UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

Software pro správu a konfiguraci Z-Wave zařízení

Bc. Michal Lauterbach

Diplomová práce

2017

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Michal Lauterbach**
Osobní číslo:	**I15216**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Software pro správu a konfiguraci Z-Wave zařízení**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem práce je vytvoření softwaru pro správu a konfiguraci zařízení využívajících bezdrátový komunikační protokol Z-Wave.
Aplikace bude umožňovat přidávání a odebírání zařízení z centrálního prvku sítě, tzv. kontroléru. Dále bude umožňovat tvorbu a rušení vazeb mezi zařízeními při využití asociačních skupin protokolu Z-Wave. U každého zařízení bude možno identifikovat jeho typ, případně také výrobce a obchodní označení. Dále bude umožněno zjišťovat a nastavovat parametry zařízení a také zasílat příkazy pro jeho zapnutí a vypnutí. Dostupné konfigurační parametry zařízení budou načteny z poskytnutého XML souboru, eventuálně bude umožněn jejich manuální výběr pomocí kódu parametru, datového typu a jeho hodnoty. Aplikace bude naprogramována v programovacím jazyce C#, výstupem bude spustitelná aplikace pro systém Microsoft Windows. Kromě kvalitní implementace by měl být kladen důraz také na uživatelskou přívětivost a snadné použití aplikace.

Rozsah grafických prací:

Rozsah pracovní zprávy: **cca 65 stran**

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

***NAGEL, Christian. C# 2008: Programujeme profesionálně. Brno: Computer Press, 2009. ISBN 978-80-251-2401-7.**
***PAETZ, Christian. Z-wave Basics: Remote Control in Smart Homes. S.l.: Create Space, 2013. ISBN 978-1490537368.**
***MARTIN, Robert C. Čistý kód: návrhové vzory, refaktorování, testování a další techniky agilního programování. Brno: Computer Press, 2009. ISBN 978-80-251-2285-3.**

Vedoucí diplomové práce: **Ing. Jiří Paar**
Steinel Technik

Konzultant diplomové práce: **Ing. Zdeněk Šilar, Ph.D.**
Katedra informačních technologií

Datum zadání diplomové práce: **31. října 2016**

Termín odevzdání diplomové práce: **17. května 2017**

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

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

V Pardubicích dne 15. listopadu 2016

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.
Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.
Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 10. května 2017

Michal Lauterbach

Poděkování

Tímto bych rád poděkoval vedoucímu práce panu Ing. Jiřímu Paarovi za odborné vedení a užitečné rady v průběhu práce. Rád bych také poděkoval firmě Steinel Technik, kde jsem tuto diplomovou práci realizoval.

Děkuji také mé rodině a všem, kteří mě podporovali po celou dobu studia.

Anotace

Cílem práce je vytvoření softwaru pro správu a konfiguraci zařízení využívajících bezdrátový komunikační protokol Z-Wave. V teoretické části je nejprve představeno několik protokolů, které jsou v praxi využívány k účelům domácí automatizace. Dále je zde detailněji představen protokol Z-Wave a jeho specifika. Část práce je také věnována programovacímu jazyku C# a některým jeho funkcionalitám. V praktické části je popsána samotná implementace aplikace a její použití. Z hlediska implementace je zde popsán význam a funkce jednotlivých tříd aplikace a také jsou zde vysvětleny principy klíčových funkcí aplikace. Výsledná aplikace je naprogramována v programovacím jazyce C#, výstupem je spustitelná aplikace pro systém Microsoft Windows.

Klíčová slova

Z-Wave, C#, konfigurace, automatizace, protokol, aplikace

Title

Software for management and configuration of Z-Wave devices

Annotation

The aim of this thesis is creation of the software for management and configuration of devices based on Z-Wave wireless communication protocol. Theoretical part describes several protocols which are used for home automation. Also there is deeply described Z-Wave protocol and its specifics. Afterwards there is described C# programming language and some of its functionalities. Practical part of the thesis describes implementation of the application and its usage. There is described meaning and function of individual classes and key functions of the application. Application is programmed in C# programming language with executable application for Microsoft Windows as a result.

Keywords

Z-Wave, C#, configuration, automation, protocol, application

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

ACK	Acknowledge
BCI	Batibus Club International
CLR	Common Language Runtime
EHB	European Home System
EIB	European Installation Bus
ETS	Engineering Tool Software
GUI	Graphical User Interface
ID	Identity
IEEE	Institute of Electrical and Electronics Engineers
KNX	Konnex Networks
LINQ	Language Integrated Query
MAC	Media Access Control
PAN	Personal Area Network
PIR	Passive Infrared Sensor
PLINQ	Parallel Language Integrated Query
SQL	Structured Query Language
XML	Extensible Markup Language

Úvod

Domácí automatizace je jedním z trendů dnešní doby. Na trhu existuje velké množství více či méně známých výrobců elektronických zařízení, která mohou být k domácí automatizaci využita. Kromě toho existuje i relativně velké množství protokolů, které tato zařízení mohou používat. Jedním z nejznámějších, a v současné době často využívaných protokolů, je protokol Z-Wave. Právě vývoji aplikace pro správu a konfiguraci zařízení, která jsou na této technologii založená, je věnována tato diplomová práce.

Jak již bylo zmíněno, kromě Z-Wave je v praxi využíváno i mnoho dalších protokolů. První část práce tak čtenářům některé z nich představí. Následující kapitola se pak bude detailněji věnovat samotnému protokolu Z-Wave. Nejprve bude krátce popsána jeho historie a propojení s některými technologickými firmami. Poté bude protokol rozebrán z technologického hlediska – využívaných frekvencí, významu jednotlivých vrstev protokolu apod. Dále budou popsány třídy používaných zařízení a také třídy příkazů, se kterými tato zařízení pracují.

Následuje představení jazyka C#, který byl použit pro vývoj zmíněné aplikace. Nejprve bude představen samotný jazyk a platforma Microsoft .NET Framework. Následuje popis základních stavebních prvků aplikací napsaných v tomto jazyce. Poté bude popsán framework Windows Forms jako jedna z možností pro tvorbu okenních aplikací ve zmíněném jazyce. Vysvětlena bude také práce s výjimkami, technologie LINQ a zpracování XML souborů.

Poté je popsána samotná implementace aplikace. Nejprve je čtenáři představena struktura tříd, které tato aplikace využívá. Následně jsou detailněji popsány některé stěžejní části a funkcionality aplikace. V kapitole je také popsáno, jak byla aplikace testována. Poslední část se pak věnuje popisu použití aplikace a také principu demo režimu určeného pro prezentaci aplikace bez nutnosti připojení USB kontroléru.

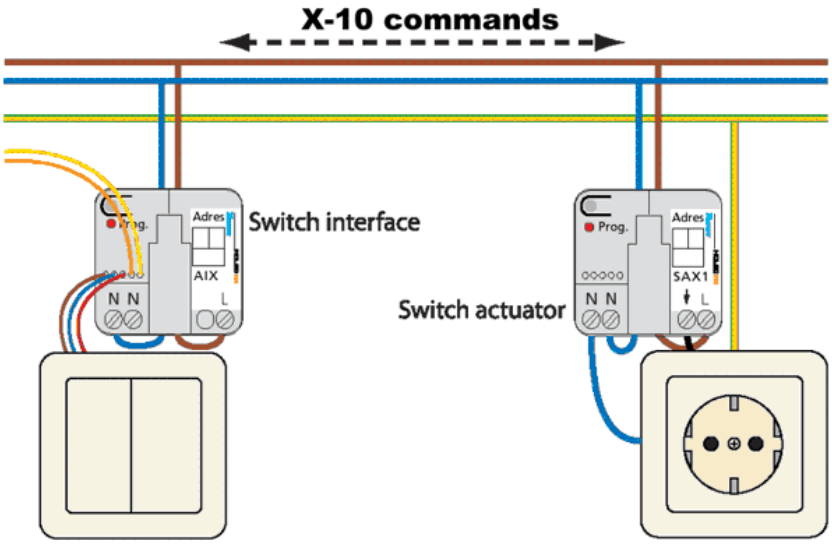
1 Protokoly pro domácí automatizaci

Existuje veliké množství protokolů používaných pro domácí automatizaci. Některé z nich používá jen úzká skupina výrobců, některé z nich jsou naopak považovány za celosvětový standard a jsou v zařízeních využívány velkým množstvím výrobců. V následujících kapitolách jsou představeny některé z těch nejznámějších a nejpoužívanějších.

1.1 X10

Protokol X10 byl vyvinut už v roce 1975 společností Pico Electronics, mimo jiné známou výrobou první kalkulačky, která k fungování využívala jen jeden čip. Jedná se o první komerčně rozšířený protokol pro domácí automatizaci. Velkému rozšíření zařízení pomohlo spojení s velkým americkým prodejcem elektroniky, společností Radio Shack. Zařízení pracující s tímto protokolem komunikují po klasickém rozvodu elektrické energie, jak je znázorněno na obrázku 1. Zařízení mezi sebou posílají dva typy signálů – adresy a příkazy. Adresy jsou kombinací písmen z rozsahu A-P a čísel z rozsahu 1-16. Celkem je tak možno adresovat až 256 zařízení. Na jedné adrese může být dostupných i více zařízení, ta jsou pak ovládána najednou. Kromě základních příkazů pro zapnutí a vypnutí je možno posílat také příkazy pro ztlumení, nebo naopak zesílení jasu světla apod. Výhodou protokolu X10 byla dostupnost zařízení a jejich cena. Nevýhodou je pak rychlost a nespolehlivost komunikace (způsobená velkým rušením na napěťových rozvodech) a také složitější instalace pro běžného uživatele. [1], [2]

Vzhledem ke stáří a nástupu modernějších řešení je tento protokol používán méně než v minulosti. Nicméně i dnes je na trhu k dostání relativně velké množství zařízení pracujících právě na protokolu X10. Protokol je také často využíván v zabezpečovacích systémech pro domácnost.



Obrázek 1: Schéma zapojení zařízení s protokolem X10. [1]

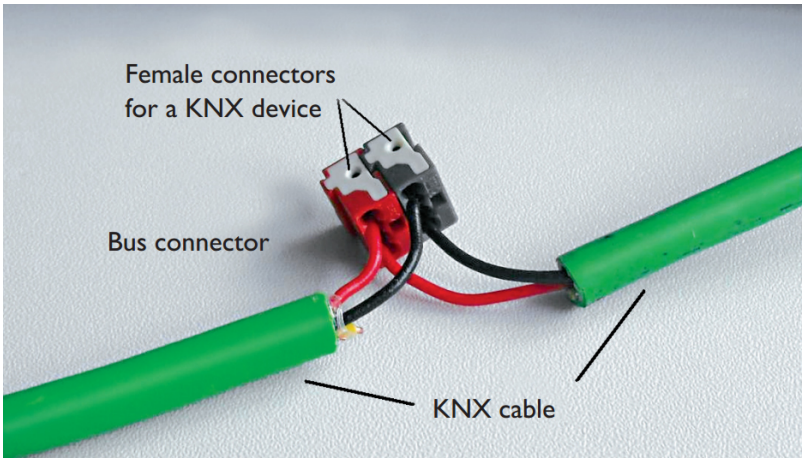
1.2 KNX

Asociace KNX vznikla v roce 1997 spojením tří evropských asociací, které byly založeny již na počátku devadesátých let. Jde o francouzskou asociaci BCI, která stála za systémem Batibus, holandskou EHS Association, která vyvinula systém EHS a belgickou EIB Association, stojící za systémem EIB. Právě systém EIB se stal základem pro protokol KNX, který byl v Evropě standardizován v roce 2002. Původní systém EIB byl doplněn o přenosová média a nové konfigurační mechanismy, které byly původně vyvinuty právě pro Batibus a EHS. V roce 2006 pak došlo k uznání standardu na mezinárodní úrovni. Nyní je KNX jediným celosvětově uznávaným standardem pro automatizaci domů a budov. [3] Instalace KNX se skládá z následujících částí:

- napájecí zdroj 29 V,
- sběrnicové vedení – kroucená dvoulinka,
- snímače – například tlačítka, snímače teploty, vlhkosti apod.,
- aktory – např. relé, ventily topení, lampy apod.

Zařízení se ke KNX sběrnici připojují pomocí speciálních konektorů, nazývaných *bus terminal*. Tyto konektory umožňují připojení nebo odpojení zařízení, aniž by byla jakkoliv

ovlivněna funkčnost zařízení okolních. Právě to je jedním z největších benefitů technologie KNX. Ukázka zmíněného konektoru společně s typicky používanou kabeláží je ukázána na obrázku 2 [4]



Obrázek 2: Konektor a kabeláž pro technologii KNX. [4]

Kromě fyzické instalace jednotlivých částí je třeba také jejich následná konfigurace pomocí ETS softwaru. Tu provádí nejčastěji instalační technik. Aby byla konfigurace kompletní, je třeba zadat jedinečné adresy jednotlivých zařízení, nastavit jejich parametry a přiřadit skupinové adresy, které slouží k provázání funkcí snímačů a akčních členů. [5]

1.3 ZigBee

Zařízení fungující na podobném protokolu, jako je ZigBee, byla vyvíjena už na konci devadesátých let. Samotná specifikace protokolu byla představena v roce 2005. Protokol ZigBee byl původně vyvinut pro použití v sítích PAN (Personal Area Networks), podobně jako Bluetooth. Je založen na frekvenční komunikaci s nízkým výkonem, konkrétně na standardu IEEE 802.15.4. Pracuje na frekvencích 868 MHz a 2,4 GHz. Na nižší frekvenci umožňuje rychlost přenosu 20 kbit/s, na vyšší až 250 kbit/s. Maximální dosah zařízení využívajících tento protokol je cca 70 m.

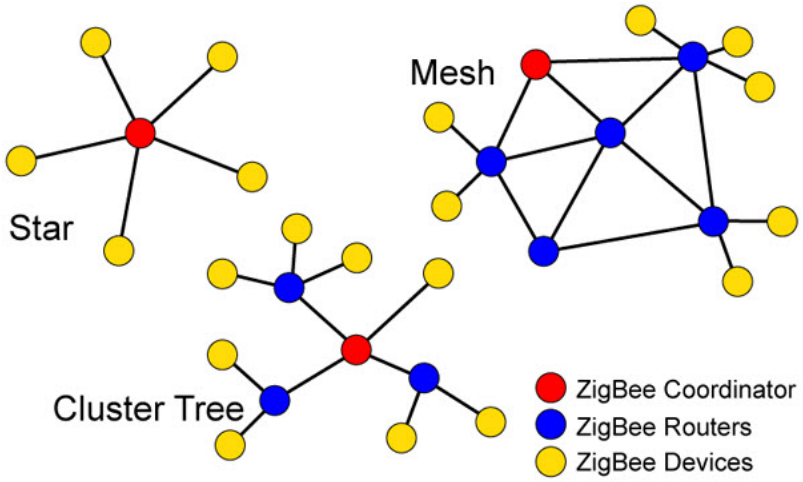
Mezi výhody protokolu patří šifrovaná komunikace, velká dostupnost produktů na trhu, jednoduchost instalace, nízká spotřeba (možnost použití bateriově napájených zařízení)

a velký dosah. Nevýhodou je pak nízká šířka frekvenčního pásma a s ní související nižší přenosové rychlosti. [6]

Protokol umožňuje tvorbu sítí, ve kterých může být využíváno přeposílání dat na vzdálenější uzly, a tím zvýšení dosahu. V ZigBee sítích jsou rozlišovány tři druhy zařízení:

- coordinator (koordinátor) – stará se o samotnou tvorbu sítě (přidávání a odebírání prvků, přiřazování adres apod.), v každé síti musí být alespoň jeden,
- router (směrovač) – umožňuje přeposílání dat na další uzly,
- end device (koncové zařízení) – neumožňuje přeposílání dat (typicky bateriová zařízení). [7]

Pro tvorbu ZigBee sítí je nejčastěji využívána topologie *Star* (česky hvězda), *Mesh* (česky síť), nebo *Cluster Tree* (česky strom). Jak tyto topologie vypadají, je ukázáno na obrázku 3. Červeně je znázorněn typ prvku koordinátor, modře směrovač a žlutě koncová zařízení.



Obrázek 3: Používané topologie sítí u protokolu ZigBee. [8]

1.4 Proprietární protokoly

Mnoho výrobců se vydává cestou vývoje a použití vlastních komunikačních protokolů namísto užití univerzálních standardizovaných řešení. Některé protokoly již umožňují obousměrnou spolehlivou komunikaci mezi zařízeními. Ta je většinou zajištěna pomocí paketů potvrzujících správnost přijatých dat (tzv. ACK pakety).

