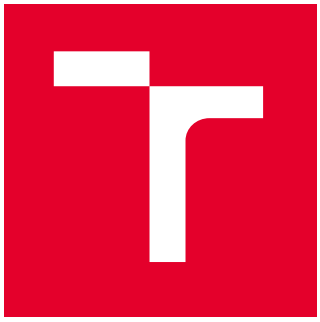


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

SW APLIKACE PRO NASTAVENÍ PLC MODEMŮ

SW FOR CONFIGURATION OF PLC MODEMS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Vojtěch Tichý**

AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Petr Fiedler, Ph.D.**

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Vojtěch Tichý ***ID:*** 221025

Ročník: 3 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

SW aplikace pro nastavení PLC modemů

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvoření PC aplikace pro nastavení PLC modemů fy. ModemTec. Pomocí aplikace se definuje požadovaná komunikační síť, nastaví jednotlivé parametry modemů a přenese vytvořena konfigurace do připojených modemů.

1. Seznamte se s PLC modemem a jeho parametry pro nastavení.
2. Vyberte vhodné nástroje pro vytvoření konfigurační aplikace pro PC.
3. Navrhněte aplikaci s ohledem na snadnost použití.
4. Implementujte aplikaci ve zvoleném prostředí.
5. Otestujte aplikaci s reálnými PLC modemy.

DOPORUČENÁ LITERATURA:

1. Interní dokumentace firmy Modemtec
2. PECINOVSKÝ, Rudolf. Návrhové vzory. Brno: Computer Press, 2007. ISBN 8025115828.

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 23.5.2022

Vedoucí práce: doc. Ing. Petr Fiedler, Ph.D.

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

ABSTRAKT

Cílem této bakalářské práce byl návrh aplikace pro snadnou konfiguraci modemů, od společnosti ModemTec, komunikujících po silnoproudém vedení. Práce se nejdříve věnuje krátkému vysvětlení komunikace po silnoproudých vedení, která povrchově seznámí čtenáře k čemu bude aplikace využívána. Dále se pro pochopení významu jednotlivých nastavitelných parametrů modemu, vysvětluje sériový port, pomocí kterého bude modem komunikovat s nadřazeným zařízením, jenž sbírá data, ale také s počítačem, na kterém bude aplikace spuštěna a pomocí které bude modem konfigurován. Poté jsou čtenáři obeznámeni s návrhem uživatelského rozhraní a architektury aplikace, jejíž součástí jsou základní návrhové principy, podle kterých byla aplikace implementována. Následuje popis implementace aplikace a výsledky testování konfigurace modemů společnosti ModemTec. Závěrem jsou shrnuty dosažené výsledky a možnosti pro budoucí modifikaci.

KLÍČOVÁ SLOVA

Komunikace po silnoproudém vedení, ModemTec, PLC modem, WPF, desktopová aplikace, C#, .NET

ABSTRACT

The aim of this bachelor thesis is to design an application for easy configuration of modems, from ModemTec, communicating over power lines. The bachelor's thesis first deals with a brief explanation of the communication over the power line, which will give the reader a general idea of what the application will be used for. Furthermore, to understand the meaning of the individual adjustable parameters of the modem, the serial port is explained, through which the modem will communicate with the parent device that collects data, but also with the computer on which the application will run and with which the modem will be configured. Then, readers are introduced to the design of the user interface and architecture of the application, which includes the basic design principles which the application implementation was based on. The following is a description of the application implementation and the results of ModemTec modem configuration testing. Finally, the achieved results and possibilities for future modification are summarized.

KEYWORDS

Powerline Communication, ModemTec, PLC modem, WPF, desktop application, C#, .NET

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Vojtěch Tichý
VUT ID autora:	221025
Typ práce:	Bakalářská práce
Akademický rok:	2021/22
Téma závěrečné práce:	SW aplikace pro nastavení PLC modemů

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu doc. Ing. Petru Fiedlerovi Ph.D. za podnětné návrhy ke zpracování bakalářské práce a udělení možnosti k vypracování zadání od zadavatele - společnosti ModemTec. Zároveň bych tak rád poděkoval i společnosti ModemTec, která mi umožnila vypracovat bakalářskou práci pro ně.

Obsah

Seznam obrázků

Úvod

Se vzrůstajícím počtem obnovitelných zdrojů energie, stoupá požadavek na jejich využití, protože jsou oproti fosilním zdrojům energie nevyčerpatelné a zároveň přívětivější pro naše prostředí. Bohužel je ale množství energie, kterou jsou schopny dodat, závislé na vnějších vlivech, což je činí nestabilními. Pokud bychom chtěli obnovitelné zdroje využívat co nejefektivněji, museli bychom být schopni neustále monitorovat jejich dostupnost a komunikovat tyto data do systémů, které slouží k distribuci elektrické energie. Tento účel je jedním z důvodů, proč se využívá komunikace po silnoproudých vedeních. [1]

Společnost ModemTec se zabývá problematikou komunikace po silnoproudých vedeních a jedním z poskytovaných výrobků jsou modemy umožňující přeposílání dat po silnoproudé síti, získaná ze zařízení, ke kterým jsou modemy přes sériový port připojeny. Tato práce se zaměřuje na vývoj aplikace, která umožní co nejsnazší konfiguraci parametrů modemů pro komunikaci jak se zařízeními, která slouží ke sběru dat, tak vzájemně mezi sebouparametry sítě, pomocí kterých mezi sebou modemy komunikují.

Cílem této práce je vytvořit aplikaci *SemSet*, která bude sloužit pro snadnou konfiguraci PLC modemů od společnosti ModemTec. Dalším cílem zadavatele je obeznámení autora bakalářské práce se způsobem vývoje softwarových aplikací a nabytí potřebných znalostí o vývojový principech a návrhových vzorech, které budou přínosné pro implementace v budoucích projektech. Návrh aplikace se bude řídit existencí možnosti budoucí modifikace nebo rozšíření funkcionality aplikace, kdy daný požadavek musí být co nejsnáze přidán, aniž by se musela předělávat většinu zdrojového kódu aplikace.

1 Komunikace po silnoproudém vedení

Původním záměrem silnoproudého vedení bylo přenášení elektrické energie. Nutnost ochrany rozvodů energie v případě poruch vedla k prvnímu využití datových přenosů po vedení, kde vyžadujeme co nejrychlejší výměnu informací za účelem co největšího zmírnění škod. Toto nám umožňuje technologie s názvem Power Line Communication (PLC) neboli komunikace po silnoproudém vedení. [2]

První generace digitálního přenosu dat po napájecí síti dosahovala nízkých rychlostí. Především sloužila k ochraně vedení a později k hlasové komunikaci. Mnohonásobně zvýšit rychlosti přenosu dat se povedlo až s příchodem širokopásmové PLC technologie. Návrh rozvodných sítí však nebral v úvahu možnou implementaci komunikace, proto využívané médium není ideální a má své nedostatky. Napříč všem problémům dnes zůstává technologie PLC silným konkurentem bezdrátových sítí pro aplikace v domácí automatizaci, inteligentních sítích a telemetrie, ale největší využití nachází v dálkovém sběru dat. [3] [4]

1.1 Princip

Pro přenos dat pomocí PLC technologie se využívá znalosti frekvence přenosu elektrické energie, která je v ČR 50 Hz. Díky tomu jsou známá volná frekvenční pásma, která mohou sloužit pro přenos dat. Odesílaná data jsou nejdříve namodulována na napětí v napájecím vedení a přenesena po síti. Poté se tento modulovaný signál na straně příjemce demoduluje a užitečná data jsou extrahována. PLC technologie dělíme dle frekvenčních pásem, pomocí kterých přenáší data, na:

- Ultra úzkopásmovou technologii s rozsahem frekvencí od 30 do 300 Hz, rychlostí 10 kb/s a vzdáleností komunikace až 150 km.
- Úzkopásmovou technologii s frekvenčním rozsahem od 3 do 148,5 kHz, který je definován Evropskou normou CENELEC. Rychlost komunikace se pohybuje v řádech desítek kb/s.
- Širokopásmovou technologii s nejširším rozsahem frekvencí (2 - 30 MHz) dosahujících rychlostí v řádech až stovek Mb/s, ale využívá složitějších modulací a nepřenáší na velké vzdálenosti. [4] [5]

1.2 Výhody a nevýhody

Instalace vedení pro nízko nebo vysoko rychlostní sítě v soukromých obydlích vyžaduje velké množství práce a prostředků. PLC technologie využívá již existujících rozvodů elektrické energie, takže její instalace je levnější. Vysoká rozšířenost silnoproudých rozvodů odpovídá dostupnosti, kterou PLC technologie nabízí, díky které je možné

PLC sloučit jako redundantní kanál pro zvýšení ochrany. V domácích infrastrukturách s vysokou koncentrací elektrických zásuvek a spotřebičů, představuje efektivní způsob komunikace a sdílení dat mezi inteligentními zařízeními. [4] [6]

Vysoká rozšířenost PLC technologie, však ubírá na bezpečnosti přenosu dat. Zároveň dochází k vysokému rušení, z důvodu neoptimálního návrhu silových rozvodů, které vede k chybovému příjmu dat. Takže ačkoliv je instalace napájecího vedení již implementována, pro komunikaci jsou zapotřebí drahé PLC modemy (dále jen "modemy"), které dokážou data zpracovávat. [4] [6]

2 Seznámení se s modemy firmy ModemTec

Pro uskutečnení komunikace po silnoproudém vedení, je nutné využít zařízení, které dokáže přijímat a posílat data skrz PLC síť a zároveň komunikovat s nadřazeným zařízením (Masterem), ze kterého například sbírá data. K tomuto účelu jsou využity modemy. V následujících podkapitolách si přiblížíme funkčnost modemů společnosti ModemTec.

2.1 Základní popis

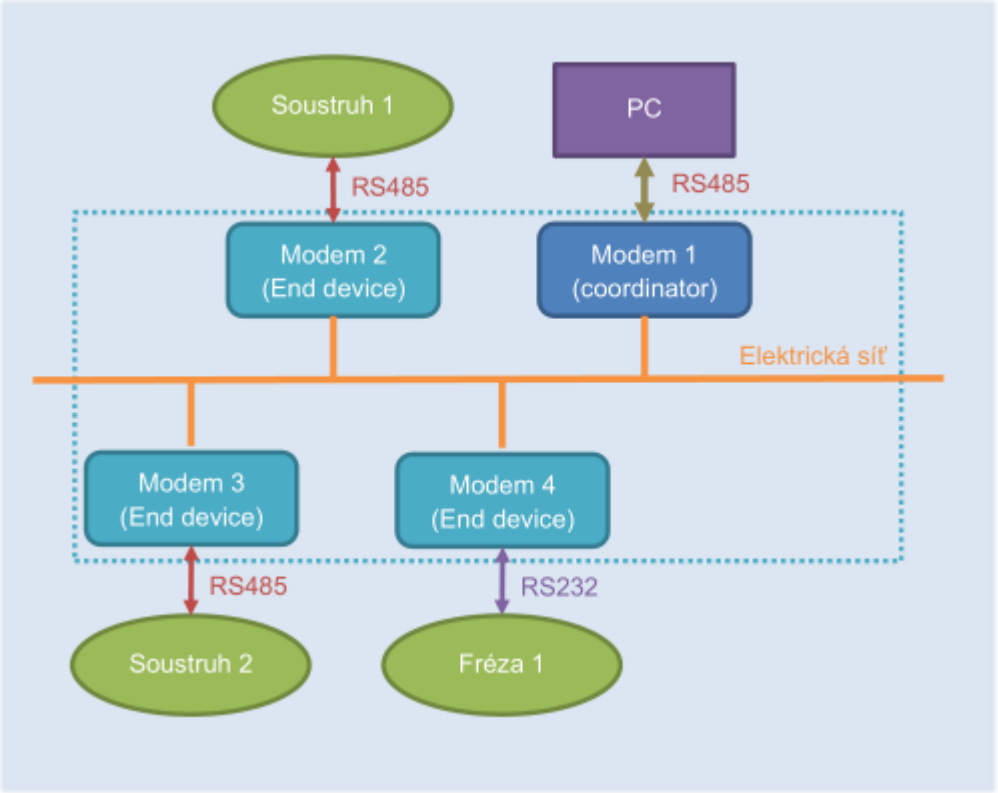
Společnost ModemTec využívá modemy založené na procesoru STM32 a komunikačním čipu Semitech SM2480. S odpovídajícím firmwarem pracuje modem v rámci standardu G3-PLC, pomocí kterého je komunikace mezi jednotlivými modemy v PLC síti zabezpečena. Z jedné strany je modem tedy vybaven PLC rozhraním pro připojení do G3-PLC sítě a ze strany druhé je vybaven sériovým portem pro poskytnutí připojení nadřazenému zařízení. [7]

Modemy, které jsou součástí PLC sítě, se dělí dle jejich role na koordinátory (Coordinator) a koncová zařízení (End Device), viz obrázek 2.1. Koordinátor komunikuje se všemi modemy v síti a je vždy pouze jeden, protože síť, po které komunikace probíhá, zakládá. Koncová zařízení jsou modemy sloužící k propojení se zařízeními, která například sbírají data a dalšími zařízeními, se kterými chceme po PLC síti komunikovat.

