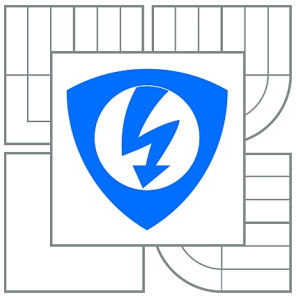
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

SBĚR DAT Z WEBOVÝCH SERVERŮ V PROSTŘEDÍ BEZDRÁTOVÉ SENZOROVÉ SÍTĚ

WEB-BASED DATA COLLECTION IN WIRELESS SENSOR NETWORKS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

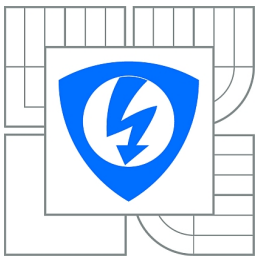
AUTOR PRÁCE — Bc. PETR VELKÝ

AUTHOR

VEDOUCÍ PRÁCE — Ing. VLADIMÍR ČERVENKA

SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Bc. Petr Velký ***ID:*** 102410
Ročník: 2 ***Akademický rok:*** 2010/2011

NÁZEV TÉMATU:

Sběr dat z webových serverů v prostředí bezdrátové senzorové sítě

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je prostudovat problematiku REST architektury v prostředí bezdrátové senzorové sítě. Seznámit se s objektově relačním mapováním dat. Navrhnout a implementovat serverové řešení pro sběr dat a řízení uzlů bezdrátové senzorové sítě v objektovém jazyce. Dále vytvořit klientskou aplikaci REST serveru na uzlech senzorové sítě. Získané informace pak budou uloženy v relační SQL databázi.

DOPORUČENÁ LITERATURA:

[1] DUNKELS, Adam; VASSEUR, Jean-Philippe. Interconnecting Smart Objects with IP : The Next Internet . California : Morgan Kaufmann, 2010. 432 s. ISBN 0123751659.

[2] The Contiki Operating System [online]. 2010-09-06 [cit. 2010-10-11]. Dostupné z WWW: <http://www.sics.se/~adam/contiki/docs/main.html>.

Termín zadání: 7.2.2011 ***Termín odevzdání:*** 26.5.2011

Vedoucí práce: Ing. Vladimír Červenka

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

ABSTRAKT

Tato práce se zabývá realizací sběru dat z webových serverů spuštěných na uzlech senzorové sítě. Zaměřuje se na senzorovou síť založenou na standardech IEEE 802.15.4 a 6LoWPAN. Samotná práce je rozdělena na stranu uzlu a stranu serveru. Pomocí REST webové služby na webovém serveru uzlu se poskytují data ze senzorů. Tyto data jsou sbírány JAVA aplikací spuštěnou na serveru a následně jsou ukládány do databáze.

KLÍČOVÁ SLOVA

server, senzorová síť, 6LoWPAN, webový server, REST, Contiki, AVR Raven

ABSTRACT

This thesis deals with the feasibility collecting data from web servers running on the nodes in sensor networks. It focuses on sensor network based on IEEE 802.15.4 and 6LoWPAN. The work itself is divided to the node side and server side.REST web services on the node's Web server provides sensor data. These data are collected with Java application running on the sensor network server. This servers stores collected data in a database.

KEYWORDS

server, sensor network, 6LoWPAN, web server, REST, Contiki, AVR Raven

VELKÝ, Petr *Sběr dat z webových serverů v prostředí bezdrátové senzorové sítě*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2011. 62 s. Vedoucí práce byl Ing. Vladimír Červenka

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Sběr dat z webových serverů v prostředí bezdrátové senzorové sítě“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Tato práce se zabývá řešením sběru dat z webových serverů uzlů bezdrátové senzorové sítě. Úvodní část práce se věnuje bezdrátovým senzorovým sítím a nejnovějším standardům, které v tomto oboru začínají nabírat na významu. Zejména standardům IEEE 802.15.4 a 6LoWPAN.

V druhé části je představeno prostředí bezdrátové senzorové sítě, ve kterém byla práce realizována a podrobněji je rozebráno jak se dá vyměňovat data pomocí webových serverů. Konkrétně je probrána problematika webových služeb typu REST a SOAP.

V kapitole 3 a 4 je podrobněji popsáno jak byla realizována celá práce. V 3. kapitole je popsána strana uzlu bezdrátové senzorové sítě, pro který byl implementován program realizující REST webové služby, které zpřístupňují data z uzlu. Ve 4. kapitole je práce popsána z druhé strany, tedy z pohledu serveru, který z REST webových služeb uzlů sbírá data a ukládá je do databáze. V závěru kapitoly je pak popsáno i datové úložiště serveru.

V poslední kapitole jsou shrnuty výsledky práce. Dále jsou demonstrovány možnosti firmware pro uzel AVR Raven a možnosti serverové aplikace a výsledky realizovaných sběrů dat z webových serverů na uzlech bezdrátové senzorové sítě.

1 BEZDRÁTOVÉ SENZOROVÉ SÍTĚ

Bezdrátové senzorové sítě, dále jen WSN (Wireless Sensor Network), mají za úkol monitorovat předem definované parametry (např. teplota, průtok řeky, lokace a další) a tyto data pak zpřístupnit nebo doručit k uživateli (serveru) pro následné zpracování. Rozsahem patří senzorové sítě do skupiny PAN (Personal Area Network) respektive WPAN (Wireless PAN). Do stejné skupiny patří např. technologie Bluetooth a její rozsah je několik desítek metrů. Díky rozvoji WSN v poslední době mohou uzly ve WSN komunikovat v IPv6 síti upravené pro nízko-energetická (low-power) zařízení. To otevírá nové možnosti a zařízením v takové sítí se začalo říkat smart objects (chytré objekty). Protože taková zařízení mohou komunikovat přes IP protokol je snadné je propojit s dalšími sítěmi nebo dokonce s Internetem. Tuto problematiku popisuje.

1.1 IEEE 802.15.4

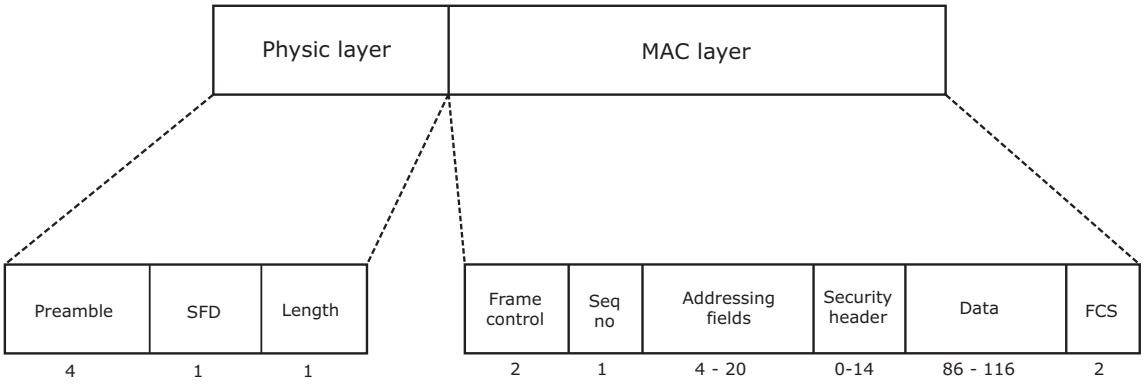
IEEE 802.15.4 je standard pro nízkoenergetické bezdrátové sítě s malými datovými přenosy . Maximální datový přenos je 250 kb/s a výstupní výkon 1mW [1]. Zařízení založena na tomto standardu mají dosah pár desítek metrů a vystačí s levnými a jednoduchými radio přijímači, proto je tento standard oblíbený.

IEEE 802.15.4 specifikuje následující vrstvy:

- Fyzická vrstva - určuje jak jsou přijmuty/vysílány zprávy přes (bezdrátové) médium
- MAC vrstva (MAC - Media Access Control) - definuje jak zpracovat zprávy přicházející z fyzické vrstvy

Ačkoliv standard specifikuje mechanizmy ve fyzické a MAC vrstvě, často nejsou využívané všechny části specifikace a jsou definované až na vyšší vrstvě. Například WirelessHART standard využívá specifikaci fyzické vrstvy a formát hlavičky paketu MAC vrstvy. Nevyužívá ale kompletní chování MAC, místo toho si přidává vlastní logiku nad MAC formátem.

Maximální velikost paketu je 127B [1]. Na obrázku 1.1 je obsah hlavičky fyzické a MAC vrstvy [1]. MAC vrstva vkládá hlavičku do každého paketu, takže pro data zbude prostor o velikosti mezi 86 a 116B. Proto protokoly na vyšší vrstvě často poskytují mechanizmy pro fragmentaci větších datových částí.



Obr. 1.1: Formát hlaviček fyzické a MAC vrstvy.

Skladba hlaviček fyzické vrstvy (Physic layer):

- Preamble - Záhlaví
- SFD - Start of frame delimiter - indikuje konec záhlaví
- Length - velikost paketu (maximálně 127B)

Skladba hlaviček MAC vrstvy (MAC layer):

- Frame control - obsahuje příznaky, které příjemci řeknou jak má rámec zpracovat
- Seq no - sekvenční číslo
- Addressing fields - obsahuje adresu odesílatele a příjemce rámec a identifikátory odesílající a přijímající PAN
- Security header - obsahuje data pro bezpečné zpracování paketu např. MIC (Message integrity Check) pro kontrolu kryptografické integrity
- Data
- FCS - Zápatí

IEEE 802.15.4 je implementován kombinací hardware a software. Nižší části - fyzická vrstva a část MAC vrstvy je implementována do hardware, vyšší části jako logika MAC vrstvy do software. Existuje několik implementací standardu [1].

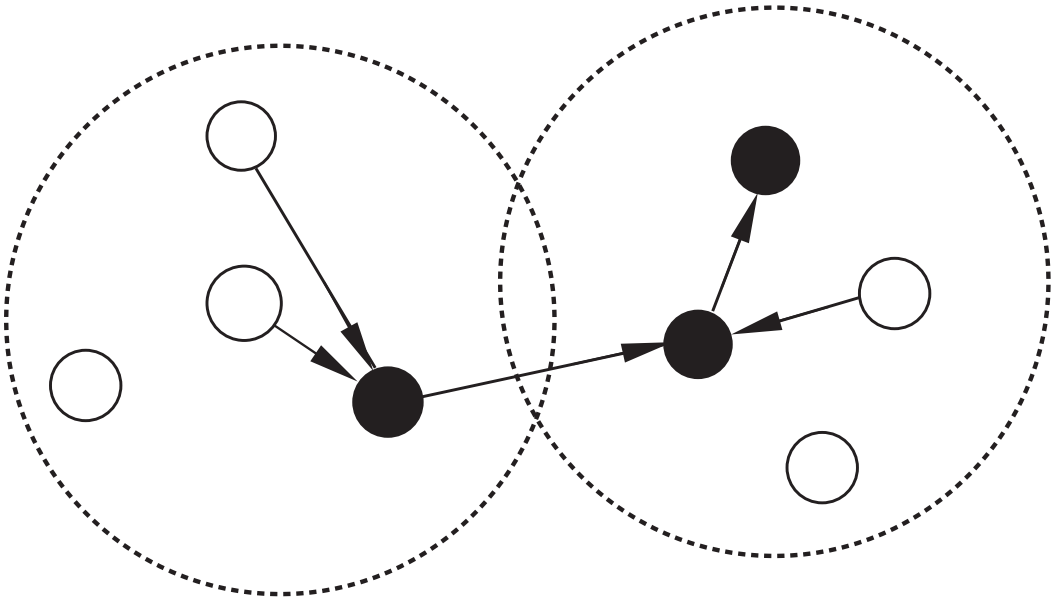
Sítě IEEE 802.15.4 jsou rozděleny do více PAN. Každá PAN má svůj PAN koordinátor a několik PAN členů. Pakety, posílané přes PAN obsahují 16 b identifikátor, který určuje jeho cílovou PAN. Jedno zařízení může fungovat jako koordinátor jedné PAN a jako člen druhé PAN.

Zařízení připojená do sítě IEEE 802.15.4 mohou být dvojího typu:

- Plně funkční - FFD (Fully Functional Device)
- Se sníženou funkčností - RFD (Reduced Functional Device)

FFD mohou komunikovat s FFD i RFD a mouhou zastávat funkci PAN koordinátora. RFD jsou mnohem jednodušší zařízení a mohou komunikovat pouze s FFD. Jednoduchý příklad dvou PAN síti je na obrázku 1.1

IEEE 802.15.4 specifikuje tři typy síťové topologie - hvězdicová, polygonální a shlukový strom - většina protokolů fungujících nad IEEE 802.15.4 však tyto topologie nepoužívá a vytváří si vlastní.



Obr. 1.2: Příklad dvou PAN sítí. Bílé kruhy jsou zařízení typu RFD, černé FFD.

Díky všudypřitomnosti IEEE 802.15.4 a dostupnosti kompatibilních radio přijímačů byla vyvinuta řada nízkoenergetických specifikací a standardů postavena právě na tomto standardu (např. 6LoWPAN a ZigBee).

1.2 6LoWPAN

6LoWPAN (IPv6 over Low power Wireless Personal Area Networks), čili IPv6 pro nízkoenergetické bezdrátové sítě, je standard umožňující společně použít IEEE 802.15.4 a IP [2]. Protože IPv6 podporuje linky s maximální přenosovou jednotkou (MTU - Maximum Transmission Unit) 1280B a linky IEEE 802.15.4 mají pouze MTU 127B, bylo zapotřebí specifikovat adaptační vrstvu pod IP, která se bude starat o fragmentaci a sjednocení paketů.

Následující výpočet ukazuje, proč a co konkrétně musí řešit adaptační vrstva 6LoWPAN. MTU IEEE 802.15.4 rámce je 127B od toho odečteme sadu polí protokolu [2]:

- Maximalní velikost režie MAC rámce: Řízení rámce (2B) + sekvenční číslo (1B) + adresní pole (20B) + FCS (2B) = 25B
- Bezpečnostní hlavička MAC: 21B (AES-CCM-128), 13B (AES-CCM-64) a 9B (AES-CCM-32)

V nejhorším případě tedy zbyde 81B (127B - 25B - 21B = 81B) pro data IPv6 paketů. Hlavička IPv6 má velikost 41B, zbývá 40B [2]. Dále je potřeba odečíst hlavičku transportního protokolu (8B pro UDP a 20B pro TCP), takže nakonec zbude pro aplikační vrstvu pouze 21B, což je velmi málo.

Adaptační vrstva 6LoWPAN poskytuje tři hlavní funkce:

- Fragmentaci a sjednocení paketu
- Komprese hlavičky
- Přesměrování na linkové vrstvě při použití multi-hop

Ve většině případů použití efektivních kompresních technik umožňuje většině aplikacím poslat data v jednom IPv6 paketu, navíc v bezdrátových senzorových sítích jsou aplikace navrženy tak aby posílali co nejmenší objemy dat.

