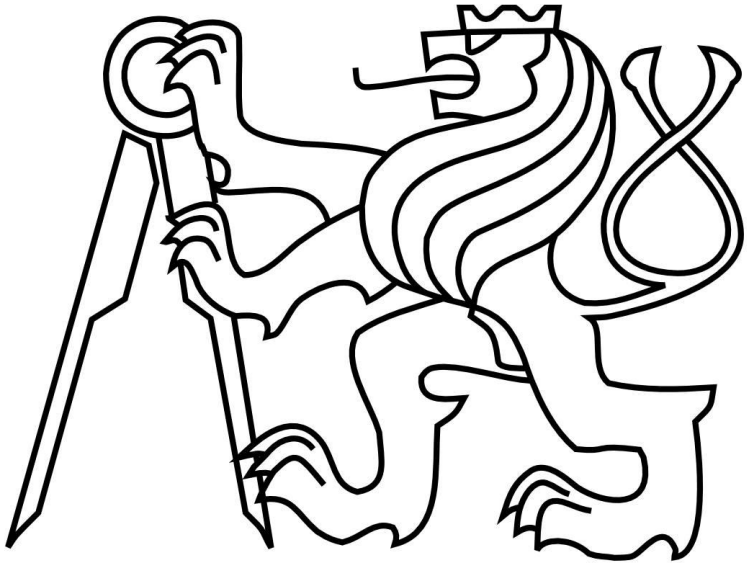
Bakalářská práce



Ovládání kolony vozidel ze systému Android pomocí protokolu Zigbee

České Vysoké Učení Technické

Katedra Řídící Techniky

Alexander Dubeň

2015

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 10. května 2015 .....

Alexander Dubeň

Abstrakt

Náplní této bakalářské práce je vytvoření aplikace v systému Android, díky níž bude možné ovládat testovací platformu, sloužící k experimentálnímu ověřování metod distribuovaného řízení kolon aut, za pomocí tabletu. Tato platforma je vyvíjena na katedře řídící techniky fakulty elektrotechnické na ČVUT. Je založena na větším počtu modelů aut, která jsou vybavena procesory, spolu s dalšími nezbytnými periferiemi (senzory atd.). I přestože jsou auta schopna se řídit autonomně, pro parametrizaci celého testu je nutný externí přístup ke všem modelům z centrálního počítače, přičemž implementace této centrální monitorovací jednotky je hlavním cílem práce.

Ke komunikaci mezi modely je využíván rádiový protokol Zigbee. Jelikož tablety v současné době nepodporují nativně tento způsob komunikace, je tato funkčnost řešena pomocí USB periferie připojené k tabletu.

Práce se tedy nejprve zaměřuje na zprovoznění komunikace mezi zařízením Android a komunikačním čipem pomocí USB. Následně pojednává o konfiguraci tohoto komunikačního čipu tak, aby byla vytvořena funkční Zigbee rádiová síť. Poslední částí práce je tvorba aplikační vrstvy a také uživatelského rozhraní v systému Android.

This bachelor thesis aims to create an Android application for tablet which will enable control over testing platform which is used for experimental testing of methods for distributed control of car platoons. The platform is developed at Department of Control Engineering at Faculty of Electrical Engineering at Czech Technical University. It is based on larger amount of models of cars, which are equipped with processors and other peripheries such as sensors etc. Even though the cars are able to control themselves, it is necessary to control whole experiment from central computer. Implementation of this central monitoring unit is the main goal of this work.

Wireless radio protocol Zigbee is used for communication between models. Because present-day tablets do not support natively this type of communication it was enabled using USB periphery connected to the tablet.

First of all this thesis concentrates on deployment of USB communication between the tablet and the network processor. Secondly it aims on dealing with configuration of network processor so a wireless Zigbee network is established. Finally the thesis deals with development of application layer and also user interface for Android based devices.

Klíčová slova

Universal Serial Bus, USB CDC class, Serial communication, Serial port, USB OTG, Android, Android USB, USB Host API, Android UI, Zigbee, Texas Instruments CC2531

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Alexander D u b e ň

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Ovládání kolony vozidel ze systému Android pomocí protokolu Zigbee

Pokyny pro vypracování:

1. Zprovozněte komunikaci se Zigbee modulem v tabletu se systémem Android.
2. Implementujte jednoduché komunikační API.
3. Připravte grafické rozhraní pro ovládání kolony vozidel na autodráze.

Seznam odborné literatury:

[1] Drew Gislason: Zigbee Wireless Networking, Newnes 2008.
[2] Grant Allen: Android 4 - Průvodce programováním mobilních aplikací, Computer Press Brno,2013.
[3] Pavel Herout: Učebnice jazyka Java, KOPP, 2010.

Vedoucí bakalářské práce: Ing. Ivo Herman

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 21. 1. 2015

OBSAH

1 ÚVOD

1.1 MOTIVACE

V současné době zažívá automatizace dopravy velký rozkvět a není pochyb o tom, že autonomně počítači řízená doprava je otázkou několika let. Vždyť již dnes automobily umožňují automatické parkování, jízdu v zácpě či nouzové zastavení auta při hrozící kolizi. Například Google již úspěšně testuje své autonomní prototypy. [17]

Všechny tyto příklady spadají do kategorie autonomního řízení, které ovšem probíhá pouze v rámci jednoho vozidla. Druhou cestou jak dosáhnout autonomie, je vytvoření distribuovaného systému, kdy zamýšlené automatizace je dosaženo komunikací mezi agenty v systému. Jednou z aplikací této myšlenky je tzv. Car Platooning.

Při Car Platooningu se auta pohybují po silnicích ve skupinách (platoons), většinou i uspořádána v koloně. Pokud by se veškerá doprava pohybovala v takto uspořádaných formacích, došlo by k obrovskému zvýšení efektivity využití dopravních komunikací a hlavně ke zvýšení bezpečnosti.

I přesto, že ve vědeckých kruzích již existuje celá řada návrhů na obdobné řídicí systémy, je zde zásadní problém s převedením teorie do praxe a to, že většina výzkumných týmů si prostě nemůže dovolit testovat na koloně reálných automobilů z finančních i legislativních důvodů. Platforma vyvíjená na katedře řídící techniky elektrotechnické fakulty by mohla nabídnout levnou alternativu, jak tyto teorie otestovat v praxi.

Na této platformě v současné době pracuje pod vedením Ing. Ivo Hermana a Ing. Dana Martince několik studentů. Mým úkolem v rámci této bakalářské práce je implementace spolehlivější rádiové komunikace mezi auty, než je využívána doposud a také vytvoření uživatelského rozhraní pro zjednodušení a zpříjemnění ovládání celého experimentu. Díky kvalitnímu rozhraní systému Android, slouží tato práce také k prezentačním účelům.

Studentka Anastasia Vlasova současně pracuje na komplementárním projektu, který se zabývá komunikací ze strany modelů aut.

1.2 CÍL PRÁCE

Cílem projektu je vytvoření aplikace v systému Android spojující grafické uživatelské rozhraní s aplikační vrstvou k ovládání experimentu.

Jednotlivé modely aut jsou vybaveny procesory ARM Cortex-M4, dále tříosým akcelerometrem, gyroskopem a magnetometrem. K měření rychlosti slouží inkrementální senzor a k měření vzdálenosti optický IR senzor. K bezdrátové komunikaci byl původně využíván čip RF24L01 od firmy Nordic Semiconductors. Pro účely tohoto projektu by ovšem byla do budoucna vhodnější standardizovaná rádiová komunikace. Jako nejvhodnější byl zvolen standard Zigbee. RF24L01 již slouží pouze k ladícím účelům. [18]

Jednotlivé modely aut byly vybaveny komunikačním čipem od firmy Texas Instruments CC2530. Tento čip je již vybaven hotovou implementací Zigbee komunikačního protokolu, který je obsažen přímo v jeho firmwaru. Komunikace mezi tímto čipem a procesorem v modelu auta je náplní komplementárního vedlejšího projektu.

V rámci tohoto projektu je využíván USB dongle obsahující Zigbee čip TI CC2531. Pro komunikaci s tímto čipem je tedy možné využít jakékoliv zařízení, které je schopné pracovat jako USB master. V tomto případě to je tablet s operačním systémem Android.

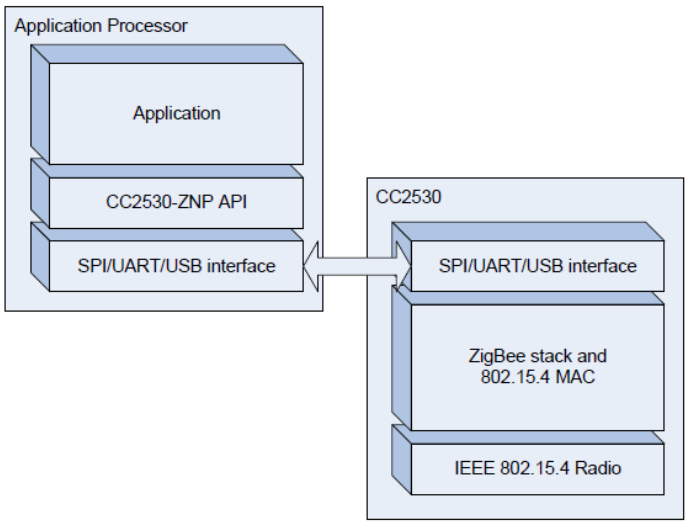
V celém projektu je nutné vyřešit 3 hlavní části. Nejprve zprovoznit komunikaci přes USB mezi tabletem a komunikačním čipem. S tímto krokem získáme přístup k jeho firmwaru, tedy hlavně k jeho Zigbee protokolu, který následně nakonfigurujeme tak, abychom vytvořili funkční Zigbee rádiovou síť. Po této síti následně budeme přenášet požadovaná data, jako příkazy k ovládání modelů aut, či hodnoty z jejich senzorů. Nakonec bude v systému Android vytvořeno vhodné uživatelské rozhraní, pomocí kterého bude uživatel schopen komfortně ovládat celou platformu.

Zprovoznění USB komunikace jsem vyřešil v rámci předmětu Individuální projekt předcházejícího bakalářské práci.

Tato zpráva se zabývá obecným popisem a postupným řešením dílčích problémů.

2 NÁVRH IMPLEMENTACE

Zjednodušené schéma celého projektu je vidět na následujícím grafu (Obrázek 1). V našem případě je Application Processor tablet se systémem Android. Toto zařízení bylo zvoleno z důvodu jednoduché manipulace a vhodného uživatelského rozhraní sloužícímu i k demonstraci, které je zajištěno dotykovým displejem.



Obrázek 1 - Struktura projektu [5]

Aniž bychom museli zasáhnout do hardwaru tabletu, musíme využít ke komunikaci se Zigbee čipem sběrnici USB. Všechna zařízení typu tablet sice obsahují USB rozhraní, ale často jsou navržena tak, aby pracovala jako koncová zařízení a nikoliv hostitel (anglicky host). Tato funkčnost je nutná u téměř všech zařízení se systémem Android, jelikož umožnuje nabíjení a rychlou správu uživatelské paměti z osobního počítače.

Tento fakt ovšem není překážkou, jelikož USB standard definuje zařízení typu USB OTG (On The Go), které umožňuje duální režim komunikace, jak v režimu USB host tak v režimu koncového zařízení. Ne všechna zařízení založena na systému Android tento formát podporují, proto je nutné zvolit zařízení, které hardwarově i softwarově umožňuje USB OTG.

Příkazy přenášené přes USB zpracovává komunikační čip (v našem případě USB dongle CC2531 viz 2.1), přičemž o správu všech funkčních vrstev (USB, Zigbee, MAC, rádio) se již stará jeho firmware.

Jelikož je USB dongle jako jediný v přímém spojení s řídícím počítačem celé platformy, je nejvhodnější, aby pracoval v rámci Zigbee sítě jako tzv. koordinátor (viz 4.2.1). Mimo zprostředkování přenosu dat mezi řídící aplikací a modely aut tedy bude mít na starost vytvoření a správu rádiové sítě.

Modely aut se budou chovat jako koncová zařízení a jejich úkolem bude přijímat konfigurační příkazy a také zpět řídící aplikaci odesílat data.

Řídící aplikace by měla být podle požadavků interpretovat přijímaná data a vhodně na ně reagovat. Hlavní funkcí by měla být možnost parametrizace celého experimentu, tedy

například nastavení regulátorů v modelech a také referencí. Další důležitou funkcí je vizualizace dat ze senzorů a jejich uložení k pozdějšímu zpracování. Jelikož je aplikace zamýšlena i k prezentaci celého projektu pro laickou a odbornou veřejnost, měla by být graficky líbivá.

2.1 POUŽITÝ HARDWARE

Jádrem projektu je dongle osazený čipem CC2531 ([Obrázek 2]). Hlavní výhodou tohoto čipu je možnost komunikace přes USB s řídícím počítačem. Čip nicméně disponuje širokou škálou dalších periferií. [3]

Obrázek 2 - CC2531 USB Dongle [2]

Nejdůležitějším prvkem je RF vysílač/přijímač pracující v pásmu 2,4 GHz. Tento radiový vysílač je kompatibilní se standardem IEEE 802.15.4, který mimo jiné využívá také protokol Zigbee. Vyznačuje se velice dobrou citlivostí a také odolností vůči rušení.

Aby byla umožněna USB komunikace, je zde přítomen vestavěný USB kontrolér. Pracuje v souladu se standardem USB 2.0 s přenosovou rychlostí až 12 Mbps. Spravuje 5 endpointů (+ control endpoint 0), které mohou pracovat ve všech dostupných režimech (bulk, interrupt, isochronous). K ukládání paketů využívá 1 KB SRAM FIFO paměť. USB controller komunikuje s hlavním čipem formou přerušení a také má na starosti komunikaci při enumeraci zařízení na sběrnici USB. Dále je čip vybaven nezbytnou pamětí, a to 256 KB programovatelnou flash a také 8 KB RAM. Obsahuje čtyři timery s obecnou funkčností, jeden sleep timer a watchdog timer. Mimo to je zde hardwarově podporována přístupová metoda ke sdílenému médiu CSMA/CA, teplotní senzor a dvanácti bitový ADC převodník s osmi kanály a nastavitelným rozlišením. Nakonec jsou zde také dvě rozhraní USART s podporou několika sériových protokolů.

Při vývoji byly použity dva vysílače TI CC2531. Jeden k vytvoření Zigbee sítě a druhý sloužící k simulaci modelů aut.

Tablet, pro který byla aplikace primárně vyvíjena je model Acer Iconia Tab 10. Při testování ovšem aplikace fungovala i na jiných tabletech s 10 palcovým displejem. Aplikace teoreticky může fungovat i na zařízeních s jinou velikostí displeje, ale například na menších zařízeních je velice nepřehledná. Tablet Acer je vybaven procesorem MediaTek MT8127 Quad-core 1.30 GHz, 1 GB RAM paměti, 10 palcovým displejem s rozlišením 1280x800 a 16 GB Flash paměti. Tato hardwarová výbava je dostatečná i pro náročné aplikace a bohatě postačuje našim potřebám.

2.2 POUŽITÝ SOFTWARE

Původní dongle je z výroby vybaven firmwarem, který umožňuje jeho použití jako packet sniffer. Dongle s tímto firmwarem je tak možné využít k testovacím účelům při Zigbee komunikaci. Pro vytvoření funkční rádiové sítě je ovšem nutné použít firmware implementující celý komunikační stack Zigbee protokolu.

Mimo hardware firma Texas Instruments také dodává několik typů již hotových softwarových výbav pro tento typ komunikačního čipu s již vytvořeným plně funkčním komunikačním stackem (Z-Stack [4]). V projektu byl použit firmware ze softwarového balíku Z-Stack Home Automation. Tento balík obsahuje několik verzí Zigbee protokolu vhodných pro několik typů hardwaru a také řadu ukázkových aplikací, sloužících k demonstraci předpokládaného využití této Zigbee implementace.

Do programové paměti komunikačního čipu byla nahrána implementace ZNP (Zigbee Network Processor) přímo určená pro verzi čipu CC2531. Nebyla nutná žádná dodatečná kompilace, jelikož TI dodává k dispozici přímo zkompilované programy v hex formátu (CC2531ZNP-Pro.hex). Firmware kromě přístupu ke kompletnímu Zigbee stacku také umožňuje komunikaci s čipem pomocí USB. Dongle se chová jako standartní USB zařízení třídy CDC/ACM.

K programování Android aplikace bylo využito vývojové prostředí Eclipse se zásuvným modulem pro vývoj Android aplikací s plnou sadou vývojových nástrojů. Celý systém Android je open-source a kdokoliv může používat tyto nástroje, aniž by musel platit licenci. Jediným finančním výdajem je registrační poplatek, pokud vývojář chce své aplikace publikovat na Google Play. Jedním z prvků, které nabízí tato sada nástrojů, který jsem ocenil asi nejvíce je možnost debuggingu aplikace přes Wi-Fi. Tímto způsobem lze aplikaci nejenom krokovat, ale i číst systémové logy a nahrávat aktualizace aplikace do tabletu. Teto způsob debuggingu není příliš obvyklý, jelikož většina vývojářů používá jednodušší způsob pomocí USB kabelu, což ovšem není u této aplikace možné, jelikož jediný USB konektor je obsazen Zigbee vysílačem. I díky tomu a možná i kvůli podcenění průzkumu všech možností jsem tento způsob začal využívat až při hledání chyby při přerušení komunikace v průběhu čtení rámce (viz 3.7.1). Debugging po Wi-Fi následně několikanásobně urychlil vývoj grafického rozhraní. Do příloh k této práci (C) přikládám návod, jak tento způsob debuggingu zprovoznit.

Program simulující modely aut byl pro změnu psán v IDE Netbeans s nainstalovanou knihovnou RXTX. Nevyužil jsem stejné vývojové prostředí jako pro tvorbu aplikace, jelikož osobně preferuji Netbeans, nicméně podpora vývoje pro Android v Netbeans není natolik propracovaná jako v Eclipse.

Pro tvorbu ikon, využitých v aplikaci jsem použil freeware editor Paint.net, s využitím stránky Android Asset Studio ([39]).

3 USB KOMUNIKACE

Universal Serial Bus (USB) je průmyslový standard vyvinutý v devadesátých letech. Definuje kabely, konektory, komunikační protokoly a napájení mezi počítači a elektronickými zařízeními. Byl navržen tak, aby unifikoval a standardizoval připojení počítačových periferií jako například klávesnice, myši, tiskárny, externí paměti a síťové adaptéry včetně jejich napájení. Toto rozhraní postupně nahradilo starší typy jako sériové či paralelní porty.

V současné době je možné využít standard USB verze 3.1, který umožňuje přenášet data až rychlostí 10 Gbit/s. Tato verze nicméně ještě není plně rozšířena skrz celé spektrum technologií využívajících USB a často zařízení pracují pouze s verzí 2.0. To je případ i pro dongle TI CC2531, který podporuje USB verze 2.0 a to pouze režim Full Speed, který umožňuje dosahovat přenosové rychlosti 12 Mbit/s.

Jednou z hlavních výhod USB je podpora tzv. Plug&Play, což je funkčnost zajišťující automatickou detekci připojení a konfiguraci zařízení bez nutnosti instalace speciálních driverů či restartu systému. Přenosová rychlost také není pevně daná a liší se od jednotek Mbit/s po Gbit/s (podle varianty low nebo super speed). USB podporuje různé typy přenosů lišící se například spolehlivostí, velikostí bloků dat či podporou isochronního přenosu. [6]

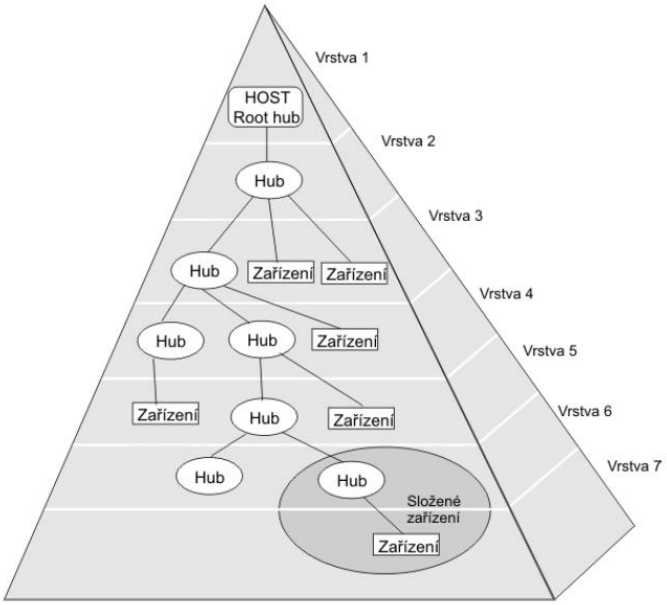
3.1 CHARAKTERISTIKA USB SYSTÉMU

USB systém má asymetrickou stromovou topologii tvořenou třemi typy uzlů: hostitel, hub a koncové zařízení. Komunikace probíhá sériově a polo duplexně (USB 3.0 i plný duplex).