Největší nevýhodou využití vlastních protokolů je velmi nízká univerzálnost. Typicky jsou tyto protokoly implementovány jen v zařízeních jednoho výrobce, případně několika málo spolupracujících výrobců. S tím také souvisí problémy s implementací plné automatizace, ke které je často potřeba využití i produktů jiných výrobců. Dalším problémem může být dlouhodobá využitelnost daných zařízení. Nezřídka se stává, že výrobce v nových zařízeních využije novou verzi svého proprietárního protokolu, který však není kompatibilní se starší verzí. Výhodou zařízení s vlastním komunikačním protokolem je pak snadnost jejich použití a často také jejich nižší cena. [9]

2 Protokol Z-Wave

V této kapitole je podrobněji rozebrán protokol Z-Wave. Nejprve je krátce popsána jeho historie a současnost. Dále jsou specifikovány použité frekvence a je zde popsán vrstvový model protokolu. Zbytek kapitoly je pak věnován třídám zařízení a třídám příkazů.

2.1 Historie a současnost

Protokol Z-Wave byl původně vyvíjen dánskou společností Zen-Sys. Tato společnost byla založena již na konci devadesátých let a první generaci Z-Wave čipu uvedla na trh v roce 2003. Tento čip byl kombinací standardního mikrokontroléru značky Atmel s transceiverem[1]. Prvními většími zákazníky byly firmy v USA, kde díky protokolu X10, vyvíjeném od roku 1975, byla domácí automatizace již celkem známá. Prvním větším evropským výrobcem, využívajícím protokol Z-Wave, byl dánský výrobce termostatů Danfoss. V roce 2005 začalo docházet k masivnímu rozšíření na americkém a evropském trhu a následně i v Asii. [9]

Velkým milníkem byl odkup společnosti velkým americkým výrobcem čipů, společností Sigma Designs. V roce 2005 bylo také založeno konsorcium Z-Wave Alliance. To zaštiťuje výrobce elektroniky, kteří využívají protokol Z-Wave ve svých produktech. V současné době je členem přes 450 společností. Kromě propagace organizuje také různá školení k rozšíření znalostí a povědomí o této technologii a zajišťuje také certifikaci jednotlivých zařízení. Pokud je zařízení certifikováno, může na něm být použito logo Z-Wave, které je vyobrazeno níže. [9], [10]

Obrázek 4: Logo Z-Wave. [11]

[1] Transceiver je síťový prvek kombinující vysílač a přijímač signálu v jedné součástce. Název je odvozen z anglických slov transmitter (vysílač) a receiver (přijímač).

2.2 Technologie

Z-Wave využívá bezlicenční frekvenční pásma. V Evropě pracuje na frekvenci 868,4 MHz. Ta spadá do pásma 868–870 MHz, kde je umožněno využívání zařízení krátkého dosahu. Jednou z motivací využití této frekvence je cena. Jak bylo uvedeno dříve, jedná se o bezlicenční pásmo, není zde tedy třeba platit licenci k provozu bezdrátových zařízení. Dalším důvodem je přeplněnost pásma 2,4 GHz, na kterém operují např. Wi-Fi sítě a Bluetooth zařízení. Jednou z výhod plynoucí z nižší frekvence je také spolehlivost přenosu. V tabulce níže jsou uvedeny frekvence používané pro protokol Z-Wave v různých státech světa. [9], [12]

Region	**Frekvenční pásmo**
Austrálie	921,4 MHz
Brazílie	921,4 MHz
Čína	868,4 MHz
Evropa	868,4 MHz
Hong Kong	919,8 MHz
Indie	865,2 MHz
Japonsko	951–956 MHz 922–926 MHz
Jižní Afrika	868,4 MHz
Malajsie	868,1 MHz
Mexiko	908,4 MHz
Nový Zéland	921,4 MHz
Rusko	869 MHz
Singapur	868,4 MHz
Spojené arabské emiráty	868,4 MHz
USA a Kanada	908,4 MHz

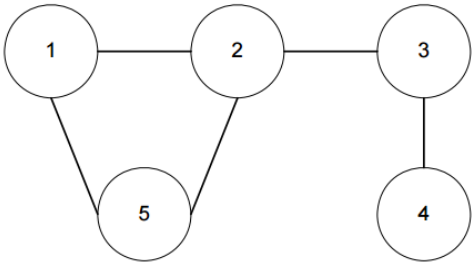
Tabulka 1: Frekvenční pásma používaná v jednotlivých regionech [9]

2.3 Vrstvový model

Protokol Z-Wave je složen celkem z pěti vrstev. Souhrnně jsou tyto vrstvy nazývány jako *Z-Wave Protocol Stack*. Dle pořadí od nejvyšší úrovně po nejnižší se skládá z vrstvy aplikační, síťové, transportní, vrstvy zajišťující správný přístup k přenosovému médiu (nazývané zkráceně MAC vrstva) a vrstvy fyzické.

Aplikační vrstva se stará o datový obsah (anglicky payload) rámce. V případě rámce Z-Wave je tímto obsahem třída příkazu, typ příkazu a případně také jeho parametry. Detailnějšímu popisu tříd a typů příkazů je věnována kapitola 2.5.

Síťová vrstva se stará o správné směrování rámců a také o skenování síťové topologie a na něm závislém aktualizování směrovacích tabulek uložených v kontroléru. Směrovací tabulky mají formu bitového pole. Existující přímé spojení mezi dvěma zařízeními značí jednička, neexistující spojení naopak nula. Ukázka síťové topologie a odpovídající směrovací tabulky je vyobrazena na obrázku níže.



	1	2	3	4	5	6
1	0	1	0	0	1	0
2	1	0	1	0	1	0
3	0	1	0	1	0	0
4	0	0	1	0	0	0
5	1	1	0	0	0	0
6	0	0	0	0	0	0

Obrázek 5: Síťová topologie a odpovídající směrovací tabulka. [14]

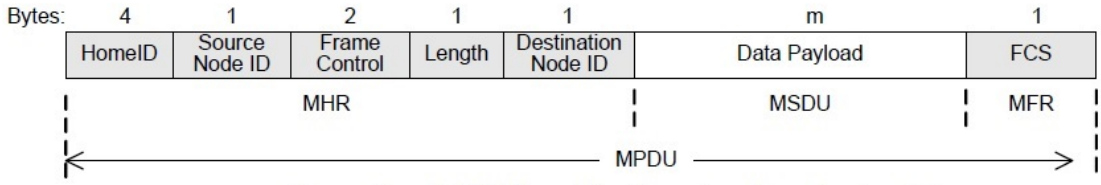
Se síťovou vrstvou je celkem úzce spjatá vrstva transportní. Ta je zodpovědná za samotné odesílání a přijímání rámců a zajištění spolehlivosti přenosu. K tomu může využít opakování přenosu (anglicky retransmission) a také ACK pakety, které potvrzují správné přijetí rámce. Dalším prvkem k zajištění spolehlivosti je kontrolní součet, který je přidáván na konec transportního rámce.

Na úrovni transportní vrstvy je také rozlišováno, o jaký typ rámce jde. K dispozici jsou čtyři typy rámců:

- Singlecast – rámec tohoto typu je posílán do jednoho specifikovaného zařízení. Příjem rámce je potvrzován rámcem typu ACK. Je tak možno ověřit, zda došlo k úspěšnému doručení, nebo zda je potřeba odeslání opakovat.
- ACK – tento rámec slouží k potvrzení příjmu zařízením, jemuž byl rámec doručen.
- Multicast – rámce tohoto typu jsou posílány zároveň do více než jednoho zařízení. V tomto případě nedochází k potvrzování příjmu pomocí ACK rámců.
- Broadcast – tyto rámce jsou poslány všem zařízením v dané síti. Stejně jako v předchozím případě, ani zde nedochází k potvrzení příjmu danými zařízeními. [13]

Další vrstvou je vrstva MAC (Media Access Control). Jde o vrstvu, která zajišťuje přístup k přenosovému médiu. K tomu je využit například mechanismus předcházení kolizí. Ten zajišťuje, aby zařízení nezačalo posílat data v případě, kdy data již posílá jiné zařízení v síti. Rámce na této vrstvě již obsahují většinu informací potřebných k přenosu. Struktura MAC rámce je vyobrazena na obrázku 6. Skládá z následujících částí:

- Home ID – unikátní identifikátor sítě. Všechna zařízení v dané Z-Wave síti mají tento identifikátor identický a je jim přidělen již v okamžiku přidávání zařízení do sítě. Má velikost 4 B.
- Source Node ID – unikátní identifikátor zařízení, které je odesílatelem rámce. Má velikost 1 B.
- Frame Control – definuje, o jaký typ rámce se jedná. Kromě toho obsahuje také další informace, týkající se adresování rámce apod. Má velikost 2 B.
- Length – celková délka rámce v bytech. Má velikost 1 B.
- Destination Node ID – identifikátor zařízení, jemuž je rámec posílán. V případě broadcastového rámce se používá 0xFF. Má velikost 1 B.
- Data Payload – datový obsah rámce, jako je například třída příkazu, typ příkazu apod. ACK rámce tuto část neobsahují. Velikost této části je proměnlivá v závislosti na typu příkazu, počtu parametrů apod.
- FCS – kontrolní součet (anglicky checksum), který slouží pro ověření kompletnosti přijatých dat. Má velikost 1 B. [15]



Obrázek 6: Z-Wave MAC rámec. [15]

Poslední vrstvou je vrstva fyzická. Jedním z jejích hlavních úkolů je modulace a kódování přenášených dat a také řízení samotného přenosu mezi vysílačem a přijímačem. K MAC rámci, jehož struktura je uvedena výše, přidává část nazývanou preambule, která uvozuje každý rámec a také části značící začátek a konec rámce (SoF – Start of Frame, EoF – End of Frame). [13]

2.4 Třídy zařízení

Každé Z-Wave zařízení je specifikováno pomocí tříd zařízení. Funkce (přesněji třídy příkazů), které musí být v zařízení implementovány, záleží právě na třídách zařízení, do kterých dané zařízení spadá. Rozlišujeme tři kategorie tříd.

Základní kategorií je *Basic Device Class* (česky základní třída zařízení). Tato třída specifikuje, zda je zařízení typu controller, slave, nebo routing-slave. Pokud je typu controller, umožňuje mimo jiné přiřazování identifikátorů (*Node ID*) jednotlivým zařízením a podporuje směrování – udržuje tzv. směrovací tabulku. Typ slave je nejjednodušším typem zařízení. Nemůže samo o sobě iniciovat jakýkoliv přenos dat do jiného zařízení, kromě odpovědi na typ příkazu *Get* (typy příkazů jsou podrobně popsány v kapitole 2.5). Zařízení musí umožňovat přidání a odebrání ze sítě a také přeposlání příkazů od zdrojového zařízení, pokud je cílové zařízení mimo jeho dosah. Routing-slave je pak rozšířením předchozího zmíněného typu a může samo o sobě iniciovat přenos dat na více zařízení v síti.

Další kategorií je *Generic Device Class* (česky generická třída zařízení). Tato třída charakterizuje hlavní funkcionalitu daného zařízení. Určuje tedy například, zda se jedná o vypínač, termostat, snímač apod. K této třídě dále náleží určitá *Specific Device Class* (česky specifická třída zařízení). Ta přesněji specifikuje, o jaký druh zařízení se jedná. [9], [16]

2.5 Třídy příkazů

Každá zpráva, kterou si mezi sebou Z-Wave zařízení posílají, je nazývána příkaz. Příkazy můžeme rozdělit do třech základních kategorií:

- Set příkazy – požadujeme, aby zařízení něco udělalo,
- Get příkazy – požadujeme, aby nám zařízení poskytlo nějaké informace,
- Report příkazy – odpověď zařízení na příkaz Get.

Každý příkaz pak patří do určité třídy příkazů (anglicky command class). Třída příkazů reprezentuje určitou funkcionalitu zařízení a patří do ní tedy všechny příkazy spojené s touto funkcionalitou. Seznam podporovaných tříd příkazů poskytuje zařízení pomocí tzv. NIF (Node Information Frame) zprávy. Níže jsou uvedeny některé důležité třídy příkazů, které jsou v aplikaci využívány. [9]

2.5.1 Třída Basic

Základní třídou, používanou pro ovládání zařízení, je třída *Basic*. Tato třída umožňuje ovládání základních funkcionalit zařízení, například v případě klasické lampy ji umožňuje rozsvítit nebo zhasnout. Zároveň také umožňuje zjistit stav zařízení. Zda zařízení tuto třídu příkazů musí podporovat, je dáno jejich generickým a specifickým typem, viz kapitola 2.4.

Na této třídě příkazů je také vhodné detailněji ukázat, v jakém formátu musí být data posílaná, nebo naopak přijímaná ze zařízení. Příkaz je uvozen hodnotou třídy příkazu. Hodnoty odpovídající jednotlivým třídám příkazů je možno nalézt v tabulce v dokumentu *Z-Wave Command Class Specification*, dostupném na webu společnosti Sigma Designs. V tomto případě má třída příkazu hodnotu 0x20. Následuje typ příkazu a případně také jeho hodnota. V případě příkazu třídy *Basic* má typ *Set* hodnotu 0x01, typ *Get* 0x02 a typ *Report* 0x03. Hodnota příkazu nabývá rozsahu 0x00 až 0xFF. U všech částí příkazu jsou uváděny hodnoty v šestnáctkové soustavě. Níže je přiložena struktura rámce pro příkaz Basic Set s hodnotou 0xFF. V záhlaví tabulky je ilustrováno rozložení bitů, z tabulky je tak patrno, že každá část příkazu v tomto případě zabírá 8 bitů, tedy jeden byte. [17]

7	6	5	4	3	2	1	0
Třída příkazu: 0x20 (Basic)							
Typ příkazu: 0x01 (Set)							
Hodnota: 0x00 až 0xFF							

Tabulka 2: Struktura příkazu Basic Set [17]

Struktura rámce pro příkaz *Basic Get* je částečně podobná s rámcem předchozím. Třída příkazu zůstává stejná. Ke změně dochází v typu příkazu, v tomto případě je použita hodnota 0x02, která odpovídá typu příkazu *Get*. Logicky zde není hodnota příkazu, protože právě na ni se zařízení dotazujeme.

7	6	5	4	3	2	1	0
Třída příkazu: 0x20 (Basic)							
Typ příkazu: 0x02 (Get)							

Tabulka 3: Struktura příkazu Basic Get [17]

Jakmile je zařízení zaslán příkaz *Basic Get*, zařízení odpovídá příkazem *Basic Report*. Takto můžeme například zjistit, zda je lampa rozsvícená, nebo zhasnutá. Jako třída příkazu je opět použita třída *Basic*. Typem příkazu je nyní *Report* (tedy 0x03), dále následuje samotná hodnota. V případě, že by lampa byla zhaslá, dostáváme hodnotu 0x00. V případě, že by byla naopak rozsvícená, dostáváme hodnotu 0xFF. Struktura příkazu je opět uvedena v tabulce níže.

7	6	5	4	3	2	1	0
Třída příkazu: 0x20 (Basic)							
Typ příkazu: 0x03 (Report)							
Hodnota: 0x00 až 0xFF							

Tabulka 4: Struktura příkazu Basic Report [17]

2.5.2 Třída Configuration

Příkazy ze třídy *Configuration* umožňují nastavování parametrů zařízení na hodnotu požadovanou uživatelem, případně na výchozí hodnotu, která je specifikována výrobcem. Zároveň umožňuje čtení hodnot jednotlivých parametrů ze zařízení.

Každý z parametrů má specifické číslo a velikost. Tyto údaje jsou vždy uvedeny v návodu k zařízení. Číslo parametru a velikost jsou potřebné pro zaslání příkazu typu *Set* a také pro získání hodnot pomocí příkazu typu *Get*. Tabulka níže ilustruje strukturu příkazu *Configuration Set*, který má hodnotu typu příkazu 0x04. V příkazu se nachází také tzv. flag „Výchozí“. Pokud je tento bit nastaven, dojde k nastavení výchozí hodnoty. Z tabulky je také patrno, že vzhledem k různé velikosti parametrů může být hodnota rozdělena do několika po sobě jdoucích bytů. [17]

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Třída příkazu: 0x70 (Configuration)							
Typ příkazu: 0x04 (Set)							
Číslo parametru: 0x00 až 0xFF							
Výchozí	Rezervováno				Velikost: 1/2/4		
Hodnota parametru: 0x00 až 0xFF							
...							
Hodnota parametru: 0x00 až 0xFF							

Tabulka 5: Struktura příkazu Configuration Set [17]

Příkaz *Configuration Get* se jako obvykle skládá z hodnoty třídy příkazu a typu příkazu, v tomto případě má typ *Get* hodnotu 0x05. Za nimi pak následuje číslo parametru, jehož hodnotu chceme zjistit.

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Třída příkazu: 0x70 (Configuration)							
Typ příkazu: 0x05 (Get)							
Číslo parametru: 0x00 až 0xFF							

Tabulka 6: Struktura příkazu Configuration Get [17]

Příkaz *Configuration* Report má velmi podobnou strukturu jako příkaz *Configuration Set*. Liší se hodnotou typu příkazu, ta je nyní 0x06. Navíc zde, oproti zmíněnému příkazu, není prostor pro flag Výchozí.

