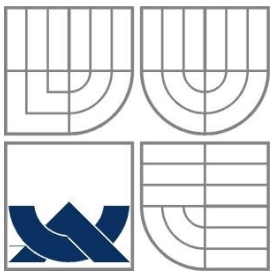
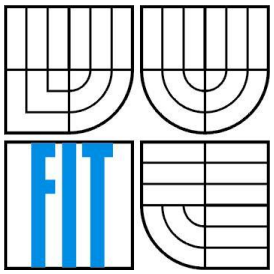


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORTMATION SYSTEMS

KOMUNIKACE V ZIGBEE SÍTÍCH

COMMUNICATION IN ZIGBEE NETWORKS

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ MAJER

VEDOUCÍ PRÁCE
SUPERVISOR

Mgr. ROMAN TRCHALÍK

BRNO 2010

Zadání diplomové práce/9359/2009/xmajer10

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2009/2010

Zadání diplomové práce

Řešitel: **Majer Tomáš, Bc.**
Obor: Počítačové systémy a sítě
Téma: **Komunikace v ZigBee sítích**
Communication in ZigBee Networks
Kategorie: Počítačové sítě

Pokyny:

1. Seznámení se s technologií ZigBe, a popište principy komunikace a adresování v této síti.
2. Navrhněte koordinátor sítě ZigBee tak, aby dokázal příjmout a přeposlat libovolnou zprávu ze/do sítě ZigBee přijatou přes sériový port.
3. Proveďte implementaci navrženého systému a otestujte na realném zařízení.
4. Zhodnoďte dosažené výsledky a uveďte příklady využití toho systému.

Literatura:

- Ilyas, M., Mahgoub, I.: Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems, CRC Press LLC, Boca Raton, FL, USA, 2004.
- National Institute of Standards and Technology, NIST IEEE 1451, [online] < http://www.motion.aptd.nist.gov>
- ZigBee Alliance: ZigBee Specification v 1.1. ZigBee Alliance Board of Directors, 2006. [online] < http://www.zigbee.org>
- Pužmanová, R.: Routing and Switching, Time of convergence, ISBN: 9780201398618
- Furht, B., Ilyas, M.: Wireless Internet Handbook, ISBN:9780849315022

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1-3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Trchalík Roman, Mgr.**, UIFS FIT VUT
Datum zadání: 21. září 2009
Datum odevzdání: 26. května 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

V dnešní době se stále více setkáváme se senzory a senzorovými sítěmi, které jsou uplatňovány převážně v průmyslových oblastech. V této diplomové práci se zaměřuji na senzorovou síť ZigBee a její využití. Je zde podrobně představena vnitřní struktura komunikačních rámců na aplikační vrstvě APS, které slouží pro ovládání senzorů ve standardu ZigBee. Hlavním záměrem diplomové práce je návrh a implementace aplikační brány mezi Internetem a senzorovou sítí ZigBee. Představuji zde možné řešení komunikačního protokolu přes síť Internet a jeho zpracování koordinátorem pro senzorovou síť tak, aby umožňoval vzdálenou správu senzorové sítě. Diplomová práce je napsána tak, aby mohla být využita jako návrh řešení pro aplikaci do praxe. Výsledky řešení implementace jsou znázorněny na modelových situacích a následně uvádím možná vylepšení určitých částí implementace.

Abstract

Now days, we are more often meet sensors and sensor networks, which are used mainly in industrial fields. In this master thesis my target is sensor network Zigbee and usage of it. Internal structure of communication frames of APS application layer is presented here in details as well, which is used for control of sensors inside of sensor network. Main purpose of this master thesis is to design and implement application gateway between Internet and sensor network ZigBee. I present possible solution of communication protocol for transport over the Internet and processing of it by ZigBee coordinator. Thesis is written in style suitable for practical solution and results of solution are presented on model situations, which include discussion about possible improvements.

Klíčová slova

Senzor, senzorová síť ZigBee, 802.15.4, aplikační brána, komunikační protokol, ZigBee koordinátor, struktura rámců APS.

Keywords

Sensor, sensor network ZigBee, 802.15.4, application gateway, communication protocol, ZigBee coordinator, structure of APS frames.

Citace

Majer Tomáš: Komunikace v ZigBee sítích, diplomová práce, Brno, FIT VUT v Brně, 2010

Komunikace v ZigBee sítích

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Mgr. Romana Trchalíka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Majer
20.5.2010

Poděkování

Chtěl bych poděkovat vedoucímu mé diplomové práce Mgr. Romanu Trchalíkovi, který mi poskytl odbornou pomoc při jejím vypracování. Také bych mu chtěl poděkovat za četné konzultace, které mě vždy směrovaly tím správným směrem.

Obsah

Obsah 1
1 Úvod 3
2 Senzorová síť a IEEE specifikace 4
2.1 Senzorové sítě 4
2.1.1 Komponenty senzorové sítě 4
2.1.2 Použití senzorových sítí 4
2.1.3 Architektura senzorových sítí 5
2.1.4 Vlastnosti senzorových sítí 6
2.2 Standard IEEE 802.15.4 7
2.2.1 Souhrnný popis IEEE 802.15.4 8
2.2.2 Fyzická vrstva IEEE 802.15.4 10
2.2.3 Podvrstva MAC IEEE 802.15.4 10
3 Technologie ZigBee 12
3.1 Přehled základních vlastností 12
3.2 ZigBee struktura vrstev 13
3.2.1 Struktura vrstev pro ZigBee 13
3.2.2 Popis jednotlivých částí vrstev modelu pro normu ZigBee2006 13
3.3 Všeobecný APDU formát rámce (APDU Frame Format) 14
3.3.1 Formát APS rámce (APS FF – APS Frame Format) 14
3.3.2 Formát jednotlivých typů APS rámců 17
3.4 ZigBee Cluster Library 19
3.4.1 Knihovna ZigBee Cluster (ZCL-ZigBee Cluster Library) 19
3.4.2 Formát ZCL rámce (ZCL Frame Format) – hlavička 20
3.4.3 Datová část ZCL rámce (Frame Payload) 21
4 Návrh aplikační brány 23
4.1 Komunikační protokol a komunikace 23
4.1.1 Komunikační protokol pro přenos dat přes Internet 23
4.1.2 Popis struktury komunikace 28
4.2 Návrh implementace aplikační brány 29
4.2.1 Návrh implementace koordinátoru 30
4.2.2 Návrh implementace serveru 31
5 Implementace aplikační brány 33
5.1 Úvodní přehled implementace 33
5.1.1 Přehled a představení implementačních prostředí 33

5.1.2 Klientská aplikace 34
5.2 Implementace ZigBee koordinátoru 35
5.2.1 Hardware pro ZigBee koordinátor 35
5.2.2 ZigBee knihovny – ZigBee Stack 37
5.2.3 Implementace zpracování dat 37
5.3 Implementace serveru 42
5.3.1 Stavební kameny implementace serveru 42
5.3.2 Popis běhu serveru 57
5.4 Modelové situace 60
5.4.1 Modelová situace 1 61
5.4.2 Modelová situace 2 62
5.4.3 Modelová situace 3 64
5.4.4 Zátěžový test aplikační brány 66
6 Závěr 68

1 Úvod

Předkládaná diplomová práce se věnuje především problematice komunikace v senzorových sítích ZigBee. Záměrem je navrhnout a implementovat aplikační bránu pro komunikaci mezi Internetem a senzorovou sítí ZigBee. Aplikační brána umožňuje ovládání senzorů pomocí vzdálené aplikace, jež může být spuštěna na počítači kdekoliv na světě s přístupem k Internetu. Komunikace přes bránu je obousměrná, abychom byli schopni nejen ovládat specifické senzory, ale i přijímat zprávy ze senzorové sítě. Návrh aplikační brány obsahuje i návrh komunikační protokolu pro přenos příkazů přes síť Internet. Diplomová práce může sloužit jako praktický návod pro využití senzorové sítě v praxi. Dává čtenáři přehled všeho potřebného tak, aby mohl pochopit aspekty, které jsou nutné při přizpůsobování implementace programu pro senzorovou síť ZigBee, tak aby vyhovovala specifickým požadavkům čtenáře na senzorovou síť.

Druhá kapitola obsahuje všeobecné vlastnosti senzorových sítí, kde se především zaměřuji na topologie senzorových sítí a přehled komunikace v nich. Je zde uveden souhrnný popis standardu IEEE 802.15.4, který definuje nejnižší vrstvy (fyzickou a linkovou) komunikačního modelu.

Třetí kapitola nabízí celkový přehled technologie ZigBee s podrobným rozborem komunikačních rámců vyšších vrstev senzorové sítě ZigBee, které slouží pro ovládání či příjem zpráv od senzorů. Tyto vrstvy jsou Application Support Layer a ZigBee Cluster Library.

Čtvrtá kapitola se zabývá návrhem komunikace přes aplikační bránu a návrhem komunikačního protokolu pro komunikaci přes Internet. Dále je zde detailní návrh aplikační brány jako celku. Aplikační brána se skládá ze dvou částí (serverová část a ZigBee koordinátor), jejichž návrhy jsou v kapitole představeny.

Pátá kapitola se věnuje implementaci jednotlivých částí aplikační brány. Jsou zde představeny stavební kameny implementace, implementace koordinátoru senzorové sítě ZigBee, serveru pro komunikaci s Internetem, podrobný popis ovládání serverové části a možnosti jejího nastavení. Funkčnost a testování implementace aplikační brány je znázorněna na modelových situacích a zátěžovém testu. V této kapitole budou diskutovány i problémy, které nastaly při implementaci.

Poslední kapitolou je závěr, kde hodnotím výsledky návrhu a implementace aplikační brány pro komunikaci se senzorovou sítí ZigBee a také vzniklé problémy. Jsou zde diskutovány úpravy pro aplikační bránu, které by vedly ke zlepšení zpracování příkazů pro senzorovou síť.

2 Senzorová síť a IEEE specifikace

2.1 Senzorové sítě

Senzorová síť (Sensor Network-SN) je infrastruktura složená z měřících, výpočetních a komunikačních elementů, které dávají administrátorovi schopnost dokumentovat, sledovat a reagovat na události a jevy ve specifickém prostředí. Administrátor je typicky civilní, vládní, komerční nebo průmyslová entita. Prostředí může být fyzický svět, biologický systém nebo systém v informačních technologiích [12].

Systémy využívající SN jsou brány jako důležitá sledovací technologie, která bude v budoucnu velmi využívána v mnoha odvětvích a to nejen průmyslových.

2.1.1 Komponenty senzorové sítě

V SN jsou 4 základní komponenty:

- *Skupina distribuovaných nebo lokálních senzorů*
- *Propojovací síť* - obvykle bezdrátově založená komunikace
- *Centrální bod pro shromažďování informací* (koordinátor)
- *Množina výpočetních zdrojů* (nacházejících se na koordinátoru) - zajišťuje korelaci dat, zjišťování tendence událostí, získávání stavu sítě a získávání znalostí z dat.

Z kontextu je vidět, že měřící a výpočetní uzly jsou brány jako část SN, dokonce některé výpočty mohou být prováděny samotnou sítí. Z důvodu získávání, přenášení a shromažďování potencionálně velkého množství datových informací v SN hrají kvalitní algoritmy pro řízení dat velmi důležitou roli.

Výpočetní a komunikační infrastruktura dané SN je velmi často specifická pro prostředí, ve kterém je umístěna. Neméně důležitou vlastností pro SN je způsob napájení jak koordinátoru tak jednotlivých senzorů.

2.1.2 Použití senzorových sítí

Tradičně jsou SN využívány v průmyslových odvětvích, ale čím dál tím více si nacházejí uplatnění i v odvětvích, kde bychom to nečekali.

Zde uvádím krátký přehled aplikačních oblastí využití systémů založených na SN:

- Komerční sféra
    - Měření škodlivých látek uvnitř průmyslových budov
    - Sledování automobilů a jejich rozpoznávání
    - Kontrola dopravních situací

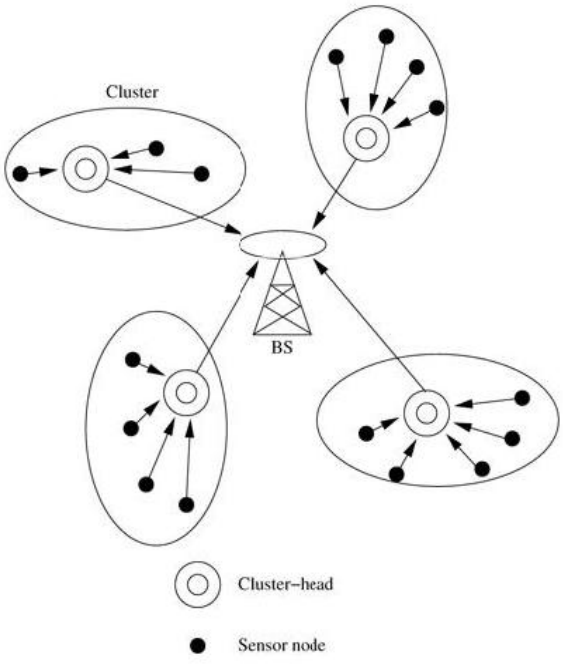
- Armáda
    - Monitorování nepřátelských akcí či jednotek
    - Zaměřování cílů
- Životní prostředí
    - Detekce rizika nebo vzniku ohně v lesích
    - Detekce záplav
- Domácnosti
    - Monitorování majetku proti vniknutí
    - Monitorování vzniku ohně a jeho příznaků (teplota, kouř)

2.1.3 Architektura senzorových sítí

Architekturu senzorových sítí lze rozdělit na seskupenou a vrstvenou [20]. Některé obrázky jsou s popisky v anglickém jazyce, neboť jsou převzaty z literatury.

2.1.3.1 Seskupená architektura

Jednotlivé uzly (sensor node) sítě jsou organizovány do skupin (clusters) a každá skupina má svůj řídící uzel. Koncové uzly komunikují s řídícím uzlem skupiny (cluster-head), který shromažďuje data a provádí výpočty. Výsledky těchto operací jsou zasílány na základní stanici (koordinátor), která je hlavním uzlem senzorové sítě. Základní stanice je komunikačním bodem s okolním světem mimo senzorovou síť a může mít mnoho dalších funkcí, které jsou ovšem specifické u každé základní stanice. Jak vypadá typická vrstvená architektura SN je znázorněno na obrázku 1 (obrázek byl převzat z literatury [20]).

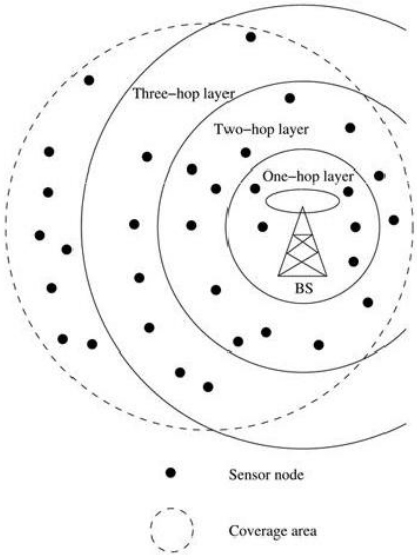


Obrázek 1 Seskupená architektura

2.1.3.2 Vrstvená architektura

Zde jsou uzly (sensor node) rozprostřeny okolo základní stanice do vrstev podle vzdálenosti od základní stanice. Ta díky tomu, že má často plnohodnotný zdroj napájení (naproti tomu uzly jsou ve většině případů napájeny z baterie) může komunikovat se všemi uzly přímo (má možnost generovat signál s větším dosahem). Ovšem uzel může komunikovat se základní stanicí pouze pokud je přímo u stanice, jinak uzel komunikuje přes uzly ze sousední vrstvy. Stejně jako u seskupené architektury je základní stanice výstupním komunikačním bodem SN s okolním světem.

Vrstvená architektura má tedy méně energeticky náročnou komunikaci uvnitř sítě a také je levnější na pořízení, neboť nepotřebuje řídící body jako seskupená architektura. Nevýhodou může být větší zatížení uzlů při přesměrování komunikace přes jednotlivé vrstvy. Další nevýhoda je horší škálovatelnost topologie sítě, kdy u seskupené architektury můžeme mít složitější topologii a snadnější práci při přidávání dalších skupin senzorů. Proto výběr architektury SN by měl být promyšlen i s ohledem na budoucí růst počtu senzorů. Na obrázku 2 je znázorněna vrstvená architektura (obrázek byl převzat z literatury [20]).



Obrázek 2 Vrstvená architektura

2.1.4 Vlastnosti senzorových sítí

Pro návrh senzorové sítě je velmi důležité znát její vlastnosti, které jsou uváděny v literatuře [9].

- *Topologie senzorové sítě* – popis rozmístění uzlů a jejich vzájemné propojení. Nejčastěji to bývá hvězdicová nebo stromová topologie či jejich kombinace.
- *Propojenost sítě* – definuje, které uzly sítě jsou v kontaktu po celou dobu komunikace a jestli je komunikace přerušovaná.
- *Velikost sítě* – nezabývá se plochou, kterou senzory pokrývají, ale počtem navzájem propojených zařízení uvnitř SN.
- *Pokrytí sítě* – určuje hustotu rozmístění senzorů v rámci jedné oblasti.

- *Pohyblivost senzorů* – říká, zda uzly mají schopnost se pohybovat z místa na místo nebo jsou připojeny k pohybujícím se částem objektů.
- *Způsob komunikace* – komunikace nemusí být výhradně rádiová, ale může být i světelná či zvuková.

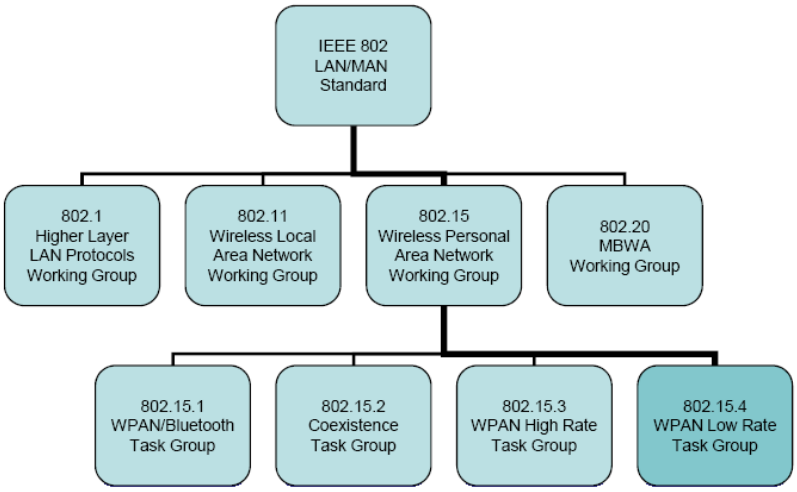
V některých vlastnostech jsou velmi podobné klasickým bezdrátovým sítím. Pro přehled uvádím v čem jsou SN odlišné od klasických bezdrátových sítí.

- Zdrojem napájení jsou v mnoha případech baterie => nižší příkon
- Mnohem větší počet komunikačních uzlů
- Nižší rychlost přenosu dat a rozdílné velikosti přenášených rámců
- Síť je zaměřena především na sběr a vyhodnocování dat
- Autentizace do sítě je převážně automatizovaná.

2.2 Standard IEEE 802.15.4

IEEE 802.15.4 protokol specifikuje fyzickou vrstvu a podvrstvu řízení přístupu na médium (Medium Access Control - MAC) pro nízko-rychlostní bezdrátové soukromé sítě (Low-Rate Wireless Private Area Networks – LR-WPAN). Standard se zaměřuje na sítě s nižší rychlostí transportu dat a s nižšími energetickými nároky, což typicky vyhovuje požadavkům senzorových sítí [21].

IEEE 802.15.4 protokol je převážně spojován se ZigBee protokolem. ZigBee aliance spolupracuje s IEEE na způsobu specifikace plnohodnotného protokolu pro bezdrátovou komunikaci s nízkými náklady, nízkou spotřebou a nízkou přenosovou rychlostí pro celosvětové použití. Právě ZigBee specifikace (vydaná v roce 2004 a její následník vydaný v roce 2006) definuje vrstvy nad IEEE 802.15.4, kterými jsou síťová vrstva (zahrnující bezpečnostní služby) a aplikační vrstva, jenž bude podrobně probrána v kapitole 3.2. Na obrázku 3 uvedena struktura standardu IEEE 802 a začlenění standardu IEEE 802.15.4 do něj (obrázek byl převzat z literatury [20]).



Obrázek 3 Přehled standardu IEEE 802

2.2.1 Souhrnný popis IEEE 802.15.4

2.2.1.1 Síťová zařízení

Na základě standardu IEEE 802.15.4 LR-WPAN podporuje dva typy zařízení:

a) *Plně funkční zařízení (Full Function Device - FDD)* – Zařízení podporující 3 operační módy, ve kterých slouží jako:
   - *Koordinátor pro osobní síť (Personal Area Network - PAN)* – hlavním řídící prvek pro PAN. Toto zařízení identifikuje vlastní síť, do které mohou být další zařízení přiřazena.
   - *Koordinátor* – poskytuje synchronizaci služeb přes přenášená data. Koordinátor musí být přiřazen do nějaké PAN sítě, ale sám o sobě žádnou netvoří.
   - *Jednoduché zařízení* – ostatní zařízení, která neimplementují předchozí funkce (např. senzor).

b) *Zařízení s omezenou funkčností (Reduced Function Device-RFD)* – zařízení pracující s minimální implementací IEEE 802.15.4 protokolu. RFD je zaměřeno na velmi jednoduché aplikace, kterými jsou přepínač světel nebo pasivní infračervený senzor. Není z něj potřeba zasílat velké množství dat a musí být připojeno právě jen k jednomu FDD.

LR-WPAN síť musí obsahovat minimálně jedno zařízení FDD, které implementuje funkci PAN koordinátoru, aby mohla poskytovat synchronizaci síťových služeb a řízení potenciálních FDD nebo RFD zařízení uvnitř senzorové sítě.

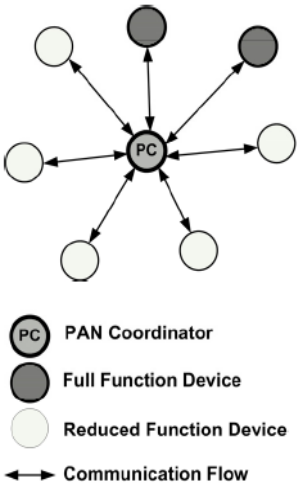
2.2.1.2 Topologie sítě

Standard IEEE 802.15.4 definuje dvě základní topologie a to jsou *hvězda* a *peer-to-peer*. Často se ještě objevuje další topologie *cluster-tree*, která je ovšem brána jako speciální případ *peer-to-peer* topologie.

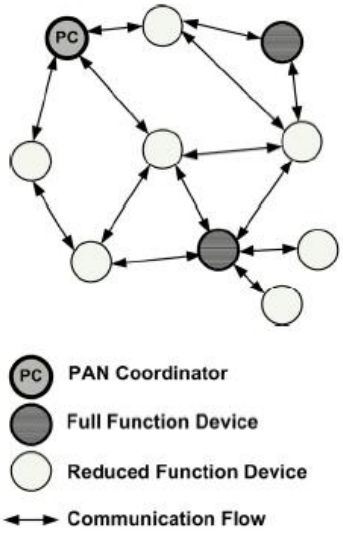
a) *Topologie hvězdy (Star topology)* – tato topologie obsahuje jeden PAN koordinátor, přes který probíhá veškerá komunikace mezi zařízeními, z čehož plyne, že tento PAN koordinátor by měl být napájen stálým napájením, zatímco ostatní prvky senzorové sítě mohou mít napájení z baterie. Ve standardu je doporučeno využívat topologii hvězdy jen pro domácí využití či pro ovládání periferií k počítači.

b) *Topologie Peer-to-peer* – komunikace je zde decentralizována (samozřejmě hlavním prvkem je opět PAN koordinátor). Každé zařízení může přímo komunikovat s jakýmkoliv jiným zařízením uvnitř senzorové sítě. Tato topologie umožňuje rozsáhlejší použití v praxi a je shodná s vrstvenou architekturou (popsána v kapitole 2.1.3.2).

Na obrázku 4 je zobrazena topologie hvězdy (obrázek byl převzat z literatury [4]). Grafické znázornění topologie je na obrázku 5 (obrázek byl převzat z literatury [4]).

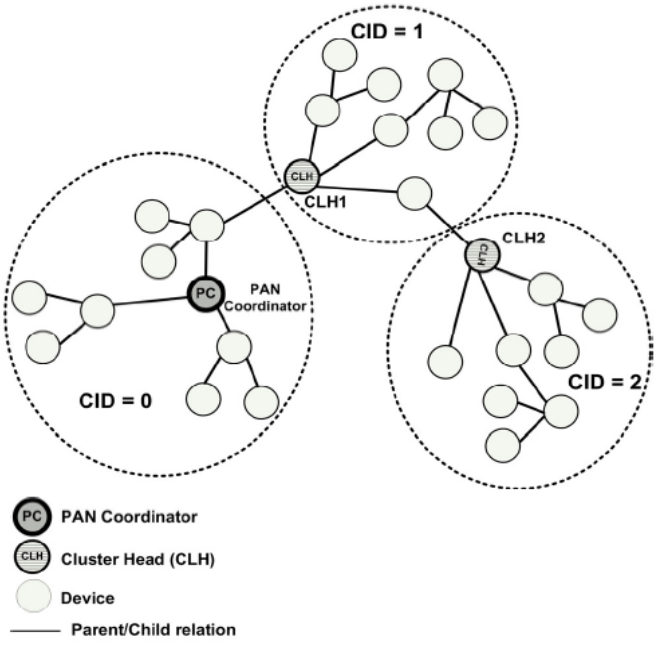


Obrázek 4 Topologie hvězdy



Obrázek 5 Topologie peer-to-peer

c) *Topologie cluster-tree* – speciální případ peer-to-peer topologie, ve které jsou převážně obsaženy FDD zařízení. Má stejnou strukturu jako seskupená architektura (kapitola 2.1.3.1), kde funkci základní báze plní PAN koordinátor. Funkci řídících center plní koordinátory a jako uzly jsou RFD zařízení. Topologie se používá pro velmi rozsáhlé senzorové sítě. Na obrázku 6 můžeme vidět tuto topologii (obrázek byl převzat z literatury [4]).



Obrázek 6 Topologie cluster-tree

2.2.2 Fyzická vrstva IEEE 802.15.4

Fyzická vrstva je zodpovědná za přenos a příjem dat, kdy používá specifický kanál s odpovídající modulací a šířící technikou rozprostření signálu. Tabulka 1 přehledně uvádí frekvenční pásma, parametry rozprostření signálu a rychlosti pro přenos dat.

Některé položky tabulku nechávám v anglickém jazyce, aby nedošlo k jejich dezinterpretaci.

Frekvence [MHz]	Parametry rozprostření		Datové parametry		
	Chip rate [kchip/s]	Modulace	Bit rate [kbps]	Symbol rate [ksymbol/s]	Znaky (Symbols)
868	300	BPSK	20	20	binární
915	600	BPSK	40	40	binární
2400	2000	O-QPSK	250	250	hexadecimální

Tabulka 1 Přehledová tabulka pro parametry fyzické vrstvy standardu IEEE 802.15.4

Fyzická vrstva standardu IEEE 802.15.4 se stará o následují úlohy:

- Aktivace a deaktivace rádio vysílače/přijímače
- Detekce přenášené energie v signálu
- Indikace kvality na přenosovém médiu
- Kontrola přístupu na médium
- Výběr frekvenčního kanálu

2.2.3 Podvrstva MAC IEEE 802.15.4

MAC podvrstva tvoří rozhraní mezi fyzickou vrstvou a vyššími vrstvami LR-WPAN. Podvrstva MAC protokolu IEEE 802.15.4 používá mnoho prostředků protokolu stejně jako IEEE 802.11. Těmito prostředky jsou například přístupový protokol CSMA/CA (Carrier Sense Multiple Access/Contention Avoidance) a podpora nespojovaných nebo spojovaných period. MAC je ovšem více zaměřená na požadavky LR-WPAN, jako je například vyloučení používání RTS/CTS mechanismu k redukci kolizí.

