VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

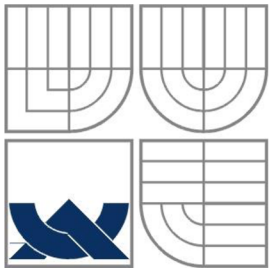
APLIKACE PRO SBĚR A VYHODNOCENÍ DAT Z ROZHRANÍ OBDII PRO ANDROID

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

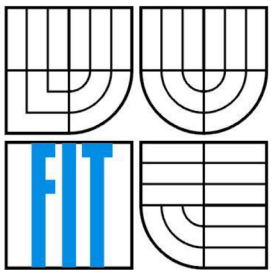
AUTOR PRÁCE
AUTHOR

Bc. MICHAL KABELKA

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

APLIKACE PRO SBĚR A VYHODNOCENÍ DAT Z ROZHRANÍ OBDII PRO ANDROID

APPLICATION FOR DATA ACQUISITION AND PROCESSING FROM OBDII INTERFACE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAL KABELKA

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Dr. Ing. PAVEL ZEMČÍK

BRNO 2015

Abstrakt

Táto práce se zabývá problematikou získávání dat z automobilu přes rozhraní OBDII. Postupně rozebírá měřené veličiny, senzory a také řídicí systémy v automobilech a princip komunikace mezi nimi. Shrnuje základní principy tvorby mobilních aplikací pro systém Android. Dále se zaměřuje na návrh aplikace, která využívá mikrokontrolér ELM327 pro komunikaci s elektronickými systémy v automobilu. Získané data zobrazuje aplikace formou, díky které může zlepšit ekonomické návyky řidiče. Vytvořená aplikace také demonstruje možnosti komunikace s automobilem mimo standardu OBDII, v tomto případě ovládáním vestavěného rádia v automobilu.

Abstract

This thesis is focused on obtaining data from a car via the OBDII interface. A successive analysis of the measured values, car sensors, control systems and communication principles between them is done. Basic principles for creating Android mobile applications have been also summarized here. However, the main focus is on designing an application which uses a microcontroller ELM327 to communicate with the electronic systems of a real car. The application shows the obtained data in a form that enables the driver to improve his/her driving habits in terms of fuel economy. The application also demonstrates communication with a car outside of the OBDII standard, in this scenario, by controlling a built-in radio in the car.

Klíčová slova

Android, OBD, OBDII, ELM327, CAN BUS, Bluetooth

Keywords

Android, OBD, OBDII, ELM327, CAN BUS, Bluetooth

Citace

Kabelka Michal: Aplikace pro sběr a vyhodnocení dat z rozhraní OBDII pro Android, diplomová práce, Brno, FIT VUT v Brně, 2015

Aplikace pro sběr a vyhodnocení dat z rozhraní OBDII pro Android

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Prof. Dr. Ing. Pavla Zemčíka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Kabelka
26.7.2015

Poděkování

Děkuji panu Prof. Dr. Ing. Pavlu Zemčíkovi za odborné vedení této diplomové práce.

Obsah

1 Úvod

V posledních několika desetiletích zaznamenala oblast informačních technologií obrovský růst. Různá elektronická zařízení se stala součástí našeho běžného života, aby nám mnohdy ulehčila práci nebo poskytla zábavu. Tento trend se samozřejmě nemohl vyhnout ani automobilovému průmyslu, kde moderní automobily jsou protkány mnohými elektronickými systémy. Těch multimediálních si je většina lidí vědoma, zatímco řídicí systémy zůstávají pro běžného uživatele skryté.

Po letech relativně nekoordinovaného vývoje bylo potřeba standardizovat alespoň část těchto systémů. Proto vznikl koncem devadesátých let standard OBDII, který umožnil provádět diagnostiku některých systémů v automobilech jednotným způsobem, bez ohledu na to, kdo automobil vyrobil. Tento standard zastřešuje převážně ty části automobilů, které jsou zodpovědné za množství emisí produkovaných vozidlem. Hlavně jsou to tedy informace o motoru, ale obsahuje i některé jiné, které mohou být zajímavé pro běžného řidiče. Bohužel zbylé systémy a informace, které nejsou součástí tohoto standardu, zůstávají aj nadále specifické pro konkrétní automobily a výrobci k nim úmyslně neposkytují bližší informace.

S masivním rozšířením mobilních zařízení, jako jsou chytré telefony a tablety, se začaly na trhu ve velkém objevovat jakási rozhraní, která dokáží zprostředkovat informace z automobilu přímo na displej těchto zařízení. Na jejich základě pak vzniklo velké množství aplikací pro mobilní zařízení, která využívají data dostupná díky standardu OBDII a prezentují je uživateli nejrůznějšími způsoby.

Ačkoliv takovýchto aplikací existují stovky, neexistuje prakticky žádná, která by se snažila získávat také informace neobsaženy v tomto standardu nebo používala pro komunikaci s vozidlem způsob specifický pro konkrétní vozidlo.

Cílem této práce je zmapovat podmínky pro vytvoření systému, který dokáže získávat od vozidla data dostupná přes standard OBDII a na jejich základě poskytnout řidiči informace, která v některých automobilech chybí. Také se pokusit na testovaném automobilu o zpřístupnění nestandardní funkcionality, jako třeba ovládání rádia automobilu přes mobilní aplikaci. Dalším cílem je takovýto systém navrhnout a vytvořit pro něj aplikaci určenou pro operační systém Android.

Celý dokument je rozdělen do několika kapitol. V kapitole 2 budou popsány elektronické systémy a informační technologie, která se používají v současných automobilech a také způsob jakým mezi sebou systémy v automobilech komunikují. Kapitola 3 popíše systém Android a základní principy tvorby aplikací pro tento systém. Dále v kapitole 4 budou zhodnocena současná existující řešení, jak hardwarová, tak softwarová a bude udělán náčrt budoucího systému. V kapitole 5 bude rozebrán podrobněji návrh aplikace celého systému pro komunikaci s vozidlem a zobrazení přijatých dat. Následující kapitola 6 se věnuje samotné realizaci navrženého systému. V kapitole 6.8 bude popsáno testování systému v automobilu za jízdy a v závěru budou shrnuty dosažené výsledky realizovaného systému a diskutovány jeho možné další vylepšení.

2 IT technologie ve spojení se současnými automobily

Současné automobily jsou složité a komplexní systémy a každým rokem se stávají ještě složitější a komplexnější. Může za to jednak velký rozvoj výpočetní techniky v poslední době, ale také snaha výrobců o zjednodušení návrhu, či nutnosti dodržování přísných emisních norem. Do automobilů se proto integruje mnoho mikroprocesorů, které zabezpečují různé funkce, od řízení motoru přes diagnostiku až po ovládání rádia či zrcátek. V této kapitole budou diskutovány některé důležité části těchto systémů a hlavně principy, na kterých funguje komunikace mezi těmito systémy.

Následující část rozhodně není encyklopedickým výpisem všech vědomostí daného oboru, snaží se spíš přiblížit a osvětlit fakta potřebná pro pochopení této práce. Ačkoli existuje snaha o standardizování systémů používaných v automobilech, mnoho výrobců v současnosti používá vlastní systémy, pro které navíc neexistují veřejně dostupné dokumentace. Některé informace v této práci je tedy možné brát jako všeobecně platné, jiné jsou naopak úzce spjaté s konkrétním vozidlem. Pro testování této práce bylo k dispozici vozidlo Opel Astra H GTC s rokem výroby 2006, proto budou všechny informace vztaženy právě k tomuto vozidlu. Pokud se bude některá informace týkat jenom konkrétního automobilu, či systému, bude na to v dalším textu upozorněno.

2.1 Řídicí systémy v automobilech

Současným automobilům už nestačí k činnosti jeden řídicí systém, ale používají jich od několika jednotek a po několik desítek kusů [26]. V následujících odstavcích budou popsány základní z nich.

ECU – Řídicí jednotka

ECU je zkratka z anglického Electronic Control Unit nebo též elektronická řídicí jednotka. Je to název pro vestavěné elektronické zařízení, které čte a zpracovává signály ze senzorů umístěných různě ve vozidle a na základě dat z těchto senzorů provádí požadované automatizované úkony. V moderních vozidlech je standardně obsaženo několik různých řídicích systémů. Tady je výčet několika základních, které se nacházejí prakticky v každém moderním vozidlu [2]:

- ECM – Engine Control Module – nebo také řídicí jednotka motoru (někdy výrobci označována také jako ECU – Engine Control Unit, což je mírně matoucí, proto dále v této práci bude řídicí jednotka motoru označována jak ECM a výraz ECU bude používán pro jakoukoliv řídicí jednotku ve vozidlu) [2].
- EBCM – Electronic Brake Control Module – nebo také elektronický systém pro řízení brzd – stará se například o funkci ABS (Anti Blocking System). Je to systém vyvinutý v sedmdesátých letech minulého století pro zlepšení brzdných vlastností vozidel, bez ohledu na stav silnice nebo povětrnostní podmínky [2].
- PCM – Powertrain Control Module – je řídicí jednotka, která kontroluje rychlost vozidla, řazení v případě automatické převodovky a sleduje časování ventilů [2]. PCM někdy kombinuje dohromady řídicí jednotku motoru (ECM) a řídicí jednotku převodovky [26].

- VCM – Vehicle Control Module – je řídící jednotka, která má na starosti bezpečnostní systémy ve vozidlu jako ESC (Electronic Stability Control – elektronický stabilizační systém), ACS (Airbag Control System – systém řízení airbagů), ale také EPS (Electronic Power Steering – elektronický posilovač řízení). VCM je typicky instalována ve střední části vozidla, mezi kabinou a motorovým prostorem [26]. Je připojena k mnoha senzorům tak, aby ovládala různé systémy ve vozidlu. Jednotka také přijímá údaje od nárazových senzorů (mikro akcelerometry), senzorů detekujících váhu cestujících, polohu sedadla, použití bezpečnostních pásů, a to za účelem určení síly s jakou mají být čelní airbagy rozbaleny. Podobně přijímá i data od senzoru natočení volantu, senzoru rychlosti kol, senzoru bočního (laterálního) zrychlení, aby poskytla ESC všechny potřebné vstupy pro maximální bezpečnost za jízdy [2].
- BCM – Body Control Module – je řídící jednotka, která se stará o činnost například elektronického řízení sedadel, kontrolu stěračů, elektrického stahování oken nebo třeba o elektronické stahování střechy u kabrioletů [2].

Mnoho moderních, hlavně luxusních automobilů obsahuje desítky dalších řídicích systémů, které jsou ale opět závislé na konkrétním modelu a jejich znalost není stěžejní pro pochopení této práce, proto je tady zmiňovat nebudeme.

Komponenty ECU

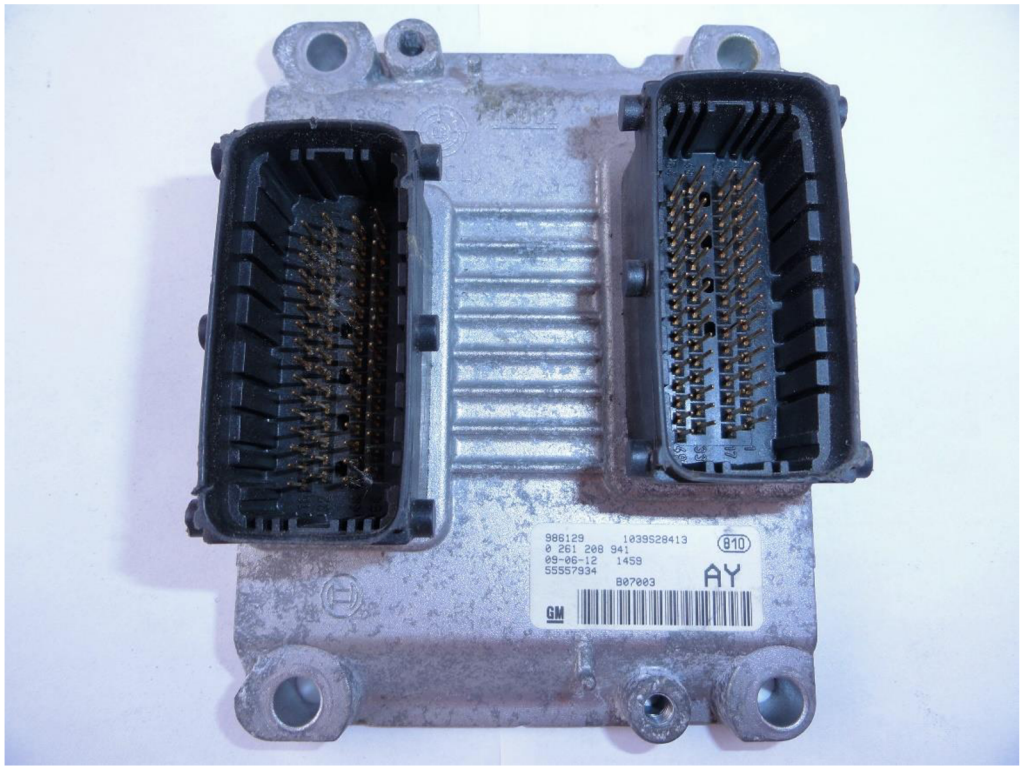
Procesor je součástí jednoho modulu spolu se stovkami dalších komponent. V dalším textu se podíváme na ty nejzákladnější [1]:

- Analogově-digitální převodníky (A-D) – signál z analogových senzorů je potřeba převést do digitální reprezentace. Například lambda sonda (senzor množství kyslíku ve výfukových plynech) má většinou výstup v rozmezí 0-1,1V. Tento signál je potřeba převést na 10-bitovou reprezentaci [1].
- Vysoko-úrovňové digitální výstupy (High Level Digital Outputs) – na moderních automobilech ECU řídí zapalovaní svíček, otevírá nebo zavírá vstřikovače nebo spouští ventilátor chlazení. K tomu je potřeba mnohem vyššího napětí a proudu než je schopen procesor jednotky poskytnout. Jedná se v podstatě o relé, kde procesor malým proudem pomocí tranzistoru spíná mnohem větší proud, který je již pak dostačující pro sepnutí opravdového relé, například na ventilátoru chlazení. Relé pak je schopno poskytnout proud řádově v desítkách ampérů [1].
- Digitálně-analogové převodníky (D-A) – pro řízení některých částí motoru je potřeba analogového signálu. Protože řídící jednotka je digitální zařízení je potřebné převést tento signál na analogové napětí [26].
- Úprava signálů (Signal Conditioners) – některé vstupy je potřeba upravit předtím než mohou být přečteny. Například A-D převodník může být nastaven tak, aby přijímal signály v rozmezí 0-5V, ale lambda sonda poskytuje signál v rozmezí 0-1,1V. Proto je potřeba tento signál upravit před dalším zpravováním. Obvod pro úpravu signálu pak upraví signál tak, že výstupní napětí z lambda sondy vynásobí čtyřmi. Získáme tak signál v rozmezí 0 až 4,4V, což umožní A-D převodníku číst napětí s mnohem větší přesností [1].
- Komunikační obvody – tyto obvody mohou implementovat různé standardy, ale až na výjimky dnes výrobci prakticky ve všech vozidlech používají rozhraní CAN BUS (z anglického Controller-Area Network) [4]. Existuje několik používaných variant a rychlostí

této technologie. Rozhraní CAN BUS se budeme podrobně věnovat v kapitole 2.33 této práce [26].

ECM – řídící jednotka motoru

ECM, zkratka z anglického Engine Control Module, nebo také řídící jednotka motoru, je mikroprocesor, který řídí chod motoru. Kdysi, když ještě neexistovali emisní limity, bylo možné postavit motor bez řídící elektroniky. Dnes se zavedením přísných emisních norem jsou potřeba sofistikované řídící procesy k regulování bohatosti směsi paliva a vzduchu tak, aby katalyzátory (zařízení pro snížení množství škodlivin ve výfukových plynech [19]) mohly odstranit většinu znečištění z výfukových plynů. Řízení motoru je výpočetně náročný proces, kde řídící jednotka motoru musí vykonat miliony operací za vteřinu. K tomu potřebuje posbírat data z desítek senzorů, od teploty chladicí kapaliny až po množství kyslíku ve výfukových plynech. Nad těmito daty pak provádí řídící jednotka motoru výpočetní operace, od jednoduchých jakými je vyhledávání hodnot v tabulce až po vyhodnocování složitých rovnic. Vše proto, aby zabezpečila co nejefektivnější spalování paliva a tím i co nejmenší spotřebu a množství emisí. [1]



Obrázek 2.1: Ukázka ECM Bosch motoru Z14XEP z automobilu Opel Astra H[1].

2.2 Senzory v automobilech

Získávat hodnoty ze senzorů v automobilu umožňuje standard OBDII (viz sekce 2.3). Znalost činností těchto senzorů není bezpodmínečně nutná pro pochopení této práce, nicméně získávání dat

[1] Zdroj: http://www.megavaux.co.uk

tvoří základ aplikace, tvorbě které se tato práce podrobně věnuje. Proto následuje popis nejdůležitějších signálů, které ECM používá pro výpočet správného vstřikování paliva do motoru pro efektivní spalování. [2]

- Senzor teploty chladicí kapaliny (Coolant Temperature Sensor – CTS)
  CTS senzor používá termistor pro zjištění teploty chladicí kapaliny v motoru [2]. Termistor je jednoduchá součástka, která mění svůj odpor v závislosti na teplotě. Podle typu senzoru může odpor s rostoucí teplotou buď růst, nebo klesat. U prvního typu NTC (Negative Temperature Coefficient) vnitřní odpor s rostoucí teplotou klesá a obráceně. Druhý typ PTC (Positive Temperature Coefficient) pracuje přesně obráceně. V automobilech se ve většině případů používá první varianta a měření samotné teploty probíhá tak, že řídící jednotka pošle na senzor referenční napětí (většinou 9V) [1]. Jelikož se změnou teploty se mění také odpor, mění se úbytek napětí na výstupu senzoru. Z rozdílu referenčního a výstupního napětí na senzoru je pak řídící jednotka schopna vypočítat teplotu. Pro ECM je teplota motoru nezbytný údaj, protože studený motor potřebuje jiné časování a poměr paliva než motor v provozní teplotě [19]. ECM tento údaj o teplotě posílá také dál po CAN sběrnici, aby ji mohl přečíst a použít libovolný modul, který tento údaj potřebuje (například Instrument Cluster – přístrojová deska, která ji zobrazí jako informaci pro řidiče).

- Lambda sonda (Oxygen Sensor – O2S)
  Součástí motoru v každém moderním automobilu je lambda sonda, nebo též kyslíkový senzor. Tento senzor je součásti systému pro kontrolu emisí a poskytuje data pro ECM. Úlohou senzoru je pomoci motoru běžet co nejefektivněji a zároveň produkovat co nejméně emisí. Benzínové motory spalují palivo za přítomnosti kyslíku. Bylo zjištěno, že ideální poměr mezi vzduchem a benzínem je 14,7:1 [19] (jiné druhy paliva mají jiný poměr). V případě, že je k dispozici méně vzduchu, než je ideální poměr, bude zůstávat nespálené palivo. Tomuto se říká bohatá směs, která způsobuje znečistění ovzduší ve formě prachových částic (u dieselových motorů je toto znečištění kvůli technologii vstřikování velký problém, proto se řeší pomocí známého filtru pevných částic, nebo vstřikováním močoviny do výfukového potrubí – tzv. AdBlue) [3]. Pokud máme více vzduchu, než říká ideální poměr, pak máme v spalovací komoře přebytek kyslíku a mluvíme o chudé směsi. Při této chudé směsi zase vzniká více znečišťujících látek oxidu dusíku, a navíc v takovém případě vzniká ve válcích vyšší teplota, což může v extrémních případech vést k poškození motoru [19].

  Důvod proč řídící elektronika motoru potřebuje lambda sondu je ten, že množství kyslíku, které je motor schopen nasát, závisí od mnoha faktorů, jakými jsou nadmořská výška, teplota vzduchu, teplota motoru, atmosférický tlak, zatížení motoru a další [19]. Proto je nutné množství vstřikovaného paliva upravovat podle aktuálních podmínek v reálním čase.

  Z hlediska fyzické konstrukce se jedná o relativně jednoduchý senzor, který je umístěný ve výfukovém potrubí (v moderních automobilech se používá trojcestný katalyzátor, který disponuje dvěma lambda sondami – jednou před a jednou za katalyzátorem [19]) a který porovnává okolní (referenční) vzduch s výfukovými plyny [1]. Senzor je vyroben z keramického zirkonia, které vytváří napětí přibližně od 1V při bohaté směsi paliva po 0V při směsi chudé. Pro svoji činnost potřebuje být senzor zahřát na přibližně 300°C. Pro rychlejší zahřátí se dnes proto používají většinou vyhřívané senzory [2].

  Hodnoty z lambda sondy není možné číst přímo z CAN sběrnice, respektive možné to je, ale není to součástí standardu, ani žádné dostupné dokumentace. Přes ODBII rozhraní je však možné číst hodnoty už zpracované procesorem ukazující bohatost, respektive chudost směsi

v procentech (hodnoty označované jako „long time fuel trim“ a „short time fuel trim“, pro dlouhodobý, respektive krátkodobý ukazatel).

- Poloha plynového pedálu
  Kdysi byl plynový pedál přímo propojen se škrtící klapkou a mechanicky nastavoval její polohu. Dnes je v drtivé většině automobilů snímána poloha plynového pedálu elektronicky a signál posílán na zpracování řídící jednotce motoru [1].

- Senzor pozice škrtící klapky (Throttle Position Sensor – TPS)
  Senzor je většinou umístěn přímo na hřídeli klapky a může tak snímat její aktuální polohu. Jak je zmiňováno v předchozím odstavci o plynovém pedálu, kdysi byla klapka spojena přímo s plynovým pedálem, většinou pomocí bowdenu. Teď už jsou škrtící klapky řízeny téměř výhradně elektronicky. Obsahují krokový motorek, který pohybuje hřídelí a senzor poskytující zpětnou vazbu procesoru pro přesné nastavení polohy klapky. Její polohu řídí ECM, takže nemusí přesně opisovat polohu pedálu, ale závisí od algoritmu obsaženého v řídící jednotce. V současnosti se používají převážně bezdotykové senzory na principu Halova efektu nebo indukční senzory [1]. Poloha škrtící klapky je také přístupná přes OBDII rozhraní udávající procento otevření klapky (viz sekce 2.4).

- Senzor rychlosti vozidla (Vehicle Speed Sensor – VSS)
  Senzor rychlosti vozidla je většinou umístěn na výstupu převodovky nebo na zadní poloose při zadním náhonu. Snímá otáčky hřídele a na jejich základě generuje signál, který posílá na zpracování do řídící jednotky motoru. Snímačů existuje více druhů a principy snímání se mohou pro každý typ vozu lišit. ECM přijatý signál zpracuje a podle velikosti kola přepočte na rychlost vozidla [13]. Tento údaj je pak také přístupný přes OBDII (viz sekce 2.4).

- Otáčky motoru
  Pro měření otáček motoru se většinou používá senzor na klikové hřídeli. Ten potřebuje řídící jednotka motoru k tomu, aby dokázala správně načasovat jak vstřikování paliva, tak zapalování. A proto, že už známe přesnou polohu klikové hřídele a tedy také její otáčky, není potřeba používat další speciální snímač jenom pro otáčky motoru [19]. Většinou se používají snímače na principu Halova jevu, ale existují také varianty založené na optickém nebo indukčním principu [13] a opět se liší podle konkrétního modelu auta. Signál přijatý a zpracovaný ECM je pak přístupný přes OBDII (viz sekce 2.4).

- Absolutní tlak v sání motoru (Manifold Absolute Pressure – MAP)
  MAP senzor měří změny tlaku v sacím potrubí motoru způsobené změnou rychlosti a zatížení motoru. ECM vysílá 5V referenční signál do MAP senzoru. Jak se mění tlak v sacím potrubí, mění se také odpor MAP senzoru. Sledováním napětí na výstupu senzoru může řídící jednotka určit hodnotu tlaku v sání motoru. Za určitých podmínek je MAP senzor použit také pro měření barometrického tlaku, což umožňuje řídící jednotce přizpůsobit se nadmořské výšce [2][19]. Hodnota MAP senzoru je přístupná přes rozhraní OBDII v kPa (viz sekce 2.4).

- Senzor množství vzduchu (Mass Air Flow – MAF)
  Je to senzor pro měření množství protékajícího vzduchu sáním motoru. Jedná se o velice důležitý senzor. Spolu s údaji od lambda sondy jsou data z MAF senzoru použity řídící jednotkou motoru pro výpočet správného množství paliva, které má být do válce vstřiknuto.

Starší typ senzoru (také nazývaný jako VAF – Vane Air Flow) pracoval na mechanickém principu, kde v komoře se nacházela klapka připojena na potenciometr [19]. Proudící vzduch klapku otevíral a potenciometr měnil svůj odpor (a tedy také výstupní napětí) v závislosti na poloze klapky [10]. V moderních automobilech se již využívá mírně odlišný princip založen na odporovém drátě (hot wire). Základ tohoto systému tvoří termistor, platinový vyhřívaný drátek a řídící obvod. Termistor měří teplotu nasávaného vzduchu. Na drátek je přivedeno konstantní napětí, aby jej ohřálo. Platinový drátek je umístěn do proudu vzduchu ve vzduchovém tunelu a je ochlazován proudícím vzduchem. Jak klesá teplota drátku, klesá také jeho odpor, což umožňuje, aby drátkem protékalo více proudu a udrželo přednastavenou teplotu. Měří se pak velikost protékajícího proudu, která je transformována na frekvenci, nebo napětí a poslána ECM na zpracování. Hodnota zpracována ECM je také přístupná přes OBDII rozhraní v gramech za sekundu (viz sekce 2.4) [10][19].

2.3 Komunikace mezi elektronickými systémy v automobilech – sběrnice CAN BUS

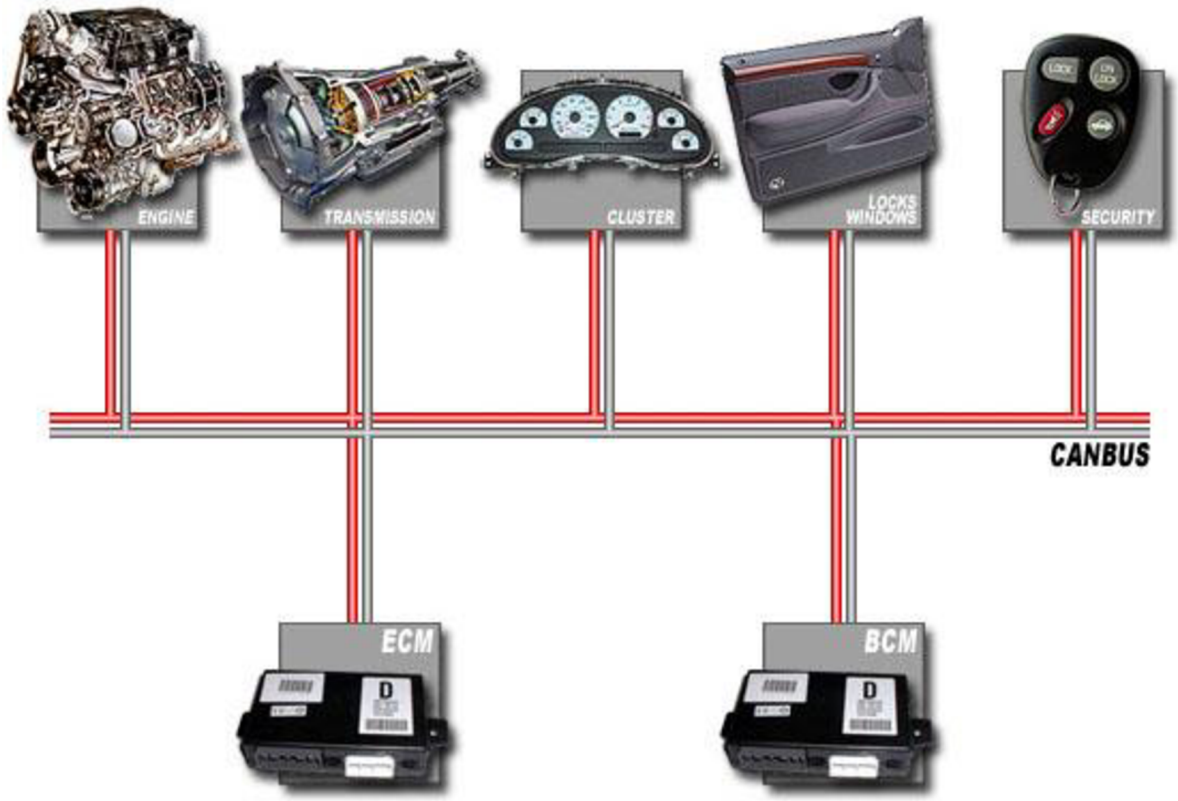
V posledních letech se zvyšujícími se požadavky na bezpečnost, komfort, jízdní vlastnosti a v neposlední řade snaha o co největší snižování emisí a spotřeby paliva, dochází v automobilovém průmyslu k masivní implementaci elektronických systémů v konstrukci automobilů. Motory jsou řízeny čistě elektronicky, málokterý ještě disponuje mechanickým plynovým pedálem, či karburátorem. Některý výrobci zašli dokonce tak daleko, že jejich auta nemají přímo mechanicky spojený ani brzdový pedál s brzdovým okruhem, jako u systému SBC od Mercedes-Benz (vyvinut společností Daimler ve spolupráci s firmou Bosch)[2]. Použití jedné centrální řídící jednotky už samozřejmě není možné, nebo to minimálně není nejefektivnější řešení. Jak postupem času rostl počet elektronických systémů ve vozidlech, začínalo být množství vodičů spojujících tyto systémy neúnosné, proto vznikla potřeba propojit je sofistikovanější sběrnicí, která by snížila potřebný počet vodičů a zároveň zvýšila spolehlivost celého systému. Pro tyto účely vyvinula firma Bosch sběrnici CAN (viz předchozí kapitoly), která přesně splňuje tyto požadavky. Za posledních třicet let se z ní stal standard nejen v automobilovém průmyslu [7]. Tato sběrnice se stala také součástí standardu OBDII, který má umožnit základní diagnostiku moderních automobilů jednotným způsobem a ulehčit tak dodržování emisních norem (viz sekce 2.4).

