proces je ilustrován na obrázku níže (Obrázek 17).



Obrázek 17 – Vývojový diagram procesu startu aplikace, Zdroj: vlastní

6.2 Zpracování příchozích rámců od zařízení

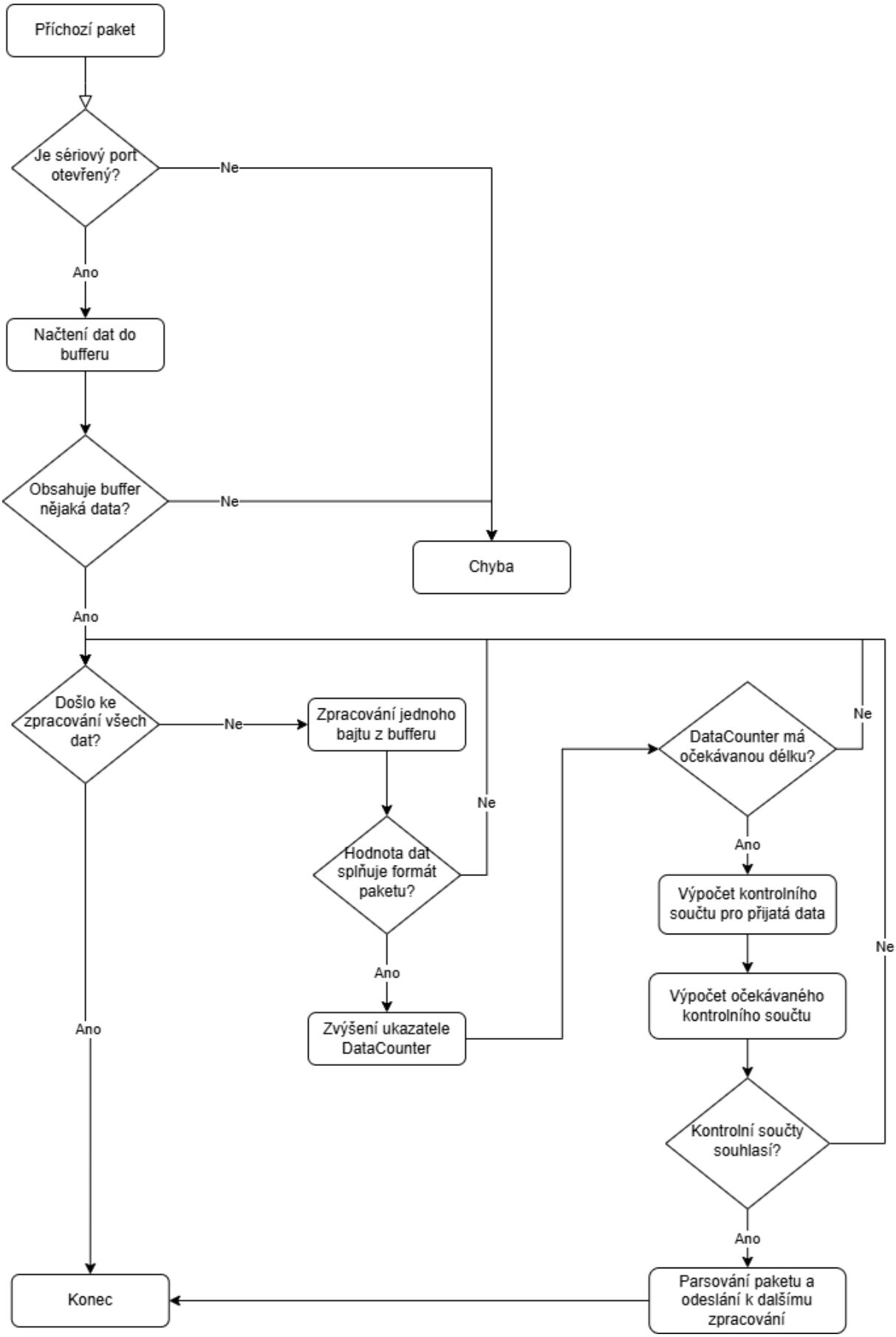
Komunikace mezi aplikací a hardwarovým zařízením je realizovaná prostřednictvím sériového portu, kde příchozí data jsou ve formátů rámců (paketů). Data přicházejí zpravidla každých 200 milisekund s velikostí okolo 40 bajtů. Proces prvotního zpracování je rozdělen do tří hlavních částí: přijetí a sestavení rámce, ověření integrity dat a další rozdělení pro následující zpracování dle typu přijatého paketu. Základní logika pro zpracování rámců se nachází ve třídě „SerialPortManager“, která deklaruje obslužnou metodu „OnDataReceived“. Tato obslužná metoda se zavolá pokaždé, když přijde na vstupu nový blok dat od zařízení. V prvním kroku se nejprve zkontroluje, zda je vůbec sériový port otevřen a aplikace z něj tedy může přijmout data. Přijatá data jsou načtena do dočasného bufferu, který slouží pro procházení aktuálně přijaté sekvence dat a přidávání do seznamu bajtů, který sestavuje paket.

Data jsou zpracovávána bajt po bajtu, přičemž se sleduje řídící proměnná „DataCounter“, která určuje, na jakém místě v rámci by měla být jaká hodnota. Při zpracování každého bajtu

se hodnota „DataCounter“ zvětší o jedničku. Proměnná se nastavuje zpět na nulu, pokud selže prvotní synchronizace nebo se překročí maximální rozmezí datového typu byte v absolutní hodnotě. Když je dosažena předpokládaná délka paketu, tak dojde k výpočtu kontrolního součtu. Tudíž pokud je detekován první bajt, „DataCounter“ má hodnotu 0 a přijatá hodnota z bufferu je „0xAA“ inicializuje se nový rámec. Další hodnotou přijatou z bufferu by měla být hodnota „0x55“ s hodnotou „DataCounter“ rovno 1 a tím je dokončena prvotní synchronizace. Poté co jsou přijaté synchronizační bajty je načase přečíst další dva bajty s hodnotou délky rámce. Podle toho je potřeba nastavit kolik ještě přečíst bajtů, aby paket byl kompletní. S nastavenou očekávanou délkou rámce se přijímají další bajty, dokud není dosaženo této délky.

Celý přijatý paket se musí ověřit pomocí kontrolního součtu (CRC) a potvrdit tak jeho integritu, že při přenosu nenastaly žádné problémy. Z přijatého paketu je extrahována hodnota kontrolního součtu, která byla zaslána zařízením. Poté je vypočítán nový kontrolní součet pomocí přijatých dat. V případě, že se obě získané hodnoty shodují znamená to, že data zaslaná zařízením jsou platná a může se pokračovat v dalším zpracování. Dalším zpracováním je nově vytvořená instance objektu „Packet“, která reprezentuje přijatý validní paket. Ten je následně poslán k dalšímu zpracování pomocí delegátu, který je volán v hlavním aplikačním vlákně. Tím se zajistí, že přijatá data jsou bezpečně zpracována a zobrazena v uživatelském rozhraní bez přímého narušení jinými vlákny.

Paket analyzuje dále ve třídě „SerialPacketManager“ pomocí metody „ProcessData“. Metoda rozhoduje, jak dále naložit s přijatým paketem. Výběr dalšího zpracování se provádí pomocí hodnoty atributu „Object ID“ příslušného paketu. Například v případě, že je hodnota „Object ID“ rovna 1 aplikace ví, že se jedná o paket s informacemi popisující identifikační data zařízení. Aplikace takový paket zpracuje a namapuje ho na novou instanci třídy „IdentificationFrame“, která obsahuje všechny potřebné atributy pro identifikaci zařízení. Po mapování je vyvolána další událost „IdentificationFrameProcessed“, která informuje další část aplikace, že byla přijata identifikační data zařízení. Všechny pakety bez ohledu na jejich hodnotu „Object ID“ jsou zpracovány událostí, která má na starosti logování přijatých bajtů. Finální zpracování paketů probíhá posléze na jednotlivých stránkách na frontendu. Toto zpracování bude popsáno v další kapitole níže. Přehledové zobrazení procesu popisovaného v této podkapitole je vidět na dalším obrázku. (Obrázek 18).



Obrázek 18 – Vývojový diagram prvotního zpracování příchozího paketu, Zdroj: vlastní

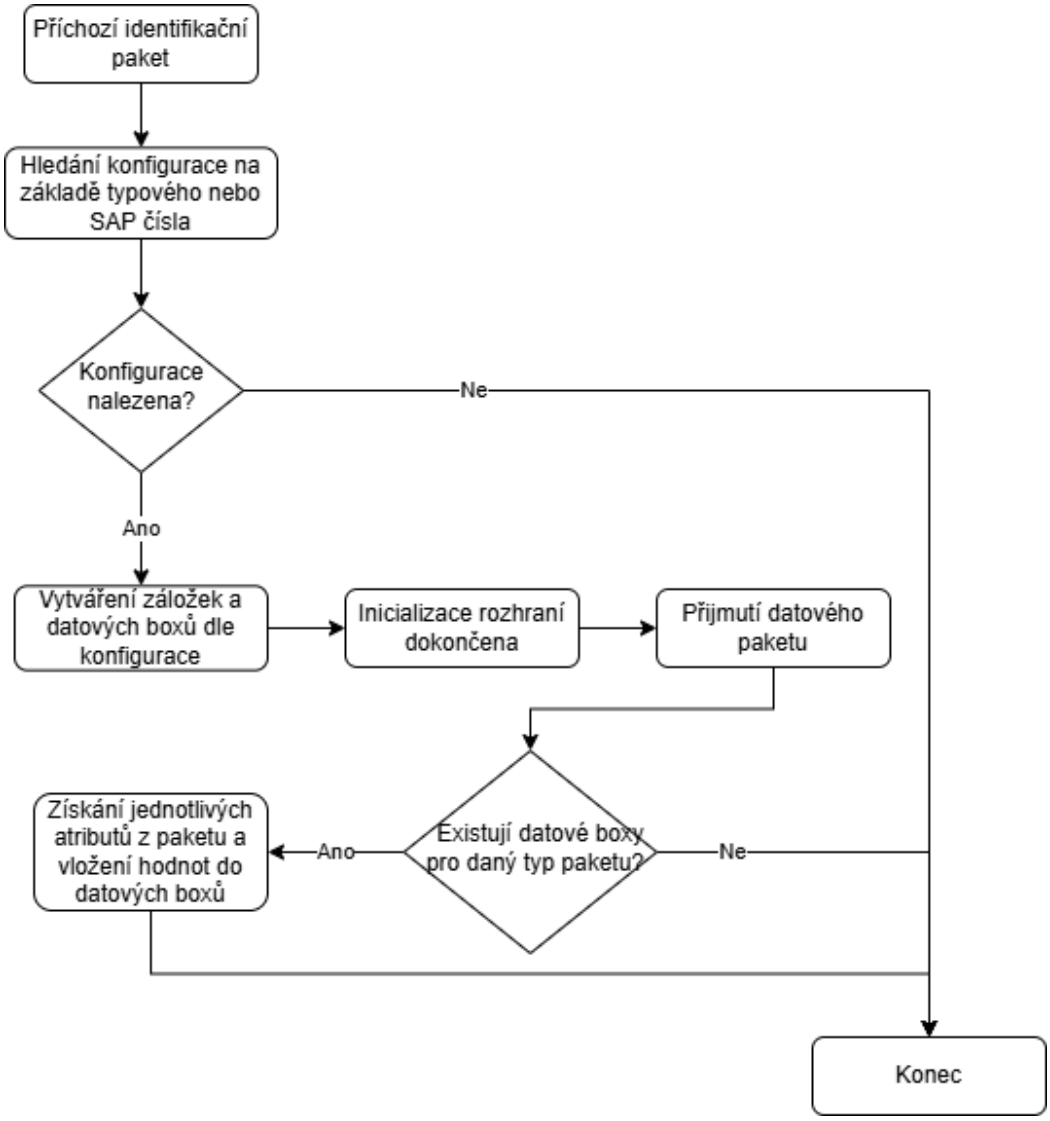
6.3 Zpracování paketů na frontendu

Po úspěšném zpracování paketu na pozadí aplikace přichází jeho další zpracování na konkrétní stránce WPF, která zajistí kompletní vizualizaci příchozích informací z původně surových nezpracovaných dat. V tomto případě se o vizualizaci stará stránka „HomePage“, která je při spuštění aktivní jako první. Vizualizace paketů probíhá ve dvou krocích, nejprve jsou vytvořeny datové boxy na základě popisu zařízení načteného z konfiguračního souboru JSON a poté probíhá dynamická aktualizace hodnot boxů v reálném čase.