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Třída příkazu: 0x70 (Configuration)							
Typ příkazu: 0x06 (Report)							
Číslo parametru: 0x00 až 0xFF							
Rezervováno					Velikost: 1/2/4		
Hodnota parametru: 0x00 až 0xFF							
...							
Hodnota parametru: 0x00 až 0xFF							

Tabulka 7: Struktura příkazu Configuration Report [17]

2.5.3 Třída Manufacturer Specific

Třída příkazů *Manufacturer Specific* umožňuje získání informací o výrobci a typu zařízení. Tyto informace jsou poskytovány ve formátu dvoubytových čísel. Na základě nich je například v realizované aplikaci rozhodováno, zda je zařízení od firmy Steinel a také to, o jaký výrobek se jedná. [17]

Příkaz *Manufacturer Specific Get* má podobnou strukturu jako dříve uvedený příkaz *Basic Get*. Obsahuje pouze hodnotu třídy příkazu (0x91) a hodnotu typu příkazu (0x04). Jeho struktura je uvedena níže.

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Třída příkazu: 0x91 (Manufacturer Specific)							
Typ příkazu: 0x04 (Get)							

Tabulka 8: Struktura příkazu Manufacturer Specific Get [17]

Odpovědí na příkaz *Manufacturer Specific Get* je příkaz *Manufacturer Specific Report*. Ten obsahuje konkrétně hodnoty ID výrobce, ID typu produktu a ID produktu.

7	6	5	4	3	2	1	0
Třída příkazu: 0x91 (Manufacturer Specific)							
Typ příkazu: 0x05 (Report)							
ID výrobce: 0x00 až 0xFF							
ID výrobce: 0x00 až 0xFF							
ID typu produktu: 0x00 až 0xFF							
ID typu produktu: 0x00 až 0xFF							
ID produktu: 0x00 až 0xFF							
ID produktu: 0x00 až 0xFF							

Tabulka 9: Struktura příkazu Manufacturer Specific Report [17]

2.5.4 Třída Association

Jednou ze základních tříd příkazů je třída *Association*. Ta umožňuje tvorbu asociačních vazeb mezi zařízeními a je tedy důležitá pro tvorbu jakékoliv automatizace v rámci Z-Wave sítě. Je využita například pro vytvoření vazby mezi PIR čidlem a lampou. Pohyb zaznamenaný PIR čidlem může vést k rozsvícení lampy apod. Kromě klasických typů příkazů, tedy *Get*, *Set* a *Report*, je ve třídě *Association* použit ještě typ *Remove*, který je popsán dále v této kapitole. [17]

Pro přidání jednotlivých zařízení do dané asociační skupiny slouží příkaz *Association Set*. Těchto skupin může zařízení obsahovat hned několik a jejich popis bývá uveden v návodu k zařízení. Příkaz je uvozen hodnotou třídy příkazu (0x85), následovanou jeho typem (0x01). Poté následuje identifikátor asociační skupiny a po něm následují identifikátory jednotlivých zařízení (*Node ID*). V jednom příkazu je možné přidat do skupiny najednou více zařízení.

7	6	5	4	3	2	1	0
Třída příkazu: 0x85 (Association)							
Typ příkazu: 0x01 (Set)							
Identifikátor skupiny: 0x01 až 0xFF							
Identifikátor zařízení 1: 0x00 až 0xFF							
...							
Identifikátor zařízení N: 0x00 až 0xFF							

Tabulka 10: Struktura příkazu Association Set [17]

K odebrání zařízení z asociační skupiny je použit příkaz *Association Remove*. Jeho struktura je prakticky stejná jako struktura příkazu *Association Set*. Liší se jen typem příkazu, který zde má hodnotu 0x04. Stejně jako v předchozím případě, i zde je možno ze skupiny odebrat najednou několik zařízení.

7	6	5	4	3	2	1	0
Třída příkazu: 0x85 (Association)							
Typ příkazu: 0x04 (Remove)							
Identifikátor skupiny: 0x01 až 0xFF							
Identifikátor zařízení 1: 0x00 až 0xFF							
...							
Identifikátor zařízení N: 0x00 až 0xFF							

Tabulka 11: Struktura příkazu Association Remove [17]

Ke zjištění, jaká zařízení se nacházejí v dané asociační skupině, je možno využít příkazu *Association Get*. Třída příkazu zůstává stejná jako v předchozím případě, typ příkazu má hodnotu 0x05 a po něm následuje hodnota identifikátoru skupiny.

7	6	5	4	3	2	1	0
Třída příkazu: 0x85 (Association)							
Typ příkazu: 0x05 (Get)							
Identifikátor skupiny: 0x01 až 0xFF							

Tabulka 12: Struktura příkazu Association Get [17]

Odpovědí na příkaz *Association Get* je příkaz *Association Report*. Ten má opět stejnou třídu příkazu jako předchozí zmíněný příkaz. Následuje typ příkazu s hodnotou 0x03. Poté následuje identifikátor skupiny, maximální počet zařízení v dané skupině, jejich aktuální počet a samotné identifikátory jednotlivých zařízení.

7	6	5	4	3	2	1	0
Třída příkazu: 0x85 (Association)							
Typ příkazu: 0x03 (Report)							
Identifikátor skupiny: 0x01 až 0xFF							
Maximální počet zařízení ve skupině: 0x00 až 0xFF							
Aktuální počet zařízení ve skupině: 0x00 až 0xFF							
Identifikátor zařízení 1: 0x00 až 0xFF							
...							
Identifikátor zařízení N: 0x00 až 0xFF							

Tabulka 13: Struktura příkazu Association Report [17]

3 Jazyk C#

C# je moderní objektově orientovaný programovací jazyk. Podobně jako Java nebo C++ patří mezi vyšší programovací jazyky. Ty jsou mimo jiné charakterizovány vyšší mírou abstrakce, podstatně menší závislostí programů na konkrétním technickém vybavení a také vyšší čitelností zdrojového kódu. Aktuálně patří mezi jeden z nejpoužívanějších programovacích jazyků na světě. Jazyk C# je součástí platformy Microsoft .NET Framework. Ta kromě programovacích jazyků (C#, VB.NET apod.) obsahuje také sadu vývojářských nástrojů a knihoven. Kromě toho také poskytuje prostředí pro běh programů napsaných právě v jazycích z rodiny .NET, které se nazývá Common Language Runtime (CLR). Zdrojové kódy aplikací napsaných v jazyce C# jsou uloženy v souborech s příponou `.cs`. Tyto soubory jsou kompilátorem přeloženy do formy spustitelných souborů s příponou `.exe`, případně knihoven s příponou `.dll`. Pro běh programů musí na počítači být nainstalována minimálně taková verze .NET Framework, pro kterou byla daná aplikace vyvinuta. [18], [19]

V následujících kapitolách jsou představeny základy použití jazyka C# a také některé technologie a postupy, které byly použity pro implementaci aplikace, která je předmětem této diplomové práce.

3.1 Základní stavební prvky

Vzhledem k tomu, že jazyk C# je objektově orientovaným jazykem, základním stavebním prvkem jsou třídy. Třídy slouží jako předpisy pro jejich instance, tedy objekty. Definují jejich vlastnosti (atributy) a chování (metody). Pro tvorbu objektů se využívá speciální metody, nazývané konstruktor. Tato metoda je specifická tím, že má stejný název jako třída, ve které je definována a také tím, že nemá žádný návratový typ. [18]

Níže je uvedena ukázka třídy `Osoba`. Ta obsahuje vlastnosti `jmeno` a `prijmeni`, obojí datového typu `string` (text). Dále obsahuje parametrický konstruktor, který v jeho těle umožňuje nastavení atributů třídy na požadované hodnoty. Dále obsahuje metodu `GetCeleJmeno()`, která má návratový typ `string` a vrací celé jméno osoby.

```
class Osoba
{
    // Datova pole (vlastnosti)
    private string jmeno;
    private string prijmeni;

    // Konstruktor
    public Osoba(string jmeno, string prijmeni)
    {
        this.jmeno = jmeno;
        this.prijmeni = prijmeni;
    }

    // Metoda, ktera vraci cele jmeno
    public string GetCeleJmeno()
    {
        return jmeno + " " + prijmeni;
    }
}
```

3.2 Windows Forms

Windows Forms (někdy nazýván zkráceně WinForms) je prvním frameworkem z rodiny Microsoft .NET, který vývojáři umožňuje snadnou tvorbu formulářových aplikací pomocí grafického designéru. Windows Forms umožňuje použití velkého množství připravených grafických komponent, zároveň také umožňuje použití vlastních komponent. Kromě zmíněného frameworku je k tvorbě formulářových aplikací možno využít také modernější framework WPF (Windows Presentation Foundation). Ten zajišťuje lepší oddělení logiky od grafického výstupu, rychlejší vykreslování, možnost využití animací apod. Windows Forms ale není považován za zastaralý a díky jeho jednoduchosti (ve srovnání se zmíněným WPF) je pro tvorbu formulářových aplikací stále hojně využíván. [20]

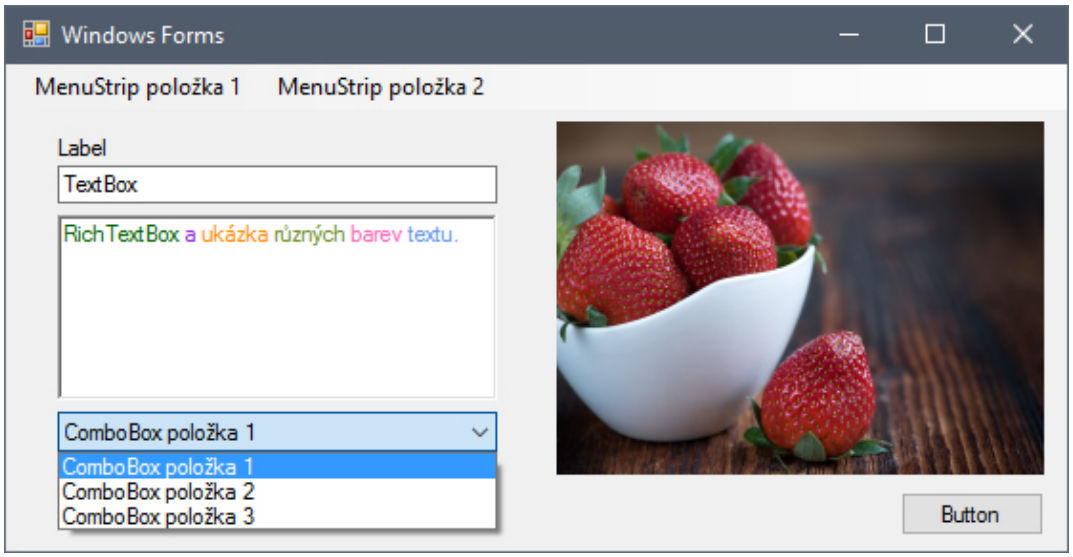
Vzhledem k jednoduššímu použití a hlubší znalosti byl pro tvorbu aplikace, která je předmětem této diplomové práce, vybrán právě framework Windows Forms. Další motivací byly problémy s přesouváním prvků pomocí myši (tzv. drag and drop), které se u WPF objevily a které se nepodařilo úspěšně vyřešit. V následujících kapitolách jsou popsány některé formulářové prvky, které jsou součásti Windows Forms a také princip tvorby vlastních komponent.

3.2.1 Formulářové prvky

Windows Forms obsahují celou řadu grafických komponent. V této kapitole jsou představeny nejpoužívanější z nich. Jednou z nejpoužívanějších je `Label`. Tato komponenta umožňuje umístění textového popisku do formuláře. U popisku je mimo jiné možno měnit jeho formátování – tedy velikost a barvu písma, font, zarovnání apod. K zadávání textu je nejčastěji využita komponenta `TextBox`, případně `RichTextBox`. Druhá zmíněná komponenta umožňuje komplexnější práci s obsaženým textem, možnost změny barvy jednotlivých slov apod. Pro výběr z několika možností z rozbalovacího seznamu je k dispozici komponenta `ComboBox`. Zde je možno nastavit několik různých typů zobrazení, které se liší tím, zda jde o pevně definovaný seznam, případně zda je umožněno zadávání textu ze strany uživatele apod. Další ze základních komponent je `Button`. Jedná se o klasické tlačítko, u kterého je možno kliknutím vyvolat určitou akci. K umístění obrázku do formuláře může být využita komponenta `PictureBox`. Tato komponenta mimo jiné umožňuje pomocí vlastnosti `SizeMode` volbu různých režimů přizpůsobení velikosti obrázku.

K vytvoření známé vodorovné lišty s menu v horní části aplikace slouží komponenta `MenuStrip`. Ta umožňuje přidávání jednotlivých položek, které mají typicky formu seznamu. Každá položka pak může po kliknutí na ni vyvolávat nějakou akci, sloužit jako zaškrtávací pole, případně otevírat další podmenu. Pro členění okna na jednotlivé části (skupiny) může být použita komponenta `Panel`. Panel je možno umístit kamkoliv do formulářového okna, také je možno mu nastavit různé typy okrajů apod. Výhodou umístění dílčích komponent do panelu je, že změna vlastnosti `Enabled` („povolení“ komponenty) a `Visible` (viditelnost komponenty) ovlivní i její obsah. Pokud tedy chceme nastavit viditelnost všech komponent umístěných v daném panelu, není třeba nastavovat viditelnost každé z nich, ale stačí změnit patřičnou vlastnost u panelu, který je obsahuje. [21]

Ukázka vzorového formulářového okna, které obsahuje výše zmíněné komponenty, je přiložena na Obrázku 7.



Obrázek 7: Ukázka formuláře Windows Forms.

3.3 Výjimky

Výjimka je v podstatě upozornění, že došlo k přerušení normálního běhu programu. Výjimky tedy programátorovi umožňují detekovat a reagovat na neočekávané události. Pokud během provádění programu dojde k výjimce, aktuální stav programu je uložen, standardní běh programu je přerušen a řízení programu je předáno obslužné rutině výjimky, pokud taková rutina existuje.

Výjimka je v platformě .NET objekt. Tento objekt signalizuje chybu nebo událost, která není součástí standardního běhu programu. Může jít například o chybu během otevírání souboru, pokud daný soubor neexistuje. Pokud k takové chybě dojde, je „vyhozena“ (anglicky throw) výjimka. Vznikne tedy objekt, který obsahuje typ chyby a část programu, kde k chybě došlo a několik dalších informací. Objekt také mimo jiné obsahuje tzv. *stack trace*. Ten programátorovi umožňuje vysledovat posloupnost akcí, které k výjimce vedly a tedy k snazšímu nalezení problematické části aplikace a snazší opravě chyby. Základní konstrukce kódu pro obsluhu výjimek, tzv. blok try-catch, je uvedena níže. V kódu může být i více bloků `catch`, každý z nich může zpracovávat jiný typ výjimky. [18], [19]

```
try
{
   // Misto, kde muze dojit k chybe
}
catch (Exception e)
{
   // Misto, kde je chyba zpracovana
}
```

Pokud během vykonávání programu dojde k výjimce, nedojde k provedení zbytku kódu, který se nachází za místem, kde k chybě došlo. Často je však potřeba, aby po nastalé chybě bylo zajištěno uvolnění určitých zdrojů. K tomu může být použit blok `finally`, jehož obsah je vykonán za všech okolností – nezáleží tedy na tom, zda chyba nastane, nebo nenastane.

V programu mohou být využity i vlastní výjimky. V takovém případě je nutné vytvořit třídu, která bude potomkem třídy `Exception`. Doporučuje se, aby název třídy byl ukončen slovem Exception (např. UserNotFoundException). Samotná třída obsahuje konstruktor, případně několik konstruktorů s různými parametry. Níže je přiložena ukázka implementace vlastní výjimky se třemi různými konstruktory, každý s jinými parametry. [19], [22]

```
public class UserNotFoundException: Exception
{
    public UserNotFoundException()
    {
    }

    public UserNotFoundException(string message)
      : base(message)
    {
    }

    public UserNotFoundException(string message, Exception inner)
      : base(message, inner)
    {
    }
}
```

3.4 LINQ

LINQ (Language Integrated Query) je technologie umožňující dotazování nad různými zdroji dat. Jako zdroj dat může sloužit jakákoliv kolekce, která implementuje rozhraní `IEnumerable<T>`, relační databáze, XML dokument apod. LINQ byl jako koncept představen v roce 2005, poté došlo k jeho zabudování do .NET Frameworku 3.5. Ve verzi .NET Frameworku 4.0 pak byla přidána možnost paralelního dotazování nazývaná PLINQ (Parallel LINQ). Hlavní výhodou technologie LINQ je univerzálnost dotazů. Stejný dotaz může být použit pro přístup k datům v databázi, stejně tak pro přístup ke kolekci dat uložené v paměti. Další výhodou je také syntaktická kontrola dotazů již při překladu programu.

