VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

DIAGNOSTIKA AUTOMOBILU NA MOBILNÍCH TELEFONECH S OS GOOGLE ANDROID

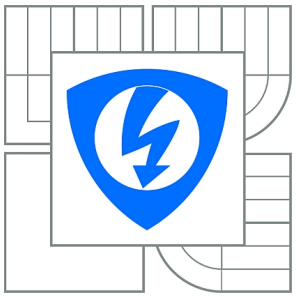
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PETR ŠŤASTNÝ
AUTHOR

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

DIAGNOSTIKA AUTOMOBILU NA MOBILNÍCH TELEFONECH S OS GOOGLE ANDROID

CAR DIAGNOSTICS ON MOBILE PHONES WITH OS GOOGLE ANDROID

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

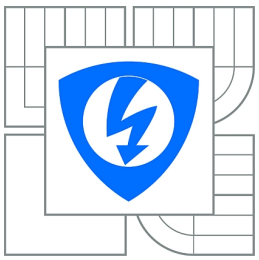
AUTOR PRÁCE — PETR ŠŤASTNÝ

AUTHOR

VEDOUCÍ PRÁCE — Ing. ONDŘEJ LUTERA

SUPERVISOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Petr Šťastný	***ID:***	134633
Ročník:	3	***Akademický rok:***	2012/2013

NÁZEV TÉMATU:

Diagnostika automobilu na mobilních telefonech s OS Google Android

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvořit aplikaci pro mobilní telefony s operačním systémem Google Android, která umožní diagnostiku automobilu a zobrazení provozních hodnot automobilu na displeji mobilního telefonu.

1. Seznamte se s diagnostickým rozhraním OBD-II
2. Seznamte se s vývojovou platformou pro Android SDK
3. Navrhněte vhodný způsob vyčtení a zobrazení provozních hodnot automobilu na mobilních telefonech s OS Google Android a ověřte funkčnost navrhnutého řešení
4. Na základě navrženého řešení vytvořte software pro získávání diagnostických a provozních dat z automobilu. Software odladťe a ověřte funkčnost v automobilu.

DOPORUČENÁ LITERATURA:

[1] MURPHY, Mark L. Android 2: průvodce programováním mobilních aplikací. Vyd. 1. Brno: Computer Press, 2011, 375 s. ISBN 978-80-251-3194-7.

[2] COX, Roy. Introduction to OBD II. Student ed. Clifton Park, NY: Thomson Delmar Learning, c2006, 214 p. ISBN 14-180-1220-3.

[3] HENDERSON, Bob a John Harold HAYNES. The Haynes OBD-II. Student ed. Newbury Park, Calif.: Haynes Pub. Group, c2006, 1 v. (various pagings). ISBN 15-639-2612-1.

Termín zadání:	11.2.2013	***Termín odevzdání:***	5.6.2013

Vedoucí práce: Ing. Ondřej Lutera
Konzultanti bakalářské práce:

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

ABSTRAKT

Práce se zabývá zobrazením informací z palubního počítače na mobilních přístrojích s operačním systémem Android. Přenos informací je realizován pomocí integrovaného obvodu ELM327, připojeného k Bluetooth modulu pro bezdrátovou komunikaci s mobilním přístrojem.

KLÍČOVÁ SLOVA

OBD-II, Android, Google, ELM327, Bluetooth

ABSTRACT

This thesis deals with the display of information from the onboard computer to the mobil devices with Android operating system. Information transfer is implemented using intergrated circuit ELM327 connected to the Bluetooth module for wireless communication with mobile instrument.

KEYWORDS

OBD-II, Android, Google, ELM327, Bluetooth

ŠŤASTNÝ, Petr *Diagnostika automobilu na mobilních telefonech s OS Google Android*: bakalářská práce. BRNO: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2012. 43 s. Vedoucí práce byl Ing. Ondřej Lutera

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Diagnostika automobilu na mobilních telefonech s OS Google Android“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

BRNO ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Ondřeji Luterovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

BRNO . . . . . .

. . . . . .

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Současné automobily jsou vybaveny množstvím elektronických systému. Jedním z nich je i systém palubního počítače. Tyto počítače informují řidiče například o průměrné spotřebě nebo o stavu nádrže. Ne všechny automobilky nabízí tak sofistikované palubní počítače jako konkurence. Dokonce některé automobily nemají palubní počítač vůbec. I tento automobil však shromažďuje všechny informace, které mohou být zobrazeny na palubním počítači.

Pokud tedy automobil nemá palubní počítač, bylo by vhodné, podávat tyto informace řidiči přes zařízení, které většina lidí již vlastní. Nejlépe na „chytrý“ mobilní telefon.

Cílem této bakalářské práce je propojit tyto dva světy. V první části práce se nachází problematika palubních počítačů a s ní spojený standard OBD-II. Ve druhé části bude představen operační systém Android, který se dodává na „chytré“ mobilní telefony. Dalším bodem práce bude vlastní komunikace mezi automobilem a mobilním přístrojem. Tento postup bude následně ověřen na testovací aplikaci. Závěrem se bude práce věnovat vývoji komplexní aplikace pro výčet informací z automobilu.

Toto řešení by mohlo mít tu výhodu, nad standardními palubními počítači, že bude přenositelné na jakýkoliv automobil a mobilní přístroj s operačním systémem Android. Dále by mohlo zobrazovat více informací než dostupné palubní počítače a využít plný potenciál OBD-II.

1 ON-BOARD DIAGNOSTICS

[1]

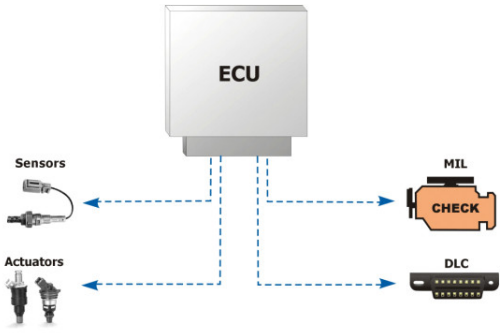
1.1 Co je to On-Board Diagnostics

On-Board Diagnostics, zkráceně OBD, je komunikační rozhraní, pomocí kterého může automobil shromažďovat informace o svém stavu. Tyto informace je pak možné zobrazovat přímo na palubní desce nebo k nim můžeme přistupovat pomocí externího zařízení. Od 80. let 20. století, kdy se poprvé začaly objevovat palubní počítače ve vozidlech ze sériové výroby, se tento systém postupně vyvíjel a neustále vyvíjí. S postupným vývojem se rozšiřovala paleta dostupných informací.

Dalo by se říci, že základem dnešních systémů OBD je propojení ECU (Electronic Control Unit - elektronická řídící jednotka) se senzory (např. senzor tlaku vzduchu) a mechanickými částmi motoru (např.: vstřikování). V průběhu vývoje se pak přidal tzv. „Check Engine Light“ nebo také známý jako MIL (Malfunction Indicator Light), což je oranžová kontrolka motoru na palubní desce(obr. 1.1). Ta informuje řidiče o poruše vozidla. U moderních automobilů je rozsvícení této kontrolky doprovázeno ještě stručným popisem závady (např. u Škody Octavia se vypíše „Motor do dílny!“ nebo „Emise do dílny!“).

Obr. 1.1: Check Engine Light neboli MIL

Pokud se tato kontrolka rozsvítí, je pak nevyhnutelné napojit vozidlo na diagnostický program, který je na externím zařízení. To se provádí pomocí standardizovaného DLC (Diagnostic Link Connector) konektoru. Základní schéma je pak vidět na obr. 1.1



Obr. 1.2: Základní schéma OBD (převzato z [2])

1.2 Standardní rozhraní

1.2.1 OBD-I

OBD-I vznikl na začátku 80. let 20. století. V Californii tou dobou zemřelo několik lidí na otravu. Tu jim způsobila enormní smogová situace. Proto se tehdejší guvernér rozhodl reagovat tak, že zakázal prodávat a provozovat vozidla, která do ovzduší vypouštěla nepřípustné množství emisí výfukových plynů. Z předpisů se postupem času stalo doporučení pro výrobu automobilů.

Jelikož měli výrobci volné ruce při zavádění těchto předpisů, nebylo dost dobře možné, aby se předpisy daly efektivně kontrolovat. Každý výrobce si vymyslel vlastní konektor a vlastní modely komunikace. Tato situace vedla k tomu, že nezávislé kontroly STK musely nakoupit hodnotná zařízení k detekci těchto předpisů. Lidé pak museli dávat mnoho peněz za tyto kontroly a raději na kontroly nejezdili. V polovině 90. let se tedy začalo přemýšlet, jak sjednotit tyto předpisy. Tyto myšlenky daly poté vzniknout standardu OBD-II.

1.2.2 OBD-II

Tento standard navazuje na snahu o snížení emisí výfukových plynů. Obsahuje přesně stanovené hodnoty, které musí automobil plnit, aby byl schopen provozu na pozemních komunikacích. Výsledkem je, že všude na světě se technik může připojit k vozidlu přes standardizovaný konektor J1962. Dále může z autem komunikovat jedním z předepsaných komunikačních protokolů uvedených v normě. Norma dále obsahuje seznam všech komunikačních modelů a seznam doporučených přenosových médií.

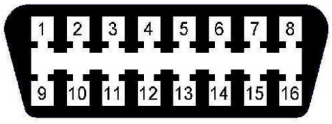
Nalezneme v ní také, jak mají vypadat chybová hlášení tzv. DTC (Diagnostic Trouble Codes). Ta jsou stanovena jako 4 číslice s prefixem v podobě písmene. Existují 4 druhy prefixů: P (Powertrain) pro motor a převodovku, B (Body) pro karosérii, C(Chassis) pro podvozek a U pro komunikační síť. 4 číslice pak znamenají číslo chyby. Pokud je první číslo 0 znamená to, že je chyba standardní, pokud je 1 znamená to, že se jedná o chybu specifikovanou výrobcem. Význam dalších čísel je pak závislý na prefixu. Význam těchto kódů je uveden u diagnostických zařízeních anebo na různých internetových stránkách.

