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

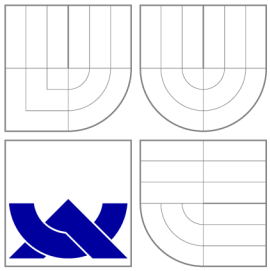
SPRÁVA NÁKLADŮ NA PROVOZ AUTOMOBILU PRO ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

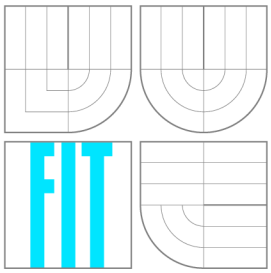
AUTOR PRÁCE
AUTHOR

JAN DORAZIL

BRNO 2011



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

SPRÁVA NÁKLADŮ NA PROVOZ AUTOMOBILU PRO ANDROID

COSTS OF RUNNING A CAR ADMINISTRATION ANDROID APPLICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN DORAZIL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK KUBÍČEK

BRNO 2011

Abstrakt

Bakalářská práce se zabývá mobilní platformou Android. V rámci práce byla vytvořena aplikace pro jednoduchou správu nákladů na provoz automobilu. Práce řeší návrh a implementaci této aplikace a rozebírá problematiku výpočtu spotřeby automobilu a emisí oxidu uhličitého. Aplikace používá SQLite databázi, asynchronní úlohy a moderní prvky uživatelského rozhraní.

Abstract

The bachelor thesis deals with mobile Android platform. The application for administration of costs of running a car was created within this thesis. It describes design and implementation of the application and analyses calculation of car's consumption and carbon dioxide emissions. The application uses SQLite database, asynchronous tasks and modern user interface components.

Klíčová slova

Android, Google, provoz automobilu, spotřeba automobilu, emise CO_2, databáze, SQLite, grafické uživatelské rozhraní, GUI, mobilní aplikace, Java.

Keywords

Android, Google, car administration, car consumption, CO_2 emissions, database, SQLite, graphical user interface, GUI, smartphone application, Java.

Citace

Jan Dorazil: Správa nákladů na provoz automobilu pro Android, bakalářská práce, Brno, FIT VUT v Brně, 2011

Správa nákladů na provoz automobilu pro Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Kubíčka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Dorazil
10. května 2011

Poděkování

Rád bych poděkoval panu Ing. Radkovi Kubíčkovi, vedoucímu bakalářské práce, za odbornou pomoc, ochotu a čas, který mi při tvorbě práce věnoval.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Na trhu s mobilními telefony jsou čím dál více populární telefony s dotykovým ovládáním. S nimi zažívají rozmach také mobilní operační systémy, u nichž je kladen důraz na přehlednost, jednoduchost a pohodlné ovládání. To umožňuje rozšíření telefonů s operačními systémy, které byly dříve výsadou spíše zkušenějších uživatelů, mezi širokou masu spotřebitelů. Zároveň je zachována volnost v psaní programů, jelikož aplikace mohou plně využít hardwarové a softwarové vybavení telefonu. Jedním z moderních mobilních operačních systémů je právě Android od společnosti Google.

Společně s vývojem nových mobilních operačních systémů – Android, iOS, Windows Phone 7 – byly vytvořeny služby Android Market, App Store a Marketplace pro centralizovanou distribuci aplikací ke koncovým uživatelům. Tento model se v praxi velmi osvědčil, protože je výhodný pro všechny zúčastněné strany. Uživatel má k dispozici výkonné zařízení se spoustou funkcí a možností rychlého získání velkého množství dalších aplikací a her přímo z telefonu. Pro vývojáře je platforma také zajímavá, jelikož nabízí rychlou a jednoduchou distribuci vytvořených aplikací mezi mnoho uživatelů. Výrobce pak těží z prodaných telefonů a obvykle má též podíl z prodaných aplikací a her.

Cílem bakalářské práce je vytvořit aplikaci pro jednoduchou správu nákladů na provoz automobilu. Tato problematika je zajímavá zejména proto, že potenciální základna uživatelů je velká, neboť v dnešní době vlastní automobil spousta lidí. Navíc peníze na správu automobilu tvoří nemalou část rodinného rozpočtu a je dobré o nich mít přehled ideálně kdykoliv a kdekoliv, protože například účty za tankování mohou být snadno ztraceny nebo jejich evidence zapomenuta. Jelikož je aplikace určena pro mobilní platformu Android, bude ji mít uživatel stále při sobě. To jí dává výhodu oproti některým stávajícím řešením, jež mají pouze webové rozhraní.

První část práce pojednává o platformě Android. Je vysvětleno, co si představit pod pojmem aplikace pro Android, jsou popsány její části a je nastíněn jejich životní cyklus, jehož pochopení je klíčové pro správnou funkčnost aplikace.

V kapitolách 3 a 4 jsou popsány etapy návrhu aplikace, zejména model databáze, uživatelské rozhraní a problémy, jež bylo nutné řešit při výpočtu spotřeby automobilu. Dále jsou uvedeny zajímavosti týkající se implementace aplikace. Především vytvoření vlastního seznamu zobrazujícího provoz automobilu a práce s asynchronními úlohami, jež zajišťují běh dlouhotrvajících operací na pozadí.

Na závěr je nastíněn budoucí vývoj projektu – možnosti rozšíření funkcionality aplikace a její distribuce přes Android Market.

Kapitola 2

Platforma Android

Informace uvedené v této kapitole jsou čerpány z knih [1, 11] zabývajících se vývojem aplikací pro platformu Android OS (dále jen Android) a z internetu [3, 8, 9, 15].

Android je open-source softwarová platforma určená pro mobilní zařízení. Obsahuje operační systém (založený na Linuxu), middleware[1] a soubor základních mobilních aplikací spolu s API[2] knihovnami, jež umožňují psaní aplikací pro tuto platformu.

Android byl původně vyvíjen firmou Android Inc., kterou v roce 2005 odkoupila společnost Google. Android byl oficiálně ohlášen 5. listopadu 2007. Současně vzniklo sdružení firem OHA (Open Handset Alliance) za účelem prosazování otevřených standardů ve světě mobilních zařízení. Jeho členy se staly významné společnosti IT průmyslu – Google, Texas Instruments, Qualcomm, HTC, Intel, Nvidia, LG, Motorola, Samsung Electronics ad. Jako první produkt konsorcia OHA byl ohlášen právě Android. Odsavec byl převzat z [15].

2.1 Architektura

Obsah této podkapitoly byl čerpán z [1, 8, 9, 11]. Architektura Androidu sestává z vrstev a prvků zobrazených na obrázku 2.1. V anglické literatuře je často označována jako *Android software stack*.

2.1.1 Linux kernel

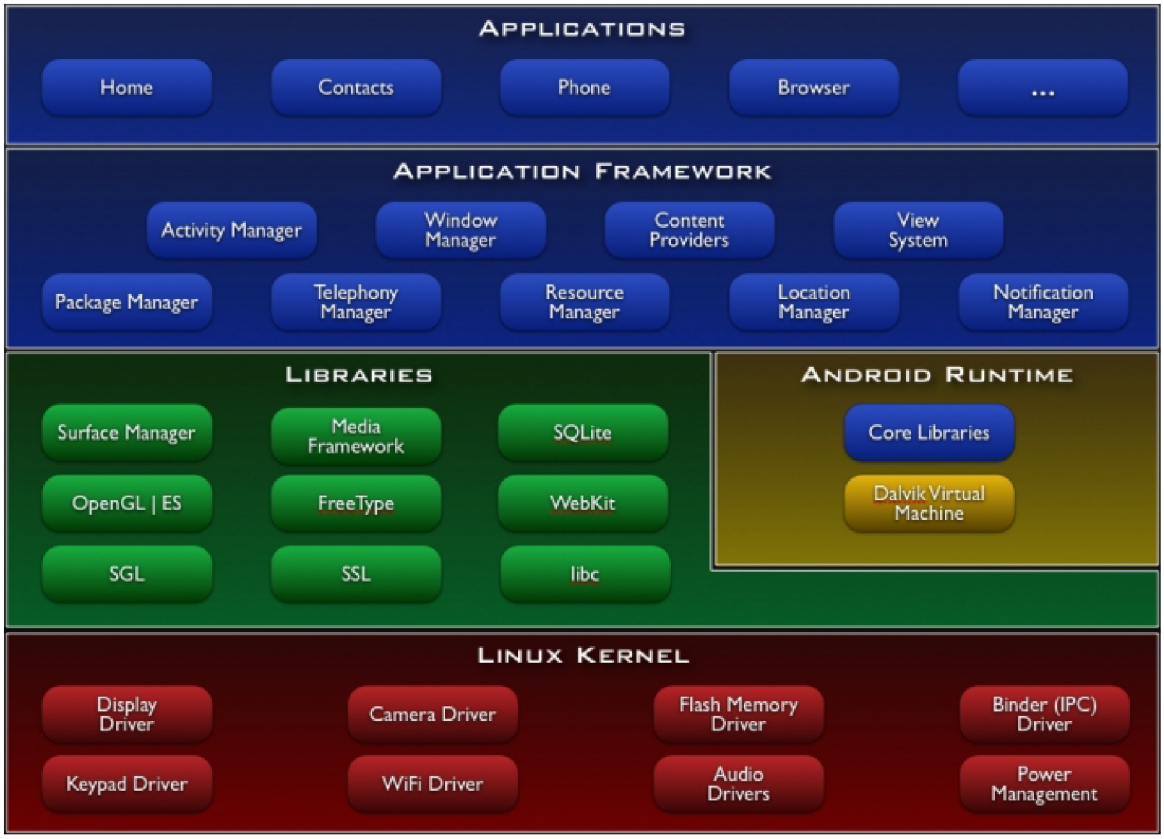
Vrstva poskytující základní systémové služby – hardwarové ovladače, správu procesů a paměti, bezpečnost, síťové služby a správu napájení. Je založena na linuxovém jádře verze 2.6. Slouží také k abstrakci hardwaru od ostatních vrstev architektury [8].

2.1.2 Knihovny

Vrstva ležící nad linuxovým jádrem obsahuje knihovny napsané v programovacích jazycích C a C++. Jsou kompilovány pro konkrétní hardwarovou architekturu a na zařízení předinstalovány výrobcem. Nejedná se o samostatné aplikace, jejich funkce jsou volány z programů vyšších vrstev.

[1] Softwarová vrstva ležící mezi operačním systémem a aplikacemi.

[2] Zkratka pro *Application Programming Interface* neboli aplikační programové rozhraní.



Obrázek 2.1: Architektura Androidu [8].

Některé základní knihovny [1, 8]:

- **Systémová knihovna jazyka C** – knihovna odvozená z BSD[3] impementace standardní systémové knihovny jazyka C upravená pro použití ve vestavěných systémech založených na Linuxu.
- **Multimediální knihovny** – knihovny pro podporu přehrávání a nahrávání populárních formátů audia, videa a obrázků.
- **Surface manager** – spravuje přístup k displeji zařízení. Spojuje 2D a 3D grafické vrstvy více aplikací, čímž umožňuje vytváření různých grafických efektů – např. průhlednost nebo animované přechody.
- **WebKit a SSL** – vykreslovací jádro internetových stránek a zabezpečení síťové komunikace. WebKit je použit nejen ve webovém prohlížeči, ale také ve vestavitelné komponentě *web view*.
- **Grafické knihovny** – obsahují knihovny SGL[4] a OpenGL ES[5] pro 2D a 3D grafiku.

[3] *Berkeley Software Distribution*. Operační systém odvozený od Unixu, distribuovaný Kalifornskou univerzitou v Berkeley [13].

[4] Více na http://sgl.sourceforge.net/.

[5] Více na http://www.khronos.org/opengles/.

- **FreeType** – slouží k vykreslování bitmapových a vektorových fontů.
- **SQLite** – jednoduchá a výkonná relační databáze.[6]

2.1.3 Android runtime

Android runtime leží nad linuxovým jádrem a spolu s knihovnami (2.1.2) tvoří základ pro aplikační framework.

Jelikož aplikace pro Android jsou psány v jazyce Java, Android runtime se skládá z knihoven poskytujících většinu funkcionality dostupné v základních knihovnách tohoto programovacího jazyka. Dále jsou zde knihovny specifické pro Android a *Dalvik Virtual Machine* (virtuální stroj Dalvik, dále jen DVM) [8].

DVM je implementace Javy od společnosti Google, optimalizovaná pro mobilní zařízení. Optimalizace se týkají zejména efektivního běhu více instancí DVM. K tomuto účelu poskytuje linuxové jádro podporu vláken a správu paměti na nízké úrovni [1, 11].

2.1.4 Aplikační framework

Vrstva obsahuje základní stavební bloky, které slouží k vytváření aplikací na vysoké úrovni. Důležitou vlastností Androidu je, že všechny aplikace – včetně standardních – používají stejné API. Je tedy možné *plnohodnotně* nahradit standardní aplikaci jinou aplikací třetí strany (např. aplikaci pro práci se zprávami) [1, 8].

Některé důležité části frameworku:

- ***Activity manager* (správce aktivit)** – řídí životní cyklus aplikací (viz 2.5) a udržuje zásobník sloužící k navigaci mezi aplikacemi a jejich aktivitami – tzv. *back stack* (viz 2.3).
- ***View system*** – rozšiřitelný systém komponent GUI.[7] Obsahuje textová pole, tlačítka, seznamy a další.
- ***Resource manager* (správce zdrojů)** – poskytuje přístup ke zdrojům, které nejsou součástí kódu aplikace. Jsou to např. lokalizované řetězce, grafika nebo deklarace GUI.
- ***Notification manager* (správce upozornění)** – umožňuje všem aplikacím zobrazovat upozornění ve stavové liště.
- ***Content providers* (poskytovatelé obsahu)** – dovolují sdílet perzistentní data mezi aplikacemi (např. kontakty).

2.1.5 Aplikace

Všechny aplikace jsou napsány v programovacím jazyce *Java* [1]. Tato nejvyšší vrstva architektury se skládá ze dvou části – *aplikace* a *widgety*. Aplikace jsou programy, jež můžou převzít kontrolu nad celou obrazovkou a interagovat s uživatelem.

Naproti tomu widgety zabírají pouze malý obdelník na ploše domovské aplikace. Umožňují zobrazovat důležité informace přímo, bez nutnosti otevírat aplikaci [8].

