VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

POUŽITÍ INTELIGENTNÍCH AGENTŮ V BEZDRÁTOVÝCH SENZOROVÝCH SÍTÍCH

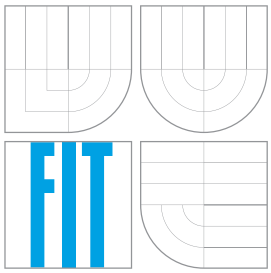
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR ŽÍDEK

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

POUŽITÍ INTELIGENTNÍCH AGENTŮ V BEZDRÁTOVÝCH SENZOROVÝCH SÍTÍCH

USAGE OF INTELLIGENT AGENTS IN WIRELESS SENSOR NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. PETR ŽÍDEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. FRANTIŠEK ZBOŘIL, Ph.D.
SUPERVISOR

BRNO 2011

Abstrakt

Tato práce si klade za cíl seznámit čtenáře s technologií bezdrátových senzorových sítí spolu s nástroji pro vývoj a simulaci aplikací pro ně určených. Dále má čtenáře seznámit s projektem WSnageNt, který je určen pro podporu agentů v bezdrátové senzorové síti. Hlavním cílem této práce je ovšem rozšíření projektu WSageNt o nové prvky, které umožní vytvořit agenty s většími možnostmi a schopnostmi. Těchto prvků bude poté využito pro rozšíření projektu WSageNt o sledování pohybu uzlů v síti.

Abstract

This thesis is focused on wireless sensor networks and implementation tools for creation and simulation of applications. It describes WSnageNt project, which is intended to support agents in wireless sensor networks. The main objective of this thesis is to extend the WSageNt project with new features that will allow to create agents with more capabilities. These features will be then used to extend the WSageNt project with network node movement tracking capability.

Klíčová slova

Bezdrátové senzorové sítě, agent, WSnageNt, Iris, TinyOS, NesC, TOSSIM, ALLL, pachové stopy.

Keywords

Wireless sensor networks, agent, WSnageNt, Iris, TinyOS, NesC, TOSSIM, ALLL, footprint.

Citace

Petr Žídek: Použití inteligentních agentů v bezdrátových senzorových sítích, diplomová práce, Brno, FIT VUT v Brně, 2011

Použití inteligentních agentů v bezdrátových senzorových sítích

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana Ing. Františka Zbořila, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Žídek
24. května 2011

Poděkování

Tímto bych chtěl poděkovat panu Františku Zbořilovi, který byl vedoucím práce a Janu Horáčkovi za jeho přínosné rady. Dále bych chtěl poděkoval Marku Housťovi za odstranění některých chyb v původní verzi platformy. V neposlední řadě bych rád poděkoval mé přítelkyni a rodičům za trpělivost a ochotu pomoci.

Obsah

Kapitola 1

Úvod

Počátky technologie bezdrátových senzorových sítí sahají do roku 1949 a jsou spjaty s vývojem protiponorkových zbraní americkou armádou. Typ těchto projektů již v té době definoval základní koncepty a myšlenky, které se dnes u této technologie uplatňují.

Pokrok v síťových technologiích a technologiích vestavěných zařízení umožnil vytvoření malých, flexibilních a levných zařízení, která mohou mezi sebou a jejich okolním prostředím interagovat. Tato zařízení mohou obsahovat senzory různých druhů a mohou tak snímat různorodé veličiny, jako např. teplotu, tlak nebo intenzitu slunečního záření.

Zařízení také mohou mezi sebou komunikovat a vyměňovat si tak například získaná data nebo data posílat do centrálního bodu. V tomto místě mohou být data uložena pro pozdější použití nebo na nich může být provedena analýza.

Na rozdíl od tradičních desktopových počítačů jsou tato zařízení používána spíše pro sběr dat a řízení než obecně pro výpočty. Výpočetní výkon je u nich relativně malý, nicméně pro jejich potřeby plně dostačující.

Zařízení mohou být mobilní a z toho plyne nutnost jim zajistit nějaký zdroj elektrické energie. Pro svůj provoz mohou dokonce využívat obnovitelných zdrojů, jako např. slunečních paprsků.

Využití technologie bezdrátových senzorových sítí je především v monitorování okolního prostředí, domácí automatizaci nebo monitorování lidského zdraví. Vzhledem k tomu, že tato technologie je relativně mladá, hledají se stále nová pole působnosti pro její uplatnění.

Již v současné době je možné pozorovat stále širší uplatnění těchto technologií. Byly vyvinuty a vyrobeny například miniaturní bezdrátové implantáty[3] měřící tlak a teplotu v oku, pasivní senzory pro měření vakua v moderních tepelně izolujících oknech a hybridní mikrosystémy spojující na jednom čipu chemické a biologické senzory s vyhodnocovacími mikroelektronickými obvody.

Tato práce se věnuje popisu technologie bezdrátových senzorových sítí. Čtenáři budou nastíněny základní principy a koncepty pro vývoj a simulaci aplikací určených pro tyto sítě. Práce se bude také věnovat popisu skutečných zařízení, z kterých může být bezdrátová senzorová síť složena.

Práce má dále čtenáři poskytnout informace o projektu WSageNt, který je určen pro podporu agentů v bezdrátové senzorové síti. WSageNt je výstupem dvou diplomových a jedné bakalářské práce. Projekt je možné rozdělit na část interpretu, platformy a webového rozhraní. Úkolem interpretu je interpretovat kód agenta popsaného jazykem ALLL. Platforma interpret řídí a nabízí mu své služby, jako např. práce se seznamy, radio komunikace, měření hodnot pomocí senzorů, atd. Webové rozhraní pak slouží pro vytvoření uživatelského prostředí, ve kterém je možné jednoduše využívat služeb platformy.

Hlavním cílem této práce je ovšem rozšíření projektu WSageNt. Budeme se věnovat použití inteligentních agentů pro sledování pohybu uzlů v bezdrátových senzorových sítích. Pro tento úkol bude nutné rozšířit interpret a platformu o nové služby. Tyto služby budou pracovat s informacemi (pachovými stopami), které agent po přesunu z jednoho uzlu na druhý uzel zanechá. Nové služby budou s těmito informacemi pracovat a budou nabízet např. informace odkud přišel agent poprvé nebo zda na daném uzlu již byl. To umožní vytvářet agenty s většími schopnostmi. Bude možné např. zjišťovat trasu agenta, jak procházel sítí nebo bude možné vytvořit agenta, který projde celou síť a vrátí se zpět na uzel, z kterého byl odeslán. Agenta, který projde celou síť, využijeme pro získání hodnot síly signálu mezi jednotlivými uzly v síti. Na základě těchto informací bude možno provést vizualizaci rozmístění uzlů ve webovém rozhraní. Periodickým vysíláním tohoto agenta pak docílíme vizualizace pohybu uzlů.

Práce je členěna do kapitol, jejichž stručný popis je zde uveden. V kapitole 2 je popsána historie vzniku technologie bezdrátových senzorových sítí, včetně toho, které prvky síť tvoří. Kapitola 3 se zaměřuje na operační systém TinyOS a ukazuje práci se simulační knihovnou TOSSIM. Následující čtvrtá kapitola se zaměřuje na popis základních konceptů programovacího jazyka NesC. Kapitola 5 obsahuje stručný popis inteligentních agentů a jejich vlastností. Kapitola 6 se zabývá projektem WSageNt, který se zaměřuje na použití agentů v prostředí bezdrátových senzorových sítí. Text kapitoly 7 se věnuje návrhu rozšíření projektu WSageNt. Obsahem osmé a deváté kapitoly je popis vlastní implementace provedených rozšíření. Kapitola 10 se zabývá testováním provedeného rozšíření. V poslední kapitole se nachází závěr, který hodnotí dosažené výsledky. Zároveň je zde naznačen směr pokračování další práce.

Kapitola 2

Bezdrátové senzorové sítě

2.1 Stručný popis

Na počátku vzniku této technologie byl projekt americké armády SOSUS (*Sound Surveillance System*)[19], který se zabýval vývojem systému pro protiponorkové zbraně. Vznik tohoto projektu se datuje k roku 1949. Poté následoval další navazující projekt nazvaný IUSS (*Integrated Undersea Surveillance System*).

Typ těchto projektů definoval základní koncepty technologie, tedy existenci autonomních zařízení, majících schopnost mezi sebou komunikovat a sbírat data ze svého prostředí.

V roce 2001 pak následovaly projekty Smart Dust a NEST, které tyto základní vize dále více obohatily. Jako řada technologií pocházejících z armádního výzkumu si i tato našla cestu k širší veřejnosti.[1] Těchto myšlenek se poté chopily některé univerzity a soukromé společnosti a začaly je dále rozvíjet.

Bezdrátové senzorové sítě, neboli *wireless sensor networks* (WSN), jsou tedy složeny z autonomních zařízení, která mohou mezi sebou komunikovat pomocí bezdrátové technologie. Komunikace může být realizována např. využitím standardu IEEE 802.15.4-2003. Tato zařízení také obsahují senzory, které umožňují snímání různorodých fyzikálních veličin, jako např. tlak, teplota atd.

Bezdrátová senzorová síť je tvořena dvěma typy zařízení. První z nich je nazýváno *základní stanice* (dále jen basestation). Ta je připojena pomocí některého standardního rozhraní (USB, RS232) k PC. Na počítači může být spuštěn např. program pro analýzu nasbíraných dat ze sítě.

Druhým typem je *senzorový uzel* (node[2]), který obsahuje senzory pro měření fyzikálních veličin a výsledky může posílat ostatním uzlům nebo basestation.

V současné době je možné vidět aplikaci těchto technologií v oblastech monitorování lidského zdraví, životního prostředí nebo automatizace v domácnostech.

Zařízení mají určitá omezení, která do jisté míry určují jejich použití. Napájení je realizováno ve formě baterií, solárních článků nebo obojího, což má sice za následek omezenou dobu provozu, ale naopak to umožňuje větší mobilitu. Zařízení se proto snaží pracovat efektivně, aby došlo k prodloužení doby jeho činnosti. Jako další limity lze uvést omezenou šířku pásma a malou velikost paměti RAM (4-8 KB).

Jak již bylo zmíněno, současným problémem je zajištění napájení uzlů po dlouhou dobu jejich činnosti. Vizí bezdrátových senzorových sítí budoucnosti[8] je vytvoření permanentně

[1] Jako příklad, dnes již běžně používaných technologií pocházejících z vojenského výzkumu bychom mohli uvést např. GPS, počítače nebo Internet.

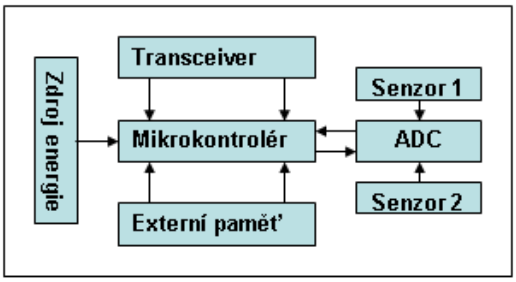
[2] V Severní Americe se pro senzorový uzel používá výraz mote.

fungujících uzlů, které by měly dokázat využít okolní energie pro své průběžné dobíjení.

2.2 Senzorový uzel

Při vývoji tohoto zařízení byl kladen velký důraz zejména na malé rozměry uzlů, malou spotřebu energie a dobré přizpůsobení okolním podmínkám. Senzorový uzel je mikroelektronické zařízení, které je schopno vykonávat běh programu. Mimoto pak také může sbírat informace o okolních podmínkách a komunikovat s okolními uzly. Uzel je při své činnosti ovlivňován událostmi z okolního prostředí (např. změna teploty). Obrázek 2.1 znázorňuje blokovou strukturu základních komponent.

Mezi základní komponenty se řadí[21]: mikrokontrolér, vysílač/příjmač, externí paměť, zdroj energie a v neposlední řadě také senzory.



Obrázek 2.1: Struktura senzoru.

Důležitou součástí senzorového uzlu jsou **senzory**. Ty určují typ snímaných dat z okolního prostředí. Data jsou získána nejprve analogově a poté jsou převedena analogově digitálním převodníkem (*ADC*) do digitální podoby. Mohou tak být následně uložena pro pozdější analýzu. Typy senzorů mohou být např. mechanické, tepelné, chemické, elektromagnetické či světelné.

Cílem **mikrokontroléru** je vykonávat procesy a řídit činnost ostatních komponent. Velkou výhodou mikrokontroléru je malá spotřeba v řádech mW. Dále je také možné část mikrokontroléru vypnout a tak spotřebu ještě snížit až řádově na nW, což je výhodné především pro prodloužení doby provozu uzlu. Mezi další dobré vlastnosti se řadí také dobrá programovatelnost, flexibilita, malé rozměry a také nízká cena. Nevýhodou je oproti tomu však malá rychlost v řádu jednotek MHz, která ovšem pro tyto účely plně dostačuje.

Transceiver je zařízení, které obsahuje vysílač a přijímač radiových vln. Jedním z používaných frekvenčních pásem pro bezdrátové senzorové sítě je pásmo 2.4-2.4835 GHz, protože je vhodné pro všesměrové vysílání.

Nejčastěji používanou **externí pamětí** je paměť typu flash. Je to z důvodu malé energetické náročnosti, velké kapacity a také dnes již nízké ceny.

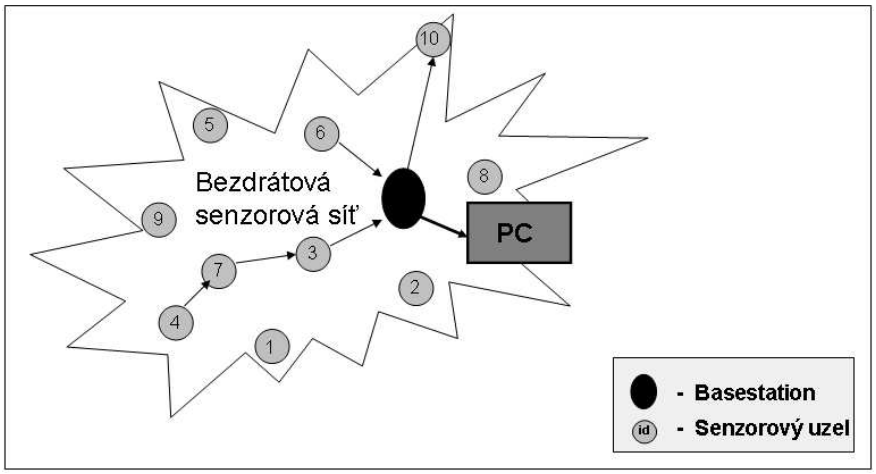
Další důležitou součástí každého senzorového uzlu je **zdroj energie**. Ten je využíván všemi komponentami uzlu. Největší nároky na spotřebu energie jsou vynaloženy při komunikaci zařízení, a proto je žádoucí ji omezit na minimum.

Obrázek 2.2: Senzorový uzel (MicaZ).

2.3 Senzorová síť

Typická senzorová síť tvoří bezdrátovou *ad-hoc*[3] síť. V ad-hoc síti jsou si všechna zařízení rovna (*peer-to-peer*). Síť může být složena i z několika basestation a velkého množství uzlů, které tak mohou pokrýt rozsáhlou oblast.

Síť by měla umožňovat multihop komunikaci. Ta je ilustrována na obr. 2.3, který znázorňuje bezdrátovou senzorovou síť, která obsahuje několik uzlů a jednu *basestation* připojenou k počítači. Pokud uzel číslo 4 chce komunikovat s basestation, probíhá komunikace přes uzly číslo 7 a 3. Výhoda multihopu je tedy v možnosti nepřímé komunikace prvků v síti za pomoci mezilehlých prvků.



Obrázek 2.3: Bezdrátová senzorová síť.

[3]Příkladem tohoto typu sítě může být například propojení několika notebooků přes bezdrátovou síťovou kartu (*wifi*)

2.4 Hardware pro bezdrátové senzorové sítě

V této podkapitole se budeme zabývat popisem konkrétního hardware pro bezdrátové senzorové sítě od firmy Crossbow. Tato firma vznikla v roce 1995 a je jedním z předních dodavatelů řešení založených na bezdrátových senzorových sítích a senzorových systémech.

2.4.1 Iris platforma

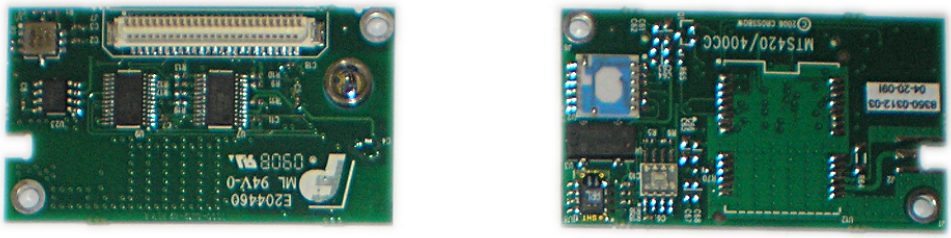
Iris[6] je poslední generací uzlů od firmy Crossbow. Platforma XM2110 (IRIS) obsahuje transceiver Atmel RF230, který slouží pro příchozí a odchozí radio-frekvenční komunikaci. Použitým mikrokontrolérem je Atmega1281, který je navržen tak, aby jeho spotřeba byla velmi nízká. Iris má 8KB velkou paměť RAM, což je dvakrát více než u předchozí generace uzlů MicaZ. Velikost flash paměti pro program je 128KB a velikost externí paměti pro data je 512KB.

Deska dále obsahuje 51-pinové vstupně/výstupní rozhraní, přes které je možné připojit různé senzorové desky obsahující senzory pro měření fyzikálních veličin. Přes toto rozhraní se také připojují uzly k basestation pro programování. Iris, stejně jako MicaZ platforma, obsahuje také tři barevné LED diody. Zajištění napájení uzlů je realizováno dvěma tužkovými bateriemi.

2.4.2 Senzorová deska

Platforma Iris dovoluje přes 51-pinový konektor připojit různé desky pro měření fyzikálních veličin. Senzorová deska **MTS420/400CC**[7] (obr. 2.4) obsahuje:

- senzor pro měření teploty a vlhkosti
- senzor pro měření barometrického tlaku
- senzor pro měření intenzity okolního osvětlení
- dvouosý akcelometr

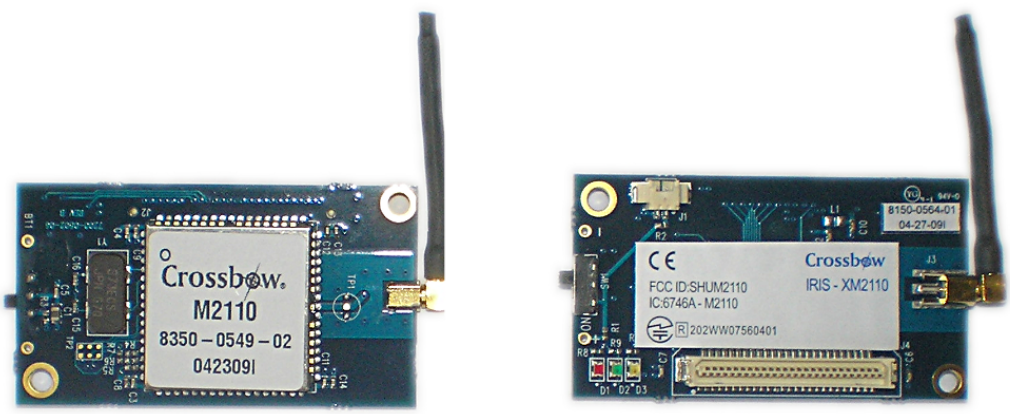


Obrázek 2.4: MTS420/400CC.

