Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Operační systém Google Android

Petr Koula

Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Akademický rok 2010/2011

ZADÁNÍ DIPLOMOVÉ PRÁCE

Petr Koula

obor Informatika

Vedoucí katedry Vám ve smyslu Studijního a zkušebního řádu ČZU v Praze čl. 17 odst. 2 určuje tuto diplomovou práci.

Název práce: **Operační systém Google Android**

Osnova diplomové práce:

1. Úvod
2. Cíl práce a metodika
3. Teoretická východiska
4. Vlastní řešení
5. Výsledky a diskuse
6. Závěr
7. Seznam použitých zdrojů
8. Přílohy

Rozsah hlavní textové části: 60 - 80 stran

Doporučené zdroje:

MEIER R. Professional Android Application Development, Wrox 2008, 432str., ISBN 0470344717.
ROGERS, R., LOMBARDO, J. Android Application Development,
1st Edition. Cambridge: O`Reilly Media, Inc. 2009, 336s., ISBN 978-0-596-52147-9.
TOPLEY, Kim. J2ME v kostce - Pohotová referenční příručka. Praha: GRADA, 2004,536s., ISBN 80-247-0426-9.
Zdroje na internetu:
Android - http://code.google.com/android/
Android SDK for Windows - http://code.google.com/android/documentation.html
Symbian Developer: home - http://www.symbian.com/developer/index.html
Windows Mobile Developer Center - http://msdn2.microsoft.com/en-us/windowsmobile/default.aspx

Vedoucí diplomové práce: **Ing. Čestmír Halbich, CSc.**

Termín odevzdání diplomové práce: duben 2011

L.S.

..... Vedoucí katedry

..... Děkan

V Praze dne: 28. 2. 2011

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Operační systém Google Android" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne _____

Poděkování

Rád bych zde poděkoval vedoucímu diplomové práce panu Ing. Čestmíru Halbichovi, CSc. za vstřícné konzultace a cenné připomínky.

Operační systém Google Android

Operating system Google Android

Souhrn

Diplomová práce zahrnuje v první části komplexní popis operačního systému Google Android z pohledu jeho historie, architektury, funkcionalit a vývoje verzí. Věnuje se také porovnání platformy Android s největšími konkurenčními systémy. Druhá část je věnována praktické ukázce procesu implementace reálné aplikace, jejíž zdrojové kódy popisují použití částí Android API, přičemž jsou podrobně komentovány. Část je také věnována grafickému návrhu uživatelského rozhraní v souladu se standardy a doporučeními. Na závěr je popsán postup zkompletování aplikace a její publikace ve službě Android Market. Na operační systém Google Android je tedy pohlíženo jak ze strany pokročilého uživatele, tak ze strany vývojáře aplikací pro tuto platformu.

Summary

The thesis includes complex description of Google's Android operating system from perspective of its history, architecture, functionality and versions. It also compares Android platform with competitive systems. The second part is devoted to practical demonstration of the implementation process of the real application whose source codes describes how to use parts of the Android API and is commented in detail. The part of the text deals with graphic design of the user interface in accordance with the standards and recommendations. In conclusion, it describes how to assemble the developed application and publish it on the Android Market. The Google Android operating system is perceived from the perspective of advanced users and developers of the applications for this platform.

Klíčová slova: operační, systém, google, android, architektura, historie, porovnání, vývoj, api, příklad

Keywords: operating, system, google, android, architecture, history, comparison, development, api, example

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

1 Úvod

Nároky na osobní mobilní zařízení, která již zdaleka neplní funkci pouze mobilního telefonu, rostou každým rokem, a to stále rychlejším tempem. Stejně jako na osobní počítače, tak i na chytré telefony a osobní přenosná zařízení platí Mooreův zákon, díky němuž se stávají dominantní světovou komunikační platformou. Tato zařízení jsou mnohdy i několikanásobně výkonnější než byly superpočítače před několika roky (1). Uživatele ale neuspokojuje pouze výkonný hardware, ale především kvalitní a všestranné softwarové vybavení zařízení. Dnešní běžný uživatel využívá svůj mobilní telefon stejným způsobem jako pokročilý uživatel před několika lety. Chytré telefony slouží jako jakési terminály pro spojení se světem Internetu, sociálních sítí a různých komunikačních kanálů. Kromě tohoto jsou využívány jako navigační zařízení v autodopravě, turistice i sportu, dále jako fotoaparáty, čtečky elektronických knih či audio/video přehrávače. Všechny tyto uživatelovy požadavky kladou vysoké nároky na operační systém aplikovatelný na více či méně odlišné architektury výrobců mobilních zařízení, který samozřejmě musí zvládat obsluhu široké škály hardwarových prvků, ale také, a to se stává stále důležitějším, musí nabízet uspokojivé množství aplikací všeho druhu prostřednictvím komunity vývojářů. A zde se dostáváme k problému, který dnes velmi ovlivňuje úspěšnost či neúspěšnost operačního systému, aspirujícího na masivní používání v oblasti mobilních zařízení, a tím je podpora vývoje a distribuce aplikací.

V době, kdy na trh vstupuje nový operační systém, má na rychlost jeho rozšíření vliv právě zájem vývojářů o implementování svých aplikací na této platformě. Jednoduše řečeno, jen obtížně se masově rozšíří operační systém, pro který se jen těžko shání software. Na druhou stranu existují i specifické případy, kdy se operační systém nerozšířil díky přirozeně stoupající oblíbenosti mezi uživateli a vývojáři, ale díky hromadnému nasazení na většinu modelů výrobce, jak tomu bylo v případě finské Nokia a jejího Symbianu. Dalším specifickým případem jsou mobilní zařízení BlackBerry společnosti Research In Motion (RIM), které si získaly oblibu v oblasti byznysu, a to především díky bezkonkurenční hardwarové klávesnici a výbornému emailovému klientu.

Podporou vývoje rozumíme například veřejně dostupné a dobře zdokumentované API, vývojové prostředí (IDE), tutoriály a diskusní fóra. Oblíbeným způsobem podpory

distribuce aplikací jsou online markety nabízející zakoupení a stažení požadované aplikace přímo prostřednictvím mobilního zařízení.

Na světovém trhu mobilních zařízení se utkává v boji o uživatele pětice nejrozšířenějších operačních systémů. Jedná se o operační systém Symbian, Android, BlackBerry OS, iOS a Windows Phone (dříve Windows Mobile). I přesto, že je tato práce zaměřená na operační systém Android, popíšeme si v kapitole 3 pro srovnání i ostatní operační systémy z této pětice. Navzdory tomu, že společnost Google reálně vstoupila s Androidem na trh mobilních zařízení až v druhé polovině roku 2008, dokázal Android velice rychle zaujmout místo mezi nejrozšířenějšími mobilními platformami a jak uvidíme později, očekává se, že by v roce 2014 mohl celosvětově bojovat se systémem Symbian o první příčku (2). Právě proto, že operační systém Google Android má tak velký potenciál, stojí za to se jím detailně zabývat, popsat si jeho architekturu, srovnat jej s konkurenčními systémy, zaměřit se na vývoj aplikací a jejich distribuci a pokusit se určit směr, jakým se bude Android ubírat v budoucnu.