CAN BUS

Sběrnice CAN BUS (z anglického Controller Area Network) byla vyvinuta firmou Bosch v roce 1983 pro automobilový průmysl a poprvé použita v sériové výrobě v roku 1986. V případě prvního vozu bylo díky použití sběrnice CAN BUS ušetřeno dva kilometry kabelů [5]. Postupně našla uplatnění nejen v automobilovém průmyslu, ale také v průmyslové výrobě [6]. Byla vyvinuta jako systém s více hlavními uzly (multi-master), vysílající zprávy všem uzlům (message broadcast), který specifikuje maximální přenosovou rychlost na úrovni 1 Megabit za vteřinu (Mbps). Na rozdíl od tradičních sítí jako USB nebo Ethernet, CAN neposílá velké bloky dat z uzlu A do uzlu B (point-to-point) za dohledu centrálního uzlu (central bus master). V sítích CAN je mnoho krátkých zpráv, jako

[2] Zdroj: http://autoweek.com/article/car-news/mercedes-cancels-wire-brake-system-decision-blow-technologys-future

teplota chladicí kapaliny nebo otáčky motoru, vysílaných (broadcastovaných) do celé sítě, což poskytuje konzistenci dat ve všech uzlech sítě [7][6].

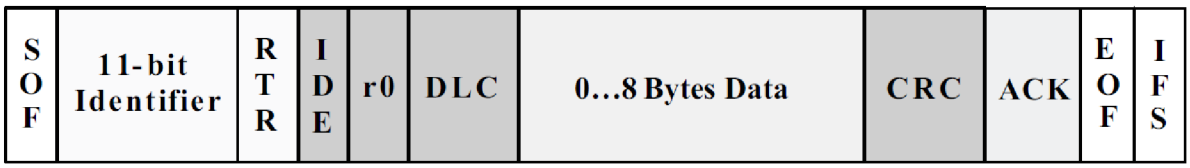


Obrázek 2.2: Ukázka propojení sběrnice CAN BUS v moderním automobilu [6].

CAN je sériová komunikační sběrnice definovaná ISO (International Standardization Organization) pro automobilový průmysl, vytvořená tak, aby nahradila množství spojů a kabelů za jednoduchou dvou vodičovou sběrnici. Komunikační protokol CAN je definován v ISO-11898: 2003 a popisuje, jak jsou informace vyměňované mezi zařízeními v síti. Specifikace požaduje vysokou odolnost proti rušení a schopnost detekce a opravy chyb. Tyto vlastnosti dopomohly velké popularitě CAN v různých oblastech průmyslu, včetně automatizace, medicíny nebo výroby [7].

Standardní a rozšířený CAN identifikátor

Komunikační protokol sběrnice CAN je CSMA/CD+AMP (Carrie-Sense, Mulltiple-Access with Collision Detection and Arbitration on Message Priority) [7]. CSMA vyjadřuje, že každý uzel na sběrnici musí čekat určitý definovaný časový okamžik před tím, než se pokusí vysílat zprávu na sběrnici. CD+AMP znamená, že kolize jsou řešeny pomocí bitové arbitráže (bit-wise arbitration), založené na předprogramované prioritě pro každou zprávu, obsažené v identifikátoru zprávy. ISO standard původně definoval zprávy s 11 bitovým identifikátorem poskytující rychlosti od 125kbps po 1Mbps. Později byl doplněn o rozšířený (extended) 29 bitový identifikátor poskytující až 537 milionů identifikátorů [7][5]. Podle některých zdrojů se používá také jiný princip přístupu na CAN sběrnici a to CSMA/CA (collision avoidance), tedy vyhýbání se kolizím místo jejich detekce. Vypadá to, že jako všechno v automobilovém průmyslu, tak také CAN nemá standardizovanou a stejnou podobu u všech výrobců automobilů [16].



Obrázek 2.3: Formát CAN zprávy se standardním (11bit) identifikátorem [7].

Na obrázku 2.3 je formát standardní CAN zprávy s 11 bitovým identifikátorem. Jednotlivé položky zprávy jsou vysvětleny v následujícím odstavci [7][5]:

- SOF (The single dominant start of frame) – je to bit označující začátek zprávy, používá se pro synchronizaci uzlů na sběrnici po tom, co byly uzly nečinné.
- 11 bit Identifier – 11 bitový identifikátor, určuje prioritu zprávy. Čím nižší je jeho binární hodnota, tím vyšší je priorita zprávy.
- RTR (Remote transmission request) – tento bit je nastaven, pokud uzel požaduje nějaká data od jiného uzlu. Zpráva je doručena všem, ale identifikátor specifikuje příjemce. Odpověď také obdrží všechny uzly, ale použijí jenom tací, které tato hodnota zajímá. To zabezpečuje konzistenci informací v systému.
- IDE (Identifier extension) – indikuje, zda je použit standardní CAN identifikátor bez rozšíření nebo rozšířený 29 bitový identifikátor.
- r0 (Reserved) – rezervovaný bit pro případná rozšíření v budoucnosti.
- DLC (The 4-bit data length code) – čtyř bitový indikátor počtu přenášených datových bitů.
- Data – datová část CAN zprávy, až 64 bitů může být přenášeno.
- CRC (Cyclic redundancy check) – je to 16 bitů (15 bitů plus jeden oddělovač) obsahujících kontrolní součet dat, pro detekci případných chyb při přenosu.
- ACK (Acknowledge) – každý uzel, který příjme zprávu v pořádku, přepíše tento bit v původní zprávě na 1, říkajíc tak, že byla zpráva přenesena bezchybně. Pokud přijímající uzel detekuje chybnou zprávu, nechá tento bit nezměněn, zprávu zahodí a odesílající uzel zprávu odešle znovu. Na základě tohoto principu, každý uzel potvrdí (ACK) integritu svých dat. ACK jsou dva bity, jeden potvrzovací a druhý slouží jako oddělovač.
- EOF (End-of-frame) – 7 bitů značících konec CAN zprávy.
- IFS (Interframe space) – je to 7 bitů, které slouží jako oddělovač zpráv. Poskytují čas, který potřebuje kontrolér, aby přesunul přijatá data bufferu.

Obrázek 2.4: Formát CAN zprávy s rozšířeným (29bit) identifikátorem [7].

Jak je patrné z obrázku 2.4, rozšířená CAN zpráva je shodná se standardní, má však navíc některé položky [7]:

- SRR (Substitute remote request) – je bit, který nahrazuje RTR bit a v rozšířeném formátu zprávy zastupuje jeho funkci.

- IDE (Identifier extension) – 0 na místě IDE indikuje, že následují další bity identifikátoru, přesněji dalších 18 rozšiřujících bitů.
- r1 – byl přidán jako další rezervovaný bit pro budoucí použití.

2.4 On Board Diagnostics

OBD je zkratka z anglického On-Board Diagnostics používána v automobilovém průmyslu, která odkazuje na schopnost samodiagnostiky a hlášení chyb vozidla. OBD rozhraní poskytuje přístup k datům různých senzorů a stavu subsystémů. Množství informací dostupných přes OBD se od jeho vzniku v osmdesátých letech výrazně změnilo. První verze jednoduše rozsvěcovaly kontrolku na palubní desce, ale neposkytovaly bližší informace o chybě, která nastala. OBD rozhraní v moderních automobilech se už řídí standardy a poskytuje data ze senzorů v reálném čase jako doplněk k standardizovaným diagnostickým kódům (DTC – diagnostic trouble codes), které dovolují rychle identifikovat a odhalit závadu na vozidle [8].

OBDII

Verze OBDI byl takovým prvním pokusem výrobců o implementaci diagnostiky přímo ve vozidle, nebyl příliš standardizovaný. Každý výrobce měl vlastní konektor, vlastní umístění konektoru, dokonce vlastní definici chybových kódů [8].

OBDII je výrazným vylepšením první verze co se týče standardizace, tak také možností diagnostiky. Standard přesně definuje diagnostický konektor, rozložení pinů konektoru a také dostupné komunikační protokoly a formát zprávy. Součástí konektoru je také napájení pro připojené diagnostické zařízení, odpadá tedy nutnost vlastního napájení diagnostického přístroje. OBDII standard definuje rozšiřitelnou sadu diagnostických kódů (DTC), takže pomocí jediného zařízení je možné diagnostikovat prakticky všechny moderní automobily [8]. Hlavním důvodem vzniku OBDII byly požadavky na emise a jejich dodržování, proto jsou požadovány jenom kódy vztažené k emisím. Přesto většina výrobců už další konektor ve vozidle neimplementuje a používá OBDII konektor také pro diagnostiku a programování všech systémů ve vozidle. Tato funkcionalita je však nad rámec standardu, proto si ji každý výrobce implementuje podle sebe a většinou není součásti žádné veřejně dostupné dokumentace [8]. Od roku 1996 je OBDII povinný pro všechny výrobce osobních automobilů v USA. V Evropské unii je EOBD (viz dále) povinný pro všechny vozidla se zážehovými (benzínovými) motory od roku 2001 a pro vznětové (dieselové) motory od roku 2003 [14].

Chybové kódy (DTC)

Standardní diagnostické kódy pozůstávají ze čtyř číslic následujících po písmeně, kde písmeno reprezentuje typ nebo skupinu chyby (např.: P0101):

- P – powertrain – reprezentuje chyby motoru, převodovky a emisního systému.
- B – body – pro chyby systému klimatizace, osvětlení, airbagů a dalších.
- C – chassis – brzdový systém, ABS systém, elektronický posilovač řízení, elektronické odpružení.
- U – reprezentuje chyby sítě – chyby CAN sítě a jednotlivých modulů [12][8].

První číslo reprezentuje:

- 0 – pro standardní kód
- 1 – pro specifický kód definovaný výrobcem automobilu

Druhé číslo:

- 1 – řízení emisí (emission management)
- 2 – obvod vstřikovačů (Injector circuit)
- 3 – zapalování (Ignition)
- 4 – doplňkové emise (Auxiliary emission)
- 5 – rychlost vozidla a ovládání volnoběhu (Vehicle speed & idle control)
- 6 – řídící jednotka a výstupní obvody (Computer & output circuit)
- 7 – převodovka (Transmission)

Příklad některých chybových kódů:

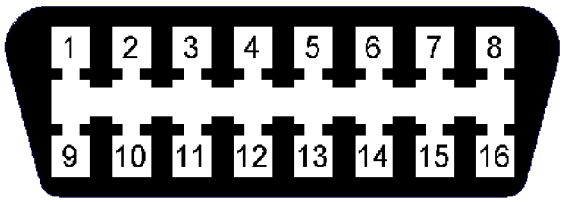
- P0300 – vynechání jiskry zapalování na více válcích
- P0301 – vynechání jiskry zapalování na prvním válci
- P0094 – zjištěn malý únik paliva
- P0128 – porucha termostatu chlazení [12]

Práci s chybovými kódy se věnuje množství jiných standardních aplikací dostupných na internetu, v případě zájmu je možné popis standardních kódů najít v literatuře [8] nebo na internetu[3].

OBDII konektor

Typ OBDII konektoru, jeho poloha v kokpitu vozu a zapojení jednotlivých pinů je definováno standardem podle ISO 15031-3:2004 [8]. Ne všechny piny jsou podle standardu obsazeny a výrobci si tedy mohou podle potřeby do konektoru přidat vlastní rozhraní pro komunikaci. Většina moderních vozidel má OBDII na protokolu CAN podle SAE J-2284 a ISO 15765-4, používají tedy piny 6 a 14 pro CAN high, respektive CAN low, jak je vidět z tabulky 2.1.

Kupříkladu společnost General Motors definuje na pinu číslo 1 rozhraní SW CAN (Single Wire CAN), tedy variantu CAN rozhraní vedenou pouze jedním signálovým vodičem (standardně používá CAN rozhraní pro komunikaci kroucenou dvoulinku) [8].



Obrázek 2.5: Číslování pinů konektoru podle ISO 15031-3:2004 (pohled na čelní stranu konektoru umístěnou v automobilu)[4].

[3] Web: http://www.troublecodes.net/obd2/
[4] Zdroj: http://www.obdtester.com/obd2_connector

Pin	Zapojení	Pin	Zapojení
1.	Definované výrobcem	9.	Definované výrobcem
2.	J1850 Bus +	10.	J1850 Bus -
3.	Definované výrobcem	11.	Definované výrobcem
4.	Zemnění karoserie V--	12.	Definované výrobcem
5.	Zemnění signálu V--	13.	Definované výrobcem
6.	CAN High (J-2284)	14.	CAN Low (J-2284)
7.	ISO 9141-2 K-Line	15	ISO 9141-2 K-Low
8.	Definované výrobcem	16.	Napájení V++ / Vcc

Tabulka 2.1: Tabulka zapojení pinů diagnostického konektoru v automobilu podle ISO 15031-3:2004 [8].

PIDs

Protokol OBDII pracuje na principu dotaz-odpověď (request-response), to znamená, že na jeden dotaz vrátí právě jednu odpověď. Výjimkou jsou odpovědi, které sestávají z více řádků, to pak přijde každý řádek jako samostatná odpověď a příjemce se musí postarat o jejich složení. Takovým příkladem je dotaz na VIN číslo (Vehicle Identification Number), které obsahuje 17-20 znaků. Odpověď tedy dorazí v pěti samostatných zprávách za sebou obsahujících pořadí potřebné pro složení odpovědi [11].

OBDII PIDs (z anglického parameter IDs, dále v textu bude používána originální zkratka PID/PIDs, protože je to tak zaužívané v oboru) jsou kódy, pomocí kterých se diagnostické nástroje dotazují na data od řídící jednotky. Dotaz na data pozůstává z tzv. módu, což je hexadecimální číslo v rozsahu 0x01 až 0x0A, definující dotazovací mód a čísla PID, které definuje požadovaný typ informace.

Ve standardu SAE J-1979 je definováno mnoho základních PIDs, ale výrobci mají možnost si definovat další kódy specifické pro dané vozidlo. Výrobci vozidel ale nejsou povinni implementovat všechny módy a ani v rámci módů nemusejí definovat všechny PIDs [11]. V tabulce 2.2 jsou uvedeny všechny módy obsažené ve standardu. PIDs je velké množství, proto v tabulce 2.3 je možné vidět výpis jenom těch základních. Kompletní seznam je možné nalézt v literatuře [8].

Mód (hex)	Popis
01	Ukázat aktuální data
02	Ukázat „freeze frame“ data
03	Ukázat DTC – diagnostické chybové kódy
04	Vymazat DTC
05	Výsledky testů, lambda sonda
06	Výsledky testů
07	Ukázat nevyřešené chybové kódy
08	Speciální kontrolní mód
09	Dotaz na informace o vozidlu
0A	Dotaz na trvalé chybové kódy

Tabulka 2.2: OBDII módy definované ve standardu podle SAE J-1979 [8].

Základním módem je mód 01, kterým se zobrazují data z automobilu v reálném čase. Mód 02 zobrazuje stejnou skupinu dat jako předchozí mód, ale data nejsou aktuální, nýbrž uložená z doby,

kdy se vyskytla poslední chyba v systému. Pomocí módu 03 je možné zobrazit uložené chybové kódy, které se vyskytly od posledního vynulování těchto kódů. Mód 04 jednoduše vymaže všechny aktuálně uložené chybové kódy [12]. To byl popis nejpoužívanějších módů.

PID (hex)	Datových bytů	Popis	Min. hodnota	Max. hodnota	Jednotka	Vzorec pro výpočet
04	1	Zatížení motoru	0	100	%	A*100/255
05	1	Teplota chladicí kapaliny	-40	215	°C	A-40
0C	2	Otáčky motoru	0	16383	rpm	((A*256)+B)/4
0D	1	Rychlost vozidla	0	255	Km/h	A
10	2	MAF (Air Flow Rate)	0	655,35	g/s	((A*256)+B)/100
11	1	Poloha škrtící klapky	0	100	%	A*100/255
5E	2	Rychlost toku paliva	0	3212.75	L/h	((A*256)+B)/20

Tabulka 2.3: Seznam základních PIDs [8].

Ukázka dotazu na data od automobilu

V následujících odstavcích je ukázka dotazu na rychlost vozidla. Datová část CAN zprávy pro dotaz je v tabulce 2.4 a v tabulce 2.5 pak přijatá odpověď. Požadujeme aktuální data, tedy mode 01 a PID pro rychlost je 0D (viz tabulky 2.2 a 2.3). Dostáváme odpověď, ve které je definován počet datových bytů, na jaký dotaz je to odpověď a požadovaná hodnota. Přijatá odpověď má hodnotu 0x55, což je dekadicky 85 km/h. Rychlost vozidla má definovanou hodnotu na jednom bytě, další tři datové byty zůstávají tedy volné.

PID dotaz na data od řídící jednotky je poslán vozidlu přes sběrnici CAN na adresu 0x7DF (hexadecimální formát čísla), která slouží jako všesměrová (broadcast) adresa. Odpověď pak přichází z některé z adres 0x7E8 až 0x7EF, což je rozsah osmi adres. Tento princip dovoluje, aby na dotaz reagovala konkrétní (jedna z osmi) řídicí jednotka, které se tento dotaz týká [8].
Za adresou následuje osm datových bajtů:

- Bajt 0 – počet doplňkových datových bitů
- Bajt 1 – mode
- Bajt 2 – PID
- Bajty 3-7 – nepoužité

Byte							
0	1	2	3	4	5	6	7
Počet doplňkových bajtů: 2	Mode: 0x01 = ukaž aktuální data	PID: 0x0D = rychlost vozidla	Nepoužité				

Tabulka 2.4: Příklad standardního OBDII dotazu na rychlost vozidla poslaného na všesměrovou adresu 0x7DF s osmi datovými bajty - Datová část CAN rámce [12].

Byte							
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Počet doplňkových bajtů: 3	Mode: 0x41 = stejné jako dotaz + 0x40	PID: 0x0D = rychlost vozidla	Požadovaná hodnota: bajt 0: 0x55	Požadovaná hodnota: Bajt 1-3 (volitelné)			Nepoužité Může být: 0x00,nebo 0x55

Tabulka 2.5: Příklad odpovědi na OBDII dotaz z tabulky 2.4 od ECU z adresy 0x7E8 - Datová část CAN rámce [12].

EOBD

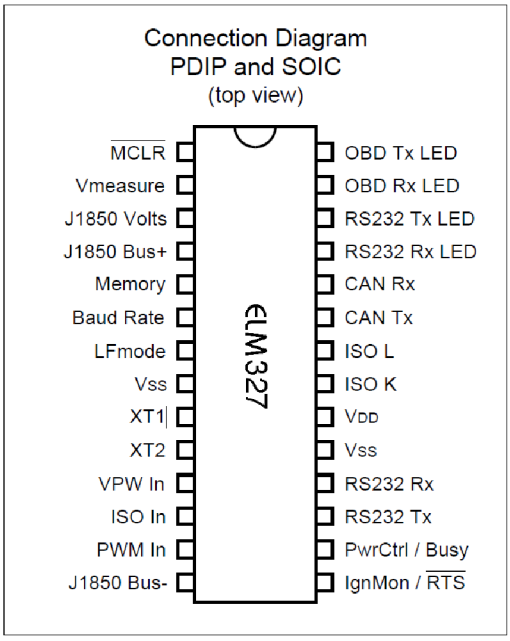
EOBD (zkratka pro European On Board Diagnostics) jsou pravidla vydané Evropskou unií, které jsou závazné pro všechny nově zaregistrované automobily ve všech členských státech Evropské unie. Pro benzínové motory platí od 1. ledna 2001 a pro dieselové od 1. ledna 2004 [14].

Technická implementace je naprosto shodná s normou OBDII, používá stejný SAE J-1962 konektor a stejný signální protokol [8]. Pouze definuje vlastní práh pro emise, pro jednotlivé emisní normy. V době psaní této práce byla v platnosti emisní norma Euro 6, zavedená od roku 2014 [15].

2.5 ELM327

Od téměř všech dnes vyráběných automobilů je zákonem vyžadováno, aby jejich vozidla disponovala rozhraním pro připojení diagnostického vybavení. Data přenášena těmito rozhraními sledují několik standardů, ale žádný z nich není přímo použitelný s počítačem nebo chytrým zařízením. ELM327 je navržen jako most mezi tímto rozhraním palubní diagnostiky (OBD) a standardním RS232 sériovým rozhraním. Kromě toho, že dokáže automaticky detekovat a interpretovat devět OBD protokolů, poskytuje také podporu pro vysokorychlostní komunikaci, režim nízké spotřeby a také podporu standardu J1939 pro kamiony a autobusy. Je také zcela přizpůsobitelný pro potřeby konkrétní aplikace [11].

Následující text popisuje, jak se používá ELM327 pro získávání informací z automobilu. Začíná popisem jak komunikovat s integrovaným obvodem za použití počítače, pak následuje vysvětlení použití „AT“ příkazů a nakonec několik příkladů jak získat informace od řídící jednotky vozidla. Pro další podrobnější informace chybějící v této části, je možné je najít v dokumentaci (datasheetu) k ELM327 nacházejícího se v odkazech na literaturu [11]. Základní použití ELM327 je relativně jednoduchou záležitostí, pokud nehodláme měnit nastavení a využívat všechny možnosti tohoto obvodu. Pak nám stačí jenom počítač s terminálovým programem (např. HyperTerminal nebo ZTerm) a základní znalost OBD příkazů. V této práci zajdeme samozřejmě dál než jen na pouhé zadávání základních příkazu do terminálu.



Obrázek 2.6: Pohled na mikrokontrolér ELM327 a rozložení pinů [11].

Komunikace s ELM327

ELM327 používá pro komunikaci s počítačem sériovou linku RS232. Většina moderních počítačů však už sériovou linkou nedisponuje, existuje, ale několik způsobů jak můžeme virtuální sériovou linku vytvořit. V této práci budeme používat dvě různé verze. Pro komunikaci s počítačem verzi s převodníkem USB na RS232 a pro komunikaci s chytrým telefonem a systémem Android verzi s převodníkem Bluetooth na RS232. Před použitím terminálového programu z počítače je nutné udělat několik nastavení. Nejdřív je nutné zvolit správný COM port a pak zkontrolovat nastavení rychlosti portu. Standardně je to buď 9600 Bd (baudů) (pokud je pin 6 = 0V při zapnutí), nebo 38400 Bd (za předpokladu, že parametr PP OC nebyl změněn) [11]. Po zvolení nesprávného portu nebude možné posílat ani přijímat žádná data. Při zvolení špatné rychlosti sice komunikace bude zdánlivě fungovat, ale přijaté data budou zkomolená a nečitelná jak pro terminál, tak pro ELM327. Dále je potřeba nastavit počet datových bitů na 8, žádné paritní bity, počet stop bitů na 1 a také nastavit korektní mód pro ukončování řádků. Všechny odpovědi od ELM327 jsou ukončeny jednoduchým znakem CR (Carriage Return), volitelně také LF (Line Feed), záleží na konkrétním nastavení. Výchozí nastavení sériového rozhraní ELM327 je možné vidět v tabulce 2.6.

Parametr	**Hodnota**
Rychlost (baudů)	9600
Datové bity	8
Parita	žádná
Počet stop bitů	1

Tabulka 2.6: Výchozí nastavení RS232 rozhraní mikrokontroléru ELM327 [11].

Správně připojené ELM327 rozbliká čtyři ledky, jednu za druhou v pořadí jako test a pošle zprávu:

```
ELM327 v1.5 >
```

Jednak tím identifikuje verzi mikrokontroléru, ale také přijetím čitelné zprávy potvrdí správné nastavení komunikace mezi počítačem a mikrokontrolérem (i když ještě stále neproběhla žádná komunikace mezi mikrokontrolérem a vozidlem, tedy stav tohoto spojení zůstává zatím neznámý). Znak „>“, který následuje za verzí mikrokontroléru, je tzv. „prompt character“ (dalo by se přeložit jako znak výzvy). Ten indikuje, že zařízení je v nečinném stavu a je připravené přijímat znaky na vstupu. V případě, že se neobjeví verze mikrokontroléru, ani znak „>“, je možné rozhraní restartovat příkazem *AT Z* následované enterem [11]. Pokud ani pak není komunikace úspěšná, jsou pravděpodobně špatně nastavené parametry spojení.

Znaky poslané z počítače mohou být určeny buď přímo pro mikrokontrolér, nebo jsou zpracovány a přeposlány řídící jednotce vozidla. ELM327 monitoruje obsah příchozích zpráv, takže dokáže rychle odhadnout, jestli je zpráva určena pro něj samotného nebo pro vozidlo. Příkazy, které jsou určeny pro ELM327 začínají znaky „AT“, kým OBD příkazy pro vozidlo mohou obsahovat jenom ASCII kódy hexadecimálních čísel (0-9 a A-F). Ať už se jedná o interní příkaz nebo hexadecimální řetězec pro OBD sběrnici, všechny zprávy pro ELM327 musí být ukončeny novým řádkem (CR – Carriage Return – 0x0D) před tím, než jsou zpracovány. Jedinou výjimkou je, pokud je poslán nekompletní řetězec bez ukončujícího znaku. V takovém případě je zpráva automaticky zrušena na základě interního časovače po přibližně dvaceti vteřinách a ELM327 vypíše otazník na výstup („?“), na znak toho, že zprávě nebylo porozuměno [11]. Pokaždé, když se ve zprávě vyskytne syntaktická chyba, je to signalizováno právě jednoduchým otazníkem. Tohle zahrnuje nekompletní zprávy, nesprávné AT příkazy nebo neplatné hexadecimální čísla, ale neindikuje to, jestli byla nebo nebyla zpráva pochopena vozidlem. Musíme mít na paměti, že ELM327 jen interpretuje OBD příkazy a hlídá syntaktickou správnost, ale nezkoumá příkaz samotný. Přijme data určená pro vozidlo, spojí je dohromady, přidá k nim všechny potřebné údaje (hlavičku, počet datových bajtů, kontrolní součet a další, viz kapitola 2.3) a pošle na OBD sběrnici. Zatímco zpracovává OBD příkazy, ELM327 aktivně čeká jak na RTS vstupu, tak na RS232 portu. Každý z nich způsobí přerušení mikrokontroléru a rychle vrátí kontrolu uživateli, ale s možností, že bude přerušena jakákoliv právě probíhající operace. Po vygenerování signálu k přerušení by měl software vždy počkat na znak „>” (0x3E) nebo na přechod výstupu BUSY do nízké úrovně před tím, než začne posílat další příkazy [11]. U ELM327 nezáleží na velikosti znaků (není case sensitive), tedy příkaz „ATZ“ je identický s příkazem „aTz“. Také ignoruje všechny bílé a řídicí znaky jako mezera nebo tabelátor. ELM327 také umožňuje zopakovat poslední provedený příkaz posláním jenom samotného CR (nový řádek) znaku. Disponuje pamětí jenom pro jeden příkaz, není tedy možné zopakovat více příkazů najednou [11].

AT příkazy

Některé vnitřní parametry ELM327 mohou být nastaveny tak, aby cíleně změnili chování mikroprocesoru, například změna nastavení protokolu, změna hlavičky CAN paketu nebo nastavení hodnoty časovače. Aby bylo možné tyto parametry změnit, je potřebné použít vnitřní „AT“ příkazy. Jedná se o stejný princip, jaký používaly kdysi počítačové modemy pro změnu vnitřní konfigurace. ELM327 používá shodný princip, vždy sleduje data poslaná z počítače a hledá zprávy začínající znakem „A“ následovaným znakem „T“. Pokud takovou zprávu nalezne, jsou další znaky interpretovány jako příkaz pro změnu vnitřní konfigurace a po přijetí znaku nového řádku, daný příkaz vykoná. Pokud příkaz mění pouze některý z vnitřních parametrů, mikrokontrolér jednoduše odpoví „OK“ na znak, že příkaz byl úspěšně proveden [11].

Příkaz	Popis příkazu
<CR>	Zopakovat poslední příkaz
BRD hh	Baud Rate Divisor hh
D	Nastavit výchozí nastavení (Defaults)
L0, L1	Nový řádek vypnout/zapnout (Linefeeds)
Z	Resetovat rozhraní
PP xx OFF	Zakázat programovatelný parametr (PP) xx
PP xx ON	Povolit programovatelný parametr xx
PP xx SV yy	Nastavit PP xx na hodnotu yy
BI	Bypass inicializační sekvenci
DP	Popíš protokol (describe protocol)
H0, H1	Výpis hlaviček vypnout/ zapnout
RV	Přečíst vstupní napětí
MA	Monitorovat veškerý provoz (monitor all)
MR hh	Monitorovat vše pro hh (monitor for receiver)
MT hh	Monitorovat vše od hh (monitor for transmiter)
PC	Zavřít současný protokol (protocol close)
SH xyz	Nastavit hlavičku na xyz (set header)
SP h	Nastavit protokol na h (set protocol)
SR hh	Nastavit adresu příjemce na hh (set receive address)
CAF1 CAF0	Zapnout vypnout auto formátování pro CAN zprávy
CFC1 CFC0	Zapnout vypnout FlowControl pro CAN zprávy
CM hhh	Nastavit masku pro CAN ID na hhh
CF hhh	Nastavit filtr pro CAN ID na hhh
CRA hhh	Nastavit adresu příjemce pro CAN na hhh

Tabulka 2.7: Základní vnitřní příkazy mikrokontroléru ELM327 [11].

Popis základních příkazů pro mikrokontrolér ELM327

V tabulce 2.7 je možné najít vypsané některé základní vnitřní příkazy pro ELM327. Ty důležitější a potřebné pro účely této práce jsou pak podrobněji popsané v následujícím textu.

Příkaz BRD hh

Tento příkaz se používá pro změnu rychlosti RS232 portu (změnu dělitele rychlosti) na hexadecimální hodnotu hh v době, kdy je mikrokontrolér připojen k počítači. Protože některé obvody nejsou schopny pracovat při vyšších rychlostech, příkaz BRD pošle a příjme sekvenci několika bytů. Jakákoliv chyba v přenosu způsobí návrat k předchozí funkční rychlosti spojení. To umožňuje otestovat několik rychlostí a vybrat tu nejspolehlivější z nich. Pokud je test rychlosti úspěšný, je nová rychlost nastavena na hodnotu 4000 vydělenou číslem hh (v kbps) [11].