MAC protokol podporuje dva operační módy, které budou představeny v následujících dvou podkapitolách.

2.2.3.1 Beacon-enabled mód

Signály (beacons) jsou periodicky generovány koordinátorem k synchronizaci připojených zařízení a k identifikaci PAN. Signalizační rámec je (první) částí super-rámce, který dále obsahuje všechny datové rámce, které se zasílají mezi jednotlivými uzly a PAN koordinátorem.

Pokud si koordinátor vybere využít tento mód, pak musí využívat strukturu super-rámce k řízení komunikace mezi zařízeními. Formát super-rámce je definován PAN koordinátorem a přenášen k ostatním zařízením mezi jednotlivými signalizačními rámci, které jsou periodicky

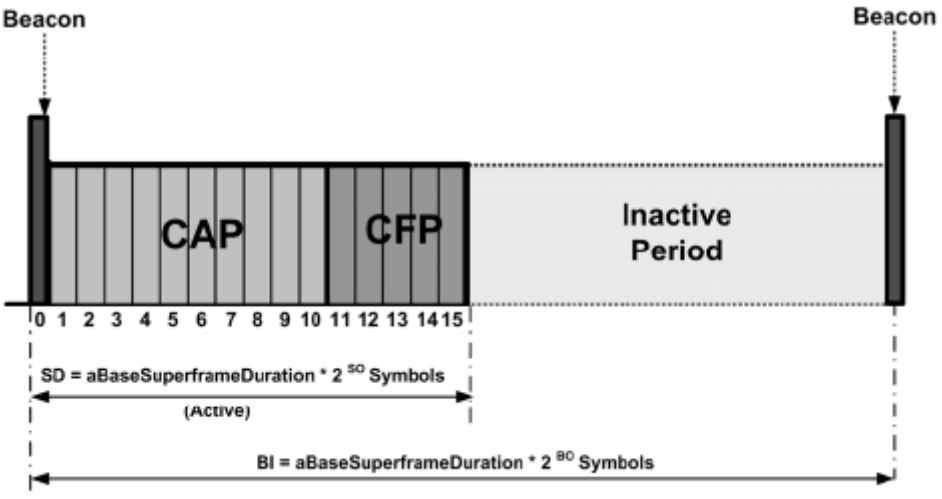
vysílány PAN koordinátorem. Super-rámec je rozdělen do 16-ti stejně velkých slotů a za ním následuje předem definovaný časový úsek tzv. neaktivity. Pokud zařízení chce vysílat musí vše stihnout před koncem super-rámce.

Signalizační interval, jenž je ohraničen dvěma za sebou jdoucími signalizačními rámci, obsahuje dvě části Contention-Access Period (CAP), kde mohou zařízení pomocí CSMA soutěžit o volné rámce, a Contetntion-Free Period (CFP). Pro znázornění obecné struktury super rámce uvádím obrázek 7 (obrázek byl převzat z literatury [4]).

Obrázek 7 Obecná struktura super-rámce

2.2.3.2 Non Beacon-enabled mód

V tomto módu mohou zařízení posílat svá data použitím unslotted CSMA/CA. Super rámec se zde nepoužívá. Všechny zasílané zprávy musí vyhovovat tomuto mechanizmu kromě potvrzovacích rámců, které se používají pro potvrzování požadavků. Na obrázku 8 je zobrazena struktura vysílání pro Non Beacon-enabled mód (obrázek byl převzat z literatury [4]).



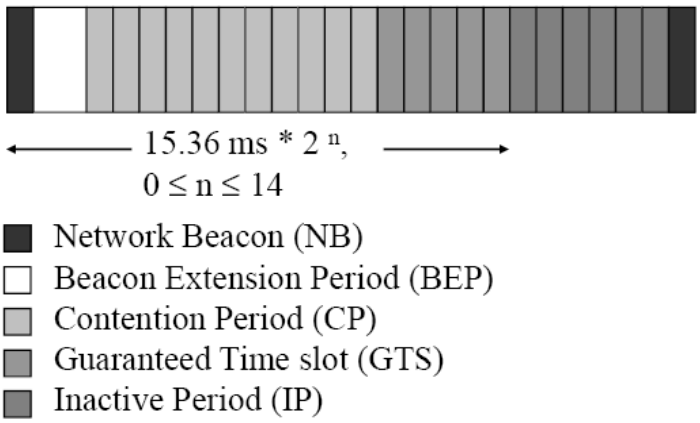
Obrázek 8 Struktura vysílání

3 Technologie ZigBee

3.1 Přehled základních vlastností

ZigBee je bezdrátová komunikační technologie s následujícími vlastnostmi [4]:

- Vystavěna na standardu IEEE 802.15.4
- Určena pro sítě s velmi malým provozem a je optimalizována pro aplikace s komunikační zátěží pod 0,1%.
- Šíření signálu je založeno na technice přímého rozprostření světla DSSS (Direct Sequence Spread Spectrum) s využitím modulací BPSK a QPSK.
- Topologie a typy zařízení jsou přebrány ze standardu IEEE 802.15.4.
- Při posílání zpráv se využívá konstrukce zvaná super-rámec, která má za následek velmi dlouhé mezery mezi vysíláním a tím dochází k velmi dobré výdrži baterií. Vnitřní struktura super-rámce je na obrázku 9 (obrázek byl převzat z literatury [4]).



Obrázek 9 Vnitřní struktura super-rámce

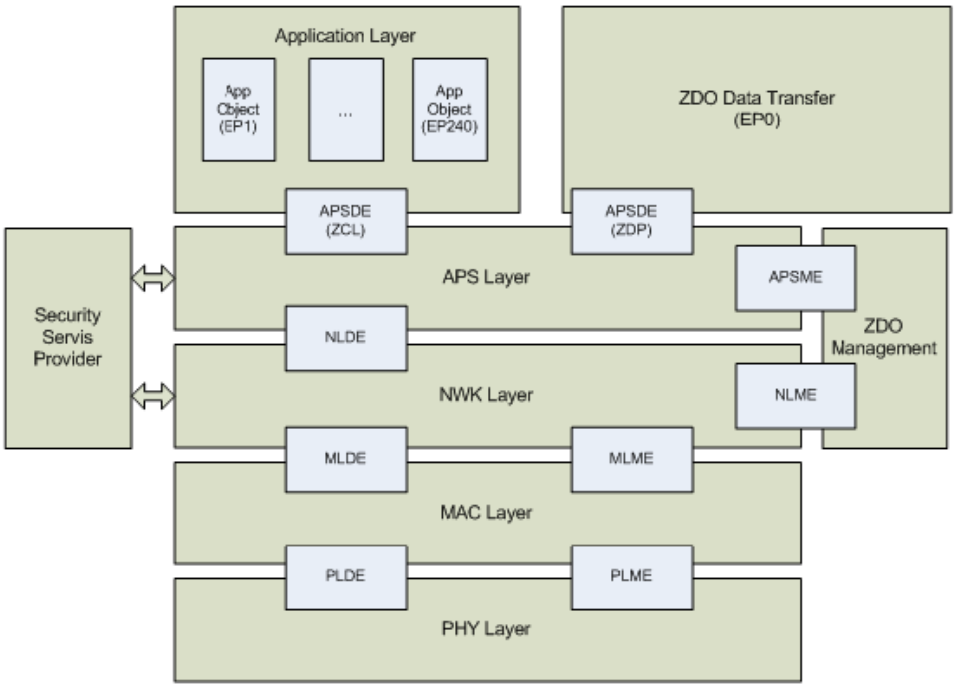
Popis jednotlivých částí super-rámce:

- *Network Beacon (NB)* – slouží k časové synchronizaci mezi jednotlivými uzly
- *Beacon Extension Period (BEP)* – nečinná doba před soupeřením o garantované časové sloty
- *Conection Period (CP)* – soupeření o garantované časové sloty
- *Guaranteed Time Slots (GTS)* – garantované časové sloty slouží k přenosu dat mezi uzly
- *Inactive Period (IP)* – doba neaktivity komunikace uzlů (max. 4 minuty)

3.2 ZigBee struktura vrstev

3.2.1 Struktura vrstev pro ZigBee

Na obrázku 10 je znázorněna struktura vrstev pro ZigBee standard z roku 2006 (obrázek byl převzat z literatury [21]).



Obrázek 10 ZigBee2006 struktura síťových vrstev

3.2.2 Popis jednotlivých částí vrstev modelu pro normu ZigBee2006

Fyzická vrstva (PHY – Physical Layer)

Nejnižší vrstva modelu, která vykonává odpovídající funkci kontrolovanou MAC vrstvou.

Vrstva pro kontrolu pro přístup k přenosovému médiu (MAC – Media Access Control Layer)

Jejím úkolem je kontrola zařízení. Nachází se mezi vrstvami MAC a NWK. Vrstva MAC vyhovuje normě IEEE 802.15.4-2003. Vrstva MAC obsahuje tyto komunikační entity:

- ***MAC Datová entita*** *(MLDE – MAC Data Entity)* – primitiva pro přenos MAC dat.
- ***MAC Řídící entita*** *(MLME – MAC Management Entity)* – primitiva pro řízení MAC dat

Síťová vrstva (NWK – Network Layer)

Vrstva obsahující funkce pro řízení ZigBee sítí. Vrstva NWK se nachází mezi vrstvami MAC a APS. Vykonává datový přenos (utváření, spojování a řízení) a směrovací funkce. Vrstva NWK obsahuje tyto komunikační entity:

- ***NWK Datová entita*** *(NLDE – NWK Data Entity)* – primitiva pro přenos NWK dat.
- ***NWK Řídící entita*** *(NLME – NWK Management Entity)* – primitiva pro řízení NWK dat.
- ***NWK informační báze*** *(NIB – Information Base)* – definuje parametry používané v NWK.

Vrstva aplikační podpory (APS – Application Support Layer)

Obsahuje funkce pro podporu aplikační vrstvy a ZigBee Device Object (ZDO). APS je mezi vrstvami aplikační a ZDO a vrstvou NWK. APS má na starost datový přenos, „binding" a řízení skupin.Vrstva APS obsahuje tyto komunikační entity:

- ***APS Datová entita*** *(APSDE – APS Data Entity)* – primitiva pro přenos APS dat.
- ***APS Řídící entita*** *(APSME – Management Entity)* – primitiva pro řízení APS.
- ***APS informační báze*** *(IB – APS Information Base)* – definuje parametry používané v APS.

ZigBee objekt zařízení (ZDO – ZigBee Device Object)

ZDO je část APS zodpovědná za určení funkce zařízení uvnitř sítě (ZigBee koordinátor nebo koncová zařízení), jeho inicializace a odpovídání pro „binding" a objevování požadavků. Také je zodpovědná za zřízení bezpečného spojení mezi dvěmi zařízeními v sítí. Všechna ZigBee zařízení by měla mít ZDO a endpoint nastaveny na 0.

Knihovna ZigBee Cluster (ZCL – ZigBee Cluster Library)

Knihovna definuje ZigBee standard pro strukturu rámců a clusterů. Podrobnější popis knihovny ZigBee Cluster je v kapitole 3.4.1.

3.3 Všeobecný APDU formát rámce (APDU Frame Format)

3.3.1 Formát APS rámce (APS FF – APS Frame Format)

Pro seznámení s problematikou formátu APS rámce uvádím v této kapitole přehled specifikace ZigBee z roku 2006 [21]. APS FF je složen z APS hlavičky a APS datové části. Jednotlivá políčka APS hlavičky mají pevné pořadí, avšak adresovací políčka nemusí být obsažena ve všech rámcích.

Tabulky s popisky v anglickém jazyce jsou převzaty z literatury [21] a nejsou překládány, neboť při překladu do češtiny by mohlo dojít k dezinterpretaci pojmů.

V tabulce 2 je uvedeno, jak vypadá APS rámec.

Octets:1	0/1	0/2	0/2	0/2	0/1	1	Variable
Frame control	Destination port	Group address	Cluster identifier	Profile identifier	Source endpoint	APS Counter	Frame Payload
	Addressing fields						
APS Header							APS Payload

Tabulka 2 APS formát rámce

3.3.1.1 Kontrolní pole rámce (FC - Frame Control Field)

FC má délku 8 bitů a obsahuje informace definující typ rámce, adresovací políčka a další kontrolní flagy. V tabulce 3 je ilustrován vzhled FC.

Bits: 0-1	2-3	4	5	6	7
Frame type	Delivery mode	Indirect address mode	Security	Ack. Request	Reserved

Tabulka 3 Kontrolní část rámce

Typ rámce (FC-FT - Frame Type Sub-Field)

FC-FT se skládá ze 2 bitů, jejichž hodnoty udávají typ rámce. V tabulce 4 je přehled typů rámců.

Frame Type Value b_1 b_0	Frame type name
00	Data
01	Command
10	Acknowledgement
11	Reserved

Tabulka 4 Typy rámců

Doručovací mód (FC-DM - Delivery Mode Sub-Field)

FC-DM se skládá ze 2 bitů. V tabulce 5 je přehled typů FC-DM.

Delivery mode Value b_1 b_0	Delivery mode name
00	Normal unicast delivery
01	Indirect addressing
10	Broadcast
11	Group addressing

Tabulka 5 Přehled typů doručení

Mód nepřímé adresy (FC-IAM - Indirect Address Mode Sub-Field)

FC-IAM má délku 1 bit a specifikuje, zda SE nebo DE políčka jsou přítomna v rámci APS FF v případě, že je nastaveno nepřímé adresování v FC-DM.

Bezpečnost (FC-S - Security sub-field)

FC-S je řízeno Security Services Provider.

Potvrzovací požadavek (FC-AR - Acknowledgement Request Sub-Field)

FC-AR má délku 1 bit a specifikuje, zda aktuální přenos vyžaduje, aby příjemce zaslal potvrzovací rámec pro stvrzení přenosu.

3.3.1.2 Cílové zařízení (DE - Destination Endpoint Field)

DE má velikost 8 bitů a specifikuje zařízení (endpoint) koncového příjemce rámce.

3.3.1.3 Skupina adres (GA - Group Address Field)

GA má délku 16 bitů. Je přítomen jen v případě, že FC-DM v kontrolním rámci je nastaveno na 0b11. Jestliže APS hlavička rámce obsahuje GA, pak bude rámec doručen všem zařízením, pro které tabulka skupin v zařízení obsahuje asociaci mezi zařízením a skupinou identifikovanou obsahem v GA.

3.3.1.4 Identifikátor Clusteru (CI - Cluster Identifier Field)

CI má délku 16 bitů a specifikuje identifikátor clusteru, který je použit v „binding“ operaci na ZigBee koordinátoru nebo na zařízení identifikovaného SrcAddr v požadavku. Operace „binding“ je zde myšlena jako mapování určité funkce pro jednotlivá zařízení.

3.3.1.5 Identifikátor profilu (PI - Profile Identifier Field)

PI má délku 16 bitů a specifikuje ZigBee identifikátor profilu, pro který je rámec určen, a bude použit během filtrování zpráv na každém zařízení, které rámec obdrží.

3.3.1.6 Zdrojové zařízení (SE - Source Endpoint Field)

SE má velikost 8 bitů a specifikuje zařízení výchozího tvůrce/odesílatele rámce.

3.3.1.7 APS čítač (APS čítač - APS counter)

APS čítač má délku 8 bitů a je použit proti přijetí podvrženého rámce se stejnou hodnotou v poli APS čítače. Po každém přenosu je čítač inkrementován o 1.

3.3.1.8 Datová část rámce (FP – Frame Payload Field)

FP má proměnnou délku a obsahuje informace specifikující jednotlivé typy rámců.

3.3.2 Formát jednotlivých typů APS rámců

Jsou definovány 3 typy rámců – datový rámec, APS příkaz a potvrzovací rámec.

3.3.2.1 Formát datového rámce (DFF - Data Frame Format)

Uspořádání polí v DFF je stejné jako u všeobecného APS rámce, což ilustruje tabulka 6.

<table>
<tr><td>Octets:1</td><td>0/1</td><td>0/2</td><td>2</td><td>2</td><td>1</td><td>1</td><td>Variable</td></tr>
<tr><td rowspan="2">Frame control</td><td>Destination port</td><td>Group address</td><td>Cluster identifier</td><td>Profile identifier</td><td>Source endpoint</td><td rowspan="2">APS Counter</td><td rowspan="2">Frame Payload</td></tr>
<tr><td colspan="5">Addressing fields</td></tr>
<tr><td colspan="7">APS Header</td><td>APS Payload</td></tr>
</table>

Tabulka 6 Struktura datového rámce

Datový rámec – hlavička (DFF-APS hlavička - APS header field)

DFF-APS hlavička pro datové rámce musí obsahovat následující políčka:

- *Frame control* - podrobný popis viz kapitola 3.3.1.1.
- *Cluster identifier* - podrobný popis viz kapitola 3.3.1.4.
- *Profile identifier* - podrobný popis viz kapitola 3.3.1.5.
- *Source endpoint* - podrobný popis viz kapitola 3.3.1.6.
- *APS counter* - podrobný popis viz kapitola 3.3.1.7.

Cílové zařízení (DE - Destination Endpoint field) musí být vloženo do datového rámce v závislosti na hodnotě v FC-DM. V FC-FT musí být uložena hodnota 00, která podle tabulky 3 indikuje typ rámce jako datový.

Datový rámec – tělo (DFF-DP - Data payload field)

Pro odchozí datové rámce musí datová část obsahovat část nebo všechny sekvence oktetů, které jsou vyžadovány vyšší vrstvou datových služeb APS pro přenos.

Pro příchozí datové rámce musí datová část obsahovat sekvenci oktetů, které byly přijaty datovou službu APS a ty se budou uplatňovat na cílových zařízeních nebo budou doručeny vyšším vrstvám v případě, že koordinátor je jedna z cílových položek.

3.3.2.2 Formát příkazového rámce (CFF-APS Command Frame Format)

Uspořádání rámce musí být ve formátu uvedeném v tabulce 7.

<table>
<tr><td>Octets: 1</td><td>0/2</td><td>1</td><td>1</td><td>Variable</td></tr>
<tr><td>Frame control</td><td>Group Address</td><td>APS counter</td><td>APS command identifier</td><td>APS command payload</td></tr>
<tr><td colspan="3">APS leader</td><td colspan="2">APS payload</td></tr>
</table>

Tabulka 7 Struktura APS rámce určeného pro specifikaci příkazu

Příkazový rámec – hlavička (CFF-APS hlavička – Command Frame APS Leader Field)

CFF-APS hlavička musí obsahovat FC a APS čítač. Skupinová adresa musí být obsažena v rámci v případě, že FC-DM je nastaven pro skupinové adresování.

Příkazový rámec – tělo (CFF-APS payload – Command Frame APS Payload Field)

CFF-APS datová část je dále rozdělena na:

- *APS Command Identifier Field* – identifikace použití příkazu.
- *APS Command Payload Field* – obsahuje samotný příkaz.

3.3.2.3 Formát potvrzovacího rámce (AFF – Acknowledgement Frame Format)

Uspořádání rámce AFF musí být ve formátu uvedeném v tabulce 8.

Octets: 1	0/1	2	2	0/1	1
Frame control	Destination endpoint	Cluster identifier	Profile identifier	Source endpoint	APS counter
APS header					

Tabulka 8 Struktura potvrzovacích rámců

AFF hlavička

AFF hlavička musí obsahovat následující:

- *Frame control* – podrobný popis viz kapitola 3.3.1.1.
- *Cluster identifier* - podrobný popis viz kapitola 3.3.1.4.
- *Profile identifier* - podrobný popis viz kapitola 3.3.1.5.
- *APS counter* - podrobný popis viz kapitola 3.3.1.7.

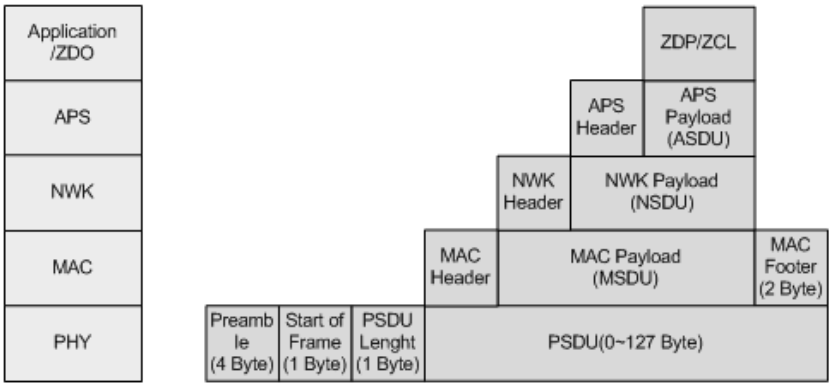
Další vlastnosti AFF hlavičky:

- Jestliže FC-DM indikuje přímé adresování, pak AFF musí obsahovat SE i DE.
- Jestliže FC-DM indikuje nepřímé adresování, pak AFF musí obsahovat SE i DE podle hodnoty FC-IAM.
- APS čítač musí obsahovat stejnou hodnotu jako rámec, který je potvrzován.

3.4 ZigBee Cluster Library

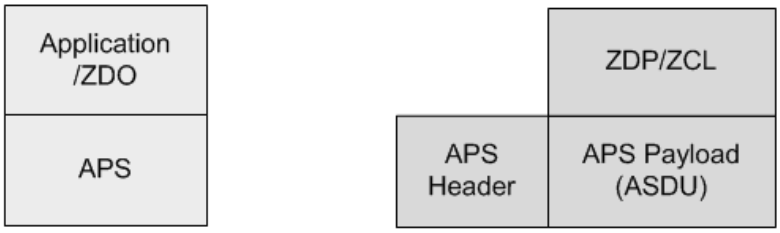
3.4.1 Knihovna ZigBee Cluster (ZCL-ZigBee Cluster Library)

Pro seznámení s problematikou formátu ZCL uvádím v této kapitole přehled podle specifikace ZigBee z roku 2006 [21]. Na obrázku 11 je zobrazen formát rámců pro jednotlivé vrstvy ZCL (obrázek byl převzat z literatury [21]).



Obrázek 11 ZigBee2006 Struktura rámců na jednotlivých vrstvách síťového modelu

ZCL definuje formát APS datové části (ASDU) při přechodu na aplikační vrstvu. Endpoint je nastaven na hodnotu 0, jestliže jsou zasílána a přijímána data typu APS data primitive (APS-DATA.req, APS-DATA.ind). Jsou to data pro ZDP (ZigBee Device Profile) a jsou zpracovávána v ZigBee Device Object (ZDO). Je-li Endpoint nastaven na 1, pak jsou data zpracovávána na aplikační vrstvě. V tomto případě má APS datová část formát stejný jako ZCL, což je znázorněno na obrázku 12 (obrázek převzat z [21]).



Obrázek 12 Formát rámce

3.4.2 Formát ZCL rámce (ZCL Frame Format) – hlavička

Tabulka 9 znázorňuje formát ZCL rámce.

Octets : 1	0/2	1	1	Variable
Frame control	Manufacturer code	Transaction Sequence Number	Command ID	Frame Payload
ZCL Header				ZCL payload

Tabulka 9 Struktura ZCL rámce

3.4.2.1 Kontrolní políčko ZCL (FCF - ZCL Frame Control Field)

Tabulka 10 znázorňuje formát FCF.

Bits : 0-1	2	3	4	5-7
Frame Type	Manufacturer Specific	Direction	Disable Default Response	Reserved

Tabulka 10 Kontrolní část rámce

Nyní uvádím přehled významů jednotlivých políček FCF pro hodnoty, kterých mohou nabývat. Hodnota v políčku, pro kterou je popisován případ, je uveden před tímto popisem.

Typ rámce (FCF-FT - Frame Type)

0 – Příkaz platí přes celý profile. Políčko Command ID má hodnotu vyhovující ZCL standardu.
1 – Příkaz je specifický na cluster. Políčko Command ID má definovanou hodnotu v definovaném clusteru.

Výrobní specifikace (FCF-MS - Manufacturer Specific)

0 – ZCL hlavička obsahuje výrobní kód (Manufacturer Code).
1 – ZCL hlavička neobsahuje výrobní kód (Manufacturer Code).

Směr (FCF-D - Direction)

0 – Rámec je zasílán ze serveru na klienta.
1 – Rámec je zasílán z klienta na server.

Zakázání implicitní odpovědí (FCF-DDS - Disable Default Response)

0 – Příjemce zasílá Response Command.
1 – Příjemce nezasílá Response Command.

3.4.2.2 Políčko výrobního čísla (ZCL Manufacturer Code Field)

Jestliže je použit výrobcem definovaný profile, cluster a atribute, tak by tyto hodnoty měly by být obsaženy v hlavičce ZCL rámce. Pokud chceme přenášet tuto hodnotu musí být v FCF nastavena hodnota FCF-MS na 1.

3.4.2.3 Sekvenční číslo transakce (ZCL Transaction Sequence Number Field)

Obsahuje sekvenční číslo, které je inkrementováno o 1 při novém vygenerování rámce.

3.4.2.4 Identifikátor příkazu (ZCL Command ID Field)

Obsahuje příkaz daného rámce. Pokud je FCF-FT nastaveno na hodnotu 0, pak je v identifikátoru příkazu uložena hodnota definovaná ve standardu ZCL. Je-li naopak nastaveno na 1, pak by měl být použit příkaz definovaný ve všech clusterech.

3.4.3 Datová část ZCL rámce (Frame Payload)

V této části se zaměříme na datovou část ZCL příkazu. Formát datové části se liší na základě identifikátoru příkazu (Command ID). V dalších podkapitolách uvedu přehled formátů datových částí hlavních ZCL příkazů.

3.4.3.1 Přečti atribut (RA – Read Attribute)

Příkaz slouží k získání hodnot atributů vzdáleného zařízení. Tabulka 11 znázorňuje formát datové části při použití příkazu RA.

Octets: variable	2	2	…	2
ZCL Header	Attribute ID 1	Attribute ID 2	…	Attribute ID n

Tabulka 11 Čtení atributů

Nastavení pro ZCL hlavičku (ZCL Header)

Frame Type = 0, Command ID = 0x00

3.4.3.2 Odpověď pro příkaz přečtení atributu (RAR - Read Attribute Response)

Příkaz je použit v rámci odpovědi, jestliže zařízení obdrží příkaz RA. Tabulka 12 znázorňuje formát datové části při použití příkazu RAR.

Octets: variable	variable	variable	…	variable
ZCL Header	Read attribute status record 1	Read attribute status record 2	…	Read attribute status record n

Tabulka 12 Struktura odpovědí na příkaz čtení

Nastavení pro ZCL hlavičku (ZCL Header)

Frame Type = 0, Command ID = 0x01

Formát políčka Status záznamu pro čtení atributu (RASR - Read Attribute Status Record Field)

Tabulka 13 znázorňuje formát políčka RASR při použití příkazu RAR. Políčka Attribute Data Type a Attribute Data pokud Status zobrazí úspěch.

Octets: 2	1	0/1	0/variable
Attribute ID	Status	Attribute Data Type	Attribute Data

Tabulka 13 Formát RASR

3.4.3.3 Zapiš atribut (WA – Write Attribute)

Příkaz se používá pro zapsání atributů na vzdálené zařízení. Tabulka 14 znázorňuje formát datové části při použití příkazu WA.

