VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

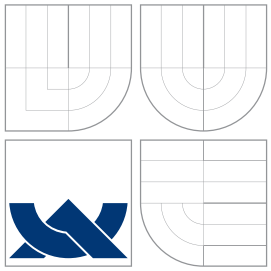
ZÁKLADNOVÁ STANICE PRO AGENTNÍ PLATFORMU WSAGENT S VYUŽITÍM GSM MODULU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

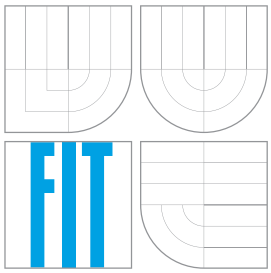
AUTOR PRÁCE
AUTHOR

Bc. JOSEF MOLÁK

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ZÁKLADNOVÁ STANICE PRO AGENTNÍ PLATFORMU WSAGENT S VYUŽITÍM GSM MODULU

BASESTATION FOR WSAGENT AGENT PLATFORM WITH USING GSM MODULE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JOSEF MOLÁK
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN HORÁČEK
SUPERVISOR

BRNO 2012

Abstrakt

Tato diplomová práce popisuje softwarové a hardwarové rozšíření multiagentní platformy WSageNt o možnost komunikace mezi základnovou stanicí senzorové sítě a řídícím webovým rozhraním pomocí sítě GSM. Jako základnová stanice je použit senzorový uzel FITmote a jako rozhraní do sítě GSM je použit GSM modem od firmy Teltonika. Fyzické propojení mezi těmito zařízeními tvoří nově navržený propojovací most, jež řeší různé napěťové úrovně jejich sériových rozhraní. Aplikace na základnové stanici byla rozšířena o sériovou komunikaci s GSM modemem. Toto rozšíření umožňuje komunikaci přes GSM síť do sítě Internet. Upravena byla také aplikace na straně řídícího serveru pro komunikaci přes TCP/IP.

Abstract

This master's thesis describes the software and hardware extension of multiagent platform WSageNt to ensure communication between basestation and web interface Control Panel using GSM network. Sensor node FITmote is used as a basestation and modem Teltonika is used as a interface to GSM. The physical interconnection of the devices implementing the newly created bridge to connect different logic voltage levels of their serial interface. Application for basestation was extended to serial communication with modem. This extend provide communication over GSM network to Internet. Application on control server also was modified for TCP/IP communication.

Klíčová slova

TinyOS, nesC, bezdrátová senzorová síť, GSM, FITmote, GPRS, WSageNt, základnová stanice

Keywords

TinyOS, nesC, wireless senzor network, GSM, FITmote, GPRS, WSageNt, basestation

Citace

Josef Molák: Základnová stanice pro agentní platformu WSageNt s využitím GSM modulu, diplomová práce, Brno, FIT VUT v Brně, 2012

Základnová stanice pro agentní platformu WSageNt s využitím GSM modulu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jana Horáčka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Josef Molák
23. května 2012

Poděkování

Chtěl bych velice poděkovat Ing. Janu Horáčkovi za vedení této diplomové práce. Především za jeho ochotu, připomínky a rady, které mě přivedli k jejímu zdárnému dokončení.

Obsah

Kapitola 1

Úvod

Pojem senzorové sítě se v poslední době rozšiřuje do celé řady oblastí lidské činnosti. Jediný senzor pro měření teploty, který by měl řídit například topení v budově není v současné době již sám schopen v takové míře zajistit šetrnost objektu ke zdrojům energie či tepelnou pohodu jako je to schopna zajistit senzorová síť. Ta totiž může kromě teploty brát při řízení topné soustavy v úvahu ku příkladu přítomnost osob, vlhkost vzduchu či rozdílnost tepelných ztrát různých částí domu. Navíc toto všechno je schopna provádět téměř bez jakéhokoli zásahu člověka. Do této oblasti se započítávají v dnešní době stále více zmiňované tzv. inteligentní budovy. Ty by měli zajišťovat pomocí senzorových sítí a agentů prakticky samostatný chod celé budovy. Podobně je tomu také u senzorových sítí starajících se o podporu řízení, bezpečnosti a provozu letadel (*EFCS*[1]). V dnešní době by prakticky nebylo bez této techniky možné se supermoderními dopravními či vojenskými letouny vůbec létat. Jen těžko by bylo možné provádět při dnešních rychlostech a velikostech letadel například čistě ruční ovládání klapek náběžných a odtokových hran křídel či spojlerů. Senzorové uzly se ve všech těchto systémech starají o dosažení společného cíle celé sítě. Toho dosáhnou vzájemnou komunikací, učením a výpočetním výkonem jednotlivých uzlů sítě.

Protože se poslední dobou stále více rozvíjí různá mobilní zařízení, zvyšují se i požadavky na mobilitu senzorových sítí. Chceme například pozorovat místa, kde není vybudována žádná infrastruktura či chceme být schopni rychle síť přesunout. Nebo ji rozmístit pouhým shozem z letadla do nějakého nehostinného místa. Proto postupně vznikly bezdrátové senzorové sítě, které by tyto požadavky měli splňovat. S tímto však přichází celá řada problémů. Jedním z nich je požadavek na nízkou energetickou náročnost uzlů sítě. Dalším problémem je přenos získaných dat na vzdálenější pozorovací či řídící centrum. Proto je zapotřebí mít v síti jeden centrální prvek (basestation[2]), který bude získaná data ze všech uzlů distribuovat například na počítač pro pozorování stavu sítě. Tím se zatíží větší měrou pouze jeden prvek dané sítě.

Vzhledem k tomu, že je v dnešní době penetrace[3] *GSM* sítě většiny civilizovaných zemí na vysoké úrovni, je výhodné použít pro spojení základnové stanice se vzdáleným centrem datový přenos přes síť *GSM*. Zároveň je možné s použitím dostupné *GSM* techniky docílit poměrně nízké spotřeby celé základnové stanice. Před zhotovením této práce bylo pro propojení základnové stanice a řídícího počítače využíváno spojení pomocí sériového rozhraní, což právě přílišnou mobilitu sítě vůči řídícímu počítači neumožňovalo. Nově navrhované a implementované řešení má daný problém vyřešit.

[1] Electronic Flight Control System - systém starající se o bezpečné řízení letadla.
[2] Základnová stanice.
[3] Dostupnost vzhledem k území

V druhé kapitole je obecně vysvětlena problematika senzorových sítí, jejich architektura a komunikační protokol *ZigBee*. Dále je zde probráno nastavení prostředí v operačním systému Linux pro práci s *TinyOS* a samotný systém *TinyOS*, programování senzorového uzlu *FITmote* a také základní práce s programovacím jazykem *nesC*. V kapitole je popsáno i zprovoznění všech potřebných součástí, jež jsou nutné pro běh aplikací na řídící počítačové stanici.

Třetí kapitola podrobně popisuje multiagentní platformu *WSageNt* [12]. Také se věnuje základní charakteristice agentního jazyka *ALLL*, jeho sémantice a následnému použití v senzorových sítích.

Další kapitola pojednává o technologii *GSM*. Modemu, který bude použit jako součást senzorového uzlu pro konstrukci základnové stanice. Popisují se tady také příkazy pro ovládání modemu a přenos dat pomocí *GPRS*. V neposlední řadě je zde také popsán problém nevyžádaných zpráv, jež přicházejí od modemu.

V páté kapitole je proveden návrh komunikace a fyzického propojení senzorového uzlu s *GSM* modemem. Stejně tak je zde proveden návrh komunikace mezi modemem a webovým rozhraním dříve zmíněné agentní platformy.

Předposlední část práce je věnována fyzickému provedení desky plošného spoje pro komunikace mezi uzlem *FITmote* a *GSM* modemem. Také je zde popsána implementace výsledné komunikace ze senzorové sítě až po webové rozhraní *Control Panel*.

V kapitole popisující testování výsledného systému je popsán průběh při průchodu agentů přes celý systém, získání výsledných dat při použití různých typů senzorů připojených k různým druhům senzorových uzlů.

Kapitola 2

Bezdrátové senzorové sítě

Tato kapitola popisuje základní principy senzorových sítí, jejich strukturu, programování, operační sytém *TinyOS* a také charakterizuje senzorový uzel *FITmote*. Jsou zde také uvedeny základní konstrukce programovacího jazyka *nesC*, jež se hojně používá právě pro programování senzorových uzlů.

Bezdrátová senzorová síť je uskupení určitého počtu uzlů (*mote*) tvořených z mikrokontroléru s bezdrátovým komunikačním rozhraním a připojenými senzory. Uzly spolu musí být schopny, určitým způsobem, přes bezdrátová rozhraní komunikovat. Mikrokontrolér zajišťuje fyzickou logiku uzlu. Díky ní je schopen snímat stav ze svých senzorů, předvídat budoucnost a reagovat na dané situace.

2.1 Architektura

Původně byly senzorové sítě koncipovány jako samostatné senzory připojené pomocí pevného drátového spojení s centrálním uzlem, který se staral o sběr dat z jednotlivých senzorů a podle vyhodnocení výsledků prováděl různé činnosti. Takový model dnes představují například průmyslové počítače s připojenými zařízeními a senzory ve větších výrobních procesech.

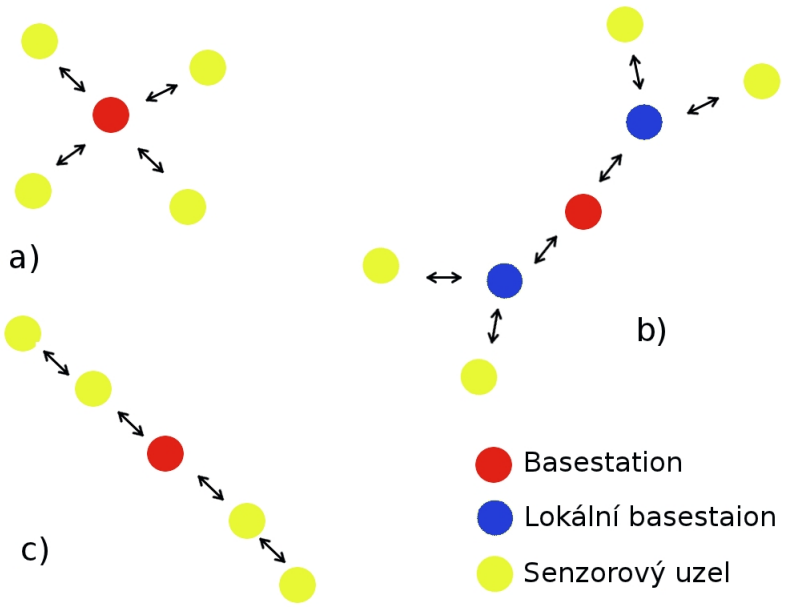
V dnešní době jsou senzorové sítě ve většině případů koncipovány pomocí mnoha uzlů představující multiagentní systém. Tyto uzly společně komunikují a tím si vytváří obraz sledované oblasti (dochází k učení agentů). Získaná data jsou poté určtým způsobem předána na hlavní uzel sítě (základnovou stanici) a odtud putují na řídící a monitorovací stanoviště, kde se s nimi dále pracuje. Komunikace může probíhat pomocí zvlášť vytvořené kabelové infrastruktury nebo může využívat infrastrukturu, jež je primárně určena pro jiné účely jako například Ethernet. Stále více se však rozšiřují sítě, u nichž probíhá komunikace ve speciálních bezdrátových standardech jako jsou *Bluetooth* nebo *Zigbee*, jež je podrobněji popsána v kapitole 2.1.2.

2.1.1 Základní topologie

Podle toho jak probíhá v senzorové síti komunikace mezi jednotlivými uzly a základnovou stanicí je možné rozlišovat několik základních topologií. V případě, že senzorové uzly napřímo komunikují se základnovou stanicí a obráceně, také ona komunikuje s jednotlivými uzly, jedná se o topologii hvězda. Přičemž základnová stanice vlastní konektivitu na zařízení, které se stará o sběr a správu dat získaných s celé sítě.

V senzorových sítích se setkáváme i s jinými topologiemi. Dalším typem topologie senzorových sítí může být síť uspořádaná do stromu. Zde se v každé menší oblasti s několika senzorovými uzly nachází menší základnová stanice, která sbírá data od podřízených uzlů. Sama komunikuje s hlavní základnovou stanicí, která se připojuje na centrální stanoviště. Tím se sníží energetická náročnost jednotlivých senzorových uzlů, které nemusí provádět komunikace na tak velké vzdálenosti. Lokální základnové stanice mají pak ve většině případů zajištěno napájení ze zdroje s větší kapacitou, nebo jsou napájeny přímo z elektrické sítě.

Jinou topologií může být model mesh[1] síť s hlavní základnovou stanicí, která je umístěna záměrně téměř ve středu monitorované oblasti a kolem ní mohou být rovnoměrně rozmístěny senzorové uzly. Každý uzel pak zná svého nejbližšího souseda směrem k základnové stanici. Pak pokud probíhá komunikace směrem od uzlu k základnové stanici, pošle daný uzel data na sousední uzel blíž ke stanici. Takto se postup opakuje dokud se zpráva nebo agent nedostane až na základnovou stanici. V obráceném směru je postup komunikace podobný. Toto řešení má menší nároky na spotřebu energie jednotlivých uzlů, díky přenosu dat na menší vzdálenosti. Ale naopak narůstá zahlcení komunikační sítě daty. Dalším problémem je zjištění polohy uzlů vůči stanici, k tomu se používají speciální poměrně výpočetně náročné algoritmy.



Obrázek 2.1: Typy topologií bezdrátových senzorových sítí: a) Hvězdicová topologie. b) Stromová topologie. c) Mesh topologie.

Jednotlivé uzly senzorových sítí jsou obvykle tvořeny řídícími, komunikačními, senzorovými a akčními prvky. Řídící prvky představují mikroprocesory a programovatelné části obvodů, které zajišťují aplikační logiku daného uzlu. Tyto prvky umožňují již na samotném uzlu jednoduché vyhodnocení získaných dat, zasílání agentů a samotné učení agentů a také ovládání případných akčních prvků uzlu vzhledem k dané situaci. Akčními prvky mohou být různá elektromechanická zařízení jako servomotory, relé a podobně. Komunikační část představuje rozhraní a část starající se o komunikaci mezi dalšími uzly či nějakým centra-

[1] Typ sítě, kde probíhá komunikace každý s každým.

lizovaným zařízením pro pozorování a úpravu řízení senzorové sítě. Komunikace může být prováděna pomocí kabelového spojení nebo pomocí bezdrátové komunikace. Zda je užito první či druhé možnosti záleží na okolnostech použití a prostředí, kde má být síť nasazena. Jak již bylo popsáno v úvodu výhodou bezdrátových sítí je především jednoduché rozmístění, použití na místech bez infrastruktury a jednodušší skrytí před odhalením. Naopak sítě propojené pomocí kabeláže je jednodušší zásobovat energií, nemají takové problémy se ztrátovostí dat při přenosu a komunikace probíhá zpravidla rychleji. Poslední avšak neméně podstatnou součástí senzorových uzlů, jak již z jeho názvu vyplývá, jsou senzory. Pokud akční prvky uzlu představují jeho ruce, řídící prvky mozek, pak senzory jsou jeho očima. Starají se o zaznamenání změn v daném okolí. Právě v reakci na tyto změny provádí uzel akce a komunikuje se zbytkem celé sítě.

2.1.2 ZigBee

ZigBee je standardem v bezdrátové komunikaci podle IEEE 802.15.4 určený především pro použití v aplikacích, kde není hlavním měřítkem přenosová rychlost a dosah, ale především malé rozměry, nízká cena a spotřeba energie. Podle specifikace může pracovat ve třech různých zkoordinovaných pásmech a to pro Ameriku a Austrálii v pásmu 915 MHz s 10 kanály a teoretickou rychlostí 40 kb/s, pro Evropu v pásmu 868 MHz s jedním kanálem a teoretickou rychlostí 20 kb/s. Pro globální využití je vyhrazeno pásmo 2,4 GHz na 16 kanálech a rychlostí 250 kb/s. Pro přístup k médiu je zde využito přístupové metody *CSMA/CA*[2]. Tato metoda zabraňuje kolizím tak, že každá stanice, jež chce vysílat vyšle před počátkem vysílání krátký paket RTS (příprava k vysílání). Pokud příjemce tuto zprávu přijme odpoví na ni pomocí paketu CTS (pásmo je volné). Poté může odesílatel začít vysílat data. Po přijetí všech dat příjemcem dojde odeslání paketu ACK (potvrzení) odesílateli. Ostatní účastníci sítě tyto pakety zaznamenávají a podle toho jsou schopni rozpoznat, kdy mohou začít vysílat [3].

Topologie používané v sítích podle *ZigBee* jsou shodné s topologiemi popsanými v obecných senzorových sítích výše. Pro adresování se používá binárních čísel o délce buď 64 bitů pro větší sítě nebo 16 bitů pro sítě menších rozsahů. Adresování je dále rozšířeno o dalších 16 bitů pro určení ID celé sítě. To umožňuje provozovat na jednom místě více nezávislých sítí podle standardu IEEE 802.15.4. Celou síť vždy řídí centrální prvek. Ten zajišťuje mimo jiné synchronizaci celé sítě pomocí speciálních datových paketů. Ty slouží například k přepínání komunikačního rozhraní jednotlivých uzlů sítě na požadovanou dobu do režimu spánku, čímž se právě velkou měrou snižuje spotřeba jednotlivých uzlů. Po tuto dobu může uzel shromažďovat data, provádět měření a podobně. Poté co se probudí ke komunikaci, může data například odeslat na řídící prvek sítě.

2.1.3 Bezdrátový senzorový uzel FITmote

Tento senzorový uzel byl vytvořen na FIT VUT v Brně. Jeho základem je bezdrátový modul pro síť *ZigBee*[3] od firmy Atmel s označením *ATZB-24-A2*. Podle [2] je jádrem modulu procesor *ATmega1281*, k němuž je přes *SPI* sběrnici připojen radiový modul *AT86RF230* s dvojitou čipovou anténou.

Díky obsahu procesoru *ATmega1281* má modul spoustu dobrých vlastností hodících se právě pro použití v bezdrátových senzorových sítích. Tou hlavní je především malý nárok

[2] Carrier Sense Multiple Access with Collision Avoidance.
[3] Bezdrátová síť vycházející ze standardu IEEE 802.15.4.

Obrázek 2.2: Bezdrátový modul ATZB-24-A2 pro ZigBee.

na napájení. Pracovní rozsah napájení se pohybuje v rozmezí 1,8 až 3,6 V. Proto je pro celý senzorový uzel použito knoflíkové napájecí baterie s napájecím napětím 3 V. Tato baterie je výhodná z důvodů jejich malých rozměrů při umístění na desku plošného spoje, ale nedosahuje takových kapacit jako by měly například dvě tužkové baterie typu AA, které jsou použity například u platforem *MICAz* či *Iris*. Ale pro dané použití je i tato kapacita dostačující na dlouhodobý provoz, jelikož spotřeba modulu se pohybuje okolo 19 mA při vysílání a v režimu spánku okolo 6 μA. Pouze je zapotřebí uvažovat pro připojované periferie, že nedosáhneme takových úrovní napětí a proudu jako je tomu u klasických zapojení. Procesor má 8 KB RAM, 128 KB paměti flash a jeho pracovní frekvence jsou 4 MHz. Mezi další důležité vybavení procesoru patří rozhraní *UART* pro komunikaci s dalšími zařízeními a *JTAG*[4] a *SPI* pro programování. Dále je vybaven 4-bitovým ADC převodníkem (až 9 bitovým při zakázání *JTAG*).

Rádiový modul je konstruován pro frekvence v pásmu 2,4 GHz, přičemž je schopen pracovat až na šestnácti různých kanálech. Odpovídá, jak již bylo výše zmíněno, standardu *IEEE 802.15.4* a *ZigBee*. Jeho přenosová rychlost je pak 250 kbps. A výkon vysílače se pohybuje od -17 do + 3 dBm a impedance je pro vyvážený výstup 100 Ω.

Modul *ATZB-24-A2* je osazen na desce plošného spoje, kde je pro programování vyvedeno rozhraní *JTAG* jako boční sběrnice. Dále se na desce nachází, v podobě jednotlivých pinů, dobře přístupné rozhraní *UART* pro připojení různých zařízení. Jako periferie jsou na desce připojeny na piny procesoru tři LED diody a senzor pro měření teploty. Ze zadní strany plošného spoje je připevněn držák pro vložení knoflíkové baterie pro napájení. Toto kompaktní zařízení tedy tvoří bezdrátový senzorový uzel *FITmote*. Samozřejmě pro něj platí stejná pravidla jako pro uzly *Iris* či *MICAz* a to, že na něm může běžet i kód pro základnovou stanici, který je pro tuto diplomovou práci zásadní. Bez jakýchkoli úprav zdrojového kódu, by měla aplikace, jež funguje na jedné ze dvou výše uvedených platforem, běžet vzhledem k shodnému procesoru a celé konfiguraci obvodu i na senzorovém uzlu *FITmote*. Oproti ostatním modulům jako jsou předchozí dva zmíněné, však tento modul vyniká svojí nízkou cenou, pokud se tedy zanedbá práce určená pro návrh desky plošného spoje a její osazení. Za zmínku stojí také velice malé rozměry.

[4] Joint Test Actoin Group, ale dnes vžit tento název jako speciální sériový protokol pro přístup, testování a programování obvodů.

2.2 TinyOS

TinyOS je podle [16] velice nenáročný operační systém, určený díky svým parametrům pro aplikace, kde je hlavním kritériem nízká spotřeba energie. Jeho další významnou vlastností je malá náročnost na paměť. Ze zdroje [8] je patrné, že pro paměť programu i dat může postačovat pouhých 400 bytů. Primárně je tedy určen pro malé nízkoenergeticky náročné mikroprocesory. Systém podporuje celou řadu prostředků, pro efektivní práci se senzory, komunikaci po síti, ukládání do paměti, práci s časovači a uspávání nepotřebných procesů. Je naprogramován v jazyce *nesC*, popsaném v kapitole 2.2.4.

TinyOS je založen na komponentním přístupu vzhledem k použitým architekturám. Každá komponenta může být buď softwarově fyzicky napsaná knihovna nebo pouze jakási nástavba nad konkrétním hardwarem typická pro danou architekturu. Proto je pak možné při programování různých architektur postupovat totožně i přes jejich rozdílnou vnitřní stavbu. A navíc také zbytečně nezatěžovat mikroprocesor funkcemi, které nebudeme v naší aplikaci potřebovat.

Dále je pro tento systém typické speciální plánování funkcí pro komunikaci mezi komponentami v podobě příkazů (*commands*), úkolů (*tasks*) a událostí (*events*). V jeden okamžik smí v systému běžet vždy pouze jeden příkaz, událost či jeden úkol. Přičemž ani jeden z nich není v jejich běhu až dokonce přerušen. Výjimku tvoří pouze naplánování úkolu na obsluhu nějakého přerušení, které je samo o sobě událostí s nejvyšší prioritou. Úkoly se plánují při každém jejich zavolání v komponentě do plánovače úloh. A až přijde jejich naplánovaný čas, tak se provedou. Na rozdíl od nich je u událostí možné, že je před vykonáním předběhne jiná událost či úkol. Vzhledem k tomu, že každý úkol se provede vždy až dokonce, je nutné jednotlivé úkoly psát krátké. Tím se zamezí zpoždění u důležitějších úkolů, protože dostanou příležitost na svůj běh.

Poslední zvláštností *TinyOS* jsou tzv. split-phase operace. Ty představují rozhraní mezi dvěma komponentami. Když se v jedné komponentě zavolá příkazem druhá, tak se zpětně v té první signalizuje dokončení operace pomocí události.

Právě díky vlastnostem popsaným výše je jasně patrné, že se *TinyOS* hodí pro použití v uzlech bezdrátových senzorových sítí. Zde je totiž většinou hlavním požadavkem bezúdržbový běh senzorového uzlu s některou nízkokapacitní baterií v řádu let. Toto je možné zajistit kromě speciálního hardwaru právě nízkou energetickou náročností samotného operačního systému.