Příkaz H0 a H1 [Headers off or on]

Tento příkaz nastavuje, zda mají být zobrazovány další (hlavičkové) bajty obsažené v informaci od vozidla. Tyto nejsou za normálních okolností zobrazovány, ale v některých případech může být potřebné vidět celý paket (hlavně v případě, kdy dostáváme více odpovědí a chceme znát od kterého

modulu je která odpověď). Zapnutí zobrazování hlavičky pomoci H1 ve skutečnosti zobrazí více než jen hlavičkové bajty, je možné vidět kompletní přenesenou zprávu, včetně kontrolních a PCI bajtů. Verze mikrokontroléru 1.4b neumí zobrazit bajty kontrolního součtu [11].

Příkaz MA [Monitor All messages]

Příkaz MA přepíná ELM327 do monitorovacího módu, ve kterém kontinuálně monitoruje a zobrazuje veškerý provoz zachycený na OBD nebo CAN sběrnici. Jedná se o tiché monitorování, kdy mikrokontrolér nevysílá na sběrnici žádné data (response, acknowledges, wakeup, keep-alive messages). Monitorování pokračuje, dokud není zachycen jakýkoliv znak na RS232 vstupu nebo RTS pinu. Pro zastavení monitorování stačí tedy poslat mikorkontroléru jakýkoliv znak a počkat na znak „>“ jako odpověď nebo na přechod BUSY pinu do nízké úrovně. Je nezbytné počkat na „prompt character“, protože doba odezvy se liší podle toho, jakou operaci právě mikrokontrolér prováděl. Kupříkladu pokud právě vypisoval řádek, nejdřív dokončí výpis, pak vytiskne „STOPPED“ a až poté se vrátí do stavu, ve kterém přijímá příkazy a pošle „>“, na znak toho, že je. Pokud jenom jednoduše čekal na příkazy, bude čas odezvy výrazně kratší. Je potřeba poznamenat, že jakýkoliv znak, který zastaví monitorování, bude zahozen a nebude mít vliv na následné příkazy. Pokud je monitorování použité spolu s CAN protokoly a jsou nastaveny filtr a/nebo maska (pomocí CF, CM, CRA příkazů), pak MA příkaz bude ovlivněn těmito nastaveními. Pokud je například adresa příjemce předem nastavena pomocí CRA 4B0, pak příkaz AT MA uvidí pouze zprávy s ID 4B0. To nemusí být vždy záměrem, proto je potřeba si dát pozor na tyto nastavení, případně tyto filtry nejdříve vynulovat (pomocí příkazů AT AR, nebo AT CRA) [11].

Příkaz SP h [Set Protocol to h]

Příkaz SP se používá pro nastavení komunikačního protokolu pro komunikaci mezi ELM327 a automobilem, specifikovaným číslem protokolu h. Nastavený protokol je také uložen, bez ohledu na nastavení AT M0/M1. Je možné vybrat jeden z dvanácti protokolů, z kterých poslední tři jsou uživatelsky nastavitelné. Seznam protokolů, které je možné nastavit je uveden v tabulce 2.8.

První protokol (0) je nejjednodušší způsob jak oznámit ELM327, že protokol vozidla není známý a že by měl provést hledání. Mikrokontrolér zkusí všechny protokoly, pokud je to potřebné, aby našel jeden, který se podaří úspěšně inicializovat. Pokud je nalezen platný protokol a povolena funkce paměti (AT M1), je protokol uložen do paměti a nastaven jako výchozí. Pokud je uložen touto cestou, bude automatické hledání stále aktivní a pokud se příště nepovede mikrokontroléru připojit pomocí uloženého protokolu, znovu prohledá všechny možnosti. Pokud bude zvolen protokol jiný než 0 (např. AT SP 3), stane se výchozím protokolem a bude jediný, který ELM327 použije. V případě, že se nepovede inicializovat spojení pomocí tohoto protokolu, povede to na odpověď „BUS INIT: …ERROR“ a žádný další protokol nebude testován [11].

Protokol	Popis
0	Automatický
1	SAE J1850 PWM (41.6 kBd – kilo baudů)
2	SAE J1850 PWM (10.4 kBd)
3	ISO 9141-2 (5 baud init)
4	ISO 14230-4 KWP (5 baud init)
5	ISO 14230-4 KWP (fast init)
6	ISO 15765-4 CAN (11 bit ID, 500 kBd)
7	ISO 15765-4 CAN (29 bit ID, 500 kBd)
8	ISO 15765-4 CAN (11 bit ID, 250 kBd)
9	ISO 15765-4 CAN (29 bit ID, 250 kBd)
A	SAE J1939 CAN (29 bit ID, 250* kBd)
B	User1 CAN (11* bit ID, 125* kBd)
C	User2 CAN (11* bit ID, 50* kBd)

Tabulka 2.8: Protokoly podporované v mikrokontroléru ELM327, hodnoty označené * jsou výchozí a uživatelsky nastavitelné [11].

PP hh SV yy [Prog. Param. hh: Set the Value to yy]

ELM327 obsahuje několik paměťových míst, které uchovávají svoje data i po vypnutí napájení. Při každém startu mikrokontroléru jsou tyto data načtena a použita pro nastavení výchozích hodnot. Jedná se o nastavení jako například, jestli se mají zobrazovat také hlavičky zprávy nebo jak často se mají posílat „wakeup“ zprávy. Tyto nastavení a parametry mohou být změněny kdykoliv pomocí jednoduchých příkazů. Příkazy pro programovatelné parametry jsou klasické AT příkazy s jednou výjimkou, že každý vyžaduje dva kroky na dokončení. Tento extra krok poskytuje jakousi formu ochrany proti nechtěné změně parametrů. Například pro změnu parametru PP 01 na hodnotu 00, slouží jednoduchý příkaz [11]:

```
>AT PP 01 SV 00
OK
```

Tohle změní hodnotu asociovanou s PP 01, ale nepovolí ji. To je potřebné udělat pomocí příkazu:

```
>AT PP 01 ON
OK
```

Všechny programovatelné parametry, které je možné použít s mikrokontrolérem ELM327 je možné nalézt v literatuře [11].

SH xx yy zz [Set the Header to xx yy zz]

Tento příkaz dovoluje kontrolovat, jaké hodnoty jsou posílány jako hlavičkové bajty zprávy (nebo také identifikátor u CAN zpráv). Normálně jsou tyto přidělovány mikrokontrolérem automaticky a není potřebné je měnit, ale jsou situace, kdy může být žádoucí je upravit, hlavně pokud chceme experimentovat s fyzickým adresováním ne OBD paketů na síti. Není to bezpodmínečně nutné, ale je lepší nastavit hlavičkové bajty až poté, co je protokol již aktivní. Tímto zprávy, které udržují spojení aktivní (jako wakeup zprávy a jiné) zůstanou na výchozích hodnotách. Hlavičkové bajty jsou

definovány jako hexadecimální čísla, kde xx bude použit jako první bajt nebo priorita, yy jako druhý bajt nebo příjemce a zz jako třetí bajt nebo odesílatel. Toto nastavení zůstává platné, dokud nejsou obnoveny výchozí hodnoty příkazy D, WS, nebo Z. Pokud jsou hlavičkové bajty nastaveny před tím, než je inicializované spojení s vozidlem a výběr protokolu není nastaven jako plně automatický (tzn. jiný než 0), budou tyto hodnoty použity pro inicializaci spojení. Pokud toto nepovede na správnou odpověď od vozidla a je nastaven automatický výběr protokolu, bude ELM327 pokračovat v hledání protokolu s použitím výchozích hlavičkových bajtů. Jakmile je nalezen odpovídající protokol, jsou hlavičkové bajty nastaveny zpátky na to, co bylo nastaveno pomocí příkazu AT SH. Tento příkaz se používá pro přidělení hlavičkových bajtů ať už pro J1850, ISO 9141, ISO 14230, nebo CAN systémy. CAN systémy použijí tyto tři bajty pro naplnění bitů 0 až 23 části ID (pro 29 bitové ID) nebo použijí jenom nejnižších jedenáct bitů (nejvíc vpravo) pro 11 bitové CAN ID a zbytek bitů bude ignorován. Zbylých pět bitů potřebných pro 29 bitový CAN systém je nastaveno pomocí příkazu AT CP [11].

SH xyz [Set the Header to 00 0x yz]

Za normálních okolností při zadávání 11 bitového CAN ID (hlavičky) je potřebné přidávat nuly na začátek (např. AT SH 00 07 DF), tento příkaz to však zjednodušuje. Příkaz AT SH xyz přijme tři čísla jako argument, vezme nejpravějších jedenáct bitů, přidá počáteční nuly a výsledek uloží jako hlavičku. Kupříkladu, AT SH 7DF je platný příkaz a je velmi užitečný při práci s jedenáct bitovými CAN systémy. Ve skutečnosti způsobí uložení hlavičky jako 00 07 DF [11].

3 Popis systému Android a základních stavebních kamenů Android aplikací

Android je operační systém, který byl vyvinut primárně pro mobilní zařízení jako mobily, tablety, ale v poslední době také chytré televize, auta nebo náramkové hodinky. Na rozdíl od iOS firmy Apple, svého největšího konkurenta, je Android šířen volně pod licencí open-source (software s otevřeným zdrojovým kódem). Android pohání stovky milionů mobilních zařízení ve více jak 190 zemích světa. Je nejrychleji rostoucí platformou s více jak milionem nových instalací každý den[5]. Jeho jediným přímím konkurentem je zmiňovaný iOS, avšak momentálně má Android na trhu jednoznačné prvenství, minimálně co se do počtu zařízení týče [18].

V následujících odstavcích budou popsány základní elementy, z kterých se skládají Android aplikace a také popsán systém Android sám o sobě.

Vývoj operačního systému Android

Prvotním záměrem společnosti bylo vyvinout pokročilý operační systém pro digitální kamery. Když si však uvědomili, že tento trh je příliš malý, přeorientovali se na vývoj operačního systému pro chytré telefony. V té době se jednalo o začínající odvětví, kde se chystali být přímými konkurenty zaběhnutých systémů Symbian a Windows Mobile[6].

Přibližně rok od oznámení o vývoji systému Android přichází Google s první verzí, která byla nasazena v mobilním telefonu výrobce HTC a spolu s ním byla uvolněna první verze SDK 1.0 (Software Development Kit - soubor nástrojů pro vývoj softwaru, který umožňuje vytváření aplikací pro určitou softwarovou platformu, hardwarovou platformu aj.). Od roku 2008 dostává Android pravidelné aktualizace, které přidávají nové vlastnosti a opravují chyby z předchozích verzí. Každá nová hlavní verze je pojmenována v abecedním pořadí podle názvů různých zákusků a sladkostí, např. Donut, Ice Cream Sandwich nebo KitKat. Během svého života si Android vybudoval silnou základnu a z konkurenta na poli operačních systémů pro chytré telefony se stala světová jednička s více jak miliardou běžících zařízení [18].

Rozšířenost verzí systému Android a k nim přislouchajících verzí API

V době psaní této práce byl nejnovějším systémem Android verze 5.1.1 Lollipop z dubna 2015 a k tomu přislouchající verze API level 22. Na nejvyšší verzi zatím běží samozřejmě minimum zařízení. Tabulka 3.1 ukazuje jaké je rozšíření jednotlivých verzí systému Android (data jsou měřena pomocí aplikace Google Play Store, která podporuje Android 2.2 a novější, proto zařízení běžící na starší verzi nejsou v tabulce zahrnuté. Nicméně v srpnu 2013 používalo starší verzi Android 2.2 jenom kolem jednoho procenta zařízení)[7].

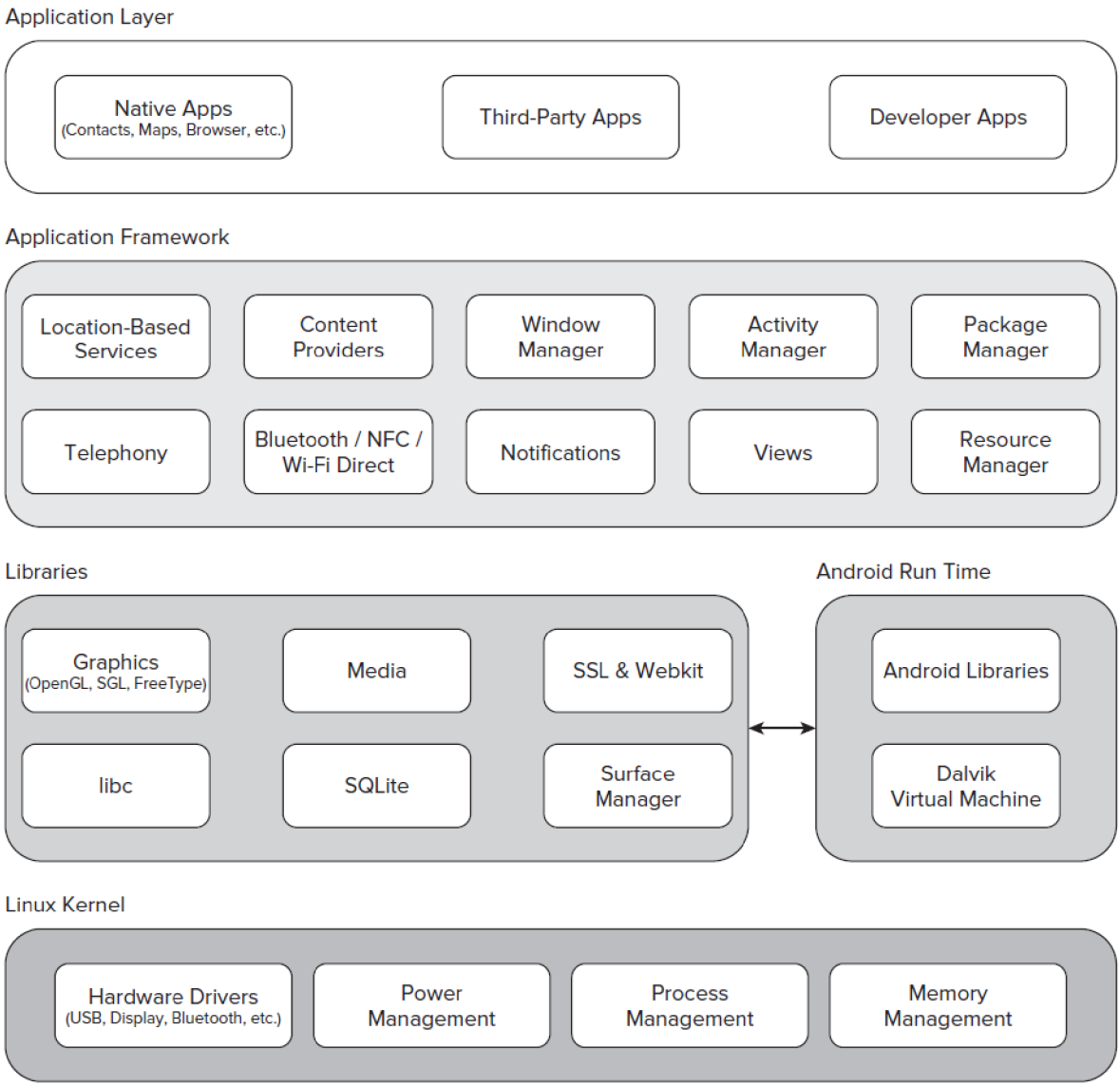
[5] Zdroj: http://developer.android.com/about/android.html
[6] Zdroj: http://www.theverge.com/2013/4/16/4230468/android-originally-designed-for-cameras-before-smartphones
[7] Zdroj: https://developer.android.com/about/dashboards/index.html?utm_source=suzunone

Verze	Kódové jméno	API verze	Použití
2.2	Froyo	8	0,3 %
2.3.3 – 2.3.7	Gingerbread	10	5,6 %
4.0.3 – 4.0.4	Ice Cream Sandwich	15	5,1 %
4.1.x	Jelly Bean	16	14,7 %
4.2.x		17	17,5 %
4.3		18	5,2 %
4.4	KitKat	19	39,2 %
5.0	Lollipop	21	11,6 %
5.1		22	0,8 %

Tabulka 3.1: Rozšíření jednotlivých verzí systému Android a k nim přislouchajících verzí API (k 1. červnu 2015)[7].

3.1 Architektura systému Android

Android aplikace jsou psané klasicky v Javě, ale spouštěné ve virtuálním prostředí nazývaném Dalvik, místo tradičního Java VM (Java Virtual Machine). Každá Android aplikace běží v samostatném procesu ve své vlastní instanci Dalvik-u. Aplikace přenechává plnou zodpovědnost za správu paměti a procesů na Android runtime, který zastavuje a ukončuje procesy podle potřeby pro efektivní řízení zdrojů. Dalvik a Android runtime běží nad linuxovým jádrem, které zabezpečuje interakci s hardwarem na nízké úrovni (low-level) včetně ovladačů a správy paměti, zatímco sada API rozhraní poskytuje přístup ke všem základním službám, funkcím a hardwaru [21].



Obrázek 3.1: Architektura systému Android [21].

Jednoduše řečeno, architektura systému Android sestává z linuxového jádra a kolekce C/C++ knihoven přístupných přes aplikační Framework, který poskytuje služby a management pro runtime aplikace. Jak je patrné z obrázku 3.1, architektura systému Android sestává z pěti částí [21]:

- Linuxové jádro (Linux kernel) – na nejnižší úrovni se nachází vhodně upravené linuxové jádro, které obsahuje ovladače pro správu hardwaru a zabezpečuje správu paměti a procesů, bezpečnost, síťové služby a správu napájení. Poskytuje také abstraktní vrstvu mezi hardwarem a zbytkem architektury. Do verze androidu 3.2 se používalo linuxové jádro 2.6, pro Android 4.0 a novější je to jádro verze 3.0 [25].
- Knihovny (Libraries) – na vyšší úrovni se nachází knihovny vytvořené v programovacím jazyce C/C++, které slouží k obsluze funkcí na nižší úrovni linuxového jádra. Patří sem například libc, SSL knihovna, ale také multimediální knihovny pro přehrávání audia a videa, či grafické knihovny jako SGL, OpenGL pro 2D a 3D grafiku, tak také SQLite pro podporu databází [21].

- Android runtime – runtime je to, co dělá Android spíše Androidem než mobilní implementací Linuxu. Obsahuje základní knihovny a virtuální stroj Dalvik VM (Dalvik Virtual Machine). I když většina aplikací pro Android je napsaných v Javě, Dalvik není Java VM. Na rozdíl od JVM neobsahuje knihovny AWT, Swing a další a je plně přizpůsoben potřebám systému Android. Dalvik VM je optimalizován tak, aby na jednom zařízení mohlo efektivně běžet několik jeho instancí [21].
- Aplikační Framework (Application Framework) – o úroveň výš se nachází aplikační Framework, který poskytuje třídy potřebné pro tvorbu Android aplikací. Pro vývojáře se jedná pravděpodobně o nejdůležitější vrstvu, obsahuje totiž služby nezbytné při vývoji aplikací [23]. Jednotlivým částem tohoto frameworku bude věnovaná následující kapitola.
- Aplikační vrstva (Application layer) – sestává jak z nativních aplikací, tak aplikací třetích stran. Běží spolu s Android runtimem a poskytuje svoje třídy a služby aplikacím, přístupné pomocí aplikačního Frameworku [21].

Jednotlivé komponenty tvořící Android aplikace

Architektura systému Android je navržena tak, aby podporovala znovu použití komponent, což umožňuje tvůrcům aplikací, aby publikovali a sdíleli svoje Aktivity, Služby a data s ostatními aplikacemi s přístupem řízeným bezpečnostními omezeními definovanými programátorem. Ten samý mechanizmus, který dovoluje nahradit správce kontaktů nebo telefonní dialer, také umožňuje vystavit části aplikace tak, aby mohli jiný vývojáři na nich dál stavět vytvořením nového uživatelského rozhraní nebo rozšířením původní funkcionality[21][24].

Tyto aplikační služby jsou základní stavební kameny každé aplikace pro systém Android [21] (názvy následujících služeb budou pro srozumitelnost uváděny v originálním anglickém tvaru tak, jak se vyskytují v literatuře):

- Activity Manager a Fragment Manager – kontroluje životní cyklus Aktivit a Fragmentů (Activities & Fragments) a zahrnuje také správu zásobníku Aktivit (Activity stack).
- Views – používá se pro konstrukci uživatelského rozhraní pro Aktivity a Fragmenty.
- Notification Manager – poskytuje konzistentní a nedotěrný mechanismus pro upozornění uživatelů.
- Content Providers – dovoluje aplikacím sdílet data.
- Resource Manager – umožňuje externalizovat nekódové zdroje jako textové řetězce a grafiku.
- Intents – poskytuje mechanizmus pro přenos dat mezi aplikacemi a jejich komponenty.

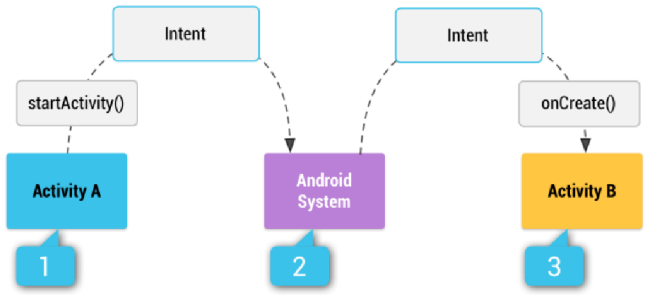
Activities

Aktivita (Activity – dále v textu bude používán výraz Aktivita) reprezentuje jednu obrazovku aplikace s uživatelským rozhraním. Jednotlivé prvky uživatelského rozhraní (GUI) jsou členěny do tzv. View (pohledů), pomocí kterých je možné vyvolat určité akce [23]. Většina běžných aplikací bude pravděpodobně obsahovat více takových Aktivit, které budou dohromady tvořit celek aplikace, ale samy o sobě budou nezávislé a bude možné kteroukoliv z nich spustit také z jiné aplikace (pokud to původní aplikace dovolí). Tím je možné vytvářet v systému Android aplikace, které mezi sebou spolupracují, případně využívají služeb jiných aplikací. Mezi akce, které může Aktivita vykonávat tedy patří také spuštění jiné Aktivity, což většinou znamená přechod na jinou obrazovku. Přechody mezi jednotlivými obrazovkami jsou řízeny pomocí tzv. Intents, které budou popsány dále v textu. O řízení Aktivit (spouštění a ukončování) se stará tzv. Intent Manager, nacházející se ve vrstvě

aplikačního frameworku. Aktivita je implementována jako podtřída třídy android.app.Activity (v jazyce Java použitím klíčového slova extends) [24][21].

Intents

Tři ze čtyř typů komponent – Activity, Services a Broadcast receivers - je aktivováno pomocí asynchronní zprávy nazývané Intent. Jedna Aktivita tedy nemůže jen tak sama spustit jinou Aktivitu nebo službu, toto je právě ponecháno v režii Intentu.



Obrázek 3.2: Ukázka doručení implicitního Intent přes systém[8].

Intent spojuje individuální komponenty za běhu programu, či už se jedná o komponenty stejné aplikace nebo jiné. Na Intent se dá nahlížet jako na zprávy, které požadují po komponentě vykonat nějakou činnost. Intent může být explicitní nebo implicitní. Na obrázku 3.2 je vidět postup jak je implicitní Intent doručen přes systém pro spuštění jiné Aktivity. Pro Aktivity a služby definuje Intent akce, které se mají vykonat (například zobrazit nebo poslat), může také specifikovat URL (Uniform Resource Locator – jednotná adresa zdroje) na data, nad kterými se má vykonat akce. Kupříkladu Intent může předat Aktivitě zprávu, že má být zobrazen obrázek nebo otevřena webová stránka. V některých případech je možné spustit Aktivitu za účelem vrácení nějakého výsledku. Tento výsledek je pak vrácen jako Intent (například uživatel vybere ze seznamu Bluetooth zařízení, ke kterému se chce připojit, pak Intent vrácený volající Aktivitě obsahuje adresu vybraného zařízení) [24].

Services

Služba (Service) je komponenta, která běží na pozadí a většinou vykonává nějakou dlouhotrvající operaci nebo úlohu pro vzdálený proces. Služba většinou neposkytuje žádné uživatelské rozhraní, proto její Aktivita nemusí být přímo viditelná. Služba v pozadí může například přehrávat hudbu, zatímco je uživatel aktivní na jiné aplikaci nebo stahovat data ze sítě bez toho, aby blokovala interakci uživatele s Aktivitou. Jiná komponenta jako například Aktivita může spustit službu a nechat ji běžet na pozadí nebo naopak se spojit s jinou běžící Službou za účelem vzájemné interakce. Služba je implementována jako podtřída třídy andoid.app.Service [24].

Content providers

Aplikace může ukládat data do souborového systému, SQLite databáze, na web nebo do jakéhokoliv permanentního datového úložiště, do kterého má přístup. Content provider se stará o sdílená data

[8] Zdroj: http://developer.android.com/guide/components/intents-filters.html

aplikace. Pomocí něj mohou ostatní aplikace přistupovat k těmto datům nebo dokonce je modifikovat, pokud to Content provider dovolí. Například systém Android poskytuje Content provider, který spravuje informace o kontaktech uživatele. A tedy jakákoliv aplikace s příslušnými oprávněními se může dotázat Content providera (napr. ContactsContract.Data) o povolení číst nebo modifikovat data o konkrétní osobě. Content provider poskytuje čtyři základní operace nad uloženými daty, a to insert, query, update, delete, případně další rozšíření nad těmito základními metodami. Content provider je implementovaný jako podtřída třídy android.content.ContentProvider a musí implementovat standardní sadu APIs, které dovolují ostatním aplikacím vykonávat transakce nad daty [23][24].

Broadcast receivers

Broadcast receiver je komponenta, která reaguje na vysílání celo-systémových oznámení. Většina jich pochází od systému, například oznámení, že byla vypnuta obrazovka zařízení, že dochází baterie, nebo že byl pořízen snímek. Aplikace mohou také iniciovat vysílaní vlastních oznámení, pokud chtějí upozornit ostatní aplikace, že byla stažena nějaká data a že jsou pro ně připravena. I když Broadcast receivers nemají žádné uživatelské rozhraní, mohou vytvořit oznámení v notifikační liště pro upozornění uživatele, když taková událost nastane. Ve většině případů je Broadcast receiver ale jenom bránou pro ostatní komponenty, kterým zprostředkovává informace a je určen dělat jenom minimum práce. Například může iniciovat Službu, která vykoná nějakou práci jako reakci na přijetí zprávy. Broadcast receiver je implementován jako podtřída třídy android.content.BroadcastReceiver a každé vysílání je přijato jako objekt typu Intent [24].

Resources

Bývá dobrým zvykem, aby nekódové části aplikace jako obrázky a textové řetězce nebyly součásti zdrojového kódu, ale drženy mimo něj. Android podporuje externalizaci prostředků (Resources), od jednoduchých hodnot jako jsou řetězce a barvy, až po složitější jakými jsou obrázky (Drowables), animace, témata a menu. Asi nejsilnějším externalizačním prostředkem jakým systém Android disponuje, jsou Layouts (budou popsány dále v textu). Držení těchto prostředků v externích strukturách zjednodušuje jejich údržbu a aktualizaci. Díky tomu je možné jednoduše vytvářet alternativy pro jazykovou lokalizaci textu a podporu různého hardwaru, zejména různé velikosti a rozlišení obrazovky. Při startu aplikace Android automaticky vybere příslušné prostředky pro danou aplikaci. Všechny tyto prostředky se nacházejí v podadresáři `res` a jsou rozděleny do následující adresářové struktury [21][23]:

- `drawable` – Drawable je obecný koncept pro grafické objekty, které se mohou v aplikaci vykreslovat na obrazovku. Zahrnují bitmapy a tzv. NinePatches (jsou to PNG soubory, které je možné „natahovat“), ale také složené Drawables jako LevelListDrawables a StaleListDrawables, které jsou definovány pomocí XML. Všechny Drawables se naházejí v `res/drawable` a každý z nich by měl být uložen v adresáři odpovídajícím jeho velikosti a rozlišení. Standardně obsahuje `res/drawable` tyto podadresáře [21]:
  - `drawable-mdpi`
  - `drawable-hdpi`
  - `drawable-xhdpi`
  - `drawable-xxhdpi.`

Jednotlivé soubory by měly být optimalizovány pro správné zobrazení na daném rozlišení displeje [23].

- `layout` – Layout (dá se přeložit jako rozložení grafického uživatelského rozhraní) soubory dovolují oddělit prezentační vrstvu od aplikační logiky tím, že rozvržení uživatelského rozhraní je definováno v XML souboru místo toho, aby bylo součástí zdrojového kódu. Pomocí Layout je možné definovat jakoukoliv komponentu uživatelského rozhraní, včetně Aktivit, Fragmentů a Widgetů. Poté co je Layout definován v XML souboru musí být „nafouknut" (inflated) do uživatelského rozhraní. Uvnitř Aktivity se tohle děje pomocí `setContentView` (obvykle uvnitř metody `onCreate`). Použití Layout pro konstrukci jednotlivých obrazovek pomocí XML je nejlepším způsobem vytváření uživatelského rozhraní v systému Android. Oddělení rozvržení od kódu dovoluje vytvářet optimalizované Layouty pro různé hardwarové konfigurace jako jsou různé velikosti displejů, orientace, přítomnost klávesnice či dotykové obrazovky. Každý Layout je definován v samostatném souboru, který obsahuje pouze jedno rozložení v adresáři `res/layout`. Jméno souboru se stává zároveň i identifikátorem daného Layoutu [21].