Octets: variable	variable	variable	…	variable
ZCL Header	Write attribute record 1	Write attribute record 2	…	Write attribute record n

Tabulka 14 Zápis atributů

Nastavení pro ZCL hlavičku (ZCL Header)

Frame Type = 0, Command ID = 0x02

Formát políčka Záznam pro zápis atributu (WAR – Write Attribute Record Field)

Tabulka 15 znázorňuje formát políčka WAR při použití příkazu WA.

Octets: 2	1	variable
Attribute ID	Attribute Data Type	Attribute Data

Tabulka 15 Formát záznamu pro zápis atributu

4 Návrh aplikační brány

Hlavním cílem je návrh aplikační brány pro komunikaci mezi vzdálenou aplikací a senzorovou sítí ZigBee tak, aby bylo možné zasílat příkazy pro senzorovou síť, ale i naopak přijímat hlášení a odpovědi ze senzorové sítě.

Nejprve je představen v kapitole 4.1.1 návrh komunikačního protokolu na aplikační úrovni (dále jen komunikační protokol) pro přenos příkazů přes síť Internet do senzorové sítě ZigBee. Samotný návrh aplikační brány je představen v kapitole 4.2. Návrh aplikační brány se skládá ze dvou částí. První je serverová část aplikační brány (dále jen server), která se stará o příjem a zpracování dat ze sítě Internet. Druhou část představuje ZigBee koordinátor, jenž je řídícím prvkem senzorové sítě ZigBee a má na starost komunikaci s elementy senzorové sítě (senzory).

4.1 Komunikační protokol a komunikace

4.1.1 Komunikační protokol pro přenos dat přes Internet

Komunikační protokol je navržen tak, aby mohl být univerzální pro jakoukoliv strukturu senzorové sítě ZigBee. V komunikačním protokolu jsou zahrnuta klíčová slova určená pro příkazy, tak jak je tomu třeba u Post Office Protocol verze 3 (POP3 [10]). Přehled klíčových slov i s jejich stručným významem v komunikačním protokolu je uveden v tabulce 16.

Příkaz	Význam příkazu
AUTH	Autorizace klienta
COMM	Příkaz pro senzorovou síť
CHCK	Kontrola zda se na serveru nenachází odpověď ze senzorové sítě
CLSE	Odhlášení a uzavření spojení se serverem

Tabulka 16 Stručný přehled komunikačního protokolu

Podrobně jsou příkazy protokolu i odpovědi od serveru rozebrány v následujících podkapitolách. Pokud klient zašle příkaz na server ve špatném tvaru nebo neplatný příkaz, tak je na tuto skutečnost upozorněn odpovědí ze serveru. Zpracování příkazů serverem je case-sensitive (rozlišuje VELKÁ a malá písmena).

Legenda k příkazům:

< parametr > – povinná část příkazu

[parametr] – nepovinná část příkazu

4.1.1.1 Příkaz AUTH

Příkaz AUTH slouží pro autentizaci uživatele, který se chce přihlásit na server, aby mohl komunikovat se senzorovou sítí. Pokud není klient autentizován, tak mu není umožněno zasílat či přijímat příkazy pro senzorovou síť. Server v případě neplatného přihlašovacího jména klienta uzavře

spojení. Příkaz se používá pro zajištění alespoň základní bezpečnosti komunikace. Pokud bychom chtěli zajistit větší bezpečnost, tak bychom ji zajistili buď pomocí hesla, nebo ještě lépe pomocí šifrované komunikace s využitím například Secure Socket Layer (SSL) [13].

Syntaxe příkazu AUTH:

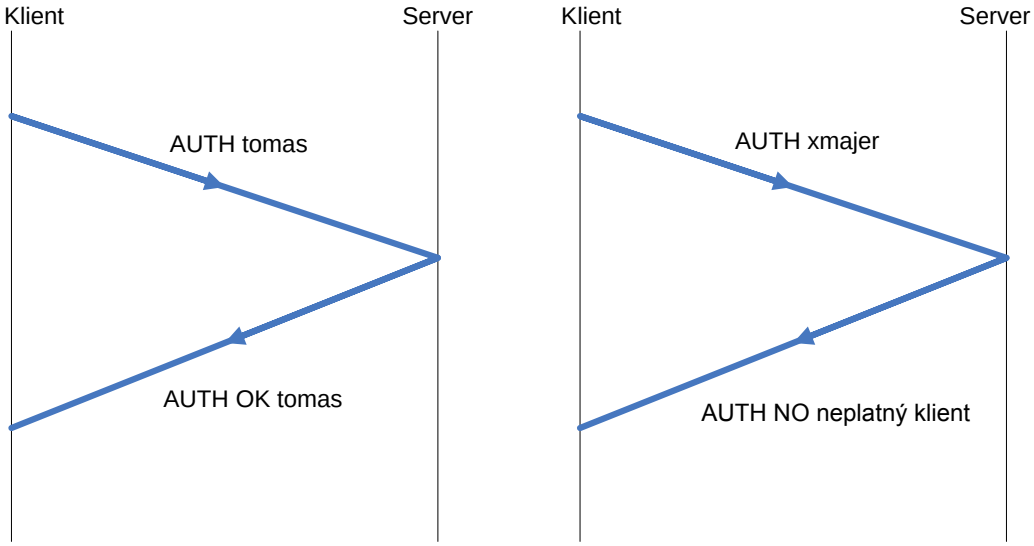
```
AUTH <uživatelské_jméno>
```

Syntaxe odpovědi:

```
AUTH <status> <info_zpráva>
```

Příklad komunikace:

Na obrázku 13 jsou znázorněny příklady komunikace pro příkaz AUTH.



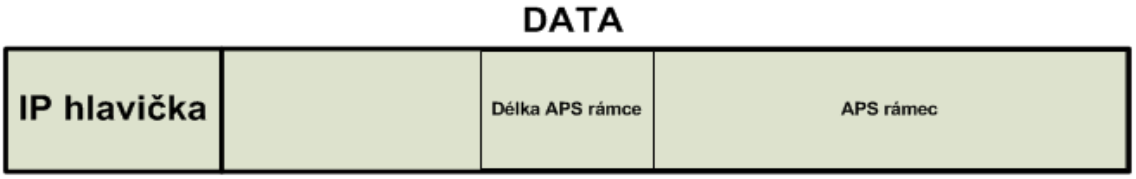
Obrázek 13 Přiklad komunikace pomocí příkazu AUTH

4.1.1.2 Příkaz COMM

Příkaz COMM slouží pro zaslání příkazů pro senzorovou síť. Pokud není klient autentizován pomocí příkazu AUTH, tak příkaz pro senzorovou síť je zahozen a klient je informován o tom, že se má nejprve autentizovat. Jako parametr příkazu COMM je použit vlastní APS rámec, ve kterém jsou uloženy příkazy pro senzorovou síť zpracovávané na koordinátoru. Díky komplexnosti APS rámce, který poskytuje získání statusu sítě nebo daného elementu senzorové sítě a možnost vložení příkazu, příkaz COMM jako parametr obsahuje jen velikost APS rámce a vlastní APS rámec.

Univerzálnost protokolu spočívá v tom, že je možné do parametru příkazu COMM vložit jakýkoliv vnitřní rámec pro vybranou strukturu senzorové sítě ZigBee. V reálném provozu bude APS rámec i jeho délka zasílána v hexadecimální kódu, ovšem pro větší přehlednost budu v diplomové práci uvádět textový řetězec.

Na obrázku 14 je uvedeno jak vypadá vnitřní část příkazu COMM v reprezentaci IP paketu přicházejícího ze sítě Internet do senzorové sítě a platí to i pro opačný směr (příkaz CHCK). Obrázek neobsahuje příkaz COMM z důvodu větší přehlednosti.



Obrázek 14 Vnitřní struktura komunikace přes Internet

Syntaxe příkazu COMM:

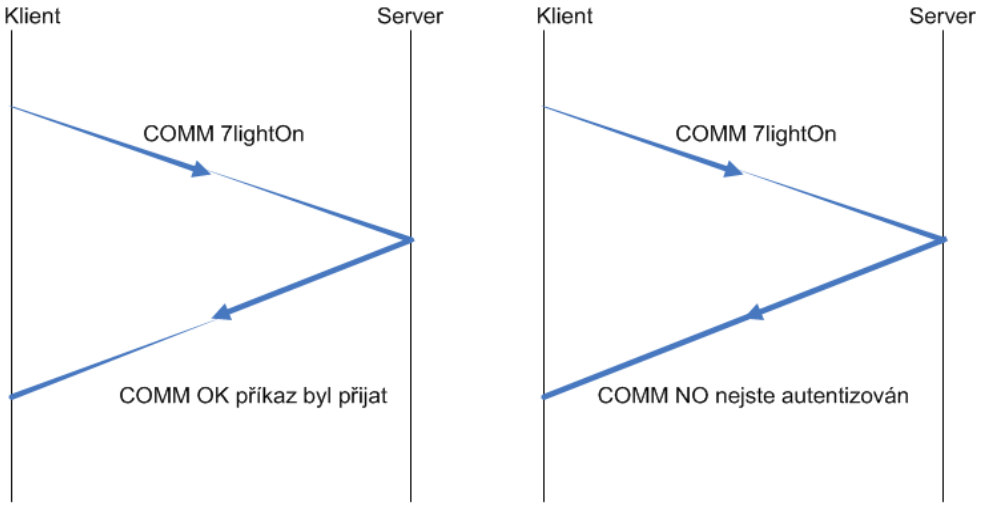
```
COMM <délka_APS_rámce><APS_rámec>
```

Syntaxe odpovědi:

```
COMM <status> <info_zpráva>
```

Příklad komunikace:

Na obrázku 15 jsou znázorněny příklady komunikace pro příkaz COMM.



Obrázek 15 Příklady komunikace pomocí příkazu COMM

4.1.1.3 Příkaz CHCK

Příkaz CHCK slouží pro kontrolu, zda na serveru je už uložena odpověď od senzorové sítě. Pokud není klient autentizován pomocí příkazu AUTH, tak je příkaz pro senzorovou síť zahozen a klient je informován o tom, že se má nejprve autentizovat. Zprávu od senzorové sítě můžeme získat dvěma způsoby, kde v prvním pomocí CHCK můžeme získat odpověď synchronně. U druhého způsobu je zpráva obdržena od serveru asynchronně, aniž bychom zaslali na server příkaz CHCK, ale vnitřní mechanismus serveru kontroluje, zda server obdržel od senzorové sítě zprávu pro klienta (v případě události v senzorové síti – například kouřový senzor zaslal zprávu o požáru). Jakmile server získá zprávu, tak ji zašle příslušnému klientovi ve formátu odpovědi příkazu CHCK. V odpovědi na příkaz CHCK od serveru je vložen stejně jako u příkazu COMM APS rámec a jeho délka, aby se dosáhlo kompatibility na úrovni komunikace.

Syntaxe příkazu CHCK:

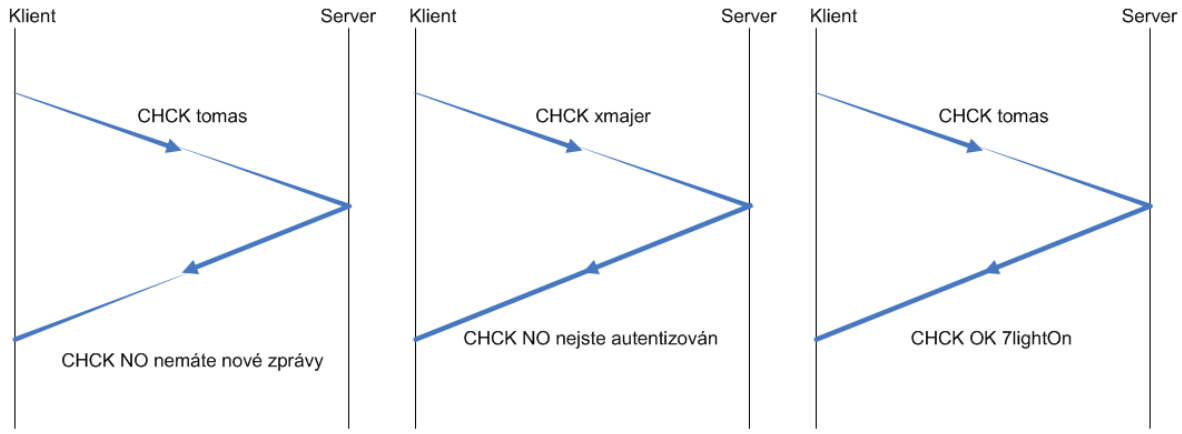
```
CHCK <uživatelské_jméno>
```

Syntaxe odpovědi:

```
CHCK <status> [info_zpráva] [délka_APS_rámce][APS_rámec]
```

Příklad komunikace:

Na obrázku 16 jsou znázorněny příklady komunikace pro příkaz CHCK.



Obrázek 16 Příklady komunikace pomocí příkazu CHCK

4.1.1.4 Příkaz CLSE

Příkaz CLSE slouží k odhlášení a odpojení od serveru.

Syntaxe příkazu CLSE:

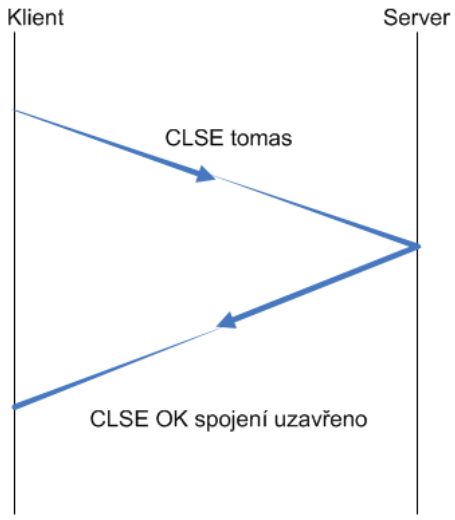
```
CLSE <uživatelské_jméno>
```

Syntaxe odpovědi:

```
CLSE <status> <info_zpráva>
```

Příklad komunikace:

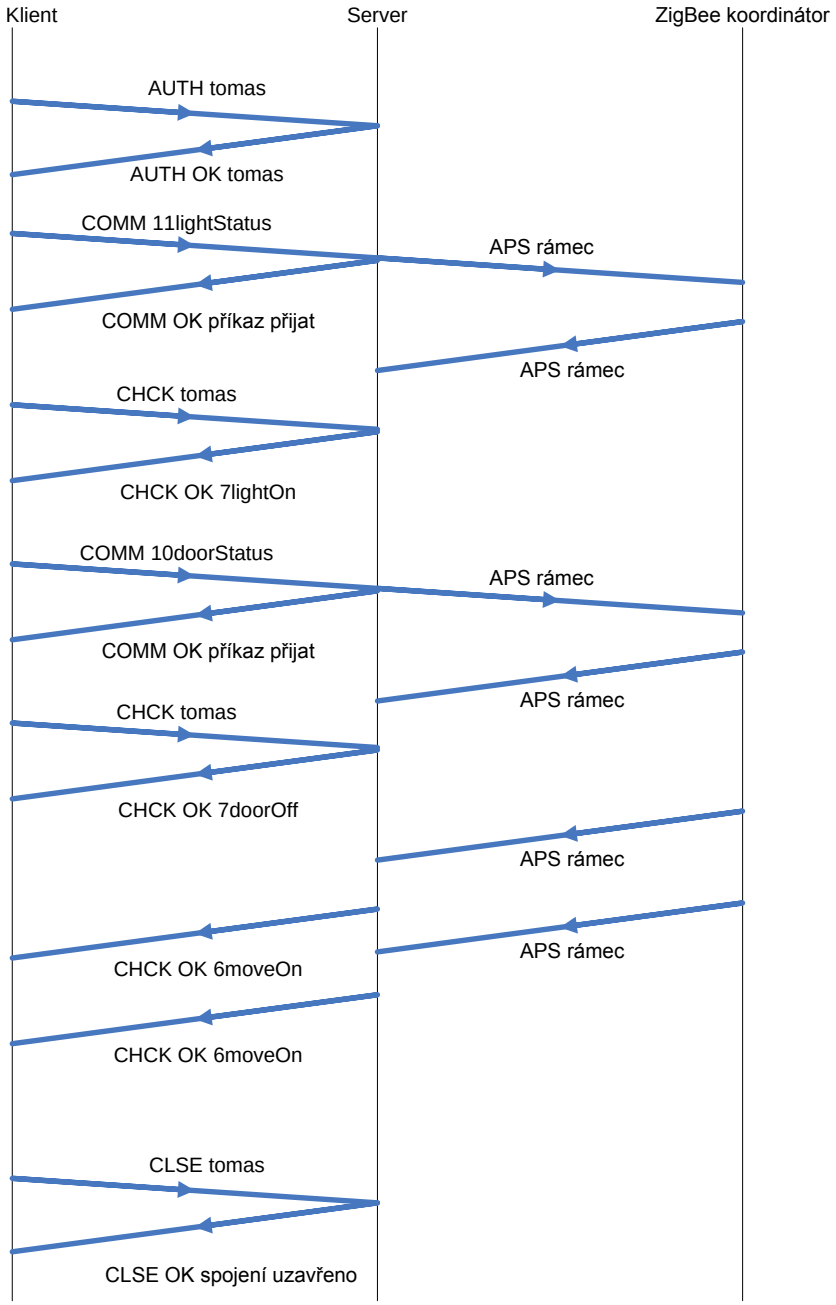
Na obrázku 17 jsou znázorněn příklad komunikace pro příkaz CLSE.



Obrázek 17 Příklad komunikace pomocí příkazu CLSE

4.1.1.5 Příklad průběhu komunikace pomocí komunikačního protokolu

Na obrázku 18 je uveden příklad celkové komunikace pomocí příkazů navrženého protokolu.



Obrázek 18 Pohled na příklad celkové komunikace

Komunikaci začíná klient příkazem AUTH pro svou autentizaci na serveru, který si ověří uživatelské jméno a zašle potvrzení. Klient zasílá příkaz COMM, ve kterém se dotazuje senzoru, zda je světlo rozsvíceno či zhasnuto. Od serveru dostává odpověď, že byl příkaz přijat. Server zasílá APS rámec na ZigBee koordinátor. Poté, co koordinátor získá odpověď od senzoru, zašle APS rámec s odpovědí na server, kde se čeká na dotaz od klienta. Klient pomocí příkazu CHCK získává odpověď, ve které je řečeno, že světlo je zapnuto. V další části ukázky je stejným způsobem zpracován dotaz na senzor hlídající bezpečnostní dveře. Poté přichází asynchronní událost ze strany

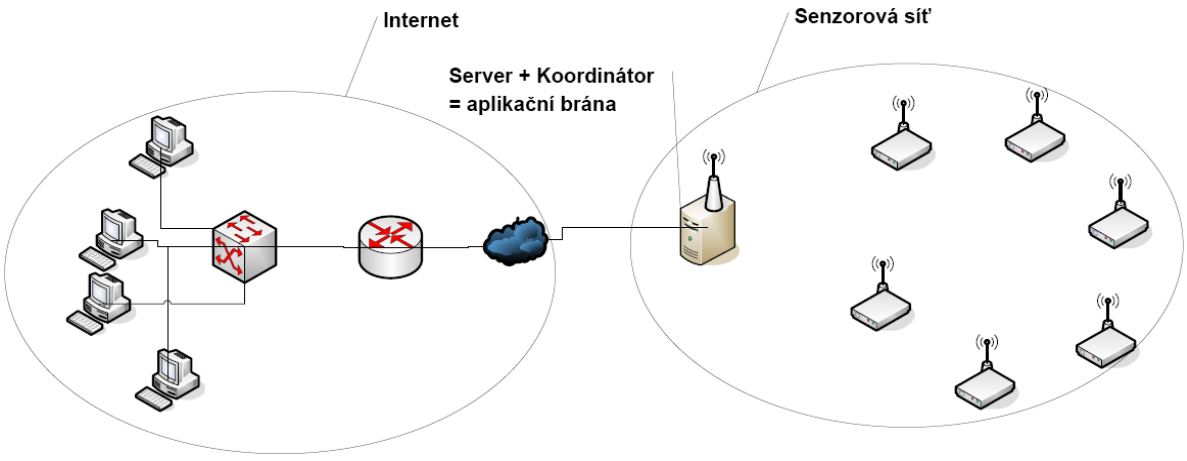
senzorové sítě. ZigBee koordinátor na tuto skutečnost upozorní APS rámcem, ve kterém je obsažena zpráva o události. Server APS rámec vloží jako parametr odpovědi příkazu CHCK a zasílá tuto odpověď klientovi. Klient obdrží zprávy o pohybu v místnosti a obsluha klienta může vyvolat alarm. Nakonec je komunikace mezi klientem a server ukončena.

4.1.2 Popis struktury komunikace

Aplikační brána je navržena tak, aby umožňovala obousměrnou komunikaci ze sítě Internet do senzorové sítě ZigBee a opačným směrem. Ve směru komunikace ze sítě Internet do senzorové sítě ZigBee obdrží koordinátor APS rámec, jenž bude zpracovávat a na jehož základě provede požadovanou akci v senzorové síti Zigbee. Pod pojmem požadovaná akce se skrývají veškeré možné akce a požadavky v senzorové síti ZigBee. Do senzorové sítě ZigBee z koordinátoru už putují příkazy pro jednotlivé elementy senzorové sítě. Ve směru komunikace ze senzorové sítě ZigBee do sítě Internet obdrží koordinátor příkaz od elementu senzorové sítě ZigBee a vytvoří APS rámec, který zašle serverové části aplikační brány.

Z důvodů odstranění přímé komunikace koordinátoru se sítí Internet, což by způsobilo jeho větší složitost a spotřebu energie, je jako mezistupeň zvolen serverová část aplikační brány (dále jen server). Server zajišťuje komunikaci s uživateli a ke koordinátoru je připojen přes sériové komunikační rozhraní RS-232 a komunikační přemostění (bridge). Server slouží jako vyrovnávací paměť mezi rychlou sítí Internet a senzorovou sítí ZigBee. Dále server zajišťuje přesměrování dat ze sítě Internet na sériové rozhraní, takže odpadá získávání dat z TCP/IP paketu na koordinátoru, který obdrží už jen APS rámec. Server nepřeposílá pouze data oběma směry, ale také ukládá požadavky do databáze. Na základě těchto dat je možno provádět analýzu přístupů k senzorové síti nebo získávání znalostí z databáze. Možnosti programu jsou tedy omezeny jen požadavky na analýzu dat.

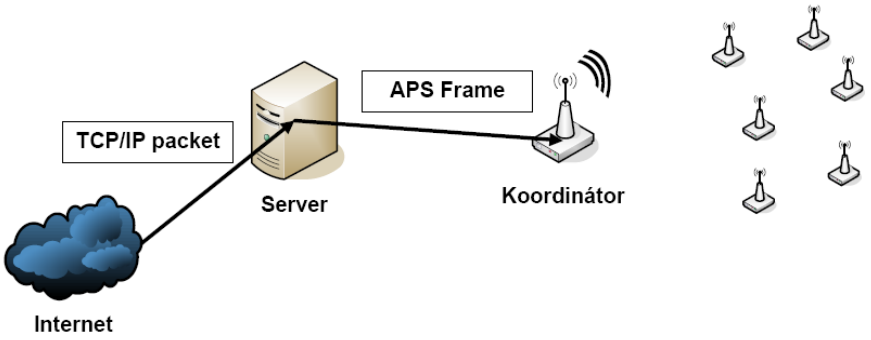
Pro představu jak bude vypadat struktura komunikace (a to nejen pro aplikační bránu) je na obrázku 19 zobrazeno komunikační schéma.



Obrázek 19 Komunikační schéma

Popis změny datových struktur při komunikaci

Na Obrázku 20 je zobrazen podrobný popis změny dat na aktivních prvcích komunikace.



Obrázek 20 Změny datových struktur na aktivních prvcích komunikace

V tabulce 17 uvádím přehled změn datových struktur při komunikaci mezi aktivními prvky.

Z Internetu do SN			Ze SN do Internetu		
Typ aktivního prvku [zdroj]	Typ aktivního prvku [cíl]	Formát datové struktury	Typ aktivního prvku [zdroj]	Typ aktivního prvku [cíl]	Formát datové struktury
Vzdálená aplikace	Server	TCP+IP+ [length + APS frame]	SN ZigBee	Koordinátor	[real ZigBee APS frame]
Server	Koordinátor	[length + APS frame]	Koordinátor	Server	[length + APS frame]
Koordinátor	SN ZigBee	[real ZigBee APS frame]	Server	Vzdálená aplikace	TCP+IP+ [length + APS frame]

Tabulka 17 Struktura dat na aktivních prvcích komunikace

4.2 Návrh implementace aplikační brány

Tato kapitola se zabývá návrhem implementace aplikační brány, nikoliv samotnými detaily implementace, která je představena v kapitole 5.

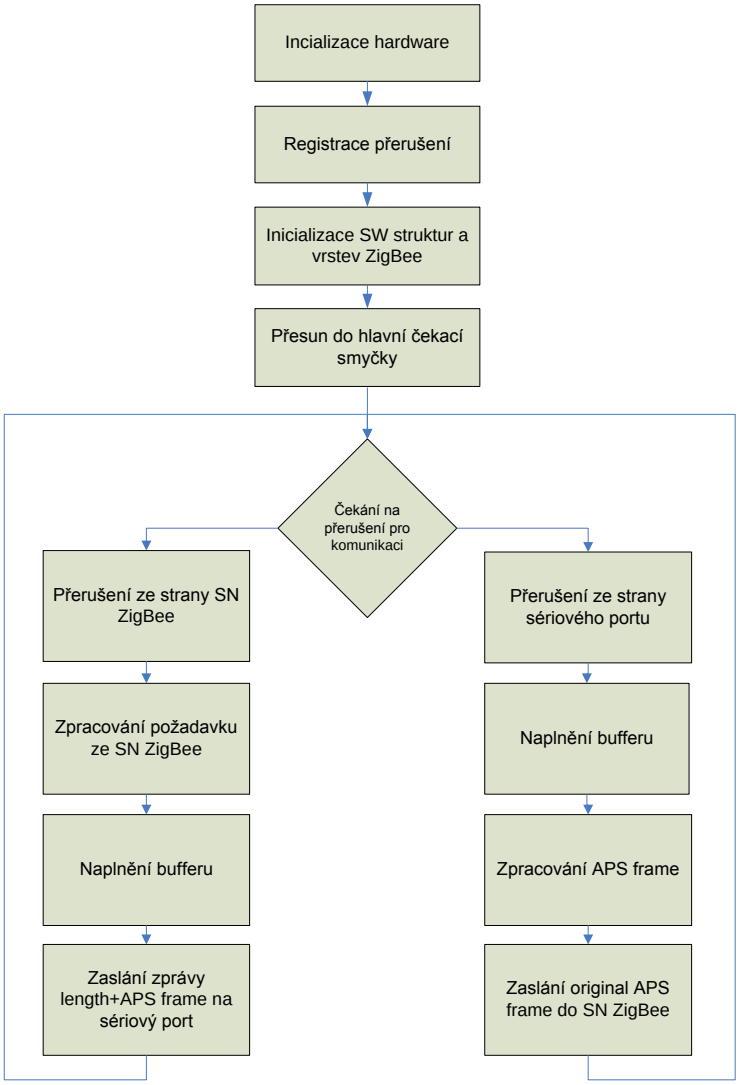
Jelikož se aplikační brána skládá ze dvou částí, je i návrh rozdělen na dvě části. První návrh implementace se zabývá programem ZigBee koordinátoru. Druhý návrh implementace se věnuje serverové části (dále jen server), která je spuštěna na počítači s operačním systémem Linux. Rozdíl v těchto dvou programech je i z pohledu programovacího jazyka či prostředí, ve kterém bude program fungovat. Pro implementaci serveru je použit programovací jazyk C++ a příslušné rozšiřující knihovny. Tento fakt umožňuje využít objektově orientované programování, knihovny a objekty starající se o síťovou komunikaci či další užitečné knihovny a funkce, které poskytuje OS Linux. Na straně koordinátoru je použit programovací jazyk ANSI C. O správu paměti, přerušení a periferie se stará program mikrokontroléru PIC18F4620.