2.2.1 Příprava prostředí v OS Linux

Pro nahrávání programů na senzorové uzly *FITmote* je zapotřebí nejprve řádně připravit prostředí v operačním systému. Pro implementaci zadání byl zvolen operační systém Linux, konkrétně distribuce Kubuntu. V tomto systému je zapotřebí pro správné fungování *TinyOS* učinit několik dílčích kroků popsaných v této podkapitole. Při popisu je částečně vycházeno z [13]. Nejprve je nutné přidat do systému nový zdroj programových balíčků pro stažení a instalaci všech knihoven na práci s *TinyOS*. Po vlastním testování několika zdrojů se mi pro programování senzorových uzlů *FITmote* osvědčil zdroj pro operační systémy Kubuntu s kódovým označením Lucid. Samotné přidání tohoto zdroje je možné provést otevřením souboru sources.list v konzoli pomocí příkazu[5]:

```
sudo gedit /etc/apt/sources.list
```

[5] Místo programu Gedit je možné použít jakéhokoli jiného textového editoru jako KWrite či Kate.

Na konec tohoto souboru je následně zapotřebí přidat tento řádek, rozšiřující zdroje software pro daný operační systém:

```
deb http://tinyos.stanford.edu/tinyos/dists/ubuntu lucid main
```

Dále je nutné provést aktualizaci zdrojů softwaru a samotnou instalaci všech knihoven potřebných pro překlad a práci s *TinyOS*. Mezi tyto úkony patří i instalace a nastavení prostředí pro běh a programování aplikací v jazyce *JAVA*, které jsou v *TinyOS* používány podle [11] pro komunikaci se základnovou stanicí. Nastavení a instalace se provede příkazy:

```
sudo apt-get update
```

```
sudo apt-get install tinyos-2.1.1 sun-java6-jdk sun-java6-jre sun-java6-
plugin
```

```
sudo update-java-alternatives -s java-6-sun
```

Poté se musí upravit prostředí pro bash, aby po jeho spuštění bylo možné provádět překlady programů v *nesC* a také samotné programování uzlů senzorové sítě pomocí různých programátorů. To se provede úpravou souboru */home/<uživatel>/.bashrc* pomocí:

```
gedit /home/<uživatel>/.bashrc
```

Pak se přidá do tohoto souboru, jako nový zdroj pro nastavení, cesta ke skriptu */opt/tinyos-2.1.1/tinyos.sh*. Do souboru .bashrc se tedy přidá řádek:

```
source /opt/tinyos-2.1.1/tinyos.sh
```

Protože je po instalaci balíčků implicitně nastavena cesta pro Javu chybně, je už jen zapotřebí upravit cestu k Javě ve výše zmíněném souboru *tinyos.sh*. A to otevřením daného souboru v konzoli takto[6]:

```
sudo gedit /opt/tinyos-2.1.1/tinyos.sh
```

A následnou změnou řádku s proměnnou *CLASSPATH* na:

```
CLASSPATH=$TOSROOT/support/sdk/java/:$TOSROOT/support/sdk/java/tinyos.jar:.
:$CLASSPATH
```

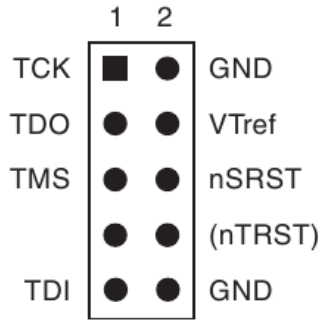
2.2.2 Programátor AVR JTAGICE mkII

Pro programování byl vybrán programátor *JTAGICE mkII* s rozhraním *JTAG* od firmy AVR. Tento programátor umožňuje provádět ladění softwaru přímo na fyzickém čipu, což nám ulehčuje vývoj aplikací. Tuto funkcionalitu umožňuje rozhraní *On-Chip* debug, jež se stará o komunikaci mezi programem v počítači a běžící aplikací na procesoru. Dále ho lze také použít pro programování přes *ISP* rozhraní, jež používají procesory řady ATtiny. Obrovská výhoda tohoto zařízení je také možnost připojení k počítači pomocí dvou nejrozšířenějších rozhraní a to jak *USB* tak i *RS-232*.

[6] Viz. strana 9 tamtéž.

Snad jedinou nevýhodou tohoto programátoru je, že veškeré postupy a nastavení jsou ze strany výrobce popsány pro operační systém Windows s nainstalovaným programem AVR studio. Proto bylo nutné všechny postupy a nastavení provést pro zvolený operační systém Linux. To je popsáno v následujících podkapitolách.

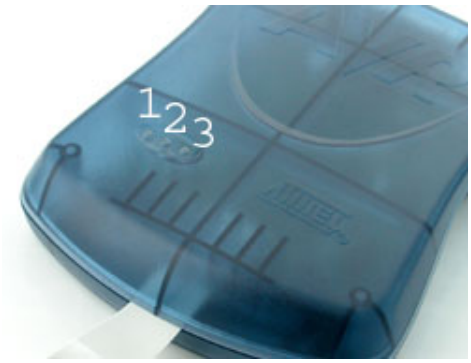
Programátor obsahuje pro připojení k procesoru přes *JTAG* kabel zakončený konektorem *JTAG10PIN* zobrazeným níže viz. 2.3.



Obrázek 2.3: Konektor JTAG10PIN na AVR JTAGICE mkII [1].

Tento konektor je připojen na speciální desku plošného spoje, kde je upraven na malou n-bitovou sběrnici, jež je připojitelná na bezdrátový senzorový uzel *FITmote*.

Programátor obsahuje také 3 LED diody pro indikaci jeho stavu. Vyobrazení LED na programátoru je znázorněno na obrázku 2.4.



Obrázek 2.4: Indikační LED na programátoru [4].

Zelená dioda číslo 1 značí, že je programátor připojen na zdroj napájení ať už externí či přes rozhraní do PC. Červená dioda číslo 2 značí fungování rozhraní *JTAG*. Dioda číslo 3 při zelené barvě značí přenos dat do procesoru. Při červené, že není programátor připojen k řídícímu programu v PC, a když nesvítí a zároveň svítí dioda číslo 2, programátor je připojen pouze k řídícímu programu v PC.

2.2.3 Programování senzorového uzlu FITmote

Po instalaci všech balíčků *TinyOS* nemáme stále zajištěnou funkčnost programování senzorového uzlu *FITmote* přes rozhraní *JTAG* pomocí příkazu make s přepínači pro programování.

Nejprve je nutné do složky */opt/tinyos-2.1.1/support/make/* nahrát soubor *fitmote.target* poskytnutý Fakultou informačních technologií Vysokého učení technického v Brně (dále jen FIT VUT v Brně).

Pro správné nahrávání programu na senzorový uzel přes rozhraní *JTAG* pomocí programátoru *AVR JTAGICE mkII*, zmíněného v dřívější kapitole, chybí totiž v základu operačního systému *TinyOS* skript pro program avrdude[7]. Pro pohodlí a stejný přístup k programování pro různé platformy rozhraní jako *mib510* použitého u platforem *MICAz* či *IRIS* je zapotřebí tento skript vytvořit.

V adresáři */opt/tinyos-2.1.1/support/make/avr/* jsou pomocné skripty pro programování přes různá rozhraní procesorů AVR a tím pádem i k nim určených programátorů. Zde je nutné vytvořit nový skript pro rozhraní senzorového uzlu *FITmote*. Protože jde u našeho senzorového uzlu a tedy i programátoru o rozhraní *JTAG2* vytvoříme soubor *jtag2.extra*. Při vytváření skriptu je možné vycházet z jiných skriptů v adresáři avr například pro isp mkII rozhraní. Základem našeho nového souboru je totiž především příkaz pro *avrdude2*. A ten se pro různá rozhraní mění jen kosmeticky.

Pokud tedy budeme vycházet ze souboru k isp rozhraní *avrispmkii.extra* přejmenujeme proměnnou `AVRISPMKII` proměnnou `JTAG2`, proměnou `PROGRAM` nastavíme na hodnotu `jtag2` a nakonec je potřeba pro *JTAG2* rozhraní nastavit proměnnou `PROGRAMMER_FLAGS` určující přepínače pro *avrdude2*. Nově bude tato proměnná v takovémto tvaru:

```
PROGRAMMER_FLAGS= -c jtag2 -P $(JTAG2) -U hfuse:w:$(AVR_FUSE_H):m -U
lfuse:w:$(AVR_FUSE_L):m $(PROGRAMMER_PART) $(PROGRAMMER_EXTRA_FLAGS)
```

Přičemž za přepínačem *-c* je určeno rozhraní pro avrdude v našem případě *jtag2*, za *-P* rozhraní PC s připojeným programátorem a za *-U* horní a dolní konfigurační slovo, které je závislé na typu procesoru a nastavuje se pomocí vnitřních proměnných `AVR_FUSE_H` a `AVR_FUSE_L`. Ty jsou inicializovány ve složce */opt/tinyos-2.1.1/support/make/* pro *FITmote* konkrétně v souboru *fitmote.target*. Podle [21] určují například zvolený procesor, oscilátor procesoru a další neměnná nastavení za běhu.

V neposlední řadě je zapotřebí do adresáře */opt/tinyos-2.1.1/tos/platforms/* nahrát složku *fitmote* získanou opět z FIT VUT v Brně. Tato složka obsahuje zdrojové kódy aplikací napsaných v jazyce *nesC*, jež jsou typické právě pro daný senzorový uzel. Tyto aplikace představují ovladače pro jednotlivé periferie a hardware uzlu *FITmote*, jež se liší od obecných ovladačů jiných senzorových uzlů používaných v *TinyOS*.

Po provedení všech nastavení výše je možné naprogramovat senzorový uzel v místě, kde se nachází kód a makefile dané aplikace pomocí příkazu:

```
sudo chmod a+rw /dev/bus/usb

make fitmote install,<id uzlu> jtag2,usb  // pro základnovou stanici
                                          // je id rovno jedné
```

Prvním příkazem nastavíme práva pro přístup ke sběrnici *USB* pro připojení programátoru. Druhý příkaz slouží pro překlad a naprogramování senzorového uzlu. Parametr `fitmote` představuje typ senzorového uzlu, který je programován. Podle něj se vhodně nastaví konstanty, pro správný běh programu na uzlu. Dále `<id uzlu>` označuje, které číslo

[7] Avrdude je program sloužící pro programování přes různé druhy programátorů z konsole operačního systému.

bude uzlu přiřazeno jako adresa pro komunikaci. Další parametr `jtag2` značí typ programátoru. Toto nastavení je nutné pro správné řízení programátoru programem *avrdude* při nahrávání. A v neposlední řadě pomocí `usb` je zvolena sběrnice počítače, kam je připojen programátor.

Makefile obsahuje pouze název aplikace (hlavní komponenty), připojení správných pravidel, popřípadě inicializaci speciálních globálních proměnných určujících například číslo skupiny, ve které má uzel komunikovat či jeho ID. Makefile má tedy obvykle tvar:

```
COMPONENT=<název aplikace>
include $(MAKERULES)
```

2.2.4 Jazyk nesC

Jazyk *nesC* [9] je rozšířením jazyka *C*, jež byl přímo navržen pro programování aplikací pro operační systém *TinyOS*, který je sám v tomto jazyce napsán. *NesC* je statickým jazykem, nepoužívá se zde dynamické alokování paměti a tedy i call-graph je známý již v době překladu.

Jeho hlavním rysem je rozdělení celé aplikace do jednotlivých komponent, přičemž každá komponenta je dále členěna na dvě části. Jsou jimi její specifikace, která představuje jména instancí jejího obousměrného rozhraní a druhá část pro funkčnost celé komponenty. Komponenty v sobě zapouzdřují úkoly, příkazy a obsluhy událostí [8].

Komponenta se v jazyce *nesC* zapisuje jako fráze *module* za níž následuje její název a ve svorkách je uveden výčet rozhraní. Dále se může vyskytovat implementační část uvedená klíčovým slovem *implementation*. Za ní se nachází oblast ohraničená dalšími svorkami, kde se může nacházet aplikační kód implementující jedno nebo více rozhraní. Struktura kódu komponenty v jazyce *nesC* může tedy vypadat nějak takto:

```
module /*JMÉNO*/ {
  provides {
    /*SEZNAM PROVIDES INTERFACE*/
  }

  uses {
    /*SEZNAM USES INTERFACE*/
  }
}
implementation {

  /*IMPLEMENTAČNÍ KÓD*/

  return SUCCESS;
}
```

Zvláštním případem komponent jsou tzv. konfigurace uvozených frází *configugurations*. Konfigurace slouží k propojení mezi komponentami. Zde se provede fyzické napojení rozhraní jedné komponenty na druhou. Za klíčovým slovem následuje, podobně jak je tomu u komponent, seznam rozhraní a implementační část, kde se uvede seznam použitých komponent uvozených slovem *components*. Poté se přiřadí pomocí operátoru `->` vstupní rozhraní

jedné komponenty výstupnímu rozhraní druhé komponenty. Konfigurace má tedy takovouto strukturu:

```
configuration /*JMÉNO*/ {
  provides {
    /*SEZNAM PROVIDES INTERFACE*/
  }

  uses {
    /*SEZNAM USES INTERFACE*/
  }
}
implementation {
components /*SEZNAM KOMPONENT*/;

/*VSTUPNÍ ROZHRANÍ -> VÝSTUPNÍ ROZHRANÍ*/;
}
```

Rozhraní mohou být dvou základních typů. Prvním typem jsou interface typu *provides*, které představují rozhraní ve směru od dané komponenty k jiné (k poskytnutí). Typ *uses* pak obráceně slouží pro směr k dané komponentě (k využití). Samotná rozhraní je možné zapsat v jazyce *nesC* v prvním případě přímo pomocí výčtu jednotlivých deklarací funkcí:

```
module /*JMÉNO*/ {
  provides command /*DATOVÝ TYP*/ /*JMÉNO FUNKCE*/(/*PARAMETRY FUNKCE*/);
  uses command /*DATOVÝ TYP*/ /*JMÉNO FUNKCE*/(/*PARAMETRY FUNKCE*/);
}
```

Druhou možností je pak vytvoření samotného souboru rozhraní pod označením *interface*, který v sobě zapouzdřuje skupinu většinou souvisejících funkcí. Blok interface může opět obsahovat příkazy či události. Pak takový blok rozhraní je možné použít v komponentě následovně:

```
interface /*JMÉNO*/ {
  command /*DATOVÝ TYP*/ /*JMÉNO FUNKCE*/(/*PARAMETRY FUNKCE*/);
  event /*DATOVÝ TYP*/ /*JMÉNO FUNKCE*/(/*PARAMETRY FUNKCE*/);
}
```

```
module /*JMÉNO*/ {
  provides interface /*JMÉNO*/;
  uses interface /*JMÉNO*/;
}
```

Kapitola 3

Multiagentní platforma WSageNt

Tato multiagentní platforma slouží pro správu, sledování a jednoduchou práci s uzly bezdrátové senzorové sítě. Byla navržena a je nadále vyvíjena na FIT VUT v Brně [12]. V této kapitole bude proveden základní popis jednotlivých částí celé platformy, vycházející především z diplomových prací [20], [7] a [11].

První částí popisu této platformy je v následující podkapitole seznámení s agentním jazykem *ALLL*, jež je na jednotlivých senzorových uzlech interpretován a upravuje tak funkčnost uzlu za běhu hostitelské aplikace. Tím je umožněn běh agentů v senzorové síti, jejich přenos z uzlu na uzel a změna funkčnosti jednotlivých uzlů podle potřeby. Dále bude popsána aplikace pro mobilní agenty nainstalovaná na každém obecném uzlu senzorové sítě, která právě představuje onu hostitelskou aplikaci tvořící základ celé platformy.

Další částí platformy určené k rozboru je upravený systémový modul na FIT VUT v Brně z *TinyOS* s názvem *BaseStation*, běžící na jednom z uzlů senzorové sítě a představující základnovou stanici. Podkapitola popisující jeho význam se zaměřuje hlavně na primární funci směrovovače dat z řídícího počítače na senzorovou síť a zpěz ze sítě na řídící počítač přes sériové rozhraní. Propojení je fyzicky provedeno pomocí speciálního hardwarového mostu. Ten zajišťuje vzájemný převod dat mezi rozhraním *UART* a *USB*. Pomocí *USB* je pak připojena základnová stanice k počítači.

V závěrečné části této kapitoly je uveden popis aplikací *BSComm* a *Control Panel* běžících na centrální počítačové stanici, určené k ovládání a pozorování celé sítě. *Control Panel* zde představuje grafické uživatelské rozhraní řídícího počítače a *BSComm* jeho aplikační logiku. Architektura celé platformy *WSageNt* je vidět na obrázku 3.1.

3.1 Agentní jazyk ALLL

Pokud by mělo být pro řízení funkcí senzorového uzlu použito pouze jazyka *nesC*, vykonával by senzorový uzel stále stejnou činnost, kterou získal po nahrátí přeloženého programu v daném jazyce do svojí paměti. Nebylo by tedy možné poté vzdáleně měnit funkci celého uzlu a tím umožnit flexibilitu funkčnosti celé sítě. Jazyk *nesC* totiž, jak je již uvedeno v předcházející kapitole, používá statickou alokaci paměti. Ta neumožňuje měnit průběh programu v tomto jazyce za běhu. Jelikož však platforma měla umožňovat podporu řízení, monitorování a především přenos agentů mezi jednotlivými uzly senzorové sítě, vznikl na FIT VUT v Brně jazyk *ALLL*. Ten zajistil v senzorové síti modifikovatelnost služeb poskytovaných jednotlivými uzly. A také přenos informací a dokonce i celých agentů v podobě kódu v jazyce *ALLL* mezi uzly sítě.

Obrázek 3.1: Architektura půdvodní platformy WSageNt.

3.1.1 Základní charakteristika jazyka ALLL

Jazyk *ALLL* patří mezi jazyky s nízkou úrovní abstrakce. Je to dáno tím, že je navržen právě pro běh agentů v bezdrátových senzorových sítích, které se zpravidla vyznačují nízkou energetickou náročností provozu na úkor hardwarové výkonnosti. Tento hardwarový výkon tedy poté schází pro interpretaci vyšší úrovně abstrakce agentního kódu v daném jazyce. Proto se pracuje pouze se základními strukturami a jednoduchými konstrukcemi.

Věta v jazyce *ALLL* se vytváří pomocí seznamu akcí, představující plán k jejich vykonání. Každá akce poté může být tvořena opět vnořeným seznamem akcí. V každé akci může být místo fyzické hodnoty symbol, jež zastupuje registr. Tento symbol je nahrazen, než započne vykonání akce, hodnotou daného registru. Registr tedy může být použit jako vstup do akce či jako jedna z částí libovolné n-tice. Strukturou se tedy jazyk *ALLL* velice podobá jazykům typu *Prolog* a *Lisp*. Navíc pokud se některá ze zanořených akcí plánu nepovede provést, je tato část smazána a pokračuje se vykonáváním části plánu o úroveň výš, než byla původní akce. Agent reprezentovaný pomocí tohoto jazyka obsahuje čtyři základní seznamy a tři pomocné registry pro uchování různých dat při výpočtech. Seznamy představují různé báze možných činností a provozních informací agenta. Ty jsou popsány dále podle [20].

První z nich je báze plánů označovaná jako `PlanBase`. Ta představuje seznam možných akcí pro vykonání, které musí být označeny vždy nějakým jménem. Podle tohoto jména je pak možné, danou akci volat znovu v některé ze zanořených částí báze plánů nebo například v samotném záměru agenta popsaném dále.

Záměr agenta, neboli `Plan`, přímo vyjadřuje, jakou činnost bude agent vykonávat. Jak bylo uvedeno v předchozím odstavci, může tento seznam akcí obsahovat mimo přímo uvedených výčtů akcí také jména jednotlivých seznamů akcí z báze plánů. Jednotlivé seznamy mohou být do sebe zanořovány, čímž vzniká hierarchická struktura výsledného programu. Pokud dojde k chybě v některé akci daného plánu, přejde se ve vykonávání agentního kódu do vyšší úrovně celé struktury.

Dalším seznamem určeným pro agenta v jazyce *ALLL* je báze znalostí `BeliefBase`. Tento seznam slouží jako uložiště dat potřebných pro výpočty apod. Může být naplněn před vysláním agenta nebo si ji vytváří agent sám během doby, kdy interaguje s okolím, odkud data získá. Toto představuje učení agenta. Formát dat, jež tuto bázi tvoří, má podobu n-tic

s danou délkou. Přítomnost vybrané n-tice lze ověřit pomocí speciální operace unifikace, která porovnává shodnost hledaného vzoru s n-ticemi v určeném rozsahu.

Poslední agentní bází je vstupní báze `InputBase`. Je podobná bázi znalostí, ale ukládají se do ní veškeré zprávy a naměřené hodnoty automaticky. Každý záznam je tvořen n-ticí se dvěma údaji. Prvním údajem n-tice je typ údaje, který určuje zda jde o příchozí zprávu nebo data získaná měřením popřípadě výpočtem. Druhá část nese právě fyzickou hodnotu těchto dat. Pokud chce agent některá z těchto dat získat, stačí mu je vybrat pouze pomocí výběru typu dat a do aktuálního registru se nahraje jejich hodnota. Pro jazyk *ALLL* přináší oddělení báze znalostí od vstupní báze možnost agentovi plně rozhodovat o tom, které z příchozích a naměřených hodnot zařadí do svojí báze znalostí a které ne, což vede k jeho větší samostatnosti a rozmanitosti.

3.1.2 Sémantika v jazyce ALLL

V jazyce *ALLL* se užívá především dvou základních struktur a těmi jsou seznam a tabulka. Pro tabulku je význačné, že nezáleží na pořadí jejích prvků. Těmi mohou být právě seznamy. U nichž na pořadí jejich jednotlivých položek záleží. Položkami tétu struktury jsou tedy uspořádané akce. Každý seznam je uzavřen z obou stran v klasických závorkách `(` a `)`. Jednotlivé akce nejsou oddělovány speciálním znakem, ale každá akce v popisovaném jazyce má speciální uvozovací znak. Podle něj je možné rozpoznat, kde začíná jedna a končí druhá akce. Uvozovací znaky s jejich významy jsou uvedeny v tabulce 3.1.

Uvozovací znak	**Vstupní parametry**	**Význam akce**
`+`	n-tice | registr	Přidání do `BeliefBase`, unifikací se kontroluje duplicitní vkládání
`-`	n-tice | registr	Odebrání položek z `BeliefBase` pomocí unifikace
`!`	číslo | registr n-tice | registr	Odeslání zprávy zadané n-ticí nebo registrem na mote se zadanou adresou
`&`	číslo	Změna aktivního registru
`*`	n-tice | registr	Test `BeliefBase` na zadanou n-tici nebo registr, výsledek se uloží do aktivního registru
`?`	číslo | registr | znak	Test `InputBase` na zprávu od mote/senzoru se zadanou adresou, výsledek se uloží do aktivního registru
`@`	seznam akcí	Přímé spuštění, akce se vloží na zásobník se zarážkou
`^`	jméno | registr	Nepřímé spuštění, hledá se plán v `PlanBase` se stejným jménem
`$`	písmeno (,n-tice | registr)	Volání služeb platformy, první parametr (písmeno) je kód operace, druhý parametr jsou parametry služby, nemusí být u všech služeb
`#`	nejsou	Zarážka za plánem, sémanticky tato akce nemá žádný význam

Tabulka 3.1: Uvozovací znaky akcí v jazyce ALLL podle [11].

Jednotlivé operace v jazyce *ALLL* tedy umožňují činosti jako ovládání připojených zařízení na senzorovém ulzu, čtení hodnot ze senzorů a přenost zpráv a celých agentů

skrz senzorovou síť. K vykonání těchto úkonů, se využívá několika základních kontrukcí popisovaného jazyka. Mezi ty nejzákladnější podle [20] patří:

- **Přidání a odebrání z báze znalostí** - při této operaci je nejdříve pomocí unifikace zkoumána daná tabulka, zda již vybranou n-tici nebo registr neobsahuje. Pokud ne data jsou uložena na začátek tabulky. Příkladem může být operace: `+(12,13)`, která přidá do tabulky se znalostmi tuto n-tici. Možné je taky použít například `+(14,&1)`, kde se přidá nejprve tato n-tice a poté dojde k nahrazení registru označeného `&1` za jeho skutečnou hodnotu. Podobně je tomu i pro odebírání, kde se však pomocí unifikace hledá vhodná n-tice pro smazání. Pokud bude v bázi znalostí například uloženo `(12)(13)(14)(15,16)`, pomocí akce `-(13)` se daná entice odebere. Je možné také použít pro mazání anonymních proměnných jako třeba `-(_)`, která odebere všechny jednoprvkové n-tice.