[6] Více na http://www.sqlite.org/.
[7] Zkratka pro *Graphical User Interface* neboli grafické uživatelské rozhraní.

Aplikace v mnoha ohledech „žije ve svém vlastním světě“ [9]:

- Pokud není určeno jinak, každá aplikace běží ve svém vlastním linuxovém procesu. Android vytváří proces kdykoliv je třeba vykonávat kód aplikace a ruší jej, pokud již není potřeba a systémové prostředky (např. paměť RAM) jsou vyžadovány jinými aplikacemi.
- Každemu procesu je přidělena vlastní instance DVM. Kód aplikace je tak vykonáván v izolaci od kódu ostatních aplikací.
- Každé aplikaci je přiděleno unikátní linuxové uživatelské ID. Oprávnění jsou nastavena tak, že soubory aplikace jsou viditelné pouze pro tohoto uživatele a aplikaci samotnou. Avšak existují způsoby, jak data aplikací sdílet (viz 2.2.4).

2.2 Komponenty aplikace

Obsah této podkapitoly vychází z [9]. Narozdíl od aplikací v jiných operačních systémech, aplikace pro Android nemají jediný vstupní bod – nenajdeme zde žádnou funkci `main()`. Aplikace se skládají z *komponent*, jejichž instance mohou být systémem podle potřeby vytvářeny a spouštěny. Existují čtyři typy komponent – *aktivity*, *služby*, *přijímače broadcastu* a *poskytovatelé obsahu*.

2.2.1 Aktivity

Aktivita (*activity*) představuje obrazovku s grafickým uživatelským rozhraním. Také zajišťuje reakce na události v tomto GUI vyvolané. Ačkoliv více aktivit tvoří dohromady ucelené uživatelské rozhraní, každá z nich je nezávislá na ostatních. Aktivita musí být implementována jako potomek bázové třídy `Activity`.

Aplikace se obvykle skládá z více aktivit, kdy každá řeší jinou část programu. Jedna z nich může být označena jako výchozí (viz 2.2.8), což zajistí její spuštění při startu aplikace. Pro přechod z jedné aktivity na jinou je použit mechanismus aktivace (viz 2.2.5).

Každá aktivita má k dispozici okno, do kterého může GUI vykreslit. Obvykle je velikost okna taková, aby vyplnila obrazovku zařízení, smí být ale i menší. Aktivita může využít další okna například pro výběr položky ze seznamu nebo pro dialogové okno.

Grafický obsah okna zajišťuje hierarchie objektů odvozených od bázové třídy `View`. Každý objekt ovládá určitou obdelníkovou část okna. Android obsahuje řadu již hotových objektů – tlačítka, textová pole, check boxy a další. Hierarchie objektů *view* je umístěna do okna aktivity voláním metody `Activity.setContentView()`, které je předán kořenový objekt hierarchie.

2.2.2 Služby

Služba (*service*) běží po neurčitou dobu na pozadí bez přímé interakce s uživatelem. Nemá grafické uživatelské rozhaní. Služba může být použita například pro přehrávání hudby, stahování dat z internetu nebo pro dlouhotrvající výpočty. Každá služba je odvozena od bázové třídy `Service`.

Typickým příkladem použití služby je hudební přehrávač. Skládá se z několika aktivit – pro výběr skladeb, ovládání přehrávané skladby, nastavení, ... Je zřejmé, že uživatel očekává souvislé přehrávání, zatímco prochází skladbami, upravuje nastavení aplikace nebo

se dokonce v aplikaci nenachází a provádí zcela jinou činnost. Proto nemůže být přehrávání hudby řízeno aktivitou. Je tedy vhodné použít službu, která zajistí přehrávání vybraného seznamu skladeb na pozadí.

K probíhající službě se lze připojit pomocí operace *bind*. Pokud služba neběží, může být spuštěna. Po navázání spojení je možné komunikovat se službou přes rozhraní, jež poskytuje. Rozhraní hudební služby umožňuje například pauzu, přetočení nebo zastavení přehrávání. Více o službách viz 2.5.2.

Jako aktivity a další komponenty, služby jsou prováděny v hlavním vlákně aplikačního procesu. Aby nedocházelo k blokování jiných komponent nebo uživatelského rozhraní, služby si často vytváří vlastní vlákno, pokud potřebují vykonávat dlouhotrvající činnost (jako přehrávání hudby). Více o procesech a vláknech viz 2.4.

2.2.3 Přijímače broadcastu

Přijímač broadcastu (*broadcast reciever*) je komponenta, která naslouchá a reaguje na broadcastová oznámení. Spousta oznámení má svůj původ v systému – např. oznámení o změně časové zóny, o nízkém stavu akumulátoru, o pořízení fotografie nebo o změně nastavení jazyka. Také aplikace mohou být zdrojem broadcastových oznámení – např. oznámení o dokončení stahování dat z internetu a jejich připravenosti k dalšímu použití.

Všechny přijímače broadcastu jsou odvozeny od bázové třídy `BroadcastReciever`. Přijímače broadcastu nemají uživatelské rozhraní, avšak mohou spustit aktivitu jako reakci na obdrženou informaci nebo využít systém notifikací k získání pozornosti uživatele.

2.2.4 Poskytovatelé obsahu

Poskytovatel obsahu (*content provider*) slouží ke sdílení určité množiny dat mezi aplikacemi. Aplikace jeho prostřednictvím povoluje jiným aplikacím manipulaci se svými daty. Data mohou být uložena v souborovém systému nebo SQLite databázi.

Poskytovatel obsahu je odvozen od bázové třídy `ContentProvider` a musí implementovat standardní množinu metod, které dovolují ostatním aplikacím získávat a ukládat data, jež jsou poskytovatelem spravována. Aplikace tyto metody nevolají přímo, ale používají objekt typu `ContentResolver` a jeho metody. Content resolver je schopen komunikovat s jakýmkoliv poskytovatelem obsahu.

2.2.5 Aktivace komponent

Poskytovatelé obsahu jsou aktivováni, jsou-li cílem požadavku content resolveru. Zbývající tři typy komponent – aktivity, služby a přijímače broadcastu – jsou aktivovány asynchronní zprávou zvanou *intent* (česky úmysl nebo záměr). Intent je objekt typu `Intent`, který nese obsah zprávy. Pro aktivity a služby je – mimo jiné – obsahem zprávy jméno požadované akce a cesta (URI[8]) k zpracovávaným datům. Například požadavek na spuštění aktivity pro zobrazení fotky a URI fotky, kterou má aktivita zobrazit. Pro příjímače broadcastu intent nese jméno oznamované akce – např. oznámení o stisku tlačítka fotoaparátu.

Metody pro aktivaci komponent:

- **Aktivita** je spuštěna předáním objektu intent metodě `Context.startActivity()`. Jedna aktivita často spouští další aktivitu. Pokud od ní očekává vrácení výsledku,

[8]Zkratka pro *Uniform Resource Identifier* neboli jednoznačný identifikátor zdroje.

namísto výše uvedené volá metodu `Activity.startActivityForResult()`. Výsledek je vrácen v intent objektu, jenž je předán metodě `OnActivityResult()` ve volající aktivitě. Příkladem nechť je volání aktivity pro výběr fotky a následné vrácení fotky uživatelem zvolené.

- **Služba** je spuštěna předáním objektu intent metodě `Context.startService()`. Pokud již služba běží, jsou jí předány nové instrukce. Android volá metodu služby `onStartCommand()` a předává jí intent.

  Intent lze také předat metodě `Context.bindService()` pro ustavení spojení mezi volající komponentou a službou. Služba obdrží intent v metodě `onBind()`. Pokud služba neběží, `bindService()` ji může volitelně spustit. Po ustavení spojení lze z volající komponenty volat metody služby. Například aktivita s GUI ovládajícím hudební přehrávač se připojí ke službě zajišťující přehrávání hudby a volá její metody. Tento mechanismus je implementován pomocí volání vzdálených procedur.[9]

- **Broadcastové vysílání** je iniciováno aplikací předáním instance objektu intent metodě `Context.sendBroadcast()` nebo některé z jejích variant. Android doručí intent všem přijímačům broadcastu, jež mají o daný typ oznámení zájem, voláním jejich metody `onRecieve()` (viz 2.5.3).

2.2.6 Ukončení komponent

Poskytovatel obsahu je aktivní pouze ve chvíli, kdy odpovídá na dotaz content resolveru. Podobně přijímač broadcastu je aktivní, pouze když reaguje na broadcastovou zprávu. Tyto komponenty tedy není třeba explicitně ukončovat.

Na druhou stranu aktivity a služby mohou setrvávat v aktivním stavu velmi dlouho. Jelikož aktivita poskytuje uživatelské rozhraní, zůstává aktivní po celou dobu komunikace s uživatelem. Také služba může být aktivní dlouho – např. během přehrávání hudby na pozadí. Proto má Android metody umožňující řádné ukončení aktivit a služeb:

- **Aktivita** ukončí samu sebe voláním své metody `finish()`. Aktivita může ukončit jinou aktivitu voláním metody `finishActivity()`, ale jen v případě, že ji předtím spustila voláním `startActivityForResult()`.

- **Služba** je ukončena voláním své metody `stopSelf()`, případně voláním metody `stopService()` z jiné komponenty.

Komponenty mohou být také ukončeny systémem, pokud již nejsou používány nebo pokud Android potřebuje uvolnit pamět pro více aktivní komponenty (více viz 2.5).

2.2.7 Manifestový soubor

Než Android spustí některou z komponent, musí se nejprve dozvědět o její existenci. Každá aplikace deklaruje své komponenty v *manifestovém souboru*, jenž je zabalen do instalačního balíčku aplikace – soubor s příponou `apk`, obsahující kód aplikace, zdroje a další potřebné soubory.

Manifest je strukturovaný XML[10] soubor. Musí být součástí každé aplikace a vždy je pojmenován `AndroidManifest.xml`. Ačkoliv jeho hlavním účelem je deklarovat komponenty

[9] Více na http://developer.android.com/guide/developing/tools/aidl.html.
[10] Zkratka pro *Extensible Markup Language* neboli rozšiřitelný značkovací jazyk.

aplikace, slouží také například k identifikaci oprávnění, jejichž udělení aplikace očekává (čtení a zápis dat na paměťovou kartu, plný přístup k internetu, snímání polohy pomocí GPS, . . .).