2.2 Pracovní režimy

Modem může pracovat v příkazovém nebo transparentním režimu. V příkazovém režimu je veškerá komunikace s nadřazeným zařízením realizována formou příkazů, které jsou předávány přes sériový port. Tyto příkazy jsou zapouzdřeny do paketů, jejichž formát je předem definovaný firemním protokolem pro sériovou linku. Forma komunikace mezi modemem a nadřazeným zařízením funguje především způsobem dotaz-odpověď. Výjimkou je asynchronní událost příjmu dat z PLC sítě, kdy modem pošle přijatá data po sériové lince zapouzdřené do IPV6 paketu. Všechny příkazy pomocí kterých lze komunikovat s nadřazeným zařízením si uvedeme v podkapitole 2.3.2.

Transparentní režim se od příkazového liší tím, že veškerá komunikace směrem od nadřazeného zařízení na sériový port modemu je považována za data. Modem nadřazenému zařízení neodpovídá, ale pouze zabalí přijatá data do IPV6 paketu a pošle jej na druhý modem (popřípadě na všechny modemy) skrz PLC síť. Na druhé



Obr. 2.1: Příklad hierarchie komunikační sítě [7]

straně jsou data z paketu rozbalena a pomocí sériového portu modemu přeposlána nadřazenému zařízení. [7]

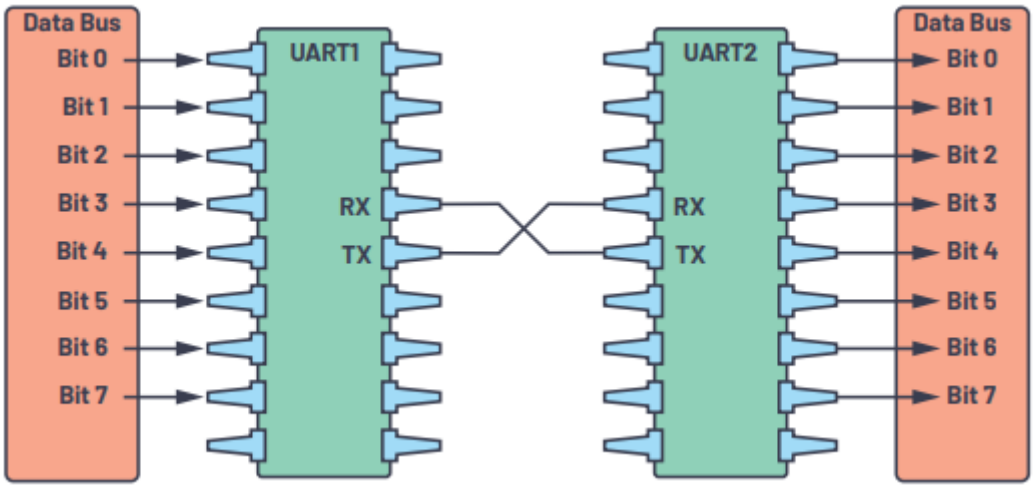
2.3 Komunikace s nadřazeným zařízením

Modem komunikuje s nadřazeným zařízením pomocí sériového portu. V následujících podkapitolách je vysvětleno, jak je sériový port využit pro přenos dat mezi zařízeními. Dále je vysvětlen protokol, který využívá společnost ModemTec k přenosu příkazů a odpovědí po sériové lince. V poslední části jsou uvedeny příkazy, které nabízí rozhraní sériového protokolu společnosti ModemTec.

2.3.1 UART

Jeden z nejrozšířenějších protokolů pro komunikaci mezi dvěma zařízeními je universal asynchronous receiver-transmitter (UART) neboli univerzální asynchronní přijímač-vysílač, často označován jako sériový port. Při sériové komunikaci dochází k přenosu dat bit po bitu po jediném vodiči. Z toho vyplývá, že nevyžadujeme nadbytečné vodiče pro přenos synchronizačních hodin, což sníží cenu implementace. [8]

Každé zařízení využívající UART má alespoň dva signálové piny. Jeden pro příjem dat (Rx) a druhý pro odesílání dat (Tx). Díky tomu může probíhat sériová komunikace plně duplexně neboli může přijímat i posílat data zároveň. Na obrázku 2.2 vidíme příklad propojení dvou zařízení. Data ze sběrnice jsou paralelně předána zařízení UART1. Poté jsou zpracována pro sériové odeslání, bit po bitu, na přijímací zařízení UART2. Jak již bylo zmíněno, zařízení nepotřebují synchronizační hodinový cyklus, který by bylo nutné přenášet dalším vodičem. Pro úspěšnou komunikaci pomocí UART protokolu je nutné dodržet, aby se nastavená přenosová rychlost obou zařízení blížila stejné hodnotě. Kdyby tato podmínka splněna nebyla, rozdíl v přenosových rychlostech by způsobil zpomalení, respektive zrychlení intervalů, ve kterých mají být bity čteny, což vede k nekorektnímu příjmu dat. [8]



Obr. 2.2: Propojení protokolu UART s datovou sběrnicí [8]

Máme-li data, která chceme přenést přes sériový port, je potřeba upozornit přijímací zařízení, kdy má začít a kdy skončit číst ze sériové linky. K tomuto slouží start a stop bity, které spolu s paritním bitem a datovým rámcem tvoří UART paket, který lze vidět na obrázku 2.3. Datový rámec je tvořen datovými bity, které odpovídají datům, která chceme přenést přes sériovou linku. Velikost datového rámce není pevně určena. Počet datových bitů v datovém rámci přenesených na jeden UART paket, je možné měnit. Minimální počet datových bitů je 5. Maximální počet datových bitů je 9. Přítomnost paritního bitu, který může být součástí UART paketu, sníží maximální počet přenesených datových bitů na paket o 1 bit. Díky přítomnosti paritního bitu, je však možné kontrolovat, zda je přenesený paket po sériové lince úplný nebo zda došlo k chybě. Hodnota paritního bitu závisí na nastavení sudé nebo liché parity přenosu. Je-li nastavení parity sudé, bude úroveň paritního bitu taková,

aby při odeslání paketu počet datových bitů s úrovní 1, tvořil společně s paritním bitem sudé číslo. Při nastavení liché parity bude výsledný počet lichý. [8]

Obr. 2.3: UART paket [8]

2.3.2 ModemTec Serial Protocol

Zřetězením jednotlivých UART paketů lze vytvořit rámec protokolu, sloužící pro bezpečnou a sjednocenou komunikaci mezi jednotlivými zařízeními. ModemTec Serial Protocol je jedním z těchto protokolů. Rámec protokolu se skládá z hlavičky (Header), datového obsahu (Payload) a konce rámce (Tail). Díky složitějšímu a komplexnějšímu sestavení rámce, se umožní posílání předem definovaných zpráv mezi zařízeními. Tyto zprávy mohou například být příkazy, které jsou poslány z jednoho zařízení na druhé. Druhé zařízení má možnost odpovídat, zda zprávu přijalo a úspěšně příkaz vykonalo nebo zda naopak selhalo. Důležité je, že komunikace je kontrolovaná, takže nemůže dojít k chybnému přenosu signálu bez toho, aniž by byla chyba detekována a buďto zpráva rekonstruována, nebo odeslána odpovědět žádající o opakování zprávy. Zároveň díky zprávám ve formě příkazů, mohou být jednotlivá zařízení konfigurována, například parametry modemů, o kterých pojednává následující kapitola. [7]

2.4 Přehled parametrů modemu

V naší aplikaci se zabýváme nastavením jednotlivých parametrů modemu pro požadovaný chod sítě. Některé parametry jsme již zmínili v předchozích podkapitolách. Přesto si je zde uvedeme znovu s krátkým vysvětlením jejich funkčnosti.

2.4.1 Nastavení sériového portu

Umožňuje nastavit parametry pro přenos dat mezi modemem a nadřazeným zařízením pomocí sériového portu.

Přenosová rychlost

Udává rychlost, kterou se jednotlivé bity přenášejí, respektive čtou. Standardní přenosová rychlost je nastavena na 9600 b/s. [7]

Počet datových bitů

Určuje kolik bitů dat se přenese v jednom UART paketu. Modem umožňuje volbu mezi sedmi nebo osmi datovými bity. Standardně je posíláno 8 datových bitů. [7]

Počet stop bitů

Udává počet bitů na konci UART paketu, které slouží k signalizaci konce poslaného paketu. Standardně je paket zakončen jedním stop bitem, ale může obsahovat i dva. [7]

Parita

Slouží k definování paritního bitu, který je možné přidat za datový rámec v UART paketu. Standardně komunikuje modem s nadřazeným zařízením bez paritního bitu. Avšak modem umožňuje další čtyři možnosti nastavení parity. Parita *mark* přidá vždy paritní bit s hodnotou 1. Opakem toho je parita *space*, kdy paritní bit bude přidán vždy s hodnotou 0. Poslední dvě možnosti jsou sudá a lichá parita, které slouží ke kontrole úplnosti přenesených dat. Hodnoty paritních bitů v těchto režimech jsme si uvedli v podkapitole 2.3.1. [7]

Řízení toku

Slouží k řízení přenosu dat, při komunikaci rychlého a pomalého zařízení pomocí UART protokolu. Modem ve standardním nastavení žádné řízení toku nevyužívá, ale nabízí tok dat hardwarově řídit pomocí pinů RTS (Request To Send) a CTS (Clear To Send), které jsou součástí sériových rozhraní jako třeba RS-232. RTS pin jednoho zařízení je připojený k CTS pinu druhého zařízení a naopak. V případě, že pomalejší zařízení již není schopné přijímat další data, změní hodnotu na svém RTS pinu. Tuto změnu rychlejší zařízení zaznamená na svém CTS pinu a přeruší posílání dat. Jakmile bude pomalejší zařízení opět schopné přijímat data, nastaví hodnotu RTS pinu zpět a komunikace se obnoví. [7][9]

2.4.2 Síťová nastavení

Umožňuje nastavit parametry pro komunikaci s ostatními modemy v PLC síti.

Adresa

Udává IPV6 adresu zařízení, díky které lze rozpoznat adresáta, kterému byly data určeny. [7]

Čas kontroly připojení

Slouží k nastavení periodického intervalu, po kterém se pošle kontrolní zpráva, která vyhodnocuje stav připojení modemu k síti. [7]

Před sdílený klíč

Pro zabezpečení komunikace po G3-PLC síti, využívá každé zařízení svůj unikátní náhodně vygenerovaný klíč. Využívá se k autentizaci a identifikaci připojeného zařízení. [7]

Směrování

Udává, zda modem posílá data pouze jednomu dalšímu modemu (Unicast) a nebo jestli posílá data všem v síti (Broadcast). [7][10]

3 Návrh aplikace

Tato kapitola se zabývá popisem požadavků na funkcionalitu aplikace a význačnými rysy, které by aplikace měla obsahovat. Dále je prezentován návrh uživatelského rozhraní, rozdělení jednotlivých oken a popis k čemu slouží. Poslední část kapitoly vysvětluje různé druhy architektur softwarových aplikací a způsoby jejich implementací. Uvádí dělení projektů aplikace a jejich zasazení do vrstev architektury, dle kterých bude aplikace implementována.

3.1 Popis funkcionality aplikace

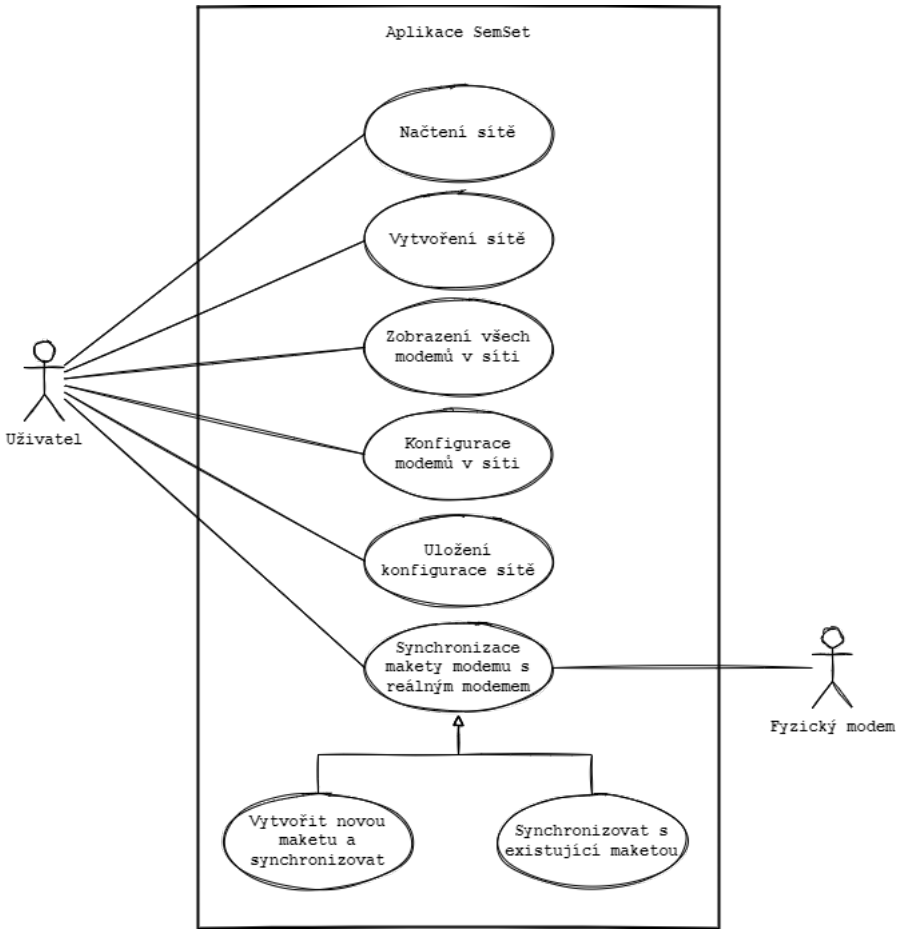
Námi vytvořená aplikace bude sloužit k utvoření osobní sítě na elektrické síti (viz modré ohraničení na obrázku 2.1). Samotná aplikace nebude síť přímo vytvářet, ale s její pomocí se nahraje vytvořená konfigurace do modemů, které za pomoci firmwaru společnosti ModemTec síť vytvoří.