Výrobci automobilů využili této standardizace ve svůj prospěch. Začali nabízet automobily s palubními počítači v příplatkové výbavě. Ze začátku se instalovaly do luxusních modelů luxusních značek. Postupem času však byly dostupné v běžně dostupných modelech. V dnešní době však ještě není pravidlem, že palubní počítač má každé vozidlo. Typicky tyto počítače zobrazují hodnoty jako průměrná a okamžitá spotřeba, dojezd, ujeté kilometry nebo informace o poruše vozidla.

Diagnostický konektor OBD-II

Součástí standardu OBD-II je i standardizované hardwarové rozhraní. Jedná se o 16pinový (2x8) konektor J1962. Musí být umístěn v kabině vozu maximálně 60 cm od volantu (někdy je možné se setkat s tím, že je konektor umístěn někde jinde, ale pořád v dosahu řidiče, záleží na výrobci).

Organizace SAE (Society of Automotive Engineers) ustanovila výstupy všech pinů konektoru J1962, jež je vidět v obr. 1.3 a v tab. 1.2.2.



Obr. 1.3: Konektor J1962

1	–	9	–
2	SAE J1850 PWM a SAE J1850 VPW sběrnice +	10	SAE J1850 PWM a SAE J1850 VPW sběrnice −
3	–	11	–
4	Baterie GND (−)	12	–
5	Signal GND	13	Signal GND
6	ISO 15765 CAN High	14	ISO 15765 CAN Low
7	ISO 9141-2 a ISO 14230 KWP2000 K linka	15	ISO 9141-2 a ISO 14230 KWP2000 L linka
8	–	16	Baterie +

Tab. 1.1: Zapojení výstupních pinů konektoru J1962

1.2.3 EOBD

Směrnice EOBD (European On-Board Diagnostic) je evropským ekvivalentem k OBD-II, která se užívá u vozidel kategorie M1[1]. Tato směrnice oficiálně platí v EU od 1.1.2001 pro benzínové a od 1.1.2004 pro naftové motory. Pro nové modelové řady byla uvedena v platnost již o rok dříve, tj. pro benzínové motory 1.1.2000 a pro naftové 1.1.2003. Pro vozidla přesahující celkovou hmotnost 2500 kg, platí tato regule od roku 2002 pro benzínově poháněná vozidla a od roku 2007 pro naftově poháněná vozidla.

Technická implementace EOBD je stejná jako u OBD-II. Užívá stejný konektor (J1962) a stejné přenosové protokoly.

[1] nemají více než 9 míst a jejich celková hmotnost nepřesahuje 2500 kg

1.3 Signální protokoly OBD-II

Celosvětově se používá 5 standardizovaných signálních protokolů s rozhraním OBD-II. Většina vozidel podporuje pouze jeden z těchto 5 signálních protokolů. Většinou se dá zjistit, jakým protokolem vozidlo komunikuje podle zapojení výstupních pinů konektoru J1962.

Všechny protokoly OBD-II používají stejný konektor, který má výstupní piny zapojeny podle toho, přes který protokol komunikuje. Pouze 2 piny jsou fixní a to:

- pin 4: Battery GND
- pin 16: Battery +

1.3.1 SAE J1850 PWM

Používá jej automobilka Ford. Využívá pulsně-šířkovou modulaci pro přenos signálu. Rychlost tohoto signálu je 41,6 kB/s. Úroveň logické 1 je +5 V, délka slova je 12 bitů, přepočítaná pomocí CRC[2]. Pro přístup k fyzickému médiu se využívá metoda CSMA/NDA[3].

1.3.2 SAE J1850 VPW

Užívaný automobilkou General Motors. Používá variabilní pulsně-šířkovou modulaci pro přenos. Rychlost přenosu dat 10,4 kB/s nebo 41,6 kB/s. Úroveň logické 1 je +7 V. Délka slova je 12 bitů s použitím CRC. Pro kontrolu přenosu se požívá CSMA/NDA.

1.3.3 ISO 9141-2

Využívá jej automobilka Chrysler a také evropské a asijské vozy. Tento protokol používá nesynchronní sériovou komunikaci o rychlosti přenosu dat 10,4 kB/s. Je podobný přenosu po RS232. Má ale jiné napěťové úrovně, komunikace je pouze na jednom kabelu v obou směrech a nepoužívá handshake.

Délka slova je 12 bitů s CRC, signalizace pomocí UART. Řízení aktivního nebo dominantního stavu je řešeno nízkým napětím na otevřeném kolektoru.

1.3.4 ISO 14230 KWP2000

Vylepšená verze předchozího protokolu. Mají stejnou fyzickou vrstvu, liší se v rychlostech a délce slova. Rychlost přenosu dat je 1,2–10,4 kB/s, délka slova může být

[3]Cyklická redudantní kontrola – kontrolní součet všech bitů, pro ověření správnosti přenosu

[3]Carrier Sense Multiple Access with Non-Destructive Arbitration – jedná se o přístupovou metodu, která přistupuje k fyzickému médiu v době, kdy na tomto médiu není žadný provoz. Tato metoda je nedestruktivní vůči přenášenému slovu.

až 255 bitů.

1.3.5 ISO 15765 CAN

CAN (Controller area network) protokol byl vyvinut společností Bosch a byl poprvé představen veřejnosti v roce 1986. Na rozdíl od jiných, výše uvedených, standardů OBD se tohoto standardu také využívá pro komunikaci různých mikrokontrolerů v průmyslových továrnách.

CAN je „textově“ orientovaný protokol a byl vytvořen k zjednodušení komunikace mezi různými senzory a řídícími jednotkami. CAN má svůj vlastní mikrokontroler, který shromažďuje informace od senzorů a řídících jednotek a poté je posílá přes konektor J1962 ke čtecímu zařízení. Tím se zjednodušila komunikace mezi všemi dostupnými zařízeními.

2 OPERAČNÍ SYSTÉM ANDROID

[5]
Android je operační systém vyvinutý hlavně pro dotyková mobilní zařízení jako jsou smartphone nebo tablet. Android je majetkem společnosti Android Inc., která patří pod skupinu Google Inc. Poprvé se objevil na trhu v roce 2007. V tomto roce také spolu s ním vznikla Open Handset Alliance, což je společenství 86 harwarových, softwarových a telekomunikačních společností, které se věnuje prosazování open standardů v oblasti mobilních zařízení.

Google vydal Android jako open sourcový systém s liciencí Apache. Původně chtěl Google vyvíjet všechny aplikace sám, ale po nějaké době se „androidoví“ nadšenci sami pustili do vývoje aplikací. Není to nic složitého pro toho, kdo umí s Javou, protože aplikace pro Android se píší v upravené verzi Javy. Aplikace je pak možno stáhnout nebo nahrát na Google Play (původně Android Market), kde je většina z nich zdarma. Jen pro zajímavost, v září roku 2012 bylo na Google Play umístěno kolem 700000 aplikací pro Android a od začátku tohoto serveru bylo staženo přes 25 miliard aplikací.

První telefon s OS Android se začal prodávat v říjnu 2008 a koncem roku 2010 se stal nejprodávanějším operačním systémem pro mobilní telefony. Porazil tak Symbian (operační systém telefonů Nokia), který byl do té doby na prvním místě. Celosvětově se Android dodává v 75 % všech mobilních zařízeních, s 500 milióny aktivovanými zařízeními a s 1,3 milióny aktivací každý den. V budoucnu se Android hodlá zaměřit nejen na smartphony a tablety, ale také chce implementovat svůj systém do televizí, netbooků nebo fotoaparátů.

2.1 Uživatelské prostředí

Uživatelské prostředí Androidu je založeno na přímé interakci mezi uživatelem a operačním systémem pomocí dotykové obrazovky. Prostředí je velmi interaktivní a tzv. „user friendly“. Jelikož ne všechno se dá pomocí dotyků na obrazovce ovládat, je zařízení většinou vybaveno akcelerometry a gyroskopem pro efektivnější ovládání systému. Pomocí akcelerometrů se například rozhoduje o natočení obrazovky.

2.1.1 Aplikace

Počet aplikací pro Android pořád stoupá, hlavně stoupá jejich počet od 3. stran. Aplikace mohou vývojáři nebo uživatelé nahrávat na on-line obchod Google Play nebo na Amazon Appstore. Další možností je nahrát aplikaci do přístroje přímo, například přes kabel z počítače (toho se využívá především při vývoji k testování na

hardwarovém zařízení). Play obchod umožňuje uživatelům sdílet, updatovat nebo stahovat aplikace přímo od Googlu nebo od ostatních vývojářů a také je nainstaluje do zařízení s plnou požadovanou kompatibilitou. Aplikace mohou být tříděny do různých skupin například podle typu přístroje (jsou pak viditelné jen uživatelům s vybraným typem přístroje) nebo například podle regionů prodeje.

Aplikace se píší v jazyce Java s použitím nástroje Android software development kit (Android SDK). SDK obsahuje celý set vývojových nástrojů včetně debuggerů, softwarových knihoven, telefonní emulátor založený na QEMU, dokumentaci a příklady základních příkazů a kódů. Oficiálně podporované vývojové prostředí je Eclipse, který využívá zásuvný modul ADT (Android Development Tools). K mání jsou i jiné způsoby vývoje, například pomocí NDK (Native Development Kit), který se využívá pro programování v jazycích C a C++.