V prvním kroku musí přijít nejprve identifikační paket zařízení, aby bylo možné cokoliv zobrazovat. Identifikační paket obsahuje důležité informace, jako je číslo SAP a typové číslo, které je kombinací zvoleného identifikačního čísla a čísla aktuální verze vyvíjeného firmwaru. Obě čísla jsou aplikací kontrolována a v případě, že zařízení odpovídá alespoň jedné z hodnot je načten odpovídající objekt „DeviceDescription“ obsahující informace, jak se budou data mapovat na jednotlivé datové boxy. Po zpracování identifikačního rámce je vytvořena záložka „Product Info“ na domovské stránce a v ní příslušný výpis atributů reprezentovaný jednotlivými datovými boxy. Každý datový box obsahuje název atributu a jeho hodnotu, přičemž hodnota může být reprezentovaná buď prostým textem, indikátorem měnícím barvu podle stavu nebo ukazatel průběhu měnící se podle hodnoty v určitém rozsahu.

Pro úspěšně identifikované zařízení aplikace načte všechny podporované objekty a pro každou z nich vytvoří další záložku na domovské stránce, včetně datových boxů. Po načtení má aplikace vše potřebné pro mapování komponent a jejich hodnot. Nyní se přistoupí ke zpracovávání datových paketů, které obsahují měřené hodnoty a stavová data zařízení. Při přijetí datového paketu se nejprve kontroluje, zda jsou vytvořené příslušné datové boxy a je tak možné někam data vložit. V kladném případě dojde k aktualizací polí v datových boxech na základě informacích poskytnutých z načtených mapovacích objektů. Jednotlivé atributy mohou být zakódovány jiným způsobem, proto se používají specifické metody pro získávání dat z paketu. Pokaždé co aplikace obdrží nový datový paket jsou přepisovány datové boxy, tím je zajištěna dynamická aktualizace informací v reálném čase. Datové boxy implementují rozhraní „INotifyPropertyChanged“, které pomáhá s automatickou aktualizací datových polí při změně jejich obsahu. Jakmile je zaznamenána změna hodnoty vyvolá se událost „PropertyChanged“ a dojde k překreslení prvku vázaného na danou hodnotu. Tím se zajistí, že pouhá změna hodnoty v datovém boxu způsobí přepsání informace na domovské stránce bez potřeby rozhraní aktualizovat ručně. Mechanismus tak zajišťuje efektivní a automatizovanou

vizualizaci dat v reálném čase bez cyklického přepisování statických hodnot. Celkový přehled popisovaného procesu inicializace rozhrání na frontendu a následné zpracování paketů je ilustrováno na další obrázku (Obrázek 19).



Obrázek 19 – Vývojový diagram zpracování paketů na frontendu, Zdroj: vlastní

6.4 Práce s grafy

Aplikace používá dialogová okna s grafy pro detailnější vizualizaci průběhu různých hodnot. Vyvolání okna s grafem proběhne kliknutím do libovolného datového boxu na domovské stránce. Aplikace nejdříve kontroluje, zda již nějaké okno s grafem sledující danou hodnotu neexistuje, v případě, že ano, vybere ho jako aktivní a další okno nevytváří. Všechna vytvořená aktivní okna se přidávají do kolekce pro snadnou správu aktivních oken. Ještě před vytvořením samotného okna se zaregistruje nová událost „ValueUpdated“, díky které se bude graf

aktualizovat v reálném čase tak jak přicházejí hodnoty. Následně se vytvoří dialogové okno a graf se inicializuje za pomoci knihovny LiveCharts2. V grafu se používají dvě datové kolekce, jedna pro hodnoty a druhá pro časová razítka hodnot. Datové kolekce jsou nastaveny jednotlivým osám grafu. Jedná se kartézskou soustavu souřadnic.

Po vytvoření grafu se automaticky přes dříve definovanou událost volá metoda „UpdateGraph“, která má za úkol periodicky přidávat nové záznamy do grafu. Metoda přijímá v parametru hodnoty pro osu X a Y, které pak přidává do příslušných kolekcí. Dále také udržuje definovaný maximální počet bodů, tím že vypočítá, kolik datových bodů překračuje povolený limit a tolik nejstarších datových bodů z grafu odstraní. Jedním z dalších úkolů metody je výpočet maximální hodnoty z příchozích dat a také průměru, který se počítá z kumulativního součtu hodnot a celkového počtu vzorků. Tím pádem hodnota maxima a průměru je vždy počítána na základě historických dat a nezávisí pouze na aktuálních hodnotách v grafu. V případě, že je průběh grafu pozastaven uživatelem jsou všechna příchozí data do metody „UpdateGraph“ zahazována. Po zavření dialogového okna se odpojí událost „ValueUpdated“ od příslušného datového boxu a dojde ke smazání daného záznamu z kolekce aktivních oken s grafy.

6.5 Přepínání barevných režimů

Uživatel má v aplikaci možnost si vybrat ze světlého a tmavého režimu barev, což poskytuje osobní preferenci, kterou si aplikace bude pamatovat v následujících spuštěních. Možnost změny režimu je dostupná v záložce „Nastavení“ v hlavním menu. Na výběr má uživatel ze tří možností: světlý režim, tmavý režim nebo automatický režim, který se řídí výběrem režimu podle operačního systému. Jednotlivé barevné motivy jsou definovány v souborech XAML. Aplikace používá dva soubory: „CustomAccent.xaml“ pro definici vzhledu komponent světlého režimu a „CustomAccentDark.xaml“ pro komponenty v tmavém režimu.

Hlavní logika pro správu motivů je umístěna ve třídě „ThemeService“, tato třída zajišťuje nastavení motivů a získání právě používaného motivu. Nastavení nového motivu probíhá pomocí metody „SetTheme“. V této metodě je na vstupu parametr typu řetězec, který reprezentuje právě vybranou možnost pro nastavení režimu, tato možnost je okamžitě uložena do konfiguračního souboru. Posléze se zajistí přepnutí režimu tím, že je právě používaný motiv odstraněn z „ResourceDictionary“, která se stará o jednotlivé grafické zdroje aplikace. Když je starý motiv odstraněn přidá se nový do „ResourceDictionary“, který odpovídá právě zvolenému stylu v parametru metody. V případě, že je zvolena volba „Automatický“ je nutné v metodě „SetTheme“ zavolat pomocnou metodu „GetSystemTheme“. Metoda přistupuje k hodnotě aktuálně používaného motivu v registrech Windows a je schopna tuto hodnotu vyčíst a dle toho nastavit daný motiv i v aplikaci. Ukázka tohoto přístupu:

```
using (RegistryKey? key = Registry.CurrentUser.OpenSubKey(@"Software\
          Microsoft\Windows\CurrentVersion\Themes\Personalize"))
{
     if (key != null)
     {
          object? lightThemeValue = key.GetValue("AppsUseLightTheme");
          if (lightThemeValue != null && int.TryParse
               (lightThemeValue.ToString(), out int isLightTheme))
          {
               return isLightTheme == 1 ? "Light.Custom" : "Dark.Custom";
          }
     }
}
```

Kód 12 – Získání aktuálně používaného barevného režimu z registru Windows

V ukázce na prvních třech řádcích aplikace otevírá konkrétní registr, ve kterém je uložena hodnota o barevném režimu. Použití „CurrentUser“ specifikuje, že se čte pouze nastavení aktuálně přihlášeného uživatele. Pokud je úspěšně získán přístup k registru, tak v tom případě registr „Personalize“, přistoupí ke čtení hodnoty „AppsUseLightTheme“ z registru. Tato hodnota říká, zda systém používá světlý režim. Pokud je získaná hodnota 1, bude pro aplikaci nastaven světlý režim, v opačném případě tmavý režim. Tímto způsobem aplikace zajišťuje, že bude mít vždy takový barevný režim, jaký uživatel preferuje v rámci operačního systému.

6.6 Přepínání mezi jazyky aplikace

Jazyková lokalizace aplikace je založená na systému „resx“ souborů, ve kterých se nachází přeložené řetězce pro různé jazyky. Třída, která má na starosti správu jazyků se nazývá „LanguageManager“, tato třída má mimo jiné na starosti také aktualizaci uživatelského rozhraní

při změně jazyka. Jako je to u jiných preferencí i jazyková verze je po spuštění načtena z konfiguračního souboru aplikace. Načtená hodnota je předána metodě „ChangeLanguage“ ve třídě „LanguageManager“, která provede nastavení příslušné jazykové mutace. Poté metoda prochází všechna otevřená okna aplikace a každému oknu nastaví nový kontext, který zajistí přepsání všech lokalizovaných řetězců. Ke konci se aktualizují překlady pro položky komponent typu „ComboBox“, které nelze přepsat při nastavování nového kontextu v předchozím kroku.

Třída „LanguageManager“ využívá instanci třídy „ResourceManager“, jenž umožňuje načítání překladů ze souborů typu „resx“. Instance „ResourceManager“ má nastavený jmenný prostor, ve kterém má překladové soubory hledat. V daném jmenné prostoru jsou již jednoduše definované jednotlivé soubory lokalizace ve formátu například „Strings.cs.resx“, kde „cs“ znamená, že soubor definuje české překlady. Na stejném principu jsou definované ostatní jazyky s výjimkou souboru „String.resx“, který obsahuje výchozí jazyk aplikace, což je v případě budované aplikace angličtina. V situaci, kdy uživatel změní jazyk aplikace je vyvolána událost, která cílovým prvkům vázaných na lokalizované texty řekne, že došlo ke změně. Kontexty se nastavují přímo na stránkách WPF, kde v tom případě je kontext nastaven na „LanguageManager.Instance“. Po nastavení kontextu se již nastavují konkrétní místa, kde bude text lokalizovaný. Stačí místo konkrétního řetězce nastavit binding, jako je tomu v tomto příkladu:

```
<TextBlock Text="{Binding [DeviceName_Label]}"/>
```

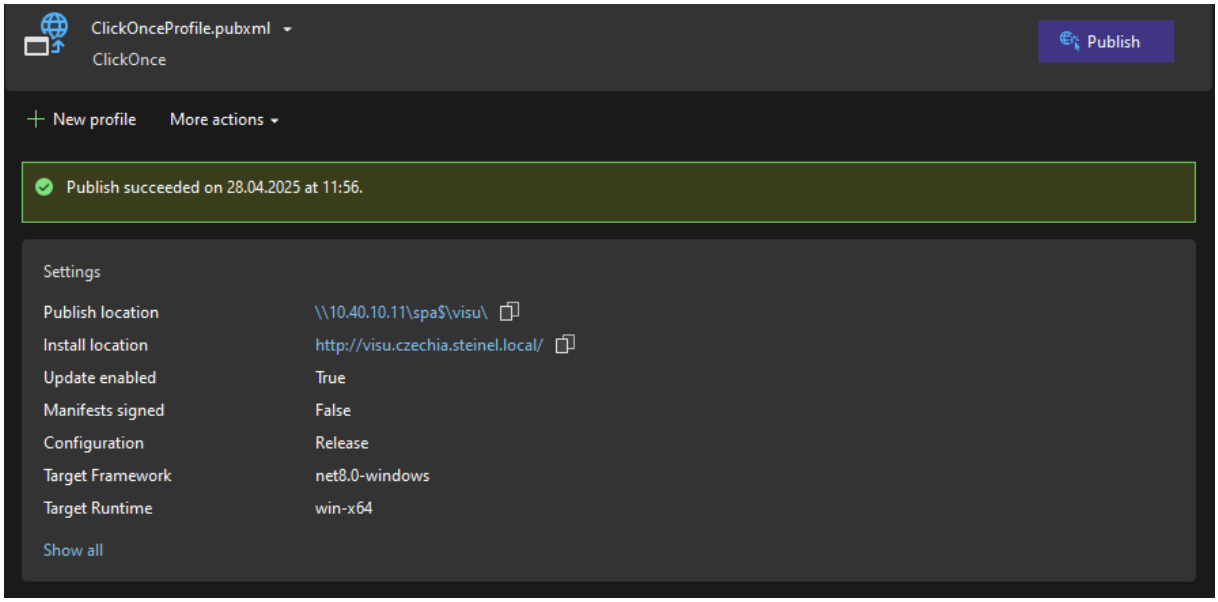
Kód 13 – Vázání textového bloku s překladem

Při zpracování se systém podívá do konkrétního „resx“ souboru pro daný jazyk a najde identifikátor, který koresponduje s překladem. Tudíž v každém „resx“ souboru se nachází sloupec se jmény identifikující dané komponenty, v nichž je potřeba dosadit hodnotu a druhý sloupec obsahuje dosazovanou hodnotu. Pomocí tohoto mechanismu není nutné aplikaci restartovat při změně jazyka a všechny změny proběhnou dynamicky v reálném čase.