Popis návrhu implementace jednotlivých částí aplikační brány jsem pro větší přehlednost rozdělil na dvě podkapitoly, kde se každá zabývá jiným směrem komunikace se senzorovou sítí

ZigBee. Samotná implementace obou částí aplikační brány bude podrobně zpracována v kapitolách 5.2 a 5.3.

4.2.1 Návrh implementace koordinátoru

Obrázek 21 zobrazuje blokový diagram implementace koordinátoru, který je z důvodu přehlednosti na vyšší úrovni abstrakce.



Obrázek 21 Implementace na koordinátoru

4.2.1.1 Zpracování dat z Internetu

Na začátku je provedena HW inicializace, registrace přerušení a SW inicializace struktur, bufferů a modulů. Při zpracování přerušení ze strany sériového portu je spuštěna příslušná funkce, která načte přijatá data do bufferu, jehož funkce jsou implementovány v jiném modulu než je hlavní program koordinátoru.

Zde je vidět jedna z výhod navrhnutého komunikačního protokolu, kde před samotným APS rámcem je definována jeho délka. Díky tomu můžeme data ze sériového portu přijímat po bytech.

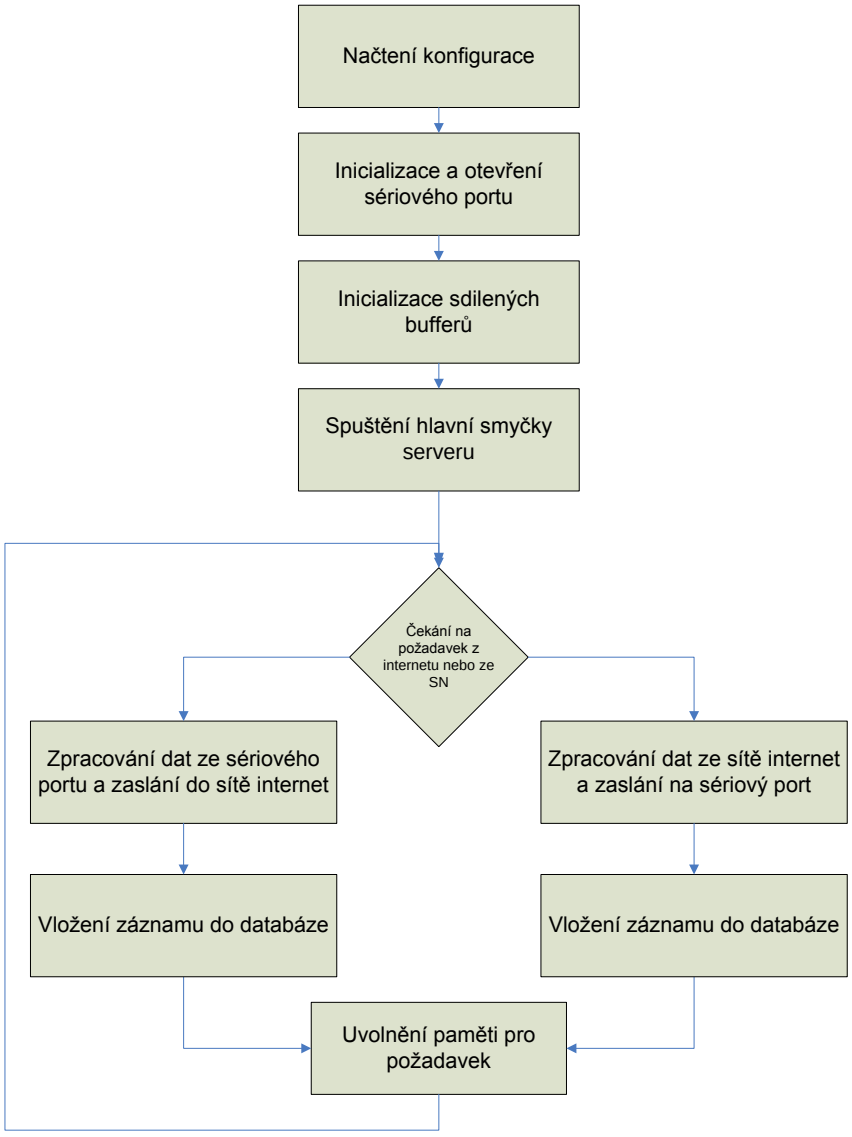
Jakmile je buffer naplněn daty, tak může začít zpracování APS rámce. Podle hodnot uložených v APS rámci se vybírá akce, která se v senzorové síti ZigBee provede. Dojde k naplnění příslušných struktur, které umožňují přesné poskládání rámců pro senzorovou síť ZigBee. Poté je zavoláno vysílání do senzorové sítě ZigBee. Tento postup se opakuje při každém přijetí dat ze sériového portu.

4.2.1.2 Zpracování dat ze senzorové sítě ZigBee

Implementace koordinátoru při obdržení zprávy ze senzorové sítě ZigBee zajistí vytvoření APS rámce. O vytvoření dat APS rámce se stará část programu, do které se přejde při indikaci požadavku ze strany senzorové sítě. Následně je APS rámec vložen do bufferu a z něj se zasílají data přes sériový port na server. Chyby při zpracování nebo při přijetí koordinátorem jsou také zasílána na server.

4.2.2 Návrh implementace serveru

Obrázek 22 zobrazuje blokový diagram implementace serveru.



Obrázek 22 Návrh implementace serveru

Zpracování síťové komunikace s klientskými aplikacemi je navrženo jako konkurentní na úrovni vláken, aby nedocházelo k blokování při přijímání či zasílání dat. Pro nastavení serveru jsou načítány hodnoty z konfiguračního souboru, ve kterém jsou uloženy i hodnoty pro nastavení a inicializaci komunikace se sériovým rozhraním. Pro komunikaci mezi klientskými vlákny a vláknem zabezpečujícím komunikaci se sériovým rozhraním budou sloužit sdílené buffery. Tyto sdílené paměti budou uzamykány, aby při zápisu nebo čtení jednotlivými vlákny nevznikaly nechtěné chyby.

4.2.2.1 Zpracování dat z Internetu

Po přijetí dat od klientské aplikace se zpracuje příkaz komunikačního protokolu. Server zajišťuje autentizaci uživatelů podle jejich uživatelského jména. Pokud jsou data určena pro senzorovou síť, zašlou se přes sériové rozhraní na ZigBee koordinátor. Do databáze se ukládá záznam o příkazu i s detaily potřebnými pro pozdější analýzu.

4.2.2.2 Zpracování dat ze senzorové sítě ZigBee

Při přijetí dat ze strany sériového rozhraní jsou data zaslána příslušnému uživateli na klientskou aplikaci. Vlákno sériového rozhraní zajišťuje kontext příkazů pro senzorovou síť od uživatelů, aby nedocházelo k chybnému zaslání informací jinému uživateli. Záznam o odeslaných datech je uložen do databáze. Do všech záznamů se ukládá i čas jejich zápisu do databáze a směr komunikace.

5 Implementace aplikační brány

Jelikož je kapitola 5 Implementace aplikační brány rozsáhlá, uvádím zde přehled toho, co lze v jednotlivých podkapitolách nalézt. V kapitole 5.1 jsou uvedeny parametry implementačních prostředí jednotlivých částí aplikační brány a jsou popsány požadavky na implementaci klientské aplikace. V kapitole 5.2 je představena implementace ZigBee koordinátoru. V kapitole 5.3 jsou uvedeny implementační detaily serverové části aplikační brány, kde jsou nejprve představeny stavební kameny serveru a poté je popsán celkový běh programu. V kapitole 5.4 jsou představeny modelové situace, na kterých byl testován reálný běh serveru.

5.1 Úvodní přehled implementace

Kapitola představuje implementační prostředí pro vývoj programu aplikační brány a také klientskou aplikaci pro komunikaci s aplikační bránou přes síť Internet. Pro klientskou aplikaci jsou určeny požadavky, které musí splňovat pro vzdálenou správu senzorové sítě ZigBee.

5.1.1 Přehled a představení implementačních prostředí

Kapitola ukazuje nástroje využité pro implementaci programů pro ZigBee koordinátor a server.

5.1.1.1 MPLAB IDE

Vývojové prostředí MPLAB IDE [8] spolu s překladačem jazyka C tvoří základní SW vybavení pro implementaci programu, který může být po přeložení nahrán pomocí programátoru MPLAB ICD 2 do samotného HW ZigBee koordinátoru (kapitola 5.2.1). MPLAB IDE i překladač je potřeba nainstalovat v operačním systému Windows (existují verze i pro jiné operační systémy). MPLAB IDE disponuje grafickým uživatelským rozhraním s možností vytvoření či úpravy projektu, ve kterém se nachází SW implementace chování ZigBee koordinátoru. Pro nahrání přeloženého programu do mikrokontroléru musí být k počítači připojen programátor MPLAB ICD 2 a také samotný ZigBee koordinátor.

MPLAB IDE kromě standardních vlastností vývojového prostředí pro implementaci programů do integrovaných čipů nabízí i možnost emulace chování ZigBee koordinátoru. Dále dává uživateli přehled o tom, jaká část dané paměti programu či dat má jaké procentuální využití.

Pro ladění chování implementace programu Zigbee koordinátoru doporučuji využít program Putty [11], pomocí kterého lze testovat zda program reaguje správně na podněty ze sériového rozhraní.

5.1.1.2 KDevelop

Pro vývoj implementace serveru jsem využil KDevelop [3], jenž umožňuje velmi pohodlné objektové programování v jazyce C++ se vším, co by mělo moderní integrované prostředí pro vývoj nabízet. KDevolop je primárně určen pro vývoj pod operačním systémem Linux.

5.1.2 Klientská aplikace

Návrh a implementace programu klientské aplikace není úkolem mé diplomové práce. V této kapitole popíši využitý program, který při testování nahradil klientskou aplikaci, a představím požadavky na klientskou aplikaci pro případ její implementace.

5.1.2.1 NetCat

Pro testování aplikační brány jsem jako klientskou aplikaci využil program NetCat [18]. NetCat je aplikace určená pro komunikaci po síti. Komunikačním protokolem je TCP/IP. Nabízí spoustu možností nastavení přenosu dat po síti, proto jsem ji použil jako náhradu za klientskou aplikaci pro testování funkčnosti aplikační brány. Podrobnější popis lze nalézt na oficiálních webových stránkách projektu NetCat [18].

5.1.2.2 Požadavky na klientskou aplikaci

Klientská aplikace by měla být implementována na míru pro využití praktické aplikace senzorové sítě. Aby klientská aplikace mohla využívat navrženou a implementovanou aplikační bránu, tak musí splnit následující požadavky:

- Neblokující čekání pro příjem a zasílání dat.
- Používat pro komunikaci se serverem navržený komunikační protokol (kapitola 4.1.1) na aplikační úrovni.
- Funkce pro vytvoření APS rámce na základě příkazu uživatele klientské aplikace.
- Funkce pro načtení APS rámce a jeho následnou reprezentaci uživateli ve srozumitelné podobě.
- Uložení uživatelského jména a případně hesla do zabezpečené části programu.
- Uživatelské rozhraní pro ovládání klientské aplikace.

Další funkce klientské aplikace už závisejí na požadavcích ze strany uživatelů. Doporučuji ovšem vytvořit grafické uživatelské rozhraní pro klientskou aplikaci, aby byla využitelnost celého projektu vhodná pro co největší spektrum uživatelů. Složitost grafického uživatelského rozhraní a jeho ovládání už záleží na potřebách uživatelů. Klientská aplikace může být využita i jako součást většího systému pro sledování a ovládání senzorové sítě.

5.2 Implementace ZigBee koordinátoru

Hlavním záměrem kapitoly je popsat problematiku implementace ZigBee koordinátoru. V kapitole 5.2.1 si představíme hardware ZigBee koordinátoru. Kapitola 5.2.2 nám ukáže využití implementace knihoven představujících ZigBee stack. Nakonec je v kapitole 5.2.3 představena implementace zpracování dat přicházejících na ZigBee koordinátor a popis, jak propojit koordinátor pro správnou komunikaci se serverovou částí aplikační brány.

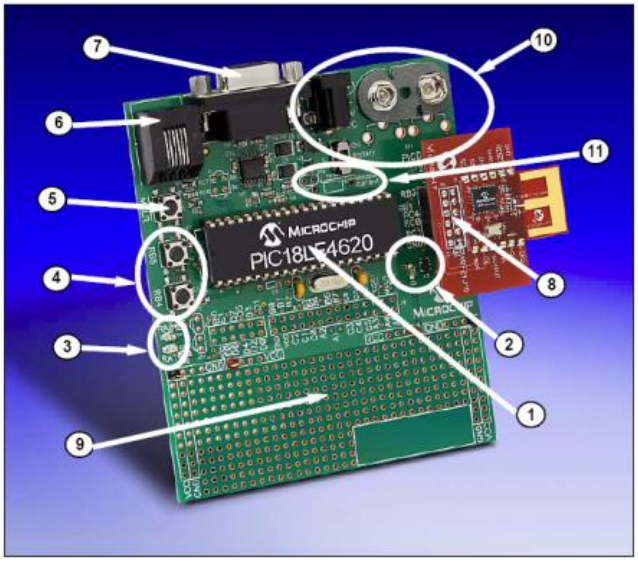
5.2.1 Hardware pro ZigBee koordinátor

5.2.1.1 Základní deska PICDEM Z

Základní deska PICDEM Z obsahuje [6]:

1. Mikrokontrolér PIC18LF4620 (kmitočet 4 MHz) osazený v 40 pinnové patici
2. Teplotní senzor Microchip TC77
3. Led diody D1 a D2
4. 2 tlačítka S2 a S3 s možností nastavení funkčnosti
5. Tlačítko s funkcí reset
6. RJ-11 konektor sloužící k připojení programátoru ICD 2
7. RS-232 konektor slouží k připojení základní desky k počítači
8. Konektor pro desku sloužící jako bezdrátový přijímač/vysílač
9. Rozšiřující pole pinů pro možné rozšíření funkčnosti desky
10. Konektor pro napájení desky (doporučené napájecí napětí 9V, maximální 16 V)
11. Konektor pro připojení měřících přístrojů

Na obrázku 23 je znázorněna základní deska PICDEM Z s číselným popisem odpovídajícím výše uvedenému seznamu. Obrázek 23 byl převzat z literatury [6].



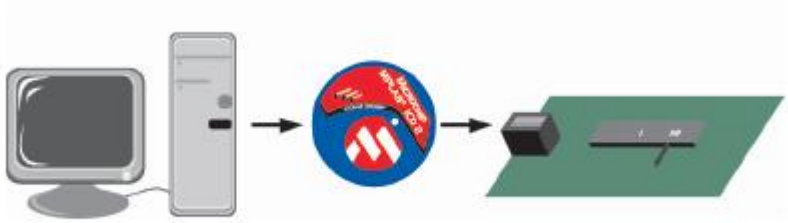
Obrázek 23 Základní deska PICDEM Z

Pro bezdrátovou komunikaci se senzory v senzorové síti ZigBee je použita deska PICDEM Z 2.4GHz RF, která je osazena přijímačem/vysílačem MRF24J40MA. Bezdrátový přenos probíhá na frekvenci 2.4 GHz. Na obrázku 24 je zobrazena deska PICDEM Z 2.4GHz RF. Obrázek byl převzat z literatury [6].

Obrázek 24 Deska pro bezdrátový přenos PICDEM Z 2.4GHz RF

5.2.1.2 Programátor MPLAB ICD 2

MPLAB ICD 2 (In-Circuit Debugger 2) [7] umožňuje spolu s vývojovým prostředím MPLAB IDE (kapitola 5.1.1.1) ladění a programování PIC a dsPIC Flash mikrokontrolerů. MPLAB ICD 2 programátor je při programování mikrokontroléru PIC18LF4620 propojen s počítačem přes rozhraní USB nebo RS-232 .Se základní deskou PICDEM Z je programátor propojen přes RJ-11 konektor. Po naprogramování mikrokontroléru je doporučeno rozpojit tento spoj. Na obrázku 25 lze vidět výše popsané propojení počítače, programátoru a základní desky. Obrázek 26 obsahuje programátor MPLAB ICD 2. Obrázky 25 a 26 byly převzaty z literatury [7].



Obrázek 25 Propojení počítače, programátoru a základní desky

Obrázek 26 Programátor MPLAB ICD 2

Vlastnosti programátoru MPLAB ICD 2

- Rozhraní USB a RS-232 pro připojení k hostitelskému počítači.
- Kompatibilnost s MPLAB IDE s možností ladění programu v reálném čase.
- Zápis a čtení paměti dat a programu z mikrokontroleru, možnost vymazání paměti programu.
- Možnost vkládání breakpointů pro ladění.

5.2.2 ZigBee knihovny – ZigBee Stack

Firma Microchip, která je výrobcem zařízení (koordinátor, směrovač nebo koncové zařízení) pro senzorovou síť ZigBee, poskytuje soubor knihoven určenou pro vývoj programů na těchto zařízeních. Tento soubor knihoven se nazývá ZigBee stack [5] (v některých literaturách se může objevit název Microchip stack), který je softwarovou implementací architektury vrstev v jazyce ANSI C. Podrobný popis architektury vrstev je v kapitole 3.2 ZigBee struktura vrstev.

ZigBee stack primárně zajišťuje funkci nižších vrstev (fyzická vrstva, vrstva přístupu k médiu a síťová vrstva) a aplikační vrstvě poskytuje rozhraní pro její specifikaci a implementaci. Pro rozhraní mezi aplikační vrstvou a ostatními vrstvami je v ZigBee stacku definována datová struktura *ZIGBEE_PRIMITIVE* a podle hodnoty proměnné definované touto strukturou se určuje aktuální stav ZigBee stacku.

Hlavní vlastnosti ZigBee stacku

- Certifikovaná platforma vyhovující standardu ZigBee 2006.
- Podpora bezdrátového přenosu dat o kmitočtu 2.4 GHz.
- Funkčnost pro všechna zařízení definována ZigBee protokolem.
- Implementace perzistentní uložiště pro skupinové tabulky a jiné kritické síťové parametry jako je nalezení souseda nebo správa směrovacích tabulek.
- Přenositelnost mezi velkým počtem mikrokontrolerů rodin PIC16 a PIC24.
- Podpora multicast adresování zařízení a jejich opětovné připojování do senzorové sítě.

5.2.3 Implementace zpracování dat

Pro svou implementaci ZigBee koordinátoru jsem využil knihoven ZigBee stacku, který je volně dostupné na webových stránkách firmy Microchip [22]. Součástí komprimovaného souboru s knihovnami ZigBee stacku je i soubor *Coordinator.c*, ve kterém je implementace základní kostry ZigBee koordinátoru. Implementace kostry zahrnuje hardwarovou inicializaci ZigBee koordinátoru, softwarovou inicializaci ZigBee struktur, funkci pro registraci přerušení a strukturu hlavní smyčky určenou pro zpracování proměnné typu *ZIGBEE_PRIMITIVE*. Zdrojové kódy knihoven ZiBbee stacku a součástí programu pro ZigBee koordinátor jsou přiloženy na DVD jako příloha 3.

5.2.3.1 Komunikace se senzorovou sítí - Hlavní smyčka

Zpracování příkazů a komunikace se senzorovou sítí spadající pod řídící ZigBee koordinátor je implementována v hlavní smyčce programu, která představuje stavový automat, jehož stav se mění podle hodnoty proměnné typu *ZIGBEE_PRIMITIVE*. Podle stavu proměnné začíná zpracování příchozích informací ze senzorové sítě. Pro ilustraci, jak stavový automat uvnitř hlavní smyčky vypadá, uvádím zde strukturu stavového automatu. Proměnná *currentPrimitive* je typu *ZIGBEE_PRIMITIVE*.

```
switch (currentPrimitive)
{
  case NLME_DIRECT_JOIN_confirm:          //potvrzení přímého spojení
  case NLME_NETWORK_FORMATION_confirm:    //vytvoření sítě koordinátorem
  case NLME_PERMIT_JOINING_confirm:       //potvrzení přijetí uzlu do sítě
  case NLME_ROUTE_DISCOVERY_confirm:      //směrování dat
  case NLME_JOIN_indication:              //připojení nového uzlu do sítě
  case NLME_LEAVE_indication:             //odpojení uzlu ze sítě
  case NLME_RESET_confirm:                //reset zigbee stacku
  case NLME_LEAVE_confirm:                //potvrzení odpojení uzlu ze sítě
  case APSDE_DATA_confirm:                //příjem zprávy od senzoru
  case APSDE_DATA_indication:             //indikace nové zprávy
  case APSME_ADD_GROUP_confirm:           //sestavení skupiny senzorů
  case APSME_REMOVE_GROUP_confirm:        //odstranění skupiny senzorů
  case APSME_REMOVE_ALL_GROUPS_confirm:   //odstranění všech skupin senzorů
  case NO_PRIMITIVE:                      //zpracování zpráv pro senzory
}
```

Bližší popis jednotlivých stavů lze nalézt v dokumentaci ZigBee stacku [5]. Stav automatu *NO_PRIMITIVE* je důležitý z hlediska zpracování dat ze strany serveru aplikační brány a bude popsán v kapitole 5.2.3.3.

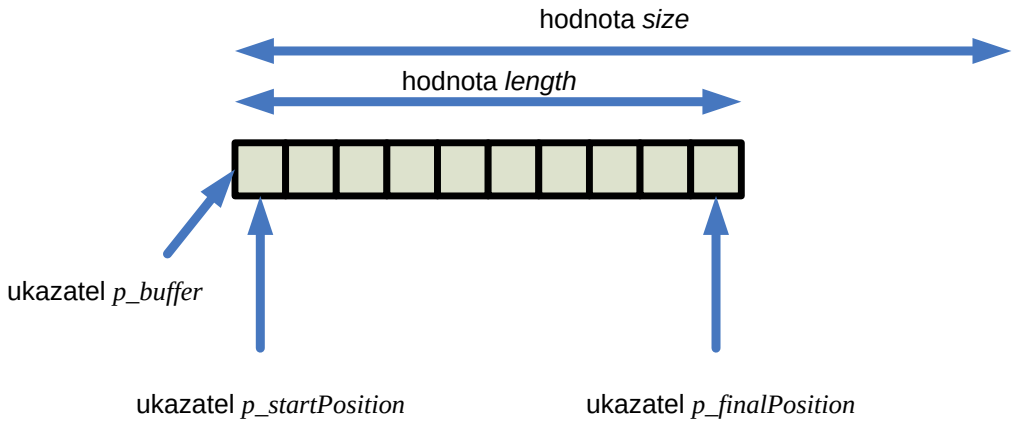
5.2.3.2 Implementace bufferu

Ještě předtím, než uvedu popis implementace zpracování dat ze strany serveru, chtěl bych prezentovat vyrovnávací paměť určenou pro komunikaci se sériovým rozhraním ZigBee koordinátoru. Abychom mohli rychle a správně zpracovávat APS rámce v koordinátoru, potřebujeme je mít v kompaktním bloku dat. Po přerušení ze strany sériového rozhraní můžeme ovšem získávat data pouze po jednom byte, což vede k implementaci modulu, který by zajišťoval funkci bufferu. Do bufferu se postupně ze sériového rozhraní načte celý APS rámec, který můžeme začít zpracovávat. Podle obsahu APS rámce koordinátor provede požadovanou akci v senzorové síti. Modul bufferu implementovaný v jazyce ANSI C obsahuje datovou strukturu *s_BUFFER* a funkce pro jednoduchou správu bufferu.

Datová struktura s_*BUFFER* má následující datové položky

- *byte *p_buffer* – ukazatel do paměti na byty (samotný buffer), do kterých se budou ukládat informace
- *int size* – celočíselná proměnná obsahující maximální velikost bufferu
- *int length* – celočíselná proměnná obsahující aktuální počet bytů v bufferu
- *byte *p_startPosition* – ukazatel na začátek paměti bufferu
- *byte *p_finalPosition* – ukazatel na konec paměti bufferu

Maximální velikost bufferu *size* je při inicializaci proměnné typu *s_BUFFER* nastavena na hodnotu odpovídající nejvyššímu počtu bytů možných pro vložení do jednoho APS rámce. Abychom zbytečně neplýtvali pamětí programu, počet bytů uložených v paměti je jen tak velký, jak je definována velikost právě získaného APS rámce. Nyní je vidět výhoda návrhu komunikačního protokolu (kapitola 4.1.1), neboť vnitřní část příkazu *COMM* obsahuje délku APS rámce a až poté následuje samotný APS rámec. Díky tomu můžeme nejprve načíst délku APS rámce a touto hodnotou nastavit hodnotu *length*, čímž získáváme dynamicky se měnící velikost bufferu podle velikosti APS rámce. Musíme ovšem zajistit kontrolu, zda právě přijímaný APS rámec není větší než maximální velikost bufferu (což může naznačovat chybný přenos informací). Ukazatele *p_startPosition* a *p_finalPosition* slouží pro kontrolu rozsahu paměti bufferu. Pro lepší představu je na obrázku 27 znázorněna struktura bufferu v paměti programu.



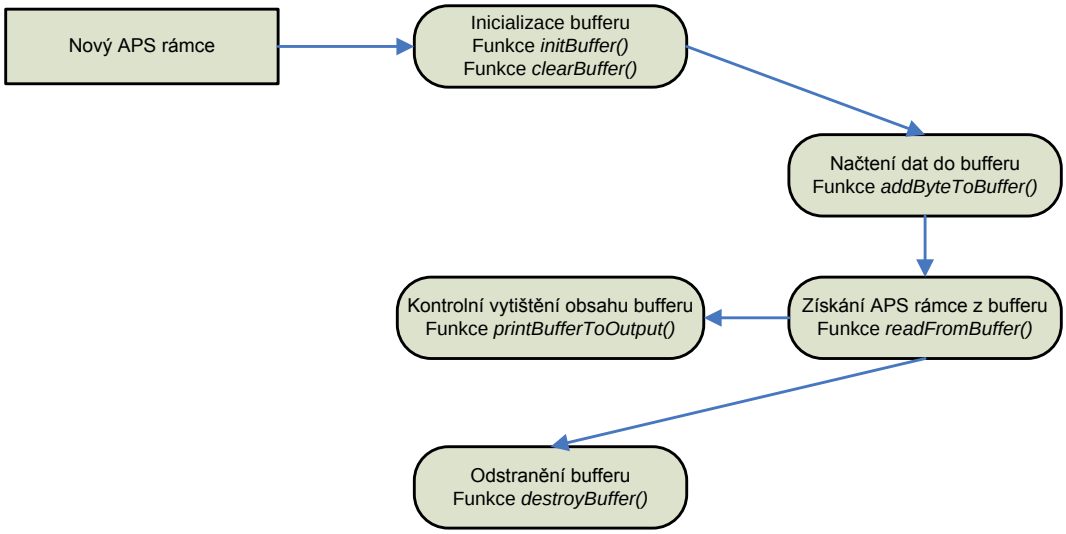
Obrázek 27 Znázornění struktury bufferu

Pro snadnější a bezpečnější práci s bufferem (správná alokace paměti, kontrola velikosti APS rámce, vyčištění paměti bufferu) jsem implementoval funkce určené pro jeho správu. Do přehledu funkcí neuvádím jejich parametry (ve všech je jako parametr ukazatel na strukturu *s_BUFFER* a proměnná, jejíž hodnota upravuje buffer podle názvu funkce) ani jejich návratovou hodnotu. Tyto informace jsou uvedeny v příloze 2, kde je vložen celý zdrojový kód rozhraní modulu bufferu i s komentáři.