Prvním typem je hostitel (anglicky host). Tento uzel je v systému vždy pouze jeden a tvoří kořen stromu. Typicky je to PC nebo v našem případě tablet. Řídí datové přenosy v celém systému včetně procesu enumerace koncových zařízení a power managementu. Veškerá komunikace je iniciována hostitelem, tedy zařízení nemá možnost samo o sobě hostitele kontaktovat.

Druhý typ uzlu je rozbočovač neboli hub. Tento uzel má možnost detekce připojení zařízení na down-stream portech a také distribuuje datové toky. Povoluje přenosy na jednotlivých portech a řídí datový tok (například konverze z full speed na low speed). Z důvodu omezení zpoždění na sběrnici standard povoluje pouze 5 propojených hubů, přičemž každý hub umožňuje připojení 7 koncových zařízení ([Obrázek 3]). Maximální počet koncových zařízení je tedy 127 na jediného hosta. [8]

Koncová zařízení musí z pohledu USB sběrnice splňovat vlastnosti, které se souhrnně nazývají standartní třída (Standart USB Device Class). Tato vlastnost je součástí tzv. uživatelské vrstvy protokolu a umožňuje podporu celé standartní třídy zařízení jediným ovladačem v operačním systému PC. Standartní třída definuje veškerou funkčnost, kterou zařízení musí splňovat. Jsou to například zařízení typu Mass Storage, Human Interface Device, Printer a v našem případě CDC – Communication Device Class.



Obrázek 3 - Struktura USB systému [7]

3.2 FYZICKÁ VRSTVA

USB využívá symetrické datové vedení (dvojce kabelů D+ a D-). Využívá se kódování NRZI (Non Return To Zero Inverted) a také bit-stuffing sloužící k synchronizaci. Napěťové úrovně se liší podle rychlosti sběrnice. Režim Low-Speed umožňuje rychlost 1,5 Mbit/s a Full-speed 12 Mbit/s. Verze USB 2.0 zavedla režim Hi-Speed umožňující komunikaci rychlostí až 480 Mbit/s. USB 3.0 umožňuje režim Super-Speed s rychlostí 5-10 Gbit/s.

USB 2.0 je sériová linka využívající pouze 1 datový pár, společnou zem a vodič sloužící k napájení. Varianta USB OTG umožňující duální režim zařízení (host i device) obsahuje ještě pátý vodič sloužící k vyjednání role hosta na sběrnici.

USB 3.0 navíc využívá další 2 datové páry sloužící k zajištění plně duplexní komunikace.

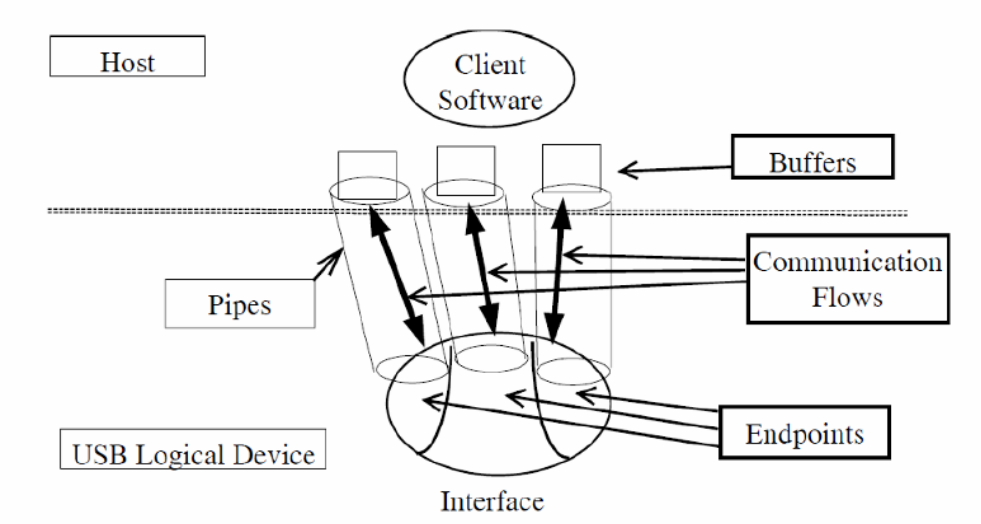
Dříve než proběhne enumerace, host rozpozná nové zařízení na základě napěťových úrovní na jednotlivých vodičích, na něž zařízení připojí pull-up a pull-down rezistory. Tímto způsobem také zařízení sdělí svoji rychlost a je ho možné resetovat. Následně započne proces enumerace (viz 3.4.2).

Maximální odběr každého zařízení při napájení je 500 mA, přičemž reálnou hodnotu odběru, kterou zařízení potřebuje, musí sdělit hostu při enumeraci. [9]

3.3 PRINCIP KOMUNIKACE

Aby se data v zařízení dostala k zamýšleným funkcím, je komunikace založena na tzv. pipes. Pipe (roura) je logický datový kanál spojující hostitele a logickou entitu na koncovém zařízení nazvanou endpoint. USB zařízení může obsahovat až 32 endpointů, ale v praxi jich většinou má mnohem méně. Endpointy jsou seskupeny do tzv. rozhraní a mají předem definovaný typ ([Obrázek 4]). Výčet rozhraní a jednotlivých endpointů

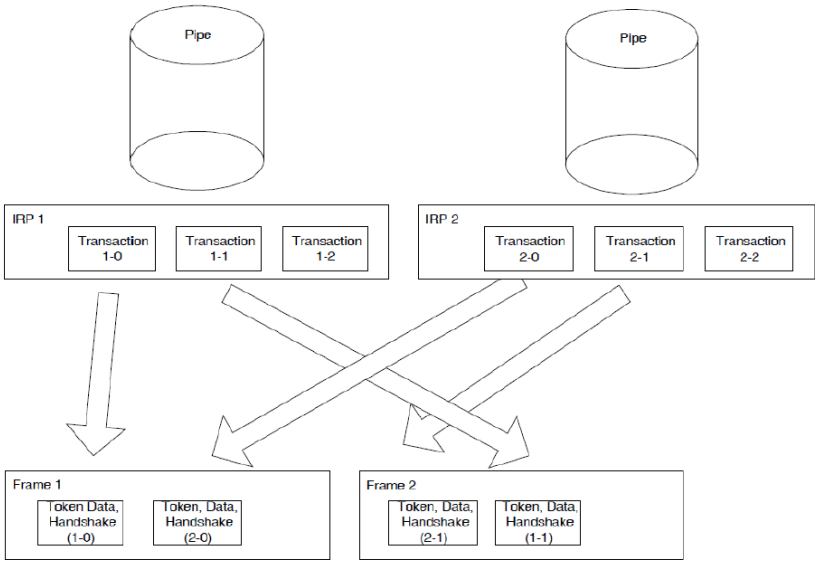
zařízení sdělí hostiteli při enumeraci. Existují dva hlavní typy datových rour (stream pipe a message pipe). [8]



Obrázek 4 - Logické uspořádání komunikačních entit [8]

3.3.1 Datové přenosy

Datové přenosy probíhají cyklicky v rámcích (anglicky frame). Rámec je časový interval, ve kterém jsou posílány jednotlivé pakety. V USB 2.0 je tento frame dlouhý 1 ms popřípadě 125 µs pro rychlý režim. Každý přenos probíhá v tzv. transakcích, které se skládají ze tří paketů (Token, Data, Handshake). V každém rámci může být přeneseno několik paketů, přičemž pakety mohou souviset s různými transakcemi, ba dokonce mohou být určena různým zařízením ([Obrázek 5]). Začátek každého rámce je uvozen paketem SOF (Start Of Frame) čímž se zařízení mohou synchronizovat. [8]



Obrázek 5 - Komunikace formou transakcí v rámcích [8]

3.4 STANDARTNÍ TŘÍDY ZAŘÍZENÍ

USB standard definuje širokou škálu různých typů zařízení s definovanou funkčností. Zařízení splňující předepsané parametry spadá do dané standartní třídy USB a komunikaci se všemi zařízeními tohoto typu zprostředkuje jediný driver v řídícím PC.

Mezi často používané třídy patří například Mass Storage definující zařízení jako flash disky či externí hard disky, Printer (tiskárny a skenery) a HID (klávesnice, myši a jiné). [8]

3.4.1 Deskriptor

Veškeré vlastnosti zařízení jsou shrnuty v datovém objektu zvaném deskriptor, který má v sobě zařízení uloženo. Je to hierarchická struktura tvořena několika vrstvami deskriptorů.

Nejdůležitějším je deskriptor zařízení. Je jediný a obsahuje 14 položek. Identifikuje základní vlastnosti zařízení, podle kterých se chová. Mezi tyto informace patří VID (id výrobce), PID (product id), kód USB třídy zařízení, verze USB a následně odkazy na další deskriptory.

Následuje deskriptor konfigurace (může jich být i více). Tento deskriptor definuje počet logických rozhraní v zařízení a odkazy na jejich deskriptory. Mimo to také obsahuje informace o napájení a spotřebě.

Deskriptory rozhraní říkají, kolik každý interface obsahuje endpointů a také odkazy na jejich deskriptory.

Deskriptory endpointů definují jejich adresy, směr přenosu a také typ přenosu.

Všechny tyto deskriptory obsahují data číselně zakódována. Tento formát ovšem není pro uživatele nejvhodnějším, a proto existují textové deskriptory. Ty obsahují výše uvedené informace o zařízení v čitelné podobě pro člověka. Často má v sobě zařízení uloženo i několik jazykových verzí těchto dat. [8]

Při vývoji se téměř vždy kompletní znalost informací o zařízení hodí. Pro vyčtení deskriptorů v čitelné podobě lze využít programů na PC, které deskriptor převedou do přehledné podoby. Do přílohy A přidávám kompletní deskriptor k CC2531 USB dongle ve formátu HTML.

3.4.2 Enumerace

Při připojení zařízení hostitel rozpozná, že je na sběrnici nové neznámé zařízení a započne proces enumerace, tedy proces registrace a konfigurace zařízení na sběrnici. Nejprve provede hostitel reset zařízení přivedením nulového napětí na oba vodiče dvojlinky. Následně vyčte deskriptor zařízení, ovšem k zajištění spolehlivosti zařízení znovu resetuje a deskriptor vyčte podruhé. Zařízení je poté přidělena unikátní adresa v rámci celé sběrnice (často je na sběrnici jediné). Hostitel následně vyčte další deskriptory a na základě těchto informací zařízení nakonfiguruje a také zvolí správný driver, kterému předá kontrolu nad tímto zařízením. [8]

3.5 CDC TŘÍDA

Firmware, který byl nahrán do TI CC2531 zajišťuje, že se zařízení chová jako standartní zařízení třídy CDC (Communication Device Class). Tato třída zahrnuje širokou škálu telekomunikačních a síťových zařízení jako například analogové telefony a modemy, také COM-port zařízení a ADSL modemy či Ethernet adaptéry.

CDC zařízení má tři základní úkoly. Správu zařízení a upozornění hosta na určité události, správu spojované komunikace (např. hovory) a také posílání aplikačních dat.

Podtřída ACM (Abstract Control Model) je schopna emulovat virtuální sériový port (COM-port). Stejnou podtřídu implementuje i firmware v TI CC2531. Podtřída se dále vyznačuje jedním interrupt IN endpointem pro posílání upozornění pro hosta a dvěmi bulk endpointy (IN, OUT) pro posílání aplikačních dat. [10]

3.6 USB DRIVER V OS ANDROID

Jak již bylo zmíněno, firmware v TI CC2531 je schopen emulovat klasickou COM-port komunikaci. Skutečně tomu tak je na PC s operačním systémem Windows. Stačí nainstalovat driver spravující CDC zařízení a s komunikačním čipem je možno komunikovat příkazy standardní sériové linky, jelikož se zařízení v systému hlásí jako COM-port device.

Naše řídící aplikace nicméně poběží na tabletu se systémem Android a bohužel většina Android zařízení v jádře systému neobsahuje drivery pro komunikaci přes sériový port. Starší zařízení dokonce nepodporují USB OTG, a tedy nejsou schopna pracovat v režimu USB host. Zařízení musí mít vhodnou hardwarovou a softwarovou výbavu. Fakt, že v systému chybí COM-port driver by šel vyřešit ručním napsáním ovladače a následnou modifikací jádra systému tak, aby tuto funkčnost umožňoval. Tento postup se ovšem neobejde bez tzv. rootu zařízení a vytvoření hardwarového driveru by bylo velice složité.

Nalezení řešení daného problému bylo zřejmě nejobtížnější částí celé práce, jelikož podpora USB periferií je v systému Android poměrně nová, z čehož plyne, že ještě není tolik využívaná, což znamená velmi málo zdrojů s informacemi jak postupovat při vývoji a také obtížný debugging. Reálné schopnosti tabletů a smartphonů při práci s USB se liší zařízení od zařízení. Často je i přes deklarovanou podporu USB OTG komunikace plná chyb, funguje pouze pro úzký výběr zařízení či nefunguje vůbec.

Po dlouhém průzkumu internetových zdrojů se následující řešení jeví jako nejschůdnější. Od verze systému Android 3.1 existuje v systému knihovna USB Host API. Tato knihovna nabízí sadu tříd, která umožňuje přístup k široké škále funkcí USB sběrnice. Lze tedy vytvořit tzv. soft-driver, což umožní řídící aplikaci zastoupit i roli USB ovladače. [11]

Tento postup bohužel není univerzální, jelikož USB Host API zdaleka neposkytuje veškerou funkčnost USB standardu. Jedním z omezení je například podpora pouze řídících a blokových přenosů. V našem případě to naštěstí není překážkou, jelikož CDC třída pro svoji funkci ani jiné typy přenosů nevyžaduje.

Výhodou USB Host API je na druhé straně kompletní správa fyzické vrstvy (kódování, modulace), správa vyšších vrstev komunikačního protokolu (provádění transakcí, kontrola chyb, spolehlivost přenosu) a také proces enumerace.

Vývojář v praxi nemusí znát USB standard do hloubky, aby ho byl schopen úspěšně zapracovat do svého projektu, jelikož USB API odstiňuje velkou část samotného USB přenosu a nabízí základní funkce typu blokový přenos, řídící přenos, vytvoření USB spojení či metody pro správu většího počtu USB zařízení.

I přes všechny plusy, které tato knihovna nabízí, ale vyvstává další nepříjemný problém, a to že vlastně přicházíme o možnost velice jednoduché sériové komunikace s čipem. Tohoto chování je možné dosáhnout vytvořením výše zmiňovaného soft-driveru, který vhodně využívá USB API a pro vyšší vrstvy aplikace poskytuje pouze funkce connect, close, serial read/write popř. řízení datového toku RTC/CTS.

3.7 USB SOFT-DRIVER

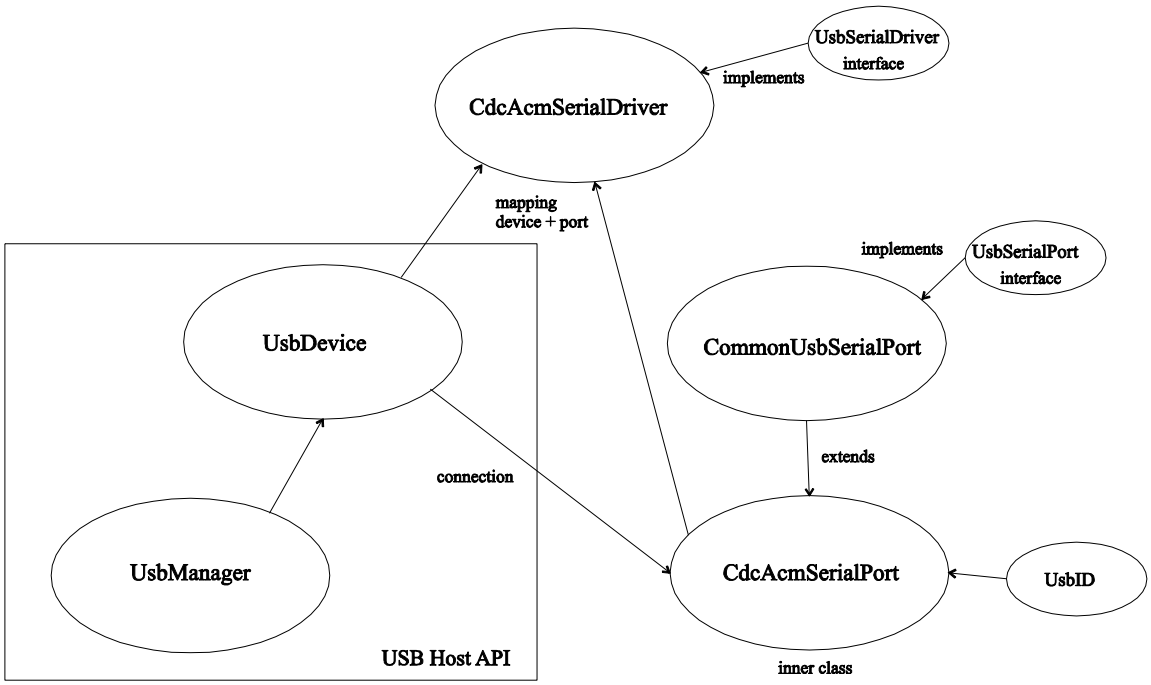
V průběhu vyhledávání informací při tvorbě tohoto projektu jsem narazil na několik zdrojů věnujících se podobné problematice (sériová komunikace s USB zařízeními v OS Android). Zprovoznění této komunikace je sice pro celý projekt kritické, ovšem zdaleka není hlavním cílem aplikace a navíc lze vždy použít například PC s jiným operačním systémem, který již tuto komunikaci podporuje. Lze předpokládat, že v budoucnu se určitá forma této funkčnosti v OS Android objeví, ale prozatím si ji každý vývojář musí zajistit sám.

Velká část z již výše zmiňovaných projektů tento problém vyřešila za pomocí knihovny USB-serial-for-android, což je volně šiřitelný projekt pod licencí GNU LESSER GENERAL PUBLIC LICENSE od autora Mikea Wakerlyho. [12]

Tento projekt se věnuje emulaci sériové komunikace v systému Android s využitím USB Host API. Velké části této knihovny vychází z knihovny od firmy FTDI (libFTDI) umožňující správu převodníků UART/USB, tedy také zajištění sériové komunikace po USB sběrnici ovšem pro jinou platformu než je OS Android.

V tomto projektu je sice využita tato knihovna, bohužel v průběhu testování se ukázalo, že knihovna není funkční a musela být částečně upravena pro potřeby této práce.

3.7.1 Struktura ovladače



Obrázek 6 - Struktura ovladače

Samotný ovladač se skládá z několika tříd ([Obrázek 6]). Na nejvyšším stupni hierarchie je třída CdcAcmSerialDriver. Tato třída implementuje obecné rozhraní UsbSerialDriver a její hlavní funkcí je mapování objektu UsbDevice a objektu implementujícího rozhraní UsbSerialPort.

Při vytváření instance tohoto objektu je nutno jako parametr předat objekt UsbDevice. USBDevice je již obsažen v USB Host API a reprezentuje zařízení připojené k Android hostiteli. Hlavními atributy jsou identifikační parametry zařízení a také jaké obsahuje rozhraní. Z objektu rozhraní jsme následně schopni získat odkazy na endpointy.

Samotný objekt UsbDevice získáme pomocí instance třídy UsbManager z USB Host API. Tato třída se také používá pro vytvoření spojení se zařízením (třída UsbDeviceConnection). Dříve než je toto spojení vytvořeno, aplikace musí mít oprávnění od uživatele, který explicitně povolí komunikaci s daným zařízením při startu aplikace, jinak spojení nebude vytvořeno.

Druhým atributem CdcAcmSerialPort je objekt implementující rozhraní UsbSerialPort. Tento objekt slouží k emulaci sériového přenosu a zajišťuje hlavní funkce tohoto driveru. Vlastní funkčnost je definována v rámci vnitřní třídy CdcAcmSerialPort, která je potomkem třídy CommonUsbSerialPort (tato abstraktní třída implementuje požadované rozhraní).