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
  <application ...>
    <activity android:name=".Login">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".MainTabs"></activity>
    <activity android:name=".TrafficTab"></activity>
    <activity android:name=".CarTab"></activity>
    <activity android:name=".ProfileTab"></activity>
    ...
  </application>
  ...
</manifest>
```

Výpis 2.1: Příklad deklarace aktivit a intent filtru.

Ve výpisu 2.1 je uveden příklad deklarace několika aktivit. Atribut `name` prvku `<activity>` určuje podtřídu bázové třídy `Activity`, která implementuje aktivitu.

Zbylé komponenty jsou deklarovány stejným způsobem – prvky `<service>` pro služby, `<reciever>` pro přijímače broadcastu a `<provider>` pro poskytovatele obsahu. Aktivity, služby a poskytovatelé obsahu, kteří nejsou deklarováni v manifestu, nejsou pro systém viditelní a tudíž nemohou být nikdy spuštěni. Naproti tomu přijímač broadcastu smí být buď deklarován v manifestu nebo dynamicky vytvořen v kódu jako instance objektu `BroadcastReciever` a zaregistrován v systému voláním `Context.registerReciever()`.

2.2.8 Intent filtry

Pokud intent explicitně jmenuje cílovou komponentu, Android ji vyhledá na základě deklarace v manifestovém souboru a aktivuje. V opačném případě musí najít nejlepší možnou komponentu, která je schopna na intent odpovědět. Hledání provádí porovnáváním intentu s *intent filtrem* potenciálních cílových komponent. Intent filtr informuje Android o typech intentů, jež je komponenta schopna zpracovat. Intent filtry komponent jsou deklarovány v manifestovém souboru.

Ve výpisu 2.1 je vidět příklad deklarace intent filtru. Tento typ filtru je velmi častý. Označuje aktivitu, jež má být spuštěna, pokud uživatel spustí aplikaci. Jde tedy o vstupní bod aplikace.

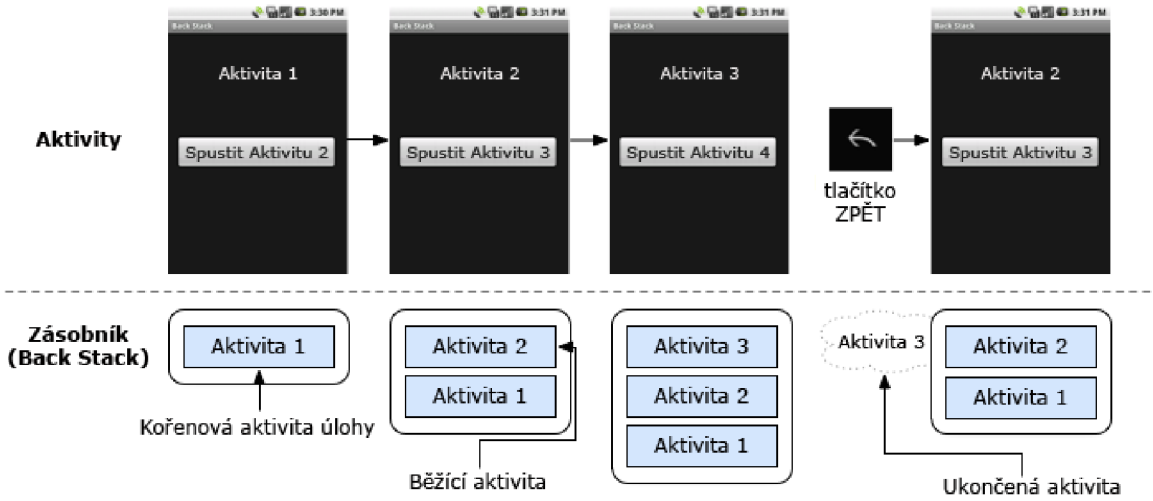
Komponenta smí mít libovolný počet intent filtrů. Pokud nemá žádný, jediný způsob, jak ji aktivovat, je pomocí intentu, který ji explicitně jmenuje jako cílovou komponentu.

2.3 Úlohy

Obsah této podkapitoly vychází převážně z [3, 9]. Jak již bylo uvedeno, aktivita může spouštět jiné aktivity a to včetně těch, jež jsou definovány v jiné aplikaci. Například pokud aplikace potřebuje zobrazit adresu na mapě – k tomuto účelu již v systému aktivita existuje – stačí vytvořit patřičný intent a předat jej metodě `startActivity()`. Prohlížeč map poté mapu zobrazí. Pokud uživatel zmáčkne tlačítko „zpět“, na obrazovce se znovu objeví předchozí aktivita. Uživateli se to jeví, jakoby byl prohlížeč map součástí aplikace, ačkoliv je definován v jiné aplikaci a běží v jejím aplikačním procesu.

Přestože jsou obě aktivity definovány v různých aplikacích a běží v jiných procesech, Android je uchovává v jedné *úloze* (anglicky *task*). Úloha je to, co se z pohledu uživatele jeví jako jedna „aplikace“. Jde o skupinu souvisejících aktivit uspořádaných do *zásobníku* – v angličtině je označován jako *back stack*. Aktivita na dně zásobníku je ta, která úlohu začala – obvykle jde o aktivitu, jež spustil uživatel kliknutím na ikonu aplikace. Aktivita na vrcholu zásobníku je v běžícím stavu, připravena reagovat na akce uživatele. Když aktivita spustí jinou, nová aktivita je vložena na vrchol zásobníku a přepnuta do běžícího stavu, předchozí v zásobníku zůstává. Pokud uživatel stiskne tlačítko „zpět“, nová aktivita je vyjmuta ze zásobníku – předchozí pokračuje v běhu.

Tento princip demonstruje obrázek 2.2, ve kterém Aktivita 1 spustí Aktivitu 2 a ta poté spustí Aktivitu 3. Pod přerušovanou čarou je naznačeno, jak jsou aktivity postupně ukládány do zásobníku. Následně je zmáčknuto tlačítko „zpět“, což má za následek ukončení Aktivity 3 a obnovení Aktivity 2. Více o stavech, v nichž se může aktivita během svého životního cyklu nacházet viz 2.5.1.



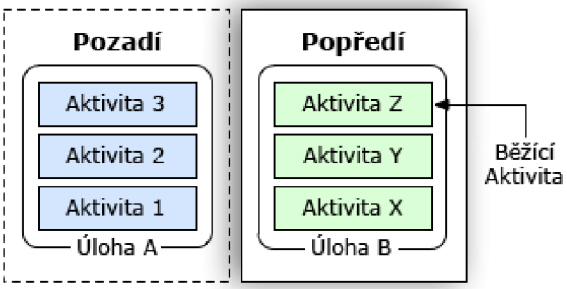
Obrázek 2.2: Zásobník aktivit tvořící jednu úlohu [3].

Zásobník obsahuje objekty, pokud je tedy v úloze otevřeno více instancí jedné podtřídy `Activity` (např. více prohlížečů map), zásobník má oddělený záznam o každé z nich. Aktivity v zásobníku nemohou být nikdy přeskládány, jediné povolené operace jsou vložení a vyjmutí položky (často označované anglickými termíny *push* a *pop*).

Všechny aktivity v úloze jsou společně přesouvány jako celek. Celá úloha (zásobník) smí být přesunuta do popředí nebo na pozadí. Tímto způsobem systém Android umožňuje přepínání mezi úlohami – *multitasking*. Při stisknutí tlačítka „domů“ je aktuální úloha pře-

nesena na pozadí a je zobrazen spouštěč aplikací. Uživatel může spustit jinou aplikaci – začít novou úlohu. Po dlouhém stisku klávesy „domů“ může uživatel opět aktivovat předchozí úlohu – stávající je přesunuta na pozadí a předchozí úloha je opět přenesena do popředí. Funkce klávesy „zpět“ se vždy vztahuje k úloze (jejímu zásobníku), která je v popředí.

Na obrázku 2.3 jsou zobrazeny dvě úlohy. Úloha A je na pozadí a čeká na obnovení. Úloha B je v popředí a její Aktivita Z, jež je na vrcholu zásobníku, reaguje na podněty uživatele. Ačkoliv úloh čekajících na pozadí smí být více, mohou být systémem ukončeny z důvodu potřeby uvolnění operační paměti [3].



Obrázek 2.3: Úloha na pozadí a v popředí [3].

2.4 Procesy a vlákna

Obsah této podkapitoly byl čerpán z [9]. Ve chvíli, kdy potřebuje být spuštěna první komponenta aplikace, Android pro ni vytvoří proces s jedním hlavním vláknem. Ve výchozím nastavení běží všechny komponenty aplikace v tomto procesu a vlákně.

Nicméně je možné spouštět komponenty v jiných procesech, stejně tak lze pro každý proces vytvořit další vlákna.

2.4.1 Procesy

Proces, v němž běží aplikační komponenty, je nastaven v manifestovém souboru. Každý z prvků <`activity`>, <`service`>, <`reciever`> a <`provider`> má atribut `process`, jenž definuje, ve kterém procesu má být komponenta spuštěna. Atributy smí být nastaveny tak, že každá komponenta běží ve svém vlastním procesu nebo naopak některé komponenty sdílí proces, zatímco jiné mají svůj vlastní. Taktéž komponenty různých aplikací mohou běžet v jednom procesu – aplikace musí sdílet linuxové uživatelské ID a musí být podepsány stejnou autoritou. Atribut `process` prvku <`application`> slouží k nastavení výchozí hodnoty pro všechny komponenty aplikace.

Instance všech komponent běží v hlavním vlákně daného procesu. Systémová volání komponentám jsou vysílána z tohoto vlákna. Metody, jež odpovídají na systémová volání se tedy vždy nacházejí v hlavním vlákně – například `View.onKeyDown()`, která reaguje na stisk tlačítka nebo metody indikující změnu stavu komponenty (viz 2.5). Z toho vyplývá, že žádné komponenty by zde neměly provádět dlouhotrvající operace (síťová komunikace, složité výpočty, . . .), protože tím zablokují ostatní komponenty v procesu, což může mít za

následek mimo jiné výrazné zpomalení odezvy GUI. Namísto toho by měly využít oddělené vlákno viz 2.4.2.

Pokud je v systému nedostatek paměti, Android ukončí méně důležité procesy na úkor důležitějších. Spolu s procesem jsou ukončeny i všechny komponenty, které v něm běží. Proces je znovu spuštěn ve chvíli, kdy jsou jeho komponenty opět zapotřebí. Při rozhodování, který proces ukončit, Andorid používá hierarchii důležitosti procesu, viz 2.5.4.

2.4.2 Vlákna

Uživatelské rozhraní musí vždy rychle reagovat na podněty uživatele. Ve vlákně s aktivitami proto nesmí být přítomny operace, jež vykonávají časově náročnou činnost – například stahování dat z internetu. Všemu, co nemůže být vykonáno rychle, by mělo být přiřazeno vlastní vlákno.

Vlákna se vytvářejí v kódu za použití standardního objektu `Thread` jazyka Java. Lze také s výhodou využít třídu `AsyncTask`, která umožňuje provádět operace ve vlákně na pozadí a výsledky poté publikovat v hlavním vlákně [6]. Více viz 4.4.

2.5 Životní cykly komponent aplikace

Obsah této podkapitoly vychází z [4, 5, 7, 9]. Komponenty aplikace mají svůj životní cyklus – od vytvoření nové instance jako odpovědi na intent až po její zrušení. Mezitím mohou být aktivní či neaktivní nebo, v případě aktivit, viditelné nebo neviditelné pro uživatele. Tato podkapitola se zabývá životními cykly aktivit, služeb a poskytovatelů obsahu. Popisuje stavy, ve kterých se mohou nacházet, a metody sloužící k upozornění na změnu tohoto stavu.

Dále se podkapitola zabývá životním cyklem procesů a s tím související hierarchií důležitosti procesu, jež se uplatňuje při rozhodování, který proces má být ukončen, pokud systém potřebuje uvolnit paměť.

2.5.1 Životní cyklus aktivity

Aktivita se může nacházet ve třech stavech [4]:

- **Obnovena (*resumed*)** – aktivita je v popředí na obrazovce a má tzv. *focus*, tedy jsou na ni směrovány akce uživatele. Aktivita se nachází na vrcholu zásobníku aktuálně běžící úlohy (viz 2.3). Tento stav bývá také označován jako běžící (*running*).
- **Pozastavena (*paused*)** – aktivita nemá *focus* a je *z části* překryta jinou aktivitou nebo zcela překryta *průhlednou* aktivitou. Pozastavená aktivita je stále uchována v operační paměti, ačkoliv při extremě nízkem stavu paměti může být systémem ukončena.
- **Zastavena (*stopped*)** – aktivita je zcela překryta jinou neprůhlednou aktivitou a již *není pro uživatele viditelná* ani z části. Přestože je zastavená aktivita v pozadí, stále je uchována v operační paměti. Aktivita může být systémem ukončena, je-li potřeba uvolnit paměť.

Pokud se aktivita přepíná mezi výše popsanými stavy, je o tom informována pomocí volání speciálních metod. Tyto metody mohou být přepsány a rozšířeny o akce, jež jsou při změně stavu potřeba vykonat.

Ve výpisu 2.2 je uveden příklad přepsání metody `onStart()`. Pomocí anotace `@Override` říkáme překladači, že přepisujeme implementaci nadtřídy – tato anotace není povinná. Naproti tomu nezbytnou součástí je volání implementace nadtřídy – `super.onStart()`, která musí vždy předcházet jakékoli jiné operaci.