Funkce určené pro správu bufferu

- *initBuffer()* –inicializuje strukturu bufferu na počáteční hodnoty a alokuje paměť bufferu na správnou velikost podle délky APS rámce.
- *addByteToBuffer()* – slouží pro přidávání dat o velikosti 1 byte do bufferu.
- *clearBuffer()* – vyprázdnění bufferu a nastavení velikosti aktuálního počtu dat uložených v bufferu na 0.
- *readFromBuffer()* – získání celý blok dat (APS rámce) uložený v bufferu.
- *printBufferToOutput()* – výpis obsah bufferu na standardní výstup.
- *destroyBuffer()* – uvolnění paměti, kterou buffer používal k ukládání dat.

Na obrázku 28 je znázorněn typický postup práce s bufferem v programu ZigBee koordinátoru.



Obrázek 28 Životní cyklus bufferu

5.2.3.3 Komunikace se serverovou částí aplikační brány

Na komunikaci se serverem lze nahlížet ze dvou pohledů, kde první je zaměřen na příjem dat ze serveru a druhý na zasílání dat na server.

Příjem a zpracování dat ze serveru

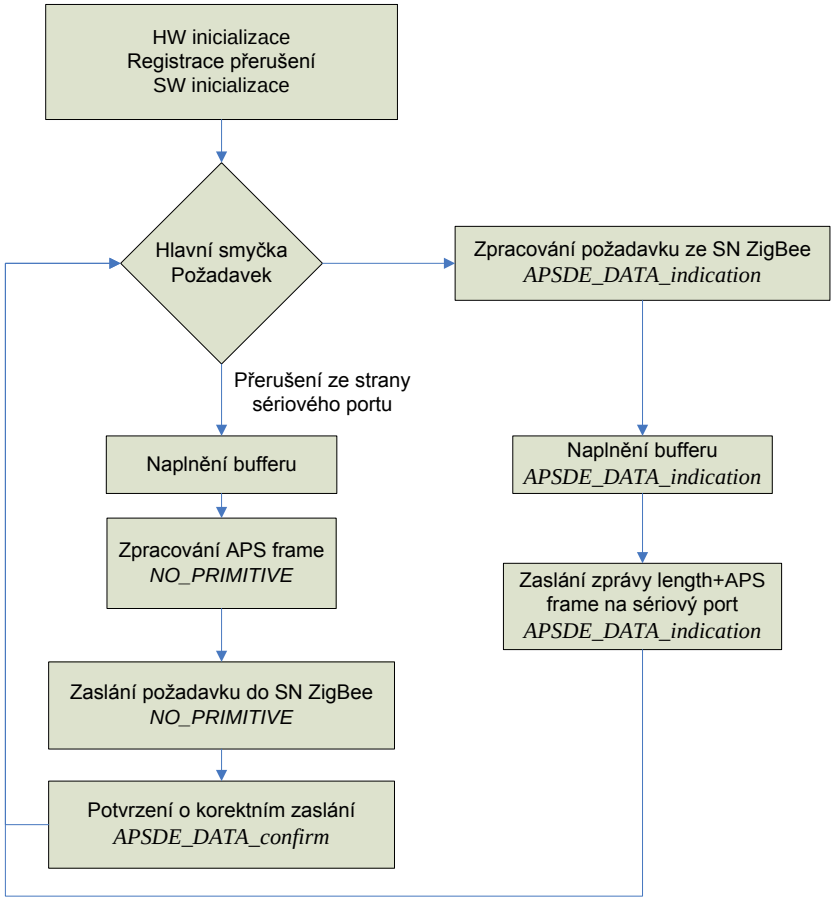
Data ze serverové částí aplikační brány jsou získávána přes sériové rozhraní ZigBee koordinátoru. Při přerušení ze strany sériového rozhraní jsou ukládána do bufferu (kapitola 5.2.3.2). Jakmile je APS rámec načten do bufferu, nastaví se proměnná fungující jako flag. V části *NO_PRIMITIVE* stavového automatu hlavní smyčky je tento flag kontrolován a pokud je nastaven, víme že můžeme začít APS rámec zpracovávat. Při jeho zpracovávání a převodu na příkaz pro senzorovou síť je nejprve z hlavičky APS rámce načteno kontrolní pole rámce, ze kterého zjistíme podrobnosti o hlavičce APS rámce. V ní jsou uloženy informace o typu rámce a dalších důležitých vlastnostech rámce. Podrobný popis APS rámce a jeho hlavičky je v kapitole 3.3.1 Formát APS rámce. Příslušnými částmi APS

rámce je naplněna struktura určená pro komunikaci se senzorovou sítí a data jsou zaslána vybraným aktivním prvkům senzorové sítě (směrovače, koncové zařízení). Informace o tom, na která zařízení nebo skupiny zařízení budou zaslána data, jsou uloženy v hlavičce APS rámce. Na konci každé části reprezentující stav automatu musí být nastaveno, do jakého stavu se přejde po ukončení aktuálního stavu. V našem případě se přejde do stavu *APSDE_DATA_confirm*, který zajistí kontrolu, zda data byly správně zaslány senzorům.

Zasílaní dat na server

Po příchodu dat ze strany senzorové sítě se v hlavní smyčce programu koordinátoru vstoupí do části *APSDE_DATA_indication*. Zde se data zpracují a pokud jsou data určena jako odpověď (zde nejsou brány v potaz potvrzovací odpovědi aktivních prvků senzorové sítě o tom, že korektně obdržely odpověď), vytvoří se APS rámec, který se vloží do bufferu. Velikost APS rámce a samotný APS rámec je z bufferu jako odpověď od senzorové sítě zaslán na server přes sériové rozhraní. Do dalšího stavu automatu tvořícího tělo hlavní smyčky se přejde podle typu přijatých dat.

Jak vypadá běh programu implementovaného na ZigBee koordinátoru je znázorněn na obrázku 29, na kterém je společný pohled na komunikaci se serverovou částí aplikační brány a komunikace se senzorovou sítí. V diagramu jsou znázorněny i stavy automatu z hlavní smyčky ZigBee koordinátoru.



Obrázek 29 Diagram běhu programu ZigBee koordinátoru

5.3 Implementace serveru

Kapitola se zabývá implementací programu serveru. V kapitole 5.3.1 jsou představeny detailní informace o stavebních kamenech implementace tak, aby v kapitole 5.3.2 mohl být popsán běh programu serveru v přehledné formě.

Hlavní vlastnosti serveru

- Zprostředkování komunikace se senzorovou sítí.
- Řízení komunikace se ZigBee koordinátorem přes sériové rozhraní.
- Nastavení serveru pomocí konfiguračního souboru.
- Konkurentní zpracování požadavků od klientské aplikace.
- Chování serveru je nezávislé na senzorové síti.
- Uživatelské rozhraní pro správu serveru.
- Databáze určená analýzu komunikace se senzorovou sítí.
- Řízení autentizace uživatelů.
- Implementace komunikačního protokolu.

5.3.1 Stavební kameny implementace serveru

Předtím, než se budeme zabývat samotným během programu, chtěl bych představit jednotlivé stavební kameny implementace serveru. Pro každou důležitou část implementace serveru jsem vytvořil kapitolu, ve které vysvětluji podrobnosti implementace dané části. Pro tento způsob popisu implementace jsem se rozhodl, abych předešel zmatku v kapitole 5.3.2, která se zabývá celkovým popisem běhu serveru. V kapitole 4.2.2 u návrhu implementace serveru jsem nastínil, jak server pracuje. Čtenář nyní získá podrobný přehled o jednotlivých částech implementace a poté se bude lépe orientovat v celkovém popisu běhu serveru. Může se to ovšem udělat naopak, kdy si čtenář nejdříve projde popis v kapitole 5.3.2 a poté se bude vracet ke kapitolám podrobně popisujících stavební kameny implementace.

5.3.1.1 Konfigurační soubor

Konfigurační soubor slouží pro nastavení konfigurace pro správný běh serveru. Je jedním z argumentů programu, které se načítají při spuštění serveru. Nastavení konfigurace se dělí na dvě části, kdy jedna část slouží pro konfiguraci síťového rozhraní serveru a druhá část je určena pro nastavení sériového portu, který server používá pro komunikaci se sériovým rozhraním. Pro zopakování možností nastavení sériového portu doporučuji nahlédnout do literatury [17].

Konfigurační soubor je ve formátu, který je obvykle používán pro programy běžící v operačním systému Linux. Pro komentáře se na začátku řádku používá znak „#“. V komentářích jsou u každého konfigurační parametru popsány informace pro specifické nastavení jednotlivých

parametrů. Parametry pro nastavení jsou textové řetězce, které jsou odděleny od klíčových slov mezerou. Pro větší variabilitu využití konfiguračního souboru je možné si nadefinovat vlastní konfigurační parametry. Pokud je konfigurační soubor změněn za běhu serveru, tak se změny projeví až při dalším spuštění serveru, neboť hodnoty z konfiguračního souboru jsou načteny při spuštění serveru.

Konfigurační parametry serveru

- **Síťové rozhraní**
  - *interface* – definuje síťové rozhraní, na kterém bude server naslouchat a čekat na nová příchozí připojení od klientů.
  - *port* – definuje číslo portu, na kterém bude server naslouchat a čekat na nová příchozí připojení od klientů.
  - *timeout*– definuje nastavení časové jednotky, po jejímž uplynutí budou kontrolovány sdílené buffery (kapitola 5.3.1.4) z důvodů asynchronní kontroly odpovědí senzorové sítě.
  - *buffer_size*– definuje velikosti bufferů sloužících pro příjem zpráv od klientů.

- **Rozhraní sériového portu**
  - *device* – definuje cestu k umístění sériového zařízení v adresářové struktuře operačního systému Linux.
  - *speed* – definuje rychlost přenosu na sériovém rozhraní.
  - *data_bits* – definuje počet datových bitů použitých při přenosu na sériovém rozhraní.
  - *parity*– definuje počet paritních bitů použitých při přenosu na sériovém rozhraní.
  - *stop_bits* – definuje počet stop bitů použitých při přenosu na sériovém rozhraní.
  - *flow_control* – definuje řízení toku při přenosu na sériovém rozhraní.

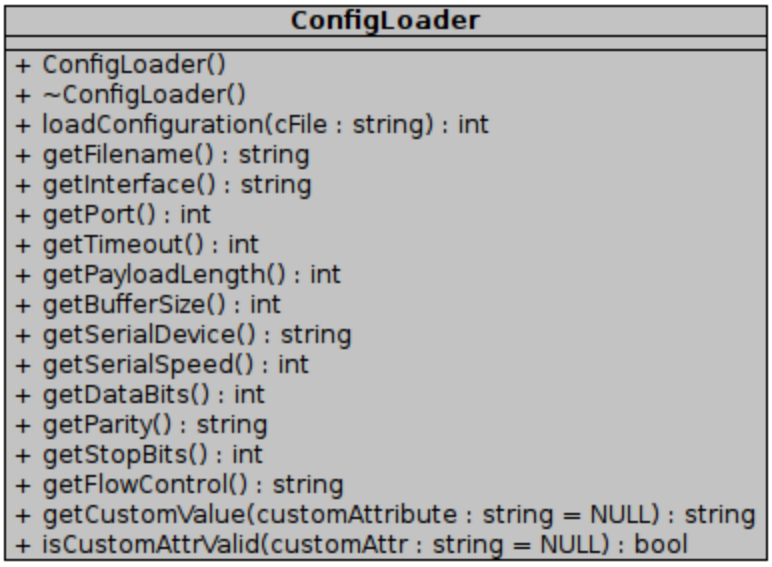
Pro všechny konfigurační parametry jsou v komentářích popsány rozsahy hodnot, které mohou být vloženy jako parametr ke klíčovému slovu konfiguračnímu portu. Pro lepší představu zde uvádím příklad z konfiguračního souboru *server.cfg* používaného při implementaci serveru, kde jsou uvedeny komentáře, konfigurační parametry a hodnoty parametrů. Celkový výpis ze souboru je umístěn v příloze 4.

Příklad ze zdrojového textu konfiguračního souboru server.cfg

```
# Definice rychlosti přenosu na sériovém rozhraní.
# Možné hodnoty: 75, 110, 134, 150, 200, 300, 600, 120, 1800, 2400, 4800,
#                9600, 19200, 38400, 57600, 115200
# Výchozí hodnota:   9600
speed 9600
```

Implementace načtení konfigurace

Pro načtení konfiguračních hodnot slouží třída *ConfigLoader*, která zajišťuje správné načtení a uložení do vhodných datových typů. Veřejné rozhraní metody objektů třídy *ConfigLoader*, které slouží k zisku jednotlivých hodnot, je na obrázku 30.



Obrázek 30 Veřejné rozhraní třídy ConfigLoader

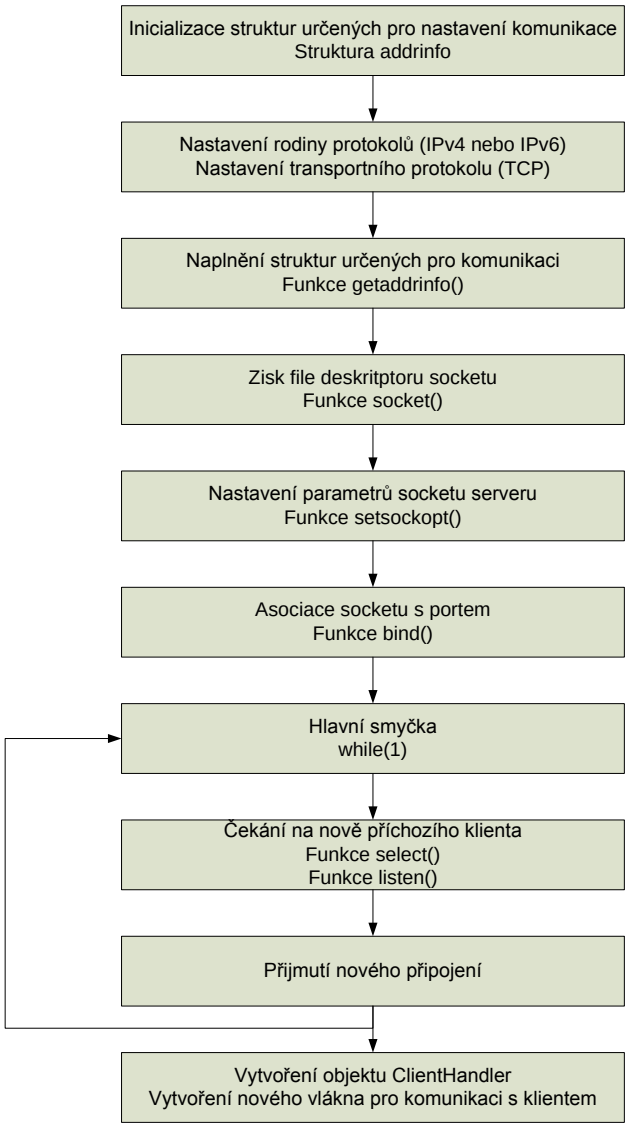
Nejdůležitější metodou třídy *ConfigLoader* z hlediska načtení hodnot z konfiguračního souboru je *loadConfiguration()*, jejíž parametrem je řetězec obsahující cestu v souborovém systému k tomuto souboru. Metoda otevře soubor, načte hodnoty a uzavře soubor. Pro načtení vlastních konfiguračních parametrů slouží metoda *getCustomValue()*. Parametrem metody je řetězec reprezentující název konfiguračního parametru. Metoda vrací příslušnou hodnotu uloženou v řetězci. Pro kontrolu, zda se v konfiguračním souboru opravdu nachází uživatelsky definovaný konfigurační parametr, slouží metoda *isCustomAttrValid()*, která vrací *true*, pokud se tam nachází a *false* při opačném případě.

Další metody jsou intuitivně pojmenovány podle toho, jaké hodnoty chceme načíst. Podrobnou programovou dokumentaci ke třídě *ConfigLoader* a jejímu veřejnému rozhraní lze nalézt na přiloženém DVD v sekci programová dokumentace. Přesnější informace o umístění programové dokumentace jsou v příloze 3.

5.3.1.2 Sockety pro síťovou komunikaci

Pro implementaci komunikace s klienty přes síť Internet využívám sockety [2] operačního systému Linux, jejichž API modul lze získat pomocí připojení knihovny *sys/socket.h*. Abych dosáhl neblokující obsluhy klientů, používám pro připojení nového klienta funkci *select()* a pro zpracování požadavků již připojeného klienta samostatné vlákno (kapitola 5.3.1.3), které se vytvoří, jakmile server přijme nového klienta pomocí funkce *accept()*.

Postup volání jednotlivých funkcí pro komunikaci s použitím socketů je znázorněn na obrázku 31. Programovou dokumentaci k jednotlivým funkcím lze nalézt v literatuře [19].



Obrázek 31 Průběh nastavení síťové komunikace serveru

Pro zasílání zpráv mezi serverem a klientem se používá funce *send()* a pro příjem zpráv se používá funkce *read()*, které jsou použity v implementaci třídy Clienthandler (kapitola 5.3.1.5), která zajišťuje komunikaci s připojeným klientem.

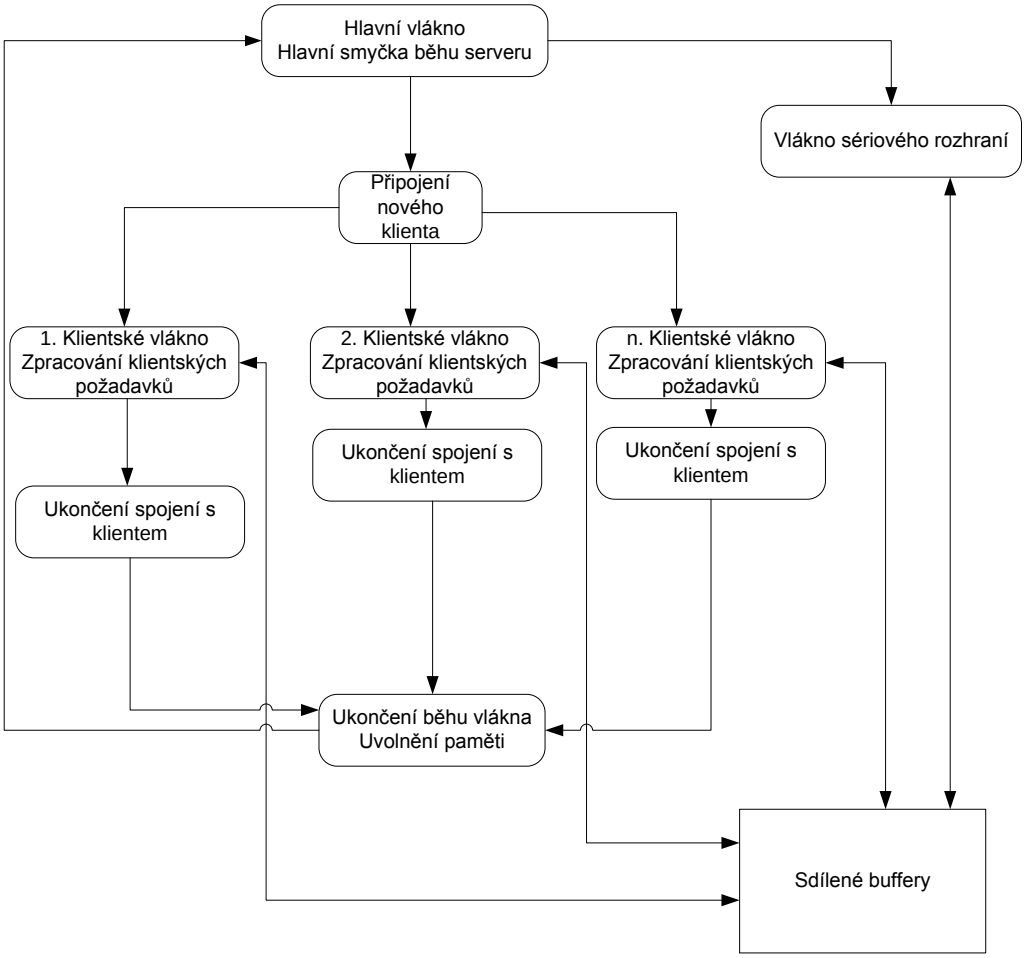
5.3.1.3 Konkurentnost serveru

Aby zpracování požadavků připojených klientů na straně serveru byla na sobě nezávislá, implementoval jsem server jako konkurentní na úrovni vláken. Pro implementaci této problematiky jsem použil POSIX vlákna [1]. Pro možnost práce s vlákny je potřeba pomocí direktivy *#include* připojit knihovnu *pthread.h* a při překladu programu použít přepínač *-lpthread*.

Zpracování požadavků od klientů je pro každého klienta prováděna ve vlastním vlákně, které je vytvořeno hlavním vláknem serveru po přijetí nového spojení s klientem. Zpracování komunikace se

sériovým rozhraním (5.3.1.6) a ZigBee koordinátorem (5.2.3) je také prováděno ve vlastním vlákně. Pro komunikaci mezi vlákny klientů a vláknem sériového rozhraní se používá dvou sdílených bufferů (5.3.1.4).

Obrázek 32 znázorňuje běh vláken zpracovávající požadavky klientů a komunikaci s vláknem sériového rozhraní přes sdílené buffery. Bližší popis komunikace vláken přes sdílené buffery jsou uvedeny v kapitole 5.3.1.4 a popis zpracování klientských požadavků je v kapitole 5.3.1.5.

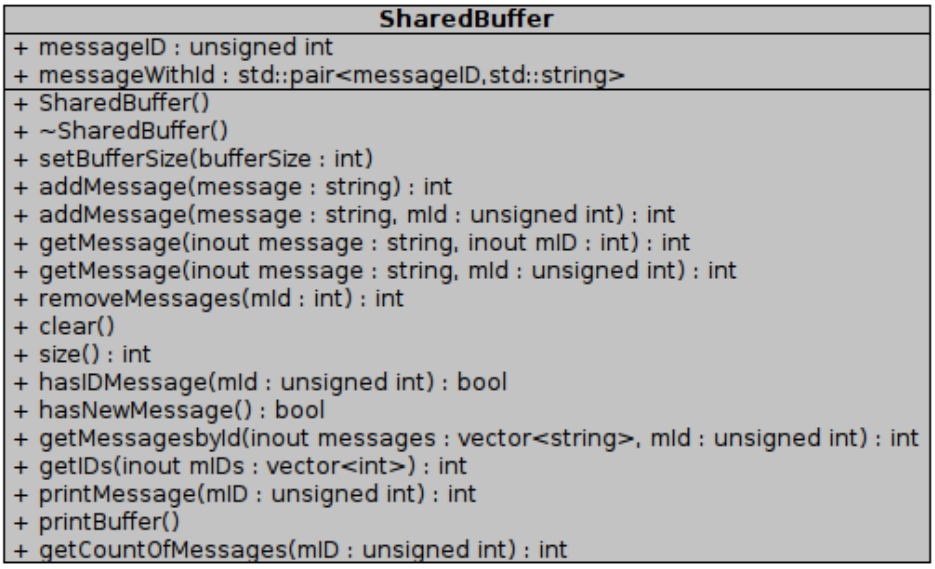


Obrázek 32 Běh vláken v programu

5.3.1.4 Sdílené buffery

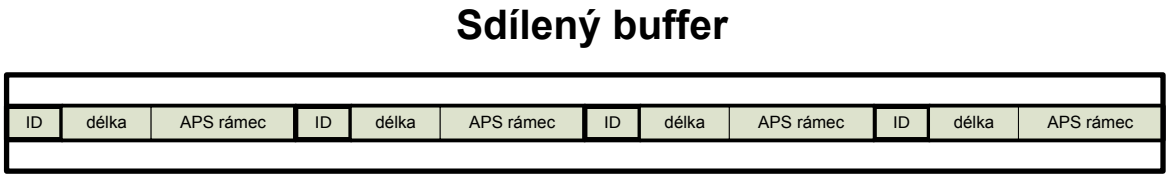
Sdílené buffery jsou datové struktury sloužící pro komunikaci mezi vlákny klientů a vláknem starajícím se o komunikaci se sériovým rozhraním. Buffer je implementován ve třídě *SharedBuffer*. Některé z metod veřejného rozhraní budou představeny dále v textu, ale podrobnou programovou dokumentaci ke třídě *SharedBuffer* a jejímu veřejnému rozhraní lze nalézt na přiloženém DVD v sekci programová dokumentace. Přesnější informace o umístění programové dokumentace jsou v příloze 3.

Veřejné rozhraní objektů této třídy je vyobrazeno na obrázku 33.



Obrázek 33 Veřejné rozhraní třídy Sharedbuffer

Pro komunikaci mezi vlákny jsou použity dva sdílené buffery (vstupní sdílený buffer a výstupní sdílený buffer), které jsou alokovány v paměti programu (heap) a přistupuje se k nim pomocí ukazatelů na objekty třídy *SharedBuffer*. Každá položka uložená v bufferu má identifikační číslo klienta (identifikační číslo klienta se načítá z databáze při autentizaci klienta a je uloženo pro každého klienta v objektu třídy *ClientHandler* – detaily jsou uvedeny v kapitole 5.3.1.5), který do bufferu zprávu vložil a samotnou zprávu. Identifikační číslo se používá pro rozlišení získávání zpráv ze sdíleného bufferu. Zprávy obsahují APS rámce i s jeho délkou pro komunikaci se senzorovou síti ZigBee. Vnitřní struktura je znázorněna na obrázku 34.



Obrázek 34 Vnitřní struktura sdíleného bufferu

Jelikož do sdílených bufferů se přistupuje z více klientských vláken a vlákna sériového rozhraní, implementoval jsem pomocí mutexů POSIX vláken uzamykání bufferu při jakémkoliv přístupu k datovým položkám v bufferu, abych předešel chybnému přepsání dat v této sdílené datové struktuře nebo se vyhnul ztrátě dat. Z tohoto plyne, že objekty třídy *SharedBuffer* jsou tzv. thread safe (bezpečné pro sdílení mezi více vlákny najednou).

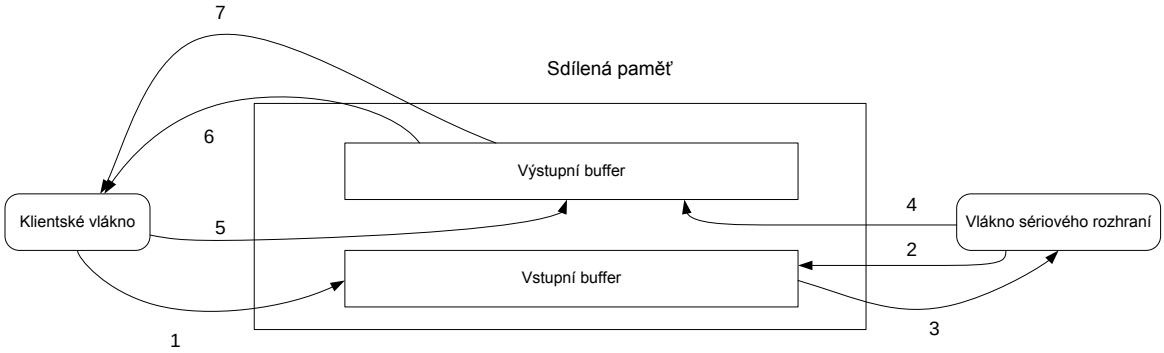
Funkce vstupního a výstupního sdíleného bufferu

- **Vstupní sdílený buffer** – slouží pro komunikaci mezi klientskými vlákny a vláknem sériového rozhraní ve směru komunikace příkazů pro senzorovou síť od klientů. Klientské

vlákno po obdržení příkazu COMM od klienta vloží pomocí metody *addMessage()* příkaz ve výše uvedeném formátu do vstupního bufferu (akce 1 na obrázku 27). Vlákno sériového rozhraní v pravidelných intervalech (jejichž časové rozpětí lze nastavit pomocí konfiguračního souboru 5.3.1.1) kontroluje tento vstupní buffer pomocí metody *hasNewMessage()* (akce 2) a pokud obdrží kladnou odpověď, vyjme metodou *getMessage()* zprávu z bufferu (akce 3) a začne ji zpracovávat (podrobný popis zpracování je v kapitole 5.3.1.6).