CdcAcmSerialPort nabízí všechny klasické funkce sériového portu. Při komunikaci je nejprve nutné otevřít spojení metodou open. Následně nakonfigurovat sériový kanál (počet datových bitů, parita, stop bit, baudrate). Lze zapnout řízení datového toku RTS/CTS. Poté máme k dispozici metody read/write, zajišťující sériový přenos dat. Pro ukončení komunikace se volá metoda close.

Tato třída také využívá vedlejší objekt UsbID spravující seznam identifikačních údajů kompatibilních zařízení.

Metoda open slouží k otevření sériového portu. Z pohledu USB komunikace se vyčte řídící a datový interface a jejich endpointy. U endpointů se zjistí jejich typy a směr datového toku. Získáme jeden řídící endpoint a dva datové endpointy typu bulk. Nyní jsme schopni provádět klasické USB přenosy.

Metoda setParameters slouží k nastavení parametrů sériového kanálu. Jmenovitě to je modulační rychlost, počet datových bitů, stop bitů a parita. Parametry jsou USB zařízení poslány přes řídící endpoint metodou controlTransfer, kterou lze zavolat nad objektem UsbDeviceConnection v paketu s definovanou strukturou (viz dokument [14], kapitola 6.2.2.). V tomto dokumentu lze najít i další parametrizaci chování zařízení, jako například nastavení řízení datového toku.

Metoda read umožňuje datový přenos ze zařízení do hostitele. Umožňuje synchronní a asynchronní přenos. Synchronní přenos je zajištěn metodou bulkTransfer. Asynchronní přenos využívá objektu UsbRequest a metody queue a requestWait. Na základě příspěvků na diskuzních fórech od uživatelů pracujících na podobných projektech se asynchronní čtení jevilo jako vhodné z hlediska spolehlivosti ([23]). Nicméně při testování zdánlivě náhodně docházelo k pádům aplikace. Prakticky náhodou jsem vypozoroval souvislost mezi pády aplikace a zhoršením signálu sítě například sevřením donglu do dlaně, či pokud se vysílače dostaly mimo dosah vysílání. V této fázi jsem ještě nevěděl o možnosti debbugingu aplikace pomocí Wi-Fi a jediný volný USB port, přes který by jinak ladění probíhalo, byl obsazen vysílačem. Tato chyba nakonec vedla ke zprovoznění debuggingu za pomocí Wi-Fi viz 2.2. Díky tomuto nástroji se ukázalo, že chyba nastává v USB driveru, pokud se vnějším zásahem přeruší příjem rámce. Bohužel stack-trace chyby indikoval, že k chybě dochází až na úrovni nativních knihoven (části Java knihoven psaných v jazyku C s využitím Java Native Interface) a uživatel na tak nízké úrovni ke kódu nemá přístup.

Zkusil jsem tedy využít synchronní čtení s využitím metody bulkTransfer. K chybám docházelo i v tomto případě, ale tentokrát výjimka vznikala na úrovni jazyka Java, jmenovitě IOException. Tento typ chyby již uživatel může ošetřit dle uvážení. Pro zachování co nejvyšší spolehlivosti přerušený rámec jednoduše zahodím, přesněji neposkytnu rámec vyšší vrstvě. Pokud se tablet dostane z dosahu modelů aut, vysílání jednoduše ustane do doby, než se vysílač dostane do vzdálenosti umožňující komunikaci, kdy se znovu automaticky obnoví bez nutnosti programově zasahovat do Zigbee sítě.

Metoda write využívá k synchronnímu přenosu dat z hostitele do zařízení také metodu bulkTransfer.

3.8 INICIALIZACE A VYUŽITÍ OVLADAČE

Pokud chceme využít ovladač v Android aplikaci, je nutné provést několik inicializačních kroků. Nejprve je nutné získat povolení pro práci s USB zařízením od uživatele. Toho se dá dosáhnout vytvořením tzv. filtru záměrů s parametrem USB_DEVICE_ATTACHED. Díky tomuto filtru bude aplikace upozorněna při připojení USB zařízení k tabletu a následně spuštěna s potřebnými přístupovými právy k zařízení. Pokud nevyužijeme filtr záměrů, lze vygenerovat dialog o povolení přístupu přímo v aplikaci.

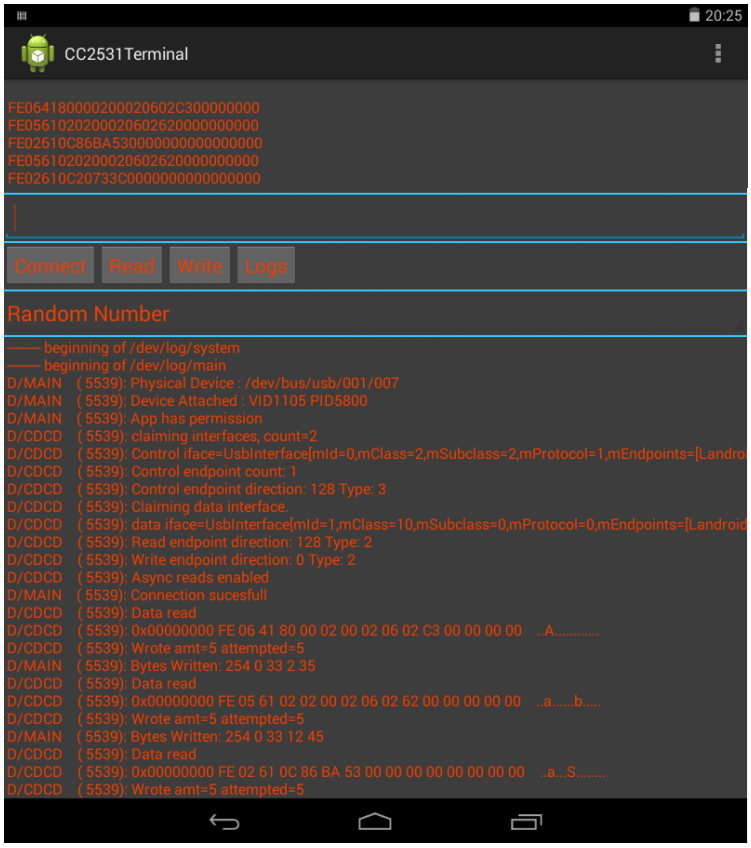
Dalším nezbytným krokem je vytvoření objektu UsbManager. Tento objekt se stará o zprávu připojených USB zařízení. Lze z něj také zjistit, jestli máme povolení pro práci s daným zařízením a vytvořit spojení.

Získáme-li zařízení, které odpovídá požadovaným parametrům (PID, VID), můžeme vytvořit instanci třídy CdcAcmSerialDriver a začít s ní pracovat. Při otevírání sériového portu budeme také potřebovat objekt odkazující na spojení se zařízením (UsbConnection), který získáme pomocí UsbManager.

Správná inicializace USB komunikace musí být zajištěna částečně i hardwarově. Po vytvoření spojení se zařízením je nutné komunikační čip manuálně resetovat. Slouží k tomu hardwarové tlačítko na USB donglu s označením S2. Při resetu zařízení odešle zprávu indikující tento proces. Nyní již můžeme komunikačnímu čipu sériově posílat příkazy, na které nám bude čip reagovat a generovat odpovědi.

3.9 Test USB komunikace na OS Android

Komunikace mezi čipem CC2531 a tabletem se systémem Android byla nakonec úspěšně zprovozněna. Funkční komunikaci lze demonstrovat na vzorové aplikaci chovající se jako jednoduchý komunikační terminál (Obrázek 7).



Obrázek 7 - Komunikační terminál Android

Aplikace umožňuje odesílání příkazů ve formátu podle specifikace Z-Stack [5]. Příkazy lze psát ručně, či využít příkazů předdefinovaných.

3.10 Test USB komunikace na PC

I přesto, že je aplikace vyvíjena pro Android, bylo v průběhu výhodné vytvoření obdobné aplikace i na PC s Windows, hlavně z toho důvodu, že nebylo nutné aplikaci testovat přímo na fyzické platformě s modely aut, ale za pomocí druhého CC2531 donglu bylo možné simulovat chování celé platformy virtuálně. V praxi aplikace v tabletu využívá vlastní dongle a chová se, jakoby pracovala s reálnou platformou, ovšem veškerá komunikace ze strany platformy a modelů aut je simulována z počítače. Tento způsob je velmi vhodný zejména pro testování nestandartních situací.

Úloha sériové komunikace je mnohem méně obtížná, jelikož klasické PC je více standardní a také lépe přizpůsobeno ke komunikaci s periferiemi. Pro úspěšné připojení donglu k počítači je nutné nainstalovat drivery sloužící ke komunikaci s CDC zařízeními. Tento driver je obsažen v doprovodné softwarové výbavě k CC2531. Po instalaci a připojení zařízení se dongle tváří jako COM port a lze s ním pracovat jako se sériovou linkou.

Sériovou komunikaci se zařízením lze ve Windows uskutečnit mnoha způsoby. Jedním z nich je například vlastní aplikace v jazyce C++, který již obsahuje knihovny obsluhující sériový port. Pokud ovšem vezmeme v úvahu fakt, že aplikace v Androidu se programují v Javě, je nejvhodnější volbou psát tento program právě v tomto jazyce. Díky tomu lze stejný kód používat na obou systémech. Bohužel s použitím Javy vyvstává další problém a tím je neexistence knihoven pro práci se sériovým portem.

Naštěstí existuje open-source projekt RXTX library. Tento projekt se zabývá vytvářením portu knihovny Java Communications API. Tato knihovna umožňuje emulaci a komunikaci ve standardu EIA232, nicméně je dostupná pouze pro systémy Linux a Solaris. Projekt RXTX tedy umožňuje využití jejich funkcí i na systému Windows. [19] [20]

Sériová komunikace byla úspěšně otestována na obdobné aplikaci jako na tabletu. Program nabízí funkce jednoduchého komunikačního terminálu. Nyní je již možné psát vyšší vrstvu programu obsahující logiku vytvoření Zigbee sítě pro oba operační systémy, s tím že stejný kód bude fungovat na obou zařízeních. Tento fakt značně usnadňuje například debugging, jelikož vývoj programu na PC je daleko snadnější.

Do přílohy B k této práci přikládám návod na instalaci RXTX knihovny a její využití v IDE Netbeans, jelikož instalace není úplně triviální a do budoucna by se to mohlo někomu hodit.

4 ZIGBEE

Zigbee je protokol umožňující vytvoření a správu rádiových sítí, často tvořených rádiovými vysílači s nízkým výkonem a malými rozměry. Zigbee je založeno na standartu IEEE 802.15.4. Dosah vysílání je od desítek po stovky metrů, přičemž vysílání je v ISM pásmu (2,4 GHz). Maximální rychlost je 250 Kbps a samotný radiový vysílač potřebuje k funkci velmi malé množství energie (na baterii je zařízení schopno fungovat několik let). Tyto parametry se hodí hlavně pro komunikaci mezi senzory či vstupně výstupními zařízeními. Mezi často využívané aplikace patří bezdrátové spínače k osvětlení, řízení vytápění domů či řízení dopravy. Standard je poměrně mladý (od roku 2003), a proto se teprve pomalu rozšiřuje do mnoha průmyslových odvětví.

Rádiové vysílače jsou většinou integrovány na jediném čipu s řídícím mikrokontrolérem a vstupně výstupními periferiemi. Častou výbavou je také sériové rozhraní nebo USB pro komunikaci s PC. [15]

Zigbee protokol je založen na klasickém modelu distribuovaných systémů ISO/OSI. Implementací Zigbee protokolu se v současné době zabývá několik firem, přičemž mezi nimi může být mnoho rozdílů. Skupina Zigbee Alliance se zabývá testováním těchto protokolů a vydáváním potvrzení, že splňují daný standard a mohou využívat označení Zigbee.

V našem projektu využíváme Zigbee implementaci Z-Stack Home Automation od Texas Instruments. Tato implementace drží certifikaci Golden Unit, která indikuje, že stack splňuje standard s velkou mírou detailu a funguje jako vzorové řešení pro další Zigbee protokoly od jiných firem, které jsou vůči němu porovnávány. Mimo Texas Instruments tuto certifikaci drží také firmy Freescale, Ember a Integration. Tyto implementace jsou navzájem kompatibilní, díky čemuž jsou výrobky od těchto výrobců schopny spolupráce. [13]

4.1 IEEE 802.15.14

Tento standard definuje fyzickou a spojovou vrstvu pro bezdrátové sítě (PAN – personal area networks) s nízkou přenosovou rychlostí. Není využíván pouze Zigbee technologií, ale například i technologiemi 6LoWPAN nebo MiWi. [16]

4.1.1 Fyzická vrstva

Na fyzické vrstvě se k zajištění dobré spolehlivosti i ve velice zarušeném prostředí využívá modulace DSSS a O-QPSK. [13]

Základní modulací je O-QPSK využívající fázovou modulaci k zakódování symbolů.

DSSS je modulace s rozprostřeným spektrem, kdy se k toku dat v modulátoru přidává i pseudonáhodný signál o datovém toku vyšším než původní data. Tato modulace zajišťuje odolnost vůči úzkopásmovému rušení a umožňuje sdílení pásma většímu počtu vysílačů. Částečně zajišťuje i obtížnější odposlech, jelikož k dekódování signálu je nutno znát rozmítací signál. [8]

4.1.2 Spojová vrstva

Spojová vrstva umožňuje přenos dat, pokud jsou uzly v přímém dosahu. Rozumí spojové vrstvě protokolu, což znamená, že zajišťuje i adresaci. Mimo to se stará i o spolehlivost, k čemuž využívá několik postupů.

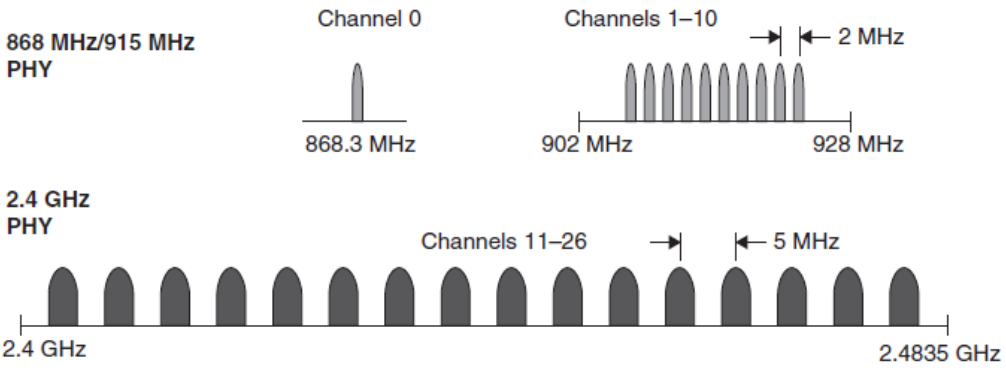
V první řadě implementuje přístupovou metodu ke sdílenému médiu ve variantě CSMA/CA. Zigbee je samo o sobě asynchronní sítí, tedy uzly mohou vysílat kdykoliv, ovšem s jediným omezením a to, že musí respektovat tuto přístupovou metodu, aby nedocházelo ke kolizím. Pokud chce zařízení vysílat, musí nejprve otestovat, zda je kanál volný. Pokud ano, tak může odeslat data. Je-li kanál obsazený, musí čekat na jeho uvolnění, poté zvolit náhodnou čekací dobu z intervalu s exponenciálně rostoucí horní hranicí, a pokud i po tomto čekání je kanál stále volný, může vysílat. K čekání dochází i při neúspěšném přenosu. [8]

K detekci chyb rámce obsahují šestnácti bitové CRC, přičemž se zde využívá potvrzování bezchybně přijatých rámců (ACK). Pokud nepřijde odesílateli potvrzení o bezchybném přijetí, je odeslání rámce zopakováno (maximálně třikrát).

Mimo vysílání unicast lze vysílat i broadcast a multicast (při těchto režimech se nepoužívá ACK).

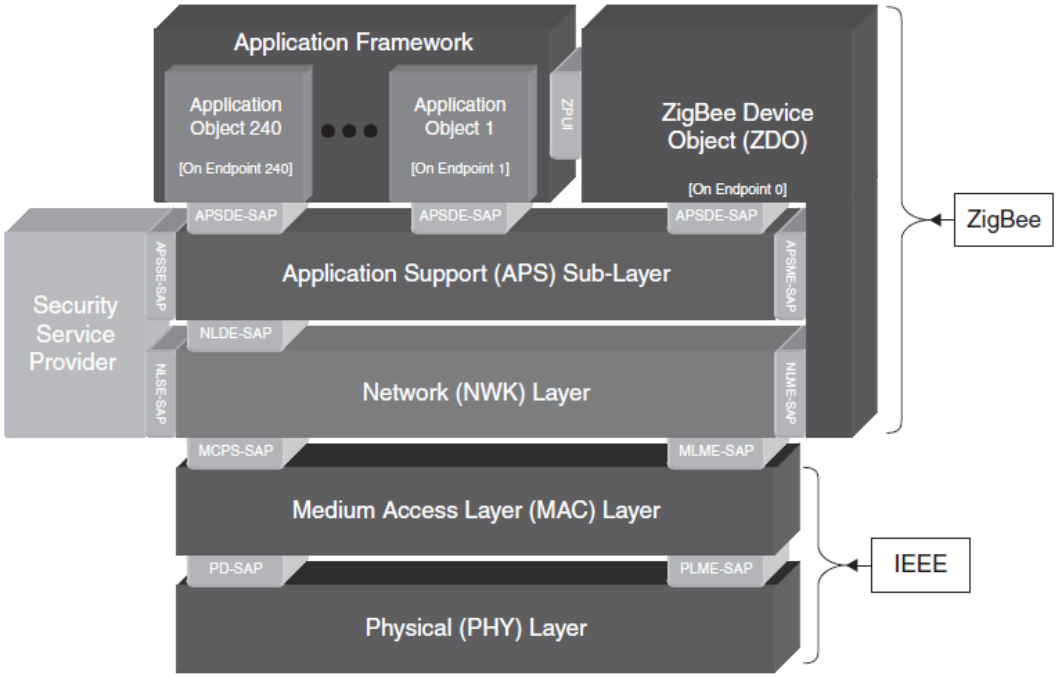
Standard 802.15.4 definuje mechanismy pro vytváření a objevování sítí a také připojování se do nich. Umožňuje skenovat jednotlivé kanály, zjistit míru jejich rušení a následně kanál, na kterém síť pracuje změnit.

Fyzické médium je rozděleno na 16 kanálů, přičemž rozestupy mezi kanály jsou 5 MHz ([Obrázek 8]). [13]



Obrázek 8 - Rozdělení fyzického média [13]

4.2 ZIGBEE PROTOKOL



Obrázek 9 - Zigbee protokol [13]

Všechny vyšší vrstvy protokolu (síťová až aplikační) již zajišťuje standard Zigbee ([Obrázek 9]).

V rámci Zigbee sítě se rozlišují tři typy zařízení: Zigbee Coordinator, Zigbee Router a Zigbee Device.

4.2.1 Zigbee Coordinator

Zigbee Coordinator je jediné zařízení, které může vytvořit síť. Při vytváření volí kanál, na kterém bude síť pracovat a její identifikační číslo (PAN ID). Volí také další parametry sítě na aplikační vrstvě, jako například profil Zigbee protokolu a jiné. Po vytvoření sítě se již chová jako router. [13]

4.2.2 Zigbee Router

V možnostech routeru je nalezení a připojení se k síti, rozšiřování broadcast vysílání po síti a povolení k připojení koncového zařízení k síti. [13]

4.2.3 Zigbee End Device

Hlavní výhodou koncových zařízení je, že mohou trávit dlouhé časové intervaly v režimu sleep aniž by ze sítě vypadly. [13]

4.2.4 Síťová vrstva

Nejdůležitější prvek zajištěný síťovou vrstvou je mesh networking. Topologie sítě nemá pevně danou strukturu, ale každý uzel je spojen se všemi uzly, které jsou v dosahu. Pokud chceme odeslat paket uzlu, který je v síti, ale není v přímém dosahu, je tato zpráva doručena na základě síťové adresy přes několik uzlů, přičemž cesta je dynamicky vyhledávána.