Technologie je založená na tzv. deklarativním paradigmatu. Jeho podstatou je, že specifikujeme, jaká data chceme získat a nemusíme se starat o to, jakým způsobem jsou prakticky získávána. Syntaxe dotazů je velmi podobná jazyku SQL. Stejně jako v jazyce SQL se i zde používají klíčová slova *select*, *from*, *where*, *join* apod. Hlavním rozdílem je pořadí jednotlivých částí. V LINQ je dotaz uvozen klíčovým slovem *from*. Za ním následuje název proměnné, která reprezentuje prvek z kolekce. Poté následuje klíčové slovo *in* a za ním zdroj dat. Poté mohou následovat další klíčová slova, jako např. *where*, které nám umožňuje určit podmínku pro výběr dat. Příkaz je zakončen klíčovým slovem *select*, pomocí kterého specifikujeme, co vybíráme. [23], [24]

Jako jednoduchý příklad může posloužit dotaz, který z kolekce `prijmeni` vybere taková příjmení, která mají víc jak osm znaků.

```
var vysledek = from prijmeni in vsechnaPrijmeni
               where (prijmeni.Length > 8)
               select prijmeni;
```

Kromě výše uvedeného přístupu mohou být použity i dotazy založené na volání metody (anglicky method-based). Jako parametry jsou zde využívány tzv. lambda výrazy. [25] Stejného výsledku, jako u dotazu výše, je možno dosáhnout následujícím dotazem:

```
var vysledek = vsechnaPrijmeni.Where(prijmeni => prijmeni.Length > 8);
```

3.5 Práce s XML soubory

Jazyk C# nabízí několik způsobů, jakými lze pracovat s XML soubory. Jedním z nejznámějších je využití třídy `XmlReader` pro čtení a `XmlWriter` pro zápis do souboru. Další možností je použití třídy `XmlDocument`. Její výhodou je například snadnější úprava již existujícího XML souboru. Nevýhodou může být vyšší paměťová náročnost a také mírně složitější zdrojový kód. Jako další možnost se nabízí využití třídy `XDocument` v kombinaci s LINQ dotazy. A právě tento způsob bude podrobněji rozebrán.

Představme si XML dokument nazvaný *products.xml*. Tento dokument obsahuje seznam několika produktů a mohl by vypadat například takto:

```
<Products>
  <Product>Laptop</Product>
  <Product>Printer</Product>
  <Product>Mouse</Product>
</Products>
```

Soubor je nejprve otevřen pomocí funkce `XDocument.Load()`. Jako parametr funkce slouží cesta k souboru. V tomto případě je soubor umístěn ve stejném adresáři jako spustitelný program, stačí tedy specifikovat název souboru. Dále můžou být do souboru přidány nové položky, anglicky nazývané *Element*. Jednou z výhod využití třídy `XDocument` je, že lze přesně specifikovat, kam se má položka vložit. Položka může být vložena například na konec stávajícího seznamu produktů, nebo naopak na začátek. Jak je tyto operace možno provést, je ukázáno v kódu níže. Nejprve je vybrán element, se kterým chceme pracovat, v tomto případě *Products*. Poté se vybere patřičná operace, tedy například `Add()` pro přidání na konec, nebo `AddFirst()` pro přidání na začátek. Parametrem těchto funkcí je objekt typu `XElement`, v jehož konstruktoru je uveden název elementu, který má být vložen a jeho obsah. [26]

```
XDocument xdoc = XDocument.Load("products.xml");

// pridani noveho elementu za jiz existujici
xdoc.Element("Products").Add(new XElement("Product", "Mobile"));

// pridani noveho elementu na zacatek
xdoc.Element("Products").AddFirst(new XElement("Product", "Tablet"));
```

Po těchto operacích do dokumentu přibudou dva nové elementy *<Product>*. Struktura celého souboru pak vypadá takto:

```
<Products>
  <Product>Tablet</Product>
  <Product>Laptop</Product>
  <Product>Printer</Product>
  <Product>Mouse</Product>
  <Product>Mobile</Product>
</Products>
```

K uložení dat do nového XML souboru je použita funkce `Save()`. Parametrem je opět cesta k souboru, v tomto případě je soubor ukládán na stejné místo, na kterém se nachází soubor původní, jen s novým názvem *products_new.xml*. Ke stávajícímu zdrojovému kódu je tedy přidán následující řádek:

```
xdoc.Save("products_new.xml");
```

Nyní si představme situaci, kdy z tohoto souboru chceme vybrat všechny produkty, které začínají na písmeno „M“ a následně je vypsat do konzole. K tomu může být využit dříve zmíněný LINQ. Vybírány jsou tedy všechny „podelementy“ (přesněji následníci) elementu *Product* a dotaz je zpřesněn podmínkou, že obsah tohoto elementu musí začínat na dané písmeno. Kód bude vypadat následovně:

```
var produkty = from element
               in xdoc.Element("Products").Descendants("Product")
               where (element.Value.StartsWith("M"))
               select element.Value;

foreach (var produkt in produkty)
{
   Console.WriteLine(produkt);
}
```

4 Implementace aplikace

Aplikace je naprogramována v programovacím jazyce C#. Jako cílový framework byl zvolen Microsoft .NET Framework ve verzi 4.0. V následujících kapitolách je nejprve popsána základní struktura aplikace z hlediska tříd a také jejich základní popis. Další kapitoly pak popisují implementaci některých stěžejních částí a funkcionalit aplikace.

4.1 Struktura aplikace a popis tříd

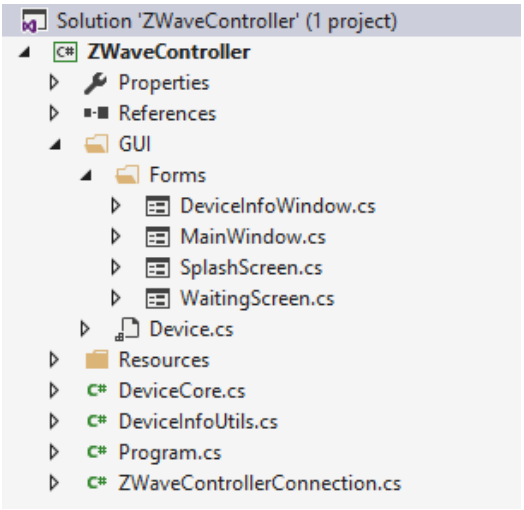
Aplikace se skládá z několika tříd, které jsou uloženy v souborech s příponou *.cs*. Kromě těchto tříd jsou v aplikaci také využity některé knihovny, které usnadňují obsluhu Z-Wave kontroléru a samotnou komunikaci s jednotlivými zařízeními. Tyto knihovny mají příponu *.dll*. Pro větší přehlednost jsou třídy přímo se týkající grafického uživatelského rozhraní umístěny v adresáři *GUI*. Zde dochází ještě k dalšímu větvení, kdy jsou jednotlivé třídy, reprezentující různá okna aplikace, umístěny v adresáři *Forms*. Přímo v adresáři *GUI* je umístěna pouze třída `Device`. Ta představuje grafickou reprezentaci zařízení a je detailněji popsána v kapitole 4.4. Jak už bylo zmíněno, nachází se zde i adresář *Forms*. Ten obsahuje následující třídy:

- `DeviceInfoWindow` – okno umožňující nastavování a získávání parametrů a také zobrazení základních informací o zařízení. Podrobněji je popsána v kapitole 4.11.
- `Main Window` – hlavní okno aplikace. Do tohoto okna jsou přidávána jednotlivá zařízení, umožňuje jejich přesouvání do skupiny, vyvolávání dialogů pro přidávání nebo odebírání zařízení apod. Detailněji se této třídě věnuje kapitola 4.3.
- `SplashScreen` – okno, které se zobrazí při spuštění aplikace a informuje uživatele o průběhu načítání zařízení z kontroléru do aplikace. Zároveň také zajišťuje základní inicializaci zařízení, tedy aktualizaci názvu a stavu tohoto zařízení.
- `WaitingScreen` – toto okno se zobrazuje během operace přidávání, případně odebírání zařízení. Informuje uživatele o krocích, které je potřeba během přidávání, respektive odebírání zařízení udělat.

V kořenovém adresáři se nachází jen třídy, které nepředstavují žádnou grafickou část aplikace. Jde o následující třídy:

- `DeviceCore` – tvoří pomyslné jádro každého zařízení v rámci aplikace. Je nositelem informací o zařízení a také zajišťuje zpracování přijatých dat. Tato třída je rozebrána v kapitole 4.5.
- `DeviceInfoUtils` – tato třída obsahuje různé pomocné metody a kolekce dat. Mimo jiné zajišťuje načítání a zpracování XML souborů obsahujících informace o generických třídách zařízení, příkazech apod. Podrobněji se jí věnuje kapitola 4.7.
- `Program` – základní třída aplikace, která obsahuje metodu `Main`, jež je vstupním bodem do programu. V této třídě dochází k vyhodnocení parametrů, se kterými je aplikace spouštěna. Na základě parametrů může být aplikace spuštěna v tzv. demo režimu, který umožňuje demonstraci většiny funkcí a ukázku chování aplikace. Tento režim je popsán v kapitole 5.1.
- `ZWaveControllerConnection` – tato třída představuje připojení ke kontroléru. Zpřístupňuje zařízení, která jsou v tomto kontroléru uložena, umožňuje jejich přidávání, odebírání, posílání a příjem dat apod. Protože jde o jednu z nejzásadnějších tříd, je důkladněji popsána v kapitole 4.6.

Struktura výše popsaných tříd, tak jak ji zobrazuje aplikace Visual Studio 2015, je vyobrazena na obrázku 8. Kompletní diagram tříd je ve formátu PDF přiložen na CD, které je přílohou této diplomové práce.



Obrázek 8: Struktura tříd aplikace.

4.2 Knihovny Zensys

Pro usnadnění operací se Z-Wave kontrolérem jsou využity knihovny nabízené společností Zensys. Tyto knihovny obsahují velké množství metod, které umožňují například přidávání a odebírání zařízení z kontroléru, odesílání a příjem příkazů, získání seznamu zařízení uložených v kontroléru a podobně. Knihovny jsou uloženy v souborech s příponou *.dll* a jsou umístěny v adresáři se spustitelným souborem aplikace. Pro jejich použití musí být přidány mezi tzv. reference. Ve vývojovém prostředí Visual Studio 2015 se reference přidávají pravým kliknutím na položku *References* ve struktuře otevřeného projektu. Ze zobrazeného kontextového menu je pak třeba vybrat položku *Add reference*. V zobrazeném okně jsou následně vybrány požadované knihovny.

Zásadním problémem při použití zmíněných knihoven je jejich velice špatná dokumentace. Ta totiž obsahuje pouze názvy jednotlivých nabízených tříd a jejich metod, avšak bez jakéhokoliv dalšího popisu jejich funkce. Jejich funkci tak bylo třeba často zkoušet metodou pokus-omyl, což vývoj aplikace zdržovalo. Na tomto místě je tedy vhodné některé použité třídy, respektive jejich metody, detailněji popsat. Všechny třídy, které jsou součástí knihoven, implementují svá rozhraní. Jedněmi z nejčastěji využívaných rozhraní jsou rozhraní `IController` a `IDevice` z knihovny `Zensys.ZWave`. Ty obsahují metody, které pomáhají s operacemi prováděnými s kontrolérem a zařízeními v síti. Níže je uveden seznam důležitých metod včetně jejich případných parametrů a popisu.

- `AddNodeToNetwork()` – přepne kontrolér do režimu, ve kterém čeká na přidání nového zařízení. Opětovné volání této funkce přepíná kontrolér zpět do standardního režimu.
- `RemoveNodeFromNetwork()` – přepne kontrolér do režimu, ve kterém naopak čeká na odebrání zařízení. Stejně jako v předchozím případě, opětovné volání metody přepíná kontrolér do standardního režimu.
- `SendData(byte nodeId, byte[] data, TransmitOptions txOptions)` – odešle data (předaná v parametru `data`) do zařízení s *Node ID* specifikovaným parametrem `nodeId`. Posledním parametrem, nazvaným `txOptions`, je nastavení typu přenosu. V aplikaci je využit typ `TransmitOptionAcknowledge`. Poslané příkazy jsou tedy potvrzovány ACK paketem. Dostupné typy jsou uloženy ve výčtu `Zensys.ZWave.Enums.TransmitOptions`.

- `SendData(byte nodeId, byte[] data, TransmitOptions txOptions, int timeout)` – přetížení předchozí zmíněné metody. Obsahuje navíc parametr `timeout`, kterým lze specifikovat, jak dlouho se čeká na potvrzení úspěšného příjmu odeslaných dat.
- `GetNodes()` – tato metoda vrací seznam (kolekce typu List), který obsahuje všechna zařízení, aktuálně přidaná v kontroléru. Zde je třeba dát pozor na to, že zařízením uloženém na prvním místě v seznamu je samotný kontrolér.
- `Open(string portName)` – otevře spojení se sériovým portem, který je specifikován názvem předaným v parametru `portName` (například „COM4“).
- `Close()` – uzavře stávající spojení se sériovým portem.
- `GetCapabilities(), GetControllerCapabilities(), GetSUCNodeID()` – tyto tři metody je nezbytné zavolat po otevření sériového portu. Slouží k jakési inicializaci kontroléru. Bez jejich zavolání není možno kontrolér používat.

4.3 Hlavní okno aplikace

Hlavní okno je zobrazeno ihned po získání zařízení z kontroléru a jejich následné inicializaci. Během té je zobrazen tzv. splash screen, tedy jakási úvodní obrazovka. Hlavní okno je zobrazeno automaticky v maximalizovaném režimu, tak aby byl dostatek místa na zobrazení zařízení a různých ovládacích prvků.

Hlavní okno je rozděleno do tří základních oblastí – panelu obsahujícího zařízení ve skupině, panelu obsahujícího všechna zařízení, mimo zařízení již umístěných ve skupině a kontextového menu v horní části aplikace. Většinu plochy okna zabírá komponenta `SplitContainer`. Ta rozděluje plochu do dvou panelů. Levou část zabírá panel nazvaný `panelGroup`. Ten, jak již bylo zmíněno, obsahuje zařízení ve skupině – tedy zařízení, mezi kterými jsou vytvořeny asociační vazby, viz kapitola 4.10. Pravá část, obsahující panel nazvaný `panelAllDevices`, je fixní, nemění tedy velikost při změnách velikosti okna. Menu v horní části okna obsahuje různé kontextové nabídky pro správu zařízení, zobrazení nápovědy apod. Jak hlavní okno vypadá, je vyobrazeno na obrázku 9.