- **Výstupní sdílený buffer** – slouží pro komunikaci mezi klientskými vlákny a vláknem sériového rozhraní ve směru komunikace příkazů od senzorovou sítě ke klientům. Vlákno sériového rozhraní obdrží APS rámec i s jeho délkou od koordinátoru přes sériové rozhraní (akce 4). Vloží zprávu do výstupního sdíleného bufferu, který je v pravidelných intervalech (délku intervalu lze rovněž nastavit stejně jako pro případ vstupního sdíleného bufferu) kontrolován klientským vláknem (akce 6) a to z důvodu možnosti asynchronní události, která může nastat v senzorové síti. Výstupní sdílený buffer je kontrolován klientským vláknem i v případě, že vlákno obdrží příkaz CHCK od klienta (akce 5). Pokud klientské vlákno obdrží pozitivní odpověď od metody *hasIDMessage()*, jehož parametrem je identifikační číslo klienta, může z bufferu vyjmout zprávu metodou *getMessage()* (akce 7) a zaslat obsah zprávy klientovi.

Obrázek 35 znázorňuje komunikaci klientského vlákna a vlákna sériového rozhraní prostřednictvím vstupního a výstupního sdíleného bufferu. Popisy akcí pro jednotlivá čísla v obrázku jsou uvedeny v předchozím textu.

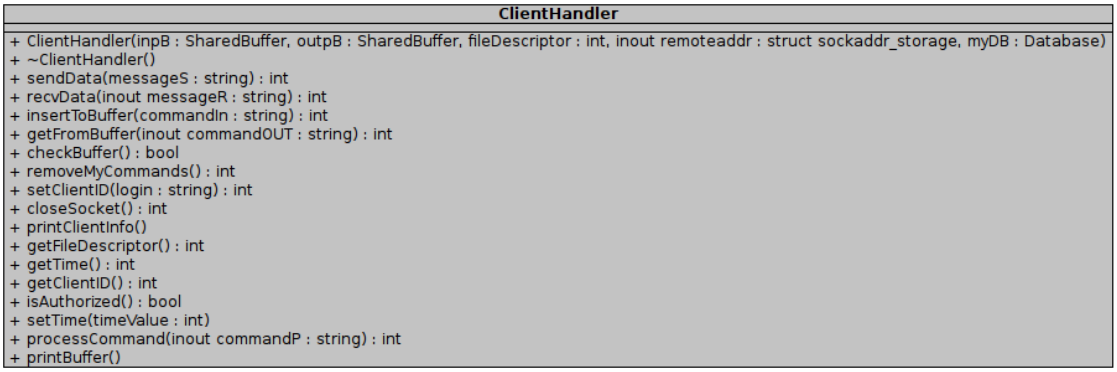


Obrázek 35 Komunikace mezi vlákny přes sdílené buffery

5.3.1.5 Zpracování příkazů klienta

Zpracování příkazů klienta probíhá pro každého klienta v samostatném vlákně, do kterého je předán jako argument vlákna objekt třídy *ClientHandler*, který byl vytvořen a inicializován před samotným vytvořením vlákna. Programovou dokumentaci ke třídě *ClientHandler* a jejímu veřejnému rozhraní lze nalézt na přiloženém DVD v sekci programová dokumentace. Přesnější informace o umístění programové dokumentace jsou v příloze 3.

Na obrázku 36 je znázorněno veřejné rozhraní objektů této třídy.



Obrázek 36 Veřejné rozhraní třídy ClientHandler

Ve vlákně probíhá uvnitř hlavní smyčky zpracování příkazů od klienta funkce *select()* (kapitola 5.3.1.2), která zajišťuje neblokující komunikaci s klientem. Zde se běh hlavní smyčky dělí na dvě části, kdy do první části zpracování se vstupuje při zachycení události na množině čtecích file deskriptorů, ve které je vložen file deskriptor klienta. Do druhé části zpracování se vstupuje při vypršení času ve funkci *select()*.

První část zpracování

Jak již bylo výše uvedeno, do této části se vstupuje při události na množině čtecích file deskriptorů ve funkci *select()*, což znamená, že na server přišla data od klienta. Data jsou přijata pomocí metody objektu ClientHandler *recvData()*, jejíž parametrem je řetězec, který bude naplněn přijatou zprávou. Při každém použití metody pro příjem data *recvData()* nebo metody *sendData()* se ukládá záznam o příkazu do databáze (bližší popis vkládaného záznamu je v kapitole 5.3.1.7). Tento řetězec je předán jako parametr metodě *processCommand()*, která zpracovává zprávu na základě hodnoty v záhlaví zprávy, jenž obsahuje příkaz komunikačního protokolu (kapitola 4.1.1). Podle příkazu se dělí běh zpracování zprávy na další části, kdy se po zpracování předá na začátek hlavní smyčky.

Typ příkazu:

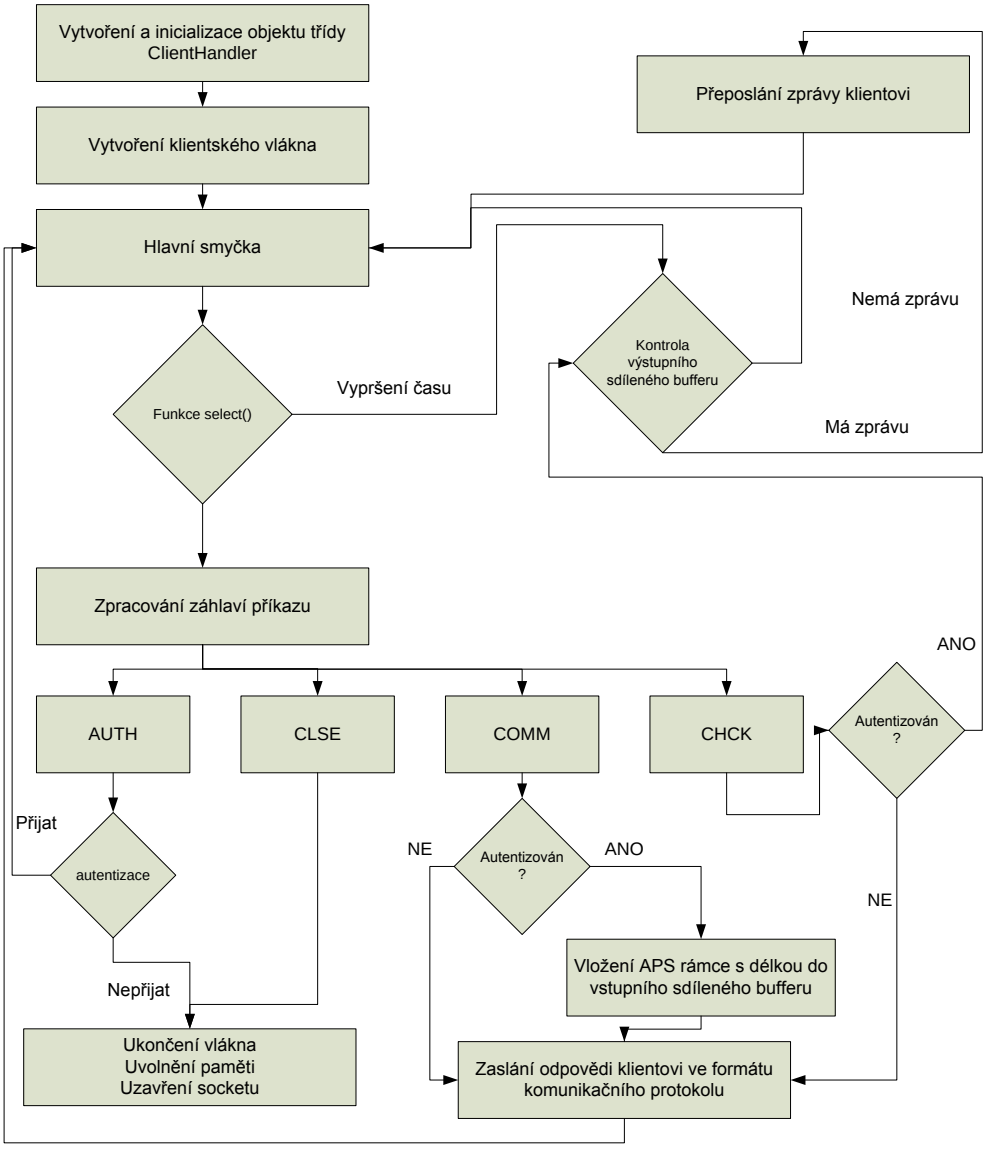
- ***AUTH*** – parametrem metody *setClientID()* se předá v řetězci uživatelské jméno z příkazu *AUTH*. Metoda zkontroluje, zda se uživatelské jméno nachází v databázi. Pokud je uživatel uložen v databázi, je klientovi přiděleno identifikační číslo pro komunikaci s vláknem sériového rozhraní přes sdílené buffery (kapitola 5.3.1.4) a objektu třídy *ClientHandler* je nastavena stavová proměnná, která určuje, zda je klient autentizován na hodnotu *true*. Není-li uživatelské jméno v databázi nalezeno je ukončeno spojení s klientskou aplikací metodou.
- ***COMM*** – na začátku se zkontroluje, zda je klient autentizován. Pokud není, zašle se klientovi oznámení v příslušném tvaru. Pokud je klient autentizován, vloží se tělo příkazu *COMM* obsahující APS rámec s jeho délkou do vstupního sdíleného bufferu a další zpracování se přenechává vláknu sériového rozhraní (kapitola 5.3.1.6).

- ***CHCK*** – na začátku se zkontroluje, zda je klient autentizován. Pokud není, zašle se klientovi oznámení v příslušném tvaru. Pokud je klient autentizován, metoda checkBuffer() zkontroluje, zda se ve výstupním sdíleném bufferu nachází zpráva pro klienta. Jestliže pro klienta zpráva existuje, je z bufferu vyjmuta a zaslána klientovi.
- ***CLSE*** – vlákno je ukončeno. Socket a spojení jsou uzavřeny. Uvolnění paměti od klienta.

Druhá část zpracování

Do této části hlavní smyčky se vstupuje při vypršení času ve funkci *select()*. Zde je implementována asynchronní kontrola sdíleného výstupního bufferu, abychom mohli získat APS rámec případné události vyvolané senzorovou sítí. Pokud je v bufferu nějaké zpráva, přeposílá se klientovi, jinak se pokračuje na začátek hlavní smyčky.

Na obrázku 37 je blokový diagram zpracování příkazů a komunikace s klientem.

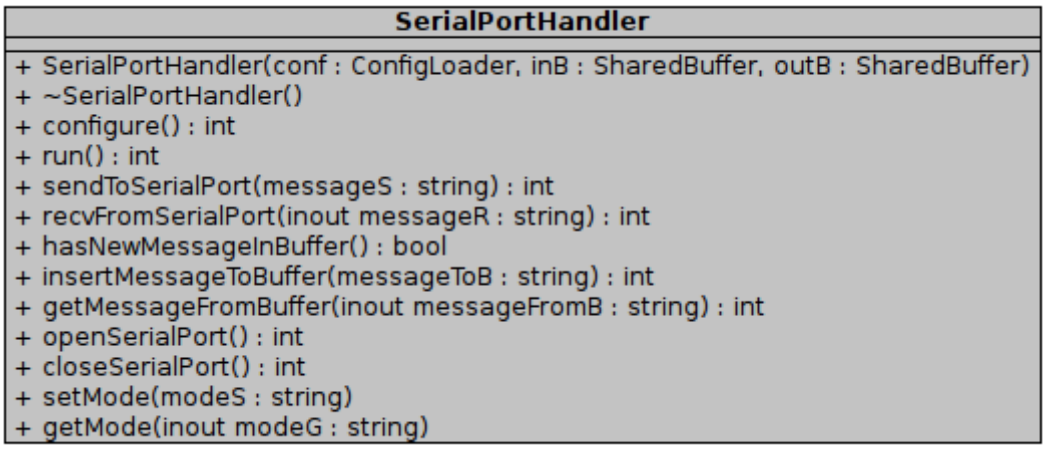


Obrázek 37 Blokový diagram zpracování komunikace s klientem

5.3.1.6 Sériové rozhraní

Pro komunikaci se ZigBee koordinátorem se využívá sériové rozhraní. Implementace programu umožňující komunikaci se sériovým rozhraním pracuje ve vlastním vlákně a o přímou komunikaci se stará objekt třídy *SerialPortHandler*. Programovou dokumentaci ke třídě *SerialPortHandler* a jejímu veřejnému rozhraní lze nalézt na přiloženém DVD v sekci programová dokumentace. Přesnější informace o umístění programové dokumentace jsou v příloze 3.

Veřejné rozhraní třídy *SerialPortHandler* je na obrázku 38.



Obrázek 38 Veřejné rozhraní třídy SerialPortHandler

Po načtení informací z konfiguračního souboru (kapitola 5.3.1.1) a vytvoření sdílených bufferů (kapitola 5.3.1.4) se vytvoří objekt třídy *SerialPortHandler*. Metoda *openSerialPort()* zajistí otevření komunikace po sériovém portu. Poté je zavolána metoda *configure()*, která provede nastavení komunikace se sériovým portem na základě načtených hodnot z konfiguračního souboru. Pokud se při otevírání spojení či nastavení komunikace objeví chyba, je vypsána na standardní chybový výstup i s doporučením k řešení.

Nyní se spouští vlákno sériového rozhraní, jehož parametrem je objekt třídy *SerialPortHandler*. Ve vlákně se pomocí metody *run()* implementuje zpracování komunikace se ZigBee koordinátorem a s klientskými vlákny přes sdílené buffery.

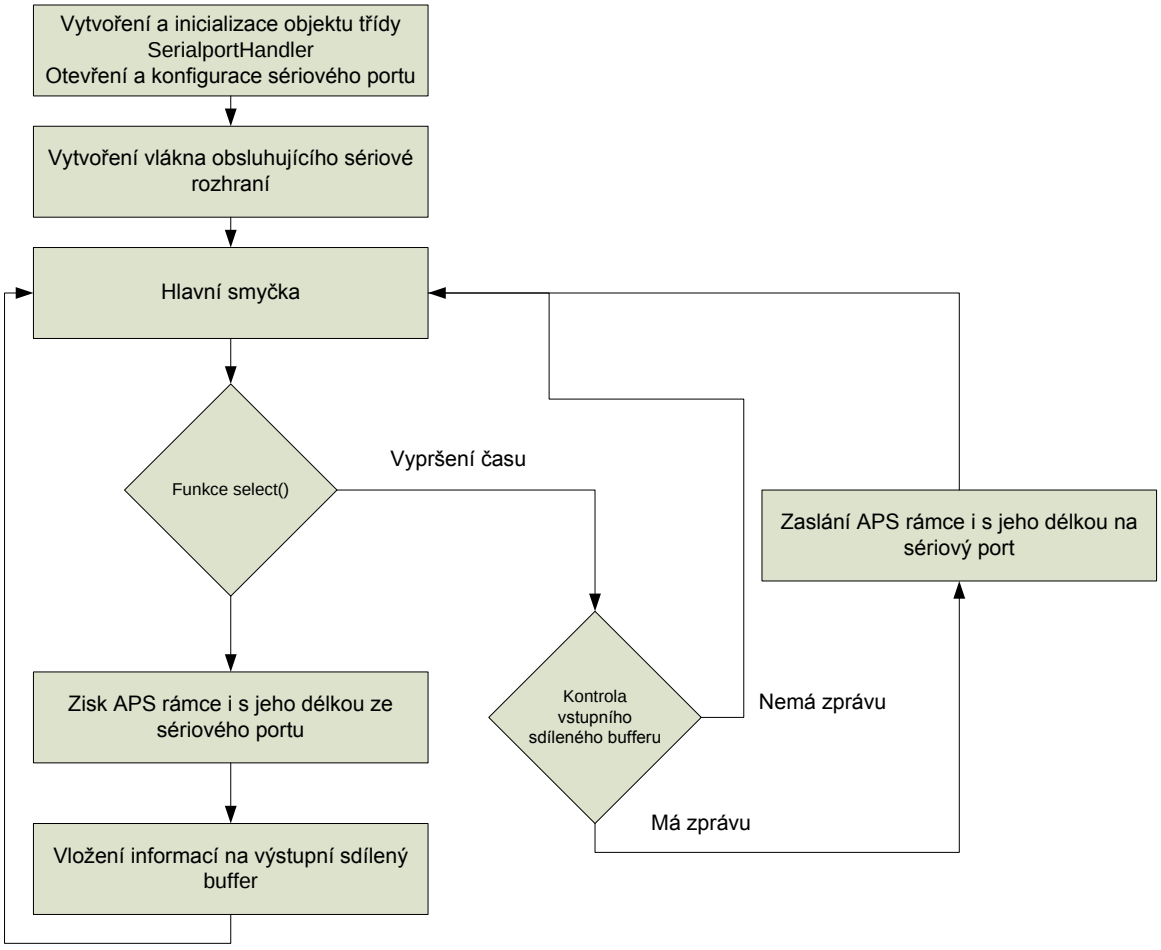
Popis běhu programu v metodě *run()*

Po vstupu do hlavní smyčky v metodě *run()* se zavolá funkce *select()* (kapitola 5.3.1.2), která rozděluje program na dvě části. Do první části programu se vstupuje po vypršení času ve funkci *select()*. Tato část se stará o komunikaci s klientskými vlákny přes sdílené buffery. Testování a získávání informací ze sdílených bufferů funguje na stejném principu jako v klientském vlákně (kapitola 5.3.1.3). Ze vstupního sdíleného bufferu získává informace, které přeposílá na sériové rozhraní pomocí metody *sendToSerialPort()*. Identifikační číslo právě zpracovávaného klienta se

uloží do fronty, aby byl zachován kontext a pořadí zpracovávaných příkazů. Nakonec se řízení programu navrátí na začátek hlavní smyčky.

Do druhé části programu se vstupuje při zachycení události ve funkci *select()* na množině čtecích file deskriptorů, ve které je uložen file deskriptor sériového portu. Zde začíná čtení dat ze sériového portu metodou *recvFromSerialPort()*. Získaná zpráva od ZigBee koridnátoru se vloží do výstupní díleného bufferu. Nakonec se řízení programu navrátí na začátek hlavní smyčky.

Na obrázku 39 je blokový diagram komunikace ve vlákně sériového rozhraní.



Obrázek 39 Blokový diagram komunikace ve vlákně sériového rozhraní

5.3.1.7 Databáze

Databáze je důležitou součástí serverové části aplikační brány, neboť jsou v ní důležité informace pro autentizaci, ukládání záznamů o komunikaci a definování vlastních příkazů pro uživatelské rozhraní serveru (kapitola 5.3.1.8). Databáze slouží jako perzistentní uložiště dat. Pokud byste chtěli provádět jakoukoliv operaci s databází, můžete do ní přistupovat přes příkazy uživatelského rozhraní serveru.

Jako jednoduchý typ databáze jsem zvolil projekt *SQLite* [14], který nabízí sadu knihoven pro práci s databází. Samotná struktura databáze je uložena do jediného souboru. Pro samotnou komunikaci s databází jsem využil C++ wrapper z projektu *Sqlite wrapped* [16]. Tento projekt nabízí třídy pro základní komunikaci s databází. Implementace projektů *SQLite* a *Sqlite wrapped* jsou

odděleny od mého zdrojového kódu a adresář s touto implementací je označen názvem contrib (zkratka pro contribution s významem poskytnutí cizího zdrojového kódu). Přesnější informace o umístění zdrojových kódů jsou v příloze 3. Dokumentace na oba projekty je v odkazech, které se nacházejí v literatuře [14], [16] .

Autentizace pomocí databáze

V databázi je uložena tabulka *users* obsahující identifikační číslo uživatele, který může komunikovat se senzorovou sítí, a jeho uživatelské jméno. Tato tabulka se využívá při autentizaci klienta (kapitola 5.3.1.5), kdy dotaz na tabulku buď vrací identifikační číslo klienta nebo chybu o tom, že klient s daným uživatelským jménem není zaregistrován.
Vnitřní struktura tabulky users:

- *id* – celé kladné číslo udávající jedinečné identifikační číslo klienta.
- *name* – řetězec obsahující uživatelské jméno.

Záznamy o komunikaci

Pro přehled příkazů, které přicházejí na server, je určena tabulka *command*, kde jsou ukládány veškeré důležité informace o příkazu komunikačního protokolu a informace o klientovi. Ukládají se informace o komunikaci v obou směrech (jak ze strany klienta, tak odpovědi serveru). Tyto ukládané záznamy mohou posloužit k analýze příkazů pro senzorovou síť, Například pro zjištění neoprávněného přihlašování na.

Vnitřní struktura tabulky *command*:

- *id* – celé kladné číslo udávající jedinečné identifikační číslo záznamu o komunikaci.
- *time* – řetězec obsahující datum a čas vložení záznamu (ve formátu z kapitoly 5.3.1.9).
- *commandH* – řetězec obsahující záhlaví příkazu komunikačního protokolu.
- *commandB* – řetězec obsahující tělo příkazu komunikačního protokolu.
- *ip* – řetězec obsahující IP adresu klienta.
- *port* – celé kladné číslo udávající číslo portu, na kterém byl klient připojen.
- *direct* – řetězec obsahující směr komunikace.
- *id_client* – celé kladné číslo udávající jedinečné identifikační číslo klienta zadefinované jako cizí klíč do tabulky *users*.

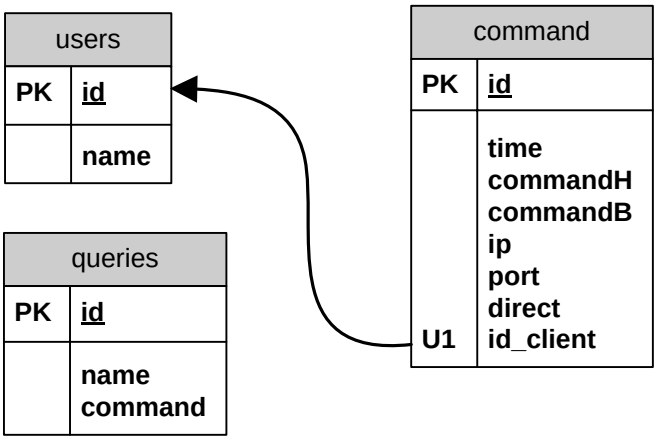
Definování vlastních příkazů pro uživatelské rozhraní

Uživatelské rozhraní serveru (kapitola 5.3.1.8) nabízí mnoho předdefinovaných příkazů, ale pokud by jste si chtěli nadefinovat vlastní, můžete vložit do tabulky queries dotaz na databázi pod zvolený alias.

Vnitřní struktura tabulky *queries*:

- *id* – celé kladné číslo udávající jedinečné identifikační číslo definovaného příkazu
- *name* – řetězec obsahující alias pro příkaz
- *command* – řetězec obsahující tělo dotazu na databázi ve formátu SQL.

Na obrázku 40 je zobrazena struktura a vztah tabulek v databázi.



Obrázek 40 Stuktura a vztahy tabulek uložených v databázi

5.3.1.8 Uživatelské rozhraní

Pro server jsem implementoval uživatelské rozhraní, které umožňuje získávat informace o všech akcích, které jsou na serveru prováděny. Toto uživatelské rozhraní má konzolový výstup na standardní výstup programu. Vstupem pro jeho ovládání je standardní vstup spuštěného programu, kde je možno zadávat příkazy. Po spuštění programu serveru a úspěšném spuštění všech jeho komponent je možné zadávat předdefinované příkazy pro výpis informací nebo je možné si nadefinovat vlastní výpisy z databáze (kapitola 5.3.1.7) pod zvolený alias. Pokud zadáte příkaz, který není definován nebo je špatně syntakticky zapsán (jsou rozlišována velká a malá písmena), jste upozorněni a program doporučí zadat příkaz *HELP*.

Přehled předdefinovaných příkazů pro uživatelské rozhraní

- **HELP** – výpis nápovědy, ve které je přehled všech předdefinovaných příkazů pro uživatelské rozhraní. Pokud si chcete nechat vypsat vámi definované příkazy, použijte příkaz pro výpis informací z tabulky *queries*.
- **SHUTDOWN** – bezpečné vypnutí serveru. Po zadání příkazu jsou odpojeni všichni připojení klienti a uzavřena komunikace se sériovým rozhraním. Na závěr je uvolněna veškerá paměť a program je ukončen. Tento příkaz doporučuji používat při zastavování běhu serveru, neboť nevzniknou úniky paměti a vše je bezpečně ukončeno.
- **SHOW ALL** – výpis všech databázových tabulek a informací uložených v nich. Při častém využívání serveru tento příkaz nedoporučuji používat, neboť výpis z tabulky *command* je

velmi dlouhý, což značně znepřehlední výstup uživatelského rozhraní. Proto doporučuji použít příkazy pro výpis z jednotlivých tabulek (*SHOW CLIENTS, SHOW COMMANDS*).

- **SHOW CLIENTS** – výpis informací uložených v tabulce *users*. V tomto výpisu si můžeme zkontrolovat, kteří uživatelé mají oprávnění zasílat příkazy pro senzorovou síť. Pro vložení nového klienta lze použít příkaz *SQLIN* popsaný dále.
- **SHOW COMMANDS** – výpis informací uložených v tabulce *command*. Tento výpis je stejně jako výpis příkazu SHOW ALL při častém využívání serveru velmi dlouhý, a proto doporučuji používat jiné předdefinované příkazy nebo si nadefinovat vlastní.
- **SHOW ZCOMMANDS** – výpis všech příkazů pro senzorovou síť a jejich odpovědi či události ze senzorové sítě.
- **SHOW IN** – výpis všech příkazů komunikačního protokolu přicházejících od klientů na server.
- **SHOW OUT** – výpis všech příkazů komunikačního protokolu odcházejících ze serveru na klienty.
- **SHOW ZIN** – výpis příkazů pro senzorovou síť přicházejících od klientů na server.
- **SQLIN** – Příkaz slouží pro práci s databází ve smyslu vkládání SQL příkazů typu *CREATE, INSERT, UPDATE* nebo *DELETE*. Popis syntaxe příkazu a příklady použití jsou z důvodů své složitosti uvedeny pod tímto výpisem.
- **SQLOUT** – Příkaz slouží pro práci s databází ve smyslu vkládání SQL příkazu *SELECT* s jeho parametry, čímž nám umožňuje přesnější výpisy nebo definování vlastních příkazů. Popis syntaxe příkazu a příklady použití jsou z důvodů své složitosti uvedeny pod tímto výpisem.

Popis příkazu SQLIN

Příkaz slouží pro vytvoření nových tabulek, vkládání nových řádků, změnu vybraných řádků, mazání řádků, mazání tabulek či smazání celé databáze. V podstatě umožňuje s vnitřní strukturou databáze provádět veškeré akce definované v literatuře [15] kromě výpisů pomocí SQL příkazu *SELECT*.