- **Odeslání a příjem zprávy z radiového modulu** - Při těchto akcích se využívá funkcí, které musí být naimplementovány na senzorovém uzlu. Jak bude popsáno dál, v patformě *WSageNt* jsou tyto funkce implementovány a tudíž je možné je v jazyce *ALLL* používat. Pro odeslání zprávy se tedy vytvoří n-tice, kde první prvek bude představovat adresu cílového uzlu a druhým může být i víceprvková n-tice, představující přímo data určená k odeslání jako například `!(1,(16,20))`. Při příjmutí zprávy se provede její uložení do vstupní báze. Proto pokud příjde například zpráva `(16,20)` od uzlu 2, uloží se do vstupní báze jako `(2,(16,20))`. Pro přečtení je tedy třeba provést její nalezení v dané bázi. Pro tento účel lze použít příkaz `?(2)`. Po němž se zpráva odebere z této báze a její druhá n-tice `(16,20)` se uloží do aktivního registru. Stejně tak jako u předcházející a všech následujících konstrukcí, je možné použít jako vstupní parametr této funkce místo čísla registr.

- **Přímé a nepřímé spuštění** - Při přímém spuštění se zavolá příkaz `@()`, kde jeho parametrem může být seznam akcí, které se mají provést. Ale aby při chybě některé z těchto akcí, nedošlo k chybě i u plánů o úroveň výš, vloží se na zásobník před nový plán ještě zarážka, která by případný neúspěch akce již nenesla dál. Podobně je na tom také přímé spuštění, kde však vstupním parametrem může být pouze název plánu, jež je uložen v bázi plánů. Příklad pro přímé zpuštění: `@(!(5,(2)))` a pro nepřímé: `^(pocitej)`.

- **Volání základních služeb senzorového uzlu** - Kromě již výše zmíněných služeb pro odesílání a příjem dat, implementuje aplikace senzorového uzlu ještě další, které je možné ovládat pomocí kódu v jazyce *ALLL*. Jednoduše je však možné přidat postupně další a poté je opět začít používat. Mezi ty základní patří například ovládání LED diod na senzorovém uzlu, jež se provádí pomocí příkazu `$(l,(<B>,<H>))`, kde za `<B>` se doplní barva diody z množiny {`r, g, y`} a za `<H>` se zadá číslo `1` pro rozsvícení nebo `0` pro zhasnutí. Další významnou službou je přesunutí agenta daného uzlu senzorové sítě na jiný. To lze provést pomocí příkazu `$(m,<A>)`, kde za `<A>` patří doplnit číslo uzlu, kam se má kód agenta zkopírovat. Další významné služby jsou zastavení činnosti interpretu uzlu, zapnutí naslouchání na příchozí zprávy nebo pozastavení vykonávání kódu agenta na daný čas.

- **Změna aktivity registru a testování znalostí** - Jelikož se u většiny operací používá pro vrácení výsledku jeho uložení do aktivního registru, je třeba, pokud budeme chtít pracovat s několika hodnotami najednou, měnit volbu aktivního. To lze udělat

pomocí příkazu `&(<R>)`. Přičemž místo `<R>`, je třeba zadat číslo registru od 1 do 3, jež se má stát aktivním. Jak bylo zmíněno výše, pro vyhledávání v bázi znalostí se užívá operace unifikace. Jejím parametrem je vzor entice zvolené pro vyhledání. Ty n-tice z vybrané báze, které odpovídají hledanému vzoru, jsou postupně ukládány v podobě seznamu do aktivního registru. V případě, že máme například v bázi znalostí n-tice jako `(15)(18)` a zadáme příkaz `*(18)`, do aktuálně aktivního registru se vloží n-tice `(18)`.

3.2 Součásti platformy WSageNt pro senzorovou síť

Tatu část platformy představují dva typy aplikací, určených a otestovaných pro běh na senzorových uzlech *MICAz* nebo *Iris*. Prvním typem je aplikace pro obecné senzorové uzly. Tato aplikace umožňuje komunikovat s ostatními uzly sítě a základnovou stanicí. Dále pak dokáže provádět interpretaci kódu v jazyce *ALLL* a podle ní pak provádět různé akce. Agentní kód může být na uzel přijmut buď od řídícího rozhraní na počítači nebo od jiného senzorového uzlu.

Druhý typem aplikace je program pro základnovou stanici, jež je tvořena jedním ze senzorových uzlů připojených přes *USB* bránu do počítače. Toto zařízení představuje přepínač v komunikaci mezi počítačem a jednotlivými uzly senzorové sítě.

Obě aplikace jsou napsané v jazyce *nesC* pomocí prostředků popsaných v kapitole 2.2.4. Komunikace mezi všemi uzly sítě i základnovou stanicí probíhá pomocí komunikační technologie *ZigBee*.

3.2.1 Aplikace pro obecné uzly senzorové sítě

Danou aplikaci navrhli a naimplementovali Ing. Jan Horáček a Ing. Pavel Spáčil na FIT VUT v Brně ve své diplomové práci [11]. Je určena pro nahrátí na všechny obecné uzly vybrané senzorové sítě. Každý uzel musí dále mít při jeho naprogramování zvoleno vždy unikátní číslo představující jeho adresu oproti ostatním uzlům v síti. To je nutné kvůli jejich rozlišení při komunikaci. Pro řízení činnosti uzlu se používá systémových modulů obsažených přímo v *TinyOS*.

Hlavní program je možné rozdělit na dvě základní část. Kde první představuje hlavní ovládání činnosti uzlu, bez ohledu na činnost agentů. V první fázi se po zapnutí napájení daného uzlu provede inicializace všech jeho základních komponent a interpretu. Poté se provede nastartování činnosti interpretu. Při jeho běhu se musí obě části aplikace synchronizovat, tak aby první část byla schopna zpracovávat vnější události jako příjem zpráv, řízení komunikace se základnovou stanicí, nahrávání nového agentního kódu či samotné požadavky na vykonání nějaké činosti zjištěné intepretací agentního kódu. Pro provádění interpretované akce se využívá knihovních modůlů, jež jsou součástí *TinyOS*. [11]

Druhá část je implementována jako interpret jazyka *ALLL*. Jeho činnost vzhledem k jednoduchosti agentního jazyka je řízena pomocí stavového automatu. Běh interpretu představuje hlavní smyčku celého programu pro uzel. Před zpracováváním agentního kódu se vždy na začátku smyčky zkontroluje zda, první část nepřijala nového agenta, pokud ano nahradí původního novým. Dále se ještě zkontroluje zda nepřišla nějaká data ze senzorů či obecná zpráva od základnové stanice nebo jiného uzlu. Ve smyčce se pak vždy v jednom kroku zpracuje jedna akce agenta. Při vykonání dané akce se využívá poskytovaných služeb, jež jsou implementovány v první části programu. [11]

V souhrnu tedy senzorový uzel obsahuje vždy platformu uloženou v paměti flash a po přijetí kódu agenta a jeho interpretaci je platforma využita k ovládání funkcí daného uzlu. Tato část celé platformy nebude touto diplomovou prací měněna. Musí s ní však být schopna v nezměněné podobě komunikovat nově upravovaná základnová stanice.

3.2.2 Aplikace pro základnovou stanici

Tato aplikace je bez malé úpravy přímou součástí knihovních aplikací *TinyOS*. Jejím úkolem je tvořit rozhraní mezi řídícím počítačem, na němž běží aplikace *BSComm* a *Control Panel* a senzorovou sítí. Pro komunikaci s počítačem využívá svého rozhraní *UART*[1]. Pomocí něj je připojena v případě použití senzorových uzlů *MICAz* nebo *Iris* na speciální hardwarovou *USB* bránu s označením *MIB520* do řídícího počítače. Jelikož *USB* i *UART* pracují téměř se stejnými napěťovými úrovněmi, nemusí brána řešit jejich převod.

Při komunikaci ve směru z a do senzorové sítě se používá základního datové paketu popsaného v kapitole 5.2.1. Podle jeho hlavičky provede stanice odeslání paketu získaného z počítače na vybraný senzorový uzel. V případě broadcastu, na celou síť s odpovídajícím číslem skupiny, jež je opět uvedena v hlavičce základního paketu. Pokud kterýkoli z uzlů odešle zprávu na základnovou stanici, přepošle se upravená zpráva na řídící počítač.

Při komunikaci mezi řídícím počítačem je základní datový paket určený pro senzorovou síť rozšířen o hlavičku a metadata pro zajištění spolehlivé komunikace po sériovém rozhraní. To umožňuje provádět číslování paketů a kontrolu integrity přenášených dat pomocí kontrolního součtu. Číslování paketů je zavedeno pro zjištění, zda některý paket nebyl ztracen během komunikace. Podobrobnější popis struktury paketů je uveden v kapitole 5.2.1. Toto nutí provádět základnovou stanici konverzi paketů při jejich přepínání mezi rádiovým a sériovým rozhraním.

Součástí této diplomové práce je právě změna kódu tohoto systémového modulu, aby místo přeposílání dat přes rozhraní *UART* na řídící počítač byla data posílána na *GSM* modem a odsud na počítač umístěný v síti Internet.

3.3 Řídící a monitorovací část platformy WSageNt

Podkapitola s daným názvem popisuje část platformy *WSageNt*, jež je umístěna na řídícím počítači a komunikuje se senzorovou sítí. Tato část platformy slouží pro správu senzorové sítě, její monitorování, zjišťování vzájemné polohy uzlů, posílání agentních nebo obyčejných zpráv do sítě a také jejich příjem a zpracování ve směru opačném.

Na řídícím počítači implementují popsané funkce dvě aplikace. O jejich návrh a následný vývoj se postaral Bc. Martin Gábor ve svojí diplomové práci [7]. První z těchto aplikací má název *Control Panel*. Představuje grafické uživatelské rozhraní celé platformy v podobě webových stránek. Ty jsou navrženy tak, aby v nich bylo snadné provádět základní operace se senzorovou sítí a přitom vše se přehledně zobrazovalo.

Druhou je aplikace *BSComm*, jež tvoří komunikační rozhraní mezi webovou stránkou a základnovou stanicí. Ta přebírá určitý formát dat od webového rozhraní, ten modifikuje a odesílá na základnovou stanici. Naopak při příjmu ve směru ze senzorové sítě data uloží do databáze, popřípadě sdělí příslušnou událost webovému rozhraní. Zároveň musí provádět řízení spolehlivé komunikace se sítí.

Ke dvěma výše zmíněným aplikacím lze zařadit i databázi, kam se provádí ukládání některých dat. Informace z ní jsou pak zobrazovány také ve webovém rozhraní, kde je

[1] Universální asynchronní přijímač/vysílač.

možné s nimi dále pracovat a provádět jejich úpravu. Rozhraní obou aplikací pro databázi je přednastaveno pro připojení k serveru *MySQL*, kde musí být vytvořena databáze podle pokynů v kapitole 3.4, jinak nebude serverová část platformy pracovat správně.

3.3.1 Aplikace BSComm

BSComm je konzolová aplikace, která tvoří nejnižší vrstvu na počítači, kde bude prováděn monitoring a správa senzorové sítě. *BSComm* komunikuje přímo se základnovou stanicí pomocí sériového rozhraní počítače. Dále také komunikuje oboustraně s webovým uživatelským rozhraním pomocí vytvořeného soketu. Přitom musí provádět analýzu příchozích zpráv od grafického rozhraní, aby z nich mohl sestavovat pakety pro senzorovou síť. *BSComm* je implementován v jazyce *JAVA*, konkrétně v platformě *JAVA SE*. [7]

Tato aplikace, jak již bylo zmíněno dříve, přistupuje také k databázovému serveru *MySQL* pomocí rozhraní vytvořeného ve frameworku *Hibernate*. Na databázovém serveru tedy pak musí běžet správná databáze s určenými právy podle 3.4, kam ukládá příchozí zprávy a agenty od jednotlivých uzlů sítě.

Nejdůležitější částí aplikace *BSComm* je třída *Comm* vycházející ze třídy *MessageListener*, jež pochází z knihovních tříd z *JAVA SDK* pro *TinyOS*. Třída *Comm* přímo zabezpečuje komunikaci se základnovou stanicí. Dále se stará o vytváření správného formátu paketů zasílaných po sériové lince a ověřování integrity paketů doručených od základnové stanice. Především však musí pomocí zasílání a příjmu potvrzujících paketů zabezpečovat spolehlivou komunikaci přes rozhraní počítače. [7]

Další částí této práce je tedy rozbor aplikace *BSComm* a hlavně třídy *Comm*, pro zjištění možnosti jak by se dalo místo sériového portu komunikovat se základnovou stanicí přes *GSM* modem pomocí sítě Internet.

3.3.2 Webové rozhraní Control Panel

Webové rozhraní *Control Panel* je grafická nadstavba nad aplikací *BSComm* napsaná opět pomocí programovacího jazyka *JAVA* ve frameworku *Struts 2* pro platformu *JAVA EE*. Umožňuje uživatelsky příjemné a jednoduché možnosti práce s bezdrátovou senzorovou sítí.

Mezi základní podporované funkce patří přehled aktivních uzlů sítě z předdefinovaného seznamu a nástroje určené k jejich správě. Dále pak umožňuje zasílání agentů a jednoduchých zpráv na zvolené uzly. V neposlední řadě je také možné provést zobrazení topologie senzorové sítě na základě silý signálů jednotlivých uzlů a seznam všech agentních i obyčejných zpráv přijatých ze sítě.

Komunikační protokol mezi grafickým rozhraním a řídící logikou je implementován pomocí speciálního formátu agentní a obyčené zprávy, jež byl navržen v práci [7]. Pro agentní zprávu je používán tento formát:

```
typ_zprávy ODDĚLOVAČ adresa_uzlu ODDĚLOVAČ planBase ODDĚLOVAČ
plan ODDĚLOVAČ beliefBase ODDĚLOVAČ inputBase
```

Pro obyčejné zprávy je to tento formát:

```
typ_zprávy ODDĚLOVAČ adresa_uzla ODDĚLOVAČ obsah_zprávy
```

Takováto zpráva je sestavena ze vstupních komponent webového formuláře a po potvrzení tlačítkem je odeslána na *BSComm*. Navíc se po určitých časových intervalech vygeneruje samovolně zpráva pro ověření živosti každého ulzu sítě, jež je uložen v databázi. V případě příjmutí jejího potvrzení aplikací *BSComm* a oznámení webovému rozhraní, se vykreslí na stránce ikona s číslem aktivního ulzu.

Webové rozhraní přistupuje také k databázi *MySQL*, kde je umožněno ukládat jednotlivé odchozí agenty, zprávy a podobně. Později je možné je zase načíst a pracovat s nimi.

Tato část platformy opět v mojí práci měněna nebude, jelikož to zádání nevyžaduje. Bude muset být pouze zachován komunikační protokol s tímto uživatelským rozhraním, tak aby výstupy a vstupy z této aplikace mohly být zachovány.

3.4 Instalace aplikací BSComm a Control Panel na řídící počítač

Pro uvedení dané platformy do provozu je zapotřebí provést několik dílčích kroků podle [12], kde prvním je stažení všech součástí z daného zdroje. Naprogramování zařízení do módu základnové stanice nebo do módu obyčejného senzorového uzlu je popsáno v kapitole 2.2.3. Abychom uvedli do provozu serverovou část aplikace bude nutné dát na serverové stanici do provozu aplikaci *BSComm* a *Control Panel*.

Nejprve je třeba do systému nainstalovat *Apache server* a aplikační server *Apache Tomcat* pro běh rozhraní *Control Panel*. Instalace lze provést pomocí příkazů:

```
sudo apt-get install apache2 tomcat6 tomcat6-admin
```

Poté je třeba do souboru */var/lib/tomcat6/conf/tomcat-users.xml* na serverové stanici vložit kód pro umožnění správy aplikací běžících na webovém serveru. Tento kód může vypadat například následovně:

```
<tomcat-users>
  <role rolename=''manager-gui''/>
  <role rolename=''manager-script''/>
  <role rolename=''manager-jmx''/>
  <role rolename=''manager-status''/>
  <role rolename=''manager''/>
  <role rolename=''admin''/>
  <user username=''root'' password=''ab5rrt6'' roles=''admin, manager,
  manager-gui, manager-script, manager-jmx, manager-status''/>
</tomcat-users>
```

Pak se na server nahraje aplikace představující grafické rozhraní řídící aplikace. V prohlížeči se tedy zadá adresa *localhost:8080/manager*, kde lze provést po přihlášení jménem "root" a heslem "ab5rrt6" instalaci nové aplikace s příponou *war*. Proto stačí stáhnout a rozbalit aplikaci *Control Panel* z [12]. Soubor z umístění *ControlPanel/dist/ControlPanel.war* je pak možné načíst na server.

V dalším kroku se provede instalace *MySQL* serveru. To jde pro systémy operační systémy typu *Ubuntu*, *XUbuntu* a podobně provést pomocí příkazu:

```
sudo apt-get install mysql-server
```

Dále je vhodné nainstalovat nějaké klientské grafické rozhraní pro zprávu databáze jako je například *phpMyAdmin*, ale zprávu databáze je možné samozřejmě provádět i přes konzoli pomocí standardních příkazů. Pro instalaci rozhraní stačí zadat:

```
sudo apt-get install phpmyadmin
```

Nyní lze vytvořit, přes *phpMyAdmin* na adrese ve webovém prohlížeči *localhost/phpmyadmin* či příkazů v konzoli, databázi se jménem "wsagent", uživatelem také "wsagent" a heslem "hesloveslo". V konzoli operačního systému, kde je databázový server nainstalován, se stejného výsledku dosáhne pomocí těchto příkazů:

```
mysql --user=root --password=<root heslo>

mysql> create database wsagent;

mysql> GRANT ALL PRIVILEGES ON wsagent.* TO wsagent@,,%‘‘ IDENTIFIED BY
’hesloveslo’;
```

Pro spuštění původní aplikace *BSComm* je možné zadat v umístění *BSComm/dist*, kde je *BSComm* stažen a rozbalen, příkaz:

```
java -jar BSComm.jar serial@/dev/ttyUSB1:<BAUDRATE> 7777
```

Poté je již rozhraní na serveru dostupné a plně funkční společně s aplikací *BSComm* na adrese *localhost:8080/ControlPanel*.

Kapitola 4

GSM

Zkratka *GSM* pochází podle [25] původně z francouzského pojmenování Groupe Spéciale Mobile a později byl její význam změněn na výstižnější název Global System for Mobile Communication. Jde o celoevropský standard pro mobilní komunikaci zavedený institucí *ETSI*[1]. V dnešní době jej využívá více než 82% všech mobilních sítí na celém světě. Mezi jeho nesporné výhody patří hlavně možnost bezdrátové komunikace se stejným přístrojem i telefonním číslem téměř po celém světě, díky *SIM* kartě možnost změnit mobilní telefon bez změny čísla a tarifu, kvalitní zabezpečení před útoky na síť *(SIM, PIN)* a podpora nejen hlasových, ale i datových služeb. Má však i některé nevýhody jako neexistence šifrování od jednoho koncového uzlu až po druhý[2], téměř stálé elektromagnetické vyzařování v okolí mobilu a vysílačů a také nekompatibilita některých částí *GSM* standardu.

4.1 Stavba GSM sítě

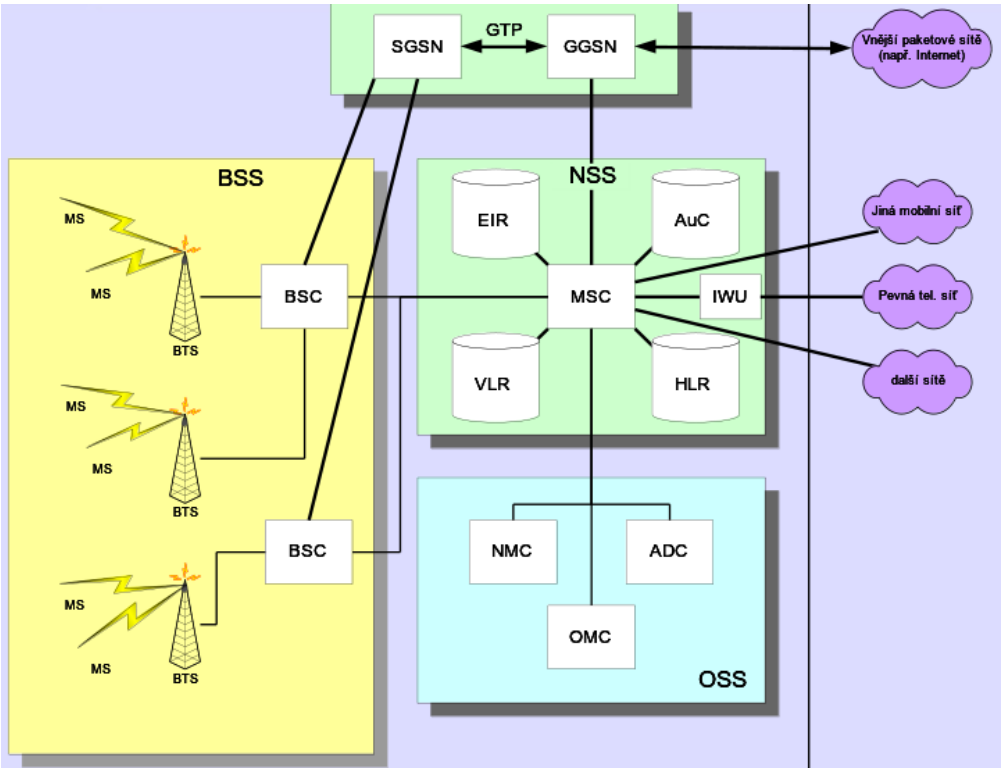
GSM síť je celulární síť, jež je tedy tvořena menšími uzemní buňkami. Každé zařízení se tedy do *GSM* sítě přihlašuje k řídící stanici v nejbližší buňce. Řízení několika buněk má na starosti vždy nějaká větší řídící stanice. Toto rozdělení na buňky je výhodné v tom, že se mohou ve vzdálenějších buňkách opakovat stejné komunikační frekvence. *GSM* samo řeší přechod mobilního zařízení mezi buňkami. Nejčastěji pracuje *GSM* síť na dvou základních řídících frekvencích a to 900 MHz a 1800 MHz. Základní struktura celé *GSM* sítě je tvořena podle [25] přibližně takto:

- *BSS* - Base Station Subsystem - tato část se stará o připojování a komunikaci *MS* se sítí *GSM* a řídí také správné přidělování frekvencí do buněk, aby se vzájemně nepřekrývali. Do této sekce patří Mobile station (*MS*), představuje zařízení se *SIM* kartou odpovídající standardu *GSM*, Base transciever station (*BTS*), pracující jako vysílací a přijímací stanice a Base station controller (*BSC*), řídící sílu signálu, přechod *MS* mezi buňkami a řízení přístupu *MS* do svých timeslotů.
- *NSS* - Network and Switching Subsystem - zde se řídí spojování hovorů, účtování, autentizace zákazníků a přepojování sítí. Obsahuje především Mobile switching center (*MSC*), které zajišťuje přepínání kanálů, řízení hovoru, přístup k databázi zákazníků a přepínání mezi jinými *MSC* a do sítí jiných operátorů. Dále je zde například ještě Authenticity centre (*AUC*) provádějící autentizaci klientů v síti daného operátora

[1] European Telecommunications Standardisation Institute
[2] V GSM se šifruje se pouze přenos z mobilní stanice po nejbližší základnovou stanici a opačně.

a *GPRS* Core Network, jež představuje rozšíření *GSM* sítě pro funkci přepínání paketů pro datově orientované přenosy.

- *OSS* - Operations Subsystem - je plnohodnotným počítačovým systémem pro správu, monitorování a řízení celé *GSM* sítě.



Obrázek 4.1: Základní struktura GSM sítě [25].