- `menu` – Stejně jako při vytváření Layoutů z předchozího odstavce, je i při vytváření menu nabídek aplikace, vhodné je definovat v externím XML souboru raději než přímo v kódu. Můžeme použít Menu prostředek pro definování jak menu pro Aktivitu, tak kontextového menu v rámci aplikace. Definování Menu v XML souboru poskytuje stejné možnosti jako jeho definování uvnitř kódu. Menu je pak do aplikace „nafouknuto" přes metodu `inflate` služby `MenuInflator`, obvykle v metodě `onCreateOptionsMenu`. Definice každého Menu je uložena v samostatném souboru v adresáři `res/menu`, přičemž se jméno souboru stává zároveň identifikátorem daného menu prostředku [21].

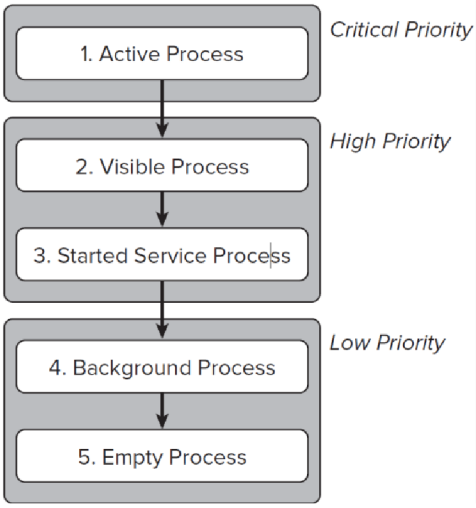
- `values` – Adresář `values` obsahuje hodnoty, které jsou používány za běhu aplikace. Opět jsou definovány pomocí XML a mohou nést hodnoty, které definují rozměry komponent, styly nebo textové řetězce. Například soubor strings.xml se stará o poslední zmiňovanou možnost, obsahuje dvojice hodnot: identifikátor a hodnotu – tedy řetězec, což umožňuje, aby všechny textové řetězce použité v uživatelském rozhraní byly definovány přehledně v XML souboru. Takový přístup pak výrazně zjednodušuje vytváření jazykových lokalizací. Dokonce i člověk bez programátorských znalostí může vytvořit překlad programu do jiného jazyka jenom úpravou XML souboru [21].

- `mipmap` – Obsahuje Drawables soubory se spouštěcími ikonami aplikace pro různé velikosti displeje v adresářích[9]:
  - `mipmap-mdpi`
  - `mipmap-hdpi`
  - `mipmap-xhdpi`
  - `mipmap-xxhdpi.`

[9] Zdroj: http://developer.android.com/guide/topics/resources/providing-resources.html

3.2 Životní cyklus aplikací v systému Android

Na rozdíl od tradičních operačních systémů, aplikace běžící na systému Android mají velmi limitovanou kontrolu nad vlastním životním cyklem. Jejich jednotlivé komponenty musí naslouchat změnám stavu aplikace, reagovat na ně a být připraveny, že mohou být kdykoliv ukončeny. Standardně je každá Android aplikace spuštěna ve svém vlastním procesu, z kterých každý běží na své vlastní instanci Dalvik VM (sekce 3.1). Řízení procesů a paměti je prováděno zásadně za běhu [21].

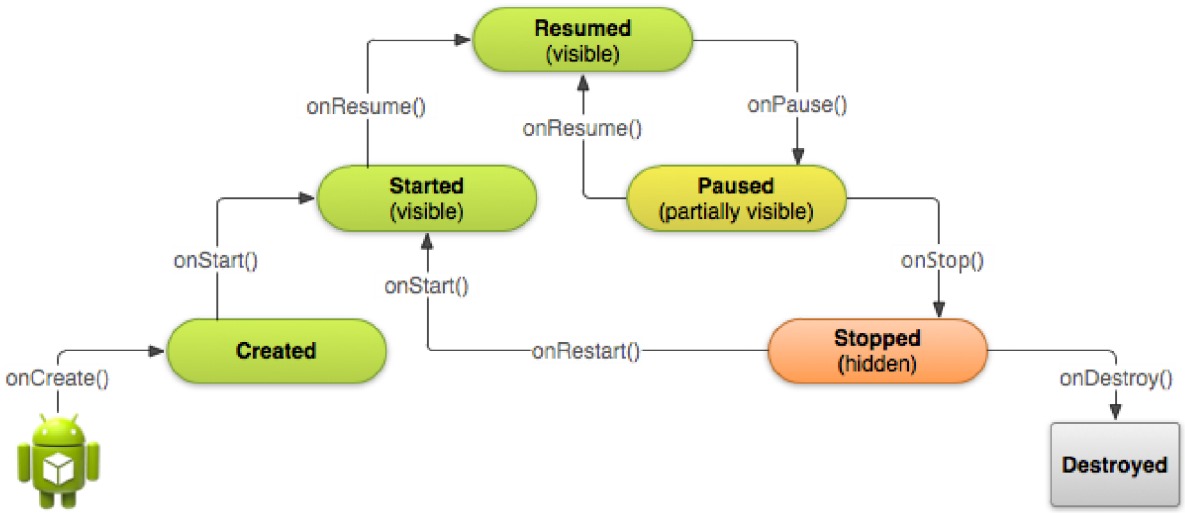


Obrázek 3.3: Priorita procesů v systému Android [21].

Systém Android agresivně spravuje vlastní zdroje a dělá vše nezbytné, aby zajistil plynulý a stabilní chod aplikací. V praxi to znamená, že proces (nebo jeho hostitelská aplikace) může být ukončena v některých případech bez varování, aby se uvolnily systémové prostředky pro více prioritní aplikace. Pro lepší pochopení priorit v systému Android je na obrázku 3.3 ukázka jak Android spravuje priority procesů. Pokud mají dvě aplikace stejnou prioritu, jako první bude ukončen proces, který byl na této prioritě po delší čas. Priorita procesů je také ovlivněna závislostmi mezi procesy. Tedy pokud aplikace závisí na Službě, kterou poskytuje druhá aplikace, má druhá aplikace minimálně stejnou nebo vyšší prioritu než aplikace, které svojí službu poskytuje. Systém Android má ještě jednu důležitou vlastnost, a to, že všechny aplikace běží dál a zůstávají v paměti, pokud nejsou jejich prostředky potřeba pro jiné aplikace [21][24].

Metody pro správu životního cyklu Android Aktivit (Activity)

Na rozdíl od jiných programovacích paradigmat, ve kterých jsou aplikace spouštěny pomocí metody *main()*, systém Android iniciuje kód instance Activity pomocí voláni specifických callback metod, které odpovídají určitým fázím životního cyklu dané Aktivity. V systému Android tedy existuje posloupnost callback metod, které jsou volány jak při startu Aktivity, tak také posloupnost metod volaných pro zastavení činnosti dané Aktivity [21][20].



Obrázek 3.4: Zjednodušený náčrt ilustrující životní cyklus Activity v systému Android [20].

V průběhu života Aktivity systém volá základní sadu metod pro správu jejího životního cyklu v pořadí podobném pyramidě, jak je ilustrováno na obrázku 3.4. Každý stupeň této pyramidy představuje samostatní fázi života Aktivity. Jak systém vytváří novou instanci Aktivity, každá callback metoda posouvá danou Aktivitu a stupeň blíže vrcholu. Vrchol pyramidy pak představuje stav, ve kterém je Aktivita na popředí a uživatel s ní může aktivně interagovat [21].

Jakmile začne uživatel opouštět Aktivitu, systém volá metody, které posouvají stav této Aktivity zpátky dolů pyramidou, za účelem jejího postupného zničení. V některých případech se Aktivity dolu pyramidou posunou jenom částečně a čekají (když například uživatel přepne na jinou Aktivitu). Z tohoto bodu se mohou jednoduše přesunout zpátky na vrchol (pokud se uživatel vrátí k Aktivitě) a pokračovat, kde byly zanechány naposled [20].

Na obrázku 3.4 jsou ilustrovány stavy, kterými může Aktivita v sytému Android procházet, avšak jenom tři z nich mohou být statické. To znamená, že existují pouze tři stavy, ve kterých může Aktivita setrvat delší dobu [20]:

- Resumed – V tomto stavu je Aktivita na popředí a může probíhat komunikace mezi ní a uživatelem (Někdy je také tento stav označován jako „běžíci“ nebo také „running“).
- Paused – V tomto stavu je Aktivita částečně překryta jinou Aktivitou, to znamená, že Aktivita, která je na popředí je polo-průhledná nebo jednoduše nepokrývá celou obrazovku. Aktivita, která je pozastavena, nepřijímá žádné vstupy od uživatele a nemůže vykonávat ani žádný kód.
- Stopped – V tomto stavu je Aktivita kompletně překryta jinou Aktivitou a není viditelná pro uživatele, má se za to, že je v pozadí. Zatím co je Aktivita ve stavu „stopped“, její instance a všechny informace o jejím stavu jako členské proměnné jsou zachovány, ale nemůže vykonávat žádný kód.

Zbylé dva stavy (Created a Started) jsou jenom přechodné a systém se z nich rychle přesune do dalších stavů voláním callback metod. Kupříkladu, potom co systém zavolá `onCreate()`, rychle volá `onStart()`, který je opět rychle následován `onResume()` [20].

3.3 Vývojové prostředí – Android Studio

Po dlouhou dobu byl pro Android aplikace oficiálně podporovaným vývojovým prostředím Eclipse v spojení s ADT (Android Development Tools). ADT je plugin pro Eclipse, který mu poskytuje sadu integrovaných nástrojů a vlastností, které zjednodušují vývoj Android aplikací. Také poskytuje grafický přístup k některým nástrojům příkazového řádku dostupných přes SDK (Standard Development Kit), stejně tak nástroje pro vytváření a design uživatelského rozhraní aplikace [25].

V roce 2013 představila společnost Google Android Studio jako nové vývojové prostředí pro Android aplikace. Od té doby prošlo výrazným vývojem a stalo se z něj propracované a pro vývoj Android aplikací optimalizované prostředí. Také je momentálně na stránkách Androidu označované jako jediné oficiálně podporované vývojové prostředí[11]. V době psaní této práce byla nejnovější stabilní verze Android Studia 1.2.2. To byla také verze, na které se vytvářela aplikace této práce.

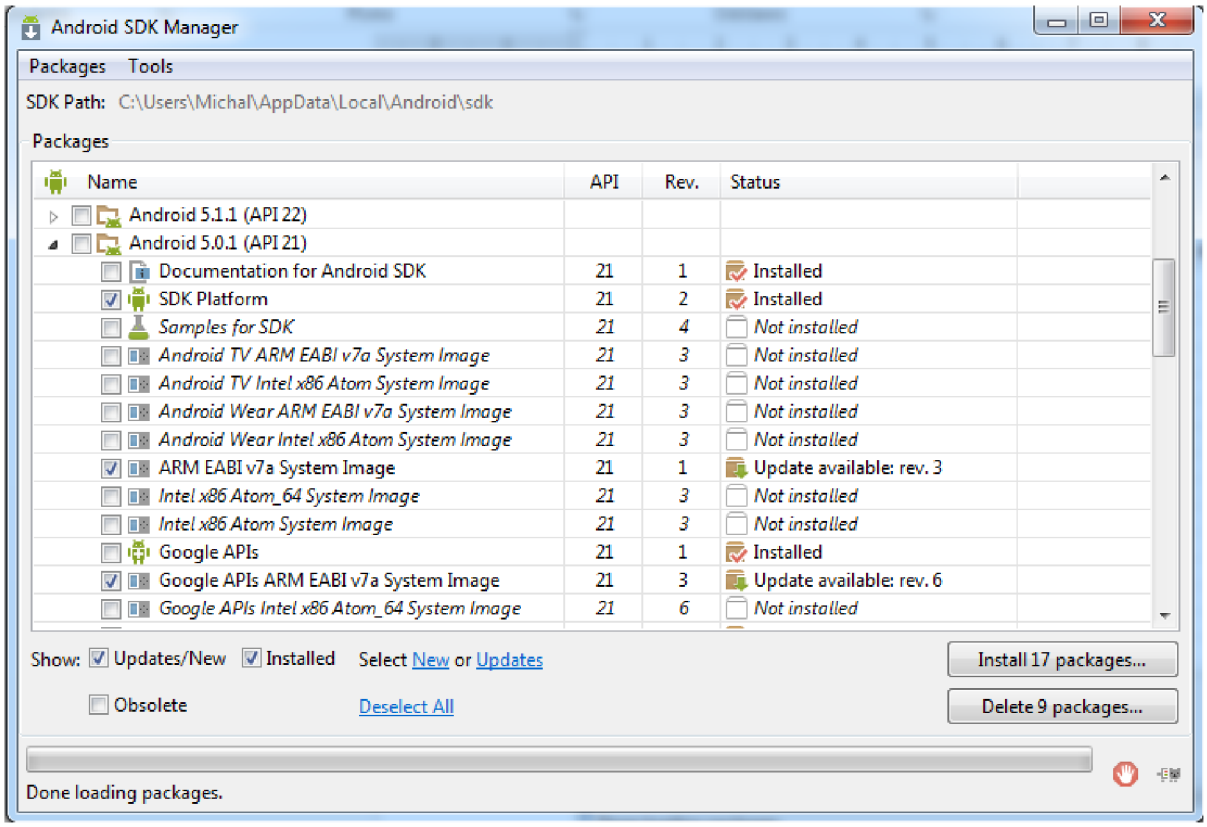
Jednotlivé kroky pro zprovoznění Android Studia, vytváření nové aplikace a základní popis prostředí budou popsány v následujících odstavcích této podkapitoly.

Instalace a konfigurace Android Studia

Android Studio je volně ke stažení z oficiálních stránek[10] a je šířeno pod licencí Apache Licence 2.0 [22]. Nacházejí se tam verze jak pro Windows, tak pro Mac OS X a Linux. Jelikož jsou Android aplikace primárně psané v jazyce Java, je potřebné mít před samotnou instalací Android Studia nainstalované JDK (Java Development Kit). Systém musí být pak schopen tuto instalaci Java najít. To může být dosaženo nastavením proměnné prostředí s názvem `JAVA_HOME`, která musí ukazovat na adresář JDK [22]. Podle informací z oficiálních stránek je podporovaná jenom verze Oracle JDK[11], která ale není dostupná v repositářích Ubuntu (v průběhu tvorby aplikace bylo ale otestováno, že Android Studio pracuje také s verzí OpenJDK, což je Open Source verze Javy). Ve Windows se instalace spouští pomocí exe souboru a spolu s prostředím Android Studia se instaluje také Android SDK. Pokud není nastaveno jinak, je SDK naistalováno do adresáře `\Users\<user_name>\Appdata\Local\Android\android-studio`, kde `Appdata` je většinou skrytý adresář [22]. Pro Linux je jenom potřeba rozbalit TGZ soubor a spustit skript `studio.sh` umístěný v adresáři `/home/user_name/android-studio/bin/`.

Ve výchozím nastavení Android SDK neobsahuje vše potřebné pro vývoj aplikací. SDK odděluje nástroje, platformy a jiné komponenty do balíčků, které je možné stáhnout pomocí Android SDK Manager. Obrazovka SDK Manageru je zobrazena na obrázku 3.5.

[10] Web: https://developer.android.com
[11] Zdroj: https://developer.android.com/sdk/index.html



Obrázek 3.5: Obrazovka aplikace Android SDK Manager.

Jak je vidět z obrázku 3.5 SDK Manager poskytuje jednotlivé balíčky rozdělené podle verzí API. Jako minimum je potřebné nainstalovat následující balíčky [22]:

- Android SDK Tools,
- Android SDK Platform-tools,
- Android SDK Build-tools (v nejvyšší dostupné verzi).

A také pro konkrétní verzi API (v ukázce API 21 pro verzi systému Android 5.0.1):

- SDK Platform,
- Systémový obraz pro emulátor, například ARM EABI v7a Systém Image.

Pro plynulý chod emulátoru systému Android je ještě zapotřebí naistalovat:

- Intel x86 Emulator Accelerator (HAXM installer – Hardware Accelerated Execution Manager).

Tento akcelerátor k tomu aby fungoval, vyžaduje hardwarovou podporu technologie Intel VT (Intel Virtualization Technology) v procesoru počítače [22].

Vytváření projektu

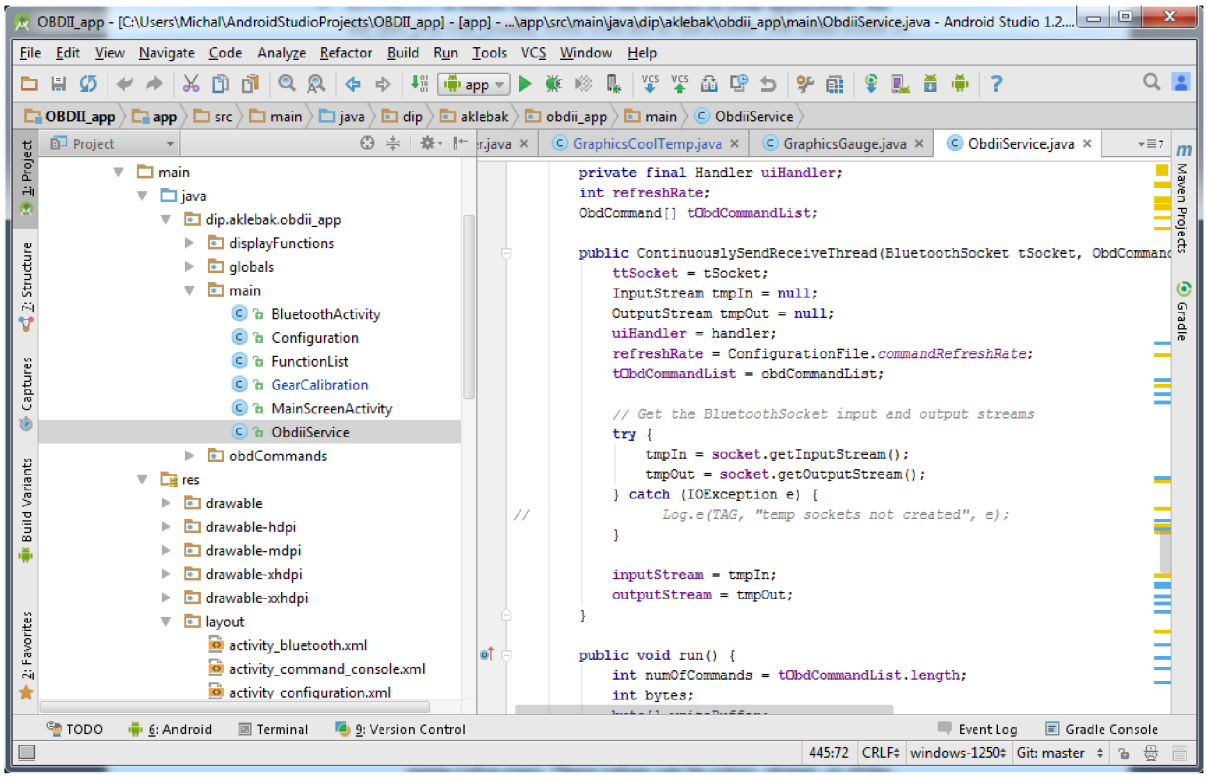
Průvodce pro vytvoření nového projektu je možné vyvolat z uvítací obrazovky, nebo pokud ta není zobrazena, pomocí File | New | New Project. Průvodce postupně zobrazuje obrazovky, kde v první se

nastavuje název projektu a jeho umístění na disku. V druhé pak cílová zařízení, tedy platformy, které má výsledná aplikace podporovat (jako mobil, tablet, televize a jiné) spolu s minimální verzí API, která v podstatě určuje nejstarší podporovanou verzi systému Android. Více o verzích Android API bylo popsáno v kapitole 3. Třetí a čtvrtá obrazovka umožňují přidat do aplikace základní Aktivitu [22].

Navigace v projektu a editoru zdrojového kódu

Po vytvoření projektu se otevře hlavní okno Android Studia. Na pravé straně kliknutím na záložku projekt v navigačním panelu je možné otevřít projektové menu, ve kterém se zobrazují všechny právě otevřené projekty. V tomto menu je možné spravovat soubory projektu, přidávat nové, kopírovat je, porovnávat atd. Jak je možné vidět na obrázku 3.6, po kliknutí na požadovaný projekt se zobrazí jednotlivé struktury projektu se všemi soubory. Nejdůležitější elementy struktury projektu jsou [22]:

- *build/:* obsahuje přeložené soubory po sestavení projektu
- *libs/:* obsahuje knihovny potřebné v projektu
- *src/main/:* obsahuje zdrojové soubory aplikace a je dále rozdělen na podadresáře:
    - *java/:* obsahuje Java třídy organizované do balíčků
    - *res/:* obsahuje zdroje (resources) projektu, dále organizovány do podadresářů:
        - *drawable/:* obsahuje obrázky a ikony použité v aplikaci, většinou ve více rozlišeních
        - *layout/:* obsahuje XML soubory definující vzhled obrazovek aplikace
        - *menu/:* obsahuje XML definice jednotlivých menu nabídek v aplikaci
        - *values/:* obsahuje XML soubory definující hodnoty jako barvy, řetězce
    - *AndroidManifest.xml:* je to nezbytný soubor v Android aplikacích, který je generován automaticky a nese základní informace potřebně pro spuštění aplikace [25].

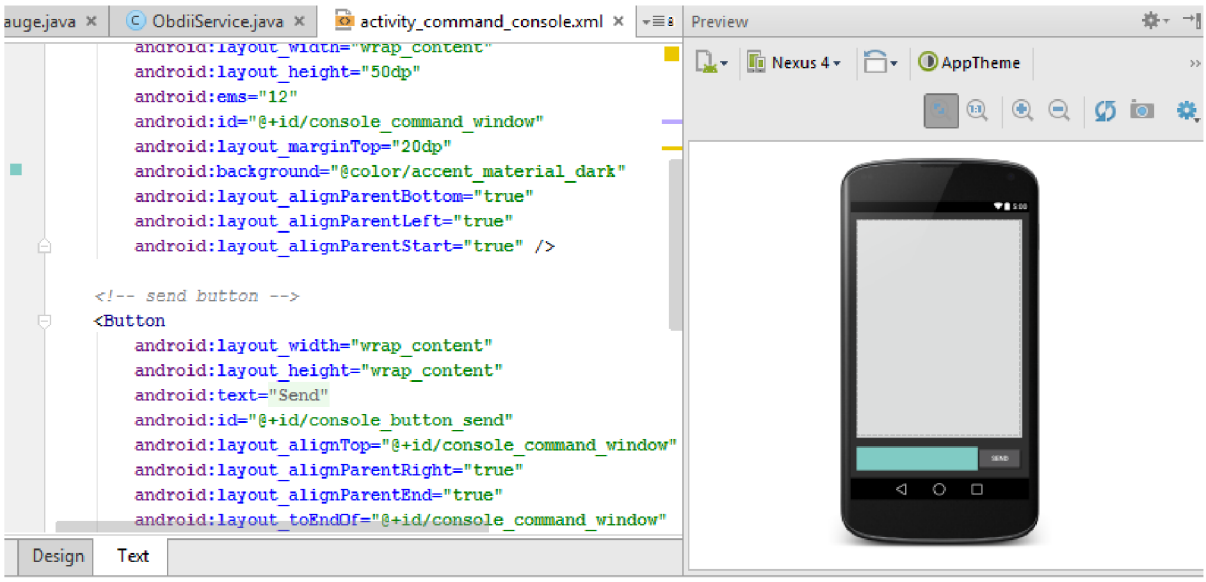


Obrázek 3.6: Otevřený projekt v prostředí Android Studia.

Pro editaci zdrojových kódů slouží v Android Studiu pravá část okna, jak je patrné z obrázku 3.6. Editor dovoluje všechny pokročilé funkce jako dokončování kódu, automatické generování kódu, zobrazování dokumentace a mnoho dalších. Android Studio je pokročilý nástroj, postavený na InteliJ IDEA, což je propracované vývojové prostředí pro tvorbu Java aplikací. V tomto odstavci byly zmíněny jenom základní funkce a vlastnosti, kompletní popis celého prostředí se všemi detaily je možné v případě zájmu nalézt v odkazech na literaturu [22].

Tvorba grafického uživatelského rozhraní

Android Studio umožňuje tvorbu grafického uživatelského rozhraní pomocí grafického nebo textového editoru. Jak již bylo popsáno v předchozí kapitole, definice grafického uživatelského rozhraní je uložena v XML souborech ve složce layouts/ [21].



Obrázek 3.7: Textový editor grafického uživatelského rozhraní v prostředí Android Studia.

Na obrázku 3.7 je zobrazen textový editor pro úpravu uživatelského rozhraní. Komponenty jsou přidávány do rozložení pomocí XML deklarací v levé části obrazovky. Na obrázku 3.7 vlevo je část definice tlačítka, ve které je definováno jeho id, umístění, popisek a další parametry. V levé části obrazovky se pak nachází náhled vytvářeného rozložení. V ovládacím panelu je možné si zvolit typ zařízení, na kterém se má náhled zobrazovat. Tak je možné připravit více verzí rozložení obrazovky pro různé velikosti displejů.

Uživatelské rozhraní je možné vytvářet také v grafickém editoru, který obsahuje náhled rozložení podobný tomu v textovém editoru s tím rozdílem, že je možné měnit pozice jednotlivých elementů a přidávat nové přímo v tomto náhledu jednoduchým tažením myší [22].

4 Zhodnocení současného stavu

V této kapitole se zaměřím na stručný popis a zhodnocení některých existujících hardwarových a softwarových řešení zabývajících se problematikou komunikace a zpracováním dat z moderních automobilů. Také nastíním, jakým směrem by se mělo ubírat samotné řešení této práce.

4.1 Zhodnocení současných hardwarových řešení

Tak jako všeho v dnešní době, existuje také zařízení pro komunikaci s automobily velké množství. Jak již bylo zmiňováno v předchozí kapitole, většina takových zařízení je specifických pro konkrétního výrobce nebo dokonce model vozidla. Těch skutečně univerzálních existuje jenom pár. Proto budou tady zmiňována jenom zařízení, která jsou kompatibilní s testovaným vozidlem (Opel Astra H Gtc).

Řešením přímo od výrobce je zařízení nazývané GM Tech2, které používají dealeři značky Opel a také ostatních značek ze skupiny General Motors (jako Chevrolet, Saab a další). Dokáže komunikovat se všemi jednotkami ve vozidle a spouštět na nich diagnostické testy. Pokrývá vozidla vyrobená v rozmezí let 1992 až přibližně do roku 2013, takže také testované vozidlo. GM Tech2 je vestavěné zařízení, které nevyžaduje připojení k počítači. Po koupi zařízení už nevyžaduje obnovování licence, jen v případě potřeby dokoupení aktualizace pro podporu novějších vozidel. Jedná se ale o profesionální zařízení v ceně přes sto tisíc korun, kterého koupě nemá pro běžného uživatele velký význam. Na internetu jsem narazil také na čínské kopie tohoto zařízení v ceně kolem deseti tisíc korun, ale jestli poskytují stejnou funkcionalitu jako originální zařízení nebo jestli vůbec fungují, se je možné jenom domnívat.

Od roku 2010 začal General Motors nahrazovat GM Tech2 zařízením MDI (Multiple Diagnostic Interface), což je komunikační modul, který zabezpečuje přenos dat mezi řídícími jednotkami ve vozidle a počítačem. MDI funguje v spolupráci se softwarem GDS2 (Global Diagnostic Software 2) a spolu poskytují přibližně stejnou funkcionalitu jako GM Tech2. Cenově to ale stále není nic pro běžného uživatele, cena MDI se pohybuje kolem čtyřiceti tisíc korun a ročné předplatné GDS2 přibližně od deseti tisíc. Tady také existují čínské verze, jejichž funkčnost lze opět jen stěží odhadnout.

Tohle byla řešení určena pro profesionální použití, které si mohou oficiálně pořídit jenom autorizovaná servisní střediska, neoficiálně samozřejmě kdokoliv, kdo je ochotný za ně zaplatit. Pravděpodobně, ale moc hobby nadšenců ochotných zaplatit takové peníze nebude.

Na opačné straně stojí zařízení, která jsou určena primárně pro hobby použití a poskytují spíš jenom základní funkcionalitu. Jedním takovým jsou zařízení postavená na mikrokontroléru ELM327 od Elm Electronics, který byl podrobně popsán v kapitole 2.5 této práce. Je vytvořen tak, aby umožňoval komunikaci s automobily na protokolu OBDII a umožnil tedy zjišťování informací od systémů automobilu vztažených hlavně k emisím. To je také účel, ke kterému se používá ve většině případů. Při jeho vytváření ale do něj přidali také sadu příkazů pro komunikaci na CAN sběrnici, což umožňuje alespoň částečné rozšíření jeho základní funkcionality. ELM327 zaznamenal v poslední době obrovský rozmach, paradoxně kvůli rozšíření čínských kopií tohoto mikrokontroléru. Díky tomu je ho možné pořídit v řádech sto korun a existuje v různých mutacích na USB, Bluetooth nebo RS232, dokonce v poslední době se začali objevovat také verze s Wi-Fi.

Dalším hobby řešením podobným ELM327 je mikrokontrolér STN1110. Sice je od jiného výrobce, ale kvůli kompatibilitě s ELM327 používá stejnou sadu AT příkazů, kterou rozšiřuje o další

příkazy. Jeho hlavní předností proti ELM327 je rychlost UART rozhraní, která je až 10 Mbps proti 500kbps u ELM327, což ho může činit vhodnějším pro použití s moderními automobily. Používá jej například OBDII řešení pro Arduino, nazývaný Vehicle OBD2 Shield od výrobce Sigalabs s cenou lehce přes tisíc korun.

4.2 Zhodnocení současných softwarových řešení

Profesionální řešení tady podrobně popisovat nebudu. Jedním takovým je GDS2 již zmiňovaný v kapitole 4.1 a druhým TIS2000, oba určené pro vozy skupiny General Motors. Z hobby OBDII softwarových řešení dostupných pro platformu Android jsem v průběhu vytváření této práce přišel do kontaktu s dvěma aplikacemi, obě dostupné přes Android Play Store (Obchod s aplikacemi v systému Android). První z nich byla aplikace s názvem „OBD Car Doctor“ od PNN Soft, která má přes milion stažení. Pro komunikaci s vozidlem vyžaduje ELM327 v Bluetooth verzi. Dá se tedy předpokládat, že většina těch, který si tuto aplikaci stáhli, ELM327 také vlastní. Z výše uvedeného si můžeme tedy odvodit poměrně značnou rozšířenost těchto zařízení.