2 KOMUNIKACE V BEZDRÁTOVÉ SENZOROVÉ SÍTI

V této kapitole je představeno základní schéma a popis senzorové sítě, pro kterou bylo navhrnuto a realizováno řešení sběru dat. Dále je na příkladech popsáno, jakým způsobem se dá řešit komunikace mezi uzly sítě a serverem. Další část práce upřesňuje vlastní realizaci sběru dat a jejich následné zpracování.

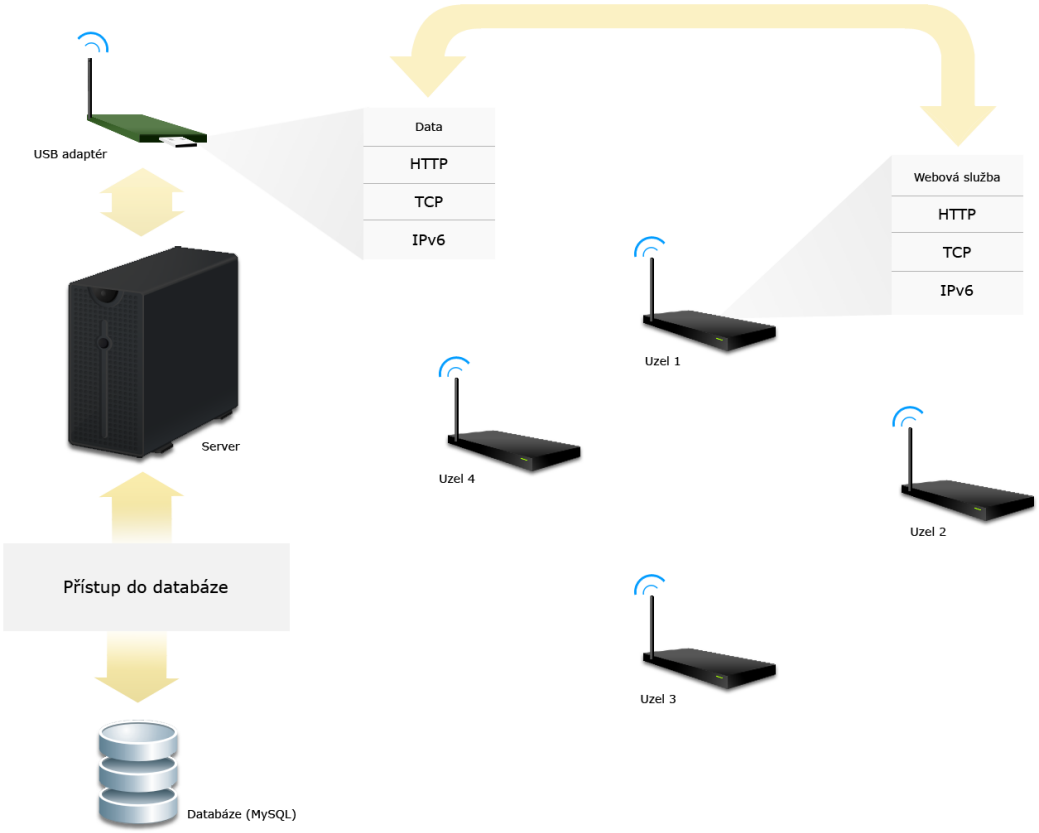
2.1 Prostředí bezdrátové senzorové sítě

Základem senzorové sítě je standard 6LoWPAN popisovaný v předchozí kapitole. Na rozdíl od neméně populárního Zigbee podporuje IP protokol verze 6. Z tohoto hlediska je 6LoWPAN do budoucna perspektivnější a i proto je síť na tomto standardu založena. Na úvod je třeba zmínit, že řešení není cíleno pouze na jednu platformu. Zdrojové kódy lze po úpravě použít i na jiné uzly od jiného výrobce. Takto může vzniknout větší nebo několik menších senzorových sítí s různými uzly, které snímají různé jevy a sdílejí společnou databázi.

Základní prvky senzorové sítě:

- Uzel - má radio přijímač a senzory
- Server - pomocí aplikace sbírá data v senzorové síti
- USB adaptér - brána do senzorové sítě - připojen k serveru
- Databáze - uchovává informace o celé síti a o naměřených hodnotách

Na obrázku 2.1 je vidět struktura senzorové sítě. Komunikace mezi serverem a uzlem je znázorněna ve 4 vrstvách. Nejnižší IPv6 dále TCP a HTTP, na 4. "imaginární" vrstvě je naznačeno, že data se budou získávat z webové služby uzlu. Uzly jsou typu Atmel Raven, které podporují 802.15.4 i 6LoWPAN, a které popisuji dále v kapitole 3.1. Pod serverem si lze představit jakékoli PC, notebook nebo speciální zabudované zařízení podporující JVM (Java Virtual Machine) a s nainstalovanou aplikací WSNSCC (WSN Server Control Centre), která řídí sběr dat v senzorové síti. Databázový server nemusí být nezbytně umístěn na této stanici, může být na jiné ale současně k ní musí být připojena buď pomocí internetu nebo lokální sítě. Klíčový je pro server Atmel USB Stick plnící funkci brány do senzorové sítě.



Obr. 2.1: Základní schéma senzorové sítě včetně serveru.

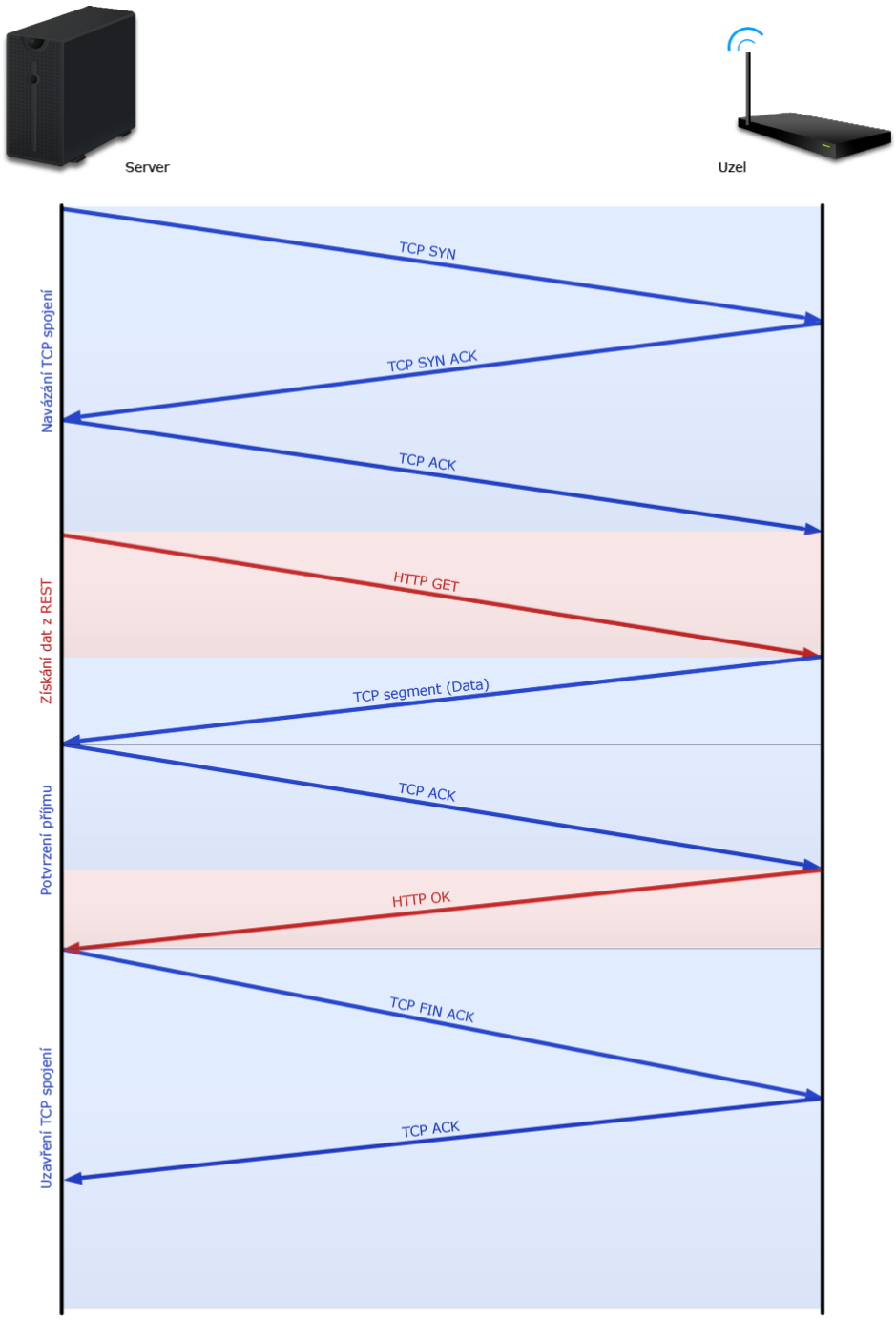
2.2 Komunikace mezi serverem a uzly

Zadání specifikuje použití webových serverů, takže komunikace byla navržena prostřednictvím HTTP protokolu a tím pádem také pomocí TCP protokolu transportní vrstvy. Na uzlu byl pomocí Contiki vytvořen webový server pro komunikaci se serverem. Pro stranu serveru byla vytvořena aplikace v jazyce JAVA. Iniciátor komunikace je tedy server a uzel na jeho požadavky reaguje a odesílá data. Při všem je využit IP protokol verze 6, díky jeho podpoře operačním systémem Contiki.

Kroky komunikace mezi serverem a uzlem:

- Navázání TCP spojení
- Požadavek o data (HTTP GET)
- Odeslání dat z REST
- Potvrzení přijmu dat
- Ukončení TCP spojení

Průběh komunikace je podrobněji naznačen na obrázku 2.2. V následující části bu-



Obr. 2.2: Průběh komunikace při získání dat z REST webserveru umístěném na uzlu. Zaznamenáno ve Wiresharku.

dou popsány výhody použití TCP v porovnání s UDP a možnosti zabezpečení komunikace.

UDP protokol je nespojový a nespolehlivý. V případě jeho použití by byla minimalizována režie a tím pádem i menší energetická spotřeba uzlů. U senzorových sítí však může často docházet ke ztrátám paketů vlivem ztráty signálu. Proto by využití tohoto protokolu bylo vhodné spíše pro sběr dat v reálném čase, kde nedoručení některých dat tolik nevadí. Takto realizovaný sběr dat by však nemohl využívat HTTP protokol, protože nad UDP protokolem jsou definovány jiné protokoly aplikační vrstvy např. RTP.

Protokol TCP, na úkor větší spotřeby energie uzlů kvůli potvrzování a spojově orientovanému přenosu, zajistí spolehlivost komunikace. V aplikaci, která byla vytvořena pro tuto práci neprobíhá sběr dat v reálném čase ale data se sbírají z jednotlivých uzlů pravidelně v časových intervalech. I proto je vhodnější použití TCP pro spolehlivost i za cenu větší spotřeby. HTTP protokol na aplikační vrstvě nabízí řadu technologií, pomocí kterých je možno implementovat komunikaci mezi uzly a serverem viz další podkapitoly.

Zabezpečit komunikaci v senzorové síti by mohl na TCP protokolu definovaný zabezpečovací protokol SSH. Pro využití v senzorové síti se však nehodí. Šifrování a podepisování odesílaných dat na straně uzlu by zatěžovalo procesor a tím pádem by šla i spotřeba energie nahoru. Dalším způsobem jak dosáhnout alespoň základního stupně zabezpečení by mohlo být zapsání do každého uzlu jedinečné přístupové heslo, jehož hash (otisk) by se na uzel posílal společně s požadavkem o data. Další možnost jak zabezpečit komunikaci je možnost IEEE 802.15.4 šifrovat data algoritmem AES128 [1]. V Contiki k tomu slouží funkce v hlavičkovém souboru cc2420-aes.h, který je součástí operačního systému [3]. Aby bylo možné data šifrovat musí však být uzel vybaven radio vysílačem CC2420.

2.2.1 Webové služby

Webové služby, dále jen WS, poskytují mechanizmy, pomocí kterých je možno získat datové zdroje z jiného zařízení, než na kterém jsou umístěny. Tyto služby mohou být implementovány na jakémkoli inteligentním zařízení, které je připojeno do IP sítě a podporuje funkci webového serveru. Uzly s WS pak mohou přes síť poskytovat data ze svých senzorů aplikaci běžící na serveru.

Mimo jiné existují dva typy WS:

- Založené na SOAP
- Založené na REST architektuře

2.2.2 SOAP (Simple Object Access Protocol)

Tento protokol původně určený pro webové aplikace je organizací W3C [6] doporučený od verze 1.2. Slouží pro výměnu informací v decentralizovaném, distribuovaném prostředí. SOAP definuje operace, pomocí kterých se přistupuje k datům a definuje v jakém tvaru se odešlou odpovědi. Je založen na XML zprávách, které se skládají ze tří částí:

- Envelope - definuje framework, tedy co je ve zprávě a jak jí zpracovat
- Header - sada pravidel pro vyjádření aplikací definovaných datových typů
- Body - konvence reprezentující vzdáleně volané procedury a odpovědi

XML zprávy jsou relativně velké a musejí se parsovat. Což je náročné na výpočetní výkon i na velikost operační paměti. Výhoda těchto zpráv je, že jsou přehledné a snadno čitelné pro člověka.

Často se na tzv. SOAP serveru definuje WSDL [6]. Tento soubor obsahuje seznam, všech operací, které lze na serveru provést, a také definici vlastních datových struktur. Výhoda je, že funkce, na které WSDL odkazuje jsou nezávislé na jazyku. Mohou tak být implementované v čemkoli, záleží jen na tvaru dat na výstupu těchto funkcí. SOAP kromě standardních datových typů (int, double, string, array) umožňuje definovat i vlastní typy.

Ačkoliv existují projekty, které se snaží uplatnit protokol SOAP i v senzorových sítích (např. RomXOAP [9]), ve srovnání s architekturou REST se hodí spíše pro webové servery nebo pro komunikaci s inteligentnímy zabudovanými zařízeními se stabilním zdrojem napájení. Studií těchto dvou webových služeb v senzorové síti se zabývá zpráva [12]. Tato zpráva hodnotí energetickou náročnost a rychlost obou typů webových služeb. Jejím závěrem je, že REST je rychlejší a méně náročný na spotřebu. Implementace SOAP serveru zabírá více paměti a navíc potřebuje vlastní parser. Proto je zde SOAP uveden pouze pro srovnání.

2.2.3 REST (Representational State Transfer)

REST se dá nazvat architekturou, je to styl implementace nebo způsob přístupu k datům [10]. Stejně jako předchozí typ je i tato architektura založena na principu klient - server. Klient si vyžádá konkrétní zdroj a server mu vrátí požadovaná data.