4.2 Modem GSM

Pro projekt byl podle zadání použit modem Teltonika ModemCOM/G10 [24] jež je možné vidět na obrázku 4.2. Tento modem je původně určen k připojování počítačů k sítí *GSM* například pro možnost přístupu na síť Internet. Představuje standard ve všech parametrech a možnostech využití na svém poli působnosti. Pomocí něj je možné přijímat a odesílat hovory, posílat *SMS* a využívat datového připojení *GPRS*. Původně je dodáván s programem pro ovládání výše zmíněných základních funkcí, ale ten nebude v naší aplikaci použit, proto zde nebude více zmiňován.

4.2.1 Základní parametry modemu

Modem dokáže pracovat ve všech základních *GSM* pásmech 850/900/1800/1900 MHz. A pro spojení pomocí *GPRS* je zařazen do třídy 10, jež umožňuje přenos dat pro odesílání rychlosí 16 až 24 kbps a 32 až 48 kbps pro jejich stahování. Dále obsahuje vrstvu pro práci s protokolem *TCP/IP*, která bude využívána pro komunikaci základnové stanice se serverem.

Obrázek 4.2: Modem Teltonika ModemCOM/G10.

Ovládání je možné provádět přes sériové rozhraní *RS-232* s parametry uvedenými v tabulce 4.1. K ovládání je však potřeba znát alespoň základní AT příkazy. Ty jsou popsány v následující kapitole. Zároveň je patrné, že přidáním dalšího zařízení v podobě tohoto modemu při komunikaci mezi základnovou stanicí a řídícím rozhraním přinese zpomalení přenosu dat. Tato zkutečnost je dána nejen nutností přenášet data na větší vzdálenost, ale především režií při přenášení dat, která vznikne vlivem odesílání a příjmu řídících AT příkazů mezi modemem a základnovou stanicí a celkově parsováním velkého množství těchto příkazů. Původně totiž v platformě WSageNt při přenosu dat mezi základnovou stanicí a řídícím rozhraním docházelo k přenosu pouze čistých dat, nyní bude nutné navíc provádět řízení modemu a vyzvedávání příchozích dat z bufferu modemu, jež je omezen na 1024 bytů. Zároveň nabude množství přenášených dat vlivem obalování odchozích dat do AT příkazů a zpomalení vznikne také oddělením příchozích dat ze serveru od AT příkazu, jež je zapouzdřuje.

Parametr	**Hodnota**
Baudrate	115200 b/s
Datové bity	8 b
Stop bity	1 b
Handshake	Hardwarový
Parita	Není

Tabulka 4.1: Parametry rozhraní RS-232 modemu Teltonika ModemCOM/G10.

Modem tedy obsahuje rozhraní *RS-232*, zdířku pro připojení externího napájení, tři indikační LED diody, konektor s přípojnou anténou pro *GSM* a slot pro vložení *SIM* karty. Napájecí adaptér je součástí balení tohoto modemu. Přesto, že by bylo pro naši aplikaci v senzorové síti vhodné na místo napájení modemu ze sítě použít nějaký bateriový zdroj energie, bude tato skutečnost zanedbána. Při reálném použití by nebylo problémem takovou baterii, zastupující zmíněny zdroj, obstarat a zapojit. První indikační LED dioda představuje indikaci připojeného napájení. Druhá LED dioda indikuje připojení k *GPRS* síti. Pokud bliká rychle, zařízení není připojeno k *GPRS*, pokud bliká pomaleji, nastala chyba s připojením a pokud bliká vždy dvakrát po sobě, není připojena do modemu *SIM*

karta. Třetí LED dioda indikuje připojení zařízení do *GSM* sítě. Pokud bliká v intervalu po dvou, není vložen *PIN* kód, pokud bliká pomalu, provádí se přihlašování do sítě. Pokud svítí modem je připojen do *GSM*.

4.3 Řídící příkazy pro modem

Jak bylo zmíněno dříve modem lze ovládat přes sériové rozhraní pomocí AT příkazů. Ty se staly téměř standardem, i když se pro některé modemy liší. Jsou jednoduchou možností pro ovládání takového zařízení. Umožňují vytáčet a přijímat telefonní hovory, odesílat a číst *SMS* zprávy, provádět datové přenosy a podobně.

Základní filozofie AT příkazů spočívá ve dvou možnostech, jak bude probíhat komunikace s modemem. První možností je, že nám od modemu samovolně přijde nevyžádaná zpráva. Tento režim lze označit za asynchronní, jelikož se to může stát kdykoli, i když například probíhá čekání na zprávu, kterou požadujeme. To přináší komplikace pro komunikaci s modemem. Tento problém bude řešen v kapitole s návrhem. Druhou možností typu komunikace s modemem je synchronní komunikace. U této komunikace odešleme modemu naši žádost a[3] on nám na ni odpoví.

Existují tyto základní druhy AT příkazů zakončených vždy znakem `<CR>` :

- Základní - například pro položení hovoru příkazem `ATH<CR>`, jsou tvořeny jedním slovem.
- Ověřovací - dostaneme odpověď ve stylu ano nebo ne. `AT+CPWROFF=?<CR>`, pro dotaz zda je možné vypnout mobilní stanici.
- Zjišťovací - dostaneme odpověď s požadovanou hodnotou. `AT+CPIN?<CR>`, pro ověření zda byl zadán správný *PIN*.
- Nastavovací - nastavíme nějakou hodnotu v modemu či SIM kartě. `AT+CPIN=''1234'' <CR>`, vloží *PIN* pro přihlášení do *GSM* sítě.

Na dané příkazy je možné dostat odpovědi začínající a končící `<CR><LF>` jako:

- Pokud to byl dotaz na hodnotu nebo pro nastavení dostaneme ve většině případů odpověď ve tvaru `<CR><LF>+<TYP ODPOVĚDI>: <HODNOTA><CR><LF>`.
- Ale vždy dostaneme rozhodně odpověď `<CR><LF>OK<CR><LF>` nebo `<CR><LF>ERROR <CR><LF>`.

Souhrn základních AT příkazů a nevyžádaných zpráv je možné vidět v tabulce 4.2:

4.4 Datové přenosy

Jak bylo řečeno dříve datové přenosy na modemu Teltonika budou prováděny pomocí *GPRS*. Jelikož modem obsahuje sadu AT příkazů přímo pro práci s *TCP/IP*, budou pro komunikaci tyto příkazy použity. Jejich použitím odpadne řada problémů ze zajištěním spolehlivé komunikace na server.

V následující podkapitole bude obecně popsána technologie *GPRS* určená pro komunikaci základnové stanice se serverem na síti Internet. Další podkapitola přinese popis ovládání modemu pomocí AT příkazů právě pro datové přenosy přes *GPRS*.

[3] V GSM se šifruje se pouze přenos z mobilní stanice po nejbližší základnovou stanici a opačně.

AT příkaz / nevyžádaná odpověď	Činnost AT příkazu / význam příchozí zprávy
`AT+CPIN?`	Dotaz zda byl zadán PIN.
`ATH`	Dojde k položení příchozího hovoru.
`AT+CPIN=''1234''`	Zadá na SIM katu PIN kód 1234.
`AT+CMGL`	Vypíše seznam SMS zpráv.
`AT+CMGD=2`	Smaže zprávu číslo 2.
`RING`	Nevyžádaná zpráva oznamující příchozí hovor.
`+CMTI: ''SM'',0`	Nevyžádaná zpráva oznamující příchozí SMS.

Tabulka 4.2: Tabulka se základními příkazy podle [14].

4.4.1 GPRS

GPRS byl prvním standardem *ETSI*[4] pro přenost dat pomocí přepínání paketů v síti *GSM*. Jelikož byla tato síť určena nejprve pouze pro přenos hlasu pracovala na principu přepojování okruhů, ale s roustoucí poptávkou po datových službách se začala používat i pro přenos dat. Tento způsob přenášení informací byl značně neefektivní, jelikož pro přenos informací byl vždy zabrán celý přenosový kanál mezi odesílatelem a příjemcem v podobě obsazování stále stejného timeslotu[5] v přenosové periodě. Efektivně je možné v základní podobě tohoto způsobu přenosu dosáhnout rychlosti 13,5 kbit/s. Později se tato rychlost pomocí využití více přenosových kanálů znásobovala. Bohužel zvyšování rychlosti neměnilo nic natom, že při spojení byl zabrán celý kanál s protistranou, ikdyž zrovna nebyl používán. Tomu také odpovídalo účtování za tuto službu, kde se platilo za čas a místo kam bylo zařízení připojeno. [26]

Právě z výše popsaných důvodů vzniklo *GPRS*. Pro jeho zavedení museli mobilní operátoři rozšířit svojí síť o nové prvky, jež budou představovat nadstavbu původní sítě. Tyto nové prvky označované jako *SGSN*, mohou být připojeny přes řídící uzly vysílačů až ke komunikujícímu zařízení. *SGSN* je pak přes speciální rozhraní připojeno ke klasiské IP síti. Takto upravená síť umožňuje, aby její účastníci svými datovými pakety obsazovali timesloty, jež nejsou používány k telefonním hovorům. Toto přináší hlavní výhodu v tom, že je komunikační pásmo mnohem efektivněji využíváno. Ale přináší to také nevýhodu v tom, že pokud bude v daném pásmu probíhat mnoho hovorů, bude volných pouze několik timeslotů a tudíž bude malá rychlost přenosu dat. Přenosová rychlost se také odvíjí od toho kolik timeslotů je schopno komunikující zařízení obsadit. *GPRS* implementuje mezi *SGSN* a komunikujícím zařízením IP komunikaci. Data jsou dále z *SGSN*, přes výše zmíněnou bránu, přenášena podle cílové adresy do cílové IP sítě. [26]

4.4.2 Přenos dat s využitím AT příkazů

Z dokumentu [14] byly zjištěny základní příkazy, jež je potřeba provést pro inicializaci *GPRS* spojení, odesílání dat a jejich příjem. Tyto příkazy uvádí tabulka 4.3. Podle těchto příkazů se nejprve provede nastavení a poté i spuštění profilu pro *GPRS*. Dále vytvoření *TCP* soketu, připojení k serveru a odeslání a příjem dat. Nakonec je uveden příkaz pro

[4] European Telecommunication Standard Institute.
[5] V čase se obsazuje stále stejný datový úsek pásma.

uzavření soketu.

AT příkaz	**Činnost AT příkazu**
`AT+NPSD=0,1,''internet''`	Provede nastavení profilu 0 na síť Internet.
`AT+NPSDA=0,1`	Uloží se aktuální profil 0.
`AT+NPSDA=0,3`	Aktivuje profil 0.
`AT+NSOCR=6`	Vytvoří soket pro TCP (6 = TCP).
`AT+NSOCO=0,''147.229.176.14'',1111`	Provede se připojení na server 147.229.176.14 a port 1111.
`AT+NSOWR=0,5`	Sdělí modemu, že bude odesláno 5 bytů dat přes soket 0. Až Server odpoví @, lze posílat data.
`AT+NSOCL=0`	Uzavře soket 0.
`AT+NSORD=0,15`	Pokud nám přišla nevyžádaná zpráva oznamující příchozí data v soketu o délce 15, lze ji takto vyzvednout.

Tabulka 4.3: Tabulka s AT příkazy pro práci s GPRS podle [14].

Modem nám může také zaslat nevyžádané zprávy. Mohou to být upozornění například na příchozí hovor, příchozí datové spojení přes *GPRS*, či oznámení o příchozí *SMS* zprávě. Tyto příkazy vždy začínají znakem +. Ty které bude nutné v aplikaci obsluhovat jsou uvedeny v tabulce 4.4.

Příchozí zpráva	**Význam**
`+NUSORD: 0,15`	Na soketu 0 přišla data o délce 15 bytů.
`+NUSOCL: 0`	Došlo k uzavření soketu.

Tabulka 4.4: Příchozí nevyžádané zprávy týkající se GPRS podle [14].

Kapitola 5

Návrh rozhraní mezi základnovou stanicí a řídícím serverem

Návrh rozhraní mezi senzorovým uzlem FITmote a aplikací *BSComm* umístěné na řídícím serveru je možné rozdělit na dvě základní části. První částí je realizace fyzického spojení senzorového uzlu, představující část základnové stanice a *GSM* modemu. Návrh této části celého rozhraní bude představovat tvorbu schématu pro následnou implementaci desky plošného spoje. Základním požadavkem na tuto desku je možnost připojit na jedné straně pomocí konektoru senzorový uzel a na druhé modem. Zároveň musí být schéma navrženo tak, aby nebylo nutné fyzicky upravovat žádným způsobem senzorový uzel ani modem a přitom byla mezi nimi zaručena bezproblémová komunikace.

Druhou částí je návrh dvou aplikací, kde první bude umístěna na senzorovém uzlu a druhá na řídícím serverovém počítači v síti Internet. Navržená aplikace pro senzorový uzel nahradí v platformě původní aplikaci *BaseStation* základnové stanice. Ta bude provádět přesměrovávání a formátování zpráv ve směru ze senzorové sítě na řídící server a zpět s využitím popsaného modemu přes sériové rozhraní. Nová aplikace určená pro server bude náhradou za aplikaci *BSComm*, jež tvoří aplikační logiku pro webové řídící rozhraní a komunikuje se základnovou stanicí. Původní aplikace umožňovala činnosti jako je sestavování a odesílání datových paketů, které umí dekódovat základnová stanice, práci s databází, komunikaci s webovým rozhraním *Control Panel*, či zpětné přijímání paketů od základnové stanice a jejich analýzu. Nová aplikace bude muset být kromě těchto činností navíc schopna ustanovit a řídit datové spojení s *GSM* modemem, popřípadě obnovit a vyresetovat dané spojení v případě výpadku.

5.1 Fyzické spojení senzorového uzlu a GSM modemu

Na senzorovém uzlu *FITmote* bylo pro spojení s *GSM* modemem Teltonika ModemCOM/G10 zvoleno sériové rozhraní *UART*. Tato volba byla výhodná z toho důvodu, že použitý *GSM* modem je možné ovládat pouze pomocí jeho sériového rozhraní *RS-232*, které je s obecným rozhraním *UART* v principu přenosu dat prakticky shodné. V obou případech jde totiž o sériové porty, jejichž základní fyzická linka je tvořena jedním vodičem *RxD* pro příjem dat, jedním vodičem *TxD* pro odesílání dat a vodičem *GND* představující společný zemnící vodič. Obě rozhraní navíc poskytují další dva vodiče pro řízení komunikace, ty však pro naše účely nejsou zapotřebí a nebudeme je tedy v této práci popisovat.

Jelikož však senzorový uzel a *GSM* modem pracují u výše zmíněných rozhraní s růz-

nými napěťovými úrovněmi, je nutné vyřešit jejich konverzi. Pro standard *RS-232* platí, že napěťové úrovně jsou v rozmezích podle tabulky 5.1.

Logická úroveň	**Úroveň napětí**	
	Pro vysílání	**Pro příjem**
Logická 0	+5 V až +15 V	+3 V až +25 V
Logická 1	-5 V až -15 V	-3 V až -25 V
Nedefinováno	-3 V až +3 V	

Tabulka 5.1: Rozsahy napětí logických úrovní pro RS-232 [19].

Přijímač sériového rozhraní modemu tedy rozpoznává přijaté signály, až při dosažení napětí na vstupu nad +3 V pro logickou nulu a pod -3 V pro logickou jedničku. Protože je jádrem senzorového uzlu FITmote modul *ATZB-24-A2*, jehož základ tvoří dle [2] procesor AVR ATmega1281, pracuje celý uzel s napěťovými úrovněmi odpovídající nízkoenergeticky náročným obvodům typu *CMOS*. U obvodů tohoto typu jsou napěťové úrovně pro logickou jedničku a nulu přímo závislé na napájecím napětí daného obvodu. Podle [6] platí pro obvody *CMOS*:

Logická úroveň	**Úroveň napětí**
Logická 0	0 až $0,3U_{cc}$
Logická 1	$0,7U_{cc}$ až U_{cc}
Nedefinováno	$0,3U_{cc}$ až $0,7U_{cc}$

Tabulka 5.2: Rozsahy napětí logických úrovní pro obvody CMOS. U_{cc} značí napájecí napětí daného obvodu.

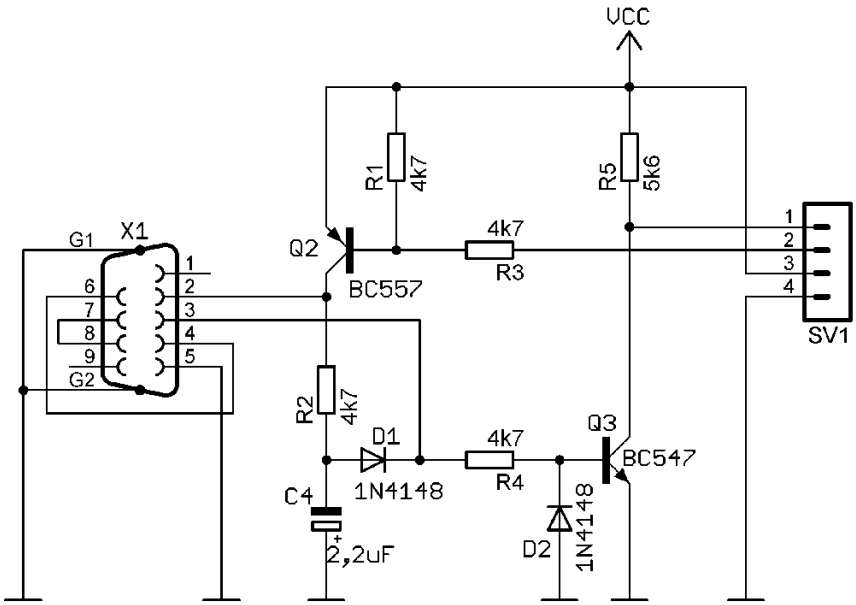
Jak, již bylo zmíněno v kapitole 2.1.3 je senzorový uzel FITmote napájen pomocí knoflíkové baterie s napětím 3V. Podle 5.2 tedy senzorový uzel používá pro logickou nulu napětí 0 až 0,9 V a logickou jedničku 2,1 až 3 V.

Je tedy jasně patrné, že se napěťové úrovně používané modemem a senzorovým uzlem liší. Ani v krajním případě, kdy by se při připojení k modemu použilo u senzorového uzlu obrácených napěťových úrovní, by nedošlo k opuštění hranice pro nedefinovaný stav logických úrovní u modemu. Pokud se k této skutečnosti ještě domyslí průběžný pokles napájení z baterie senzorového uzlu vlivem jejího vybíjení a potřeba snižovat napěťové úrovně ve směru od modemu k uzlu na hodnoty určené pro *CMOS* obvody, stává se nutností najít komplexní řešení všech výše popsaných problémů. V další části této kapitoly bude tedy provedeno několik možností návrhu pro řešení tohoto problému pomocí speciálního převodního můstku.

5.1.1 Zapojení řízené pomocí tranzistorů

První možností je použít zapojení řízené pomocí tranzistorů. V tomto zapojení by bylo na desku plošného spoje přivedeno externí napájení s velikostí napětí rovné velikosti logické nuly pro *RS-232* podle [19]. Tranzistory by pak v zapojení 5.1 spínali kladné či záporné napětí na vodiče *RxD* pro příjem dat a *TxD* pro odesílání dat podle logických úrovní sériových rozhraní z *GSM* modulu a podobně i ze senzorového uzlu. Správné úrovně napětí a jejich polarity by nastavovali rezistory, kondenzátor a usměrňovací diody podle [15]. Toto řešení však pro danou aplikaci není úplně vhodné, jelikož přidání dalšího energeticky závislého modulu, požadující napájení o vyšších úrovních než by poskytl senzorový uzel, by

vyžadovalo přidat další napájecí zdroj. Bylo by sice možné použít napájení určené pro *GSM* modem, ale jelikož má být převodní můstek spíše součástí senzorového uzlu nezávislém na *GSM* modemu, je nutné najít jiné řešení.



Obrázek 5.1: Schéma navrženého propojení mezi základnovou stanicí a *GSM* modemem s externím napájením vytvořené v programu EAGLE [15].

5.1.2 Schéma tvořené obvodem s nábojovými pumpami

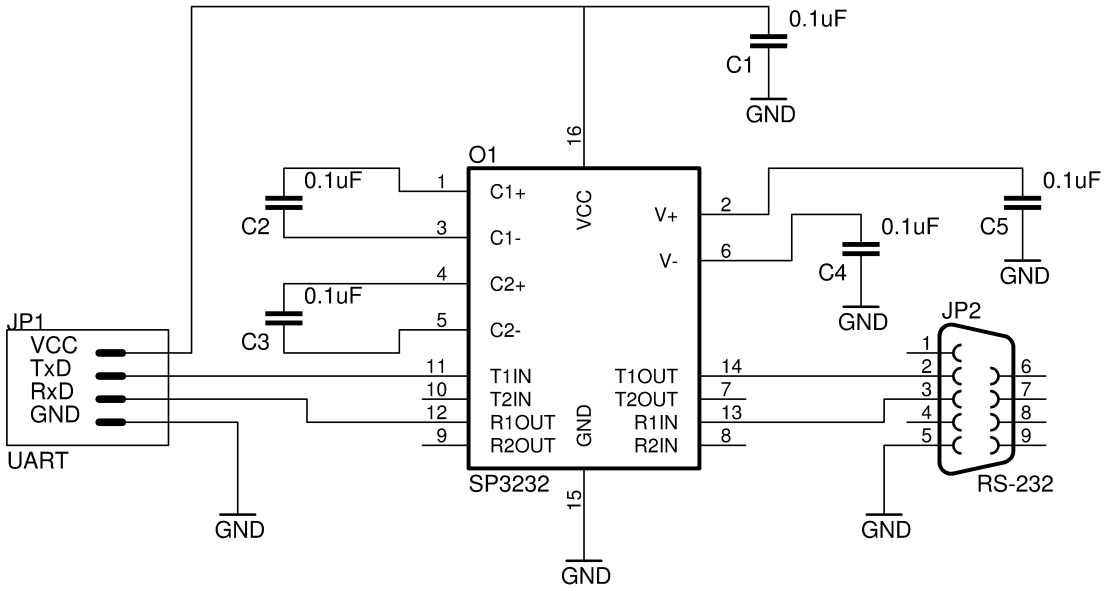
Další možností je použití některého elektronického obvodu řady 232 jako je například *MAX232* [18] od firmy Maxim. Jde o jeden z nejznámějších převodníků pro převod napěťových úrovní z *TTL*[1]*/CMOS* do úrovní pro *RS-232* a zpět. Tento obvod pracuje na základě využití tzv. nábojových pump. Je to speciální uspořádání kondenzátorů, jež se řízeně nabíjejí a vybíjejí a tím mohou vytvářet z napájecího napětí +5 V přibližně napětí o úrovních -8 V a +8 V pro výstup na *RS-232*, na základě vstupu v logických úrovních *TTL/CMOS*. Zároveň je tento obvod schopen podle úrovní na vstupu *RS-232* vytvářet odpovídající úrovně výstupního napětí pro *TTL/CMOS* rozhraní. Daný obvod tedy dokáže realizovat žádané funkce, ale vyžaduje opět vyšší napájení něž poskytuje daná konfigurace zapojení bez většího hardwarového rozšíření. Z něho však vznikl dále obvod *MAX3232* [17], jemuž stačí napájecí napětí již pouze s velikostí +3,3 V. Však i této napěťové úrovně jen těžko docílíme přivedením napájení ze senzorového uzlu, jež je popsáno v kapitole týkající se vybraného uzlu FITmote.

Naštěstí se podařilo najít další obvod řešící poslední nedostatek, jež vychází ze dvou předchozích zařízení popsaných výše a tím je jeden z jejich klonů s označením *SP3232* [5] od firmy Sipex. Stavba tohoto obvodu je tedy obdobná obvodům od firmy Maxim, avšak integrované nábojové pumpy dokážou vytvořit potřebné napětí do *RS-232* rozhraní z menší úrovně napájení. Podle [5] postačuje tomuto obvodu pro napájení pouhých +2,7 V, což

[1] Transistor transistor logic

je i s rezervou taková úroveň napětí, kterou pokryje napájecí napětí ze senzorového uzlu *FITmote*.

Při návrhu převodního mostu je podle [5] zapotřebí doplnit obvod o pět kondenzátorů o kapacitě 0,1 μF představující nábojové pumpy. Pak stačí pouze připojit vývody senzorového uzlu a *GSM* modemu *TxD* a *RxD* správně na dané piny obvodu a připojit uzemnění. Výsledné schéma je pak vidět na obrázku 5.2.