Druhou aplikací byla „Elm327 OBD Terminal“ od Qbek. Jak název napovídá, jedná se o aplikaci ve stylu příkazového řádku, která dovoluje přímo posílat příkazy pro ELM327 bez dalšího zpracování. Při testování jednotlivých příkazů je to velice dobrý pomocník.

Pro systém Android existuje mnoho podobných aplikací poskytujících přibližně stejnou funkcionalitu. Stačí zadat výraz „OBD“ do vyhledávače v Android Play Store a dostanete seznam desítek různých aplikací. Všechny jsem samozřejmě neprošel, ale většina z nich umožňuje zobrazit a vymazat chybové kódy, zobrazuje dostupná data v reálném čase a některé z nich ukazují také spotřebu paliva. Nachází se tam také jedna aplikace speciálně pro automobily značky Opel s názvem „ScanMyOpel“. Zobrazuje několik údajů, které se ve standardu OBDII nevyskytují, jako detailní informace o ECU a informace z některých snímačů, ale nepřináší žádný zásadní přínos vůči „obyčejným aplikacím“.

Pro komunikaci s ELM327 z prostředí počítače postačuje jednoduchý emulátor terminálu, který se dokáže připojit k sériovému portu. Z prostředí Linuxu to může být například Picocom, který jsem používal já v průběhu této práce.

Od verze jádra 2.6.25 je v Linuxu obsažen SocketCAN, což je nízkoúrovňový CAN framework (Low Level Can Framework), který umožňuje pracovat s CAN rozhraním stejným způsobem, jako by se jednalo o síťové rozhraní. Jeho použití by bylo ideální pro monitorování provozu na komunikační sběrnici vozidla. Následně by bylo možné interpretovat význam některých zpráv zachycených na sběrnici. Bohužel SocketCAN není kompatibilní s ELM327 a pro účely této práce nebylo k dispozici jiné kompatibilní zařízení.

4.3 Vylepšení současných řešení a prvotní návrh aplikace

Jelikož bude výslední aplikace určena pro použití v automobilech, je logickou podmínkou její použití v mobilních zařízeních. Chytré telefony nebo tablety dnes vlastní téměř každý. Podle kapitoly 3 je patrné, že nejrozšířenějším operačním systémem pro mobilní zařízení je v současnosti systém Android. Navíc já také vlastním telefon s Androidem a jako programovací jazyk preferuji Javu, aplikace by tedy měla být vytvořena právě pro tuto platformu.

Jako další je potřeba vybrat rozhraní pro komunikaci mezi Android zařízením a automobilem. Mělo by to být cenově dostupné zařízení, které bude možné si bez problémů pořídit tak, aby se toto nestalo hlavní brzdou rozšíření aplikace mezi uživatele. Dále by bylo vhodné, aby se toto rozhraní připojovalo k aplikaci bezdrátově. Jelikož asi každé mobilní zařízení v dnešní době disponuje Bluetooth rozhraním, bude toto asi nejlepší volba. Těmto požadavkům bez problémů vyhovuje zařízení založené na mikrokontroléru ELM327 (viz kapitola 2.5), dostupné také ve verzi s Bluetooth rozhraním.

Aplikací, které poskytují obecní funkcionalitu spojenou s OBDII rozhraním existuje pro Android velké množství (viz. kapitola 4.2). Proto by bylo podle mně zbytečné vytvářet další aplikaci, která by jenom kopírovala stejné funkce, jaké poskytují tyto aplikace. Z tohoto důvodu bych vůbec neimplementoval například načítání a mazání chybových kódů, na které jsem narazil prakticky v každé aplikaci. Naopak bych se zaměřil na věci, které podle mého názoru v těchto aplikacích chybí. Aplikace by měla být navíc navržena tak, aby bylo možné ji v budoucnu podle potřeby dále rozšiřovat a přidávat další funkce.

Jedna část aplikace by tedy sloužila jako rádce pro ekonomickou jízdu. Zobrazovala by rychlost vozidla a otáčky motoru. Určitě tam nesmí chybět také spotřeba paliva, jak průměrná, tak aktuální. Dobrý motivační faktor by byl ukazatel, který by zobrazoval cenu jízdy, tedy kolik zatím jízda stála. Ve stylu počítadla kilometrů, ale místo vzdálenosti by na něm naskakovala suma. Spotřeba také záleží od schopnosti správně řadit rychlosti, proto bych tam přidal také ukazatel aktuálně zařazeného převodového stupně spolu s ukazatelem, který by doporučoval, jestli zařadit vyšší nebo nižší převodový stupeň. Dále také ukazatel polohy plynového pedálu, aby si byl řidič vědom, jak moc naň šlape.

Další obrazovka aplikace by obsahovala příkazový řádek, který umožní uživateli komunikovat přímo s mikrokontrolérem ELM327. To při znalosti příkazů tohoto mikrokontroléru dovolí provádět úkony a zjišťovat údaje, které nebudou standardně součástí této aplikace.

Pro potřeby obrazovky rádce pro ekonomickou jízdu budou implementovány příkazy pro zjišťování všech potřebných údajů od automobilu. Jelikož už bude přijímání těchto dat stejně součástí aplikace, umožnil bych na samostatné obrazovce graficky zobrazit tyto jednotlivá data, jako rychlost vozidla, otáčky motoru, teplotu chladicí kapaliny, polohu škrtící klapky a další.

Aplikace by dále měla umožnit uživateli měnit některá nastavení podle svých preferencí. Jako třeba cenu paliva pro výpočet ceny jízdy v obrazovce rádce pro ekonomickou jízdu nebo podmínky, podle kterých bude aplikace rozhodovat, zda by se měl zařadit vyšší nebo nižší převodový stupeň v té samé obrazovce.

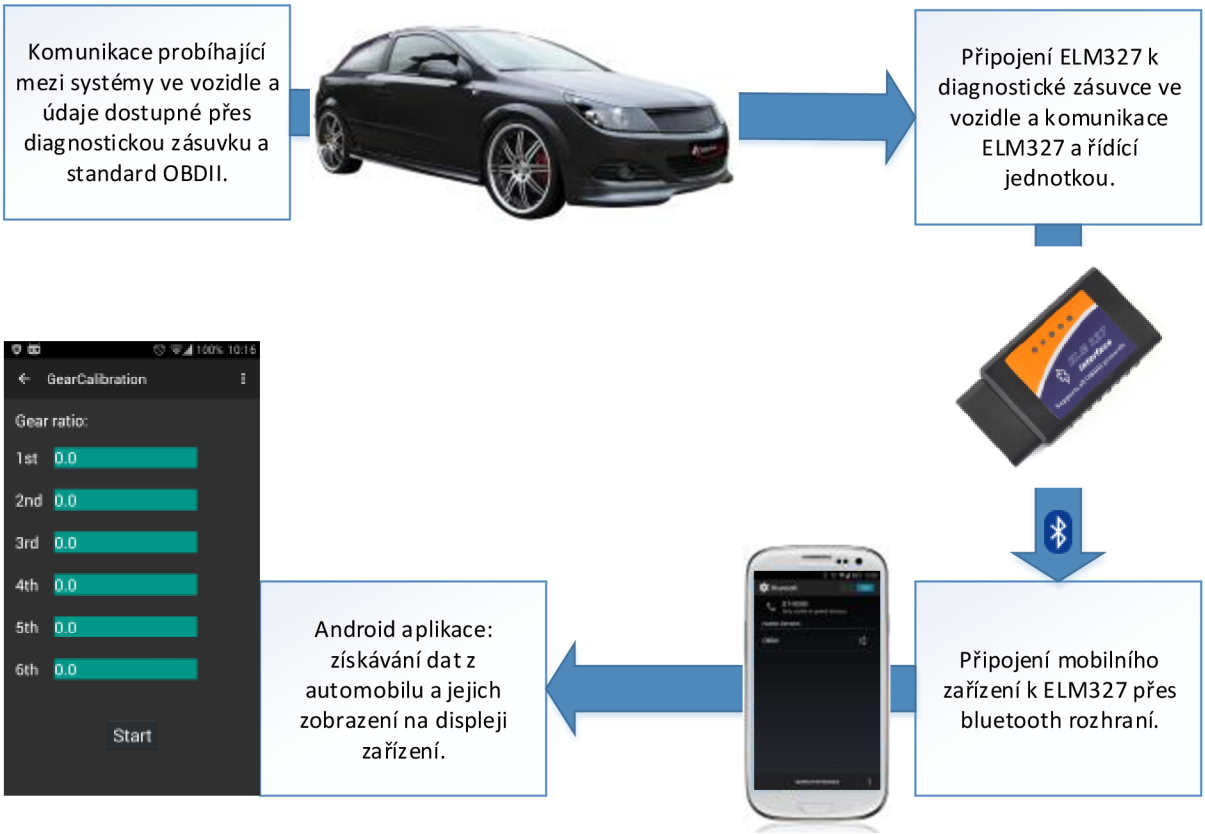
Rád bych do aplikace také zařadil funkci, na kterou jsem nenarazil u žádné z aplikací, které jsem zatím viděl. Bohužel se jedná o něco, co bude úzce spjaté s konkrétním vozidlem, v tomto případě s testovaným Opel Astra H Gtc. A tedy využít toho, že většina automobilů používá stejnou sběrnici, na které komunikuje protokol OBDII, také pro komunikaci mezi vlastními systémy ve vozidle. Pokud se tedy povede najít na stejné sběrnici třeba zprávy, které posílá modul volantu do rádia, bylo by možné toto rádio ovládat odesláním stejné zprávy zpět na sběrnici. Takto by bylo možné ovládat třeba stahování oken nebo spuštění stěračů nebo cokoliv jiného co je ovládáno pomocí zpráv na CAN sběrnici. V této fázi samozřejmě není jasné, jaké a jestli vůbec nějaké systémy komunikují na stejné sběrnici jako OBDII nebo jestli bude možně se případně připojit na jinou sběrnici. Zjištění těchto informací a také návrh spolu s popisem implementace tady načrtnuté aplikace budou součástí dalších kapitol.

5 Návrh řešení pro komunikaci s elektronickými systémy v automobilu

Následující kapitola se věnuje návrhu samotné aplikace pro komunikaci s vozidlem přes diagnostickou zásuvku pomocí protokolu OBDII, respektive EOBD, ale také komunikací mimo standardu. Na základě znalostí a informací o možnostech komunikace s a mezi elektronickými systémy v automobilech, popsanými v kapitole 2 a také na základě neformálního návrhu popsaného v kapitole 4, jsem navrhl systém, který umožní:

- Zobrazení údajů z automobilu v reálném čase, dostupných přes diagnostickou zásuvku a definovaných podle OBDII standardu.
- Zobrazení údajů, které nejsou dostupné přes diagnostickou zásuvku, ale je možné je z těchto údajů vypočítat.
- A nakonec, který bude schopný komunikovat přímo s některým systémem v automobilu a ovlivňovat jeho funkčnost (například rádio).

Koncepce celého systému je ukázána na blokovém diagramu na následujícím obrázku 5.1. Poté je popsán princip činnosti celého systému a také detailněji popsány stěžejní části návrhu aplikace.



Obrázek 5.1: Zjednodušené blokové schéma navrženého řešení.

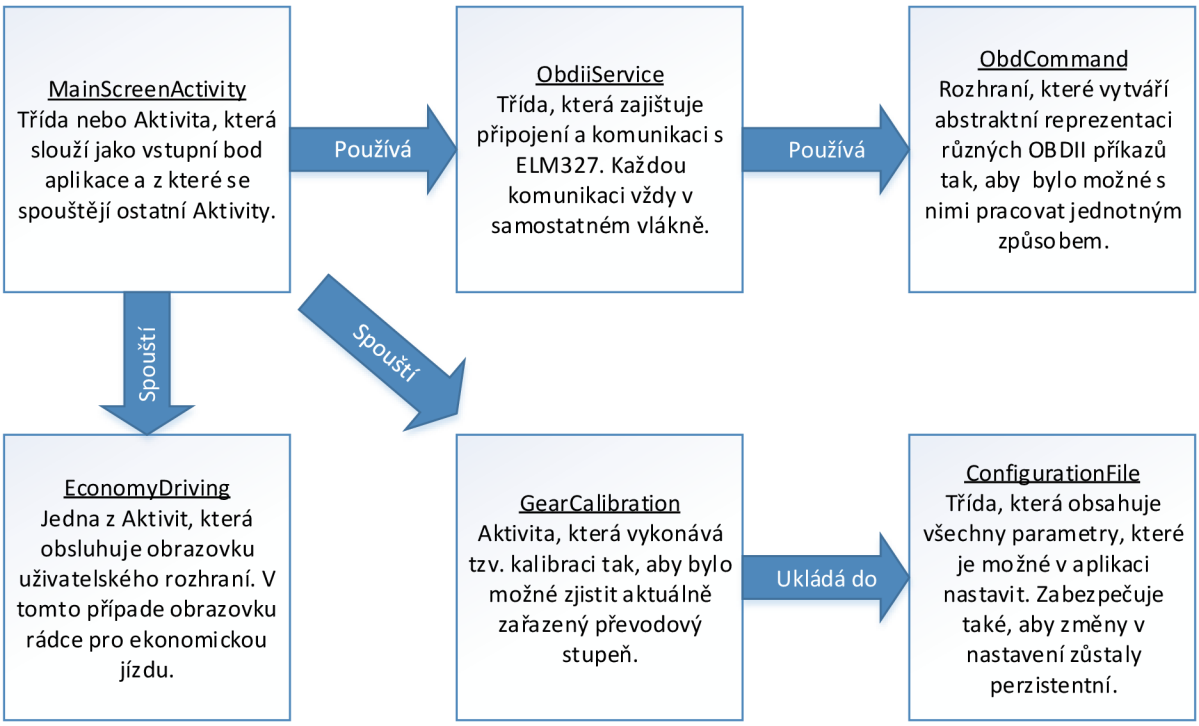
Navržený systém pro komunikaci s elektronickými systémy v automobilu, kterého zjednodušené blokové schéma je zobrazeno na obrázku 5.1, bude založen na následujícím principu:

- V nastartovaném automobilu neustále probíhá komunikace mezi jednotlivými řídicími systémy a moduly (určitá komunikace probíhá také při vypnutém motoru, je však výrazně menší). Tato komunikace probíhá na sběrnicích, které jsou většinou přístupné přes diagnostický konektor. Na něm je dostupné také OBDII rozhraní.
- K tomuto konektoru bude připojeno zařízení s mikrokontrolérem ELM327, které bude sloužit jako překladač mezi OBDII, případně CAN rozhraním na jedné straně a sériovou linkou na straně druhé (přístupné přes Bluetooth rozhraní).
- Mobilní zařízení, na kterém poběží výslední aplikace, se bude k ELM327 připojovat přes Bluetooth rozhraní (na straně mobilního zařízení je ELM327 přístupné jako virtuální sériová linka).
- Na mobilním zařízení poběží aplikace, která bude komunikovat s ELM327 a získávat přes něj údaje od automobilu. Tyto údaje pak bude prezentovat uživateli na displeji chytrého telefonu.

V následujících podkapitolách bude popsán návrh základních tříd aplikace (podkapitola 5.1) a také návrh grafického uživatelského rozhraní aplikace (podkapitola 5.2). Implementace jednotlivých částí navrženého systému pro komunikaci s elektronickými systémy v automobilu a zobrazení přijatých dat je popsána v kapitole 6.

5.1 Návrh aplikace pro systém Android

Na obrázku 5.2 je zobrazeno schéma základních tříd navrhované aplikace. Je to zjednodušené schéma, proto neobsahuje všechny třídy, ale jenom ty nejdůležitější.



Obrázek 5.2: Zjednodušené schéma tříd aplikace.

Na začátku stojí Aktivita `MainScreenActivity`, která bude vstupním bodem aplikace. Bude obsluhovat základní obrazovku, z které bude možné a také nezbytné se připojit k ELM327 přes Bluetooth rozhraní a také bude sloužit jako rozcestí pro přístup k dalším obrazovkám, respektive funkcím aplikace. Jednou z nich bude `EconomyDriving`, která je zobrazena také ve schématu na obrázku (ostatní byly pro přehlednost ze schématu vynechány). Ta bude zobrazovat rádce pro ekonomickou jízdu a zabezpečovat všechny potřebné výpočty pro zobrazení na této obrazovce.

Samotné připojení k ELM327 a proces komunikace bude mít na starosti třída `ObdiiService`, která pro tyto účely vytvoří vždy nové vlákno tak, aby nebylo přetěžováno primární vlákno aplikace. Jednotlivé příkazy pro ELM327 budou reprezentovány pomocí rozhraní `ObdCommand`, to z důvodu, aby bylo možné s různými příkazy pracovat jednotným způsobem.

Některé funkce aplikace budou pro svoji činnost vyžadovat uživatelsky nastavitelné parametre. Ty budou uloženy ve třídě `ConfigurationFile` tak, aby byly přístupné z kterékoliv části aplikace. Třída `ConfigurationFile` se bude starat také o to, aby změny v nastaveních zůstaly perzistentní i po ukončení aplikace. Poslední Aktivitou, která je zobrazena v diagramu na obrázku 5.2, je `GearCalibration`. Pomocí této Aktivity bude prováděna kalibrace převodových stupňů (zjištění poměru mezi rychlostí a otáčkami pro jednotlivé převodové stupně) tak, aby bylo možné v případě potřeby v aplikaci zjistit aktuálně zařazený převodový stupeň.

V této kapitole byl návrh a popis jenom základních tříd. Aplikace bude samozřejmě obsahovat ještě další důležité třídy a také řadu pomocných, které byly pro zjednodušení z popisu vynechány.

Výběr verze Android API

V době psaní této práce byl nejnovějším systémem Android verze 5.1.1 Lollipop z dubna 2015 a k tomu přislouchající verze API level 22. Na nejvyšší verzi zatím běží samozřejmě minimum zařízení (jak je možné vidět z tabulky 3.1), proto není rozumné volit tuto jako cílovou verzi platformy. Dál bylo samozřejmě potřeba brát v úvahu hardware dostupný pro testování vytvářené aplikace. K dispozici byl chytrý telefon Samsung Galaxy S3 (i9300) s neoriginálním systémem CyanogenMod ve verzi 10.2.0 postaveném na Androidu 4.3.1, tedy Jelly Bean.

Z tabulky 3.1 je patrné, že přes 94% zařízení používá Android 4.0.3 a novější. Jak již bylo zmiňováno, na testovacím zařízení běží Android 4.3.1, vyžaduje tedy maximálně API level 18. Z tohoto důvodu a z důvodu maximální zpětné kompatibility byl tedy zvolen jako minimální API level 15. Volit ještě nižší by bylo z hlediska většího počtu podporovaných zařízení prakticky zbytečné, jen bychom přicházeli o některé důležité funkce nedostupné v starších verzích. Pro potřeby testování aplikace bohužel nebyl kdispozici žádný tablet se systémem Android a ani jiný chytrý telefon, proto bude aplikace vyladěna primárně pro Samsung Galaxy S3 a zařízení s podobnou velikostí displeje. Bude samozřejmě fungovat aj na jiných zařízeních se systémem Android 4.0.3 nebo novějším, ale nebude zaručeno správné rozložení jednotlivých prvků uživatelského rozhraní.

5.2 Návrh grafického rozhraní aplikace

Aplikace je navržena primárně pro chytré telefony, protože tablet se systémem android nebyl k dispozici. Neznamená to, že by aplikace na větším displeji nefungovala, ale rozložení jednotlivých prvků uživatelského rozhraní je uzpůsobené displeji o velikosti 4,8 palce. Android aplikace je možné testovat také na virtuálním stroji, na kterém běží systém Android. Avšak pro běh takového systému použitelnou rychlostí je vyžadována podpora technologie Intel VT-x v procesoru, kterou bohužel počítač, na kterém byla aplikace vyvíjena, nedisponoval.

Vzhled aplikace je laděn spíše do decentního tmavého vzhledu a vychází z Android šablony „Holo Dark“, která je výchozí šablonou alespoň pro systém CyanogenMod založený na Androidu, který byl naistalován na dostupném zařízení.

Základním předpokladem příjemného uživatelského rozhraní je, že musí být jednoduché a dostatečně intuitivní. Proto byl vytvořen návrh, jak by na sebe jednotlivé obrazovky uživatelského rozhraní měly navazovat. Návrh byl vytvořen ve formě diagramu aktivit, který se nachází v příloze B. Tento diagram pouze ukazuje jednotlivé obrazovky aplikace, které jsou na něm zachyceny pomocí modrých oválů. Na hranách jsou pak uvedeny uživatelské interakce, které je možné z dané obrazovky provádět (jedná se pouze o ty nejdůležitější).

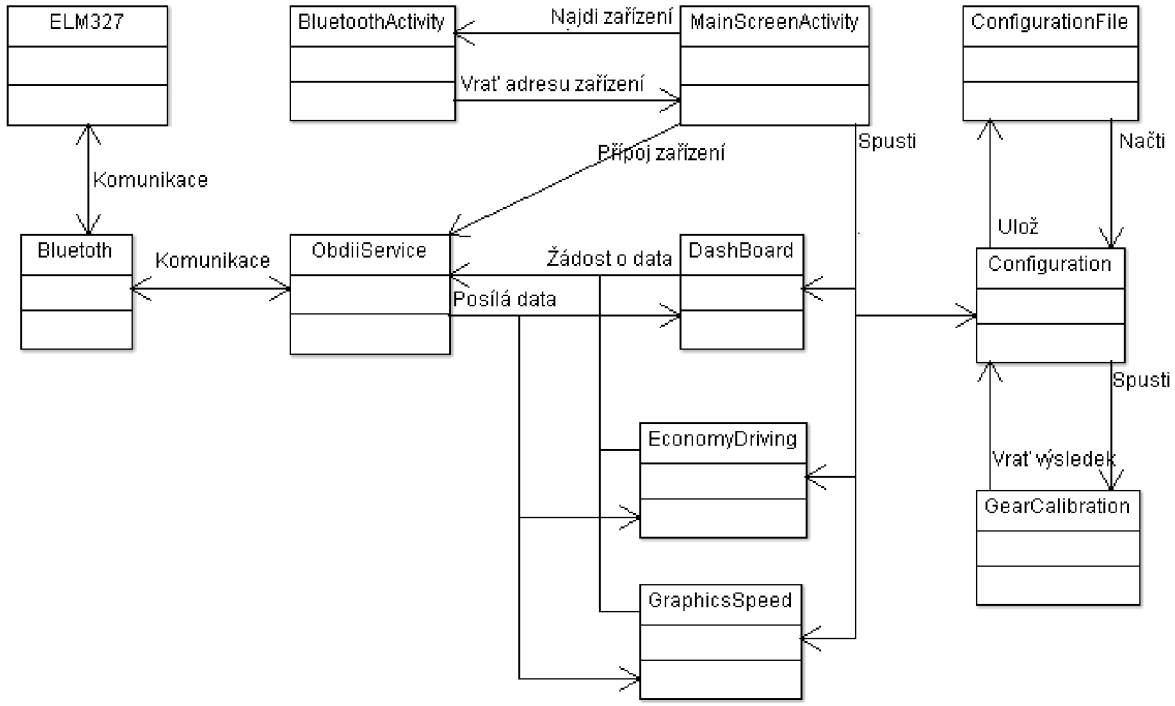
Uživatelské rozhraní bylo navrženo s ohledem na co největší přehlednost. Základem je úvodní obrazovka, která slouží jako rozcestník pro další funkce aplikace, je z ní možné (a také nezbytné) se připojit pomocí Bluetooth k rozhraní ELM327 a poté se navigovat na jednu z následujících obrazovek:

- DashBoard – poskytuje zobrazení dat od vozidla v reálném čase, jako rychlost vozidla, otáčky motoru, teplota motoru a další (podobně jako palubní přístroje ve vozidle).
- Live data – umožňuje vybrat ze seznamu živá data, které je aplikace schopna zobrazit. Vždy zobrazuje pouze jeden údaj graficky. Je možné zobrazit tyto data:
    - Rychlost vozidla
    - Otáčky motoru
    - Teplota motoru (chladicí kapaliny)
    - Poloha škrtící klapky
- Economy driving – poměrně komplexní obrazovka, která zobrazuje některá aktuální data od vozidla a některá dopočítaná data. Je myšlena jako pomocník řidiče pro ekonomickou jízdu. Bude podrobně probrána v kapitole 6, zaměřující se na realizaci navržené aplikace.
- Console – obrazovka ve stylu příkazového řádku pro zadávání příkazů přímo pro ELM327.
- Control Car Radio – obrazovka, která ovládá vestavěné rádio v automobilu.

6 Realizace navrženého systému

V předešlých kapitolách byly stručně shrnuty nejdůležitější poznatky z oblasti komunikace s elektronickými systémy v automobilech a také získávání dat z automobilů. Byly představeny standardy, které upravují některé aspekty této komunikace a prezentován návrh řešení.

Tato kapitola se věnuje popisu procesu samotné implementace aplikace podle návrhu diskutovaného v kapitole 5. Uvádí hlavní a nejdůležitější části, které spolu vytvářejí celek aplikace. Podává je od abstraktního pohledu až po popis konkrétních tříd a metod.



Obrázek 6.1: Diagram tříd navrženého řešení.

Na obrázku 6.1 je zobrazen zjednodušený diagram tříd, který podává abstraktní pohled na propojení základních tříd a komponentů aplikace.

Pro komunikaci s automobilem bylo jako rozhraní vybrané zařízení obsahující mikrokontrolér ELM327 a Bluetooth rozhraní. Na jedné straně je zařízení připojeno k diagnostické zásuvce automobilu (viz Podkapitola 6.1), na straně druhé k mobilnímu zařízení pomocí Bluetooth rozhraní. Toto připojení zabezpečují třídy `ObdiiService` a `BluetoothActivity`, detaily ohledně Bluetooth připojení jsou popsány v sekci 6.2. Další důležitou částí aplikace je samotná komunikace s vozidlem prostřednictvím ELM327 zařízení, o které pojednává podkapitola 6.3. V ní se nacházejí implementační detaily třídy `ObdiiService`, která představuje stěžejní třídu celé aplikace, zabezpečující veškerou komunikaci s vozidlem. V sekci 6.4 pak bude popsána primární obrazovka aplikace, včetně některých důležitých implementačních detailů. Je ní obrazovka rádce pro ekonomickou jízdu (Economy driving), která využívá data dostupná od vozidla způsobem, který by měl řidiče motivovat k ekonomičtější jízdě. Kromě obrazovky pro ekonomickou jízdu obsahuje aplikace také další obrazovky, které zobrazují údaje od vozidla nebo poskytují jiné možnosti komunikace s vozidlem, jako třeba obrazovka `Console`, obsahující příkazový řádek (viz podkapitola 6.5). Takováto aplikace se samozřejmě neobejde bez různých nastavení, které si může

uživatel upravovat podle svých potřeb. Tohle mají na starosti třídy `Configuration` a `ConfigurationFile`, popsané v sekci 6.6. Poslední funkcionalitou aplikace, která bude popsána v podkapitole 6.7 je komunikace s automobilem mimo standardu OBDII. Tato podkapitola bude zahrnovat nalezení a otestování dalších komunikačních rozhraní v testovaném vozidle, ale také odchycení proprietárních zpráv od vozidla a interpretaci jejich významu.

V této kapitole budou postupně popsány základní principy a důležité třídy celé aplikace, některé více, jiné méně detailně. V poslední podkapitole 6.8 bude aplikace otestována v reálném provozu, za účelem odhalení případných chyb, které unikly pozornosti při průběžném testování. Aplikace bude také otestována na dalších mobilních zařízeních a automobilech pro posouzení její kompatibility.

6.1 Připojení ELM327 k diagnostické zásuvce automobilu

Pro testovací účely byl k dispozici automobil Opel Astra H GTC (dále v textu bude většinou označován jednoduše jako testovaný automobil), rok výroby 2006. Je to automobil s benzinovým motorem vyrobený po roce 2001, tedy podle směrnic Evropské unie musí byt vybaven rozhraním OBDII, respektive EOBD (viz 2.4). Pro komunikaci se použije rozhraní s mikrokontrolérem ELM327 připojeným k diagnostické zásuvce automobilu.

Umístění diagnostické zásuvky v testovaném vozidle

Standard předepisuje polohu konektoru v kabině vozu tak, aby byl umístěn pod volantem nebo v dosahu z řidičova místa, ale ne dál jako dvě stopy (přibližně 60 centimetrů) od volantu[12], pokud to dovolují prostorové možnosti ve vozidle. Výjimky jsou samozřejmě možné, ale konektor musí být bezpodmínečně umístěn vně kabiny vozu.