Aplikace umožní vytvořit model reálné sítě, u které se nastaví její identifikátor a popis. Do sítě se pak přidají makety modemů v roli koordinátora nebo koncového zařízení, které budou reprezentovat skutečné modemy. Připojením opravdového modemu k počítači bude možné synchronizovat parametry makety, tak aby se rovnaly parametrům modemu. Bude-li naopak potřeba upravit parametry nebo nahrát aktuální nastavení sítě do skutečného modemu, změní se parametry makety v aplikaci a přes sériový port se nahraje její nastavení do modemu. Pokud se tímto způsobem nakonfiguruje každý modem v síti, bude vytvořená síť odpovídat modelu sítě v aplikaci.

Jakmile by uživatel vytvořil síť a nakonfiguroval modemy, bylo by nepraktické po něm požadovat udržení chodu aplikace pro zachování vytvořené konfigurace. Proto je přínosné, zavést možnost ukládání a načítání různých konfigurací sítí pro případné budoucí změny. Diagram případu užití pro uživatele aplikace lze vidět na obrázku 3.1.

3.2 Požadavky na aplikaci

Hlavním požadavkem na aplikaci je snadnost a efektivnost jejího použití. Požaduje se, aby uživatelé nemuseli hledat jednotlivé funkce, popřípadě přemýšlet k čemu co slouží. Domníváme se, že s pomocí ikon je možné vytvořit intuitivnější grafické rozhraní aplikace. Pokud by i přesto nastaly nejasnosti, implementace nápověd, které se zobrazí při najetí kurzoru myši, by mohly pomoci tento problém vyřešit. V případě jakékoliv chyby aplikace je nutné informovat uživatele o nevydařené operaci pomocí vyskakovacích oken. Pro zaručení co největší šance, aby k žádné takové chybě



Obr. 3.1: Diagram případů užití pro uživatele navrhované aplikace

nedošlo, aplikace by měla kontrolovat zadané hodnoty uživatelem, aby nedošlo k uložení chybného nastavení, které by mohlo nepříznivě ovlivnit chod aplikace.

Zákazníci společnosti ModemTec pocházejí i ze zemí, kde se nemluví českým jazykem. Z tohoto důvodu je potřeba zajistit, aby aplikaci byli schopni ovládat všichni zákazníci. Toho je možno docílit zavedením změny jazyka aplikace do angličtiny.

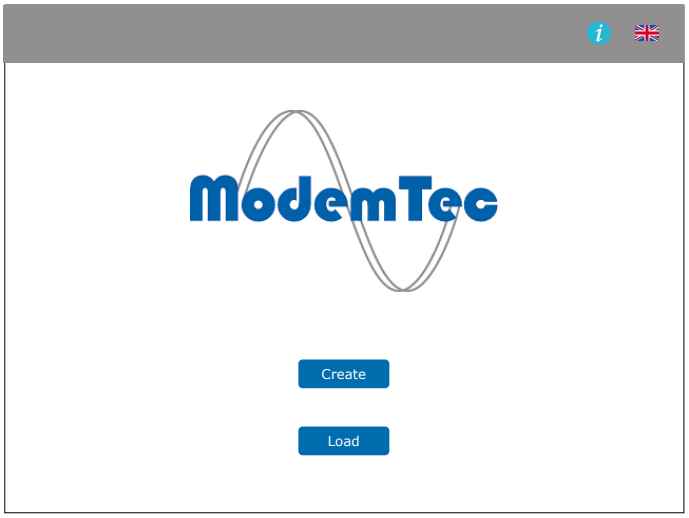
V budoucnosti by mohlo nastat, že by se konfigurace modemů nemusela provádět skrz propojení s počítačem přes sériový port, ale bylo by možné konfiguraci uskutečnit pomocí bezdratových signálů. Z tohoto důvodu by mohlo dojít k rozšíření funkčnosti aplikace. Je proto nutné, aby se v tomto případě nemusela velká část softwaru předělávat, ale pouze stačilo přidat další možný způsob konfigurace, bez rozbití jakékoli předešlé funkčnosti.

3.3 Návrh grafického rozhraní

Grafické rozhraní bude sloužit uživateli jako způsob ovládání aplikace. Rozhraní bude rozděleno na navigační lištu a dynamicky měnící se stránku, viz obrázek 3.2. Navigační lišta se skládá z navigačních tlačítek pro přechod mezi jednotlivými stránkami a tlačítek pro uložení konfigurace sítě, přepnutí jazyka a zobrazení informací o programu. Navigační tlačítka budou přepínat mezi čtyřmi různými stránkami, které se, společně se stránkou pro konfiguraci modemu, vysvětlují v následujících podkapitolách.

3.3.1 Úvodní strana

Spuštěním aplikace se uživatel octne na úvodní stránce, kterou lze vidět na obrázku 3.2. Na této stránce bude možnost volby mezi vytvořením nové konfigurace PLC sítě a modemů nebo načtením existující konfigurace z uloženého souboru. Pokud se uživatel na úvodní straně nenachází, ale chce například načíst odlišnou konfiguraci, lze tak učinit pomocí prvního tlačítka navigační lišty s tvarem domečku.

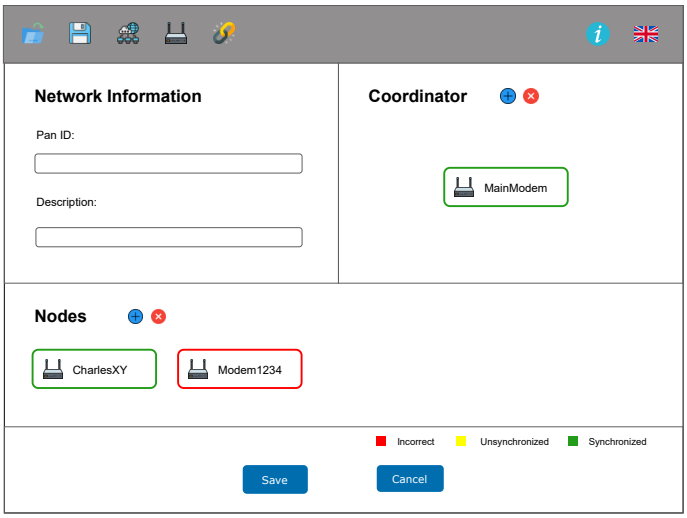


Obr. 3.2: Úvodní strana aplikace

3.3.2 Přehled aktuální sítě

Návrh stránky pro přehled aktuální sítě vidíme na obrázku 3.3. Tato stránka bude sloužit k rozvržení hierarchie sítě a jejímu popisu. Tlačítko s tvarem znaménka plus umožní uživateli přidat nový koordinátor nebo koncové zařízení. Tlačítka ve tvaru znaménka mínus přidají ke každému modemu zaškrtávací políčko, pomocí kterého si uživatel bude moci vybrat, jaký modem chce ze sítě odstranit. Barevné rámečky kolem jednotlivých modemů symbolizují stav ve kterém se nachází. Pouze modemy,

které mají kolem sebe zelený rámeček odpovídají fyzickým modemům připojeným k PLC síti. Ostatní jsou pouze makety, se kterými se skutečné modemy musí propojit. Kliknutím na obrázek modemu se změní stránka, o které pojednává následující podkapitola 3.3.3.

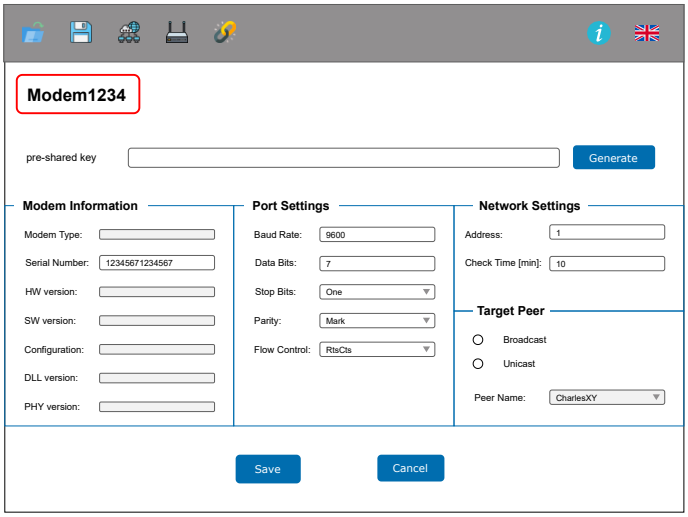


Obr. 3.3: Přehled modelu PLC sítě

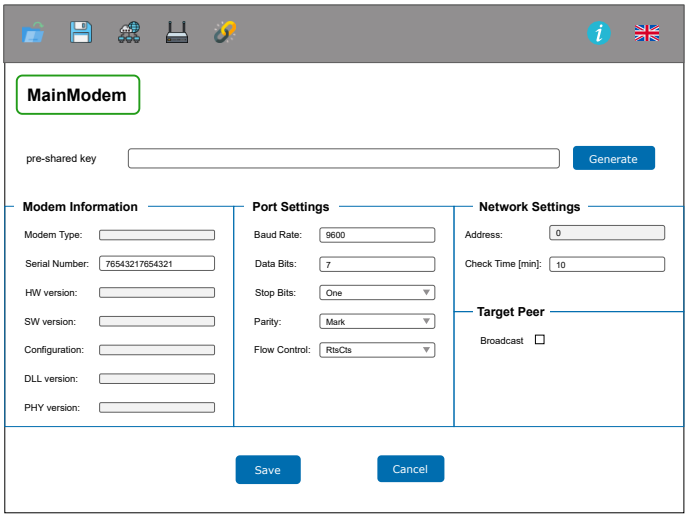
3.3.3 Konfigurace parametrů modemu

Přidáním nové makety modemu v přehledu sítě a následným kliknutím na její obrázek se bude zobrazovat stránka pro konfiguraci parametrů modemu, které je vidět na obrázku 3.4. Rámeček s textem *Modem1234* bude sloužit k nastavení jména, pomocí kterého lze jednotlivé modemy v přehledu sítě lépe rozpoznat. Většina nastavitelných parametrů již byla uvedena v kapitole 2.4. Parametry v části *Modem Information* nebude možno nastavit, protože to jsou hardwarové informace fyzického modemu. Jedinou výjimkou bude sériové číslo, kdy v případě shody sériového čísla makety se sériovým číslem skutečného modemu, budou při synchronizaci všechny parametry vyplněny automaticky.

Stránka konfigurace modemu s rolí koordinátora a koncového zařízení se v jednom parametru liší, tak jak lze vidět na obrázku 3.5. Koncové zařízení narozdíl od koordinátora má možnost posílat data po síti pouze jednomu dalšímu modemu (Unicast), oproti každému modemu v síti (Broadcast). Koordinátor tuto možnost nemá, protože vždy bude komunikovat se všemi modemy.