Teplotní senzor dokáže měřit teplotu v rozsahu -10 ° až +60 ° a vlhkost v rozsahu 0 až 100% RH nekondenzující. Senzor pro měření barometrického tlaku pracuje v rozmezí 300 až 1100 mbar. Senzor měřící intenzitu okolního osvětlení dokáže snímat hodnoty v rozsahu 400-1000 nm.

2.4.3 Radio transceiver

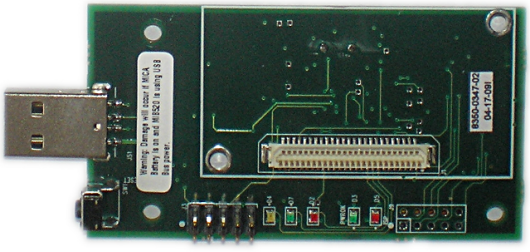
Použitý transceiver **XM2110** (obr. 2.5) vyhovuje standardu IEEE 802.15.4, který je určen pro nízko-energetická bezdrátová zařízení. Využívá rádio čip Atmel AT86RF230, který používá O-QPSK *(offset quadrature phase-shift keying)* modulaci a DSSS *(digital direct sequence spred spectrum)*. Transceiver pracuje ve frekvenčním rozsahu od 2405 MHz do 2480 MHz. Radiový rozsah je uváděn do 300 m pro venkovní prostředí a do 50 m pro vnitřní prostředí, což je třikrát více než u generace uzlů MicaZ. Maximální přenosová rychlost komunikace je až 250Kbps.



Obrázek 2.5: XM2110.

2.4.4 Basestation

Basestation **MIB520CB** (obr. 2.6) se připojuje k PC prostřednictvím standardního USB rozhraní. Tímto rozhraním je zajištěna komunikace mezi PC a basestation a napájení. Basestation obsahuje 51-pinový konektor, přes který je možno připojit uzly. Podporovanými typu uzlů jsou Mica2, MicaZ a Iris. Po připojení uzlu k basestation je možné z PC do uzlu nahrát program. Basestation také po připojení uzlu umožňuje komunikaci s bezdrátovou senzorovou sítí.



Obrázek 2.6: MIB520CB.

Kapitola 3

Operační systém TinyOs

Na počátku vzniku tohoto systému stála spolupráce mezi University of California, Berkeley a firmou Intel. TinyOs[16] je otevřený, na zdroje nenáročný operační sytém, který je určený pro bezdrátové senzorové sítě. Byl napsán v programovacím jazyku NesC.

TinyOs poskytuje hardwarovou abstrakci pro vývoj aplikací, které mohou být pomocí TinyOs knihoven přeloženy pro konkrétní hardwarovou *platformu*. Tento přístup poskytuje výhodu, která spočívá v tom, že je zdrojový kód aplikace stejný i pro odlišný hardware.

V současné době je aktuální verzí verze 2.x., jejíž vznik byl motivován nedostatky vyplývající z verze starší. V té bylo obtížné propojování komponent a hledaní interakcí mezi nimi. Ve verzi 2.x. je již většina základních služeb virtualizována, což přispělo k snadnějšímu programování aplikací pro bezdrátové senzorové sítě. Je zde ovšem důležité upozornit na nekompatibilitu[1] mezi verzemi 1.x. a 2.x..

3.1 Překlad

Překlad[2] TinyOs aplikací je možný pomocí utility *make*, která je vhodná pro automatizaci překladu. Make poskytuje poměrně silný a rozšiřitelný prostředek, který umožňuje přidání nových platforem a různých voleb pro překlad. Program make nejprve hledá v lokálním adresáři soubor `Makefile`. Pokud takový soubor neexistuje, make aplikuje globální nastavení pravidel[2] pro překlad, která jsou standardně umístěna v souboru /opt/tinyos-2.x/support/make/Makerules. Překlad pro platformu Iris se provede příkazem:

```
make iris
```

3.2 Nahrání programu do zařízení

V této části se budeme zabývat nahráním aplikace do zařízení v prostření TinyOs. V tomto případě ilustrujeme postup instalace pro uzly platformy Iris, nicméně pro jiné platformy je postup obdobný.

Basestation obecně obsahuje některé standardizované rozhraní (USB, RS232, atd), kterým je možné ji připojit k PC. Basestation platformy Iris může obsahovat USB port. Ten

[1] Existuje možnost upgrade zdrojového kódu aplikace napsané pro verzi 1.x do verze 2.x. Více informací lze nalézt na http://www.tinyos.net/tinyos-2.x/doc/html/porting.html

[2] Pravidla, která se uplatňují při překladu je možné vypsat zadáním příkazu `printenv MAKERULES` do konzole.

je použit pro nahrávání programu do zařízení i pro komunikaci s počítačem. Po připojení basestation k PC by měla být systémem rozpoznána[3] a měly by být vytvořeny dva virtuální porty. Například odkaz na zařízení dev/ttyUSB0 je určen k programování uzlů přes basestation a odkaz /dev/ttyUSB1 pro komunikaci s PC. Nahrání programu se provede takto:

```
make iris install.id mib510,/dev/ttyUSB0
```

`Id` je nezáporné celé číslo, které specifikuje zvolené id uzlu. Za ním následuje typ basestation, přes kterou probíhá programování a port, na kterém je připojena.

3.3 TOSSIM

TOSSIM[18] je simulátor, který je určen pro simulaci TinyOs aplikací. Umožňuje simulovat sítě, které mohou obsahovat i tisíce uzlů, tvořících bezdrátovou senzorovou síť. V té je v každém uzlu spuštěna vlastní instance aplikace. Zdrojové kódy je možné nalézt v adresáři */tos/lib/tossim*. Ten dále obsahuje užitečná data pro vytvoření modelu.

TOSSIM je tedy diskrétně událostní simulátor, který nahrazuje některé nízko-úrovňové komponenty TinyOs, využité v aplikaci vlastní implementací pro simulaci. Když je spuštěn, tak ukládá události do událostní fronty podle času a pak je následně vykonává. Simulační události mohou reprezentovat hardwarová přerušení nebo nějaké vysoko-úrovňové události, jako např. příjem paketů.

3.3.1 Překlad aplikace pro TOSSIM

Pro vytvoření simulačního modelu je nejprve nutné přeložit aplikaci pro TOSSIM. Překlad NesC aplikace pro simulaci[4] se provede příkazem:

```
make micaz sim
```

TOSSIM nejprve použitím nástroje nesc-dump vytvoří XML dokument, který obsahuje jména a typy všech proměnných v aplikaci. Poté jsou změněny cesty include souborů pro simulaci a kompilaci aplikace. Dále se zkompiluje podpora pro C++ a Python programové rozhraní, přes které je možné vytvořit simulační model. V návaznosti na to je pak vytvořen sdílený objekt obsahující TOSSIM kód a podporu C++, Pythonu.

TOSSIM podporuje Python a C++ programové rozhraní pro vytvoření simulačního modelu. Python navíc umožňuje interakci s modelem v reálném čase.

V následujícím textu se budeme věnovat rozhraní Pythonu, nicméně pro rozhraní C++ je postup velice podobný.

S modelem je možné pracovat dvěma způsoby. Jednak je možné vytvořit soubor se simulací, který Pythonem spustíme a následně se provede dávkově simulace. Mimoto je možné spuštění Pythonu a interaktivní práce s modelem, kde zadáváme příkazy Pythonu.

[3] Pokud TinyOs nenalezne použitý hardware, je třeba nainstalovat ovladače dostupné na http://www.ftdichip.com/Drivers/VCP.htm.

[4] V současné době je jedinou podporovanou platformou TOSSIMu platforma MicaZ. Ta se ale např. od platformy Iris liší jen typem transceiveru a velikostí paměti RAM a proto je možné provést simulaci platformy Iris.

3.3.2 Vytvoření simulačního modelu

V každé simulaci je nejprve nutné importovat TOSSIM a vytvořit TOSSIM objekt. To se provede následovně:

```
from TOSSIM import *
t = Tossim([])
```

Hranaté závorky mohou obsahovat proměnné, ke kterým chceme přistupovat během simulace. V tomto případě nejsou specifikovány žádné. TOSSIM objekt obsahuje několik užitečných funkcí, které je možné zjistit funkcí `dir()`, jejímž parametrem je sám objekt.

Na začátku simulace nejsou vytvořeny žádné uzly. Vytvoření uzlu m se provede následovně:

```
m = t.getNode(2)
```

V kulatých závorkách je uvedeno číslo uzlu, které si můžeme zvolit. Existuje zde nepsaná konvence, která říká, že hodnota id=1 je vyhrazena pro basestation.

Výše uvedeným přiřazením se vytvoří uzel, který není ovšem aktivní. Zapnutí uzlu se může provést následovně:

```
m.bootAtTime(4 * t.ticksPerSecond() + 242119)
```

Vykonání jednoho kroku simulace se provede následovně:

```
t.runNextEvent()
```

Pro vykonání více kroků simulace je vhodné použít cyklus:

```
while t.runNextEvent():
  pass
```

TOSSIM umožňuje zobrazování výstupů z aplikace skrz kanály *gdb*. Gdb vytvoří kanál, přes který aplikace posílá zprávy simulátoru. Na straně aplikace musíme tedy vytvořit pojmenovaný kanál a ten pak importovat v simulaci. V následujícím nesC kódu je demonstrováno použití gdb:

```
event void Boot.booted() {
  call Leds.led0On();
  dbg("Boot","Node booted at %s \n", sim_time_string());
  call AMControl.start();
}
```

Na straně modelu musíme nejprve importovat knihovnu *sys* (from sys import *) a poté přidat požadované kanály. V našem případě kanál Boot, tedy:

```
t.addChannel("Boot", sys.stdout)
```

TOSSIM se pokouší poskytnout několik nízko-úrovňových primitiv pro simulaci širokého rozsahu transceiverů a jejich chování. Implicitní hodnoty radio modelu jsou založeny na transceiveru CC2420, který používají např. platformy rodiny MicaZ.

Simulace transceiveru se provádí přes radio objekt, který obsahuje funkce pro radiokomunikaci:

```
r = t.radio()
```

Specifikace rozmístění uzlů se provede přidáním záznamu do radio objektu, kde prvním argumentem je zdrojový uzel, druhým je cílový uzel a posledním je zisk v dBm. V případě rozsáhlé sítě je výhodné načíst data do simulace z externího souboru:

```
r.add(1, 2, -54.0)
r.add(2, 1, -50.0)
```

TOSSIM také simuluje radio-frekvenční rušení[14] a přeslechy mezi uzly využitím CPM *(Closest Pattern Matching)* algoritmu. Ten na základě stopy šumu vygeneruje statistický model. Adresář /tos/lib/tossim/noise pak obsahuje soubory s příklady šumu, které je možné požít jako vstup pro CPM:

```
noise = open("meyer-heavy.txt", "r")
lines = noise.readlines()
for line in lines:
  str = line.strip()
  if (str != ""):
    val = int(str)
    for i in range(1, 3):
      t.getNode(i).addNoiseTraceReading(val)

for i in range(1, 3):
  t.getNode(i).createNoiseModel()
```

3.4 Komponenty TinyOS

V následujících podkapitolách budou uvedeny některé základní systémové komponenty TinyOs. Tyto komponenty je možné použít jako základní stavební bloky v nových aplikacích.

3.4.1 Inicializace

TinyOs obsahuje modul MainC, který nabízí jediné rozhraní `Boot`[1]. To je určeno pro počáteční inicializaci uzlu. Při spuštění uzlu proběhne bootovací sekvence. Ta obsahuje několik kroků, jimiž jsou inicializace plánovače, inicializace komponent, signalizace informace o dokončení bootovacího procesu a také spuštění plánovače. Rozhraní obsahuje událost *booted*. Tomuto rozhraní je v kroku signalizace informace o dokončení zaslán signál. Jelikož se jedná o událost, je nezbytné, aby byla v případě použití tohoto rozhraní implementována.

3.4.2 Přístup k flash paměti

TinyOs poskytuje tři typy přístupů k flash paměti, jimiž jsou *konfigurace*, *logování* a *blok* [4].

Konfigurace je určena pro záznam a čtení konfiguračních nastavení pro aplikaci. Některá data nemusejí být v průběhu překladu známa, nebo se může jednat o různé hardwarově specifické hodnoty, které mohou být neuniformní napříč uzly. Konfiguračními daty mohou tak být např. různé kalibrační nastavení pro senzory, identifikace uzlů, informace o umístění nebo informace o způsobu získávání hodnot ze senzorů. Pro tento typ přístupu k paměti obsahuje TinyOs rozhraní `ConfigStorageC`.

Logování je určeno pro sekvenční záznam a čtení malých objemů dat. Nová data jsou v případě zápisu vždy uložena na konec stávajících. V případě čtení jsou záznamy opět čteny sekvenčně. V případě, že již došlo k zaplnění přiděleného bloku dat, může být zápis nových dat zastaven, nebo data mohou postupně přepisovat blok znovu od začátku. Nevýhodou tohoto způsobu práce s pamětí je velmi neefektivní přístup k datům v náhodném pořadí. Naproti tomu by mohlo být výhodou, že manipulace s daty se děje atomicky a máme tak jistotu, že nedojde k přerušení operace. Pro tento typ přístupu TinyOs poskytuje rozhraní `logRead` a `LogWrite`.

Blokový režim se zpravidla používá pro uložení velkých dat, které by nebylo možné uložit do RAM. Velkou výhodou tohoto přístupu je možnost efektivně přistupovat k datům v náhodném pořadí jak pro čtení, tak i pro zápis. Rozhraní pro čtení z paměti je v TinyOs pojmenováno `BlockRead` a pro zápis `BlockWrite`.

TinyOs umožňuje vývojáři rozdělit flash paměť do několika samostatných bloků s pevně přidělenou velikostí. Tato konfigurace se použije v době překladu pro rozdělení paměti. Tyto informace jsou uloženy v XML konfiguračním souboru, jehož název musí být ve tvaru volumes-CHIPNAME.xml, kde CHIPNAME je název flash čipu[5]. Konfigurační záznamy jsou uloženy v tabulce, která obsahuje informaci o názvu bloku a jeho velikosti v bytech. S jednotlivými bloky je možno pracovat pomocí výše uvedených rozhraní.

Příklad konfigurace může vypadat následovně:

```
<volume_table>
  <volume size="16384" name="DATA" />
</volume_table>
```

3.4.3 Komunikace

Pro možnost komunikace obsahuje TinyOS strukturu nazvanou *message_t*[15] a čtyři základní komponenty, kterými jsou AMSenderC, AMReceiverC, AMSnooperC a AMSnoopingReceiverC.

Před prací s tranceiverem je nutné provést inicializaci systémovou funkcí `start` modulu AMControl. Po nějakém čase je modulu, který funkci volal, zaslán signál *startDone*. Ten informuje modul, zda se podařilo transceiver zapnout.

Rozhraní `Recieve` je určeno pro příjem paketů, které poskytuje modul AMReceiverC. Pro odeslání paketů je možné použít rozhraní `AMSend` nebo Send. Rozdíl mezi nimi je ten, že AMSend umožňuje specifikovat cílovou adresu.

Komunikace mezi uzly je realizována prostřednictvím posílání paketů. Každý paket obsahuje hlavičku, data, metadata a patičku (viz obr. 3.1). Každá hardwarová platforma definuje vlastní specifikaci těchto polí v struktuře message_t.

Obrázek 3.1: Struktura message_t.

Hlavička je tvořena polem bytů, které jsou především určeny pro směrování paketu. Příkladem hodnot tohoto pole je zdrojová nebo cílová adresa. Platforma Iris (CC2420) obsahuje 11-bytovou hlavičku.

[5]Platformy MicaZ a Iris používají čip od firmy Atmel at45db.

Datová část je určená pro přenášení vlastních dat a dokáže pojmout data o velikosti dané konstantou TOSH_DATA_LENGTH. Její implicitní hodnota je 28 bytů.

Patička může sloužit pro ukončení dat, u některých platforem nemusí být využita.

Metadata uchovávají informace, které nejsou posílány v paketu. Tento mechanismus umožňuje datové vrstvě uložit např. informace o RSSI.

3.4.4 Časovač

TinyOs nabízí rozhraní `Timer`, které poskytuje rozsáhlý systém časovačů. Ten zahrnuje časovače s různorodým rozsahem časování (8, 16, 32 nebo 64-bitové), přesností (milisekundové, mikrosekundové) a dalších možností. Pro všechny časovače platí dva společné rysy, jimiž jsou měření času a periodické generování událostí v čase. Časovač tedy představuje komponentu, která měří čas a v určitém čase vygeneruje událost, nebo periodicky generuje události se zvolenou periodou. Dále systém poskytuje velice přesné asynchronní časovače (jsou označeny slovem async).

Kapitola 4

NesC

NesC (*network embedded systems C*) je programovací jazyk určený pro vývoj aplikací pro vestavěné systémy, které jsou mezi sebou propojeny a vytváří tak síť. Příkladem takových zařízení mohou být bezdrátové senzorové sítě. Tato kapitola čerpá informace ze zdrojů [17] a [9].

Jazyk NesC je velice podobný programovacímu jazyku C, ale obsahuje navíc některá rozšíření (např. komponenty a jejich propojení) speciálně navržená pro potřebu bezdrátových senzorových sítí. Je důležité si uvědomit, že uzly v síti jsou na rozdíl od tradičních počítačů používány spíše pro sběr dat a řízení okolního prostředí, než pro výpočty. To má za následek, že jsou uzly řízeny spíše událostmi (reakce na změny v okolních podmínkách), než interaktivně nebo dávkovým zpracováním.

Aplikace je tvořena několika vzájemně propojenými *komponentami*. Ty mohou skrz svá *rozhraní* [1] poskytovat funkce ostatním komponentám nebo naopak mohou využívat funkce jiných komponent. Komponenty jsou podobné objektům z programovacího jazyka C++. Jedním z rozdílů je, že komponenty používají lokální jmenné prostory.

Aplikace jsou tvořeny znovupoužitelnými systémovými komponentami, které jsou spojeny s aplikačně specifickým kódem. Tabulka 4.1 obsahuje konvenci názvů souborů v TinyOs.

Jméno souboru	**Typ souboru**
Foo.nc	Rozhraní
Foo.h	Hlavičkový soubor
FooC.nc	Modul
FooP.nc	Skrytý modul

Tabulka 4.1: Konvence názvů souborů v TinyOs.

4.1 Komponenty

V NesC existují dva typy komponent, jimiž jsou *moduly* a *konfigurace*. Moduly obsahují vlastní implementaci funkcí a událostí. Oproti tomu konfigurace jsou používány k propojení ostatních komponent navzájem.

[1] Rozhraní v tomto smyslu má stejný význam jako u jiných programovací jazyků, jako například C++.