V případě zadání špatného SQL příkazu program upozorní na chybu a oznámí jaký typ chyby jste při jeho zadávání udělali. V syntaxi příkazu je uveden volitelný příkaz, což značí veškeré příkazy povolené výše. Závorky v syntaxi mají stejný význam, jako při definování komunikační protokolu (kapitola 4.1.1).

Základní syntaxe příkazu:

```
Vstup: SQLIN <volitelný_příkaz>
Výstup:[výpis_chyby]
```

Příklad příkazu:

- Vložení nového uživatele: `SQLIN insert into users values(100,'login')`

Pomocí příkazu *SQLIN* můžeme definovat vlastní příkazy pro výpis pod určitý alias. Do tabulky queries vložíme pod určitým jménem SQL dotaz, který se provede v případě, že zadáme příkaz ve tvaru *SHOW jméno_sql_dotazu*.

Příklad předefinovaných příkazů:

- Definice příkazů pro zobrazení všech předdefinovaných příkazů uložených v databázi s jeho následným použitím pomocí příkazu *SHOW*:

```
SQLIN insert into queries values(NULL, 'dotazy', 'select * from
queries')
SHOW dotazy
```

- Definice příkazů pro zobrazení všech příkazů od uživatele tomas pro senzorovou síť uložených v databázi s jeho následným použitím pomocí příkazu *SHOW*:

```
SQLIN insert into queries values(NULL, 'tomas', 'select * from
command, users where users.id=102 and command.commandH='AUTH')
SHOW tomas
```

Popis příkazu SQLOUT

Příkaz slouží pro výpisy z databáze pomocí SQL příkazu *SELECT*, kde si můžeme nechat vypsat jakoukoliv informaci uloženou v databázi. Tělo příkazu *SELECT* může obsahovat jakýkoliv dotaz podle syntaxe příkazu definovaného v literatuře [15]. V případě zadání SQL příkazu SELECT v nesprávné syntaxi vás program upozorní na chybu a oznámí, jaký typ chyby jste při jeho zadávání udělali.

Základní syntaxe příkazu:

```
Vstup: SQLOUT <select> <volitelný_příkaz>
Výstup: <výpis_pro_příkaz> [výpis_chyby]
```

Příklad příkazu:

- Ze kterých IP adres a na jakém portu se přihlašoval uživatel tomas:

```
SQLOUT select command.ip, command.port from command, users where
       users.id=command.id_client and users.name='tomas'
```

- Zobraz datum a čas přihlašování uživatele tomas:

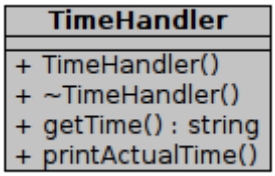
```
SQLOUT select command.time from command, users where
       users.id=command.id_client and users.name='tomas'
```

- Počet zaslaných příkazů pro senzorovou síť od uživatele 102:

```
SQLOUT select count from command where command.id_client=102 and
                    command.commandH='COMM'
```

5.3.1.9 Aktuální datum a čas

Pro ukládání záznamů o příchozích či odchozích příkazů na serveru do databáze je důležité mít možnost zjistit přesné datum a čas vložení záznamu do databáze, abychom mohli provést analýzu dotazů, například z hlediska bezpečnosti. Pro získání těchto hodnot slouží implementace *TimeHandler*. Veřejné rozhraní objektů této třídy je znázorněno na obrázku 41.



Obrázek 41 Veřejné rozhraní třídy TimeHandler

Nejdůležitější metodou třídy *Timehandler* je getTime(), která vrací řetězec obsahující aktuální datum a čas. Datum a čas jsou v následujícím formátu:

```
DD-MM-RRRR HH:NN:SS
```

Legenda k formátu řetězce:

- *DD* – den v měsíci v rozsahu 01 – 31
- *MM* – měsíc v roce v rozsahu 01 – 12
- *RRRR* – rok v rozsahu 1900 – 2100
- *HH* – hodina ve 24 hodinovém formátu v rozsahu 00 – 23
- *NN* – minuta v hodině v rozsahu 00 – 59
- *SS* – sekunda v minutě v rozsahu 00 – 59

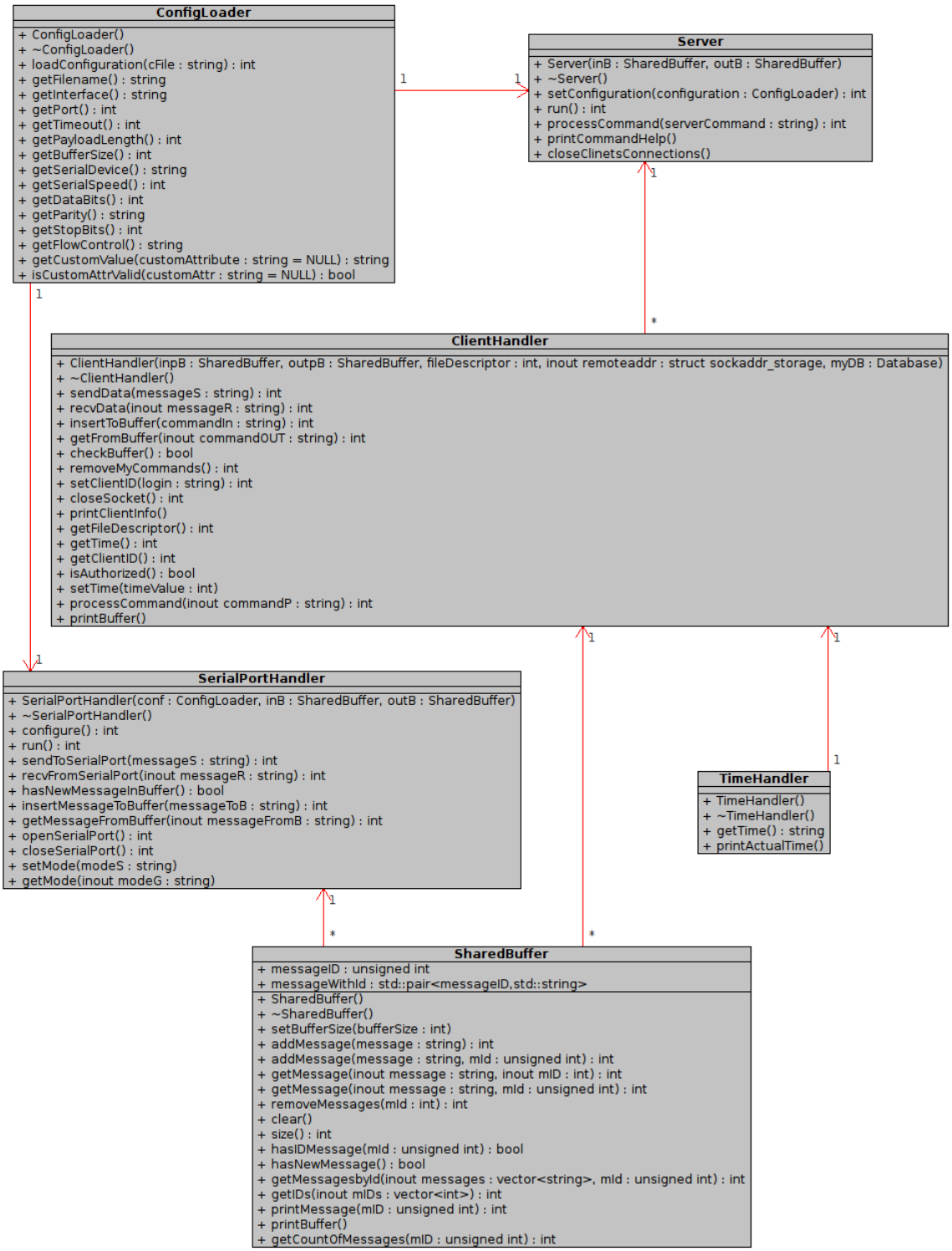
Podrobnou programovou dokumentaci ke třídě *Timehandler* a jejímu veřejnému rozhraní lze nalézt na přiloženém DVD v sekci programová dokumentace. Přesnější informace o umístění programové dokumentace jsou v příloze 3.

5.3.2 Popis běhu serveru

Nyní by měl mít čtenář přehled o detailech základních stavebních kamenů implementace serveru. V této kapitole si představíme celkový diagram tříd implementace, blokový diagram běhu programu a popíšeme si slovně běh programu serveru.

5.3.2.1 Diagram tříd

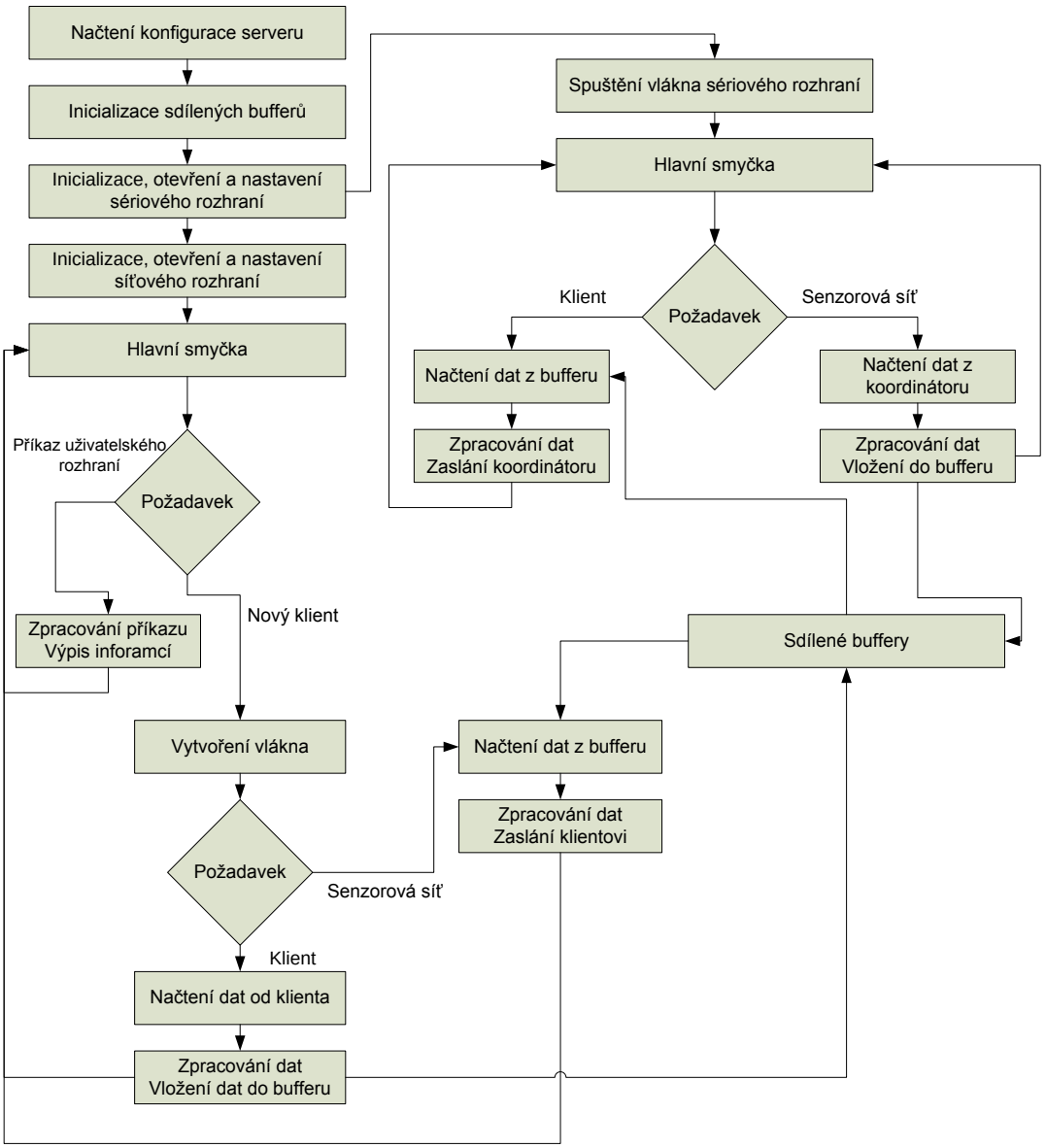
Abychom získali přehled nad celkovou implementací serveru, nachází se na obrázku 42 diagram tříd s veřejnými rozhraními tříd a vztahy mezi třídami ve formátu UML 2.0 (http://www.uml.org/). Diagram tříd byl vytvořen v programu Umbrello UML Modeller (http://uml.sourceforge.net/).



Obrázek 42 Diagram tříd ve formátu UML 2.0

5.3.2.2 Blokový diagram serveru

Na obrázku 43 je blokový diagram běhu programu serveru. Některé části nejsou až tak detailní, neboť by byl diagram nepřehledný. Detailní popisy těchto částí diagramu jsou popsány v kapitole 5.3.1 Stavební kameny implementace serveru v příslušných podkapitolách. Do diagramu nejsou zahrnuty části vedoucí k zastavení běhu programu (například příkaz uživatelského rozhraní *SHUTDOWN*).



Obrázek 43 Blokový diagram běhu programu

5.3.2.3 Popis běhu programu

Popis běhu programu serveru se odvíjí od obrázku 43. Nebudou zde popisovány detaily z kapitoly 5.3.1. Veškeré nekorektní vstupy, příkazy nebo hodnoty jsou ošetřeny a na standardní chybový výstup serveru jsou vypisována varování a odůvodnění těchto chyb, často s doporučením pro jejich odstranění.

Po spuštění serveru je načtena jeho nastavení z konfiguračního souboru (kapitola 5.3.1.1). Následuje inicializace sdílených bufferů (5.3.1.4) a poté inicializace, otevření a konfigurace sériového rozhraní (kapitola 5.3.1.6). O zpracování příkazů pro nebo od ZigBee koordinátoru se stará vytvořené vlákno sériového rozhraní (kapitola 5.3.1.3). Pokud by při otevírání nebo konfiguraci nastala chyba, program se ukončí a uživateli je na standardní chybový výstup vypsán typ chyby a doporučené řešení vedoucí k odstranění chyby. Nyní se inicializuje a konfiguruje nastavení síťového rozhraní serveru. V případě chyby je program ukončen a uživateli je na standardní chybový výstup vypsán typ chyby a doporučené řešení vedoucí k odstranění chyby.

Třída *Server* zajišťuje veškerou správu síťových připojení, kde nejdůležitější metodou je metoda *run()*, která spouští hlavní smyčku síťové komunikace. V této hlavní smyčce je zavolána funkce *select()*, kde ve čtecí množině file deskriptorů je vložen file deskriptor hlavního socketu serveru a file deskriptor standardního vstupu. Ve funkci *select()* se čeká do doby, než na některém file deskriptoru z čtecí množiny nastane událost. Pokud nastala událost, vstupuje se do části programu, která zjišťuje, jestli nastala událost na standardním vstupu nebo na hlavním socketu serveru. Je-li zjištěna událost na standardním vstupu, je předáno řízení metodě *processCommand()*, která zajistí zpracování a příslušný výpis příkazu uživatelského rozhraní (kapitola 5.3.1.8). Po zpracování se vrací řízení programu zpět na začátek hlavní smyčky. Je-li zjištěna událost na hlavním socketu serveru, provede se připojení nového klienta (kapitola 5.3.1.5) a vytvoření nového klientského vlákna, kterému je předáno řízení komunikace. Informace o připojené klientské aplikaci se do vlákna předají pomocí objektu *ClientHandler*.

Nyní se o komunikaci mezi klientem (vzdálenou aplikací) a senzorovou sítí starají vlákna připojených klientů, sdílené buffery a vlákno sériového rozhraní. Veškeré podrobnosti o ukončení serveru, odhlášení klientů a uzavření sériového rozhraní jsou uvedeny v příslušných podkapitolách kapitoly 5.3.1 Stavební kameny implementace serveru.

Příklad přeložení zdrojových kódů implementace serveru a spuštění programu je v příloze 1 Manuál.

5.4 Modelové situace

Pro předvedení funkčnosti a testování návrhu implementace aplikační brány (serverová část a ZigBee koordinátor) byly po domluvě s vedoucím diplomové práce zvoleny modelové situace namísto reálné senzorové sítě ZigBee. V těchto modelových situacích je simulováno chování senzorové sítě, které se odvíjí od modelové situace. Jsou simulovány odpovědi od senzorové sítě na příkazy z klientské aplikace. Dále jsou simulovány události, které se vyskytly v senzorové síti, jenž jsou zasílány klientským aplikacím. Byly zvoleny 3 modelové situace, kde by každá měla otestovat, jestli je návrh a implementace dostatečně pružná v přizpůsobení se specifické situaci a jestli aplikační brána reaguje korektně. V rámci testování byl vytvořen zátěžový test, jehož popis a výsledky jsou součástí této

kapitoly. Bližší informace o modelových situacích jsou uvedeny v kapitolách 5.4.1, 5.4.2, 5.4.3 a zátěžový test je představen v kapitole 5.4.4.

V modelových situacích nejsou při jejich popisu zobrazovány reálné APS rámce, ale příkazy pro senzorovou síť a její odpovědi jsou reprezentovány pro lepší srozumitelnost v textových řetězcích, které svým obsahem uvádějí, o jaký příkaz se jedná. Pro každou z modelových situací musí být program serverové části spuštěn s jiným parametrem, který je použit jako přepínač mezi modelovými situacemi. Příklad spuštění programu je uveden v manuálu k programu v příloze 1. V každé z modelových situací v rámci testovaných parametrů jsou tučně vyznačeny důležité funkce aplikační brány.

5.4.1 Modelová situace 1

5.4.1.1 Popis modelové situace

Firma má nainstalované ZigBee senzory, pomocí kterých lze ovládat světelné zdroje ve všech výrobních halách v areálu firmy. K jejich vzdálené ovládání se využívá aplikační brána. Ke klientským aplikacím pro vzdálenou správu senzorů mají přístup ředitel firmy, správce areálu a jeho zástupce. Správa senzorové sítě zahrnuje zapínání, vypínání a získávání statusu jednotlivých světelných zdrojů. Status umožňuje zjistit, zda je světelný zdroj ve stavu zapnuto nebo vypnuto.

K uživatelskému rozhraní serverové části aplikační brány má přístup hlavní IT technik, který poskytuje výpisy z databáze aplikační brány řediteli, jenž předem definuje, o které informace má zájem. Takový výpis může například obsahovat informace o začátcích pracovní doby v jednotlivých halách za předpokladu, že k práci v halách je zapotřebí zapnutí světelných zdrojů.

Parametry modelové situace

- Identifikační čísla světelných zdrojů jsou značeny *IDXX*, kde za *XX* je vloženo identifikační číslo světelného zdroje v rozsahu odpovídajícímu počtu světelných zdrojů
- Spuštěný ZigBee koordinátor
- Spuštěna serverová část aplikační brány, kde spouštěcí příkaz je ve tvaru:

```
./server server.cfg SIM1
```

- Klientská aplikace pro zasílání příkazů pro senzorovou síť
- **Příkazy pro senzorovou síť**
  - *COMM light on IDXX* – zapnutí světelného zdroje s identifikačním číslem *XX*
  - *COMM light off IDXX* – vypnutí světelného zdroje s identifikačním číslem *XX*
  - *COMM status IDXX* – získání statusu světelného zdroje s identifikačním číslem *XX*

- **Odpovědi senzorové sítě**
  - *COMM OK light on IDXX* – světelný zdroj s identifikačním číslem XX je ve stavu zapnuto
  - *COMM OK light off IDXX* – světelný zdroj s identifikačním číslem XX je ve stavu vypnuto

Testované parametry aplikační brány

- Autentizace uživatelů
- Řízení komunikace mezi vzdálenou aplikací a senzorovou sítí
- **Zpracování příkazů pro senzorovou síť**
- **Přiřazování odpovědí od senzorové sítě připojeným klientům**
- Analýza komunikace z informací ukládaných do databáze
- Chování sdílených bufferů

5.4.1.2 Zhodnocení výsledků a chování aplikační brány

Modelová situace testovala základní chování aplikační brány, kdy jsou na aplikační bránu z klientských aplikací zasílány příkazy pro senzorovou síť, která zasílá odpověď zpět přes aplikační bránu na klientskou aplikaci. Všechny příkazy přijaté na aplikační bránu a odeslané odpovědi senzorové sítě byly uloženy jako záznamy do databáze pro další analýzu, kterou umožňuje uživatelské rozhraní serverové části aplikační brány. Pokud uživatelé nebyli autentizováni, neměli povoleno zasílat či přijímat zprávy od/do senzorové sítě a byli na tuto skutečnost upozorněni. Sdílené buffery si uchovaly během modelové situace integritu i dostupnost pro všechna vlákna, která přes ně komunikovala.

Hlavním záměrem modelové situace bylo ověření, zda jsou příkazy pro senzorovou síť správně zpracovávány. Dalším podrobně zkoumaným chováním aplikační brány bylo správné přiřazování odpovědí od senzorové sítě klientům, jenž zaslali příkaz pro senzorovou síť. U obou podrobně zkoumaných hlavních testových parametrů modelové situace bylo chování vyhodnoceno jako korektní i například u záměrného ukončení klientské aplikace. Toto ukončení klientské aplikace mělo simulovat přerušení spojení či pád klientské aplikace. I při této události jsou data pro klienta držena v paměti serverové části aplikační brány. Jakmile se klient připojí a je autentizován, zprávy jsou mu zaslány.

5.4.2 Modelová situace 2

5.4.2.1 Popis modelové situace

Firma využívá ZigBee senzory ke kontrole místnosti, kde má uloženy cenné informace a dokumenty včetně finanční hotovosti. Senzory hlídají místnost a při událostech, které by mohli vést ke ztrátě či zničení cenností, upozorňuje šéfa ochranky, jenž pro správu senzorové sítě používá aplikační bránu, ke které je vzdáleně připojen přes klientskou aplikaci. Použité senzory se dělí na senzory snímající

pohyb, senzory umožňující ovládání bezpečnostních kamer, senzory snímající kouř a senzory sloužící k ovládání hasícího systému. Správa senzorové sítě zahrnuje zapnutí či vypnutí kamerového systému, senzorů monitorujících pohyb a spuštění hasícího systému. Dále umožňuje přijímat hlášení o detekci pohybu a kouře ve sledované místnosti.

Parametry modelové situace

- Identifikační čísla senzorů jsou značeny *IDXX*, kde za *XX* je vloženo identifikační číslo senzoru v následujících rozsazích
  - *ID01- ID03* – detektory pohybu
  - *ID04-ID06* – detektory kouře
  - *ID07-ID08* – kamery z monitorovacího systému
  - *ID09-ID11* – hasící přístroje z protipožárního systému
- Spuštěný ZigBee koordinátor.
- Spuštěna serverová část aplikační brány, kde spouštěcí příkaz je ve tvaru:

```
./server server.cfg SIM2
```

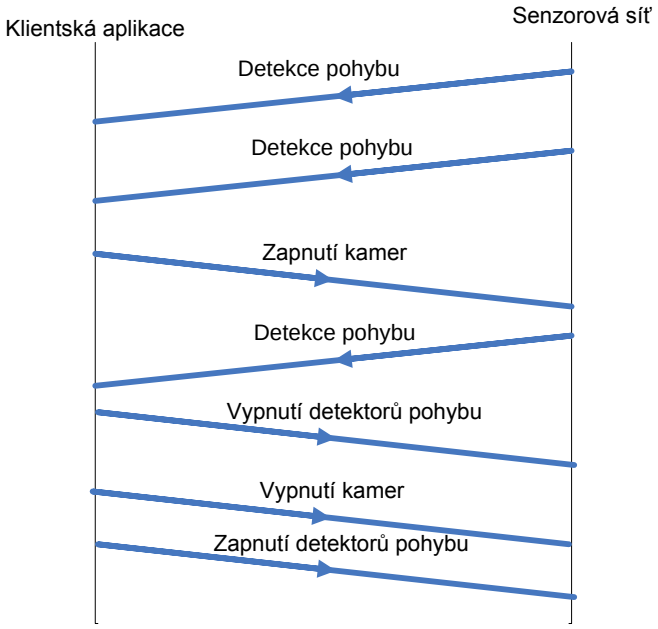
- Přístup k uživatelskému rozhraní serverové části
- Klientská aplikace pro zasílání příkazů pro senzorovou síť
- **Příkazy pro senzorovou síť**
  - *COMM motion on* – zapnutí detektorů pohybu
  - *COMM motion off* – vypnutí detektorů pohybu
  - *COMM cameras on* – zapnutí monitorovacího systému
  - *COMM cameras off* – vypnutí monitorovacího systému
  - *COMM fireprotectors on* – zapnutí protipožárního systému
  - *COMM fireprotectors off* – vypnutí protipožárního systému
- **Odpovědi senzorové sítě**
  - *COMM OK IDXX on* – senzor s identifikačním číslem XX je ve stavu zapnuto
  - *COMM OK IDXX off* – senzor s identifikačním číslem XX je ve stavu vypnuto
- **Příkazy asynchronních událostí v senzorové síti**
  - *COMM OK IDXX motion* – detekce pohybu na senzoru s identifikačním číslem XX
  - *COMM OK IDXX fire* – detekce kouře na senzoru s identifikačním číslem XX

Testované parametry aplikační brány

- Autentizace uživatelů
- Řízení komunikace mezi vzdálenou aplikací a senzorovou sítí
- **Reakce na asynchronní události vyvolané senzorovou sítí**
- Zpracování příkazů pro senzorovou síť

- Přiřazování odpovědí od senzorové sítě připojeným klientům
- Analýza komunikace z informací ukládaných do databáze
- Chování sdílených bufferů

Příklad chování senzorové sítě a komunikace je znázorněna na obrázku 44.



Obrázek 44 Příklad z běhu modelové situace 2

5.4.2.2 Zhodnocení výsledků a chování aplikační brány

V modelové situaci bylo testováno kromě základního chování i schopnost aplikační brány reagovat na událost v senzorové sítí, která nebyla vyvolána příkazem z klientské aplikace ale senzory, které mají například při detekci pohybu upozornit na tuto skutečnost vzdálenou aplikaci. Aplikační brána na tyto asynchronní události v senzorové síti reagovala korektně a zprávy od senzorů zasílala určenému uživateli na klientskou aplikaci. V praktickém řešení této modelové situace by muselo být zaručeno doručení informací o pohybech či požáru v monitorované místnosti i pomocí jiných prostředků, než jen pouhým zasláním upozornění na klientskou aplikaci. K aplikační bráně by bylo vhodné připojit alarm či SMS bránu.

5.4.3 Modelová situace 3

5.4.3.1 Popis modelové situace

Firma využívá senzory ZigBee pro ovládání elektronických zámků dveří od kanceláří nebo konferenčních místností, které se nacházejí v řídící budově firmy. Senzory zasílají informace o změně stavu zámku a umožňují měnit stav zámku. Ke vzdálené správě zámků se používá klientská aplikace, která komunikuje se senzorovou sítí přes aplikační bránu. Klientské aplikace využívají ředitel firmy, správce řídící budovy a jeho zástupce.

K uživatelskému rozhraní serverové části aplikační brány má přístup hlavní IT technik, který poskytuje výpisy z databáze aplikační brány řediteli, jenž předem definuje, o které informace má zájem. Takový výpis může například obsahovat informace o příchodech/odchodech zaměstnanců z/do práce nebo využití konferenčních místností.

Správa zahrnuje zamykání/odemykání a zjištění stavu jednotlivých zámků dveří, skupin dveřních zámků nebo uzamčení všech zámků v rámci řídící budovy. Dále jsou přijímány informace při změně stavu zámku. Počet adresovatelných skupin je 4, kde každá skupina má 5 zámků.