Na serveru jsou tedy definovány zdroje, které se dají adresovat. V adrese (např. `http://88de:c121:ff1::45d2:1/weather` se mohou předávat proměnné a tím přesněji specifikovat požadovaný zdroj. REST server na těchto adresách respektive URI (Uniform Resource Identifier - jednotný identifikátor zdroje) tak poskytuje data (datové objekty), které mohou být reprezentovány v libovolném tvaru. REST tedy není

závislý na žádném datovém formátu.

Datový objekt, který může být brán odkudkoli (např. z databáze nebo ze senzoru), je zdroj a REST definuje jak se k němu dá přistupovat. Nejčastěji je používán společně s HTTP, ale může být použit i s jiným protokolem aplikační vrstvy. K přístupu k těmto objektům je možné použít čtyři operace HTTP [8]:

- GET - požadavek na daný objekt
- POST - odesílá data na server, následně server zobrazí objekt jako u GET
- PUT - nahrává data do souboru na serveru, je potřeba oprávnění
- DELETE - smaže požadovaný objekt, je potřeba oprávnění

REST tedy díky tomu, že funguje na HTTP protokolu může data nejen posílat ale také přijímat. Konkrétně k tomu lze využít HTTP požadavky typu POST a PUT. Operace PUT a DELETE v dnešní době nahradily protokoly SCP a FTP [8], takže pro interakcii s REST službou zbývá POST a GET.

V porovnání se SOAP není tak objemný, není závislý na XML. Nevyžaduje vlastní protokol a proto může klient snadněji volat WS založenou na REST. Narozdíl od SOAP může REST zdroje zobrazovat z cache což je při použití v senzorové síti velká výhoda [12]. Pokud v daném okamžiku není nutné získat aktuální hodnotu ze senzoru, může se žískat z cache a tím se ušetří energie za inicializaci senzoru a načtení jeho hodnoty.

Pokud se nějaký systém řídí REST principy, hovoří se o tzv. RESTful systému. REST je osvobozen od XML a záleží pouze na tvůrci aplikace v jakém tvaru budou data odeslána. Často se používá ve spojení s datovým formátem JSON, který se díky jeho malé velikosti hodí i pro senzorovou síť.

JSON (JavaScript Object Notation)

JSON je platformě nezávislý, odlehčený datový formát, určený pro výměnu dat. Pomocí jej můžeme zapsat libovolné datové typy, pole i složité datové struktury jako řetězec.

Jeho formátovací pravidla, vycházejí z Javascriptu a definuje ho dokument RFC 4627. Používá čtyři základní datové typy (řetězce, čísla, booleovské hodnoty a null) a dva strukturově orientované datové typy - pole a objekt.

Tento datový formát je přehledný, umožňuje zapsat jakkoli složitou datovou strukturu a má jednoduché parsovací pravidla. Příklad takové struktury je uveden v ukázce 2.1

Z příhlednutím na vlastnosti obou typů webových služeb, byla pro implementaci zvolena webová služba REST, díky její nezávislosti na XML a menší energetické

Zdroj. kód 2.1: Ukázka JSON objektu

```
{
    "Node":
    {
        "sensor":
        {
          "name": "Temperature",
          "value": 22.3
        },
        "location":
        {
          "lattitude": 49.5219,
          "longitude": 43.3112
      }
    }
}
```

náročnosti. Zároveň jako prostředek pro komunikaci byl zvolen JSON. Je s touto architekturou často propojen stejně tak jako s JAVA aplikacemi.

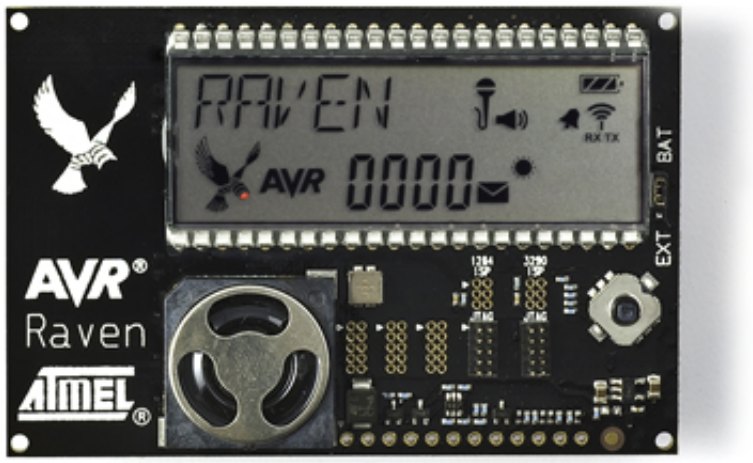
3 UZEL SENZOROVÉ SÍTĚ

Tato kapitola popisuje implementaci webového serveru s REST webovými službami na uzlu bezdrátové senzorové sítě.

Pro tuto práci byl k dispozici hardware AVR Raven. Pro něj byla vytvořena aplikace webového serveru s REST webovými službami v operačním systému Contiki viz kapitola 5.1. Podle URI požadované služby se spustí obslužná funkce, která vrací data formátována do JSON. Klíčové části kódu však nejsou platformě závislé a dají se zkompilovat i pro jiné platformy.

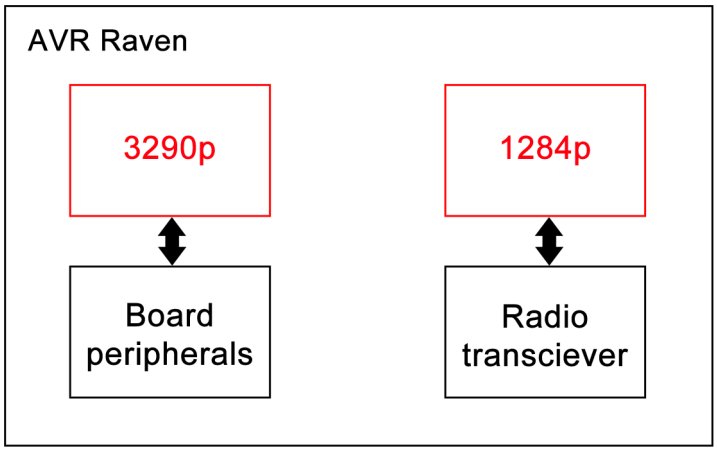
3.1 AVR Raven

Uzel senzorové sítě tedy tvoří AVR Raven viz obrázek 3.1, který má 2.4 GHz radio vysílač [13]. Toto zařízení se hodí pro ladění a demonstraci vytvořené aplikace ale ne pro praktické použití v opravdové senzorové síti.



Obr. 3.1: AVR Raven.

Na desce jsou dva procesory viz obrázek 3.2. K procesoru 3290p jsou připojeny periferie jako LCD, teplotní senzor reproduktor a pákový ovladač. Důležitější je ale procesor 1284p, na který je připojen radiový přijímač/vysílač. Na tomto procesoru tedy poběží aplikace REST serveru. Pokud by tedy tento uzel měl fungovat jako senzor je nutné implementovat komunikaci mezi těmito procesory přes UART (sériové rozhraní). Při realizaci této práce jsou senzory simulovány a nahrazeny generátorem náhodných hodnot. Rovněž je také možné povolit režim spánku - Idle sleep, který ale simulován ze stejného důvodu, tedy protože by bylo třeba implementovat komunikaci mezi procesory a to nebylo smyslem této práce.



Obr. 3.2: Procesory na desce AVR Raven.

3.2 Operační systém Contiki

Contiki [3] je otevřený, přenosný, víceúlohový operační systém pro vestavěné síťové systémy a bezdrátové senzorové sítě s efektivní práci s pamětí. Je navrhnut pro mikrokontroléry s malou operační pamětí. Typická konfigurace Contiki je 2 kB RAM a 40kB ROM [3]. Poskytuje komunikaci přes IPv4 a IPv6.

Díky tomu, že se jedná o open source projekt vznikla kolem něj poměrně velká komunita. Contiki poslední dobou využívá velké množství projektů s bezdrátovými senzorovými sítěmi a díky tomu již existuje podpora pro celou řadů platforem. V současné době je aktuální verze 2.5 RC1 a je téměř jisté, že podporovaných platforem bude přibývat.

Kromě Contiki existuje několik dalších operačních systémů, některé jsou otevřené a jiné komerční distribuce. Jako alternativa ke Contiki existuje ještě další volně dostupný systém - TinyOS [5].

3.3 Webový server s webovými službami REST

REST server zajišťuje prostřednictvím navrženého stromu zdrojů přístup ke všem definovaným phenomenon (jevům) umístěných na uzlu. Kromě získání dat, umožňuje také nastavovat parametry uzlu. Nastavitelné parametry jsou:

- Povolení Idle sleep - režim spánku
- Nastavení parametru measurement pro každy phenomenon (počet měření, ze kterých se má udělat průměr a odeslat hodnota)

Tyto parametry se pak uchovávají v proměnných programu uzlu zdroje, ale jejich funkce je pouze simulována. V práci je tedy řešeno jak tyto parametry nastavit na uzlu pomocí webové služby, ale jejich opravdová funkce na uzlu již implementována není.

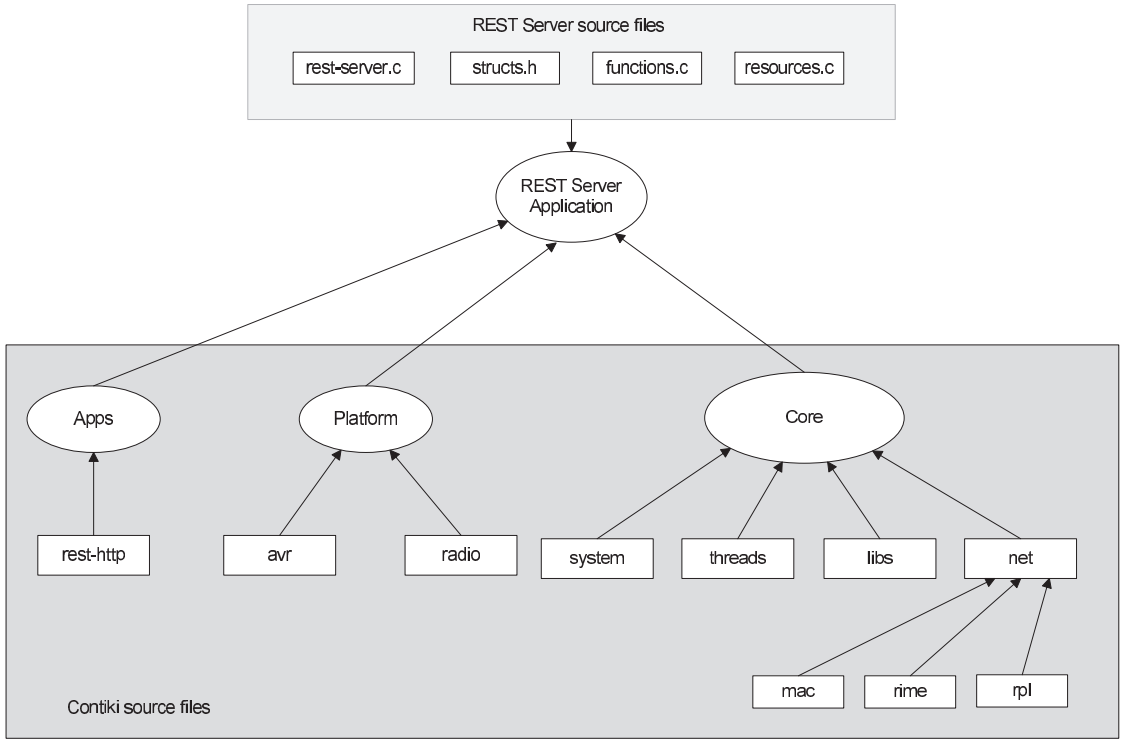
Hlavní části implementace jsou zdrojové kódy (REST Server source files) `rest-server.c`, `structs.h`, `functions.c` a `resources.c` - nacházející se v `contiki/projects/rest-server/`. Program byl navržen tak, že v souboru structs.h se definuje jaké jevy bude REST obsahovat. V resources.c jsou definovány jednotlivé zdroje pro jevy a obslužné funkce těchto zdrojů. Pomocné funkce pro práci s definovanými strukturami jsou v souboru functions.c a větev celého procesu REST serveru se spouští v rest-server.c - zde se také aktivují jednotlivé zdroje podle nadefinovaných jevů ve structs.h. Program pro webové služby je navržen tak, aby šlo zdroje dále přidávat. Jak přidat další zdroj popisuje kapitola 5.1.1, pro demonstraci funkce jsou definován zdroje - temperature (teplota), humidity (vlhkost) a acceleration (zrychlení).

Program REST server také linkuje jednotlivé části Contiki (Contiki source files) (obrázek 3.3). Jádro systému (Core) se skládá z bloku systémový prostředků (práce s procesy, časovou tabulí), vláken, knihoven pro práci s pamětí a časovači a bloku pro síťovou interakci (uip, tcp). Platform definuje pro jakou platformu se aplikace bude kompilovat. Kromě "hardwarových" platforem lze definovat i virtuální platformu, na které lze aplikaci simulovat bez fyzického zařízení. Kompilací pro platformu AVR Raven se zabývám níže. Část Apps linkuje prostředky pro definování REST zdrojů a prostředky pro spuštění webového serveru.

Nejprve se tedy nadefinují jevy, pro které mají být vytvořeny zdroje. Tyto jevy se deklarují ve zdrojovém kódu contiki/projects/rest-server/structs.h v proměnné *phenos*, což je pole struktur phenomenon (pole struktur jevů) viz ukázka 3.1. Tato struktura definuje name - název jevu, unit - jednotka jevu, parametr measurements - jak bylo psáno výše jedná se o simulovaný parametr počtu měření ze senzoru před odesláním hodnoty a rovněž simulovaný parametr offset - hodnota pro korekci měření senzoru.

Zdroj. kód 3.1: Pole struktur phenos a její deklarace v structs.h.

```
struct phenomenon phenos[16] = {
    {
        .name = "temperature",
        .unit = "C",
        .offset = 0,
        .measurements = 1
    },
```



Obr. 3.3: Přehled linkovaných částí programu webového serveru.

. . .

Pro každý phenomenon jsou definované dva zdroje:

- resource_phenomenonName - vrací hodnotu, respektive hodnoty jevu pokud jich je více
- resource_phenomenonName_opt - tvrací nastavení jevu a umožňuje nastavit parametr measurements

Tyto zdroje kromě definice, také potřebují obslužné funkce, které obstarávají data pro zobrazení zdroje. Tohle vše se řeší ve zdrojovém kódu resource.c a podrobněji to popisuje kapitola 3.3.4 a všechny zdroje na uzlu v kapitole 3.3.1.

Nakonec tedy po definici jevu, po definici zdroje jevu se ve vlákně procesu rest-server ve zdrojovém kódu contiki/rest-server/rest-server.c podle nadefinovaných jevů aktivují zdroje (ukázka 3.2).