Každý modul obsahuje blok kódu, specifikující které funkce nebo rozhraní ostatních komponent využívá a které naopak poskytuje ostatním. Vedle toho musí modul obsahovat také blok kódu s vlastní implementací. Komponenta, která modul využívá, obsahuje klíčové slovo *uses*. Rozhraní, které modul nabízí ostatním komponentám, předchází klíčové slovo *provide* a musí být dále v modulu implementována.

Příkladem jednoduchého modulu je SmoothingFilter, který slouží pro práci s nasbíranými daty:

```
module SmoothingFilterC {
  provides command uint8_t topRead(uint8_t* array, uint8_t len);
  uses command uint8_t bottomRead(uint8_t* array, uint8_t len);
}

implementation
{
...
}
```

SmoothingFilter poskytuje funkci topRead a proto ji musí implementovat, ostatní komponenty ji pak mohou využívat. Naproti tomu využívá funkci bottomRead, která je součástí jiné komponenty.

4.2 Propojení komponent

Moduly jsou základními kameny nesC aplikací. Konfigurace vzájemně propojují moduly vytvořením instancí modulů nebo připojením jejich rozhraní na rozhraní komponenty. Konfigurace se skládá ze dvou bloků kódu. V bloku začínajícím klíčovým slovem *configuration* je možné specifikovat, která rozhraní komponenta používá a která naopak nabízí ostatním, stejně tak jako u modulů.

Za klíčovým slovem *implementation* následuje blok kódu s konfigurací. Nejprve jsou specifikovány reference na komponenty. V níže uvedeném zdrojovém kódu z Bink aplikace jsou to MainC, BlinkC, LedsC a tři časovače.

NesC používá operátory $\rightarrow$, $\leftarrow$ a $=$ pro napojení rozhraní. Šipka směřující vpravo určuje vztah uživatel $\rightarrow$ poskytovatel. Oproti tomu šipka směřující vlevo určuje vztah inverzní. Rovnost pak určuje, že rozhraní komponenty je přímo napojeno na rozhraní vnitřního modulu. Jako příklad konfigurace zde můžeme uvést konfiguraci BlinkAppC, která je následující:

```
configuration BlinkAppC {
}
implementation {
  components MainC, BlinkC, LedsC;
  components new TimerMilliC() as Timer0;
  ...
  BlinkC -> MainC.Boot;
  BlinkC.Leds -> LedsC;
  ...
}
```

4.3 Rozhraní

V běžné praxi není příliš obvyklé, aby komponenty deklarovaly jednotlivé funkce, jako na příkladě SmoothingFilteru. Místo toho nesC využívá rozhraní, která obsahují souhrn souvisejících funkcí. Rozhraní tedy vytvářejí jakési standardizované body pro přípojení komponent. Rozhraní specifikují množinu příkazů *(commands)*, což jsou funkce implementované poskytovatelem rozhraní. Mimoto specifikuje rozhraní také množinu událostí *(events)*.

Příkladem rozhraní je rozhraní Send, které je určeno pro posílání dat v bezdrátové senzorové síti:

```
interface Send {
  command error_t send(message_t* msg, uint8_t len);
  event void sendDone(message_t* msg, error_t error);
  ...
}
```

Většina běžných operací v TinyOs, jako např. odeslání paketu do sítě, měření dat pomocí senzorů nebo práce s flash pamětí, je rozdělena na části. Je to proto, aby operace byly neblokující a bylo umožněno paralelní provádění operací a systém tak nemusel čekat na dokončení právě prováděné operace. Pro toto rozdělení budeme dále používat termín *split-phase*.

Důležitou charakteristikou split-phase je, že operace jsou obousměrné tzn. že se zahájení split-phase operace děje na základě zavolání příkazu (command) a dokončení operace je oznámeno odesláním signálu události (event).

Split-phase operaci je možné ilustrovat na výše uvedeném rozhraní Send. To obsahuje příkaz `send`, který se pokusí odeslat paket do sítě a událost `sendDone`, která signalizuje dokončení operace. Po zavolání příkazu send dojde tedy k pokusu o odeslání poketu do sítě. Po zavolání tohoto příkazu je možné provést nějaký kód. Rozhraní send po ukončení pokusu o odeslání pošle signál události sendDone. Tato událost obsahuje v argumentu výsledek operace a je jí předáno vykonávání kódu.

Pokud modul využívá rozhraní, které obsahuje split-phase operace, musí být všechny události z tohoto rozhraní v modulu implementovány.

Kapitola 5

Agenti

Tato práce se zabývá tématem použití inteligentních agentů v bezdrátových senzorových sítích a z tohoto důvodu budou v této kapitole stručně zmíněny některé základní informace, které se týkají agentů.

Následující text bude čerpat informace ze zdrojů [13] a [23].

Agent[1] je podle jedné z definic[13] něco, co vnímá své okolní prostředí skrz senzory a samostatně vykonává akce v prostředí pomocí efektorů tak, aby vyhověl stanoveným požadavkům.

Typů agentů existuje velké množství. Lidský „agent“ používá oči, uši a jiné orgány jako senzory, naproti tomu jeho akčními členy jsou ruce, nohy, hlas, atd. Robotický agent může používat jako senzory kamery, čidla a jako akční členy různé motory pro ovládání pohybu.

Agent je situován do prostředí, ve kterém působí. To může být rozlišováno na několik kategorií podle sledovaných vlastností. Prostředí tedy může být např. deterministické, nedeterministické, statické, dynamické, diskrétní, spojité, aj..

Deterministické prostředí je takové prostředí, kde každá provedená akce má právě jeden garantovaný efekt a není v něm tedy žádná neurčitost ohledně jeho výsledného stavu po provedení akce. Naproti tomu v nedeterministickém prostředí může mít provedení akce několik efektů. Jako příklad nedeterministického prostředí zde lze uvést "fyzický svět".

Statické prostředí je měněno jen akcemi, které provede agent. V tomto prostředí se samo od sebe nic nemění. Naproti tomu prostředí dynamické je měněno dalšími probíhajícími procesy a prostředí je tak měněno bez kontroly agenta.

Hlavní vlastností agenta je *autonomie*, což je schopnost jednat nezávisle a tím plně kontrolovat své jednání.

5.1 Inteligentní agent

Inteligentní agent je počítačový systém, který je schopen flexibilně vykonávat akce v prostředí. Flexibilitou zde chápeme tyto vlastnosti:

- **reaktivní** - udržuje interakci s prostředím, bezprostředně reaguje na změny v prostření adekvátní akcí,
- **proaktivní** - dokáže se ujímat iniciativy a sám tak ovlivňovat prostředí, aby dosáhl svých cílů,

[1] Tento pojem pochází z latinského slova *agentem*, jehož význam je "ten, kdo koná".

- **sociální** - má schopnost jednat s ostatními agenty, spolupracovat nebo řešit konflikty.

Agenti mohou mít i další vlastnosti, mezi které se řadí např. racionalita, adaptivita nebo mobilita. *Racionalita* znamená, že agent koná s cílem nejlepšího předpokládaného výsledku. Pokud má znalost, že některá z akcí, kterou může vykonat, by vedla k k cíli, pak ji vykoná. *Adaptivita* jako další vlastnost umožňuje agentovi časem zlepšovat svoji výkonnost. *Mobilita* dává možnost agentovi přemísťovat se v rámci sítě.

5.2 BDI agenti

Model BDI agentů *(Belief Desire Intention)* rozděluje agenta na tři části:

- **belief** - množinu znalostí o okolním prostředí,
- **desire** - množinu plánů,
- **intention** - plán který se rozhodl plnit.

Agent tedy obsahuje množinu znalostí o okolním prostředí (světě), množinu plánů a plán, který se rozhodl plnit. Plán zde představuje posloupnost akcí, které vedou ke splnění zvoleného cíle.

Kapitola 6

WSageNt

V této kapitole se budeme zabývat popisem projektu WSageNt. Tento projekt je složen z několika částí, které mezi sebou spolupracují.

6.1 ALLL

ALLL *(Agent Low Level Language)*[22] je jazyk pro popis agentů, který byl navržen s ohledem na požadavky bezdrátových senzorových sítí. Jazyk ALLL je lehce interpretovatelný, avšak dovoluje vytvářet agenty, kteří jsou mobilní, mající schopnost komunikace, jsou robustní a proaktivní. Oproti jiným podobným jazykům a kalkulům přidává tento jazyk navíc některé užitečné vlastnosti, mezi které patří především obsluha výjimek, meta-rozhodování nebo klonování. Jazyk je složen z vět, které reprezentují jednotlivé plány agenta. To jsou opět věty, které jsou složené z jednotlivých akcí. Jazyk tak umožňuje hierarchické zanořování plánů. Tyto plány lze spouštět přímo, kdy akce podplánu jsou přímo parametrem této akce nebo spouštět nepřímo, kdy jsou pojmenované plány uloženy v bázi plánů. Pokud se nepodaří provedení nějaké akce v podplánu, je podplán smazán a pokračuje se v provádění plánu na vyšší úrovni. Další zajímavou vlastností je, že ALLL pro vyhledávání svých znalostí používá operaci zobecňování v bázi znalostí.

Jazyk umožňuje vykonání několika typů akcí, jimiž jsou např. spuštění podplánu, operace pro práci s bází znalostí a volání služeb platformy. Významnou akcí je volání služeb platformy. Při požadavku na službu platformy je platformě předáno jméno služby a parametry a platforma poté provede konkrétní akci.

6.2 Platforma pro agenty

Tato část projektu vznikla v roce 2009 jako diplomová práce Ing. Jana Horáčka[11]. Jejím účelem bylo vytvoření vhodného prostředí pro činnost agentů, která může zahrnovat např. radiovou komunikaci, mobilitu či měření dat ze senzorů.

Platforma tedy vytváří mezivrstvu mezi agentem, popsaným v jazyce ALLL a mezi hardware a jeho operačním systém. Platforma nabízí interpretu sadu služeb, které může agent využít pro interakci s okolím. Mezi tyto služby patří zejména odesílání zpráv a agentů na jinou platformu, ovládání LED diod na uzlech, pozastavení činnosti na určitý čas, měření dat ze senzorů nebo množinu algoritmů pro aritmetické operace a v neposlední řadě také práci se seznamy.

Celá platforma je rozdělena na část platformy a část interpretu. Tyto obě nezávislé části mezi sebou pomocí sady rozhraní komunikují. Platforma je zodpovědná za řízení činnosti interpretu a poskytuje mu abstraktní funkce pro řízení systémových modulů. Řízení je možné rozdělit na inicializaci interpretu, provedení jednoho kroku a informování o dokončení vnější činnosti nebo změně okolí. Při inicializaci se provádí nastavení interpretu, plán agenta je vložen na zásobník a jsou naplněny tabulky počátečními daty. Inicializace se také provádí po spuštění nebo příjmu agenta, aby mohl provádět svou činnost. Po inicializaci je spuštěna smyčka interpretace kódu nebo se platforma uspí a čeká do té doby, než je kód agenta přijat přes transceiver.

Při provedení jednoho kroku interpretu předá platforma řízení interpretu, který provede první akci v plánu. Ten pak podle výsledku informuje platformu o dalším průběhu. Poté může dojít k pozastavení interpretu a čekání na vnější událost nebo pokračování v interpretaci.

V následujícím textu budou popsány některé služby agentní platformy, včetně způsobu jakým pracují. Souhrn všech služeb, včetně parametrů a stručného popisu, je možné nalézt v tabulce 6.1. Tato tabulka obsahuje informace z doby, kdy vznikla platforma.

6.2.1 Ovládání LED diod

Uzel platformy Iris obsahuje 3 LED diody (červená, zelená a žlutá), které je možné využít např. k signalizaci nějakého stavu nebo činnosti na uzlu uživateli. Rozhraní *LedkyI* umožňuje interpretu rozsvítit, zhasnout nebo překlopit stav kterékoliv z diod. Toto rozhraní je synchronní a proto již žádnou další synchronizaci nevyžaduje.

Tyto služby mají v jazyce ALLL společný identifikátor "l". Poté následuje n-tice parametrů, kde první specifikuje s kterou LED diodou se bude pracovat (r, g, y) a druhý volitelný je hodnota buď 1 pro rozsvícení nebo 0 pro zhasnutí. Pokud není druhý parametr uveden, dojde k překlopení stavu na diodě. Rozsvícení zelené diody se provede kódem: `$(l,(g,1))`.

6.2.2 Služby pro řízení uspání

Platforma umožňuje uspat interpret na zvolenou dobu. Tato služba má v jazyce ALLL identifikátor "w". Uspání interpretu na jednu vteřinu se provede kódem: `$(w,(1000))`. Platforma dokonce nabízí agentovi uspání interpretu, než je mu zaslána nějaká zpráva.

6.2.3 Služby pro zasílání zpráv

Jelikož je vyžadována komunikace mezi jednotlivými uzly, obsahuje platforma služby pro odesílání a příjem zpráv. Služba pro odeslání zprávy je v jazyce ALLL volána symbolem '!', kde následuje n-tice parametrů. Prvním parametrem je id cílového uzlu a v druhé n-tici je obsažena samotná zpráva. Tedy např. `!(2,(zpravaprouzel2))` odešle uzlu číslo 2 zprávu "zpravaprouzel2".

Kód	Parametry	Popis
a	žádné	Aktivace sledování příchozích zpráv při běžícím interpretu.
f	seznam|registr	Vloží do aktivního registru první prvek seznamu jako jednoprvkovou n-tici nebo první ze seznamů, je-li jich více.
k	žádné	Zastaví činnost interpretu.
l	seznam|registr	Ovládá LED diody na uzlech. Parametr musí obsahovat kód barvy LED(r, g, y) a za ním může být stav LED (0 - nesvítí, 1 - svítí), není-li zadán, dojde k přepnutí stavu.
m	seznam|registr	Parametr této služby obsahuje adresu (číslo) platformy, kam se má kód agenta přesunout. Celý kód se zkopíruje do cílové platformy a provádění pokračuje na obou platformách dále. Je-li cílová adresa stejná jako ta, kde agent momentálně je, provedením této služby se nic nestane. Za adresou může být parametr s, který značí, že se má zastavit provádění kódu na zdrojové platformě.
n	žádné	Objevování sousedů.
r	seznam|registr	Vloží do aktivního registru zbytek seznamu bez prvního prvku nebo všechny seznamy bez prvního z nich. Když je seznam pouze jeden a obsahuje jediný prvek, vloží se do akt. registru prázdná n-tice.
s	žádné	Zastaví provádění kódu, dokud nepřijde zpráva z rádia. Bylo-li aktivováno sledování a zpráva už byla přijata, provádění pokračuje dál.
w	seznam|registr	Pozastavení provádění kódu na zadaný čas v milisekundách.
d	žádné|seznam|registr	Zavoláním této služby bez parametru dojde ke změření aktuální teploty, v opačném případě je nutné zadat typ a hodnoty (a - průměr, m - minimum, M - maximum) a počet hodnot, z kolika se má hodnota vypočítat. Když není naměřen dostatečný počet hodnot pro výpočet, končí služba chybou.

Tabulka 6.1: Přehled služeb platformy.

6.3 Interpret

Tato část vznikla jako bakalářská práce Bc. Pavla Spáčila[20] současně s agentní platformou. Interpret slouží k vykonávání kódu agenta, který je popsán v jazyce ALLL. Interpret pracuje na principu zásobníkového automatu, kde je plán agenta uložen po jednotlivých akcích na zásobníku. Během provádění akce může interpret zavolat v každém kroku nejvýše jednu službu platformy, aby nedošlo k přepsání nějaké části agenta nebo k problémům se synchronizací. Hlavními typy použitých datových struktur v interpretu je tabulka a zásobník.

Tabulka je tvořena seznamy, které nejsou uspořádány, tedy na jejich pořadí nezáleží. Všechny tabulky v interpretu mají stejnou strukturu a operace, ovšem význam jednotlivých

tabulek je různý. Do tabulek je možné vkládat seznamy, mazat je nebo unifikovat s jinými seznamy. Seznam zde naproti tomu obsahuje uspořádanou posloupnost akcí. Každá akce je uvozena svým unikátním symbolem a proto není třeba akce nějak oddělovat. Začátek seznamu je identifikován znakem "(" a konec znakem ")".

Zásobník je použit pro aktuálně zvolený cíl. Prvky zásobníku představují akce, které se mají provést. Zásobník umožňuje vkládat prvky na vrchol, zjistit, který je na vrcholu a po provedení akce jej smazat.

Agent byl rozdělen na 7 níže uvedených částí, přičemž stále splňuje definici BDI agentů. **Báze plánů** (planBase) v sobě obsahuje pojmenované plány, které je možné vložit na zásobník v průběhu interpretačního cyklu. V **bázi vstupů** (input base) jsou uloženy n-tice obdržené od platformy, jako např. naměřené hodnoty ze senzorů nebo přijaté zprávy. **Báze znalostí** (belief base) obsahuje konkrétní hodnoty získané během své činnosti. Aktuální **plán**, který se provádí a **tři registry** (Registr1-3), které jsou určeny pro dočasné uložení výsledků některých služeb platformy.

6.3.1 Sémantika jednotlivých akcí

Tabulka 6.2 obsahuje seznam typů akcí v jazyce ALLL.

Kód akce	**Parametry**	**Popis**
+	n-tice|registr	Přidání do belief base, unifikací se kontroluje duplicitní vkládání.
-	n-tice|registr	Odebrání položek z belief base pomocí unifikace.
!	číslo|registr n-tice|registr	Odeslání zprávy zadané n-ticí nebo registrem na uzel se zadanou adresou.
?	číslo|registr	Test input base na zprávu od uzlu/senzoru se zadanou adresou.
@	seznam akcí	Přímé spuštění. Akce se vloží na zásobník se zarážkou.
∧	jméno|registr	Nepřímé spuštění, kdy se hledá plán v PlanBase se stejným jménem.
&	číslo	Změna aktivního registru.
*****	n-tice|registr	Test belief base na zadanou n-tici nebo registr. Výsledek se uloží do aktivního registru.
$	písmeno {n-tice|registr}	Volání služeb platformy, kde první parametr (písmeno) je kód operace a druhým parametrem jsou parametry služby(nemusí být u všech služeb)
#	žádné	Zarážka za plánem, sémanticky tato akce nemá žádný význam.

Tabulka 6.2: Seznam typů akcí v ALLL.

6.3.2 Přidání do báze znalostí

Tato akce je v jazyce ALLL identifikována symbolem "+". Akce nejprve unifikuje zadanou n-tici nebo registr s tabulkou a v případě, že nebyla n-tice nalezena, vloží ji na začátek tabulky. V každé akci je možné namísto konkrétní hodnoty použít zástupný symbol & a číslo registru. Ten se před prováděním akce nahradí aktuálním obsahem zadaného registru.