2 Cíl práce a metodika

Cílem této práce je poskytnout čtenáři komplexní popis operačního systému Google Android. Popíšeme si jeho vlastnosti a architekturu, srovnáme jej s konkurenčními systémy a především se zaměříme na vývoj aplikací pro tuto platformu. Ukážeme si, jakým způsobem začít s vývojem, jak nakonfigurovat vývojové prostředí a na vzorové aplikaci si popíšeme API. Tato aplikace nám pomůže zahrnout celý vývojový cyklus, od první obrazovky (aktivity) až po její zveřejnění na Android Marketu. Na názorných příkladech bude dobře pochopitelné jakým způsobem vytvářet a provazovat aktivity aplikace, jak vytvářet menu a ukládat nastavení, obohatíme aplikaci o widget a notifikace, použijeme GPS modul mobilního zařízení ke zjištění polohy, ukážeme si jakým způsobem využívat SQL databázi a nakonec propojíme aplikaci s Google Maps. Budeme postupovat jako při reálném vývoji a výsledkem bude funkční aplikace dostupná přes službu Android Market všem uživatelům. Aby vše nebylo popisováno jen v teoretické rovině, uvedeme si i části kódu. Tímto vývojem pojmeme značnou část Android API a zároveň se dozvíme, jak

standardizovat grafické prvky aplikace, aby vyhovovaly doporučením, které k systému Android společnost Google vydala.

3 Teoretická východiska

3.1 Historie Google Android

3.1.1 Počátky Androidu

V červenci 2005 se objevují zprávy, že společnost Google koupila firmu Android, Inc. za neupřesněnou cenu. Odborníci a analytici ICT trhu se shodují na tom, že Google tímto krokem míří na klíčový segment mobilních zařízení (3). Společnost Android, Inc., v té době téměř neznámá společnost, se řadila mezi tzv. „start-up“ firmy, které se snaží prorazit na trh díky vývoji nové technologie, která tvoří na trhu díru. Jejími zakladateli jsou Andy Rubin, Rich Miner, Nick Sears a Chris White. Společnost se začala specializovat na vývoj aplikací pro mobilní zařízení a přichází s konceptem mobilní platformy založené na Linuxu. V tuto dobu vstupuje do hry Google, který Android, Inc. kupuje v rámci své strategie investování do slibně se rozvíjejících společností. Google tohoto způsobu rozrůstání se do různých segmentů ICT použil již dříve a díky svým finančním možnostem si zvolil firmu přicházející v tomto segmentu s něčím novým a tu jednoduše koupil. Na druhou stranu nemusel tak investovat do počátečních analýz a vývoje a mohl se rovnou pustit do produkce, čímž šetří čas a zvětšuje náskok před konkurencí, který je ve světě IT jednou z největších konkurenčních výhod.

Zdroj: archiv autora

Po akvizici společnosti Android, Inc. s Googlem se Andy Rubin stává v Google ředitelem vývoje pro mobilní platformy a jeho tým má za úkol vyvinout flexibilní a jednoduše rozšiřitelný operační systém. V prosinci 2007 se společnosti Google, HTC, Motorola, Intel, Qualcomm, Sprint Nextel, T-mobile a NVidia spojují do uskupení Open Handset Alliance, v rámci kterého se pokoušejí definovat otevřené standardy nové mobilní

platformy. Výsledkem je operační systém Android založený na linuxovém jádře, vydaný pod licencí Apache jako otevřený systém, který je možné libovolně upravovat, přidávat součásti a aplikace a prodávat jej. (4)

Od této doby začíná boj o prvenství v uvedení nového zařízení s Androidem na trh. Jako první velkosériově vyráběné zařízení se objevuje v září 2008 oblíbený T-Mobile G1 s verzí Google Android 1.0. Ten v rozmezí několika měsíců dostává aktualizace na nové verze 1.1, 1.5 a 1.6. Android se stává oblíbeným operačním systémem a objevuje se na zařízeních dalších výrobců jako je Samsung nebo LG. V roce 2010 dokonce Google uvádí na trh svůj vlastní model Nexus One, jehož hardware pochází z dílen HTC a má se stát vlajkovou lodí Google na poli samotných mobilních zařízení. Jeho úspěch ovšem nenaplňuje očekávání, a tak se objevují spekulace o tom, že Google trh mobilních zařízení opustí a dále se bude věnovat pouze vývoji operačního systému a aplikací nad ním. Ovšem na konci roku 2010 je ohlášen nový model Nexus S, který jako první přinese novou verzi Androidu 2.3 Gingerbread. Oblíbenost Androidu také zapříčinila, že jeho nasazení expanduje také na tabletová zařízení, netbooky či dokonce televizory. Po chytrých telefonech se objevuje nový fenomén a tím jsou multidotykové tablety. Prvním velmi úspěšným zařízením tohoto typu je Apple iPad, který svým hardwarem a uživatelsky lákavým operačním systémem poměrně razantně předstihl konkurenci, která ale na sebe s odpovědí nenechá dlouho čekat a přichází s tabletem, jehož platformou je Android. Tímto zařízením je Samsung Galaxy Tab. Android ovšem není do této doby zcela připraven na tento typ zařízení a nemá dobrou podporu vysokého rozlišení a dalších funkcí, které uživatel tabletů požaduje. Proto vzniká verze Androidu 3.0, která by měla tento nedostatek plně odstranit a umožnit tak velký rozmach Androidu i v této oblasti.

3.1.2 Vývoj verzí

Následující tabulka ukazuje vývoj operačního systému Google Android a popisuje nejzásadnější změny, které daná verze přinesla.

Označení verze	Popis
1.0	Datum vydání: 23. září 2008
1.1	Datum vydání: 9. února 2009 - Drobné změny a opravy

1.5 Cupcake	Datum vydání: 30. dubna 2009 - založeno na linuxovém jádře 2.6.27 - přidáno nahrávání a přehrávání videa - nahrávání videí na YouTube a obrázků do Picasa - nová klávesnice - podpora Bluetooth A2DP a AVRCP - nové widgety - nové animace obrazovek
1.6 Donut	Datum vydání: 15. září 2009 - verze linuxového jádra 2.6.29 - vylepšen Android Market - vylepšeno hlasové vyhledávání - vylepšeno hledání v záložkách, kontaktech, historii a na webu - podpora pro CDMA, 802.1x, VPN, a text-to-speech - podpora WVGA displejů - vylepšena rychlost systému
2.0/2.1 Eclair	Datum vydání: 26. října 2009 - verze linuxového jádra 2.6.29 - optimalizována rychlost - podpora více rozlišení obrazovek - nový webový prohlížeč - vylepšení grafických prvků - nový seznam kontaktů - nové Google Maps - podpora Microsoft Exchange Server ActiveSync 2.5 - podpora pro vestavěný blesk fotoaparátu - digitální zoom fotoaparátu - vylepšeno multidotykové rozhraní - vylepšena dotyková klávesnice - podpora Bluetooth 2.1 - podpora interaktivních pozadí plochy
2.2 Froyo	Datum vydání: květen 2010 - verze linuxového jádra 2.6.32 - celkové vylepšení rychlosti, práce s pamětí a výkonu - implementována nová verze JavaScript do prohlížeče - vylepšena podpora Microsoft Exchange Serveru - přidán Wifi hotspot - možnost vypínání dat přes mobilní síť - vylepšení Android Marketu - telefonování a posílání kontaktů přes bluetooth - vylepšení webového prohlížeče - možnost instalovat aplikace na paměťové karty - podpora Adobe Flash 10.1 - podpora vyššího rozlišení displejů