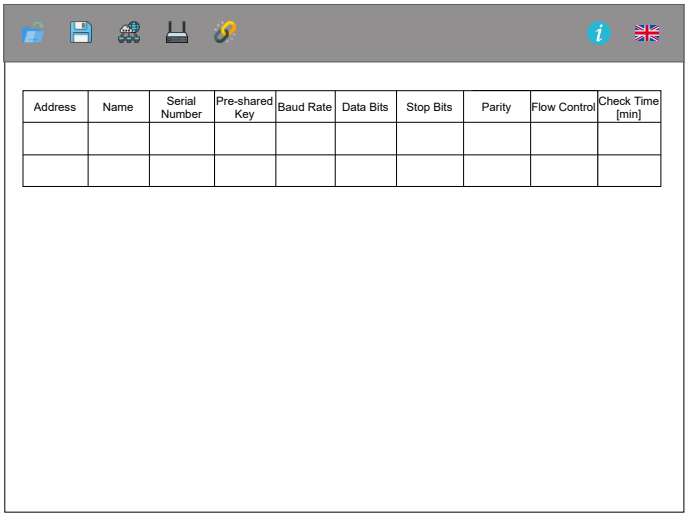
Obr. 3.4: Konfigurace parametrů koncového zařízení v aplikaci



Obr. 3.5: Konfigurace parametrů koordinátora v aplikaci

3.3.4 Přehled všech modemů

Pokud by uživatel potřeboval vidět všechny nakonfigurované parametry a stavy jednotlivých modemů, které jsou součástí sítě, poslouží mu k tomu stránka pro přehledu všech modemů, viz obrázek 3.6. Tímto dojde k urychlení vyhledávání požadovaného modemu například podle sériového čísla, bez potřeby zkoumání jedné konfigurační stránky za druhou.



Obr. 3.6: Přehled všech modemů a jejich stavů v aplikaci

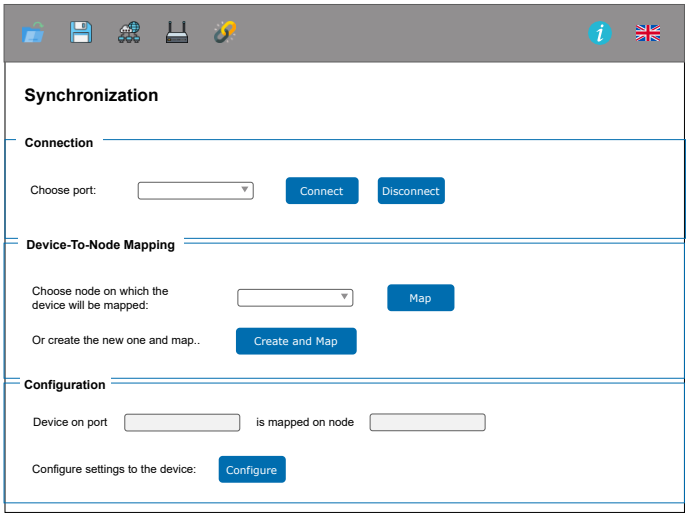
3.3.5 Synchronizace

Jakmile uživatel propojí počítač s fyzickým modemem a chce spojit aplikaci s realitou, využije stránka synchronizace, viz obrázek 3.7. Tato stránka umožňuje nahrát konfiguraci fyzického modemu do jedné z maket vytvořené uživatelem anebo do automaticky vytvořené makety koncového zařízení, která bude odpovídat připojenému modemu. K úpravě parametrů připojeného fyzického modemu slouží část *Configuration* na stránce synchronizace. Uživatel vybere v aplikaci maketu modemu, ze které chce konfiguraci přenést a pomocí tlačítka *Configure* se přepíšou nastavitelné parametry připojeného modemu.

3.4 Architektura a principy návrhu aplikace

Architektura aplikace je podstatnou součástí návrhu softwaru. Jedná se o obecný konstrukční plán, který určuje, jak by měly být jednotlivé části aplikace rozděleny a jak by tyto části spolu měly interagovat. Implementací architektury se zajistí přehled o částech aplikace a jejich funkcionalitě. V případě, že by byl software vyvíjen bez předem definovaného plánu, bude sice vyvinut rychleji, ale jakmile bude potřeba software rozšířit, modifikovat nebo testovat, výhoda rychlosti vývinu mizí. [11]

Při návrhu architektury aplikace je především dbáno na její udržitelnost a rozšiřitelnost. K dodržení těchto požadavků se budeme především řídit následujícími principy pro návrh a implementaci aplikace. Nejedná se o jediné principy, podle kterých bude aplikace navrhována a vyvíjena, ale domníváme se, že tato poslouží k porozumění jakým způsobem budou jednotlivé části aplikace rozděleny a spojeny.



Obr. 3.7: Synchronizační stránka pro propojení s modemy v aplikaci

3.4.1 Návrhové principy

Oddělení zodpovědností

Princip oddělení zodpovědností se zakládá na rozdělení softwaru dle druhu práce kterou vykonává. Například pokud se v naší aplikaci uživatel rozhodne odstranit zvolené modemy v síti, tak aby byl princip oddělení zodpovědností dodržen, logika aplikace zajišťující zobrazení a výběr modemů se oddělí, od logiky, která modemy odstraní z dat sítě. [12]

Aplikace se může sestavit na základě tohoto principu tak, že se oddělí základní obchodní chování aplikace od dat aplikace a logiky uživatelského rozhraní. Ideálně by do samostatných projektů měly být odděleny i obchodní pravidla a logika, které by neměly záviset na ostatních projektech aplikace. Tímto se zaručí snadné testování a možnost rozšíření obchodního modelu, bez závislosti na implementaci nižší úrovně. Toto oddělení do samostatných projektů je hlavním aspektem při využití vrstev v architekturách aplikací. [12]

Zapouzdření

Princip zapouzdření tvrdí, že by se jednotlivé komponenty a vrstvy aplikace měly od sebe vzájemně izolovat. To znamená, že by jedna komponenta měla být schopna ovlivnit vnitřní stav druhé, pouze za pomoci definovaných funkcí. Díky zapouzdření zajistíme volné propojení jednotlivých komponent, které budou moci měnit svou interní funkcionalitu, aniž by bylo potřeba měnit části aplikace, které tyto komponenty využívají. [12]

Inverze závislostí

Představte si, že existují dvě třídy, třída A a třída B. Třída A obsahuje členskou proměnnou třídy B, která ke své inicializaci využije proměnné, která byla předána jako parametr konstruktoru třídy A. Nyní lze říci, že třída A je závislá na třídě B, protože ke konstrukci instance třídy A, je nutné vytvořit objekt třídy B. Tento způsob pro vkládání závislosti je návrhovým vzorem s názvem injekce závislosti, v tomto případě za pomoci konstruktoru.

Ve výše uvedeném případě bude třída A velice úzce spjata s implementací třídy B. Inverze závislostí říká, že by se závislosti měly tvořit směrem k abstrakcím, nikoli ke konkrétním implementacím, čímž se dosáhne volnějších závislostí mezi třídami a lépe modifikovatelného kódu. Zkombinuje-li se tento princip společně s principem zapouzdření, bude možné vytvářet komponenty, jejichž implementace se budou moci kompletně nahradit alternativními implementacemi, aniž by bylo potřeba jakkoliv měnit části aplikace, které tyto komponenty využívají. [12]

Princip explicitní závislosti

Princip explicitních závislostí uvádí, že by třídy a metody měly explicitně vyžadovat všechny spolupracující objekty, které potřebují pro svoji korektní funkčnost. Jsou-li závislosti na spolupracujících objektech vytvářeny implicitně, znamená to, že jsou vytvářeny uvnitř metod nebo tříd, kde jsou požadovány, což vede k problematičtějšímu testování a složitější údržbě či modifikaci aplikace. [13]

3.4.2 Modely architektur aplikace

Nejjednoduššími modely aplikací jsou samostatné jednotky odpovídající spustitelnému souboru a obsahující veškerou logiku v jednom projektu. Tento přístup slouží velmi dobře interním a menším veřejným aplikacím. Jakmile však aplikace roste na komplexitě a velikosti, přibývá i počet souborů, které nejsou nijak logicky uspořádány a logika aplikace je rozptýlená napříč složkami a není jasné, které třídy ve kterých složkách by měly záviset na ostatních. Proto jsou aplikace děleny do více projektových řešení, kde každý projekt je logicky rozdělen do určité vrstvy aplikace. [14]

Vrstvy architektury jsou děleny na základě jejich povinností nebo starostí. Tento způsob následuje princip oddělení zodpovědností, uvedený v kapitole 3.4.1, umožňující udržet zdrojový kód aplikace organizovaný a přehledný, takže vývojáři mohou snadno najít, kde se určitá implementace funkcionality nachází. [14]

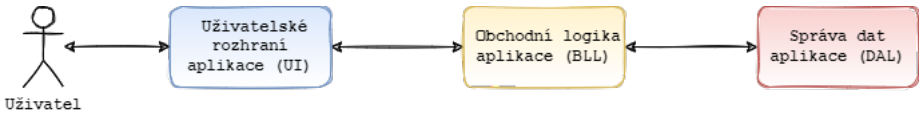
Organizace kódu však není jediným důvodem, proč by měla být aplikace stavěna dle architektur využívajících vrstev. Dělení kódu do vrstev nabízí mnohé výhody. Mezi některé patří:

- Dosažení principu zapouzdření, protože lze určit, které vrstvy mohou komunikovat s dalšími vrstvami.
- Nahrazení funkcí v aplikaci. Například použití jiného média pro synchronizaci modemů.
- Opakované užití běžnch funkcí na nízkých úrovní. [14]

Existuje více způsobů, jak lze logiku aplikace rozdělit do jednotlivých vrstev. V následujících podkapitolách je uveden nejběžnější způsob, takzvaná *N-Layer* architektura. Tato architektura však obsahuje určité problémy, a proto je jako další způsob uvedena takzvaná *Clean* architektura.

N-Layer architektura

Běžné rozdělení vrstev dle N-Layer architektury je zobrazeno na obrázku 3.8. Na obrázku lze vidět i způsob komunikace mezi jednotlivými vrstvami. Uživatel interaguje s vrstvou uživatelského rozhraní (UI), skrz kterou se zasílají požadavky do vrstvy s obchodní logikou (BLL). BLL poté může volat vrstvu spravující data aplikace (DAL) pro židost o přístup k datům. Neměl by nastat případ, kdy UI může přímo volat DAL a naopak. [14]



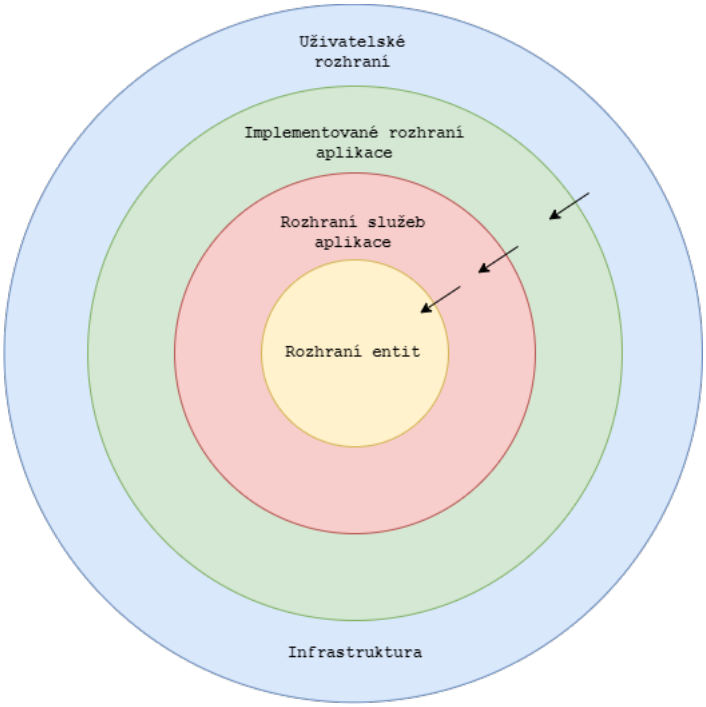
Obr. 3.8: Rozdělení vrstev dle N-Layer architektury. Vysvětlené zkratky: UI - user interface, BLL - business logic layer, DAL - data access layer

Jednou z nevýhod tohoto tradičního přístupu dělení vrstev je to, že závislosti v době kompilace aplikace směřují shora dolů. To znamená, že vrstva uživatelského rozhraní závisí na BLL, která závisí na DAL. Takže BLL, která většinou obsahuje nejdůležitější logiku aplikace závisí na podrobnostech implementace vrstvy spravující data aplikace. Testování obchodní logiky v takové architektuře je často obtížné a vyžaduje testovací databázi. K vyřešení tohoto problému lze využít princip inverze závislostí, jenž využívá čistá architektura. [14]

Čistá architektura

Čistá architektura umisťuje aplikační model a obchodní logiku do středu, utvářející jádro aplikace. Dochází k inverzi závislosti mezi obchodní logikou a správou dat nebo jiných obavách infrastruktury aplikace. Podrobnosti o infrastruktuře a její implementaci závisí na jádru aplikace, které je definováno pomocí abstrakcí nebo

rozhraní, jak lze vidět na obrázku 3.9. Na vnějšku architektury se nachází vrstvy uživatelského rozhraní, a infrastruktury aplikace, které závisí na aplikačním jádře, ale nikoli nutně na sobě. [14]



Obr. 3.9: Rozdělení vrstev aplikace dle čisté architektury. Šipky ukazují směr závislostí vrstev

Díky čisté architektuře bude vrstva uživatelského rozhraní v době kompilace pracovat s rozhraními, které jsou definované v jádru aplikace a v ideálním případě by neměla vědět o implemntačních typech definovaných ve vrstvě infrastruktury. V době běhu jsou ale tyto implementační typy potřeba ke spuštění aplikace, proto je nutné, aby byly v jádře aplikace přítomné a propojené s rozhraními za pomoci injekce závislostí. [14]

3.4.3 Organizace vrstev aplikace

Architektura aplikace bude využívat rozdělení do vrstev dle čisté architektury, přičemž každá vrstva bude složena z jednoho nebo více projektů.