[12] Zdroj: https://en.wikipedia.org/wiki/On-board_diagnostics

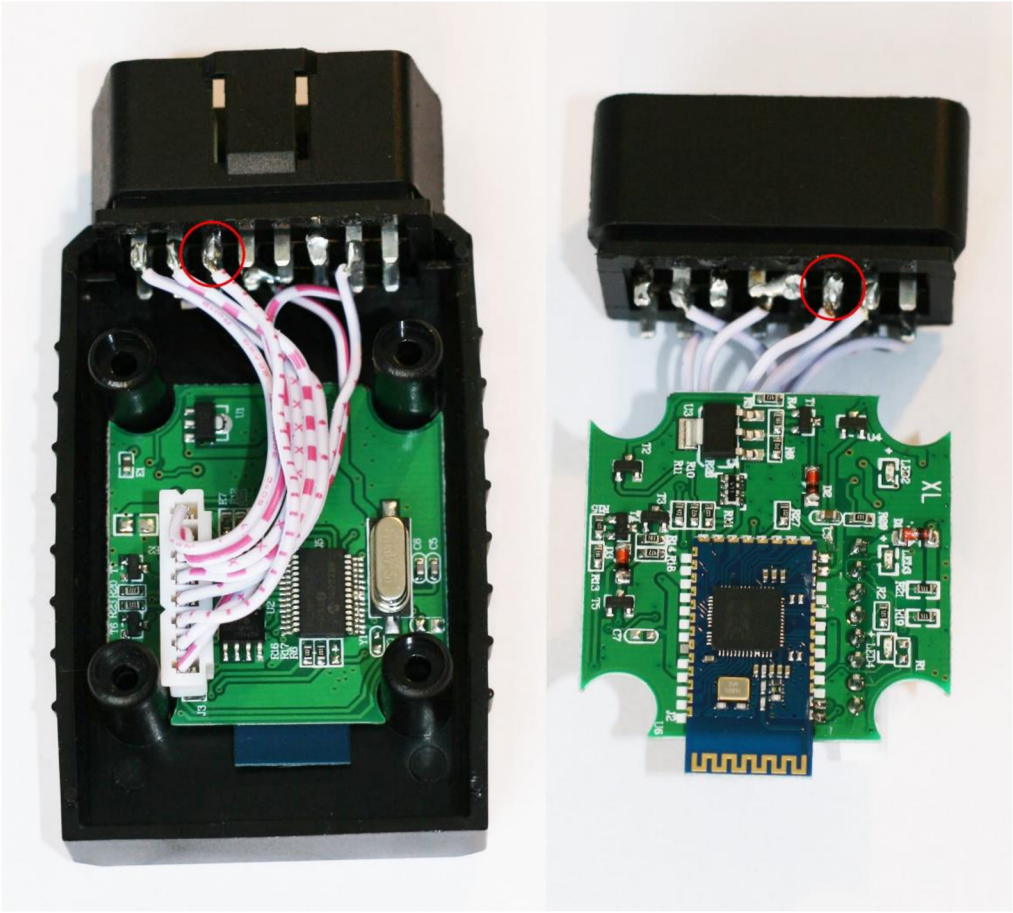
Obrázek 6.2: Umístění diagnostického konektoru ve vozidle Opel Astra H Gtc ve středovém tunelu pod pákou ruční brzdy.

V testovaném automobilu se konektor nachází ve středovém tunelu mezi vodičem a spolujezdcem, ukrytý pod plastovým krytem, pod pákou ruční brzdy, jak je vidět na obrázku 6.2. Pro přístup k němu stačí vytáhnout nahoru ruční brzdu a zatažením za zadní část odstranit plastový kryt. Poté se ukáže 16 pinový konektor (samice), definovaný podle normy SAE J1962.

Zapojení pinů konektoru v rozhraní s mikrokontrolérem ELM327

Jak bylo popsané v kapitole 2.4 a tabulce 2.1Tabulka 2.1, standard definuje pouze některé piny tohoto konektoru. Na obrázku 6.3, po otevření obalu dostupného ELM327, prozkoumání připojení konektoru a porovnání se standardem, je patrné, že jsou implementovány tyto komunikační protokoly:

- Komunikační protokol CAN podle standardu ISO 15765-4, na pinech 6 a 14
- Pak také komunikační protokoly ISO 9141-2 a KWP2000 na pinech 7 a 15
- A komunikační protokol podle SAE J1850 VPW a PWM



Obrázek 6.3: Otevřená krabička rozhraní ELM327 z obou stran. V červeném kroužku vyznačeny piny 6 a 14 představující CAN BUS rozhraní. Modrá destička plošného spoje je rozhraní Bluetooth.

Testování mikrokontroléru ELM327

Pro účely této práce byly k dispozici dvě rozhraní. Obě založené na čipu ELM327. Při testování se ukázalo, že v obou případech se jedná o napodobeniny, což už bylo ale evidentní také v době nákupu podle ceny, za kterou se prodává. Ta byla kolem 300 Kč za jeden kus, kdežto podle údajů z oficiální stránky společnosti Elm electronics, je cena jenom samotného mikrokontroléru ELM327 v nejnovější verzi 2.1 21$. Paradoxně řešení používající originální mikrokontrolér na našem trhu prakticky nejsou k dostání a tak jsem se rozhodl pro použití této neoriginální verze. Jelikož většina lidí pravděpodobně vlastní právě takováto zařízení, bude alespoň možné ověřit případná omezení těchto přístrojů. Dále v textu budou tyto rozhraní označována jako ELM327 i když se nejedná o originální produkty.

Obrázek 6.4: Originální verze mikrokontroléru ELM327 disponuje podle stránek výrobce velkým nápisem ELM327 na "zádech“ pouzdra mikrokontroléru[13].

To, že se nejedná o originál, je patrné z porovnání obrázků 6.3 a 6.4, kde je k vidění originální mikrokontrolér (obrázek 6.4), tak také neoriginální verze používaná v této práci (obrázek 6.3Obrázek 6.3). Originální disponuje velkým nápisem ELM327 na „zádech“ mikrokontroléru, zatím co padělaná verze nápisem PIC18F2480-I/SO, což je podle dokumentace dostupné v literatuře [17] programovatelný mikrokontrolér v ceně kolem čtyř dolarů. Z dokumentu v literatuře [11] také vyplývá, že originální ELM327 je založen na shodném mikrokontroléru PIC18F2480 od firmy Microchip Technology Inc.

Po připojení rozhraní ELM327 a zadání příkazu `AT I`, se mikrokontrolér hlásí jako verze 1.5. Podle dokumentů dostupných ze stránek výrobce[13] existují v současnosti jenom verze 1.3a, 1.4b a 2.1, žádná verze s označení 1.5 tam nefiguruje. To je tedy dalším důkazem, že se nejedná o originální mikrokontrolér. Není možné tedy zjistit, z které verze daný mikrokontrolér vychází. Některé zdroje uvádí, že klony jsou založeny na úplně první verzi ELM327, tedy 1.0, která neobsahovala ochranu proti kopírování kódu a bylo možné ji tedy naklonovat[14]. Nicméně tuto informaci se mi nepovedlo ověřit.

Ať už je verze dostupného mikrokontroléru jakákoliv, ukázalo se to jako nedůležité, protože v průběhu vytváření této práce jsem nenarazil na výraznější omezení ze strany neoriginálního kontroléru a všechno fungovalo podle datasheetu k originální verzi 1.4b.

6.2 Připojení aplikace k ELM327 pomocí Bluetooth rozhraní

K dispozici byly dvě verze rozhraní ELM327. Jedno připojitelné pomocí USB, druhé pomocí Bluetooth. Některé Android zařízení podporují funkci USB host (to znamená, že se mohou k nim připojovat jiná USB zařízení a ony slouží jako hostitel) a k takovým je možné použít také USB verzi. Sice chytrý telefon, který byl k dispozici, disponoval také touto funkcí, ale připojení přes Bluetooth je mnohem elegantnější řešení. USB verze bude použita pro připojení k počítači, při testování některých funkcí.

Pro práci s Bluetooth bylo použito Android Bluetooth API. Veškerou funkcionalitu spojenou s vyhledáváním zařízení vykonává třída `BluetoothActivity` nacházející se uvnitř balíčku `main`. Připojení je iniciované stiskem tlačítka z hlavní obrazovky aplikace. Pokud není Bluetooth aktivní je dotázán uživatel, jestli povolí aplikaci ho spustit. Nejdříve ze všeho je načten seznam již spárovaných zařízení, poté se pomocí metody `startDiscovery()` začnou hledat dostupná

[13] Zdroj: http://elmelectronics.com/obdic.html
[14] Zdroj: https://en.wikipedia.org/wiki/ELM327

zařízení v okolí. Pro naslouchání změnám ve vyhledávání se vytvoří *BroadcastReceiver* (viz kapitola 3.1).

Jakmile skončí vyhledávaní a uživatel vybere požadované zařízení ze seznamu, začne proces párování. Pro tyto účely je uvnitř třídy *ObdiiService* (třída bude podrobněji popsána v další kapitole 6.3) definována třída *ConnectThread*, která vytvoří nové vlákno a provede připojení. Tady je ale potřeba vytvořit Socket pro zabezpečené spojení mezi Android zařízením a ELM327. K tomu je potřeba tzv. UUID, který slouží jako identifikátor Bluetooth zařízení. Podle specifikace Bluetooth[15] je UUID 128 bitové číslo, které vznikne kombinací základního UUID (BASE_UUID) rovné 00000000-0000-1000-8000-00805F9B34FB a identifikátoru služby. V případě ELM327 se jedná o sériový port, který má číslo 0x1101. Kombinací těchto dvou zmiňovaných čísel vznikne potřebné UUID, které je možné vidět na následující ukázce zdrojového kódu z třídy *ConnectThread*.

```
private static final UUID MY_UUID_SECURE =
        UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
// ...
socket = device.createRfcommSocketToServiceRecord(MY_UUID_SECURE);
// ...
socket.connect();
```

6.3 Komunikace s ELM327

ObdiiService z balíčku *main* je primární třída, která zabezpečuje komunikaci s ELM327 zařízením. V souladu s doporučeními pro tvorbu Android aplikací tak, aby nedocházelo k přetěžování hlavního vlákna a zpomalení reakce uživatelského rozhraní, jsou všechny déle trvající operace spouštěny v samostatném vláknu. Mezi takové operace se samozřejmě řadí připojování a komunikace s ELM327 zařízením.

Jak již bylo zmiňováno v předchozí kapitole, při připojování vytvoří třída *ConnectThread* samostatní vlákno, které vytvoří a otevře spojení s ELM327 zařízením. Po připojení vlákno zaniká, ale vytvořené spojení zůstává k dispozici pro další komunikaci.

Ta Aktivita aplikace, která chce získat data od vozidla, si vytvoří seznam Obd příkazů (*CommandList*, bude popsán v následující podkapitole) a do něj vloží jednotlivé příkazy, o které má zájem. Poté zavolá metodu *ContinuouslySendReceive()* z třídy *ObdiiService*, které předá tento seznam spolu s odkazem na speciální metodu, tzv. Handler. Následuje ukázka definice Handler metody s částí kódu pro zpracování odpovědi.

```
private final Handler uiHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
// ...
case Constants.COMMAND_RESPONSE_RECEIVED:
    switch (msg.arg1) {
        case Command.SPEED:
            responseBuf = (byte[]) msg.obj;
            // ... zpracování přijaté zprávy od ELM327
```

[15] Zdroj: https://www.bluetooth.org/en-us/specification/assigned-numbers/service-discovery

Handler je obslužná metoda, která musí být definována v každé třídě požadující nepřetržité přijímání aktuálních dat od vozidla. Je volána pokaždé když komunikační vlákno z *ObdiiService* obdrží odpověď od vozidla a je ji předána přijatá zpráva. Zprávu je potřeba zpracovat a zjistit o jakou informaci se jedná. Na základě této informace ji pak předat dál, aby mohla být zpracována a v případě potřeby zobrazena uživateli nebo použita pro další výpočty.

Pokaždé, když je volána metoda *ContinuouslySendReceive()*, je nejdřív ukončeno jakékoliv jiné vlákno, které by komunikovalo s ELM327. To z toho důvodu, že není dobré, aby se v jeden okamžik dotazovalo víc vláken na data od vozidla, protože by nebylo možné rozlišit pro koho je odpověď určena. Jedinou možností by bylo synchronizovat vlákna tak, aby v jeden okamžik komunikovalo jenom jedno vlákno. Vzhledem k povaze aplikace by však bylo zbytečné podobnou situaci řešit, jelikož vždy může být aktivní jenom jedna obrazovka, která požaduje data od vozidla.

Po ukončení starého vlákna, pokud nějaké běželo, je vytvořeno nové. Jelikož není možné zjistit, v které fázi komunikace bylo staré vlákno ukončeno, na kterém protokolu komunikovalo (jedna část aplikace komunikuje na odlišném protokolu než ostatní) nebo zda nezůstala komunikace „viset“ kvůli chybě, je mikrokontrolér ELM327 pro jistotu pokaždé znovu inicializován. Pomocí příkazu *AT Z* se provede restart celého mikrokontroléru a pomocí *AT SP 0* se nastaví výběr protokolu na automatický. V případě testovaného vozu je vybrán protokol číslo 6 a tedy CAN 11bit a 500Kbps (viz tabulka 2.8). Bylo by tedy možné nastavit přímo tento protokol a zkrátit tím proces inicializace, ale kvůli kompatibilitě s jinými automobily je vždy lepší nechat výběr protokolu na mikrokontrolér.

Před tím než je možné komunikovat s ELM327, musí být pomocí již aktivního Bluetooth spojení otevřené input a output Streamy (je možné přeložit jako vstupní a výstupní datové toky). Do nich je pak možné jednoduše zapisovat pomocí metod *write()*, respektive číst pomocí *read()*, jak je vidět v následující zjednodušené ukázce kódu.

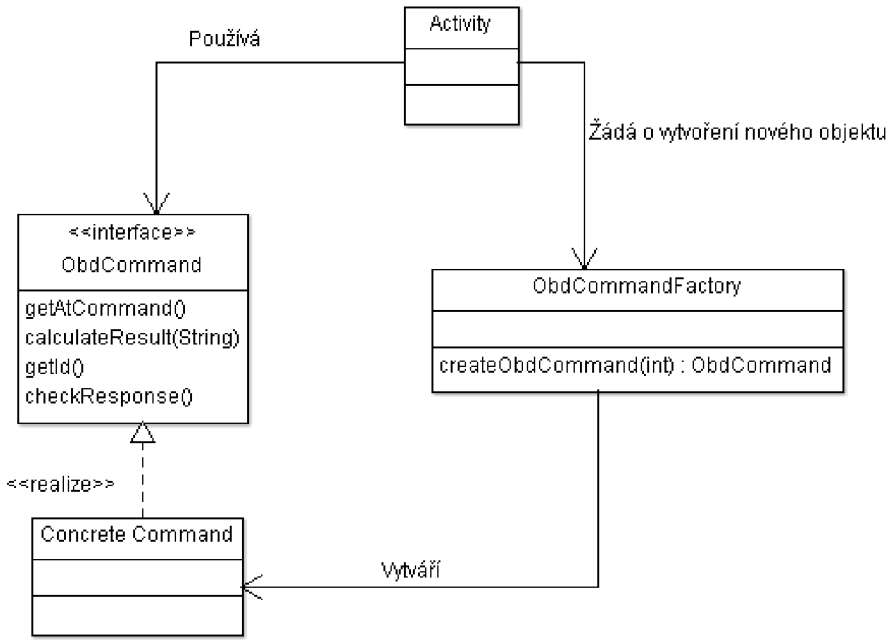
```
inputStream = socket.getInputStream();
outputStream = socket.getOutputStream();
// ...
outputStream.write(buffer);
bytes = inputStream.read(readBuffer);
```

Vlákno poté posílá příkazy a přijímá odpovědi v nekonečné smyčce, v které postupně přechází příkazy z *CommandList*. Po každé iteraci je vložená pauza, které velikost je definována v nastaveních aplikace jako obnovovací frekvence příkazů (command refresh rate) v milisekundách.

ObdCommand

Protože každá obrazovka aplikace potřebuje trochu jinou kombinaci dat od vozidla, nebylo by rozumné, aby každou obsluhovalo vlastní speciální vlákno, ale spíš jedno společné pro všechny, kterému bude možné předat jenom seznam požadovaných příkazů a ono bude Aktivitu na oplátku zásobovat vždy čerstvými daty od vozidla. Takové vlákno se nachází v třídě *ObdiiService* a bylo zmiňováno v předchozí podkapitole. Teď se zaměříme spíš na způsob jak tomuto vláknu předat seznam příkazů.

Je důležité, aby bylo možné s každým příkazem pracovat jednotným způsobem. Za tímto účelem bylo prostudováno několik návrhových vzorů a vybrán *Factory method* (tovární metoda, také označována jako Factory pattern, tedy tovární vzor) jako nejvhodnější. Jedná se o velice jednoduchý přístup, kterého princip je ilustrován na následujícím diagramu.



Obrázek 6.5: Diagram tříd zobrazující použití návrhového vzoru Factory method (Tovární metoda).

Princip metody je následující:

- Aktivita potřebuje příkaz *(ObdCommand)*, ale místo toho, aby si ho vytvořila sama přímo pomocí operátoru *new*, požádá objekt *ObdCommandFactory* o jeho vytvoření s tím, že mu poskytne informace o typu objektu, který potřebuje.
- *ObdCommandFactory* vytvoří novou instanci konkrétního příkazu *(ObdCommand)* a vrátí ji zpátky Aktivitě, ale zabalenou do abstraktní třídy.
- Aktivita používá tento *ObdCommand* jako abstraktní objekt, bez znalosti její konkrétní implementace.

Každá Aktivita, která požaduje data od vozidla, si vytvoří seznam takovýchto příkazů *(obdCommandList)*, do kterého vloží všechny příkazy na data, které potřebuje. Tento seznam pak předá spolu s odkazem na Handler metodu (viz výše) metodě *ContinouslySendReceive*. Tento postup je možné vidět na části kódu z Aktivity *DashBoard*, která zobrazuje data podobné přístrojovému panelu v automobilu.

```
ObdCommandFactory factory = new ObdCommandFactory();
obdCommandList = new ObdCommand[4];
obdCommandList[0] = factory.createObdCommand(Command.SPEED);
obdCommandList[1] = factory.createObdCommand(Command.RPM);
// ...
obdiiService.ContinouslySendReceive(uiHandler, obdCommandList);
```

Od této chvíle Aktivita dostává pravidelně data přes Handler metodu s frekvencí definovanou v nastavení aplikace. Pro ověření správnosti přijatých dat a zpracování odpovědi má každý objekt *ObdCommand* definované metody. Tedy když Aktivita dostane například data rychlosti vozidla,

nejdříve volá metodu *checkResponse()*, aby ověřila správnost přijatých dat a poté z nich za pomoci metody *calculateResult()* dostane samotnou hodnotu rychlosti vozidla.

```
obdCommandList[0].checkResponse(speedValue)
int speed = obdCommandList[0].calculateResult(speedValue);
// ... použít hodnotu rychlosti vozidla
```

Nejjednodušší je vysvětlení právě na rychlosti vozidla. Jako dotaz na rychlost je na ELM327 poslán příkaz „01 0D“. Zpátky přichází odpověď „41 0D XX“. „41 0D“ značí, že se jedná o odpověď na dotaz rychlost vozidla. To je tedy kombinace, kterou hledá metoda *checkResponse* v přijaté odpovědi, aby bylo možné odfiltrovat případné špatné odpovědi. ELM327 pracuje sice na principu dotaz-odpověď, takže na jeden dotaz bychom měli dostat právě jednu odpověď. Ale ne všechny odpovědi jsou správné, proto je lepší pokaždé zkontrolovat, že se jedná o platnou odpověď na náš dotaz. Znaky XX v odpovědi představují hexadecimální hodnotu rychlosti. Zrovna v tomto případě je hodnota rychlosti přímo hodnota XX převedená do desítkové soustavy. Například přijatá odpověď „41 0D 57“ říká, že rychlost vozidla je 87 kilometrů za hodinu (0x57 se rovná 87). U většiny ostatních příkazů se ale hodnoty vypočítávají podle jednoduchých vzorců. Příkazy použity v této práci jsou uvedeny v tabulce Tabulka 2.3, kompletní seznam je možné nalézt v literatuře [12].

6.4 Rádce pro ekonomickou jízdu

Jedna z hlavních obrazovek aplikace je obrazovka zobrazující rádce pro ekonomickou jízdu. Jelikož za standardy OBDII a EOBD stojí snaha o dodržování emisních limitů, má tato obrazovka na základě dat získaných z těchto rozhraní pomoci řidiči se chovat za volantem ekonomicky. Třída, která má na starosti tohoto rádce je *EconomyDriving* z balíčku *displayFunctions*.

Obrazovka ekonomického rádce zobrazuje standardní údaje jako rychlost vozidla a otáčky motoru, ale také teplotu chladicí kapaliny či polohu škrtící klapky (ve většině případů je stejná jako poloha plynového pedálu). To jsou všechno data, která sice poskytují řidiči základní informace o vozidle, nijak zvlášť však nemohou přispět k lepší ekonomičnosti jízdy. Právě proto jsou na tuto obrazovku přidány další údaje, které již mohou mít zásadní vliv na chování řidiče.

První takový údaj je spotřeba paliva, jak aktuální, tak průměrná. Aktuální spotřeba je sice informace, která je definována v standardu OBDII a měla by být dostupná v módu 01 pod PID 5E (viz sekce 2.411 o OBDII rozhraní), ale bohužel v případě testovaného vozidla tomu tak nebylo. To proto, že výrobci automobilů jsou sice povinni implementovat OBDII rozhraní do všech svých automobilů, ale seznam podporovaných parametrů (neboli PIDs) je len doporučený a ne všechny parametry musí být automobilem podporovány (viz 2.4). Bylo potřeba nalézt alternativu, jak zjistit množství aktuálně vstřikovaného paliva.

Pravděpodobně všechny současné automobily mají motor řízený na principu uzavřeného okruhu (closed loop), hlavně kvůli nutnosti dodržování emisí. To znamená, že řídící jednotka motoru (ECM) neustále upravuje poměr vzduchu a paliva ve směsi nasávané do válce motoru na základě zpětné vazby od senzorů (lambda sonda, váha vzduchu a další) tak, aby byl splněn stechiometrický poměr. Ten zjednodušeně řečeno udává ideální poměr mezi benzínem a vzduchem tak, aby při jejich vzájemném hoření vznikalo minimum nežádoucích látek. Pro benzín byla tato hodnota určena na 14,7 dílu vzduchu k 1 dílu benzínu [19]. Vzhledem k tomu, že vzduch stejně jako benzín jsou hodně široké pojmy (jejich složení se mohou lišit), je tato hodnota jenom přibližná. Pro naše účely bude, ale dostačující.

Z přechozího odstavce je patrné, že na 14,7g vzduchu spálíme 1g benzínu. Pokud bychom tedy znali množství vzduchu, které je nasáváno do motoru, mohli bychom vypočítat spotřebu paliva. Množství nasávaného vzduchu do motoru měří tzv. MAF senzor (Mass Air Flow – popsaný v kapitole 2.2). Data z něj jsou přístupná přes OBDII rozhraní v módu 01 pod PID 10. Opět se pravděpodobně může stát, že některé automobily tento údaj nebudou poskytovat, v případě testovaného vozidla tomu tak naštěstí nebylo.

Výslednou spotřebu v litrech za hodinu můžeme dostat následovně. Když si tedy jako MAF označíme množství vzduchu nasávaného do motoru v g/s pak:

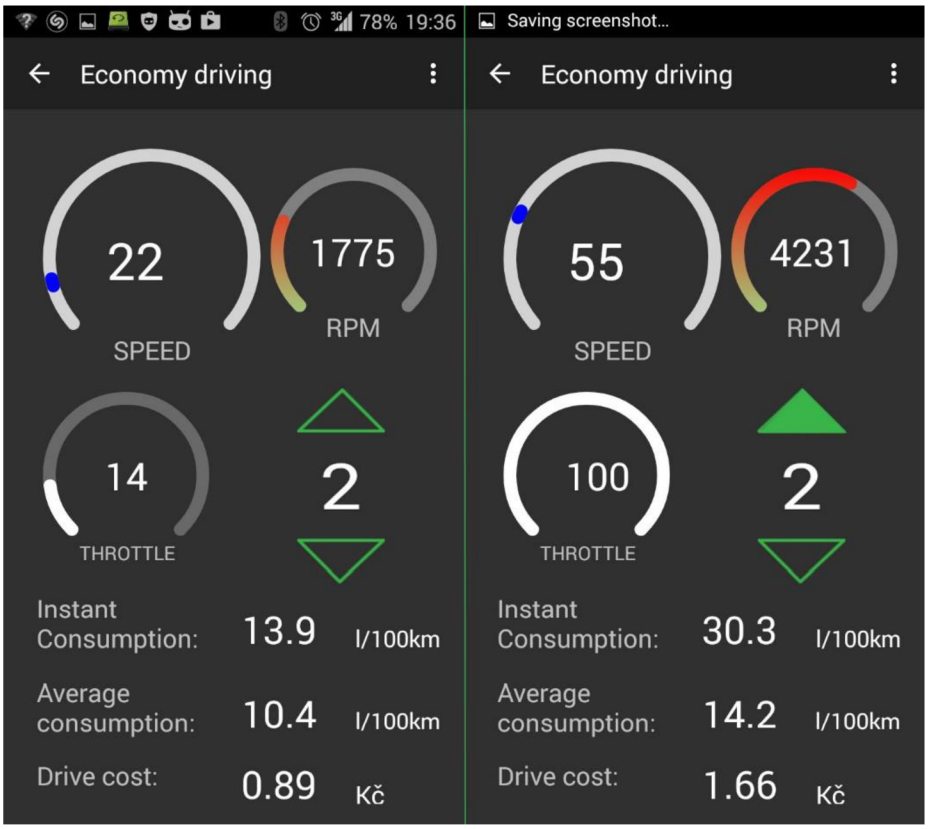
- $\frac{MAF}{14,7}$ = množství spotřebovaného benzínu v g/s (5.1)

- $\frac{MAF}{14,7} * 3,6$ = množství spotřebovaného benzínu v kg/h (5.2)

Teď už víme spotřebu paliva automobilu v kilogramech za hodinu. Hustota benzínu se pohybuje v rozmezí 700-750g/l (opět to ovlivňuje více faktorů, jako složení nebo teplota), proto budeme uvažovat průměrnou hodnotu 725g/l. Z toho víme, že jeden kilogram benzínu má objem 1,379 litru. Dostáváme tedy výsledný vztah:

- $\frac{MAF}{14,7} * 3,6 * 1,379 =$ spotřeba paliva v l/h (5.3)

Stejný princip platí také pro naftové motory jen s tím rozdílem, že stechiometrický poměr nafty je 14,6:1 a hustota přibližně 840 gramů na litr. Proto bylo potřeba do nastavení aplikace přidat možnost volby mezi benzínovým a naftovým motorem.



Obrázek 6.6: Obrazovka rádce pro ekonomickou jízdu během jízdy automobilu. Vpravo s doporučením pro přerazení na vyšší převodový stupeň.

Na obrázku 6.6 je pořízený snímek obrazovky pro ekonomickou jízdu. Na obrazovce je zobrazována, jak aktuální, tak průměrná spotřeba. Zobrazování těchto hodnot bylo inspirováno chováním reálného palubního počítače, kde jsou tyto hodnoty zobrazovány v l/100km. Mimo případy kdy automobil stojí nebo jede příliš pomalu, tehdy se přepne zobrazování aktuální spotřeby do l/h. Tato hranice byla pokusně nastavena na 5km/h. Průměrná spotřeba zůstává stále v l/100km, bez ohledu na rychlost.

Aktivita *EconomyDriving* přijímá data od vozidla jako asynchronní zprávy přes Handler metodu od *ObdiiService* (viz kapitola 6.3). To znamená, že čas, který uplyne mezi přijetím dvou zpráv není známý. Proto, aby bylo možné vypočítat průměrnou spotřebu a projetou vzdálenost, bylo potřeba zavést jednoduchou funkci časovače. Její kód je na následující ukázce.

```
// calculate time
private void setTime() {
    long curr_time = System.currentTimeMillis();
    diff_time = last_time - curr_time;
    last_time = curr_time;
}
```

Díky tomu, že známe přesný čas, po který můžeme považovat poslední data od vozidla za platná, umíme z rychlosti zjistit vzdálenost, kterou vozidlo přešlo a také aktuální a průměrnou spotřebu v litrech na 100 kilometrů.

Na obrazovce rádce se nacházejí ještě další dva údaje, které by měly pomoct řidiči chovat se za volantem ekonomicky. Jedním z nich je cena jízdy (Drive Cost), umístěná v dolní části obrazovky. Je

to v podstatě počítadlo, na kterém naskakuje místo projetých kilometrů suma, kterou zatím jízda stála. Samozřejmě čím ekonomičtější jízda, tím naskakuje počítadlo pomaleji a obráceně. To by měl být hlavní motivační faktor, jelikož mám za to, že peníze hrají nemalou roli u běžných řidičů. Tato funkce počítá s cenou paliva, která se zadává v nastavení aplikace jako první parametr (Fuel Cost).

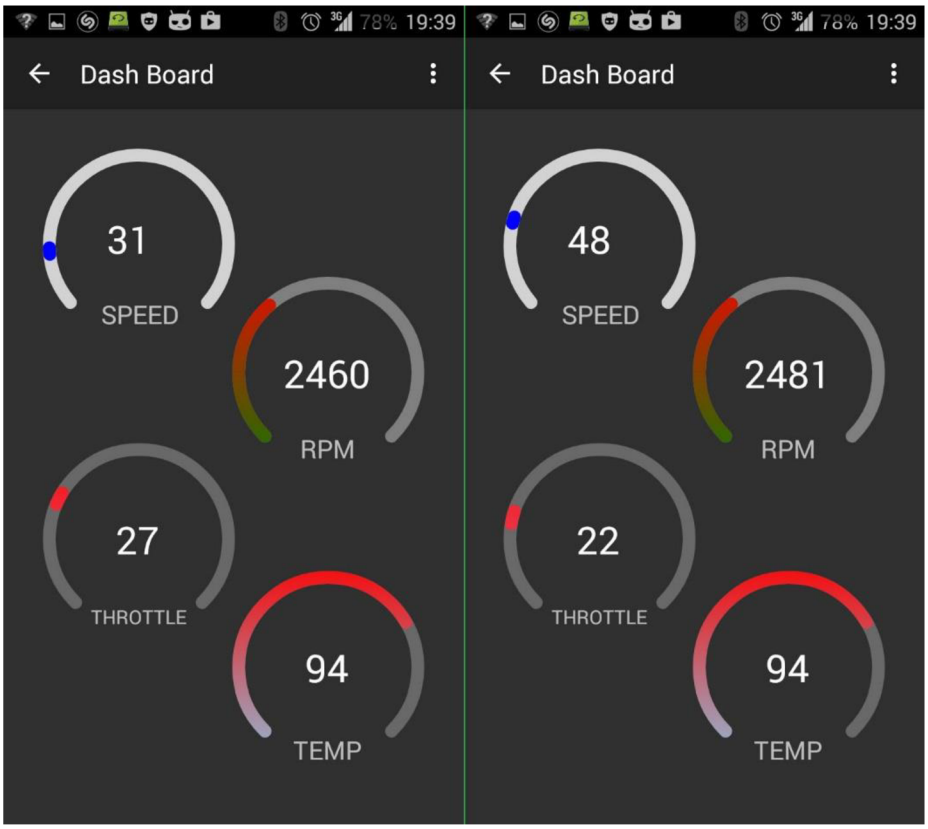
Poslední údaj na této obrazovce je, dalo by se říct, rádce pro řazení převodových stupňů. Nachází se pod ukazatelem otáček motoru. Uprostřed ukazuje aktuálně zařazenou rychlost a navíc jsou tam šipky nahoru a dolů, které radí řidiči, jestli přeřadit na vyšší, nebo nižší převodový stupeň. Pro zjištění zařazeného převodového stupně používá princip, který bude podrobně popsán v kapitole 6.6 v části Gear Calibration. Vypočte si poměr mezi rychlostí a otáčkami motoru a tento porovná s referenčními hodnotami nastavenými pomocí kalibrace převodových stupňů. Pokud to padne do některé z nastavených hodnot, ví přesně, který stupeň je zařazený. Pokud jsou otáčky motoru vyšší než 2500 a zatížení motoru menší než 30 procent, rozsvítí se šipka nahoru, indikující potřebu zařadit vyšší převodový stupeň. Naopak šipka dolů se ukáže, pokud jsou otáčky menší než 1500 a zároveň je zatížení motoru větší než 50 procent. Otáčky motoru samy o sobě neříkají nic o tom, co se děje právě s motorem, proto je do rozhodování přidán také parametr zatížení motoru. Pomůže to odhalit momenty, kdy jsou vyšší otáčky žádoucí, třeba při jízdě do kopce, nebo by bylo naopak zbytečné podřazovat, když motor běží na minimum výkonu. Hodnoty, které aplikace používá pro rozhodování, jsou nastaveny jako výchozí a je možné je upravit podle potřeby v nastaveních aplikace pod `Gear up` a `Gear down Conditions`.