V souvislosti s hlavním oknem bylo třeba řešit několik problémů týkajících se zobrazení. V případě pravého panelu (tedy panelu všech zařízení) bylo třeba vyřešit změnu jeho šířky pokud došlo k zobrazení, případně skrytí posuvníku. Automaticky totiž ke změně šířky nedochází. Posuvník, který se zobrazil, pokud panel obsahoval více zařízení, než

Obrázek 9: Ukázka hlavního okna aplikace.

bylo možno najednou zobrazit, tedy překrýval již umístěná zařízení. Jednou z možností bylo stálé odsazení komponent zařízení zprava o velikost posuvníku, to ale z grafického hlediska nepůsobilo pěkně. Problém byl tedy vyřešen reakcí na událost `ClientSizeChanged` zmíněného panelu. Pokaždé, když k této události dojde, je ověřeno, zda se již zobrazil posuvník. Pokud ano, je upravena šířka panelu připočtením šířky vertikálního posuvníku. Pokud posuvník není zobrazen, nastaví se standardní šířka panelu, což je 210 pixelů. V obou případech také musí dojít k nastavení vzdálenosti oddělovače panelů (splitteru), které je počítáno od levého okraje. Kód pro řešení této situace je uveden níže.

```
private void panelAllDevices_ClientSizeChanged(object sender, EventArgs e)
{
   if (panelAllDevices.VerticalScroll.Visible)
{
      panelAllDevices.Width = 210 +
          System.Windows.Forms.SystemInformation.VerticalScrollBarWidth;
   }
   else
   {
      panelAllDevices.Width = 210;
   }

   splitContainer1.SplitterDistance = splitContainer1.Width -
       panelAllDevices.Width;
}
```

Dalším problémem, který bylo třeba vyřešit, byla eliminace vzniku prázdných míst při přesunu zařízení z panelu všech zařízení do panelu skupiny. V tomto případě je při každé operaci, při které se mění počet zařízení umístěných v panelu, volána funkce `ReorganizePanelAllDevices`. Ta postupně nastaví korektní souřadnice všem zařízením v panelu. Při změně souřadnic je třeba počítat s tím, že se posuvník vždy nemusí nacházet nahoře. Je tak potřeba vždy odečíst daný počet pixelů, o který je tzv. odscrollováno. Nulové souřadnice (tedy bod, který má souřadnice X i Y rovny nule) totiž nezačínají v levém horním rohu panelu, nýbrž v levém horním rohu jeho aktuálně zobrazené oblasti. Mění se tedy v závislosti na posunu posuvníku. Implementace zmíněné funkce je uvedena níže.

```
private void ReorganizePanelAllDevices()
{
   int x = 5;
   int y = 5;
   foreach (Device device in panelAllDevices.Controls)
   {
      device.Location = new Point(x, y-(-panelAllDevices.VerticalScroll.Value));
      y += 205;
   }
}
```

Další podstatnou funkcionalitou hlavního okna je korekce souřadnic zařízení v panelu skupiny. Bylo totiž žádoucí, aby se střed grafického prvku zařízení při operaci „drag and drop“, tedy přesunu myší, nacházel v místě, kde se aktuálně nachází kurzor. Standardně

by se zde totiž nacházel levý horní roh daného prvku. Zároveň bylo třeba zajistit, aby nemohla nastat situace, kdy grafický prvek „přeteče“ za kraj panelu. Jeho pozice je vždy upravena tak, aby se celý nacházel v panelu. V tomto případě je vždy třeba ověřit, jestli je zařízení umísťováno přes levou, pravou, horní, nebo dolní hranu. Zdrojový kód metody `GetCorrectControlLocation`, která počítá korektní souřadnice, je uveden níže.

```
private Point GetCorrectControlLocation(Point point, Device control, bool dropped)
{
   Point newLocation = new Point(point.X, point.Y);

   if (dropped)
   {
      // Center of control position
      newLocation.X -= control.Width / 2;
      newLocation.Y -= control.Height / 2;
   }

   // Avoid overflow over the panel border
   if (newLocation.X <= 0) newLocation.X += newLocation.X * (-1) + 5;
   if (newLocation.X + control.Width >= panelGroup.Width) newLocation.X =
       panelGroup.Width - control.Width - 5;

   if (newLocation.Y <= 0) newLocation.Y += newLocation.Y * (-1) + 5;
   if (newLocation.Y + control.Height >= panelGroup.Height) newLocation.Y =
       panelGroup.Height - control.Height - 5;

   return newLocation;
}
```

4.4 Grafická reprezentace zařízení

Grafickou reprezentaci zařízení v rámci aplikace zajišťuje třída `Device`. Aby bylo možné s touto třídou pracovat jako s jakýmkoliv jiným grafickým prvkem (tlačítko, vstupní textové pole apod.), je třída děděna od třídy `Control`. Pro lepší představu popisovaných částí je na obrázku 10 vyobrazena ukázka jednoho zařízení. V případě ukázky se jedná o zařízení od firmy Steinel, konkrétně svítidlo L 810 IHF. Z ukázky je také patrné, že svit je nastaven na maximum, tedy 100 %.

Třída `Device` se z grafického hlediska skládá ze dvou panelů – `TopPanel` a `BottomPanel`. Oba panely jsou definovány jako samostatné třídy. Toto řešení bylo zvoleno z důvodu nutnosti vlastní implementace děděné metody `OnPaint` v každé z nich. Zmíněné metody

Obrázek 10: Ukázka grafického znázornění zařízení.

obsahují část kódu pro vykreslení okrajů jednotlivých panelů. Pokud nebyly samostatně implementovány, docházelo k problémům s vykreslováním. Jednotlivé části byly vykreslovány ve špatném pořadí a okraje se navzájem překrývaly.

Třída `TopPanel` představuje horní část grafického znázornění zařízení. Jako obrázek na pozadí je standardně použito logo technologie Z-Wave. Tento obrázek může být změněn v závislosti na identifikaci zařízení. Pokud je zjištěno, že zařízení je jedním z produktů firmy Steinel, pro které jsou dostupné produktové obrázky, je pozadí změněno. Kromě změny pozadí je změněn i název zařízení. Obě tyto změny zajišťuje metoda `DeviceCoreOnNameChanged`. Tato metoda je registrována jako obslužná metoda události `NameChanged` ze třídy `DeviceCore`. Na panelu se dále nachází textový popisek stavu zařízení. Tento popisek typicky zobrazuje procenta svitu (pokud je zařízením svítidlo). Protože Z-Wave zařízeními jsou často i tzv. multi-level svítidla, která mohou svítit např. na 10 %, bylo toto znázornění vhodnější než například pomocí dvou ikon „rozsvíceno“ a „zhasnuto“. Hodnota tohoto popisku je nastavována metodou `DeviceCoreOnStateChanged`. Tato metoda čerpá data opět z třídy `DeviceCore`, konkrétně atributu `DeviceState`. Stejně jako předchozí jmenovaná metoda, i tato je registrována jako obslužná metoda události – v tomto případě události `StateChanged`. Dále je na panelu umístěn další malý panel. Tento panel slouží pro vyvolání okna sloužícího ke zjištění informací o zařízení a nastavení parametrů. Tomuto oknu je věnována kapitola 4.11. Původní logickou myšlenkou bylo použití klasického tlačítka, ale z hlediska možností úprav vzhledu (např. použití ikony, neviditelných okrajů apod.), bylo vhodnější použití panelu.

Třída `BottomPanel` představuje dolní část grafického znázornění zařízení. V této části je zobrazován název zařízení, obsahuje tedy jen textový popisek. Protože některé názvy zařízení jsou dlouhé a nebylo by možné je zobrazit celé, je text názvu zkrácen na dvacet pět znaků a doplněn o tři tečky.

4.5 Třída DeviceCore

Třída `DeviceCore` zastává pomyslné jádro každého zařízení, se kterým je v rámci aplikace pracováno. Obsahuje veškeré informace o zařízení, které jsou v aplikaci používány a také metody pro zpracování přijatých dat – reportů. Níže je uveden seznam jednotlivých atributů spolu s jejich základním popisem:

- `NodeID` – číselný identifikátor zařízení, který je přidělován kontrolérem.
- `DeviceName` – jméno zařízení, může být měněno v závislosti na tom, zda je zařízení identifikováno jako zařízení od firmy Steinel.
- `DeviceID` – číselný identifikátor, který je poskytován přímo zařízením. Slouží pro identifikaci obchodního označení zařízení.
- `DeviceStat` – informace o stavu zařízení (typicky úroveň svitu).
- `GenericClass` – generická třída zařízení.
- `SpecificClass` – specifická třída zařízení.
- `IsSteinelDevice` – informace o tom, zda je zařízení od firmy Steinel. Využívá se např. při rozhodování, zda přiřazovat obrázek zařízení.
- `IsResponding` – informace o tom, zda zařízení odpovídá. Při vytvoření instance třídy je nastavena na hodnotu `false`. K nastavení na hodnotu `true` dochází až při přijetí dat z tohoto zařízení.

Kromě těchto atributů třída obsahuje také události, které mohou být v rámci třídy vyvolány. Tyto události jsou využívány jinými třídami, na základě nich například dochází ke změně popisku se jménem zařízení, popisku se stavem zařízení apod. Události jsou následující:

- `NameChanged` – tato událost je vyvolána, pokud je zpracován příkaz typu *Manufacturer Specific Report*. V tu chvíli je totiž možno rozhodnout, o jaké zařízení se jedná a podle toho změnit jeho název a na tuto změnu reagovat v jiné třídě.

- `StateChanged` – tato událost je vyvolána, pokud dojde k přijetí příkazu typu *Basic Report*, případně *Switch Multilevel Report*. Říká, že došlo ke změně stavu zařízení, např. jeho rozsvícení.
- `ParameterValueReceived` – tato událost informuje o tom, že došlo k přijetí hodnoty parametru. Dochází k ní při zpracování příkazu typu *Configuration Report*.

Jak již bylo zmíněno v úvodu kapitoly, třída také obsahuje několik metod. Tyto metody zajišťují zpracování přijatých dat z daného zařízení. Jednou ze zásadních metod je metoda `ProcessReceivedData`. Ta totiž kontroléru potvrzuje přijetí dat, aby mohl začít posílat další příkazy čekající ve frontě. Dále nastavuje hodnotu atributu `IsResponding` na hodnotu `true`, takže je zařízení dále považováno za „odpovídající“. Posledním úkolem této metody je rozhodnutí, o jakou třídu a typ přijatého příkazu se jedná a následné rozhodnutí, která metoda pro zpracování daného příkazu bude volána. Pro zpracování každého typu příkazu je použita samostatná metoda, a to z důvodu vyšší přehlednosti kódu. Níže je pro ilustraci uvedena část zmíněné metody, která zajišťuje zavolání patřičné metody pro zpracování příkazu typu *Basic Report*.

```
if (receivedData.CommandClassKey == 0x20 && receivedData.CommandKey == 0x03)
{
   ProcessBasicReport(receivedData.CommandBuffer);
}
```

Jak je z ukázky patrné, rozhodnutí je založeno na hodnotách atributů `CommandClassKey` a `CommandKey`. První z uvedených hodnot představuje třídu příkazu, druhá pak typ příkazu. V tomto případě má třída příkazu hodnotu 0x20, této hodnotě odpovídá třída *Basic*. Hodnota typu příkazu je pak 0x03, té odpovídá typ příkazu *Report*. Třídy příkazů jsou podrobněji rozebrány v kapitole 2.5.

4.6 Třída ZWaveControllerConnection

Hlavním úkolem třídy `ZWaveControllerConnection` je správa připojení k USB kontroléru, přidávání a odebírání zařízení z kontroléru a zároveň také udržování seznamu zařízení (instancí třídy `DeviceCore`) pro použití v rámci aplikace. Kromě toho také zajišťuje odesílání

a příjem dat mezi kontrolérem a zařízeními. Odesílání a příjmu dat se podrobněji věnují kapitoly 4.6.1 a 4.6.2.

Třída je založena na návrhovém vzoru *Singleton*. Taková třída je specifická tím, že v rámci aplikace existuje právě jedna její instance. Toho je docíleno tak, že třída obsahuje statický atribut – instanci sebe sama. Ten slouží jako jediný přístupový bod do této třídy. Dalším specifikem je privátní konstruktor. Instanci takové třídy pak nelze vytvořit v ostatních třídách. Tento návrhový vzor je vhodné použít tam, kde je potřeba, aby v rámci programu byla používána pouze jedna instance dané třídy – typicky jde o připojení k zařízení, databázi apod. [27]

Jedním z úkolů, které je třeba v třídě zajistit, je detekce zařízení komunikujících na sériovém portu. Mezi nimi je i zmíněný USB kontrolér, tvořící tzv. virtuální sériové zařízení. K získání seznamu zařízení je využito knihovních funkcí z knihovny Zensys. Při této operaci často docházelo k „zamrzání“ aplikace a tím i k enormně dlouhé době jejího spouštění. Problém byl vyřešen tak, že jsou do seznamu zařízení přidávána jen ta zařízení, která poskytují službu *usbser*. Ta specifikuje, že zařízení představuje USB sériový port. Pokud zařízení splňuje tuto podmínku, je zařazeno do seznamu. Pro uchování informací o sériově komunikujícím zařízení byla vytvořena třída `SerialDevice`. Tato třída obsahuje atributy specifikující popis zařízení, komunikační port a knihovnu zařízení. Tyto údaje jsou poté využívány pro identifikaci správného zařízení, viz níže.

Zmíněný seznam dostupných zařízení je využíván v metodě `Connect`, která zajišťuje samotné připojení ke kontroléru – tedy otevření patřičného sériového portu. V cyklu jsou procházena všechna dostupná zařízení a u každého je ověřováno, zda jeho popis odpovídá řetězci „UZB“ a zda zařízení podporuje knihovnu nazvanou „ControllerStaticLib“. Kromě těchto dvou podmínek musí být ještě splněna podmínka, že již neexistuje spojení s tímto zařízením. Pokud jsou všechny tyto podmínky splněny, může dojít k inicializaci kontroléru. Při ní dochází k vytvoření instance tzv. Z-Wave Manageru, který dále poskytuje funkcionality pro vytvoření kontroléru. Jakmile je kontrolér vytvořen, je otevřen patřičný sériový port a volány další potřebné metody. Poté je ještě inicializován časovač pro zpracovávání fronty příkazů a dále jsou načteny zařízení přidané v kontroléru.

Načtení zařízení z kontroléru zajišťuje metoda `LoadNodes`. Ta pro každé zařízení v kontroléru, kromě prvního (což je sám kontrolér), zajistí zjištění generické a specifické třídy

zařízení. K zjištění těchto údajů je využita třída `DeviceInfoUtils`. Tato třída také využívá návrhový vzor Singleton, aby v rámci aplikace existovala jen jedna instance a nebylo třeba vždy vytvářet instance nové a znovu načítat veškeré potřebné XML soubory apod. Tato třída obsahuje všechny potřebné informace, ve kterých je možno vyhledávat. K prohledávání patřičných kolekcí dat je využit LINQ. Následně je na základě získaných dat vytvořena instance třídy `DeviceCore` a je přidána do seznamu (přesněji slovníku) zařízení. Ukázka popisované metody je přiložena níže:

```
private void LoadNodes()
{
   foreach (var node in Controller.GetNodes().Skip(1))
   {
      var generic = DeviceInfoUtils.Instance.GenericDevicesList.Find(
      x => x.GenDevKey == node.Generic);

      var specific = generic.SpecificDevices.Find(
      x => x.SpecDevKey == node.Specific);

      var deviceCore = new DeviceCore(node.Id, generic.GenDevName,
          generic.GenDevName, specific.SpecDevName);

      Devices.Add(node.Id, deviceCore);
   }
}
```

4.6.1 Odesílání příkazů

Každý odesílaný příkaz je tvořen objektem typu `ZWaveCommand`. Ten obsahuje *Node ID* cílového zařízení, samotná data příkazu a příznak, zda je po odeslání příkazu třeba čekat na odpověď (typicky při odesílání příkazů typu *Get*). Kromě toho ještě může obsahovat informaci o hlavičce očekávaného příkazu, který by měl být přijat jako odpověď.

Všechny příkazy, které se mají odeslat, jsou nejprve zařazovány do fronty příkazů. Jde o instanci klasické kolekce typu `Queue`, ve které se data vkládají na konec a odebírají ze začátku. Ihned po zařazení příkazu do fronty je zavolána metoda `ProcessQueue`. Jejím hlavním úkolem je zajistit, aby data byla odeslána až v případě, kdy není očekávána odpověď na předchozí odeslané příkazy. Toto specifikuje atribut `_waitingForResponse`, který je nastaven na hodnotu `true` v případě, kdy je třeba před odesláním dalších dat počkat na odpověď na předchozí příkaz. Pokud je atribut nastaven na hodnotu `false`,

znamená to, že odpověď již byla přijata, případně nebyla ani požadována. V tuto chvíli je možné příkaz načíst z fronty a odeslat do zařízení.

Níže jsou pro lepší představu přiloženy ukázky metod pro poslání příkazu *Basic Set*, u kterého není očekávána odpověď a dále příkazu *Manufacturer Specific Get*, u kterého je naopak odpověď očekávána. V druhém zmíněném případě je patrné, že jako poslední parametr při vytvoření instance třídy `ZWaveCommand` je použita právě hlavička (první dva byty) očekávaného příkazu, který má přijít jako odpověď.

```
public void SendBasicOn(byte nodeId)
{
   CommandQueue.Enqueue(
   new ZWaveCommand(nodeId, new byte[] { 0x20, 0x01, 0xFF }, false)
   );
   ProcessQueue();
}

public void SendManufacturerSpecificGet(byte nodeId)
{
   CommandQueue.Enqueue(
   new ZWaveCommand(nodeId, new byte[] { 0x72, 0x04 }, true, new byte[] { 0x72,
       0x05 })
   );
   ProcessQueue();
}
```

4.6.2 Příjem dat

Příjem dat ze zařízení do kontroléru je řešen samostatným vláknem. Toto vlákno je vytvořeno vždy, pokud dojde k příjmu dat ze zařízení do kontroléru a zajišťuje ho metoda `ControllerOnApplicationCommandHandlerEvent`. Ve vzniklém vláknu je pak využívána funkce `ReceiveData`, která jako parametr přijímá právě ona přijatá data. V této funkci poté dochází k přetypování přijatých dat do potřebného datového typu. Data obsahují kromě jiného i informaci o tom, z jakého zařízení byla odeslána. Právě tato informace je využita k vyhledání patřičného zařízení, přesněji jeho jádra – tedy instance třídy `DeviceCore`. Pokud je tato instance nalezena, je poté volána její metoda pro zpracování přijatých dat nazvaná `ProcessReceivedData`. Ukázka funkce řešící příjem dat je přiložena níže.

```
private void ReceiveData(object receivedData)
{
   var data = (DeviceAppCommandHandlerEventArgs)receivedData;

   // Find matching DeviceCore
   var senderDeviceCore = Devices[data.SourceNodeId];

   if (senderDeviceCore != null)
   {
      // Call method of device which data belongs to
      senderDeviceCore.ProcessReceivedData(data);
   }
}
```

4.7 Třída DeviceInfoUtils

Třída `DeviceInfoUtils` slouží jako podpůrná třída pro třídy ostatní. Kromě různých pomocných metod obsahuje seznamy generických tříd zařízení, kde ke každé generické třídě náleží navíc seznam specifických tříd zařízení. Tyto seznamy jsou vytvořeny na základě XML dokumentu, jehož zpracování je dalším z úkolů popisované třídy. Dalším úkolem je načítání konfiguračních parametrů jednotlivých zařízení z XML souborů. Třída dále také udržuje seznam známých obchodních označení zařízení. Všechny udržované kolekce jsou pak zdrojem dat pro ostatní třídy. Jsou využívány pro zjištění generických a specifických tříd zařízení na základě jejich číselného označení. Také tato třída využívá návrhový vzor *Singleton*. Ten je využit z toho důvodu, aby nebylo třeba při každém vyhledávání dat tvořit novou instanci, znovu načítat a zpracovávat veškeré soubory a podobně.