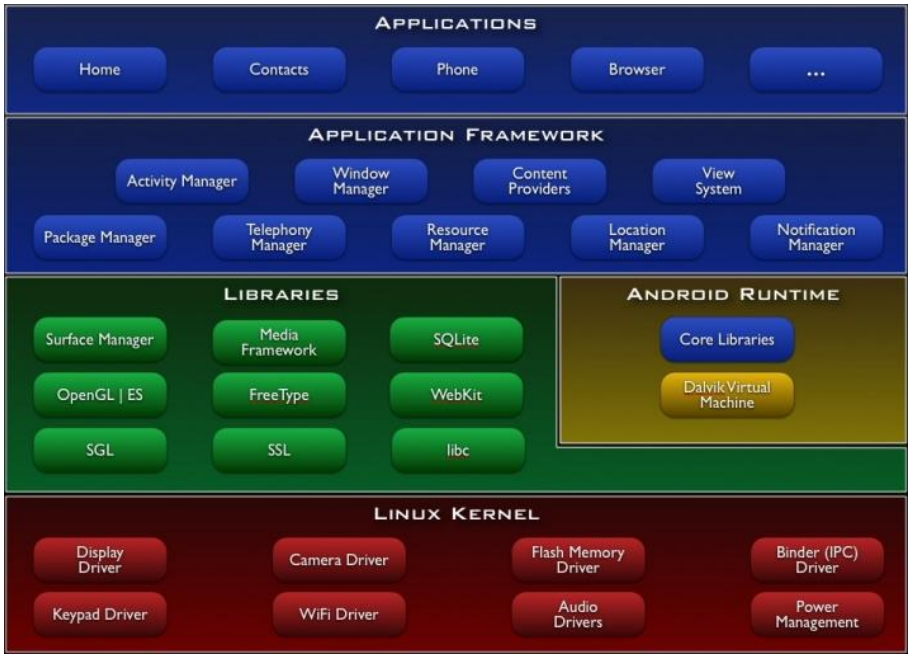
2.3 Gingerbread	Datum vydání: prosinec 2010 - verze linuxového jádra 2.6.35 - vylepšeno grafické prostředí - podpora vysokého rozlišení displejů - podpora pro VoIP telefonii - podpora pro WebM/VP8 video a AAC audio - nové audio efekty - podpora NFC - nový systém copy-paste - nová multidotyková klávesnice - podpora vývoje v nativním kódu - vylepšení audio a grafiky pro herní vývojáře - paralelní garbage collector pro zvýšení výkonu - podpora pro nové senzory - gyroskop a barometr - nový manažer stahování - vylepšeno řízení napájení - podpora více kamer/fotoaparátů - přechod na souborový systém ext4
3.0 Honeycomb	Datum vydání: 26. ledna 2011 - optimalizace pro tablet - nové uživatelské rozhraní - třírozměrná pracovní plocha s novými widgety - vylepšený multi-tasking - vylepšený webový prohlížeč - podpora video chatu přes Google Talk - hardwarová akcelerace

Tabulka 1 - Verze systému Google Android (3)

3.2 Vlastnosti a architektura

3.2.1 Vrstvy operačního systému

Android není jen čistě operační systém, ale komplexní softwarový balík obsahující operační systém, dále mezivrstvu mezi linuxovým jádrem a aplikačním frameworkem – tzv. middleware a nakonec sadu základních klíčových aplikací a uživatelské prostředí. Následující obrázek zobrazuje vrstvy systému a jednotlivé komponenty rozdělené do těchto vrstev. Vyšší vrstva znamená vyšší abstrakci oproti jádru a hardware zařízení.



Obrázek 1 - Vrstvy systému Android (4)

Na nejnižší vrstvě se nachází linuxové jádro, v němž jsou zavedeny ovladače základních hardwarových komponent jako je displej, klávesnice, fotoaparát, čtečka externích flash pamětí, bezdrátové síťové rozhraní, zvukové zařízení nebo obsluha a řízení napájení. Z toho vyplývá, že tato vrstva je přímou abstrakcí hardware a poskytuje služby pro jeho obsluhu vrstvám vyšším.

Ve vrstvě nad linuxovým jádrem se nachází vrstva knihoven a běhové prostředí Androidu, kterým je Dalvik Virtual Machine.

V nejvyšší vrstvě je pak aplikační framework, což je samotné Android API, které využívají vývojáři pro vývoj aplikací na platformě Android, avšak použití aplikačního frameworku není nutnost. Aplikace je možné vyvíjet v nativním kódu, tedy v jazyku C nebo C++. To může být výhodné především při vývoji nového ovladače nebo spouštění procesorově náročné úlohy. Ne vždy je ale vhodné nativní kód (Android NDK) použít a ne vždy to může mít za následek zrychlení aplikací. I přes to, že je kód napsaný v nativním jazyce, tak je spouštěn ve virtuální mašině, nikoli přímo jako samostatné linuxové vlákno.

Nad aplikačním frameworkem se již nacházejí jen samotné aplikace, a to jak tzv. vestavěné – grafické prostředí domácích obrazovek (homescreen), telefon, kontakty, internetový prohlížeč atd., tak i výrobcem nebo uživatelem doinstalované aplikace.

3.2.2 Dalvik Virtual Machine

Dalvik je virtuální běhové prostředí speciálně uzpůsobené pro zařízení s omezenou pamětí a výkonem procesoru. Operační systém Android tedy nepoužívá standardní Java Virtual Machine, ale vlastní virtuální mašinu, která je založená na otevřené implementaci Javy Apache Harmony. Na rozdíl od JVM, která je zásobníkově orientována, je Dalvik VM registrová a nepoužívá Java bytecode, nýbrž vlastní kompilovaný formát, který oproti bytecode eliminuje duplicitní konstanty a je optimalizován pro použití v přenosných zařízeních.

Ve standardních JVM jsou všechny třídy objektů kompilovány do bytecode a pro každou třídu se vytvoří samostatný .class soubor. Na platformě Android jsou třídy také kompilovány do bytecode, ale následně jsou převedeny a optimalizovány do DEX formátu, který je pro Dalvik VM nativním. Pro každou třídu se ale nepoužije samostatný .dex soubor, ale v jednom .dex souboru je více tříd, mnohdy dokonce všechny. Nejzásadnější paměťovou optimalizací je použití sdíleného adresáře typově-specifických konstant, do kterého se zanesou vícekrát použité řetězce v kódu právě jednou a na všech místech, kde se vyskytují, jsou uvedeny jen indexy odkazující se do tohoto adresáře.

Pro názornost si uveďme rozdíl mezi paměťovou náročností komprimovaného a nekomprimovaného Java bytecode a DEX formátem.

Kód	Nekomprimovaný Java bytecode	Komprimovaný Java bytecode (JAR)	Nekomprimovaný DEX formát
Systémové knihovny	21,4 MB (100%)	10,7 MB (50%)	10,3 MB (48%)
Internetový prohlížeč	0,47 MB (100%)	0,23 MB (49%)	0,21 MB (44%)

Tabulka 2 - Srovnání paměťové náročnosti Java byte code a .dex formátu

Jak je na první pohled vidět, tak nekomprimovaný DEX formát je na fyzické místo na disku méně náročný než komprimovaný Java bytecode, což značí, že tato optimalizace má značný přínos a v oblasti přenosných zařízení má – alespoň zatím – své opodstatnění.