6.5 Další obrazovky aplikace

Obrazovku s rádcem pro ekonomickou jízdu popsanou v předchozí kapitole je možné považovat za hlavní obrazovku. Avšak není jediná, aplikace ještě obsahuje několik dalších, které budou popsány v této podkapitole.

Obrazovka palubní desky (DashBoard)

Obrazovka palubní desky je určena jako základní obrazovka pro zobrazování dat získaných z automobilu přes OBDII rozhraní v reálném čase. Jak už název napovídá, zobrazuje podobné informace, jaké je možné najít na přístrojích palubní desky. Ve většině automobilů jednoduše supluje data, která jsou standardně k dispozici pro řidiče, ale třeba v případě testovaného automobilu, kde chybí na palubní desce ukazatel teploty motoru, doplňuje tento důležitý údaj pro řidiče.



Obrázek 6.7: Obrazovka DashBoard (Palubní deska) během jízdy automobilu.

Z obrázku 6.7 je patrné, že obrazovka zobrazuje čtyři různé údaje, a tedy rychlost vozidla, otáčky motoru, teplotu chladicí kapaliny a polohu škrtící klapky. Z funkčního hlediska Aktivita, která tuto obrazovku obsluhuje, není nic složitého. Na začátku vytváří seznam příkazů `obdCommandList` (viz 6.3) s požadovanými daty od vozidla. Tento předává spolu s ukazatelem na Handler metodu třídě `ObdiiService` (viz 6.3). Ta pro něj nastartuje nové vlákno, které se bude starat o zásobování čerstvými daty od vozidla. Obsahuje tedy ještě zmiňovanou Handler metodu a další čtyři metody, z kterých každá přijímá, zpracovává a zobrazuje na obrazovku jeden konkrétní údaj.

Grafické zobrazení údajů (LiveData)

Aplikace umožňuje zobrazit samostatně každý parametr od vozidla, kterého příkaz byl do aplikace implementován. Rozhodně neobsahuje všechny příkazy použitelné přes OBDII protokol. Účelem nebylo sestrojit další aplikaci, která jenom zobrazuje data, ale poskytnout vyšší funkcionalitu dostupnou přes diagnostický konektor. Proto jsou zde dostupné jenom základní údaje, které byly potřebné v jiných částech aplikace.

Obrázek 6.8: Obrazovka pro výběr funkce zobrazení (vlevo) a obrazovka zobrazující otáčky motoru (vpravo).

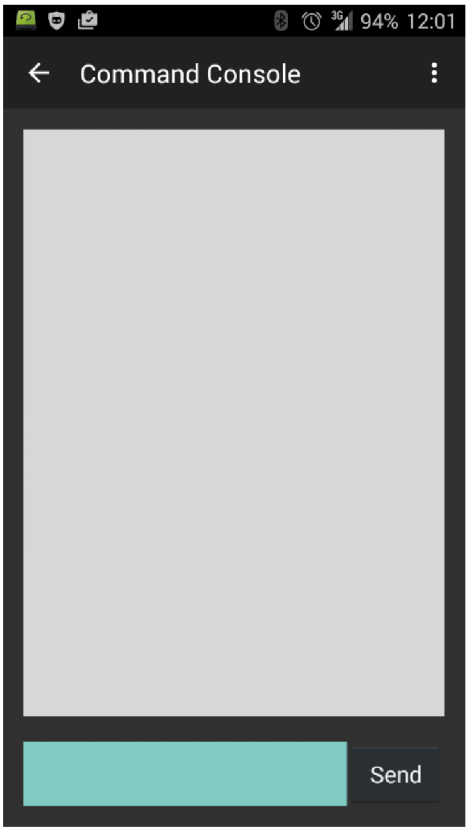
Z hlavní obrazovky přes položku Live Data je dostupná obrazovka, na které je možné vybrat požadovanou funkci k zobrazení (viz obrázek 6.8 vlevo). Seznam poskytuje na výběr všechna data, pro které jsou definovány příkazy ve třídě `Command`. Pro uživatele jsou pro zobrazení na výběr tyto parametry:

- Rychlost vozidla (0 - 255 km/h)
- Otáčky motoru (0 – 16383 rpm)
- Teplota chladicí kapaliny (-40 – 215 °C)
- Poloha škrtící klapky (0 – 100 %)
- Zatížení motoru (0 – 100 %)
- Množství nasávaného vzduchu – MAF (0 – 655,35 g/s)

Seznam funkcí je možné vidět také na obrázku 6.8 vlevo. Poté co si uživatel vybere jednu požadovanou funkci, otevře se nová obrazovka s grafickým ukazatelem hodnoty ve stylu tachometru a číselnou reprezentací této hodnoty uprostřed. Mimo grafické zobrazení je hodnota vypsána také číselně. Jak vypadá obrazovka zobrazující otáčky motoru, je možné vidět na obrázku 6.8 vpravo. Vzhled jednotlivých obrazovek je definován vždy v příslušném xml souboru. O grafické zobrazovaní hodnot se stará třída `CustomGauge`. Je v ní definován vzhled ukazatele, který je možné vidět na obrázku 6.8 vpravo. Vychází ze stejnojmenné knihovny, která byla upravena pro potřeby tohoto projektu.

Obrazovka příkazového řádku (Console)

Dlouho jsem zvažoval, jestli do aplikace zařadit také příkazový řádek, v kterém by bylo možné přímo pomocí příkazů pro ELM327 komunikovat s vozidlem a dostávat zpátky nezpracované odpovědi tak, jak je mikrokontrolér posílá. Jelikož je část aplikace zaměřena na funkcionalitu, která není součástí standardů a byla zkoumána právě z prostředí příkazového řádku počítače, bylo logické zařadit příkazový řádek také do této aplikace. ELM327 navíc může být nakonfigurováno tak, že umožňuje komunikovat na čisté CAN sběrnici a ne jenom pomocí OBDII protokolu (více o této problematice v kapitole 2.5), byla by tedy škoda připravit aplikaci o možnost takového rozšíření.



Obrázek 6.9: Obrazovka příkazového řádku.

Jak je vidět na obrázku 6.9, obrazovka příkazového řádku je velice jednoduchá, tvoří ji jenom tři elementy a to:

- Pole pro zadávání příkazů
- Tlačítko pro odeslání příkazu (Send)
- Okno pro zobrazení odeslaných příkazů a přijatých odpovědí

Pro potřeby příkazového řádku byla v `ObdiiService` definována metoda `sendRecv()`, která nespouští samostatné vlákno pro posílání příkazu jako ostatní metody v ní, ale místo toho pošle jenom jeden požadovaný příkaz a po obdržení odpovědi od mikrokontroléru tuto vrátí volající Aktivitě.

6.6 Možnosti nastavení aplikace

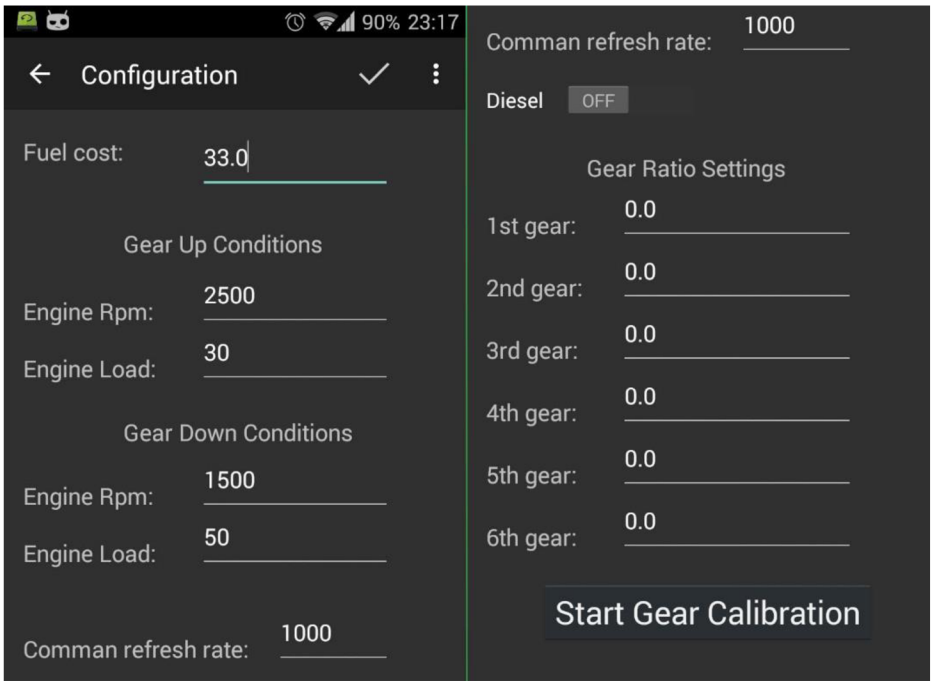
Některé výpočty v aplikaci se provádějí na základě údajů, které jsou, buďto specifické pro dané vozidlo, nebo si je uživatel může nastavit na míru sobě. Zpracování těchto údajů má na starosti třída *Configuration* z balíčku *main*. Pro jednotlivé Aktivity aplikace jsou konfigurační data přístupná přes statické proměnné třídy *ConfigurationFile* z balíčku *globals*. Tento přístup byl zvolen z důvodu, že v původním návrhu se počítalo s tím, že se při startu aplikace načtou tyto data z konfiguračního souboru do zmiňovaných proměnných a při jejich změně bude naopak obsah těchto proměnných uložen do souboru.

V průběhu implementace se ukázalo jako lepší řešení použít funkcionalitu dostupnou v Android API nazývanou *SharedPreferences* (nebo také sdílené vlastnosti) z balíčku *android.content.SharedPreferences*. K těmto *SharedPreferences* je možné přistupovat z kteréhokoliv místa aplikace, ale z důvodu, aby nemusela být přepracována celá aplikace, jednotlivé Aktivity přistupují k proměnným stále přes třídu *ConfigurationFile*. Jenom při spuštění aplikace se načtou proměnné ze *SharedPreferences* a při jejich změně se tam zase uloží. Takto zůstávají změny v nastavení perzistentní mezi jednotlivými spuštěními aplikace. Na následujícím kódu je ukázka načtení z, respektive uložení do *SharedPreferences*.

```
// get values
SharedPreferences settings = getPreferences(0);
ConfigurationFile.fuel_cost = settings.getFloat("fuel_cost", 33.0f);

// edit values
SharedPreferences settings = getPreferences(0);
SharedPreferences.Editor editor = settings.edit();
editor.putFloat("fuel cost", ConfigurationFile.fuel cost);
```

Jak je možné vidět na předchozí ukázce, metoda *settings.getFloat(„fuel_cost", 33.0f)* načítá hodnotu parametru *fuel_cost* a také jako druhý parametr definuje výchozí hodnotu, kterou vrací v případě, že požadovaný parametr zatím nebyl vytvořen. Samozřejmě to nemusí být jenom *getFloat*, ale také *getInt*, *getString* a další. *SharedPreferences* také umožňuje definovat více souborů pro různé skupiny vlastností identifikovaných podle jména. Tady však postačuje použití jenom jednoho základního souboru, není potřeba využívat další.



Obrázek 6.10: Obrazovka nastavení aplikace. Vlevo začátek obrazovky, vpravo pokračování.

Na obrázku 6.10 je obrazovka nastavení aplikace, na které je možné upravit jednotlivá nastavení. Každý parametr byl podrobněji probrán v kapitole zaměřené na funkcionalitu, s kterou je daný parametr svázán. Tady následuje jenom základní vysvětlení jednotlivých parametrů:

- Fuel cost – cena paliva – parametr používán v obrazovce rádce pro ekonomickou jízdu (Economy Driving) na výpočet ceny cesty.
- Gear Up/Down Conditions – parametry pro nastavení podmínek pro rádce volby převodových stupňů v obrazovce rádce pro ekonomickou jízdu. Je možné nastavit otáčky a také zátěž motoru.
- Command refresh rate – parametr, který definuje četnost obnovování dat od vozidla (popsán v kapitole 6.3)
- Gear Ratio Settings – parametry používá obrazovka economy driving. Jedná se o nastavení, pomocí kterého se určuje aktuální převodový stupeň (viz část Gear Calibration).
- Tlačítko Start Gear Calibration – pomocí něj se spouští kalibrační proces právě těchto Gear Ratio Settings.
- Diesel – volba mezi benzínovým a naftovým motorem. Je potřebné pro rozdílný výpočet spotřeby paliva.

Gear Calibration

Část aplikace, která funguje jako rádce pro ekonomickou jízdu, potřebuje pro svoji činnost znát aktuálně zařazený převodový stupeň. Takovýto údaj ale bohužel není součástí standardní sady dat, které je možné od vozidla získat. U některých vozidel se pravděpodobně na sběrnici vyskytuje, ale ve formě proprietárních a pro každé vozidlo specifických dat. Což znamená, že neexistuje způsob, jak obecným způsobem tuto informaci od vozidla získávat přímo.

Proto bylo potřebné nalézt metodu, jak si tento údaj vypočítat z dat, které jsou dostupné od každého vozidla. Jelikož klasická manuální převodovka má pro každý převodový stupeň pevně daný

poměr mezi otáčkami na vstupní a výstupní hřídeli, je poměr mezí rychlostí vozidla a otáčkami motoru pro daný převodový stupeň také pevný. Jak již bylo více krát v průběhu této práce napsáno, patří otáčky motoru a rychlost vozidla mezi údaje definované v standardu, které musí být dostupné v každém moderním automobilu přes diagnostickou zásuvku a rozhraní OBDII, respektive EOBD (viz kapitola 2.4). Ale abychom mohli z těchto dvou údajů zjistit aktuální převodový stupeň, musíme znát poměry mezi rychlostí a otáčkami pro každý z těchto převodových stupňů. Tyto hodnoty je možné zadat v nastavení aplikace v části Gear Ratio Settings, nebo spustit kalibraci, která tyto hodnoty vypočte v průběhu tohoto procesu.

Obrázek 6.11: Obrazovka kalibrace převodových stupňů (Gear Calibration). Vlevo před začetím a vpravo v průběhu probíhajícího procesu kalibrace s aktuálně zařazeným čtvrtým převodovým stupněm (zvýrazněn červenou barvou).

Proces kalibrace funguje tak, že uživatel při stojícím vozidle zmáčkne tlačítko start, tím kalibrace začne a postupně musí projít za jízdy přes všechny převodové stupně, od jedničky po pětku, případně šestku. Algoritmus funguje následujícím způsobem:

- Přibližně každých 200 milisekund se počítá aktuální poměr znovu
- Bylo potřeba odfiltrovat případy, kdy je vytlačená spojka, protože tehdy není poměr mezi rychlostí a otáčkami stálý. Porovná se aktuálně vypočtená hodnota z několika předchozími. Bere do úvahy posledních deset hodnot, a pokud nejsou stejné, současná hodnota se ignoruje.
- V případě, že se rovnají, nejdříve se podívá, jestli se nejedná o stejný převodový stupeň jako posledně. Pokud ne a současný poměr je menší, předpokládá se, že bylo přeřazeno na vyšší převodový stupeň. Ukazatel na aktuálně zařazený převodový stupeň se tedy zvýší o jedna a vypočtený poměr se přiřadí k novému převodovému stupni.

Tímto způsobem jsou zjištěny postupně všechny převodové stupně. Pokud proces skončí na pátém převodovém stupni, má se za to, že automobil šestým nedisponuje. Pro odfiltrování hraničních situací, kdy automobil buď stojí, nebo jede na volnoběh, se berou do úvahy jenom hodnoty, kde je rychlost vozidla větší než 10 kilometrů za hodinu a otáčky motoru nejsou menší než 1200 za minutu. Příklad poměrů převodových stupňů pro testovaný automobil je možné vidět na obrázku 6.11 vpravo, kde je zobrazený probíhající proces kalibrace se zvýrazněným aktuálně zařazeným převodovým stupněm. Pro skončení kalibrace je potřeba stlačit tlačítko Finish. To z důvodu, aby v případě, kdy automobil nedisponuje šestým převodovým stupněm, aplikace nečekala na přeřazení, ale mohla ukončit proces kalibrace.

6.7 Komunikace s automobilem mimo standardu OBDII

Problematika komunikace a diagnostiky v současných automobilech je hodně složitá. Každý výrobce si vyvíjí vlastní systémy, případně ty současné upravuje do podoby, aby nebylo možné používat univerzální přístroje pro komunikaci s nimi. Proto všechny informace obsažené v této podkapitole je možné brát jako sto procentní jenom pro automobily příbuzné tomu, na kterém byly zjištěny a otestovány, a tedy Opel Astra H GTC s rokem výroby 2006 a kódem motoru Z14XEP. Dokonce je možné, že některé informace se budou lišit i u stejného vozidla, ale s jinou motorizací nebo výbavou. Je těžké odhadnout, proč si výrobci zvolili právě tuhle cestu. Možná je to jenom důsledek vývoje, kdy se každý výrobce snaží generaci od generace svoje automobily vylepšovat. Pravděpodobnější však bude snaha výrobců o to, aby servisní dílny musely používat jejich originální diagnostické nástroje, za které si účtují nemalé poplatky a zákazníky donutili chodit do autorizovaných servisů, jelikož některé závady není možné odhalit bez použití příslušných nástrojů. Ať už je to tak, nebo onak, důsledkem je, že neoriginální diagnostické systémy se musí vyvíjet vždy na míru konkrétního vozidla, většinou principem reverzního inženýrství a metodou pokus omyl.

Táto podkapitola se bude věnovat právě komunikaci s automobilem, která není definována v standardu OBDII. Veškeré testování popsané v této podkapitole probíhalo z příkazového řádku počítače (přes terminálový program picocom) připojeného k automobilu pomoci ELM327 ve verzi s USB. Na začátku bude popsán postup nalezení a otestování komunikačních sběrnic v automobilu, které jsou dostupné přes diagnostický konektor. V další části se pokusím nalézt na jedné ze sběrnic zprávy, které posílá modul ovládání rádia na volantu do samotného rádia.

Rozhraní a komunikační protokoly použité v testovaném vozidle

Po připojení ELM327 ke konektoru automobilu a spojení s terminálem na počítači bylo možně zjistit, který protokol použije ELM327 pro komunikaci s vozidlem. V této fázi je to zatím jediná možnost jak zjistit komunikační protokol, jelikož zatím nemáme k dispozici žádnou servisní dokumentaci k vozidlu (později, v dalších kapitolách bude zapojení konektoru diskutováno také na základě schématu elektrického obvodu vozidla). Pro zjištění protokolu se použije sekvence příkazů:

```
> AT Z
ELM327 v1.5
> AT SP 0
OK
> AT DP
AUTO, ISO 15765-4(CAN 11/500)
```

Pomocí příkazu *AT Z* se provede restart zařízení. Není to nutné, jelikož bylo zařízení právě připojeno k vozidlu, ale vždy je lepši ho resetovat na začátku před jakýmkoliv dalším nastavováním, aby toto nebylo ovlivněno případnými předchozími změnami hodnot. Zařízení odpoví hodnotou aktuální verzí použitého mikrokontroléru. Následující příkaz AT SP 0 nastaví komunikační protokol na automatický (viz kapitola 2.5 a popis příkazu SP), tedy zařízení samo vyzkouší všechny protokoly a zvolí odpovídající. Tato možnost se nám hodí pro zjištění komunikačního protokolu, který používá testovaný vůz. Poslední příkaz AT DP (viz kapitola 2.5 a popis příkazu DP) vypíše používaný protokol. Z předchozí sekvence příkazů bylo tedy možné zjistit, že Opel Astra H používá pro OBDII protokol číslo 6 z tabulky 2.8, tedy ISO 15765-4 CA používající 11 bitový identifikátor a běžící na rychlosti 500 kBd.

Toto je jenom protokol, na kterém se komunikuje pomocí standardu OBDII, ale neříká to vůbec nic o případných dalších rozhraních ve vozidle, které jsou implementovány výrobcem nad rámec standardu a které se používají pro komunikaci a diagnostiku ostatních elektronických systémů ve vozidlu. Tomuto problému se bude věnovat další podkapitola.

Zapojení diagnostického konektoru v testovaném vozidlu a používané protokoly

Obrázek 6.12 ukazuje konektor vytažený se středového tunelu a je možné na něm vidět, které piny jsou zapojeny. Je to šestnácti pinový konektor definovaný ve standardu. Vpravo dolů je pin číslo jedna, vlevo nahoře pak pin číslo 16. Jak je možné si odvodit, ale také podle definice pinů z tabulky 2.1, bude červený vodič (pin 16) napájení (V++), dva hnědé vodiče (piny 4 a 5) budou zemnění (ground), bílé vodiče signálové „Low“ a zelené vodiče signálové „High“. Na pinech 6 a 14 je tedy protokol ISO 15765-4, což je CAN. Potvrzuje to také informaci od ELM327, že komunikačním protokolem je CAN s 11 bitovým identifikátorem a rychlostí 500 kBd. Význam zbylých signálových vodičů zatím zůstává neznámý.

Obrázek 6.12: Zapojení diagnostického konektoru v testovaném vozidlu.

Je pravděpodobné (a později se také ukáže jako jisté), že na stejném rozhraní, kde je implementován protokol OBDII budou komunikovat také jiné systémy nebo budou dostupné jiné funkce. Zbylé signálové kabely, ale napovídají, že se nejedná o jedinou komunikační sběrnici ve vozidle, a že některé systémy budou využívat jiných komunikačních cest.

Jednou z možností, jak zjistit význam zbylých signálových vodičů a odhalit tak další komunikační cesty v testovaném vozidlu, bylo získat schéma elektrických obvodů pro Opel Astra H. Samozřejmě nic takového není veřejně dostupné. Naštěstí se mi povedlo získat přístup k softwaru TIS2000, což je kompletní diagnostický software sloužící jako doplněk k diagnostickému zařízení GM Tech2, včetně pracovních postupů a schémat elektrických obvodů pro vozidla skupiny General Motors Company. TIS2000, který byl k dispozici, byl ve verzi 99.0G z prosince 2008. V dnešní době je to již zastaralý systém nahrazený novějším, jelikož ale testované vozidlo je z roku 2006, obsahoval všechny informace o daném vozidle a pro účel této práce byl tedy plně postačující.

Po seznámení se s programem a prostudování schémat elektrických obvodů ve vozidlu, bylo nalezeno také schéma zapojení diagnostického konektoru (kompletní schéma je uvedeno v příloze). Díky tomu bylo možné zjistit význam zbylých pinů v konektoru. Funkce všech pinů je uvedena v tabulce 6.1.

Pin	Zapojení	Pin	Zapojení
1.	LSCAN-H	9.	Nepřipojen
2.	Nepřipojen	10.	Nepřipojen
3.	MSCAN-L	11.	MSCAN-L
4.	Zem karoserie V--	12.	Nepřipojen
5.	Zem signálu V--	13.	Nepřipojen
6.	HSCAN-H	14.	HSCAN-L
7.	TRW	15	Nepřipojen
8.	Nepřipojen	16.	Napájení V++ / Vcc

Tabulka 6.1: Zapojení pinů diagnostického konektoru v Opel Astra H Gtc zjištěných pomocí programu TIS2000.

Víme tedy, že testované vozidlo obsahuje tři komunikační sběrnice a to:

- HSCAN – High Speed CAN – vysokorychlostní CAN sběrnice.
- MSCAN – Middle Speed CAN - CAN sběrnice se střední rychlostí.
- LSCAN – Low Speed CAN – nízko rychlostní sběrnice.

Rychlosti sběrnic v testovaném vozidlu

Jak bylo již napsáno na začátku této podkapitoly, rychlost nejrychlejší sběrnice HSCAN je 500 kBd. Rychlost zbylých dvou sběrnic nám ale zatím zůstává neznámá. Tato informace nebyla součásti ani již zmiňovaného TIS2000, proto bude potřeba tyto sběrnice otestovat a zjistit jejich rychlost. Nejelegantnější řešení by bylo samozřejmě připojit je na osciloskop a podívat se na frekvenci signálů. Nicméně ten k dispozici nebyl, takže bylo nutné použít metodu pokus omyl a zkusit několik nejpravděpodobnějších nastavení a podívat se jestli data přicházející po sběrnici nejsou nesmyslná. Pokud dostaneme data, která vypadají jako CAN rámec, pak je nastavená rychlost pravděpodobně správná.

Jak jejich název napovídá, jsou to sběrnice využívající CAN protokol. Jelikož ta nejrychlejší, které parametry jsou známé, používá 11 bitový identifikátor, budeme předpokládat, že zbylé dvě používají také 11 bitový identifikátor. ELM327 počítá s CAN rozhraním jenom na pinech 6 a 14, pro otestování MSCAN bylo proto potřebné tento konektor přepájet na piny 3 a 11. LSCAN je u skupiny General Motors nazýván také jako SWCAN (Single Wire CAN – tedy jednovodičové CAN rozhraní). Jak je možné vidět také podle zapojení pinů v diagnostické zásuvce testovaného vozidla, z tabulky 6.1, LSCAN používá jenom jeden signálový vodič (zbylé dvě sběrnice používají vždy kroucenou dvoulinku, kvůli lepší odolnosti proti rušení). ELM327 umí jenom CAN se dvěma signálovými vodiči. Pokusem bylo ale zjištěno, že když se zbylý vodič připojí na zemnění v diagnostickém konektoru automobilu (pin 5), pak je ELM327 schopno bez problému komunikovat také na SWCAN.

Když bylo ELM327 připojeno na požadovanou sběrnici, bylo možné začít se zjišťováním přenosové rychlosti na daných sběrnicích. Bylo potřebné udělat několik změn v nastavení mikrokontroléru pomocí následujících příkazů:

```
> AT Z            // provede reset rozhraní
ELM327 1.5
> AT PC           // deaktivuje otevřený protokol
OK
> AT PB C0 0F     // nastaví parametry protokolu B
OK
> AT SP B         // aktivuje protokol B
OK
```

Na začátku se pro jistotu provede restart mikrokontroléru. Protože po restartu může ELM327 automaticky otevřít komunikační protokol (pokud je nastaven protokol na 0 - automatický), pomocí AT PC se provede deaktivace jakéhokoliv otevřeného protokolu.

A teď k samotnému nastavení. ELM327 podporuje dva uživatelsky definované protokoly označené jako B a C. Pomocí AT PB se provede nastavení prvního z nich. Hodnota C0 nastavuje CAN identifikátor na délku 11 bitů (protože C0 je binárně 1100 0000). Rychlost komunikace je definovaná jako 500/XX kBd, kde XX je poslední hodnota z příkazu AT PB. Tady je nastavená na

hodnotu 33,33 kBd, tedy 500/15 (0F je dekadicky 15). Po provedení popisovaných nastavení se musí už jenom protokol B nastavit jako aktivní.

Tímto způsobem je ELM327 připraveno komunikovat na připojené sběrnici na CAN protokolu o rychlosti 33 kBd s 11 bitovým identifikátorem. Teď je potřebné otestovat správnost nastavení. Následuje ukázka výpisu přijatých zpráv při monitorování CAN sběrnice.

```
> AT MA           // aktivuje monitorování
450 46 07 05 00               // zachycené CAN zprávy
501 61 00 42 16 08 80 20 00
206 08 93 01
506 16 00 43 0B 40 80 01 00
501 61 00 42 16 01 80 20 00
188 46 00 00 16 01 80
```

AT MA přepne ELM327 do monitorovacího módu, kdy zobrazuje veškerou komunikaci, která se objeví na sběrnici. Pokud jsou všechna nastavení správná, měla by data vypadat podobně těm následujícím hned za příkazem AT MA v předchozí ukázce výpisu přijatých zpráv. V případě, že se vedle dat často objevuje „CAN ERROR“, je buď špatně nastavená rychlost nebo počet bitů identifikátoru a bude pravděpodobně potřeba zkusit jiná nastavení.

Stejným postupem bylo zjištěno, že MSCAN běží na rychlosti 100 kBd s 11 bitovým CAN identifikátorem a LSCAN na rychlosti 33,33 kBd a také 11 bitovým identifikátorem. V tabulce 6.2 je shrnutí rozhraní, které jsou v automobilu Opel Astra H přístupná přes diagnostickou zásuvku, včetně protokolů, pomocí kterých je možné na nich komunikovat.