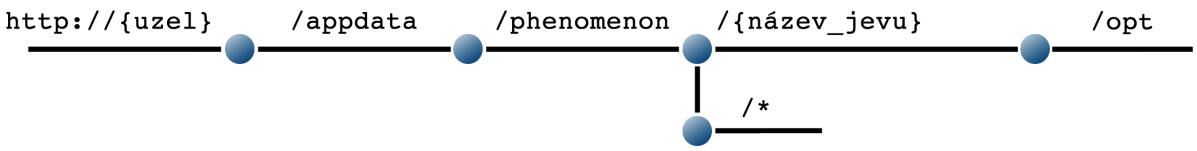
Zdroj. kód 3.2: Aktivace zdrojů v rest-server.c.

```
int i;
int n=sizeof(phenos)/sizeof(phenos[0]);
for(i=0;i<n;i++)
```

```
{
    //phenomenonTemperature
    if(!strcmp(phenos[i].name,"temperature"))
    {
      rest_activate_resource(&resource_phenomenonTemperature);
      rest_activate_resource(&resource_phenomenonTemperature_opt);
    }
    ...
}
```

3.3.1 Strom zdrojů

Webová služba REST spuštěná na uzlu definuje zdroje pro přístup k datům z uzlu. Tyto zdroje jsou dostupné pod URI, kterou má každý zdroj jedinečnou. Strom adres, na kterých jsou tyto zdroje dostupné je na obrázku 3.4. V obrázku uzel reprezentuje adresu k uzlu (IP adresu).



Obr. 3.4: Strom adresovatelných zdrojů REST serveru.

Pokud by měl uzel například jeden jev temperature, jeho zdroje by byly:

1. http://{uzel}/appdata - zdroj pro informace o uzlu
2. http://{uzel}/appdata/phenomenon - zdroj se seznamem jevů
3. http://{uzel}/appdata/phenomenon/temperature - zdroj pro zobrazení hodnoty jevu
4. http://{uzel}/appdata/phenomenon/temperature/opt - zdroj pro zobrazení vlastností jevu
5. http://{uzel}/appdata/phenomenon/* - zdroj pro zobrazení hodnot všech jevů

Tyto zdroje poskytují data z uzlu, která jsou formátována do JSON objektů. Struktura těchto objektů je popsána v kapitel 3.3.3 Každý zdroj má své parametry. Ty jsou popsány v následující kapitole.

3.3.2 Parametry zdrojů

V předchozí podkapitole bylo popsáno k čemu slouží adresy REST zdrojů. Nyní bude popsáno jaké používají parametry. Parametry jednotlivých adres zdrojů (tab. 3.1) mohou být čteny, ale i zapisovány ({phenomenon_name} - název jevu). Čtení probíhá pomocí HTTP požadavku typu GET a zápis pomocí POST. Jak bylo zmíněno dříve REST server posílá data v JSON strukturách.

URI zdroje	Parametry	Čtení/Zápis
/appdata	sent	Čtení
	received	Čtení
	ver	Čtení
	idlesleep	Čtení/Zápis
/appdata/phenomenon	names[]	Čtení
/appdata/phenomenon/{název_jevu}	val[]	Čtení
	seq	Čtení
/appdata/phenomenon/{název_jevu}/opt	unit	Čtení
	offset	Čtení
	measurements	Čtení/Zápis
/appdata/phenomenon/*	array(name,val[])	Čtení
	seq	Čtení

Tab. 3.1: Přehled REST zdrojů a jejich parametrů.

Význam jednotlivých parametrů:

- /appdata
  - sent - počet přijatých paketů uzlu
  - received - počet odeslaných paketů uzlu
  - ver - verze firmware uzlu
  - idlesleep - true/false parametr pro indikaci povolení režimu uzlu Idle sleep
- /appdata/phenomenon
  - name[] - pole názvů jevů, které jsou na uzlu adresovatelné (každý název je shodný s identifikátorem {phenomenon_name})
- /appdata/phenomenon/{phenomenon_name}
  - val[] - pole hodnot jevu (maximálně tři hodnoty)
  - seq - sekvenční číslo měření
- /appdata/phenomenon/{phenomenon_name}/opt
  - unit - jednotka jevu (např. °C)

  - offset - hodnota pro korekci měření jevu
  - measurements - počet měření jevu před odesláním
- /appdata/phenomenon/*
  - array(name,val[]) - pole s identifátorem a hodnotou všech jevů na uzlu (name je vždy {phenomenony_name})
  - seq - sekvenční číslo měření

Tyto parametry jsou zobrazovány na příslušných URI zdroje v JSON strukturách. V další podkapitole je specifikován jejich formát.

3.3.3 Formát JSON odpovědí

Pro bezproblémovou komunikaci mezi serverem a uzlem je potřeba určit přesnou podobu JSON objektů. Jak bylo popsáno výše, webová služba REST definuje několik zdrojů a ty jsou adresovatelné pomocí URI. Každá URI vrací data v JSON objektu. Některé adresy mají definované parametry jak pro čtení, tak pro zápis, proto některé URI vracejí data, jak pro HTTP GET požadavek, tak pro POST. Tyto JSON struktury se vytvářejí v obslužných funkcích zdrojů. Příklad vytvoření takové struktury je v ukázce 3.8 Data, které jsou obsaženy v těchto JSON objektech se berou z programu (senzorů) uzlu.

Adresa `/appdata` vrací JSON s parametry sent,received, ver a idlesleep (zdrojový kód 3.3). Protože je parametr idlesleep i zapisovatelný, je pro tento zdroj definovaná také odpověď pro HTTP požadavek typu POST. V této odpovědi se odesílá nové nastaveního parametru idlesleep.

Zdroj. kód 3.3: Formát JSON pro URI /appdata

```
HTTP GET:       {uzel}/appdata
HTTP RESPONSE:
{
        "sent": 142,
        "received": 142,
        "ver": "1.0",
        "idlesleep": "On" / "Off"
}
HTTP POST:      {uzel}/appdata
HTTP RESPONSE:
{
        "idlesleep" : "On" / "Off"
}
```

Na adrese `/appdata/phenomenon` zdroj webové služby zobrazuje JSON s názvy všech jevů uzlu (zdrojový kód 3.4). V této demonstraci byli definovány tři jevy, může jich být méně i více. JSON je pole se jmény jevů, které jsou zároveň identifikátory - {název_jevu} v URI.

Zdroj. kód 3.4: Formát JSON pro URI /appdata/phenomenon.

```
HTTP GET:      {uzel}/appdata/phenomenon
HTTP RESPONSE:
{
        "names": ["temperature","humidity","acceleration"]
}
```

Další adresa `/appdata/phenomenon/{název_jevu}` vrací JSON s hodnotou, případně hodnotami (maximálně 3) - parametr val[] (pole hodnot) a sekvenčním číslem měření - parametr seq. V ukázce 3.5 je JSON pro odeslání hodnoty jevu temperature.

Zdroj. kód 3.5: Formát JSON pro URI /appdata/phenomenon/temperature.

```
HTTP GET:      {uzel}/appdata/phenomenon/temperature
HTTP RESPONSE:
{
        "val": 31.2,
        "seq" : 3465
}
```

Zdroj na adrese `/appdata/phenomeon/{název_jevu}/opt` vrací JSON s parametry (unit, offset, measurements) pro daný jev (zdrojový kód 3.6). Parametr measurements lze přes POST měnit a pokud dojde HTTP požadavku typu POST, zdroj vrátí JSON s novou hodnotou pro parametr measurements.

Zdroj. kód 3.6: Formát JSON pro URI /appdata/temperature/opt.

```
HTTP GET:      {uzel}/appdata/phenomenon/temperature/opt/
HTTP RESPONSE:
{
        "unit" : "C",
        "offset" : 0.3,
        "measurements" : 1
}

HTTP POST:   {uzel}/appdata/phenomenon/temperature/opt/
```

```
HTTP RESPONSE:
{
        "measurements" : 2
}
```

A adresa `/appdata/phenomenon/*` vrací JSON s hodnotami všech definovaných jevů na uzlu (zdrojový kód 3.7. Je to pole názvů jevů (shodných s identifikátorem {název_jevu} a příslušné hodnoty respektive pole hodnot.

Zdroj. kód 3.7: Formát JSON pro URI /appdata/phenomenon/*.

```
HTTP GET:     {uzel}/appdata/phenomenon/ *
HTTP RESPONSE:
{
        {
                "name": "temperature",
                "val": 86.77
        },
        {
                "name": "acceleration",
                "val": 15.93
        },
        {
                "name": "humidity",
                "val": 35.86
        },
         "seq": 1
}
```

3.3.4 Definice zdroje a jeho obsluha

Do této části byla popsána deklarace jevů, strom zdrojů a jejich parametrů a teď bude popsáno jak se vlastně definuje samotný zdroj a funkce pro obsluhu zdroje. Definice zdroje probíhá pomocí funkce RESOURCE a její parametry jsou:

- Adresa zdroje - URI
- Povolené metody - typy HTTP požadavků
- Název obslužné funkce

Tímto se dané URI přiřadí funkce, která ve svém těle získá data a v požadovaném formátu - JSON objekt - sestaví HTTP odpověď.

V obslužné funkci REST zdroje jsou na vstupu dva parametry - ukazatel na request (požadavek) a ukazatel na response (odpověď). Request obsahuje parametry HTTP požadavku jako jsou například proměnné POST a do response se v obslužné funkci uloži data, která bude zdroj vracet.

Konkrétní příklad definice zdroje REST a jeho obslužné funkce je na ukázce 3.8. Jedná se o zdroj s vlastnostmi jevu temperature - `appdata/phenomenon/temperature/opt`.Ve funkci RESOURCE povoluje metody GET a POST a jako osblužná funkce je definována phenomenonTemperature_opt - tato funkce má pak v kódu své definice za svým názvem ještě přidáno _handler. V těle obslužné funkce phenomenonTemperature_opt_handler se načtou do proměnné *phenon* aktuální hodnoty pro daný jev. V následující podmínce se testuje pokud je definována POST proměnna measurements v HTTP *requestu*. Podle toho program pozná že se jedná o požadavek typu POST a uloží novou hodnotu z proměnné POST do struktury *phenos* a následně se do proměnné *output_string* uloží řetězec podle definovaného JSON formátu pro odpověď s nově přiřazenou hodnotou parametru measurements. V obou případech - tedy i požadavek typu POST i GET - se do proměnné *response* ukládá řetězec z proměnné *output_string* a definuje se typ HTTP odpovědi - v tomto případě JSON (APPLICATION_JSON).

Zdroj. kód 3.8: Část zdrojového kódu resources.c s definicí zdroje a jeho obslužné funkce.

```
RESOURCE(phenomenonTemperature_opt, METHOD_GET | METHOD_POST,
"appdata/phenomenon/temperature/opt");

void phenomenonTemperature_opt_handler(REQUEST* request,
 RESPONSE* response)
{
  char *output_string[100];
  struct phenomenon phenon;
  char param[10];
  int measurements, index;

  phenon = find_phenomenon("temperature");
if (rest_get_post_variable(request, "measurements", param, 10)){
      measurements = atoi(param);
      index = find_phenomenon_index("temperature");
      phenos[index].measurements = measurements;
      sprintf(output_string,"{ \"measurements\": %i}",
phenos[index].measurements);
```

```
  }
  else {
    sprintf(output_string,"{ \"unit\": \"%s\", \"offset\": %f,
\"measurements\": %i }",phenon.unit,phenon.offset,phenon.
measurements);
  }

  rest_set_header_content_type(response, APPLICATION_JSON);
  rest_set_response_payload(response, output_string,
strlen(output_string));
}
```

Zdrojový kód resources.c obsahuje definici všech možných zdrojů a jejich obslužných funkcí a ty pak podle nadefinovaných jevů ve struktuře *phenos* (structs.h) jsou v hlavním souboru rest-server.c aktivovány.

3.4 Kompilace Contiki pro platformu AVR Raven

Pro správnou kompilaci na rozdíl od návodu [4] je potřeba určit cíl avr-raven a příponu výsledného souboru pro kompilaci elf. Toho dosáhneme malou úpravou kompilačního příkazu - `make TARGET=avr-raven rest-server.elf`. Soubor po kompilaci obsauje část pro Program, Data a EEPROM. Ten se pak nahraje pomocí JTAG a AVR studia do uzlu.

Při používání vývojové verze z oficiálního git repositáře:
`git://contikis.git.sourceforge.net/gitroot/contiki/contiki` bylo při kompilaci programu pro platformu avr-raven několik chyb, kvůli kterým nešlo program zkompilovat. Nejprve bylo třeba opravit zdrojový kód
`contiki/apps/rest-common/static-routing.c`, zakomentováním řádku `#include node-id.h` a připsáním `short node_id = 1`, protože pro platformu avr-raven není hlavičkový soubor node-id.h definován a tudíž není definována proměnná node_id. Druhá chyba byla s dvojím uvolňováním paměti. Dále se podařilo objevit a opravit chybu ve zdrojovém kódu `contiki/apps/rest-common/buffer.c`.

4 SERVER SENZOROVÉ SÍTĚ

V této kapitole je popsán návrh aplikace serveru pro sběr dat v bezdrátové senzorové síti a datové úložiště, do kterého se sesbíraná data ukládají.

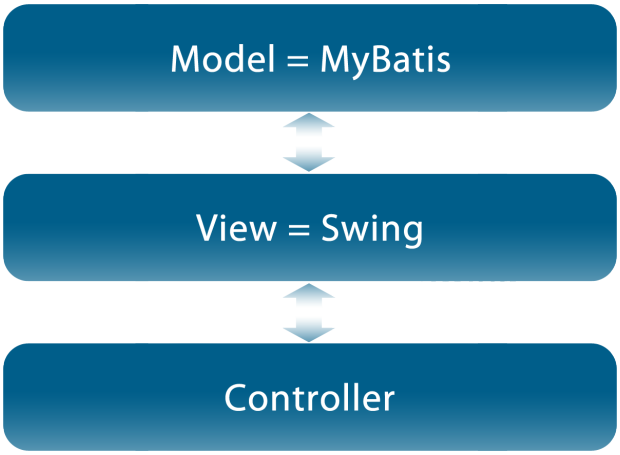
4.1 Aplikace serveru pro sběr dat

Serverová aplikace je realizovaná v jazyce JAVA. Tento jazyk jí zajišťuje platformní nezávislost díky JVM (Java Virtual Machine). Aplikace řídí a získává data ze senzorových uzlů respektive z jejich REST webových služeb běžících na webovém serveru. Data ukládá do relační databáze MySQL, která je také platformě nezávislá. K funkci celé aplikace je zapotřebí síťové rozhraní, které komunikuje se senzorovou sítí - brána.

Hlavní funkce aplikace serveru:

- Sběr dat z REST webových služeb v časových cyklech a intervalech
- Čtení a nastavování některých vlastnosti a parametrů uzlu
- Zpracování dat a uložení do databáze
- Klíčové parametry aplikace se dají konfigurovat (konfigurace se ukládá do externího souboru)

Návrh aplikace dodržuje principy architektury MVC (obr. 4.1) [7]. Tato architektura odděluje aplikaci na tři vrstvy - datový model, uživatelské rozhraní a řídicí logiku tak, aby na sobě byli co nejméně závislé. Pro zachování přehledu o tom, který zdrojový kód patří do jaké vrstvy budu v kapitole každé vrstvy vypisovat jejich seznam. Tyto zdrojové kódy se nacházejí v adresáři se zdrojovými kódy D.