2.2 Vývoj Androidu

2.2.1 Linux

Android se zakládá na Linuxovém jádru Linux Kernel s knihovnami, s API napsanými v C a aplikacemi, které jsou kompatibilní s Java knihovnami založenými na licenci Apache. Android používá Dalvik virtual machine[1] s just-in-time kompilací k rozběhnutí dex[2] kódů, které jsou překládány z Java bytecodů. V praxi to funguje tak, že zdrojový kód napsaný v Javě je přeložen do .jar aplikace, která je přes Dalvik přeložena na dex kód, který je pak vyhodnocen systémem a spuštěn. Takto komplikované je to hlavně kvůli úspoře paměti a výpočetního výkonu CPU.

2.2.2 Organizace paměti

Jelikož zařízení s OS Android jsou především napájeny přes baterie, je tento systém designován s pamětmi RAM k udržení co nejmenší spotřeby energie. Tímto se liší od stolních počítačů, které jsou připojeny do rozvodné sítě a spotřeba energie není až tak důležitá. Když se aplikace již nepoužívá je uložena do paměti, odříznuta od jakékoli spotřeby energie a čeká na své opětovné spuštění. Navíc to ve skutečnosti zrychluje její nabíhání, jelikož systém ji nemusí pokaždé spouštět od začátku, ale naváže v ní na tom místě, kde byla odložena do paměti, je pouze vyvolána a nepotřebuje tolik energie jako při studeném startu.

[1] DVM–virtuální prostředí, ve kterém se spouští aplikace. Navrhnuto a realizováno lidmi z Google. Chová se jako Java Virtual Machine ale je jednodušší a ořezanější.

[2] Zdrojový kód aplikací na úrovní assembleru.

Android si také sám organizuje paměť. Pokud je jí nedostatek tak aplikace, která byla použita jako první a je tedy „nejstarší“, je ukončena a na její místo se posouvá 2. „nejstarší“. Tato funkce běží na pozadí a je uživateli skryta.

2.2.3 Dostupnost aktualizací

Google poskytuje hlavní aktualizace systému Android každých 6–9 měsíců. Jsou poskytovány koncovým uživatelům přes Internet. Poslední aktualizací je 4.2 Jelly Bean.

V porovnání s jinými mobilními operačními systémy, jmenovitě iOS od Applu, se nové verze Androidu dostávají k uživatelům o mnoho později, typicky o několik měsíců od uvedení na trh. Je to hlavně díky tomu, že Android je poskytován několika výrobcům hardwaru a každý z nich má jiné nároky na výkon systému. Výrobci si tyto verze musí upravovat a to právě způsobuje to zpoždění, které například u iOS nehrozí. Někteří výrobci mobilních zařízení někdy ani nechtěli poskytovat nové verze systému na starší modely, ale raději vytvořili nový telefon a ten nabízeli s novou verzí operačního systému. Kvůli tomu se Google rozhodl v roce 2011 vytvořit úmluvu Android Update Aliance, ve které říká, že každý mobilní přístroj musí být aktualizován novou verzí systému minimálně jednou za 18 měsíců[3].

2.3 Vývoj aplikací

[6]

Operační systém Android využívá především mobilní telefony. I když Google oznámil, že by Android mohl najít daleko rozsáhlejší uplatnění (počítače, palubní desky automobilů), v současné době se klade důraz na vývoj pro mobilní telefony. Pro vývojáře to má své výhody i nevýhody.

Mezi nesporné výhody patří, že jsou „chytré“ telefony nezvykle přitažlivé pro uživatele. Počátek internetových služeb v mobilních telefonech se datuje již do poloviny 90. let od vytvoření jazyka Handheld Device Markup Language (HDML). Skutečně se telefony schopné připojení k Internetu prosadili až v posledních několika letech. Díky trendům, jako je psaní textových zpráv nebo díky zařízení iPhone od Apple, získávají telefony, prostřednictvím nichž se lze připojit k Internetu, rychle na popularitě.

Problém ovšem nastává při samotném programování. Při programování pro mobilní telefony narážíme na to, že telefony jsou ve všech ohledech malé.

- Telefony mají velmi malou obrazovku.

[3] Jelikož je rok 2012, tak se tato dohoda ještě neověřila v praxi.

- Klávesnice jsou velmi malé (pokud je jimi telefon vybaven).
- Rychlost CPU a velikost paměti je v porovnání s PC malá (v dnešní době už to není takový problém).
- Při psaní je možno použít jakýkoliv jazyk nebo prostředí, ale jenom pokud se to výrobce telefonu rozhodl podporovat.

Kromě toho jsou zde také úskalí s tím, že se jedná o mobilní telefon a tuto funkci musí plnit především. Nesmí se nám tedy stát, že telefon přestane fungovat například z těchto důvodů:

- Aplikace vytíží procesor telefonu takovým způsobem, že telefon není schopný přijmout příchozí hovor.
- Aplikace odmítá tiše zmizet do pozadí, když telefon upozorní uživatele na příchozí hovor nebo uživatel chce někomu zavolat, protože nespolupracuje se zbytkem operačního systému.
- Aplikace způsobí zhroucení operačního systému, například v důsledku masivních úniků paměti.

Z těchto důvodů se tedy vývoj aplikací pro telefon liší od vývoje programů pro stolní počítače, webové stránky nebo obslužné procesy serverů. Liší se vývojové nástroje, jinak se chová aplikační rámec a funkce programů podléhají mnoha různým omezením.

Android se pak snaží o kompromis:

- Umožňuje psát aplikace v běžně dostupném používaném programovacím jazyce (Java), používat některé běžné knihovny (například některá Apache Commons API) a nástroje, na které jsme zvyklí (Eclipse).
- Nabízí poměrně jasně stanovený a oddělený aplikační rámec, ve kterém programy musí běžet, takže mohou být „dobrými občany“ telefonu a nenarušují funkci ostatních programů nebo samotného telefonu.

2.3.1 Složení aplikací

Když píšeme aplikaci pro stolní počítač, jsme „pány své vlastní domény“. Spustíme její hlavní okno a tolik oken jeho potomků – například dialogů – kolik potřebujeme. Z našeho úhlu pohledu se jedná o náš vlastní svět, ve kterém sice využíváme vlastnosti operačního systému, ale příliš se nestaráme o další programy, které mohou být v počítači spuštěny současně s naším. Pokud s nimi nějak intereagujeme, děláme to obvykle prostřednictvím nějakého API, jako je například Java Database Conectivity (JDBC) nebo na něm založený aplikační rámec, abychom mohli komunikovat s MySQL nebo jinými databázemi.

Systém Android pak využívá podobné koncepce, ale jinak zabalené a strukturované s ohledem na větší ochranu telefonů proti zhroucení systému. Aplikace pro

systém Android sestávají z následujících komponentů:

Aktivity

Stavebními kameny pro tvorbu uživatelského rozhraní jsou aktivity. Můžeme je chápat jako analogii oken či dialogů aplikace pro stolní počítač. Ačkoliv je možné, aby aktivity neměly uživatelské rozhraní, většina zdrojového kódu bez uživatelského rozhraní bude zabalená spíše ve formě dodavatelů obsahu anebo služeb.

Dodavatele obsahu

Dodavatele obsahu zajišťují úroveň abstrakce jakýchkoliv dat uložených v zařízení, která jsou přístupná více různým aplikacím. Vývojový model systému Android nás podporuje v tom, abychom zpřístupnili svá data i jiným aplikacím než pouze své vlastní. Dosáhneme toho pak právě vytvořením dodavatele obsahu, který nám současně poskytuje úplnou kontrolu nad způsobem přístupu k našim datům.

Služby

Aktivity a dodavatele obsahu jsou entity s krátkou životností a lze je kdykoliv vypnout. Služby jsou oproti tomu navržené tak, aby, pokud je to potřeba, pokračovaly ve své práci nezávisle na jakékoliv aktivitě. Službu můžeme použít k detekci aktualizací RSS zdroje nebo k přehrávání hudby, které pokračuje, dokonce i když byla příslušná ovládací aktivita uzavřena.

Záměry

Záměry jsou systémové zprávy upozorňující aplikace na výskyt různých událostí, změnami hardwarové konfigurace počínaje (například vložení SD karty), přes události související s příchozími daty (například přijetí SMS zprávy) a událostmi aplikace konče (například její spuštění z hlavního menu zařízení). Na záměry pak můžeme nejenom reagovat, ale můžeme také vytvářet své vlastní záměry a spouštět jejich prostřednictvím jiné aktivity nebo je využívat k detekci specifických situací (například vyvoláme takový a takový záměr, když se zařízení dostane do vzdálenosti 100 metrů od takové a takové lokace).

2.3.2 Vlastnosti a nástroje systému

Systém Android nabízí množství vlastností a nástrojů, které jsou využitelné při vývoji:

Uložiště

Datové soubory, jejichž obsah se nebude nijak měnit (například ikony nebo pomocné soubory), můžeme přibalit přímo k aplikaci. Navíc také můžeme ukořistit trochu volného místa v zařízení a uložit data do databází nebo souborů obsahujících uživatelem zadávaná či načítaná data, která aplikace potřebuje.

Síť

Zařízení využívající operační systém Android jsou obecně vzato připravená pro připojení k Internetu prostřednictvím různých druhů komunikačních médií. Přístup k Internetu můžeme využít na jakékoliv úrovni chceme, surovými Java sockety počínaje a zabudovaným widgetem webového prohlížeče založeným na jádře WebKit konče.

Multimédia

Zařízení využívající OS Android umí přehrávat a zaznamenávat audio–videozáznamy. Ačkoliv se konrétní specifika těchto schopností liší zařízení od zařízení, můžeme se zařízení zeptat, jakými schopnostmi pro přehrávání a záznam audio a videozáznamů disponuje, a poté tyto schopnosti využít, jak uznáme za vhodné – ať už k přehrávání hudby, pořizování fotografií fotoaparátem nebo záznamu audiopoznámek prostřednictvím mikrofonu.

