TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Tvorba aplikace pro přímé ovládání a parametrování měniče z PC pomocí USS protokolu

Bakalářská práce

Studijní program: B2646 – Informační technologie

Studijní obor: 1802R007 – Informační technologie

Autor práce: **Kateřina Lebedová**

Vedoucí práce: Ing. David Lindr, Ph.D.

Liberec 2017

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Development of an application for direct control and parameterisation of the frequency converter from a PC using the USS protocol

Bachelor thesis

Study programme:	B2646 – Information Technology
Study branch:	1802R007 – Information Technology

Author:	**Kateřina Lebedová**
Supervisor:	Ing. David Lindr, Ph.D.

Liberec 2017

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: 2016/2017

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	Kateřina Lebedová
Osobní číslo:	M14000050
Studijní program:	B2646 Informační technologie
Studijní obor:	Informační technologie
Název tématu:	Tvorba aplikace pro přímé ovládání a parametrování měniče z PC pomocí USS protokolu
Zadávající katedra:	Ústav mechatroniky a technické informatiky

Zásady pro vypracování:

1. Seznamte se s dokumentací, procesem parametrizace měničů Siemens Sinamics a principem komunikace prostřednictvím USS protokolu.
2. Vytvořte aplikaci pro PC, pomocí které bude možné jednoduše nastavovat parametry měniče, ukládat aktuální parametrickou sadu do souboru, případně otevírat uloženou sadu.
3. Vypracujte závěrečnou textovou dokumentaci bakalářské práce, zhodnoťte výsledky vaší práce. Do elektronických příloh uveďte mimo jiné i zdrojové kódy vytvořených programů a vlastní aplikaci.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **30–40 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **J. Hammer, J.D. Reid, B. Perkins, Beginning C# 6 Programming with Visual Studio 2015, John Wiley & Sons Inc, United States, 2015, ISBN / EAN: 9781119096689 / 9781119096689.**

[2] **SINAMICS Měnič SINAMICS V20 - Uvedení do provozu Sinamics V20, 03/2015, A5E34559884-002, doustupné z: http://stest1.etnetera.cz/ad/current/index.php?ctxnh=356cc6f440&ctxp=doc_manualy**

[3] **SINAMICS Měnič SINAMICS V20 - Návod k použití, 03/2015, A5E34559884-002, dostupné z: http://stest1.etnetera.cz/ad/current/index.php?ctxnh=356cc6f440&ctxp=doc_manualy**

Vedoucí bakalářské práce: **Ing. David Lindr, Ph.D.**
Ústav mechatroniky a technické informatiky

Konzultant bakalářské práce: **Ing. Martin Diblík, Ph.D.**
Ústav mechatroniky a technické informatiky

Datum zadání bakalářské práce: **10. října 2016**

Termín odevzdání bakalářské práce: **15. května 2017**

Technická univerzita v Liberci

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2016

PROHLÁŠENÍ

Byla jsem seznámena s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědoma povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracovala samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 15.5.2017

Podpis: Kubová

PODĚKOVÁNÍ

Na tomto místě bych ráda poděkovala mému vedoucímu práce panu Ing. Davidu Lindrovi, Ph.D. za odborné vedení, vstřícnost při konzultacích, poskytnutí frekvenčního měniče ke zkoušení a vývoji aplikace a za čas, který mi věnoval v průběhu vypracovávání bakalářské práce.

ABSTRAKT

Tato bakalářská práce pojednává o návrhu a následném vývoji desktopové aplikace určené pro parametrování frekvenčního měniče Siemens SINAMICS V20. Komunikace probíhá po sériové lince RS485 pomocí protokolu USS. Aplikaci jsem vytvořila s využitím programovacího jazyka C# a technologie WPF. Výsledná aplikace poskytuje uživateli jednoduchý způsob čtení a nastavování parametrů frekvenčního měniče, ukládání aktuální sady parametrů do souboru a načítání uložené parametrické sady.

KLÍČOVÁ SLOVA

C#, frekvenční měnič, USS protokol, Siemens SINAMICS V20

ABSTRACT

This bachelor thesis describes the process of design and subsequent development of a desktop application designed to parameterize the frequency converter Siemens SINAMICS V20. Communication takes place via the RS-485 serial line using the USS protocol. I have developed the application using the C# programming language and WPF technology. The resulting application provides the user with a simple way to read and set the frequency converter parameters, store the current set of parameters in a file and load the stored parameter set.

KEYWORDS

C#, frequency converter, USS protocol, Siemens SINAMICS V20

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM CLASS DIAGRAMŮ

SEZNAM ZKRATEK

USS	Universal Serial Interface Protocol
WPF	Windows Presentation Foundation
MVVM	Model-View-ViewModel
DLL	Dynamic-link library
EXE	Executable file
GUI	Graphic User Interface
PC	Personal Computer
PLC	Programmable Logic Controller

TELEGRAM

STX	Start of Text	SP	Parameter Change Report Bit
LGE	Length	PNU	Parameter Number
ADR	Address	IND	Index
PKW	Parameter ID Value	PWE	Parameter Value
PKE	Parameter ID	PZD	Process Data
AK	Task/Response ID	BCC	Block Check Character

1 Úvod

Siemens SINAMICS V20 je řada jednoduchých, nízkonapěťových jednofázových a třífázových frekvenčních měničů navržených pro ovládání rychlosti otáček asynchronních motorů. Tato řada patří mezi cenově dostupnější, k čemuž přispívá i možnost velice snadného nastavení pomocí předpřipravených maker pro základní aplikace jako jsou například odstředivá čerpadla, kompresory, různé typy dopravníků a jiné.

Kromě různých výhod má ale i jednu nevýhodu – neumožňuje připojení k nějakému již existujícímu parametrizačnímu softwaru. V případě jednoduchých použití toto není až tolik omezující díky přednastaveným makrům v měniči, ale jestliže je nutné nastavit měnič detailněji, je potřeba to vše provést skrze vestavěný ovládací panel, což může být zbytečně složitý a zdlouhavý proces. Právě z tohoto důvodu se nabízelo vytvoření vlastního jednoduchého softwaru pro PC určeného výhradně pro parametrizaci frekvenčních měničů přes sériovou linku, který by uživatelům umožnil snadné nastavení bez nutnosti používání ovládacího panelu.

Toto téma jsem si vybrala z toho důvodu, že jej bylo možné uskutečnit s využitím nástrojů .NET a programovacího jazyka C#. Zabývám se vývojem aplikací, takže toto téma mi je blízké a zároveň pro mě představuje možnost získání nových zkušeností, které můžu v budoucí praxi využít. Chtěla bych lépe porozumět problematice komunikace po sériové lince a vyzkoušet si implementaci již existujícího protokolu. Byla bych ráda, kdyby se výsledná aplikace opravdu používala a její realizace by tedy byla přínosem pro lidi, kteří s frekvenčními měniči pracují.

2 CÍLE PRÁCE

Hlavním cílem této práce je vytvoření jednoduše ovladatelné aplikace pro PC, která bude uživateli poskytovat možnost čtení a nastavení parametrů frekvenčního měniče Siemens SINAMICS V20. Je potřeba, aby aplikace byla přehledná, uživatel se v ní rychle zorientoval a samotný proces parametrizace nebyl složitý.

Předpokládá se lišta s ovládacími prvky a tabulkový přehled parametrů s jejich vlastnostmi. Tabulka má dovolovat editaci požadované hodnoty parametru. Uživatel by měl být pomocí aplikace schopný zahájit (případně ukončit) komunikaci s měničem s možností nastavení potřebných parametrů komunikace jako je název portu, přenosová rychlost, počet datových bitů aj., uložit aktuálně zobrazenou parametrovou sadu a načíst již existující sadu parametrů ze souboru. V případě výskytu chyby v komunikaci je potřeba realizovat viditelný indikátor chyby, který na ni uživatele upozorní s informacemi, o jakou chybu se jedná a u kterého parametru nastala. Dále by pro příjemnější ovládání měla aplikace umožňovat řazení parametrů dle jejich čísla a fulltextové vyhledávání v tabulce.

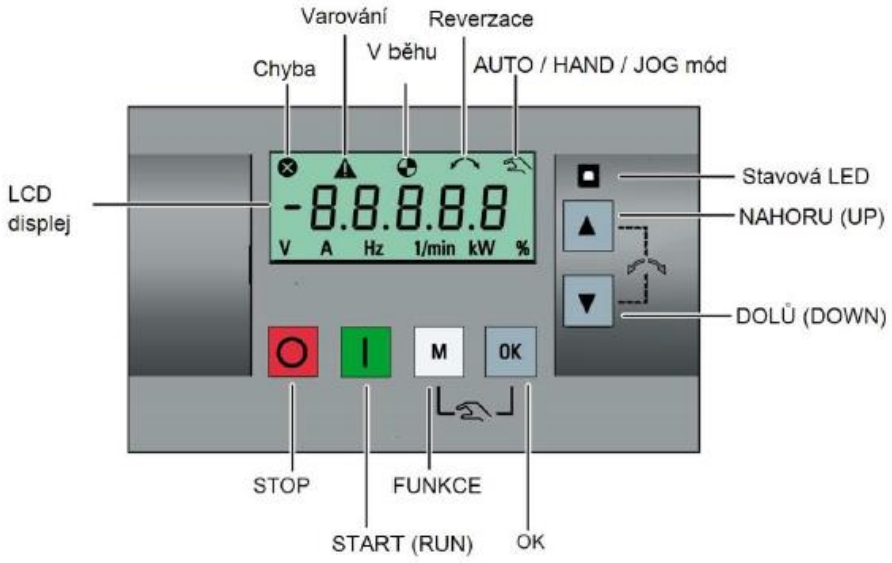
Dalším úkolem této práce, který vyplývá ze zadání, je vytvoření knihovny umožňující komunikaci po sériové lince, která bude využívat komunikačního protokolu USS. Knihovna by neměla být vázána na samotnou aplikaci, měla by být použitelná i v jiných projektech a snadně rozšiřitelná pro implementaci jiných úkonů, než je samotná parametrizace.

3 PODROBNĚJŠÍ PŘEDSTAVENÍ PROBLEMATIKY

Tato kapitola nejprve pojednává o možnostech ovládání měničů. Následně se zabývá popisem již existujícího softwaru určeného k přímému ovládání a parametrizaci frekvenčních měničů Siemens a jejich porovnáním konkrétně s řadou SINAMICS V20. Na závěr popisuje princip samotné komunikace prostřednictvím USS protokolu.

3.1 OVLÁDÁNÍ FREKVENČNÍCH MĚNIČŮ

Frekvenční měniče je možné ovládat více způsoby – ručně s využitím ovládacího panelu, prostřednictvím externích ovládacích prvků pomocí digitálních vstupů a výstupů a analogového vstupu na řídicí svorkovnici měniče anebo dálkově pomocí nadřazeného řídicího systému po standardní průmyslové sběrnici. [3]



Obr. 1 – Ovládací panel frekvenčního měniče Siemens SINAMICS V20; zdroj 49[2]

3.1.1 Ovládací panel

Měnič může být vybaven základním typem integrovaného ovládacího panelu, případně je možné dokoupit externí. Většinou se jedná o alfanumerický LCD display s několika tlačítky. Ovládací panel je možné použít jak pro ovládání motoru, tak i programování měniče. Toto ovládání však není příliš praktické, protože je proces nastavení zdlouhavý

a uživatel nemusí mít vždy přímý přístup k měniči. Schéma ovládacího panelu řady Siemens SINAMICS V20 [2] je vidět na Obr. 1.

3.1.2 Analogové a digitální vstupy a výstupy

Je možné připojit vlastní ovládací prvky na řídicí svorkovnici měniče a tím měnič ovládat. Na digitální vstupy lze připojit vlastní spínače a nastavit je pomocí příslušných parametrů měniče tak, aby po sepnutí vyvolaly požadovanou funkci (např. start/stop motoru, změna směru otáčení, kvitování poruchy aj.). Pomocí analogového vstupu s připojením potenciometru je poté možné nastavovat žádané otáčky.

3.1.3 Nadřazený řídicí systém

K frekvenčnímu měniči je možné se připojit a ovládat jej jiným řídicím systémem jako je PC nebo PLC. Komunikace poté probíhá po standardních průmyslových sběrnicích prostřednictvím různých řídicích protokolů jako jsou PROFINET, PROFIBUS, USS, Modbus, Ethernet IP apod. Siemens SINAMICS V20 využívá konkrétně komunikace po sériové lince RS-485 přes protokoly USS nebo Modbus. [13]

3.2 SOFTWARE PRO OVLÁDÁNÍ FREKVENČNÍCH MĚNIČŮ

Jak již bylo zmíněno v kapitole 3.1.3, frekvenční měniče je možné ovládat pomocí nadřazeného řídicího systému. Konkrétně pro PC již existuje software určený pro přímé ovládání, parametrizování a monitorování měničů značky Siemens, nicméně pro standardní verzi SINAMICS V20 žádná aplikace zatím není. V následujících kapitolách představím tři aplikace, které je možné využít pro řízení ostatních měničů Siemens – STARTER, DriveMonitor a SINAMICS V20 Smart Access.

3.2.1 STARTER

Asi nejrozšířenějším softwarem pro ovládání Siemens měničů je aplikace s názvem STARTER, jež slouží k ovládání, parametrizaci a monitorování frekvenčních měničů řad Siemens SINAMICS, MICROMASTER 4 a SIMATIC ET200 FC. Konkrétně řadu Siemens SINAMICS V20 ale nepodporuje. Tato aplikace je volně ke stažení z webových stránek společnosti Siemens [5].

Aplikace je velmi komplexní, obsahuje mnoho ovládacích prvků a pro nového uživatele může být velice složité se v ní zorientovat. Pro snadnější orientaci je ale k dispozici nápověda obsahující návody a rady k obsluze aplikace.

Ve STARTERu se pracuje s tzv. projekty. Ty slouží ke správě a uložení konfiguračních dat i jiných informací o jednotlivých měničích. Každý projekt se skládá z různých částí – samotná řídicí jednotka měniče, kontrolní modul ovládající ostatní komponenty měniče, zdroj napětí, vstupy a výstupy aj., přičemž lze samostatně konfigurovat parametry týkající se těchto částí. Nastavení je možné provést několika způsoby, a to pomocí:

- průvodce nastavením (wizard), který uživatele provede procesem parametrizace v oknech bez potřeby znalosti významu jednotlivých parametrů (viz Obr. 8),
- grafických schémat, které názorně ukazují propojení jednotlivých komponent a uživatel tak může projít nastavením postupně od jednotky měniče až po připojený motor (viz Obr. 9),
- expert listu, kde jsou tabulkově vypsané jednotlivé parametry a jejich hodnoty (viz Obr. 10).