Obr. 4.1: MVC vrstvy aplikace - Model View Controller.

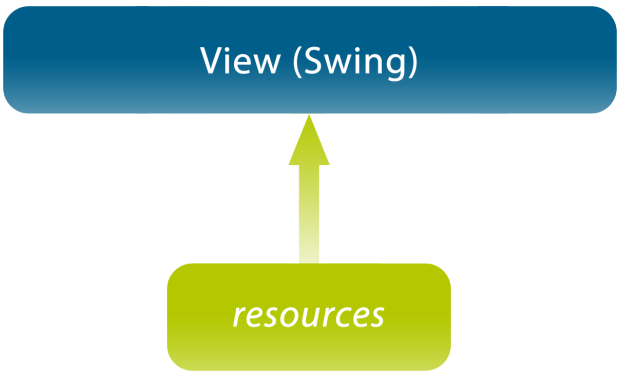
4.1.1 Brána do senzorové sítě

Jako brána do senzorové sítě 6LoWPAN byl použit AVR RZ USBstick. Součástí distribuce Contiki je připravená aplikace pro směrování paketů v senzorové síti. Tato aplikace se nachází v contiki/examples/ravenusbstick/. Postup instalace je popsán v kapitole 5.2.

Při prvním spuštění je potřeba nakonfigurovat a spustit směrování pro IPv6 - v Linux systému se jedná o démon radvd - a nakonfigurovat správnou IP adresu. Postup je rovněž popsán v kapitole 5.2.

4.2 Vrstva View (Swing)

Vrstva View (obr. 4.2) řeší část aplikace, se kterou příjde do styku uživatel - uživatelské rozhraní. Tuto vrstvu reprezentují standardní JAVA knihovny Swing a balíček *resources*. Pomocí knihoven Swing bylo vytvořeno grafické rozhraní celé aplikace, skládající se ze čtyř panelů. Každý obsahuje ovládací prvky, v jejichž obslužných funkcích se využívají třídy ostatních vrstev. Ovládací prvky jsou všechny definovány právě v balíčku *resources*. Tyto panely plní každý jinou funkci a jsou popsány v kapitolách 4.2.1, 4.2.2, 4.2.3 a 4.2.4.



Obr. 4.2: Vrstva View a její balíčky.

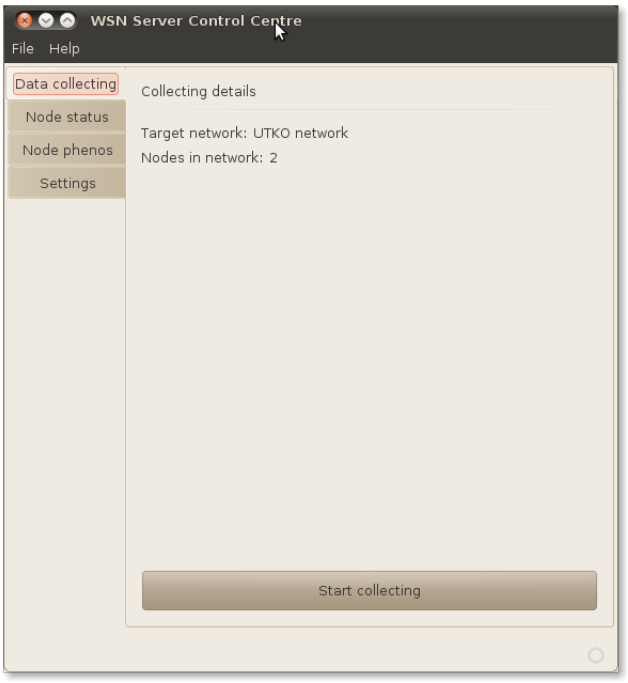
Zdrojové kódy této vrstvy: (vztah mezi odrážkami je balíček - zdrojový kód)

- wsnscc
  - WSNSCCAboutBox.Java
  - WSNSCCApp.Java
  - WSNSCCView.Java
- wsnscc.resources
  - WSNSCCAboutBox.properties

– WSNSCCAboutBox.properties
– WSNSCCView.properties

4.2.1 Data collecting

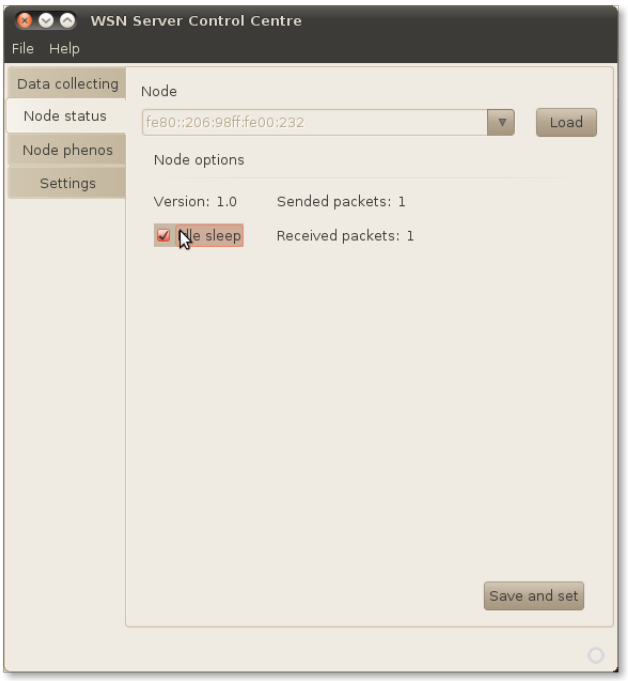
Panel Data collecting (obr. 4.3) slouží pro spuštění sběru dat. Zde se také, podle nastaveného počtu cyklů a časového intervalu mezi jednotlivými sběry v panelu 4.2.4, řídí cyklické sběry dat. Obsahuje stavové informace o aktuálním sběru a dole ve stavovém řádku se signalizuje dokončení operace.



Obr. 4.3: Panel pro kontrolu sběru dat.

4.2.2 Node status

Na tomto panelu (obr. 4.4) lze načíst stav uzlu z nastavené sítě. Tento panel pracuje s REST zdrojem /appdata. Načte data z vybraného uzlu a ty pak vizualizuje. Panel tedy zobrazuje informace o počtu přijatých a odeslaných paketech, verzi firmwaru nahraném na uzlu a také se zde dá nastavit režim spánku (Idlesleep), který je pouze simulován viz kapitola 3.3. V prvním kroku je třeba vybrat uzel a načíst jeho stavové informace. Po té lze nastavovat režim spánku uzlu.



Obr. 4.4: Panel pro zjištění stavových informací o uzlu.

4.2.3 Node phenomenons

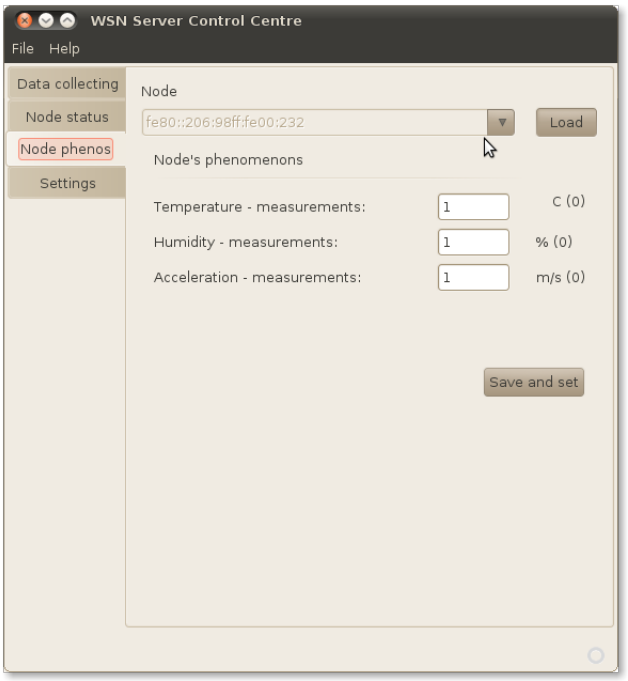
Tento panel (obr. 4.5) pracuje se zdrojem /appdata/phenomenon/{název_jevu}/opt. Pracuje tedy s nastaveními jednotlivých jevů, které jsou na uzlu definované. Parametr measurements je pomocí GUI prvků editovatelný. Funkce tohoto panelu jsou:

- Načte všechny definované jevy na uzlu
- Umožňuje nastavit parametr measurements pro jednotlivé jevy

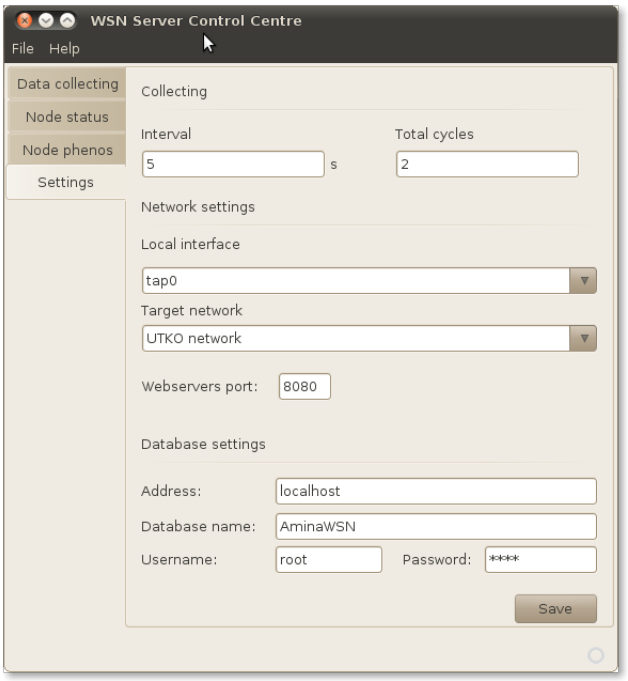
4.2.4 Settings

V tomto panelu obr. 4.6) se nastavuje celá aplikace serveru. K uchovávání konfigurace slouží konfigurační soubor conf.xml B.2. Ukládání nastavení a jeho čtení zajišťuje třída Config 4.3.3.
Funkce panelu settings:

- Konfigurace časových parametrů pro sběr dat
- Konfigurace síťového rozhraní a volba cílové sítě pro sběr dat
- Konfigurace přístupových údajů databáze



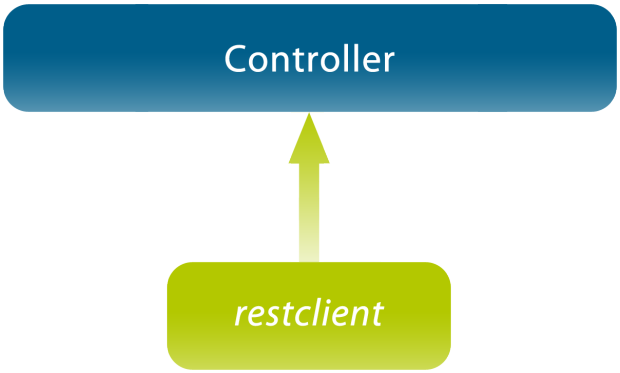
Obr. 4.5: Panel pro nastavení parametru measurements..



Obr. 4.6: Panel pro konfiguraci aplikace.

4.3 Vrstva Controller

Tato vrstva (obr. 4.7) zahrnuje třídy řešící logiku jednotlivých částí celé aplikace. Jsou to samostatné třídy, které jsou nezávislé na ostatních a mohou být použity i v jiných aplikacích. Atributy a definice metod těchto tříd se nachází v příloze B.1



Obr. 4.7: Vrstva Controller a její balíčky.

Zdrojové kódy této vrstvy:

- wsnscc.controls
  - Config.Java
  - NetworkInterfaces.Java
- wsnscc.controls.restclient
  - JSONData.Java
  - RestClient.Java

Třídy této vrstvy ve svých metodách používají knihovny, které nejsou ve standartním repozitáři JAVA knihoven. Proto je potřeba k buildování serverové aplikace do projektu vložit tyto knihovny:

- jersey-bundle-1.5.jar
- jsr311-api-1.1.1.jar
- gson-1.6.jar

4.3.1 RestClient

Tato třída je stěžejní celé aplikace. Řeší veškerou komunikaci s REST službami na uzlech senzorové sítě. Všechny metody posílají a přijímají data tak, jak byli specifikované v kapitole 3.3.2. Hlavní metody jsou getResource() a postResource().

Pro nastavení URI webové služby je potřeba nastavit nejen IP adresu ale také síťové rozhraní, které je připojeno k senzorové síti a port, na kterém běží webové servery na uzlech.

Jersey framework

Jersey framework implementuje pomocí balíčku JAX-RS prostředky pro práci s REST serverem i klientem. Díky ní je možné posílat HTTP požadavky typu GET a POST a následně zpracovávat přijaté data. Umožňuje i zpracovávat výjimky a chybové stavy. Tento framework je využit ve třídě RestClient.

google-gson

Tuto volně dostupnou knihovnu využívá třída RestClient ke konverzi přijatých JSON struktur do JAVA objektů. Tyto objekty pak mohou zpracovávat třídy na vrstvě Model a uložit tak data do databáze.

4.3.2 NetworkInterfaces

Tato třída obsahuje pouze metodu, která vrací pole řetězců s názvy dostupných síťových rozhraní stanice. K tomu využívá knihovny z balíčku java.net.*.

4.3.3 Config

Třída Config generuje XML konfigurační soubor podle nastavených parametrů z aplikace. Kromě toho také parsuje konfigurační XML soubor a načítá z něj nastavené parametry. XML soubor, ve kterém se uchovávají nastavené parametry se nazývá conf.xml. Struktura konfiguračního souboru je v příloze B.2.

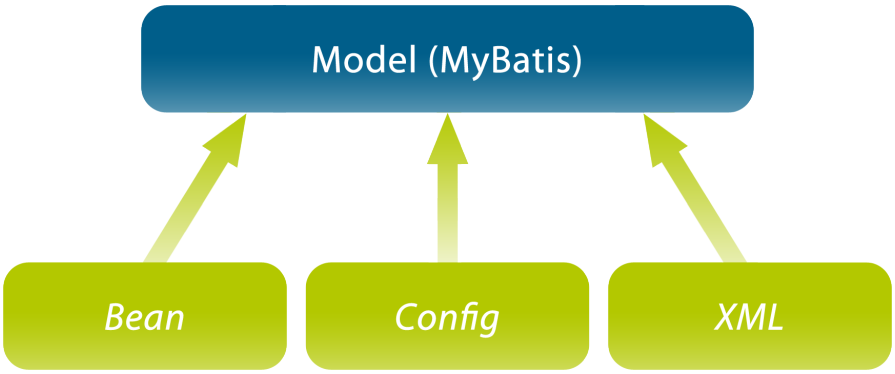
4.4 Vrstva Model (MyBatis)

Hlavní úkol této vrstvy (obr. 4.8) je práce s daty a s databází. Vybírá z databáze požadovaná data a vrací je v potřebném tvaru. A v opačném směru funguje obdobně. Tuto vrstvu řeší framework MyBatis [11], který ulehčuje práci s relační databází v objektově orientovaných aplikacích.