Dalším zásadním rozdílem mezi JVM a Dalvik VM je způsob spouštění instancí. Totiž Dalvik pro každou aplikací spustí zvláštní linuxové jádro a navíc, každá aplikace běží ve vlastní instanci Dalvik VM. Proto je nutné, aby spouštění virtuální mašiny probíhalo velice rychle a samotná mašina by měla mít co nejmenší nároky na operační paměť zařízení. Pro tento účel používá Android takzvaný koncept Zygote, který zaručuje maximální míru sdílení načtených knihoven a zdrojů mezi jednotlivými instancemi virtuálních mašin. Proto není nutné, aby každá nová instance znovu načítala systémové knihovny, které jsou pouze pro čtení, a žádný jiný proces je tedy nemůže měnit. Tím se docílí úspory operační paměti, sníží se počet přístupů na souborový systém a rychlost inicializace nové instance virtuální mašiny se výrazně zvýší.

Jak již bylo řečeno, Dalvik VM je registrově orientovaná mašina. Tento fakt na jednu stranu přináší nevýhodu zhruba o 25% většího kódu než je tomu u zásobníkových VM – avšak to je do značné míry eliminováno použitím výše zmíněného sdíleného adresáře konstant. Na stranu druhou, tato architektura přináší výhodu v podobě snížení počtu spouštěných instrukcí, a to v průměru o 47% méně než u zásobníkových VM. (5)

3.2.3 Knihovny a aplikační framework

Android obsahuje řadu nativních knihoven v jazyku C a C++, které mohou být využívány různými komponentami celého systému, nebo instalovanými aplikacemi skrze aplikační framework. Vývojáři Androidu uvádějí následující sadu knihoven:

- **Systémová knihovna C**
  - BSD implementace standardní knihovny C (libc), optimalizovaná pro zařízení s vestavěnou linuxovou distribucí
- **Knihovny médií**
  - Založené na PacketVideo OpenCORE; knihovny pro podporu přehrávání a nahrávání populárních audio a video formátů a obrazových souborů – např. MPEG4, H.264, MP3, AAC, AMR, JPG, PNG
- **Surface manager**

    - Řídí přístup k podsystému displeje a komponuje 2D a 3D grafické vrstvy z různých aplikací
- **LibWebCore**
    - Knihovna pro renderování webového obsahu, kterou využívá jak vestavěný webový prohlížeč Androidu, tak i jiné aplikace, které potřebují zobrazit webové stránky ve vlastním pohledu
- **SGL**
    - 2D grafický engine
- **3D knihovny**
    - Implementace 3D knihovny založená na OpenGL ES 1.0; umí využívat jak hardwarovou akceleraci, tak i softwarové renderování
- **FreeType**
    - Knihovna pro renderování bitmapového a vektorového písma
- **SQLite**
    - Poměrně mocná, ale přitom nenáročná relační databáze, která je dostupná všem aplikacím

Aplikační framework Androidu je otevřenou vývojovou platformou, umožňující vývojářům stavět aplikace z bohatého množství komponent. Díky němu lze snadno využít hardware přístroje a standardní služby operačního systému jako jsou například notifikace, zjišťování pozice pomocí GPS, služby na pozadí, připojení k Internetu atp. API frameworku je napsané v jazyce Java, ve kterém se také drtivá aplikací pro Android vyvíjí.

Framework je využíván jak instalovanými aplikacemi, tak i vestavěnými aplikacemi systému. Architektura je navržena tak, aby dovolovala využívat již hotové komponenty, či dokonce nově vyvinuté sdílet mezi různými aplikacemi. (4) Toto řešení zároveň poskytuje možnost uživateli nahradit určitou komponentu jinou komponentou, která více vyhovuje jeho požadavkům. Operační systém Android se tímto stává velice flexibilním a mnoho výrobců mobilních zařízení toho využívá k vlastním úpravám (tzv. customizacím) systému. Ten pak dodávají s vlastní sadou aplikací, s vlastním grafickým prostředím a s různorodou nabídkou služeb.

Aplikace na platformě Android se typicky skládají ze čtyř základních komponent (ne vždy musí být využity všechny typy). Detailní pohled na komponenty a jejich

provázanost bude lépe zřejmý z praktické ukázky v kapitole 4, nyní si ale ve stručnosti popišme jejich rozdělení. První typ se nazývá **Activities**, neboli Aktivity. Aktivita v systému Android reprezentuje vizuální uživatelské prostředí a jeho chování. Její součástí je obrazovka zobrazující určité grafické prvky, např. seznam kontaktů, formulář se vstupními poli, mapu, nebo nastavení. Aktivita ale neurčuje pouze to, co se má uživateli zobrazit, ale i to, co se s těmito prvky bude dít po uživatelově interakci a jaké bude chování jednotlivých prvků v závislosti na různých vstupech. Aktivita se může nacházet v několika stavech: aktivní, pozastavená, zastavená a neaktivní. Na tyto stavy je někdy vhodné reagovat např. zavíráním, nebo opětovným otvíráním zdrojů. (5 str. 69) Existují i aplikace, které žádné aktivity nemají a poskytují pouze služby na pozadí či widget na domácí obrazovce. Obvyklá aplikace, která má grafické prostředí (nyní neuvažujeme widget), se skládá alespoň z jedné aktivity, častěji ale z více aktivit různě na sebe navazujících. Typickým příkladem tří provázaných aktivit může být zobrazení seznamu kontaktů jako první aktivita, po vybrání kontaktu se jako další aktivita nastartuje zobrazení detailu kontaktu a po stisknutí tlačítka pro úpravu kontaktu se zobrazí poslední aktivita s editovatelnými poli. Vizuální obsah okna je členěn do hierarchie tzv. pohledů (Views). Aplikační framework obsahuje základní třídu **View**, ze které poté dědí další, již předpřipravené a různě zaměřené pohledy. Vývojář má možnost využít některý z hotových pohledů, nebo si stávající upravit či naprogramovat úplně nový pohled. Hierarchie pohledů se dá různými způsoby členit, takže pokud si zvolíme například pohled, který má za úkol zobrazit na obrazovce obrázek a vsadíme jej do pohledu, který na obrazovce vytvoří posunovatelný seznam, vznikne nám nový komplexní pohled – posunovatelný seznam obrázků. Takovýmto komplexním pohledům, sdružujícím další pohledy v určitém rozložení se říká **Layout**.

Dalším typem komponenty využívané aplikacemi je Služba, neboli **Service**. Služba je část kódu aplikace, která nemá žádné grafické prostředí, ale její úkol je vykonávat požadované operace na pozadí. Příkladem může být výpočet nějaké hodnoty, stažení dat z internetu nebo přehrání zvukového souboru. Obvykle služba provede určitou operaci a výsledek poskytne aktivitě, která tento výsledek potřebuje pro zobrazení dat uživateli. Služba také bývá častou využívána pro periodické operace, kdy je potřeba v pravidelných časových úsecích provádět některé úkony, např. kontrola emailové schránky, automatické vyhledávání bezdrátových sítí a podobně. Každá služba

musí v aplikačním frameworku systému Android rozšiřovat základní třídu Service. Výhodou služby je možnost vystavit rozhraní, přes které může probíhat komunikace s jinou službou nebo s aktivitou. Skrze rozhraní může aktivita získávat data ze služby a také může službu ovládat. Služba je spouštěna ve stejném vlákně jako celá aplikace a neblokuje běh dalších aktivit.