6.7 Distribuce aplikace

Budovaná aplikace byla distribuována na interní server společnosti Steinel prostřednictvím technologie ClickOnce, která umožňuje snadnou distribuci a aktualizaci desktopových aplikací v operačních systémech Windows. Velkou výhodou bylo, že nástroj byl již součástí instalace

vývojového prostředí Visual Studio, tudíž nebylo nutné instalovat dodatečné nástroje. Při publikaci byly nastaveny parametry pro lokaci, kam bude instalátor aplikace nasazen a z jaké adresy ho budou moci uživatelé stáhnout. Parametry nastavení publikace v prostředí Visual Studia je možné vidět na obrázku 20.



Obrázek 20 – Nastavení publikace aplikace ve Visual Studiu, Zdroj: vlastní

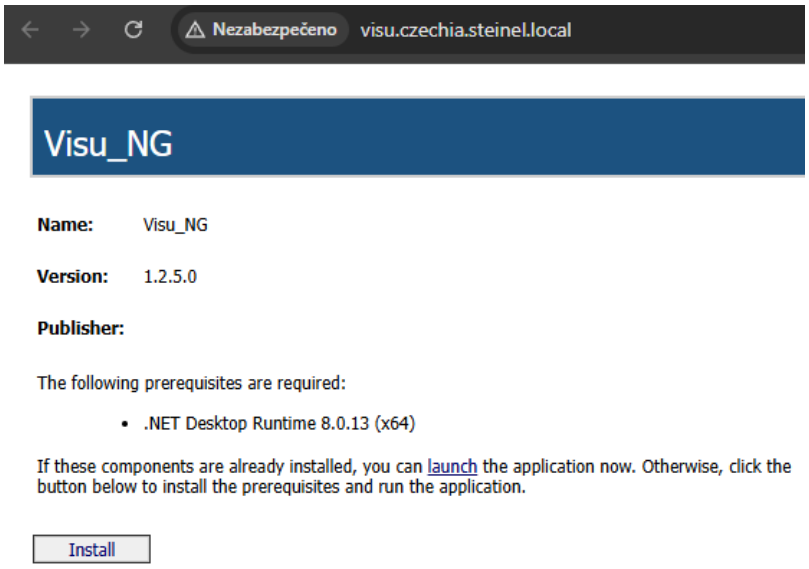
Výhodou tohoto přístupu jsou automatické aktualizace, když je na server nasazena nová verze aplikace, ClickOnce zajistí, že uživatelé budou vyzváni k aktualizaci aplikace při jejím spuštění. Tím je zajištěna efektivní správa všech instalací aplikace a zajištění, že uživatelé vždy pracují s nejnovější verzí aplikace.

7 POUŽITÍ APLIKACE

V této kapitole se práce snaží poskytnout čtenáři komplexního a detailního průvodce uživatelským rozhraním aplikace a způsobem jakým se aplikace používá. Cílem je vysvětlit rozložení aplikace a co lze na jednotlivých místech najít, aby uživatelé mohli efektivně využívat dostupné funkce. Tato příručka slouží nejen jako návod pro běžného uživatele, ale může posloužit i jako referenční příručka pro další vývojáře. V případě, že bude požadováno v budoucnu rozšířit stávají implementaci o nové funkcionality. Příručka je podpořena praktickými ukázkami a popisem věcí, se kterými se uživatel může setkat.

7.1 Stažení aplikace

Stažení je možné provést z jakéhokoliv počítače připojeného do interní sítě. Uživatel přistoupí na server pomocí adresy „visu.czechia.steinel.local“ a na této adrese se zobrazí webová stránka sloužící pro stažení instalátoru. Podoba této stránky je vidět na obrázku níže (Obrázek 21).



Obrázek 21 – Stránka pro stažení aplikace, Zdroj: vlastní

Instalátor je stažen do stažených souborů na lokálním úložišti a odsud může uživatel spustit instalaci.

7.2 Požadavky na systém a instalace

Doporučené systémové požadavky pro běh aplikace jsou následující:

- Operační systém: Windows 10 nebo Windows 11
- RAM: 4 GB
- Úložiště: 100 MB volného místa na disku
- Grafická karta: Integrovaná nebo dedikovaná s podporou DirectX 11
- .NET Verze: 8.0

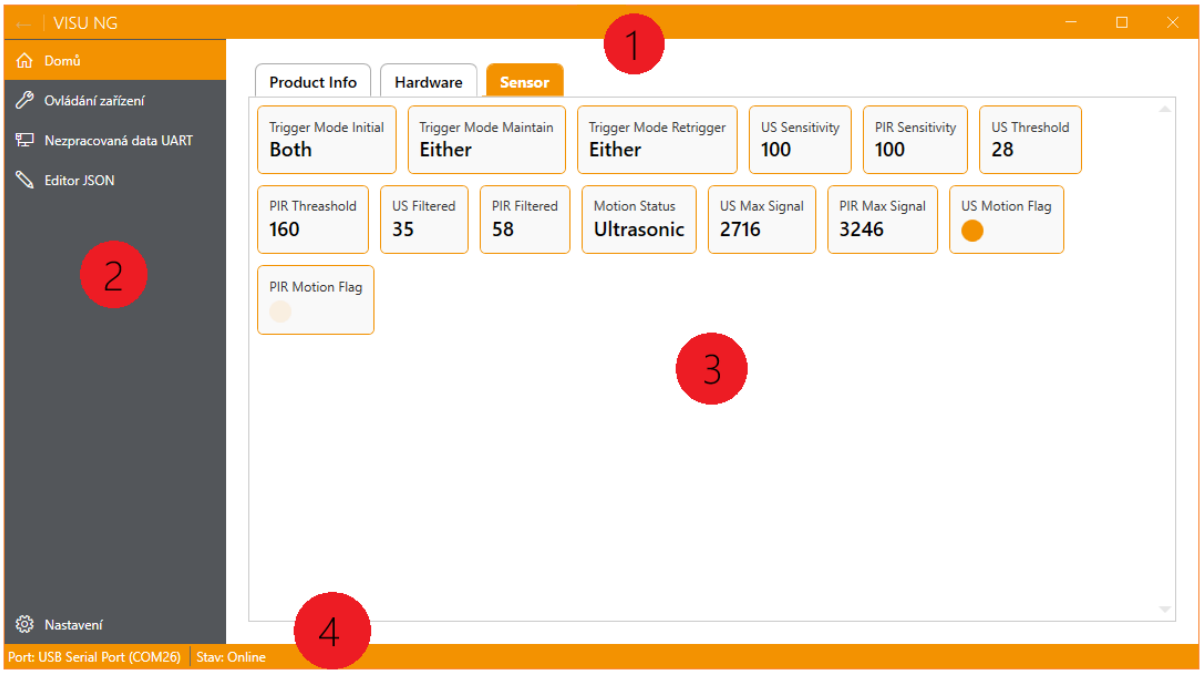
Připojení k internetu při běhu aplikace není v aktuální verzi vyžadováno. Aplikace je nainstalována automaticky pomocí instalátoru ve formátu „exe“, který si uživatel stáhl v předchozí kapitole. Poté lze najít aplikaci v nabídce start mezi ostatními programy.

7.3 Hlavní okno aplikace

Po spuštění aplikace se zobrazí hlavní okno, které okamžitě může zobrazovat údaje o zařízení v případě, že je zařízení připojeno. Vzhled a struktura celého okna je popsána v následující podkapitole.

7.3.1 Struktura hlavního okna

Vizualizace hlavního okna aplikace je znázorněna na obrázku 22. Jednotlivé části uživatelského rozhraní jsou v tomto obrázku označeny číselnými značkami a jejich popis je uveden v následující tabulce číslo 1. Popis se zaměřuje na funkční členění hlavního okna a roli jednotlivých oblastí v rámci práce se zařízením.



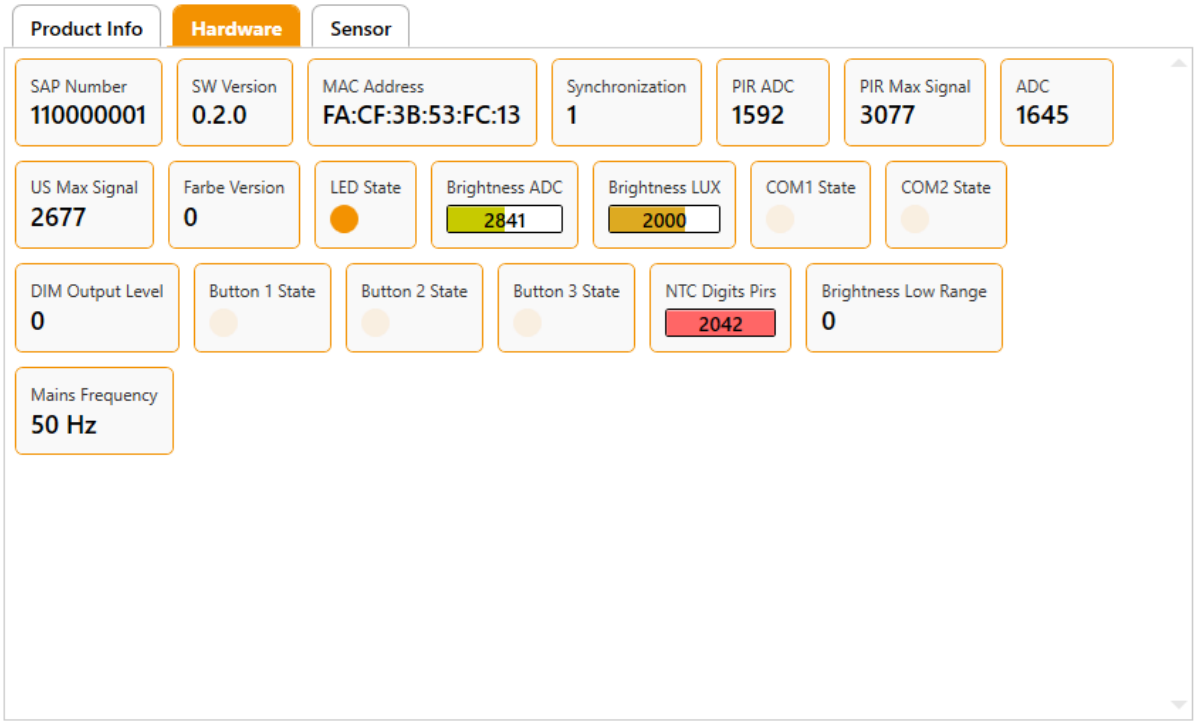
Obrázek 22 – Vizuální podoba aplikace po spuštění, Zdroj: vlastní

Číslo	Popis
1	Horní lišta aplikace obsahuje základní ovládací prvky okna. První zleva je tlačítko šipka zpět, jenž má za funkci návrat na předchozí stránky. Dále je zde uveden název aplikace a standardní tlačítka okna (minimalizace, maximalizace, ukončení) na pravé straně.
2	Navigační panel realizovaný jako postranní menu, které je dostupné napříč všemi stránkami. Menu je pevně ukotveno na levé straně okna a umožňuje uživateli rychle přecházet mezi jednotlivými sekcemi aplikace.
3	Hlavní pracovní oblast, ve které se zobrazují údaje nebo komponenty podle vybrané stránky v postranním menu.
4	Stavový řádek nacházející se na spodní liště aplikace. Informuje uživatele o aktuálním stavu komunikace se zařízením. Pokud zařízení odesílá data a aplikace je přijímá je zobrazen stav Online, v opačném případě je zařízení Offline. Dále stavový řádek popisuje název připojeného zařízení a na jaké portu COM se nachází. Podobně jako menu je stavový řádek dostupný napříč všemi stránkami.