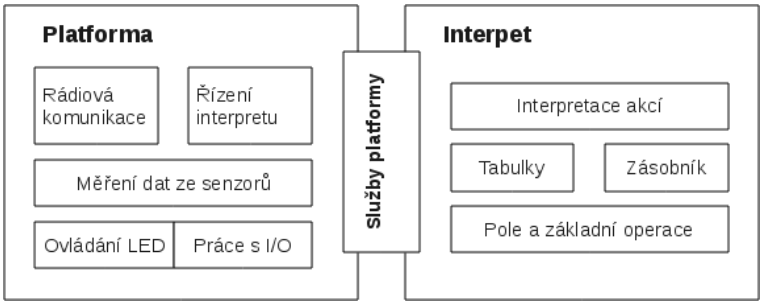
Akce přidání položek do báze znalostí nejprve vloží n-tice tak, jak jsou uvedeny v akci a poté dojde k nahrazení zástupných symbolů.

Např. pokud je v prvním registru r1 n-tice (456), pak akce +(123,&1) vloží n-tici(123, 456) na začátek báze znalostí. V případě opakovaného volání akce to nebude mít žádný význam, protože báze znalostí již takovou n-tici obsahuje.

6.4 Propojení platformy a interpretu

Pro spojení interpretu a platformy je určené rozhraní PlatformSvcI. Agent má schopnost řízení běhu programu, nemá ovšem žádné výpočetní možnosti. Z tohoto důvodu existuje sada služeb platformy (tab. 6.1), které může agent pro svou činnost využít. Přes akci volání služby platformy je v modulu PlatformSvcC zjištěn na základě obsahu n-tice typ služby a příslušné parametry. Poté je přes rozhraní požadované služby[1] zavolána příslušná funkce z platformy.

Obrázek 6.1 ilustruje sturkturu platformy a interpretu.



Obrázek 6.1: Struktura platformy a interpretu.

6.5 Webové rozhraní

Webové rozhraní vzniklo v roce 2010 jako výsledek diplomové práce Bc. Martina Gábora[10]. Navazuje na práci Ing. Horáčka, který vytvořil platformu pro agenty a Bc. Spáčila, který vytvořil interpret ALLL agentního kódu.

Cílem této práce bylo vytvořit uživatelské prostředí, ve kterém je možné jednoduše využívat služeb platformy projektu WsageNt a jednoduše tak pracovat s bezdrátovou senzorovou sítí. Aplikace je založena na technologii JavaEE a využívá architektury MVC *(Model View Controller)*. V práci je dále použit framework *Hibernate* pro perzistentní mapování objektů do relační databáze a framework Struts2. Výstupem práce jsou dvě samostatné, ale vzájemně spolupracující aplikace:

- **BSComm**: konzolová aplikace sloužící pro komunikaci s basestation,
- **ControlPanel**: webové rozhraní pro interakci s agenty a zobrazování výsledků.

[1] Rozhraní jednotlivých služeb platformy, které může interpret využít se nachází ve složce Interface.

6.5.1 BSComm

Tato aplikace byla vytvořena na platformě Java SE a její hlavní úlohou je přeposílání zpráv od webového rozhraní do basestation a naopak. Při spuštění aplikace očekává na vstupu cestu k sériovému portu, kde je připojena basestation a port soketu, na kterém probíhá komunikace s webovým rozhraním.

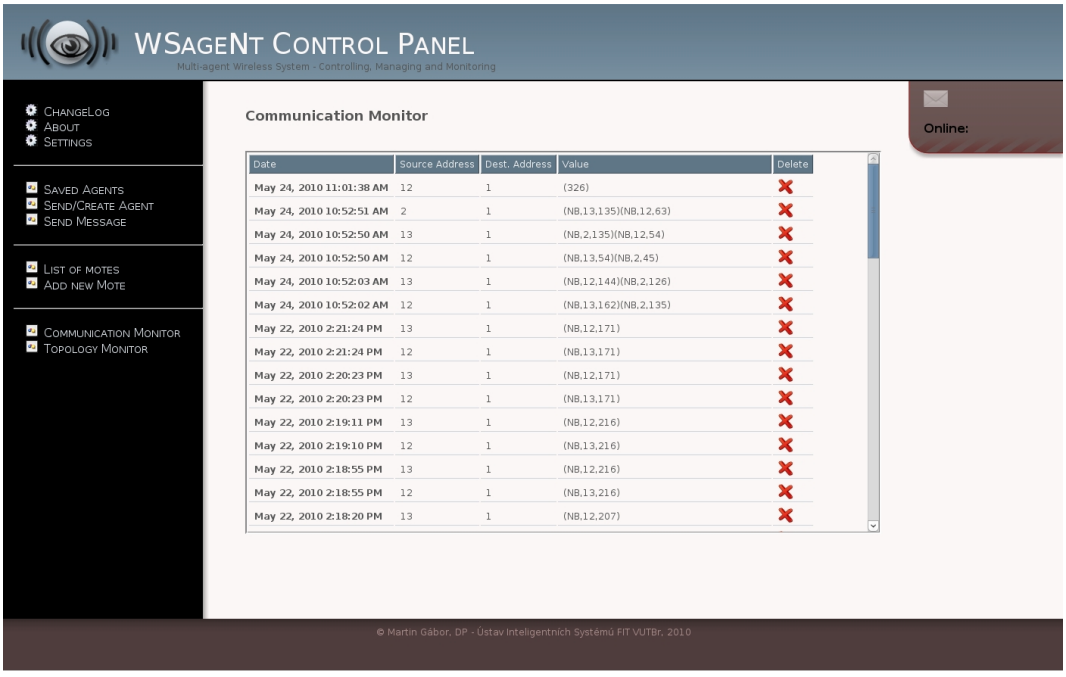
Příklad: `serial@/dev/ttyUSB1:iris 7777`

6.5.2 ControlPanel

Control Panel vytváří webové rozhraní, které slouží pro zjednodušení interakce s bezdrátovou senzorovou sítí. Výrazy webové rozhraní a Control panel budou pro další označení ekvivalentní. Obrázek 6.2 ilustruje webové rozhraní pro práci se sítí. Webové rozhraní obsahuje panel pro kontrolu nových přijatých zpráv ze sítě a informaci o aktuálně aktivních uzlech v síti.

Před prací se sítí je nejprve nutné, aby uživatel v rozhraní postupně přidal jednotlivé uzly, které se v sítí nachází. Ty jsou poté uloženy v databázi a není nutné v případě stejných uzlů tento proces opakovat.

Do bezdrátové senzorové sítě je možné posílat buď jednoduché zprávy, nebo přímo agenty. Webové rozhraní umožňuje vytvořit agenta pomocí kódu jazyka ALLL, poslat ho do sítě a v neposlední řadě ho také uložit do databáze pro pozdější použití.



Obrázek 6.2: ControlPanel.

Control Panel na základě odeslání agenta na všechny uzly, které jsou v rádiovém dosahu basestation, využitím služby objevování sousedů(`$(n)`) umožňuje zjistit přibližné rozmístění uzlů v síti. Výpočet rozmístění uzlů se provádí na základě získané síly signálu mezi uzly a analytické geometrie rovinných útvarů. V původní verzi, tak jak ji vytvořil M.Gábor[10], je možné zjistit rozmístění uzlů pouze v rádiovém dosahu uzlu, na který byl

agent odeslán. Tato skutečnost výrazným způsobem omezuje použití tohoto přístupu pro větší sítě. Navíc je ještě před touto činností nezbytné znát uzly, které síť tvoří, aby bylo možné na ně odeslat agenta.

6.5.3 Komunikační protokol

Za účelem komunikace mezi konzolovou aplikací BSComm a webovým rozhraním Control panel existuje komunikační protokol pro výměnu dat. ControlPanel vysílá požadavky do bezdrátové senzorové sítě přes aplikaci BSComm, která vyžívá některých tříd programu Listen z TinyOs pro komunikaci se sítí. Požadavky v sobě obsahují zprávu a adresu koncového uzlu, kterému má být zpráva doručena.

Zpráva může být buď *agentní* (obsahující agenta) nebo *jednoduchá* (obsahující jen data).

Formát agentní zprávy:

```
typ_zprávy ODDĚLOVAČ adresa_uzlu ODDĚLOVAČ planBase ODDĚLOVAČ plan ODDĚLOVAČ
belief base ODDĚLOVAČ input base
```

Formát jednoduché zprávy:

```
typ_zprávy ODDĚLOVAČ adresa_uzlu ODDĚLOVAČ obsah_zprávy
```

Kde:

- typ_zprávy: je roven textovému řetězci A, což znamená, že se jedná o agenta
- ODDĚLOVAČ: je roven textovému řetězci ;.,WS,-;
- adresa_uzlu: určuje id uzlu, na kterou se pošle agent, zpráva
- planBase: báze plánů agenta
- belief base: báze znalostí agenta
- input base: přijaté zprávy, naměřené hodnoty agenta
- plan: záměr(plán) agenta
- obsah_zprávy: textový řetězec definující obsah zprávy

Příklad: M;.,WS,-;3;.,WS,-;(100)

Po přijetí požadavku se BSComm pokusí provést požadovanou akci a informuje Control panel o jejím výsledku. V současné době jsou implementovány tyto odpovědi:

Číslo	**Konstanta**	**Význam**
0	SENDING_SUCCES	Úspěch operace
1	SENDING_ERROR	Neúspěch operace
2	SENDING_EXPIRED	Vypršení času
3	SENDING_DEV_BUSY	Zařízení je zaneprázdněno

Tabulka 6.3: Odpovědi Control panelu.

Kapitola 7

Návrh rozšíření projektu WSageNt

V předchozích kapitolách jsme se zabývali popisem technologie bezdrátových senzorových sítí včetně konkrétních zařízení. Byly nastíněny základní nástroje a koncepty, které se uplatňují při vývoj aplikací pro bezdrátové senzorové sítě. Dále byl také čtenář seznámen se současným stavem projektu WSageNt, který se zabývá použitím agentů v prostředí bezdrátových senzorových sítí.

V následujícím textu se budeme nejprve věnovat měření síly signálu mezi dvěma uzly platformy Iris a poté návrhem rozšíření projektu WSageNt. Zaměříme se na možnost použití pachových stop agentů v projektu WSageNt a vytvoření podpory práce s těmito stopami. Těchto nových prvků bude poté možné využít k vytvoření nových agentů s většími schopnostmi. Jako příklad bychom mohli uvést vytvoření agenta, který projde celou bezdrátovou síť a získá z ní nějaké informace. Těchto pachových stop bude především využito pro vytvoření agenta, který bude sloužit pro sledování pohybu uzlů v bezdrátové senzorové síti.

Původní verze webového rozhraní umožňuje zobrazit rozmístění uzlů v síti. Tato funkce pracuje tak, že z basestation je na každý uzel v jejím dosahu odeslán agent. Ten zjistí sílu signálu ke svým okolním uzlům a vrátí se zpět s těmito informacemi do basestation. Rozsah uzlů, který je možné takto zobrazit, je značně omezen přímým radiovým dosahem k basestation. Naproti tomu rozšíření projektu WSageNt o prvek pachových stop a služeb s nimi pracujících umožní vytvořit agenta, který projde celou síť a bude tak možné zobrazit všechny uzly v síti.

7.1 Měření síly signálu

Platforma Iris obsahuje transceiver RF230. Ten ukládá informace o RSSI do 8-bitového registru, ve kterém je hodnota RSSI uložena v dolních 5-ti bitech. Získaná hodnota je celé číslo od 0 do 28, která označuje sílu obdrženého signálu jako lineární křivku na logaritmické stupnici vstupního signálu (dBm) s rozlišením 3 dB. Hodnota RSSI 0 indikuje, že síla vstupního radio-frekvenčního signálu je $\leq$ 91dBm. Naopak hodnota 28 říká, že síla signálu je $\leq$ -10dBm. Pro zjištění skutečné hodnoty RSSI v rozsahu 1-28 je nutné použít následující vzorec[5]:

```
P=RSSI_BASE_VAL + 3(RSSI-1)
```

RSSIBASE_VAL je minimální citlivost RSSI, která je dána hodnotou -91dBb. RSSI je získaná hodnota z registru.

Pro potřeby konverze hodnot získaných z platformy na reálnou hodnotu RSSI v dBm byla vytvořena tabulka 7.1. Ta byla z důvodu velikosti rozdělena na pět částí. Transceiver umožňuje nastavit velikost vysílacího výkonu, uvedená tabulka a další měření platí pro standardně nastavený vysílací výkon.

RF230	0	1	2	3	4	5	6
Platforma	0-4	5-13	14-22	23-31	32-40	41-49	50-58
RSSI [dBm]	< -91	-91	-88	-85	-82	-79	-76
RF230	7	8	9	10	11	12	13
Platforma	59-67	68-76	77-85	86-94	95-103	104-112	113-121
RSSI [dBm]	-73	-70	-67	-64	-61	-58	-55
RF230	14	15	16	17	18	19	20
Platforma	122-130	131-139	140-148	149-157	158-166	167-175	176-184
RSSI [dBm]	-52	-49	-46	-43	-40	-37	-34
RF230	21	22	23	24	25	26	27
Platforma	184-193	194-202	203-211	212-220	221-229	230-238	239-247
RSSI [dBm]	-31	-28	-25	-22	-19	-16	-13
RF230	28						
Platforma	248-255						
RSSI [dBm]	> -10						

Tabulka 7.1: Konverze hodnot na RSSI.

7.1.1 Postup měření

Pro měření síly signálu jsme použili dva uzly XM2110 platformy Iris a basestation MIB520CB, která byla pomocí USB rozhraní připojena k notebooku s operačním systémem Ubuntu 10.04. V notebooku byl spuštěn komunikační most BScomm a webové rozhraní Control panel, z kterého byli posíláni agenti na jednotlivé uzly. Po jejich navrácení byly z webového rozhraní odečítány získané hodnoty. Měření bylo prováděno na otevřeném prostranství za přímé viditelnosti uzlů, abychom omezili vliv odrazů na sílu signálu a také v uzavřené místnosti. Rozsah vzdáleností pro měření byl zvolen v intervalu 1 - 30 m pro venkovní prostředí a 0.5 – 4000 cm pro vnitřní prostředí.

Pro měření signálu mezi jednotlivými zařízeními poskytuje platforma službu objevování sousedů. Ta je v jazyce ALLL identifikována symbolem "n". Výsledkem této služby jsou n-tice, které jsou uloženy do belief base. N-tice jsou tvaru (NB, id, strenght), kde NB je identifikační řetězec služby objevování sousedů, id je číselná hodnota identifikující uzel a strenght je číslo udávající sílu signálu k uzlu. Hodnota strenght je v platformě převedena normalizační funkcí na celé číslo z intervalu <0, 255>. Tato funkce[1] je použita proto, že různé hardwarové platformy dávají jako výsledek získání síly signálů odlišné hodnoty.

Kód agenta, který jsme použili pro měření byl v jazyce ALLL následující[10]:

```
$(n)$(m,(1))
```

Agent nejprve zjistí sílu signálu ke svým sousedům. Tyto informace uloží do belief base, počká náhodně zvolenou dobu a poté následně provede přesun na uzel č. 1. Uzel č. 1 byl

[1] Více informací lze nalézt v modulu NeighbourDiscoveryM.nc v adresáři Platforma

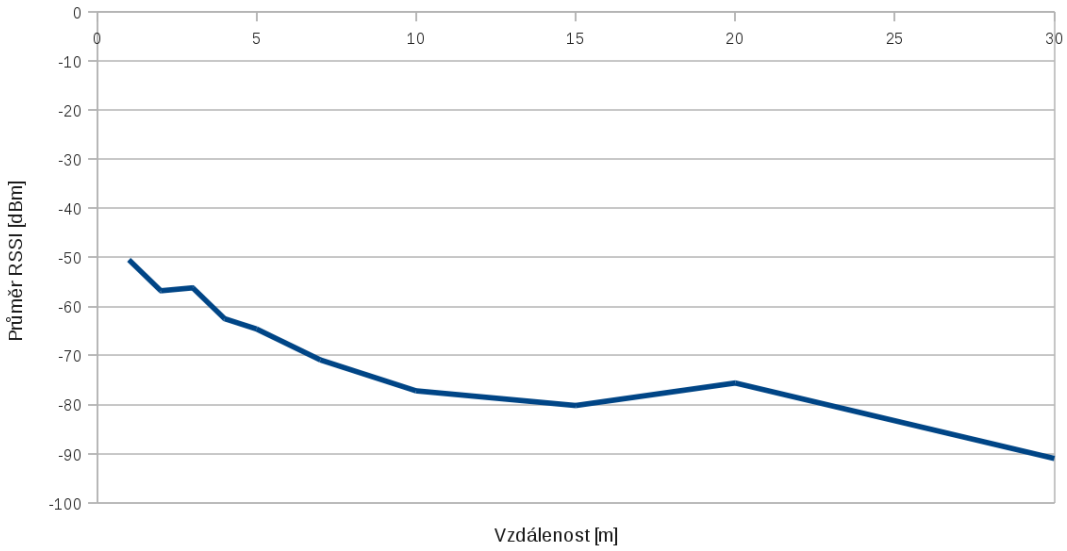
v tomto případě basestation. Takto získané hodnoty je nezbytné převést podle tabulky 7.1 na hodnoty v jednotkách dBm.

7.1.2 Naměřené hodnoty

Měření pro jednotlivé hodnoty vzdáleností bylo provedeno desetkrát, přičemž každý uzel měřil hodnotu ke svému sousedovi pětkrát. Tabulka 7.2 obsahuje výsledky naměřených hodnot pro venkovní prostředí. Levý sloupec obsahuje vzdálenost v metrech mezi dvěma uzly. V horním sloupci jsou uvedeny informace o měření ve tvaru k.l, kde k je k-té měření a l je l-tý uzel, který prováděl měření ke svému sousednímu uzlu. V sloupci a je vypočtený aritmetický průměr naměřených hodnot. Závislost hodnot síly signálu na vzdálenosti je zobrazen v grafu 7.1.

l[m], P[dBm]	**1.1**	**1.2**	**2.1**	**2.2**	**3.1**	**3.2**	**4.1**	**4.2**	**5.1**	**5.2**	**a**
1	-52	-49	-52	-49	-49	-49	-52	-49	-52	-52	-50.2
2	-61	-55	-58	-55	-58	-55	-58	-55	-58	-55	-56.8
3	-58	-55	-58	-55	-55	-55	-58	-55	-55	-58	-56.2
4	-61	-61	-61	-61	-64	-64	-64	-64	-64	-61	-62.5
5	-64	-64	-64	-64	-67	-64	-64	-64	-67	-64	-64.6
7	-70	-70	-70	-70	-73	-70	-73	-73	-70	-70	-70.9
10	-73	-73	-79	-79	-79	-79	-79	-79	-79	-73	-77.2
15	-85	-85	-82	-82	-76	-73	-82	-82	-82	-73	-80.2
20	-91	-91	-82	-82	-85	-82	-85	-82	-85	-82	-75.6
30	-91	-91	-91	-91	-91	-91	-91	-91	-91	-91	-91

Tabulka 7.2: Naměřené hodnoty venku.