Třetím typem komponenty jsou **Broadcast receivers**. Tyto komponenty mají jedinou funkci, a to čekání na definovanou událost, a pokud událost nastane, tak vyvolají určitou akci. Systém Android sám o sobě rozesílá různé typy událostí, např. vybitá baterie, změna jazykového nastavení prostředí, pořízení fotografie fotoaparátem přístroje, zapnutí či vypnutí bluetooth adaptéru atd. Vývojář má možnost definovat vlastní události a vlastní Broadcast receivery. (6)

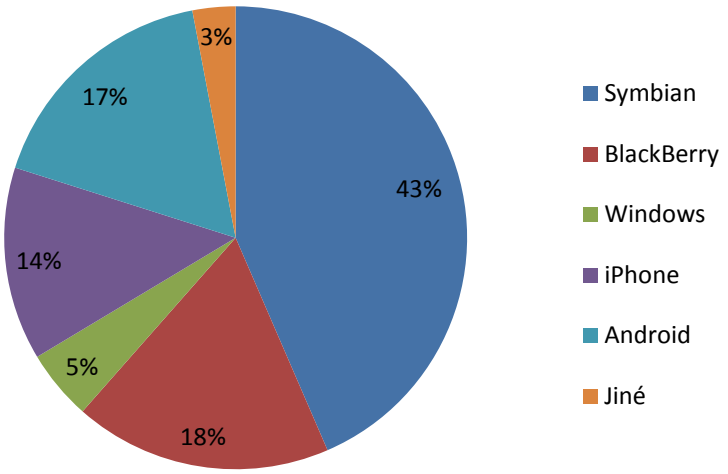
Posledním základním typem aplikačních komponent jsou **Content providers**, neboli Poskytovatelé obsahu. Content provider slouží k ukládání, modifikaci a čtení dat aplikací a je to jediný způsob, jak sdílet data napříč aplikacemi. Android již sám o sobě nabízí celou řadu poskytovatelů obsahu pro běžné datové formáty jako je audio, video, obrázky, kontaktní informace a tak dále. Obsah poskytovatelů je chráněn přístupovými právy, která musí daná aplikace mít, pokud chce data číst nebo zapisovat. Každý poskytovatel obsahu jakožto třída musí rozšiřovat základní třídu ContentProvider. Způsob, jakým jsou data reálně uložena, záleží na vývojáři, ale způsob na jejich dotazování a manipulaci je dán jednoznačně rozhraním, které musí být implementováno. Content provider je identifikován svou adresou URI, začínající řetězcem „content://“ a aby byla aplikace schopna dotázat se na data, tak musí kromě URI znát ještě jména požadovaných polí a jejich datový typ. Samotný dotaz se pak provede pomocí objektu ContentResolver, který vystavuje metody pro komunikaci s rozhraním poskytovatele obsahu. Data jsou pro každý dotaz vrácena v kurzoru, který je z databázových systémů dobře známý a umožňuje iterovat přes vrácené záznamy dané struktury a číst jednotlivá pole záznamu. Pro modifikace dat je použit klasický databázový model WHERE klauzule, mající SQL syntaxi, takže poměrně jednoduše určíme, které záznamy budou změněny či vymazány. Obvyklý způsob, jakým jsou data reálně uložena, je použití souborově orientované SQLite databáze, která je součástí systému Android. Není ovšem nutné ve vyvíjené aplikaci používat Content provider, který by data ukládal do databáze, nýbrž aplikace může svá data ukládat do vlastní SQLite databáze sama a napřímo. Jak

uvidíme v kapitole 4, aplikační framework systému Android myslí i na tuto alternativu a nabízí k použití pomocné třídy pro práci s databází (7).

3.3 Konkurenční operační systémy

3.3.1 Rozdělení trhu mobilních platforem

Pro celkový obraz operačního systému Android na trhu mobilních platforem je důležité se stručně zmínit i o jeho konkurenčních systémech. V následujících kapitolách si popíšeme největší hráče na trhu mobilních operačních systémů. V této krátké podkapitole si pouze ukážeme podíl mobilních platforem na trhu v polovině roku 2010. (8)



Obrázek 2 - Rozdělení trhu mobilních platforem (Q2/2010) (8)

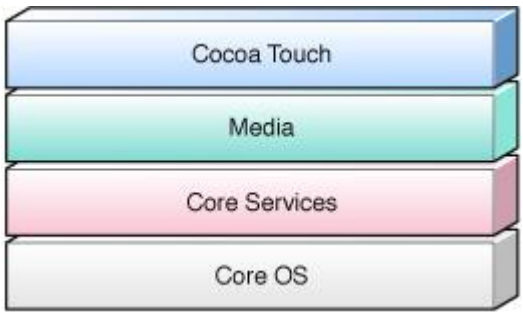
Graf na předchozím obrázku zachycuje situaci na trhu chytrých mobilních telefonů, to znamená, že nezachycuje další zařízení, jako jsou dotykové tablety atp. Ty v polovině roku 2010 nehráli na tomto trhu nijak zásadní roli, ovšem během pár měsíců se situace změnila a společnost Apple přišla se svým tabletem iPad, který je následován spoustou dalších podobných zařízení od konkurence, především z rodiny Android. V prvním čtvrtletí 2011 je pravděpodobně podíl systémů iOS a Android o něco vyšší než zobrazuje graf.

3.3.2 Apple iOS

Zdroj: archiv autora

Jako první, který si uvedeme, je iOS, operační systém společnosti Apple, který je nasazován na mobilní zařízení iPhone, iPad a iPod Touch. První verze systému byla vydána v roce 2007 a jako první přichází s multidotykovým ovládání navrženým tak, aby nebyl potřeba stylus, který byl do té doby běžnou výbavou dotykových zařízení. Velkou oblibu si získává právě díky propracovanému dotykovému ovládání, které je navíc umocněno použitím kapacitního displeje na zařízení iPhone. Novinkou je také využití nových senzorů, mezi které patří především akcelerometr detekující polohu a náklon přístroje. iOS částečně vychází ze systému Mac OS X, ze kterého bylo pro iOS použito jádro Mach. Celý systém je samozřejmě, stejně jako Android, navržený a optimalizovaný pro hardwarově omezená zařízení.

Operační systém iOS je rozdělen do čtyř vrstev a aplikace vyvíjené nad touto platformou mohou zasahovat do jakékoliv vrstvy podle své potřeby. Podobně jako u Androidu, každá vrstva udává jistou míru abstrakce vrstev nižších.



Obrázek 3 - Vrstvy operačního systému iOS (9)

Na nejnižších dvou vrstvách se nachází jádro systému a systémové služby vystavující rozhraní základních systémových funkcí, jako je přístup k souborům, síťovým zařízením, databázím či unixovým vláknům. Drtivá část těchto dvou vrstev je napsána v jazyce C.