Obrázek 5.2: Schéma navrženého propojení mezi základnovou stanicí a *GSM* modemem vytvořené v programu EAGLE.

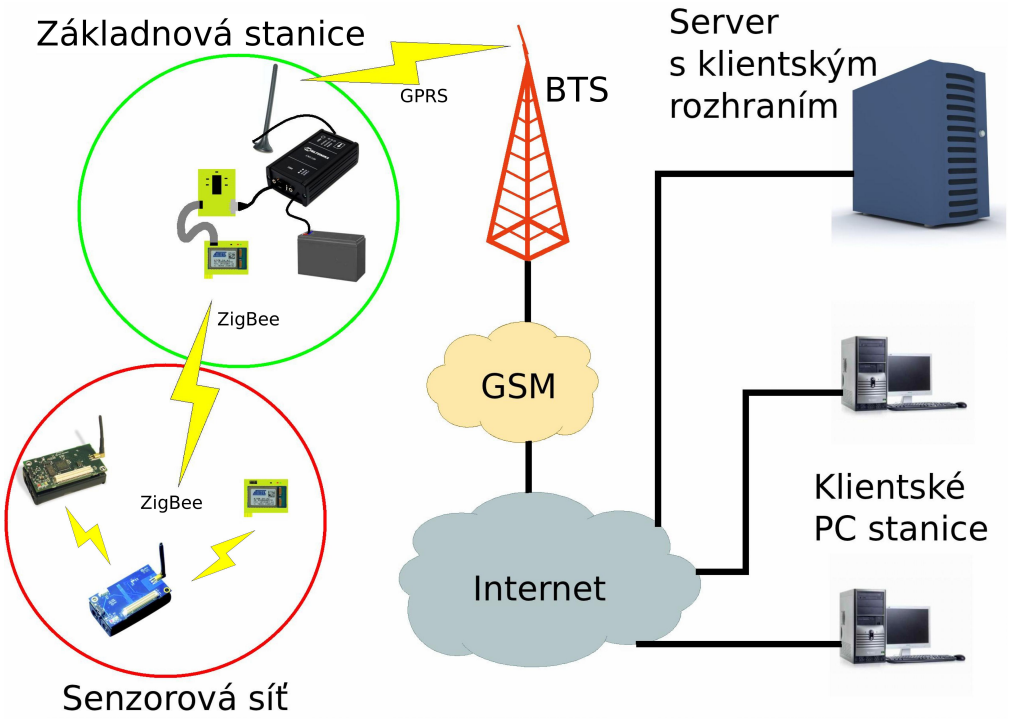
5.2 Komunikační protokol mezi základnovou stanicí a serverem

Na základnové stanici běží podle specifikace platformy *WSageNt* popsané v kapitole 3 aplikace *BaseStation*. Přes ni probíhá komunikace mezi senzorovou sítí a aplikací *BSComm* pomocí rozhraní *UART*. Jelikož je záměrem diplomové práce zaměnit rozhraní *UART* komunikací přes síť *GSM*, je v této části práce proveden návrh protokolu pro komunikaci od základnové stanice pomocí GPRS až po aplikaci *BSComm* umístěné na serveru.

Návrh se v první řadě zaměřuje na rozšíření aplikace *BaseStation* o nové rozhraní pro přístup do *GSM* modemu přes sériové rozhraní. Navržené rozšíření bude sloužit pro snadné ovládání modemu a také rychlý přenos dat pomocí GPRS spojení, vytvořeného pomocí tohoto modemu. Důležitou součástí komunikačního protokolu je vyřešit tvorbu paketů zasílaným a přijímaných ze serveru.

Další část návrhu se věnuje úpravě aplikace *BSComm* pro příjem a odesílání dat přes síť Internet. Zde se bude při návrhu vycházet z původní aplikace a navíc dojde k jejímu rozšíření, jež umožní naslouchat na příchozí spojení od základnové stanice, po němž se ustálí komunikace. Dále je nutné vyřešit správu probíhajícího spojení a stejně jako u aplikace *BaseStation* analyzovat práci s přenášenými pakety.

Po dokončení této diplomové práce bude možné provádět ovládání senzorové sítě z klientských počítačových stanic přistupujících k webovému rozhraní *Control Panel*, jež je umístěno společně s nově navrženou aplikací nahrazující *BSComm* na serveru s veřejnou IP adresou v síti Internet. Celé schéma komunikace nově upravené platformy *WSageNt* je možné vidět na obrázku 5.3.



Obrázek 5.3: Návrh architektury celé aplikace.

5.2.1 Formát dat přenášených mezi modemem a serverem

Jelikož v platformě *WSageNt* probíhá veškerá komunikace pomocí paketů o různých formátech, bude vhodné pro přenos dat mezi modemem a serverem opět využívat určitý formát paketů. Při jeho návrhu jsem vycházel ze stavby paketů přenášených pomocí sériového rozhraní mezi základnovou stanicí a aplikací na řídícím počítači původní platformy *WSageNt*. Tento formát umožňuje provádět spolehlivý přenos dat i při ztrátě a poškození paketů přenášených po sériové lince pomocí číslování paketů a kontrolního součtu. Jeho další výhodou je poměrně snadné provádění konverze mezi pakety přijatými od řídícího počítače na pakety určené pro odeslání na senzorovou síť a obráceně. Při těchto konverzích je výhodou využití některých knihovních modulů *TinyOS*. Také pro sestavení paketů určených pro základnovou stanici z dat přicházejících z webového rozhraní se v původní platformě používá rozšiřujících tříd v jazyce *JAVA* opět z *TinyOS*. Toto platí i pro obrácenou konverzi z paketů od základnové stanice na data předávaná webovému rozhraní a databázi.

Základnová stanice plní jak v původní platformě tak novém návrhu funkci přepínače. Pokud tedy ze serveru obdrží jakákoli data, zjistí z nich pro který uzel v senzorové síti jsou určena a tomu je zašle, naopak při obdržení paketu ze sítě je každý odeslán na server. Komunikace unitř senzorové sítě a mezi senzorovou sítí a základnovou stanicí je tedy závislá především na hlavičce daného paketu. Tento paket je tedy zpravidla tvořen pomocí hla-

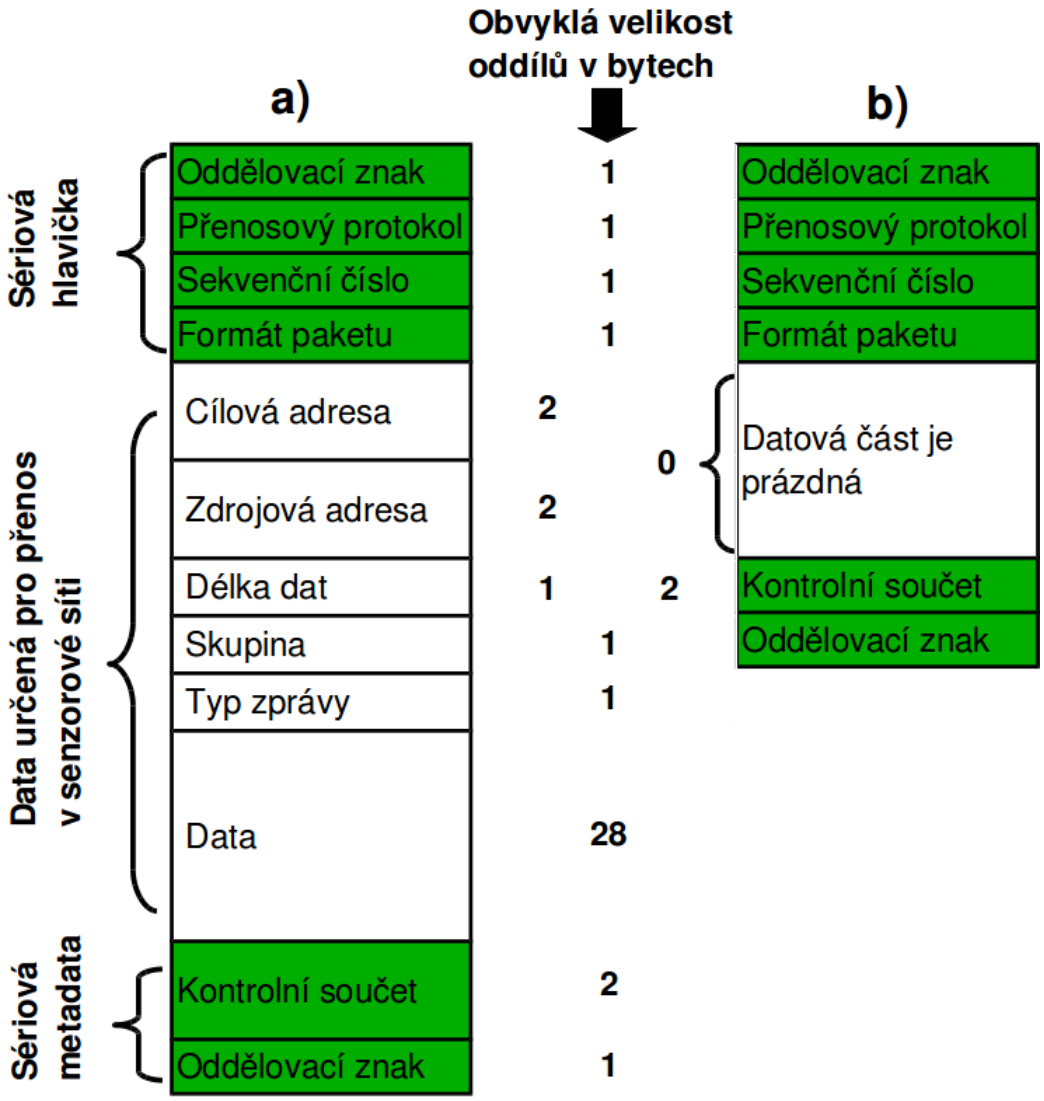
vičky popsané v souboru *tinyos-2.1.1/tos/lib/serial/Serial.h* strukturou `serial_header_t`. Ta určuje, že daný paket musí obsahovat adresu cílového uzlu s délkou 2 byty, dále pak adresu zdrojového uzlu zabírající 2 byty, délku přenášených užitečných dat s velikostí 1 byte, jež se řídí konstantou `TOSH_DATA_LENGTH` s implicitní hodnotou 28, skupinu do které náleží komunikující uzly s délkou 1 byte, typ zprávy zabírající prostor 1 byte a nakonec samotná užitečná data s délkou odpovídající konstantě `TOSH_DATA_LENGTH`. Avšak první byte datové části paketu představuje sekvenční číslo paketu v rámci jedné zprávy. To je nutné pro případy, kdy datová část zprávy překročí délku `TOSH_DATA_LENGTH - 1`. V tomto případě se tedy celá zpráva rozdělí na několik paketů. Volitelně může paket obsahovat i další metadata.

Speciálním typem výše popsaného paketu, který je vždy přenášen před odesláním sady paketů patřících do jedné zprávy je tzv. hlavičkový paket. Tento paket se od datových paketů liší obsahem svojí datové části. Ta totiž, kromě prvního bytu s pevnou hodnotou nula značící právě hlavičkový paket, obsahuje důležítá data, podle nichž rozpoznává senzorový uzel, pro který je zpráva určena, zda mu již dorazila celá zpráva nebo zda nedošlo přenosem k jejímu poškození. Datová část tedy obsahuje dva typy mechanizmů pro ověření správného doručení. Prvním je uvedení délky zprávy. A to tak, že ve druhém bytu paketu se uvede číslo odpovídající délce celé zprávy vydělené 256 a třetím paketu číslo odpovídající zbytku po dělění mezi délkou zprávy a 256. Druhým mechanizmem je provedení stejných operací nad součtem hodnot všech znakům celé zprávy. Tím se naplní další dva byty datové části paketu. Toto platí pro obnovovací a klasické zprávy určené pro přenos mezi serverem a konkrétním uzlem sítě. U zpráv nesoucích agentní kód jsou navíc přidány obě dvě výše popsané operace v podobě dvou bytů pro délku každé části agentní zprávy.

Paket určený pro přenos dat mezi senzorovými uzly, jež je popsán výše, bude tvořit datovou část paketu určeného pro komunikaci mezi základnovou stanicí a řídícím serverem. Po prozkoumání tvorby a příjmu paketů mezi serverem a základnovou stanicí je pak jasně patrné, že tyto pakety mimo datové části obsahují dále navíc hlavičku a metadata. Při hlubším zkoumání jsem došel k závěru, že formát tohoto paketu odpovídá paketu posaném v [10].

Pakety jež se budou přeposílat v nově upravené platformě mezi řídícím serverem a *GSM* modemem budou tedy mít stejný formát jako měli pakety při přenosu přes sériovou linku v původní platformě. Hlavička je tvořena podle [10] oddělovacím znakem o velikosti 1 byte, podle kterého je možné poznat začátek nového paketu. Tento oddělovací znak je nastaven konstantou `HDLC_FLAG_BYTE` ze souboru *tinyos-2.1.1/tos/lib/serial/Serial.h* na hodnotu 126. Tento knihovní soubor je implicitně součástí *TinyOS*. Další znak hlavičky představuje jakou část přenosového protokolu daný paket implementuje. Opět má délku 1 byte a může nabývat čtyřech základních hodnot popsaných v tabulce 5.3. Po tomto znaku následuje sekvenční číslo paketu o velikosti 1 byte. To znamená, že může nabývat hodnot od 0 do 255. Sekvenční číslo paketu bude využito při potvrzování o doručení paketu. Posledním znakem hlavičky je 1 byte označující formát paketu. Ten je v našem případě roven u všech paketů konstantě `TOS_SERIAL_ACTIVE_MESSAGE_ID` s hodnotou 0 ze stejného souboru jako je oddělovací znak. Tato konstanta určuje právě formát datové části celého paketu, která odpovídá formátu paketu pro přenos v senzorové síti popsaném v předchozích odstavcích. Za touto hlavičkou tedy následuje dříve popsaná datová část a po ní část s metadaty. Ta je tvořena 2 byty, kde je uveden kontrolní součet paketů od bytu určujícího v hlavičce odpovídající protokol paketu až po poslední byte datové části paketu. Kontrolní součet je možné vypočítat pomocí funkce `crcByte` opět knihovní součástí *TinyOS* v souboru *tinyos-2.1.1/tos/system/crc.h*. Celý paket je zakončen stejným oddělovacím znakem jako se nachází na začátku celého paketu. Celou strukturu paketu popsaného v tomto odstavci je možné vidět na obrázku 5.4

v části označené jako a).



Obrázek 5.4: Formáty paketů pro komunikaci mezi základnovou stanicí a serverem: a) Obecný formát paketů pro přenos všech typů zpráv. b) Formát potvrzovacího paketu.

Jediným formátem paketu, jež se však liší od výše popsaného a to pouze ve svojí datové části, je paket zasílaný základnovou stanicí jako potvrzení o úspěšném příjmutí paketu senzorovým uzlem. Tento paket je tedy význačný hlavně tím, že jako sekvenční číslo je použito sekvenční číslo paketu, který byl úspěšně doručen. A další významným znakem je absence celé datové části paketu. Struktura tohoto paketu je zobrazena na obrázku 5.4 v části označené jako b).

Formáty paketu, které jsou popsány v předchozích odstavcích, však nezaručují unikátnost výskytu počátečního a koncového oddělovacího znaku od ostatních částí paketu. Jelikož je při přenosu a zpracování tohoto formátu po jednotlivých bytech z modemu na senzorový uzel využito sériového rozhraní, je žádoucí zajistit tuto unikátnost z důvodu rozlišení začátku a konce přenášeného paketu. Navíc s touto unikátností počítá i samotná aplikace na jednotlivých uzlech senzorové sítě platformy *WSageNt* při příjmu a odesílání paketu.

Konstanta	Význam paketu v protokolu
SERIAL_PROTO_ACK	Paket představuje potvrzení o doručení.
SERIAL_PROTO_PACKET_ACK	Paket bude po doručení potvrzen příjemcem.
SERIAL_PROTO_PACKET_NOACK	Paket nebude po doručení potvrzen příjemcem.
SERIAL_PROTO_PACKET_UNKNOWN	Paket neimplmentuje žádnou část protokolu.

Tabulka 5.3: Význam konstant určujících jakou část komunikačního protokolu paket implementuje. Konstanty jsou uloženy v knihovním souboru *TinyOS tinyos-2.1.1/tos/lib/serial/Serial.h*.

Podle zdroje [10] se unikátnost tohoto znaku zajišťuje nahrazením dvěma byty, kde první odpovídá konstantě HDLC_CTLESC_BYTE s hodnotou 125 a druhý byte bude mít přímo hodnotu 94. Aby však při zpětném získávání dat z paketu bylo možné tuto náhradu rozpoznat a navrátit původní hodnotu, je nutné z počátečního paketu přijatého od modemu před odesláním odstranit také byty, jejichž hodnota odpovídá konstantě HDLC_CTLESC_BYTE. Jelikož by pak mohla nastat situace, kde se provede nahrazení popsané dvojice bytů, která však byla součástí původního paketu. Řešení je provedeno podobně jako u předchozího případu. Tudíž byty původního paketu s hodnotou HDLC_CTLESC_BYTE jsou nahrazeny dvojicí, kde první byte bude stále roven HDLC_CTLESC_BYTE, ale druhý byte bude mít hodnotu 93.

5.2.2 Průběh komunikace mezi uzlem senzorové sítě a serverem

Pro návrh celého komunikačního protokolu mezi senzorovou sítí a serverem jsem stejně jako u formátu paketů vycházel z původního protokolu pro přenos dat mezi obecným uzlem senzorové sítě a řídícím počítačem platformy *WSageNt*.

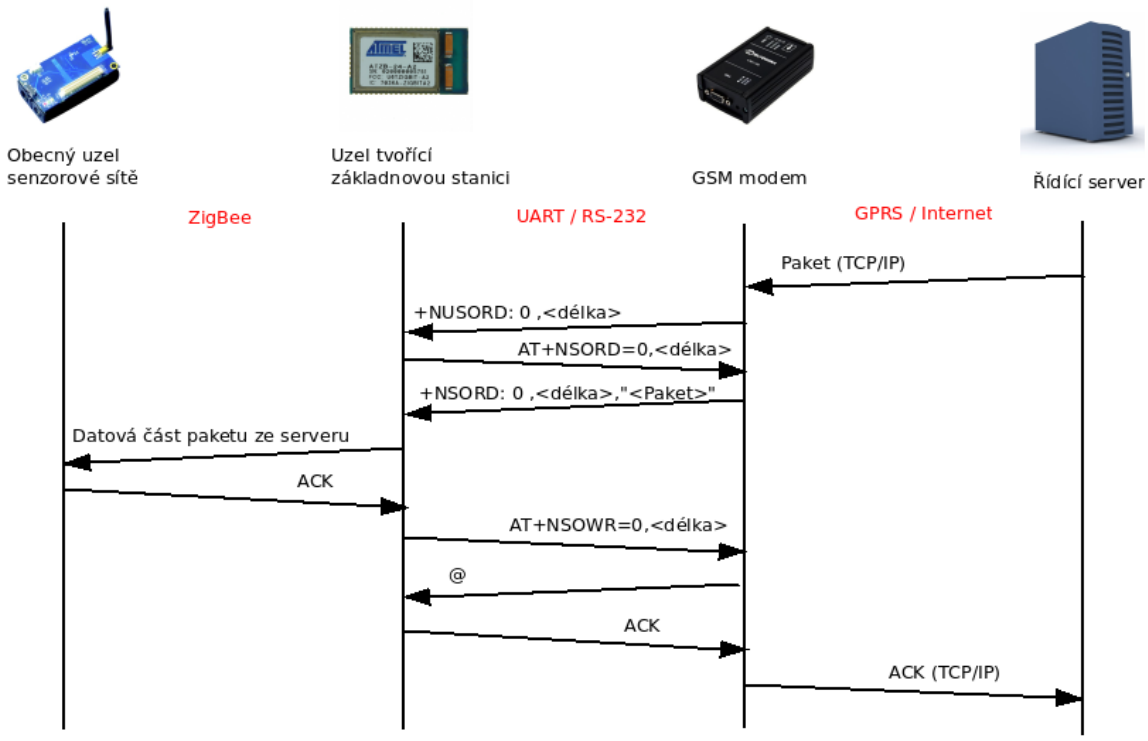
Ve všech částech původní platformy, jež je rozšiřována touto diplomovou prací, se používá shodný model komunikace. Tento model má dvě základní specifika. Prvním je, že se před odesláním vlastních dat dané zprávy přenáší speciální hlavičkový paket. Ten obsahuje základní informace o zprávě pro zajištění spolehlivé komunikace. Druhou významnou částí komunikačního protokolu je rozdělení původní zprávy do sady paketů, podle velikosti jejich datové části popsané v předchozí podkapitole. Vlastní přenos tedy probíhá tak, že iniciátor komunikace odešle v první řadě na cíl hlavičkový paket a čeká na potvrzení o bezchybném přenosu tohoto paketu od cílového uzlu. Po tomto potvrzení odešle první datový paket a opět čeká na potvrzení jeho doručení. Takto se komunikace opakuje tak dlouho, než dojde k potvrzení posledního paketu zprávy, čímž se zakončí přenos celé jedné zprávy. Pokud by v některém případě nedošlo k doručení potvrzení o přenosu ve stanoveném limitu, daný paket se zašle znovu. Jestliže se neprovede potvrzení ani na tento přenos v ještě delším časovém limitu, prohlásí se cíl jako nedostupný. Tímto způsobem tedy probíhá komunikace ve všech částech původní platformy a jinak tomu nebude ani po její úpravě s rozšířením o *GSM* modem.

Komunikační protokol, jež bude využíván v této práci a opět vychází z původní platformy, obsahuje dva základní mechanizmy pro provedení spolehlivého přenosu dat. Prvním je možnost zajištění správného pořadí paketů při sestavování jedné zprávy. To je umožněno pomocí číslování paketů na nejnižší úrovni. Tím se myslí uvedení pořadového čísla přímo jako první byte datové části paketu. Ze získaných čísel je tedy možné sestavit původní zprávu. Druhým mechanizmem je kontrola integrity zprávy. Pro tuto kontrolu zprávy byl zaveden hlavičkový paket, jenž nese čtyři základní kontrolní byty. Hlavičkový paket je podrobněji popsán v kapitole 5.2.1. Poté co je podle délky zprávy uvedené v tomto paketu

doručen paket odpovídající poslednímu, provede se kontrola integrity. Ta se provádí porovnáním součtu znaků datové části všech přijatých paketů s hodnotou získanou z hlavičkového paketu.

Nejprve zde provedu popis toho jak bude probíhat komunikace ze strany řídícího serveru na senzorový uzel. Pokud je přes webové rozhraní odeslána žádost o přenos nějaké zprávy na jeden z uzlů senzorové sítě, vytvoří se nejdříve základní formát zprávy ze vstupních komponent webového rozhraní. Takto vytvořená zpráva je odeslána na server. Zde ho příjme program jež představuje aplikační logiku serverové části platformy. Zde se zpráva rozdělí na předem určené části a z nich se vygeneruje speciální formát dat popsaný v kapitole 5.2.1. Poté se provede postupné odeslání paketů dané zprávy společně s uvozujícím paketem na již dříve připojený *GSM* modem pomocí protokolu *TCP/IP*. Aplikace následně čeká na příjem potvrzení jednotlivých paketů od modemu. V případě, že do určeného limitu nedorazí potvrzení o doručení opakovaně odesílaných paketů zprávy, vyresetuje se spojení do stavu naslouchání na daném portu.