Parametry modelové situace

- Identifikační čísla zámků jsou značeny *DIDXX,* kde za *XX* je vloženo identifikační číslo zámku v rozsahu DID01-DID20.
- Skupiny zámků jsou označovány
  - *Skupina GID01* – rozsah DID01-DID05
  - *Skupina GID02* – rozsah DID06-DID10
  - *Skupina GID03* – rozsah DID11-DID15
  - *Skupina GID04* – rozsah DID16-DID20
- Pro adresování všech zámků je definována skupina *AID* s rozsahem DID01-DID20
- Spuštěný ZigBee koordinátor
- Spuštěna serverová část aplikační brány, kde spouštěcí příkaz je ve tvaru:

```
./server server.cfg SIM3
```

- Přístup k uživatelskému rozhraní serverové části
- Klientská aplikace pro zasílání příkazů pro senzorovou síť
- **Příkazy pro senzorovou síť**
  - *COMM lock on AID* – zamknutí všech zámků
  - *COMM lock off AID* – odemknutí všech zámků
  - *COMM lock on GIDXX*– zamknutí skupiny zámků s identifikačním číslem *XX*
  - *COMM lock off GIDXX* – odemknutí skupiny zámků s identifikačním číslem *XX*
  - *COMM lock on DIDXX* – zamknutí zámku s identifikačním číslem *XX*
  - *COMM lock off DIDXX* – odemknutí zámku s identifikačním číslem *XX*
  - *COMM status AID* – zjištění stavu všech zámků
  - *COMM status GIDXX* – zjištění stavu skupiny zámků s identifikačním číslem *XX*
  - *COMM status DIDXX* – zjištění stavu zámku s identifikačním číslem *XX*

- **Odpovědi senzorové sítě**
  - *COMM OK DIDXX locked* – zámek byl uzamknut

- *COMM OK DIDXX unlocked* – zámek byl odemknut

- **Příkazy asynchronních událostí v senzorové síti**
  - *COMM OK DIDXX locked* – zámek byl uzamknut
  - COMM OK DIDXX unlocked – zámek byl odemknut

Testované parametry aplikační brány

- Autentizace uživatelů.
- Řízení komunikace mezi vzdálenou aplikací a senzorovou sítí
- Reakce na asynchronní události vyvolané senzorovou sítí
- **Adresování skupin senzorů**
- **Adresování typu broadcast (zaslání příkazů všem senzorům)**
- Zpracování příkazů pro senzorovou síť
- Přiřazování odpovědí od senzorové sítě připojeným klientům
- Analýza komunikace z informací ukládaných do databáze
- Chování sdílených bufferů

5.4.3.2 Zhodnocení výsledků a chování aplikační brány

Hlavním záměrem modelové situace je zjistit chování aplikační brány při zasílání příkazů na skupiny senzorů a zaslání příkazů všem senzorům najednou (adresování typu broadcast). Chování aplikační brány bylo vyhodnoceno jako korektní, ale byl nalezen problém v části sdílených bufferů, kde při zasílání odpovědí od senzorové sítě na klientskou aplikaci u příkazů, které byly adresovány skupině senzorů nebo dokonce všem senzorům, bylo zjištěno, že sdílené buffery zasílají odpovědi podle toho, jak byly naplněny ZigBee koordinátorem. Což při větším počtu připojených klientských aplikací zasílajících výše zmíněné příkazy by mohlo způsobit, zdržování odpovědí pro klientské aplikace v bufferech.

Proto jsem se rozhodl, že modelové situace doplním zátěžovým testem, který bude zaměřen na zdržování odpovědí od senzorové sítě ve sdílených bufferech při větším počtu připojených klientů (kapitola 5.4.4). V zátěžovém testu je navrhnuto i řešení tohoto problému pomocí doplnění sdílených bufferů o řízení zisku dat z bufferů.

5.4.4 Zátěžový test aplikační brány

5.4.4.1 Popis zátěžového testu

K aplikační bráně je připojeno přes klientské aplikace 10 uživatelů. Uživatelé zasílají libovolné množství požadavků. Využívá se požadavků z modelové situace 1 (kapitola 5.4.1) k získání stavu světelného zdroje. Hlavním záměrem je zjistit, zda při větším zatížení se aplikační brána chová korektně. Sleduje se, zda odpovědi od senzorové sítě jsou zasílány těm klientům, kteří zaslali příkaz

pro senzorovou síť. Dále se zaměřuji na kritické místo, jímž jsou sdílené buffery, kde se sleduje správný přístup k uloženým datům z klientských vláken a vlákna sériového rozhraní.

Parametry modelové situace

- Identifikační čísla světelných zdrojů jsou značeny *IDXX*, kde za *XX* je vloženo identifikační číslo světelného zdroje v rozsahu odpovídajícímu počtu světelných zdrojů
- Spuštěný ZigBee koordinátor
- Spuštěna serverová část aplikační brány, kde spouštěcí příkaz je ve tvaru:

```
./server server.cfg TEST
```

- Klientská aplikace pro zasílání příkazů pro senzorovou síť
- **Příkazy pro senzorovou síť**
  - *COMM status IDXX* – získání statusu světelného zdroje s identifikačním číslem *XX*
- **Odpovědi senzorové sítě**
  - *COMM OK light on IDXX* – světelný zdroj s identifikačním číslem XX je ve stavu zapnuto
  - *COMM OK light off IDXX* – světelný zdroj s identifikačním číslem XX je ve stavu vypnuto

Testované parametry aplikační brány

- **Řízení komunikace mezi vzdálenou aplikací a senzorovou sítí**
- **Zpracování příkazů pro senzorovou síť**
- **Přiřazování odpovědí od senzorové sítě připojeným klientům**
- Analýza komunikace z informací ukládaných do databáze
- **Chování sdílených bufferů**

5.4.4.2 Zhodnocení výsledků a chování aplikační brány

Hlavním záměrem zátěžového testu bylo zjistit chování sdílených bufferů aplikační brány.Hlavně bylo sledováno zpoždění zasílání odpovědí ve sdílených bufferech, ze kterých jsou přiřazovány požadavky na zpracování ZigBee koordinátorem, a při zasílání příkazů více klienty najednou. Bylo zjištěno, že příkazy jsou z bufferů zpracovávány ZigBee kordinátorem sekvenčně (tak jak byly vloženy do vstupního sdíleného bufferu), čímž dochází ke zpoždění odpovědí při více současně přijatých příkazech od připojených uživatelů.

Řešením tohoto problému by bylo doplnění implementace sdílených bufferů o rozhodovací logiku, která by přidávala prioritu zpracování zpráv nebo by zajišťovala pokročilou techniku pro výběr dat ze sdílených bufferů, jakou je například algoritmus Round-robin.

6 Závěr

Cílem diplomové práce bylo navrhnout a implementovat aplikační bránu pro komunikaci mezi klientskou bránou umístěnou v sítí Internet a senzorovou sítí ZigBee. Aplikační brána umožňuje ovládání senzorů pomocí vzdálené aplikace, jež může být spuštěna na počítači kdekoliv na světě s přístupem k Internetu. Komunikace přes bránu je obousměrná, abychom byli schopni nejen ovládat specifické senzory, ale i přijímat zprávy ze senzorové sítě. Návrh aplikační brány obsahuje i návrh komunikační protokolu přes síť Internet.

V diplomové práci je představen popis senzorových sítí a jejich vlastností. Převážně se tento popis zaměřuje na senzorovou síť ZigBee (kapitola 3.2), kde byla představena architektura technologie ZigBee a protokol pro komunikaci v Zigbee sítích. Důležitou částí je popis podrobné struktury APS rámce (kapitola 3.3.1), který je využíván pro řízení senzorové sítě pomocí ZigBee koordinátoru. Návrh komunikačního protokolu (kapitola 4.1.1) na aplikační úrovni pro řízení komunikace mezi klientskou aplikací a serverovou částí aplikační brány byl implementován a zhodnocen jako velmi vhodný, jelikož zajišťuje univerzální použití pro jakýkoliv příkaz senzorové síě ZigBee nebo její odpověď. Implementace ZigBee koordinátoru využívá pro své řízení knihoven ZigBee stack (kapitola 5.2.2). Je doplněna o prostředky umožňující jednoduché přijímání či zasílání dat z/do senzorové sítě ZigBee. Implementace serverové části aplikační brány byla nejdříve představena na jednotlivých stavebních kamenech (kapitola 5.3.1) a poté byl popsán její celkový chod (kapitola 5.3.2).

Testování implementace a chování aplikační brány proběhlo na 3 modelových situacích a zátěžovém testu (kapitola 5.4). Každá modelová situace má za úkol otestovat různé události v senzorové síti, které se mohou při nasazení aplikační brány do praxe objevit. Výsledky testování jsou v každé z modelových situací analyzovány. Zátěžový test aplikační brány byl zaměřen na konkrétní problém, který se vyskytl při testování a bylo navrženo jeho řešení. Modelové situace a zátěžový test ukázaly, že návrh a implementace aplikační brány a komunikačního protokolu je velmi pružný a dostatečně univerzální, aby mohl být využit pro nasazení do praxe.

Jako rozšíření implementace aplikační brány bych se zaměřil na sdílené buffery, jenž jsou využívány jako sdílená paměť pro vlákna klientů a sériového rozhraní. Možným rozšířením je upravení bufferů na prioritní buffery, které umožňují přednostní zpracování příkazů od uživatelů s vyšší prioritou. Zde by ovšem musela být zajištěna možnost spravedlivého přidělování sdílených bufferů podle pokročilých algoritmů pro přístup do sdílené paměti tak, aby například nenastala situace, kde je tato sdílená paměť využívána jen uživatelem s nejvyšší prioritou. Dalším možným rozšířením sdílených bufferů je implementace správy bufferů přes uživatelské rozhraní serverové části aplikační brány nebo automatizace této správy. Příkladem této správy může být promazávání příkazů z bufferů, které jsou v bufferech udržovány déle než zvolený počet hodin. Pro využití v praxi

bych jako rozšíření aplikační brány jako celku doporučil připojení modulů zabezpečujících doručení informací při důležitých událostech přes SMS bránu, vyvolání zvukových alarmů a šifrovanou komunikaci přes síť Internet nebo v rámci senzorové sítě. Toto ovšem záleží na konkrétním příkladu nasazení aplikační brány do praxe.

Zdrojové kódy implementace serverové části aplikační brány mohou být díky objektovému návrhu využity pro další použití či rozšíření. Ke zdrojovým kódům je dodána podrobná programová dokumentace, která popisuje všechny třídy a jejich veřejná rozhraní. Při návrhu a implementaci jednotlivých tříd bylo bráno v potaz jejich další použití. V části ZigBee koordinátoru byl představen modul bufferu, který umožňuje efektivně přijímat či zasílat informace ze sériového rozhraní ZigBee koordinátoru.

Diplomová práce může sloužit jako návod pro nasazení celého konceptu aplikační brány do praxe. Pro analýzu chodu aplikační brány je možné využít uživatelské rozhraní s možností nadefinování si vlastních příkazů pro výpisy informací z databáze. Byly představeny i požadavky na klientskou aplikaci (kapitola 5.1.2.2), která umožňuje zasílání příkazů pro senzorovou síť ZigBee. Aplikační brána může být po malé úpravě konfiguračního souboru a implementace využita jako multibrána pro více ZigBee koordinátorů. Multibrána zajišťuje zasílání a přijímaní příkazů na/od více ZigBee koordinátorů, čímž můžeme zajistit větší přizpůsobení požadavkům na senzorovou síť.

Návrhová část diplomové práce byla představena a obhájena jako článek před komisí v soutěži *STUDENT EEICT 2010*. Článek byl přijat do sborníku a získal ocenění v podobě *1. místa* v sekci *M12 Inteligentní a počítačové systémy*. Sborník STUDENT EEICT 2010 i s příslušným článkem lze nalézt na URL: <http://www.feec.vutbr.cz/EEICT/2010/sbornik/index.html> v části Magisterské projekty, sekce Počítačové systémy.

Literatura

[1] Blaise Barney: POSIX Threads Programming, Lawrence Livermore National Laboratory UCRL-MI-133316. Dostupné na URL: <https://computing.llnl.gov/tutorials/pthreads/>

[2] Hall Brian: Beej's Guide to Network Programming - Using Internet Sockets. Dostupné na URL: <http://beej.us/guide/bgnet/>

[3] KDevelop: Integrated Development Enviroment. Dostupné na URL: <http://www.kdevelop.org/>

[4] Koubaa Anis, Alves Mário and Tovar Eduardo: IEEE 802.15.4 for Wireless Sensor Networks: A Technical Overview. HURRAY-TR-050702. Dostupné na URL: <http://www.hurray.isep.ipp.pt/docs/ieee+802.15.4+for+wireless+sensor+networks:+a+technical+overview/222/>

[5] Lattibeaudiere Derrick P.: Microchip ZigBee-2006 Residential stack protocol, Microchip Technology Inc., 2008 .
Dostupné na URL: <http://ww1.microchip.com/downloads/en/AppNotes/00965c.pdf>

[6] Microchip Technology Inc.: PICDEM™ Z Demonstration Kit User's Guide, DS51524C, 2008. Dostupné na URL: <http://ww1.microchip.com/downloads/en/AppNotes/PICDEM%20Z%20Users_Guide_51524C.pdf>

[7] Microchip Technology Inc.: MPLAB®ICD 2 In-Circuit Debugger/Programmer, DS51264B, 2003. Dostupné na URL: <http://ww1.microchip.com/downloads/en/DeviceDoc/51264B.pdf>

[8] Microchip Technology Inc.: MPLAB IDE - User guide. Dostupné na URL: <http://ww1.microchip.com/downloads/en/DeviceDoc/33014K.pdf>

[9] Murthy Ram Siva C., Manoj S. B.: Ad Hoc Wireless Networks Architectures and Protocols, Prentice Hall, 2004, ISBN 978-0-13-147023-1.

[10] Post Office Protocol verze 3, RFC 1939. Dostupné na URL: <http://www.ietf.org/rfc/rfc1939.txt>

[11] Putty. Dostupné na URL: < http://www.putty.org/>

[12] Sohraby Kazem, Minoli Daniel, Znati Taieb: Wireless Sensor Network - Technology, protocols and applications, Wiley-Interscience, 2007,ISBN: 978-0471743002. Zapůjčeno jako knižní vydání od vedoucího diplomové práce.

[13] SSL 3.0 specification. Dostupné na URL: <http://wp.netscape.com/eng/ssl3/>

[14] SQLite project: Description. Dostupné na URL: <http://www.sqlite.org/about.html>

[15] SQLite project: SQL syntax. Dostupné na URL: <http://www.sqlite.org/lang.html>

[16] Sqlite wrapped project. Dostupné na URL: <http://www.alhem.net/project/sqlite/index.html>

[17] Sweet Michael R.: Serial Programming Guide for POSIX Operating Systems. Dostupné na URL: <http://www.easysw.com/~mike/serial/serial.html>

[18] The GNU NetCat project. Dostupné na URL: < http://netcat.sourceforge.net/>

[19] The Open Group: The sys/socket UNIX ® Specification, Version 2. Dostupné na URL: <http://www.opengroup.org/onlinepubs/007908799/xns/syssocket.h.html>

[20] Trchalík Roman: Senzorové sítě - ZigBee. Dostupné na URL: <https://www.fit.vutbr.cz/study/courses/PDS/private/lecture/pds-pr13zigbee.pdf>

[21] ZigBee Standards Organization: ZigBee Specification 2006. Dostupné na URL: <http://www.medialab.ch/labor/cc2430/Z-Stack/054024r01ZB_AFG-ZigBee-Specification-2006-Download.pdf>

Seznam příloh

Příloha 1. Manuál

Příloha 2. Zdrojové texty vybraných částí implementace

Příloha 3. DVD

Příloha 4. Zdrojový text konfiguračního souboru server.cfg

Příloha 1. Manuál

Tato příloha slouží k návodu pro přeložení programu aplikační brány a jejího spuštění. Manuál k programu aplikační brány je rozdělen na dvě části – serverová část a ZigBee koordinátor.

ZigBee koordinátor

Pro práci s programem pro ZigBee koordinátor je potřeba mít nainstalovaný MPLAB IDE s překladačem jazyka C (kapitola 5.1.1.1), programátor MPLAB ICD 2 (kapitola 5.2.1.2) a samozřejmě základní desku PICDEM Z (dále jen základní deska), na které je mikrokontrolér PIC18LF4620 (kapitola 5.2.1.1). MPLAB umožňuje překlad programu, případné ladění programu a nahrání přeloženého kódu do hardware (PICDEM Z) plnící funkci ZigBee koordinátoru.

Pro nahrání přeloženého kódu potřebujete MPLAB ICD 2 programátor připojený k Vašemu počítači a zároveň k základní desce. Po nahrání programu se odpojí programátor MPLAB ICD 2 od základní desky. Program se spustí tím, že se na základní desce přesune přepínač pro zapnutí z polohy OFF do polohy ON nebo BATTERY, podle toho jak je základní deska napájena, a stisknutím tlačítka pro restart (kapitola 5.2.1.1).

Pro komunikaci se serverovou částí musí být základní deska propojena s počítačem pomocí sériového kabelu s rozhraním RS-232 nebo případně přes redukci USB -> RS-232.

Serverová část

Program serveru je implementován pro běh v operačním serveru Linux a se ZigBee koordinátorem je propojen přes sériové rozhraní viz výše.

Překlad programu

Pro překlad je nutné mít nainstalovaný překladač programovacího jazyka C++ a nástroj CMake (http://www.cmake.org/), který je určen pro sestavení samotného překladu. Minimální požadovaná verze CMake je 2.6. Při překladu se musíte nacházet v adresáři */Program/server/* a pro spuštění překladu stačí v konzoli napsat příkaz *make*. Pokud by při překladu nastala chyba vypíše se upozornění a důvod chyby.

Spuštění programu

Před samotným spuštění zkontrolujte zda je ZigBee koordinátor korektně připojen k počítači a je zapnut. Pro spuštění serverové části aplikační brány je potřeba mít konfigurační soubor pro nastavení serveru a komunikace se sériovým rozhraním (kapitola 5.3.1.1), jehož příklad je uveden v příloze 4.

Pro spuštění programu je nutné mu předat v jeho argumentech následující informace. V prvním argumentu programu se předává relativní adresářová cesta ke konfiguračnímu souboru i s jeho názvem a případnou příponou. Pokud se konfigurační soubor nachází ve stejném adresáři jako

spustitelný soubor programu, jenž byl vytvořen při překladu, tak stačí jen zadat jméno konfiguračního souboru s případnou příponou. Druhým argumentem je mód běhu programu serverové části aplikační brány. V tabulce 18 je uveden přehled těchto módů i s jejich významem.

Mód	Význam
NORM	Spuštění programu pro přímou komunikaci a ladění ZigBee koordinátoru
SIM1	Spuštění programu pro běh modelové simulace 1
SIM2	Spuštění programu pro běh modelové simulace 2
SIM3	Spuštění programu pro běh modelové simulace 3
TEST	Spuštění programu pro běh zátěžového testu

Tabulka 18 Přehled módů programu serverové části aplikační brány

Příklad spuštění serverové části aplikační brány:

```
./server server.cfg SIM1
```

Po spuštění programu jej lze ovládat přes uživatelské rozhraní, jehož příkazy slouží pro vypnutí serveru, správu databáze a výpisu z komunikace (kapitola 5.3.1.8).

Pro komunikaci s aplikační bránou jako klientskou aplikaci využijte program Netcat (kapitola 5.1.2). Při komunikaci s aplikační bránou musí být dodržen komunikační protokol (kapitola 4.1.1).

Spuštění programu NetCat:

```
./nc <IP_adresa|localhost> <port_serveru>
```

Nápověda programu

Nápověda programu se dělí na dvě části. První nápověda se tiskne na standardní výstup při špatně zadaných argumentech programu. Druhá nápověda slouží pro výpis předdefinovaných příkazů pro uživatelské rozhraní. Výpis se tiskne na standardní výstup a jsou v něm obsaženy veškeré důležité informace potřebné k ovládání uživatelského rozhraní.

Chybové hlášení programu

Chybové hlášení programu se tiskne na standardní chybový výstup a jeho účelem je upozornit správce serveru na vzniklou chybu. Ve většině případů je po výpisu chyby i nabídnuto řešení problému. Pokud klientská aplikace zašle příkaz komunikačního protokolu v chybném tvaru, server zasílá informaci o chybné syntaxi příkazu.

Příloha 2. Zdrojové texty vybraných částí implementace

Deklarace hlavičkového souboru modulu bufferu buffermodul.h

```
/**********************************************************************
 *   Autor: Tomas Majer, xmajer10                                     *
 *   xmajer10@stud.fit.vutbr.cz                                       *
 **********************************************************************
**/
#ifndef BUFFERMODUL_H
#define BUFFERMODUL_H

/**
 Struktura obsahující buffer s pomocnými proměnnými.
**/
typedef struct {
  byte *p_buffer;             //samotný buffer
  int size;                   //velikost bufferu
  int length;                 //počet bytů v bufferu
  byte *p_startPosition;      //ukazatel na začátek řetězce
  byte *p_finalPosition;      //ukazatel na konec řetězce
} s_BUFFER;

/**
 Funkční prototypy funkci modulu buffermodul.h
**/

/**
Funkce sloužící pro inicializaci struktury s_buffer. Alokace paměti pro
buffer. Nastaveni velikosti bufferu. Nastaveni pomocných ukazatelů na
začátek bufferu.
@param p_bufferStruct Ukazatel na strukturu, u které se provede
inicializace.
@param sizeOfBuffer Hodnota určující velikost bufferu.
@return V případě chybné alokace paměti vrací -1, jinak vrací 1.
**/
extern int initBuffer(s_BUFFER *p_bufferStruct,int sizeOfBuffer);
```

```
/**
Funkce sloužící pro vložení 1 byte dat do paměti bufferu.
@param p_bufferStruct Ukazatel na strukturu bufferu.
@param sizeOfBuffer Hodnota vkládaného byte.
@return V případě chybného vložení -1, jinak vrací 1.
**/
extern int addByteToBuffer(s_BUFFER *p_bufferStruct,byte input);

/**
Funkce sloužící pro vyčištění bufferu a uvodního nastaví ukazetelů.
@param p_bufferStruct Ukazatel na strukturu bufferu.
**/
extern void clearBuffer(s_BUFFER *p_bufferStruct);

/**
Funkce sloužící pro načtení dat z bufferu do pole bytů.
@param p_bufferStruct Ukazatel na strukturu bufferu.
@param output Ukazatel na pole bytů, které se bude plnit daty z bufferu.
@return V případě chybného čtení -1, jinak vrací 1.
**/
extern int readFromBuffer(s_BUFFER *p_bufferStruct,byte *output);

/**
Funkce sloužící k vytištění obsahu bufferu a velikosti dat v bufferu.
@param p_bufferStruct Ukazatel na strukturu bufferu.
**/
extern void printBufferToOutput(s_BUFFER *p_bufferStruct);

/**
Funkce sloužící k vytištění nastavení pomocných ukazatelů na začátek a
konec dat v bufferu.
@param p_go Ukazatel na strukturu bufferu.
**/
void setPointersToSizeOfData(s_BUFFER * p_go);
```

Příloha 3. DVD

Tato příloha obsahuje diplomovou práci, soubory obsahující zdrojové kódy jednotlivých částí programu, programovou dokumentaci serverové části aplikační brány, než byla vygenerována pomocí nástroje Doxygen, soubory umožňující přeložení programu. Dále jsou zde všechny obrázky a zdrojové soubory diagramů použitých v diplomové práci.

Adresářová struktura DVD

- */Diplomova_prace.pdf*
  Text diplomové práce uložená v PDF formátu.
- */Program/*
  Adresář obsahující podadresáře se zdrojovými kódy jednotlivých částí aplikační brány a se soubory určené k překladu programů.
- */Program/server/*
  Adresář obsahující zdrojové kódy implementace serverové části aplikační brány a soubory určené k překladu serverové části.
- */Program/server/contrib/*
  Adresář obsahující zdrojové kódy k rozhraní SQLite a SQLite wrapper.
- */Program/coordinator/*
  Adresář obsahující podadresáře se zdrojovými kódy implementace ZigBee koordinátoru.
- */Program/coordinator/PIC18Coordinator/*
  Adresář obsahující zdrojové kódy s implementací ZigBee koordinátoru, modulu bufferu a projekt pro MPLAB IDE sloužící k naprogramování ZigBee koordinátoru.
- */Program/coordinator/Microchip/*
  Adresář obsahující zdrojové kódy implementace ZigBee stacku.
- */Dokumentace/*
  Adresář obsahující programovou dokumentaci. Vstupním bodem do dokumentace je index.html.
- */Podklady/*
  Adresář obsahující podadresáře se všemi obrázky nebo diagramy použitých v diplomové práci.
- */Podklady/literatura_obrázky*
  Adresář obsahující obrázky, které byly převzaty z literatury a použity v diplomové práci.
- */Podklady/obrázky*
  Adresář obsahující mé vlastní obrázky, které byly použity v diplomové práci.
- */Podklady/visio/*
  Adresář obsahující zdrojové soubory všech diagramů použitých v diplomové práci ve formátu .vsd (MS Visio).

Příloha 4. Zdrojový text konfiguračního souboru server.cfg

```
#+------------------------------------------------------------+
# Tomáš Majer, xmajer10, diplomová práce
# Konfigurační soubor
#+------------------------------------------------------------+
#+------------------------------------------------------------+
# Nastavení síťové části serveru
#+------------------------------------------------------------+
# Definice síťové rozhraní, na kterém bude server naslouchat.
# Hodnoty musí odpovídat reálným parametrům.
# Možné hodnoty: řetězce odděleny mezerami.
# Výchozí hodnota:   lo eth0
interface lo eth0

# Definice čísla portu, na kterém bude server naslouchat.
# Doporučené číslo portu: 1024-65535
# Možné hodnoty: 1 - 65535
# Výchozí hodnota:   12001
port 12001

# Časový interval, po kterém budou kontrolovány sdílené buffery na
asynchronní události.
# Doporučené hodnoty: 1-10 (v sekundách)
# Možné hodnoty: 1 - velikost datového typu int (závislé na platformě)
# Výchozí hodnota:   1
timeout 1

# Velikost bufferů pro příjem zpráv od klientů (v bajtech).
# Možné hodnoty:: 1 - velikost datového typu int (závislé na platformě)
# Výchozí hodnota:   256
buffer_size 256

#+------------------------------------------------------------+
# Nastavení sériového portu
#+------------------------------------------------------------+
# Specifikace umístění zařízení komunikující pomocí sériového portu v
struktuře operačního systému Linux.
```

```
# Možné hodnoty: řetězec popisující umístění
# Výchozí hodnota:   /dev/ttyS0
device /dev/ttyUSB0

# Definice rychlosti přenosu na sériovém rozhraní.
# Možné hodnoty: 75, 110, 134, 150, 200, 300, 600, 120, 1800, 2400, 4800,
#                  9600, 19200, 38400, 57600, 115200
# Výchozí hodnota:   9600
speed 9600

# Definice počtu datových bitů pro přenos na sériovém rozhraní.
# Možné hodnoty: 5, 6, 7, 8
# Výchozí hodnota:   8
data_bits 8

# Definice typu parity použitém při přenosu na sériovém rozhraní.
# Možné hodnoty: none, even, odd
# Výchozí hodnota:   none
parity none

# Definice počtu stop bitů použitém při přenosu na sériovém rozhraní.
# Možné hodnoty: 1, 2
# Výchozí hodnota:   1
stop_bits 1

# Definice typu řízení toku při přenosu na sériovém rozhraní.
# Možné hodnoty: hardware, xon/off, none
# Výchozí hodnota:   none
flow_control none
#+------------------------------------------------------------+
```