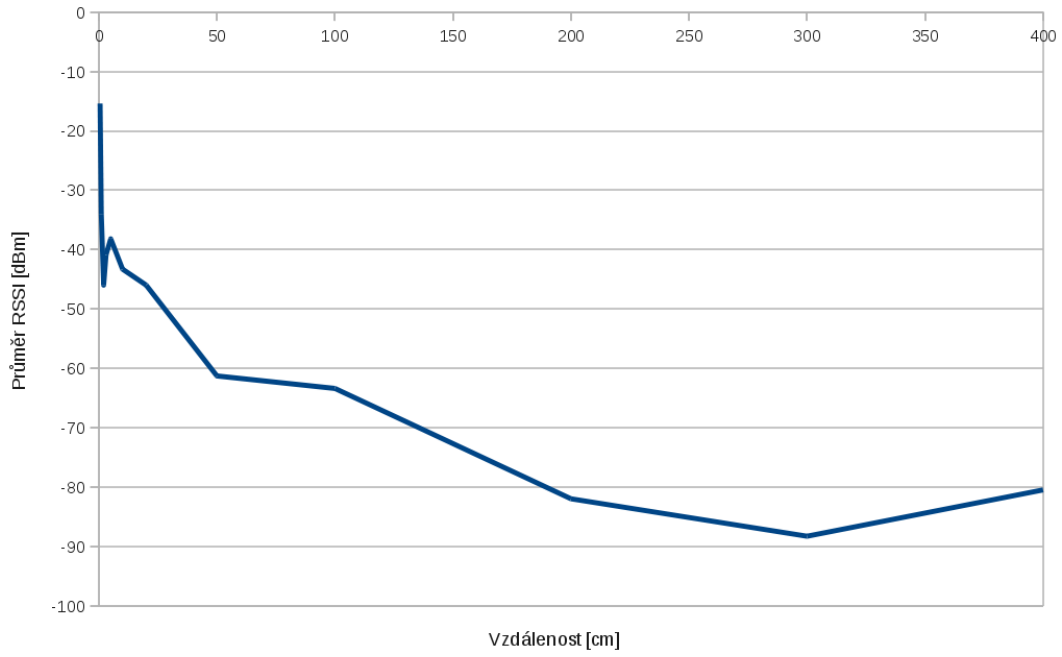
Obrázek 7.1: Graf hodnot vzdálenosti a průměru RSSI pro venkovní prostředí.

Tabulka 7.3 obsahuje naměřené hodnoty síly signálu pro vnitřní prostředí. Závislost hodnot síly signálu na vzdálenosti je zobrazen v grafu 7.2.

Je důležité zde upozornit na to, že když srovnáme průměrné hodnoty sil signálu ve vnitřním a venkovním prostředí, tak se vzdálenosti mezi uzly významně liší. Např. hodnotě síly signálu -88.3 dBm pro vnitřní prostředí odpovídá vzdálenost mezi uzly 3 m. Podobné hodnotě síly signálu -91 dBm ve venkovním prostředí ovšem odpovídá vzdálenost mezi uzly 30 m.

l [cm], P[dBm]	**1.1**	**1.2**	**2.1**	**2.2**	**3.1**	**3.2**	**4.1**	**4.2**	**5.1**	**5.2**	**a**
0.5	-16	-16	-16	-16	-16	-16	-16	-13	-13	-16	-15.4
1	-34	-34	-34	-34	-34	-34	-34	-34	-34	-34	-34
2	-43	-49	-43	-49	-43	-49	-43	-49	-43	-49	-46
3	-40	-43	-40	-40	-40	-40	-40	-43	-40	-43	-40.9
5	-37	-40	-37	-40	-37	-40	-37	-37	-40	-37	-38.2
10	-43	-43	-43	-43	-43	-43	-43	-43	-46	-43	-43.3
20	-46	-46	-46	-46	-46	-46	-46	-46	-46	-46	-46
50	-61	-64	-61	-61	-61	-61	-61	-61	-61	-61	-61.3
100	-64	-64	-64	-64	-64	-64	-64	-61	-64	-61	-63.4
200	-82	-82	-82	-82	-82	-82	-82	-82	-82	-82	-82
300	-85	-91	-88	-91	-88	-91	-88	-91	-85	-85	-88.3
400	-79	-82	-79	-79	-82	-79	-82	-79	-82	-79	-80.5

Tabulka 7.3: Naměřené hodnoty uvnitř budovy.



Obrázek 7.2: Graf hodnot vzdálenosti a průměru RSSI uvnitř budovy.

7.2 Pachové stopy agentů

Aby bylo možné agenta nějakým způsobem identifikovat, je nutné zajistit jeho jedinečnou identifikaci v bezdrátové senzorové síti. Pro tento účel bude tedy nezbytné přidat nové agentní prvky, které toto umožní. Dále bude nezbytné vytvořit potřebné služby platformy, které budou pracovat s těmito informacemi. Těchto nových prvků a služeb bude možné využít pro konstrukci agentů s většími schopnostmi, které do této doby nebyly možné. Bude možné např. vytvořit agenta, který pomocí těchto nových prvků projde celou bezdrátovou senzorovou síť nebo stopovat agenta, jak procházel sítí.

Identifikace agentů bude zajištěna identifikátory id a třídou agenta. Tyto hodnoty by měly být v rámci sítě jedinečné, přičemž třída agentů může být pro podobné agenty stejná, ovšem id musí být v rámci jedné třídy jedinečné.

Jelikož byl agent rozdělen do několika částí, bylo by možné aby měl agent tyto informace o sobě uložené v některé své části, např. v belief base. My však tyto informace o agentovi budeme uchovávat v několika globálních proměnných uvnitř platformy. Při přesunu z jednoho uzlu na druhý budou informace o agentovi v hlavičce spolu s dalšími daty odeslány a na cílovém uzlu budou tyto informace uloženy do globálních proměnných a také do externí paměti flash.

Agent by mohl potřebovat zjistit informaci, zda již na daném uzlu byl či nikoli. K řešení této problematiky se nabízejí dvě možnosti[12]. Bylo by možné do belief base ukládat identifikátory id navštívených uzlů a podle těchto informací zjišťovat, zda na daném zlu již agent byl nebo ne. Druhou možností by bylo zaznamenávat do paměti uzlu informace o agentovi, který se na uzel přesunul a v těchto záznamech pak požadovanou informaci získat. První možnost by byla pravděpodobně jednodušší z pohledu implementace. Ta by spočívala spíše ve vytvoření příslušného agenta v jazyce ALLL a rozšíření platformy a interpretu by bylo spíše okrajové. Druhá možnost bude naproti tomu vyžadovat rozsáhlejší rozšíření platformy i interpretu, nicméně se zdá být výhodnější z důvodu perzistentního uložení záznamů v uzlech. Data bude pak možné využít např. pro zjištění trasy agenta i po ukončení činnosti agenta.

Jak již bylo předesláno, platforma do své paměti uloží informace o příchozím agentovi. Zaznamenávanými informacemi budou tyto následující:

- **from** - id uzlu ze kterého agent přišel,
- **agentId** - id agenta,
- **agentClassId** - id třídy agenta,
- **pathCounter** – pořadí na cestě, v kterém agent navštívil uzel.

Hodnota pathCounter se po přesunu agenta z jednoho uzlu na druhý zvýší o hodnotu 1. Tato hodnota bude tedy říkat, v jakém pořadí navštívil agent uzel, když se pohyboval v síti. Je důležité zde upozornit na to, že uložení této informace umožní zjistit např. v jakém pořadí agent uzly navštívil.

Služby pracující s pachovými stopami budou v jazyce ALLL označeny symbolem "t", kde funkcionalita služeb silně závisí na daných parametrech. V následujícím textu budou jednotlivé argumenty služby "t"popsány[12].

7.2.1 Identifikace uzlu

Služba "t"s jediným parametrem "i"je určena pro identifikaci uzlu. V tomto případě naplní služba aktuálně vybraný registr hodnotou id uzlu, kde se agent právě nachází. Tato služba je jako jedna z mála synchronní.

Příslušný kód v jazyce ALLL je následující:

```
$(t,(i))
```

7.2.2 Backtracking

Další možností služby "t"jsou argumenty (b,f). V tomto případě naplní služba aktuální registr hodnotou id uzlu, ze kterého přišel agent poprvé. Jinou možností je zavolat službu s argumenty (b,l). V tomto případě se do aktuálního registru uloží hodnota id uzlu, ze kterého přišel agent naposledy. Tato služba je asynchronní.

Kód této služby je v jazyce ALLL následující:

```
$(t,(b,f))
```

7.2.3 Dotazování

Za účelem dotazování je zde argument "q". Dalším povinným parametrem je id uzlu, kterého se agent bude ptát. Hodnota id může být zadána buď jako číslo, nebo jako odkaz na některý ze tří registrů. V případě, že tato služba skončí s výsledkem, že agent na dotazovaném uzlu nebyl, agent pokračuje dále v aktuálně prováděném plánu. V opačném případě je aktuálně prováděný plán smazán. Tato služba pracuje asynchronně.

Této službě odpovídá v jazyce ALLL kód:

```
$(t,(q, id))
```

7.3 Agent pro průchod sítí

V této části se budeme zabývat popisem agenta, který projde všechny uzly v bezdrátové senzorové síti. Pro svou činnost využije výše navržených služeb, které pracují s pachovými stopami.

Agent pro průchod sítí bude pracovat následujícím způsobem:

```
1. uložím do belief base informaci o uzlu, na kterém se právě nacházím
2. zjistím, které sousední uzly jsou v mém rádiovém dosahu a tyto informace
   uložím do belief base jako n-tice (NB,id,strenght)
3. vyjmu první n-tici tvaru (NB,_,_) z belief base, získám z ní id sousedního
   uzlu a uložím ho do registru r2
4. jestliže nebyl vyjmut žádný prvek, tj. belief base neobsahuje informace
   o nalezených sousedech, pokračuji bodem 8, jinak pokračuji bodem 5
5. dotáži se uzlu, jehož id je dáno hodnotou r2, zda jsem na něm již byl
6. jestliže jsem na uzlu byl, vrátím se na bod 3, jinak pokračuji bodem 7
7. smažu z belief base zbylé n-tice s nalezenými sousedy, přesunu se na uzel
   daný hodnotou r2 a pokračuji bodem 1
8. zjistím id uzlu z kterého jsem přišel poprvé, přesunu se na tento uzel
   a pokračuji bodem 1
```

Kód tohoto agenta v jazyce ALLL je zobrazen v tabulce 7.4 a činnost je následující[12].

Provádění agenta začne spuštěním plánu `nb`, který v sobě obsahuje pojmenované plány addblf,cycle a back a požadavek na službu objevování sousedů. Nejprve je tedy spuštěn podplán **addblf**, který využitím služby identifikace uzlu "$(t,(i))"zjistí hodnotu id uzlu, na kterém se agent právě nachází. Tuto hodnotu uloží do své belief base jako n-tici tvaru (vis, id).

Po skončení provádění podplánu se agent navrátí zpět do plánu nb, kde zavolá službu objevování sousedů. Ta naplní obsah belief base agenta n-ticemi o nalezených sousedech.

Po ukončení hledání sousedů agent začne vykonávat plán `cycle`. Agent uloží do aktivního registru z belief base jednu ntici s informací o nalezeném sousedovi. Následně je tato n-tice z belief base odstraněna a dojde k vykonávání plánu `testnmove`.

Uvnitř tohoto plánu nejprve agent spustí podplán `getid`, který zpracuje z prvního registru záznam ze služby objevování sousedů a do druhého registru uloží id objeveného uzlu ze záznamu. Dále se pokračuje v plánu testnmove. Agent poté zavolá službu platformy pro pachové stopy s parametrem "q"pro dotazování. Tato služba v případě zjištění, že agent na daném uzlu již byl, smaže aktuální vykonávaný podplán. Tím tedy dojde znovu ke spuštění plánu cycle a výše uvedené činnosti se opakují do té doby než plán testnmove zjistí, že agent na dotazovaném uzlu ještě nebyl. V tomto případě nedojde ke smazání plánu, ale k jeho pokračování. V návaznosti na to jsou pak smazány všechny záznamy o objevených sousedech z belief base a agent se přesune na dotazovaný uzel, kde je spouštěn plán nb a situace se opakuje. V případě, že v plánu cycle došlo ke zpracování všech záznamů z belief base na všechny n-tice a tedy operace `*(NB,_,_)` žádný záznam nenajde, dojde ke smazání plánu cycle a tedy ke spuštění plánu back.

Plán `back` je určen pro navracení agenta zpět na místo odkud přišel. V tomto podplánu je zavolána služba pachových stop backtracking "$(t,(b,f))", která do aktuálního registru uloží id uzlu, ze kterého agent přišel poprvé. Na tento uzel se poté agent přesune a jeho aktuální plán je nastaven na nb. Po provedení tohoto plánu dojde v případě nenalezení nových nenavštívených uzlů opět k vykonání plánu back. Toto se děje do té doby, než se agent přesune na uzel, ze kterého byl vyslán.

Část	**Obsah**

PlanBase	`(nb,(^(addblf)$(n)^(cycle)#^(back)))`
	`(addblf,(&(1)$(t,(i))+(vis,&1)))`
	`(cycle,(&(1)*(NB,_,_)&(2)$(f,&1)-&2^(testnmove)^(cycle)))`
	`(testnmove,(^(getid)$(t,(q,&2))-(NB,_,_)$(m,(&2,s))^(nb)))`
	`(getid,(&(1)$(r,&2)&(2)$(f,&1)))`
	`(back,(&(3)$(t,(b,f))$(m,(&3,s))^(nb)))`

Plan	`^(nb)`

Tabulka 7.4: Agent pro průchod sítí[12].

7.4 Modifikace agenta pro odeslání z uzlu

Agent navržený v předchozí podkapitole je určen pro vyslání do sítě z basestation. V případě jeho návratu zpět využíváme faktu, že je zde spuštěn program Basestation[2] a tak nedochází k interpretaci agentního kódu. Činnost agenta je zde proto ukončena a data agenta přeposlána do PC.

V případě vyslání agenta do sítě z uzlu je nutná modifikace agenta z důvodu, aby jeho činnost byla ukončena při návratu na počáteční uzel. Modifikace zde spočívá v uložení id uzlu, na kterém agent začne svou činnost do belief base. Příslušný kód v jazyce ALLL je následující:

```
&(1)$(t,(i))+(S,&1)
```

Dále je nezbytné upravit plán back pro návrat agenta:

```
(back,(^(addblf)&(3)$(t,(b,f))$(m,(&3,s))*(S,&1)$(k)^(nb)))
```

Agentní platforma WSageNt v současné době neobsahuje podporu multihop směrovacích protokolů. Je proto nezbytné upozornit na to, že námi navržený agent je pomocí nových služeb platformy schopen projít celou síť i bez těchto směrovacích mechanismů. Je tak tedy možné doručit např. nějaká data konkrétnímu uzlu v sítí, který není v přímé viditelnosti.

7.5 Rozšíření objevování sousedů

V předchozí podkapitole jsme se zabývali popisem agenta, který projde bezdrátovou síť využitím pachových stop a služeb s nimi pracujících. Tento agent využívá služby platformy objevování sousedů, která do belief base uloží nalezené uzly a sílu signálu mezi nimi. S těmito informacemi je v průběhu činnosti agenta manipulováno za účelem získání id okolního uzlu. Při přesunu agenta na jiný uzel jsou tyto informace smazány.

Pro sledování pohybu uzlů budeme potřebovat v agentovi uložit informace o síle signálu mezi uzly v síti. Za tímto účelem bude rozšířena původní služba objevování sousedů, která neobsahovala žádný parametr, o parametr "m".

Rozšířená služba objevování sousedů bude pracovat obdobným způsobem jako služba původní. Nejvýznamnějším rozdílem bude jiný formát n-tice, který se bude ukládat do belief base:

```
(M, actId, ndId, strenght)
```

- M - identifikační řetězec rozšířeného objevování sousedů,
- actId - id uzlu, na kterém se agent nachází,
- ndId - id nalezeného sousedního uzlu,
- strenght - síla signálů mezi uzly actId a ndId.

Rozšířené službě objevování sousedů odpovídá v jazyce ALLL kód:

```
$(n,m)
```

[2] V basestation je nahrán tento program proto, aby bylo možné přeposílat požadavky z webového rozhraní do sítě a naopak.

7.6 Agent pro sledování pohybu uzlů

Pro potřeby sledování pohybu uzlů byl navržen agent, který využívá nových agentních prvků pro průchod bezdrátovou sítí. Agent projde celou bezdrátovou síť a při své zpáteční cestě do belief base uloží pomocí modifikované služby objevování sousedů (viz. 7.5) informace o síle signálu mezi jednotlivými uzly sítě. Agent pracuje obdobným způsobem jako agent pro průchod sítí z kapitoly 7.4 - činnost agenta je také ukončena po jeho návratu do basestation. Tento agent nicméně nemusí být odeslán do sítě z basestation, ale může začít svou činnost na nějakém uzlu v síti. V tomto případě je ale nezbytné provést jeho drobnou modifikaci, obdobně jako v kapitole 7.4.

Agent pro sledování pohybu uzlů bude pracovat následujícím způsobem:

1. `zjistím, které sousední uzly jsou v mém rádiovém dosahu a tyto informace uložím do belief base jako n-tice tvaru (NB,id,strenght)`
2. `vyjmu první n-tici tvaru (NB,_,_) z belief base a uložím ji do registru r1`
3. `jestliže nebyl vyjmut žádný prvek, tj. belief base neobsahuje informace o nalezených sousedech, pokračuji bodem 8, jinak pokračuji bodem 4`
4. `z registru r1 získám hodnotu id sousedního uzlu, kterou uložím do registru r2`
5. `dotáži se uzlu, jehož id je dáno hodnotou r2, zda jsem na něm již byl`
6. `jestliže jsem na uzlu byl, vrátím se na bod 2, jinak pokračuji bodem 8`
7. `smažu z belief base zbylé n-tice s nalezenými sousedy, přesunu se na uzel daný hodnotou r2 a pokračuji bodem 1`
8. `uložím do belief base pomocí rozšířené služby objevování sousedů informace o síle signálu mezi mnou a okolními uzly`
9. `zjistím id uzlu z kterého jsem přišel poprvé, přesunu se na tento uzel a pokračuji bodem 1`

Kód navrženého agenta v jazyce ALLL je zobrazen v tabulce 7.5.

Část	**Obsah**

PlanBase	`(nb,($(l,(r,1))$(w,(500))$(l,(r,0))$(n)^(cycle)#^(back)))`
	`(addblf,($(l,(r))$(n,m)))`
	`(cycle,(&(1)*(NB,_,_)&(2)$(f,&1)-&2^(testnmove)^(cycle)))`
	`(testnmove,(^(getid)$(t,(q,&2))-(NB,_,_)$(m,(&2,s))^(nb)))`
	`(getid,(&(1)$(r,&2)&(2)$(f,&1)))`
	`(back,(^(addblf)&(3)$(t,(b,f))$(m,(&3,s))^(nb)))`

Plan	`^(nb)`

Tabulka 7.5: Agent pro sledování pohybu uzlů.

Agent po návratu na uzel, z kterého byl vyslán, obsahuje v belief base n-tice s informacemi o nalezených uzlech a síle signálu.

7.7 Použití systémových komponent TinyOs

Platforma Iris obsahuje paměť RAM a externí flash paměť pro manipulaci s daty. Paměť RAM je určena pro rychlé čtení a zápis dat. Její nevýhodou ovšem je, že její velikost není

příliš velká (8KB). Jako další možná nevýhoda této paměti by mohla být ztráta dat po vypnutí uzlu.