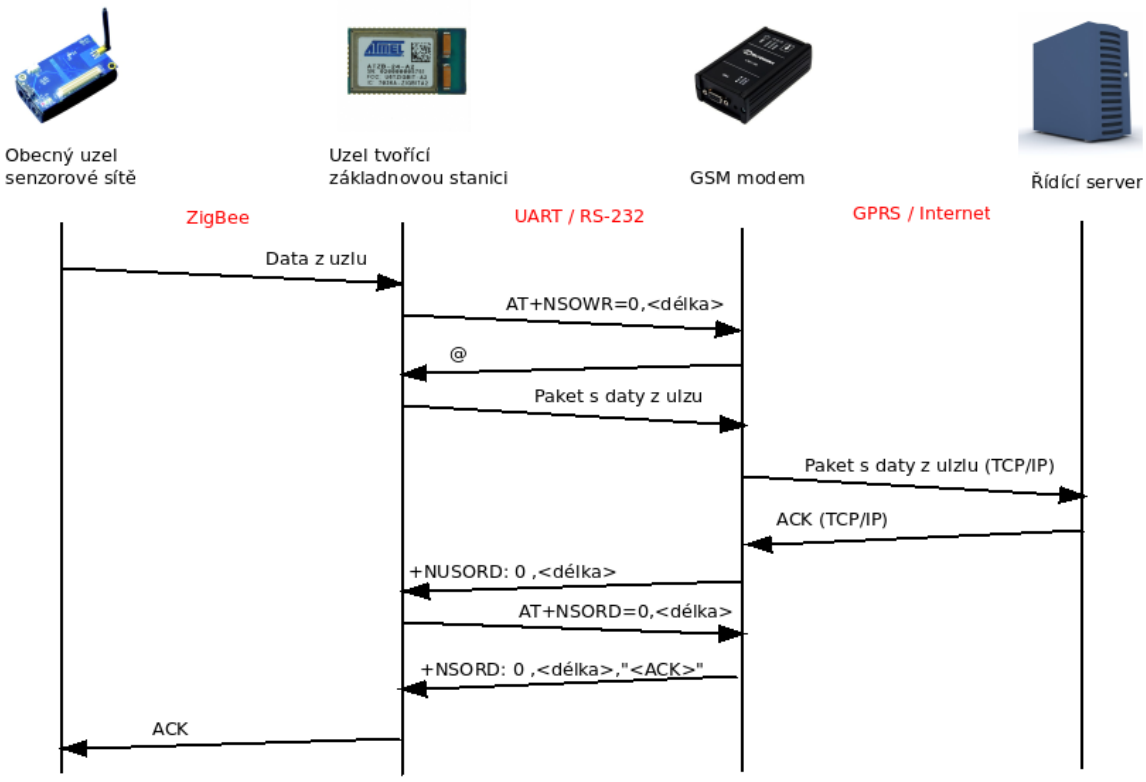
Pokaždé, kdy je na *GSM* modem doručen příchozí paket ze serveru, odešle se na senzorový uzel, představující základnovou stanici, *AT* zpráva o příchodu dat dané délky. Poté co se základnová stanice o datech dozví, pošle na modem AT příkaz s žádostí o přečtení příchozích dat. Modem data stanici přes rozhraní *RS-232* pošle. Ta je hned zpracuje na formát paketu určený pro přenos v senzorové síti popsané v kapitole 5.2.1 a pak odešle na určený uzel sítě. Poté co uzel příjme neporušený paket, odešle přes základnovou stanici na server potvrzení o bezchybném doručení. V opačném případě nepošle nic. Potvrzující zpráva se na základnové stanici upraví a odešle pomocí AT příkazů ve správném formátu přes modem na server. Celá tato komunikace je vidět na obrázku 5.5.



Obrázek 5.5: Průběh přenosu datového paketu od serveru na určený uzel senzorové sítě. ACK značí paket potvrzující doručení.

Komunikace iniciovaná senzorovým uzlem ve směru na server bude probíhat v podobném duchu. Tohoto typu komunikace se v platformě využívá například, když bude chtít zdělit senzorový uzel serveru naměřenou hodnotu teploty, kterou má odesílat podle agentního kódu na server každou půlhodinu. U tohoto typu komunikace nejprve pro zprávu určenou serveru sestaví uzel hlavičkový paket, který odešle na uzel s adresou odpovídající jedné. Na této adrese totiž působí v platformě *WSageNt* základnová stanice. Poté čeká senzorový uzel na potvrzení o přijetí paketu od serveru. V případě, že potvrzení dorazí, může postupně sestavovat pakety získané rozkladem původní zprávy a ty odesílat na základnovou stanici. To však vždy až po té, kdy získá od serveru potvrzení o přijetí předchozího paketu.

V případě, že na základnovou stanici dorazí paket ze senzorové sítě, je rozšířen na formát určený pro přenos na řídící server. Následně se odešle pomocí AT příkazu na *GSM* modem. Ten příchozí paket pomocí *TCP/IP* spojení odešle na řídící server. Na něm se zjistí zda paket dorazil přes síť Internet v pořádku pomocí ověření kontrolního součtu v metadatech paketu. V kladném případě se dále zjistí o jaký typ zprávy se jedná. Jestli nejde o zprávu potvrzující doručení dat na uzel, odešle se na *GSM* modem zpět paket s formátem potvrzující správné doručení. Pak tedy modem odešle na základnovou stanici AT upozornění o příchozích datech ze sítě. Tyto data si stanice pomocí AT příkazu vyžádá. Po jejich přijmutí v podobě vyžádané AT odpovědi, data z paketu zpracuje a převede je na tvar pro mezi uzluvou komunikaci. Tento paket pak odešle na daný uzel. Zde se paket příjme a může dojít k odeslání dalšího v pořadí. Průběh celé komunikace při přenosu dat z uzlu senzorové sítě na řídící server je ukázáno na obrázku 5.6.



Obrázek 5.6: Průběh přenosu datového paketu z uzlu senzorové sítě na server. ACK značí paket potvrzující doručení.

5.3 Návrh úpravy původní platformy

Návrh úprav původní platformy je možné rozdělit na dvě základní části. První částí je návrh rozšíření aplikace určené pro základnovou stanici. Zde je nutné navrhnout takové rozšíření, které zajistí spolehlivou a dostatečně rychlou komunikaci mezi vybraným uzlem senzorové sítě, jež bude tuto základnovou stanici představovat a *GSM* modemem pomocí sériového rozhraní obou součástí. Jedním z hlavních problémů, které bude muset základnová stanice řešit je ošetření stavu kdy probíhá synchronní komunikace s modemem a ve stejný okamžik dorazí asynchronní zpráva. Toto může nastat, když na modem budem odesílat pomocí AT příkazů data určená pro server, ale mezitím na modem dorazí data od serveru, o kterých nás modem bude asynchroně informovat.

Druhá část představuje návrh pro úpravu programu představujícím aplikační logiku webového rozhraní řídícího serveru. Místo původního modelu komunikace, kde data určená pro základnovou stanici byla přenášena sériovou linkou, je nutné vytvořit nový model pro přenos dat pomocí soketů. Tento návrh musí zaručovat schopnost obnovit spojení při výpadku základnové stanice a ověřování integrity dat přenášených mezi senzorovou sítí a serverem. Dále bude muset návrh řešit analýzu paketu tak, aby z něj získal potřebná data pro databázi a webové rozhraní.

5.3.1 Rozšíření aplikace BaseStation

Toto je nejdůležitější část návrhu celé práce. Od ní se bude odvíjet, jak rychle bude pracovat základnová stanice, dále jak dlouhou budou mít životnost baterie v ní a také jak bude snadné použít nové rozšíření v původní aplikaci. Po promyšlení celé problematiky a zjištění všech okolností z předchozích kapitol jsem se rozhodl vytvořit návrh rozhraní pro ovládání *GSM* modemu se třemi základními vrstvami v podobě několika modulů, jak jsou známy z jazyka *nesC*. Výsledkem tohoto rozšíření musí být samostatně použitelná aplikace, jež bude mít přístupnou funkci pro odeslání správného formátu paketu na server přes modem. Dále pak funkci, která rozpozná celý správně přijatý paket od serveru a oznámí to pomocí signálu. V neposlední řadě funkci, jež po startu aplikace nastaví modem pro *GPRS* přenosy.

Jelikož aktuální práce je vytvářena pro specifický *GSM* modem a přistupujeme na něj přes sériové rozhraní RS-232, první vrstva bude představovat nadstavbu nad použitým rozhraním. Tento způsob je výhodný hlavně z hlediska budoucího použití jiného typu modemu, který lze ovládat například stejnými příkazy, ale jeho datové připojení je provedeno přes jiný typ rozhraní jako například *I2C* nebo *USB*, které může být přímo součástí nějakého senzorového uzlu plánovaného pro funkci základnové stanice. V takovém případě by pak při nasazení takovéhoto modemu stačilo provést pouze změnu první vrstvy této aplikace. První vrstva tvořená modulem *ModemUARTInterface* tedy umožňuje přístup vyšší vrstvě k operacím pro odeslání a příjem holých dat z rozhraní *UART* pomocí systémového modulu *Atm128Uart0C*.

Další vrstva představující modul *GSMModemCommandP*, která tvoří jádro návrhu, pracuje jako generátor a dekodér AT příkazů podle požadavků nejvyšší vrstvy. Obsahuje tedy příkazy, které se starají o inicializaci modemu, jeho přihlášení do *GSM* sítě a nastavení *GPRS* po spuštění celé základnové stanice. Pokud se inicializace modemu nedaří, nastaví se časovač a inicializace neprovedené části nastavení se po vypršení časovače opakuje. Toto se děje až do doby než se inicializace provede. Dále tato vrstva umožňuje odesílat a přijímat data přes *GPRS*, odmítat příchozí hovory, jelikož ty nejsou pro tuto práci podstatné a aplikaci by jen zpomalovali a v neposlední řadě mazat příchozí *SMS* zprávy, aby nedocházelo

k přeplnění paměti *SMS*. To samozřejmě pouze v případě, že jsou nějaké *SMS* zprávy na *SIM* kartě uloženy.

Dále musí být základnová stanice schopna pomocí této vrstvy zpracovávat veškeré příchozí AT odpovědi od modemu. Každý příchozí znak z modemu bude uložen do zásobníku. Tento zásobník bude pak nezávisle na příchozích znacích čten a překopírováván do dalšího bufferu, u kterého pak dojde po přidání každého znaku k analýze o jakou zprávu se jedná. Po rozpoznání se ze zprávy získají potřebná data, jež se mohou dále použít v aplikaci popřípadě předat vyšší vrstvě.

U odpovědí, které jsou reakcí na nějaký požadavek od základnové stanice, nedochází k problému přečtení nevyžádané (asynchronní) zprávy od modemu. Ale jak již bylo zmíněno dříve v kapitole 4.3, k problému může dojít pokud nějaká nevyžádaná zpráva od modemu dorazí v době, kdy probíhá mezi uzlem a modemem synchronní komunikace. Pokud bude o něco žádat základnová stanice, bude vždy upřednostněna odpověď na tento dotaz, před všemi nevyžádanými zprávami. Pokud tedy v okamžiku čekání na vyžádaný dotaz přijde nějaká nevyžádaná zpráva, tato událost se poznačí do předem určeného zásobníku a hned po přijmutí posledního znaku vyžádané zprávy se provede obsloužení nevyžádané zprávy. V případě, že nevyžádaná zpráva dorazí v čase, kdy žádná vyžádaná odpověď očekávána není, dojde k okamžitému zpracování příchozí zprávy a rázem se komunikace mění na synchronní.

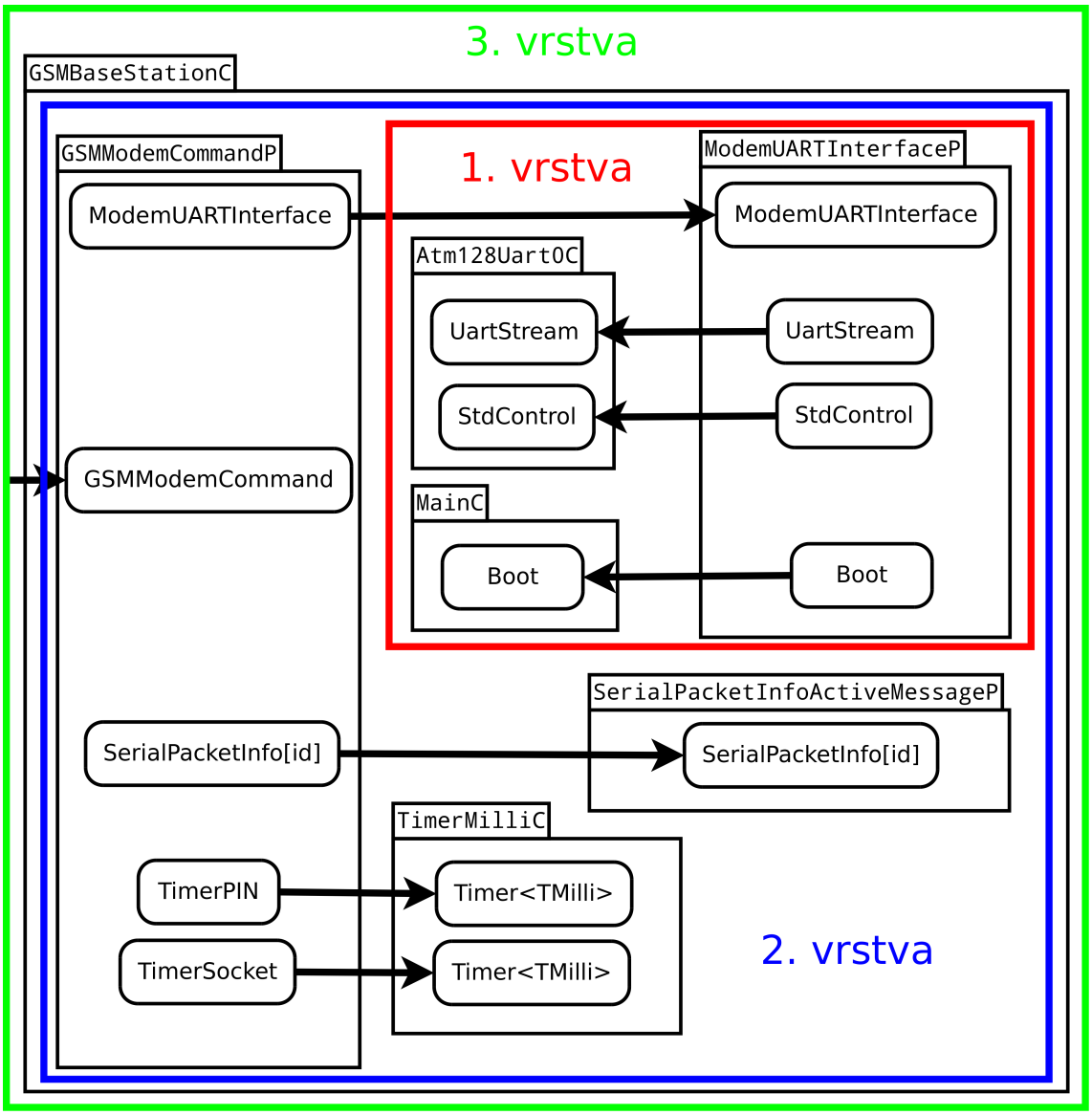
Další částí, kterou musí návrh této vrstvy řešit je zpracování datového paketu z vyžádané AT zprávy. Po přijmutí kompletní AT zprávy od modemu obsahující data ze serveru dojde k jejich výběru z AT zprávy a následně jsou tato data zpracována a předána vyšší vrstvě. Zpracování spočívá ve vytvoření formátu paketu, jež je právě určen ke komunikaci v senzorové síti, tak jak je to popsáno v kapitole 5.2.1. Naopak pokud bude zavolána metoda pro odeslání dat na server, tak se z dat pro odeslání a údajů o odesílajícím uzlu sítě vytvoří paket s formátem pro komunikaci s řídícím serverem. Ten se dále zaobalí do AT příkazu a odešle se s využitím nižší vrstvy na modem.

V neposlední řadě je nutné pomocí této vrstvy řídit spojení s *GSM* modemem. Pokud se základnové stanici po jejím zapnutí nepodaří připojit na řídící server, nastaví se časovač a o připojení se znovu pokusí po vypršení daného časového limitu. V případě neúspěchu se tato akce opakuje stále dokola. A pokud dojde k výpadku spojení se serverem, stanice se pokusí o uzavření soketu a opětovné obnovení spojení. Základnová stanice rozpozná výpadek tak, že jí od serveru dorazilo oznámení o uzavření soketu nebo pokud jí nedorazí do určeného časového limitu potvrzení o přijetí zprávy serverem.

AT příkazy a další globální konstanty, jako *PIN* pro přihlášení do sítě *GSM* nebo IP adresa a port, kde běží aplikace řídícího serveru, budou uloženy ve speciálním hlavičkovém souboru. Při použití jiného modemu by pak stačilo v této vrstvě provést pouze záměnu AT příkazů za jiné, které jsou schopny nový modem ovládat. V případě naprosto odlišného přístupu k ovládání modemu je možné nahradit celou vrstvu. Ve druhé vrstvě tedy probíhá komunikace po rozhraní *UART* s využitím nižší vrstvy na úrovni AT příkazů popsaných v kapitole 4. Zároveň se zde provádí analýza příchozích dat od serveru a sestavování paketů, jenž jsou určeny pro řídící server.

Poslední vrstva tvořená rozhraním modulu *GSMBaseStationC* zastřešuje celý *GSM* modem a poskytuje přístupné rozhraní, přes které se snadno provede inicializace modemu a odeslání či příjem dat bez vědomí toho, co se děje na nižších vrstvách. V původní aplikaci *BaseStation* tedy bude stačit nahradit funkce pro odeslání a příjem dat funkcemi z této vrstvy a navíc po spuštění provést inicializaci *SIM* karty a *GPRS* spojení. Výsledná struktura návrhu aplikace a propojení jejich modulů pomocí vstupních a výstupních rozhraní je

vidět na obrázku 5.7.



Obrázek 5.7: Propojení navržených modulů pomocí jejich rozhraní.

5.3.2 Úprava aplikační logiky na straně serveru

K dovršení návrhu celé komunikace je zapotřebí zajistit komunikaci základnové stanice s rozhraním na serverové stanici přes Internet. Pro vytváření spojení lze uvažovat dva základní přístupy.

První možností je, že by základnová stanice vzhledem k síti měla statickou a navíc i veřejnou IP adresu. Dané řešení přináší hlavní výhodu v tom, že by se prakticky nemusela upravovat serverová část aplikace. Tuto zkutečnost jsem zjistil při hlubším zkoumání třídy *Comm*, jejíž instance se používá v hlavním programu celé aplikace *BSComm*. O samotnou komunikaci v dané třídě se stará instance třídy *MoteIF* [23]. Do té se totiž v konstruktoru nastaví pomocí parametru typu *PhoenixSource* port, platforma a také rozhraní, se kte-

rým bude vlastně probíhat komunikace. Podle zdroje [23] by měla instance této třídy být schopna nejen komunikovat pomocí sériového rozhraní počítače, jak tomu bylo v doposud používaných aplikacích, ale také pomocí *TCP* protokolu. Podle stránek 3 se také samotná aplikace *BSComm* spouští s prvním parametrem určujícím právě rozhraní, port a platformu se kterou se bude komunikovat. A tento parametr se předá vytvářené instanci třídy *Comm* do jejího konstruktoru. Z uvedených zjištění jsem vyvodil, že pro správné připojení tedy stačí zadat správný parametr při spouštění zmíněné aplikace. Po delším úsilí se mi podařilo najít webové stránky [22], kde jsou uvedeny možnosti pro nastavení různých zdrojů pro připojení k základnové stnici. Podle zjištěného parametru jsem tedy sestavil příkaz pro spuštění rozhraní *BSComm* v následujícím formátu:

```
java -jar BSComm.jar network@<IP BASESTATION>:<ČÍSLO PORTU> 7777
```

V daném příkazu by se místo `<IP BASESTATION>` zadala IP adresa *GSM* modemu a místo `<ČÍSLO PORTU>` by se uvedlo číslo portu, na kterém by modem očekával příchozí připojení od serveru. Pro výše uvedené řešení by však musela být pro *GPRS* statická veřejná IP adresa podporována mobilním operátorem vybrané *GSM* sítě, což není u nás samozřejmostí. Nyní v České republice podporuje danou službu pouze společnost Telefónica Czech Republic. Je však pouze na vyžádání a navíc je ještě zpoplatněna další částkou oproti základnímu tarifu. Další problém přináší stav, kdy se spouští *BSComm* a přitom není základnová stanice zapnuta či není momentálně dostupná. V takovém případě by se aplikace zbytečně pokoušela o připojení.

Proto je možné použít ještě druhou variantu návrhu, kde by se tyto problémy neobjevily. Sice bude v implementaci nutná rozsáhlejší úprava kódu aplikace *BSComm*, ale zase se vyvarujeme obou zmíněných problémů. Při návrhu tedy dojde k vytvoření nové aplikace *IPBSComm*, která převezme určitou část tříd z aplikace *BSComm* a navíc do ní bude doimplementováno několik nových, které budou zajišťovat správnou komunikaci pomocí soketů se základnovou stanicí.

Z pohledu této varianty se tedy bude předpokládat, že připojení se vytvoří až po zapnutí základnové stanice a jejím pokusu o připojení na předem určený server. Zde totiž bude po spuštění čekat aplikace *IPBSComm* na příchozí připojení na daném portu. Až teprve po té, co ze základnové stanice dorazí na server žádost o připojení, se vytvoří komunikační soket a bude tím umožněna komunikace na obou stranách celé aplikace *WSageNt*. K tomuto účelu bude vytvořena pro serverovou část nová třída *IPComm*, jež se právě bude starat o vytvoření spojení. Tato nová třída nahradí původní třídu *Comm* pro zasílání a příjem zpráv. Nebude již tedy zapotřebí pro spuštění třídy *IPBSComm* volat parametr určující, kde je připojena základnová stanice, ale místo něj bude parametr `<PORT SERVERU>` nastavující port serveru, kde se čeká na připojení. Tvar pro spuštění tedy bude:

```
java -jar IPBSComm.jar <PORT SERVERU> 7777
```

Pokud by během komunikace došlo k výpadku spojení, aplikace *IPBSComm* by se opět uvedla do stavu před vytvořením spojení se základnovou stanicí. To znamená pokusila by se uzavřít komunikační soket, kudy se přenášela data a uvolnit veškeré prostředky, jež byly využívány pro komunikaci. Poté by čekala na přijetí oznámení o novém pokusu o připojení od základnové stanice, aby mohla vytvořit soket pro opětovný přenos dat. Za výpadek se bude považovat to, když základnová stanice neodpoví ve stanoveném limitu na žádost od serveru nebo pokud od základnové stanice dorazí na server oznámení o uzavření soketu *GSM* modemem.

Celá aplikace *IPBSComm* bude hned po svém startu tedy naslouchat na připojení od modemu. Až se tak stane bude stejně tak, jak to dělala původní aplikace *BSComm*, naslouchat příchozím paketům z webového rozhraní. Tyto pakety zanalyzuje a poté znich vytvoří vhodný formát paketů, jež jsou určeny pro základnovou stanici, na niž je pak může odeslat. Samozřejmě bude muset být také schopna přijímat příchozí pakety od modemu. Ty pak opět analyzovat a data, které z nich získá uložit do databáze nebo popřípadě předat webovému rozhraní. Do databáze bude ukládat pakety typu obyčejná zpráva od senzorového uzlu a agentní zprávy. Pomocí potvrzovacích zpráv zobrazí ve weobovém rozhraní například zprávu o úspěšném odeslání dat na senzorový uzel. V případě zjištění, že základnová stanice neodpovídá na dotazy do vypršení určeného časového limitu, prohlásí základnovou stanici za nedostupnou a vyresetuje se do stavu, čekání na příchozí připojení od základnové stanice.

Kapitola 6

Fyzické zhotovení základnové stanice a implementace komunikace

Implementaci celého rozšíření a úpravy původní platformy *WSageNt*, tak aby splňovala požadavky návrhu, je možné rozdělit do tří základních částí. Toto rozdělení odpovídá tvorbě samostatně funkčních a také fyzicky jednotlivě odělitelných částí platformy.

První částí, jež se bude týkat tvorby hardwaru, je popis zhotovení a osazení desky plošného spoje pro propojení senzorového uzlu, představující základnovou stanici a *GSM* modemu. Tento můstek pak bude řešit napojení komunikačního rozhraní *UART* senzorového uzlu na rozhraní modemu *RS-232*.

Další části se týkají softwarového vývoje platformy. Jako první z nich bude popsána implementace pro novou aplikaci základnové stanice, tak aby mohla řídit modem a pomocí něj odesílat data na řídící server umístěný v síti Internet. Samozřejmě bude umožňovat i obrácenou komunikaci.