Vrstva skládající se z rozhraní entit bude obsahovat rozhraní pro data aplikace. Bude se jednat o osobní síť, modemy v roli koncového zařízení a v roli koordinátora a klienta, který bude použit pro uchovávání dat při propojování mezi aplikací a fyzickým modemem.

V další vrstvě budou definovány rozhraní služeb aplikace, které budou implementovány o vrstvu výš a ve vrstvě infrastruktury. Jedná se o služby pro obchodní logiku a pro správu dat:

- Synchronizační služba, pro synchronizaci fyzického modemu s danou maketou.
- Konfigurační služba, která nastaví konfiguraci fyzického modemu dle makety.
- Služby, které budou využité ve vrstvě uživatelského rozhraní.
- Služba pro správu dat.
- Služba pro ukládání a načítání sítě do a z souboru.

Další vrstva bude obsahovat zmíněnou implementaci služeb pro obchodní logiku, ale také bude obsahovat implementaci rozhraní entit, které služby využívají pro jejich funkcionalitu.

Vrstva infrastruktury bude obsahovat implementaci služeb, které budou pracovat s daty sítě a také síť ukládat a načítat ze souborů. Opět budou muset být implementovány i entity, se kterými bude služba pro ukládání a načítání sítě pracovat.

Poslední vrstvou bude vrstva uživatelské rozhraní, která se bude zabývat veškerou komunikací s uživatelem skrz grafické rozhraní a na základě různých akcí bude využívat různé služby, které jsme definovaly ve vnitřních vrstvách. Uživatelské rozhraní je většinou implementováno za pomocí některého z aplikačních rámců, o kterých bude pojednávat kapitola 4.2.

4 Softwarové nástroje

Tato kapitola se zabývá výběrem vhodných nástrojů, které budou sloužit pro vývin aplikace. Uvádí, pro jakou platformu bude aplikace vyvinuta a v jakém vývojovém prostředí. Dále jsou popsány jednotlivé aplikační rámce, které se využívají pro vývin uživatelského rozhraní a je zdůvodněn výběr jednoho z nich. Následně jsou vysvětleny pomocné knihovny, které daný aplikační rámec využívá.

4.1 Platforma pro aplikaci

Výběr platformy, na které bude aplikace spustitelná, se domníváme, že je prvním rozhodnutím při implementaci aplikace. Existují tři druhy platformových aplikací:

- mobilní
- desktopová
- webová

Ke konfiguraci modemu se využívá propojení s počítačem, na kterém bude aplikace spuštěna. Modem se s počítačem propojí pomocí sériového portu, po kterém bude probíhat veškerá komunikace k nastavení parametrů. Operační systém počítačů je Windows. Domníváme se, že aplikace by mohla být využívána i v nepříznivých prostředích, kde nebude možnost připojení k internetu. Z těchto důvodů plyne využití desktopové platformy.

4.2 Aplikační rámce

Aplikační rámec (Framework) je softwarová knihovna, implementující základní strukturu aplikace. Využívají se především pro rychlejší a snazší vývoj softwaru s grafickým uživatelským rozhraním. [16]

Požadavky při výběru frameworku pro vývoj aplikace byly:

- Snadnost, tedy i rychlost vývoje aplikace
- Možnost separace uživatelského rozhraní od aplikační vrstvy neboli logiky aplikace
- Rozšiřitelnost a udržitelnost aplikace
- Podpora pro vývoj aplikace na platformu Windows
- Schopnost propojení aplikačního rozhraní pro ModemTec Serial Protocol

Pro tento seznam požadavků byly nalezeny následující aplikační rámce.

4.2.1 Qt

Qt je multiplatformní framework pro tvorbu uživatelských rozhraní a aplikací. Nabízí velké množství knihoven a nástrojů, které umožňují softwarovým vývojářům vyvíjet aplikace, bez nutného zaobírání se technickými detaily specifickými pro danou platformu. Pro vývin aplikací ve frameworku Qt se využívá jejího specifického vývojové prostředí *Qt Creator*, které v sobě má vestavěný editor kódu a nástroj podporující návrh grafických rozhraní buď za pomocí kódu anebo pomocí uživatelského rozhraní. [17]

Stejně jako aplikační rozhraní sériového protokolu společnosti ModemTec, framework Qt využívá programovací jazyk C++. Díky tomu, by propojení knihoven pro komunikaci s modemy a kódem aplikace nepředstavovalo žádný problém. Domníváme se, že C++ vyčnívá svou výkonností a rychlostí softwaru, avšak má strmější křivku učení, proto k využití jeho plného potenciálu je potřeba obětovat více času na naučení.

4.2.2 .NET

.NET je vývojová platforma, obsahující velké množství knihoven, pro vývoj různých druhů aplikací. K jejich vývoji lze využít více programátorských jazyků, mezi ně patří C#, Visual Basic a F#. .NET má čtyři různé varianty, označované jako implementace. Tři z implementací se dají použít pro vývoj desktopové aplikace na Windows, avšak jedinou implementací, která podporuje i ostatní operační systémy je technologie .NET 5 a její novější verze (.NET 6). [18] [19]

Při kompilaci kódu z jednoho z podporovaných jazyků, je kód zkompilován nejdříve do CIL (Common Intermediate Language), který není závislý na dané platformě. Následně je pro specifickou platformu CIL přeložen do strojového kódu za pomoci CLR (Common Language Runtime). [20]

Modul CLR (Common Language Runtime), spouští kód a poskytuje služby, které usnadňují proces vývoje. CLR usnadňuje návrh komponent a aplikací, jejichž objekty vzájemně komunikují napříč jazyky. Objekty napsané v různých jazycích mohou vzájemně komunikovat a jejich chování může být integrováno. Díky tomu lze vytvořit program v C++/CLI, což je varianta jazyka C++ pro .NET, mířený na CLR, za pomoci kterého by se daly využít knihovny společnosti ModemTec. [20] [21]

Windows Forms

Windows Forms nebo zkráceně WinForms je UI framework pro vytváření desktopových aplikací. Nabízí přívětivé prostředí pro návrh jednoduchého uživatelského rozhraní. K návrhu využívá vestavěného grafického uživatelského rozhraní ve Visual

Studiu, kde se uživatelské rozhraní aplikace vytváří pomocí přetažení požadovaných komponent ze sad nástrojů na požadované místo v oknu. Tento způsob vytváření aplikací je jak rychlý, tak jednoduchý.

Nevýhodou WinForms je omezenost na předem definované komponenty, jejich složitá modifikace a komplikovaný vývoj komplexnějších uživatelských rozhraních. Vkládání jednotlivých komponent je spojeno s absolutní pozicí v okně, takže v případě změny rozlišení se aplikace nepřizpůsobí. Způsob, kterým komunikuje uživatelské rozhraní s logikou aplikace, je úzce svázaný, a tak nenabízí takové možnosti jejich oddělení. [22]

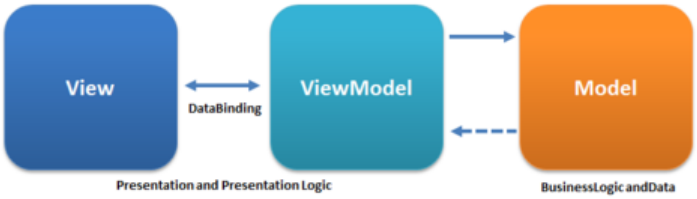
Windows Presentation Foundation (WPF)

Dalším aplikačním rámcem, co .NET nabízí je WPF. Jedná se o nejpopulárnější UI framework pro vývoj desktopových aplikací. Umožňuje podobný způsob návrhu uživatelského rozhraní jako WinForms, ale není limitován na běžné ovládací prvky postavené na základě Windows. Toto umožňuje přepisovat staré a vytvářet vlastní ovládací prvky jako třeba ikonové tlačítko, které bude využito v naší aplikaci. [24] [23]

WPF oproti WinForms umožňuje návrh uživatelského rozhraní nejen pomocí přetahování komponent do okna, ale především využívá deklarativního značkovacího jazyka XAML (eXtensible Application Markup Language), který je založen na XML (eXtensible Markup Language) a následuje nebo rozšiřuje jeho strukturální pravidla. XAML slouží k deklaraci jednotlivých instancí typů objektů, které definují rozložení a obsah uživatelského rozhraní aplikace. Při kompilaci okna definovaného za pomoci XAMLu, se společně kompilují dvě částí. První částí je značkovací soubor psaný pomocí XAMLu. Částí druhou je takzvaný *code-behind* soubor neboli soubor obsahující kód psaný v programovacím jazyku pro danou aplikaci. Tento soubor většinou obsahuje způsob jakým bude okno inicializováno nebo logiku objektů, definovaných ve značkovací části. Výsledkem kompilace jsou značkově definované objekty, ke kterým je připojen kód z code-behind části. [25] [26] [27]

Implementace XAMLu v .NETu využívá abstraktní třídy MarkupExtension jako základ pro všechny rozšíření značek, se kterými jsou jednotlivé typy definovány. Konkrétní rozšíření značek, které WPF implementuje, jsou často určena k poskytování prostředků, které umožní referencovat ostatní existující objekty. Například jednoduché datové vazby se ve WPF provádí zadáním *{Binding}* na místě, kde jsou normálně uvedeny hodnoty pro danou vlastnost. Díky těmto rozšířením značek je možné implementovat sjednocené styly pro stejné objekty v aplikaci, nebo použít třídy, které mění barvu rámečku na základě hodnoty proměnné objektu, se kterým je daný typ propojen. [26]

Při vývoji UI aplikací ve WPF se převážně využívá návrhového vzoru MVVM (*Model-View-ViewModel*), který odděluje logiku aplikace od ovládání uživatelského rozhraní, viz obrázek 4.1. Část *View* by měla obsahovat všechny třídy sloužící k návrhu uživatelského rozhraní, která jsou v aplikaci používána. Návrh rozhraní se provádí v XAMLu a v code-behind části jsou implementovány rozšíření jednotlivých prvků, které implementují logiku zobrazování uživatelského rozhraní. Každá třída pro zobrazování obsahuje vlastnost *DataContext*, která slouží jako zprostředkovatel dat pro datové vazby *DataBinding*. Objekt, který se zapisuje do vlastnosti DataContext je *ViewModel*. ViewModel je třída, která slouží k propojení dat aplikace s uživatelským rozhraním a dodává způsoby, pomocí kterých je logika aplikace volána z části uživatelského rozhraní. Poslední část *Model* je tvořena daty a logikou aplikace, které jsou propůjčovány ViewModelu. ViewModel tak obsahuje data, která byly získány z dat aplikace, ale implementovává své proměnné na základě jejich potřebnosti v uživatelském rozhraní. Dalo by se říct, že ViewModel je zprostředkovatel mezi skutečnými daty aplikace a daty, které jsou zpřístupněny uživatelskému rozhraní pomocí DataBindingu s proměnnými třídy ViewModel. Stejně tak jako zprostředkovatel způsobů, pomocí kterých může uživatelské rozhraní vyvolávat logiku aplikace a provádět tak skutečné změny v datech aplikace.



Obr. 4.1: Návrhový vzor MVVM (Model-View-ViewModel) [28]

Framework, ve kterém bude aplikace implementována, je .NET WPF za použití jazyka C#. Tato volba plyne především ze snahy, o co nejefktivnější a nejvíce korektní implementaci aplikace s ohledem na její rozšiřitelnost a modifikaci. Protože je WPF nejpopulárnějším frameworkem, na internetu existuje velké množství informací zabývající se vývojem aplikací právě v tomto frameworku. Na základě toho se domníváme, že i přes složitější návrh aplikace oproti WinForms, kde se není potřeba učit další jazyk pro návrh uživatelského rozhraní, bude aplikace rychleji a lépe implementována. Zároveň osobním cílem implementace aplikace je naučit se vyvíjet aplikace pro společnost ModemTec, a protože se domníváme, že WPF je relevantnější v oblasti vývoje aplikací, tak bude výhodnější vytvořit aplikaci právě za pomoci WPF.

4.3 Vývojové prostředí

Aplikace bude vyvíjena na operačním systému Windows. Nejrozsáhlejším vývojovým prostředím pro tento operační systém je Microsoft Visual Studio. Stejně jako framework .NET je vytvořen společnostní Microsoft. Díky tomu, obsahuje, námi zvolený framework WPF, podpůrné nástroje pro ulehčení vývoje aplikací v tomto prostředí. Z tohoto důvodu bylo zvoleno Microsoft Visual Studio jako vývojové prostředí pro vytvoření aplikace.