Druhou možností pro práci s daty je použít vestavěnou externí paměť typu flash. Ta je určena pro permanentní uložení dat v uzlu, kde data zůstanou uložena i po vypnutí nebo dokonce po přeprogramování uzlu. Paměť je rozdělena do pojmenovaných spojitých oblastí, jejichž velikost definuje v aplikaci programátor. Tato paměť nabízí oproti RAM větší kapacitu (512KB) pro uložení dat, nicméně je naproti tomu pomalejší.

Zvážením výše uvedených vlastností obou pamětí bylo rozhodnuto pro uložení pachových stop využít paměť typu flash. K této paměti je možné přistupovat několika přístupy, které byly uvedeny v kapitole 3.4.2. Z důvodu efektivního přístupu do paměti byl zvolen blokový přístup.

Pro manipulaci s obsahem paketů využijeme rozhraní `Packet` a `AMPacket`, která umožňují zjistit například zdrojovou nebo cílovou adresu a další důležité informace.

Časovač bude využit v podpoře služeb s pachovými stopami u služby dotazování. V případě, že dotazující uzel neobdrží ve zvoleném intervalu odpověď od dotazovaného, vygeneruje a odešle dotaz znovu.

Komponentu Boot použijeme pro spuštění transceiveru na uzlu po jeho zapnutí a pro nastavení některých počátečních hodnot.

7.8 Rozšíření webového rozhraní

V současné době již Control panel podporuje odeslání agenta do bezdrátové senzorové sítě. Agent je zde definován třídou Agent.java v balíčku cz.vutbr.wsagent.entity. Bude proto nezbytné tuto třídu obohatit o id a třídu agenta a rozšířit příslušnou jsp stránku WSAgentSend.jsp o tato pole ve formuláři pro odeslání agenta do sítě.

Požadavek na odeslání agenta je přes komunikační protokol přenesen do programu BSComm, který agenta odešle na požadovaný uzel. Tento protokol bude nutné pro předání těchto hodnot mezi Control panelem a BSComem také rozšířit.

V této fázi bychom již měli mít platformu rozšířenou o služby pracující s pachovými stopami a bude tedy možné z webového rozhraní odeslat agenta, který zanechá na uzlu pachovou stopu.

Webové rozhraní obsahuje modul pro zobrazování rozmístění uzlu. Ovšem zjišťování sil signálu mezi uzly se děje na základě odeslání agenta z basestation s kódem `$(n)(m,(1))` všem uzlům v dosahu basestation. Znamená to tedy, že rozsah nalezených uzlů je silně omezen dosahem basestation. Naproti tomu vyslání našeho agenta umožní získat informace o síle signálu od všech uzlů sítě.

Pro potřebu sledování pohybu uzlů, přidáme do Control panelu nové třídy pro vyslání a zpracování požadavku na sledování pohybu uzlů. Dále pak budou vytvořeny příslušné webové stránky pro vyslání požadavku a vizualizaci sledování pohybu.

Problematika zobrazení pozic uzlů v bezdrátové síti je, tak aby výsledné rozmístění uzlů odpovídalo realitě, značně rozsáhlá a složitá. Pro naši demonstraci pohybu jsme se proto rozhodli pro generování rozmístění uzlů využít výstupu diplomové práce M. Gábora, který se touto problematikou již zabýval. Control panel v současné době obsahuje třídy pro výpočet a vizualizaci rozmístění uzlů. Některé z těchto tříd budou použity jako základ pro zobrazení sledování pohybu.

Kapitola 8

Implementační rozšíření platformy

V předchozí kapitole jsme se zabývali návrhem služeb platformy pro práci s pachovými stopami, vytvořením agenta pro průchod celou sítí a návrhem rozšíření Control panelu a BSCommu pro podporu pachových stop agentů.

V této části práce bude popsána implementace funkcí a služeb pro podporu práce s pachovými stopami a také bude nastíněno jakým způsobem služby pracují. Mimoto bude dále také demonstrována funkčnost služeb pomocí simulační knihovny TOSSIM, která byla v průběhu implementace využívána pro simulaci. U některých funkcí poukážeme na některé komplikace při ověřování funkčnosti na reálných hardwarových uzlech.

8.1 Datová struktura pro popis pachové stopy

Za účelem práce s pachovými stopami byla v souladu s návrhem v souboru AMAgent.h[1] vytvořena následující datová struktura, která definuje jakých typů mohou jednotlivé položky záznamu pachové stopy nabývat.

```
typedef struct ag_footprint {
uint16_t from;
uint16_t ag_id;
uint16_t ag_class;
uint16_t path_counter;
} ag_footprint;
```

Jak již bylo předesláno, informace týkající se pachové stopy agenta nejsou uloženy v agentovi, ale nacházejí se v platformě. Z tohoto důvodu byly v tomtéž hlavičkovém souboru vytvořeny globální proměnné `ag_id, ag_class, path_counter`, ve kterých jsou tyto informace o agentovy uloženy.

Informace o příchozím agentovi jsou také zaznamenávány do flash paměti na uzlu. Služby pracující s pachovými stopami využívají těchto dat z paměti pro zjištění požadované informace. Data jsou v paměti uložena do té doby, než je uzel vypnut. Pro práci s flash pamětí je již v adresáři platformy vytvořen soubor volumes-at45db.xml, který obsahuje vyhrazený blok paměti určený pro zaznamenání naměřených dat. Do tohoto souboru byl přidán další blok pro uložení informací o pachových stopách agentů nazvaný AGFOOTPRINT. Velikost tohoto bloku byla zvolena 16384 bytů.

[1] Tento hlavičkový soubor je určen pro definici konstant a datových typů, které platforma ke své činnosti využívá.

8.2 Rozhraní a modul pro práci s pachovými stopami

Pro práci s pachovými stopami bylo vytvořeno rozhraní *FootprintI*, které je umístěno v souboru *FootprintI.nc* v adresáři Interface. Tento adresář je určen pro rozhraní platformy, která jsou přístupná[2] také interpretu. Rozhraní vypadá následovně:

```
interface FootprintI
{
  command void getNodeId();
  command void addFootprint(ag_footprint data);
  command void readFootprint(ag_footprint *data, uint16_t pos);
  command void whereAgentComeFirstFrom(uint16_t agentId, uint16_t agentClass);
  command void whereAgentComeLastFrom(uint16_t agentId, uint16_t agentClass);
  command void didAgentHere(uint16_t agentId, uint16_t agentClass, am_addr_t addr);
  command void whereAgentComeFrom(uint16_t *data, uint16_t *count,
                                  uint16_t agentId);
}
```

Všechny služby a funkce pro podporu práce s pachovými stopami agentů jsou umístěny v modulu FootprintM.nc v adresáři Platforma. Tento modul implementuje výše uvedené rozhraní.

Každá funkce z rozhraní po svém zavolání nastaví do proměnné `operationType`, která je v rámci modulu globální, typ operace. Tento modul využívá rozhraní pro práci s flash pamětí, což je split-phase operace. Nastavení typu operace je důležité proto, abychom po dokončení operace práce s pamětí (čtení, zápis) dokázali určit, která funkce (služba) danou operaci započala. Typ operace může být následující:

- FP_ADD - přidání pachové stopy do paměti,
- FP_READ - čtení pachové stopy z paměti,
- FP_DID_AG_HERE - služba dotazování,
- FP_WHERE_AG_COME_FIRST_FROM - služba backtracking,
- FP_WHERE_AG_COME_LAST_FROM - služba backtracking,
- FP_READALL - čtení všech záznamů,
- FP_QUERYRESPONSE - služba dotazování,
- FP_NODEID - služba identifikace uzlu.

Nastavení operace je důležité především z hlediska událostí *readDone*, *writeDone*, *eraseDone*. Je to proto, abychom dokázali rozlišit, která služba se provádí v okamžiku dokončení práce s pamětí.

[2]Tato společná rozhraní umožňují části interpretu při interpretaci agenta volat příslušné služby platformy, jsou tedy pro interpret a platformu společné.

8.3 Uložení a čtení pachové stopy do/z paměti

Za uložení záznamu o agentovi zodpovídá funkce addFootprint(ag_footprint data). Protože funkce pracuje s pamětí, je proto asynchronní. Nejprve se tedy pozastaví činnost interpretu a poté se nastaví typ operace na FP_ADD a také se zvýší počítadlo pachových stop, které jsou již uloženy ve flash paměti. Poté se pomocné pole naplní hodnotami z parametru funkce addFootprint (strukturou data) a zavolá se funkce rozhraní BlockWrite pro uložení dat do paměti `call FootprintWrite.write(pos_agFootprint, wbuf, sizeof(wbuf))`.

Zkopírování hodnot ze struktury data do pomocného pole wbuf by se mohlo zdát jako zbytečné. Praktické zkušenosti na hardwarových uzlech ovšem ukázaly, že je to nezbytné pro správnou funkčnost operace write a read.

Po provedení této funkce je modulem BlockRead poslán signál modulu FootprintM s výsledkem provedení operace. Tento signál se zpracovává v události `FootprintWrite.writeDone( storage_addr_t x, void* buf, storage_len_t y, error_t result)`.

Funkce `readFootprint` je určena pro přečtení dat z externí flash paměti. Činnost této funkce je také asynchronní a proto je po jejím zavolání pozastavena činnost interpretu. Funkce zašle prostředníctvím funkce `read` z rozhraní `blockRead` požadavek na přečtení dat na pozici `pos`. Po přečtení dat rozhraní blockRead zašle signál o dokončení operace modulu FootprintM, který je zpracován v události `readDone`.

Tato funkce je využívána službami platformy pracující s pachovými stopami pro čtení záznamů z paměti.

8.4 Odeslání, příjem agenta

S ukládáním pachových stop souvisí odeslání a příjem agenta v platformě.

8.4.1 Odeslání agenta

Za odeslání agenta na jiný uzel je v platformě zodpovědný modul SendM.nc. Platforma umožňuje poslání zprávy pro běžnou komunikaci nebo zprávy agentní. V případě odeslání agentní zprávy se nejprve odešle agentní hlavička a poté vlastní data agenta.

Jak již bylo řečeno, platforma v globálních proměnných obsahuje aktuální pachovou stopu agenta, který se na ní nachází. Tyto informace o agentovi se vloží na 19-24 byte do hlavičky při přesunu agenta na jiný uzel.

8.4.2 Příjem agenta

Pro příjem zpráv nebo agentů je v platformě určen modul ReceiveM. Do tohoto modulu bylo přidáno získání informací o příchozím agentovi z hlavičky, které mají být zaznamenány do externí flash paměti.

```
ag_from = call AMPacketA.source(bufPtr);
ag_id = rcm->msg[19]*256+rcm->msg[20];
ag_class = rcm->msg[21]*256+rcm->msg[22];
path_counter = rcm->msg[23]*256+rcm->msg[24] + 1;
```

Po přijetí[3] celého agenta dojde k zavolání operace addFootprint s informacemi o přijatém agentovi. Je důležité zde upozornit, že hodnota path_counter se průchodem agenta sítí

[3] V případě větších agentů se nemusí podařit celého agenta přijmout napoprvé a proto musí být některé jeho části poslány znovu.

zvyšuje. Této hodnoty je možné využít např. pro trasování agenta, jak procházel sítí.

8.5 Služba identifikace uzlu

Při žádosti agenta o službu identifikace uzlu volá interpret příkaz `getNodeId()`. Služba je jako jedna z mála synchronní a proto po jejím zavolání nedojde k pozastavení interpretu. Funkce uloží do pomocné proměnné `final_data` hodnotu id uzlu, na kterém se agent nachází. Tuto hodnotu získá ze systémové konstanty TinyOs `TOS_NODE_ID`. Dále nastaví proměnnou `have_data` na hodnotu TRUE.

Platforma umožňuje přidávat zprávy nebo data agentovi pouze na začátku každého cyklu. O vhodné době informuje hlavní smyčka pro přidání dat modul spuštěním příkazu iFootprintI.ableToAdd z rozhraní iFootprintI. V případě, že máme získanou hodnotu, kterou chceme přidat, můžeme ji uvnitř příkazu ableToAdd vložit do příslušného pole agenta. U této služby je zjištěné Id uzlu uloženo do aktuálního vybraného registru.

Pro přidání získaných hodnot z modulu FootprintM do aktivního registru agenta bylo do modulu TableC přídána funkce

```
call TableI.addFootprintDataToReg(final_data);
```

Takovýmto způsobem je i ostatních služeb pracujících s pachovými stopami prováděno uložení získané informace do agenta.

8.6 Služba backtracking

Služba backtracking je implementována ve dvou níže uvedených variantách. Obě varianty mají společné to, že jsou asynchronní.

8.6.1 Z kterého uzlu přišel agent poprvé

Při žádosti na službu backtracking s parametrem "f", která zjistí odkud přišel agent poprvé, volá interpret funkci `whereAgentComeFirstFrom`. Argumenty funkce jsou id a třída hledaného agenta.

Protože je služba asynchronní, je pozastavena hlavní řídící smyčka. Poté dojde k nastavení proměnné `recordIndex` na hodnotu 0. Tato hodnota specifikuje, od které pozice se mají data v paměti začít prohledávat. V tomto případě je to tedy od prvního uloženého záznamu ve flash paměti. Dále je nastaven typ operace na hodnotu `FP_WHERE_AG_COME_FIRST_FROM` a nastaven příznak, že se má číst další záznam. Do pomocné struktury `queryData` jsou uloženy hledané hodnoty id a třídy agenta z argumentu funkce. Následně je zavolána funkce readRecord, která přes systémové rozhraní vyšle požadavek na přečtení záznamu na pozici recordIndex.

Po dokončení čtení dat je systémovou komponentou BlockRead zaslán signál modulu FootprintM, který je zpracován událostí readDone. V této události je zjištěn typ nastavené operace dané proměnnou `operationType` a poté provedeno porovnání, zda přečtená data odpovídají hledaným kritériím.

Pokud byl nalezen záznam s požadovaným id a třídou agenta, je nastaven příznak, že další záznamy již nemají být dále čteny. Také je nastaven příznak, indikující že existují data která je možné přidat agentovi. Dále je vložena získaná hodnota, odkud agent přišel, do proměnné final_data. Následně dojde k přidání získané informace do aktivního registru a ke spuštění interpretu.

V případě, že přečtený záznam neobsahuje hledané informace, je zvýšena proměnná recordIndex o hodnotu 1 a je zavolána funkce pro přečtení záznamu. To má za následek opět poslání signálu a obsluhu událostí readDone.

Pokud jsou přečteny všechny záznamy a nedošlo k nalezení hledaného záznamu, je řízení předáno interpretu.

8.6.2 Z kterého uzlu přišel agent naposledy

Tato varianta služby backtracking je specifikována parametrem "l". Při žádosti o tuto službu interpret zavolá funkci `whereAgentComeLastFrom`.

Tato varianta backtrackingu pracuje obdobným způsobem jako předchozí varianta. Je nastaven typ operace na `FP_WHERE_AG_COME_LAST_FROM`. Nejvýznamnějším rozdílem je to, že záznamy jsou z paměti prohledávány od konce, tedy od naposledy vloženého. Modul si při ukládaní pachových stop ukládá informaci o jejich celkovém počtu. Tato hodnota je použita pro inicializaci proměnné recordIndex na poslední záznam v paměti.

8.7 Služba dotazování

Pro potřeby této služby byly v souboru PlatformaC vytvořeny komponenty AMReceiverC, AMSenderC pro příjem a odesílání zpráv mezi uzly. Jako parametr při jejich vytváření byl uveden typ zprávy[4] AM_FOOTPRINT_MSG, na který má modul reagovat.

Při požadavku na tuto službu interpret volá funkci `didAgentHere`, která má parametry id, třídu agenta a id uzlu, na který se má poslat dotaz. Id uzlu může být zadáno buď jako číselná hodnota nebo registr.

Tato služba je asynchronní a je proto tedy dočasně pozastavena činnost interpretu. Argumenty funkce jsou uloženy do pomocné struktury pro pozdější použití a je zavolána funkce `sendQuery`. Ta vytvoří zprávu, do které vloží identifikátor daný hodnotou FOOTPRINT_QUERY, hodnoty id a třídy hledaného agenta. Tuto zprávu přes systémové rozhraní AMSend funkcí `send` odešle na uzel, kterého se služba dotazuje. Dále funkce spustí časovač, který spustí událost `MilliTimer.fired` za 1000 ms., přičemž je možné tuto hodnotu považovat za timeout. V případě, že dotazující uzel obdrží před spuštěním této události odpověď od dotazovaného uzlu, je spuštění této události zrušeno. Pokud ovšem odpověď neobdrží, je v této události dotaz odeslán znovu. V případě, že je překročen maximální počet odeslaných dotazů, je časovač zastaven, následuje smazání podplánu a řízení je navráceno interpretu.

Modul FootprintM obsahuje událost `receive` rozhraní Receive. V této události jsou zpracovávány přijaté zprávy, které jsou určeny pro tento modul. Zda se jedná o přijetí dotazu nebo odpovědi je zjištěno z prvního bytu dat zprávy. V tomto bytu může být hodnota FOOTPRINT_QUERY nebo FOOTPRINT_QUERY_RESPONSE.

Na straně dotazovaného uzlu tedy po přijetí zprávy dojde, na základě prvního bytu dat, k určení, že se jedná o dotaz zda agent na daném uzlu již byl či nikoli. V návaznosti na to se pozastaví činnost interpretu a je nastaven typ operace na `FP_DID_AG_HERE`. Mimoto jsou do pomocných proměnných uložena přijatá data, tedy id uzlu z kterého přišel dotaz, id a třída hledaného agenta.

Poté jsou postupně čteny záznamy z flash paměti a v případě nalezení záznamu dle id a třídy hledaného agenta je dotazujícímu uzlu odeslána funkcí `echo` odpověď. Tato

[4] V současné době platforma rozlišuje zprávy na zprávy obsahující agentní kód, potvrzovací, pro běžnou komunikaci, pro hledání sousedů a pro pachové stopy. Každý typ zprávy je obsluhován příslušným modulem.

funkce vytvoří zprávu, kde v prvním bytu dat je hodnota dána výčtovým typem FOOTPRINT_QUERY_RESPONSE a v dalším bytu následuje hodnota 1.

V případě, že byly přečteny všechny záznamy a nedošlo ke shodě s hledanými hodnotami, je dotazujícímu uzlu poslána funkcí echo zpráva. Ta obsahuje stejnou hodnotu na prvním bytu, ovšem druhý byte zprávy je roven 0.

Na straně dotazujícího uzlu je tato zpráva přijata a na základě prvního bytu je poté zjištěno, že se jedná o odpověď na dotaz. Pokud je hodnota druhého bytu zprávy rovna 1, znamená to že agent na daném uzlu již byl. Pomocí funkce `pop_plan` rozhraní interpretu StackI je smazán aktuální podplán a poté je provedeno spuštění interpretu.

8.8 Rozšíření služby objevování sousedů

Do rozhraní NeighbourDiscoveryI byl přidán příkaz `command void discoverForMovement()`, jehož implementace se nachází v modulu NeighbourDiscoveryM.