Pokud se přesuneme do vrstev vyšších, nalezneme rozhraní pro přístup k běžným médiím (audio/video) a 2D/3D grafice. Vrstva médií je směsicí knihoven v jazyce

C a v jazyce Objective-C. Nejvyšší vrstva Cocoa Touch je kolekce aplikačních frameworků napsaných v jazyce Objective-C a poskytuje základní infrastrukturu pro aplikace nad platformou iOS – tedy velice podobně, jako je tomu u Androidu.

Apple nabízí pro vývoj aplikací vývojové prostředí iOS SDK. Nevýhodou oproti Androidu je uzavřenost systému. Distribuce aplikací však funguje na stejném principu, jako tomu je u Android Marketu – Apple provozuje službu AppStore, kde je možné nakupovat stahovat aplikace. (9)

3.3.3 Windows Phone

Původní systém Windows Mobile společnosti Microsoft byl navržený tak, aby se co nejvíce přibližoval desktopové verzi Windows, což lákalo uživatele ke koupi zařízení Pocket PC, aby jej využívali tak, jak název napovídá – jako kapesní počítač. Velkou nevýhodou se ale časem stává způsob ovládání, který vyžaduje použití stylusu (ovládací pero) a uživatelé stále více vyžadují možnost pohodlného ovládání a psaní prstem. Proto začíná být tato mobilní platforma na ústupu. Zvratem se má stát její nástupce – Windows Phone 7. Nový systém je zcela přepracovaný a jeho uživatelské grafické rozhraní nemá s původními Windows Mobile téměř nic společného. Ovládání je nyní možné pomocí prstu uživatele, byly integrovány sociální sítě, samozřejmostí je kancelářský balík Microsoft Office ve své mobilní verzi a jsou podporovány nejnovější typy senzorů a hardwarového vybavení.

Zdroj: archiv autora

O architektuře tohoto nového systému toho doposud nebylo mnoho zveřejněno. Jádro patří do rodiny Windows Embedded CE 6.0, jehož první vydání bylo v roce 2006 a nynější verze, použitá právě ve WP7, je již třetí v pořadí a vydána byla v září 2009. Inovace jádra se týkají především práce s pamětí a procesy, podpory nových souborových systémů a procesorových architektur, vestavěných aplikací pro prohlížení internetového obsahu a dokumentů, nativní podpory multidotykového ovládání apod. (10)

Po vydání nových Windows Phone 7 se ukazuje, že úspěch není tak velký, jak se očekávalo a zájem o tento systém je zastíněn převážně Androidem a iOS. Zlomem by mohl být kontrakt mezi Microsoftem a společností Nokia, díky němuž se Windows Phone stanou

hlavní platformou chytrých telefonů společnosti Nokia a dá se očekávat masivnější rozšíření. (11)

Microsoft pro vývoj aplikací doporučuje vývojové prostředí Visual Studio 2010 a prozatím je podporována platforma XNA a Silverlight. Do budoucna by mělo být uvolněno nativní API systému Windows Phone dovolující lepší využití hardware přístrojů. Aplikace jsou distribuovány do přístrojů pomocí služby Widnows Marketplace.

3.3.4 Symbian

Operační systém Symbian byl poprvé vydán v roce 2000 jako společný produkt společností Psion, Nokia, Ericsson a Motorola. Na jeho masové rozšíření se podílela právě Nokia, která jej nasazovala na své chytré telefony a nabízí pro něj aplikace prostřednictvím své služby Ovi Store. Symbian je otevřenou mobilní platformou, která prošla dlouhým vývojem a která přes velkou konkurenci si stále drží největší podíl na tru. Oblibu si Symbian zasloužil svým jednoduchým ovládáním, které částečně vycházelo z klasických firmwarů telefonů Nokia a také velkým výběrem dostupných aplikací. Nejnovější verze samozřejmě podporuje dotykové displeje a nejrůznější senzory. Nezbytností také byla integrace sociálních sítí. Domovská obrazovka je konfigurovatelná a je možné na ní umístit oblíbené widgety, které jí dělají mnohem více interaktivní.

Zdroj: archiv autora

Architektura systému je logicky rozdělena do několika vrstev, kde nejníže se opět nachází jádro systému, které je však využíváno vyšší servisní vrstvou a instalované aplikace se k němu napřímo nedostanou. Symbian se z důvodu kompatibility a robustnosti systému vydal minimalistickou cestou a do jádra zahrnul jen nejnutnější komponenty operačního systému, jako je správa paměti, plánovač vláken a ovladače základního hardware. Ostatní služby, mezi které patří síťové rozhraní, souborový systém nebo telefonie, se nacházejí v servisní vrstvě. Do té byly zařazeny také knihovny médií a grafiky. Nad servisní vrstvou se už nachází pouze platforma Java Micro Edition a UI framework pro kompozici grafického rozhraní aplikací.

Aplikace pro Symbian mohou být vyvíjeny několika způsoby. Jedním z nich je použití frameworku Qt, který je především určen pro vývoj aplikací s grafickým rozhraním a jako programovací jazyk využívá standardní C++. Dalším způsobem je programování v jazyce Symbian C++, který je nestandardní implementací jazyka C++. Jedním ze způsobů vývoje je samozřejmě Java ME, ale taky jazyk Python, pro jehož spouštění je nutná instalace interpretra. (12)

Závěrem se dá obecně o Symbianu říci, že jeho podíl se pomalu zmenšuje a postupně je nahrazován pokročilejšími a atraktivnějšími systémy. Jednodušše řečeno, Symbian přestává stíhat dynamický vývoj mobilních platforem a stále méně se mu daří uspokojovat rostoucí nároky uživatelů. I z tohoto důvodu Nokia již delší dobu hledá novou a modernější platformu. Společně se Symbianem se snažila nasazovat nové systémy Maemo a MeeGo, nakonec ale zvítězil systém Windows Phone. I nadále je ale plánován rozvoj a nasazení Symbianu, který si své uživatele zcela jistě najde. (13)

3.3.5 BlackBerry OS

Kanadská společnost Research In Motion (RIM), vyrábějící mobilní zařízení BlackBerry, používá vlastní operační systém, který se nazývá BlackBerry OS. Jedná se o poměrně uzavřenou platformu, do jejíž architektury není příliš vidět. Oblibu si mobilní zařízení s tímto systémem získala především díky propracované podpoře práce s elektronickou poštou, kalendářem a dalšími kancelářskými aplikacemi.