GPS

Zařízení využívající operační systém Android mají často přístup k dodavatelům údajů o zeměpisné poloze, například ke službě GPS, která může poskytnout informace o tom, kde na zeměkouli se zařízení zrovna nachází. Pak můžeme následně na základě těchto informací zobrazovat mapy nebo je využívat nějak jinak, například ke sledování pohybu zařízení v případě jeho odcizení.

Telefonní služby

Protože jsou zařízení využívající operační systém Android obvykle telefony, může aplikace samozřejmě také provádět telefonická volání, zasílat a přijímat SMS zprávy a provádět jakékoliv další operace, které od moderního telefonu očekáváme.

3 NÁVRH KOMUNIKACE

Při navrhování komunikace mezi OBD-II a operačním systémem Android se pokusíme vyjít z již existujících komunikačních modelů. Především pak z komunikace mezi OBD-II a PC.

Většina těchto komunikací probíhá přes USB kabel, což je nejjednodušší řešení. USB konektor má na PC v dnešní době každý a přenosové rychlosti tohoto rozhraní jsou velmi dobré. K přenosu informací přes USB musíme ovšem mít nějakou redukci.

3.1 Dostupná přenosová média

V současnosti existuje mnoho překladačů komunikace mezi OBD-II a USB. Většina z nich se dá použít pouze na vozidla některých výrobců, kteří do vývoje těchto nástrojů sami investují. Tyto překladače využívají většinou stejného principu. Na konektor J1962 se připojí zásuvný modul, který obsahuje integrovaný obvod s funkcí překladače. Dále pak obsahuje redukci pro komunikaci s USB rozhraním. Většina z těchto řešení nepodporuje všechny typy komunikačního protokolu.

Typickým představitel užití kabelu pouze na některá vozidla je VAG-COM. Ten je speciálně vytvořen pro komunikaci s automobily koncernu Volkswagen. Překládá HEX čísla na CAN, užívá se označení kabelu HEX-CAN. Dá se připojit k diagnostickému programu VCDS. Ten umí mazat chyby, zobrazovat aktuální informace a také umí měnit nastavení řídící jednotky. Jelikož se umí připojit pouze k vozidlům koncernu Volkswagen, není pro nás příliš vhodný.

Existuje také mnoho diagnostických zařízení, která vypadají spíše jako multimetry. Z nich je to například Universal CAN Pro-Series OBDII Trouble Code Reader (obr. 3.1). Ten umí pouze číst a mazat chybová hlášení. Dá se použít na všechna vozidla, což je jeho obrovská výhoda. Jeho nevýhodou je, že se nedá přepojit k žádnému jinému nástroji, proto je také nevhodný.

Protože všechna tato řešení využívají jako přenosové médium USB rozhraní, nejsou pro řešení na mobilních telefonech vhodná. Jsou velmi nepraktická, pokud by měl mít řidič vyveden USB kabel do telefonu, například z místa pod volantem. To by mu určitě překáželo. Proto by bylo vhodné najít překladač, který by byl schopen posílat data po fyzické vrstvě bezdrátově. Vhodným kandidátem může být integrovaný obvod ELM327, který tuto funkci podporuje.

Obr. 3.1: Universal CAN Pro-Series OBDII Trouble Code Reader (převzato z [9])

3.2 ELM237

ELM327 je integrovaný obvod s funkcí překladače komunikací. Podporuje přeložení datové komunikace z RS232 na OBD-II u všech norem popsaných v kapitole 1.3. Jeho výhodou je, že se dá přizpůsobit mnoha způsobům přenosu informací po fyzické vrstvě, tzn. že může například komunikovat přes USB kabel nebo bezdrátově přes Bluetooth. Z tohoto důvodu je vhodný při sestavení komunikace přes Bluetooth. Ke komunikaci bude využita běžně dostupná redukce s ELM327 a Bluetooth modulem, obr. 3.2

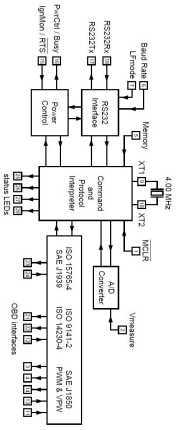
Obr. 3.2: Redukce s ELM327 a modulem Bluetooth

3.2.1 Schéma ELM327

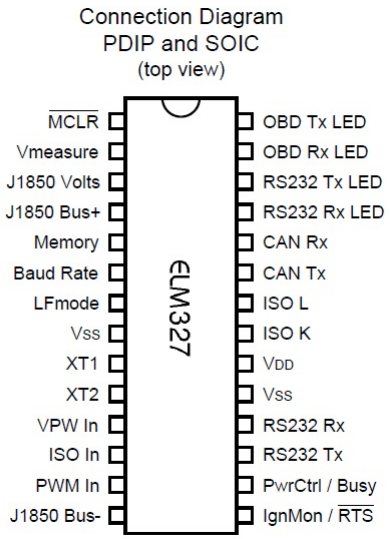
Jak jde vidět z obr.3.2.1 a obr.3.2.1 tak ELM327 má 28 pinů. Nejvíce pinů je spotřebováno na propojení komunikačních protokolů. Dále jsou zde piny pro vstupní

napětí, nastavení rychlosti přenosu a piny pro řízení komunikace. Z blokového schématu pak vidíme, že obvod má řídicí modul (Command and Protokol Interpreter), ke kterému je připojen A/D převodník. Ten převádí analogové vstupní napětí $V_{measure}$ (0-5V). Řídicí modul je taktován na 4MHz pomocí oscilátoru připojeným na piny 9 a 10 (XT1 a XT2). Dále je připojen k řídicímu modulu modul s komunikačními protokoly, modul s rozhraním RS232 a modul řízení spotřeby. Poslední 4 piny jsou rezervovány pro signální LED diody.

Pomocí pinu $\overline{MCLR}$ se dá celý obvod resetovat a uvést do původního režimu. Na pin V_{DD} je přivedena logická 1 a na pin V_{SS} je přivedena logická 0. Typická hodnota logické 1 je 5V, ale může se pohybovat mezi 4,5–5,5V.



Obr. 3.3: Blokové schéma ELM327



Obr. 3.4: Pohled shora na ELM327

3.2.2 Komunikace s ELM327

ELM327 očekává, že bude s ostatními zařízeními komunikovat přes RS232. Nicméně většina novodobých zařízení nemá RS232 port, takže součástí tohoto čipu je také převodník na USB nebo na Bluetooth. Nezáleží jak se fyzicky k ELM327 připojíme, důležitější je způsob komunikování. Nejjednodušším způsobem je připojení přes Hyperterminál nebo přes nějaký podobný program. Stačí pouze poslat dotaz, na který přijde odpověď.

K použití terminálového programu budeme potřebovat ho správně nastavit. Napřed musíme vybrat správný COM port, pak je potřeba nastavit správnou rychlost přenosu dat. Pokud na pinu 6 není přivedeno žádné napětí je tato rychlost 9600 Bd[1]nebo 38400 Bd pokud se adresa PP 0C nezmění. Pokud se zvolí špatný COM port nebudeme schopni odečítat jakékoliv informace. Jestliže zvolíme špatnou rychlost přenosu informace, mohou být neúplné nebo špatné. Nesmíme také zapomenout nastavit komunikaci na 8 bitů bez parity s jedním stop bitem. Všechny odpovědi od ELM327 jsou zakončeny jedním návratovým znakem.

Jednoduchými příkazy se dá ověřit zde náš ELM327 komunikuje s PC nebo mobilním telefonem nebo zařízením, které se dá k němu připojit ať už pomocí USB nebo Bluetooth.

AT příkazy

Pomocí těchto příkazu se dá ELM327 nastavovat. Po nastavení potřebných hodnot je potřeba obvod resetovat, aby se změna projevila. Těmito příkazy se dá nastavit například rychlost komunikace. Každý komunikační protokol pak má svou vlastní sadu AT příkazů, kterými se dá ověřit zda je přenos informací uskutečnitelný pomocí tohoto protokolu. Zvláštní sadu AT příkazů má i samotné OBD. Z těch je nejpoužívanější `AT SP`, který nastaví komunikační protokol.

3.3 Testovací program

Tato jednoduchá aplikace slouží k ověření modelu komunikace, který je navržen výše. Při psaní využijeme poznatků již zmíněných a budeme se snažit je promítnout do aplikace.

[1]Bd (Baud)–jednotka modulační rychlosti, udává počet změn stavu přenosového média za jednu sekundu.

3.3.1 Bluetooth komunikace v Androidu

Aplikace musí obsahovat třídy pro samotnou komunikaci přes Bluetooth. Jsou to třídy BTAdapter, BTConnection a Comunicator. Dále musíme implementovat knihovny API Androidu. Jsou to třídy BluetoothAdapter, BluetoothDevice a BluetoothSocet, všechny najdeme v balíku android.bluetooth. Při psaní aplikace napřed musíme získat práva pro manipulaci se zařízením Bluetooth v mobilním zařízení. To nastavíme v manifestu programu přidáním práv (permissions) `BLUETOOTH` a `BLUETOOTH_ADMIN`. Právo `BLUETOOTH` slouží pouze k povolení komunikace se spárovanými zařízeními. Naproti tomu `BLUETOOTH_ADMIN` povoluje přidat nové zařízení a spárovat se s ním. Dále musíme zjistit, zda vůbec naše zařízení má Bluetooth a zapnout ho, pokud už zapnuto není. To provedeme příkazem

```
BluetoothAdapter mBluetoothAdapter =
  BluetoothAdapter.getDefaultAdapter();

if (mBluetoothAdapter != null) {
   if (!mBluetoothAdapter.isEnabled()) {

     Intent enableBTIntent =
       new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
     startActivityForResult(enableBTIntent, DISCOVERY_REQUEST);
   }
 }
```

Třída `BluetoothAdapter` reprezentuje hardwarový adaptér Bluetooth v zařízení. Pomocí jejich metod se dá adaptér zapnout a ověřit zda je adaptér v zařízení. Tento kód potom zjišťuje zda adaptér je v zařízení a pokud ano, tak jej pomocí vytvoření nového záměru (`new Intent`) zapne a vyvolá novou aktivitu, což se projeví na obrazovce upozorněním zda opravdu chceme adaptér zapnout.