Zigbee neumožňuje komunikaci mezi jednotlivými sítěmi jako například v internetu. Jedinou možností je využití skupin, ovšem vždy musí všechna zařízení náležet k síti vytvořené pouze jedním ZC. Jedinou možností je využití brány, která komunikaci mezi sítěmi zajistí pomocí jiné technologie (WiFi, Ethernet). [13]

4.2.5 APS

APS spolu s Application Framework funguje jako aplikační vrstva ISO/OSI. Rozumí aplikačním požadavkům na Zigbee protokol. Zařízení logicky dělí na endpointy, což jsou entity, na kterých komunikují aplikace (podobné portu v IP protokolu). Zajišťuje také end-to-end ACK, tedy zajištění spolehlivého přenosu i pokud zpráva projde přes několik uzlů (multi-hop). Dále zajišťuje například filtraci duplicitních paketů, či filtraci paketů při použití skupin. Mimo jiné spravuje tzv. binding tables, které umožňují emulaci spojované komunikace. Tato vrstva také zajišťuje ochranu proti odposlechnutí komunikace za pomocí šifry AES 128 bit. [13]

4.2.6 Application Framework

Tato vrstva nemá svůj vlastní datový rámec, ale je pouze sada rutin a postupů, které standardizují přístup ke stacku pro aplikace. Zahrnuje to implementaci endpointů a jak jsou zpracovány datové operace pro jednotlivé výrobce. Tato vrstva může přistupovat k nižším vrstvám pouze prostřednictvím APS. [13]

4.2.7 Zigbee Device Object

Speciální aplikací běžící na endpointu nula je Zigbee Device Object. Tento objekt tvoří tzv. uživatelskou vrstvu, která není součásti ISO/OSI standardu. ZDO má možnost komunikace nejen prostřednictvím AF a APS, ale může i přistoupit přímo až na síťovou vrstvu. Tento adresář objektů, obsahuje objekty, které dávají informaci o celkovém stavu Zigbee zařízení. Na základě těchto objektů je celá komunikace parametrizována. [13]

4.3 Využití Zigbee v projektu

I přesto, že Zigbee protokol nabízí opravdu širokou škálu funkcí přes všechny vrstvy od fyzické, až po aplikační v našem projektu potřebujeme využít pouze nejzákladnější funkce a to vytvoření sítě a přenos datových paketů.

Z-Stack k tomuto účelu nabízí využití rozhraní SimpleAPI. Tato sada příkazů umožňuje vytvoření funkční sítě s minimálním počtem úkonů a parametrů. Bohužel se v průběhu testování projevilo mnoho nesrovnalostí s dokumentací a chování sítě vytvořené touto formou bylo velmi nedeterministické. Nakonec byla využita plnohodnotná sada příkazů dle dokumentu CC2530ZNP Interface Specification ([5]). Tento dokument nicméně nevysvětluje příliš samotné vytvoření sítě a interakce v ní, ale spíše formát a účel jednotlivých příkazů v API. Dostatek informací o vytvoření funkční sítě lze najít například v dokumentu přímo od Texas Instruments: Developing a ZigBee® System Usinga CC2530-ZNP Approach ([21]).

4.3.1 Inicializace Zigbee sítě

Podle výše zmíněného dokumentu je nejprve nutné zařízení resetovat příkazem SYS_RESET_REQ, poté nastavit čip tak, aby po startu vymazal všechny dosavadní uložené stavy a zase resetovat. Touto sadou příkazů dosáhneme toho, že se čip bude chovat vždy deterministicky.

Při testování sekvence jsem ovšem narazil na velmi významný problém. Dongle po odeslání SYS_RESET_REQ přestal vždy reagovat. Podezření nejprve padlo na systém Android a jeho implementaci USB driveru. Toto podezření nemělo dlouhého trvání, jelikož se vzápětí ukázalo, že stejný problém nastává i pokud je čip připojen k počítači.

Dongle je nutné po připojení vždy resetovat manuálně tlačítkem. Jako nejjednodušší řešení se nabízelo vynechání celé resetovací sekvence a spokojení se pouze s funkčním manuálním resetem. Toto řešení se také zpočátku jevilo jako přijatelné. Síť se povedlo vytvořit a jiná zařízení se do ní byla schopna připojit, bohužel problém nastal, jakmile jsem se pokusil změnit parametry sítě, jako například PAN ID. Čip ne vždy reagoval na tyto změny a někdy pokračoval v činnosti s neaktuálními parametry. Bylo jasné, že sekvence zajišťující kompletní vymazání všech uložených stavů byla opravdu nutná.

Při hledání informací na fórech Texas Instruments viz [25], jsem nalezl příspěvky několika uživatelů řešících stejný problém. Nakonec se ukázalo, že příkaz SYS_RESET_REQ provede tzv. hard reset, tedy kompletní odpojení zařízení a jeho znovu připojení. Tento typ resetu funguje například na verzi čipu CC2530, který komunikuje po sběrnici SPI a nepotřebuje projít procesem enumerace. Pokud tímto způsobem resetujeme USB dongle, tak datové proudy a handly k zařízení uváznou na mrtvém bodě a USB host již není schopen se zařízením komunikovat.

Naštěstí jsem na tento problém nalezl řešení v podpůrném rozhraní Z-Stack Monitor and Test API ([22]). V sadě příkazů MT_SAPI Commands je obsažen příkaz ZB_SYSTEM_RESET. Příkaz provede tzv. soft reset, tedy pouhý skok programového čítače na reset vektor. Díky tomu se nepřeruší USB komunikace a čip je schopný nadále komunikovat s hostem.

Provedeme-li nyní inicializační sekvenci, máme komunikační čip s jasně definovaným stavem a můžeme přistoupit k parametrizaci sítě. Samotný průvodce vytvoření sítě využívá pokročilejší možnosti sítě, jako zabezpečení přístupu k síti či rozšířenou PAN ID. V naší aplikaci ovšem tyto prvky nejsou nutné a jejich využívání by bylo zbytečné. Dle specifikace ZNP je nutné minimum nastavení určení PAN ID, čísla vysílacího kanálu a mód, ve kterém bude čip pracovat (Coordinator, Router, End-Device). V naší aplikaci byla zvolena hodnota PAN ID 3 a kanál 14. Modely aut fungují jako Zigbee routery a USB dongle jako koordinátor.

Posledním parametrem, který byl nastaven je ZCD_NV_ZDO_DIRECT_CB, díky kterému bude koordinátor schopen přijímat indikace připojení nového zařízení, což se osvědčilo hlavně při testování. Všechny výše jmenované parametry se nastavují jediným příkazem a to ZB_WRITE_CONFIGURATION, pouze s různými parametry pro každou veličinu.

Pro vytvoření sítě již stačí odeslat příkaz AF_REGISTER. Tento velmi obsáhlý příkaz nakonfiguruje aplikační vrstvu Z-Stacku a následně vytvoří síť. Povinným parametrem je číslo endpointu na kterém poběží aplikace a následně lze nastavit, které clustery bude

aplikace využívat. Clustery nabízí řadu funkcí pro standardizované způsoby využití Z-Stacku a slouží hlavně k interoperabilitě zařízení od různých výrobců. V naší aplikaci služeb clusterů nevyužíváme. Po odeslání AF_REGISTER čip vysílá několik zpráv po USB. Tyto zprávy indikují proces vytváření sítě a umožňují jeho monitoring. Nakonec přijde indikace úspěšné inicializace a síť je připravena k používání. USB dongle tento proces indikuje také hardwarově. Při vytváření sítě bliká červená dioda, a pokud se rozsvítí zelená dioda, inicializace byla úspěšně dokončena.

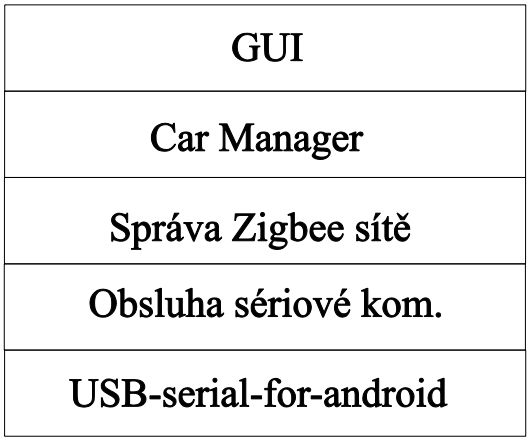
4.3.2 Používání Zigbee sítě

Nyní se již zařízení mohou do sítě připojovat. Jediný rozdíl v jejich inicializaci je příkaz určující roli v rámci sítě. Pokud se zařízení do sítě připojí, koordinátor dostane zprávu s jeho MAC adresou a Zigbee adresou. Po připojení mohou zařízení komunikovat pomocí příkazu AF_DATA_REQUEST. Odesílatel musí určit adresu cíle nebo zvolit adresu FFFF značící broadcast. Dalším důležitým prvkem je správná adresace endpointu. I přesto, že adresa cíle bude správně, tak pokud neodešleme zprávu na správný endpoint (viz 4.2.5), příjemce zprávu nezaregistruje. Dále lze nastavit několik parametrů jako poloměr broadcastu či zda bude zpráva potvrzovaná či nikoliv. Nakonec samozřejmě nesmí chybět datový payload, který může být až 128 bytů dlouhý. Odeslaná data příjemci přijdou v rámci zprávy AF_INCOMING_MSG. Mimo informací ohledně odesílatele a datového payloadu, obsahuje AF_INCOMING_MSG také informace o kvalitě spojení, způsobu doručení a času přijetí zprávy.

V rámci tohoto projektu odesíláme zprávy výlučně broadcastem a adresaci řešíme až na vyšší vrstvě v naší aplikaci. Tento způsob byl zvolen kvůli požadavku adresace modelů aut pomocí čísel jejich karoserií, přičemž ve standardu Zigbee není možná manuální adresace, jelikož adresy jsou přiřazovány automaticky tak, aby byla komunikace optimální. V praxi tedy zařízení po připojení pošle paket, ve kterém sděluje svoji přidělenou Zigbee adresu a také svoji logickou adresu (100 + číslo karoserie). Zigbee adresy jsou poté uloženy do kolekce, a slouží pouze ke zjištění toho, zda je již auto s danou adresou připojeno, či ne. Datové pakety jsou následně adresovány pouze logickou adresou a posílány broadcastem (paket přijmou všechna zařízení v síti, ovšem věnují mu pozornost, pouze pokud se pole logické adresy shoduje s jejich vlastní logickou adresou).

5 VÝVOJ ŘÍDÍCÍ APLIKACE

Vývoj řídící aplikace probíhal v několika fázích. Postupoval jsem od nejnižších vrstev pomyslného komunikačního ISO/OSI modelu tedy tzv. bottom-up návrh. Díky tomu vzniklo několik plně funkčních aplikací na PC i pro Android s různou komplexností. Každá tato verze sloužila k otestování aktuálně implementované funkčnosti. Vývoj by se dal rozdělit do čtyř hlavních fází (viz Obrázek 10).



Obrázek 10 - Rozdělení aplikace do funkčních vrstev

Nejnižší vrstvou byla sériová komunikace po USB. Tato vrstva již byla z části popsána v kapitole USB driver v OS Android. Důležitým prvkem aplikace je nicméně také způsob využití tohoto driveru. V praxi bylo nutné využití vláken pro sériovou komunikaci a vyřešit jejich konkurenci tak, aby aplikace pracovala plynule. Poněkud obtížnější bylo také řešení předávání přijatých rámců vyšším vrstvám a mezi vlákny. Jelikož oba sériové drivery (RXTX, usb-to-serial) nabízí různě bohaté možnosti komunikace, řešení muselo být zvoleno tak, aby bylo co nejvíce kompatibilní s oběma systémy s minimem nutných úprav.

Druhou vrstvou aplikace je správa Zigbee sítě. Bylo nutné vyřešit způsob, jakým bude spolehlivě vytvořena síť a jakým způsobem budou pakety předávány do vyšších vrstev. Tato vrstva je naprosto totožná pro oba využívané operační systémy.

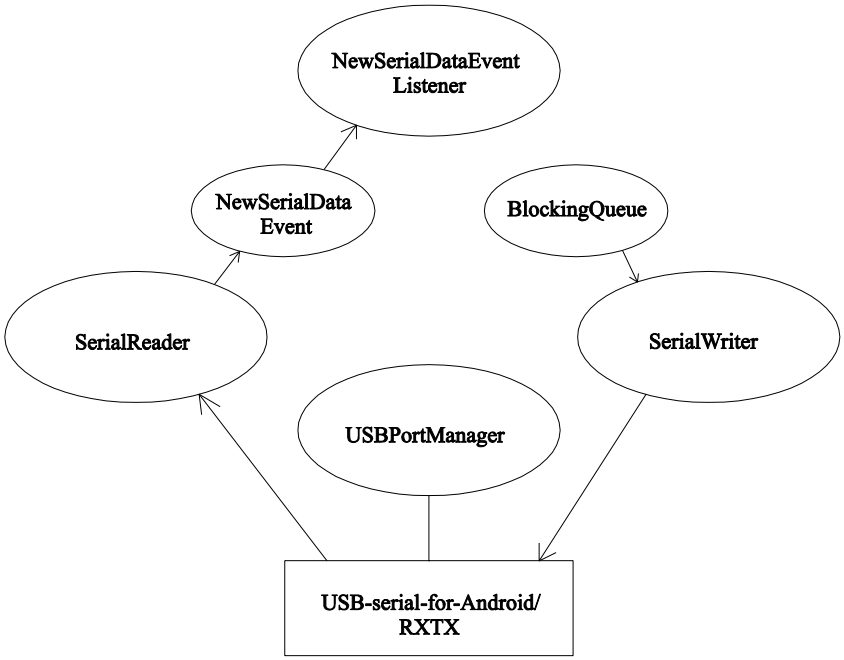
Poslední vrstvou, která byla implementována jak pro PC, tak pro Android je logika a obsluha platformy s modely aut. Do této vrstvy reprezentované třídou CarManager patří například adresace a registrace modelů, parametrizace experimentu a ukládání dat ze senzorů.

Nejsvrchnější a také nejobsáhlejší částí aplikace je grafické uživatelské rozhraní. Toto rozhraní již bylo vyvíjeno pouze pro systém Android. Nižší vrstvy jsou nicméně náležitě zapouzdřeny a proto je vytvoření grafického rozhraní v budoucnu například pro PC vcelku triviální záležitostí. GUI bylo také psáno s předpokladem, že v blízké budoucnosti se přejde v projektu na Wi-Fi komunikaci.

V následujících kapitolách postupně popíšu detaily implementace jednotlivých vrstev a také nástroje využité k vývoji.

5.1 OBSLUHA SÉRIOVÉ KOMUNIKACE

Způsob obsluhy sériové komunikace se na PC liší od Androidu z důvodu použití různých knihoven pro správu USB komunikace. Řešení pro oba systémy jsem nicméně zvolil tak, aby se co nejméně lišilo, což umožňuje využití identických vyšších vrstev.



Obrázek 11 - Správa sériové komunikace

5.1.1 Android

Vrstva využívající lehce upravený výše zmíněný soft driver pro simulaci sériové komunikace přes USB od Mikea Wakerlyho je složena ze tří hlavních tříd, jednoho rozhraní a jedné pomocné třídy.

Třída pracující na nejnižší vrstvě přímo nad USB driverem se nazývá USBPortManager. Hlavním úkolem této třídy je vytvoření a uzavření virtuálního sériového portu. Důležitým úkolem je také obstarání povolení pro práci s USB zařízením od uživatele. Obdržení povolení od uživatele je vynuceno samotným systémem a nelze jej obejít. Aplikace umožňuje dva způsoby předání tohoto povolení. Je vytvořena tak, aby rozpoznala připojený dongle CC2531, přičemž uživateli se zobrazí nabídka aplikací, které jsou vhodné pro práci s tímto zařízením. Pokud uživatel zvolí tuto aplikaci, předá tím také práva pro práci s USB. Druhou možností je připojit USB dongle až po startu aplikace. V tomto scénáři třída USBPortManager pozná, že aplikace nemá povolení a vygeneruje dialogové okno žádající uživatele o právo správy USB sběrnice. Po úspěšném otevření sériového portu předá tato třída jeho referenci vyšším a vedlejším třídám, které s ním dále pracují.

Třídy SerialReader a SerialWriter běží ve vlastních vláknech a starají se o odesílání a příjem dat ze sériového portu. Jejich implementace si vyžádala poměrně značné úsilí. Jednou z hlavních výhod knihovny RXTX je totiž možnost naslouchání na sériovém portu, přičemž pokud přijdou nová data, knihovna je schopna vygenerovat událost. To je velice vhodné, hlavně do aplikací s uživatelským rozhraním, jelikož ty jsou navrženy tak, aby byly řízeny pouze událostmi a ne cyklem. Sériová komunikace v aplikacích s uživatelským rozraním je navíc téměř výlučně řešena za pomocí vláken, protože pokud

by obsluha sériového portu běžela v hlavním vlákně obsluhujícím také GUI, aplikace by byla často neresponzivní, což značí špatný návrh.

Mým cílem tedy bylo dosáhnout obsluhy sériového portu, aniž bych omezil fungování uživatelského rozhraní a aby se nová data předávala do vyšších vrstev formou událostí. Jako vhodné řešení se nabízelo vytvoření čtecího vlákna, které by provádělo tzv. polling, což znamená periodické čtení z portu s periodou několik milisekund, přičemž pokud jsou zaznamenána nová data, třída vygeneruje událost pro vyšší vrstvy. Podobný princip by se dal využít i pro zapisovací vlákno. To by si periodicky ověřovalo, jestli má v určitém zásobníku rámců něco k odeslání. Tato úloha se dá vyřešit značně neoptimálně pouhým vytvořením dvou vláken s nekonečným cyklem, přičemž v každém kroku vlákno provede daný úkol a je uspáno na fixní čas. Z hlediska optimalizace je nicméně tento způsob velice neohrabaný a plýtvá zdroji, které jsou zrovna na zařízeních se systémem Android ještě stále omezené. Problémem je také předávání přijatých paketů mezi vlákny. Při testování tohoto návrhu část docházelo k nekonzistenci dat a následným chybám celé aplikace. Bylo nutné najít způsob využívající pokročilejší správu konkurence vláken.

Po dlouhém průzkumu možností Javy a Android knihoven jsem problém vyřešil následovně. Zapisovací vlákno využívá speciální kolekci zvanou BlockingQueue. Tato implementace fronty umožňuje svázání s vláknem, které z ní vyčítá prvky. Pokud je fronta prázdná, obsluhující vlákno je uspáno do doby, než je do ní přidán nový prvek. Zapisovací vlákno tak běží pouze tehdy, pokud má něco na práci. K Blocking Queue má přístup nadřazená třída ZBCommunicaitonManager, která do ní podle potřeby asynchronně vkládá rámce k odeslání.

Implementace čtecího vlákna byla poněkud složitější, jelikož BlockingQueue je k dispozici v klasické Javě, tím pádem i v systému Android. Při psaní aplikace na PC, která využívá grafickou knihovnu SWING, jsem objevil metodu invokeLater(). Tato metoda umožňuje vložení Runnable objektu do fronty událostí, které budou obslouženy v hlavním vlákně aplikace, které obsluhuje i grafické rozhraní. Chytrým využitím této funkce lze komunikovat mezi vlákny a dokonce předávat data tak, aby nedocházelo k nekonzistenci. RXTX umožňuje dotázat se sériového portu, zda jsou k dispozici nová data. Pokud ano, zavolá se metoda invokeLater, do které je vložen Runnable objekt, jehož jedinou náplní je vygenerování události NewSerialDataEvent (pomocná třída). Jakémukoliv objektu běžícímu v hlavním vlákně implementujícímu listener zachytávající tento typ událostí je následně umožněno událost zpracovat.