```
public class ExampleActivity extends Activity {
    ...
    @Override
    protected void onStart() {
        super.onStart();
        /* Akce prováděné předtím,
           než se aktivita stane viditelnou. */
        ...
    }
    ...
}
```

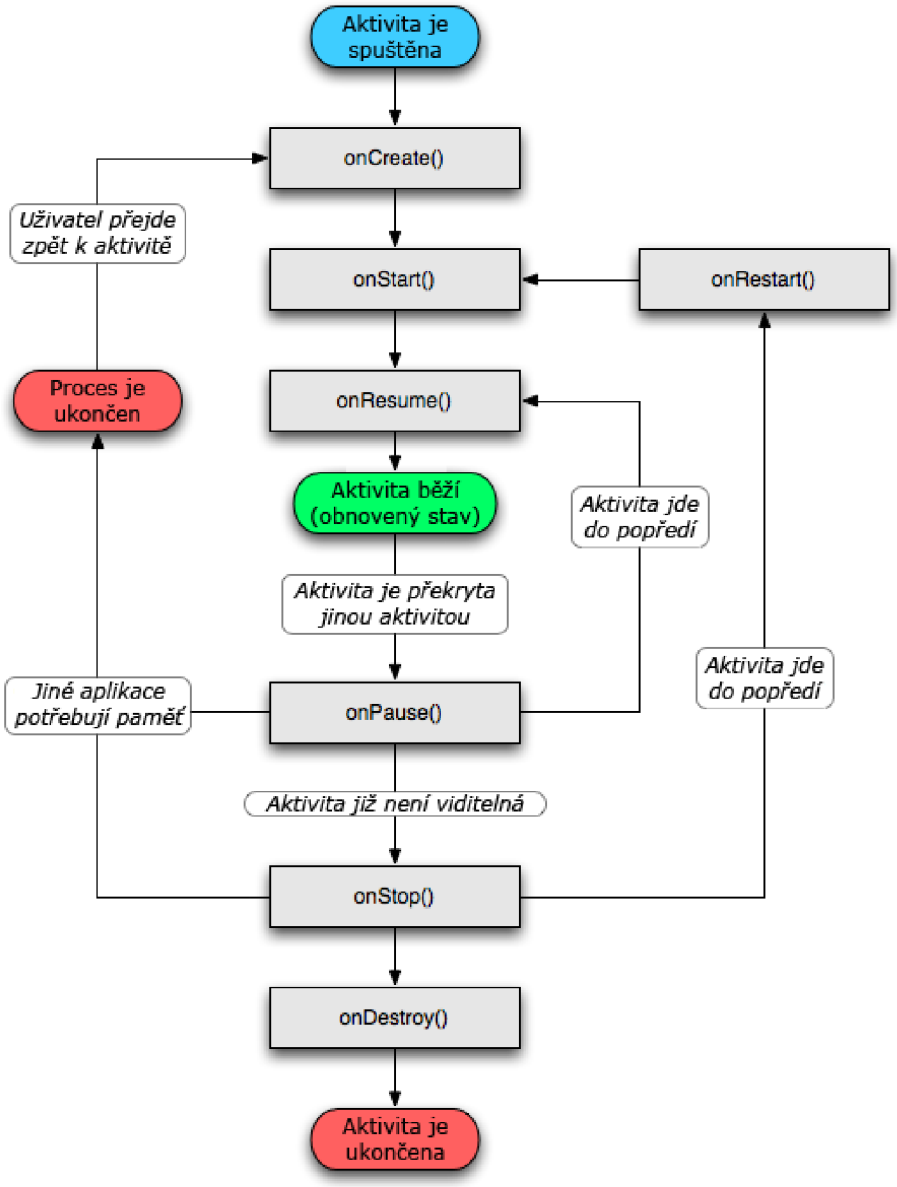
Výpis 2.2: Příklad přepsání metody `onStart()` [4].

Stejným zbůsobem smí být přepsány všechny metody informující o změně stavu aktivity:

- `void onCreate(Bundle savedInstanceState)` – aktivita je vytvářena. Parametr `savedInstanceState` může být využit k obnově předchozího stavu aktivity, byl-li uložen.
- `void onStart()` – volána těsně před tím, než se aktivita stane viditelnou.
- `void onResume()` – aktivita je viditelná, převzala *focus* a nachází se na vrcholu zásobníku úlohy (viz 2.3). Metoda je volána tesně před tím, než aktivita začne interagovat s uživatelem.
- `void onPause()` – jiná aktivita převzala *focus*. Je volána těsně před tím, než je aktivita *pozastavena*.
- `void onStop()` – aktivita není viditelná a je *zastavena*.
- `void onRestart()` – volána při přechodu ze zastaveného do pozastaveného stavu. Následuje volání metody `onStart()`.
- `void onDestroy()` – volána před ukončením aktivity.

Výše uvedené metody definují celý životní cyklus aktivity, jenž obsahuje *tři vnořené smyčky*:

1. **Celý život** aktivity probíhá mezi voláním metody `onCreate()` a `onDestroy()`. Aktivita by měla provést důležité inicializace v `onCreate()` – např. definici rozvržení uživatelského rozhraní. Naopak v metodě `onDestroy()` by měly být uvolněny všechny využívané zdroje. Například v `onCreate()` aktivita získá a otevře instanci databáze pro čtení a zápis, v `onDestroy()` pak databázi uzavře.
2. **Viditelná část** životního cyklu aktivity probíhá mezi voláním metod `onStart()` a `onStop()`. Během této doby je aktivita nebo její část viditelná na obrazovce. Metoda



Obrázek 2.4: Životní cyklus aktivity [4].

`onStop()` je volána, pokud nově spuštěná aktivita zcela překryje stávající tak, že již není viditelná žádná její část. Systém může volat `onStart()` a `onStop()` vícekrát během jednoho životního cyklu aktivity.

3. **Život v popředí** probíhá mezi voláním metody `onResume()` a `onPause()`. Během této doby je aktivita zobrazena před všemi ostatními aktivitami na obrazovce – je v obnoveném stavu. Přechod z pozastaveného do tohoto stavu a naopak je během životního cyklu aktivity velmi častý, např. pokud je zařízení uspáno nebo je zobrazeno dialogové okno. Proto by metody `onResume()` a `onPause()` měly obsahovat pouze jednoduchý kód, čímž zabrání pomalému přepínání stavu a nenutí tak uživatele dlouho čekat.

Na obrázku 2.4 je zobrazen životní cyklus aktivity spolu s výše popsanými smyčkami. Šedé obdelníky reprezentují již zmíněné metody upozorňující na změnu stavu aktivity.

2.5.2 Životní cyklus služby

Životní cyklus služby se může ubírat dvěma směry [7]:

- **Spuštěná služba (*started service*)** – služba je vytvořena ve chvíli, kdy jiná komponenta zavolá metodu `startService()`. Služba poté běží až do doby, než zavolá metodu `stopSelf()` nebo dokud není zastavena voláním `stopService()` z jiné komponenty. Jakmile je služba zastavena, systém ji ukončí.
- **Spojovaná služba (*bound service*)** – služba je vytvořena ve chvíli, kdy se k ní připojí první klient voláním metody `bindService()`. Klient poté může komunikovat se službou prostřednictvím rozhraní `IBinder`. Klient uzavírá spojení voláním metody `unbindService()`. Ke službě smí být současně připojeno více klientů. Služba je ukončena, jakmile poslední klient uzavře spojení.[11]

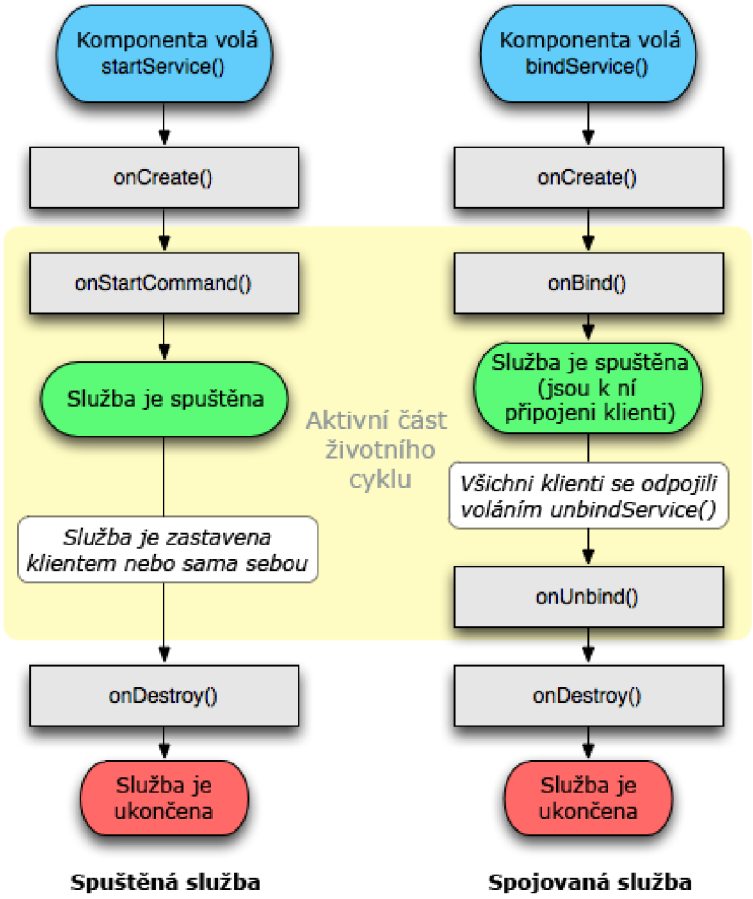
Tyto dva typy služeb mohou být navzájem kombinovány, klient se smí připojit ke službě, která již byla vytvořena metodou `startService()`. Například služba pro přehrávání hudby na pozadí bývá spuštěna metodou `startService()`, které je předán intent identifikující hudbu k přehrání. Později, když chce uživatel převzít kontrolu nad přehráváním, pozastavit přehrávání nebo zjistit informace o přehrávané skladbě, aktivita se připojí ke službě voláním `bindService()`. Jsou-li ke službě připojeni klienti, volání metod `stopSelf()` nebo `stopService()` službu nezastaví, dokud se všichni klienti neodpojí.

Jako aktivita, i služba implementuje důležité metody sloužící k upozornění na změnu stavu služby. Při jejich přepisování – narozdíl od aktivity – není nutné volat imlementaci nadtřídy. Výčet těchto metod je uveden bez parametrů a navratových typů:[12]

- `onCreate()` – volána při vytváření služby.
- `onStart()` – metoda je od verze Androidu 2.0 označena jako zavržená. V Androidu 2.0 a vyšším by místo ní měla být použita metoda `onStartCommand()`.
- `onStartCommand()` – volána při startu služby jako reakce na spuštění služby voláním metody `startService()`.

[11] Více o spojovaných službach na http://developer.android.com/guide/topics/fundamentals/bound-services.html.

[12] Kompletní výčet je uveden v [7].



Obrázek 2.5: Životní cyklus služby [7].

- `onBind()` – volána, pokud se klient připojuje ke službě metodou `bindService()`.
- `onUnbind()` – volána, pokud se všichni připojení klienti odpojili za použití metody `unbindService()`.
- `onRebind()` – volána, pokud se klient připojuje pomocí metody `bindService()` ve chvíli, kdy již byla zavolána metoda `onUnbind()`.
- `onDestroy()` – volána, pokud služba již není používána a bude ukončena.

Životní cyklus služby obsahuje dvě vnořené smyčky:

1. **Celý život** služby probíhá mezi voláním metod `onCreate()` a `onDestroy()`. Podobně jako aktivita, i služba by měla provádět počáteční inicializaci v metodě `onCreate()` a uvolňovat použité zdroje v `onDestroy()`. Například hudební služba vytvoří vlákno pro přehrávání hudby v metodě `onCreate()`, v `onDestroy()` jej zruší. Obě metody jsou volány pro všechny služby nezávisle na tom, zda byly vytvořeny metodou `startService()` nebo `bindService()`.
2. **Aktivní část** životního cyklu služby začíná voláním metody `onStartCommand()` nebo `onBind()`. Aktivita spuštěné služby končí ve stejnou chvíli jako celý její život. Spojovaná služba je aktivní do návratu metody `onUnbind()`.

Na obrázku 2.5 vlevo je zobrazen životní cyklus spuštěné služby, vpravo se nachází diagram životního cyklu spojované služby. Ačkoliv jsou životní cykly na obrázku uvedeny odděleně, mohou se prolínat, viz výše.

2.5.3 Životní cyklus přijímače broadcastu

Přijímač broadcastu má pouze jednu speciální metodu týkající se životního cyklu – `void onReceive(Context curContext, Intent broadcastMsg)`. Ve chvíli, kdy přijde zpráva pro přijímač broadcastu, Android zavolá jeho metodu `onReceive()` a předá jí intent s obsahem zprávy.

Přijímač broadcastu je *aktivní* jen po dobu vykonávání této metody. Jinak je *neaktivní*. Proces obsahující aktivní přijímač broadcatu je chráněn proti ukončení. Naopak proces, jehož všechny komponenty jsou neaktivní, smí být systémem ukončen kdykoliv je potřeba uvolnit operační paměť [9].

2.5.4 Životní cyklus procesu

Android zachovává všechny běžící aplikační procesy jak dlouho to jen jde. Dříve nebo později však musí některé ukončit, aby uvolnil místo novým a důležitějším. Při rozhodování, který ukončit, staví procesy do tzv. *„hierarchie důležitosti“* založené na komponentách, jež proces obsahuje a na jejich aktuálním stavu.

Android rozlišuje pět úrovní důležitosti. Následující výčet obsahuje popis jednotlivých úrovní, seřazený od nejvíce důležitých po nejméně důležité procesy [5]:

1. **Proces v popředí** – proces, jenž je nutný pro aktuální činnost uživatele. Proces je v popředí, splňuje-li alespoň jednu z následujících podmínek:

   - Obsahuje aktivitu, která interaguje s uživatelem – je v obnoveném stavu.
   - Obsahuje spojovanou službu, k níž je připojena aktivita interagující s uživatelem.
   - Obsahuje službu běžící v popředí – služba zavolala metodu `startForeground()`.
   - Obsahuje službu vykonávající jednu z metod svého životního cyklu – `onCreate()`, `onStart()`[13] nebo `onDestroy()`.
   - Obsahuje přijímač broadcastu, který právě vykonává svou metodu `onReceive()`.

   V jeden čas většinou existuje jenom několik málo procesů v popředí. Mohou být systémem ukončeny, ale pouze v krajních případech, kdy už není dostatek paměti ani pro běh jich samotných.

2. **Viditelný proces** – proces, který sice nesplňuje žádnou z podmínek procesu v popředí, přesto ale může ovlivnit to, co uživatel vidí na obrazovce. Proces je viditelný, splňuje-li alespoň jednu z následujících podmínek:

   - Obsahuje aktivitu v pozastaveném stavu – nemá uživatelův *focus*, ale je viditelná alespoň z části.
   - Obsahuje spojovanou službu, k níž je připojena aktivita v pozastaveném stavu.

   Viditelný proces je stále považován za velmi důležitý. Je systémem ukončen pouze v případě, že je třeba uvolnit paměť pro procesy v popředí.

[13] Metoda je zavržena viz 2.5.2.