Po zapnutí adaptéru se musí zařízení připojit k redukci, která je připojena na konektoru J1962 v automobilu. Pro zjednodušení napřed spárujeme zařízení pomocí uživatelského prostředí Androidu a připojíme se přímo na zařízení pomocí jeho MAC adresy.

```
BluetoothDevice mmDevice;
mmDevice = mBluetoothAdapter.getRemoteDevice("00:06:71:00:00:06");
```

Jakmile je zařízení připojeno musí začít vysílat a přijímat zprávy. Vytvořením třídy BTConnection se spustí komunikace mezi zařízeními. Vytvořením třídy BTConnection její konstruktor vytvoří BluetoothSocket, přes který bude komunikovat. Aby

bylo možné komunikovat je potřeba mít jedinečné UUID. Důležité je aby začínalo hodnotou 00001101. Tento začátek značí, že se bude jednat o sériovou komunikaci. Jakmile je vytvořený BluetoothSocket může aplikace zahájit komunikaci. To provede pomocí metody `run()` třídy BTConnection.

```
public void run() {

        try {
          mmSocket.connect();
          Comunicator comunicator = new Comunicator(mmSocket);
          comunicator.manageConnectedSocket(mmSocket);
        } catch (IOException connectException) {
          try {
            mmSocket.close();
          } catch (IOException closeException) {
          return;
        }
    }
```

Tato metoda otevře komunikační kanál pomocí `mmSocket.conncet();`. Dále pak vytvoří novou třídu comunicator, ve které jsou metody pro vlastní komunikaci. Ta probíhá přes InputStream a OutputStream. Přes InputStream se zapisuje do zařízení a přes OutputStream se ze zařízení čte.

Pro komunikaci bude aplikace využívat speciální PID kódy, které jsou standardizovány společně s OBD-II.

3.3.2 PID kódy

PID kódy nebo-li Parameter ID kódy jsou speciální kódy navržené pro komunikaci s OBD-II. Standardní sada těchto kódu je standardizována jako standard SAE J/1979. Obsahuje mnoho PID pro komunikaci s vozidlem, ale postupem času si většina výrobců vytvořila další sady těchto kódů. Kódy vznikly společně s OBD-II v roce 1996 a od té doby jsou podporovány u většiny osobních vozidel na diagnostických jednotkách. Jejich primárním cílem je efektivní sledování plnění emisních předpisů.

Samotná komunikace, například v autoservisech, pak probíhá takto:

1. Technik připojí diagnostické zařízení ke konektoru ve vozidle. Po připojení technik zadá PID kód.
2. Diagnostické zařízení vyhodnotí PID kód a přeloží pro komunikaci s OBD-II. Vzápětí jej pošle na konektor pomocí jednoho z komunikačních protokolů.
3. Systém ve vozidle pak vyhodnotí hodnotu PID kódu a odpoví na něj.

4. Odpověď je poté přeložena na diagnostickém zařízení a zobrazena technikovi.

Stejného principu pak bude využito v aplikaci, s tím rozdílem, že PID kódy nebude zadávat technik, ale přímo aplikace. Tato komunikace pak bude cyklovat do ukončení nebo přerušení aplikace.

Struktura PID kódu

PID kód se skládá ze dvou hlavních částí. Jedna z nich je tzv. mód a druhá je pak samotný PID kód. Vypadá například takto:

```
01 05
```

To znamená, že chceme přistoupit do módu `01`, který ukazuje aktuální informace o vozidle jako jeho rychlost nebo otáčky motoru. K němu posíláme ještě PID `05`, který vysílá dotaz na hodnotu teploty motoru.

Odpověď pak může vypadat třeba takto:

```
41 05 7B
```

To znamená, že přišla odpověď na mód `01` a PID `05`. Značí to první část odpovědi `41 05`[2]. Číslo `7B` je pak vlastní hodnota, kterou můžeme zobrazit. Je to hexadecimální číslo, které musíme převést na decimální. Následně od něj odečteme hodnotu 40, abychom dostali výslednou hodnotu teploty motoru. 40 musí být odečteno, kvůli tomu, aby bylo možné v hexadecimální soustavě zobrazovat i záporná čísla, jelikož chladící kapalina, na které je hodnota měřena, může dosahovat až $-40°C$.

Pro přepočet těchto hodnot existují vzorce, které jsou součástí normy J1979[8]. Norma také obsahuje všechny standardizované PID kódy a módy.
Vzhled hotového testovacího programu jde vidět na obrázku 3.5.

[2]41 znamená 01 + 40, 40 proto, aby bylo jasně rozeznatelné, že se jedná o odpověď a ne o dotaz. V případě obousměrné komunikace by to mohlo vytvářet náhodné stavy.

Obr. 3.5: Vzhled testovacího programu

4 NÁVRH A REALIZACE APLIKACE

V tomto bodu se práce věnuje vlastnímu návrhu software pro mobilní telefony (tablety) s operačním systémem Android. Nejprve se bude zabývat tím, jaké požadavky by aplikace měla splňovat. Dále rozebere strukturu aplikace a soubor jejích tříd, následně se kapitola bude zabývat uživatelským prostředím a na závěr budou vyhodnoceny poznatky plynoucí z testování aplikace v automobilu.

Pro vývoj bylo použito vývojové prostředí Eclipse s plug-in modulem ADT a vývojovou sadou Android SDK. Aplikace je primárně určena pro Android verzi API 15 (Adnroid 4.0.3 Ice Cream Sandwich), minimálně podporovaná úroveň je pak API 11 (Android 3.0 Honeycomb). Pro vyšší API než 15 je aplikace samozřejmě také určena.

Základem byl předchozí testovací program, na kterém byla vyzkoušena komunikace mezi mobilním přístrojem a automobilem. Program byl pouze rozšířen o několik metod a tříd, které jsou uvedeny níže.

4.1 Požadavky aplikace

Při návrhu bylo důležité zohlednit, že aplikaci budou používat především řidiči. Proto by nebylo vhodné, aby řidič (uživatel) musel za jízdy cokoliv nastavovat nebo potvrzovat. To znamená, že aplikace musí být jednoduchá a snadno ovladatelná.

Dále by měla aplikace zobrazovat užitečné informace pro řidiče. Můžou to být především okamžitá spotřeba nebo předpokládaný dojezd s palivem v nádrži. Dále by měla nabízet možnost, jak tyto informace ukládat a poté vyhodnocovat. To znamená, že bude potřeba nějaké úložiště a nejlépe takové, které nebude zabírat mnoho místa v paměti přístroje. Jednou z mnoha možností by mohla být databáze. Je celkem nenáročná na spotřebu místa v úložišti přístroje a také se k ní dá velmi lehce přistupovat.

Vyhodnocování dat by mohlo probíhat několika způsoby. Jedním ze způsobů je prosté vypsání hodnot z databáze na obrazovku. Tento způsob by nebyl moc vhodný, protože by toho moc uživateli neřekl. Další způsob zobrazení dat by mohl být pomocí nějaké srozumitelné tabulky, kde by uživatel viděl, jak se mu mění určité údaje. Tento způsob by nebyl špatný, ale nebyl by také moc přehledný. Dalším způsobem by mohlo být vykreslení dat do grafu. Ten by zobrazoval vždy závislost určité veličiny na datu v měsíci. Jako příklad by mohla být závislost počtu ujetých kilometrů v horizontu týdne nebo měsíce.

Dalším aspektem je to, že by komunikace přes Bluetooth neměla být příliš pomalá. Je potřeba navrhnout třídu, která se bude starat o komunikaci tak, aby bylo

možné posílat zároveň několik dotazů k ELM327 a dostávat zároveň několik odpovědí. Pokud by perioda aktualizace byla moc dlouhá (5 a vice sekund), musel by se zvolit jiný postup při vyčítaní informací, případně úplně přehodnotit stavbu aktivity. Aplikace by také měla zobrazovat nějaké informace o chybových hlášeních, případně by je mohla i vymazávat z řídící jednotky automobilu. Pokud aplikace bude požadovat tyto skutečnosti, její struktura bude vypadat následovně.