Tabulka 1 – Popis hlavního okna

7.4 Domovská stránka

Domovská stránka aplikace na obrázku 23 zobrazuje různé informace o připojeném zařízení. Umožňuje rychle prozkoumat základní údaje a různé další přehledy, jež jsou k dispozici. V horní části stránky se nachází lišta záložek, kde každá z nich reprezentuje nějakou určitou sekci údajů od zařízení. Pod lištou záložek je samotný panel, ve kterém se nacházejí datové boxy. Každý datový box obsahuje titulek popisující hodnotu, která pod ním následuje. Datových boxů je na každé záložce proměnlivé množství a v případě, že se již nevejdou na viditelnou plochu panelu, objeví se po pravé straně rolovací lišta.

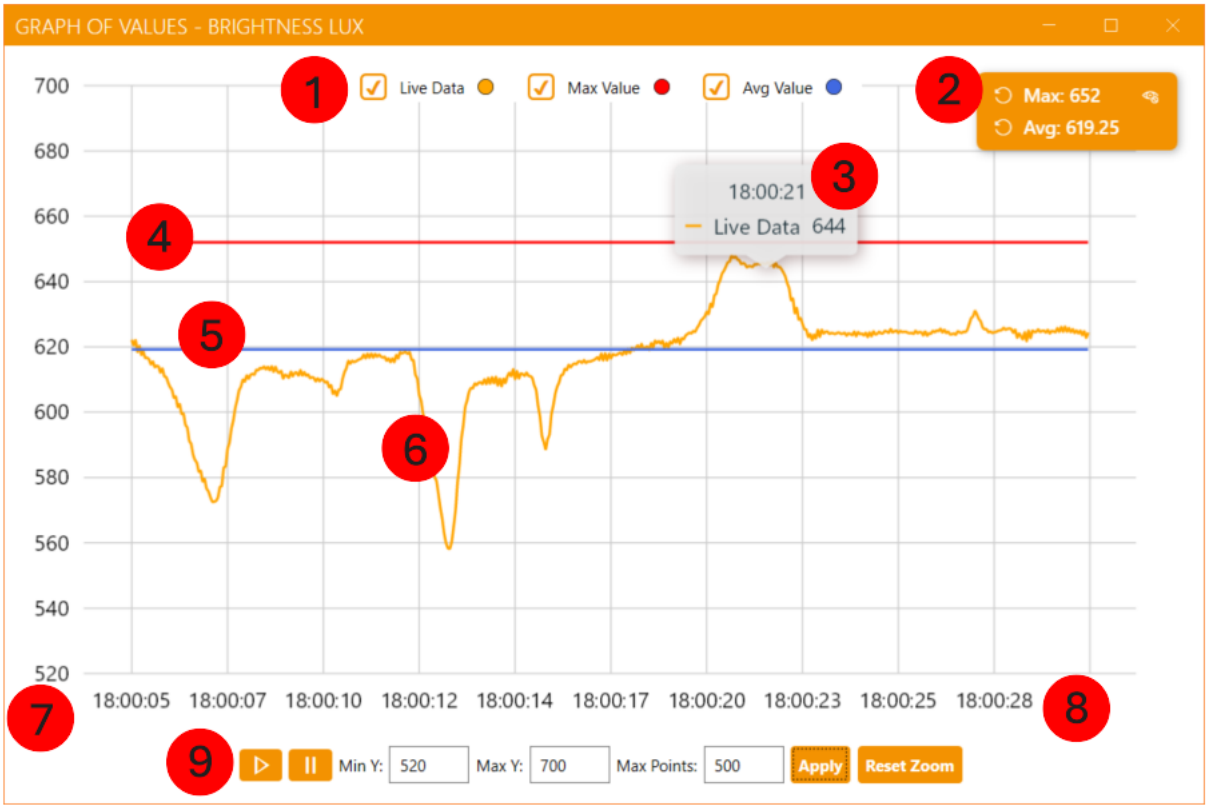


Obrázek 23 – Vizuální podoba domovské stránky, Zdroj: vlastní

Všechny hodnoty v datových boxech jsou aktualizovány dynamicky, což umožňuje sledovat změny v reálném čase. Hodnoty mohou být reprezentovány různě. Základní typ je klasickým textem, následuje ukazatel průběhu a indikátor. Ukazatel průběhu je lišta, která je procentuálně vyplněna na základě aktuální hodnoty a definice minima a maxima. Hodí se pro hodnoty, které se pohybují pouze v striktně definovaných mezích a nikdy tyto meze nepřekročí. Indikátor je realizován jako kolečko, které nabývá pouze dvou stavů. Hodí se pro hodnoty měnící se mezi dvěma stavy, například v případě 0 indikátor nesvítí a při hodnotě 1 ano.

7.4.1 Dialogové okno s grafem hodnot

V případě, že uživatel klikne na libovolný datový box na domovské stránce, otevře se mu nové dialogové okno. V tomto okně lze najít graf sloužící k vizualizaci historických hodnot daného datového boxu. Uživatel díky tomu může sledovat vývoj hodnoty parametru v čase a zároveň si přizpůsobovat zobrazení grafu dle potřeby. Dialogových oken s grafy může být v jeden moment spuštěno více najednou a lze tak monitorovat případné výkyvy mezi hodnotami. Vizuální podobu okna s grafem hodnot lze vidět na obrázku níže (Obrázek 24). O vysvětlení číselných značek se stará tabulka číslo 2 níže v této kapitole.



Obrázek 24 – Vizuální podoba grafu, Zdroj: vlastní

Číslo	Popis
1	Panel s legendou, která popisuje barvami jednotlivé čáry v grafu. U každé legendy je také zaškrtávací pole, pokud je toto pole zaškrtnuté čára je v grafu viditelná, v opačném případě se čára skryje.
2	Informační panel, který zobrazuje statistické hodnoty. Uchovává aktuální hodnoty o maximální hodnotě a průměru počítaných od spuštění instance grafu. Obě hodnoty lze resetovat za pomoci ikony reset vlevo od dané hodnoty. V tom případě aplikace

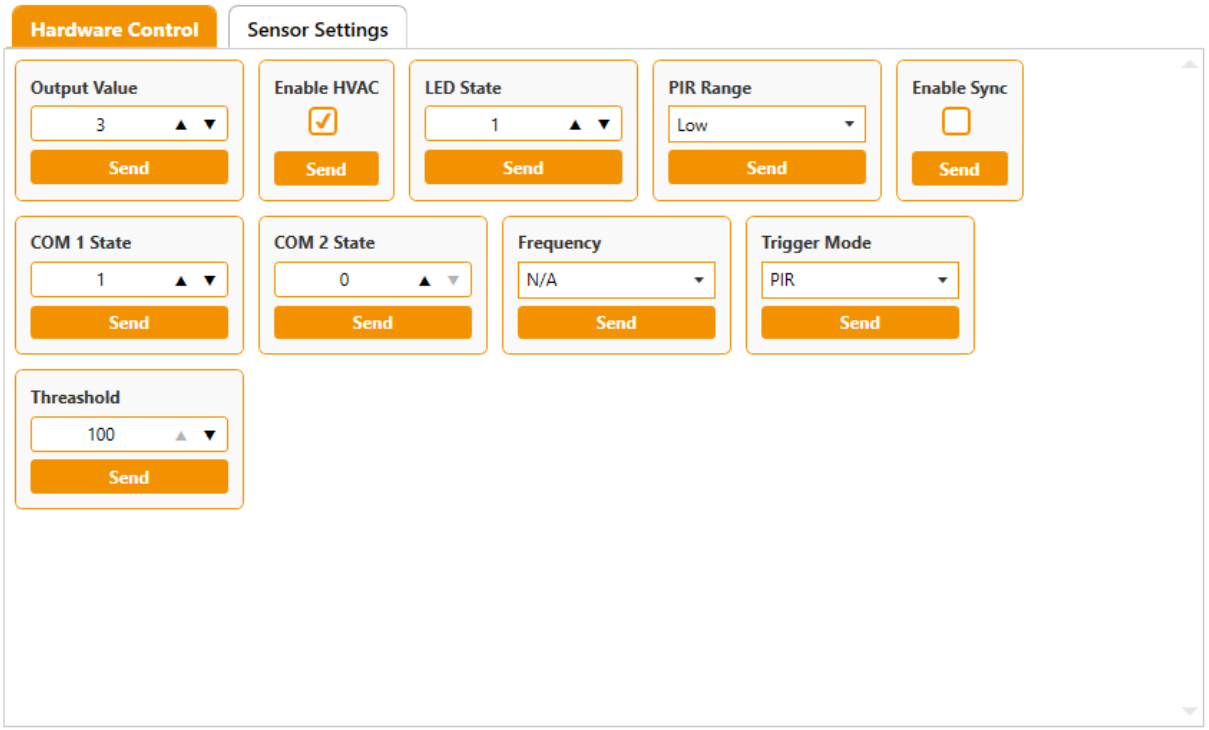
	začne maximum a průměr znovu od daného okamžiku. Jelikož informační panel částečně zasahuje do plochy grafu, je zde tlačítko pro skrytí tohoto panelu. Místo něj se objeví pouze malá ikona, díky které lze panel opět zobrazit.
3	Informační box, který se zobrazí při najetí kurzoru nad kteroukoliv část oranžové křivky. Obsahuje aktuální naměřenou hodnotu v daném časovém razítku.
4	Čára udávající hranici maximální naměřené hodnoty od spuštění grafu. Aktualizuje se pokaždé, když je nalezena nová historická maximální hodnota.
5	Čára udávající celkový průměr nad daty z aktuálního průběhu od spuštění grafu. Pozice čáry se periodicky aktualizuje nově vypočítaným průměrem při každé aktualizaci grafu. Průběh grafu si lze přibližovat buď pomocí pravého tlačítka myši a označení oblasti pro přiblížení nebo pomocí kolečka myši.
6	Příchozí data v intervalu každých 200 milisekund.
7	Osa Y udávající aktuální rozmezí hodnot.
8	Osa X grafu, která udává časová razítka hodnot, tudíž přesný časový okamžik, kdy data dorazila od zařízení.
9	Lišta s ovládáním grafu, pomocí kterého uživatel může přizpůsobit zobrazení grafu. Úplně zleva se nachází dvojice tlačítek pro spuštění nebo pozastavení živého průběhu dat. Následuje nastavení dolního a horního limitu osy Y za jehož pomoci lze nastavit přesné meze pro lepší uživatelský zážitek. Poté lze také nastavit maximální počet bodů v grafu v jeden okamžik, jelikož graf neuchovává všechny body z výkonnostních důvodů a staré body nad limit jsou průběžně mazány. Tlačítko „Apply“ slouží k potvrzení nastavení osy Y a maximálního počtu bodů, poté dojde k aktualizaci grafu. Poslední tlačítkem v liště je „Reset Zoom“, které vrátí graf do výchozí zobrazení v případě, že bylo zobrazení přiblíženo.

Tabulka 2 – Popis okna s grafem

Hlavní účelem celé této implementace grafu je poskytnutí dodatečné diagnostiky v případě nestandardních hodnot parametrů a další analýzy stability. Graf také dobře poslouží při ladění a vývoji různých embedded zařízení.

7.5 Stránka s ovládáním zařízení

Tato stránka poskytuje uživatelské rozhraní pro ruční nastavení zařízení a ovládání konkrétních parametrů, které hardwarové zařízení povoluje. Hlavní cílem této stránky je usnadnit manipulaci se zařízením při vývoji nebo testování, bez nutnosti zasahovat přímo do firmwaru zařízení. Díky přímé vazbě na zařízení skrz sériovou komunikaci je každá změna provedena okamžitě a odezva od zařízení je reflektována na domovské stránce.