3. **Proces se spuštěnou službou** – proces, který obsahuje službu spuštěnou metodou `startService()` a nespadá do žadné z předchozích úrovní. Ačkoliv tento typ procesu není přímo svázán s tím, co uživatel vidí na obrazovce, obvykle provádí činnost, jež je pro uživatele důležitá – například přehrávání hudby na pozadí nebo stahování dat z internetu. Proto je udržován v běhu, dokud není třeba uvolnit paměť pro viditelný proces nebo proces v popředí.

4. **Proces na pozadí** – proces obsahující zastavenou aktivitu – aktivita není viditelná, byla volána její metoda `onStop()`. Tento typ procesu nemá žádný vliv na to, co uživatel právě dělá a může být systémem ukončen, kdykoliv je třeba uvolnit paměť pro některý z předchozích typů procesu.

   Procesů běžících na pozadí je obvykle mnoho. Proto je Android uchovává v seznamu seřazeném podle toho, kdy byly jejich aktivity naposledy viděny uživatelem[14]. Při nedostatku operační paměti jsou ukončovány procesy, jejichž aktivity byly zobrazeny uživateli před nejdelší dobou.

5. **Prázdný proces** – proces, který neobsahuje žádnou aktivní komponentu aplikace. Slouží jako mezipamět pro rychlejší spuštění komponenty při příštím startu.

Android hodnotí důležitost procesu na základě všech v něm obsažených komponent. Vždy vybírá nejvyšší možnou důležitost. Například pokud proces obsahuje službu a pozastavenou aktivitu, Android jej ohodnotí jako viditelný proces, ne jako proces se spuštěnou službou.

[14] V originále LRU (*Last Recently Used*) list.

Kapitola 3

Návrh aplikace

Kapitola se zabývá návrhem aplikace – rozborem problematiky, modelem databáze a uživatelského rozhraní a zejména výpočtem spotřeby automobilu. Je také zmíněn výpočet emisí oxidu uhličitého (dále jen CO_2).

3.1 Rozbor problému

Cílem práce je vytvořit aplikaci umožňující jednoduchou a pohodlnou správu nákladů na provoz automobilu. Jako úvod do problematiky posloužilo prostudování několika existujících řešení. Nabyté poznatky byly užitečné při specifikaci požadavků na funkčnost vytvářené aplikace.

3.1.1 Přehled existujících řešení

Spotřeby.cz

Český server Spotřeby.cz[1] nabízí komplexní způsob sledování nákladů na provoz automobilu. Jeho výhodou je komunitní řešení, kdy uživatelé spolu mohou navzájem komunikovat a porovnávat své statistiky provozu.

Zajímavá je také síť čerpacích stanic po České republice a částečně i na Slovensku sestavená samotnými uživateli serveru. Uživatelé mohou stanice přidávat nebo využívat již vytvořené, přičemž u stanic zůstává uložena cena paliva při posledním použití některým uživatelem. To umožňuje například výběr čerpací stanice podle nejlevnější ceny paliva. Naopak nevýhodou tohoto řešení je závislost na aktivitě ostatních uživatelů, která může být zejména v odlehlejších částech republiky malá.

Spritmonitor.de

Německý server Spritmonitor.de[2] je podobný výše zmíněnému, oproti kterému však nabízí propracovanější uživatelské rozhraní, více statistik a verzi internetových stránek pro zobrazení na mobilních telefonech. Má také mnohonásobně větší základnu uživatelů – více než 220 tisíc, zatímco Spotřeby.cz mají necelých $6,5$ tisích.[3]

Nevýhodou pro českého uživatele by mohla být absence české lokalizace, momentálně existuje pouze německá a anglická.

[1] Dostupné na http://www.spotreby.cz.
[2] Dostupné na http://www.spritmonitor.de.
[3] Statistiky uvedené na internetových stránkách serverů dne 21. dubna 2011.

aCar

Na Android Marketu je dostupná aplikace aCar[4] umožňující sledování a správu provozu automobilů. Podporuje široké spektrum jednotek a měn, statistik a grafů. Placená verze dále nabízí rozšířené možnosti importu a exportu dat.

Nevýhodou je absence české lokalizace a celkově uživatelské rozhraní aplikace může (kvůli velkému počtu funkcí) uživateli působit nepřehledně.

3.1.2 Základní požadavky na aplikaci

Po prozkoumání již existujících řešení je důležité stanovení požadavků na funkčnost aplikace, na jejichž základě bude vytvořen model databáze – viz 3.2.

Aplikace podporuje více uživatelských profilů s možností zabezpečení heslem. Každý uživatel může spravovat provoz svých automobilů. Provoz automobilu se skládá z evidence *tankování* paliva, návštěv *servisů* a *poznámek*. Uživatel má možnost ke každému tankování přiřadit *profil jízdy*, který definuje způsob jízdy mezi posledním a aktuálním tankováním – typ pneumatik, míra naložení auta, styl řízení, použití příslušenství ovlivňujících spotřebu auta a další. Existují dva typy profilu jízdy:

- **Nepojmenovaný** – profil jízdy nemá název a je svázán s právě jedním tankováním. Uživatel jej vytváří spolu s tankováním.
- **Pojmenovaný** – profil jízdy má přiřazen název a je znovu použitelný pro více tankování. Jeho účelem je rychlejší zadávání profilu jízdy při opakujících se událostech – například při pravidelném dojíždění nebo služebních cestách.

Provoz automobilu může být *filtrován* na základě několika kritérií – typu provozu (tankování, servis, poznámka), spotřeby, profilu jízdy a stavu tachometru. *Statistiky provozu* jsou počítány s přihlédnutím k aktuálně zvolenému filtru. Hlavními položkami statistik jsou průměrná spotřeba, finanční náklady a emise CO_2. Aplikace dokáže zobrazovat upozornění na nadcházející servisní prohlídky a státní technické kontroly.

Aplikace podporuje jednotky fyzikálních veličin standardně používaných v Evropě (kilometry, litry) a ve Spojených státech amerických (míle, galony[5]). Lze také využít různé měny – česká koruna, euro, libra a dolar. Při změně měny má uživatel možnost zadat kurz pro přepočet. Aplikace je lokalizována do českého a anglického jazyka.

3.2 Konceptuální model databáze

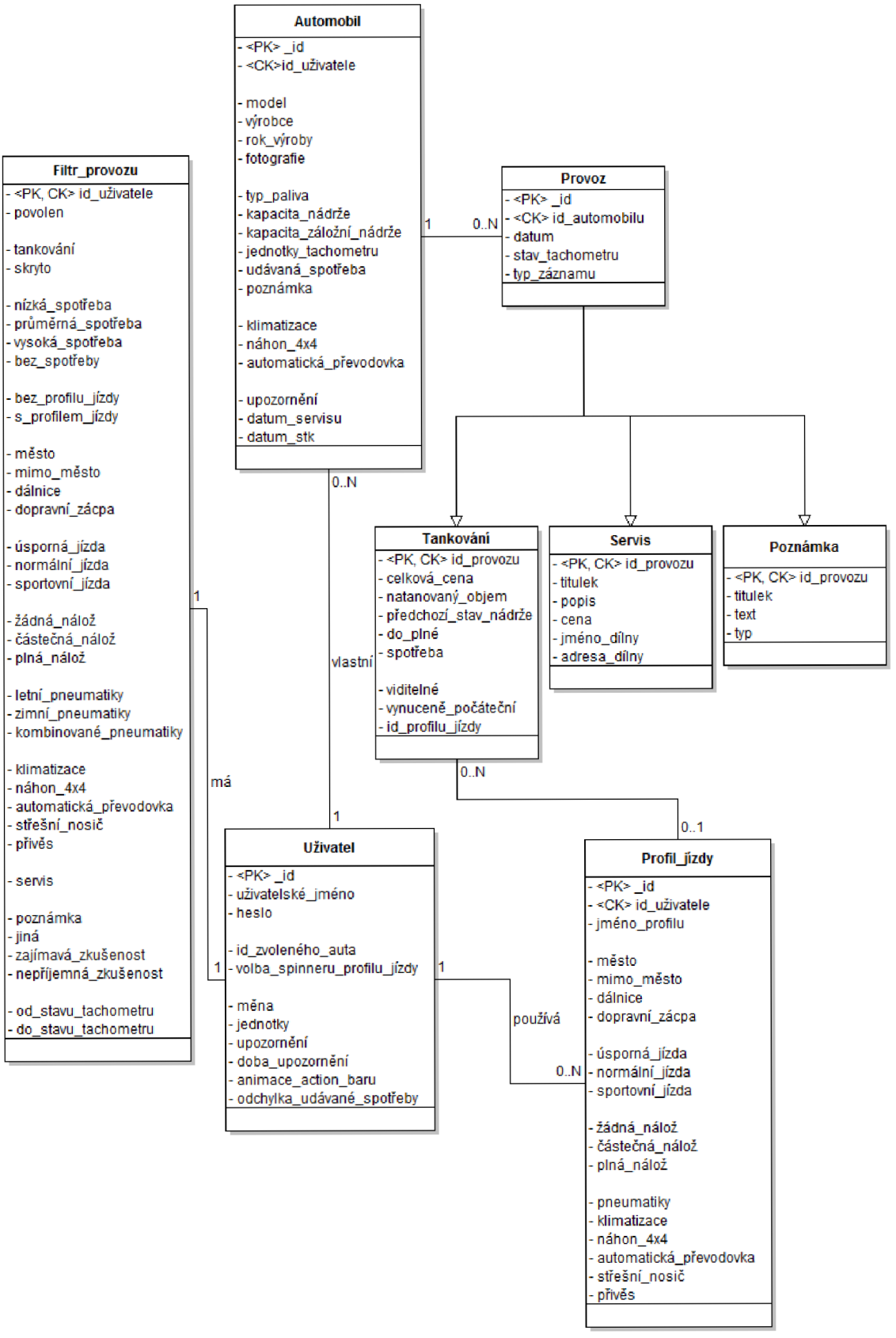
Jelikož Android disponuje plnou podporou výkonné souborové relační databáze *SQLite*, aplikace ji využije pro uložení perzistentních dat.

Schéma databáze je třeba nejprve namodelovat. K tomuto účelu byl zvolen Entity-relationship model, jelikož výsledný Entity-relationship diagram (dále jen diagram) lze snadno převést na schéma relační databáze. Diagram byl vytvořen na základě požadavků na aplikaci, viz 3.1.2. Výsledný diagram je uveden na obrázku 3.1.

Diagram obsahuje entitní množiny *Uživatel*, *Automobil*, *Provoz*, *Profil jízdy* a *Filtr provozu*. Jelikož uživatel může vlastnit několik automobilů, zatímco automobil má právě

[4] Další informace a možnost stažení na Android Marketu – https://market.android.com/details?id=com.zonewalker.acar.

[5] U.S. wet galon používaný v USA pro tekutiny [16].



Obrázek 3.1: Entity-relationship diagram databáze aplikace.

jednoho vlastníka, je mezi nimi vztah $1 : N$. Automobil smí mít evidován provoz, avšak každá položka provozu se vztahuje k právě jednomu automobilu. Mezi entitními množinami automobilu a provozu je opět vztah $1 : N$. Položkou provozu je vždy tankování, servis nebo poznámka, což je modelováno pomocí specializace. K tankování může být přiřazen maximálně jeden profil jízdy. Naopak profil jízdy smí být využit u více tankování. Vztah mezi tankováním a profilem jízdy je tedy $N : 1$. Uživatel může používat mnoho profilů jízdy. Každý profil jízdy má právě jednoho uživatele, což vede ke vztahu $1 : N$. Všichni uživatelé mají svůj vlastní filtr provozu, z čehož vyplývá vztah $1 : 1$.

Atributy entitní množiny Uživatel jsou přihlašovací udaje a uživatelské nastavení aplikace. U automobilu jsou uloženy jeho technické informace, fotografie, vybavení ovlivňující spotřebu a data pravidelných servisních prohlídek k upozornění. Provoz obsahuje důležitý atribut stav tachometru a typ záznamu určující, zda se jedná o tankování, servis nebo poznámku. Tankování nese informace o doplnění paliva, již vypočtenou spotřebu, příznaky viditelnosti a počátečního tankování (pokud byla přerušena posloupnost tankování) a odkaz na profil jízdy. Atributy profilu jízdy jsou název profilu a příznaky definující styl jízdy. Filtr provozu obsahuje příznaky uplatňující se při jeho aplikaci.

3.3 Uživatelské rozhraní

Návrh uživatelského rozhraní začal vytvářením nákresů na papír. Na základě nákresů bylo rozhraní deklarováno v aplikaci (viz 4.2) a poté odladěno na virtuálních strojích s verzemi Androidu 1.6 a 2.1-update1 a rozlišením displejů QVGA, HVGA a WVGA854[6]. Také bylo použito zařízení HTC Hero s Androidem 2.1-update1 a HVGA displejem. Byl kladen důraz na přehlednost a pohodlné dotykové ovládání, ale také na použití moderních prvků uživatelského rozhraní Androidu – záložky, ActionBar, seznamy, galerie a nabídky QuickAction. Tyto prvky by měly přispět k lepší přehlednosti aplikace a intuitivnímu ovládání. Rozhraní je přizpůsobeno pro vertikální i horizontální natočení obrazovky.

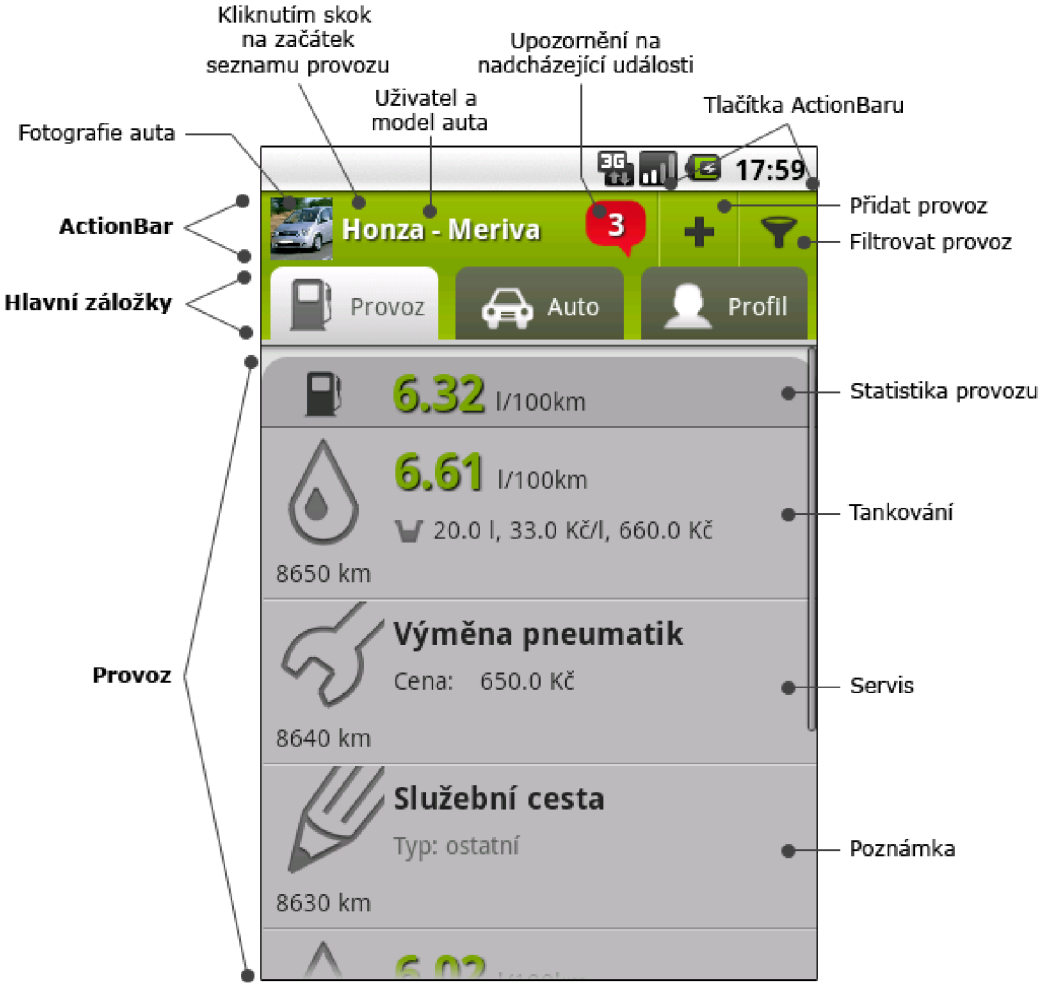
Aplikace je laděná do neutrálních šedých barevných odstínů. Výrazné prvky, jako ActionBar a formulářová tlačítka, mají zelenou barvu inspirovanou logem Androidu. Vzhled ikon napříč aplikací je převzat nebo inspirován systémovými ikonami Androidu. To podporuje intuitivnost aplikace, jelikož uživatel přichází do kontaktu s ikonami, jejichž vzhled a význam je mu již důvěrně znám. U ikon profilu jízdy, kde jejich význam nemusí být na první pohled patrný, má uživatel možnost dlouhým stiskem nad ikonou zobrazit nápovědu ve formě QuickAction, viz 3.3.5.