Do hlavičkového souboru AMagent.h, který obsahuje definice datových typů, byl přidán nový výčtový typ rozlišující varianty služby objevování sousedů.

```
ND_DISCOVER=1 - původní objevování sousedů, (NB, id, strenght)
ND_DISCOVERM=2 - objevování sousedů s parametrem "m", (M, id, strenght)
```

Původní kód z funkce, který byl určen pro objevování sousedů, byl vložen do nové interní funkce `neighbourDiscover`. Po zavolání interpretem některé z variant služby je nastavena proměnná `discoverType`, do které je vložena hodnota výčtového typu v závislosti na požadované variantě služby objevování sousedů a zavolána funkce neighbourDiscover.

Při přidávání n-tice funkcí ableToAdd do belief base je zjištěna hodnota discoverType. Na základě této hodnoty je do belief base uložena n-tice ve formátu (NB, id, strenght) pro původní službu nebo (M, id, strenght) pro službu s parametrem ”m”.

8.9 Simulace služeb pro práci s pachovými stopami

V průběhu implementace byla prováděna simulace služeb pomocí knihovny TOSSIM. Možnost využití simulace implementovaných služeb výrazným způsobem urychlila vývoji, protože práce se skutečnými zařízeními je časově náročná.

Modul FootprintM posílá TOSSIMu informace prostřednictvím komunikačního rozhraní dbg. Kanál pro komunikaci s tímto modulem byl označen *TEST-FootprintM*. Při simulaci je tedy nezbytné tento kanál importovat.

Modul PlatformaC.nc obsahuje vytvoření a napojení komponent na moduly platformy a interpretu. V práci Ing. Horáčka[11] byl vytvořen jednoduchý modul VirtualBlockM pro simulaci flash pamětí, který byl v případě simulace použit. Vytvoření 16KB virtuálního bloku paměti bylo provedeno.

```
components new VirtualBlockM(16384) as StorageFootprint;
components new TimerMilliC() as casovac_StorageFootprintM;
StorageFootprint.MilliTimer -> casovac_StorageFootprintM;
```

V případě kompilace programu pro reálná zařízení je již použita komponenta pro reálnou práci s pamětí.

```
components new BlockStorageC(VOLUME_AGFOOTPRINT) as StorageFootprint;
```

Pro simulaci pachových stop byl vytvořen testovací script v jazyce Python pojmenovaný *Footprint.py*. Tento script vytvoří 3 uzly, přičemž každý má k ostatním radiový dosah a může tak s nimi komunikovat. V průběhu implementace byly služby a agenti, které je využívají, simulovány i na větších sítích obsahující desítky uzlů.

Pro simulaci agentního kódu bylo využito části inicializující interpret. Ta se nachází v souboru AgentC.nc v adresáři Interpret. V události *booted* nejprve dojde k naplnění příslušných tabulek a registrů agentním kódem a poté k zavolání platformy příkazem ControlI.run. Po zavolání tohoto příkazu dojde k vykonávání daného kódu.

Do události ControlI.booted() byl přidáván kód pro inicializaci agentů v simulaci pro práci s pachovými stopami, tedy například:

```
#ifdef TOSSIM
  if (TOS_NODE_ID == 2){
    ag_class=2;
    ag_id=2;
    strcpy(c,"$(l,(r,1))$(m,(1,s))$(l,(g,1))&(3)$(t,(b,f))$(m,(&3,s))
              $(l,(g,1))");
    call Zasobnik.push_str(c, (uint16_t)strlen(c));
  };
#endif
```

Pro ilustraci simulace je zde uveden částečný výstup simulace pro výše uvedeného agenta, inicializovaného v události booted.

```
DEBUG (2): RED ON
DEBUG (2): sending head
DEBUG (1): CHECKSUM OK = 2463 [expected = 2463]
DEBUG (1): FootprintM: Saving footprint data to Flash from:2, ID:2 class 2
DEBUG (2): Send done
DEBUG (1): FootprintM: write done.
DEBUG (1): GREEN ON
DEBUG (1): Where agent come first command, counter 1.
DEBUG (1): FootprintM: whereAgentComeFirstFrom podarilo se nalezt zaznam.
DEBUG (1): TEST-FootprintM: pripraven pro pridani dat do aktivniho registru.
DEBUG (1): sending head
DEBUG (2): CHECKSUM OK = 677 [expected = 677]
DEBUG (2): FootprintM: Saving footprint data to Flash from:1, ID:2 class 2
DEBUG (1): Send done
DEBUG (2): FootprintM: write done.
DEBUG (2): GREEN ON
```

Agent začne svou činnost na uzlu číslo 2, kde rozsvítí červenou diodu. Poté se přesune na uzel číslo 1, kde platforma do paměti zaznamená jeho pachovou stopu a rozsvítí zelenou diodu. Poté agent pomocí služby backtracking zjistí odkud přišel poprvé a na tento uzel se přesune. Služba uloží do registru hodnotu 1. Po přesunu na tento uzel je platformou zaznamenána pachová stopa a je rozsvícena zelená dioda.

Kapitola 9

Implementace - webové rozhraní, BSComm

Tato kapitola se zabývá popisem rozšíření implementace aplikace BScomm a webového rozhraní. Tato rozšíření navazují na podporu pachových stop implementovaných v platformě a interpretu z předchozí kapitoly. Nejprve bude ukázáno, jakým způsobem byl rozšířen program BSComm a komunikační protokol mezi ním a webovým rozhraním, aby bylo možné předávat informace o pachových stopách agentů. Poté se zaměříme na popis implementace rozšíření Control panelu o přidání informací týkajících se pachových stop. V neposlední řadě bude popsáno rozšíření webového rozhraní o detekci pohybů uzlů, kde bude využito navrženého agenta pro sledování pohybu uzlů.

9.1 BSComm

Agent je v aplikaci BSComm reprezentován třídou *Agent* v balíčku cz.vutbr.wsagent.entity. Do této třídy byly proto přidány proměnné `agentId` a `classId`, sloužící pro rozlišení agentů. Dále byly ve třídě vytvořeny příslušné metody a funkce pro přidané proměnné. Poté došlo k namapování nových proměnných v souboru agent.hbm.xml, aby bylo možné je ukládat do databáze pomocí frameworku Hibernate.

Jak již bylo zmíněno, BSComm vytváří komunikační most mezi webovým rozhraním a basestation. Při požadavku na odeslání zprávy nebo agenta z webového rozhraní do bezdrátové sítě vytvoří webové rozhraní příslušný typ zprávy[1], který odešle využitím komunikačního protokolu (viz. 6.5.3) do BSCommu. Ten zprávu zpracuje a na základě získaných hodnot vytvoří novou zprávu, kterou přes sériové rozhraní pošle do basestation a poté informuje webové rozhraní o výsledku operace.

Pro odeslání agenta z webového rozhraní do sítě je určena agentní zpráva. Ta ovšem neobsahuje informace o id a o třídě agenta. Formát agentní zprávy byl proto rozšířen o pole `agent_id` , specifikující id agenta a `class_id` určující třídu agenta.

Rozšířený formát agentní zprávy je tedy následující:

```
typ_zprávy ODDĚLOVAČ adresa_uzlu ODDĚLOVAČ planBase ODDĚLOVAČ plan ODDĚLOVAČ
belief base ODDĚLOVAČ input base ODDĚLOVAČ agent_id ODDĚLOVAČ class_id
```

Definice agentní zprávy je umístěna ve třídě *AgentMessage* v balíčku cz.vutbr.wsagent-.comm. Do této třídy byly přidány proměnné agentId, classId, pathCounter a doplněny

[1] V současné době je možné posílat zprávu obsahující agenta nebo datovou zprávu.

příslušné metody set a funkce get pro jednotlivé proměnné.

BSComm po obdržení agentní zprávy získá funkcí *getAgentParsed*, která je umístěna ve třídě *MessageParser*, hodnoty jednotlivých polí ze zprávy, tedy včetně agent_id a class_id. Poté vytvoří instanci třídy AgentMessage s obsahem získaných dat.

Pro odeslání agenta do sítě je určena metoda `sendAgent` z třídy cz.vutbr.wsagent.comm, která z hodnot agentní zprávy vytvoří novou zprávu. Tato zpráva má stejný formát jako zprávy pro posílání mezi uzly sítě. Do jednotlivých polí zprávy se vloží hodnoty popisující agenta.

Hodnoty pro popis pachové stopy agenta jsou do zprávy přidány do těchto polí:

```
header[19] = (short) (agentId/256);
header[20] = (short) (agentId%256);
header[21] = (short) (classId/256);
header[22] = (short) (classId%256);
header[23] = (short) (pathCounter/256);
header[24] = (short) (pathCounter%256);
```

Zpráva je poté odeslána přes sériové rozhraní do basestation. V basestation musí být nahrán program Basestation z TinyOs, který slouží pro přeposílání paketů ze sériového portu přes transceiver do sítě a také naopak.

9.2 Control panel

Implementace rozšíření webového rozhraní proběhla ve dvou krocích. V první fázi došlo k přidání podpory pro práci s pachovými stopami agentů, tedy např. odeslání agenta, který obsahoval informace o své pachové stopě. Druhá fáze se věnovala sledování pohybu uzlů na základě využití pachových stop a navrženého agenta pro sledování pohybu.

9.2.1 Podpora práce s pachovými stopami

Agent je ve webovém rozhraní reprezentován třídou *Agent* z balíčku cz.vutbr.wsagent.entity. Tato třída byla doplněna o proměnné agentId, classId a pathCounter a příslušné metody set a funkce get pro proměnné. Dále bylo nutné provést namapování proměnných v souboru agent.hbm.xml, aby bylo možné je ukládat do databáze pomocí frameworku Hibernate.

Následně došlo k rozšíření jsp stránky *WSAgentForm.jsp*, která obsahuje formulář s informacemi o agentovi. Tento formulář je použit jednak pro vytvoření agenta, který se uloží do databáze, tak i pro jeho editaci.

Informace o uložených agentech se zobrazují ve stránce *WSSavedAgent.jsp*. Do této stránky bylo také přidáno zobrazování informací o id a třídě agenta.

Odeslání agenta do BSCommu se provádí pomocí metody `send` z třídy WSAgentSend. Tato metoda nejprve pomocí funkce constructAgentMessage z třídy *MessageConstructor* vytvoří textový řetězec, který splňuje rozšířený formát agentní zprávy a poté zprávu přes soket odešle BSCommu.

9.2.2 Sledování pohybu uzlů

Pro sledování pohybu uzlů bylo využito navrženého agenta z kapitoly 7.5. Agent využitím služeb pracujících s pachovými stopami projde celou bezdrátovou síť a získá hodnoty síly

signálu mezi uzly. Na základě těchto hodnot je pak proveden výpočet a vykreslení rozmístění uzlů. Následně je ve zvoleném intervalu agent znovu odeslán do sítě a činnost se opakuje.

Obrázek 9.1 zobrazuje rozšíření Control panelu o sledování pohybu. Na obrázku jsou zobrazeny čtyři uzly bezdrátové senzorové sítě. Každý uzel obsahuje informaci o svém id a informaci o vypočtených souřadnicích v pixelech v rámci obrázku. Vzájemná viditelnost mezi uzly je znázorněna úsečkami.

Odeslání požadavku

Požadavek na odeslání agenta do bezdrátové senzorové sítě je realizován spuštěním akce `WSN_RequestMovement`, která odkazuje na třídu *WSNRequestMovement*. Třída nejprve zjistí, zda se v databázi nachází agent pro sledovaní pohybu, který je identifikován jménem "WTAgent"v tabulce agent. Pro vybrání agenta z databáze na základě hodnoty jména (name), byla přetížena funkce *read* z třídy AgentDAO.

Pokud agent v databázi ještě neexistuje, je následně vytvořen a uložen do databáze. Jméno agenta je nastaveno na "WTAgent", třída a id je nastaveno na hodnotu 1. Obsah báze plánů a plán, který se má začít vykonávat, je stejný jako u navrženého agenta pro sledování pohybu 7.5. Při každém vykonání této třídy, tedy odeslání agenta do sítě, je hodnota id agenta zvýšena o 1. Tato činnost je velice důležitá z hlediska služby platformy pachových stop dotazování. Pokud bychom do sítě vyslali agenta se stejnými hodnotami id a třídy, došlo by k tomu, že by služba dotazování pro jednotlivé uzly v síti skončila s výsledkem, že agent na uzlech již byl. To by mělo za následek přesun agenta zpět do basestation a tím ukončení jeho činnosti, aniž by prošel síť.

Následně je pomocí třídy MessageConstructor vytvořena rozšířená agentní zpráva. Ta je přes soket odeslána aplikaci BSComm a poté dojde k přesměrování na akci WSN_Movement, která spustí vykonání třídy WSN_Movement.

Zobrazení rozmístění uzlů

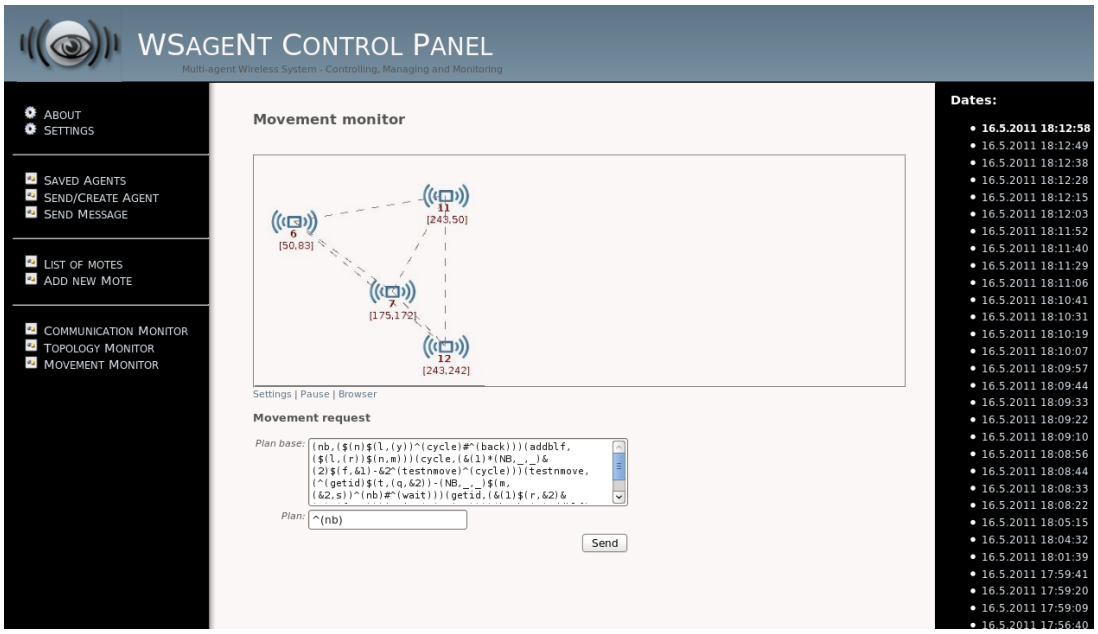
Pro vygenerování a zobrazení rozmístění uzlů bylo využito některých již existujících tříd z balíčku cz.vutbr.wsagent.util. Těmito třídami jsou: RSSIObject - obsahuje informace o síle signálu mezi dvěma uzly, RSSIRecord - obsahuje seznam objektů typu RSSIObject, TopologyMath - pro výpočet polohy uzlů, TopologyDrawer[2] - pro vykreslení topologie[10].

Pro potřebu zpracování získaných dat ze sítě byla vytvořena třída *WSNMovement*, jejíž spuštění je iniciováno akcí `WSN_Movement`. Při vykonávání této třídy jsou nejprve získána data z databáze, která agent nasbíral průchodem sítí. Tato data jsou ve formátu v jakém je uložila rozšířená služba objevování sousedů 7.5 agentovi. Jednotlivé záznamy o síle signálu mezi uzly v síti je nutné oddělit. Pro tuto činnost byla vytvořena třída *MovementParser* v balíčku cz.vutbr.wsagent.util. Ta na základě vyhledání vzoru v jakém mají být záznamy uloženy, záznamy vyhledá a vytvoří seznam objektů RSSIObject, který je typu `RSSIRecord`. Na základě těchto objektů jsou pa vypočteny pomocí třídy *TopologyMath* souřadnice uzlů a úseček mezi nimi v pixelech pro umístění prvků do obrázku. Vykreslení těchto prvků zajišťuje třída *MovementDrawer*, která prvky vykreslí do obrázku. Tento obrázek je poté zobrazen na jsp stránce WSNMovement.jsp.

Pro zobrazení aktuální polohy rozmístění uzlů a odeslání požadavku na odeslání agenta do sítě byla vytvořena stránka WSNMovement.jsp. Stránka také obsahuje menu, které ob-

[2]Námi vytvořená třída MovementDrawer se od této liší je v některých detailech, proto i tuto třídu zde uvádíme.

sahuje položky Settings, Stop a Browse. Zvolením položky Settings je vyvolána stránka obsahující nastavení pro sledování pohybu. Stop pozastaví odesílání požadavku na odeslání agenta do sítě a Browse přesměruje stránku do režimu prohlížení historie. Stránka dále obsahuje formulář pro odeslání požadavku pro odeslání agenta do sítě.



Obrázek 9.1: Movement monitor.

Prohlížení historie

Prohlížení historie záznamů sledování pohybu je spuštěno akcí `WSN_MovementBrowse`, která začne vykonávat třídu *WSNMovement*.

Zobrazení prohlížení historie zajišťuje stránka *WSNMovemenBrowser.jsp*. Na této stránce je zobrazen obrázek s rozmístěním uzlů a v pravém sloupci jsou zobrazeny časy, kdy se agent vrátil do basestation s daty ze sítě. Po kliknutí na datum je nastaven parametr `paramDate` na hodnotu id záznamu ze sloupce a zavolána akce WSN_MovementBrowse. To má za následek spuštění třídy WSNMovemnt, která na základě záznamu vygeneruje rozmístění uzlů. Druhým způsobem jak se pohybovat v historii je použít odkazy *next* a *previous*, které vyvolají akci WSNMovementBrowse.

Nastavení

Pro nastavení parametrů sledování pohybu uzlů byly přidány do třídy *Setting* z balíčku cz.vutbr.wsagent.entity dvě proměnné. Proměnná *timePeriodMovement* udává jak často bude webové rozhraní posílat agenta pro sledování pohybu do sítě. Proměnná *reqmovementReqMoteAddress* určuje id uzlu, na který se odešle agent pro sledování pohybu. Dále byla doplněna konfigurace pro Hibernate v souboru *setting.hbm.xml*. Za zobrazení nastavení zodpovídá stránka Settings.jsp, která obsahuje formulář s hodnotami. Formulář byl doplněn o dvě nová pole pro možnost nastavení uvedených hodnot pro sledování pohybu.

Kapitola 10

Testování

V této kapitole se zaměříme na testování implementace provedeného rozšíření pro sledování pohybu uzlů.