Velmi příjemné je, že je možné importovat potřebná data ze štítků komponent (program obsahuje databázi měničů, motorů, aj.) a poté je již třeba zvolit jen základní parametry pohonu a motor běží.

3.2.2 DriveMonitor

Pro starší řady měničů značky Siemens je k dispozici program DriveMonitor [4]. Tento program je jednodušší a neposkytuje tolik funkcí jako STARTER. Je určený k parametrizaci, diagnostice a nastavení setpointů měničů řad SIMOVERT, MASTERDRIVES, MICROMASTER 3 a 4 a SIMOREG.

Ovládání probíhá pomocí tabulkových přehledů parametrů (viz Obr. 11) – je možné si zobrazit parametry ovlivňující jen určitou komponentu (obecné parametry, parametry

měniče, motoru, …) nebo list všech parametrů. Je možné pracovat v offline i online módu (připojen k měniči).

3.2.3 SINAMICS V20 Smart Access

Novinkou na trhu od února roku 2017 je aplikace SINAMICS V20 Smart Access [6]. Ta jako jediná umožňuje ovládat, parametrizovat, spravovat a monitorovat měniče Siemens SINAMICS V20, a to přes Wi-Fi (viz Obr. 12). K tomu je ale zapotřebí mít měnič vybaven Smart Access web modulem, jehož cena se pohybuje kolem 3000 Kč, což nemusí být možnost pro někoho, kdo požaduje levné řešení. Zároveň v době, kdy jsem započala vývoj vlastní aplikace, toto řešení ještě nebylo k dispozici.

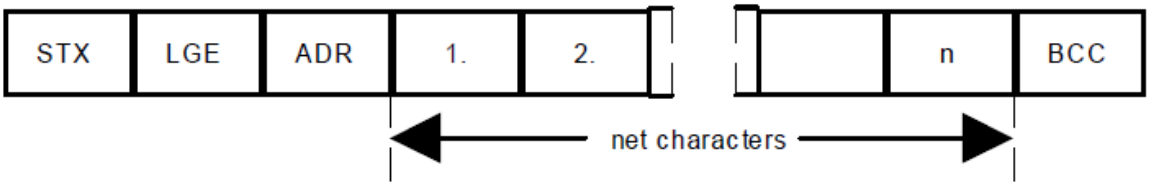
Po instalaci modulu je možné se připojit k Smart Access přes Wi-Fi jakýmkoliv zařízením vybaveným webovým prohlížečem podporujícím HTML 5 (viz Obr. 13). Není tedy potřeba stahovat žádné věci navíc, vše běží jako webová stránka. Kromě ovládání měniče a indikace připojení aplikace podporuje i zálohování dat do mobilu, přehledy chybových stavů a jejich zasílání na podporu přes email.

3.3 USS PROTOKOL

Protokol USS (Universal Serial Interface Protocol) [1] je protokol vyvinutý společností Siemens určený pro komunikaci s frekvenčními měniči od stejné firmy po sériové lince. Využívá principu master-slave. V roli master může být pouze jedno zařízení (single-master systém), a to počítač nebo PLC. Jednotlivé frekvenční měniče jsou v roli slave – těch může být až 31 a odlišeny jsou adresou.

Komunikace probíhá v poloduplexním režimu (half-duplex), kde v daném okamžiku může vysílat data pouze jedno zařízení. Master musí vždy zahájit komunikaci jako první – je možné poslat požadavek pouze jednomu měniči určenému adresou nebo všem jako broadcast. Slave sám o sobě nemůže vysílat, aniž by o to byl požádán.

Data jsou posílána ve formě předepsaných telegramů (viz Obr. 2). Kompletní struktura je znázorněna na Obr. 14.



Obr. 2 – Základní struktura USS telegramu; zdroj [1]

3.3.1 Záhlaví

Záhlaví telegramu se skládá ze tří částí: STX, LGE a ADR. Její velikost jsou 3 byty.

STX

Každý telegram musí začínat částí STX (Start of text), která má vždy velikost 1 byte a obsahuje ASCII znak STX (0x02). Značí počátek telegramu.

LGE

Část LGE (1 byte) se nachází na druhém bytu telegramu a udává jeho velikost. V hodnotě LGE je započítána velikost bloku ADR, celková velikost bloku NET DATA a velikost BCC části. Celková velikost telegramu je o dva byty větší, jelikož není započítána délka STX a LGE. Telegram může být pevné nebo proměnné délky. Maximální velikost telegramu je 256 bytů.

ADR

Blok ADR (1 byte) obsahuje adresu cíleného frekvenčního měniče na 4. až 0. bitu (hodnota 0–31) a dodatečné informace na 7. až 5. bitu. Kromě možnosti zaslání telegramu pouze jednomu měniči poskytuje USS protokol možnost zaslat telegram všem měničům nastavením 5. bitu (Broadcast) na 1. Nastavením 6. bitu (Zrcadlový telegram) na 1 je možné si vyžádat odpověď měniče zrcadlově, což poslouží například k otestování samotné funkčnosti komunikace. Na 7. bitu (Speciální telegram) je možné nastavit telegram jako speciální, nicméně takový telegram musí být nadefinovaný na měniči.

3.3.2 Net data

Samotná data se dělí na dvě hlavní části: PKW oblast a PZD oblast.

PKW

Oblast PKW (Parameter ID value) se užívá pro samotnou parametrizaci měničů. Definuje především, jak má vypadat telegram požadavku pro zápis/čtení parametru a jednotlivé stavy odpovědí měniče. PKW oblast se dělí na část PKE, IND a PWE.

- PKE (Parameter ID) je oblast, kde je definován typ požadavku/odpovědi a parametr, kterého se telegram týká. Jeho velikost je 1 word = 2 byty. PKE má další tři podčásti:
    - AK (Task/Response ID) – Na bitech 15 až 12 je zadán typ požadavku/odpovědi. Jednotlivé hodnoty jsou předepsány ve specifikaci protokolu. Liší se např. pro čtení, zápis hodnot parametrů o různé délce, čtení na indexu v poli, aj.
    - SP (Parameter change report bit) – Tento příznak se nachází na 11. bitu a slave jej nastaví na 1, pokud dojde ke změně hodnoty parametru z jiného zdroje, než který se aktuálně používá pro komunikaci.
    - PNU (Parameter number) – Na 10. až 0. bit se zapisuje číslo parametru, který je potřeba číst/zapsat. Tyto hodnoty se liší pro každý typ měniče a jsou vypsány v manuálu.
- IND (Index) o velikosti 2 bytů je využíván, pokud je potřeba číst/zapisovat v parametrech, které jsou tvořeny polem. Bity 15 až 9 jsou specifické pro jednotlivé měniče. Bity 8 a 9 jsou příznaky pro zápis a čtení textu. Na nižší byte (bity 7 až 0) se zapisuje samotný index v poli. Pokud parametr není tvořen polem, je v bytu zapsána 0.
- PWE (Parameter value) může mít velikost 1 word, 2 wordy nebo velikost proměnnou. V této oblasti už se zasílají samotné hodnoty parametru – v požadavku požadovaná hodnota, v odpovědi hodnota aktuální.

PZD

Druhou část síťových dat tvoří PZD oblast (Process data). Může mít velikost 0 až 16 wordů. Slouží k ovládání motoru. Obsahuje vždy kontrolní (v případě odpovědi stavové) slovo a hlavní setpoint (aktuální data), případně další setpointy a kontrolní

slova. Touto oblastí je např. možné zapínat a vypínat motor a nastavovat rychlost otáček.

3.3.3 Zápatí

Zápatí telegramu obsahuje pouze jednu část BCC, která má velikost 1 byte.

BCC

Na samotném konci telegramu se nachází oblast BCC (Block check character), který slouží k zabezpečení datového toku. Vypočítá se tak, že na začátku je BCC rovno 0 a od začátku telegramu se postupně po jednom bytu provádí bitový XOR (BCC_{new} = BCC_{old} XOR aktuální byte).

4 NÁVRH ŘEŠENÍ

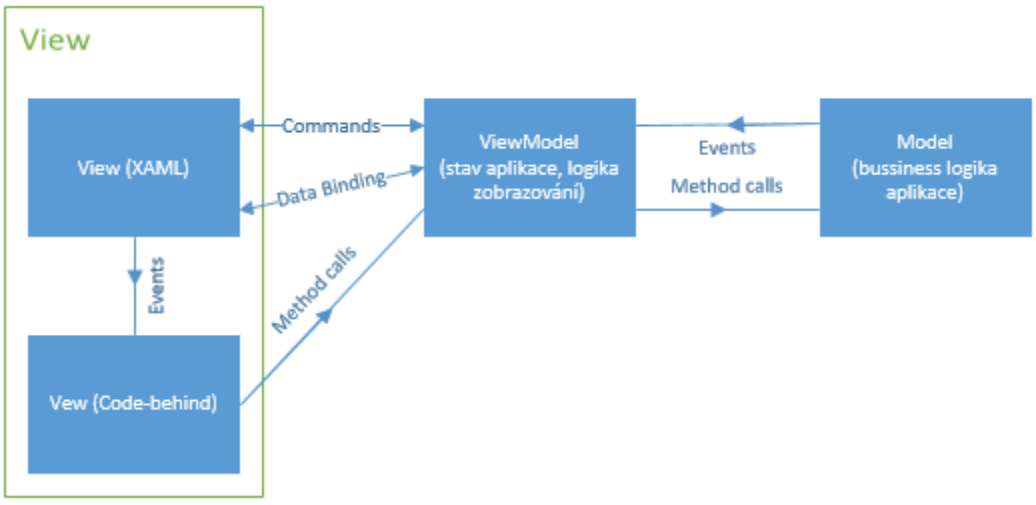
Než jsem započala vlastní realizaci řešení, bylo potřeba si připravit návrh přístupu k vývoji aplikace – rozvrhnout proces zpracování požadavku od uživatele, navrhnout princip komunikace, rozmyslet využití návrhových vzorů a vytvořit hrubý návrh GUI.

4.1 NÁVRHOVÉ VZORY

S vědomím, že aplikaci budu vyvíjet v programovacím jazyce C# jako WPF aplikaci a GUI tedy bude napsané ve značkovacím jazyce XAML, jsem se rozhodla využít některé návrhové vzory.

4.1.1 Model-View-ViewModel

Pro vývoj s využitím technologie XAML se vyloženě nabízí uplatnění návrhového vzoru Model-View-ViewModel (MVVM) [8], jehož princip je znázorněn na Obr. 3. MVVM člení aplikaci tak, aby její části byly více univerzální, rozšiřitelné a přehledné. XAML totiž umožňuje využití bindingu (navázání nějaké vlastnosti grafické komponenty na vlastnost ve view nebo view-modelu) a commadů (napojení akce např. na stisk tlačítka, čímž se vyloučí nutnost použití událostí v behind kódu view).



Obr. 3 – Princip návrhového vzoru MVVM

Tento pattern vychází z návrhového vzoru Model-View-Controller (MVC) a jeho hlavním cílem je oddělit logiku aplikace od uživatelského rozhraní. Jako vrstva Model slouží sada tříd, které popisují bussiness logiku aplikace a samotná data. Vrstvu View představuje uživatelské rozhraní – jejím jediným úkolem je zobrazovat data. View-model má za úkol propojovat model a view a uchovávat aktuální stav aplikace, dále poskytuje potřebné property, metody a commandy přiřazenému view. View-model se vyznačuje tím, že implementuje rozhraní `INotifyPropertyChanged`, díky čemuž je poté zajištěna propagace změn vlastností do view.

4.1.2 Singleton

Návrhový vzor Singleton [9] slouží k zajištění, že od jedné třídy bude možné vytvořit pouze jednu instanci, ke které bude poskytnut globální přístup. Tento pattern se hodí v případě, pokud je potřeba mít jednu instanci něčeho (např. připojení k databázi, jeden uživatel aplikace, aj.), ke které chceme přistupovat ve více místech aplikace. V mém případě Singleton použiji pro správu přístupu k sériovému portu.

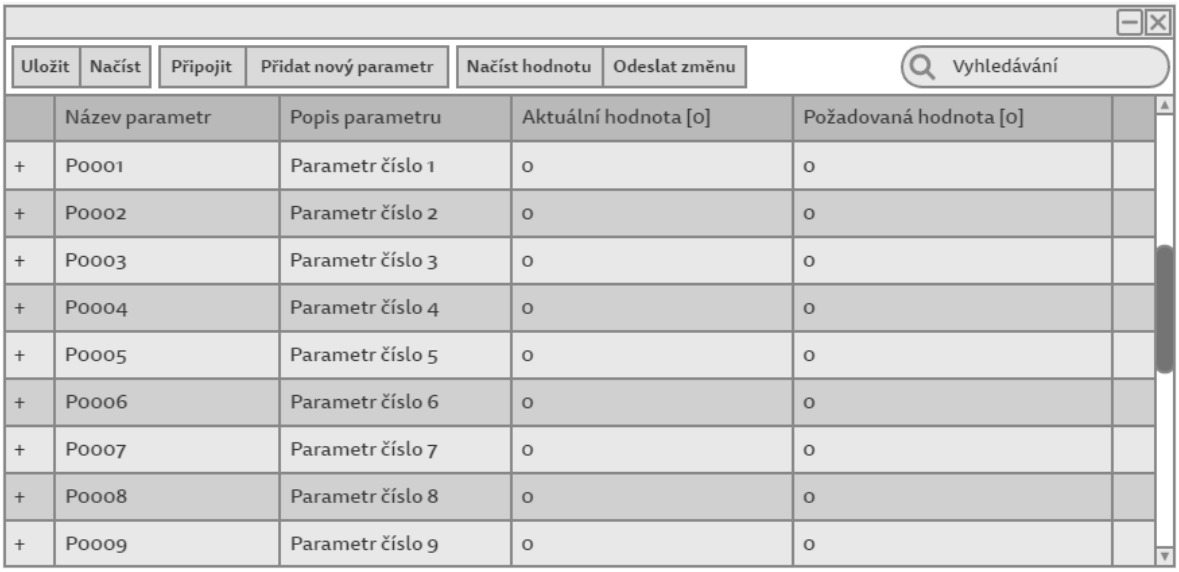
Klíčovými body v implementaci je privátní konstruktor oné třídy a statická read-only vlastnost poskytující její instanci. Třída si v případě vyžádání instance vytvoří svou instanci sama a uloží ji do již zmíněné statické vlastnosti. Uživatel tak nemá možnost vytvořit další instance.