Poslední částí je provedení popisu implementace úpravy stávající aplikace *BSComm*, jenž představuje aplikační logiku pro webové uživalské rozhraní *Control Panel*. Úprava podle návrhu z kapitoly umožní komunikaci se základnovou stanicí senzorové sítě, která má přístup k internetovému připojení, v našem případě přes síť *GPRS*.

6.1 Propojovací deska plošného spoje

Podle návrhu zapojení z kapitoly . Bylo v programu *EAGLE*, určeném pro tvorbu schémat, vytvořeno schéma plošného spoje představující převodní můstek mezi rozhraním senzorového ulzu a modemu.

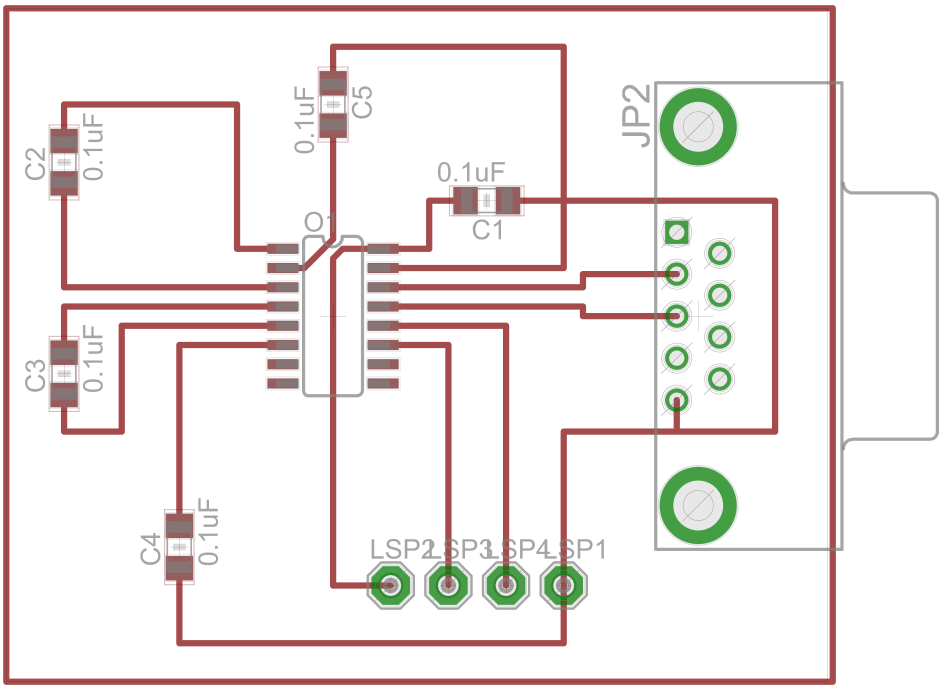
V první fázi se však muselo vytvořit v programu *EAGLE* obecné schéma zapojení. K tomu došlo již v době návrhu popsaném v kapitole . Při jeho tvorbě stačilo najít v knihovnách vybraného programu potřebné součástky, vložit je do navrhovacího okna a správně podle zjištěných údajů propojit. Seznam součástek potřebných k osazení a výrobě je uveden v tabulce 6.1.

Ve druhé fázi bylo potřeba přepnout program do módu tvorby desky plošného spoje. Jelikož nejde v zásadě o žádné složité schéma, pro tvorbu jsem zvolil jednovrstvou desku plošného spoje. Poté byla automaticky z obvodového schématu vygenerována předběžná forma výsledné desky. Ale u ní se překrývali cesty a součástky. Proto bylo potřeba nejprve rozhodnout, jak asi na výslednou desku nejlépe součástky umístit, aby se cesty pokud možno nekřížily a zároveň výsledná deska bylo co nejmenší. Poté co se tento problém podařilo

Označení součástky	Počet kusů	Cena / ks	Cena
Obvod SP3232EBCN-L	1 ks	60 Kč	60 Kč
Kondenzátor SMD 100N X7R 1206	5 ks	0,9 Kč	4,5 Kč
Konektor D-SUB-09/90° male	1 ks	14,9 Kč	14,9 Kč
Konektor DF14-5S-1.25C	1 ks	9,8 Kč	9,8 Kč
Cena celkem			89,2 Kč = 90 Kč

Tabulka 6.1: Seznam součástek pro zhotovení převodního můstku.

vyřešit, stačilo obtáhnou cesty pevnou čarou a deska byla hotova. Výsledné schéma je možné vidět na obrázku 6.1. Před finálním zhotovením, bylo ještě třeba vypnout zobrazení různých vrstev, představující například obrys součástek a podobně.



Obrázek 6.1: Deska plošného spoje na propojení modemu a uzlu FITmote.

6.2 Aplikace pro základnovou stanici

V této části implemetace bylo třeba provést záměnu funkcí implmentujících komunikaci s řídícím počítačem v aplikaci `BaseStation`. Původní funkce používané k tomuto účelu v souboru *BaseStationP.nc*, totiž sloužili pro komunikaci přes sériové rozhraní přímo do počítače. Nové musí být schopny pomocí sériového rozhraní řídit *GSM* modem a teprve přes jeho komunikační protokol přenášet data na síť Internet k řídícímu počítači a zpět. Původní funkce navíc implicitně při příjmu prováděli konverzi příchozích paketů do datového typu `message_t` pocházející z knihovního souboru *tinyos-2.1.1/tos/lib/serial/Serial.h* z *TinyOS*, se kterým se dále pracovalo. Stejně tomu je i pro obrácený směr, kde naopak tento datový typ převádějí na tvar paketu popsaný v kapitole 5.2.1.

Pro řešení popsaných problémů bylo tedy zhotoveno několik modulů v jazyce *nesC*, jež

slouží pro komunikaci s modemem. Implementace je provedena podle návrhu z předchozí kapitoly. Proto je tedy možné rozdělit i tyto moduly do tří základních vrstev podle jejich použití, jak tomu bylo v návrhu.

6.2.1 Modul pro přístup k rozhraní UART

První vrstva implementuje nadstavbu nad použitým komunikačním rozhraním *UART* ve směru na *GSM* modem. Tato vrstva je tvořena dvěma soubory. Prvním z nich je `ModemUART-Interface.nc`. Ten představuje rozhraní, které bude využito ve vyšší vrstvě pro přístup k sériové lince daného ulzu. Druhým souborem je `ModemUARTInterfaceP.nc`, kde se nachází modul tvořící aplikační logiku této vrstvy. Jelikož je pro přístup k sériové lince uzlu zvoleno systémového rozhraní `Atm128Uart0C`, je nutné ještě provést jeho konfiguraci.

Z důvodu použití specifického modemu, u něhož byla zjištěna v kapitole přenosová rychlost rozhraní *RS-233* 115200 b/s, bude nutné změnit v první řadě přenosovou rychlost pro rozhraní *UART* vybraného senzorového uzlu na stejnou úroveň. Pro základnovou stanici byl zvolen senzorový uzel *FITmote*, jehož nastavení přenosové rychlosti je možné najít v jeho specifických souborech rozhraní. První ze souborů, který je třeba upravit je soubor *tinyos-2.1.1/tos/platforms/fitmote/hardware.h*. V tomto souboru je třeba místo původní definice proměnné `PLATFORM_BAUDRATE` pomocí maker *#ifndef* a *#define* dát napevno správnou hodnotu přenosové rychlosti. To tedy lze provést nahrazením daných maker tímto kódem:

```
enum {
  PLATFORM_BAUDRATE = 115200L
};
```

Dále je zapotřebí také změnit hodnotu určující dobu pro přenesení jednoho bytu dat. Ta se odvíjí právě od přenosové rychlosti. A tedy pro 115200 b/s je přensová rychlost v bytech rovna této hodnotě vydělěné 8. Takže je to asi 14400 byte/s. Doba přenosu jednoho bytu je tedy rovna jedné vydělené vypočteným číslem. To je teda asi 69 μs, ale protože se však v nastaveních používá půlperioda, bude tedy konečná hodnota rovna 34 μs. Toto je nutné nastavit pro danou přenosovou rychlost v souboru *tinyos-2.1.1/tos/platforms/fitmote/chips/atm-1281/usart/Atm1281UartP.nc*. Upravená část souboru tedy může vypadat takto:

```
  if (PLATFORM_BAUDRATE == 19200UL)
    m_byte_time = 200; // 1 TMicor ~= 2.12 us, one byte = 417us ~= 200
  else if (PLATFORM_BAUDRATE == 57600UL)
    m_byte_time = 68;  // 1 TMicor ~= 2.12 us, one byte = 138us ~= 65
  else if (PLATFORM_BAUDRATE == 115200UL )
    m_byte_time = 34; // 1 TMicor ~= 2.12 us , one byte = 69 us~= 34
```

Základ modulu pro přístup k rozhraní *UART* je příkaz `initUART`, kterým se provede inicializace rozhraní a poté také povolení přerušení při příchozích datech na toto rozhraní. Dalším příkazem v modulu je `send` se dvěma parametry, z nichž první určuje data a druhý jejich délku pro odeslání na *UART*. Dále modul obsahuje asynchronní událost `receivedByte`, oznamující příchod jednoho bytu na dané rozhraní, která bude vytvářet signál `receive` pro vyšší vrstvu. Nakonec také obsahuje druhou událost generující stejnojmenný signál `sendDone`. Všechny tyto příkazy a signály poskytuje rozhraní `ModemUARTInterface`. Toho bude využívat vrstva aplikace popsaná v další části této kapitoly.

6.2.2 Vrstva aplikace pro práci s AT příkazy

Toto je nejrozsáhlejší část pro rozšíření původní aplikace základnové stanice. Přistupuje k rozhraní modulu nižší vrstvy, kde pracuje s rozhraním *UART*. A zároveň poskytuje svoje rozhraní vyšší vrstvě.

První částí tvořící tuto vrstvu je knihovní soubor *KonstantyBaseStation.h*. V něm jsou uloženy všechny konstanty pro popis stavů, jež se mohou naskytnout při běhu základnové stanice. Dále jsou zde ulženy základní konstanty jako `ipServeru` určující server a `port` nesoucí port, kam se má základnová stanice připojit. Dále je zde možné v konstantě `pin`, nastavit *PIN* kód *SIM* karty vložené do modemu. Co je však v tomto souboru asi nejdůležitější je seznam *AT* příkazů pro ovládání mdemu. Pokud by se tedy v budoucnu měnil pro danou implementaci typ modemu, stačilo by upravit tyto příkazy.

Druhá část je tvořena modulem `GSMModemCommandP`. Tento modul tedy přistupuje při své činnosti k rozhraní nižší vrstvy, ale nepracuje s neznámými daty, ale veškerou komunikaci zaobaluje do *AT* příkazů. Po nastartování činnosti tohoto modemu dojde k pokusu o přihlášení modemu so sítě. To se provádí opakovaně po vypršení časovače tak dloho, než k přihlášení dojde. Toto je zde naimplmentováno, kvůli možnosti zapnout senzorový uzel tvořící základnovou stanici dříve než modem. Pokud by tato možnost naimplementovaná nebyla, po prvním neuspěšném pokusu, už by nikdy k aktivaci modemu nedošlo. Toto první odeslání *AT* příkazu, na ktarý dříve či později obdrží senzorový uzel odpoveď, nastartuje rutinu pro zpracování příchozích dat.

Ta je asi nejdůležitěší částí implementace této vrstvy. Jde v ní o zpracování jednotlivých bytů, které příchází z modemu. Toto se děje po uskutečnění asynchronní události `receive` rozhraní `ModemUARTInterface`. Jelikož jde asynchronní o událost, je nutné její rychlé obloužení, aby nedocházelo v aplikaci k nepředvídaným situácím a chybám. Proto se v této události provede pouze uložení příchozího bytu na vstupní zásobník `bufferRec` a v případě, že již neběží událost `receiveChar` pro jeho synchronní čtení, naplánuje se její spuštění.

Událost `receiveChar` pracuje jako sada stavových automatů pro jednotlivé kroky inicializace a komunikace s modemem. Na vstupy těchto automatů se postupně přivádí obsah zásobníku `bufferIN`, do kterého se přidá před každým kolem vždy jeden znak ze zásobníku `bufferRec`. Pokud některý z těchto automatů zásobník příjme jako vyhovující, dojde k jeho vyprázdnění a daný automat se posune do svého dalšího kroku. V případě, že daný automat dorazí do svého konečného stavu, už se na jeho vstup zásobník dále nepřivádí. Základní situace, při kterých se provádí řízení pomocí automatu jsou:

- Nastavení *PIN* kódu *SIM* karty.
- Nastavení a následné načtení profilu pro *GPRS* spojení.
- Připojení se na danou *IP* adresu a vybraný port.
- Odmítnutí příchozího hovoru.
- Uzavření komunikačního rozhraní.
- Odeslání dat přes *GPRS* na síť Internet.
- Příjem dat přes *GPRS* ze síťe Internet.

Při každé z těchto situací dochází samozřejmě i k odesílání příkazů na modem, kterými se dávají pokyny, co má dál dělat. Jejich odesílání se provádí pomocí příkazu `send` rozhraní `ModemUARTInterface`. Samozřejmě i přepínání mezi těmito jednotlivými automaty

je řízeno pomocí hlavního automatu. To například znamená, že pokud se základnová stanice zapne, předpokládá se, že na modem nebyl zadán *PIN* kód a tudíž je zbytečné po spuštění kontrolovat jako první připojení k *GPRS*. A tak se komunikace započne dotazem zda je na modem zadán *PIN*. Pokud není, první automat podle protokolu popsaného v [14] provede jeho nastavení. Poté však, již tento automat není používán. Obdobně je tomu i u dalších situací. Pouze v případě provedení všech inicializačních akcí, se v každém kroku vždy musí kontrolovat, zda nepřišla nevyžádaná zpráva o hovoru, příchozích datech, *SMS* nebo zda nepřišla z vyšší vrstvy žádost o odeslání dat na server.

V tomto modulu se také provádí konverze paketů mezi formátem pro senzorovou síť a pro řídící server. Pokud tedy dorazí z vyšší vrstvy žádost v podobě příkazu `send` rozhraní `GSMModemCommand` pro odeslání zprávy formátu `message_t`, tato zpráva se musí doplnit rámcem pro přenos na server a poté teprve odeslat. Toto se provede nejprve postupným skládáním jednotlivých bytů, tvořících hlavičku budoucího paketu do výstupního zásobníku `bufferOUT`, následně se tam vloží celá příchozí zpráva a poté přidají metadata paketu. Takto se celý paket může odeslat na server. Opačnou situací je příjmutí celého paketu od serveru. V tomto případě dojde nejprve v události `receiveData`, představující automat zpracovávající příchozí data, k ověření kontrolního součtu paketu. Poté se musí získat základní tvar paketu pro senzorovou síť nakopírováním datové části serverového paketu do nového zásobníku `bufferReceive`, který je vrácen v podobě ukazatele na `message_t`, vyšší vrstvě nové aplikace jako příchozí data ze serveru.

Další funkčností, kterou bylo nutné v této vrstvě implementovat, je vytvoření a zpětné odeslání potvrzovacího paketu, při příjmu kontrolní zprávy ze serveru s cílovou adresou odpovídající základnové stanici. Proto se při každém příjmutí paketu od serveru provede kontrola, zda jeho cílová adresa neodpovídá adrese základnové stanice. V kladném případě se vytvoří potvrzovací paket přímo v této vrstvě a pomocí příkazu `send` rozhraní `ModemUARTInterface` odešle zabalený v *AT* příkazu na modem a odtud na server.

6.2.3 Rozhraní pro komunikaci přes GSM modem

Spolupráci jednotlivých modulů, jak je vyobrazena v návrhu této práce, implementuje konfigurační soubor `GSMBaseStationC`. Zde jsou na sebe navázány jednotlivé části obou vrstech popsaných výše a k nim je navíc připojeno výsledné rozhraní celé aplikace. Tento soubor tedy konkrétně provádí napojení příkazů a signálů modulu `Atm128Uart0C` na modul `ModemUARTInterfaceP` a rozhraní předchozího modulu na modul `GSMModemCommandP`. Jeho rozhraní `GSMModemCommand` je nakonec napojeno na hlavní rozhraní celé aplikace `GSMModemCommand`.

Toto rozhraní zastřešuje tedy celou aplikaci tak, že je použitelná přímo v aplikaci *BaseStation* bez větších zásahů do původního kódu. Rozhraní poskytuje příkaz `send` pro odeslání zprávy typu `message_t` na řídící server. Dále pak příkaz `start`, jež provede inicializaci modemu a nakonec příkaz `end`, ukončující práci s modemem. Navíc poskytuje události jako `receive` nesoucí zprávu ve formátu `message_t` z řídícího serveru nebo `sendDone` oznamující odeslání dat na modem. V neposlední řadě také událost `startDone` oznamující, že je modem nakonfigurován.

6.3 Aplikační logika serverové aplikace

Jak již bylo zmíněno v úvodu kapitoly, pro serverovou část platformy bylo nutné upravit aplikaci *BSComm* tvořící aplikační logiku webového rozhraní *Control Panel*. Změna zpočí-

vala především ve vytvoření nových tříd v jazyce *JAVA*, které nahradí původní, poskytující rozhraní pro komunikaci se základnovou stanicí. Před úpravou platformy šlo o komunikaci přes sériové rozhraní, nově půjde o přenosy sítí Internet.

Jelikož nemělo podle návrhu dojít k žádným změnám ve webovém rozhraní ani v aplikaci pro uzly senzorové sítě, muselo se zajistit vytváření správných formátů paketů pro přenos tak, aby je bylo poté snadné zpracovat pro základnovou stanici. Pro tuto činnost používala před úpravou aplikace *BSComm* knihovních tříd z *TinyOS*, které správné formáty paketů pro odesílání vytvářeli na nejnižší úrovni v podobě bytů při přímém přístupu k sériové lince. Stejně tak tomu bylo i v době příjmu, kde příchozí pakety dekódovali pro další zpracování do daného formátu, jež je popsán v kapitole 5.2.1.

6.3.1 Úprava třídy Main

Daná třída z balíčku `cz.vutbr.wsagent.bscomm` představuje hlavní část programu pro běh celé aplikace *BSComm* i novou aplikaci *IPBSComm*. Po jejím spuštění se provede nejříve zpracování vstupních parametrů hlavní funkce `main` celého programu. Prvním z nich je tedy podle nového návrhu číslo portu, kde aplikace bude naslouchat a očekávat připojení základnové stanice ze sítě Internet. Druhým parametrem, jehož význam zůstává shodný s původním, je číslo portu určené pro komunikaci s webovým rozhraním.

Dále se ve funci `main`, provádí inicializace třídy pro komunikaci se základnovou stanicí. Zde se nahradila třída `Comm` nově vzniklou třídou `IPComm`, jenž bude dále popsána v kapitole 6.3.2. Pro její inicializaci se v konstruktoru použije jako parametr pouze číslo, představující daný komunikační port pro základnovou stanici.

Třída `IPComm` se sama stará o navázání a řízení komunikace se stanicí a také příjem paketů z ní. Ty jsou s její pomocí zpracovány a předány do databáze, popřípadě jsou některé výsledky týkající se zpráv zaslány přímo webovému rozhraní. Jde například o zprávy týkající se úspěšného doručení paketu či vypršení časového limitu pro odpvěď na zprávu. V další části hlavního programu je po úspěšném navázání spojení se stanicí možné využívat tuto třídu pro odesílání vybraných zpráv. Tomu však předchází vytvoření komunikačního soketu s aplikací *Control Panel*. Odsud jsou zachytávány zprávy a předávány třídě `IPComm` pro konverzi do správného formátu a následné odeslání na senzorovou síť. Nejdříve se však určí zda se jedná o zprávu nesoucí agentní kód či obyčnou nebo kontrolní zprávu. Ta se tedy zaobalí pomcí instance třídy `Messenger` společně s adresou cílového uzlu a teprve poté je předána na odeslání.

Pro zjištění zda je spojení se základnovou stanicí stále aktivní, posílají se na ni, stejně jako na všechny uzly senzorové sítě přidané do databáze této platformy, kontrolní zprávy. Pokud na tyto zprávy neodpoví některý z obecných uzlů sítě, *IPBSComm* tuto zkutečnost pouze vypíše do konzole a komunikace pokračuje dál. Pokud však neodpoví základnová stanice, předpokládá se, že je nějakým způsobem nedostupná, atak se s ní ukončí spojení. Poté se celá aplikace *IPBSComm* navrátí do stavu čekání na připojení základnové stanice. Adresa základnové stanice je dána proměnnou `BASESTATION_ID` v rozhraní `InternalCommProtocol`.

6.3.2 Třída IPComm a IPServerComm

Třída `IPComm` plní stejné funkce jako v počáteční platformě plnila třída `Comm`. To znamená, že udržuje a řídí spolehlivu komunikaci se základnovou stanicí a provádí odesílání agentních, obyčejných a kontrolních zpráv přes základnovou stanici na jednotlivé uzly senzorové sítě. Zároveň ukládá příchozí zprávy do databáze a oznamuje webovému rozhraní výsledek přenosu zprávy.

Stejně jako třída `Comm`, implementuje i nahrazující třída `IPComm` třídu `MessageListener` z balíku `net.tinyos.message`. To je výhodné hlavně z důvodu snazší práce s příchozími zprávami pomocí předefinování činnosti funkce `messageReceived`. Sice bylo nutné implmentovat i třídy, jež příchozí zprávy pro tuto metodu vytvářejí, ale v rámci zachování původního modelu funkčnosti popsaného v kapitole 5.3.2, bylo vhodné tuto implementaci ponechat.

Dále se tato třída stará o základní fragmentaci dat tak, aby při délce překračující datovou část přenášeho paketu, byl zbytek zprávy uložen do zásobníku, určeném pro daný senzorvý uzel. Odtud bude po příchodu potvrzení předchozí části zprávy zaslána další. Samozřejmě také vytváří a zasílá před samotnou zprávou hlavičkové pakety. Podle nich pak mohou senzorové uzly jednotlivé zprávy identifikovat a ověřovat jejich integritu.

První fází po provedení konstruktoru třídy `IPComm` je vytvoření instance třídy `IPServerComm`. Právě pomocí této nové třídy, jež bude využívat i dalších nově vzniklých tříd, je provedena náhrada části původní aplikace, starající se o komunikaci se základnovou stanicí na nejnižší úrovni. Do konstruktoru této třídy se zadává port pro základnovou stanici a odkaz na instanci domovské třídy `IPComm`, kam budou právě vraceny příchozí zprávy. Ty se budou dále ukládat do zásobníku typu `HashMap`. Dále se do této třídy přidávají posluchači jednotlivých typů zpráv pomocí funkce `registerListener`. Ti představují jednotlivé typy zpráv jako objekty s pomocnými metodami, zděděné ze třídy `Message` pocházející z `net.tinyos.message`.

Poté se pomocí volání metody třídy `createConnect` vytváří čekací rutina na připojení senzorové sítě na server. Jakmile se připojení vytvoří, je možné volat metodu `send`, s adresou cílového uzlu sítě a druhým parametrem představující datovou část paketu. Této metody využívá třída `IPComm` ve všech svých veřejně dostupných funkcích, určených pro odesílání jednotlivých typů zpráv jako jsou `sendRefr`, `sendMsg` nebo `sendAgent`. Poslední veřejnou metodou, kterou je možné ve třídě `IPComm` ze třídy `IPServerComm` použít je metoda `close`, jež uzavře komunikaci se základnovou stanicí senzorové sítě po jejím zavolání. Toto se bude provádět při vypršení časového limitu, pro odpověď na kontrolní zprávu zaslanou na základnovou stanici. Provádí se to z důvodu, pokud by základnová stanice byla příliš zaneprázdněna a nestihla by odpovědět pouze všas. Potom by se i ona, po příchodu zprávy oznamující uzavření soketu, vyresetovala do módu připojování na server. Tímto se obnoví komunikace.

Samotná třída `IPServerComm` po připojení základnové stanice, vytvoří dva datové proudy, které budou služit pro komunikaci se sítí. Jeden ve směru od serveru a druhý na server. Poté se spustí další dvě vlákna pomocí konstrukce nových instancí tříd `IPPaketizer` a `IPSource`. Obě tyto třídy slouží pro formátování paketů a práci s oběma datovými proudy. Podrobně budou popsány v další kapitole 6.3.3. Nakonec se vytvoří instace dalších dvou nových tříd, kterými jsou `IPReceiver` a `IPSender`. Ty představují rozhraní pro odesílání a příjem dat využívajících předchozích tříd. Instance třídy `IPSender` je využita pro odeslání dat ve výše zmíněné metodě `send`. Podobně je tomu i pro třídu `IPReceiver`, která ovládá posluchače příchozích zpráv[1], kterým předává příchozí data od senzorové sítě.