Zdrojové kódy této vrstvy:

- wsnscc.mybatis.bean
  - Data.Java
  - Network.Java

  - Node.Java
  - Sensor.Java
- wsnscc.mybatis,config
  - MyBatisSqlSessionFactory.Java
- wsnscc.mybatis,mapper
  - StaticMapper.Java
- wsnscc.mybatis,xml
  - Configuration.xml
  - StaticMapper.xml



Obr. 4.8: Vrstva Model a její balíčky.

Tato vrstva, ke své funkci také potřebuje knihovny, které nejsou ve standartním repozitáři. Tyto knihovny jsou:

- mysql-connector-java-5.1.14-bin.jar
- mybatis-3.0.4.jar

Framework MyBatis mapuje databázi prostřednictvím mapperu, který definuje SQL dotazy prostřednictvím XML. V tomto mapperu jsou jednotlivým SQL dotazům přiřazeny třídy z balíčku Bean, což jsou objekty reprezentující danou tabulku. MyBatis umožňuje převést relační databázi do objektového jazyka. K tomu potřebuje tyto balíčky - Bean, Config a XML.

4.4.1 Bean

Třídy balíčku Bean reprezentují entity a jejich atributy jako objekt v objektovém jazyce. Obsahuje tzv. gettery a settery pro všechny atributy entity. Dále pak definuje metody, které zastupují SQL dotazy definované v StaticMapperu. Byli vytvořeny třídy pro práci s entitami Data, Network, Node a Sensor.

Data

Tato třída reprezentující entitu DataHistory, umožňuje vkládat získané hodnoty z uzlů do databáze.

Network

Třída Network definuje metodu, pomocí které je možné načíst všechny definované sítě z databáze.

Node

Metody této třídy, čtou uzly z databáze. Lze také vybrat uzly patřící do jedné sítě.

Sensor

Tato třída pracuje z daty entity Phenomenon. Umožňuje číst z databáze, jaké hodnoty může uzel snímat.

4.4.2 Config

V tomto balíčku se definuje třída, která načte konfiguraci potřebných údajů pro připojení k databázi a vytvoří spojení. Těchto tříd může být i více, pokud je potřeba spojení k více databázím. Třída MyBatisSqlSessionFactory načte konfiguraci a vytvoří připojení k databázi MySQL pro ukládání sesbíraných dat.

4.4.3 XML

Balíček XML obsahuje konfigurační a mapovací XML. Soubor Configuration.xml obsahuje všechny údaje pro spojení s databází - typ databáze, adresu, port, přístupové údaje. Soubor StaticMapper.xml definuje všechny SQL dotazy a přiřazuje je metodám, definovaným v příslušných Beanech.

4.5 Datové úložiště

Server ukládá data do databáze. V předchozí podkapitole bylo popsáno jakým způsobem vrstva Model řeší přístup do databáze. Nyní k samotné databázi. Byla použita databáze pro projekt AminaWSN [15].

4.5.1 Databáze MySQL

Pro datové úložiště byl vybrán databázový engine MySQL [14]. Je multiplatformní (Linux, Windows, MAC) relační databáze, optimalizovaná pro rychlost a je volně dostupná. Navíc je tato databáze poměrně dost rozšířená, díky čemuž vzniklo spoustu projektů, které řeší přístup do této databáze v mnoha programovacích jazycích.

4.5.2 Specifikace dat

Přístup serverové aplikace k datům databáze se dá oddělit do dvou skupin z hlediska způsobu práce s tabulkami:

- čtení dat (tabulky Network, Node, Phenomenon)
- zápis dat (tabulka DataHistory)

Aplikace serveru načítá dostupné sítě (tabulka Network) a adresy uzlů (tabulka Node) z databáze, ale tyto data nijak neupravuje pouze je čte. Stejně tak pouze čte data z tabulky phenomenon. Data se zapisují pouze do tabulky DataHistory. ERD (Entitně relační diagram) celé databáze je v příloze C.

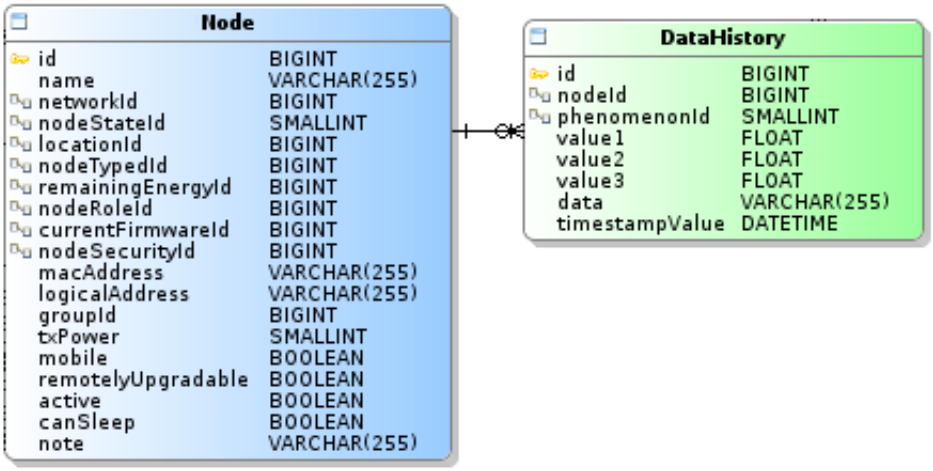
Data, která nemůže vkládat serverová aplikace a využívá je, musí tedy být do databáze vloženy ručně (např. pokud je potřeba přidat uzly, ze kterých se budou sbírat data). Předpokládá se ale, že zadávání a správu těchto dat bude řešit jiná aplikace, která mimo zadávání dat bude řešit také jejich vizualizaci.
Dále jsou popsány některé hlavní entity databáze.

4.5.3 Network

Entita Network udržuje důležité informace o síti. Hlavní je relace 1:N s entitou Node (obr. 4.9). Pomocí této vazby lze uzly třídit do skupin respektive do sítí. Má vazby na další entity FrequencyBand a NetworkProtocol. Obsahuje příznak active, pomocí, kterého lze přiřadit síti neaktivní stav a tím ho vyřadit se seznamu sítí pro sbět dat.

4.5.4 Node

Tato entita je srdcem celé databáze. Mezi hlavní vazby patří vazba k Network, NodeType a DataHistory. Definuje klíčové atributy - logickou a fyzickou adresu uzlu.



Obr. 4.9: Relace 1:N entit Node a DataHistory.

4.5.5 Phenomenon

Určuje jev - má pouze atribut name. Má relaci k DataHistory, k Units - jednotka jevu (např. m/s). Dále má relaci 1:1 k SensorPhenomenonList - přiřazuje jev k sensoru uzlu.

4.5.6 DataHistory

V této tabulce se uchovávají veškerá sesbíraná data od všech uzlů. Rozlišuje relací 1:1 o jaký se jedná phenomenon, obsahuje časovou značku. Relace 1:1 k Node určuje který uzel poslal tyto data. Hodnoty jsou maximálně tři, nebyl určen jev, který by potřeboval více (x,y,z).

5 VÝSLEDKY PRÁCE

V této kapitole je popsáno jak funguje realizovaná aplikace a co vše je potřeba pro její spuštění. Psáno pro operační systém Ubuntu.

5.1 Firmware pro AVR Raven

Zkompilovaný firmware rest-server.elf byl vytvořen jako součást této práce a je v příloze D v adresáři /node/bin , je potřeba naprogramovat do procesoru 1284p pomocí programátoru jTAG. Nastavení pojistek pro 1284p:

- EXTENDED: 0xFF
- HIGH: 0x99
- LOW: 0xE2

Přednastavená konfigurace uzlu:

- IP adresa: fe80::206:98ff:fe00: 4456 - {node_ip}
- Port webového serveru: 8080 - {webserver_port}
- Název síťového rozhraní pro komunikaci s WSN: usb0 (může se lišit) - {interface_name}
- URI uzlu: http://[{node_ip}%{interface_name}]:{webserver_port}/

Příklad URI adresy uzlu: http://[fe80::206:98ff:fe00:4456%usb0]:8080. Pod touto adresou je uzel také dostupný přes webový prohlížeč.

V případě potřeby je možné změnit port HTTP serveru ve zdrojovém kódu Contiki: contiki/appdata/rest-http/http-common.h na řádku 47 - `#define HTTP_PORT 8080`. Při kompilaci firrmware pro více uzlů je třeba změnit v Contiki IP adresu uzlu. K tomu slouží zdrojový kód raven platformy: contiki/platform/avr-raven/raven-main.c, řádek 133 - `uint8_t mac_address[8] EEMEM = {0x02, 0x11, 0x22, 0xff, 0xfe, 0x33, 0x44, 0x58}`).

Definované zdroje REST webové služby:

- /appdata
- /appdata/phenomenon
- /appdata/phenomenon/temperature
- /appdata/phenomenon/temperature/opt
- /appdata/phenomenon/humidity
- /appdata/phenomenon/humidity/opt
- /appdata/phenomenon/acceleration
- /appdata/phenomenon/acceleration/opt
- /appdata/phenomenon/*

Protože na tomto uzlu nejsou k dispozici všechny senzory, i když je možné je připojit, jsou hodnoty pro ukázku funkce komunikace generovány náhodně viz ukázka 5.1.

Zdroj. kód 5.1: Funkce pro generování náhodné hodnoty.

```
char *randval()
{
    char *val = (char *) malloc(10);
    char a[10],b[5];

    sprintf(a,"%i",rand()%100);
    sprintf(b,".%i",rand()%100);
    strcat(a,b);

    sprintf(val,a);

    return val;
}
```

5.1.1 Postup při přidání nového zdroje

Funkce webové služby REST a význam jednotlivých parametrů na uzlu senzorové sítě je popsána v kapitole 3.3. V této kapitole je popsáno jak přidat zdroj pro nový jev - svítivost. Veškeré zdrojové kódy, které je potřeba upravit se nacházejí v adresáři /contiki/projects/rest-server/. Přidání nového zdroje jevu (phenomenon) na uzel se skládá z následujících kroků:

- Deklarace nového jevu v proměnné *phenos* v souboru structs.h
- Definice nového zdroje a jeho obslužné funkce v souboru resources.c
- Aktivace nového zdroje

Deklarace jevu

Pro deklaraci nového jevu je potřeba pouze přidat atributy name, unit, offset a measurement do proměnné *phenos* (zdrojový kód 5.4).

Zdroj. kód 5.2: Přidání nového jevu do proměnné phenos (structs.h.

```
struct phenomenon phenos[16] = {
    ...
    {.name = "lightness", .unit="I", .offset = 0,
     .measurements = 1} };
```

Definice zdroje a jeho obslužné funkce

Nový zdroj se vytvoří pomocí funkce RESOURCE. Ve zdrojovém kódu 5.4 je přiřazena obslužná funkce, povolení HTTP požadavku GET a definice URI, na které bude tento zdroj dostupný. V těle obslužné funkce phenomenonLightness_handler, pak bude kód pro získání hodnoty ze senzoru a následné úpravy do řetězce znaků podle definovaného formátu z kapitoly 3.3.3. Tento řetězec se vloží do proměnné *temp*, ze které bude odeslána jako HTTP odpověd.

Zdroj. kód 5.3: Definice nového zdroje a jeho obslužné funkce (resources.c).

```
RESOURCE(phenomenonLightness, METHOD_GET,
"appdata/phenomenon/lightness");
void
phenomenonLightness_handler(REQUEST* request, RESPONSE* response){
  char temp[100] ;
  //v této části je potřeba naplnit proměnnou temp
  rest_set_header_content_type(response, APPLICATION_JSON);
  rest_set_response_payload(response, temp, strlen(temp));
}
```

Aktivace nového zdroje

Zbývá aktivovat nový zdroj pomocí funkce rest_activate_resource - její parametr je název zdroje. Je potřeba přidat podmínku, kvůli tomu, že když nebude proměnná *phenos* obsahovat jev lightness, pak se ani nebude aktivovat její zdroj.

Zdroj. kód 5.4: Aktivace nového zdroje (rest-server.c).

```
  int n=sizeof(phenos)/sizeof(phenos[0]);
  for(i=0;i<n;i++)  {
      if(!strcmp(phenos[i].name,"lightness"))       {
        rest_activate_resource(&resource_phenomenonLightness);
      }
      ...
```

5.2 WSN Server Control Centre

Požadavky serverové aplikace:

- Nainstalovaný balíček JVM
- Databáze MySQL se strukturou podle ERD v příloze C a vloženými daty se sítěmi, uzly a meřitelnými phenomenony

- Síťové rozhraní pro komunikaci se senzorovou síti

Databáze může být spuštěna na localhostu nebo na stanici, která je dostupná přes lokální síť nebo síť internet. Před prvním spuštěním aplikace je potřeba nastavit v souboru conf.xml přístupová data k databázi:

- Address - adresa stanice, na které je umístěna databáze (např. localhost)
- DatabaseName - název databáze
- Username - uživatelské jméno
- Password - heslo

Po připojení USB adaptéru je potřeba správně nakonfigurovat jeho IP adresu a povolit směrování do IPv6 sítí - spustit a nakonfigurovat démon radvd. Nastavení IP adresy pro rozhraní komunikující se senzorovou sítí se provede shell příkazy:

- `ip -6 address add fe80::0012:13ff:fe14:1516/64 scope link dev usb0`
- `ip -6 address add aaaa::1/64 dev usb0`

Démon radvd se nakonfiguruje v souboru /etc/radvd.conf podle ukázky 5.5.

Zdroj. kód 5.5: Konfigurace radvd.

```
interface usb0
{
    AdvSendAdvert on;
    AdvLinkMTU 1280;
    AdvCurHopLimit 128;
    AdvReachableTime 360000;
    MinRtrAdvInterval 100;
    MaxRtrAdvInterval 150;
    AdvDefaultLifetime 200;
    prefix AAAA::/64
    {
        AdvOnLink on;
        AdvAutonomous on;
        AdvPreferredLifetime 4294967295;
        AdvValidLifetime 4294967295;
    };
};
```

Dále je nutné povolit směrování shell příkazem:
`echo 1 > /proc/sys/net/ipv6/conf/all/forwarding`. A spustit démon příkazem: `/etc/init.d/radvd restart`.

Položky pro nastavení databáze byli popsány výše, nyní k ostatním položkám v panelu konfigurace:

- Total cycles - počet cyklů sběru dat
- Interval - čas mezi jednotlivými cykly sběru (ve vteřinách)
- Local Interface - rozhraní pro komunikaci v senzorové síti
- Target network - síť, ve které má dojít ke sběru dat (bere se z databáze)
- Webserver port - port, na kterém běží webové servery uzlů

Význam položek Total cycles a interval je vysvětlen v následující kapitole. Ostatní byli vysvětleny výše.