4.2 Uživatelské rozhraní

Inspiraci pro rozložení grafických prvků v GUI jsem si vzala v aplikaci STARTER (viz kapitolu 3.2.1), konkrétně v expert listech. Uživatelské rozhraní bude tvořeno dvěma hlavními částmi – lišta s ovládacími prvky a přehled parametrů, jako je ukázáno na Obr. 4.

Pro přehled parametrů jsem se rozhodla využít tabulku s možností vertikálního scrollování. Ta bude představovat klíčový prvek uživatelského rozhraní. Řádky budou obsahovat jednotlivé parametry a sloupce jejich vlastnosti. Jelikož mají některé parametry více indexů, ale nultý bývá nejpoužívanější, je vhodné v hlavním řádku parametru zobrazit hodnoty právě pro nultý index a dát uživateli možnost parametr

rozkliknout pro získání přehledu o ostatních indexech. Tabulku bude také možné řadit podle určitých sloupců.



	Název parametr	Popis parametru	Aktuální hodnota [0]	Požadovaná hodnota [0]	
+	P0001	Parametr číslo 1	0	0	
+	P0002	Parametr číslo 2	0	0	
+	P0003	Parametr číslo 3	0	0	
+	P0004	Parametr číslo 4	0	0	
+	P0005	Parametr číslo 5	0	0	
+	P0006	Parametr číslo 6	0	0	
+	P0007	Parametr číslo 7	0	0	
+	P0008	Parametr číslo 8	0	0	
+	P0009	Parametr číslo 9	0	0	

Obr. 4 – Návrh uživatelského rozhraní aplikace vytvořený pomocí aplikace Mockingbird [11]

Lištu s ovládacími prvky umístím nahou nad tabulku a bude obsahovat tlačítka, případně ikony, pro akce, které uživatel může provést – načtení a uložení parametrové sady, připojení a odpojení od měniče, přidání nového parametru do sady, načtení aktuální hodnoty vybraného parametru, potvrzení změny hodnoty vybraného parametru, dle potřeby i načtení a potvrzení změn všech parametrů. Lišta bude obsahovat i vyhledávací pole, které bude zarovnáno doprava jako bývá zvykem v jiných desktopových i webových aplikacích.

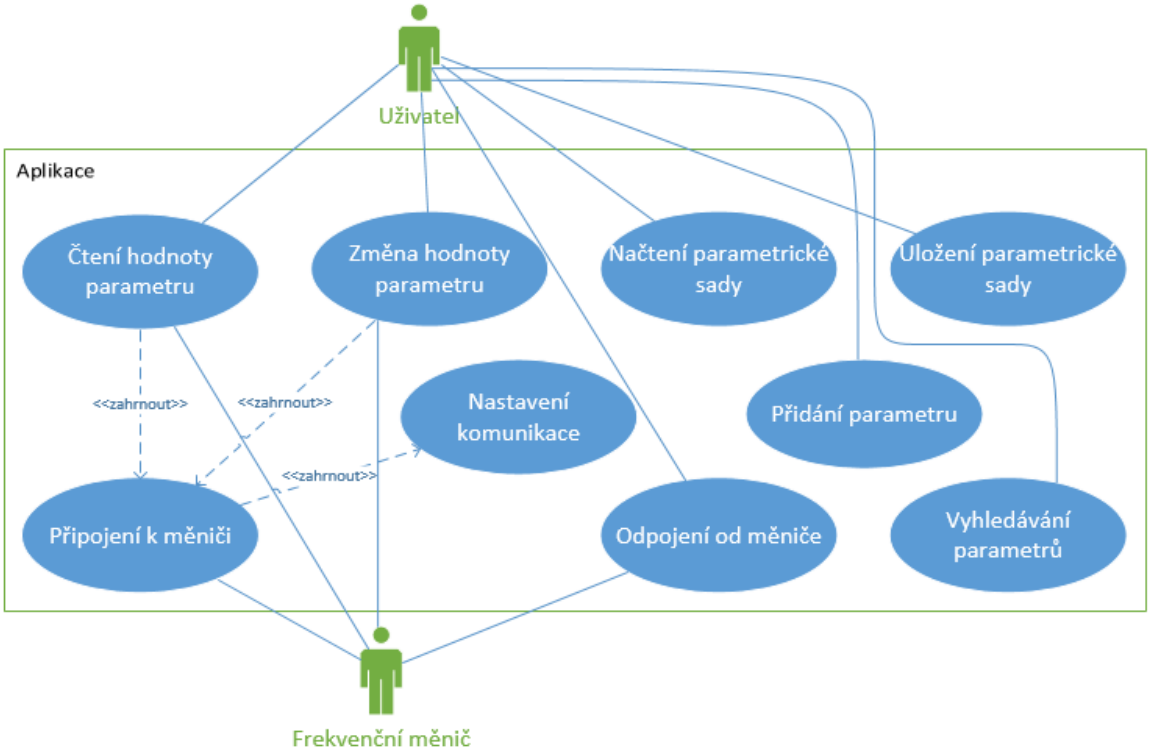
4.3 Proces zpracování požadavků

Poté, co uživatel spustí aplikaci, uvidí hlavní okno s tabulkou parametrů. Na výběr bude mít několik akcí, které může provést (viz Obr. 5).

- Připojit se/odpojit se od měniče
- Změnit a číst hodnoty parametrů
- Uložit a nahrát parametrovou sadu

- Přidat nový parametr
- Nastavit komunikaci
- Vyhledávat v parametrech

Akce pro změnu a čtení hodnoty parametru je podmíněna připojením se k měniči. K měniči je možné se připojit až po nastavení parametrů komunikace. Uložit a nahrát parametrovou sadu, přidat nový parametr a vyhledávat v parametrech může uživatel kdykoliv.



Obr. 5 – Use case diagram aplikace

4.4 Komunikace s měničem

Před realizací knihovny pro komunikaci s měničem bylo třeba si rozmyslet, jaké prostředky pro komunikaci použiji.

4.4.1 Sériový port

Pro komunikaci po sériové lince poskytuje .NET framework třídu `SerialPort` [10], která se nachází v namespace `System.IO.Ports`. Tato třída umožňuje otevřít sériový

port s požadovanou konfigurací (název portu, přenosová rychlost, počet datových bitů, handshake, parita atd.).

`SerialPort` poskytuje různé metody pro čtení a odesílání dat. V prvotním návrhu jsem předpokládala využití metody `Write(byte[] buffer, int offset, int count)` pro zápis dat a událost `SerialDataReceivedEventHandler DataReceived` pro čtení příchozích dat. Tato metoda se ale neosvědčila, protože bylo složitější přiřazovat odpovědi k požadavkům a událost neupozorňovala na všechny příchozí zprávy, pokud jich bylo více v kratším časovém intervalu. Proto jsem nakonec místo události zvolila použití metody `Read(byte[] buffer, int offset, int count)`, jejíž volání obalím do jiné metody, která pošle požadavek měniči a následně počká na odpověď. Pokud odpověď nepřijde do určitého časového intervalu, vyhodí chybu.

4.4.2 USS telegram

Mým úkolem je naprogramovat aplikaci, která je prozatím určena pouze k parametrizaci měniče. Část telegramu PZD není tedy potřeba, takže jsem se rozhodla, že ji do implementace nezařadím. Třídy realizující USS komunikaci ale napíši tak, aby bylo možné je v budoucnu případně jednoduše rozšířit o PZD oblast pro ovládání motoru a cyklickou komunikaci, která je pro to potřeba.

Pro usnadnění implementace protokolu jsem si určila pevnou délku telegramu, kde velikost PWE = 4 byty a velikost PZD oblasti = 0 bytů. Vyhnu se tak problémům se složitějším rozpoznáváním telegramu.

Jelikož je aplikace určená pro komunikaci pouze s jedním měničem, budu používat měnič vždy na adrese 0. Přenosovou rychlost si bude moci uživatel nastavit v nastavení komunikace v aplikaci – ta musí souhlasit s rychlostí nastavenou v měniči.

5 REALIZACE ŘEŠENÍ

Výsledkem mé bakalářské práce je vytvoření aplikace s názvem SiemensSinamicsV20. Aplikaci jsem napsala ve vývojovém prostředí Microsoft Visual Studio 2017 pomocí programovacího jazyka C# 6.0 a .NET Framework 4.6.2.

Ke komunikaci jsem využila měnič Siemens SINAMICS V20 3x230V 11A 2,2kW bez filtru, který jsem přes sériovou linku RS-485 připojila do USB portu počítače pomocí převodníku.

Celé solution SiemensSinamicsV20 sestává ze dvou projektů – WPF aplikace `SiemensSinamicsV20` a DLL knihovny `UssCommunication`. Řešení jsem takto rozdělila z důvodů jednodušší správy kódu, rozšiřitelnosti a případnému umožnění využití kódu pro komunikaci pomocí USS protokolu v jiných projektech.

5.1 USSCOMMUNICATION

Projekt `UssCommunication` zprostředkovává komunikaci mezi aplikací a frekvenčním měničem pomocí USS protokolu po sériové lince. Projekt se skládá ze tří hlavních managerů, několika modelů dohromady představujících USS telegram a pár dalších pomocných tříd. V dalších kapitolách jsou třídy rozebrané podrobně.

5.1.1 Models

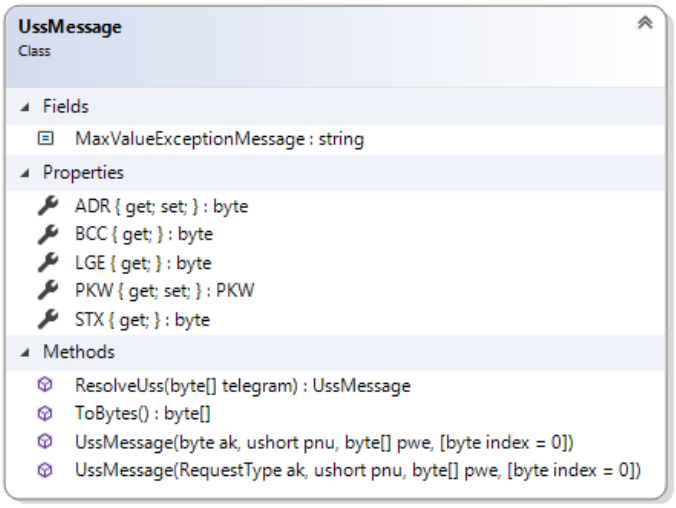
Pro usnadnění práce s USS telegramem a zprostředkování komunikace mezi aplikací a knihovnou jsem vytvořila sadu modelů.

UssMessage

Třída `UssMessage` představuje jednotku přenosu USS protokolu. Její strukturu je možné vidět na Class diagram 1.

`UssMessage` se skládá z pěti vlastností, jež každá z nich odpovídá stejnojmenným částem USS telegramu. `STX`, `LGE`, `ADR` a `BCC` jsou realizovány datovým typem `byte`, zatímco pro `PKW` jsem pro větší přehlednost vytvořila vlastní model. Pro sestavení

telegramu obsahuje `UssMessage` a každá složitější třída v ní použitá metodu `ToBytes()`, která se postará o převedení hodnot na pole bytů ve správném pořadí.



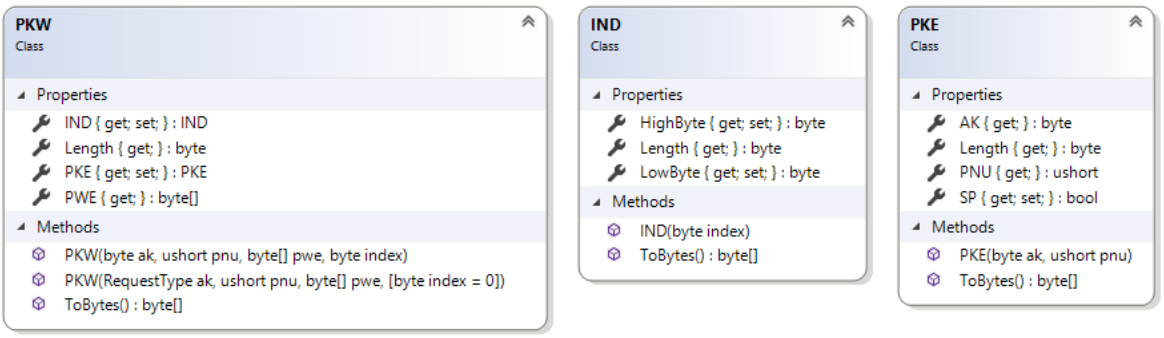
Class diagram 1 – Třída UssMessage

Při realizaci zasílání telegramů na sériový port jsem se setkala s několika zádrhely – měnič buď neposílal žádnou odpověď nebo odpovídal s chybou limitu hodnot. První problém byl s kontrolním součtem, který nebylo těžké vyřešit. Druhý zádrhel se týkal pořadí bytů ve wordu, kde jsem v metodě `ToBytes()` sice sestavila telegram ve správném pořadí, ale byty v `PWE` části byly prohozené.

PKW

Třída `PKW` slouží jako PKW část USS telegramu – obsahuje tedy vlastnosti jako číslo požadavku/odpovědi, číslo parametru, samotná data a příznak SP. Struktura je znázorněna na Class diagram 2.