10.1 Čas průchodu agenta sítí

Aby bylo možné měřit čas, za který je agent pro sledování pohybu uzlů (viz. kapitola 7.5) schopen projít celou síť, je nezbytné provést drobné modifikace. Ty spočívají v tom, že agent po odeslání z basestation na uzel rozsvítí na uzlu zelenou led diodu a po svém návratu zpět ji zase zhasne. Vlastní měření bude tedy spočívat v měření času mezi rozsvícením a zhasnutím diody. Někdy může nastat situace, že některý z uzlů je mimo rádiový dosah ostatních uzlů v síti. Z tohoto důvodu budeme kontrolovat počet zobrazených uzlů ve webovém rozhraní. Pro každé měření byla vytvořena bezdrátová senzorová síť s počtem uzlů od 2 do 6. Testování s větším počtem uzlů nebylo v době testování možné, protože uzly nebyly k dispozici. Měření času potřebného pro průchod agenta sítí bylo provedeno desetkrát pro každý počet uzlů. Vzdálenost mezi sousedními uzly byla v rozsahu 1-7 m. Vypočtený průměr z naměřených hodnot je možné nalézt v tabulce 10.1.

Počet uzlů	**Čas [ms]**
2	976
3	3021
4	5239
5	7834
6	1209

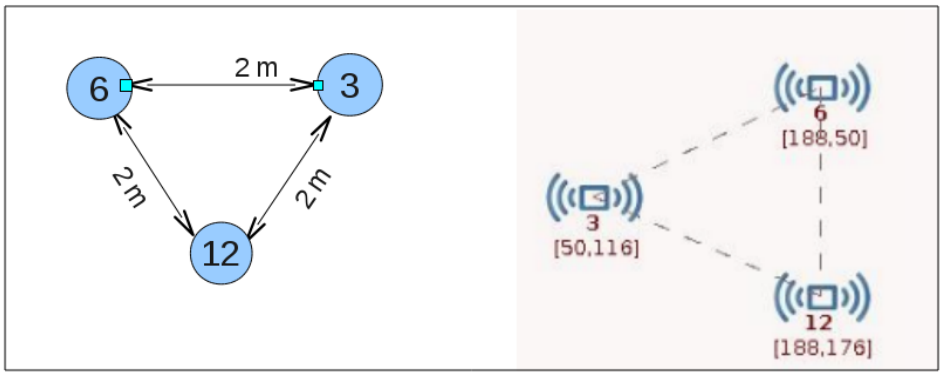
Tabulka 10.1: Průměr z naměřených hodnot času, za který agent projde síť.

10.2 Sledování pohybu uzlů

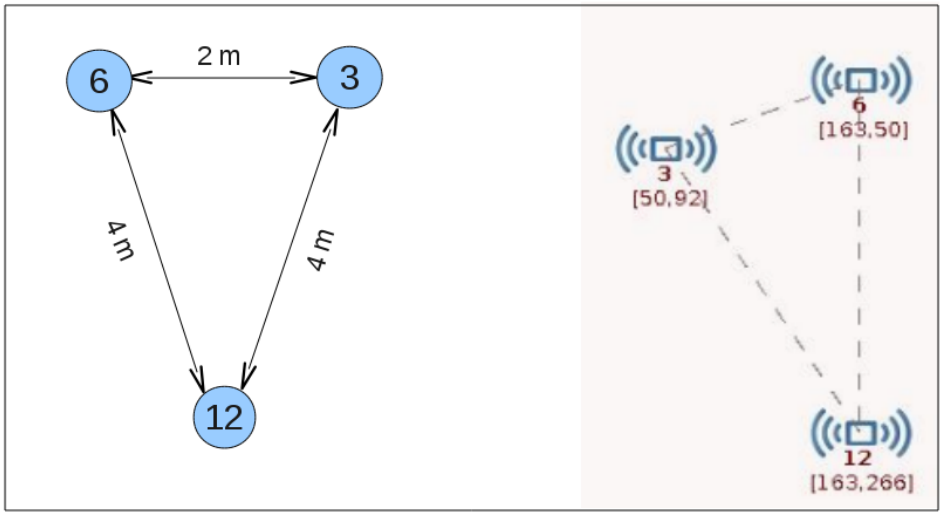
Pro testování sledování pohybu uzlů jsme postupně vytvořili dvě bezdrátové senzorové sítě, které obsahovaly 3 a 5 uzlů. Id uzlů byla zvolena na hodnoty 3, 6, 12 pro první síť. Druhá síť obsahovala uzly s hodnotami id 3, 6, 7, 11 a 12. Uzly byly pro testování označeny samolepkami s čísly odpovídajícími jejich id. Testování bylo prováděno v místnosti o rozměrech 7 x 6 m, kde byly všechny uzly v přímé viditelnosti. Uvedené obrázky z webového rozhraní

obsahují vizualizované rozmístění uzlů. Každý uzel obsahuje informaci o svém id a v hranatých závorkách jsou uvedeny souřadnice uzlu v pixelech. Tyto souřadnice se vztahují k pozici, na které se uzel nachází v rámci obrázku a nemají se skutečným umístěním žádnou souvislost. Dále je zde důležité uvést, že délka úseček mezi uzly je vypočtena na základě hodnot sil signálu mezi uzly v síti, které agent průchodem sítí získal.

Postup testování pro síť s třemi uzly byl následující. Nejprve byly uzly v síti rozmístěny tak, že mezi sebou vytvořily rovnostranný trojúhelník s délkou strany 2 m a poté bylo spuštěno sledování pohybu uzlů z Control panelu. Do této sítě byl odesílán agent na uzel 6 a perioda odesílání byla zvolena na hodnotu č. 5 s. Rozmístění uzlů zachycuje obrázek 10.1, kde je v levé části schematicky zobrazeno rozmístění uzlů ve skutečnosti a na pravé straně výsledek ve webovém rozhraní. Poté byl uzel s číslem 12 přemístěn na předem označené místo. Toto místo bylo umístěno 4 m od uzlů č. 3 a 6. Tento stav je možné vidět na obrázku 10.2.



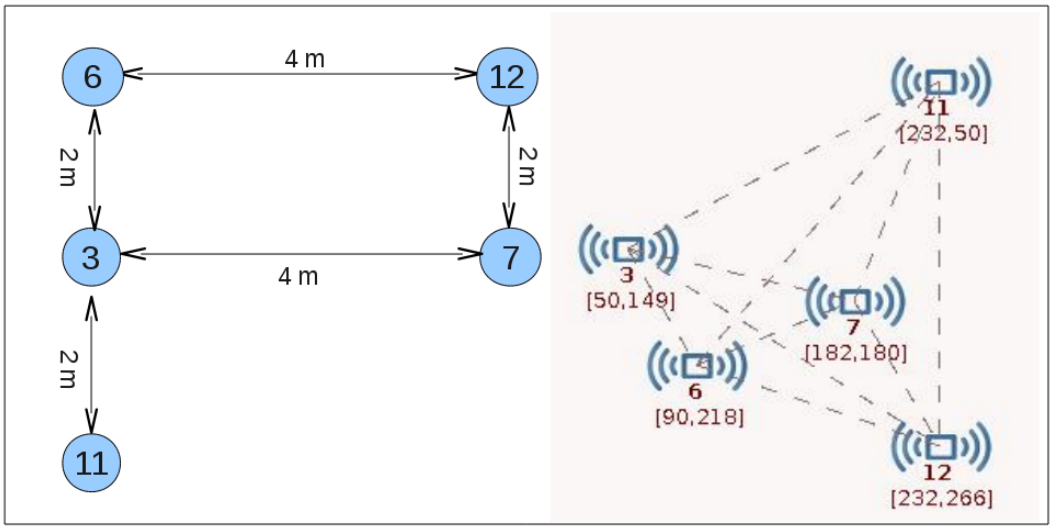
Obrázek 10.1: Skutečné a vizualizované rozmístění 3 uzlů v síti.



Obrázek 10.2: Skutečné a vizualizované rozmístění 3 uzlů v síti po přesunu uzlu č. 3.

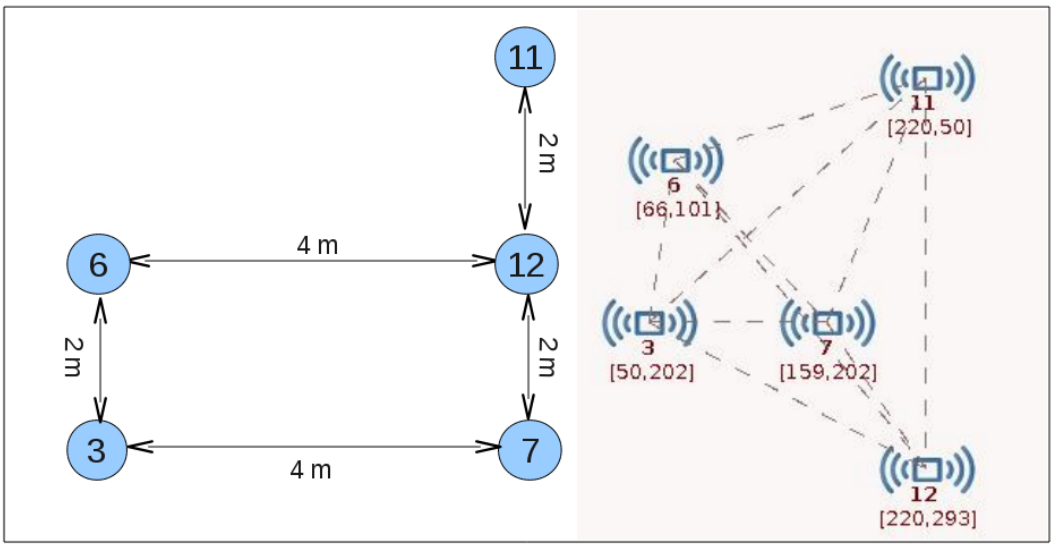
Postup testování druhé sítě s pěti uzly byl obdobný jako u předešlé sítě. Rozdílné bylo rozmístění uzlů a také to, že agent byl periodicky každých 10 s odesílán na uzel č. 3. Skutečné

rozmístění uzlů je zobrazeno v levé části obrázku 10.3. V pravé části téhož obrázku je možné nalézt výsledek z Control panelu.



Obrázek 10.3: Skutečné a vizualizované rozmístění 5 uzlů.

Obrázek 10.4 obsahuje v levé části schematické znázornění rozmístění uzlů. V pravé části obrázku je pak zobrazen výsledek z webového rozhraní.



Obrázek 10.4: Skutečné a vizualizované rozmístění 5 uzlů po přesunu uzlu č.11.

10.3 Diskuze nad dosaženými výsledky

Sledování pohybu pro síť se třemi uzly podávalo v průběhu testování poměrně dobré výsledky. Na obrázku 10.1 je vizualizovaná síť ve vertikálním směru zrcadlově převrácena,

nicméně uzly stále tvořily rovnostranný trojúhelník. Následné přemístění uzlu č. 12 na určené místo mělo požadovanou odezvu na zobrazenou síť ve webovém rozhraní. Čas pro periodické odesílání agenta do sítě byl nastaven na malou hodnotu a v Control panelu bylo proto velmi dobře vidět jak se uzel opožděně pohyboval.

Druhým případem testování bylo vytvoření sítě, která byla tvořena pěti uzly. Vizualizace počátečního stavu sítě (obr. 10.3) před přesunem uzlu č. 11 byla v horizontálním směru zrcadlově převrácena a uzel č. 11 by měl být spíše nad uzlem č. 3, než nad uzlem č. 7. Pozice uzlů č. 3, 6, 7, 12 mezi sebou vytvářely ve skutečnosti pomyslný obdelník, avšak webové rozhraní zobrazilo lichoběžník. Po přemístění uzlu č. 11 na pozici dle obrázku 10.4 již výsledek v Control panelu neodpovídal skutečnému umístění uzlů.

Z provedeného testování nabýváme poznatku, že dobrých výsledků bylo dosaženo při malém počtu uzlů v síti. Pro větší síť, v našem případě o pěti uzlech, bylo zobrazení rozmístění uzlů ve webovém rozhraní značně nepřesné. V této síti, při pomalejším přemístění skutečného uzlu č. 11, bylo možné v Control panelu sledovat opožděnou změnu polohy uzlu (pohyb uzlu), nicméně rozmístění uzlů neodpovídalo skutečnosti.

Chyba, která se vyskytla v druhém případě testování, by mohla být spůsobena nepřesným změřením síly signálu mezi uzly v síti, protože při použití uzlů v místnosti může dojít k různým odrazům. Hlavní příčinu ovšem spatřujeme ve výpočtu umístění uzlů pro jejich vizualizaci. Použitá metoda výpočtu[10] na základě analytické geometrie (průniku kružnic) neposkytovala dobré výsledky pro více uzlů v síti. Pro výpočet rozmístění uzlů ve webovém rozhraní by tedy bylo výhodnější použít jiný algoritmus nebo postup.

Kapitola 11

Závěr

Tato práce se věnovala použití agentů v bezdrátových senzorových sítích. Nejprve byly čtenáři předloženy informace týkající se vzniku této technologie. Poté byl popsán operační systém TinyOS, který poskytuje hardwarovou abstrakci pro tyto sítě. Pozornost byla také zaměřena na to, jakým způsobem je možné nahrát aplikaci do zařízení a také jak vytvořit její simulační model pro simulaci pomocí simulátoru TOSSIM. Dále bylo nastíněno několik základních konceptů programovacího jazyka NesC. Text se také zabýval projektem WSageNt, jeho částmi a možnostmi, které nabízí pro použití agentů v prostředí bezdrátových senzorových sítí. Dále byly popsány některé vlastnosti hardwarové platformy Iris, která je v projektu WSageNt podporována.

V další části se text nejprve zabýval měřením hodnot signálu mezi dvěma Iris uzly. Využitím služby objevování sousedů byly získány hodnoty sil signálů pro dvě různá prostředí. Hodnoty získané z platformy jsou z rozsahu 0-255 a neodpovídají reálným hodnotám RSSI v dBm. Proto byla pro potřeby konverze vytvořena převodová tabulka na hodnoty v jednotkách dBm.

Práce se dále také věnovala návrhu nových agentních prvků pro obohacení projektu WSageNt. Byly navrženy a implementovány služby pro podporu práce s pachovými stopami agentů. Tyto služby dávají agentům větší možnosti pro jejich činnost. Důsledkem toho je např. možnost zjištění, na kterých uzlech agent již byl. Funkčnost těchto služeb byla nejprve otestována pomocí simulační knihovny TOSSIM, která významným způsobem urychlila vývoj nových služeb. Poté došlo k následnému testování služeb na reálných zařízeních.

Těchto nových prvků bylo využito pro vytvoření agenta, který projde celou bezdrátovou senzorovou síť z a získá informace o síle signálu k okolním sousedům. Tento agent byl poté použit pro monitorování pohybu uzlů ve webovém rozhraní.

V současné době již došlo k využití pachových stop a některých služeb s nimi pracujících, a to v diplomové práci kolegy M. Houště, který je použil pro zjišťování informací, jak se agent pohyboval v síti.

Další pokračování této práce by mohlo směřovat k vytvoření přesnějšího a robustnějšího výpočtu souřadnic vizualizovaných uzlů ve webovém rozhraní.

Závěrem lze říci, že využití pachových stop u inteligentních agentů v bezdrátových senzorových sítích rozšiřuje pole použití těchto agentů.

Literatura

[1] Boot sequence. [online], [cit. 2010-04-10].
URL <http://docs.tinyos.net/index.php/Boot_Sequence>

[2] Getting Started with TinyOS. [online], [cit. 2010-03-3].
URL <http://docs.tinyos.net/index.php/Getting_Started_with_TinyOS>

[3] Novinky výstavy Sensor. [online], [cit. 2010-12-20].
URL <hhttp://www.automatizace.cz/article.php?a=1847>

[4] Storage. [online], [cit. 2011-04-3].
URL <http://docs.tinyos.net/index.php/Storage>

[5] Atmel: Atmel AT86RF230 datasheet. [online], [cit. 2010-4-14].
URL <http://www.atmel.com/dyn/resources/prod_documents/doc5131.pdf>

[6] Crossbow: Iris datasheet. [online], [cit. 2010-12-14].
URL <www.dinesgroup.org/projects/images/pdf_files/iris_datasheet.pdf>

[7] Crossbow: MTS420/400CC datasheet. [online], [cit. 2010-12-17].
URL
<http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=174%3Amts400_420>

[8] Culler, D.; Polastre, J.; Jiang, X.: Perpetual Environmentally Powered Sensor Networks. *Computer Science Department*, April 2005: s. 463 – 468.

[9] Gay, D.; Levis, P.; von Behren, R.; aj.: The nesC Language: A Holistic Approach to Networked Embedded Systems. In *In Proceedings of Programming Language Design and Implementation (PLDI)*, June 2003.

[10] Gábor, M.: *Webové rozhraní pro sledování provozu v bezdrátových sítích.* Diplomová práce, FIT VUT v Brně, Brno, 2010.

[11] Horáček, J.: *Platforma pro mobilní agenty v bezdrátových senzorových sítích.* Diplomová práce, FIT VUT v Brně, Brno, 2009.

[12] Horáček, J.; Zbořil, F.: WSageNt: A Case Study. In *Proceedings of CSE 2010 International Scientific Conference on Computer Science and Engineering*, TU v Košiciach, 2010, ISBN 978-80-8086-164-3, s. 258–264.

[13] Janoušek, V.: Inteligentní agenti. 2010, podklady k přednáškám kurzu SIN.

[14] Lee, H.; Cerpa, A.; Levis, P.: Improving wireless simulation through noise modeling. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, New York, NY, USA: ACM, 2007, ISBN 978-1-59593-638-X, s. 21–30.

[15] Levis, P.: TEP:111 - message t. [online], [cit. 2010-10-10].
URL `<http://www.tinyos.net/tinyos-2.x/doc/pdf/tep111.pdf>`

[16] Levis, P.: TinyOS 2.0 Overview. [online], [cit. 2008-04-14].
URL `<http://www.tinyos.net/tinyos-2.x/doc/html/overview.html>`

[17] Levis, P.: Tinyos programming. [online], [cit. 2010-10-14].
URL `<http://csl.stanford.edu/~pal/pubs/tinyos-programming-1-0.pdf>`

[18] Levis, P.; Lee, N.; Welsh, M.; aj.: TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *In Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[19] Pike, J.: The sound surveillance system (SOSUS). [online], [cit. 2011-04-30].
URL `<http://www.fas.org/irp/program/collect/sosus.htm>`

[20] Spáčil, P.: *Mobilní agenti v bezdrátových senzorových sítích.* Bakalářská práce, FIT VUT v Brně, Brno, 2009.

[21] Wikipedia: Sensor node. [online], [cit. 2010-11-10].
URL `<http://en.wikipedia.org/wiki/Sensor_node>`

[22] Zbořil, F.: Simulation for Wireless Sensor Networks with Intelligent Nodes. In *In 10th Inter-national Conference on Computer Modelling and Simulation*, Cambridge, GB: IEEE Computer Society, 2008, ISBN 0-7695-3114-8, str. 6.

[23] Zbořil, F.: Úvod do agentních a multiagentních systémů. 2010, podklady k přednáškám kurzu AGS.

Příloha A

Obsah CD

1. Zdrojové texty aplikace BSComm
2. Zdrojové texty aplikace ControlPanel
3. Zdrojové texty platformy projektu WSageNt
4. Zdrojové texty programu Basestation
5. Text diplomové práce
6. Plakát