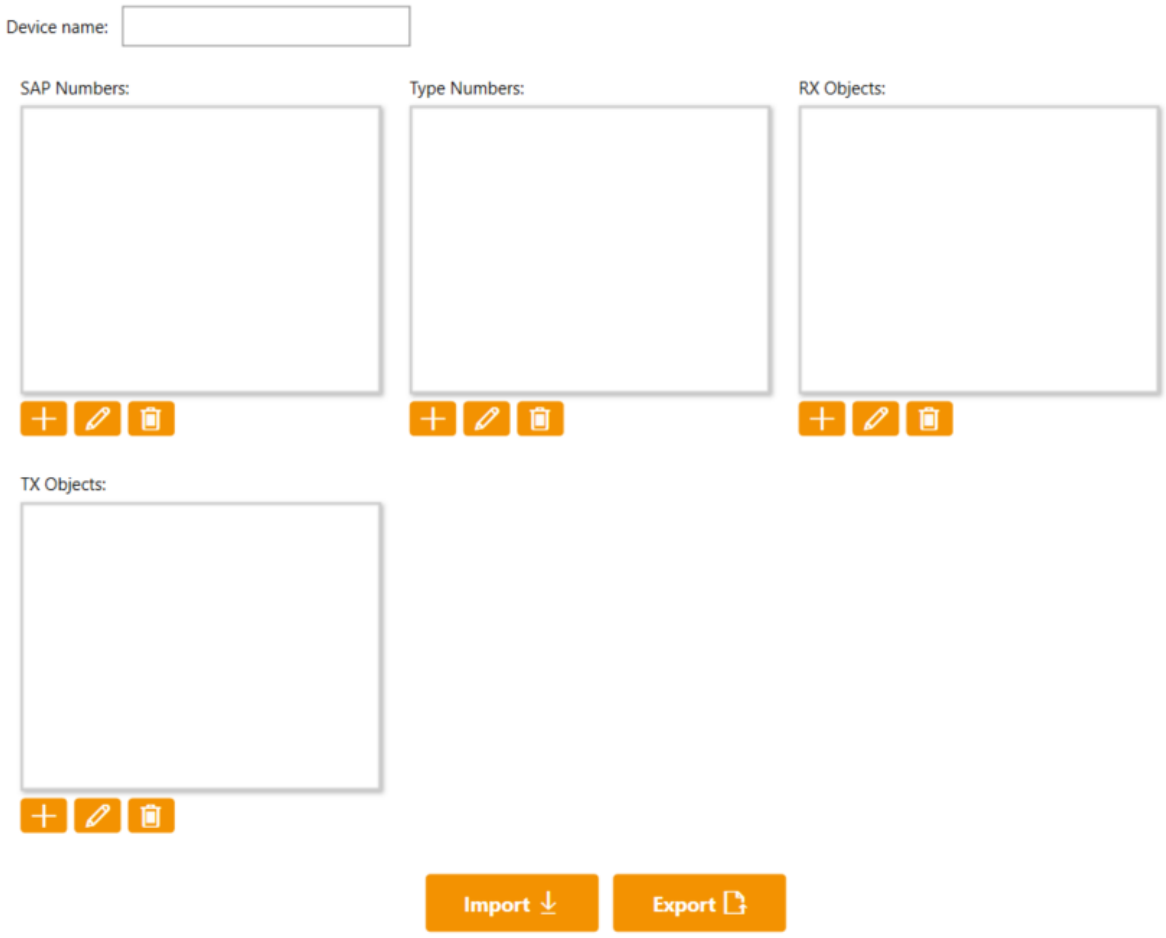
Obrázek 25 – Vizuální podoba stránky s nastavením zařízení, Zdroj: vlastní

Rozhraní zobrazené na obrázku 25 je rozděleno do samostatných ovládacích bloků a v nich jsou parametry pro nastavení zobrazeny, tak jak jsou definovány v konfiguračním souboru JSON. Jeden blok může pracovat v jednom ze dvou režimů, buď „SingleData“, kde u každého parametru je tlačítko pro odeslání paketu s právě jedním parametrem, který se má nastavit. V druhém režimu „FullTelegram“ je zde pouze jedno tlačítko, které odešle všechny parametry v bloku najednou do zařízení pro jejich nastavení.

7.6 Stránka s editorem JSON

Editor JSON má za úkol vytvářet nebo upravovat konfigurační soubory, jež jsou poté využívány pro tvorbu komponent pro reprezentaci dat na domovské stránce. Na této stránce uživatel specifikuje klíčové parametry daného zařízení, včetně SAP čísla, typového čísla a objektů RX

a TX. Položky v jednotlivých sekcích lze interaktivně přidávat, upravovat nebo odebírat. Podoba stránky s editorem je vyobrazena níže (Obrázek 26).



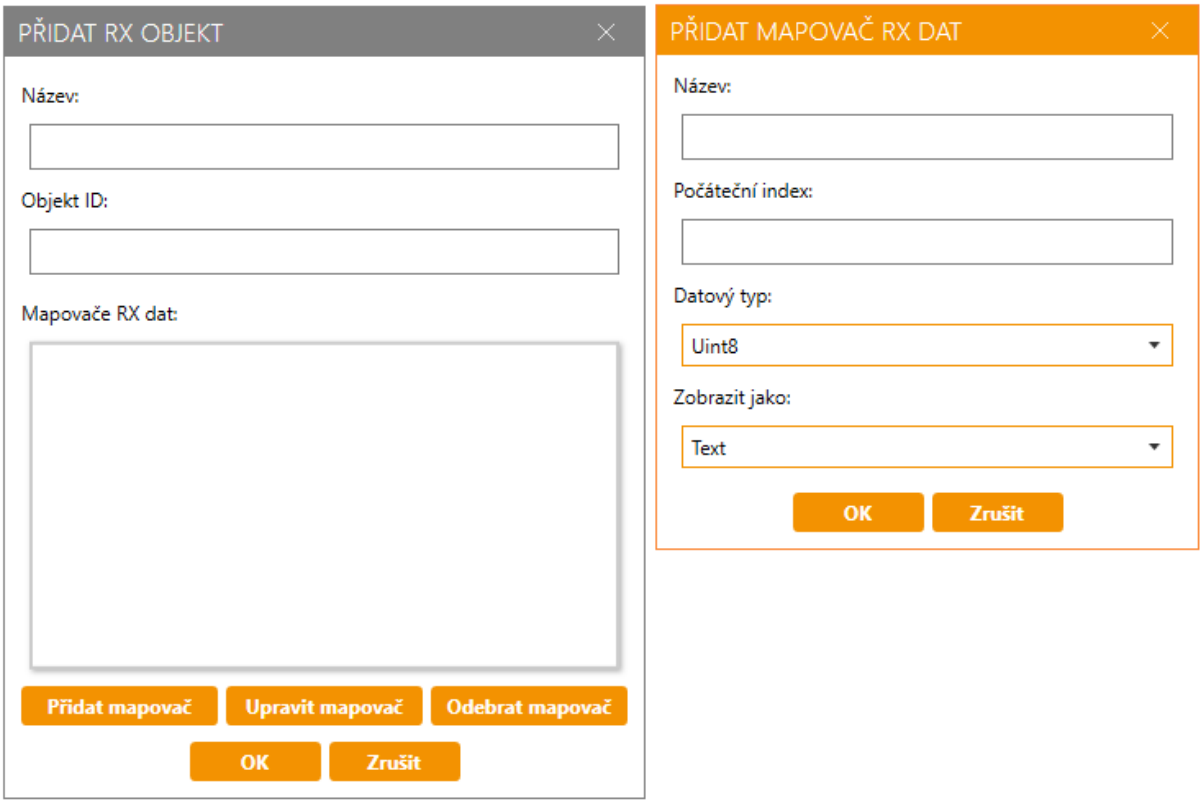
Obrázek 26 – Vizuální podoba editoru JSON, Zdroj: vlastní

V hlavní části této stránky jsou zobrazené jednotlivé údaje a výpisy objektů, které se mají uložit do souboru JSON. Na spodní části stránky jsou umístěny tlačítka pro importování nebo exportování konfigurace. Díky možnosti importovat již existující konfigurační soubor aplikace umožňuje rychlou editaci nastavení a možnost reagovat tak na drobné změny ve struktuře komunikace. Změny v konfiguraci se ihned zobrazují v editoru, ale musí být posléze exportovány pro uložení do souboru. Celý tento systém zajišťuje snadnou přenositelnost mezi počítači, rychlou správu a zálohování nastavení.

7.6.1 Dialogová okna pro objekty RX

Po kliknutí na tlačítka přidat nebo upravit pod seznamem s objekty RX se otevře nové dialogové okno. Dané dialogové okno slouží k definování přijímaných datových objektů, které se pak v aplikaci vizualizují. Každý objekt RX obsahuje jeden nebo více mapovačů, které pomáhají

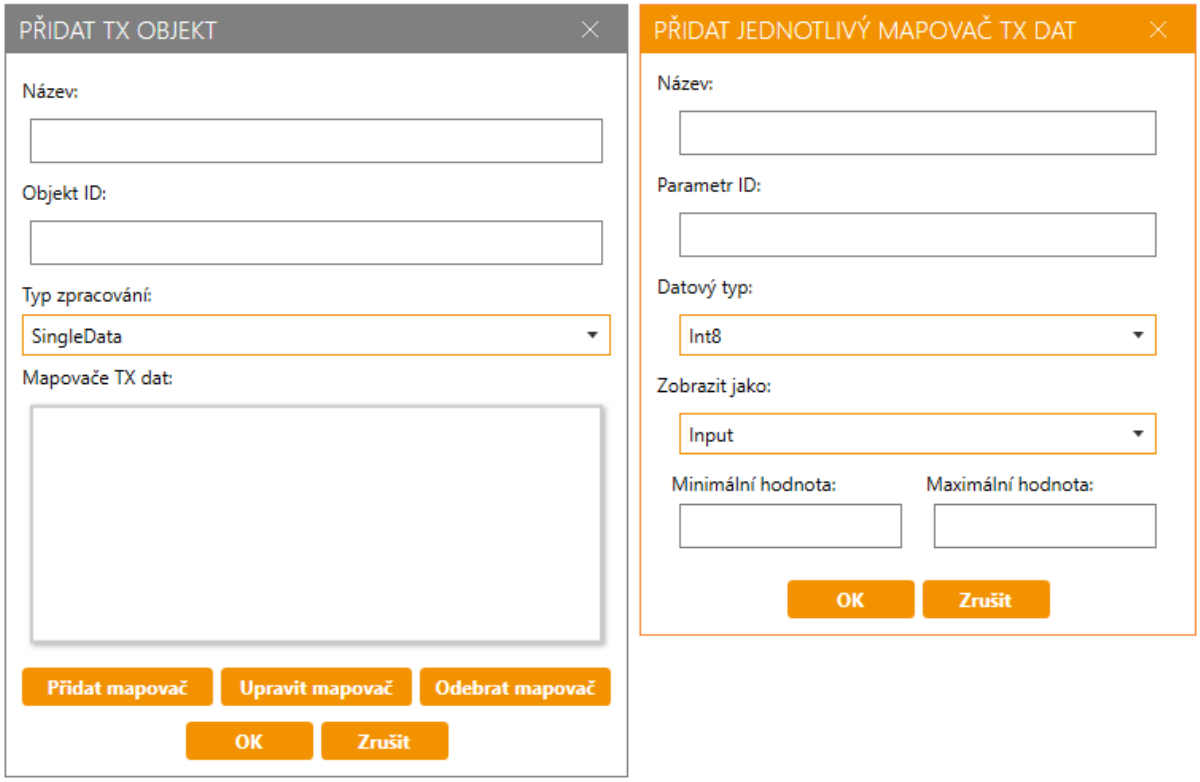
konkrétně specifikovat, jak přijímaná data zpracovat. Proto mapovače RX dat mají samostatné dialogové okno, které vychází z dialogového okna pro vytváření objektů RX. Všechny seznamy držící informace o objektech podporují standardní operace pro přidání, úpravu a odebrání. Použití těchto oken je popsáno níže v kapitole „Práce s konfiguračními soubory“. Struktury obou oken je možné vidět na obrázku níže (Obrázek 27).