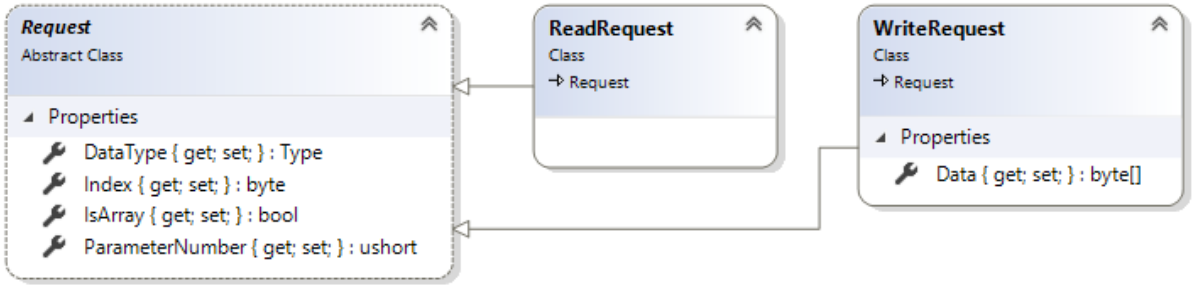
Dělí se na `IND`, `PKE` a `PWE`, kde `IND` je samotný model obsahující bytové vlastnosti `HighByte` a `LowByte` a `PKE` obsahuje již celočíselné vlastnosti `AK`, `PNU` a vlastnost `SP` typu `bool`.



Class diagram 2 – Třídy PKW, IND a PKE

Request

Dalším vytvořeným modelem je abstraktní třída `Request`. Ta slouží pro uchování dat o požadavku pro měnič – datový typ parametru, číslo parametru, zda se jedná o pole a případně index. Tvoří bázovou třídu pro další dva modely – `ReadRequest` a `WriteRequest` (viz Class diagram 3).



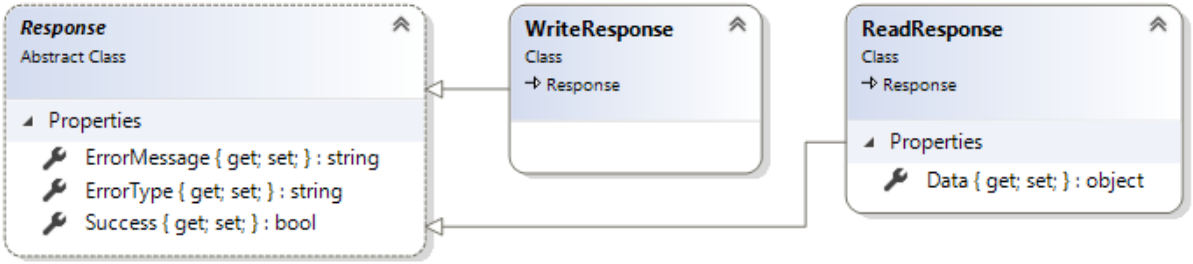
Class diagram 3 – Abstraktní třída Request a její zděděné třídy ReadRequest a WriteRequest

`ReadRequest` třída je, jak už z názvu vyplývá, určená pro požadavek na čtení hodnoty parametru a neobsahuje další vlastnosti kromě zděděných – slouží tedy spíše pro případné rozšiřování funkcionality aplikace (čtení textů, přidání obsluhy PZD části telegramu, aj.).

Třída `WriteRequest` představuje požadavek na změnu hodnoty parametru a obsahuje navíc pole bytů `Data`, kde se přenáší hodnota, kterou je potřeba do parametru nahrát.

Response

Třída `Response` je stejně jako třída `Request` abstraktní třídou a je určená k přenosu odpovědi směrem z měniče do aplikace. Obsahuje údaje o úspěšnosti splnění požadavku a případně typ a zprávu vrácené chyby. Tvoří bázovou třídu pro třídy `ReadResponse` a `WriteResponse`. Struktura tříd a jejich vztahy jsou zobrazené na Class diagram 4.



Class diagram 4 – Abstraktní třída Response a její zděděné třídy ReadResponse a WriteResponse

Model `ReadResponse` je určený k přenosu odpovědi na požadavek ke čtení hodnoty parametru – obsahuje navíc vlastnost `Data`, kde jsou přenesena data vrácená měničem.

Třída `WriteResponse` je opět pouze obálkou, neobsahuje další vlastnosti, ale umožňuje jednodušší implementaci případných rozšíření.

SerialPortSettings

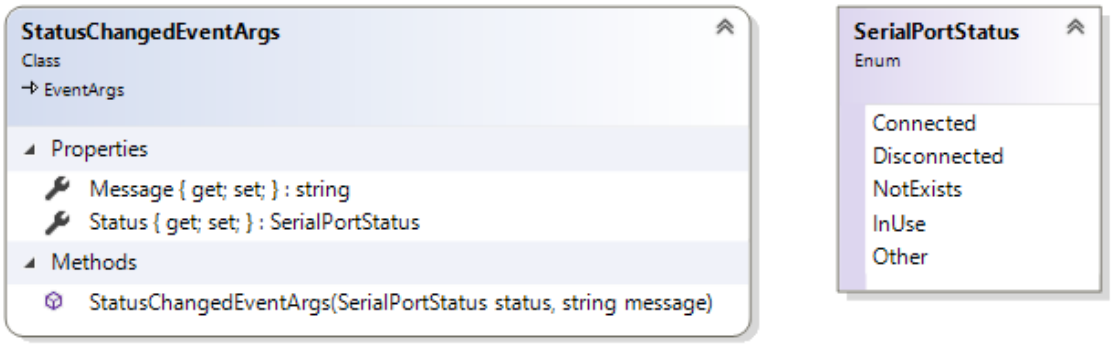
Abych nemusela napříč všemi třídami od uživatelského rozhraní po třídu komunikující se sériovým portem přenášet více proměnných pro nastavení sériového portu (např. název portu, přenosovou rychlost, aj.), vytvořila jsem model `SerialPortSettings` obalující tyto proměnné a tím pádem přenáším pouze jednu referenci na instanci třídy. Class diagram této třídy je ukázán na obrázku.

Kromě nastavitelných parametrů komunikace jsem do třídy přidala ještě časové konstanty v milisekundách `ReadTimeout` = 500 a `WriteTimeout` = 500, které jsou využity k nastavení timeoutů komunikace po sériové lince.

Class diagram 5 – Třída SerialPortSettings

SerialPortStatus a StatusChangedEventArgs

Pokud nastane nějaká změna ve stavu sériového portu – aplikace se k němu připojila, odpojila, nastala chyba, aplikace se pokusila připojit k neexistujícímu portu nebo k portu, který je již používán, tak třída obstarávající komunikaci se sériovým portem (str. 33) vyvolá událost. Abych mohla poslat potřebné údaje o stavu, který nastal, vytvořila jsem pro tuto událost vlastní argument `StatusChangedEventArgs`, který obsahuje vlastnost `Status` a `Message`.



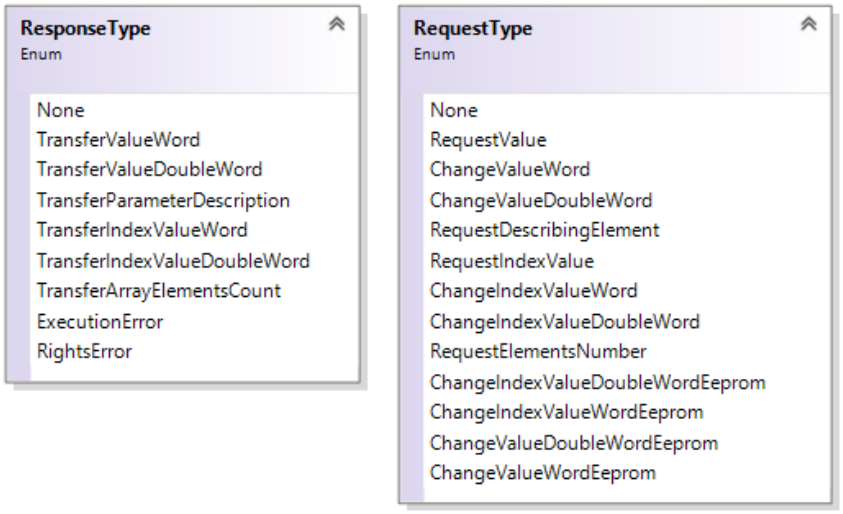
Class diagram 6 – Třída StatusChangedEventArgs a enum SerialPortStatus

Vlastnost `Message` je textový řetězec, do kterého předám zprávu chyby nebo název sériového portu, kterého se stav týká. Vlastnost `Status` je typu `SerialPortStatus`,

což je výčtový typ obsahující výše zmíněné stavy. Struktura tříd je vidět na Class diagram 6.

RequestType a ResponseType

V USS telegramu se nachází část AK, která obsahuje číslo požadavku/odpovědi měniče. Pro jednodušší práci jsem vytvořila dva výčtové typy: `RequestType` a `ResponseType` (viz Class diagram 7).



Class diagram 7 – Enumy ResponseType a RequestType

Oba enumy obsahují všechna ID požadavků a odpovědí a zároveň má každá hodnota atribut `Display[Name = "Název"]`, jež uvádí celý význam tohoto ID. To, například u chybových hlášení, napomůže uživateli identifikovat chybu, aniž by její ID musel hledat v manuálu.

ErrorMessage

Dalším vytvořeným výčtovým typem je `ErrorMessage`. Pokud dojde k chybě, že daný požadavek nemůže být zpracován (AK = 7), obsahuje PWE část číslo konkrétní chyby, která nastala. Chybou může být např. chybné číslo parametrů, zadaná hodnota mimo rozsah parametru, pokus o změnu parametru, který je pouze ke čtení, a mnoho dalších. Právě tyto chybové hlášky je třeba uživateli zobrazit, takže stejně jako u `RequestType`

a `ResponseType` jsou jednotlivé hodnoty označeny atributem `Display` obsahující celý popis chyby.

5.1.2 Managers

Pro obsluhu sériového portu, komunikaci s měničem a zpracování požadavků od uživatele jsem napsala několik managerů, kteří mají vše na starost: `SerialPortManager`, `CommunicationManager` a `UssManager`. Snažila jsem se komunikaci rozdělit do pomyslných vrstev tak, aby každý manager pracoval jen s daty, které se ho týkají – `SerialPortManager` pracuje s poli bytů, `CommunicationManager` s komplexní `UssMessage` a `UssManager` s modely `Request` a `Response`.

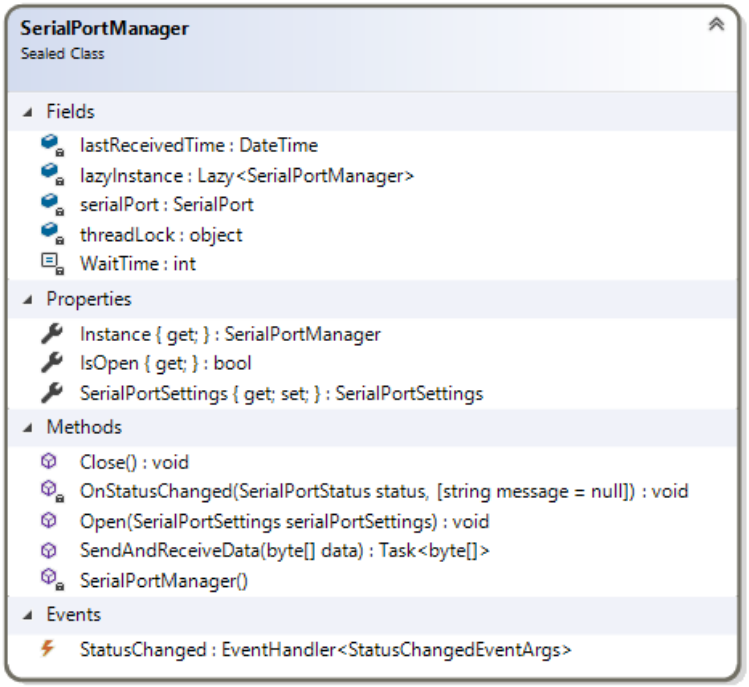
SerialPortManager

`SerialPortManager` (viz Class diagram 8) je `sealed` třída navržená pomocí návrhové vzoru Singleton, která slouží pro samotnou obsluhu sériového portu. Třídu jsem navrhla zapečetěnou a jako Singleton z toho důvodu, že chci v aplikaci mít právě jednu instanci této třídy a komunikovat s právě jedním sériovým portem. Tato třída má privátní konstruktor, aby z ní nešly udělat další instance – jedna instance je vytvořená pomocí lazy inicializace a poskytovaná přes vlastnost `Instance`.

Následně obsahuje metodu `Open`, které se předá instance `SerialPortSettings`. Dle jejích vlastností je nastaven a následně otevřen sériový port. Pokud otevření portu proběhne úspěšně, je zavolána událost `StatusChanged` s parametrem `SerialPortStatus.Connected` a zprávou daného nastavení. Událost je volána i v případě, že dojde k chybě, s parametry upřesňující nastalou chybu. Dále pak existuje metoda `Close`, která uzavře spojení s portem a opět vyšle událost o odpojení.

Poslední metodou je metoda `SendAndReceiveData`. Tato metoda přijímá jako vstupní parametr pole bytů, které mají být poslány na port (v mém případě pole bytů je již sestavený USS telegram), a výstupem je jiné pole bytů – odpověď měniče. Metoda je asynchronní, protože po odeslání požadavku čeká na odpověď od měniče. Chci tedy, aby čekání na odpověď běželo v jiném vlákně než v hlavním a uživateli se tak aplikace nejevila jako zamrzlá. V metodě se nejdříve zjistí, zda je sériový port stále otevřen

a dále proběhne kontrola, zda již uběhla potřebná doba od přijetí poslední zprávy od měniče, pokud ne, ještě se potřebnou dobu počká. Následně na sériový port zapíše potřebná data a opět počká potřebnou dobu na odpověď. Poté ji přečte a vrátí ji.



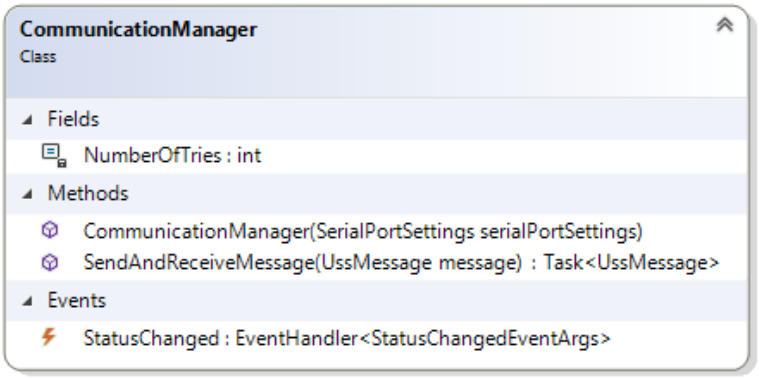
Class diagram 8 – Třída pro komunikaci přes sériový port SerialPortManager

CommunicationManager

Třída `CommunicationManager` (znázorněna na Class diagram 9) má na starost zajištění komunikace a kontroly přijatých dat. V jejím konstruktoru se otevře `SerialPortManager`, pokud ještě není otevřený, a zaregistruje se jeho událost `StatusChanged`, kterou pak třída přeposílá opět o úroveň výše.