4.4 NuGet balíčky

NuGet je nezbytný nástroj vyvinutý společností Microsoft a .NET Foundation pro prostředí .NET. Jedná se o bezplatné, open source rozšíření pro vývojové prostředí Visual Studio. Slouží jako správce balíčků, pomocí kterého, mohou vývojáři vytvářet, sdílet a používat užitečné části kódu. Tyto části kódu jsou balíčky, které lze snadno přidat do svých projektů a pouze zavolat jejich funkcionalitu ve svém kódu. [29]

5 Implementace aplikace

V této kapitole se popisují funkce implementované aplikace a vysvětluje se způsob jakým byly implementovány a na jakém principu fungují. Nejdříve je vysvětlen způsob pomocí kterého jsou dodávány závislosti určitým objektům při jejich instanciaci, následně se vysvětlují různé funkcionality aplikace jako je překlad textu v aplikaci do cizího jazyka nebo validace zadaných hodnot ze strany uživatele.

5.1 Spuštění aplikace

Výchozí bod pro start aplikace je v souboru *App.xaml*, skládající se ze dvou souborů. Značkovacího souboru, viz obrázek 5.1, a code-behind souboru, viz obrázek 5.2. Třída App v code-behind souboru, odpovídá třídě *x:Class="SemSet.UI.App"* v souboru značkovacím. Díky této referenci jsou mezi sebou propojeny a mohou být vzájemně implementovány různé funkcionality prvků.

Při spuštění aplikace dojde k zavolání metod *SetDefaultCulture* a *DependencyInjectionSetup*. Metoda *SetDefaultCulture* slouží k inicializaci členské proměnné *CurrentCulture* třídy *TranslationSource*, pomocí které je proveden překlad aplikace do jednotlivých jazyků, které jsou aplikací podporovány (v dosavadní verzi pouze angličtina a čeština). Podrobnější informace o způsobu překladu obsahuje následující kapitola 5.2.1.

Metoda *DependencyInjectionSetup* inicializuje a nakonfiguruje třídu *ServiceCollection*, která se zpřístupní při stažení NuGet balíčku *Dependency Injection*. Tato třída umožňuje vytvořit kontejner všech závislostí, které mohou třídy při instanciaci vyžadovat, a pokud by jejich závislosti obsahoval, tak je dodat, aniž by musely být konstruktoru předány jako parametr. Součástí konfigurace třídy ServiceCollection je tedy definice pro jaké závislosti mají být jaké instanciace tříd dodávány a jakým způsobem se mají vytvářet. Zda má existovat v celé aplikaci pouze jedna jejich instance (Singleton) anebo se při každém vyžádání závislosti má vytvořit instance nová (Transient). Například každá služba obsahující logiku aplikace je v kontejneru vytvořena pouze jednou a při potřebě jejich použití jsou tyto instance dodány objektu, který je vyžaduje.

Nakonec je vyvolána událost *Startup*, vidíme na obrázku 5.1, která je zachycena a obsloužena metodou *OnStartup*. Metoda OnStartup vytvoří a inicializuje objekt třídy *MainWindow*. Tato třída reprezentuje okno grafického rozhraní, skrz které uživatel ovládá aplikaci. Při zavolání metody *Show* je okno zobrazeno a další chování aplikace již závisí na uživateli.

Součástí značkovacího souboru App.xaml je také přidání stylů, které slouží pro sjednocení stylů stejných prvků uživatelských rozhraní, k definici nových prvků (jako

ikonové tlačítko), definování různých konvertorů (barva na základě hodnoty) nebo k implementování stylu, jakým bude zobrazována validace.

```
<Application x:Class="SemSet.UI.App"
     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     Startup="OnStartup">

    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="Resources/Styles.xaml" />
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
```

Obr. 5.1: Obsah značkovacího souboru App.xaml

5.2 Uživatelské rozhraní

Grafické uživatelské rozhraní je definováno objektem MainWindow, které vytváří a zobrazuje okno, se kterým uživatel ovládá aplikaci. Rozhraní je realizováno pomocí komponenty *Grid*, která slouží k rozdělení okna na daný počet řádků a sloupců, u kterých lze definovat, jak velkou část okna budou zabírat. Okno aplikace je v tomto případě rozdělena na dva řádky.

První řádek tvoří navigační lišta, která je realizována pomocí komponenty *StackPanel*, jenž řadí prvky horizontálně. Ikony navigační lišty jsou ikonová tlačítka implementována jako komponenty *Button*, pro které byl vytvořen vlastní styl, aby místo klasického vzhledu tlačítka, zobrazovaly komponentu *Image*, která zobrazuje obrázek ze zadané cesty. Navigační lišta je skrz uživatelské rozhraní neměnná a stálá, bez ohledu na obsah ve zbylé části.

Druhý řádek je oproti prvnímu řádku proměnný. Na základě stisknutého tlačítka v navigační liště se mění obsah druhého řádku a uživateli se tak zpřístupní měnit stránky uživatelského rozhraní. Toto chování umožňuje komponenta *ContentControl*, která při dosazení jednotlivé stránky do její vlastnosti *Content*, zobrazí její obsah. Stránky jsou implementované pomocí třídy *UserControl*, která umožnuje vytvářet celky uživatelského rozhraní, které jsou následně dosazeny do hlavního okna.

```
public partial class App : Application
{
    private ServiceProvider _serviceProvider;
    public App()
    {
        SetDefaultCulture();
        DependencyInjectionSetup();
    }
    private static void SetDefaultCulture()
    {
        TranslationSource.Instance.CurrentCulture = new("cs-CZ");
    }
    private void DependencyInjectionSetup()
    {
        ServiceCollection services = new();
        ConfigureServices(services);
        _serviceProvider = services.BuildServiceProvider();
    }
    private void OnStartup(object sender, StartupEventArgs e)
    {
        MainWindow mainWindow = _serviceProvider.GetService<MainWindow>();
        mainWindow.Show();
    }
```

Obr. 5.2: Necelý obsah code-behind souboru App.xaml.cs. Na obrázku chybí implementace metody ConfigureServices, protože pro pochopení funkcionality není potřeba.

Hlavní okno uživatelského rozhraní využívá třídu *MainWindowViewModel* pro DataBinding mezi aplikací a uživatelským rozhraním. Mezi konkrétní data ViewModelu patří cesta k obrázku vlajky reprezentující aktuální jazyk aplikace, instance aktuální implementace třídy UserControl, která se zobrazuje, a implementaci tříd příkazů, které jsou vázány s jednotlivými tlačítky pro vykonání daných akcí při jejich stistknutí. Způsob změn stránek a funkcionality tlačítek je popsán v následujících podkapitolách 5.2.1 a 5.2.2.

5.2.1 Překlad jazyka

Překlad všech prvků uživatelského rozhraní je proveden za pomocí WPF lokalizace. Členská proměnná *CurrentCulture* v Singleton[1] třídě *TranslationSource*, je třída typu *CultureInfo*[2]. Jedná se o vestavěnou třídu ve frameworku .NET, která na základě vloženého názvu (například cs-CZ) dokáže získat informace pro danou kulturu. Zároveň se s její pomocí může lokalizovat soubor, pro překlad aplikace do aktuálního jazyka. Zdrojové texty pro překlad jsou součástí souborů *Resources.*-*.resx*. Pro každý podporovaný jazyk musí existovat právě jeden tento soubor, kdy jednotlivé záznamy v souborech jsou jako záznamy v hash tabulce. To znamená, že pro každou hodnotu (přeložený text), existuje klíč, pomocí kterého se k hodnotě lze dostat.

Prvky uživatelského rozhraní, u kterých je potřeba získat přeložený text pro stávající jazyk, nemají text pevně daný. Místo toho je použito XAML rozšíření pro lokalizaci, které funguje na podobném principu jako DataBinding. Rozdíl je v tom, že pro zdroj hodnoty prvku se neuvádí proměnná z DataContextu zobrazovací třídy, ale dodává se klíč se specifikací, že se jedná o lokalizaci. Poté se na základě nastavené kultury v CurrentCulture proměnné, vybere soubor, odpovídající kultuře v proměnné a pomocí klíče se najde přeložený text, který se dodá prvku uživatelského rozhraní.

5.2.2 Změna stránek

Zmáčknutím ikonového tlačítka v navigační liště (kromě tlačítka s vlaječkou) dojde ke změně stránky, která je uživateli zobrazena. Postup změny se ale skládá z více kroků, kterými jsou:

- Vytvoření nového ViewModelu pro daný UserControl.
- Instanciace nového UserControlu, které je předán vytvořený ViewModel skrz konstruktor.
- Vyvolání události s parametrem obsahujícím vytvořený UserControl.
- Zachycení a obsloužení události v třídě MainWindow.

Postupu pro vytváření nejdříve ViewModelu a až poté třídy pro zobrazování (v tomto případě UserControl), se říká *ViewModel first* a měla by být preferovaným postupem pro dodávání ViewModelu do třídy UserControl. Existuje však i obrácený postup s názvem *View first*, který nepředává závislost na ViewModelu skrz konstruktor, ale s každou instanciací třídy pro zobrazování, je vytvořen nový ViewModel.

Při vytváření nového ViewModelu je využit návrhový vzor s názvem *Builder*. V tomto návrhovém vzoru je definováno rozhraní *IViewModelBuilder*, které obsahuje metody pro inicializaci ViewModelu a získání finálního výtvoru. Implementace tohoto rozhraní existuje pro každý ViewModel, který je zobrazovacími třídami vyžadován.

[1] https://refactoring.guru/design-patterns/singleton

[2] https://docs.microsoft.com/en-us/dotnet/api/system.globalization.cultureinfo?view=net-6.0

Díky tomu nemusí třída ViewModelu obsahovat závislosti na služby, které potřebuje pro své vytvoření, ale nepotřebuje pro správu funkcionality uživatelského rozhraní. [30]

Některé ViewModely vyžadují získání aktuálních dat aplikace, které jsou skrz ně zobrazovány v uživatelském rozhraní. Například u konfigurací parametrů modemů obsahuje ViewModel stejné proměnné jako entity modemů. Takže při zobrazení stránky konfigurace parametrů modemu je nutné proměnné entit získat z dat aplikace a jejich hodnoty zapsat do odpovídajících proměnných ve ViewModelu. Zároveň se při ukládání změněné konfigurace ve ViewModelu, musí v entitách modemů aktualizovat změněné parametry. Ke vzájemnému mapování je využit Nuget balíček *AutoMapper*, který na základě profilu (nastavení) automaticky najde proměnné, které si odpovídají, a jejich hodnoty zapíše z jedné entity do druhé.

Jakmile je vytvořen ViewModel pro daný UserControl, který definuje vzhled stránky, dojde k vyvolání události *ChangeDynamicContent*. Vyvolaná událost s sebou nese parametry, které budou použity pro změnu dynamické části hlavního okna. Prvím parametrem je nově vytvořená třída UserControl, která při své konstrukci přiřadí své členské proměnné DataContext předaný ViewModel a dojde k její inicializaci, která propojí code-behind část se značkovacím souborem. Druhým parametrem je název stránky, který se použije pro změnu nadpisu aplikace.

Vyvolaná událost je následně zachycena ve třídě MainWindow, kde se zavolá metoda pro její obsloužení. V této metodě dojde ke změně členské proměnné *CurrentUserControl* a *PageTitle* ve ViewModelu, který třída MainWindow používá pro svůj DataContext. A protože jsou tyto proměnné ve značkovém souboru třídy MainWindow propojeny pomocí DataBindingu s vlastnostmi *Content* prvku *ContentControl* a *Title* prvku *Window*, dojde k jejich aktualizaci a zobrazí se nová stránka s novým nadpisem.

5.2.3 Příkazy tlačítek

Každá komponenta *Button* má dva způsoby implementování činnosti, jenž se vykoná po stisknutí tlačítka. Prvním je volání událostí, které jsou zachyceny a obslouženy v code-behind části pro dané rozhraní. Druhý způsob je sice komplikovanější, ale umožňuje lepší oddělení logiky od oken aplikace a zároveň je podporován ve frameworku WPF. Tento způsob je implementován v aplikaci a jedná se o využití behaviorálního návrhového vzoru *Command*.