Mezi pomocné metody zmíněné v úvodu kapitoly patří například metoda `CreateNiceName`. Tato metoda zajišťuje převod nehezky vypadajících názvů generických a specifických tříd zařízení získaných z XML dokumentu. Jako příklad lze uvést název generické třídy *GENERIC_TYPE_SENSOR_MULTILEVEL* získaný ze souboru. Text je nejprve rozdělen na jednotlivá slova, jako oddělovač je použit znak podtržítka. Poté je každé dále používané slovo upraveno tak, aby první písmeno zůstalo velké a zbylá písmena malá. Výsledná slova jsou pak spojena do jednoho řetězce a použita jako návratová hodnota funkce. První dvě slova jsou vynechána, protože nejsou potřebná (v tomto případě slovo Generic a Type). Zdrojový kód funkce vypadá následovně:

```
private static string CreateNiceName(string original)
{
   var separated = original.Split('_');

   for (int i = 2; i < separated.Length; i++)
   {
      // Make first letter uppercase
      separated[i] = separated[i].First() + separated[i].Substring(1).ToLower();
   }

   // Join with spaces, skip first two words
   return string.Join(" ", separated.Skip(2));
}
```

Jak bylo zmíněno, jedním z hlavních úkolů popisované třídy je také načtení a uchovávání seznamu všech používaných generických a specifických tříd zařízení. K načtení z XML souboru je použita třída `XDocument` společně s technologií LINQ. Jednotlivé generické třídy představují tagy *<gen_dev>*. Uvnitř těchto tagů jsou pak uloženy tagy popisující jednotlivé specifické třídy. Pomocí jednoho dotazu jsou tedy získána všechna potřebná data – jméno generické třídy, její číselné označení a následníci tagu (tedy tagy obsahující informace o specifických třídách). Dotaz pak vypadá následovně:

```
var query = from c in xml.Root.Descendants("gen_dev")
         select new
         {
            Name = c.Attribute("name").Value,
            Key = c.Attribute("key").Value,
            Children = c.Descendants("spec_dev")
         };
```

Získaná data jsou následně v cyklu procházena. Ze získaných dat je vytvořena instance třídy `GenericDevice`, která obsahuje číselný kód generické třídy (podle něj může být následně vyhledávána), název a také seznam příslušných specifických tříd. Ten je naplněn hned vzápětí dalším cyklem, který prochází všechny následníky, tedy specifické třídy. Opět jsou vytvářeny instance, tentokrát třídy `SpecificDevice`. Ta obsahuje jen číselný kód a název specifické třídy. Následně je instance třídy `GenericDevice` uložena do kolekce. Zdrojový kód je přiložen níže.

```
foreach (var genClass in query)
{
   var genericClass = new GenericClass(Convert.ToByte(genClass.Key, 16),
       genClass.Name);

   foreach (var xElement in genClass.Children)
   {
      genericClass.SpecificClasses.Add(new
          SpecificClass(Convert.ToByte(xElement.Attribute("key").Value, 16),
          xElement.Attribute("name").Value));
   }

   genericClasses.Add(genericClass);
}
```

4.8 Přidání a odebrání zařízení

Nové zařízení je přidáváno, respektive odebíráno, na třech úrovních. V prvním případě je nejprve nutné přidat nové zařízení (přesněji node – česky uzlu) do Z-Wave sítě. To v praxi znamená, že je třeba přidat ho do kontroléru. Nejprve je třeba kontrolér přepnout do režimu, ve kterém je umožněno přidání nového zařízení. Jakmile je kontrolér přepnut do zmíněného režimu, musí uživatel provést se zařízením patřičnou akci pro jeho přidání do sítě. Typicky jde o stisknutí tlačítka, otočení potenciometru do dané polohy apod.

Jakmile dojde k přidání zařízení do sítě, následuje druhá úroveň přidávání. Tou je zjištění informací o zařízení, nastavení asociační vazby s kontrolérem a přidání zařízení do seznamu udržovaného třídou `ZWaveControllerConnection`. Nejprve je s využitím informací ze třídy `DeviceInfoUtils` zjištěna generická a specifická třída zařízení. Poté následuje vytvoření asociační vazby mezi nově přidaným zařízením a kontrolérem. K této vazbě je využita první asociační skupina, nazývaná *Lifeline*. Zařízením, která jsou uložena v této asociační skupině, jsou pravidelně posílány stavové informace o zařízení – např. úroveň svitu, hodnota okolního světla aj. Aplikace na tyto informace pak může nějakým způsobem reagovat. V případě popisované aplikace je zpracováván příkaz *Switch Multilvel Report*, případně *Basic Report*. Na základě něj je aktualizován popisek zobrazující aktuální stav zařízení. Není tak třeba se v určitých časových intervalech na tento stav dotazovat – zařízení ho sděluje samo. Poté je vytvořena instance třídy `DeviceCore`, která je přidána do zmíněného seznamu zařízení.

Jakmile je zařízení přidáno do seznamu, dochází k poslední úrovni přidávání, a tou je vytvoření grafické komponenty reprezentující zařízení přímo v okně aplikace, tedy vytvoření instance třídy `Device`. Ta kromě jiného obsahuje referenci na instanci třídy, která byla předtím přidána do seznamu zařízení. Jakmile je grafický prvek vytvořen, proběhne jeho inicializace, tedy zjištění výrobce, typu apod. Poté je také prvku přiřazena metoda pro obsluhu kliknutí myší, tak aby bylo možno ho přesouvat mezi panely a také aby byl umožněn pravý klik, který vyvolává kontextové menu. To obsahuje nabídku příkazů, které je možno zařízení poslat. Poté je již zařízení přidáno do panelu všech zařízení.

Při odebírání zařízení je kontrolér opět uveden do specifického režimu, tentokrát pro odebírání ze sítě. Jakmile je kontrolér ve zmíněném režimu, uživatel opět musí provést se zařízením patřičnou akci, tedy například stisknout tlačítko apod. Jakmile dojde k požadované akci, zařízení je odebráno z kontroléru. Poté následuje odstranění ze seznamu zařízení ve třídě `ZWaveControllerConnection` a také k odebrání grafické komponenty zařízení z okna aplikace. Zde je situace poněkud složitější než v případě přidávání. Je totiž třeba identifikovat panel, ve kterém se zařízení nachází. K tomu poslouží vlastnost `Parent`, dle které je možno zjistit, který panel je rodičovským panelem daného prvku. Zároveň je také potřeba zrušit případné vazby mezi ostatními zařízeními a právě odebraným zařízením. Odstranění vazeb mezi odebraným zařízením a ostatními zařízeními není již možno řešit. Jeho *Node ID* je totiž možno zjistit až ve chvíli, kdy je zařízení odebráno z kontroléru. V tu chvíli s ním ale již není možná jakákoliv komunikace, a tedy ani odstranění vazeb. Naštěstí na tento případ výrobci ve většině případů myslí a zařízení je při jakémkoliv opětovném přidání do sítě resetováno do výchozího nastavení – tím jsou i smazány veškeré asociace. Jakmile jsou všechny zmíněné činnosti dokončeny, dochází k reorganizaci prvků v panelu všech zařízení, tak aby se eliminovala případná vzniklá „díra“ po odebraném zařízení.

Obě činnosti, tedy přidávání i odebírání zařízení, jsou řešeny ve svých samostatných vláknech. To je provedeno proto, aby nedocházelo k zablokování vlákna, ve kterém běží samotné okno aplikace. Tím je umožněno, aby během přidávání nebo odebírání zařízení šlo stále klikat na tlačítka v zobrazeném okně, což jinak nebylo možné. Operace jsou rozděleny vždy do dvou metod. Pro přidání zařízení slouží metody `AddDevice` a metoda `AddDeviceFinal`, pro odebírání pak `RemoveDevice` a `RemoveDeviceFinal`. První zmíněná metoda vždy zahájí samotnou operaci a pokud je přidání, případně odebírání zařízení z kontroléru provedeno,

je vyvolán tzv. delegát, který zajistí zavolání druhé funkce a předá jí potřebný parametr (*Node ID* zařízení).

Při kliknutí na tlačítko pro přidání nebo odebrání zařízení je zobrazeno okno, které instruuje uživatele k určité akci se zařízením (stisknutí tlačítka) a zároveň umožňuje přidávání nebo odebírání zařízení zrušit stisknutím tlačítka *Cancel*. Právě z důvodu možného přerušení operace uživatelem, je událost uzavření zmíněného okna zpracovávána. Pokud dojde k uzavření okna, ale nedošlo k přidání, respektive odebrání, zařízení, musí přejít k přepnutí kontroléru do standardního režimu. To se, poněkud netradičně, provádí opětovným voláním funkce, která byla předtím volána právě pro uvedení kontroléru do současného stavu. V tomto případě je třeba rozlišovat, jaký byl typ okna. Jsou rozlišovány dva typy okna – pro přidávání zařízení a odebírání zařízení. V závislosti na tomto typu je pak volána patřičná funkce pro přepnutí kontroléru a následně je vlákno ukončeno.

4.9 Identifikace zařízení

Jedním z požadavků na aplikaci byla možnost snadného rozpoznávání jednotlivých zařízení. K získání potřebných informací o výrobci a obchodním označení zařízení mohou být využity příkazy tříd *Manufacturer Specific*. Pro získání názvu zařízení je určena metoda `GetDeviceName` ze třídy `Device`. Tato metoda zajistí zaslání příkazu *Manufacturer Specific Get* danému zařízení. Na tento příkaz poté zařízení odpovídá odesláním příkazu *Manufacturer Specific Report* do kontroléru.

Jakmile jsou data v kontroléru přijata, proběhne jejich zpracování. Zpracování probíhá v metodě `ProcessManufacturerSpecificReport` ze třídy `DeviceCore`. Protože aplikace je primárně určena pro správu a konfiguraci zařízení od firmy Steinel, je nejprve ověřeno, zda je zařízení právě od této firmy. Výrobce lze rozpoznat dle prvních dvou bytů přijatého příkazu. Pokud první byte obsahuje hodnotu 0x02 a druhý byte hodnotu 0x71, znamená to, že zařízení je od firmy Steinel. Detailnější identifikaci poskytuje hodnota *Device ID*. Jeho hodnota je získána z patřičných bytů přijatého příkazu. Zde už je nutné pro výpočet hodnoty použít operace bitový posun a bitový součet. Získaná hodnota je použita pro vyhledání obchodního označení zařízení. Všechna dostupná obchodní označení jsou uložena ve slovníku – tedy datové kolekci typu `Dictionary`. V ní pak probíhá vyhledávání pomocí

klíče. V tomto případě je klíčem právě hodnota *Device ID* daného zařízení. Pokud je záznam ve slovníku nalezen, dojde k jeho použití jako názvu zařízení. Pokud by záznam ve slovníku neexistoval (např. při testování prototypu zařízení, které ještě nemá specifikované *Device ID*), použije se obecný název *STEINEL device*. Pro lepší představu je níže uvedena část obsahu výše popisované metody.

```
// If manufacturer is STEINEL
if (receivedData[0] == 0x02 && receivedData[1] == 0x71)
{
   IsSteinelDevice = true;
   DeviceId = receivedData[4] << 8 | receivedData[5];

   // Find device product name by Device ID
   if (DeviceInfoUtils.Instance.DeviceNames.ContainsKey(DeviceId))
   {
      DeviceName = DeviceInfoUtils.Instance.DeviceNames[DeviceId];
   }
   else
   {
      DeviceName = "STEINEL device";
   }
}
// if manufacturer is not STEINEL
else
{
   DeviceName = "Unknown device";
}
```

Po získání názvu zařízení je vyvolána událost `NameChanged`. Pro obsluhu této události je registrována metoda `DeviceCoreOnNameChanged` ze třídy `Device`. Tato metoda zajistí aktualizaci popisku s názvem zařízení. Pokud se jedná o zařízení od firmy Steinel, nastaví mimo popisku i obrázek zařízení.

4.10 Tvorba a rušení asociačních vazeb

K tvorbě asociačních vazeb mezi zařízeními je využito asociačních skupin poskytovaných přímo protokolem Z-Wave. V rámci aplikace asociace vzniká, pokud je zařízeno umístěno do levého panelu v hlavním okně – tzv. panelu skupiny zařízení. Jakmile dojde k přesunu zařízení do zmíněného panelu, je volána metoda `AddDeviceToGroup`. Ta jako parametr přijímá *Node ID* zařízení, které chceme do skupiny přidat a dále identifikátor skupiny, do

které má být přidáno. Prvním krokem je ověření, zda jsou všechna zařízení ve skupině funkční – tedy zda mají vlastnost `IsResponding` nastavenou na hodnotu `true`. Pokud by například jedno ze zařízení, již umístěných ve skupině, nebylo dostupné (bylo odpojené od napájení, mimo dosah apod.), nedošlo by ke správnému vytvoření asociačních vazeb. Pokud je zjištěno, že jedno nebo více zařízení není dostupné, je na to uživatel upozorněn a má možnost akci přerušit. Pokud je akce přerušena, funkce vrací hodnotu `false` a tvorba asociačních vazeb ani přesun grafické komponenty zařízení z jednoho panelu do druhého dále nepokračují. V opačném případě jsou zjištěna *Node ID* všech zařízení, která se již nachází na panelu skupiny. Ta jsou následně uložena do pole. Toto pole je následně v cyklu procházeno. Každému zařízení je poslán příkaz *Association Set*. Jako *Node ID* zařízení, se kterým je tvořena asociační vazba, je použito právě *Node ID* zařízení nově přesunutého na panel. Jakmile je toto dokončeno, je třeba zajistit, aby i nově přidané zařízení mělo nastavené asociační vazby na všechny ostatní zařízení ve skupině. Je mu tedy rovněž poslán příkaz *Association Set*. Jako seznam zařízení, se kterými má být vytvořena asociační vazba, je použit výše zmíněný seznam zařízení již umístěných ve skupině. Pro snazší pochopení je část metody, která řeší právě tvorbu asociačních vazeb, uvedena níže. Za povšimnutí stojí využití technologie LINQ pro získání pole hodnot *Node ID* všech zařízení ve skupině.

```
// Get IDs of devices in group
var otherDevicesNodeIds = _devicesInGroup.Select(
device => device.DeviceCore.NodeId).ToArray();

// For every device in group set association to new device
foreach (var otherDeviceNodeId in otherDevicesNodeIds)
{
   ZWaveControllerConnection.Instance.SendAssociationSet(
   otherDeviceNodeId, groupId, deviceNodeId);
}

// Set association between dropped device and other devices in group
if (_devicesInGroup.Count > 0)
{
   ZWaveControllerConnection.Instance.SendAssociationSet(
   deviceNodeId, groupId, otherDevicesNodeIds);
}
```

Pokud má být naopak asociace mezi zařízeními zrušena, je třeba zařízení přesunout z levého panelu do pravého – tzv. panelu všech zařízení. K samotnému zrušení vazeb s tímto

zařízením slouží metoda `RemoveDeviceFromGroup`. Její princip je velmi podobný výše zmíněné metodě pro tvorbu asociačních vazeb, jen místo tvorby vazeb dochází k jejich rušení. Opět je ověřeno, zda jsou všechna zařízení ve skupině dostupná. Pokud ne, uživatel má opět možnost celou akci přerušit. V tom případě k rušení vazeb ani k přesunu komponenty z jednoho panelu do druhého nedojde. V opačném případě je třeba opět získat *Node ID* všech zařízení ve skupině. Z tohoto seznamu je ale vyjmuto ono zařízení, se kterým má být zrušena vazba (tedy to, které je přesouvané do panelu všech zařízení). Ukázka kódu pro získání zmíněného seznamu (přesněji řečeno pole) je uvedena níže.