Obsahuje asynchronní metodu `SendAndReceiveMessage`, kde jako vstupní parametr přijímá `UssMessage` k odeslání na měnič a výstupem je opět `UssMessage` představující odpověď měniče. Uvnitř metody se `UssMessage` převede na pole bytů a předá se `SerialPortManageru`. Následně počká na odpověď, kterou rozkóduje zpět do `UssMessage`. Poté se zkontroluje `BCC` část telegramu, zda předpokládaná hodnota odpovídá přijaté hodnotě. Pokud ano, proběhne další kontrola na číslo parametru. Jak

je zmíněno v [7], existuje možnost, že měnič může občas odpovědět na jiný telegram, proto zjišťuji, zda přijatá odpověď je právě ta odpověď, kterou očekávám.



Class diagram 9 – Třída CommunicationManager

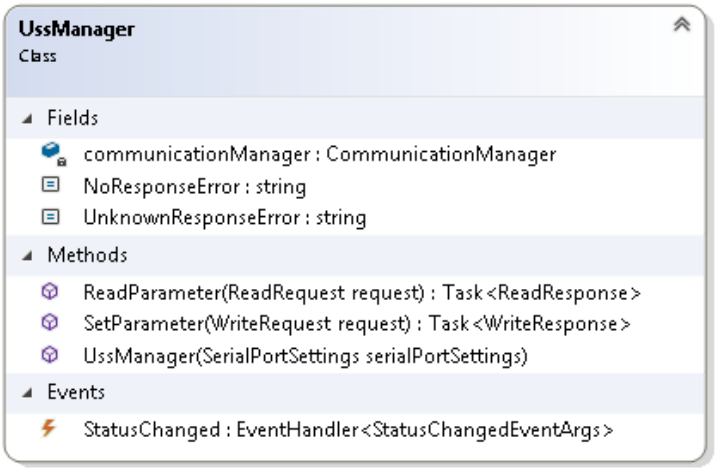
Pokud vše proběhne bez chyby, vrátí metoda přijatou `UssMessage`. Jestliže někde v průběhu nastane chyba, metoda opakuje odeslání požadavku tolikrát, kolikrát je nastavena celočíselná konstanta `NumberOfTries`, což jsou 3. Pokud ani poté nepřijde správný a bezchybný telegram, metoda vrací `null`.

UssManager

Třída `UssManager` (viz Class diagram 10) slouží pro zpracování samotných požadavků na čtení nebo modifikaci hodnoty parametru v měniči. Do konstruktoru se předá požadované `SerialPortSettings`, inicializuje se privátní instance `CommunicationManageru` a zaregistruje se událost `StatusChanged`.

`UssManager` obsahuje dvě metody – `ReadParameter` a `SetParameter`. Jak už názvy napovídají, slouží ke čtení a nastavení hodnoty parametru.

Metodě `ReadParameter` se předává jako parametr `ReadRequest`, ze kterého se vytvoří `UssMessage`, jež se následně předá `CommunicationManageru`. Tím je získána odpověď, dle které je inicializována `ReadResponse`. Stejným způsobem funguje metoda `SetParameter` akorát s typy `WriteRequest` a `WriteResponse`.



Class diagram 10 – Třída pro zpracování požadavků uživatele UssManager

5.2 SiemensSinamicsV20

Druhý z projektů je WPF projekt `SiemensSinamicsV20`. Projekt obsahuje definice uživatelského rozhraní a třídy, jež zprostředkovávají komunikaci mezi GUI a knihovnou `UssCommunication`. Projekt využívá návrhového vzoru MVVM.

5.2.1 Views

Složka Views obsahuje tři soubory typu .xaml, které definují grafické rozhraní oken aplikace:

- `MainWindow.xaml`,
- `SettingsWindow.xaml`,
- `AddParameterWindow.xaml`.

MainWindow

`MainWindow` tvoří základní okno aplikace, které uživatel využívá nejčastěji. Skládá se z jednoho `Toolbaru` a jedné editovatelné `DataGrid`. Jeho vzhled je vyobrazen na Obr. 15.

`Toolbar` obsahuje hlavní ovládací prvky aplikace jako jsou ikony pro uložení a načtení parametrové sady, ikonové tlačítko pro připojení a odpojení aplikace od měniče,

tlačítka pro přidání nového parametru, odeslání změny hodnoty parametru a načtení aktuální hodnoty parametru a vyhledávací pole.

`DataGrid` jako řádky zobrazuje jednotlivé parametry a jako sloupce jeho vlastnosti – číslo parametru, popis parametru a aktuální a požadovanou hodnotu nultého indexu. Každý řádek obsahuje `Expander`, který umožňuje otevřít seznam indexů vybraného parametru. Tento seznam je realizován opět pomocí vnořené `DataGrid`.

Kromě samotných parametrů a jejich vlastností zobrazuje GUI i informace o nastalých chybách, ať už validačních (špatný datový typ hodnoty), tak též chyb při komunikaci s měničem – žádná odpověď měniče, vypršení timeoutu, ale i hlášky přijaté přímo z měniče, např. překročení limitu hodnoty, nedostatečná oprávnění pro změnu parametru, aj. Při výskytu validační chyby se rámeček buňky, do které uživatel zapisuje, přebarví na červeno, chybná hodnota se vymaže a uživateli se zobrazí `ToolTip` s danou hláškou (například „Vstupem musí být číslo typu UInt16.“). Pokud se naskytne chyba komunikace, chyba je značena stejným způsobem, ale v buňce aktuální hodnoty, a navíc je i samotná hodnota vypsána červeně. V obou případech se také zobrazí červený vykřičník vedle `Expanderu` řádku, aby bylo uživateli dáno jasně najevo, že v nějakém indexu parametru došlo k chybě, i přesto, že zrovna nemá indexy zobrazené.

Pomocí hlavičky `DataGrid` je možné řadit parametry dle jejich čísla. K vyhledávání parametrů dle čísla nebo popisu slouží již výše zmíněné vyhledávací pole v `ToolBaru`.

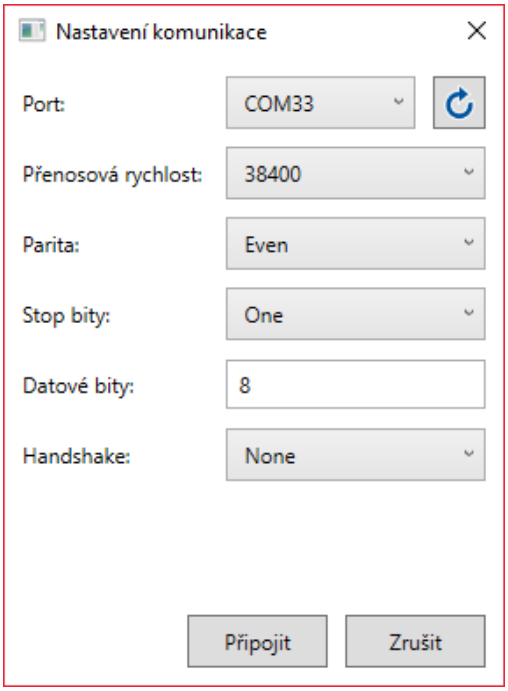
SettingsWindow

Pokud uživatel v hlavním okně klikne na tlačítko Připojit, zobrazí se mu dialogové `SettingsWindow` (viz Obr. 6). Toto okno mu umožňuje před připojením nastavit parametry komunikace přes sériový port – název portu, přenosovou rychlost, paritu, počet stop bitů, počet datových bitů a handshake.

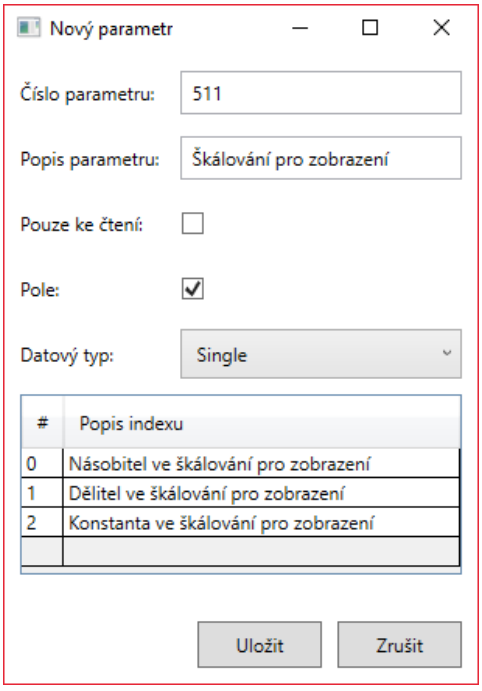
Veškeré nastavení komunikace se ukládá do nastavení aplikace, takže při znovuspuštění aplikace se zobrazí naposledy zvolená kombinace hodnot.

AddParameterWindow

Aby uživatel mohl přidat parametr do vlastní sady parametrů, vytvořila jsem okno `AddParameterWindow`, které je ukázáno na Obr. 7. To umožňuje definovat vlastnosti parametru dle manuálu (číslo parametru, popis parametru, zda je pouze ke čtení a zda je to pole, jeho datový typ a definice jeho indexů s číslem indexu a významem). Okno se otevře zvolením Nový parametr v `ToolBaru` hlavního okna. Po stisknutí tlačítka Uložit se okno zavře a parametr je přidán do seznamu parametrů v `DataGrid` v `MainWindow`.



Obr. 6 – Okno pro nastavení komunikace



Obr. 7 – Okno pro přidání nového parametru

5.2.2 Models

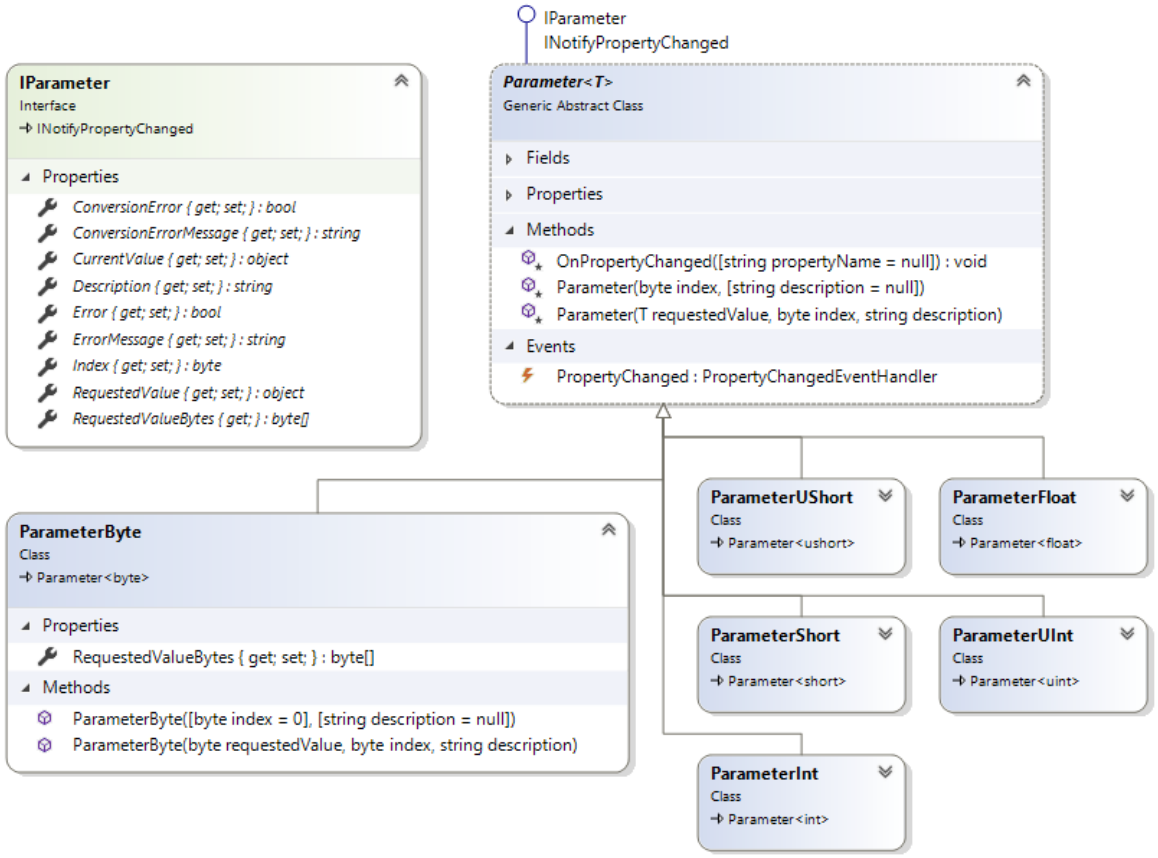
Ve složce Models jsou obsažené modely potřebné pro uchovávání dat a funkci views. Kromě tříd představující jednotlivé parametry se zde nachází i třídy `Command` a `ParameterCommand` pro napojení tlačítek k view-modelům.

Parameter

Jelikož bylo potřeba mít v aplikaci seznam všech parametrů všech datových typů pohromadě, bylo nutné vymyslet nějaký způsob, jak to provést. V aplikaci jeden `Parameter` představuje jeden index opravdového parametru v měniči.

V původním návrhu byla třída `Parameter` negenerická a datový typ byl uložen pouze jako její vlastnost. Tam ale nastal problém při konvertování požadované hodnoty na pole bytů pro telegram – kód obsahoval mnoho zbytečných ifů a přestože třída byla funkční, tak případné rozšiřování by bylo pracné a složité.

Nakonec hierarchii jednoho indexu parametru tvoří rozhraní `IParameter`, abstraktní generická třída `Parameter` a poté konkrétní implementace indexů různých datových typů (`ParameterByte`, `ParameterFloat`, `ParameterInt`, `ParameterShort`, `ParameterUInt`, `ParameterUShort`) – viz Class diagram 11.