Návrhový vzor *Command* spočívá ve vytvoření rozhraní, které bude obsahovat metodu *Execute*, jenž bude volána při stisknutí tlačítka. Rozhraní se následně implementuje pro daný příkaz, vytvoří se jeho instance (ať už přímo anebo pomocí kontejneru závislostí), která je tlačítku dodána. Ve frameworku WPF obsahuje kom-

ponenta Button vlastnost *Command*. Do této vlastnosti je pomocí Data Bindingu vložena instance třídy, implementující rozhraní *ICommand*, kde je konkrétní metoda Execute realizována. Data Binding je opět proveden přiřazením ViewModelu, který obsahuje implementaci specifického Commandu, do proměnné DataContext, pro dané zobrazení. Další výhodou využití rozhraní *ICommand* ve frameworku WPF, je jeho rozšíření o metodu *CanExecute*, která rozhoduje o tom, zda je tlačítko povolené a je možné ho stisknout, nebo povolené není a stisknout se nedá. Implementací metody CanExecute, lze tak lépe ochránit aplikaci před neočekávaným nebo nechtěným vstupem od uživatele, které by mohlo způsobit nežádané chování aplikace. [30]

5.2.4 Vysvětlivky pro uživatele

Každý prvek tlačítka nebo textového pole v uživatelském rozhraní má vlastnost *ToolTip*, která zobrazuje vysvětlivku pro daný prvek, v případě, kdy je na něj najeto myší. Text vysvětlivky využívá lokalizace, takže pro každý prvek, který využívá vlastnost ToolTip existuje klíč, pro který je definován překlad. Domníváme se, že díky těmto vysvětlivkám jednotlivých prvků, by se mělo uživatelské rozhraní stát přívětivějším a snazším na porozumění a použití.

5.2.5 Validace vstupů

Jakmile je pro konfiguraci nutno získat vstup ze strany uživatele, používají se komponenty *ComboBox* nebo *TextBox*. Komponenta *ComboBox* nabízí výběr z předem definovaných hodnot, z toho důvodu by nemělo být potřeba kontrolovat jaká data byla uživatelem zadána. V případě, kdy dostává uživatel volnost v zadávání hodnot, je využita komponenta *TextBox*, u které je však potřeba zkontrolovat zda zadaná data jsou validní, aby při jejich uložení nedošlo k rozbití funkcionality aplikace.

Implementace validace je postavena na rozhraní s názvem *INotifyDataErrorInfo*, které je součástí frameworku WPF. Pokud je toto rozhraní implementováno ve ViewModelu, který obsahuje členské proměnné, jejichž hodnota bude získávána pomocí DataBindingu z komponenty TextBox ve třídě UserControl, tak kdykoliv kdy dojde ke změně hodnoty, bude zavolán *Setter* (metoda, která zapisuje získanou hodnotu do proměnné) pro danou členskou proměnnou, jehož obsahem je jak zápis hodnoty do proměnné, tak i kontrola, zda zadaná hodnota splňuje pravidla pro daný typ. Pokud pravidla splněna nejsou, přidá se do seznamu validační chyba. Po nastavení členské proměnné ViewModelu, u které došlo ke změně hodnoty skrz uživatelské rozhraní, se zavolá metoda, která zjišťuje, zda jméno této proměnné není obsáhlé v seznamu validačních chyb. Pokud ano, dojde ke změně zobrazení komponenty TextBox a vypíše se zpráva, která odpovídá danému porušení validačních pravidel.

5.2.6 Ukládání změn

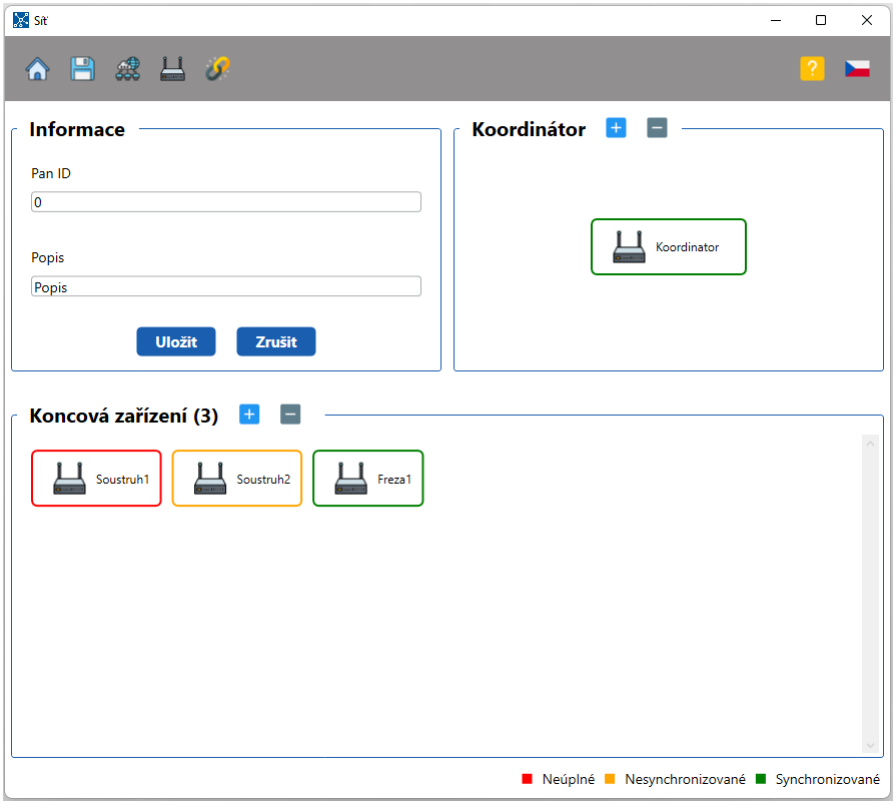
Jak již bylo zmíněno, tak úprava hodnot v uživatelském rozhraní, se projeví ve změně členských proměnných odpovídajícího ViewModelu. To znamená, že pro zapsání nastavených hodnot do dat aplikace, je potřeba využít službu, která uvedená data zapíše do dat aplikace, čímž dojde k jejich aktualizaci. Tento postup se může zdát jako zbytečný krok navíc, ale díky němu je možné implementovat navrácení změn, kdy v případě omylu ze strany uživatele, se pomocí stisknutí tlačítka *Zrušit*, namapují aktuální data aplikace na proměnné zobrazené ViewModelem a uživatel tak získá zpět původní data.

5.2.7 Přehled sítě

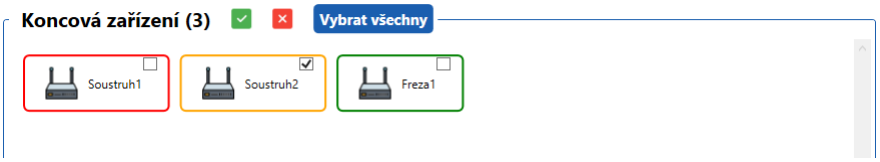
Stránka pro přehled sítě využívá k zobrazení modemů komponenty *ItemsControl*, do které je pomocí DataBindingu z ViewModelu vložena kolekce obsahující koncová zařízení a koordinátora v síti. Pro jejich zobrazení není potřeba znát jejich konfiguraci, avšak stačí pouze jejich identifikátor a stav ve kterém se nachází, protože to jsou jediné informace z entit modemů, které je potřeba uživateli na této stránce zobrazit. Z toho důvodu je vytvořen vlastní typ datového objektu (třída *ModemName*), do kterého jsou potřebná data z entit v síti namapována, a navíc implementuje proměnnou *IsChecked*, která je využita pro výběr modemů pomocí komponenty *CheckBox*, které chce uživatel ze sítě odstranit, viz obrázek 5.4. Součástí třídy *ModemName* je také jméno modemu, které je dodáno k dispozici jako parametr metody *Execute*, která je vyvolána při kliknutí na ikonové tlačítko modemu. Pomocí tohoto jména lze následně najít objekt modemu, který je uchováván v datech sítě a na základě jeho dat vytvořit nový ViewModel, který se použije jako DataContext pro stránku zobrazující podrobné informace o konfiguraci modemu.

Při odstraňování nebo přidávání modemů do sítě pomocí ikonových tlačítek plus a mínus, jsou nové modemy vytvářeny přímo v datech aplikace. V případě, kdy se má přidat nový modem, využívá se návrhového vzoru s názvem *Prototype*. Tento vzor umožňuje vytvářet nové objekty pouze se znalostí rozhraní, které implementují. Díky tomu, nemusí ve třídě, která obsahuje logiku pro vytváření nových objektů, existovat závislost na jejich konkrétní implementaci. [30]

Využití návrhového vzoru *Prototype* spočívá v definici abstraktní třídy prototypu s metodou *Clone*, která je pro každý typ objektu, který chceme vytvářet, implementována tak, aby vytvořila konkrétní objekt s inicializovanými proměnnými dle nastavené konfigurace. Následně lze tyto implementace prototypů zabalit do třídy *PrototyService*, která obsahuje metody pro produkování kopií, kterým jsou upraveny obecné parametry na parametry konkrétnější, jako například nové, neexistující jméno modemu v síti.



Obr. 5.3: Implementace stránky pro přehled sítě



Obr. 5.4: Zobrazení zaškrtávacích políček pro výběr modemů k jejich odstranění ze sítě

V případě, kdy by uživatel chtěl změnit roli modemu z koordinátora na koncové zařízení nebo naopak, není nutné modem mazat a vytvářet nový. K tomuto účelu slouží implementovaná funkcionalita *Drag and Drop*, kdy stačí modem v roli koordinátora (nebo koncového zařízení) uchopit a přetáhnout do části stránky obsahující modemy v roli koncového zařízení. Komponenta, pomocí které je tato funkcionalita realizována se nazývá *ScrollViewer*, kdy za pomoci knihoven frameworku WPF, se umožní zaznamenání této interakce, která při jejím úspěšném provedení zavolá metody, starající se o logiku interakce. V tomto případě namapování proměnných z entity modemu v roli koordinátora na entitu modemu v roli koncového zařízení, který je přidán do sítě jako nový modem v roli koncového zařízení s konfigurací modemu v

roli staré, přičemž modem, který byl přetažen se ze sítě odstraní.

5.3 Logika aplikace

Docílení správné funkcionality aplikace záviselo na korektní implementaci knihoven společnosti ModemTec, které obsahují API (Application Programming Interface) pro komunikaci s fyzickými modemy přes sériový port. Knihovny jsou psány v jazyce C++, proto bylo nutné vytvořit CLR projekty, které umožní zabalení kódu z knihoven, do .NET tříd za pomoci jazyka C++/CLI. Kompilací CLR projektů se vytvoří DLL (Dynamic Link Library) soubory, které obsahují dříve zabalený kód, avšak nyní použitelný pro referenci v projektech aplikace jako běžné třídy implementované v jazyce C#.

Součástí přílohy této práce je *.sln* soubor, který umožňuje otevřít řešení aplikace ve Visual Studiu. V originální verzi aplikace jsou CLR projekty součástí řešení, ale z důvodu zachování firemního tajemství společnosti ModemTec, jsou v přiloženém řešení projekty CLR již zkompilované do souborů *.dll*.

5.3.1 Ukládání konfigurací sítě

Běžný způsob implementace čisté architektury předpokládá přítomnost databáze, ve které jsou data uchovávána, a služby, která umožní přístup k datům i z ostatních projektů. Z důvodu neexistence databáze jsou ve vytvořené aplikaci data uchovávána v paměti běžícího programu, konkrétněji ve třídě *NetworkRepository*. Tato třída uchovává informace o nastavení vytvořené sítě, jejíž součástí jsou i informace o konfiguracích modemů. K přístupu a modifikaci dat jsou využívané metody této třídy, takže se zároveň chová jako služba i jako databáze.

Načítání a ukládání dat mimo běh aplikace zajišťuje serializace a deserializace objektů do souborů typu *.json*, která je zpřístupněna při stažení NuGet balíčku *Newtonsoft.Json* (příklad uložené sítě lze vidět na obrázku 5.5). Avšak pro korektní uložení nebo načtení dat je nutné použít entity, které mají definovaný způsob pro serializaci či deserializaci jejich proměnných, které jsou tvořeny ze složitější typů nebo kolekcí. Objekty, které jsou uloženy jako data reprezentující konfiguraci sítě, jsou ale vytvářeny ve vrstvě s implementacemi služeb, která má definované vlastní entity, takže se typy entit pro ukládání do souborů nerovnají typům entit, které jsou v aplikaci instanciovány. Z toho důvodu je při serializaci nutné namapovat (pomocí *AutoMapperu*) „non-json“ entity na json entity, které se následně serializují a uloží do .json souboru. V případě načítání sítě z .json souboru je postup stejný akorát reverzovaný.

```
{
  "$type": "SemSet.Data.Entities.JsonNetwork, SemSet.Data",
  "coordinator": {
    "$type": "SemSet.Data.Entities.JsonCoordinator, SemSet.Data",
    "address": 0,
    "checkTime": 0,
    "id": 3,
    "isBroadcasting": true,
    "modemInfo": {
      "$type": "SemSet.Data.Entities.JsonModemInfo, SemSet.Data",
      "configuration": null,
      "dllVersion": null,
      "hardwareVersion": null,
      "modemType": null,
      "phyVersion": null,
      "serialNumber": "134",
      "softwareVersion": null
    },
    "name": "NewCoordinator",
    "portSettings": {
      "$type": "SemSet.Data.Entities.JsonPortSettings, SemSet.Data",
      "baudRate": 0,
      "dataBits": 0,
      "flowControl": "None",
      "parityBit": "None",
      "stopBits": "One"
    },
    "preSharedKey": "742313526CB734D078CF71D82B78FDC8",
    "state": 1
  },
  "description": "Network description",
  "endDevices": [],
  "panId": 0
```


Obr. 5.5: Příklad uložené sítě, v *.json* souboru, obsahující jeden koordinátor a žádné koncové zařízení

6 Otestování aplikace

Tato kapitola popisuje testování logiky aplikace pomocí takzvaných *Unit* testů a popsání nástrojů k tomu využitých. Dále je popsáno testování funkcionality aplikace s reálnými modemy a poukazuje se na možný stav aplikace, který není optimální. Závěrem kapitoly je zhodnocena funkčnost aplikace na základě praktického využití.