Společnost RIM vyrábí zařízeni BlackBerry od roku 1999 a od té doby se především ve Spojených Státech drží ve špičce mobilních platforem. První zařízení, které se dá nazývat smartphonem, přichází v roce 2002. Operační systém se od té doby vyvíjel a postupně zahrnoval všechny nové trendy, takže i dnes poměrně úspěšně konkuruje ostatním operačním systémům a jako ostatní je integrován s internetovými službami a sociálními sítěmi. Je to proprietární multitaskingový systém se zabudovanými síťovými funkcemi pro práci s emailem a internetem. Běžní uživatelé používají své emailové schránky, které si synchronizují s vestavěným emailovým klientem, ale velké organizace často využívají

Zdroj: archiv autora

podnikového řešení společnosti RIM, které se nazývá BlackBerry Enterprise Server (BES). BES je integrován do podnikového emailového systému a zabezpečuje rychlé a spolehlivé doručení zpráv podnikovým uživatelům pomocí služby push email. Push email funguje na obráceném principu než běžná synchronizace emailu, kdy zařízení v pravidelných intervalech kontroluje a stahuje poštu. Pokud je emailová zpráva poslána přes firemní emailový systém, BES se postará o spojení s konkrétním zařízením a prakticky ihned do něj zprávu odešle. BlackBerry Enterprise Server je možné napojit na běžná emailová řešení, jako je Microsoft Exchange, Lotus Domino nebo Novell GroupWise. Společnost Google vyvinula pro BES svůj vlastní konektor, takže je možné jej připojit i na cloudové řešení GoogleApps. (14)

Vývoj aplikací je společností RIM podporován zveřejnění BlackBerry API. Aplikace musí být nakonec digitálně podepsány, aby mohly být nainstalovány do zařízení BlackBerry. V současné době se objevuje nový systém od RIM, nazývaný Tablet OS, který, jak název napovídá, bude používán pro tabletová zařízení BlackBerry, ale časem se stane jedinou platformou společnosti RIM a bude instalována i na chytré mobilní zařízení. Tablet OS nabízí tři typy aplikačního prostředí – Java, Adobe AIR a BlackBerry WebWorks SDK. Zatímco první dvě prostředí jsou dobře známá, k třetímu je třeba říci, že se jedná o novou aplikační platformu, zahrnující JavaScript, HTML5 a CSS, takže aplikace jsou v podstatě velice interaktivní a dynamické webové aplikace, které jsou do zařízení distribuovány pomocí služby BlackBerry App World.

BlackBerry Tablet OS je popisován jako modulární systém, který by v budoucnu mohl podporovat i další typy aplikací, a není vyloučeno, že by na něj mohly být instalovány aplikace určené pro Android. (15)

4 Vlastní řešení

4.1 Vývojové prostředí

Android SDK neboli samotné knihovny platformy Android potřebné pro vývoj se dají z internetových stránek Androidu stáhnout kompletně, nebo si může uživatel zvolit cestu instalace pomocí instalátoru, ve kterém má možnost si vybrat pouze ty verze Androidu, které k vývoji potřebuje. Pro naši aplikaci je potřeba stáhnout verzi Google API 2.2, které má oproti standardnímu Android API 2.2 podporu Google Maps a budeme jej používat pro kompilaci aplikace. Důležitou prerekvizitou je samozřejmě nainstalovaná Java Virtual Machine. Jakmile je na vývojovém počítači nainstalovaný Android SDK, zbývá jen zprovoznit samotné vývojové prostředí.

Vývojový nástroj aplikací Android je založen na platformě Eclipse a dodáván jako plugin. Název pluginu je Android Development Tool (ADT) a stačí jej jednoduše stáhnout a rozbalit do distribuce Eclipse, případně použít vestavěnou funkci Eclipse pro instalaci nových pluginů. Po spuštění Eclipse s ADT máme připravené vývojové prostředí s podporou struktury softwarových projektů Android, se všemi potřebnými editory a pohledy. Velice důležitou součástí vývojového prostředí je emulátor přístroje s operačním systémem Android (AVD – Android Virtual Device), díky kterému máme možnost během vývoje okamžitě testovat, jak se naše změny projevují v aplikaci a případně sledovat průběh kódu v módu debuggeru. Zároveň, pokud používáme v aplikaci logger pro logování informací, ladících dat a chyb, jsou tyto logy za běhu přenášeny z virtuálního zařízení do ADT, kde si je má možnost vývojář přehledně procházet. Místo virtuálního zařízení existuje možnost připojit přes USB kabel k vývojovému počítači přímo fyzický přístroj s operačním systémem Android a v ladícím režimu jej používat k ladění aplikace a přijímat z něj logovaná data. Virtuální zařízení má tu výhodu, že na něm můžeme mít nasazenu libovolnou verzi platformy Android, a tím testovat funkčnost aplikace na různých verzích API.

Dále se již budeme zabývat vývojem ukázkové aplikace v aplikačním frameworku.

4.2 Zadání ukázkové aplikace

Jak již bylo řečeno v kapitole 2, popíšeme si Android API na reálné ukázkové aplikaci. V této kapitole si proto vytvořme zadání této aplikace, abychom jejím vývojem postihli co největší část aplikačního frameworku a především jeho nejdůležitější součásti.

Aplikací bude GPS tracker pro automatické i manuální zaznamenávání polohy a seskupování zaznamenaných poloh do cest, které bude možné graficky zobrazit na mapě. Automatické a ruční detekování polohy se bude spouštět z widgetu, ze kterého se také uživatel dostane na hlavní obrazovku aplikace, kde si bude spravovat své trasy, zobrazovat si je v mapě, či si prohlížet jednotlivé polohy. Na hlavní obrazovce aplikace bude také možné vyvolat menu pro vstup do konfigurace.

Analýza aplikace není pro účely této práce podstatná, proto se jí nebudeme zabývat a vyjmenujme si pouze seznam požadavků, kterým bude třeba vyhovět, aby aplikace pokryla dostatečnou část aplikačního frameworku.

Požadavky na aplikaci jsou následující:

- Widget na domácí obrazovku
- Možnost jednorázově uložit aktuální polohu stiskem tlačítka na widgetu
- Možnost spustit/vypnout automatické ukládání polohy v nastavených periodách stiskem tlačítka na widgetu
- Ukládat adresu polohy – načítat z internetu (vypínatelná vlastnost v nastavení)
- Možnost spustit hlavní obrazovku aplikace stiskem tlačítka na widgetu
- Zobrazení seznamu vytvořených tras na hlavní obrazovce (trasy budou seřazeny od nejnovější k nejstarší a u každé bude datum vytvoření)
- Možnost vytvoření nové trasy pomocí menu
- Možnost smazání/přejmenování trasy pomocí kontextového menu pro každou trasu
- Možnost zobrazení detailu trasy – jednotlivých uložených pozic
- Možnost smazání jednotlivých uložených pozic
- Možnost přidat krátkou poznámku k uložené trase pomocí kontextového menu

- Možnost zobrazit celou trasu na mapě
- Možnost zobrazit jednotlivé pozice na mapě
- Dotykem na uloženou pozici (bod) na mapě, se zobrazí čas a datum uložení, případně adresa, pokud je uložena
- Na obrazovce mapy bude možné vyvolat menu s položkami: Jít na start trasy, Jít na konec trasy, Přepnout na družicovou mapu
- Mapu bude možné oddalovat, přibližovat a posunovat
- V hlavním nastavení aplikace bude možné definovat: periodu automatického ukládání, zapnutí vibrace při uložení pozice, možnost ukládání adresy pozice z internetu, možnost použít mobilní síť k určení pozice (kromě GPS), možnost ukládat pouze pozice odlišné od poslední známé pozice
- Uložení pozice bude uživateli oznámeno krátkou zprávou na obrazovce
- Zapnuté automatické ukládání pozic bude signalizováno ikonou aplikace v horní liště systému jako oznámení; rozbalením oznamovací oblasti a kliknutím na položku naší aplikace se uživatel dostane na hlavní obrazovku aplikace
- V hlavním menu aplikace bude možnost zobrazit informaci o verzi aplikace a o autorovi

Pro grafické zpracování aplikace a ikon použijeme standardy doporučované společností Google. Hotovou aplikaci nakonec digitálně podepíšeme vlastním certifikátem a budeme jí distribuovat pomocí služby Android Market.