Hlavní část uživatelského rozhraní tvoří obrazovka se třemi záložkami a ActionBarem, viz obrázek 3.2. Princip záložek byl zvolen pro docílení přehledné a rychlé navigace v hlavní části aplikace. Jejich účelem je sjednocení důležitých částí aplikace do jedné obrazovky. Jednotlivé záložky budou popsány v nadcházející části textu. ActionBar, jehož struktura je popsána na obrázku 3.2, slouží k zobrazení důležitých informací a k rychlému a pohodlnému provádění častých úkonů. Obsahuje obrázek zvoleného automobilu, jméno uživatele, model zvoleného automobilu a sadu tlačítek, jejichž funkčnost se různí podle aktuálně zobrazené záložky. Dále se v něm zobrazuje upozornění na nadcházející události, viz 3.3.4.

3.3.1 Záložka Provoz

Na obrázku 3.2 je popsáno uživatelské rozhraní záložky Provoz, která je tvořena seznamem položek provozu seřazených podle stavu tachometru v sestupném pořadí. Existují tři typy

[6] QVGA 320×240, HVGA 480×320 a WVGA854 854×480 pixelů.

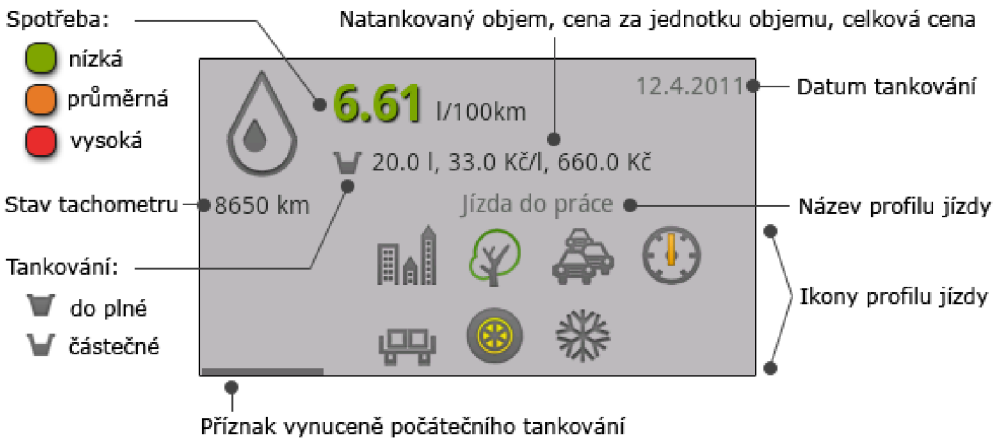


Obrázek 3.2: Uživatelské rozhraní hlavního okna a záložky Provoz.

položek – tankování, servis a poznámka. Na začátku seznamu pak leží speciální položka zobrazující statistiku provozu automobilu. Na obrázku 3.2 jsou položky provozu v zavřené podobě. Každou položku je možné kliknutím rozevřít a zobrazit tak další informace.

Rozevřená položka tankování je popsána na obrázku 3.3. Důležitým údajem je spotřeba. Proto je zvýrazněna velkými čísly a je zbarvena podle toho, zda je nižší, vyšší nebo v toleranci vůči udávané průměrné spotřebě automobilu. Dále je zde stav tachometru, natankovaný objem paliva, cena za jednotku objemu paliva, celková cena, typ tankování (zda byla nádrž zaplněna zcela nebo jen částečně), příznak počátečního tankování a konečně název a ikony zvoleného profilu jízdy. V rozevřené položce servisu je zobrazen stav tachometru, titulek, popis, datum, cena a jméno a adresa autoservisu. V rozevřené položce poznámky nalezneme stav tachometru, titulek, datum, text a typ poznámky.

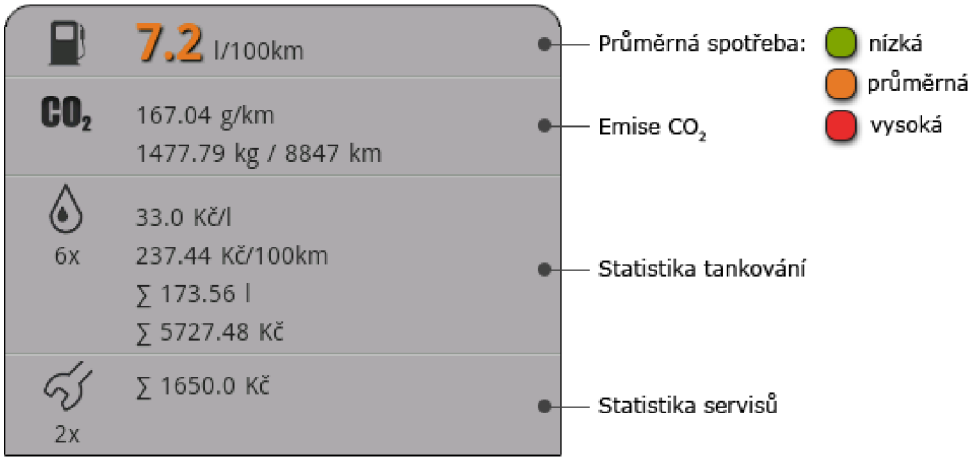
Jak již bylo uvedeno, na vrcholu seznamu se nachází položka se statistikou provozu, jež se vždy vztahuje pouze k aktuálnímu provozu v seznamu. Tedy je-li použit filtr provozu, je zobrazena statistika pouze z položek, které mu vyhovují. Pokud je položka zavřená, zobrazuje pouze průměrnou spotřebu automobilu (viz obrázek 3.2). Položka statistik v rozevřeném stavu je popsána na obrázku 3.4. Kromě spotřeby, jejíž barva je odlišena stejným způsobem jako v položce tankování výše, obsahuje vyprodukované emise CO_2 na jednotku



Obrázek 3.3: Rozevřená položka tankování.

vzdálenosti, emise CO_2 celkem za evidovanou ujetou vzdálenost[7], statistiku počtu, cen a objemů tankování a počtu a cen servisů.

Je pravěpodobné, že nejčastějšími operacemi nad seznamem provozu bude přidávání položek, popřípadě filtrování. Z toho důvodu jsou tyto dvě funkce umístěny do Action-Baru, viz obrázek 3.2. Dlouhým stiskem položky provozu je vyvolána nabídka QuickAction umožňující její úpravu či odstranění.



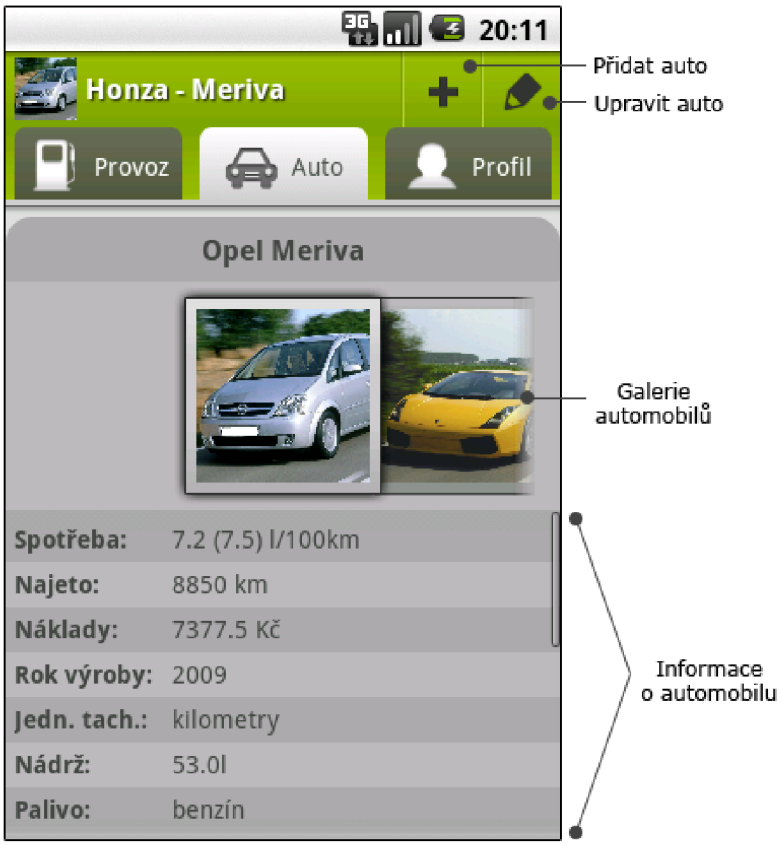
Obrázek 3.4: Rozevřená statistika provozu automobilu.

[7]Například má-li první položka seznamu provozu stav tachometru 1000 km a poslední položka 5 km, pak evidovaná ujetá vzdálenost je 995 km.

3.3.2 Záložka Auto

Na obrázku 3.5 je popsáno uživatelské rozhraní záložky Auto, která slouží ke správě automobilů uživatele a k vybrání automobilu, jehož provoz má být zobrazen v první záložce. Skládá se z posuvné fotogalerie a informací o automobilu v ní zvoleném. Informace obsahují technické a statistické údaje automobilu.

Uživatel volí aktuální auto kliknutím na jeho fotografii v galerii, popřípadě přes menu nebo nabídku QuickAction, která také umožňuje automobil upravit nebo vymazat. Možnost přidání a úpravy automobilu je též umístěna do tlačítek ActionBaru.

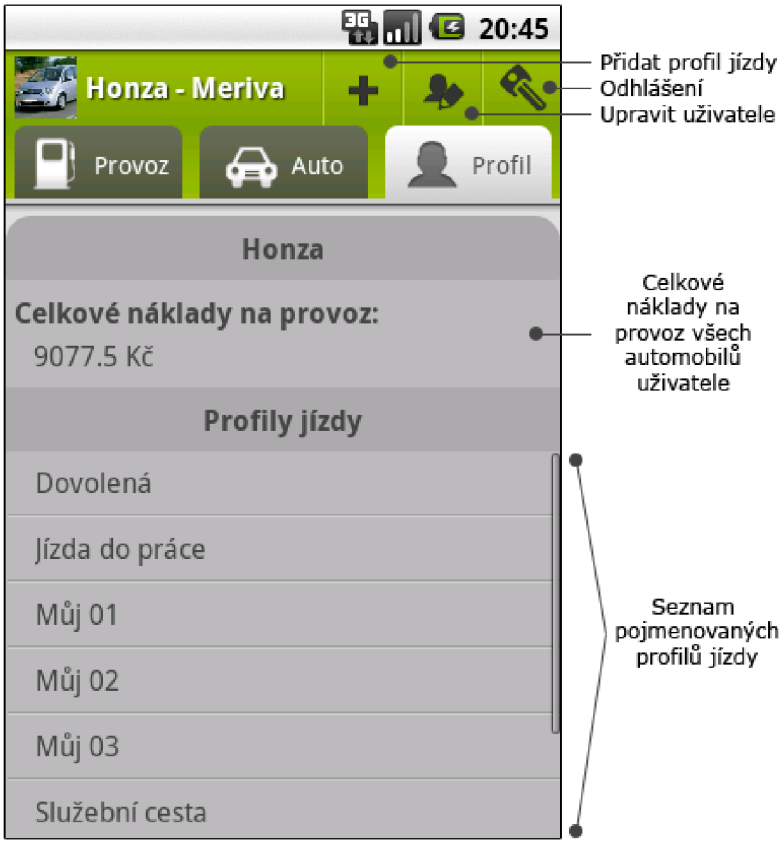


Obrázek 3.5: Uživatelské rozhraní záložky Auto.

3.3.3 Záložka Profil

Na obrázku 3.6 je popsáno uživatelské rozhraní záložky Profil. Jejím účelem je správa uživatelského účtu a pojmenovaných profilů jízdy. Lze v ní nalézt celkové náklady na provoz všech automobilů uživatele a seznam pojmenovaných profilů jízdy.

Kliknutím na položku seznamu profilů se zobrazí dialogové okno s možností úpravy, popř. odstranění profilu. Do ActionBaru je umístěna možnost přidat nový profil jízdy, upravit uživatele a odhlášení z účtu.



Obrázek 3.6: Uživatelské rozhraní záložky Profil.

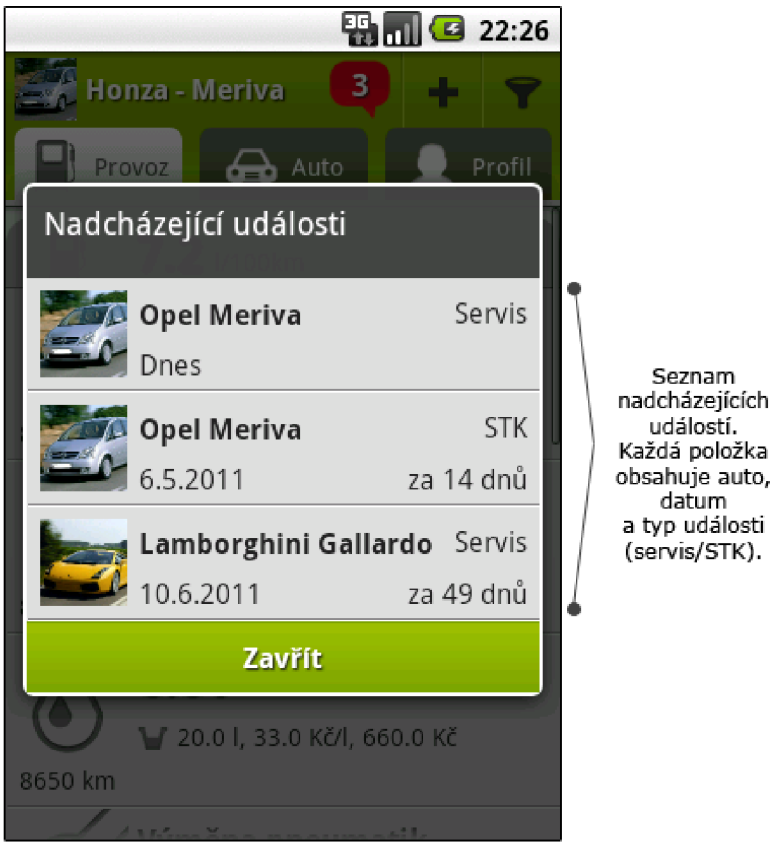
3.3.4 Nadcházející události

Aplikce umí zobrazovat upozornění na blížící se datum návštěvy servisu nebo státní technické kontroly (STK). Počet nadcházejících událostí je zobrazen v červené ikoně v ActionBaru, viz obrázek 3.2. Po kliknutí na ikonu je zobrazeno dialogové okno obsahující seznam s bližšími informacemi o událostech. Příklad okna je zobrazen na obrázku 3.7. Každá položka seznamu událostí obsahuje fotografii a název automobilu, datum události a typ – servis nebo STK. Kliknutím na položku je vyvolána nabídka QuickAction umožňující změnu data nebo zrušení události, viz obrázek 3.8.

3.3.5 Nabídky QuickAction

Nabídky QuickAction jsou dialogová okna připínající se k nějakému objektu uživatelského rozhraní. Obsahují menu položek složených z ikony a popisku. Příklad nabídky QuickAction je uveden na obrázku 3.8.

Nabídky QuickAction jsou jedním z moderních prvků uživatelského rozhraní Androidu. Jsou použity napříč aplikací v situacích, kdy je třeba vybírat z více možností. Své uplatnění také našly při zobrazení nápovědy k ikonám provozu a tlačítkům ActionBaru. Více viz 4.5.



Obrázek 3.7: Uživatelské rozhraní nadcházejících událostí.

Obrázek 3.8: Nabídka QuickAction v seznamu nadcházejících událostí.

3.4 Spotřeba automobilu

Při návrhu postupu výpočtu spotřeby automobilu byl kladen důraz na to, aby výsledná hodnota co nejvíce odpovídala realitě. K tomu je zapotřebí zjistit *ujetou vzdálenost* a *objem spáleného paliva* od předchozího tankování.

Spotřeba se pak vypočítá podle zvolených jednotek vzorcem 3.1 nebo 3.2, kde S_{eu} je spotřeba automobilu v litrech na sto kilometrů, S_{usa} je spotřeba automobilu v mílích na jeden galon paliva (MPG[8]), s je ujetá vzdálenost v kilometrech (3.1) nebo v mílích (3.2) a V_p je objem spáleného paliva v litrech (3.1) nebo galonech (3.2).

$$S_{eu} = \frac{100 * V_p}{s} \quad [\mathrm{l}/100\,\mathrm{km}] \tag{3.1}$$

$$S_{usa} = \frac{s}{V_p} \quad [\mathrm{MPG}] \tag{3.2}$$

Každá položka provozu automobilu typu tankování obsahuje mimo jiné stav tachometru, objem natankovaného paliva, příznak, zda byla nádrž zcela naplněna a pokud nebyla, je uložen také přibližný stav nádrže před tankováním. U automobilu je proto uložena kapacita nádrže, viz model databáze 3.2. Z těchto údajů lze vypočítat hodnoty proměnných s a V_p.

Nejprve je třeba zjistit stav nádrže po předchozím tankování, které je nalezeno na základě stavu tachometru aktuálního tankování. Pokud bylo předchozí tankování do plné, stav nádrže po předchozím tankování ($V_{po\_predch\_tank}$) je vypočten pomocí vzorce 3.3, jinak je použit vzorec 3.4, kde V_{nadrz_auta} je kapacita nádrže automobilu, $V_{pred\_predch\_tank}$ je stav nádrže před předchozím tankování a V_{predch_tank} je natankovaný objem při předchozím tankování.

$$V_{po\_predch\_tank} = V_{nadrz\_auta} \tag{3.3}$$

$$V_{po\_predch\_tank} = V_{pred\_predch\_tank} + V_{predch\_tank} \tag{3.4}$$

Pokud je aktuální tankování do plné, objem spáleného paliva (V_p) je vypočten za použití vzorce 3.5, jinak je aplikován vzorec 3.6, kde $V_{po\_predch\_tank}$ je stav nádrže po předchozím tankování, jehož výpočet je uveden výše, V_{nadrz_auta} je kapacita nádrže automobilu, V_{tank} je objem natankovaného paliva a V_{pred_tank} je stav nádrže před tankováním. Ujetá vzdálenost (s) je vždy vypočtena vzorcem 3.7, kde s_{tank} je stav tachometru aktuálního a s_{predch_tank} je stav tachometru předchozího tankování.

$$V_p = V_{po\_predch\_tank} - (V_{nadrz\_auta} - V_{tank}) \tag{3.5}$$

$$V_p = V_{po\_predch\_tank} - V_{pred\_tank} \tag{3.6}$$

$$s = s_{tank} - s_{predch\_tank} \tag{3.7}$$

Vypočtená *spotřeba se ukládá vždy k předchozímu tankování* a je u něj zobrazována i v uživatelském rozhraní. Je tomu tak proto, že spotřeba se vztahuje k palivu doplněnému právě v tomto tankování.

Z výše uvedených rovnic vyplývá několik podmínek, jež musí být dodrženy, aby byl výsledek korektní. Platnost podmínek je třeba ověřit před tím, než je vypočtena spotřeba a tankování uloženo do databáze. Když dvě tankování splňují následující podmínky, jsou *kompatibilní*:

[8] Z anglického *Miles Per Gallon* neboli mil na galon.

1. Natankovaný objem paliva se vždy musí vlézt do nádrže automobilu. Pokud je tankováno do plné, pak