4.2 Základní struktura programu

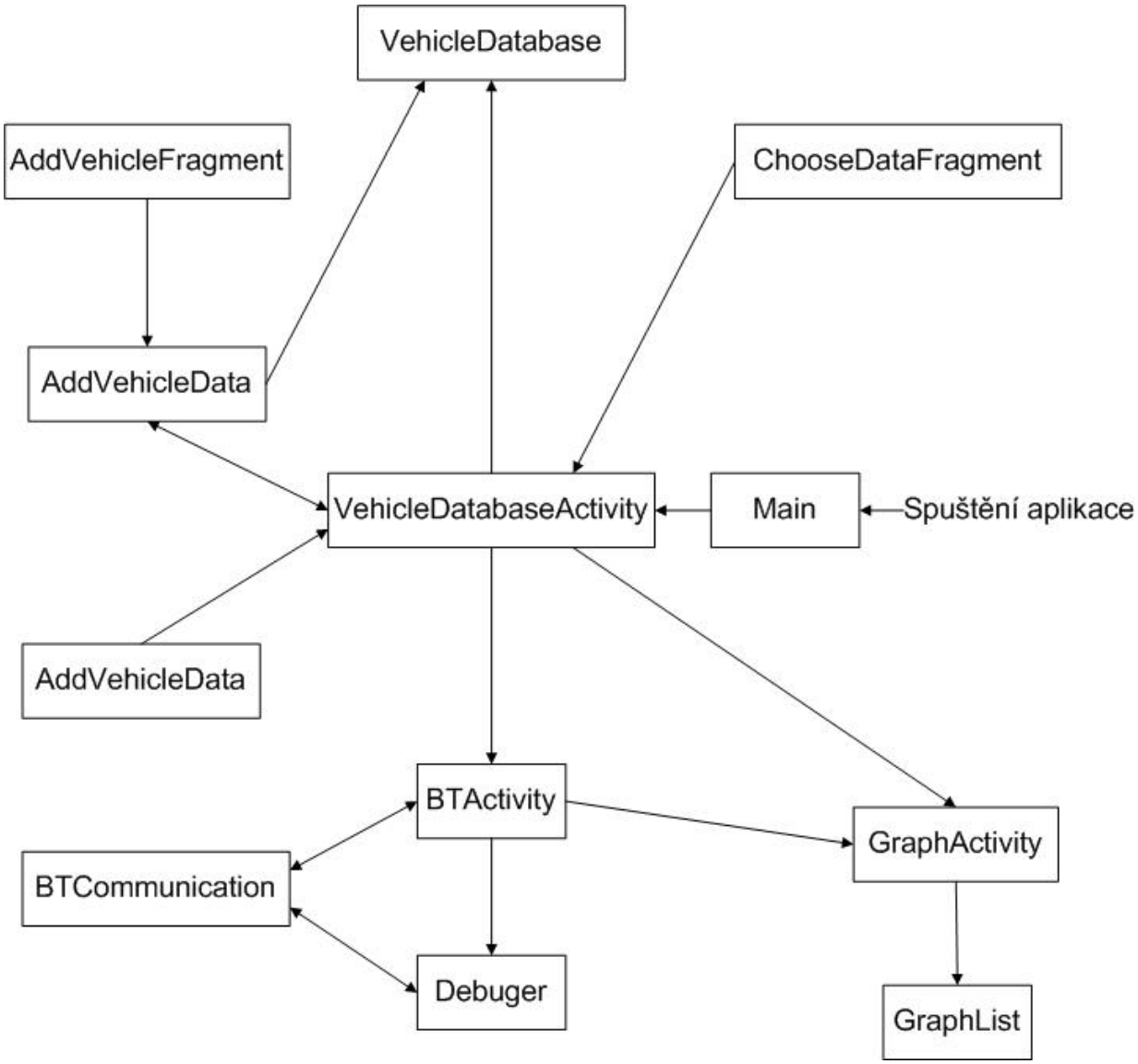
Na obrázku 4.1 jde vidět struktura programu. Při spuštění aplikace se jako první rozběhne třída `Main`. Ta nedělá nic jiného, než že po dobu 1 sekundy běží na obrazovce černé pozadí. Tento interval slouží pro zamaskování načítání informací z databáze. Při delší době používání aplikace bude množství nashromážděných dat velké. Po načtení informací z databáze do paměti se rozběhne první aktivita aplikace s názvem `VehicĺeDatabaseActivity`. Tato aktivita obstarává pohyb dat mezi databází a programem.

Databáze

Samotná databáze vzniká pomocí příkazů metod třídy `VehicleDatabase`. Pro vytvoření databáze je využito služeb knihovny `SQLiteOpenHelper`, která je součástí vývojové sady Android SDK. `SQLiteOpenHelper` pomáhá vytvořit databázi pomocí příkazů pro tvorbu SQLite databází. Součástí třídy jsou taktéž metody pro vkládání, odebírání a hledání záznamů v databázi.

Jakmile je databáze vytvořena je potřeba ji naplnit základními informacemi. Pro vytvoření nového záznamů o vozidle slouží aktivita `AddVehicleActivity`. Ta do databáze vloží zadaný název automobilu a všechna ostatní data vyplní jako prázdná. Pro přidání vozidel využívá aktivita `AddVehicleActivity` tzv. fragment, konkrétně `AddVehicleFragment`. Fragmenty ve své podstatě vytvářejí uživatelské prostředí aktivit, které se může v průběhu běhu aktivity měnit. Součástí jedné aktivity může být až několik fragmentů. Fragment má také, podobně jako aktivita, svůj životní cyklus.

`AddVehicleFragment` vytváří prostředí pro přidání vozidla. Dalším fragmentem je `ChooseDataFragment`. Ten vytváří seznam s již uloženými vozidly v databázi a zobrazuje je v aktivitě `VehicĺeDatabaseActivity`. V této aktivitě se následně odehrává samotné ukládání, výběr a odstraňování dat. Při výběru automibilu, pro který má aplikace shromažďovat v danou dobu data, se vytvoří nový záměr (Intent), který spustí následující aktivitu pro Bluetooth komunikaci. Záměr dostane extra informaci o tom, jak se dané vozidlo jmenuje a předá ji `BTActivity`.



Obr. 4.1: Vývojový diagram Aplikace

Poslední třídou pro komunikaci s databází je třída `AddVehicleData`. Jedná se o objekt typu Parcelable. V podstatě se jedná o strukturu, do které je možno přistupovat kdykoli při běhu aplikace přes prostředí Bundle. Tento objekt se dá také přidat k záměru jako extra informace. Objekt přenáší několik informací po každém jednom cyklu v `BTActivity`. Tyto informace si následně vyžádá `VehicÍeDatabaseActivity` a uloží do databáze.

Bluetooth

Pro obsluhu Bluetooth komunikace, mezi mobilním přístrojem a ELM327, se většinu času běhu aplikace stará třída `BTActivity`. Jako první věc, kterou provede je

zapnutí Bluetooth modulu v přístroji (pokud již nebyl zapnutý). Po zapnutí Bluetooth se inicializuje uživatelské prostředí a následně se začnou provádět příkazy pro inicializaci připojení.

Součástí `BTActivity` je také třída typu `Handler`. Tato třída slouží v Androidu pro obsluhu systémových zpráv. To znamená, že při změně stavu adaptéru Bluetooth, například ze stavu `STATE_CONNECTING` do stavu `STATE_CONNECTED`, pošle systémovou zprávu, kterou třída `Handler` vyhodnotí a vykoná příslušný kód. Uvedené stavy jsou součástí třídy `BluetoothAdapter`. `Handler` se stará o komunikaci se třídou `BTCommunication`. `Handler` přijímá a odesílá zprávy přes Bluetooth. Při přijetí zprávy následuje příkaz `switch`, který rozlišuje přijaté PID kódy a zobrazuje je na obrazovce. Při příjmu zprávy se vybrané informace zároveň ukládají do `AddVehicleData`.

Třída `BTCommunication` obsahuje metody pro řízení komunikace. Základním stavebním prvkem komunikace jsou dvě privátní třídy a to `ConnectingThread` a `Connected Thread`. Při inicializaci komunikace pomocí protokolu SDP by byla potřeba ještě jedna třída a to `AcceptedThread`. SDP protokol zjišťuje nabízené služby ostatních Bluetooth modulů v okolí. Může se tak stát například u aplikací pro komunikaci nebo sdílení souborů, kdy je přístroj v pozici klienta, toho co přijímá data. Úkolem `AcceptedThread` by bylo přijmout požadovanou službu od vysílací stanice. Jelikož aplikace bude využívat pouze sériovou komunikaci a nebude muset nikdy přijímat volání od ELM327 (přijímá pouze odpovědi), postačí pouze třídy dvě. `ConnectingThread` zahajuje komunikaci, vyhledá zařízení v automobilu a připojí se k němu pomocí protokolu RFCOMM. Mobilní telefon se tak chová pouze jako server. `ConnectedThread` vykonává operace za účelem samotné komunikace. Pomocí její metody `run()` probíhá čtení a zápis do respektive ze zařízení. Při zápisu je vstupní zpráva převedena na bity, při čtení je z bitů vytvořen řetězec znaků.

Bezprostředně po startu `BTActivity` se spustí aktivita `BTList`. Pomocí ní si uživatel zvolí, ke kterému zařízení se připojí. Na výběr jsou spárovaná nebo nespárovaná zařízení. Po výběru vyšle informaci o zvoleném zařízení a začíná pokus o připojení vybraného zařízení.

Debuger

Třída `Debuger` slouží k přímé komunikaci s ELM327. Jedná se o příkazovou řádku, do které uživatel zapisuje PID kódy a na obrazovce se mu zobrazují odpovědi od ELM327. Tyto odpovědi jsou v původním tvaru, to znamená, že na obrazovce budou zobrazena hexadecimální čísla a uživatel si je musí sám dekódovat. Tato funkce je spíše pro uživatele, kteří ví co dané PID kódy znamenají a vědí, jak s nimi nakládat.

Grafy

Součástí aplikace je také soubor tříd, které vykreslují grafy. Těmito třídami jsou `GraphActivity` a `GraphList`. `GraphActivity` řídí výběr informací a posílá data do třídy pro vykreslení grafu. `GrapfList` pak slouží k výběru typu grafu, který chce uživatel zobrazit.

Pro vykreslování grafů aplikace využívá knihovnu AndroidPlot dostupnou na webové adrese www.androidplot.com. Součástí této knihovny jsou metody pro vykreslování jak statických dat, tak pro vykreslení funkcí. Vykreslovat se dají i real-time data při běhu aplikace. Funkce vykreslování real-time dat není v aplikaci dostupná, především kvůli náročnosti přenosu dat do mobilního zařízení. Mohlo by to například znamenat, že by přístroj s menším výpočetním výkonem nebyl schopen data vykreslovat. Využita je pouze součást pro vykreslení statických dat. Aplikace totiž nezaznamenává žádná data pro výstup v podobě funkce.