Piny	Název	Protokol	Popis
1, (5)	LSCAN	ISO 15765-4, 11bit, 33,33 kBd	Pomalá sběrnice pro časově nekritické operace
3, 11	MSCAN	ISO 15765-4, 11bit, 100 kBd	Sběrnice se střední rychlostí, většinou pro rádio a infotainment
6, 14	HSCAN	ISO 15765-4, 11bit, 500 kBd	Rychlá sběrnice pro časově kritické operace

Tabulka 6.2: Komunikační rozhraní dostupná přes diagnostickou zásuvku v automobilu Opel Astra H Gtc.

Odchytávání proprietárních zpráv na sběrnici vozidla a interpretace jejich významu

Jelikož je ELM327 od výroby připraveno komunikovat s CAN na pinech 6 a 14, bude jako první otestována sběrnice, které se říká HSCAN (viz výše). Aby bylo možné sledovat provoz na sběrnici je potřebné ELM327 přepnout do monitorovacího módu. Abychom však dostávali relevantní informace, je potřeba ještě provést několik nastavení. K tomu se použije sekvence příkazů:

```
> AT SP 6   // nastaví protokol na CAN 11bit/500kBd
> AT L1     // zapne odřádkování
> AT H1     // zapne zobrazení hlavičky
> AT S1     // zapne zobrazení mezer
> AT AL     // dovolí zobrazení dlouhých zpráv (>7 bajtů)
> AT MA     // přepne zařízení do monitorovacího módu
```

Po každém příkazu odpoví mikrokontrolér „OK“, na znak přijetí příkazu. Pro úsporu místa je toto ale z předchozí ukázky kódu vynecháno. Význam jednotlivých příkazů je uveden přímo v ukázce. Důležité je zapnout zobrazování hlavičky zprávy, aby bylo možné vidět celou zprávu, a ne jenom její datovou část. Zbylé příkazy pak slouží pro nastavení lepší čitelnosti přijatých zpráv. Poslední příkaz přepne ELM327 do monitorovacího módu.

V tento moment se ukázala slabina ELM327, protože používá RS232 rozhraní nastavené na 38,4 kBd a sběrnice, na které se snaží naslouchat, běží na rychlosti 500kBd. Rychle dochází k zaplnění vyrovnávací paměti, do které ELM327 ukládá příchozí zprávy. Před tím než dojde k zaplnění, stihne ELM327 přeposlat kolem deseti zpráv, což je příliš málo na to, aby bylo možné zjistit cokoliv o komunikaci na této sběrnici.

Proto nezbývalo jiné, než zkusit pomalejší sběrnici a tedy MSCAN. K tomu bylo potřeba přepájet konektor tak, aby ELM327 komunikovalo s CAN na pinech 3 a 11. K nastavení ELM327 se použije stejný postup jako výše, jenom komunikační protokol se musí nastavit na 11bit a 100 kBd. Tady už zaznamenáme úspěch a od ELM327 dostáváme dlouhou sekvenci CAN zpráv, což nám umožňuje, pokusit se na této sběrnici o nalezení určitých zpráv.

Obrázek 6.13: Ovládací tlačítka rádia (vpravo) a palubního počítače (vlevo) na volantu testovaného automobilu.

Tady jsem měl za cíl nalézt zprávy, které posílá modul ovládání rádia na volantu do samotné jednotky rádia. Poté by bylo možné poslat stejné zprávy zpátky na sběrnici a ovládat tím rádio přímo z aplikace v mobilním zařízení. K nalezení těchto zpráv byl použit následující postup:

- Na začátku bylo potřeba odchytit dostatečné množství zpráv ze sběrnice a vytvořit si tak základní sadu zpráv, které se na sběrnici vyskytují za normálních podmínek.
- Poté udělat nové odchytávání, ale tento krát během něj stlačit několikrát jedno konkrétní tlačítko na volantu (třeba 5x).
- Porovnat tyto dvě sady zpráv a nalézt takové zprávy, které se vyskytují v druhé sadě, ale v první nikoliv. Takových zpráv bude víc, proto je potřeba vybrat tu, která se objevila právě tolik krát, kolik krát bylo stlačeno tlačítko na volantu.
- Takto byla nalezena zpráva tlačítka pro zvýšení hlasitosti, která je: `206 08 93 01`.

Ke zpracování zpráv takovýmto způsobem postačuje obyčejný Microsoft Excel ve Windows, nebo OpenOffice Calc v Linuxu. Stejným způsobem byly nalezeny zprávy pro všechny tlačítka na volantu. Jejich seznam a popis je uveden v tabulce 6.3.

Tlačítko	Popis	Zpráva	
1	Šipka nahoru – ladění stanic	Stlačit	206 01 91 00
		Uvolnit	206 00 91 00
2	Šipka dolů – ladění stanic	Stlačit	206 01 92 00
		Uvolnit	206 00 92 00
3	Kolečko nahoru – přidat hlasitost	206 08 93 01	
3	Kolečko dolů – ubrat hlasitost	206 08 93 FF	
4	Handsfree – přijmout hovor	Stlačit	206 01 81 01
		Uvolnit	206 00 81 01
5	Handsfree – ukončit hovor	Stlačit	206 01 82 01
		Uvolnit	206 00 82 01
6	Kolečko nahoru – volba palubní počítač	206 08 83 FF	
6	Kolečko dolů – volba palubní počítač	206 08 83 01	
6	Stlačit kolečko – potvrdit volbu	Stlačit	206 01 84 00
		Uvolnit	206 00 84 00

Tabulka 6.3: Zjištěné zprávy posílané z modulu ovládaní rádia na volantu na sběrnici MSCAN při stlačení konkrétního tlačítka v automobilu Opel Astra H Gtc.

Zprávy, které je možné vidět v tabulce 6.3 nejsou kompletní CAN zprávy tak, jak byly popsané v kapitole 2.3. Mikrokontrolér ELM327 zpracovává tyto zprávy za uživatele. Z přijatých zpráv odstraní některé položky před tím, než je zobrazí uživateli a při odeslání naopak doplní chybějící. K příkladu v zprávě `206 08 93 01`, která ovládá hlasitost rádia, první tří číslice, tedy `206` představují tzv. CAN Id, nebo také 11 bitový identifikátor. Zbylé tři dvojice čísel (tří bajty) jsou samotná data. Zbytek zprávy je pro přehlednost odstraněn mikrokontrolérem.

Obrazovka ovládající rádio

Na základě zjištění z předchozí podkapitoly, byla v aplikaci vytvořena další obrazovka, která umožňuje ovládat rádio, tedy přesněji přidávat a ubírat hlasitost a přepínat stanice, případně skladby při poslechu hudby z CD. Napodobuje funkci tlačítek 1, 2 a 3 z obrázku 6.13 nebo tabulky 6.3. Obrazovka obsahuje čtyři tlačítka, z kterých každé má na starosti odeslání příslušné zprávy. Plus a mínus představuje volbu hlasitosti, šipka nahoru a šipka dolů pak přepínání nebo ladění stanic.

Obrázek 6.14: Obrazovka pro ovládání rádia automobilu.

Pro posílání těchto zpráv je potřeba ELM327 nakonfigurovat odlišně než bylo popsáno v předchozích kapitolách při posílání OBDII zpráv. Z toho důvodu byly do třídy `ObdiiService` přidány některé metody, které mají na starosti inicializaci spojení s ELM327 a také samotné posílání zpráv.

Základním nedostatkem tohoto řešení je, že modul volantu a rádio jsou napojené na jiné sběrnici, než na které pracuje rozhraní OBDII. To znamená, že při použití klasického rozhraní s ELM327 mikrokontrolérem není možné v jeden okamžik komunikovat přes OBDII a získávat tedy data o vozidle a zároveň ovládat rádio přes druhou sběrnici.

Samotná obrazovka je skutečně velmi jednoduchá a ani nebylo v plánu, aby ovládala všechny funkce, které vestavěné rádio v automobilu poskytuje. Je to spíš ukázka toho, co všechno je možné do aplikací pro automobily implementovat a také směr, kterým by se mohla tato aplikace dále vyvíjet.

6.8 Testování

Jednotlivé komponenty aplikace byly testovány průběžně během jejich vzniku. Většina drobných chyb byla tedy vychytána už v průběhu vytváření aplikace. Testovalo se ale hlavně v stojícím automobilu. Na závěr však bylo potřeba otestovat aplikaci jako celek, aby se ověřilo jak spolu jednotlivé komponenty spolupracují, ale hlavně zkusit vše za jízdy a opravit případné nedokonalosti. Následující část práce se tedy věnuje popisu testování aplikace v reálných podmínkách.

Testování kompatibility

Účelem testování kompatibility bylo ověření správné funkčnosti aplikace na různých zařízeních, různých automobilech, různých ELM327 rozhraních. Aplikace byla primárně vyvíjena a celou dobu testována na chytrém telefonu Samsung Galaxy S3, pro které bylo také uživatelské rozhraní optimalizováno. Pro účely závěrečného testování byly ale zapůjčeny další dva chytré telefony, které uvádí tabulka 6.4. Žádný z nich ale neobsahoval starší verzi Androidu, zpětnou kompatibilitu tedy nebylo možné ověřit. Díky použití minimálního API level 15 by ale aplikace měla fungovat bez problémů od verze Androidu 4.0.3. Na větším displeji S4 se grafické uživatelské rozhraní zobrazovalo naprosto bez problémů. Galaxy S má ale výrazně menší rozlišení a velikost displeje, takže se některé prvky malinko překrývaly, ale i přesto byly všechny údaje bez problémů čitelné.

Název zařízení	Verze OS	Uhlopříčka	Rozlišení displeje
Samsung Galaxy S3	4.3.1 (Jelly Bean)	4.7 palce	720 x 1280
Samsung Galaxy S	4.4.4 (Kit Kat)	4.0 palce	480 x 800
Samsung Galaxy S4	5.0 (Lollipop)	5.0 palce	1080 x 1920

Tabulka 6.4: Seznam mobilních zařízeni dostupných pro účely testování.

Jako rozhraní pro komunikaci mezi automobilem a mobilním zařízením se používalo rozhraní postavené na mikrokontroléru ELM327. Jak bylo popsané v kapitole 6.1 toto rozhraní neobsahovalo originální mikrokontrolér od výrobce Elm Electronics, ale jeho kopii. Během vytváření aplikace, ani v průběhu závěrečného testování však nebyly odhaleny žádné nedostatky, které by tento neoriginální mikrokontrolér způsoboval. Pro porovnání nebyl k dispozici jiný mikrokontrolér s Bluetooth rozhraním, aplikace by ale měla fungovat na jakémkoliv funkčním rozhraní založeném na mikrokontroléru ELM327.

Pro testování funkčnosti aplikace byl k dispozici automobil Opel Astra H Gtc. Na něm byla aplikace průběžně testována a na něm proběhlo také závěrečné testování. OBDII je však standard, který jsou povinni implementovat do svých automobilů všichni současní výrobci. I když i v rámci tohoto standardu existují odchylky, v aplikaci jsou využívána jenom data, která by měla být dostupná v drtivé většině moderních automobilů. Pro alespoň částečné ověření tohoto tvrzení byla aplikace otestována ještě na dvou dalších automobilech, které se povedlo zapůjčit pro tyto účely. Prvním byla Toyota Auris s rokem výroby 2008 a druhým Opel Vectra z roku 2002. Jak zařízení s ELM327, tak samotná aplikace fungovali na těchto automobilech bez problémů. Samozřejmě jenom OBDII část aplikace. Ovládání rádia, které komunikuje s automobilem mimo standardu podle očekávání nefungovalo ani na jednom z nich. Ačkoliv u Oplu Vectra jsem doufal, že by fungovat mohlo, jelikož se jedná o stejného výrobce, opak byl ale pravdou.

Testování funkčnosti

Předmětem funkčního testování bylo především ověřit funkci všech operací v aplikaci. To znamená připojení k ELM327 přes Bluetooth, inicializaci spojení a samotnou komunikaci s řídicími systémy ve vozidlu, zobrazování jednotlivých údajů od vozidla až po ovládání vestavěného rádia. Cílem bylo objevit chyby, které se neprojevily během průběžného testování v průběhu vývoje aplikace. Při testovacím provozu bylo zjištěno několik nekorektně ošetřených výjimek, které se projevovaly spíše sporadicky, avšak způsobovaly selhání aplikace.

Bylo zjištěno, že aplikace nebyla schopna znovu navázat spojení s automobilem po přechodu na jinou obrazovku. To bylo způsobené zpožděním mikrokontroléru, který v době kdy ještě prováděl operace pro předchozí obrazovku, nebyl schopen zpracovat příkazy pro restartování rozhraní. Kvůli tomu bylo přidáno půl vteřinové zpoždění mezi ukončení starého a inicializaci nového spojení.

Další problém se objevil při kalibraci převodových stupňů, kdy zjištěné poměry mezi rychlostí a otáčkami motoru se mezi jednotlivými výpočty hodně měnily. To prakticky znemožňovalo tyto hodnoty vůbec vypočítat. Bylo zjištěno, že přijetí jednoho údaje od vozidla trvá po odeslání příkazu přibližně 300 až 500 milisekund. To způsobovalo hlavně při zrychlování na prvním převodovém stupni, že mezi přijetím rychlosti vozidla a otáčkami motoru se první hodnota natolik změnila, že vypočítaná hodnota nebyla použitelná. Z toho důvodu bylo optimalizováno přijímání dat od vozidla a současná doba pro přijetí jedné hodnoty je kolem 80 milisekund, což výrazně zpřesnilo výpočet, i když mírné odchylky se stále vyskytují.

Dále bylo opraveno pár drobných chyb při výpočtu spotřeby a některé malé chyby při zobrazování hodnot.

7 Závěr

Cílem této práce bylo vytvořit aplikaci pro systém Android, která by umožňovala získávat a vyhodnocovat data z automobilu přes rozhraní OBDII. Což bylo splněno a výsledkem je aplikace, která k tomuto účelu využívá mikrokontrolér ELM327.

Na začátku práce byla prostudována literatura popisující měřené veličiny a senzory v motorových vozidlech a také řídicí systémy, které využívají data z těchto senzorů. Dále také literatura o tvorbě aplikací pro platformu Android a detaily o rozhraní a standardu OBDII.

Byla vytvořena koncepce aplikace pro komunikaci s řídícími jednotkami v automobilu přes rozhraní OBDII. Jako prostředník v této komunikaci byl vybrán mikrokontrolér ELM327, s pomocí kterého komunikuje aplikace s automobilem přes Bluetooth rozhraní.

Na základě návrhu byla vytvořena aplikace, která dokáže přijímat data od vozidla a prezentovat je uživateli, buď jednotlivě, nebo ve formě, která má sloužit jako pomocník řidiče pro ekonomickou jízdu. Vytvořená aplikace také demonstruje možnosti komunikace mimo standardu OBDII, v tomto případě ovládáním vestavěného rádia posíláním zpráv na sběrnici vozidla. Tato funkce je dostupná jenom pro testovaný vůz Opel Astra H. Aby bylo možné demonstrovat celou funkcionalitu aplikace, bylo vytvořeno uživatelsky přívětivé grafické rozhraní, které graficky zobrazuje dostupné informace získané z automobilu.

Po stránce softwarové by bylo možné v práci pokračovat tak, že by se část aplikace, která pracuje na standardu OBDII rozšířila, aby poskytovala uživateli všechny informace dostupné od automobilu na základě tohoto standardu. Do části, která v této aplikaci ovládá rádio automobilu Opel Astra H, by se přidala podpora také pro další automobily, případně bych si uměl představit, že by se aplikace rozdělila na dvě samostatní a každá by pokrývala jednu tuto část původní aplikace. Dále by bylo možné nalézt, případně vyvinou alternativu za ELM327 tak, aby byla umožněna komunikace na více rozhraních najednou, což by otevřelo možnosti pro další vylepšení této aplikace. Podle časových možností bych rád v dohledné době aplikaci připravil tak, aby ji bylo možné zveřejnit v Google Play.

Vždy jsem měl vztah k automobilům i jinak než jenom z pohledu řidiče, proto mě práce na tomto projektu bavila. Navíc jsem se v průběhu vytváření této práce seznámil s mnoha technologiemi, které se v automobilovém průmyslu využívají a uvědomil si, jak velké jsou možnosti využití informačních technologií v tomto odvětví.

Literatura

[1] Barry Hollembeak: Advanced automotive electronic systems. Delmar publishing, 2010, ISBN: 978-1111038144

[2] National Instrments: ECU Designing and Testing using National Instruments Products. National Instruments, 2009. Dostupné na URL: <http://www.ni.com/white-paper/3312/en/pdf>

[3] Qianfan Xin: Diesel engine system design. Philadelphia: Woodhead Publishing, 2013. ISBN: 978-0-85709-083-6

[4] Dietrich Kuhlgatz: Journal of Bosch History, Supplement 2. Stuttgart, Robert Bosch GmbH, 2010. Dostupné na URL: <http://www.bosch.com/content2/publication_forms/en/downloads/Sonderheft_2_automotive_en.pdf>

[5] Marco Di Natale, Haibo Zeng, Paolo Giusto, Arkadeb Ghosal: Understanding and Using Controller Area Network Communication Protocol – Theory and practice. New York: Springer Science, 2012, ISBN: 978-1-4614-0314-2

[6] Martin Beran: Datové sběrnice CAN. Ústav automobilového a dopravního inženýrství, Vysoké Učení Technické v Brně. Dostupné na URL: <http://www.iae.fme.vutbr.cz/userfiles/beran/files/Datov%C3%A1%20sb%C4%9Brnice%20CAN.pdf>

[7] Steve Corrigan: Introduction to the Controller Area Network (CAN) – Application report. Dallas: Texas Instruments, 2008. Dostupné na URL: <http://www.ti.com/lit/an/sloa101a/sloa101a.pdf>

[8] Keith McCord: Automotive Diagnostic Systems – Understanding OBD-I & OBD-II. North Branch: CarTech Inc., 2011, ISBN: 978-1-934709-06-1

[9] Murat Aydin: Android 4: New features for Application Development. Birmingham: Packt Publishing Ltd., 2012, ISBN: 978-1-84951-952-6

[10] Toyota Motor: Air Flow Sensors. Toyota Motor Sales, USA, Inc. Dostupné na URL: <https://www.oscium.com/sites/default/files/MAF_autoshop101.pdf>

[11] Elm Electronics: ELM327DSH – Datasheet k mikrokontroléru ELM327 – OBD to RS232 Interpreter. Dostupné na URL: <http://elmelectronics.com/DSheets/ELM327DSH.pdf>

[12] Al Santini: OBD II: Functions, Monitors and Diagnostic Techniques. Clifton Park: Delmar, Cengage Learning, 2011, ISBN: 978-1-428-39000-3

[13] V. A. W. Hillier, Peter Coombes: Hillier's Fundamentals of Motor Vehicle Technology, Fifth edition. Cheltenham: Nelson Thornes Ltd., 2004, ISBN: 0 7487 8082 3

[14] European Parliament: Directive 98/69/EC of the European Parliament and of the Council of 13 October 1998 relating to measures to be taken against air pollution by emissions from motor vehicles and amending Council Directive 70/220/EEC [online]. 1998 [cit. 2015-06-05]. Dostupné na URL: <http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31998L0069:EN:HTML>

[15] Simon Finch, Bohumil Hnilicka, Hector Sindano: EURO6 Light-Duty Vehicle OBD project, Evaluation and Assessment of Proposed EOBD Emission Thresholds. Cambridge: Ricardo UK Limited, 2010. Dostupné na URL: <https://www.ricardo.com/Global/IA/Media/Q51760%20RD_10_320201_4_Euro6_OBD_Report.pdf>

[16] Eugen Mayer: Serial Bus Systems in the Automobile. Stuttgart: Vector Informatik, 2006. Dostupné na URL: <https://elearning.vector.com/portal/medien/cmc/press/PTR/SerialBusSystems_Part2_ElektronikAutomotive_200612_PressArticle_EN.pdf>

[17] Microchip: PIC18F2480/2580/4480/4580 - Data Sheet, DS39637D. Microchip technology Inc., 2009. Dostupné na URL: <http://www.microchip.com/wwwproducts/Devices.aspx?product=PIC18F2480>

[18] Chris Trout: Android still the dominant mobile OS with 1 bilion active users [online]. Engadget, 2014, aktualizováno 2014-06-25 [cit. 2015-06-14]. Dostupné na URL: <http://www.engadget.com/2014/06/25/google-io-2014-by-the-numbers/>

[19] R. K. Rajput: A Textbook of Internal Combustion Engines, Second edition. Laxmi Publications, 2007, ISBN: 81-318-0066-0

[20] Android Developers: Managing the Activity Lifecycle – Starting an Activity [online]. 2015 [cit. 2015-06-17]. Dostupné na URL: <http://developer.android.com/training/basics/activity-lifecycle/starting.html>

[21] Reto Maier: Professional Android 4 Application Development. Indianapolis: John Wiley & Sons, Inc., 2012, ISBN: 978-1-118-10227-5

[22] Belén Cruz Zapata: Android Studio Application Development. Birmingham: Packt Publishing Ltd., 2013, ISBN 978-1-78328-527-3

[23] Jan Koudelka: Nástroj pro správu projektů s využitím tabletů platformy Android [Diplomová práce]. Brno: Vysoké Učení Technické v Brně, Fakulta Informačních Technologií, 2014. Dostupné na URL: <http://www.fit.vutbr.cz/study/DP/DP.php?id=16847&file=t>

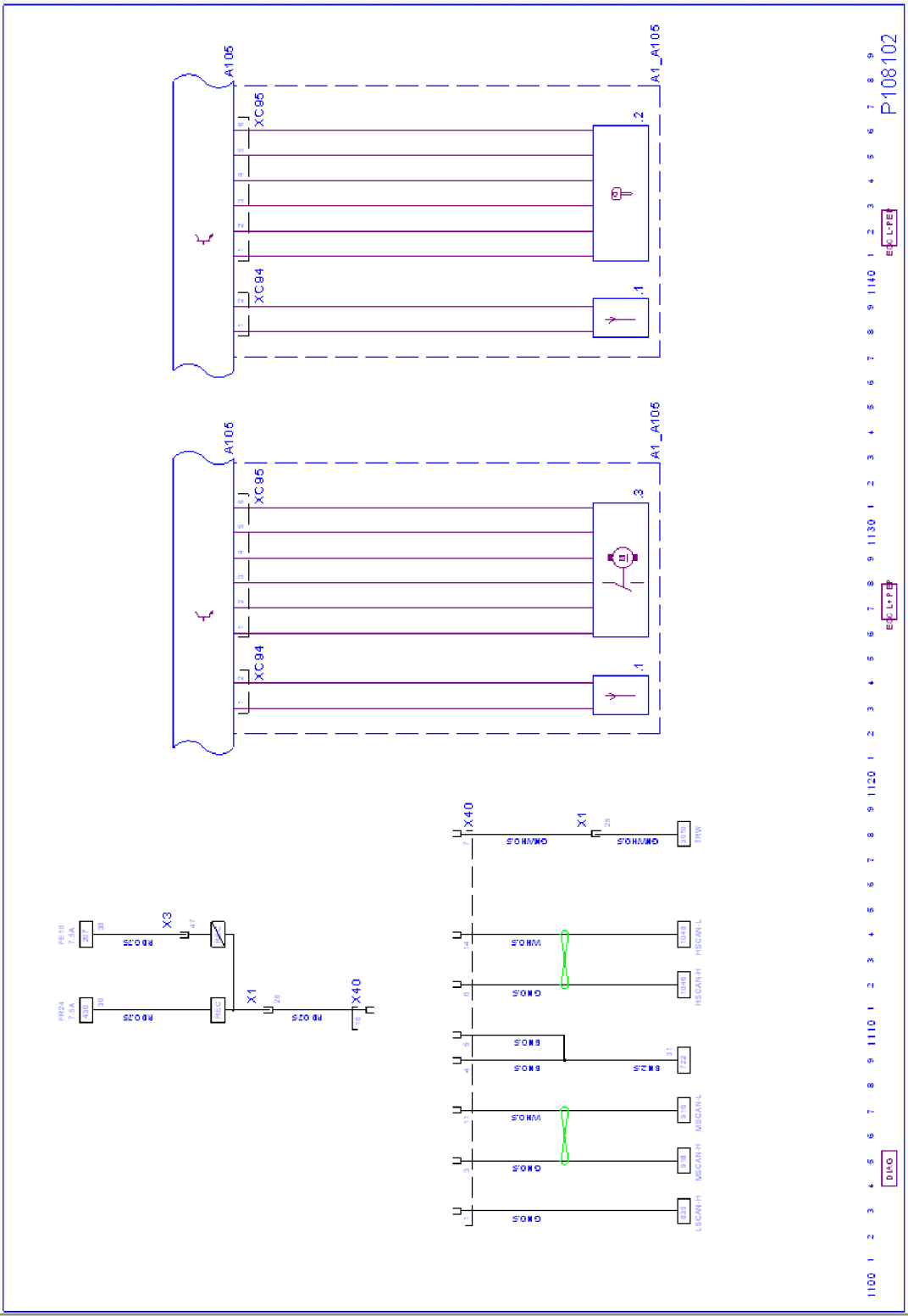
[24] Android Developers: Application Fundamentals [online]. 2015 [cit. 2015-06-15]. Dostupné na URL: <http://developer.android.com/guide/components/fundamentals.html>

[25] Mike Wolfson: Android Development Tools Essentials, First edition. Sebastopol: O'Reilly Media, Inc., 2013, ISBN: 978-1-449-32821-4

[26] Tom Denton: Automobile electrical and electronic systems, Third edition. Burlington: Elsavier Butterworth-Heinemann, 2004, ISBN: 0-7506-62190

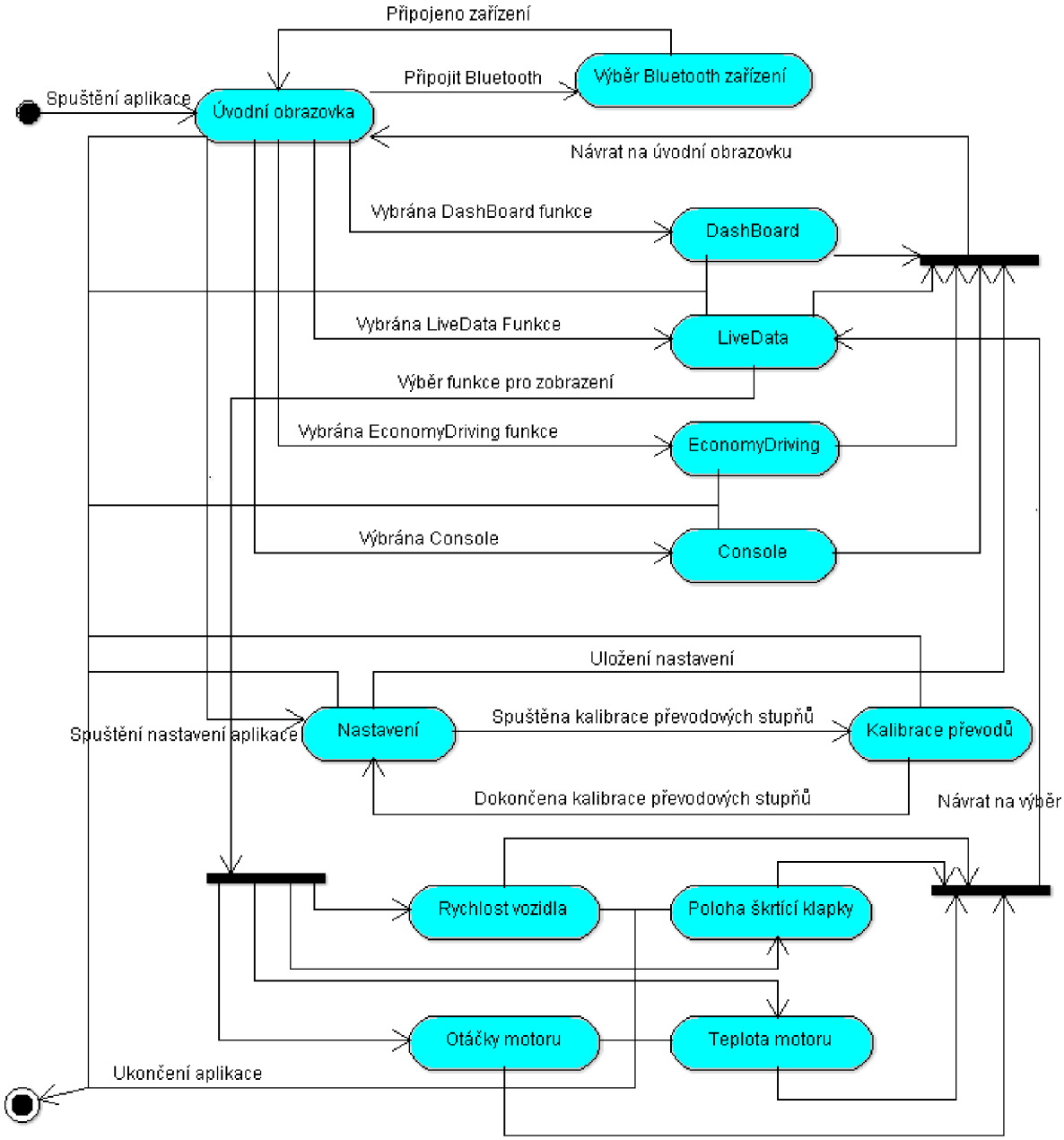
Příloha A – Schéma zapojení diagnostického konektoru

Následující obrázek ukazuje schéma zapojení diagnostického konektoru v automobilu Opel Astra H. Schéma pochází z diagnostického software TIS2000.



Příloha B – Diagram návaznosti obrazovek

Diagram vytvořený v době návrhu aplikace, který ukazuje návaznost jednotlivých obrazovek. Neobsahuje všechny obrazovky obsažené ve výslední aplikaci.



Příloha C – Obsah přiloženého DVD

Zdrojové soubory vytvořené Android aplikace
Zdrojové soubory celého projektu pro Android Studio
Text diplomové práce ve formátu Microsoft Word docx
Text diplomové práce ve formátu PDF
Přeložená aplikace pro nainstalování do mobilního zařízení se systémem Android