$$V_{tank} \leq V_{nadrz\_auta} \wedge V_{tank} > 0.$$

   Pokud není tankováno do plné, pak

$$V_{tank} \leq (V_{nadrz\_auta} - V_{pred\_tank}) \wedge V_{tank} > 0.$$

2. Pokud *je* tankováno do plné, natankovaný objem musí být dostatečně veliký, aby nádrž doplnil – na základě stavu nádrže po předchozím tankování. Tedy

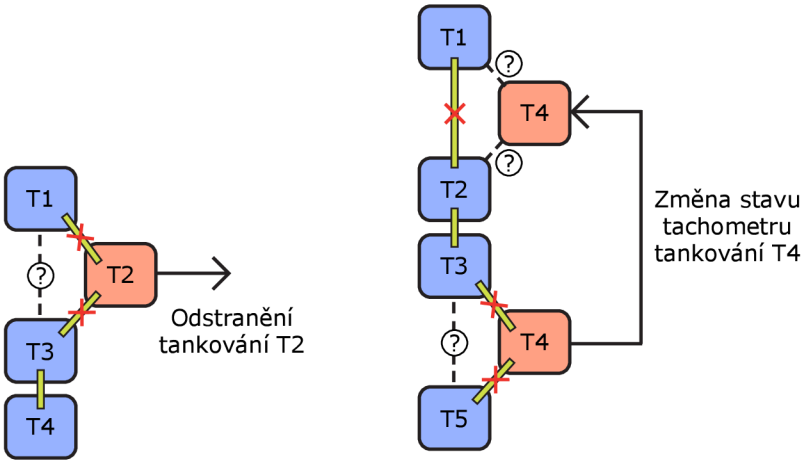
$$V_{tank} \geq (V_{nadrz\_auta} - V_{po\_predch\_tank}).$$

3. Pokud *není* tankováno do plné, musí platit

$$V_{pred\_tank} \leq V_{po\_predch\_tank} \wedge V_{pred\_tank} \leq 0.$$

   V opačném případě by v nádrži bylo před tankováním více paliva, než po předchozím tankování nebo by stav nádrže byl záporný, což není reálně možné.

Dodržení výše zníměných podmínek je klíčové nejen pro přidávání nového tankování. Problém může nastat také při odstranění některého tankování – spotřeba u předchozího se musí přepočítat a nemusí již být kompatibilní se zbylým tankováním. Podobná situace nastává i v okamžiku, kdy je tankování editováno a je mu změněn stav tachometru, čímž může být posunuto na zcela jinou pozici v posloupnosti tankování.



(a) Odstranění tankování.

(b) Editace tankování.

Obrázek 3.9: Příklad operací nad položkou tankování.

Na obrázku 3.9a je znázorněno *odstranění tankování* T2 a vliv, jaký to může mít na ostatní položky tankování. Zelený pruh spojující tankování značí jejich kompatibilitu a korektně vypočtenou spotřebu. Při vymazání tankování T2 jsou přetrhnuty vazby mezi T1 a T2 a mezi T2 a T3. Mělo by vzniknout nové spojení mezi T1 a T3, které ale nemusí být kompatibilní. Proto je nutné před vymazáním položky T2 zkontrolovat vzájemnou kompatibilitu T1 a T3. Pokud jsou tankování *nekompatibilní*, je na uživateli, aby rozhodl, která z akcí se má provést:

- Zrušit akci a tankování neodstraňovat.
- Označit tankování jako *skryté*. V tomto případě je pouze nastaven příznak viditelnosti tankování na nepravdu, což umožní skrytí tankování pomocí filtru provozu. Všechno ostatní zůstává nezměněno.
- Tankování bude vymazáno, ale *přetrhne se posloupnost* na sebe navazujících tankování. V tomto případě by položce T1 byl nastaven příznak počátečního tankování, což způsobí, že T1 se bude chovat, jakoby mu nepředcházelo žádné tankování.

  Tuto volbu může uživatel také využít, když zapomene evidovat několik tankování – pouze nastaví novému tankování počáteční příznak a může pokračovat v zadávání dalších tankování. Výpočet statistiky provozu je vyřešen tak, aby jej tyto mezery neovlivnily.

Na obrázku 3.9b je uveden příklad *editace položky tankování* T4, která způsobí přesun položky na jinou pozici v posloupnosti na sebe navazujících tankování. V tomto případě je situace o trochu komplikovanější. Je třeba ověřit kompatibilitu položek na původním místě, tankování T3 a T5, a také na novém umístění, tedy mezi tankováním T1 a T4 a mezi T4 a T2. Vazba mezi T1 a T2 je přetrhnuta. Pokud některé z ověření kompatibility selže, uživateli není dovoleno úpravu položky provést.

Přidávání nového tankování je řešeno stejně jako editace, jen není zapotřebí kontrolovat kompatibilitu položek v předchozím umístění, jelikož takové místo neexistuje.

3.5 Emise CO_2

Postup výpočtu emisí oxidu uhličitého byl převzat z webu agentury pro ochranu životního prostředí EPA [2]. Pro ověření hodnot byly použity chemické tabulky [12].

Hlavní činitel ovlivňující emise[9] CO_2 je množství uhlíku obsažené v palivu. Jelikož míra uhlíku v palivu není vždy stejná, pro výpočet emisí je použita průměrná hodnota:

- Množství uhlíku v jednom galonu *benzínu* je $2,421$ gramu.
- Množství uhlíku v jednom galonu *nafty* je $2,778$ gramu.

Na míru uhlíku v palivu musí být aplikován oxidační faktor, určující množství uhlíku, které je ve skutečnosti zoxidováno na CO_2. Pro všechny ropné produkty je oxidační faktor roven $0,99$, tzn. $99\,\%$ uhlíku je zoxidováno na CO_2, zatímco $1\,\%$ zůstáná nezoxidováno.

Dále je nutné zjistit relativní molekulovou hmotnost (M_r) uhlíku a CO_2. M_r je možno vypočítat pomocí relativní atomové hmotnosti (A_r), kterou lze nalézt v periodické soustavě prvků, viz [12]. Hodnota M_r je pak rovna součtu A_r všech atomů v molekule. Postup výpočtu M_r uhlíku a oxidu uhličitého je uveden v rovnicích 3.8, kde A_{r_C} je relativní atomová hmotnost uhlíku, A_{r_O} je relativní atomová hmotnost kyslíku, M_{r_C} je relativní molekulová hmotnost uhlíku a $M_{r_{CO_2}}$ je relativní molekulová hmotnost oxidu uhličitého.

$$\begin{aligned} A_{r_C} &= 12,001 \quad [12] \\ A_{r_O} &= 15,999 \quad [12] \\ M_{r_C} &= 1 * A_{r_C} \doteq 12 \\ M_{r_{CO_2}} &= 1 * A_{r_C} + 2 * A_{r_O} \doteq 44 \end{aligned} \tag{3.8}$$

[9] Uvolňování látek znečišťujících životní prostředí, nejčastěji spalováním [14].

Výsledná hodnota emisí na jeden galon paliva je vypočtena rovnicí 3.9, kde E_{CO_2} jsou emise CO_2, m_C je množství uhlíku obsaženém v palivu v gramech a M_{r_C} a $M_{r_{CO_2}}$ jsou relativní molekulové hmotnosti uvedené výše.

$$E_{CO_2} = m_C * 0,99 * \frac{M_{r_{CO_2}}}{M_{r_C}} \quad [\text{g/gal.}] \tag{3.9}$$

Pro benzín vychází emise 8 788 g/gal., pro naftu 10 084 g/gal. Z těchto hodnot lze pomocí převodů jednotek odvodit vzorce pro výpočet emisí na základě spotřeby automobilu. Vzorce se liší podle typu paliva a jednotek, s nimiž pracují. Při použití evropských jednotek jsou aplikovány rovnice 3.10 pro benzín a 3.11 pro naftu. E_{CO_2} je hodnota emisí CO_2 v gramech na kilometr a $S_{l/100km}$ je spotřeba automobilu v litrech na sto kilometrů.

$$E_{CO_2} \doteq 23,215 * S_{l/100km} \quad [\text{g/km}] \tag{3.10}$$

$$E_{CO_2} \doteq 26,639 * S_{l/100km} \quad [\text{g/km}] \tag{3.11}$$

Pokud jsou nastaveny jednotky používané v USA, jsou uplatněny vzorce 3.12 pro benzín a 3.13 pro naftu. E_{CO_2} je hodnota emisí CO_2 v uncích na jednu míli a S_{MPG} je spotřeba automobilu v mílích na galon paliva.

$$E_{CO_2} \doteq 19,374 * \frac{16}{S_{MPG}} \quad [\text{oz/mi}] \tag{3.12}$$

$$E_{CO_2} \doteq 22,231 * \frac{16}{S_{MPG}} \quad [\text{oz/mi}] \tag{3.13}$$

Do statistik jsou také zahrnuty emise za ujetou vzdálenost. Pro evropské jednotky je použit vzorec 3.14, kde e_{CO_2} jsou emise CO_2 v gramech na kilometr (viz výše), s je ujetá vzdálenost v kilometrech a E_{CO_2} je množství emisí vyprodukováných po ujetí s kilometrů v kilogramech. Pro jednotky užívané v USA je aplikován vzorec 3.15, kde e_{CO_2} jsou emise CO_2 v uncích na míli (viz výše), s je ujetá vzdálenost v mílích a E_{CO_2} je množství emisí vyprodukováných za ujetí s mil v librách.

$$E_{CO_2} \doteq \frac{e_{CO_2} * s}{1000} \quad [\text{kg}] \tag{3.14}$$

$$E_{CO_2} \doteq \frac{e_{CO_2} * s}{16} \quad [\text{lb}] \tag{3.15}$$

Kapitola 4

Implementace

Aplikace je implementována v programovacím jazyce Java za použití SDK[1] pro platformu Android. Minimální verze SDK je 4, tedy aplikace může být spuštěna v Androidu 1.6 nebo vyšším. Aplikace nevyžaduje udělení žadných oprávnění, kromě ovládání vibrací. Vibrační odezva je použita při skoku na začátek seznamu provozu kliknutím na ActionBar (viz obrázek 3.2).

4.1 Databáze

Konceptuální návrh databáze (3.2) je implementován třídou `MyCarDatabase`, odvozenou od speciální třídy `SQLiteOpenHelper`. Spravuje vytváření databáze a umožňuje získání instance třídy `SQLiteDatabase`, pomocí které lze provadět databázové dotazy. Data jsou uložena v jediném databázovém souboru `my_car.db`.

4.2 Deklarace uživatelského rozhraní

Všechny obrazovky uživatelského rozhraní jsou vytvořeny deklarativně jazykem XML a uloženy v samostatných souborech. Vývojové prostředí Androidu takto umožňuje oddělit uživatelské rozhraní od aplikační logiky, kde jsou komponenty GUI pouze instanciovány, pokud je vyžadováno dynamické chování, např. změna popisku nebo reakce na stisk tlačítka. Deklarace komponenty GUI se skládá z názvu elementu, jenž identifikuje typ komponenty (třídu odvozenou od třídy `View`, viz 2.2.1) a z atributů pro inicializaci jejích vlastností.

Ve výpisu 4.1 je uveden úryvek kódu souboru `form_buttons.xml`, ve kterém jsou deklarována tlačítka pro uložení a zrušení formulářů, jež jsou použita napříč aplikací. Atributem `id` je tlačítko provázáno s kódem aplikace, `text` definuje popisek (lokalizovaný řetězec), `layout_width` a `layout_height` určují šířku a výšku tlačítka, `layout_weight` udává, jaký poměr obrazovky bude tlačítko zabírat, `textStyle`, `textColor` a `textSize` nastavují styl, barvu a velikost textu popisku a `shadowColor`, `shadowDx`, `shadowDy` a `shadowRadius` přidávají popisku stín – barvu, umístění a rozsah. Poslední atribut `background` přiřazuje k pozadí tlačítka selektor. Selektor je XML soubor, který definuje různá pozadí podle toho, v jakém stavu se tlačítko nachází – tlačítko pak mění barvu podle toho, zda je označeno či stisknuto. Na obrázku 4.1 je výsledná podoba takto deklarovaných formulářových tlačítek.

[1] *Software Development Kit* – soubor nástrojů k vývoji aplikací pro určitou platformu.