Class diagram 11 – Rozhraní IParameter, abstraktní třída Parameter<T> a zděděné třídy ParameterByte, ParameterShort, ParameterUshort, ParameterInt, ParameterUInt a ParameterFloat

List nemůže obsahovat jako generický parametr datový typ s předem neznámým generickým parametrem (např. `List<Parameter<T>>`). Proto jsem vytvořila rozhraní `IParameter`, které zpřístupňuje všechny vlastnosti. Od něj následně dědí abstraktní třída `Parameter<T>`, která všechny vlastnosti implementuje, zároveň obsluhuje `PropertyChanged` událost a také slouží k serializaci. Ale z toho důvodu, že je nezbytné předem vědět, jakého datového typu parametr je, aby bylo možné jeho hodnotu převést na pole bytů, jsem naprogramovala další třídy konkrétně pro používané datové typy, které od `Parameter<T>` dědí a přepisují vlastnost `RequestedValueBytes` tak, aby vracela pole bytů správně naplněné požadovanou hodnotou.

Třída `Parameter` v sobě dále uchovává číslo a popis indexu, `bool Error` a `string ErrorMessage` pro chyby komunikace nebo chyby z měniče, `ConversionError` a `ConversionErrorMessage` pro validační chyby, aktuální hodnotu a `object` property `RequestedValue` s požadovanou hodnotou, která zároveň obaluje logiku konverze na požadovaný datový typ.

ParameterSet

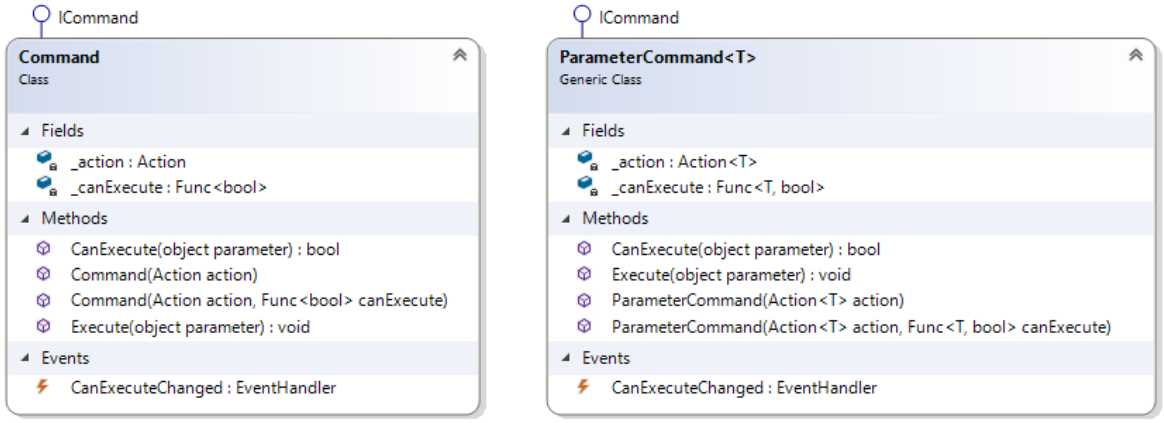
Protože `Parameter` představuje pouze jeden index parametru, existuje ještě třída představující celý parametr a obsahující seznam indexů – `ParameterSet`. Ze stejného důvodu jako u třídy `Parameter` – potřeba mít seznam všech parametrů všech datových typů – jsem opět vytvořila interface `IParameterSet` a generickou třídu, která od něj dědí, `ParameterSet<T>`, která již ale není abstraktní, jako tomu bylo u třídy `Parameter<T>`, jelikož nemá žádné metody ani vlastnosti, které by potřebovaly vědět předem, s jakým datovým typem pracují.

`ParameterSet<T>` má vlastnosti parametru měniče (číslo a popis parametru, zda je pole, zda je pouze ke čtení, datový typ a seznam jeho indexů) a navíc kvůli bindingu nastalých errorů `bool` vlastnost `Error`, která vrací true, pokud u některého z indexů došlo k chybě.

Command a ParameterCommand

Komponenta `Button` ve WPF umožňuje pro použití MVVM návrhového vzoru binding property `Command` a `CommandParameter` (viz Class diagram 12). Aby nebylo potřeba využívat behind kód view, tak se ve view-modelu vytvoří instance třídy `Command` s navázanou metodou, která se vykoná po kliku na tlačítko, jež se nabinduje právě na vlasnost `Command` daného tlačítka.

Pro tento účel jsem vytvořila dvě implementace rozhraní `ICommand` – `Command` a `ParameterCommand` (sloužící pro využití parametru). Commandy také umožňují přímo nastavovat vlastnost tlačítka `IsEnabled` podle nějaké podmínky (např. vyplnění všech povinných polí).



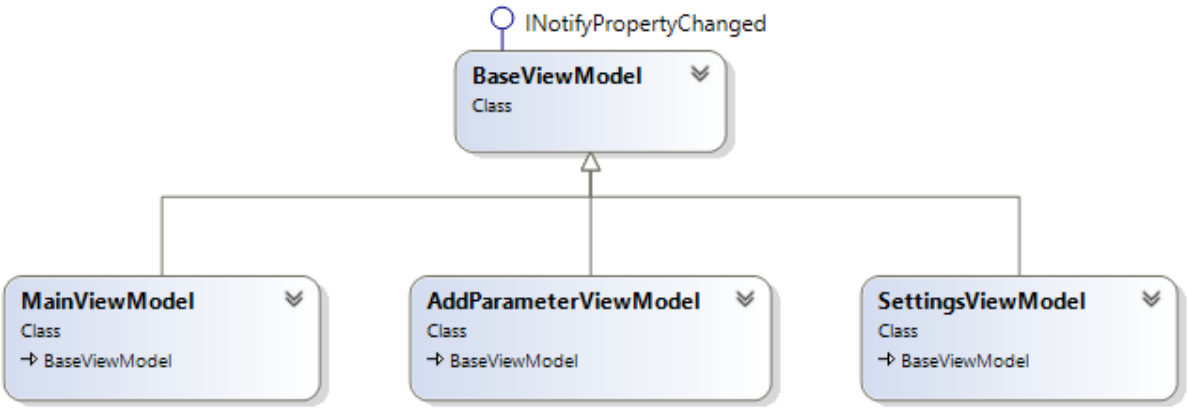
Class diagram 12 – Třídy Command a ParameterCommand<T>

ParameterIndex

Třída `ParameterIndex` je pouze pomocná třída použitá při přidávání nového parametru pro uchování informací o indexu parametru – `Index` a `Description`.

5.2.3 ViewModels

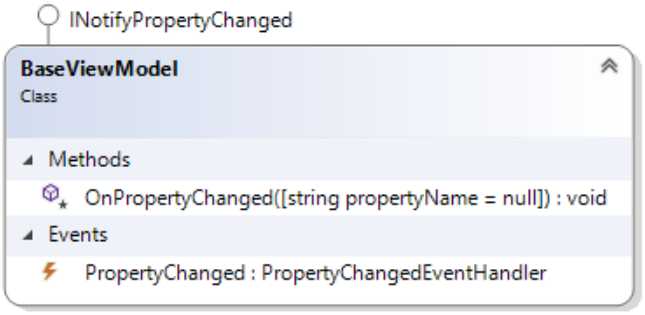
Složka ViewModels obsahuje view-modely všech view v aplikaci – `BaseViewModel`, `MainViewModel`, `SettingsViewModel` a `AddParameterViewModel`. Jejich hierarchii ukazuje Class diagram 13.



Class diagram 13 – Hierarchie ViewModelů

BaseViewModel

`BaseViewModel` (viz Class diagram 14) je bázovou třídou pro všechny ostatní view-modely v aplikaci. Dědí od rozhraní `INotifyPropertyChanged`, což poté umožňuje dvoucestný binding mezi jednotlivými view-modely a k nim náležícími view.



Class diagram 14 – Třída BaseViewModel

Obsahuje pouze jednu událost `PropertyChanged` a metodu `OnPropertyChanged`, která ji po kontrole přiřazení události invokuje s názvem změněné property. Díky atributu `CallerMemberName` u vstupního parametru `propertyName`, který je předáván metodě `OnPropertyChanged`, není potřeba pokaždé psát název vlastnosti, jež se změnila, ale je sám dosazen zjištěním názvu volající vlastnosti.

MainViewModel

Třída `MainViewModel` tvoří view-model pro hlavní okno a zpracovává všechny požadavky, které uživatel provede v GUI. Obsahuje sadu commandů a k nim přiřazené metody:

- `AddParameterCommand` – zpracovává požadavek na přidání nového parametru, otevře okno `AddParameterWindow`
- `ConnectionCommand` – zpracuje kliknutí na tlačítko Připojit/Odpojit
- `LoadFromFileCommand` – provede načtení uložené parametrové sady
- `LoadFromInverterCommand` – načte aktuální hodnotu vybraného parametru a jeho indexu z měniče
- `SaveToFileCommand` – uloží aktuálně zobrazenou parametrovou sadu do souboru
- `SendToInverterCommand` – pošle požadovanou hodnotu vybraného parametru a jeho indexu ke změně měniči

Dále uchovává `BindingList<IParameterSet> ParameterSets`, který obsahuje seznam parametrů a je bindován jako `ItemsSource` komponenty `DataGrid` v hlavním okně. K uchování aktuálně vybraného parametru a jeho indexu slouží property `SelectedParameter` a `SelectedParameterSet`.

Ukládání a načítání parametrové sady z/do souboru je realizováno pomocí třídy `BinaryFormatter`, která umožňuje serializaci i složitějších objektů do binárního souboru. Třídy, které jsou serializovány musí být označeny atributem `[Serializable]` a proměnné, jež uložit nechceme se označí atributem `[field: NonSerialized]`. Tím je možné určit vlastnosti, jejichž hodnoty uložím (např. číslo parametru, požadovaná hodnota, ale ne aktuální hodnota).

SettingsViewModel

View-model okna s nastavením komunikace `SettingsWindow` tvoří `SettingsViewModel`. Obsahuje commandy `ReloadPortNamesCommand` (znovunačtení seznamu připojených sériových portů) a `SaveValuesCommand` (uložení

nastavených hodnot do nastavení aplikace), seznamy možností pro komponenty ComboBox `BaudRates` a `PortNames` a aktuálně vybrané hodnoty: `SelectedBaudRate`, `SelectedDataBits`, `SelectedHandshake`, `SelectedParity`, `SelectedPortName` a `SelectedStopBits`.

AddParameterViewModel

Třída `AddParameterViewModel` představuje view-model pro `AddParameterWindow` se seznamem dostupných datových typů parametru `DataTypes` a hodnotami vyplněnými uživatelem `ParameterNumber`, `Description`, `IsArray`, `IsReadOnly`, `SelectedDataType` a kolekcí indexů `Indexes`.

5.2.4 Converters

V některých případech bindingu, aby se nemusela vytvářet nová property, se využívá tzv. converterů. Converter slouží k převodu jedné property ve view-modelu na jinou property komponenty ve view a naopak, např. převod `bool` na `Visibility`. Implementují interface `IValueConverter` s metodami `Convert` a `ConvertBack`.

BoolToVisibilityConverter

Pro konverzi booleovské hodnoty na hodnotu typu `Visibility` jsem vytvořila `BoolToVisibilityConverter`, který pokud je předaná `bool` hodnota z view-modelu rovnající se `true`, vrací `Visibility.Visible`, jinak vrací `Visibility.Hidden`. Tento converter využívám např. pro zobrazení vykřičníku v hlavičce řádků `DataGridu`, pokud dojde k nějaké chybě v parametru.

InvertedBoolConverter

`InvertedBoolConverter` slouží pro inverzi předané `bool` hodnoty. Pokud je předána hodnota `true`, vrací `false`, a naopak. Inverzi používám třeba u bindingu property `Focusable` buněk `DataGridu` na vlastnost parametru `IsReadOnly`, aby nebylo možné psát požadovanou hodnotu do parametru určeného pouze ke čtení.

6 ZHODNOCENÍ ŘEŠENÍ

Dle zadání jsem měla za úkol vytvořit aplikaci komunikující po sériové lince prostřednictvím USS protokolu. Velké množství času jsem strávila zjišťováním principu samotné komunikace, struktury USS telegramu a také samotným návrhem struktury třídy.

Při realizaci této bakalářské práce jsem narazila na pár menších problémů, které bylo potřeba vyřešit. Přestože jsem si strukturu modelů a rozložení zodpovědností mezi jednotlivé třídy předem navrhla, musela jsem některé třídy přepsat jiným způsobem.

6.1 VÝHODY A NEVÝHODY

Výhodou mnou vytvořené aplikace je, že poskytuje velice levný způsob, jak parametrizovat frekvenční měnič Siemens SINAMICS V20. Je pouze potřeba zajistit propojení měniče a PC pomocí sériové linky RS-485 (např. použitím převodníku na USB), což je finančně mnohem výhodnější, než třeba pořízení Wi-Fi modulu pro SINAMICS V20 Smart Access (kapitola 3.2.3). Navíc Smart Access nebyl ještě v době počátku vývoje aplikace dostupný.

Oproti Smart Access ale aplikace neumožňuje ovládání motoru, diagnostiku měniče a jiné funkce kromě parametrizace. Prozatím neumožňuje cyklickou komunikaci společně se zasíláním PZD oblasti, ale díky objektovému přístupu k návrhu aplikace by mělo být snadné aplikaci rozšířit o další funkcionalitu.

V porovnání s aplikací STARTER (kapitola 3.2.1) je ovládání mé aplikace pro nové uživatele mnohem intuitivnější, ale to může být dáno tím, že je určená pouze pro řadu Siemens SINAMICS V20 a není tedy potřeba předem nic konfigurovat. Opět neobsahuje stejné množství funkcí, nicméně ostatní funkce nebyly předmětem zadání bakalářské práce.

Další výhodou je, že oddělenou knihovnu pro USS komunikaci přes sériový port, je možné jednoduše použít jako jádro jiné aplikace, která by potřebovala komunikovat s jinými frekvenčními měniči od společnosti Siemens.

Pokud porovnám můj tabulkový přehled parametrů s tabulkovým přehledem v aplikaci DriveMonitor (kapitola 3.2.2), tak si myslím, že je má aplikace uživatelsky přívětivější kvůli možnosti rozkliknutí parametru a zobrazení všech jeho indexů spolu s jejich popisy.