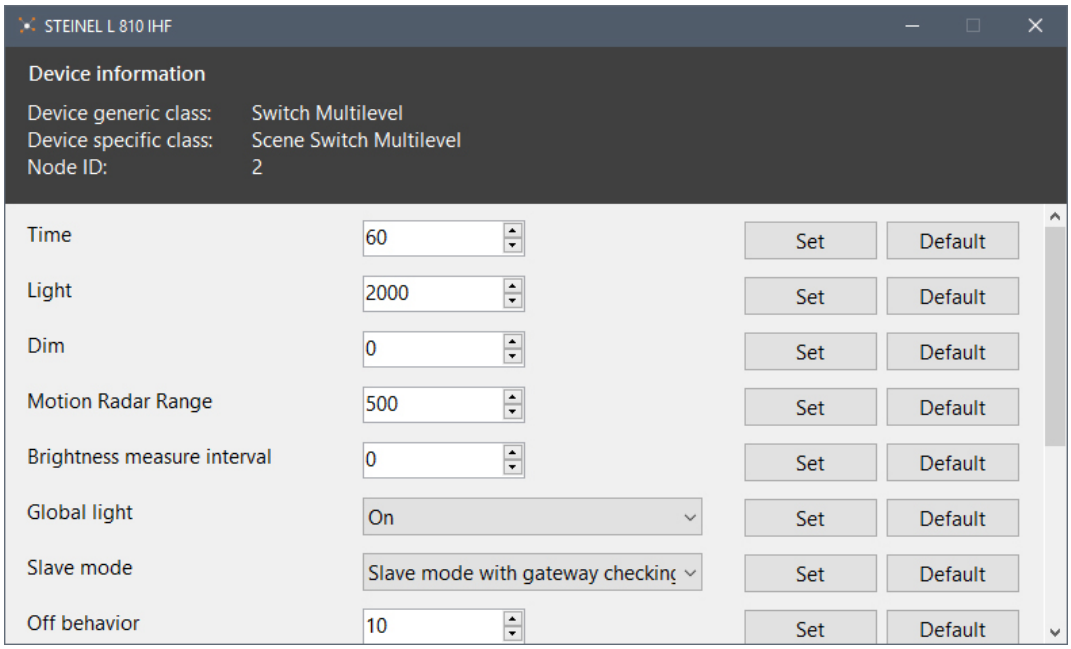
```
// Get IDs of devices in group
var otherDevicesNodeIds = _devicesInGroup.Where(
device => device.DeviceCore.NodeId != deviceNodeId).Select(
device => device.DeviceCore.NodeId).ToArray();
```

Na všechna *Node ID*, která jsou uložena v získaném poli, je poté poslán příkaz *Association Remove*. Jako *Node ID* zařízení, které se má odstranit, je použito právě *Node ID* zařízení, které přesouváme pryč ze skupiny. I zde je potřeba následně zrušit vazby mezi odstraňovaným zařízením a ostatními zařízeními ve skupině. K tomu je opět využit získaný seznam ostatních zařízení.

4.11 Získávání a nastavování parametrů

Jednou z požadovaných funkcí aplikace je možnost nastavení parametrů zařízení. K tomuto účelu slouží třída `DeviceInfoWindow`. Ta představuje okno, které se zobrazuje při stisku ikonky s písmenem „i“ na grafické komponentě zařízení. Jak zmíněné okno vypadá, je ukázáno na obrázku 11.

Okno se skládá ze dvou vodorovně oddělených panelů. V horním panelu se nachází základní informace o zařízení. Obsah dolního panelu je závislý na tom, zda je k dispozici XML dokument, který obsahuje informace o nastavitelných parametrech zařízení. V případě, že je XML dokument pro dané zařízení k dispozici, do panelu jsou umístěny další malé panely, pro každý parametr jeden. Každý z těchto panelů obsahuje popisek s názvem parametru, pole pro zadání číselné hodnoty, nebo rozbalovací seznam (v závislosti na typu parametru) a tlačítka pro nastavení požadované hodnoty do zařízení, případně pro



Obrázek 11: Okno pro nastavování parametrů

nastavení výchozí hodnoty. Po umístění každého z těchto panelů je do zařízení vyslán příkaz Configuration Get, na který zařízení odpovídá zasláním Configuration Reportu. Jakmile jsou data přijata, je v daném panelu aktualizován obsah pole pro zadání číselné hodnoty, případně je nastavena odpovídající hodnota v rozbalovacím seznamu. Výše popsaný panel je implementován ve třídě `ParameterPanel`. Tato třída kromě výše uvedených grafických prvků obsahuje také metody, které při stisku jednoho z tlačítek zajišťují zaslání hodnot do zařízení. U každé hodnoty nastavovaného parametru je důležité rozhodnout, zda jde o datový typ `byte`, nebo `short`. Část obsahu XML dokumentu, obsahujícího informace o parametrech zařízení, zobrazuje obrázek 12.

V případě, že pro dané zařízení není XML dokument k dispozici, je namísto několika panelů použit jen panel jeden. Tento panel je implementován ve třídě `SingleParameterPanel`. Obsahuje pole pro zadání čísla parametru, jeho datového typu a hodnoty. Dále obsahuje tlačítko pro nastavení parametru a také pro získání jeho hodnoty ze zařízení na základě čísla parametru. Získaná hodnota je případně načtena do textového pole pro zadávání hodnoty parametru. V případě použití tohoto panelu se uživatel musí podívat do návodu k zařízení, kde jsou uvedena jednotlivá čísla parametrů, jejich datové typy a rozsahy hodnot.

```
<Value type="short" genre="config" instance="1" index="2"
label="Light" units="Lux" min="0" max="2000" value="2000">
    <Help>
    Ambient Light treshold.
    0 - run Learn ambient light sequence.
    1 - not allowed, interpreted as 2 Lux
    2000 - is used as daylight. Still Night mode.
    Can also be set by the poti.
    </Help>
</Value>
```

Obrázek 12: Ukázka obsahu XML dokumentu popisujícího parametry zařízení

K samotnému zaslání příkazu pro nastavení konfiguračního parametru je využita metoda `SendConfigurationSet` ze třídy `ZWaveControllerConnection`. Ta je přetížena a liší se právě datovým typem hodnoty předávané v parametrech. Jednodušším případem je posílání parametru datového typu `byte`. Zde není třeba žádné další úpravy dat a může rovnou dojít k odeslání příkazu *Configuration Set*. Složitějším případem pak je, pokud je parametr datového typu `short`. V tomto případě musí dojít k „rozdělení“ hodnoty na dva byty. K tomuto účelu je využito metody `GetBytes` ze třídy `BitConverter`. Získané pole bytů pak může být využito v příkazu *Configuration Set*. Níže je přiložena ukázka obsahu popisované metody `SendConfigurationSet`, která zajišťuje poslání příkazu pro nastavení parametru typu `short`.

```
var bytes = BitConverter.GetBytes(value);
CommandQueue.Enqueue(new ZWaveCommand(nodeId,
new byte[] { 0x70, 0x04, parameterId, 2, bytes[1], bytes[0] }, false));
```

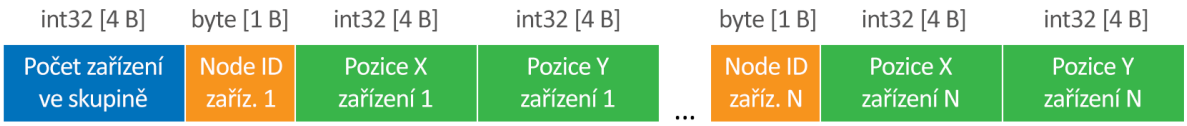
Jak bylo uvedeno výše, nejprve dochází k získání pole bytů z předávané hodnoty typu `short`. Samotný příkaz pak na prvním a druhém místě obsahuje hodnotu třídy a typu příkazu (0x70 a 0x04). Následuje číslo parametru a počet bytů daného datového typu (v případě typu `short` je hodnota dva). Poté následují dva byty získané v prvním kroku. Zde je třeba zdůraznit, že záleží na jejich pořadí. Druhý získaný byte musí být poslán jako první, protože je zde využita endianita[2] typu *Big-endian*.

[2]Endianita určuje pořadí ukládání jednotlivých bytů. Definuje, v jakém pořadí jsou v operační paměti ukládány jednotlivé řady čísel, které mají více jak jeden byte.

4.12 Uložení a načtení konfigurace GUI

Aby bylo zajištěno, že po uzavření a opětovném otevření aplikace nedojde ke zrušení asociací mezi jednotlivými zařízeními, je nutno uchovávat informace o konfiguraci grafického uživatelského prostředí. Konfigurací se rozumí pozice jednotlivých grafických znázornění zařízení (dále jen zařízení) v rámci panelu skupiny. Je totiž podstatné, zda se zařízení nachází v panelu skupiny (umístěném vlevo), nebo v panelu všech zařízení (umístěném vpravo). Všechna zařízení umístěna v levém panelu tvoří skupinu. Existují mezi nimi tedy asociační vazby, které je třeba zachovat i po zavření a opětovném otevření aplikace. Asociačním vazbám se podrobněji věnuje kapitola 4.10.

Samotná grafická konfigurace je uložena ve formě binárního souboru *guiconf.bin*. Do tohoto souboru je nejprve uložen počet zařízení, která se nacházejí ve skupině. Dále se do něj ukládají informace o jednotlivých zařízeních. Mezi ukládané informace patří identifikační číslo zařízení v rámci kontroléru (*Node ID*) a hodnoty grafických souřadnic X a Y v rámci panelu. Struktura souboru je naznačena na obrázku 13.



Obrázek 13: Struktura souboru s grafickou konfigurací

Hodnoty jsou do souboru zapisovány s využitím třídy `BinaryWriter`. Pokud soubor s grafickou konfigurací neexistuje, je vytvořen. V případě, že soubor již existuje, je přepsán. Pokud by během ukládání souboru došlo k problému, je vyvolána výjimka a uživatel je na tuto skutečnost upozorněn dialogovým oknem. K načtení grafické konfigurace je využita třída `BinaryReader`. Ze souboru je nejprve načten počet uložených zařízení. Poté je pro každé zařízení zjištěno jeho identifikační číslo a souřadnice. Na základě identifikačního čísla je pak pomocí funkce `FoundDeviceControlByNodeId` nalezen odpovídající grafický prvek v panelu všech zařízení. Poté je zařízení přesunuto na panel skupiny a zároveň odebráno z panelu všech zařízení. Po přesunu musí dojít k reorganizaci prvků v panelu všech

zařízení. Tu zajišťuje funkce `ReorganizePanelAllDevices`, jejíž implementace je podrobněji popsána v kapitole 4.3. Protože od posledního použití aplikace mohlo dojít ke změně rozlišení monitoru, jsou grafické souřadnice každého zařízení upraveny pomocí funkce `GetCorrectControlLocation`. Tato funkce je rovněž podrobněji popsána ve výše zmíněné kapitole.

4.13 Testování aplikace

Protože aplikace byla vyvinuta primárně pro účely firmy Steinel, k jejímu testování byla využita všechna dostupná zařízení od zmíněné firmy, podporující protokol Z-Wave. Kromě nich byla také použita dvě zařízení od firmy Fibaro, která je velkým hráčem na trhu se Z-Wave zařízeními. Konkrétně byla pro testování aplikace využita následující zařízení:

- STEINEL L 810 iHF – nástěnné svítidlo, která pro detekování pohybu využívá vysokofrekvenční senzor,
- STEINEL RS LED D2 – interiérové svítidlo, které pro detekování pohybu také využívá vysokofrekvenční senzor,
- STEINEL XLED HOME 2 – nástěnné svítidlo, které pro detekování pohybu využívá infračerveného senzoru (tzv. PIR čidla),
- STEINEL IS 140-2 – nástěnné pohybové čidlo, pro detekování pohybu využívá infračervený senzor, stejný jako předchozí uvedené zařízení,
- Fibaro Wall Plug – adaptér do zásuvky, který umožňuje například měření spotřeby apod.,
- Fibaro Motion Sensor – pohybový senzor napájený baterií, využívající infračervený senzor.

K testování jednotlivých částí aplikace bylo využíváno první uvedené svítidlo – STEINEL L 810 LED Z-WAVE. Toto svítidlo bylo totiž z velké části vyvíjeno ve vývojovém oddělení firmy Steinel v Pardubicích, kde byla také realizována tato diplomová práce. Nebyl tak problém získat detailní informace o fungování zmíněného zařízení a dle toho ověřovat správnou funkčnost jednotlivých částí aplikace. Finální aplikace byla pak testována se všemi dostupnými zařízeními, uvedenými v seznamu výše. Během testování bylo odhaleno několik chyb, které byly nejčastěji způsobeny špatným pochopením některých funkcí protokolu

Z-Wave. Všechny odhalené chyby byly samozřejmě opraveny, tak aby aplikace vyhovovala požadavkům, které na ní byly kladeny. V jednu chvíli bylo v síti umístěno nejvíce osm zařízení – čtyři svítidla L 810, dále RS LED D2, XLED HOME 2, IS 140-2 a Fibaro Wall Plug. I v této konfiguraci zařízení v síti se aplikace chovala korektně. Testování aplikace bude dále probíhat některými zaměstnanci firmy Steinel a případné chyby budou opraveny.

5 Použití aplikace

Aplikace je určena pro operační systém Microsoft Windows XP SP3 a novější. V operačním systému musí být nainstalován Microsoft .NET Framework ve verzi 4.0 nebo novější. Pro provoz aplikace je třeba použít USB kontrolér od firmy Sigma Designs s označením UZB-EU. Před použitím aplikace musí být tento kontrolér připojen v portu USB. Pokud by při startu aplikace bylo zjištěno, že kontrolér není připojen, případně je využíván jinou aplikací, je na to uživatel upozorněn chybovou hláškou. Uživatel má v tuto chvíli možnost kontrolér připojit, případně ukončit aplikaci, která jej využívá. Poté je možno stisknout tlačítko pro opakování inicializace, případně pro ukončení aplikace.

Pokud je kontrolér připojen, je zobrazen tzv. Splash-screen. Tím je okno, které neobsahuje záhlaví s ovládacími prvky ani rámečky. Během zobrazení tohoto okna dochází k inicializaci všech zařízení, která jsou uložena v kontroléru. O stavu inicializace je uživatel informován – je zobrazena informace, kolikáté zařízení je právě inicializováno a kolik zařízení je v kontroléru celkem. Okno kromě toho také obsahuje informaci o verzi aplikace.

Jakmile jsou všechna zařízení inicializována, je zobrazeno hlavní okno aplikace. Standardně jsou všechna dostupná zařízení umístěna v pravém panelu. Pokud však již byla aplikace dříve spuštěna a byly mezi některými zařízeními vytvořeny asociační vazby (viz kapitola 4.10), je načten soubor s uloženou grafickou konfigurací. Jeho načtení zajistí přesunutí komponent znázorňujících jednotlivá zařízení (dále jen zařízení) do panelu skupiny na místa, kde byly umístěny před zavřením aplikace. V tuto chvíli je aplikace připravena k použití. Jednotlivá zařízení je možno libovolně přesouvat v rámci panelu. Pro zajištění větší přehlednosti je například možno PIR čidla umístit vlevo a světelné zdroje naopak vpravo apod. Pro přesun zařízení je třeba na něj kliknout levým tlačítkem myši a tlačítko držet. Během držení tlačítka myši může být zařízení umístěno kamkoliv v panelu, případně přesouváno mezi panely. Jakmile dojde k uvolnění tlačítka, zařízení je umístěno na místo, kde se aktuálně nachází kurzor. Kliknutí pravým tlačítkem myši na zařízení vyvolává kontextovou nabídku obsahující příkazy, které je možno zařízení poslat. Dostupné jsou příkazy *Basic On* a *Basic Off*. Kliknutím na jeden z příkazů dojde k jeho okamžitému poslání danému zařízení. Příkaz *Basic On* je možno také poslat najednou všem zařízením, která jsou umístěna ve skupině. To je možno provést kliknutím na ikonu žluté žárovky

v pravém horním rohu panelu skupiny. Naopak kliknutím na ikonu šedé žárovky je všem zařízením ve skupině poslán příkaz *Basic Off*.

U každého zařízení je možno kliknutím na ikonku s písmenem „i“ zobrazit okno, obsahující základní informace o zařízení a umožňující zjištění a nastavení hodnot parametrů. Horní část obsahuje název generické a specifické třídy zařízení (Generic a Specific Class) a dále jeho identifikační číslo (*Node ID*). Obsah dolní části závisí na dostupnosti XML dokumentu obsahujícího informace o parametrech zařízení. V případě, že dokument dostupný není, je zobrazeno pole pro zadání čísla parametru, seznam pro výběr datového typu a pole pro zadání hodnoty, která má být nastavena. Tyto údaje poskytuje výrobce v návodu k zařízení.