6.1 Testování logiky aplikace

Testování logiky aplikace je provedeno pomocí takzvaných *Unit* testů, kdy jsou testovány ty nejmenší části kódu, které mohou být v aplikaci izolovány. Většinou se jedná o ověření správné funkcionality metod jednotlivých tříd, které jsou v aplikaci využívány. [31]

Unit testy jsou implementovány za pomoci zdarma, open-source Unit testovacího frameworku *xUnit*, který slouží k definicí a spouštění testů. Dalším nástrojem je *AutoFixture*, který umožňuje instanciaciovat třídy nebo *Mock* objekty, které jsou v testech využívány. Mock objekty slouží k vytvoření falešných instanciací objektů, u kterých lze nastavit, jakým způsobem by se v testech měly zachovat, a díky tomu není potřeba implementovat skutečnou závislost, kterou by bylo pracné nakonfigurovat nebo přímo nepraktické. Pro implementace Mock objektů je použita knihovna *Moq*.

Testování funkcionality synchronizace a konfigurace modemů není možné, protože jsou pro jejich funkčnost vyžadovány fyzické modemy, které nelze namockovat.

6.2 Testování funkcionality s modemy

Během finální části vývoje aplikace byla funkcionalita konfigurace a synchronizace modemů skrz sériový port opakovaně testovaná. Byly vyzkoušeny oba způsoby pro synchronizaci fyzického modemu, jak způsob s automatickým vytvořením nové makety modemu za pomoci tlačítka *Create and Load*, při jehož stisknutí dojde k vytvoření nové makety modemu, v roli koncového zařízení, a zapíšou se do ní parametry připojeného modemu, tak způsob, u kterého je nejdříve maketa modemu v síti vytvořena, nakonfiguruje se její sériové číslo, které odpovídá skutečnému modelu modemu, se kterým má být sesynchronizována, a poté se na stránce synchronizace tato maketa vybere z nabídky komponenty *ComboBox* a stisknutím tlačítka *Load* se nahraje konfigurace fyzického modemu do jemu odpovídající makety modemu v aplikaci. Zároveň se vyzkoušela konfigurace parametrů skutečného modemu z makety v aplikaci, kdy se po synchronizace makety s modemem, změnilo nastavení parametrů u makety a pomocí tlačítka *Configure* na stránce synchronizace se nastavily parametry fyzického modemu na základě vybrané makety.

Všechny testy funkcionality proběhly, při dodržení správného postupu, bez problému. I při nedodržení správného postupu nedošlo k rozbití funkčnosti modemu či aplikace, pouze při pokusu o připojení se k modemu, v případě, kdy modem není připojen k danému portu, ale je připojeno jiné zařízení, dojde k nerozeznání ze strany aplikace, zda došlo k propojení s modemem společnosti ModemTec, nebo s nějakým jiným zařízením, protože se pro kontrolu korektního připojení využívá logická proměnná *IsConnected* u třídy *Device* (součást knihovny ModemTec), která pro zjištění stavu připojení k modemu, kontroluje, zda byla sériová komunikace na daném portu otevřena nebo nebyla. Kvůli tomu může dojít aplikace do stavu, kdy hlásí, že modem je připojen. Tento stav umožní uživateli přístup ke tlačítkům konfigurace a synchronizace s modemem, a i když bude při jejich stisknutí hlášena chyba, může to být pro uživatele matoucí.

Nevýhodou při testování s fyzickými modemy byl jejich počet, protože při využití aplikace v praxi se předpokládá konfigurace celé sítě, po které mezi sebou modemy komunikují. Avšak při testování funkcionality aplikace, byl využit pouze modem jeden, takže nebyly otestované případy, ke kterým by mohlo docházet při připojení více modemů. Je možné namítnout, že aplikace s více modemy být testována nemusí, protože se především korektní konfigurace modemu, a pokud ta proběhla bez problému, může se předpokládat, že konfigurace každého modemu v síti, proběhne bez problému. V tomto případě by měla komunikace mezi modemy po síti, fungovat bez problému.

Na konci listopadu roku 2021 byla dokončena implementace první verze aplikace, která byla ihned zaslána zákazníkovi pro využití v praxi. V době psaní této práce (květen 2022), nebyla podána žádná stížnost směrem na funkčnost aplikace, takže lze předpokládat, že aplikace funguje bez přítomnosti jakékoli závažné chyby. Domníváme se, že i kdyby se nějaká chyba v aplikaci objevila, nebude problém chybu opravit, z důvodu vhodného návrhu aplikace.

Závěr

Cílem bakalářské práce bylo vytvoření aplikace pro konfiguraci modemů společnosti ModemTec s ohledem na snadnost jejího použití a s ohledem na možnost budoucí modifikace nebo rozšíření funkcionality aplikace. Dalším cílem bylo obeznámit se s různými způsoby vytváření softwarových aplikací a nabytí znalostí o principech stavění aplikace.

Výsledkem bakalářské práce je funkční aplikace, která umožňuje konfiguraci parametrů modemu společnosti ModemTec. Zda se aplikaci povedlo implementovat pro jednoduché použití, je subjektivní pohled uživatele. Objektivně se tohoto cíle snažilo dosáhnout pomocí implementace vysvětlivek pro všechny prvky uživatelského rozhraní, ikonovými tlačítky, rozdělením uživatelského rozhraní na stránky, u kterých je jasná jejich funkce nebo i možností změny jazyka aplikace z češtiny na angličtinu.

Aplikace byla navržena způsobem, který spočívá v práci s rozhraními namísto konkrétními implementacemi. Díky tomu je vytvořená volná závislost mezi jednotlivými částmi aplikace, což umožňuje jednoduché implementace nových rozšíření funkcionality aplikace, jako třeba konfigurace modemů bez potřeby propojení s počítačem pomocí sériového portu, ale konfigurace modemů za pomoci bezdrátových signálů.

Na základě vedlejších cílů bylo nabyto hlubší porozumění o způsobu fungování frameworku WPF, který umožňuje postavit velkou řadu desktopových aplikací s ohledem na oddělení uživatelského rozhraní od dat a logiky aplikace. Zároveň bylo dosaženo lepšího porozumění softwarového návrhu aplikací a principů podle kterých by měly být aplikace implementovány k dosažení jednoduché modifikovatelnosti a rozšiřitelnosti.

Literatura

[1] Alotaibi, Ibrahim, Mohammed A. Abido, Muhammad Khalid, and Andrey V. Savkin. *A Comprehensive Review of Recent Advances in Smart Grids: A Sustainable Future with Renewable Energy Resources* Energies 13, no. 23: 6269. Dostupné z: https://doi.org/10.3390/en13236269

[2] MAJUMDER, A a Jr CAFFERY J. *Power line communications. IEEE potentials* [online]. New York: IEEE, 2004, 23(4), 4-8 [cit. 2022-01-03]. ISSN 0278-6648. Dostupné z: doi:10.1109/MP.2004.1343222

[3] NDJIONGUE, A.R a H.C FERREIRA. *Power line communications (PLC) technology: More than 20 years of intense research. Transactions on emerging telecommunications technologies* [online]. HOBOKEN: WILEY, 2019, 30(7) [cit. 2022-01-03]. ISSN 2161-3915. Dostupné z: doi:10.1002/ett.3575

[4] BENEŠL, Lukáš. *Inteligentní systém pro dálkový sběr dat po PLC.* Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií, 2018, 51 listů : ilustrace + 1 CD-ROM

[5] ŠEBESTA, Ondřej. *Powerline komunikace pro řízení LED světel.* Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií, 2020.

[6] Abdul Mannan, D.K.Saxena, Mahroosh Banday *A Study on Power Line Communication* - published at: "International Journal of Scientific and Research Publications (IJSRP), Volume 4, Issue 7, July 2014 Edition". ISSN 2250-3153. Dostupné z: doi:10.29322

[7] Interní dokumentace firmy Modemtec

[8] PEŇA, Eric and Mary Grace LEGASPI. *UART: A hardware communication protocol understanding universal asynchronous receiver/transmitterUART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter* | Analog Devices [online]. [cit. 2022-01-03]. Dostupné z: https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

[9] Silicon Labs. *AN0059.1: UART Flow Control* [online]. [cit. 2022-01-03]. Dostupné z: https://www.silabs.com/documents/public/application-notes/an0059.1-uart-flow-control.pdf

[10] esds. *A Practical Guide to Differentiate Unicast, Broadcast & Multicast.* [online] [cit. 2022-01-03]. Dostupné z: https://www.esds.co.in/blog/difference-between-unicast-broadcast-and-multicast

[11] Ferguson Kevin. *Application Architecture* [online] [cit. 2022-05-04]. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/application-architecture

[12] Microsoft Documentation. *Architectural principles* [online] [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles

[13] DevIQ. *Explicit Dependencies Principle* [online] [cit. 2022-05-04]. Dostupné z: https://deviq.com/principles/explicit-dependencies-principle

[14] Microsoft Documentation. *Common web application architectures* [online] [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

[15] Digital Skynet. *Desktop App vs Web App: Comparative Analysis* [online] [cit. 2022-01-03]. Dostupné z: https://digitalskynet.com/blog/Desktop-App-vs-Web-App-Comparative-Analysis

[16] techopedia. *Application Framework* [online] [cit. 2022-01-03]. Dostupné z: https://www.techopedia.com/definition/6005/application-framework

[17] ZHI ENG, Lee. *Hands-On GUI Programming with C++ and Qt5: Build stunning cross-platform applications and widgets with the most powerful GUI framework* 1. edice, Packt Publishing, 2018. ISBN 978-1788397827.

[18] Microsoft Documentation. *What is .NET? Introduction and overview* [online] [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/core/introduction

[19] Microsoft Documentation. *.NET implementations* [online] [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/fundamentals/implementations

[20] Microsoft Documentation. *Common Language Runtime (CLR) overview* [online] [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/clr

[21] Microsoft Documentation. *Walkthrough: Compile a C++/CLI program that targets the CLR in Visual Studio* [online] [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/cpp/dotnet/walkthrough-compiling-a-cpp-program-that-targets-the-clr-in-visual-studio?view=msvc-170#prerequisites

[22] GeeksforGeeks. *Difference between WPF and WinForms* [online] [cit. 2022-05-04]. Dostupné z: https://www.geeksforgeeks.org/difference-between-wpf-and-winforms/

[23] WPF Tutorial. *WPF vs. WinForms* [online] [cit. 2022-05-04]. Dostupné z: https://www.wpf-tutorial.com/about-wpf/wpf-vs-winforms/

[24] Microsoft Documentation. *What is Windows Presentation Foundation (WPF)?* [online] [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2022

[25] Microsoft Documentation. *XAML overview in WPF* [online] [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/advanced/xaml-overview?view=netframeworkdesktop-4.8

[26] Microsoft Documentation. *XAML Syntax In Detail* [online] [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/advanced/xaml-syntax-in-detail?view=netframeworkdesktop-4.8

[27] Microsoft Documentation. *Code-Behind and XAML in WPF* [online] [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/advanced/code-behind-and-xaml-in-wpf?view=netframeworkdesktop-4.8

[28] Wikipedia. *Model–view–viewmodel* [online] [cit. 2022-05-04]. Dostupné z: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel

[29] Microsoft Documentation. *An introduction to NuGet.* [online] [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/nuget/what-is-nuget

[30] SHVETS, Alexander. *Dive Into Design Patterns* Anglická edice, PacRefactoring.Guru, 2018.

[31] SmartBear. *What Is Unit Testing?* [online] [cit. 2022-05-04]. Dostupné z: https://smartbear.com/learn/automated-testing/what-is-unit-testing/

Seznam symbolů a zkratek

PLC	Power Line Communication
CENELEC	Evropský výbor pro normalizaci v elektrotechnice
UART	Universal Asynchronous Receiver Transmitter
RTS	Request to Send
CTS	Clear to Send
CLR	Common Language Runtime
WPF	Windows Presentation Foundation
CLR	Common Language Runtime
API	Application Programming Interface

A Obsah paměťového média

- **xtichy28.pdf** - PDF soubor této práce
- Složka **doxyDocumentation** - Doxygen dokumentace zdrojového kódu aplikace, generována v HTML
- Komprimovaný soubor **xtichy28.zip** - řešení bakalářské práce
  - **src** - zdrojový kód aplikace
  - **lib** - .dll soubory reprezentující knihovnu společnosti ModemTec
  - **tests** - testy logiky aplikace
  - **SemSet.sln** - spustitelné řešení ve Visual Studiu 2019