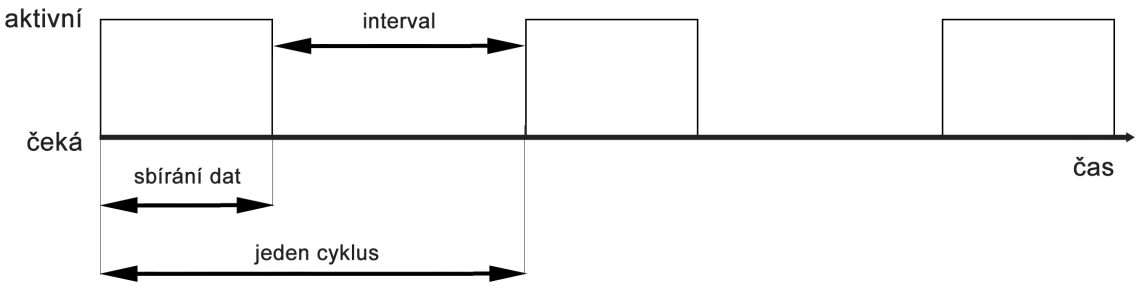
5.2.1 Sběr dat z uzlů senzorové sítě

Celý proces sběru dat se zkládá z několika časových úseků.

- sbírání dat - sekvence HTTP požadavků na REST zdroje
- interval - doba, kterou proces serverové aplikace čeká na další sběr dat
- jeden cyklus - sběr dat + interval

Na obrázku 5.1 jsou znázorněny přechody procesu sběru dat aplikace serveru z aktivního do stavu čeká. Z hlediska serverové aplikace je však důležitý pouze jeden časový parametr - interval - čas mezi jednotlivými sekvencemi sběru dat. Dalším důležitým parametrem je počet opakování. Oba tyto klíčové parametry jsou konfigurovatelné v nastavení aplikace:

- Total cycles - počet opakování sběru dat z webových serverů uzlů
- Interval - doba mezi opakovanými sběry dat



Obr. 5.1: Časový diagram sběru dat.

Samotný proces sbírání se z navrženého stromu REST zdrojů dá realizovat dvěma způsoby:

- Postupně - nejprve načte seznam definovaných jevů a poté načítá postupně hodnoty ze všech jevů
- Dávkou - načte z jediného REST zdroje hodnoty všech jevů najednou

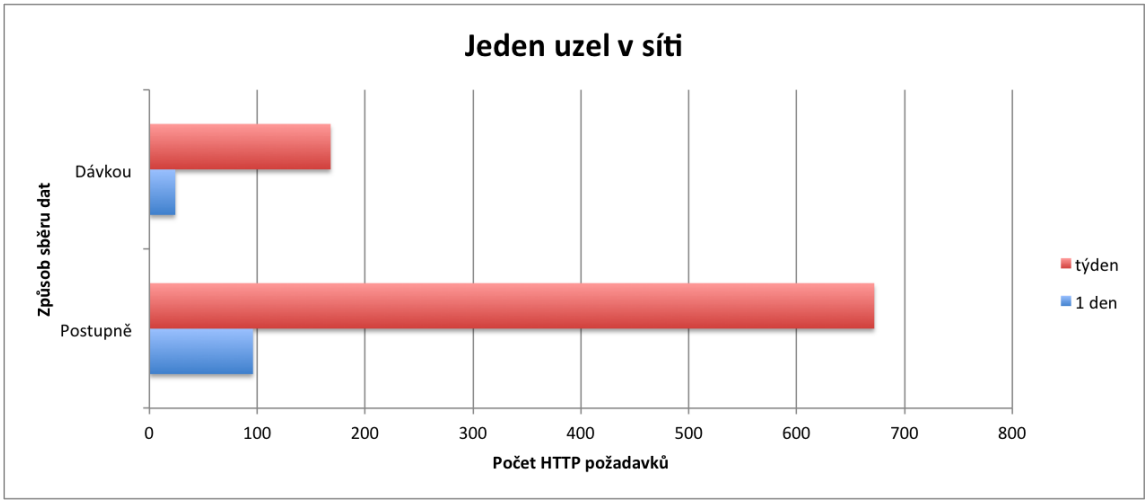
Postupný sběr dat probíhá následujícím způsobem:

- Načtení zdroje /appdata/phenomenon
- Z přijatého JSON se vytvoří URI zdrojů všech jevů a postupně se načítají
- Přijaté JSON se zpracují a data z nich se uloží do databáze

Dávkový sběr dat probíhá následujícím způsobem:

- Načtení zdroje /appdata/phenomenon/*
- Přijatý JSON se zpracuje a data se uloží do databáze

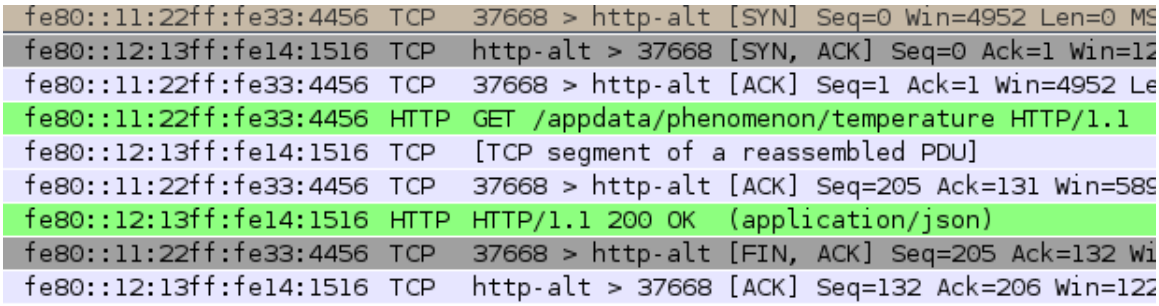
Pro představu je na obrázku 5.2 srovnání počtu potřebných požadavků (1 požadavek = 1 HTTP požadavek REST zdroje) na získání dat ze všech definovaných jevů na uzlu. Pro graf platí, že sběry probíhají v síti s jedním uzlem, na každém uzlu jsou definovány 3 jevy a interval mezi sběry je 1 hodina. Je vidět, že implementovaný způsob sběru je velmi neefektivní. Smyslem práce však nebylo navrhnout efektivní aplikaci pro použití v praxi. Druhý neimplementovaný způsob v aplikaci je zde uveden aby bylo jasné, že sběr dat lze realizovat efektivněji s ohledem na energetické podmínky bezdrátové senzorové sítě. Na straně uzlu však tento zdroj definovaný je a lze jej tedy použít.



Obr. 5.2: Porovnání počtu požadavků při sběru dávkou a při sběru po jednotlivých zdrojích.

Postupné načítání hodnot jevů je nepoužitelné pro hromadné sběry dat. Praktické využití tento způsob má, pokud je na uzlu definováno více jevů ale sběr probíhá pouze z jednoho nebo z několika málo zdrojů. Aby aplikace využila všechny navržené zdroje byl implementován tento způsob.

Jak vypadá načtení hodnoty z jednoho zdroje jevu je zachyceno na obrázku 5.3. Tyto TCP rámce byli zachyceny při získávání dat ze zdroje s URI /appdata/phenomenon/temperature. Jako odpověď dorazila požadovaná JSON struktura.



Obr. 5.3: Průběh komunikace při získání zdroje /appdata/phenomenon/temperature zachycena ve wiresharku.

5.2.2 Možností řízení uzlu

Pro řízení uzlu byly definovány dva parametry zdrojů REST:

- Parametr idlesleep zdroje /appdata
- Parametr measurements zdroje /appdata/phenomenon/{název_jevu}/opt

Ke skutečnému řízení daných funkcí však nedochází. Firmware na uzlu řeší pouze komunikaci se serverem a tedy i ovládání těchto parametrů. Na uzlu jsou tedy příslušné změny parametrů přijímány a uloženy do paměti programu, ale na funkci uzlu nemají žádný vliv.

Pokud by však jejich funkce byla doimplementována, serverová aplikace má všechny potřebné funkce implementovány a mohla by takto řídit povolení uzlu přechodu do režimu spánku. A pak také řídit počet měření ze senzoru před odesláním např. průměrné hodnoty.

5.3 Analýza komunikace s REST webovou službou v senzorové síti

Tato aplikace byla otestována v experimentální senzorové síti, Cílová síť, ve které probíhal sběr dat se skládala ze:

- Server s bránou RZ USB stick - IP fe80::12:22ff:fe14:1516
- Uzel 1 RZ Raven - IP fe80::11:22ff:fe33:4455
- Uzel 2 RZ Raven - IP fe80::11:22ff:fe33:4456

Nastavení sběru dat bylo v intervalech po 10-ti vteřinách se dvěma cykly. Na obrázku 5.4 je zachycena celá komunikace na vrstvě HTTP protokolu.

fe80::11:22ff:fe33:4455	HTTP	GET /appdata/phenomenon HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)
fe80::11:22ff:fe33:4455	HTTP	GET /appdata/phenomenon/temperature HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)
fe80::11:22ff:fe33:4455	HTTP	GET /appdata/phenomenon/acceleration HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)
fe80::11:22ff:fe33:4455	HTTP	GET /appdata/phenomenon/humidity HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)
fe80::11:22ff:fe33:4456	HTTP	GET /appdata/phenomenon HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)
fe80::11:22ff:fe33:4456	HTTP	GET /appdata/phenomenon/temperature HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)
fe80::11:22ff:fe33:4456	HTTP	GET /appdata/phenomenon/acceleration HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)
fe80::11:22ff:fe33:4456	HTTP	GET /appdata/phenomenon/humidity HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)
fe80::11:22ff:fe33:4455	HTTP	GET /appdata/phenomenon HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)
fe80::11:22ff:fe33:4455	HTTP	GET /appdata/phenomenon/temperature HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)
fe80::11:22ff:fe33:4455	HTTP	GET /appdata/phenomenon/acceleration HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)
fe80::11:22ff:fe33:4455	HTTP	GET /appdata/phenomenon/humidity HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)
fe80::11:22ff:fe33:4456	HTTP	GET /appdata/phenomenon HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)
fe80::11:22ff:fe33:4456	HTTP	GET /appdata/phenomenon/temperature HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)
fe80::11:22ff:fe33:4456	HTTP	GET /appdata/phenomenon/acceleration HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)
fe80::11:22ff:fe33:4456	HTTP	GET /appdata/phenomenon/humidity HTTP/1.1
fe80::12:13ff:fe14:1516	HTTP	HTTP/1.1 200 OK (application/json)

Obr. 5.4: Zaznamenaná komunikace při sběru dat z více uzlů senzorové sítě.

V ukázce 5.6 je vidět obsah odpovědi na první HTTP požadavek GET z předchozího záznamu komunikace. Jedná se o TCP paket, který má v datové části požadovanou JSON strukturu.

Zdroj. kód 5.6: Zachycená odpověď zdroje /appdata/phenomenon/temperature - TCP paket převedený do tisknutelné podoby.

```
"3DU‘"3DUPmHTTP/1.1 200 OK
Server: Contiki
Connection: close
Content-Type: application/json

{ "val": [40.73, 61.58, 61.50], "seq": 1 }
```

Takto získané JSON struktury serverová aplikace zpracovala a uložila do databáze viz obrázek 5.5. Ve výpisu jsou vidět získaná data z uzlů ID 1 a 2. Pro každý uzel je v databázi 6 záznamů s naměřenými hodnotami z toho 3 odpovídají jednomu sběru. Mezi časy uložených trojic záznamů jednoho uzlu je rozdíl 10 vteřin, což odpovídá nastavenému intervalu.

id	nodeId	phenomenonId	value1	value2	value3	data	timestampValue
601	1	1	40.73	61.58	61.5		2011-05-24 12:23:12
602	1	3	5.66	NULL	NULL		2011-05-24 12:23:12
603	1	2	96.22	60.58	NULL		2011-05-24 12:23:12
604	2	1	40.73	61.58	61.5		2011-05-24 12:23:13
605	2	3	5.66	NULL	NULL		2011-05-24 12:23:13
606	2	2	96.22	60.58	NULL		2011-05-24 12:23:13
607	1	1	40.11	85.68	42.18		2011-05-24 12:23:22
608	1	3	53.27	NULL	NULL		2011-05-24 12:23:22
609	1	2	34.35	81.2	NULL		2011-05-24 12:23:22
610	2	1	40.11	85.68	42.18		2011-05-24 12:23:23
611	2	3	53.27	NULL	NULL		2011-05-24 12:23:23
612	2	2	34.35	81.2	NULL		2011-05-24 12:23:23

Obr. 5.5: Sesbíraná data z uzlů uložena v databázi.

6 ZÁVĚR

Cílem této práce bylo realizovat sběr dat v bezdrátové senzorové síti z webových serverů uzlů. Pro realizaci práce byla zvolena síť 6LoWPAN, která na rozdíl od Zigbee umožňuje posílat data přes protokol IP verze 6. Realizace sběru dat byla rozdělena na část uzlu a část serveru.

Pro uzly senzorové sítě byl implementován firmware pomocí operačního systému Contiki. Tento firmware spouští na uzlu webový server s REST službou, která podle definovaných zdrojů poskytuje data ze senzorů uzlu.

Senzorová síť se skládá z uzlů, každý z nich má svou IP adresu a tyto údaje o síti a uzlech jsou uloženy v databázi. Serverová aplikace tyto data čte a podle uživatelem zvolené sítě, dotazuje webové servery s REST službami všech uzlů. Dotazy jsou HTTP požadavky typu GET pokud se jedná o čtení dat nebo typu POST při posílání dat směrem k uzlu. Data jsou JSON objekty s pevně stanovenou strukturou. Přijatá data server konvertuje do objektů a pomocí frameworku MyBatis a namapovaných Bean tříd a SQL dotazů ukládá data do databáze. Uživatel kromě volby cílové senzorové sítě v serverové aplikaci nastavuje také síťové rozhraní, které umožňuje spojení se senzorovou sítí a také konfiguruje časové intervaly, ve kterých budou data sbírány. Dále je také možnost konfigurovat přístupové údaje k databázi, díky čemuž může být databázový server umístěn na jiné stanici než na které běží aplikace pro sběr dat (např. stanice na lokální síti nebo v síti internet). Kromě toho server umožňuje také nastavovat některé parametry na uzlu, jako je nastavení spánku a konfigurace měření jevu. Tyto parametry se ale pouze nastavují do paměti programu uzlu a nemají vliv na funkci uzlu.

V poslední části práce je předvedena realizovaná aplikace s výsledky, které ukazují funkci celého řešení. Podle návrhu z kapitol 2 a 3 byl realizován sběr dat prostřednictvím webových služeb REST, spuštěném na webovém serveru uzlu bezdrátové senzorové sítě, a proto si myslím, že zadání bylo úspěšně splněno.