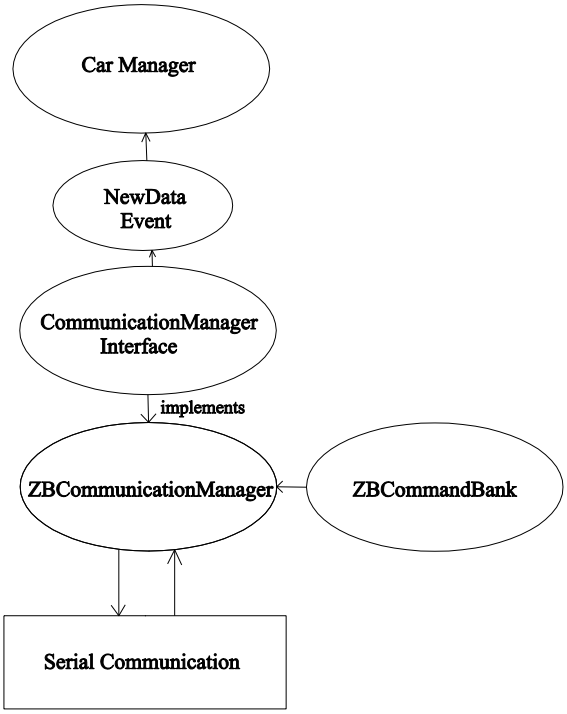
V knihovnách systému Android existuje objekt Handler. Tento objekt se sváže s vláknem, ve kterém byl vytvořen, přičemž handleru je následně možné posílat speciální zprávy nebo i objekty typu Runnable odkudkoliv z aplikace a hlavně z jiných vláken. Je ho tedy možné využít naprosto stejným způsobem jako metodu invokeLater(). Posledním rozdílem mezi RXTX a sériovým driverem na Android je způsob načtení kompletních rámců. RXTX umožňuje zjistit kolik bytů je na portu připraveno ke čtení. Stačí tedy pouze alokovat pole o stejné délce a data do něj načíst. Pokud na portu nejsou k dispozici nová data, RXTX vrátí poslední přijatý rámec. Soft driver v Androidu tuto možnost nemá, ale formou pokusů jsem zjistil, že pokud chceme z portu číst, lze to udělat pouze jednou. Pokud data nejsou k dispozici, vlákno se zablokuje do té doby, než data přijdou. Jestliže nealokujeme dostatečně dlouhé pole, čtení skončí výjimkou. Pokud ovšem vždy zajistíme dostatečnou velikost datového bufferu, máme jistotu, že vyčteme kompletní rámec, ba dokonce pokud na portu nejsou nová data, vlákno se uspí, dokud

data nepřijdou, což zase snižuje nároky na výkon. Perioda pollingu byla nastavena na 20 ms. Toto komplexní řešení zajišťuje konzistenci dat, jelikož data jsou předána vyšším vrstvám až pokud je jisté, že nový rámec došel kompletní.

5.1.2 PC

Třídy řešící obsluhu sériové komunikace s využitím knihovny RXTX mají podobnou strukturu jako v Android aplikaci. Je zde třída řešící otevření sériového portu, dvě třídy spravující zápis a čtení ve vlastních vláknech a několik pomocných tříd. Jediný rozdíl v implementaci je výše zmíněné využití metody invokeLater() místo Handleru, načítání rámců a způsob otevření sériového portu. Ke čtení a zápisu jsou také využívány standartní byte streamy. Postup otevření sériového portu je detailně popsán na stránkách projektu RXTX. [20]

5.2 VYTVOŘENÍ A OBSLUHA ZIGBEE SÍTĚ



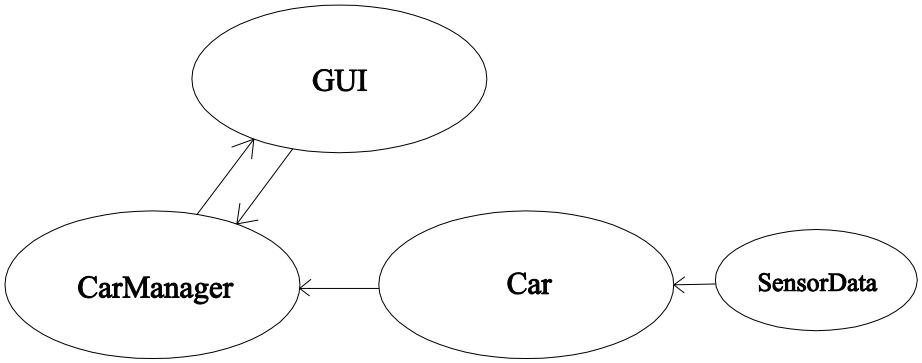
Obrázek 12 - Obsluha a vytvoření Zigbee sítě

Na této vrstvě je implementace identická pro oba operační systémy. Jediný rozdíl je v parametrech předávaných CC2531, na základě toho zda chceme, aby pracoval v režimu Zigbee Router, tedy jako model auta, nebo jako Zigbee Coordinator. V praxi jsem čip připojený k PC využíval jako Zigbee Router a čip připojený k tabletu jako Zigbee Coordinator.

Jelikož je aplikace do budoucna zamýšlena i pro komunikaci pomocí Wi-Fi, snažil jsem se tuto část aplikace vytvořit tak, aby se dalo k jiné bezdrátové technologii přejít jednoduchou úpravou zdrojového kódu. Pro tento účel jsem vytvořil rozhraní CommunicationManager definující základní funkce obsluhující bezdrátovou komunikaci, jmenovitě metody connect, disconnect a send. V budoucnu tedy stačí napsat třídu implementující toto rozhraní, popřípadě i nižší vrstvy spravující komunikaci pomocí Wi-Fi, připojit tyto třídy k projektu a upravit jedinou řádku při inicializaci CommunicationManager a aplikace by měla teoreticky fungovat.

Celá Zigbee síť je spravována třídou ZBCommunicationManager, která implementuje výše zmínění rozhraní. Třída samozřejmě obsahuje i další metody, ovšem ty jsou již specifické pro Zigbee. Jakým způsobem je síť vytvořena a jak jsou po ní posílána data z pohledu Z-Stacku je detailně popsáno v kapitole 4.3. K pohodlnému ovládání Z-Stacku jsem si vytvořil třídu ZBCommandBank, která obsahuje kompletní seznam použitých příkazů. Každá metoda sestaví požadovaný příkaz na základě jejich parametrů, přičemž jména metod se shodují se jmény příkazů. ZBCommunicationManager je kompletně řízen událostmi. Události jsou dvojího typu, události z vyšších vrstev nebo události na sériovém portu. Na základě událostí z vyšších vrstev se Manager přepíná mezi několika stavy, na základě čehož reaguje různými způsoby na příchozí zprávy ze sériového portu. Pokud je například ve stavu vytváření sítě, tak po každém inicializačním příkazu čeká na přesně danou odpověď. Příkaz opakuje, dokud mu tato požadovaná odpověď nepřijde. Ve stavu příjmu dat pouze vyřízne z rámce data (dá se říci paket) a předá je vyšším vrstvám formou události. ZBCommunicaitonManager si také vede databázi připojených zařízení, ovšem pouze pro účely debbugingu.

5.3 CarManager



Obrázek 13 - Car Manager

Třída CarManager je víceméně skutečným jádrem celé aplikace, jelikož je softwarovou reprezentací fyzické platformy. Je také nejvyšší vrstvou pomyslného komunikačního zásobníku.

Tak jako reálná platforma existuje i tato třída v různých stavech (inicializace experimentu, běžící experiment, pozastavený experiment atd.). Podobně jako v případě ZBCommunicationManager je CarManager řízen výlučně událostmi. Události generované interakcí uživatele s grafickým rozhraním přepínají stav v jakém CarManager pracuje a určuje, jak reaguje na události z nižších vrstev, tedy na příchozí datové pakety. Jelikož jediná možnost jak zjistit aktuální stav, či zaručit určitý stav je komunikace s modely aut, spravuje CarManager také databázi všech aktivních modelů. K tomu využívá objekt Car.

Car je obdobně softwarovou reprezentací modelu auta. Obsahuje veškeré parametry auta a udržuje je synchronizovány s reálným modelem. Jsou zde také dočasně ukládána data ze senzorů.

Jelikož v síti Zigbee nelze manuálně přiřazovat zařízením adresy, řeší CarManager překlad ze Zigbee adres na adresy logické. V praxi jsme ovšem spolu s kolegyní pracující na komunikaci ze strany aut postupně dospěli do stavu, kdy veškerá komunikace

probíhá zásadně broadcastem a pakety jsou adresovány pouze logickými adresami. Zigbee adresy tak nakonec slouží pouze k jedinečné identifikaci zařízení v rámci sítě.

Uživateli tedy třída nabízí vše, co nabízí i fyzická platforma. Start/stop experimentu, nastavení a vyčtení parametrů jednotlivých modelů aut i parametrů ovlivňujících celý experiment, automatické zjištění pozice modelů aut a vyčtení dat ze senzorů.

Jelikož data jsou následně vykreslována v reálném čase, probíhá zde odměřování času. Tento čas je nicméně časem aplikace a liší se od času použitého k řízení modelů. Protože je tento čas odměřován v operačním systému, který není určen pro real-time operace je poměrně nepřesný. Díky tomu, že je využit pouze při vykreslování dat pro uživatele, ovšem ani přesný být nemusí. Pro přesnější časové odměření komunikace jsme s kolegyní chtěli využít timeStamp obsažený v Zigbee rámci, ovšem kvůli nedostatku dokumentace jsme nezjistili co přesně tento timeStamp reprezentuje, čímž jsme nemohli zajistit relevanci dat, a proto jsme nakonec tyto data nepoužili.

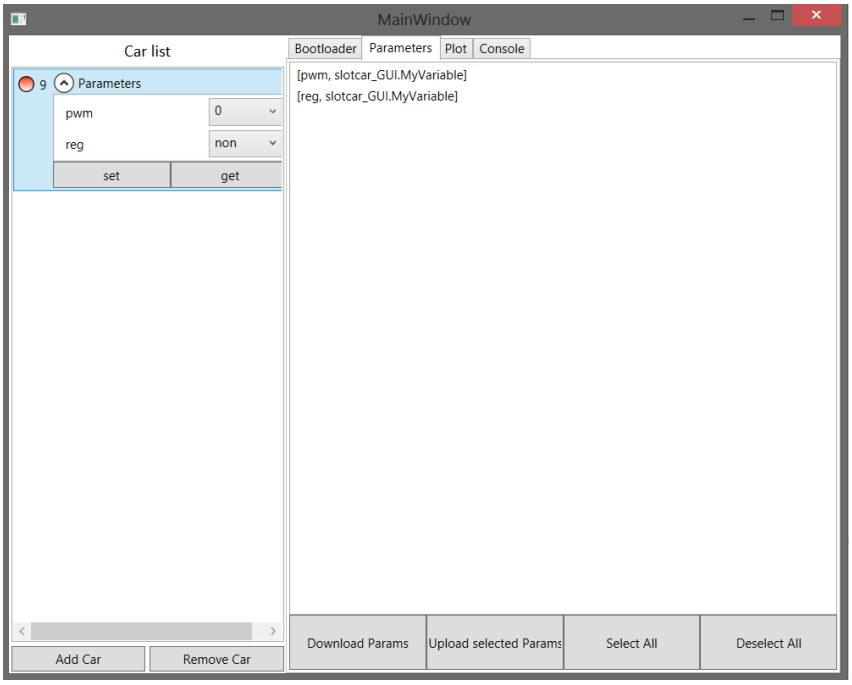
Jak již bylo řečeno, implementace celé aplikace se na PC a Android liší na nejnižších vrstvách. To nicméně platí i pro uživatelské rozhraní, jelikož Android využívá vlastních knihoven a nevyužívá standardní knihovny Javy pro tvorbu grafiky jako například Swing. Třída CarManager je naštěstí dokonale odstíněna od grafického uživatelského rozhraní a je schopna pracovat bez něj. Tím je zajištěna kompatibilita s jakýmkoliv systémem zvládajícím Javu. Je nutné pouze vytvořit vlastní GUI v souladu s možnostmi daného operačního systému. V rámci této práce jsem vytvořil plnohodnotné rozhraní pouze pro Android a pro simulaci modelů aut z PC jsem využil jednodušší verzi aplikace bez CarManageru s velice jednoduchým GUI, které sloužilo pouze pro manuální simulaci komunikace s modely.

6 GRAFICKÉ UŽIVATELSKÉ ROZHRANÍ

Grafické uživatelské rozhraní bylo zadáno následovně:

1. Rozhraní bude obsahovat seznam modelů aut v síti s jejich parametry.
2. Uživatel má možnost změnit parametry jednotlivých modelů (typ regulátoru, regulační konstanty, reference) i parametry celého experimentu (komunikační topologie, rychlost leadera, rozestupy aut).
3. Rozhraní musí být schopné vykreslovat data ze senzorů v reálném čase (pseudo-reálném) a po skončení experimentu zpětně vykreslit celý průběh.
4. Průběh experimentu bude zaznamenán do souboru *.csv.
5. Rozhraní musí být lehce modifikovatelné pro jeho využití i pro budoucí Wi-Fi komunikaci.

Jako základní inspirace posloužilo jednoduché demo vytvořené Martinem Ládem v C++ (Obrázek 14).



Obrázek 14 - Prvotní návrh GUI (autor Martin Lád)

Záměrem je vytvoření rozhraní s postranním panelem obsahujícím seznam připojeným aut a lišta umožňující výběr z několika obrazovek věnujících se různým úkonům. Jelikož jsem s vývojem aplikací pro systém Android neměl téměř žádné zkušenosti, vše jsem se učil přímo v průběhu tvoření. Tento fakt mě také nutil k poměrně dlouhému průzkumu rozličných zdrojů, díky čemuž jsem ovšem často našel několik možností, jak daný problém vyřešit. Vždy jsem se snažil zvolit takové řešení, které bylo doporučované přímo oficiálním webem Android Developers ([26]), popřípadě bylo čistější z hlediska programování i za cenu vyšší obtížnosti. Výhodou ale i nevýhodou pro vývojáře v systému Android je fakt, že se tento systém vyvíjí velice rychlým tempem. Od roku 2008 vychází nová verze systému téměř každého půl roku, přičemž každá nová verze přináší velké množství nových prvků, které často nahrazují prvky využívané v minulých verzích ([27]). I přes to, že systém obsahuje nástroje pro zajištění zpětné i dopředné

kompatibility, musí vývojáři neustále držet krok s vývojem systému a stále se učit nové věci. Často se mi stávalo, že jsem narážel na tutoriály, které již nebyly aktuální. Rozhodl jsem se aplikaci psát tak, aby byla plně kompatibilní a optimalizovaná pro verzi Android 4.4.2 KitKat.

Zajímavým prvkem při tvorbě aplikací je, způsob jakým tvůrci Androidu vynucují zvýšení bezpečnosti pro koncového uživatele aplikací. Každá aplikace, i ta, kterou vývojář nechce publikovat ve službě Google Play musí být opatřena elektronickým certifikátem neboli podpisem vývojáře. Všechna Android zařízení neumožňují instalaci necertifikovaných aplikací formou instalačních souborů. Jedinou možností jak nainstalovat nepodepsanou aplikaci je nahrát ji v režimu ladění přes USB přímo z vývojového prostředí. Pokud chce vývojář vytvořit instalační soubor, který hodlá instalovat na další zařízení, musí si vygenerovat svůj jedinečný certifikát. Certifikáty slouží k identifikaci a ověření původu aplikace. IDE Eclipse obsahuje průvodce pro vytvoření vlastních certifikátů.

Grafické rozhraní je v Androidu tvořeno tzv. widgety. Widget je element rozhraní, se kterým lze interagovat, popřípadě slouží k předání informace uživateli. Základními widgety jsou například tlačítka, popisky či posuvníky. Informace jak pracovat se základními widgety jsem čerpal hlavně z knihy Android 4, viz [28]. Kniha je sice psána pro starší verzi systému než je KitKat, ovšem základní prvky zůstávají v systému zachovány i ve vyšších verzích. Při využití pokročilejších prvků jsem již čerpal z oficiálních stránek pro developery aplikací ([26]) a také z nespočtu zdrojů udržovaných zkušenými vývojáři ve formě blogů či webových stránek. Často jsem také při hledání řešení lokálních chyb či nejasností navštívil stránku využívanou obrovským množstvím programátorů StackOverflow ([29]).

Velkou odlišností při tvorbě rozhraní v Androidu na rozdíl od čisté Javy, je možnost inicializace objektů GUI s použitím XML souborů. Tento postup není nutný, jelikož vše lze vytvořit i programově, ale je silně doporučovaný, protože tento způsob je daleko rychlejší a přehlednější. V praxi je tím také docíleno naprostého oddělení funkčního kódu od grafického rozhraní. Při startu aplikace se vytvoří všechny grafické objekty na základě XML souboru, který popisuje rozhraní a v kódu se k nim poté přistupuje na základě ID. Další výhodou je možnost překladu XML souboru v reálném čase v Eclipse. Vývojář tím zjistí, jak rozhraní bude vypadat, aniž by ho musel nahrát do přístroje.

6.1 Struktura uživatelského rozhraní

Jádrem všech Android aplikací jsou tzv. aktivity. Aktivita umožňuje uživateli určitou sadu interakcí, většinou tematicky souvisejících, přičemž je svázána s jediným oknem uživatelského rozhraní. Aplikace musí obsahovat alespoň jednu aktivitu. Jak již bylo řečeno, jádrem zvoleného grafického rozhraní má být centrální okno s volitelným obsahem, což by při využití aktivit znamenalo vytvořit novou aktivitu pro každé okno. Tento způsob byl využíván ve starších verzích systému, nicméně byl nahrazen, jelikož komunikace mezi aktivitami je složitá a musí být prováděna formou tzv. záměrů. [30]

Od verze Androidu 4.0 byly představeny fragmenty. Fragment reprezentuje část grafického rozhraní a definuje jeho chování. Fragmenty pracují v rámci rodičovské aktivity. Díky tomu, může být každá obrazovka tvořena vlastním fragmentem, přičemž rodičovská aktivita (v této aplikaci třída MainActivity) rozhoduje o tom, který fragment

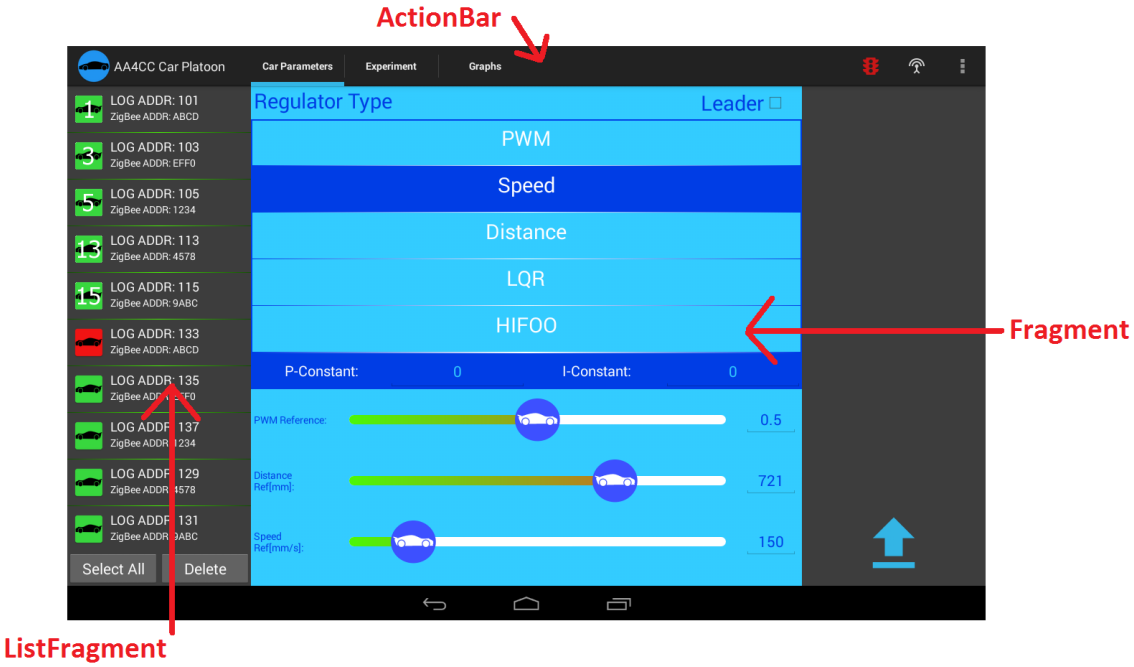
je právě aktivní. Tento způsob zobrazení je v současné době velice hojně používán v Android aplikacích (například aplikace Google Play) a tyto přepínatelné obrazovky se nazývají taby.

Jako prostředek pro použití tabů slouží tzv. ActionBar (přesněji je jejich kontejner objekt ViewPager, nicméně ActionBar obsahuje prvky pro výčet a ovládání fragmentů). ActionBar je lišta v horní části aplikace umožňující řadu funkcí (podobně jako lišta nástrojů v programech Windows). ActionBar dále obsahuje informace o celé aplikaci, umožňuje otevření postranních panelů, nápovědy nebo nastavení, také může obsahovat tlačítka. Při vytváření této části aplikace jsem se nechal inspirovat tutoriálem z webové stránky, viz [31].

Neméně důležitou částí rozhraní je postranní panel obsahující seznam připojených modelů aut. Tento seznam musí být viditelný vždy, nehledě na právě zvolený centrální tab. Pokud bychom nevyužili fragmenty, takto navržené GUI by se nedalo vůbec vytvořit, popřípadě by bylo neúnosně složité a neoptimální. K zobrazení seznamu existuje v Android specializovaný fragment ListFragment.