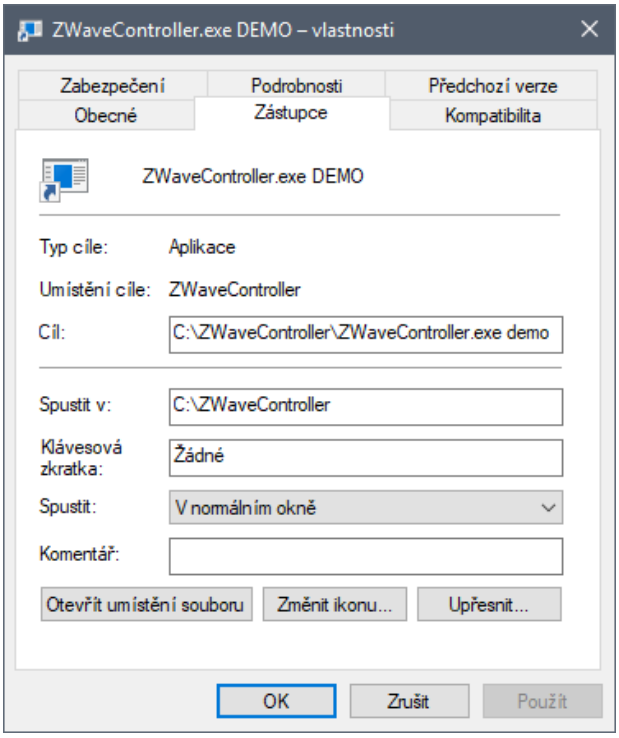
Pokud chceme přidat nové zařízení do kontroléru (a tím zároveň i do aplikace), je třeba kliknout na položku kontextového menu nazvanou *Device*. V zobrazené nabídce pak vybereme položku *Add Device*. Po kliknutí je zobrazeno okno, které uživatele instruuje k potřebné akci provedené se zařízením, typicky stisk tlačítka. Toto okno je možné stisknutím tlačítka *Cancel* zavřít a tím proces přidání zařízení zrušit. Pokud je zařízení přidáno, okno zmizí samo a v panelu všech zařízení se objeví právě přidané zařízení. Ihned poté je aktualizován jeho název a případně také zobrazený obrázek. To vše na základě přijatého příkazu *Manufacturer Specific Report*. Pro odebrání zařízení je pak třeba kliknout v menu na položku *Device*. Z nabídky pak ale vybereme položku *Remove Device*. Následně opět dojde k zobrazení okna, které uživatele instruuje k potřebné akci provedené se zařízením, jak bylo uvedeno v případě přidávání zařízení. Stejně jako v předchozím případě, i zde je možné akci odebírání přerušit stisknutím tlačítka *Cancel* v zobrazeném okně. Pokud je zařízení úspěšně odebráno, zmizí i z okna aplikace. V tomto případě nezáleží na tom, zda se zařízení nacházelo ve skupině, nebo ne. Případné asociační vazby jsou rušeny automaticky. V menu se dále nachází položka *Help*. Při kliknutí na ní se zobrazí nápověda k použití aplikace. Nápověda má formu samostatné webové stránky otevřené ve webovém prohlížeči.

Jak již bylo řečeno dříve, při uzavření aplikace dochází k automatickému uložení konfigurace grafického uživatelského rozhraní – tedy pozic jednotlivých zařízení v rámci panelu skupiny. Kromě toho také automaticky dochází k odpojení od USB Z-Wave kontroléru, tak aby nebyl sériový port dále blokován a zařízení mohla případně použít jiná aplikace.

5.1 Demo režim

Pro účely prezentace funkcionalit a designu aplikace bez nutnosti připojení USB kontroléru byl do aplikace přidán tzv. demo režim. V tomto režimu dochází k přidání pěti virtuálních zařízení – všech čtyř dostupných Z-Wave zařízení od firmy Steinel a jednoho obecného Z-Wave zařízení. V demo režimu, vzhledem k absenci připojeného kontroléru, nefunguje jakákoliv komunikace se zařízeními. Není také možno přidávat nová zařízení ani zařízení odebírat. Je možno přesouvat zařízení mezi jednotlivými panely, a tak vyzkoušet například funkcionalitu reorganizace grafických komponent v panelu všech zařízení apod. Také je možno zobrazit okno pro zobrazení informací o daném zařízení a nastavování parametrů. Dále je také možno zobrazit nápovědu k ovládání aplikace.

Pro používání aplikace v demo režimu je třeba ji spustit s parametrem `demo`. K tomu je možno využít zástupce, který bude mít v kolonce *Cíl* kromě cesty k spustitelnému souboru doplněn ještě zmíněný parametr. Jak by mohl zástupce vypadat, je ukázáno na obrázku 14.



Obrázek 14: Parametry zástupce pro spuštění aplikace v demo režimu

Závěr

Cílem této diplomové práce bylo vytvoření aplikace pro správu a konfiguraci zařízení využívajících komunikační protokol Z-Wave. Mezi hlavní požadavky na funkcionality aplikace patřilo přidávání a odebírání zařízení, jejich ovládání pomocí příkazů a také tvorba asociačních vazeb mezi nimi. Kromě toho aplikace měla umožňovat konfiguraci parametrů jednotlivých zařízení, přičemž seznam těchto parametrů, společně s jejich datovými typy a výchozími hodnotami, měl být načten z dostupných XML souborů. V případě nedostupnosti zmíněných souborů pak mělo být umožněno ruční zadávání pomocí kódu parametru, datového typu a jeho hodnoty. Dalším z požadavků na aplikaci byla také uživatelská přívětivost, snadné použití aplikace a kvalitní implementace.

V teoretické části této diplomové práce byly nejprve představeny některé další protokoly určené pro domácí automatizaci, jejich vlastnosti, výhody a nevýhody. V následující kapitole je podrobně popsán protokol Z-Wave. Nejprve je krátce představena jeho historie. Následuje popis technických aspektů protokolu – použitých frekvencí, vrstvového modelu protokolu apod. Dále jsou také popsány jednotlivé třídy zařízení a příkazů. Následně je představen jazyk C#, ve kterém byla aplikace naprogramována a některé jeho funkcionality, které byly při vývoji využity. Zbývající část práce se pak věnuje praktické implementaci aplikace. Zde je popsán význam všech použitých tříd. Některé z nich jsou pak rozebrány podrobněji v samostatných kapitolách. Dále je zde popsána implementace stěžejních funkcionalit aplikace. Poslední kapitola je pak věnována popisu použití aplikace. V této kapitole je také popsán tzv. demo režim, který umožňuje prezentaci aplikace bez nutnosti připojení USB kontroléru. V tomto režimu není logicky umožněna jakákoliv datová komunikace se zařízeními. Slouží pro demonstraci použití aplikace a dostupných funkcí.

Aplikace splňuje všechny požadavky, které na ni byly kladeny. Navíc byly po konzultaci s vedoucím práce přidány i některé funkce navíc. Jde například o možnost zapnutí, respektive vypnutí všech zařízení, které tvoří asociační skupinu, zobrazení aktuálního stavu u každého zařízení a podobně. Navíc byla přidána nápověda k použití aplikace, kde jsou vysvětleny všechny základní funkcionality aplikace. Pro vývoj aplikace bylo stěžejní nastudování principů a vlastností protokolu Z-Wave. Velice důležité bylo zejména pochopení funkce asociačních vazeb a také typů jednotlivých příkazů, které jsou pro

komunikaci mezi zařízeními používány. Aplikace je v tuto chvíli plně funkční a byla již v praxi využívána během tzv. před-certifikace jednotlivých zařízení. Dále může sloužit pro vnitropodnikové účely firmy Steinel, pro kterou byla vyvíjena a také jako zdroj know-how pro nově vznikající aplikace.

Největším problémem při vývoji aplikace i psaní samotné práce byl nedostatek literatury, která by se věnovala čistě protokolu Z-Wave. Během celého vývoje tak byla používána hlavně oficiální dokumentace, jejíž pochopení bylo ale často velmi náročné. Během samotné implementace se žádné zásadní problémy neobjevily. Jedním z nejtěžších úkolů bylo vyřešení řízení komunikace mezi kontrolérem a zařízeními. Velkým přínosem byla spolupráce přímo s vývojáři softwaru (přesněji spíše firmwaru) pro zařízení značky Steinel. Korektnost řešení některých funkcionalit tak mohla být ověřována prakticky ihned po jejich implementaci. Další výhodou byla dostupnost většího množství Z-Wave zařízení, se kterými mohla být aplikace testována.

Literatura

[1] CAPULIAKOVÁ, Kristína. Komunikačný štandard domácej automatizácie X10. *POSTERUS.sk, portál pre odborné publikovanie ISSN 1338-0087* [online]. 2010, 3(5) [cit. 2017-03-10]. Dostupné z: http://www.posterus.sk/?p=7366

[2] The X10 Home Automation Protocol - Specs and History. *BuildYourSmarthome.co* [online]. BuildYourSmarthome.co, 2014 [cit. 2017-03-10]. Dostupné z: http://buildyoursmarthome.co/home-automation/protocols/x10/

[3] Standardizace. *KNX Association [Official website]* [online]. Belgie [cit. 2017-03-15]. Dostupné z: https://www.knx.org/cz/knx/technika/standardizace/index.php

[4] The KNX standard – the basics. *KNX Finland: KNX - ratkaisusi nykyaikaiseen rakentamiseen ja ohjauksiin* [online]. [cit. 2017-04-23]. Dostupné z: knx.fi/download.php?liite_id=9748

[5] Úvod do KNX. *HW server s.r.o.* [online]. Praha, 2013 [cit. 2017-03-15]. Dostupné z: http://automatizace.hw.cz/teorie-a-praxe/knx.html

[6] The ZigBee Home Automation Protocol - Specs and Benefits. *BuildYourSmarthome.co* [online]. BuildYourSmarthome.co, 2014 [cit. 2017-03-10]. Dostupné z: http://buildyoursmarthome.co/home-automation/protocols/zigbee/

[7] ELAHI, Ata. a Adam GSCHWENDER. *ZigBee Wireless Sensor and Control Network*. Upper Saddle River, NJ: Prentice Hall, 2010. ISBN 0-13-713485-1.

[8] AGARWAL, Tarun. ZigBee Wireless Technology Architecture and Applications. *ElProCus* [online]. India: ElProCus, 2014 [cit. 2017-03-10]. Dostupné z:http: //www.elprocus.com/what-is-zigbee-technology-architecture-and-its-applications/

[9] PAETZ, Christian. *Z-wave Basics: Remote Control in Smart Homes.* S.l.: Create Space, 2013. ISBN 978-1490537368.

[10] Z-Wave Alliance Vision And Mission. *Z-Wave Alliance* [online]. USA: The Z-Wave Alliance, 2017 [cit. 2017-03-02]. Dostupné z: http://z-wavealliance.org/alliance_vision_and_mission

[11] *Z-Wave the Smartest Choice for your Smart Home* [online]. USA: Sigma Designs, 2017 [cit. 2017-03-02]. Dostupné z: http://www.z-wave.com

[12] Využívání vymezených rádiových kmitočtů. *Český telekomunikační úřad* [online]. Praha: ČTÚ, 2016 [cit. 2017-03-09]. Dostupné z: http://www.ctu.cz/vyuzivani-vymezenych-radiovych-kmitoctu

[13] Z-wave protocol stack | Z-wave protocol layer basics. *RF Wireless World* [online]. 2012 [cit. 2017-03-30]. Dostupné z: http://www.rfwireless-world.com/Tutorials/z-wave-protocol-stack.html

[14] Z-Wave Protocol Overview. In: *Tampere University of Technology: Dept. of Automation Science and Eng.* [online]. [cit. 2017-03-30]. Dostupné z: https://wiki.ase.tut.fi/courseWiki/images/9/94/SDS10243_2_Z_Wave_Protocol_Overview.pdf

[15] Z-wave MAC layer | Z-wave MAC frame types. *RF Wireless World* [online]. 2012 [cit. 2017-03-30]. Dostupné z: http://www.rfwireless-world.com/Tutorials/z-wave-MAC-layer.html

[16] Z-Wave Device Class Specification. In: *Z-Wave Public Specification* [online]. USA: Sigma Designs, 2016 [cit. 2017-03-09]. Dostupné z: http://z-wave.sigmadesigns.com/wp-content/uploads/2016/08/SDS10242-29-Z-Wave-Device-Class-Specification.pdf

[17] Z-Wave Command Class Specification, A-M. In: *Z-Wave Public Specification* [online]. USA: Sigma Designs, 2016 [cit. 2017-03-09]. Dostupné z: http://z-wave.sigmadesigns.com/wp-content/uploads/2016/08/SDS12657-12-Z-Wave-Command-Class-Specification-A-M.pdf

[18] NAKOV, Svetlin, Veselin KOLEV a kol. *Fundamentals of Computer Programming with C#: The Bulgarian C# Book*. 1. Bulharsko: Faber Publishing, 2013. ISBN 978-954-400-773-7.

[19] NAGEL, Christian. *C# 2005: programujeme profesionálně.* Brno: Computer Press, 2006. Programujeme profesionálně. ISBN 80-251-1181-4.

[20] Úvod do Windows Forms aplikací. *ITnetwork.cz* [online]. Praha [cit. 2017-03-23]. Dostupné z: http://www.itnetwork.cz/csharp/formulare/winforms/c-sharp-tutorial-windows-forms-okenni-aplikace-uvod

[21] C# Windows Forms Controls Examples. *Dot Net Perls* [online]. Sam Allen [cit. 2017-03-23]. Dostupné z: https://www.dotnetperls.com/controls

[22] How to: Create User-Defined Exceptions. *Microsoft Developer Network* [online]. [cit. 2017-03-23]. Dostupné z: https://msdn.microsoft.com/en-us/library/87cdya3t(v=vs.110).aspx

[23] LINQ v C# - revoluce v dotazování. *ITnetwork.cz* [online]. Praha [cit. 2017-03-30]. Dostupné z: http://www.itnetwork.cz/csharp/kolekce-a-linq/c-sharp-tutorial-linq-dotazy

[24] LINQ a EF - Úvod. *DotNETportal.cz* [online]. 2014 [cit. 2017-03-30]. Dostupné z: http://www.dotnetportal.cz/clanek/6413/LINQ-a-EF-Uvod

[25] LINQ: .NET Language-Integrated Query. *Microsoft Developer Network* [online]. [cit. 2017-03-31]. Dostupné z: https://msdn.microsoft.com/en-us/library/bb308959.aspx

[26] How to Start with XML and LINQ: A Beginner Guide. *CodeProject* [online]. 2012 [cit. 2017-04-07]. Dostupné z: https://www.codeproject.com/Articles/400269/How-to-Start-with-XML-and-LINQ-A-Beginner-Guide

[27] PECINOVSKÝ, Rudolf. *Návrhové vzory: [33 vzorových postupů pro objektové programování].* Brno: Computer Press, 2007. ISBN 978-80-251-1582-4.

Příloha A – Nápověda k aplikaci

Steinel Z-Wave Controller

Navigation

Introduction

This application allows you to manage and configure devices in your Z-Wave network. Z-Wave network is mainainted by USB controller. Before you start using this application, plug in the USB controller into your PC. Then open the application. Splash screen shows progress of initalization of the devices. It may take a few seconds to initialize all the devices in network. Afterwards the main window is shown and you can use the application. When you close main window, configuration of graphical user interface is saved. If you open the application again, device components are on the same place as before closing.

Device inclusion/exclusion

For inclusion of the device to the network select item *Device* from the context menu and then press *Add device*. New window is displayed. In this moment, do the action specified for inclusion of the device - typically press the button on the device. When the device is successfully included, window is automatically closed and new device is added to the panel of all devices (on the right side of main window). It may take a few seconds to initialize newly added device and show its name (if it is known) or its generic class.

If you want exlude the device from network, select item *Device* from the context menu and then press *Remove device*. New window is displayed. In this moment, do the action specified for inclusion of the device - typically press the button on the device. Device is removed from the window and also from the network. Associations are removed automatically.

Controlling devices

You can controll every device separately. To show context menu with possible commands which can be sent to the device, perform right mouse click on the device component in the main window. Afterwards select desired command, which is sent immediately.

If you want to create association between the devices, move device from the panel on the right side (panel of all devices) to the panel on the left side (panel of device group). Devices can be moved simply by operation drag & drop (like icons on your computer). Associations are created immediately when you place the device in the left panel. If you want to discard the association, simply move the device back to the right panel.

Commands *Basic On* and *Basic Off* can be sent to all devices in group by pressing yellow or gray lightbulb icon in the right-upper corner of the panel.

Parameters settings

By pressing icon with *i* on the device component in main window you can open window for configuring parameters of the device. List of parameters is loaded automatically in case of Steinel devices. In case of other devices, you have to specifiy number of parameter, datatype and value manually. By pressing *Get* button you can load current parameter value from the device. In the header of window are also basic information about the device, its Node ID, Generic Class and Specific Class.

Obsah CD

Obsah přiloženého CD je následující:

- soubor `LauterbachM_SoftwareSpravu_JP_2017.pdf` – elektronická verze práce,
- adresář `projekt` – kompletní projekt z Microsoft Visual Studio 2015 obsahující veškeré zdrojové soubory a zároveň také diagram tříd ve formátu PDF (soubor *Class_Diagram.pdf)*,
- adresář `aplikace` – spustitelná aplikace.