Obrázek 27 – Vizuální podoba oken pro vytváření objektů RX, Zdroj: vlastní

7.6.2 Dialogová okna pro objekty TX

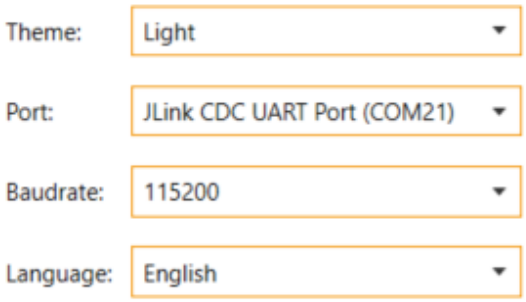
Obdobně jako je tomu u objektů RX, po kliknutí na jedno z tlačítek pro přidání nebo úpravu, zobrazí se dialogové okno pro objekty TX. Toto dialogové okno má za úkol definovat data, která budou odesílána z aplikace do zařízení. Objekty TX představují soubor parametrů, které bude možné zařízení nastavit, včetně parametrů pro přenos pomocí sériové komunikace. K objektů TX patří jejich mapovače dat TX, kde jeden mapovač dat TX představuje jeden parametr, který bude možno konfigurovat danému zařízení. Stejně jako v případě objektů RX, použití zmíněných dialogových oken je podrobněji popsáno v následující kapitole zabývající se prací s konfiguračními soubory. Struktury oken pro práci s TX objekty je možné vidět na obrázku 28.



Obrázek 28 – Struktura oken pro vytváření objektů TX, Zdroj: vlastní

7.7 Stránka nastavení

Pomocí stránky pro nastavení si uživatel může nakonfigurovat parametry aplikace. Může přizpůsobovat její vzhled nebo parametry spojené s komunikací se zařízením. Nastavení je při každé změně uloženo a při dalším spuštění opět načteno. To zajišťuje, že aplikace bude přizpůsobená dle potřeb každého uživatele. Vizuální podoba stránky s nastavením je vyobrazena na obrázku níže (Obrázek 29).



Obrázek 29 – Vizuální podoba stránky s nastavením, Zdroj: vlastní

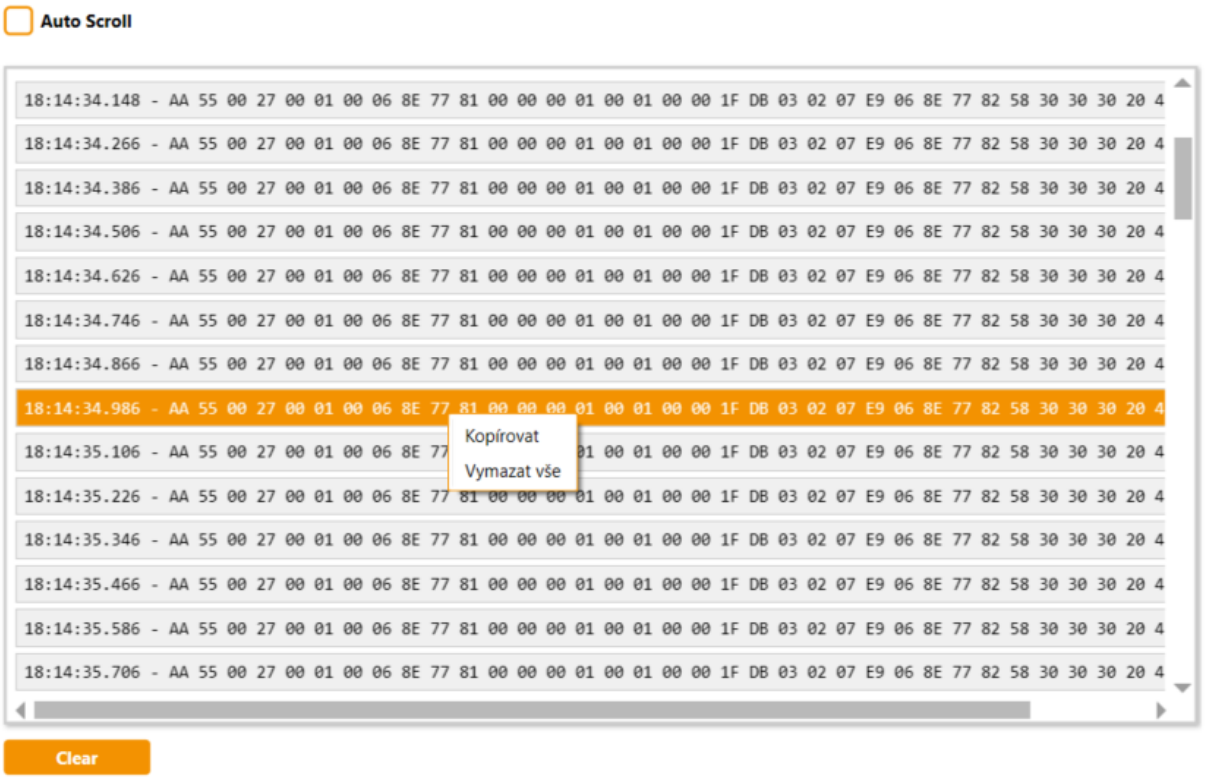
Mezi konkrétní parametry, které lze měnit patří:

- **Téma** – tato volba ovlivňuje vizuální vzhled aplikace. Zde je možné vybírat ze tří možností. První je automatické nastavení vzhledu dle operačního systému, druhá volba je světlý režim, který přizpůsobí uživatelské rozhraní do povětšinou bílé a oranžové barvy. Třetí možností je tmavý režim, jenž naopak nastaví na všech stránkách kombinaci černé a oranžové barvy.
- **Port** – touto volbou je uživatel schopný aplikaci říct jaký vybrat sériový komunikační port, od kterého se má snažit přijmout data. V rozbalovacím menu jsou uvedené všechny zařízení a jejich porty, ke kterým se aplikace může připojit.
- **Baudrate** – neboli přenosová rychlost pro sériovou komunikaci. Standardní hodnota je nastavena na hodnotu 115200, ale na výběr jsou v rozbalovací nabídce další přenosové rychlosti, aby bylo vyhověno požadavkům na rychlost pro všechna zařízení.
- **Jazyk** – poslední volbou v menu je jazyková lokalizace aplikace. Uživatel má na výběr ze tří základních jazyků (angličtina, čeština a němčina). Po vybrání jednoho jazyka z rozbalovací nabídky se celé uživatelské menu automaticky přepne do cílového jazyka bez nutnosti restartu aplikace.

Na této stránce se v levém dolním rohu také nachází informace o verzi a času sestavení ve vývojovém prostředí. Tyto informace poskytují pomoc při ladění chyb a zajištění, že uživatelé pracují se správnou verzí, která je momentálně určená pro používání.

7.8 Stránka s logováním sériové komunikace

Nezpracovaná data během komunikace jsou monitorována a zobrazována na stránce s názvem „Nezpracovaná data UART“. Na této stránce si lze zobrazit surová data přijatá prostřednictvím sériové komunikace na protokolu UART. Uživatel může sledovat příchozí pakety v reálném čase a analyzovat jejich obsah ve formě hexadecimálního výpisu, jak je vidět na obrázku 30. Takové logování může velice pomoct v případě, že je v komunikačním protokolu chyba.



Obrázek 30 – Vizuální podoba stránky s logováním dat, Zdroj: vlastní

Na stránce s logováním neprobíhá pouze výpis dat, ale lze zde také najít v horní části funkci pro automatické posouvání výpisu. Funkce je realizována pomocí zaškrtávacího pole, kde v případě, že je zaškrtnuté tak se seznam s výpisem neustále posouvá dolů. Tím pádem jsou vždy vidět ta nejnověji přijatá data. Pokud není zaškrtnuto seznam zůstane na stejném místě a uživatel má tak možnost prohlížet historii přijatých dat. V prostřední části okna je již zmiňovaný výpis dat. Každý řádek tohoto výpisu obsahuje v první řadě časový údaj neboli okamžik přijetí paketu, následovaný hexadecimálními daty. Nad každým záznamem ve výpisu lze vyvolat kontextové menu za pomoci pravého tlačítka myši. Za pomoci tohoto menu lze kopírovat jednotlivé záznamy pro potřeby vložení na jiné místo. Položky lze také vybírat a kopírovat ve větším počtu, to se realizuje klávesou Control a klikáním levým tlačítkem myši na položky, které chce uživatel zkopírovat. Ve spodní části stránky se nachází tlačítko pro vymazání veškerého obsahu v seznamu. Po jeho stisknutí se vymaže celá historie komunikace a seznam se začíná plnit od začátku. Tlačítko pro mazání historie lze najít i v kontextovém menu pro lepší uživatelský komfort. Nutno ještě podotknout, že z důvodu paměťové náročnosti není seznam pro logování nekonečný a od dovršení určitého počtu záznamů, začnou nejstarší záznamy mizet.

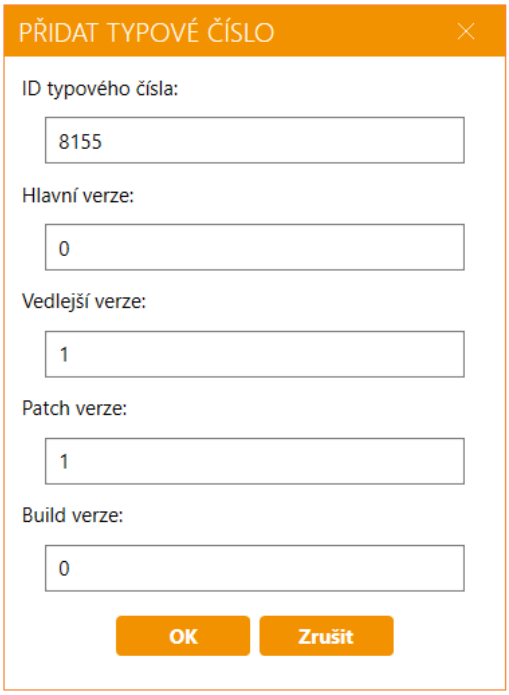
7.9 Práce s konfiguračními soubory

Následující podkapitola slouží jako podrobný průvodce tvorbou a úpravou konfiguračních souborů, které určují, jak aplikace zpracovává a vizualizuje data ze zařízení. Cílem je poskytnout návod, díky kterému bude uživatel schopný sám nastavit požadované parametry v jednotlivých dialogových oknech na stránce s editorem JSON.

7.9.1 Postup vytvořením konfiguračního souboru JSON

Po spuštění aplikace uživatel musí přejít na stránku „Editor JSON“, kde má možnost načíst již existující konfigurační soubor nebo začít vytvářet nový. Tato sekce se nejprve zaměří na vytvoření nového konfiguračního souboru od začátku. Uživatel musí vyplnit minimální parametry pro identifikaci zařízení a jeden nebo více objektů RX pro vizualizaci.

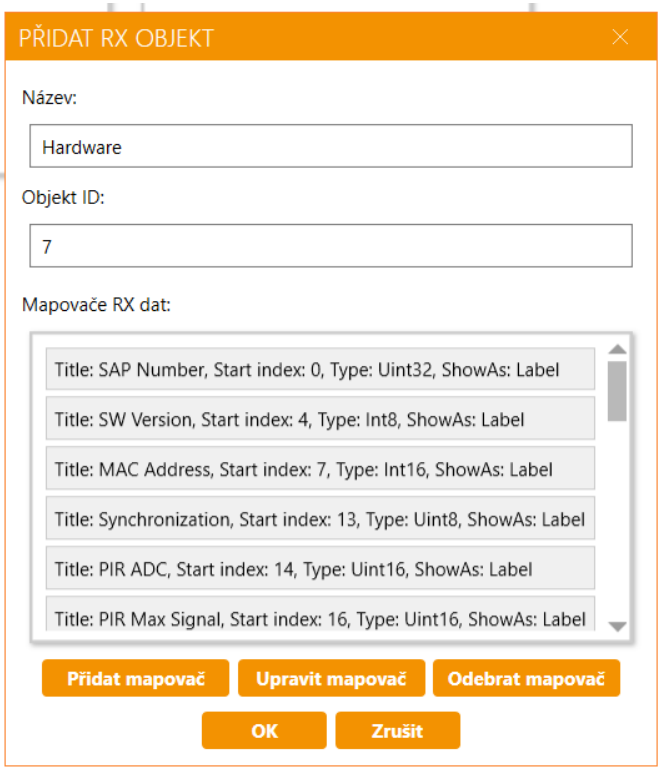
Přidání čísla SAP probíhá jednoduše stisknutím ikony plus pod seznamem čísel SAP. V nově otevřeném dialogovém okně je vyžadováno zadat validní číselnou hodnotu. Nově přidané validní číslo se zobrazí v příslušném seznamu. Stejným způsobem proběhne přidání typového čísla. Uživateli se zobrazí okno jako je na obrázku 31. Zde je nutné zadat jednotlivé verze v jakých se firmware nachází a také zvolit typové číslo, které v kombinaci s verzemi bude tvořit jedinečný identifikátor pro zařízení. Typové číslo může být jakékoliv číslo a to samé platí u jednotlivých verzí.