6.2 Distribuce

Aplikaci jsem testovala pouze na vlastním notebooku s Windows 10 a nainstalovaným .NET Frameworkem verze 4.6.2 a jednom poskytnutým frekvenčním měniči Siemens SINAMICS V20. Pokud by se aplikace použila, bylo by ji třeba otestovat i na jiných zařízeních a zároveň by bylo vhodné přidat do aplikace logování chyb. V případě používání je potřeba věnovat více času testování aplikace a ošetření neplatných akcí uživatele.

V případě distribuce by bylo potřeba zvolit metodu, jakou se bude aplikace distribuovat. Zda by to byl pouze nástroj, který je možné šířit jako EXE, nebo zda by se vytvořil instalační soubor.

7 ZÁVĚR

Cílem mé bakalářské práce bylo provedení rešerše standardních softwarů pro ovládání měničů, seznámení se s procesem parametrizace měničů Siemens SINAMICS, způsobem komunikace prostřednictvím USS protokolu a následným vytvořením aplikace určené právě pro samotnou parametrizaci. Všechny body zadání jsem splnila.

V teoretické části bakalářské práce se zabývám seznámením se s principem komunikace po sériové lince prostřednictvím USS protokolu, přehledem základních informací o tomto protokolu a strukturou jeho telegramu. Dále představuji možnosti ovládání frekvenčních měničů a již existující aplikace určené k řízení, parametrizaci, monitorování a diagnostice měničů.

Následně popisuji přístup k návrhu aplikace pro parametrizaci měniče, která byla hlavním předmětem zadání. V dokumentu jsem rozebrala postup při vývoji aplikace SiemensSinamicsV20, vysvětlila rozhodnutí učiněná v rámci realizace aplikace a popsala funkce jednotlivých tříd a jejich metod.

V rámci praktické části bakalářské práce jsem dle návrhu vytvořila WPF aplikaci pro PC, která slouží k parametrizaci měniče. Je možné ji používat v online módu (měnič připojen) i v offline módu (bez připojeného měniče), kde si uživatel připraví požadované hodnoty parametrů pro pozdější nahrání.

Aplikace poskytuje uživateli tabulkový přehled parametrů měniče. Pomocí tlačítek je možné se připojit a odpojit od měniče, číst a modifikovat hodnoty jednotlivých parametrů a přidávat parametry nové. Před připojením může uživatel upravit nastavení komunikace pro sériový port. Dále umožňuje uložení a načtení parametrických sad z/do binárního souboru. Pro jednodušší práci s parametry je poskytnuto vyhledávací pole pro filtrování dle čísla a popisku parametru a řazení tabulky podle čísla parametru. Jelikož se u různých řad měničů liší čísla parametrů, ale princip komunikace zůstává stejný, mělo by být možné použít tuto aplikaci pro parametrizaci i jiných frekvenčních měničů, než jsou Siemens SINAMICS V20.

Do budoucna se nabízí přidání možnosti alespoň základního ovládání motoru (start, stop, nastavení otáček), což by vyžadovalo rozšíření knihovny UssCommunication o cyklickou komunikaci a obsluhu PZD oblasti USS telegramu. Také by bylo vhodné umístit do GUI aplikace indikátory stavu měniče a nějaký filtr umožňující vybrat jen parametry týkající se vybraných oblastí nastavení.

SEZNAM POUŽITÉ LITERATURY

[1] MÖLLER-NEHRING, Walter a Wolfgang BOHRER. SIEMENS AG. *Universal Serial Interface Protocol Specification USS Protocol*®. Germany, 1994. 09.94. E20125-D0001-S302-A1-7600.

[2] SIEMENS. Měnič SINAMICS V20 Návod k použití. 03/2015. A5E34559884-002.

[3] Frekvenční měniče - proč a jak. *Elektromotory.cz* [online]. ©2015 [cit. 2017-05-14]. Dostupné z: https://www.elektromotory.cz/podpora/poradna-neboli-jak-na-to/frekvencni-menice-proc-a-jak

[4] DriveMonitor Version V5.5 SP2 for SIMOVERT MASTERDRIVES, MICROMASTER and SIMOREG. *Siemens Global Website* [online]. © Siemens AG, 2009-2017 [cit. 2017-05-14]. Dostupné z: https://support.industry.siemens.com/cs/document/11769381/drivemonitor-version-v5-5-sp2-for-simovert-masterdrives-micromaster-and-simoreg

[5] STARTER Commissioning Software. *Siemens Global Website* [online]. © Siemens AG, 2009-2017 [cit. 2017-05-14]. Dostupné z: http://w3.siemens.com/mcms/mc-solutions/en/engineering-software/starter-commissioning-tool/pages/starter-commissioning-tool.aspx

[6] SINAMICS V20 Basic converter. *Siemens Global Website* [online]. © Siemens AG, 2009-2017 [cit. 2017-05-14]. Dostupné z: http://w3.siemens.com/mcms/mc-drives/en/low-voltage-inverter/sinamics-v20/Pages/sinamics-v20.aspx

[7] HOSEK, P. a M. DIBLIK. Implementation of Siemens USS Protocol Into LabVIEW. *Journal of Laboratory Automation*. 2011, **16**(5), 347-354. DOI: 10.1016/j.jala.2009.07.008. ISSN 22110682. Dostupné také z: http://jla.sagepub.com/lookup/doi/10.1016/j.jala.2009.07.008

[8] Implementing the Model-View-ViewModel Pattern. *Microsoft Developer Network | MSDN* [online]. Microsoft, ©2017 [cit. 2017-05-14]. Dostupné z: https://msdn.microsoft.com/en-us/library/ff798384.aspx

[9] Implementing Singleton in C#. *Microsoft Developer Network* | *MSDN* [online]. Microsoft, ©2017 [cit. 2017-05-14]. Dostupné z: https://msdn.microsoft.com/en-us/library/ff650316.aspx

[10] SerialPort Class. *Microsoft Developer Network* | *MSDN* [online]. Microsoft, ©2017 [cit. 2017-05-14]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.io.ports.serialport(v=vs.110).aspx

[11] *Website wireframes: Mockingbird* [online]. Some Character LLC [cit. 2017-05-14]. Dostupné z: https://www.gomockingbird.com

[12] SINAMICS V20 Smart Access. Siemens AG, Unrestricted © 2016.

[13] MÝLEK, Vladimír. *Průmyslové elektrické pohony - motory, frekvenční a stejnosměrné měniče* [online]. Siemens, s.r.o., ©2015 [cit. 2017-05-15]. Dostupné z: https://w5.siemens.com/web/cz/cz/corporate/portal/home/produkty_a_sluzby/IADT/tia_na_dosah/Documents/2015_unor/Prumyslove%20elektricke%20pohony%20-%20male%20menice.pdf

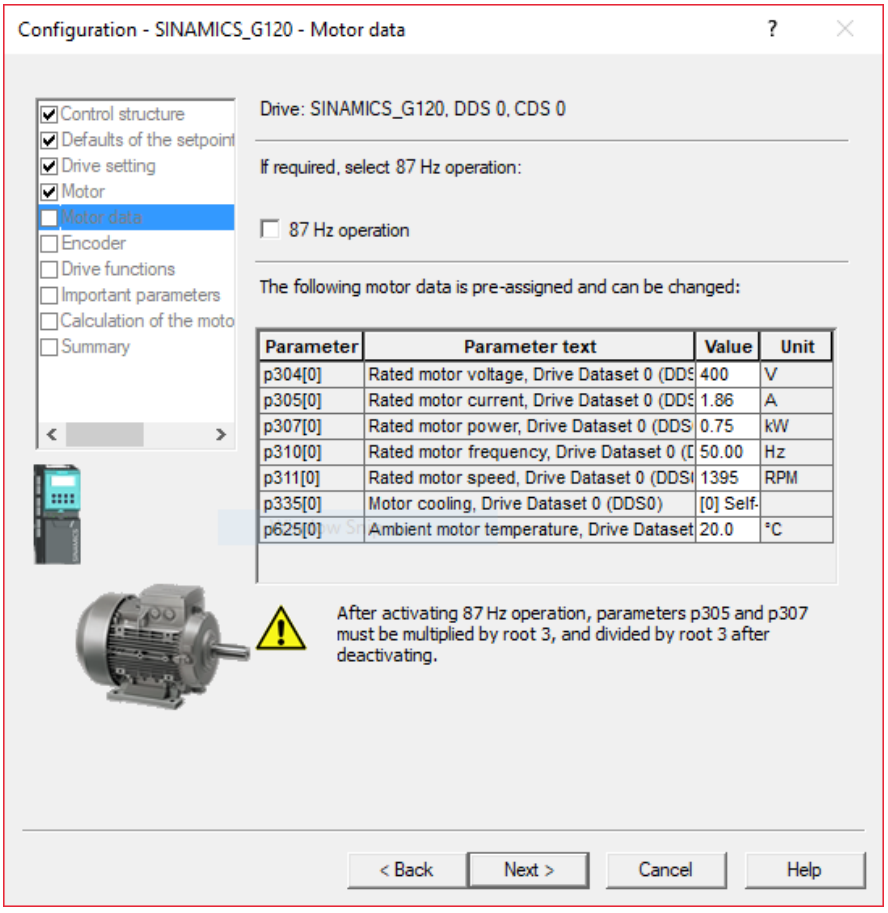
Příloha A – PŘILOŽENÉ CD

Na přiloženém CD je kromě tohoto dokumentu nahraná i vytvořená aplikace SiemensSinamicsV20 a její zdrojové kódy.

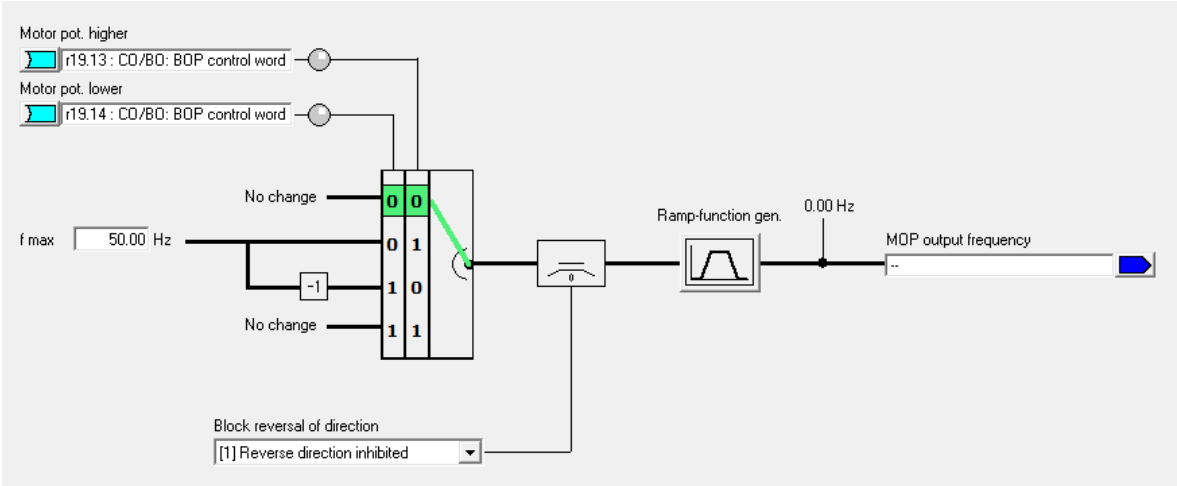
Popis jednotlivých položek:

- Adresář SiemensSinamicsV20 – obsahuje veškeré zdrojové kódy ve formě Visual Studio Solution v jazyce C#
- BP_Katerina_Lebedova.pdf – tento dokument v elektronické formě

Příloha B – STARTER



Obr. 8 – Aplikace STARTER – Průvodce nastavením



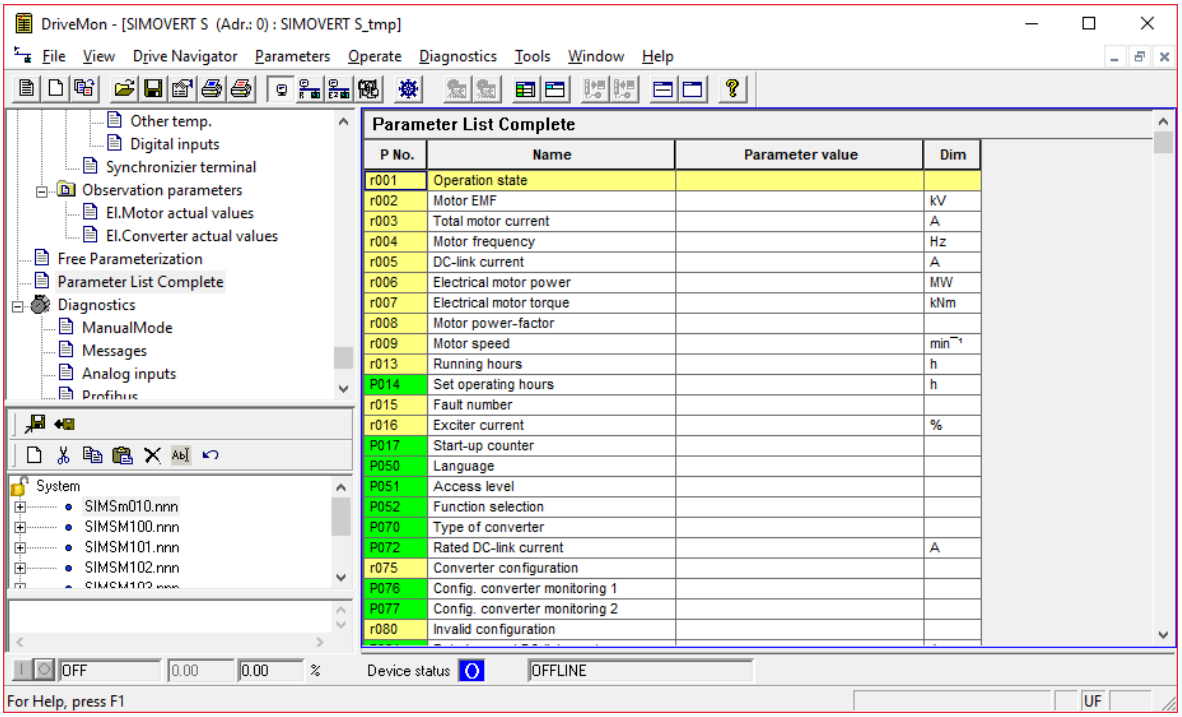
Obr. 9 – Aplikace STARTER – Nastavení pomocí grafických schémat