```
...
<Button
   android:id="@+id/save_btn"
   android:text="@string/save_btn_label"
   android:layout_width="wrap_content"
   android:layout_height="40dip"
   android:layout_weight="0.5"
   android:textStyle="bold"
   android:textColor="@color/white_cl"
   android:textSize="16sp"
   android:shadowColor="@color/dark_cl"
   android:shadowDx="1.2"
   android:shadowDy="1.2"
   android:shadowRadius="0.7"
   android:background="@drawable/button_selector">
</Button>
...
```

Výpis 4.1: Deklarace tlačítka.

Obrázek 4.1: Formulářová tlačítka.

4.3 Seznam provozu

Seznamy v systému Android jsou tvořeny grafickou komponentou `ListVew`, k níž je přiřazen adaptér. Adaptér je třída, která se stará o datovou vrstvu seznamu, zejména o vytváření položek, jež jsou v `ListView` zobrazeny. Android má řadu již hotových adaptérů, například `ArrayAdapter<T>` vytvářející položky `ListView` ze seznamu řetězců nebo `SimpleCursorAdapter` umožňující jednoduchý *binding* mezi položkami `ListView` a sloupci databáze a mnohé další.

Požadavky na vzhled a funkčnost položek seznamu provozu při návrhu (viz 3.3.1) byly nad rámec toho, co Android standardně poskytuje. Proto byl vytvořen specializovaný adaptér `TrafficListAdapter` odvozený od základního adaptéru `BaseAdapter`.

`TrafficListAdapter` spravuje načítání informací o provozu z databáze, vytváření položek provozu a jejich rozevírání a zavírání. Vytvořenou položku provozu představuje třída `TrafficView`. Podle parametrů konstruktoru pozná, o jaký typ položky se jedná – tankování, servis nebo poznámka – a vytvoří instanci grafického rozhraní daného typu položky deklarovaného v samostatném XML souboru. Tuto instanci poté naplní předanými informacemi z databáze.

Třída `TrafficListAdapter` obsahuje seznam instancí třídy `TrafficView` spolu s dalšími informacemi – stav tachometru (pro správné řazení), databázové ID a typ položky a pří-

znak, zda je položka rozevřena. Vytváření a přidávání do seznamu – načítaní dat z databáze a vytváření položek provozu – je prováděno na pozadí, takže vlákno s uživatelským rozhraním není blokováno. K tomuto účelu byla použita třída `AsyncTask`, viz 4.4.

Pokud komponenta `ListView` potřebuje vykreslit položku, zavolá metodu `getView()` svého adaptéru. Metodě je mimo jiné předána pozice položky v seznamu. Podle této pozice metoda vyhledá odpovídající položku a vrátí instanci třídy `TrafficView`. Vrácená položka je pak vykreslena na obrazovku.

Seznam provozu je načítán a zobrazován postupně. Nejprve je načteno prvních dvacet položek provozu. Když se uživatel dostane na poslední zobrazenou položku, je spuštěno načítaní dalších dvaceti položek atd.

4.4 Asynchronní úlohy

Asynchronní úloha je reprezentována třídou `AsyncTask`, viz 2.4.2. Umožňuje pohodlné vykonání operací na pozadí. Je použita napříč aplikací v místech, kde jsou prováděny náročné databázové dotazy – načítání položek provozu, výpočet statistiky provozu, načítání informací o automobilu (obsahují také statistiky) a zjišťování celkových nákladů na provoz všech automobilů uživatele. Jsou využity tři speciální metody této třídy:

- `doInBackground()` – hlavní metoda, jejíž kód je vykonán v samostatném vlákně. V této metodě jsou prováděny databázové dotazy.
- `onPreExecute()` – kód metody je proveden v hlavním vlákně před tím, než je spuštena práce na pozadí – metoda `doInBackground()`. Je použita pro zobrazení *progress baru*, který dává uživateli vědět o tom, že jsou načítána data.
- `onPostExecute()` – kód metody je proveden v hlavním vlákně poté, co je dokončena metoda `doInBackground()`. Metoda je použita pro odstranění *progress baru* a publikaci výsledků práce na pozadí v uživatelském rozhraní.

4.5 Nabídky QuickAction

Nabídky QuickAction jsou tvořeny třemi třídami – `CustomPopupWindow`, `QuickAction` a `ActionItem`. Tyto třídy byly převzaty z [10]. Autor na svém webu uvádí, že jejich šíření je možné pod licencí BSD. Třída `QuickAction` je odvozena od `CustomPopupWindow` a zajišťuje zobrazení nabídky. Pracuje s třídou `ActionItem`, jež představuje položku nabídky – obsahuje popisek, ikonu a `OnClickListener` pro vyvolání akce při stisku.

Ve třídě `QuickAction` bylo provedeno několik úprav. Byl modifikován výpočet výšky nabídky v horizontálním natočení displeje, byla přidána funkcionalita, která zajišťuje zobrazení nabídky vertikálně nebo horizontálně podle aktuálního natočení obrazovky. Dále byl přidán atribut sloužící k zapamatování pozice v seznamu. Je-li nabídka QuickAction vyvolána nad objektem umístěným v některém ze seznamů, je tento atribut nastaven a později využit k manipulaci s položkou, například pokud uživatel zvolí vymazání položky. Poslední změna spočívala v opravě chybného zobrazení, pokud je volající aktivita zobrazena jako dialogové okno, například ativita `ActiveEvents` implementující nadcházející události (viz obrázky 3.7 a 3.8).

Kapitola 5

Závěr

Cílem práce bylo vytvořit aplikaci pro správu nákladů na provoz automobilu. Kapitola se zabývá shrnutím, v jaké míře byly splněny vytyčené cíle práce a jaká je možná budoucnost projektu z hlediska rozšíření a distribuce.

5.1 Přínos práce

Největším přínosem této práce pro mě bylo seznámení s programovacím jazykem Java a zejména s SDK pro platformu Android a možnostmi, jež programátorovi nabízí – především naučení se pracovat s databází a provádění náročných dotazů tak, aby nebyla blokována odezva uživatelského rozhraní přikládám největší hodnotu, jelikož je tato dovednost v dnešní době informačních systémů nutná při tvorbě téměř jakékoliv aplikace. Dále jsem si osvojil vytváření uživatelského rozhraní a to nejen jeho návrh, ale hlavně jeho implementaci deklarativně jazykem XML a celkově moderní implementaci aplikace, kde je aplikační logika oddělena od uživatelského rozhraní a od ostatních zdrojů – například lokalizované řetězce, ikony a další.

Také jsem musel proniknout do problematiky výpočtu spotřeby automobilu, která sama o sobě není nijak náročná. Problémy nastaly až při návrhu editace a mazání položek tankování. Jde sice o operace, které pravděpodobně uživatel často nevyužije, přesto považuji za nutnost, aby je aplikace podporovala.

5.2 Budoucnost projektu

Ačkoliv cíle vytyčené v požadavcích na aplikaci (viz 3.1.2) byly splněny, je zde stále spousta prostoru pro vylepšování a rozšiřování funkcionality. Mohla by být přidána položka pro pravidelný a nepravidelný výdaj netýkající se servisu, například povinné ručení, havarijní pojištění nebo nákup oleje, pneumatik a podobně. Pravidelné výdaje by pak mohly být zobrazovány v upozorněních na nadcházející události. Také by bylo užitečné, kdyby upozornění zobrazovalo povinné servisní prohlídky podle specifikace automobilu, například po ujetí 15 000 kilometrů nebo po jednom roce (podle toho, co nastane dřív).

Aplikace obsahuje jen základní statistiky – spotřebu, emise CO_2, finanční náklady apod. Jedním z možných rozšíření by určitě mohlo být přidání statistik ve formě grafů, například vývoj spotřeby podle najetých kilometrů. Dalším rozšířením může být možnost zálohování, tedy export a import dat.

Uživatelé Androidu často používají widgety, proto by bylo vhodné, kdyby ho aplikace poskytovala. Widget by zobrazoval průměrnou spotřebu auta, upozornění na nadcházející servisní prohlídky a umožňoval by rychlé zadání tankování, bez nutnosti nejprve spouštět aplikaci. Pokud by si uživatelé aplikaci oblíbili, mohla by být přídána podpora více jednotek, měn a jazykových mutací.

Dále je v plánu aplikaci umístit na Android Market, kde mohou vývojáři distribuovat své aplikace uživatelům. Při registraci je nutné zaplatit poplatek 25 dolarů. Běžná praxe je taková, že aplikace je rozdělena na dvě verze. Odlehčená verze má omezenou funkčnost, případně obsahuje reklamy a je zdarma, plná verze je pak zpoplatněna, přičemž cena většiny aplikací se pohybuje v řádu několika desítek korun. Tady ovšem může být překážkou omezení Android Marketu, který prozatím[1] českým programátorům nepovoluje prodávání aplikací za peníze. Zpřístupněna je pouze distribuce bezplatných aplikací. Google na svém webu tvrdí, že na rozšíření portfólia zemí se neustále pracuje, což lze z vlastní zkušenosti potvrdit, jelikož nedávno byla českým uživatelům zpřístupněna možnost nakupování aplikací, předtím bylo možné stahovat pouze bezplatné aplikace. Situace bude nějakou dobu monitorována a pokud Google v dohledné době Českou republiku do podporovaných zemí nepřidá, uvažuji o zpřístupnění plné verze zadarmo, jelikož mě velmi zajímá, jak budou na aplikaci reagovat běžní uživatelé.

[1] Dne 29. dubna 2011.

Literatura

[1] BURNETTE, E.: *Hello, Android.* The Pragmatic Programers, 2010, 300 s., iSBN 1-934356-56-5.

[2] EPA: *Emission Facts: Average Carbon Dioxide Emissions Resulting from Gasoline and Diesel Fuel* [online]. 2011-04-12 [cit. 2011-04-25]. URL http://www.epa.gov/OTAQ/climate/420f05001.htm

[3] GOOGLE: *Tasks and Back Stack* [online]. 2011-03-15 [cit. 2011-03-22]. URL http://developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html

[4] GOOGLE: *Activities* [online]. 2011-03-15 [cit. 2011-03-23]. URL http://developer.android.com/guide/topics/fundamentals/activities.html

[5] GOOGLE: *Processes and Threads* [online]. 2011-03-15 [cit. 2011-03-23]. URL http://developer.android.com/guide/topics/fundamentals/processes-and-threads.html

[6] GOOGLE: *Reference: AsyncTask* [online]. 2011-03-15 [cit. 2011-03-23]. URL http://developer.android.com/reference/android/os/AsyncTask.html

[7] GOOGLE: *Services* [online]. 2011-03-15 [cit. 2011-03-23]. URL http://developer.android.com/guide/topics/fundamentals/services.html

[8] GOOGLE: *What is Android?* [online]. prosinec 2010 [cit. 2011-01-22]. URL http://developer.android.com/guide/basics/what-is-android.html

[9] GOOGLE: *Application Fundamentals* [online]. prosinec 2010 [cit. 2011-01-30]. URL http://developer.android.com/guide/topics/fundamentals.html

[10] Lorensius, W. L.: *How to Create QuickAction Dialog in Android* [online]. 2010-07-12 [cit. 2011-02-12]. URL http://www.londatiga.net/it/how-to-create-quickaction-dialog-in-android/

[11] MEIER, R.: *Professional Android^{TM} 2 Application Development.* Wiley Publishing, Inc., 2010, 543 s., iSBN 987-0-470-56552-0.

[12] MIKULČÁK, J.; KLIMEŠ, B.; ŠIROKÝ, J.; aj.: *Matematické, fyzikální a chemické tabulky pro střední školy.* Prometheus, 2004, 206 s., iSBN 80-85849-84-4.

[13] WIKIPEDIA: *BSD* [online]. 2010-09-22 [cit. 2011-01-22].
URL http://cs.wikipedia.org/wiki/BSD

[14] WIKIPEDIA: *Emise* [online]. 2010-11-21 [cit. 2011-04-25].
URL http://cs.wikipedia.org/wiki/Emise

[15] WIKIPEDIA: *Android (operating system)* [online]. 2011-01-21 [cit. 2011-01-21].
URL http://en.wikipedia.org/wiki/Android_(operating_system)

[16] WIKIPEDIA: *Galon* [online]. 2011-04-17 [cit. 2011-04-21].
URL http://cs.wikipedia.org/wiki/Galon

Příloha A

Obsah CD

K bakalářské práci je přiloženo CD obsahující adresáře:

- `app` – zdrojové soubory programu, jež mají typickou strukturu projektu aplikace pro Android, instalační `apk` balíček a programová dokumentace vygenerovaná nástrojem *javadoc*.
- `manual` – uživatelská příručka popisující ovládání aplikace formou webové prezentace.
- `thesis` – technická zpráva ve formátu `pdf` a zdrojové soubory k jejímu vytvoření.