Obrázek 31 – Ukázka přidávání nového typového čísla, Zdroj: vlastní

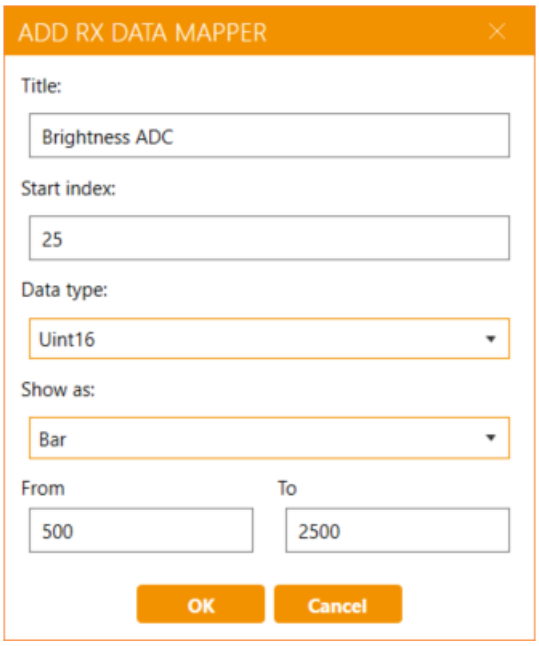
Důležité je vědět, že není nutné mít v jedné konfiguraci nastavená SAP a typová čísla zároveň. Podmínkou je mít nastavený alespoň jeden záznam z těchto dvou, což stačí pro jednoznačnou identifikaci zařízení.

Dalším krokem je přidat do seznamu objekty RX. V dialogovém okně pro tvorbu objektu RX je nutné vyplnit jeho jméno, čímž může být třeba druh informací nebo něco co jednoznačně popisuje povahu dat pro daný objekt. Do textového pole „Object ID“ je potřeba vložit hodnotu, dle které aplikace při komunikaci pozná, že se jedná o objekt určitého typu a bude schopná správně namapovat paket. Hodnoty pro „Object ID“ musí uživatel manuálně vyčíst ze specifikace protokolu a tím zjistí, jaké všechny druhy paketů je schopné zařízení poslat. Možná podoba vyplnění atributů je viditelná na dalším obrázku (Obrázek 32).



Obrázek 32 – Ukázka přidávání nového objektu RX, Zdroj: vlastní

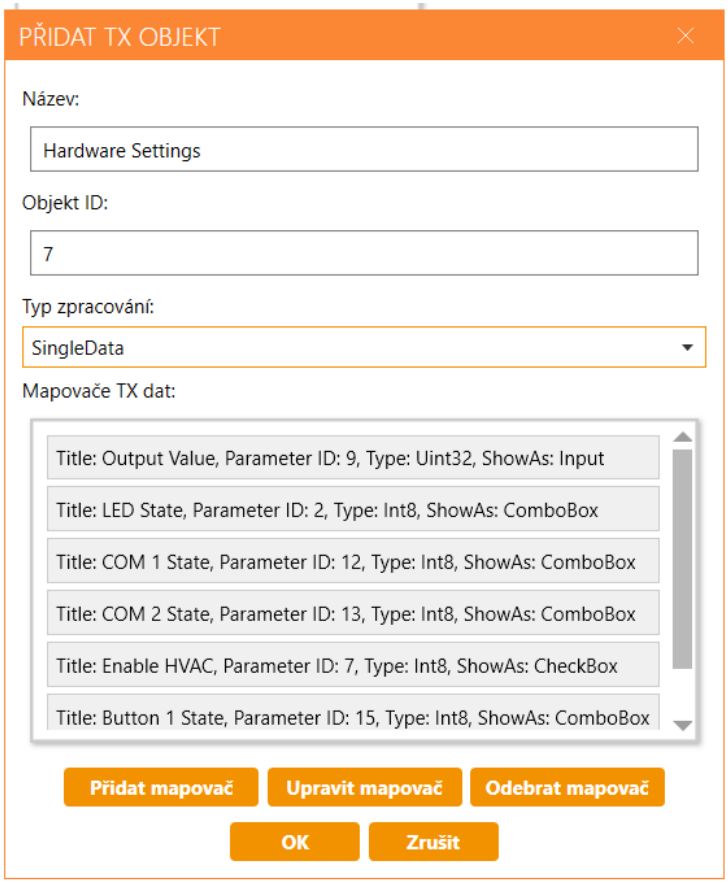
Následuje přidávání konkrétních mapovačů dat. V dalším dialogovém okně, tentokrát pro tvorbu mapovače je nutné zadat jeho název neboli nadpis, který bude potom nadepsaný nad hodnotou na domovské stránce. Další hodnotou je „Start index“, který udává pozici hodnoty v paketu, kde je opět nutné mít u sebe specifikaci paketu pro daný protokol a vyčíst kde, jaká hodnota začíná. Poté je nutné specifikovat v rozbalovacím menu datový typ hodnoty neboli kolik bajtů se má přečíst v paketu jako jedna hodnota. V poslední menu je potřeba zvolit cílovou komponentu jakou bude hodnota na domovské stránce zobrazována. Pro hodnoty, u kterých se neví jejich přesný rozsah, je doporučeno zvolit komponentu Label. U hodnot se známým rozsahem je vhodné vybrat komponentu Bar. Pokud hodnota nabývá pouze dvou stavů, například 0 a 1, nejlepší volbou je Indicator. Příklad vytváření mapovače dat je viditelný na obrázku dále v této kapitole (Obrázek 33).



Obrázek 33 – Ukázka přidávání nového mapovače pro RX data, Zdroj: vlastní

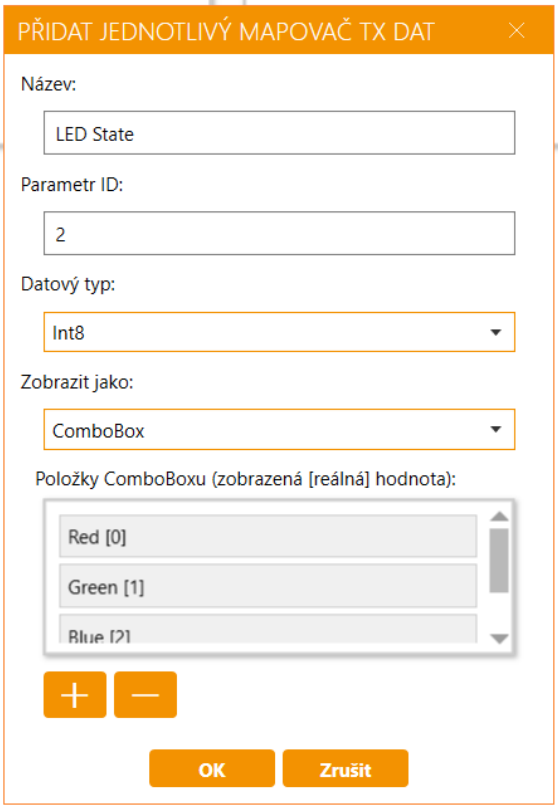
Po vytvoření mapovačů na všechny požadované hodnoty z paketu, vyplněné ID objektu a jeho jméno, má uživatel hotovou konfiguraci pro příjem jednoho typu paketu ze specifikovaného zařízení podle zadaného čísla SAP nebo typového čísla.

V případě, že je potřeba zařízení také konfigurovat, je nutné vytvořit také objekty TX. Vytváření objektů TX probíhá velice podobně, jako je tomu u objektů RX. Tudíž opět je nutné v novém dialogovém okně vyplnit jméno a ID objektu. Rozdílem oproti objektu RX je specifikace zpracování. V rozbalovací nabídce jsou možnosti „SingleData“ pro odesílání parametrů definovaných mapovači jednotlivě nebo „FullTelegram“, kde se vezmou všechny parametry a odešlou se v jednom paketu. V závislosti na vybraném typu zpracování se zobrazí příslušné okno pro mapovač, kde za použití typu „SingleData“ je vyžadováno ID parametru a v případě „FullTelegram“ se musí zase specifikovat počáteční index v paketu odesílaného do zařízení. Obrázek 34 zobrazuje možné vyplnění atributů tohoto dialogového okna.



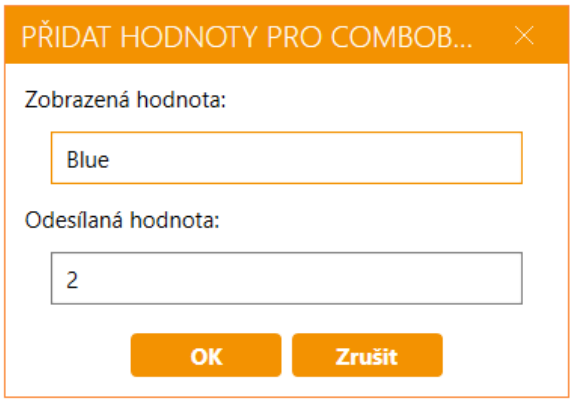
Obrázek 34 – Ukázka přidávání nového objektu TX, Zdroj: vlastní

Po zvolení předchozích atributů je načase přidat mapovače pro parametry, jenž je potřeba konfigurovat v daném zařízení. To se provede v novém okně pro tvorbu mapovačů dat TX. Zde je nutné definovat, jak se daný parametr bude jmenovat a jaké je jeho ID. Dále vybrat z nabídky datový typ parametru. Opět je důležité říct, že jak ID, tak i datový typ parametru je potřeba vyčíst z dokumentace použitého komunikačního protokolu. Na obrázku níže (Obrázek 35) lze vidět příklad vytvoření nového mapovače pro data TX.



Obrázek 35 – Ukázka přidávání mapovače pro TX data, Zdroj: vlastní

Posléze se přejde na výběr zobrazení, kde se v rozbalovací nabídce určí, jakým způsobem bude na stránce s ovládáním zařízení daný parametr volen. Na výběr jsou tři možnosti, kde na prvním místě je klasický Input hodící se pro nastavení většiny parametrů. U možnosti Input je možné specifikovat i rozsah hodnot. Tím se ošetří nesprávné vstupy. V případě, že tato možnost není vyplněna použijí se výchozí hodnoty 0 a 100. Pokud je u parametru přípustné nastavení pouze několika hodnot, například do 10 hodnot, je doporučeno použít možnost „ComboBox“. Při této možnosti se zobrazí seznam, do kterého uživatel může přidat jednotlivé hodnoty, které půjdou u parametru nastavit. Přidání hodnoty, která se stane položkou komponenty „ComboBox“ probíhá v novém okně, jak je možné vidět na obrázku níže (Obrázek 36).



Obrázek 36 – Ukázka definice hodnot pro mapovaný „ComboBox“, Zdroj: vlastní

V okně je potřeba specifikovat hodnotu, která se použije jako položka na výběr a druhá hodnota představuje reálnou hodnotu, která se pošle do zařízení. Tím se přejde jakýmkoliv špatně zadaným hodnotám při nastavování zařízení. Poslední volbou je „CheckBox“, jenž se hodí pro nastavování parametrů hodnot od 0 do 1. Na konec stačí všechna nastavení potvrdit a přidat nový objekt TX do seznamu.

Obrázek 37 – Ukázka fungující konfigurace, Zdroj: vlastní

Nyní je všechno nastaveno, jak pro vizualizaci příchozích dat na domovské stránce, tak i pro nabídku nastavení na stránce ovládání zařízení. Úplně posledním krokem je uložení celé konfigurace. To se provede tlačítkem pro exportování nacházející se v dolní části stránky s editorem, jak je možné vidět na obrázku 37. Po jeho kliknutí uživatel vybere jeho umístění na disku. Důležité je zmínit, že pro okamžité nasazení konfigurace při dalším spuštění aplikace je nutné vytvořenou konfiguraci umístit do složky „DeviceDescription“, která se nachází ve složce s aplikací.