LITERATURA

[1] DUNKELS, Adam; VASSEUR, Jean-Philippe *Interconnecting Smart Objects with IP: The Next Internet. California: Morgan Kaufmann* 2010, ISBN 0123751659.

[2] Shelby, Zach; *6lowpan: The Wireless Embedded Internet* 2010, ISBN 9780470747995.

[3] *Contiki Online Documentation* [online].
Poslední aktualizace 6. 9. 2010 [cit. 10. 11. 2010]. Dostupné z URL: <http://www.sics.se/ adam/contiki/docs/>.

[4] *Contiki Install and Compile* [online].
Poslední aktualizace 8. 12. 2010 [cit. 15. 5. 2011]. Dostupné z URL: <http://www.sics.se/contiki/install-and-compile.html>.

[5] *TinyOS Documentation Wiki* [online].
Poslední aktualizace 2011 [cit. 21. 3. 2011]. Dostupné z URL: <http://docs.tinyos.net/index.php/Main_Page>.

[6] *W3C Note: Simple Object Access Protocol (SOAP) 1.1* [online].

Poslední aktualizace 8. 5. 2000 [cit. 1. 12. 2010]. Dostupné z URL: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.

[7] *Java SE Application Design With MVC* [online].

Poslední aktualizace březen. 2007 [cit. 12. 5. 2011]. Dostupné z URL: <http://www.oracle.com/technetwork/articles/javase/index-142890.html>.

[8] *Hypertext Transfer Protocol – HTTP/1.1* [online].

Poslední aktualizace 1.9. 2004 [cit. 12. 5. 2011]. Dostupné z URL: <http://www.w3.org/Protocols/rfc2616/rfc2616.html>.

[9] *RomXoap*^{TM} [online].

Poslední aktualizace 2010 [cit. 10. 5. 2011]. Dostupné z URL: <http://www.allegrosoft.com/datasheets/RomXoap.pdf>.

[10] Roy Thomas Fielding *Architectural Styles and the Design of Network-based Software Architectures* [online].

Poslední aktualizace 2010 [cit. 10. 5. 2011]. Dostupné z URL: <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

[11] *MyBatis 3: User Guide* [online].

Poslední aktualizace 1. 3. 2011 [cit. 20. 4. 2011]. Dostupné z URL: <http://mybatis.googlecode.com/svn/trunk/doc/en/MyBatis-3-User-Guide.pdf>.

[12] Dogan Yazar, Adam Dunkels *Efficient Application Integration in IP-Based Sensor Networks* [online].

Poslední aktualizace Listopad 2009 [cit. 1. 12. 2010]. Dostupné z URL: <http://www.sics.se/ãdam/yazar09efficient.pdf>.

[13] *AVR2016: RZRAVEN Hardware User's Guide* [online].

Poslední aktualizace: 2008 [cit. 10. 5. 2010]. Dostupné z URL: <http://www.atmel.com/dyn/resources/prod_documents/doc8117.pdf>.

[14] *MySQL 5.0 Reference Manual* [online].
poslední aktualizace 2011 [cit. 12.5. 2010].
Dostupné z URL: <http://dev.mysql.com/doc/refman/5.0/en/>.

[15] *Amina - bezdrátová senzorová síť* [online].
poslední aktualizace 2011 [cit. 12.5. 2010].
Dostupné z URL: <http://www.aminawsn.org/>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

WSN Wireless Sensor Network - Bezdrátová senzorová síť

WPAN Wireless Personal Area Network - Bezdrátová osobní síť (s malým dosahem)

IPv6 Internet Protokol verze 6

6LoWPAN IPv6 over WPAN - IPv6 přes WPAN

SOAP Simple Object Access Protocol

REST Representational State Transfer

HTTP Hypertext Transfer Protocol

TCP Transmission Control Protocol

ERD Entitně relační diagram

XML Extensible Markup Language - Rozšiřitelný značkovací jazyk

JSON Java Script Object Notation - Javaskriptová objektový zápis

URI Uniform Resource Identifier - Jednotný identifikátor zdroje

ROM Read-Only Memory - Paměť pouze pro čtení

RAM Random Access Memory - Paměť s přímým přístupem

RTP Real-time Transport Protocol

UDP User Datagram Protocol

JAVA Programovací jazyk

JVM Java Virtual Machine - Virtuální stroj JAVA

SQL Structured Query Language - Strukturovaný dotazovací jazyk

GUI Graphic User Interface - uživatelské rozhraní

SEZNAM PŘÍLOH

A ČÁSTI ZDROJOVÝCH KÓDU REALIZUJÍCÍ FUNKCI REST SERVERU

A.1 Definice funkcí v functions.c

```
//searchs phenos for phenomenon with .name=name
struct phenomenon find_phenomenon(char name[32])

// returns index of phenomenon with .name=name
int find_phenomenon_index(char name[32])

// returns pointer to random float value in string generated
from interval (min,max)
char *randval(int min, int max)
```

A.2 Definice struktur v structs.h

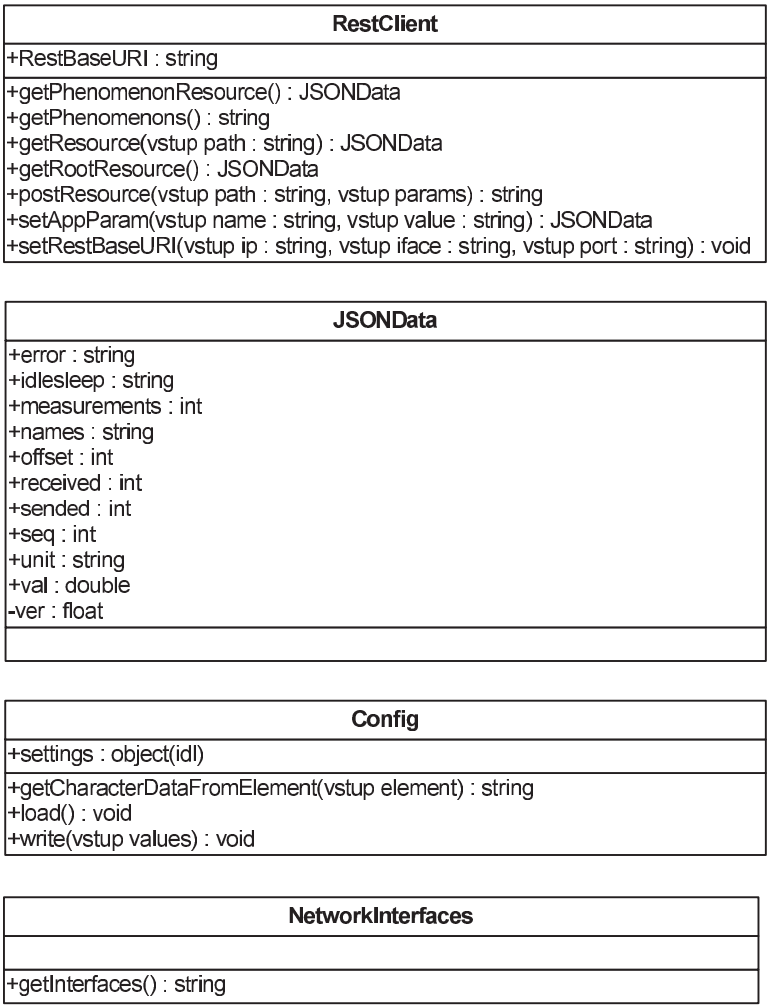
```
// aplication parameters
struct application
{
    char name[32];
    long sended;
    long received;
    char idlesleep[5];
    char ver[5];
};

//phenomenons parameters
struct phenomenon
{
     char name[32];
     char unit[16];
     float offset;
     int measurements;
};
```

A.3 Definice zdrojů a obslužných funkcí v resources.c

```
/* AppData RESOURCE: Returns and sets Application data and settings */
RESOURCE(appdata, METHOD_POST | METHOD_GET, "appdata");
void appdata_handler(REQUEST* request, RESPONSE* response)

/* PhenomenonList RESOURCE: Generates JSON array of node's phenomenons */
RESOURCE(phenomenonList, METHOD_GET, "appdata/phenomenon");
void phenomenonList_handler(REQUEST* request, RESPONSE* response)

/* PhenomenonList RESOURCE: Generates JSON array of node's phenomenons data */
RESOURCE(phenomenonData, METHOD_GET, "appdata/phenomenon/*");
void phenomenonData_handler(REQUEST* request, RESPONSE* response)

/* Phenomenon RESOURCE: Temperature */
RESOURCE(phenomenonTemperature, METHOD_GET, "appdata/phenomenon/temperature");
void phenomenonTemperature_handler(REQUEST* request, RESPONSE* response)


/* Phenomenon RESOURCE: Temperature options */
RESOURCE(phenomenonTemperature_opt, METHOD_GET |
METHOD_POST, "appdata/phenomenon/temperature/opt");
void phenomenonTemperature_opt_handler(REQUEST* request, RESPONSE* response)

/* Phenomenon RESOURCE: Humidity */
RESOURCE(phenomenonHumidity, METHOD_GET, "appdata/phenomenon/humidity");
void phenomenonHumidity_handler(REQUEST* request, RESPONSE* response)

/* Phenomenon RESOURCE: Humidity options */
RESOURCE(phenomenonHumidity_opt, METHOD_GET, "appdata/phenomenon/humidity/opt");
void phenomenonHumidity_opt_handler(REQUEST* request, RESPONSE* response)

/* Phenomenon RESOURCE: Acceleration */
RESOURCE(phenomenonAcceleration, METHOD_GET, "appdata/phenomenon/acceleration");
void phenomenonAcceleration_handler(REQUEST* request, RESPONSE* response)

/* Phenomenon RESOURCE: Acceleration options */
RESOURCE(phenomenonAcceleration_opt, METHOD_GET,
"appdata/phenomenon/acceleration/opt");
void phenomenonAcceleration_opt_handler(REQUEST* request, RESPONSE* response)
```

B WSNSCC

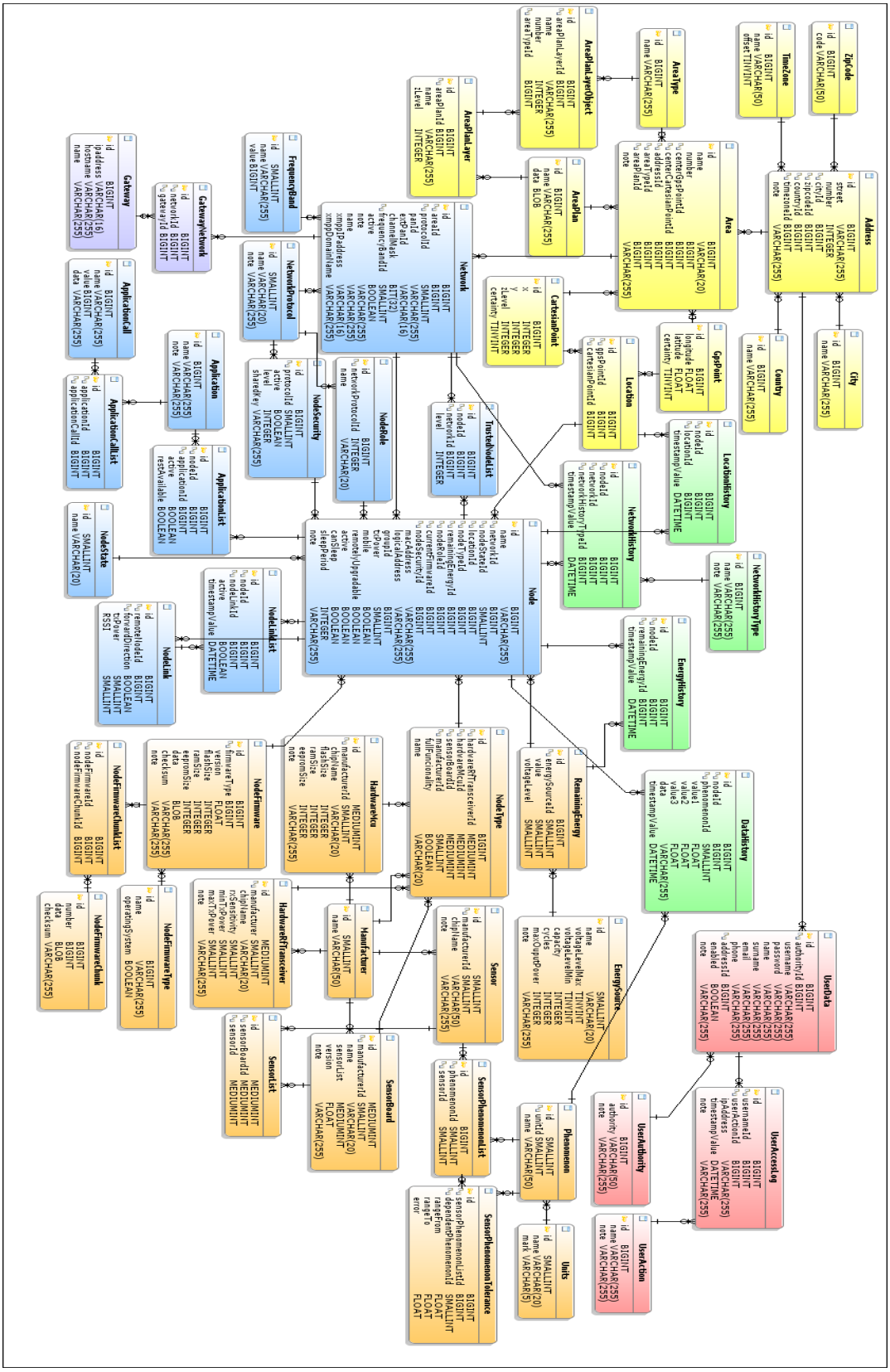
B.1 Třídy vrstvy Controllers - jejich atributy a metody



B.2 Soubor conf.xml pro uchovávání konfigurace

```
<Configuration>
        <Collecting>
                <Interval>00:10:00</Interval>
                <Cycles>5</Cycles>
        </Collecting>
        <Network>
                <Interface>tap0</Interface>
                <TargetNetwork>UTKO network</TargetNetwork>
                <Port>8080</Port>
        </Network>
        <Database>
                <Address>localhost</Address>
                <DatabaseName>AminaWSN</DatabaseName>
                <Username>root</Username>
                <Password>root</Password>
        </Database>
</Configuration>
```

C ERD DATABÁZE PRO SBĚR DAT



D CD PŘÍLOHA

Na přiloženém CD se nachází v adresáři ./node/sources/ zdrojové kódy realizující REST server pro uzel senzorové sítě společně se zdrojovými kódy Contiki verze 2.5RC1. V adresáři ./node/bin je zkompilovaný soubor firmware, který je určen pro AVR Raven. Zkompilovaná aplikace serveru pro sběr dat v senzorové síti (WSN-SCC) se nachází v adresáři ./server/bin/ a její zdrojové kódy společně s potřebnými knihovnami v ./server/sources/. Dump MySQL databáze je umístěn v /database/, pro vývoj aplikace byla použita MySQL verze 5.1.