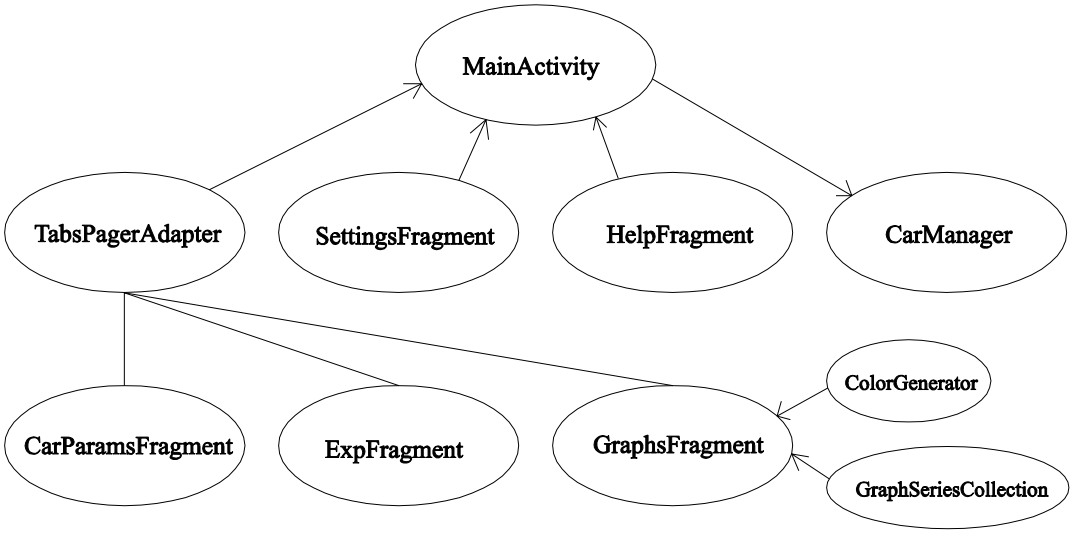
Seznam stavebních kamenů aplikace uzavírá PreferencesFragment, který nabízí služby pro vytvoření nastavení a také fragment zobrazující nápovědu.

Následující obrázek (Obrázek 15) demonstruje výše popsané prvky ve finální verzi vyvíjené aplikace. V další kapitole popíšu jednotlivé části podrobněji.



Obrázek 15 - Finální verze GUI

6.2 MODULY GRAFICKÉHO ROZHRANÍ



Obrázek 16 - Struktura grafického rozhraní

6.2.1 MainActivity

Třída MainActivity dědí od objektu FragmentActivity. Plní funkci jediné a mateřské aktivity v celé aplikaci a funguje jako centrum veškerých informačních toků aplikace mezi grafickým rozhraním a logikou správy experimentu. Inicializuje správce experimentu a bezdrátové komunikace CarManager a spravuje všechny použité fragmenty.

S fragmenty komunikuje přímo voláním jejich metod, jelikož k nim má zajištěný plný přístup. Složitost dosažení tohoto principu byla zvýšena použitím tabů, jelikož fragmenty jsou inicializovány v objektu ViewPager, který umožňuje následné přepínání obrazovek. ViewPager jsem musel implementovat v rámci vnitřní třídy MainActivity, což není úplně čisté řešení z programátorského hlediska, ovšem zaručuje přístup hlavní aplikace k referencím na fragmenty.

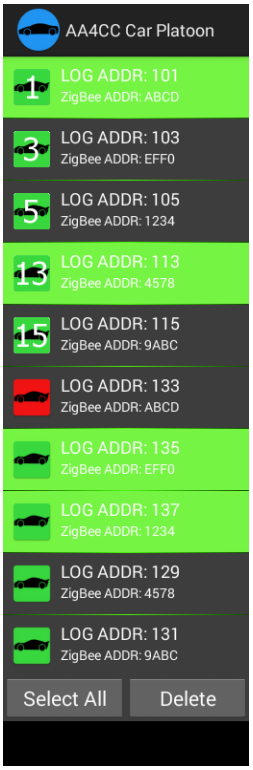
MainActivity, dále zajišťuje možnost přepínání obrazovek pomocí tzv. swipes, tedy přejetím prstu po obrazovce ze strany na stranu.

6.2.2 ActionBar

Obrázek 17 - ActionBar s rozvinutou nabídkou akcí

ActionBar je možné vygenerovat přímo při vytvoření projektu v IDE. Tato ovládací lišta obsahuje seznam tabů, mezi kterými lze přepínat kliknutím na jejich názvy. Dále obsahuje tlačítka pro inicializaci Zigbee sítě (ikona s motivem antény) a start experimentu (ikona s motivem semaforu). Pod ikonou tří čtverců, což je univerzální ikona pro rozbalovací nabídku dalších možností v systému Android, jsou schovány nabídky nápovědy a nastavení. ActionBar dále obsahuje název aplikace s její ikonou.

6.2.3 CarListFragment



Obrázek 18 – CarListFragment

Třída CarListFragment dědí od objektu ListFragment. ListFragment je klasický Fragment obsahující widget Listview, který se specializuje na vytvoření interaktivního seznamu. Nejzákladnější použití je pouhé zobrazení seznamu textových řetězců, což je velice jednoduché, na druhé straně v této aplikaci bylo nutné vytvoření pokročilejšího a hlavně líbivějšího seznamu modelů aut.

Při tvorbě tohoto fragmentu jsem vycházel z tutoriálu na této stránce [32]. Bylo nutné vytvořit XML soubor definující rozložení widgetů v samotném řádku a také objekt ArrayAdapter, který má na starosti naplnění seznamu daty.

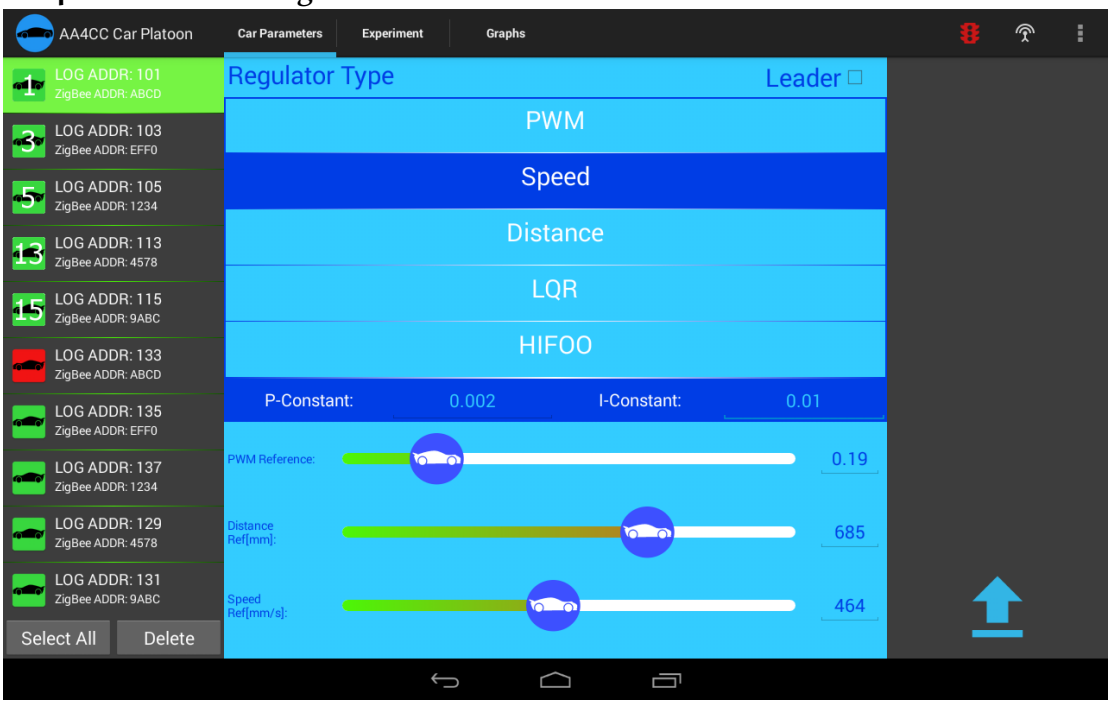
Při tvorbě této třídy jsem chtěl dosáhnout toho, aby se seznam dal ovládat univerzálním způsobem se stejně bohatými možnostmi, jako například seznam souborů v systémovém správci souborů. Cílem bylo umožnit uživateli označovat položky vylučovacím způsobem (vždy lze označit pouze jednu položku), ale i několik položek současně (pomocí tlačítka i všechny), přičemž označené položky lze ze seznamu vymazat. Mazání musí probíhat manuálně, protože pokud je auto z nějakého důvodu vyjmuto z kolony, nemá přístup k elektrickému napájení a tedy ani možnost sdílet tuto informaci v síti. Implementace těchto možností si vyžadovala vcelku hluboké pochopení principů fungování ListView, jelikož v základu je možné položky v seznamu pouze vybírat a to buď v režimu multiple choice nebo single choice, ale ne jejich kombinaci.

Jako nejtěžší z těchto úkolů se ukázalo být mazání položek ze seznamu, přesněji smazání několika položek najednou. Nikde jsem nenašel návod, jak této funkčnosti dosáhnout, který by nebyl zbytečně složitý či programátorský neoptimální. Nakonec jsem díky detailnímu krokování programu dospěl k velmi elegantnímu řešení, fungujícímu v jediném cyklu.

ListView vrací informaci o označených položkách jako mapu indexů a boolean hodnot. Faktem, na který jsem přišel až při krokování ovšem je, že se toto pole aktualizuje po každé změně, tedy i smazání prvku, aniž bych upozornil ArrayAdapter na tuto změnu, což je standardní postup. Nakonec stačilo načíst pole s označenými pozicemi pouze na začátku cyklu a mazat postupně data (která jsou indexována také pozicí v seznamu), s tím, že pokud postupujeme od vrcholu seznamu, musíme vždy daný index upravit o hodnotu doposud smazaných položek. Nakonec upozorníme ArrayAdapter na změnu dat a ten aktualizuje celý seznam.

Pokud se do sítě připojí nový model auta, CarManager za pomocí MainActivity upozorní CarList na tento fakt a přidání proběhne jednoduše novým vytvořením ArrayAdapteru, kterému dáme k dispozici aktualizovaná data.

6.2.4 CarParamsFragment



Obrázek 19 – Nabídka nastavení parametrů aut

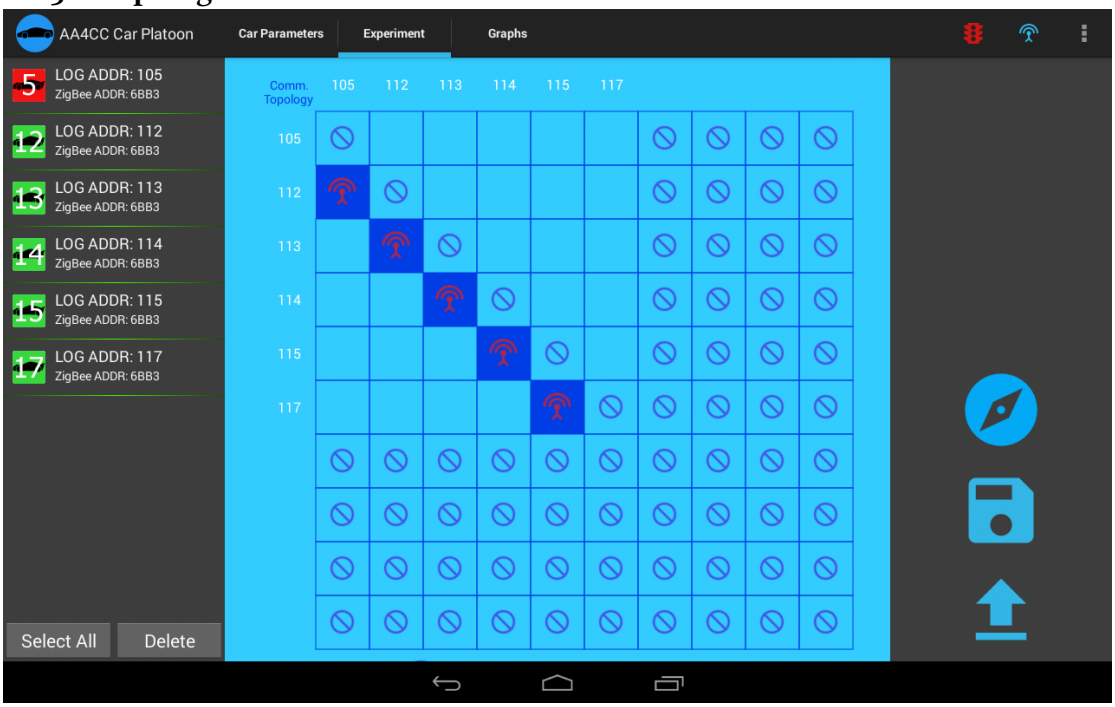
Obrazovka Car Parameters slouží k zobrazení a nastavení parametrů modelů aut. Tento fragment se skládá z většího počtu widgetů. V prvé řadě to je několik TextView, sloužících k zobrazení textových popisků například pro názvy konstant či referencí regulátorů. Seznam regulátorů je zase tvořen pomocí ListView, ovšem ve značně jednodušším provedení než seznam aut. Po kliknutí na daný regulátor se v liště konstant vypíše výběr konstant s ním souvisejících. Dalšími použitými prvky jsou widgety typu EditText. Ty slouží k zadávání hodnot uživatelem. Fragment obsahuje checkbox k indikaci toho, zda je model auta leader (tento fakt je také promítnut v barvě pozadí ikony položky v seznamu aut, kdy červená znamená, že auto je leader).

Vizuálně výraznými prvky jsou posuvníky (widget Seekbar), pomocí kterých uživatel určuje referenci jednotlivým regulátorům. Základní Seekbar je mnohem jednodušší z grafického hlediska (jedná se pouze o tenkou jednobarevnou linku s poloprůhledným kruhovým jezdcem). V rámci zvýraznění těchto prvků, jsem vytvořil vlastní vizuální motiv, přičemž jsem postupoval na základě tohoto tutoriálu [33]. Nejprve je nutné

definovat všechny vrstvy horizontálního pruhu ve zvláštním XML souboru (pozadí, tvar, překryvná vrstva atd.). Nakonec stačí vytvořit ikonu jezdce a seekbar je připraven k použití. Jelikož zadávání číselných hodnot formou posuvníku je sice vizuálně působivé ovšem značně nepřesné, doplnil jsem každý seekbar widgetem EditText. Toto pole je svázáno s daným posuvníkem a aktualizuje se v reálném čase v závislosti na poloze posuvníku. Ukazuje tedy uživateli současnou hodnotu a zároveň umožňuje její zadání manuálně, přičemž posuvník se tomuto způsobu editace okamžitě přizpůsobí. Při testování aplikace jsem narazil na nepříjemný problém a to při snaze změnit polohu posuvníku. Posuvník byl velmi neresponzivní a bylo obtížné změnit jeho polohu. To bylo očividně způsobeno tím, že stejným gestem aplikace přepíná mezi obrazovkami. Potřeboval jsem tedy nějakým způsobem zvýšit prioritu posuvníku. To jsem vyřešil vytvořením vlastního OnTouchListeneru, který využívá metody requestDisallowInterceptTouchEvent. Tuto metodu lze využít u všech interaktivních widgetů. V praxi funguje tak, že při zaznamenání doteku v oblasti Seekbaru je ViewPageru zamezeno přijímání dotekových událostí.

Nakonec fragment obsahuje tlačítko s ikonou často využívanou pro upload, které slouží k odeslání nastavených parametrů označeným autům v seznamu. Tento fragment má přístup k označeným prvkům seznamu prostřednictvím MainActivity se kterou komunikuje generováním událostí, která například událost o nastavení parametrů předává třídě CarManager, která ji následně zpracuje.

6.2.5 ExpFragment



Obrázek 20 – Nastavení parametrů experimentu

Tento fragment má podobnou strukturu jako fragment s parametry jednotlivých modelů, ovšem věnuje se nastavení parametrů, které souvisejí s celým experimentem. Jmenovitě to je rychlost leadera, rozestupy mezi modely a komunikační topologie. Obsahuje také tlačítka pro upload konfigurace do aut, uložení hodnot ze senzorů do souboru a start procedury zjišťující pozice modelů v koloně.

Centrálním prvkem je matice reprezentující komunikační topologii formou matice sousednosti. Celá matice je tvořena widgety ImageView v TableLayout, což je kontejner pro zobrazení prvků v tabulce. V současné době bohužel vytvoření libovolné komunikační topologie není možné, protože ještě není podporováno modely aut. Nyní se využívají pouze dvě topologie. Pokud uživatel neprovedl proceduru zjištění pozic aut, všechna auta se řídí pouze leaderem. Pokud jsou již pozice aut známé, řídí se každý model předcházejícím členem v koloně, kromě leadera, který se samozřejmě řídí nezávisle.

Pod maticí jsou dva posuvníky, pomocí kterých uživatel může měnit rychlost leadera a rozestupy mezi auty. K těmto posuvníkům se uživatel dostane, pokud gestem posune obrazovku dolů. Zde zase musela být využita metoda zvýšení priority Seekbaru při dotyku.

Tlačítko s ikonou uploadu funguje identicky jako v CarParamsFragment. Tlačítko s ikonou kompasu slouží ke spuštění procedury zjištění polohy modelů aut. Pokud modely aut obdrží paket požadující start této procedury, modely postupně jeden po druhém ujedou krátkou vzdálenost a na základě hodnot ze senzorů každé auto určí vlastní polohu v rámci celé kolony. Tuto informaci následně posílají koordinátorovi, přičemž v aplikaci se aktualizuje seznam s připojenými auty (položky jsou seřazeny v závislosti na poloze v koloně).

Posledním interaktivním prvkem v tomto fragmentu je tlačítko s ikonou diskety. Pokud uživatel stiskne toto tlačítko, uloží se data z průběhu celého experimentu všech vybraných aut do souboru ve formátu csv (comma separated values). Tento soubor lze s výhodou lehce zpracovat například v Matlabu.

6.2.6 GraphsFragment

Posledním a dá se říci nejdůležitějším modulem GUI je obrazovka umožňující vykreslování dat ze senzorů (Obrázek 24). Pravá strana této obrazovky je vyplněna seznamem měřených veličin a servisními tlačítky k ovládání vykreslování v reálném čase. Seznam měřených veličin je zase tvořen pomocí ListView, ve zjednodušeném provedení na rozdíl od seznamu aut. Tlačítko s ikonou často využívanou pro pauzu umožňuje pozastavení vykreslování grafu a také opětovné spuštění. Tlačítko s ikonou zámku, který je buď odemčený či zamčený přepíná mezi módy vykreslování. Pokud je zamčený, uživatel na časové ose vidí vždy fixní okno pěti sekund, dokud vykreslovaná data nedosáhnou pravé hrany obrazovky, kdy se zobrazí okno s následujícími pěti sekundami. Pokud je zámek odemčený časové okno pěti sekund se plynule posouvá s přibývajícím časem.

Centrálním prvkem této obrazovky je prvek sloužící k vykreslování dat. Při hledání zdrojů, jak tuto funkčnost implementovat, jsem si byl vědom toho, že je tento prvek často v Android aplikacích používán a určitě bude k dispozici již hotová knihovna umožňující vykreslování grafů. Bohužel oficiální knihovny nic podobného neobsahují, a proto jsem začal pátrat po open-source knihovnách. A opravdu existuje několik těchto projektů věnujících se tvorbě vývojářsky přívětivých knihoven pro vykreslování grafů. Jsou to například knihovny Android Plot ([34]), AChartEngine ([35]) a GraphView ([36]). Nakonec jsem zvolil GraphView, jelikož dle mého názoru nabízí nejlepší poměr možností a složitosti.

Open-source knihovna GraphView je neustále vyvíjena a v současné době je k dispozici ve verzi 4.0.1. Obsahuje celou řadu nástrojů pro vykreslování grafů prokládaných úsečkami, či sloupcových grafů, a to jak v reálném čase, tak vykreslení celé série dat najednou. Nabízí opravdu širokou škálu vizualizace. Umožňuje měnit barvy pozadí, barvy průběhů, styl čar, vygenerování legendy a popisků os, volbu různých typů gridů a mimo to také zoom a posuv v obou osách pomocí gest. Použití knihovny je opravdu velice intuitivní a jednoduché viz [37]. Jedinou obtížnější pasáží bylo určení toho, jaké možnosti bude mít uživatel k dispozici a jak je implementovat s využitím GraphView.