ZÁVĚR

Cílem této práce bylo navrhnout a implementovat desktopovou aplikaci pro vizualizaci dat ze zařízeních a konfiguraci jejich parametrů za použití sériové komunikace. Zadání vycházelo z reálné potřeby firmy nahradit nejednotná a hlavně zastaralá řešení. Výsledná aplikace oproti těmto řešením je nejenom technologicky aktuální, ale také více přívětivá pro koncové uživatele s velkou mírou flexibility pro budoucí potřeby vývoje.

Během práce byly analyzovány současně používané nástroje pro práci se zařízeními přes sériovou komunikaci a byly vyhodnoceny jejich nedostatky. Tato analýza potvrdila jejich nízkou úroveň přizpůsobení a problematickou správu při používání s různými zařízeními firmy Steinel. Na základě těchto problémů byla navržena aplikace, která je postavena na platformě WPF s využitím dodatečných knihoven jako jsou MahApps.Metro a LiveCharts2.

V praktické části byla vytvořena aplikace, která umožňuje vizualizaci dat ze zařízení v reálném čase, konfiguraci parametrů zařízení pomocí uživatelsky definovaných komponent a to vše díky snadno přenosné konfiguraci ve formátu JSON. Důraz byl také kladen na moderní uživatelsky přívětivé prostředí, jazykovou lokalizaci rozhraní a přepínání barevných režimů, což je klíčové při provádění testů v laboratorních podmínkách. Systém byl vytvořen s důrazem na modularitu, která umožní práci i s budoucími komunikačními protokoly nebo přidávání nových funkcionalit bez většího zásahu do stávajícího kódu. Součástí zpracování byly také jednotkové testy pro určité části systému, hlavně co se týče vytváření konfiguračních paketů nebo výpočtu kontrolních součtů.

Při vývoji aplikace bylo nutné čelit několika výzvám, zejména co se týče návrhu struktury konfigurace všech komponent pro dané zařízení a následné zpracování konfigurací pro mapování komponent v uživatelském rozhraním. Z počátku také autor neměl žádné zkušenosti s technologií WPF a bylo nutné před samotnou implementací nastudovat tuto technologii. I přes tyto výzvy se autorovi podařilo implementovat všechny stanovené požadavky na počátku zpracování praktické části.

V budoucnu je v rámci firmy Steinel plánováno tuto aplikaci rozšiřovat o nové funkce nebo vylepšovat její zabezpečení. Jedním z prvních kroků vpřed je přidání nové stránky s přehrávačem historických dat pro ještě detailnější analýzy a další funkce vyplývající z požadavků vývojářů. Závěrem lze říci, že výsledná aplikace představuje flexibilní řešení, které položilo základní kameny pro další vývoj v oblasti interních nástrojů firmy Steinel.

POUŽITÁ LITERATURA

[1] NATHAN, Adam. Online. In: *WPF 4: Unleashed*. Sams, 2010, s. 11-24. ISBN 9780672331190. Dostupné z: https://github.com/adamshe/books/blob/master/WPF-4-unleashed.pdf. [cit. 2025-04-02].

[2] MICROSOFT. *Desktop Guide (WPF .NET)*. Online. Microsoft. 10/29/2024. Dostupné z: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-8.0. [cit. 2025-04-02].

[3] MICROSOFT. *XAML overview (WPF .NET)*. Online. Microsoft. 10/29/2024. Dostupné z: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/xaml/?view=netdesktop-9.0. [cit. 2025-04-02].

[4] NATHAN, Adam. Online. In: XAML Unleashed. Sams, 2014, s. 6. ISBN 9780672337222. Dostupné z: https://books.google.cz/books?hl=cs&lr=&id=LjTtBQAAQBAJ&oi=fnd&pg=PR5&dq=XAML&ots=lb5KtMPJ2-&sig=FwM3QdhIY-Ric36VM9_GgPQFOlVA&redir_esc=y#v=onepage&q=XAML&f=false. [cit. 2025-04-02].

[5] MAHAPPS. *Documentation*. Online. MahApps. 2014, 2023. Dostupné z: https://mahapps.com/docs/. [cit. 2025-04-02].

[6] *[Demo Application]*. Online. In: GitHub. 2020. Dostupné z: https://github.com/MahApps/MahApps.Metro/blob/develop/docs/2020-05-23_14h26_59.png. [cit. 2025-04-10].

[7] RODRÍGUEZ, Alberto. *LiveCharts2*. Online. In: GitHub. 2023. Dostupné z: https://github.com/beto-rodriguez/LiveCharts2/blob/master/README.md. [cit. 2025-04-02].

[8] BASSETT, Lindsay. Online. In: *Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON*. O'Reilly Media, 2015, s. 1-9. ISBN 9781491929483. Dostupné z: https://books.google.cz/books?hl=cs&lr=&id=Qv9PCgAAQBAJ&oi=fnd&pg=PP1&dq=JSON&ots=g5BVN-LYm2O&sig=XYd3pBHxceSfkoZm3bUS6Pawd70&redir_esc=y#v=onepage&q=JSON&f=false. [cit. 2025-04-02].

[9] *Oblíbená komunikační rozhraní a protokoly*. Online. In: TME. TME Electronics Components. 2024. Dostupné z: https://www.tme.eu/cz/news/library-articles/page/60826/oblibena-komunikacni-rozhrani-a-protokoly/. [cit. 2025-04-02].

[10] PETERKA, Jiří. *Základní formy přenosů*. Online. In: EArchiv.cz. 1991. Dostupné z: https://www.earchiv.cz/a91/a140c110.php3. [cit. 2025-04-02].

[11] MANKAR, Jayant. REVIEW OF I2C PROTOCOL. Online. *International Journal of Research in Advent Technology*. January 2014, roč. 2014, č. 2, s. 474-478. ISSN 2321–9637. Dostupné z: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=314537daa1f601f83044b25b68e2af6c8f331f3f. [cit. 2025-04-09].

[12] DHAKER, Piyu. *Introduction to SPI Interface*. Online. In: Allelco. September 2018. Dostupné z: https://www.allelcoelec.in/datasheets.b8/ADG1412YCPZ-REEL7.pdf. [cit. 2025-04-09].

[13] AXELSON, Jan. Online. In: *USB Complete: The Developer's Guide*. Fifth edition. Lakeview Research, 2015, s. 23-58. ISBN 9781931448284. Dostupné z: https://dslev.narod.ru/files_STM32/usb-complete_5.pdf. [cit. 2025-04-09].

[14] PEÑA, Eric a LEGASPI, Mary Grace. *UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter*. Online. In: ANALOG DEVICES. Analog. December 2020. Dostupné z: https://fgcoca.github.io/Autocaravana-inteligente/datasheet/uart-a-hardware-communication-protocol.pdf. [cit. 2025-04-09].

[15] *UART s datovou sběrnicí*. Online. In: VOXCAFE. 2022. Dostupné z: https://www.voxcafe.cz/data/Articles/uart/datsbernice.jpg. [cit. 2025-04-09].

[16] *Software UART through emulated GPIO pins*. Online. In: JIMMYIOT. 2022. Dostupné z: https://jimmywongiot.com/2022/10/23/software-uart-through-emulated-gpio-pins/. [cit. 2025-04-09].

[17] PEÑA, Eric a LEGASPI, Mary Grace. *UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter*. Online. In: Analog Devices. 2020. Dostupné z: https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html. [cit. 2025-04-09].

[18] WANG, Kankan a LIU, Hao. *RS-485 Basics Series*. Online. In: Texas Instruments. FEBRUARY 2021. Dostupné z: https://www.ti.com/lit/wp/slla545/slla545.pdf?ts=1738389468222. [cit. 2025-04-10].

[19] SPATARU, Alex. *Introduction to Serial Studio*. Online. In: GitHub. 2024. Dostupné z: https://github.com/Serial-Studio/Serial-Studio/wiki. [cit. 2025-04-10].

[20] *[SerialTool Screenshot]*. Online. In: SerialTool. C2021-2025. Dostupné z: https://www.serialtool.com/assets/img/ScreenShots/2024/serialtool-serial-hex-terminal_windows_2000.png. [cit. 2025-04-10].

[21] *SerialTool: The Most Complete Serial Port Software*. Online. SerialTool. C2021-2025. Dostupné z: https://www.serialtool.com/_en/serial-port-software-why. [cit. 2025-04-10].

[22] *Serial Terminal Basics: Real-Term (Windows)*. Online. In: Sparkfun. 2013. Dostupné z: https://learn.sparkfun.com/tutorials/terminal-basics/real-term-windows. [cit. 2025-04-10].

PŘÍLOHY

PŘÍLOHA A – STRUKTURA IDENTIFIKAČNÍHO PAKETU

Pořadí bajtů	Název	Popis
0.	Sync_0	Synchronizační bajt, konstantní hodnota 0xAA
1.	Sync_1	Synchronizační bajt, konstantní hodnota 0x55
2.	Length_L	Délka dat
3.	Length_L	
4.	Object_ID_H	Identifikace typu objektu
5.	Object_ID_L	
6.	Status	Stav zařízení
7.	SW_Materialnumber_3	Číslo SAP firmwaru
8.	SW_Materialnumber_2	
9.	SW_Materialnumber_1	
10.	SW_Materialnumber_0	
11.	SW_Version_Major_Release_1	Číslo hlavní verze firmwaru
12.	SW_Version_Major_Release_0	
13.	SW_Version_Minor_Release_1	Číslo vedlejší verze firmwaru
14.	SW_Version_Minor_Release_0	
15.	SW_Version_Patch_Level_1	Číslo záplaty firmwaru
16.	SW_Version_Patch_Level_0	
17.	SW_Version_Build_No_1	Číslo kompilační verze firmwaru
18.	SW_Version_Build_No_0	
19.	Compilation_Date_Day	Den kompilace firmwaru
20.	Compilation_Date_Month	Měsíc kompilace firmwaru
21.	Compilation_Date_Year_1	Rok kompilace firmwaru
22.	Compilation_Date_Year_0	
23.	PCB_Materialnumber_3	Číslo SAP elektronické desky
24.	PCB_Materialnumber_2	
25.	PCB_Materialnumber_1	
26.	PCB_Materialnumber_0	
27.	PCB_Index_7	Kód fyzické desky

28.	PCB_Index_6	
29.	PCB_Index_5	
30.	PCB_Index_4	
31.	PCB_Index_3	
32.	PCB_Index_2	
33.	PCB_Index_1	
34.	PCB_Index_0	
35.	CRC_H	Hodnota kontrolního součtu
36.	CRC_L	
37.	Compilation_Time_Hour	Hodina kompilace firmwaru
38.	Compilation_Time_Min	Minuta kompilace firmwaru
39.	Compilation_Time_Sec	Sekunda kompilace firmwaru

PŘÍLOHA B – STRUKTURA AKTIVAČNÍHO PAKETU PRO DANÝ OBJEKT

Pořadí bajtů	Název	Popis
1.	Sync Byte 1	Synchronizační bajt 0xAA pro začátek paketu
2.	Sync Byte 2	Synchronizační bajt 0x55 pro začátek paketu
3.	Length High Byte	Délka paketu
4.	Length Low Byte	
5.	Object ID High Byte	Číslo objektu pro aktivaci
6.	Object ID Low Byte	
7.	Read/Write Flag	Indikace operace čtení nebo zápisu
8.	Data	Hodnota datového bajtu
9.	CRC High Byte	Kontrolní součet
10.	CRC Low Byte	

PŘÍLOHA C – STRUKTURA PAKETU PRO NASTAVENÍ PARAMETRŮ ZAŘÍZENÍ

Pořadí bajtů	Název	Popis
1.	Sync Byte 1	Synchronizační bajt 0xAA pro začátek paketu
2.	Sync Byte 2	Synchronizační bajt 0x55 pro začátek paketu
3.	Length High Byte	Délka paketu
4.	Length Low Byte	
5.	ID High Byte	Identifikátor objektu
6.	ID Low Byte	
7.	Read/Write Flag	Indikace operace čtení nebo zápisu
8.	Parameter ID	Identifikátor nastavovaného parametru
9.	Data Byte 1 (MSB)	Hodnota nastavovaná parametru
10.	Data Byte 2	
11.	Data Byte 3	
12.	Data Byte 4 (LSB)	
13.	CRC High Byte	Kontrolní součet
14.	CRC Low Byte	