Expert list

	Param...	Data	Parameter text	Offline value G120	Unit	Modifiable to	Access level	Minimum	Maximum
	All	A_	All	All	Al_	All	All	All	All
1	r2		Drive state	[0] Commissioning ...			2		
2	p3		User access level	[1] Standard: Allo...		Operation	1		
3	p4		Parameter filter	[0] All parameters		Operation	1		
4	⊞ p5[0]	D	Display selection, Drive Dataset 0 (DDS0)	21		Operation	2	2	4000
5	p6		Display mode	[2] Ready state: P0...		Operation	3		
6	p7		Backlight delay time	0		Operation	3	0	2000
7	p10		Commissioning parameter	[0] Ready		Ready to run	1		
8	p11		Lock for user defined parameter	0		Operation	3	0	65535
9	p12		Key for user defined parameter	0		Operation	3	0	65535
10	⊞ p13[0]		User defined parameter, 1st user parameter	0		Operation	3	0	65535
11	⊞ p14[0]		Store mode, USS on RS485	[0] Volatile (RAM)		Operation	3		
12	r18		Firmware version	0.00			1		
13	⊞ r19		CO/BO: BOP control word	0H			3		
14	r20		CO: Freq. setpoint before RFG	0.00	Hz		3		
15	r21		CO: Act. filtered frequency	0.00	Hz		2		
16	r22		Act. filtered rotor speed	0	RPM		3		
17	r24		CO: Act. filt. output frequency	0.00	Hz		3		
18	r25		CO: Act. output voltage	0	V		2		
19	⊞ r26[0]		CO: Act. filtered DC-link volt., Compensation DC voltage Channel	0	V		2		
20	r27		CO: Act. output current	0.00	A		2		
21	r29		CO: Flux gen. current	0.00	A		3		
22	r30		CO: Torque gen. current	0.00	A		3		

Obr. 10 – Aplikace STARTER – Expert list

Příloha C – DRIVEMONITOR

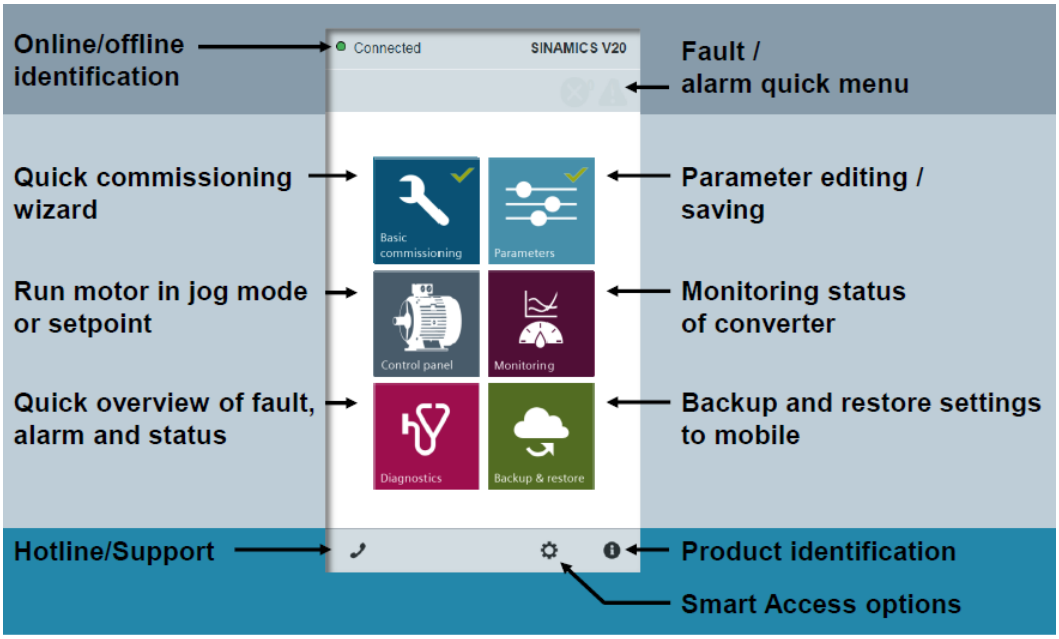


Parameter List Complete

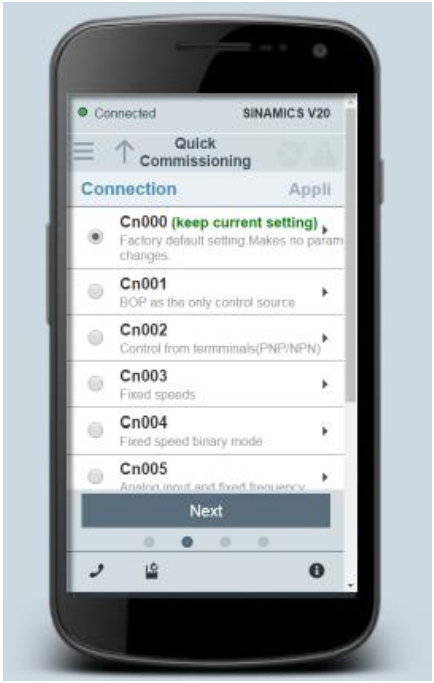
P No.	Name	Parameter value	Dim
r001	Operation state		
r002	Motor EMF		kV
r003	Total motor current		A
r004	Motor frequency		Hz
r005	DC-link current		A
r006	Electrical motor power		MW
r007	Electrical motor torque		kNm
r008	Motor power-factor		
r009	Motor speed		min^{-1}
r013	Running hours		h
P014	Set operating hours		h
r015	Fault number		
r016	Exciter current		%
P017	Start-up counter		
P050	Language		
P051	Access level		
P052	Function selection		
P070	Type of converter		
P072	Rated DC-link current		A
r075	Converter configuration		
P076	Config. converter monitoring 1		
P077	Config. converter monitoring 2		
r080	Invalid configuration		

Obr. 11 – Aplikace DriveMonitor – Parametrizace

Příloha D – SINAMICS V20 SMART ACCESS

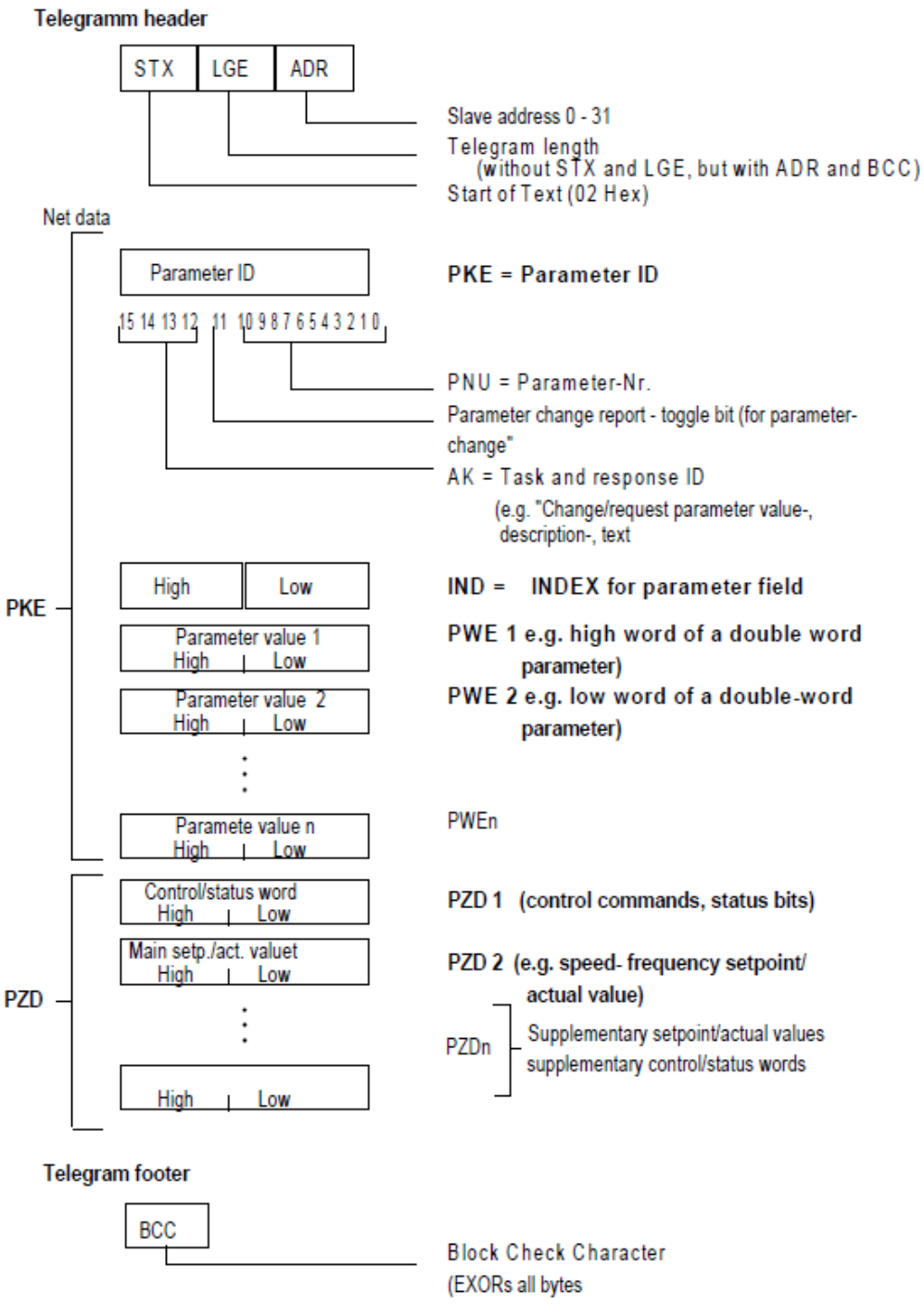


Obr. 12 – Aplikace Smart Access – Přehled funkcionalit; zdroj [12]



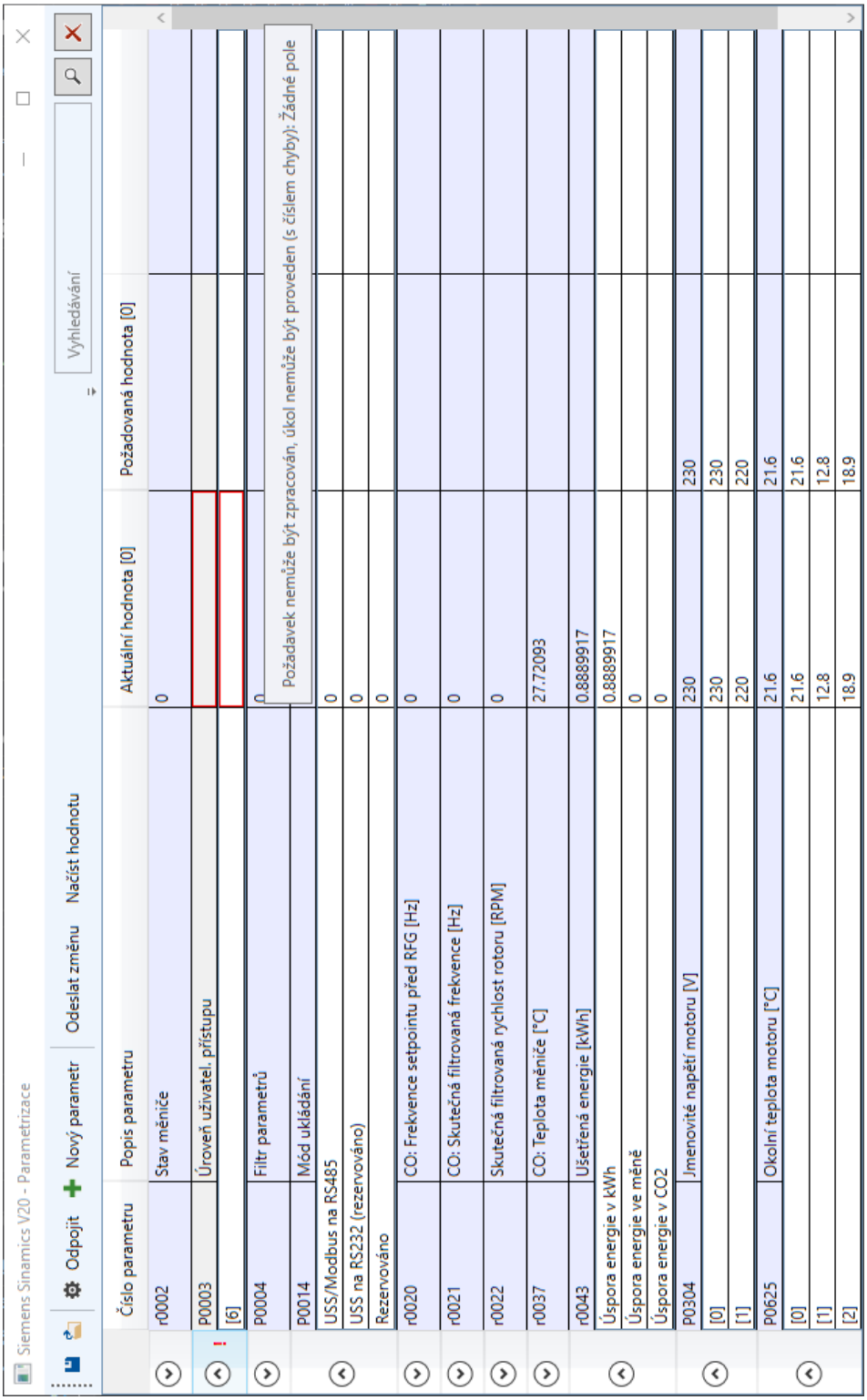
Obr. 13 – Aplikace Smart Access – Ukázka aplikace, výběr z předpřipravených nastavení; zdroj [12]

Příloha E – STRUKTURA TELEGRAMU



Obr. 14 – Kompletní struktura USS telegramu; zdroj [1]

Příloha F – SCREENSHOT APLIKACE



Obr. 15 – Screenshot hlavního okna vytvořené aplikace