Finální verze umožňuje uživateli před začátkem experimentu zvolit v seznamu aut, od kterých modelů budou data vykreslována a následně v seznamu veličin zvolit typy senzorů. Pokud chce výběr změnit v průběhu experimentu, musí nejprve pozastavit vykreslování k tomu určeným tlačítkem, změnit parametry a vykreslování zase spustit. Počet a typ vykreslovaných linií, je tedy velice variabilní a velice těžko by se výběr dal zpracovat nějakým automatizovaným způsobem formou cyklů. Zvolil jsem tedy cestu spíše hrubou silou a vytvořil třídu GraphSeriesCollection spravující všechny datové sady (i ty zrovna neaktivní) svázanou s jediným objektem typu Car. Kód vykonávající hlavní práci je metoda updateSeries, která na základě daných parametrů aktualizuje datové sady a mimo jiné umožňuje i zpětně vykreslení kompletního průběhu experimentu, protože pokud by se v paměti udržovaly všechny doposud vykreslené body při vykreslování v reálném čase, bylo by to velice náročné na hardware tabletu a aplikace by byla neresponzivní nebo by rovnou havarovala. Na tyto problémy jsem ostatně narážel i při testování a vykreslování grafů v reálném čase si vyžádalo věnovat se i optimalizaci.

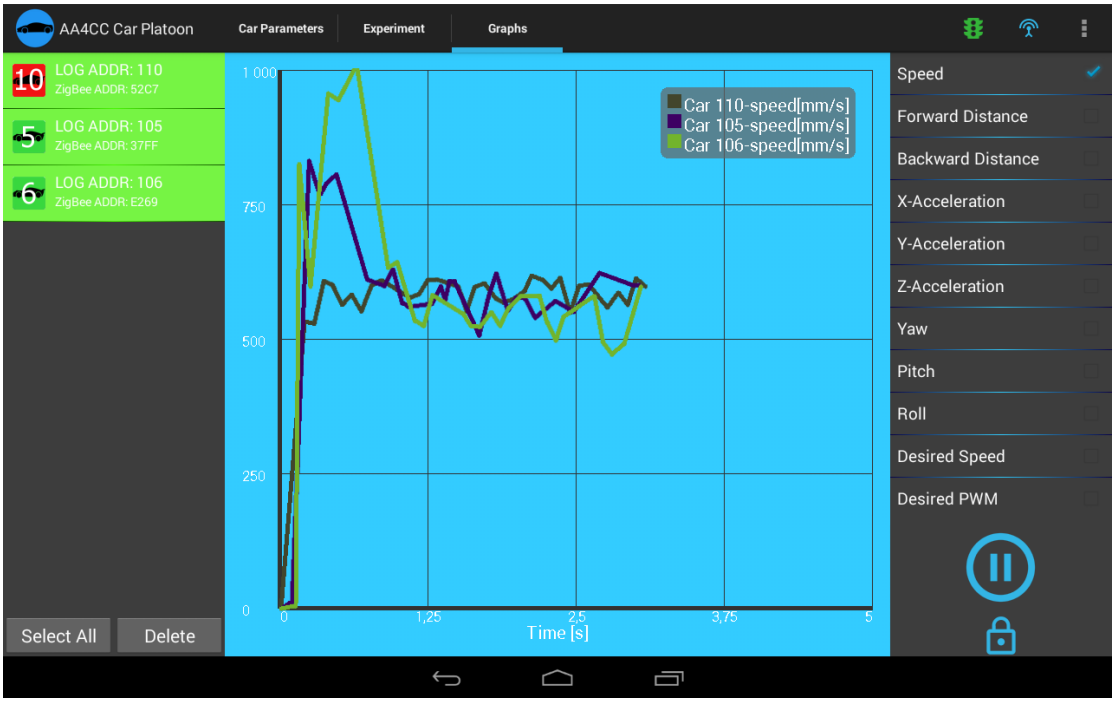
GraphView naštěstí umožňuje nastavit maximální počet vykreslených bodů, které budou zůstávat v paměti. Pokud by data chodila s periodou například 20 ms, tak při vykreslování pěti sekundového okna a pouze jediné sady dat, tak na obrazovce může být v jeden moment až 250 bodů, mezi nimiž se ještě musí vykreslit úsečky. V dokumentaci ke knihovně GraphView je doporučený maximální počet bodů při real-time vykreslování 500, což by při této frekvenci vykreslování spolehlivě umožňovalo vykreslení pouze dvou veličin současně. Tento problém se nejčastěji řeší převzorkováním příchozích dat, nicméně nakonec tato procedura ani nebyla nutná, jelikož auta odesílají data s periodou T, viz (1).

$$T = (\text{počet aut} + 1) \cdot 20\text{ms} \qquad (1)$$

Pokud budeme provádět experiment s minimálním počtem modelů (2), bude perioda 60 ms, což znamená 83 bodů na obrazovku (okno 5 sekund) na jednu veličinu. Pro větší počet aut, již bude počet vždy nižší. Velikost zásobníku byla nakonec zvolena na hodnotu 80 bodů, což bohatě postačuje k tomu, aby bylo vykreslování kontinuální a uživatel si nevšiml postupného odstraňování starých hodnot. Co se týče maximálního výkonu v této konfiguraci, tak tablet bez problémů zvládá současné vykreslení až šesti veličin (to odpovídá cca 480 bodům). Uživatel sice má možnost vykreslit veličin i více, což nedoporučuji z hlediska plynulosti aplikace, ale v praxi je navíc současné vykreslování více než čtyř veličin i zbytečné, jelikož se graf stává nepřehledným.

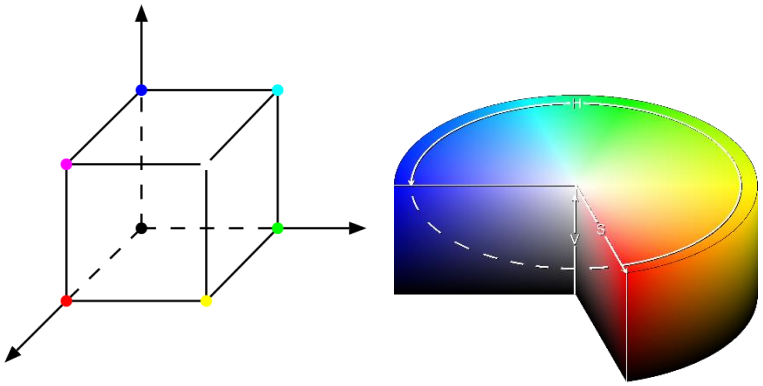
Dalším zajímavým problémem se ukázal být způsob jakým od sebe odlišit jednotlivé čáry v grafu. I přesto, že nedoporučuji vykreslování více než 6 veličin, uživatel tuto možnost má k dispozici. Vezmeme-li v úvahu, že každé auto měří 11 veličin a experimentu se může teoreticky účastnit například 20 aut, dělá to dohromady 220 průběhů, které je

nutné odlišit. Nejjednodušším řešením je odlišení barvou, jelikož barva se na první pohled zdá být i lehce generovatelná automaticky. Teoreticky stačí náhodně generovat složky RGB a pokaždé dostaneme novou barvu. To ovšem není úplně pravda. Za prvé při tomto způsobu generování barev budou vznikat i barvy jako například odstíny hnědé, černé či šedé, což není příliš vizuálně působivé. Za druhé, pokud se vygenerují dvě barvy lišící se například pouze v červené složce a ještě s malým rozdílem, uživatel bude mít problém tyto dvě barvy odlišit, viz výsledek při použití tohoto naivního přístupu na obrázku 21.



Obrázek 21 - Barvy generované náhodnými hodnotami RGB

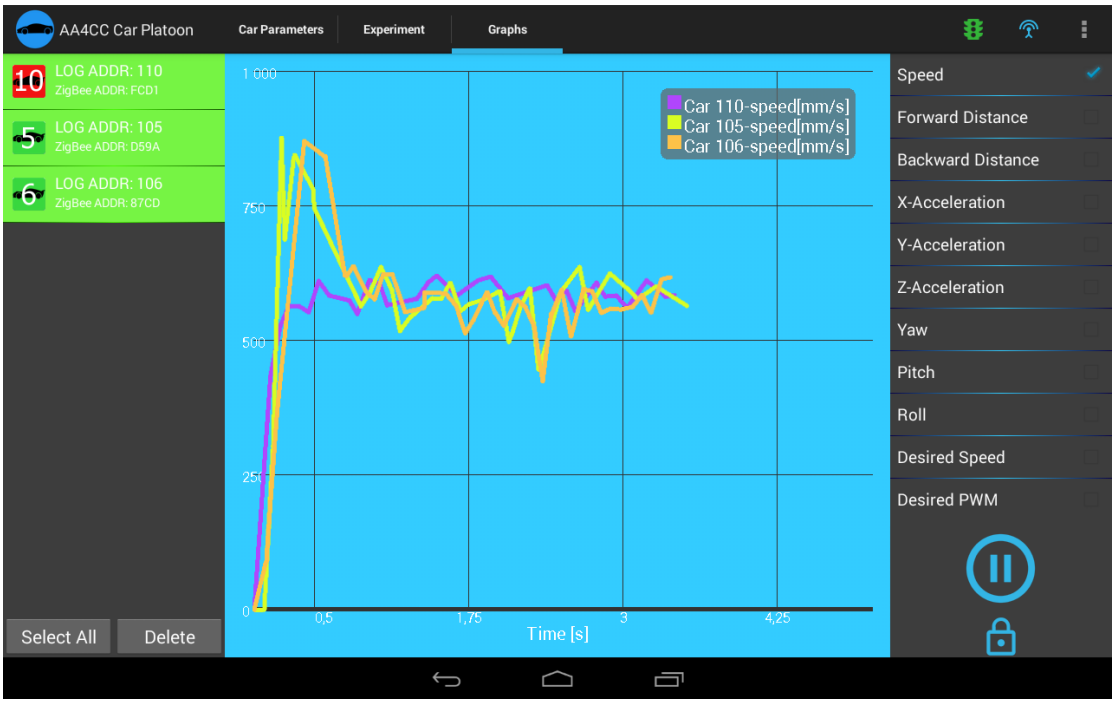
Při hledání informací na toto téma jsem zjistil, že náhodné generování líbivých barev opravdu není úplně triviální záležitostí. V prvé řadě se k tomuto účelu využívá místo barevné stupnice RGB (red-green-blue), která je kartézského typu, stupnice HSV (hue-saturation-value) válcového typu viz Obrázek 22.



Obrázek 22 - RGB vs. HSV stupnice [38]

První možností je zafixovat hodnoty value a saturation a randomizovat hodnotu hue, tedy měnit pouze odstín barvy. Tuto metodu jsem ještě lehce upravil tím, že náhodně generuji i hodnoty saturace, ovšem pouze v rámci omezeného intervalu, čímž dosáhnu

ještě větší nahodilosti. Na obrázku 23 je výsledek při použití saturace v intervalu 70-90% a value 99%.

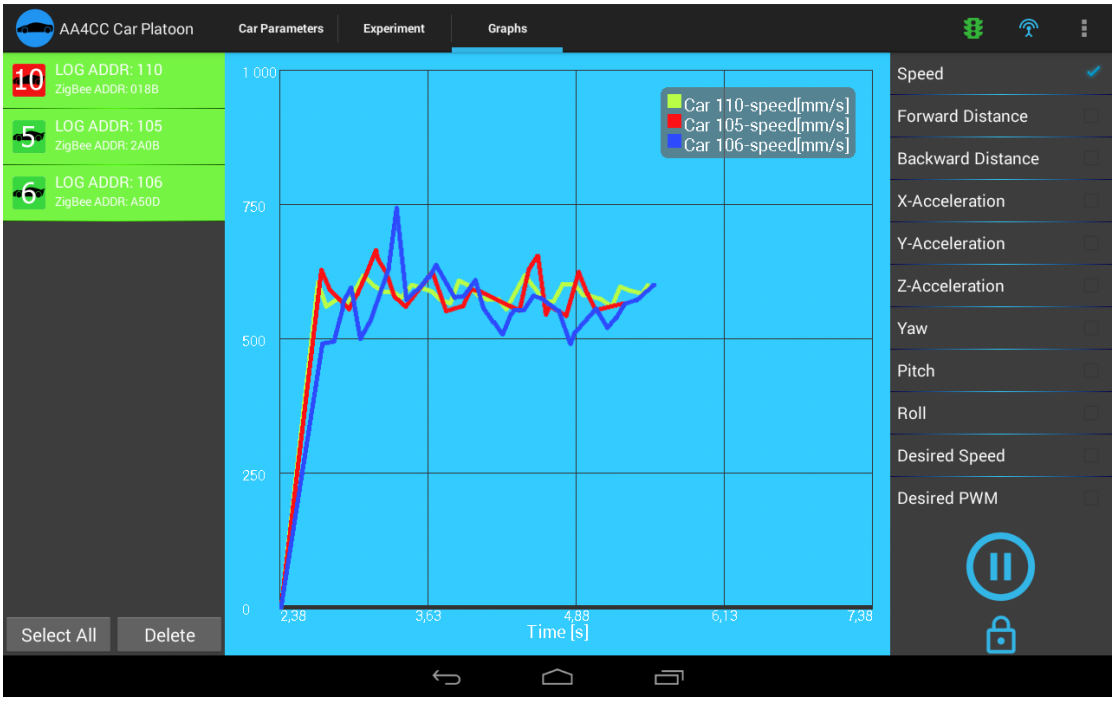


Obrázek 23 - Generování barev s využitím stupnice HSV

Poslední mnou vyzkoušený způsob je generování barev s využitím určité heuristiky. Touto heuristikou je využití konstanty φ neboli zlatého řezu (2). Rozestupy mezi odstíny jednotlivých barev vypočítáme podle vzorce (3). Tím je zajištěno velice rovnoměrné rozvržení barev přes celé spektrum. Výsledek se stejnými hodnotami saturation a value jako v předchozím příkladu je vidět na obrázku 24.

$$\varphi = \frac{1+\sqrt{2}}{5} \qquad (2)$$

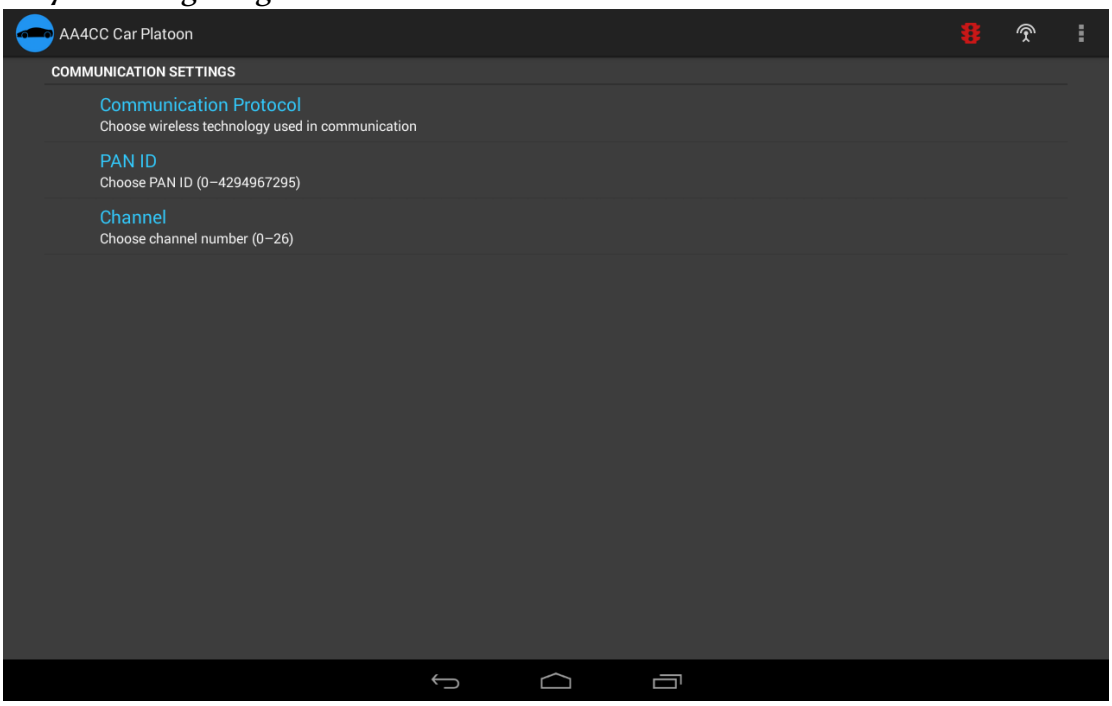
$$\text{hue} = \text{randomHue} + \varphi * \left(\frac{\text{colorPosition}}{(5 * \text{rand}(0-1))}\right) \qquad (3)$$



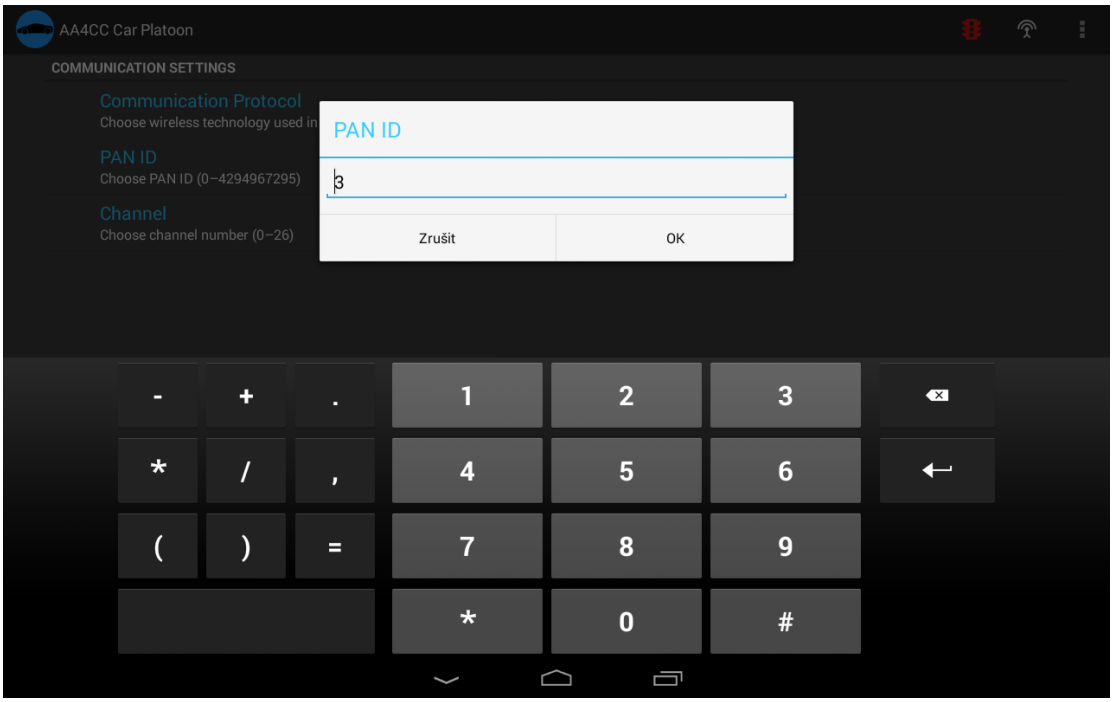
Obrázek 24 - Generování barev s využitím konstanty φ

Shrnutí výše zmíněných přístupů i s příklady je hezky zachyceno na následující stránce [38].

6.2.7 SettingsFragment



Obrázek 25 - Nabídka nastavení aplikace



Obrázek 26 - Rozvinutá nabídka nastavení PAN ID

Tento fragment slouží k zobrazení nabídky nastavení aplikace. Ke tvorbě nastavení Android obsahuje řadu specializovaných nástrojů, v tomto případě dědění od objektu PreferencesFragment. Při vyvolání nabídky uživatelem hlavní aktivita překryje celou obrazovku tímto fragmentem. S vhodným využitím tzv. backstacku, se lze z nastavení vrátit kliknutím na systémové tlačítko Zpět s tím, že vše zůstane zachováno tak, jak to uživatel opustil.

SettingsFragment umožňuje výběr použité bezdrátové komunikace Zigbee nebo Wi-Fi (bude podporováno v budoucnu) a dále parametry této sítě (prozatím pouze číslo kanálu a PANID). Při výběru jednoho z parametrů se otevře dialog s klávesnicí se znakovou sadou umožňující zadání pouze validních hodnot. Protože byl využit fragment přímo navržený pro tvorbu nastavení, ukládají se tyto hodnoty automaticky do proměnné SharedPreferences, která uchovává toto nastavení i po ukončení aplikace a ze které je nastavení načítáno při opětovném startu.