Vodorovná osa (x) vždy zobrazuje datum, kdy byla hodnota zaznamenána a svislá osa (y) zobrazuje vybraná data.

4.3 Uživatelské prostředí aplikace

Jak již bylo řečeno, aplikace je velmi jednoduchá a nepotřebuje příliš mnoho vstupů od uživatele. Po spuštění uživatel pouze vybere, k jakému vozidlu chce shromažďovat informace. Poté je vyzván k zapnutí Bluetooth v přístroji (pokud nebylo zapnuto) a následně vybere zařízení, ke kterému se chce připojit. Aplikace se ještě zeptá, zda-li chce uložit zařízení do paměti a automaticky se k němu připojovat při dalším spuštění.

Jakmile je mobilní přístroj připojen, zobrazí se obrazovka s několika informacemi, které jsou defaultně nastaveny. Vybrané informace se ukládají zároveň i do databáze. Na stavovém řádku jsou tlačítka pro spuštění dalších aktivit, jako je debuger nebo aktivita pro vykreslování grafů. Vzhled těchto aktivit je zobrazen na obrázcích 4.2, 4.3.

Na obrázku 4.4 jsou vidět některé informace, které jsou defaultně vybrány a nelze je jakkoliv přesunovat či odebírat. Z těchto informací, hlavně z první čtveřice, se získávají data, která se ukládají do databáze. Ostatní jsou zde jen na ukázku komunikace. Při nalezení nějaké chyby (trouble kódu), se pod výčtem informací zobrazí tlačítko s textem „Ukaž DTC“. Po zmáčknutí se objeví dialog, ve kterém budou zobrazeny čísla těchto chyb. Uživatel má pak na výběr, zda bude toto varování ignorovat nebo zda tyto chyby vymaže z řídící jednotky automobilu. Při mazání těchto chyb se nemusí uživatel bát, že by něco pokazil na automobilu. Pokud by chyba byla trvalejšího rázu, při dalším nastartování motoru by se obnovila a znovu

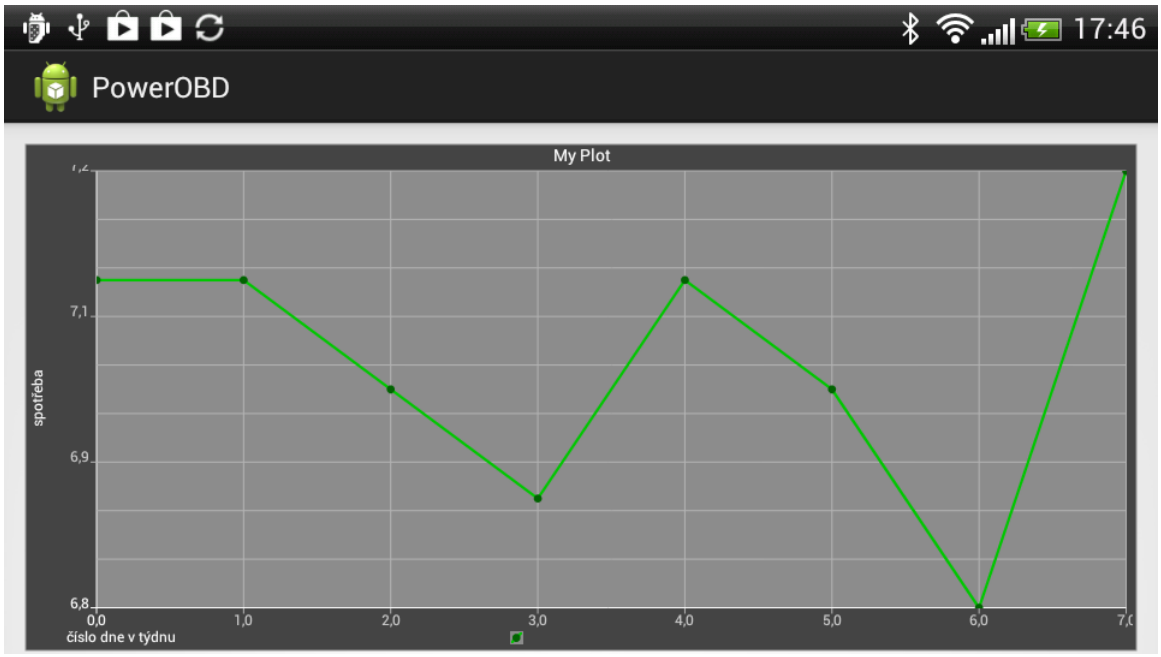
Obr. 4.2: Prostředí Debuger

ukázala. Někdy se může stát, že čidla, která kontrolují stav vozidla, mohou mít výrobní vadu a zobrazí nějakou chybu, třebaže o blíže specifikovanou chybu nejde.

Vpravo nahoře se při natočení telefonu na podélnou stranu zobrazí stavová řádka. Z této řádky lze spustit několik dalších součástí aplikace. Jedná se o Debuger (terminál) nebo aktivitu s grafy a dalšími informacemi.

V Debugeru uživatel může vysílat své doplňující PID kódy. Funguje to podobně jako Hyperterminál v OS Windows. Jedná se v podstatě o textovou řádku, kde se zapisují PID kódy a pod touto řádkou se zobrazují odpovědi od ELM327. Odpovědi nejsou nijak upraveny do formy, které by uživatel ihned porozuměl. Odpověd je stejná jak je napsáno v 3.3.2. Tato aktivita je více či méně na otestování samotné komunikace. Pro zkušenější uživatele může sloužit jako přístup k vícero informacím. Tato komunikace se ovšem neukládá do databáze.

Při spuštění grafu je uživatel dotázán, jaký typ grafu chce zobrazit. Po výběru typu se příslušný graf vykreslí na obrazovce. Uživatel může zvolit ze 6 grafů. Jedná se o grafy jako ujetá vzdálenost za den, případně i delší časový horizont (týden, měsíc), nebo vývoj průměrné spotřeby v určitém časovém horizontu. Grafy mohou



Obr. 4.3: Ukázka vykreslení grafu

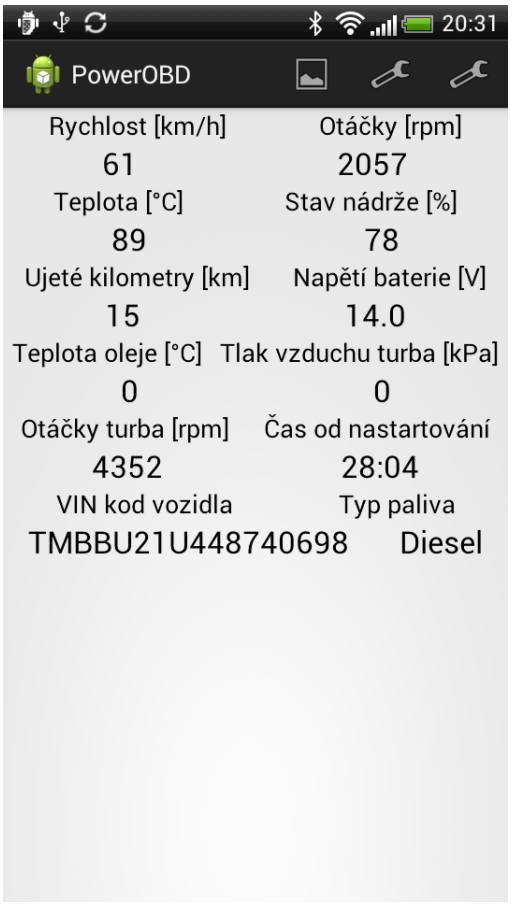
být typu křivky nebo typu sloupečkového, vše záleží na uživateli, co si při výběru zvolí.

Poslední aktivita, která se dá spustit přes stavovou řádku, je aktivita doplňkových informacích. Zde jsou zobrazeny informace, které se vypočítavají z dat uložených v databázi. Jsou jimi data jako průměrná nebo aktuální spotřeba paliva nebo aktuální dojezd. Například výpočet aktuální spotřeby probíhá tak, že se pomocí PID kódu změří průtok paliva a vydělí se aktuální rychlostí, kterou automobil jede. To znamená, že pokud automobil jede 100km/h aktuální spotřeba bude přímo aktuální průtok paliva.

4.4 Testování aplikace

Testování aplikace probíhalo na vozidle Škoda Octavia první generace a na mobilním telefonu HTC Sensation XE s verzí Android API 15 Ice Cream Sandwich. Na tomto telefonu byla aplikace také vyvíjena. Na obrázku 4.5 je zobrazeno umístění interpreteru s ELM327 ve vozidle. U Škody Octavia se OBD konektor nachází přímo pod volantem.

Prvním z mnoha testů bylo to, zda aplikace neomezuje funkčnost telefonu jako takového. Zda při svém běhu neruší telefonní hovory, nespotřebovává mnoho energie a podobně. Při 10 pokusech o navázání telefonního hovoru, se vždy aplikace uchýlila do stavu pozastaveno (pause). Po ukončení hovoru začala zase běžet tam, kde pře-



Obr. 4.4: Prostřední aplikace

stala. Při přerušení aplikace nastává však ukončení komunikace mezi automobilem a mobilním přístrojem. Tento jev by se dal odstranit například změnou struktury aplikace. Řídící aktivita aplikace by nemohla být aktivitou ale službou, tím pádem by běžela pořád na pozadí a shromažďovala informace neustále. To by ovšem mělo jisté následky, například to, že by byl pořád využíván modul Bluetooth pro komunikaci a spotřeba energie telefonu by rapidně stoupla. Služba by běžela i v případě, pokud by zařízení nebylo připojeno k modulu v automobilu a pořád by jej vyhledávalo. Pokud by byla spotřeba energie vysoká, jistě by nebyli uživatelé spokojeni s tím, že by museli telefon neustále nabíjet. Proto je zřejmě lepší, když aplikace bude shromažďovat informace pouze při jejím běhu.