6.3.3 Další třídy pro práci s pakety

Další nově přidané třídy aplikace *IPBSComm* jsou především `IPPaketizer`, `IPSource`, `IPSender` a `IPReceiver`, jež byli zmíněné již v přechozí kapitole. Třída `IPPaketizer` im-

[1] Instance tříd dědících vlasnoti a metody ze třídy `Message` pocházející z `net.tinyos.message`, jež byli použity jako parametry při volání metody `registerListener()` ve třídě `IPComm`.

plementuje tvorbu výsledného formátu paketů popsaného v kapitole 5.2.1 a určeného pro spolehlivou komunikaci se základnovou stanicí. Také umožňuje odesílání těchto paketů do určeného datové proudu typu `DataOutputStream` navázaného na daný soket a stejně tak také příjem stejného formátu ze sítě z proudu typu `DataInputStream`.

Jelikož je hlavní smyčka této třídy určená pro stálý příjem dat po bytech, běží calá třída ve vlastním vláknu. Při přijímání dat hledá odělovací znaky jednotlivých paketů pro jejich rozpoznání. Při načtení celého paketu jej uloží do zásobníku typu `LinkedList` z balíčku `java.util.LinkedList`. Ve třídě `IPSource` se pomocí neustálého volání metody `readSourcePacket` provádí načítání uložených paketů z daného zásobníku. Proto i tato třída představuje nové vlákno, které právě nezávisle na té první čte obsah zásobku, kam první data ukládá. Přičemž výsledný přečtený paket je zbaven hlavičky a metadat, určených pro spolehlivý přenos mezi senzorovou sítí a serverem. Pro odeslání paketu je ve třídě `IPPaketizer` implementována metoda `writeSourcePacket`, která se opět volá ze třídy `IPSource` v její metodě `writePacket`. Zde se paket pro senzorovu síť zaobalí do popsaného rámce a odešle se do sítě. `IPSource` slouží tedy jako rozhraní, pro odesílání a příjem dat od základnové stanice pro další třídy.

Nad těmito dvěma třídami jsou třídy `IPSender` a `IPReceiver`. Druhá jmenovaná slouží pro konverzi mezi základním formátem paketu senzorové sítě do datového typu odpovídající třídě `Message` z `net.tinyos.message`. Tento formát poté předá třídě `IPComm` do její metody `messageReceived`, kde dojde k jejímu dalšímu zpracování. Tím může být odeslání další části zprávy, jež překročila délku jednoho paketu nebo odeslání potvrzení o příjmu agentní zprávy od daného senzorového uzlu. Naopak voláním metody `send` třídy `IPSender` se provede nejdříve konverze zprávy formátu `Message` z jejího vstupního parametru na paket přenášený standardně senzorovou sítí a ten se předá dál třídě `IPSource`.

Kapitola 7

Testování výsledné aplikace

Pro testování výsledné aplikace bylo zvoleno několik základních případů vzorových modelů situací, jež se můžou v platformě *WSageNt* při běžném provozu objevit a jenž by měla zvládat platforma před svojí úpravou. První z ověřovaných případů je zjišťování dostupnosti senzorových uzlů uložených v databázi, které se provádí automaticky při neaktivitě webového rozhraní.

Další situací pro kontrolu správné funkčnosti upravené platformy, je zaslání vybraného agenta na daný uzel senzorové sítě. V tomto případě může jíz například o agenta, který bude na senzorovém uzlu rozsvicovat různým způsobem LED diody nebo zajišťovat komunikaci mezi uzly sítě, jež se může projevit opět například nějakým vizuálním efektem s LED diadami na uzlu. Poslední z vybraných situací pro otestování funkčnosti celé nově upravené platformy je záslání agenta.

Běh vybraných situací k ověření správnosti řešení byl prováděn na senzorové sítí s větším počtem senzorových uzlů typu *Iris*. Ve vybrané oblasti testování bylo také poměrně kvalitní pokrytí signálem sítě *GSM* od mobilního operátora Vodafone Czech Republic a. s. s úrvní signálu od -115 dBm až do -60 dBm. Tyto hodnoty byli získány měřením na mobilním zařízení s platformou Google Android.

7.1 Sestavení součástí platformy pro testování

Pro testování bylo použito 6 senzorových uzlů *Iris* s napájením z tužkových baterií. Uzly byly různě rozmístěny po standardním třípokojovém panelovém bytě. Dále byla sestavena a použita základnová stanice tvořená senzorovým uzlem *FITmote* a připojená přes nově vytvořený propojovací můstek s *GSM* modemem od firmy Teltonika. Na uzly *Iris* se provedlo nahrátí záladní aplikace pro senzorové uzly ze stránek [12] pomocí *USB* brány *MIB520*. A na uzel základnové stanice nově vytvořenou aplikaci pomocí programátoru *AVR JTAGICE mkII* příkazem podle části této práce 2.2.3.

Pro serverovou část platformy mi byl poskytnut školní server od mého vedoucího Ing. Jana Horáčka na veřejné internetové adrese *ihoracek.no-ip.org*. Přičemž server obsahoval již potřebný *Apache* server s aplikačním serverem *Apache Tomcat*, jež je určený pro běh webových aplikací napsaných v jazyce *JAVA*. Stejně tak již server obsahoval spuštěnou aplikaci *Control Panel*, pro vizuální správu a monitorování sítě připravenou se připojit na aplikaci *IPBSComm* na portu 7777.

Dále jsem také získal přístup k *MySQL* serveru, který je umístěn na stejném serveru jako Apache. Databáze s názvem *wsagent* určená pro správný běh platformy zde již byla

vytvořena, ale bylo žádoucí smazat z ní veškeré dříve uložené údaje. To bylo nutné hlavně kvůli hladkému průběhu testování a jednoduššího ověření, že například pomocí nově upravené platformy přibyli nějaké záznamy v databázi a podobně. Pokud by tyto tři součásti nějaký server neobsahoval, stačilo by pro jeho nakonfigurování provést kroky popsané v kapitole 3.4.

Poté bylo nutné na server nahrát také nově vytvořenou aplikaci *IPBSComm*. Nejdříve ji však bylo samozřejmě potřebné přeložit ve vývojovém prostředí NetBeans, určené právě pro vývoj *JAVA* aplikací. Pak jsem přeložený soubor umístil na moji veřejnou část školního serveru eva na adresu *http://eva.fit.vutbr.cz/ xmolak02/IPBSComm.jar*, odkud ji lze na server *ihoracek.no-ip.org* stáhnout pomocí utility `wget` po přihlášení na cílovém serveru. Nakonec tedy stačí příkazem podle kapitoly 5.3.2 *IPBSComm* spustit a testovat.

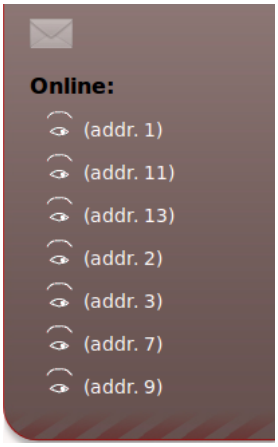
7.2 Obnovení seznamu uzlů senzorové sítě

Jak bylo posáno v kapitole týkající se původní platformy *WSageNt*, je při neaktivitě uživatelů na webovém rozhraní prováděno periodické odesílání zpráv na aplikaci *IPBSComm*. Tyto zprávy obasahující ve svojí datové části pouze znak `R` jsou postupně generovány se všemi adresami uzlů, které jsou uloženy v určené tabulce `mote` databáze `wsagent`. Tyto příchozí zprávy zabalí aplikce *IPBSComm* do určeného paketu, který odešle na základnovou stanici. Ta paket zpracuje, přeformátuje na jiný typ a ten je odeslán na příslušný uzel sítě. V případě že je uzel aktivní, odpoví na tuto zprávu potvrzením. Pokud se potvrzení dostane až na server, zobrazí se v pravém monitorvacím panelu rozhraní *Control Panel* ikona s číslem a jménem daného uzlu.

Pro otestování je tedy nejprve nutné vytvořit pomocí volby tlačítka z levého menu s názvem *ADD NEW MOTE* seznam všech uzlů senzorové sítě i se základnovou stanicí. Na obrazovce pro přidávání postačuje vyplňovat pouze pole určené pro adresu uzlu, které od sebe jednotlivé uzly rozlišuje. Toto se tedy provede pro všech 7 uzlů. Poté se může nastartovat aplikace *IPBSComm*, zapnout *GSM* modem, kam je již připojen uzel určený jako základnová stanice. A nakonec pozapínat včechny uzly senzorové sítě.

Po chvíli se začne uskutečňovat komunikace, v podobě automatického zasílání zpráv pro ověření živosti uzlů. Jednotlivé uzly na žádosti odpovídají a tím pádem získává serverová část platformy přehled o tom, které ulzy jsou na příjmu. Po odpovědích jsou vyobrazeny na weobové stránce všechny aktivní uzly sítě. Tato část testování proběhla v pořádku, všechny uzly se na webové stránce zobrazovali a to i společně s uzlem představující základnvou stanici odpovídající číslu 1. Výsledné zobrazení indikátoru připojených uzlů, je vidět na obrázku 7.1. Část výpisu výstupů aplikace *IPBSComm* při průběhu komunikace na oknu konzole je vidět v rámečku níže.

```
...
Web Client Connected. Parsing Message...
Parsing Message Successful. Sending Message to the BaseStation...
Sending Refresh message header packet
----------------
Receiving REFRESH ACK...
Transferring Refresh message Successful.
...
```



Obrázek 7.1: Indikátor zobrazující připojené uzly v rozhraní Control Panel.

7.3 Ovládání LED diod na uzlu

Pro další část testování je zvoleno příkladu, kdy se na jeden z uzlů senzorové sítě zašle agentní kód, jenž bude naslouchat příchozím zprávám od jiných agentů a na jejich základě provede rozsvícení vybrané LED diody. Pro tuto činnost se na jeden z uzlů provedlo zaslání agentního kódu uvedeného v tabulce 7.1. Tento kód představuje funkci podřízeného uzlu, který rozsvicuje svoje LED diody podle přijaté zprávy od řídícího uzlu. Po přijetí dané zprávy navíc zašle potvrzení řídícímu uzlu, který taktéž provede rozsvícení vybrané LED diody. Kód pro řídící uzel je uveden v tabulce 7.2. Ten tedy pracuje tak, že uzel nejprve do své báze znalostí uloží pořadí, v jakém budou dané LED diody blikat společně s časovým údajem určujícím také tuto dobu. Následně použije operaci unifikace nad tímto plánem znalostí, kterým se vybere jeden z údajů, který podřízenému uzlu zašle. Toto se opět po vypršení stanovené doby vybraným plánem opakuje. Tato část testování tedy měla ověřit, zda je schopna základnová stanice odeslat na jednotlivé uzly senzorové sítě agentní kód. Tyto uzly poté odešlou na základnovou stanici potvrzení o jeho příjmu. Po odeslání obou kódů z webového rozhraní na senzorové uzly se vypsalo na konzoli vždy ověření o kompletním odeslání agentů. Po chvíli také začalo na vybraných dvou uzlech sítě různé blikání LED diod. Z tohoto zjištění lze tedy usoudit, že i tato část komunikace bezchybně funguje.

Plán / Báze	**Agentní kód**

`PlanBase`	`(prijem,($(s)&(2)?(2)$(a)!(2,&2)^(blik)))`
	`(blik,(&(1)$(r,&2)&(3)$(f,&1)$(l,&3)&(2)$(r,&1)&(1)$(f,&2)$`
	`(w,&1)$(l,&3)^(prijem)))`

`Plan`	`^(prijem)`

Tabulka 7.1: Kód agenta pro podřízený uzel.

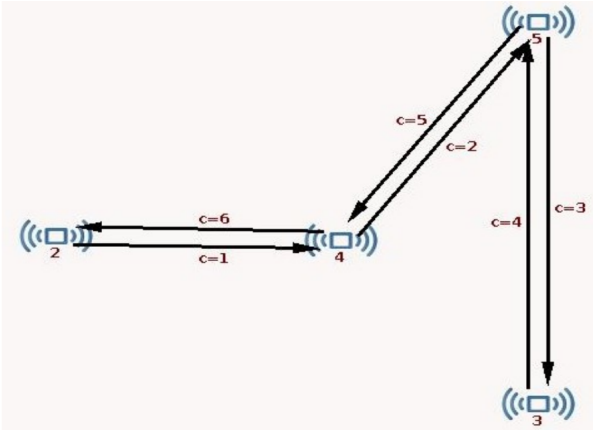
7.4 Průchod agenta sítí a získání její přibližné topologie

Tento příklad byl vybrán pro otestování příjmu zpráv ve směru ze senzorové sítě na server. Pro jeho provedení se stačí přepnout ve webovém rozhraní *Control Panel* na stránku *Tracing Agents*. Zde se musí vyplnit číslo uzlu, ze kterého bude docházet k průchodu senzorovu sítí

Plán / Báze	Agentní kód
PlanBase	(odesli,($(a)!(3,&2)&(3)$(s)?(3))) (napln,(+(led,r,600)+(led,g,700)+(led,r,500)+(led,y,800) ^(blik)#^(napln))) (blik,(&(1)*(led,_,_)&(2)$(f,&1)-&2^(odesli)&(1)$(r,&2)&(3) $(f,&1)$(l,&3)&(2)$(r,&1)&(1)$(f,&2)$(w,&1)$(l,&3)^(blik)))
Plan	^(napln)

Tabulka 7.2: Kód agenta pro řídící uzel.

a který nám sdělí výslednou trasu průchodu. Poté se ještě musí navolit libovolně číslo značící agenta a také jeho třídu. Pak už je možné tohoto agenta pomocí tlačítka *Request* odeslat do senzorové sítě. Po jeho průchodu, celou sítí a sezbírání potřebných dat, se objevila v pravém horním rohu indikátoru připojených uzlů sítě ikona značící uložení příchozích zpráv od agenta do databáze. Poté již bylo možné na výše zmíněné stránce zvolit agenta, jež byl vyslán do sítě a zobrazit cestu, kterou prošel. Tuto cestu je možné vidět na obrázku 7.2. Podle těchto zjištěných udlálostí je možné říci, že nově upravená platforma dokáže přímat také obyčejné zprávy přicházející ze senzorové sítě. Bohužel však kvůli komunikační režii, která narůstá přenosem *AT* zpráv mezi *GSM* modemem a uzlem tvořící základnovou stanici, trval přenos trasovacího agenta do sítě poměrně dlouhou dobu, díky jeho větší délce oproti testování v předchozích příkladech.



Obrázek 7.2: Zobrazení průchodu agenta senzorovou sítí.

Kapitola 8

Závěr

Podle zadáni této diplomové práce byla v první fázi zmapována oblast bezdrátových senzorových sítí. Dále pak proveden rozbor multiagnetní platformy *WSageNt*, senzorového uzlu *FITmote* a *GSM* modemu firmy *Teltonika*. Poté byl proveden návrh úpravy platformy *WSageNt*, tak aby komunikace mezi jejím řídícím počítačem a základnovou stanicí senzorové sítě probíhal přes *GSM* modem připojený ke stanici.

Na základě zjištěných informací jsem nejprve navrhl a zhotovil hardwarové propojení mezi senzorovým uzlem a modemem. Tento propojovací most řeší různé napěťové úrovně pro logickou nulu a jendičku jejich sériových rozhraní, jež standardně používají při komunikaci.

Další částí práce bylo navržení a implementace komunikačního protokolu, pro přenos dat mezi řídícím počítačem v síti Internet a základnovou stanicí pomocí *GSM* modemu. Implementace tohoto protokolu obnášela úpravu aplikace pro základnovou stanici a aplikační logiky řídícího webového rozhraní.

Po implementaci celého rozšíření platformy se při testování ověřila většina funkcí, jež ovládala i původní platforma. Při tom nebylo zjištěno žádných odchylek od chování té původní. Ověřily se přenosy všech typů zpráv a to ve směru do senzorové sítě i opačně. Jedinou nevýhodou nového řešení je pomalejší přenos a zpracování dat na základnové stanici, způsobené režií za přenos *AT* zpráv a čekání na jejich odpovědi z modemu. Bohužel byl nakonci testování nevratně poškozen prototyp senzorového uzlu *FITmote*, ale naštěstí je nově upravenou platformu možné provozovat i na jiných uzlech, jejichž základ tvoří procesor *AVR ATmega1281* jako jsou senzorové uzly *Iris* či *MICAz* firmy Crossbow.

Současný stav po úpravě platformy *WSageNt* umožňuje její nasazení v externích podmínkách mimo dosah sériového kabelu počítače. Její omezení je dáno pouze dosahem signálu zvolené *GSM* sítě. Lze tedy umístit několik senzorových uzlů společně se základnovou stanicí například v přírodě a tam sledovat třeba změny teplot z pohodlí domova prakticky kdekoli na světě.

Jako další rozšíření platformy *WSageNt* bych navrhoval především přidání nových funcí webovému rozhraní. Příkladem může být zobrazování dat ze senzorů, jako grafy s vývoji teplot během dne apod. Dále by bylo vhodné vytvořit pro uzly sítě nové senzory, například pro měření polohy *GPS* nebo tlaku vzduchu a následně práci s těmito senzory vhodně zakomponovat na serverovou část platformy.

Literatura

[1] Atmel Corporation. *JTAGICE mkII Quick Start Guide* [online]. 2006 [cit. 2011-10-24]. Dostupné z: http://www.atmel.com/dyn/resources/prod_documents/doc2562.pdf.

[2] Atmel Corporation. *Datasheed ATZB-24-A2/B0* [online]. 2009 [cit. 2011-12-29]. Dostupné z: http://www.atmel.com/dyn/resources/prod_documents/doc8226.pdf.

[3] Bradáč, Zdeněk. Bezdrátový komunikační standard ZigBee. *Automatizace* [online]. 2005 [cit. 2012-04-03]. Dostupné z: http://www.automatizace.cz/article.php?a=638.

[4] eHelp Corporation. *JTAGICE mkII User's Guide* [online]. 2002 [cit. 2011-10-24]. Dostupné z: http://support.atmel.no/knowledgebase/avrstudiohelp/mergedProjects/JTAGICEmkII/mkII/JTAGICEmkII.htm#Html/JTAGICE_mkII_Hardware_description.htm.

[5] Exar Corporation. *Datasheed SP3223E/EB/EU* [online]. 2011 [cit. 2012-01-02]. Dostupné z: http://www.exar.com/Common/Content/Document.ashx?id=619.

[6] Fábera, Vít. *Logické obvody* [online]. 2010-03-26 [cit. 2012-05-07]. Dostupné z: http://www.fd.cvut.cz/personal/xfabera/zajmove/logika/logika.pdf.

[7] Gábor, Martin. *Webové rozhraní pro sledování provozu v bezdrátových sítích* [online]. Brno: FIT VUT v Brně, 2010. Diplomová práce. Dostupné z: http://www.fit.vutbr.cz/study/DP/rpfile.php?id=8888.

[8] Gay, David, Levis, Philip, Behren, Robert von et al. *The nesC Language: A Holistic Approach to Networked Embedded Systems* [online]. 2003 [cit. 2011-11-10]. Dostupné z: http://csl.stanford.edu/~pal/pubs/nesc-pldi03.pdf.

[9] Gay, David, Levis, Philip, Culler, David et al. *NesC 1.1 Language Reference Manual* [online]. 2003 [cit. 2011-11-14]. Dostupné z: http://nescc.sourceforge.net/papers/nesc-ref.pdf.

[10] Greenstein, Ben a Levis, Philip. *Serial Communication* [online]. 2005-07-11 [cit. 2012-05-12]. Dostupné z: http://www.tinyos.net/teps/doc/html/tep113.html.

[11] Horáček, Jan. *Platforma pro mobilní agenty v bezdrátových senzorových sítích* [online]. Brno: FIT VUT v Brně, 2009. Diplomová práce. Dostupné z: http://www.fit.vutbr.cz/study/DP/rpfile.php?id=8922.

[12] Horáček, Jan, Zbořil, František a Gábor, Martin. *WSageNt: Multiagent Platform for Wireless Sensor Networks* [online]. 2010 [cit. 2011-12-30]. Dostupné z: http://www.fit.vutbr.cz/~ihoracek/WSageNt/.

[13] Horkel, Milan. *Programování procesorů ATMEL AVR* [online]. 2005-12-28 [cit. 2011-10-18]. Dostupné z: http://www.mlab.cz/Modules/AVR/Text_Prog/DOC/Programovani%20AVR.cs.pdf.

[14] HQ JSC Teltonika. *AT Commands Manual* [online]. 2007-04-27 [cit. 2012-01-04]. Dostupné z: http://www.teltonika.lt/uploads/docs/TeltonikaATcommandsEN20070427.pdf.

[15] Kubička, Matěj a Šimána, František. *RoboCraner2* [online]. 2007 [cit. 2012-01-02]. Dostupné z: http://matejk.cz/zdroje/robocraner2-dokumentace.pdf.

[16] Levis, Philip a Gay, David. *TinyOS Programming*. Cambridge: Cambridge University Press, 2009. ISBN 978-0-521-89606-1.

[17] Maxim Integrated Products. *Datasheed MAX3222-MAX3241* [online]. 1999 [cit. 2012-01-01]. Dostupné z: http://datasheets.maxim-ic.com/en/ds/MAX3222-MAX3241.pdf.

[18] Maxim Integrated Products. *Datasheed MAX220-MAX249* [online]. 2010 [cit. 2011-12-30]. Dostupné z: http://datasheets.maxim-ic.com/en/ds/MAX220-MAX249.pdf.

[19] Olmr, Vít. *HW server představuje - Sériová linka RS-232* [online]. 2005-12-12 [cit. 2011-12-28]. Dostupné z: http://www.hw.cz/rs-232#rs232_ttl.

[20] Spáčil, Pavel. *Mobilní agenti v bezdrátových senzorových sítích* [online]. Brno: FIT VUT v Brně, 2009. Bakalářská práce. Dostupné z: http://www.fit.vutbr.cz/study/DP/rpfile.php?id=7959.

[21] TinyOS Documentation Wiki. *Installing TinyOS 2.1.1* [online]. 2007-07-20 [cit. 2011-10-19]. Dostupné z: http://docs.tinyos.net/tinywiki/index.php/Installing_TinyOS_2.1.1#Two-step_install_on_your_host_OS_with_Debian_packages.

[22] TinyOS Working Groups. *Serial-line communication in TinyOS-1.1* [online]. 2003-09-03 [cit. 2012-01-03]. Dostupné z: http://www.tinyos.net/tinyos-1.x/doc/serialcomm/index.html.

[23] TinyOS Working Groups. *Class MoteIF* [online]. [cit. 2012-01-03]. Dostupné z: http://www.tinyos.net/dist-2.0.0/tinyos-2.0.0/doc/javadoc/net/tinyos/message/MoteIF.html#MoteIF().

[24] UAB Teltonika. *Teltonika ModemCom/G10* [online]. 2008-10-23 [cit. 2012-04-30]. Dostupné z: http://avl1.teltonika.lt/Downloads/Flyers/Flyer_ModemCOM_G10_EN.pdf.

[25] Ščuglík, František. *GSM* [online]. 2007-11-27 [cit. 2012-01-03]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/PKS-IT/lectures/07_GSM.ppt?cid=7552.

[26] Řeháк, Jan a Knotek, Kamil. *GPRS z hlediska aplikací* [online]. 2003-07-01 [cit. 2012-05-10]. Dostupné z: http://www.hw.cz/teorie-a-praxe/dokumentace/gprs-z-hlediska-aplikaci.html.

Dodatek A

Obsah CD

- 1. Text této diplomové práce v elektronické podobě
- 2. Zdrojové soubory textové části této práce
- 3. Zdrojové soubory aplikace IPBSComm
- 4. Spustitelný soubor aplikace IPBSComm
- 5. Zdrojové soubory aplikace BaseStation
- 6. Poster reprezentující tuto práci