6.2.8 HelpFragment

Tento fragment obsahuje pouze jediný widget typu TextView a jeho úkolem je načíst text s nápovědou ze souboru. V současné době nápověda obsahuje kopii textu nápovědy z přílohy D k této práci. Z nápovědy se lze zase s využitím backstacku vrátit systémovým tlačítkem Zpět.

7 ZÁVĚR

V rámci této bakalářské práce jsem úspěšně splnil všechny hlavní body zadání. Zprovoznil jsem funkční USB komunikaci mezi tabletem s OS Android a donglem TI CC2531, bez nutnosti rootu zařízení. Následně jsem úspěšně nakonfiguroval komunikační čip tak, aby vytvořil funkční Zigbee rádiovou síť a byl po ní schopen komunikace s modely aut. K účelu simulace aut, při vývoji jsem také vytvořil obdobnou aplikaci na PC, kdy jediný vysílač je schopen simulovat několik modelů aut současně. Finálním krokem bylo vytvoření aplikace pro OS Android, umožňující uživateli spravovat testovací platformu distribuovaného řízení kolon s využitím bezdrátové komunikace. Aplikace obsahuje grafické uživatelské rozhraní vhodné pro pohodlnou parametrizaci celého experimentu, vizualizaci v reálném čase a export naměřených hodnot do souboru. Aplikace je vhodná i pro prezentační účely celé platformy.

Pokud budu mít v budoucnu možnost na vývoji dále pokračovat, mám v plánu rozšířit aplikaci o možnost komunikace přes Wi-Fi, jelikož nejnovější verze modelů aut, která je právě ve vývoji, bude postavena právě na této technologii. Mimo komunikaci, bych chtěl ještě rozšířit grafické rozhraní o pár drobností, jako například možnost serializace nastavení a jeho načtení ze souboru, či panel shrnující informace o stavu celé platformy.

Při vývoji jsem se naučil nejenom velké množství informací například o sériové komunikaci, technologii Zigbee a USB, ale hlavně získal zkušenosti s vývojem aplikací pro OS Android. Vývoj pro tento operační systém, ale i technologie Zigbee, jsou velmi moderní trendy, z čehož plyne, že zkušenosti s nimi, bych mohl s velkou pravděpodobností uplatnit ve své budoucí kariéře.

8 ZDROJE

[1] Stránka projektu Slotcar Platooning, navštíveno 13. 5. 2015, https://support.dce.felk.cvut.cz/mediawiki/index.php/Slotcar_platooning

[2] CC2531 USB Hardware User's Guide, navštíveno 13. 5. 2015, http://www.ti.com/lit/pdf/swru221

[3] CC253x System-on-chip Solution for 2.4-GHz IEEE 802.15.4 and Zigbee Applications, navštíveno 13. 5. 2015, http://www.ti.com/lit/pdf/swru191

[4] Z-Stack, navštíveno 13. 5. 2015, http://www.ti.com/tool/z-stack

[5] CC2530ZNP Interface Specification, Z-Stack API Documentation PDF download

[6] Universal Serial Bus Wiki, navštíveno 13. 5. 2015, http://en.wikipedia.org/wiki/USB

[7] Materiály k předmětu Mikrokontroléry, Teplý T., navštíveno 13. 5. 2015, http://moodle.kme.fel.cvut.cz/moodle/mod/resource/view.php?id=13523

[8] Materiály k předmětu Distribuované systémy a počítačové sítě, Novák J., navštíveno 13. 5. 2015, http://measure.feld.cvut.cz/system/files/files/cs/vyuka/predmety/A3B38DSY/12_USB_cz.pdf

[9] Beyond Logic – USB in a NutShell, Peacock Craig, navštíveno 13. 5. 2015, http://www.beyondlogic.org/usbnutshell/usb1.shtml

[10] ARM KEIL Microcontroller Tools, navštíveno 13. 5. 2015, http://www.keil.com/pack/doc/mw/USB/html/_c_d_c.html

[11] USB Host API, navštíveno 13. 5. 2015, http://developer.android.com/guide/topics/connectivity/usb/host.html

[12] usb-serial-for-android, Mike Wakerly, navštíveno 13. 5. 2015, https://github.com/mik3y/usb-serial-for-android

[13] Drew Gislason, Zigbee Wireless Networking, Newnes 2008

[14] Universal Serial Bus Communications Class Subclass Specification for PSTN Devices, navštíveno 13. 5. 2015, http://www.usb.org/developers/docs/devclass_docs/CDC1.2_WMC1.1_012011.zip

[15] Zigbee Wiki, navštíveno 13. 5. 2015, http://en.wikipedia.org/wiki/ZigBee

[16] IEEE 802.15.4 Wiki, navštíveno 13. 5. 2015, http://en.wikipedia.org/wiki/IEEE_802.15.4

[17] Google Driverless Car, navštíveno 13. 5. 2015, http://en.wikipedia.org/wiki/Google_driverless_car

[18] Slotcar Platoon System Description, navštíveno 13. 5. 2015, https://support.dce.felk.cvut.cz/mediawiki/index.php/Slotcar_Platoon_System_Description

[19] Java Communications API, navštíveno 13. 5. 2015, http://www.oracle.com/technetwork/java/index-jsp-141752.html

[20] RXTX library, navštíveno 13. 5. 2015, http://rxtx.qbang.org/wiki/index.php/Main_Page

[21] Developing a ZigBee® System Using a CC2530-ZNP Approach, navštíveno 13. 5. 2015, http://www.ti.com.cn/cn/lit/an/swra444/swra444.pdf

[22] Z-Stack Monitoring and Test API, navštíveno 13. 5. 2015, Z-Stack Monitoring and Test API PDF download

[23] Android USB Host - bulkTransfer() is losing data, navštíveno 13. 5. 2015, http://stackoverflow.com/questions/9108548/android-usb-host-bulktransfer-is-losing-data

[24] Enabling developers mode in Android Kit-Kat, Puthur Plato, navštíveno 13. 5. 2015, http://www.technewscentral.co.uk/how-to-enable-usb-debugging-and-developer-options-in-android-4-2-and-higher-android-4-2android-4-3android-4-4/id_7250

[25] Texas Instruments Forum, CC2531 and SYS_RESET_REQ, navštíveno 6. 5. 2015, http://e2e.ti.com/support/wireless_connectivity/f/158/p/47640/168999

[26] Android Developers, navštíveno 13. 5. 2015, http://developer.android.com/index.html

[27] Android Versions History, navštíveno 13. 5. 2015, http://en.wikipedia.org/wiki/Android_version_history

[28] Grant Allen, Android 4 - Průvodce programováním mobilních aplikací, Computer Press Brno, 2013

[29] StackOverflow, navštíveno 13. 5. 2015, http://stackoverflow.com/

[30] Android Developers Activity, navštíveno 7. 5. 2015, http://developer.android.com/reference/android/app/Activity.html

[31] Android TabLayout with Swipeable Views Tutorial, Tamada Ravi, navštíveno 7. 5. 2015, http://www.androidhive.info/2013/10/android-tab-layout-with-swipeable-views-1/

[32] Using Lists in Android Tutorial, navštíveno 8. 5. 2015, http://www.vogella.com/tutorials/AndroidListView/article.html

[33] Custom Seekbar Tutorial, navštíveno 8. 5. 2015, http://www.mokasocial.com/2011/02/create-a-custom-styled-ui-slider-seekbar-in-android/

[34] Android Plot, navštíveno 9. 5. 2015, http://androidplot.com/

[35] AChartEngine, navštíveno 9. 5. 2015, http://www.achartengine.org/

[36] GraphView, Gehring Jonas, navštíveno 9. 5. 2015, http://www.android-graphview.org/

[37] GraphView Basic Tutorial, Gehring Jonas, navštíveno 9. 5. 2015, http://www.android-graphview.org/documentation/how-to-create-a-simple-graph

[38] How to generate random colors programatically, Ankerl Martin, navštíveno 9. 5. 2015, http://martin.ankerl.com/2009/12/09/how-to-create-random-colors-programmatically/

[39] Android Asset Studio, Nurik Roman, navštíveno 12. 5. 2015, http://romannurik.github.io/AndroidAssetStudio/index.html

PŘÍLOHY

A. USB Deskriptor TI CC2531

TI CC2531 Low-Power RF to USB CDC Serial Port (COM6)

Connection Status	Device connected
Current Configuration	1
Speed	Full (12 Mbit/s)
Device Address	2
Number Of Open Pipes	3

Device Descriptor TI CC2531 USB CDC

Offset	Field	Size	Value	Description
0	bLength	1	12h	
1	bDescriptorType	1	01h	Device
2	bcdUSB	2	0200h	USB Spec 2.0
4	bDeviceClass	1	02h	CDC Control
5	bDeviceSubClass	1	00h	
6	bDeviceProtocol	1	00h	
7	bMaxPacketSize0	1	20h	32 bytes
8	idVendor	2	0451h	
10	idProduct	2	16A8h	
12	bcdDevice	2	0009h	0.09
14	iManufacturer	1	01h	"Texas Instruments"
15	iProduct	1	02h	"TI CC2531 USB CDC"
16	iSerialNumber	1	03h	"__0X00124B0002F25BE3"
17	bNumConfigurations	1	01h	

Configuration Descriptor 1 Bus Powered, 50 mA

Offset	Field	Size	Value	Description
0	bLength	1	09h	
1	bDescriptorType	1	02h	Configuration
2	wTotalLength	2	0043h	
4	bNumInterfaces	1	02h	
5	bConfigurationValue	1	01h	
6	iConfiguration	1	00h	
7	bmAttributes	1	80h	Bus Powered
	4..0: Reserved		...00000	
	5: Remote Wakeup		..0.....	No
	6: Self Powered		.0......	No, Bus Powered
	7: Reserved (set to one) (bus-powered for 1.0)		1.......	
8	bMaxPower	1	19h	50 mA

Interface Descriptor 0/0 CDC Control, 1 Endpoint

Offset	Field	Size	Value	Description
0	bLength	1	09h	
1	bDescriptorType	1	04h	Interface
2	bInterfaceNumber	1	00h	
3	bAlternateSetting	1	00h	
4	bNumEndpoints	1	01h	
5	bInterfaceClass	1	02h	CDC Control
6	bInterfaceSubClass	1	02h	Abstract Control Model
7	bInterfaceProtocol	1	01h	AT Commands: V.250 etc
8	iInterface	1	00h	

Header Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	05h	
1	bDescriptorType	1	24h	CS Interface
2	bDescriptorSubtype	1	00h	Header
3	bcdCDC	2	0110h	1.10

Union Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	05h	
1	bDescriptorType	1	24h	CS Interface
2	bDescriptorSubtype	1	06h	Union
3	bControlInterface	1	00h	
4	bSubordinateInterface0	1	01h	CDC Data

Call Management Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	05h	
1	bDescriptorType	1	24h	CS Interface
2	bDescriptorSubtype	1	01h	Call Management
3	bmCapabilities	1	00h	
	7..2: Reserved		000000..	
	1: Data Ifc Usage		......0.	Call management only over Comm Ifc
	0: Call Management		0	Does not handle call management itself
4	bDataInterface	1	01h	

Endpoint Descriptor 82 2 In, Interrupt, 64 ms

Offset	Field	Size	Value	Description
0	bLength	1	07h	
1	bDescriptorType	1	05h	Endpoint
2	bEndpointAddress	1	82h	2 In
3	bmAttributes	1	03h	Interrupt
	1..0: Transfer Type		11	Interrupt
	7..2: Reserved		000000..	
4	wMaxPacketSize	2	0040h	64 bytes
6	bInterval	1	40h	64 ms

Interface Descriptor 1/0 CDC Data, 2 Endpoints

Offset	Field	Size	Value	Description
0	bLength	1	09h	
1	bDescriptorType	1	04h	Interface
2	bInterfaceNumber	1	01h	
3	bAlternateSetting	1	00h	
4	bNumEndpoints	1	02h	
5	bInterfaceClass	1	0Ah	CDC Data
6	bInterfaceSubClass	1	00h	
7	bInterfaceProtocol	1	00h	
8	iInterface	1	00h	

Endpoint Descriptor 84 4 In, Bulk, 64 bytes

Offset	Field	Size	Value	Description
0	bLength	1	07h	
1	bDescriptorType	1	05h	Endpoint
2	bEndpointAddress	1	84h	4 In
3	bmAttributes	1	02h	Bulk
	1..0: Transfer Type		10	Bulk
	7..2: Reserved		000000..	
4	wMaxPacketSize	2	0040h	64 bytes
6	bInterval	1	00h	

Endpoint Descriptor 04 4 Out, Bulk, 64 bytes

Offset	Field	Size	Value	Description
0	bLength	1	07h	
1	bDescriptorType	1	05h	Endpoint
2	bEndpointAddress	1	04h	4 Out
3	bmAttributes	1	02h	Bulk
	1..0: Transfer Type		10	Bulk
	7..2: Reserved		000000..	
4	wMaxPacketSize	2	0040h	64 bytes
6	bInterval	1	00h	

Abstract Control Management Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	04h	
1	bDescriptorType	1	24h	CS Interface
2	bDescriptorSubtype	1	02h	Abstract Control Management
3	bmCapabilities	1	02h	
	7..4: Reserved		0000....	
	3: Connection		....0...	
	2: Send Break		.....0..	
	1: Line Coding		......1.	Line Coding requests and Serial State notification supported
	0: Comm Features		0	

B. Instalace RXTX knihovny do IDE Netbeans

V současné době je na oficiálních stránkách projektu RXTX ([20]) k dispozici ke stažení verze pouze pro 32-bitové systémy. Build pro 64-bitové systémy je k dispozici na této stránce (http://fizzed.com/oss/rxtx-for-java). Stažený archiv je nutné rozbalit a zkopírovat soubor RXTXcomm.jar do složky <JAVA_HOME>\jre\lib\ext kde <JAVA_HOME> je instalační složka obsahující Javu a následně soubory rxtxSerial.dll popřípadě i rxtxParallel.dll do složky <JAVA_HOME>\jre\bin. Pozor na správnou verzi jre jelikož v systému může být nainstalováno několik verzí Javy. Správná verze je ta, kterou využívá IDE Netbeans.

Nakonec je nutné v projektu, který hodlá využívat tuto knihovnu nastavit následující parametry. Zvolit Properties – Libraries – Add Jar a vybrat cestu k RXTXcomm.jar. V Properties – Run – VM Options přidat cestu ke složce jre ve tvaru:

-Djava.library.path="<JAVA_HOME>\jre\bin "

Nyní je možné využívat všechny možnosti knihovny. Ukázkové příklady jsou k dispozici na oficiálním webu RXTX ([20]).

C. Zprovoznění ADB s využitím Wi-Fi

ADB neboli Android Debugging Bridge je sada nástrojů sloužící k debuggingu aplikace. ADB umožňuje ladění aplikace, ve vývojových prostředích jako Eclipse, či Android Studio, popřípadě jej lze používat přímo v příkazovém řádku. Při využití výše zmíněných vývojových prostředí většinou není potřeba ADB zvlášť instalovat, jelikož je v nich integrovaný.

Ze všeho nejdříve je nutné nainstalovat PC drivery dodané od výrobce tabletu a připojit jej pomocí USB kabelu k počítači. V tabletu je nutné umožnit ladění přes USB. V nově koupených tabletech je tato možnost většinou skryta a je nutno ji aktivovat sedmi poklepáními na položku Build Number v informacích o zařízení. [24]

V příkazovém řádku na PC se poté uživatel musí dostat do složky s vývojovými nástroji pro Android, jmenovitě <Android tools home>\sdk\platform-tools. Se stále připojeným zařízením se příkazem adb tcpip 5555 přepne ADB do bezdrátového módu. Nyní je nutné tablet fyzicky odpojit od počítače. Tablet se nyní musí připojit k Wi-Fi síti pokud tak již uživatel neučinil. Jedinou podmínkou je, že počítač i tablet musí být připojen ke stejnému access pointu. Teď již stačí pouze zadat příkaz adb connect ip.addrr.tabletu a zařízení je úspěšně připojeno k počítači. Pokud v Eclipse v perspektivě DDMS uživatel vybere připojený tablet, získává veškeré funkce ADB v Eclipse. V poli logcat lze číst logy celého systému i aktuálně vyvíjené aplikace a klasickým použitím breakpointů lze aplikaci krokovat. Při krokování lze také číst hodnoty všech proměnných a objektů. Aplikace je automaticky uploadována a instalována do tabletu při spuštění z vývojového prostředí.

D. Nápověda pro používání aplikace

Dongle TI CC2531 lze k tabletu připojit buď před spuštěním aplikace a v automaticky vygenerované nabídce zvolit aplikaci CC2531 GUI, nebo nejprve tuto aplikaci otevřít a až poté připojit dongle (v tomto případě se při vytváření sítě vygeneruje žádost o udělení práv). **Ať už je zvolen jakýkoliv způsob ihned po připojení donglu je nutné stisknout hardwarové tlačítko SP2 (blíže k USB konektoru) na samotném donglu.**

Po spuštění aplikace lze změnit nastavení bezdrátové sítě, v nabídce Settings ve skryté nabídce ActionBaru (ikona tří čtverců v pravém horním rohu). Mezi obrazovkami se lze přepínat swipy a také kliknutím na název tabu na horní liště.

Síť se vytvoří stiskem tlačítka ActionBaru s ikonou antény. Síť je úspěšně vytvořena, pokud toto tlačítko zmodrá a na obrazovce se objeví hláška o úspěšném vytvoření sítě. Nyní lze spouštět modely aut, které se automaticky budou připojovat do vytvořené sítě. Objeví-li se v seznamu aut tlačítka Select All a Delete, ovšem v seznamu nejsou vidět žádné položky, je nutné aplikaci restartovat a celý proces opakovat (jedná se o nevyřešený bug). Pokud vše proběhne, správně budou se v seznamu automaticky objevovat připojená auta. Kliknutím na ikonu auta v seznamu lze zobrazit jeho parametry na obrazovce Car Parameters. Pokud chceme parametry editovat, navolíme hodnoty pomocí interaktivních prvků a stiskneme ikonu uploadu. Toto lze provézt i pro několik aut současně. Stačí chvíli podržet jeden z prvků a seznam se přepne do režimu multiple-selection (všechny prvky lze vybrat i tlačítkem Select All). Pro návrat do režimu výběru jediného prvku stačí zase jeden z prvků podržet. Vybraná auta lze ze seznamu vymazat tlačítkem Delete.

Parametry celého experimentu lze nastavit na obrazovce Experiment. Tlačítko s ikonou kompasu spustí proces zjištění polohy aut v koloně. Matice komunikační topologie je prozatím pouze pasivní a nelze jí měnit reálnou topologii. Scrollováním dolů se lze dostat k parametrům Leader Speed a Car Spacing, které lze měnit až po startu experimentu.

Před startem experimentu je dobré si nejdříve nastavit vykreslování grafu na obrazovce Graphs. Uživatel musí v levém seznamu vybrat auta, která chce vykreslit a v pravém seznamu, které veličiny. Pokud chce vykreslovaná data změnit v průběhu experimentu, musí pozastavit vykreslování tlačítkem s ikonou pausy, změnit parametry a zase spustit vykreslování. Tlačítkem zámku lze přepínat, v jakém módu se bude vykreslovat časová osa.

Experiment, lze nastartovat tlačítkem s ikonou semaforu na horní liště. Pokud je ikona zelená, experiment běží a pokud červená tak ne. Opětovným stisknutím lze experiment zastavit. Restart celého experimentu, lze provést zastavením a opětovným spuštěním. Pokud je experiment zastaven, lze připojit nové modely aut. Po proběhnutém experimentu, lze na obrazovce Experiment uložit naměřená data tlačítkem s ikonou diskety. Uložené soubory jsou k nalezení ve složce /storage/emulated/0/CC2531. V programu Total Commander stačí zvolit složku Karta SD a následně složku CC2531.

E. Seznam příloh na přiloženém CD

- Složka CC2531GUI je projekt hlavní aplikace pro systém Android pro IDE Eclipse
- Složka CC2531ThreadedRW je projekt aplikace umožňující vytvoření Zigbee sítě pomocí PC pro IDE Netbeans
- Složka CC2531ThreadedZD je projekt aplikace umožňující simulaci modelů aut na PC pro IDE Netbeans
- Soubor CC2531GUI.apk je zkompilovaný instalační soubor aplikace
- Soubor Slotcar Project.mp4 je demonstrační video použití aplikace
- BachelorThesis.pdf je elektronickou verzí této práce