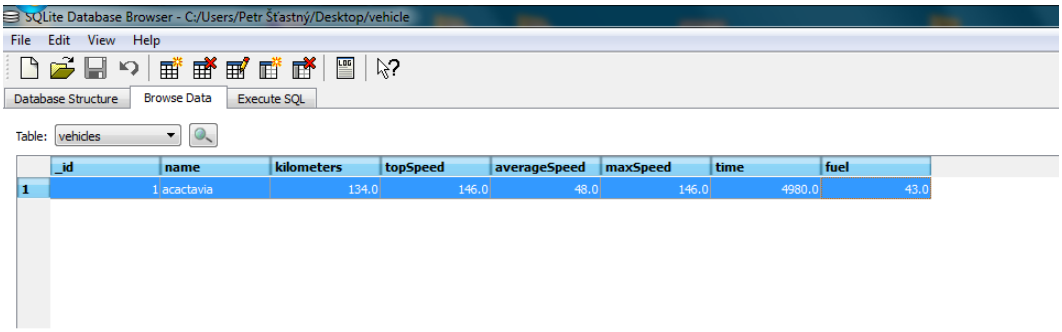
Jako další se testovala samotná komunikace. Začátek testu probíhal za pomocí Debugeru. Byly vyzkoušeny všechny možné PID kódy, které využívá aplikace v aktivitě BTActivity. Dále se testovaly některé vybrané AT příkazy pro případné rozšíření aplikace v oblasti nastavování komunikace a podobných příkazů. Jakmile bylo vše v pořádku, přešlo se na testy aktivity BTActivity. V tomto testu záleželo hlavně na tom, jak bude aktivita stíhat obnovovat hodnoty na obrazovce. Výsledná perioda

Obr. 4.5: Umístění interpreteru ve vozidle

aktualizace PID kódů se pohybovala okolo 2 sekund a to při rotaci výčtu 10 PID kódů. Poslední 2 PID kódy v aktivitě (na obrázku 4.4 úplně v dolním řádku) se aktualizují pouze jednou při startu aktivity. Jedná se o informace, které se při běhu nemůžou měnit. Výsledná perioda by mohla být přijatelná vzhledem k tomu, že se jedná o komunikaci přes Bluetooth a současně se vysílá a přijímá 10 PID kódů.

Při prokázání příjmu PID kódů a zjištění, že se průběžně aktualizují, bylo potřeba ověřit, zda funguje i zvolený způsob ukládání informací do databáze. Při zobrazení obsahu databáze přes nástroj adb v balíku Android SDK. Přes konzoli adb se dá připojit k přístroji s Androidem pro zobrazení různých systémových informací, které nejsou z telefonu přístupné. Součástí je také shell pro sqlite3 databáze, který má všechny potřebné nástroje pro nakládání s SQLite databázemi. Bohužel se z telefonu nepodařilo získat výpis databáze a její schéma. Uživateli totiž nebyla udělena práva pro zápis a čtení ze složky, kde se tento soubor nachází. Pro ověření funkčnosti byl tedy využit emulátor na počítači. Protože nemůže tento emulátor využívat Bluetooth, je tento test proveden tak, že do databáze vstupují určité hodnoty, které by mohli vstupovat přes Bluetooth. Výpis je pak vidět na obrázku 4.6

Pokud tyto všechny testy proběhly úspěšně, je už jen potřeba ověřit, zda se dá využít nashromážděná data v databázi. K tomu nám poslouží pouze 2 věci. Aktivita pro dodatečné informace a aktivita grafů. Vypočítaná průměrná spotřeba aplikací a průměrná spotřeba vypočítaná palubním počítačem vozu se liší o 0,1 l/100km. Tato odchylka může být způsobena různými okolnostmi. Jednou z mnoha okolností může být jiný přístup k výpočtu tohoto údaje. Aplikace jej počítá z počtu ujetých kilometrů, průměrné rychlosti a celkového času provozu. Způsob výpočtu palubního



Obr. 4.6: Výpis databáze

počítače se v podstatě zjistit nedá, jelikož se jedná o know-how automobilky.

Při zobrazení jakéhokoli grafu je ověřeno jestli funguje aktivita, která tyto grafy kreslí, a funguje-li výčet informací z databáze. Výsledek zobrazuje obrázek 4.3.

ZÁVĚR

Předmětem této práce bylo navrhnout komplexní aplikaci pro výčet a přenos provozních hodnot automobilu na mobilní zařízení. Řešení diagnostiky palubního počítače vozidla bylo po seznámení s patřičnou problematikou věci ověřeno na testovací aplikaci. Ta byla po ověření funkčnosti rozšířena o modul s databází. Aby byla aplikace využitelná v praxi, bylo potřeba navrhnout způsob, jak tyto informace v databázi využít. K tomu slouží součásti spojené s vykreslováním informací do grafů. Aplikace se liší od standardních palubních počítačů v automobilech tím, že je schopna využít potenciál rozhraní OBD-II. Na rozdíl od palubních počítačů umí lépe specifikovat chybová hlášení, které automobil zobrazí. Jelikož umí shromažďovat dlouhodobá data, může být více nápomocna řidiči než palubní počítač.

Aplikace byla otestována na mobilním telefonu HTC Sensation XE s Android API 15 Ice Cream Sandwich a na automobilu Škoda Octavia první generace.

Jednoduchými úpravami je možné tuto aplikaci dále rozšiřovat, například přidáním dalších měření nebo zobrazováním více grafických údajů. Aplikace by se také dala doplnit o grafický engine. Mohla by být více komplexní. Při jejím běhu by se dalo například přepínat na navigaci, média nebo využívat telefonní hovory. Tyto všechny funkce by byly využívány přímo z telefonu, takže by nebylo potřeba velkých zásahů do stávajícího kódu.

LITERATURA

[1] *On-Board Diagnostics* [online]. 2004, poslední aktualizace 29. 10. 2012 [cit. 1. 11. 2012]. Dostupné z URL: <http://en.wikipedia.org/wiki/On-board_diagnostics>.

[2] *What is OBD? | OBD Solution* [online]. 2012, poslední aktualizace 29. 10. 2012 [cit. 1. 11. 2012]. Dostupné z URL: <http://www.obdsol.com/wp-content/uploads/2011/02/obd_system_components.jpg>.

[3] *Keyword Protocol 2000* [online]. 2004, poslední aktualizace 2. 9. 2012 [cit. 1. 11. 2012]. Dostupné z URL: <http://en.wikipedia.org/wiki/Keyword_Protocol_2000>.

[4] *CAN in Automation (CiA): CAN history* [online]. 2001, poslední aktualizace 16. 5. 2012 [cit. 1. 11. 2012]. Dostupné z URL: <http://www.can-cia.de/index.php?id=161>.

[5] *Android* [online]. 2005, poslední aktualizace 27. 6. 2012 [cit. 7. 11. 2012]. Dostupné z URL: <http://www.android.com>.

[6] MURPHY, Mark L. *Android 2: Průvodce programováním mobilních aplikací*. 1. vyd. Brno, Computer Press, 2011. 375 s. ISBN 80-2513-194-7

[7] *ELM327 data sheet)* [online pdf]. 2008, poslední aktualizace 31. 10. 2012 [cit. 7. 11. 2012]. Dostupné z URL: <http://www.elmelectronics.com/DSheets/ELM327DS.pdf>.

[8] J1979. *E/E Diagnostic Test Modes*. Warrendale: Society of Automotive Engineers, 2012.

[9] *CAN Pro-Series OBDII Trouble Code Reader. In: Diydiagnostics* [online]. 2011. [cit. 2012-12-10]. Dostupné z URL: <http://www.diydiagnostics.com/images/canProseries_large.jpg>.

SEZNAM SYMBOLŮ A ZKRATEK

ADT Android Development Tools – plug-in pro Eclipse

API Application Programming Interface – rozhraní pro software

Bd Baud – jednotka změny stavu

CAN Controller Area Network – přístupový protokol

COM Component Object Model – přenosový port

CPU Central Processing Unit – procesor

CRC Cyclic Redundancy Check – cyklický kontrolní součet redundance

CSMA/NDA Carrier Sense Multiple Access with Non-Destructive Arbitration – přístupová metoda

DLC Diagnostic Link Connector – obecné rozhraní pro komunikaci s automobilem

DTC Diagnostic Trouble Codes – chybové hlášení

DVM Dalvik Virtual Machine – překladač pro mobilní telefony s Android

ECU Electronic Control Unit – řídící jednotka automobilu

GPS Global Positioning System – globální poziční systém

HDML Handheld Device Markup Language – programovací jazyk

MIL Malfunction Indicator Light – kontrolka motoru na palubni desce

NDK Native Development Kit – nástrojem pro vývoj v C, C++

OBD On-Board Diagnostic – standard pro komunikaci s automobilem

PID Parameter ID – kód pro komunikaci s OBD

QEMU Quick Emulator – software pro virtualizaci

SAE Society of Automotive Engineers – společenství automobilových inženýrů

SDK Software Development Kit – balík vývojových nástrojů

SMS Short Message – textová zpráva

STK Státní technická kontrola

UART Universal asynchronous reciever/transmitter – typ komunikačního prorokolu

USB Universal Serial Bus – přenosové médium

UUID Universally Unique Identifier – standard pro identifikaci