4.3 Vývoj aplikace

4.3.1 Vytvoření nové aplikace

4.3.1.1 Nový projekt

Ve vývojovém prostředí, které nadále budeme v této práci nazývat pouze ADT, vytvoříme nový Android Projekt, při jehož tvorbě je třeba nastavit několik parametrů. Prvním parametrem je samozřejmě název projektu, který si pro naši aplikaci zvolme

například MyWayTracker – v názvu by neměly být mezery. Název projektu je vhodné volit shodně s názvem aplikace a název aplikace volíme v našem případě tak, aby alespoň trochu napovídal její funkci a zaměření – to nám může později výrazně pomoci ve zvyšování počtu vyhledání a stažení ve službě Android Market. Dalším parametrem je cílová verze API, pomocí kterého se bude aplikace kompilovat. Zvolme si tedy verzi Google API 2.2 (verze 8), která je aktuálně nejrozšířenější verzí na trhu. Třetím parametrem je název aplikace, který, jak jsme si již řekli, je MyWay Tracker, ale nyní jej můžeme uvést s mezerou. Dále je potřeba uvést hlavní balík aplikace, který by ji měl jednoznačně identifikovat, ať je vyhledávána v jakékoliv službě. Proto je vhodné použít skládání názvu balíku tak, jak je zvykem v jazyce Java – název domény státu či jiné domény prvního řádu, k tomu připojíme název naší společnosti nebo skupiny, pod kterou aplikaci vyvíjíme a nakonec připojíme řetězec, který naši aplikaci identifikuje v rámci naší domény. Oddělovačem jednotlivých částí balíku je tečka. Název balíku si zvolme cz.byteworks.android.myway.activities. Dále máme možnost zvolit si možnost, aby pro nás ADT vytvořil první vstupní aktivitu. Pokud tuto možnost využijeme, pojmenujme si aktivitu DataActivity, protože hlavní aktivita v naší aplikaci bude zobrazovat uložená data. Posledním parametrem je minimální verze SDK (tedy API), na které aplikace poběží. Nižší verze nebudou aplikací podporovány. My budeme podporovat minimální verzi Android API 1.6, což je číslo verze 4 – to je třeba uvést jako hodnotu parametru.

Po vytvoření nového projektu se nám automaticky vygeneruje následující struktura.

Obrázek 4 - Struktura nového projektu, zdroj: archiv autora

4.3.1.2 Konfigurační soubory

Vygenerovaný nový projekt obsahuje tři konfigurační soubory, které si v této kapitole popíšeme. První je soubor default.properties. U tohoto konfiguračního souboru se nebudeme příliš dlouho zdržovat, protože je automaticky generovaný vývojových prostředím a neměl by se vývojářem editovat. Případná editace je totiž zbytečná, jelikož soubor se při sestavování aplikace vygeneruje znovu a změny, které bychom do něj provedli, se přepíší původním obsahem. Pokud bychom přece jen potřebovali nadefinovat parametry, které se použijí při sestavování aplikace systémem Ant, je třeba vytvořit nový konfigurační soubor build.properties, ve kterém tyto parametry uvedeme.

Konfigurační soubor proguard.cfg slouží pro nastavení nástroje Proguard. Ten se během sestavování aplikace stará o optimalizaci a očištění kódu od nepoužitých nebo duplicitních částí, přejmenování tříd, metod a vlastností objektů a tím redukuje velikost vzniklého APK instalačního souboru. V souboru proguard.cfg se definují pravidla říkající, jakým způsobem probíhá optimalizace kódu. Nám vystačí výchozí nastavení a pro účely popisu API operačního systému Android není potřeba se tímto konfiguračním souborem zabývat.

Zásadním a nejdůležitějším konfiguračním souborem aplikace je AndroidManifest.xml. Každá aplikace platformy Android musí mít konfigurační soubor právě tohoto jména v kořenovém adresáři. AndroidManifest.xml obsahuje základní informace o aplikaci, které musí být známé před spuštění této aplikace. (16)

Kromě jiného, soubor slouží k:

- Pojmenování Java balíku aplikace (definovali jsme při zakládání nového projektu)
- Popisu komponent aplikace – aktivit, služeb, broadcast receivers, poskytovatelů obsahu – ze kterých se aplikace skládá; ke každé komponentě definují implementující třídu a podmínky, za kterých se daná komponenta spouští
- Určení, které procesy budou používat aplikační komponenty
- Deklaraci potřebných oprávnění, která musí aplikace mít pro přístup k chráněným částem API a k jiným aplikacím

- Definici oprávnění pro ostatní aplikace, která budou vyžadována pro přístup ke komponentám naší aplikace
- Nastavení minimální požadované verze Android API, kterou aplikace potřebuje
- Vyjmenování knihoven, se kterými musí být provázána, protože jsou používány v aplikační logice

Konfigurační soubor musí být validní XML soubor. Pro názornost si ukažme, jak vypadá AndroidManifest.xml, který se vygeneroval během vytváření nového projektu. Jak je vidět, obsahuje informace, které jsme při vytváření nového projektu zadali jako parametry.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="cz.byteworks.android.myway.activities"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="4" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".DataActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
</manifest>
```

Kořenový element manifest obsahuje důležitý atribut package, jehož hodnotou je balík aplikace. Uvnitř něj se nacházejí další elementy – uses-sdk pro definici minimální verze API a application pro popis komponent aplikace. Jak je vidět, v elementu application se pomocí atributů icon a label definuje ikona aplikace a název aplikace. Hodnotou těchto atributů ale není cesta k obrázku ikony a konkrétní řetězec reprezentující název aplikace, nýbrž jsou to odkazy do souborů zdrojů. Tento způsob se při vývoji Android aplikací běžně používá a má výhodu v tom, že všechny zdroje, jako např. grafické prvky, lokalizované řetězce nebo témata aplikace, jsou externalizovány do struktury v adresáři res a v kódu aplikace se na ně odkazujeme logickými názvy. Hodnoty zdrojů jsou v kódu vyjadřovány takto:

```
@[balík:]typ:název
```

kde balík může být vynechán, pokud je zdroj ve stejném balíku jako aplikace. Typ určuje jakého typu je daný zdroj, takže například „string“ pro lokalizovaný řetězec, nebo „drawable“ pro grafický soubor. Název nakonec identifikuje samotný zdroj, tedy buď klíč do souboru řetězců, nebo název grafického souboru. O lokalizacích a grafických zdrojích si více povíme v kapitolách 4.3.1.3 a 4.3.1.4.

Jednotlivé elementy v konfiguračním souboru AndroidManifest.xml si vysvětlíme a popíšeme v dalších kapitolách, tak, jak nám budou postupně do konfigurace přibývat a jak se bude aplikace rozšiřovat. Co je ale ještě nutné nyní zmínit, je konfigurace oprávnění pro naši aplikaci.

Oprávnění definují přístupová práva aplikace k chráněným částem operačního systému nebo dat, nebo naopak mohou definovat přístupová práva ostatních aplikací k částem naší aplikace. Oprávnění, která aplikace vyžaduje k tomu, aby měla přístup k chráněným částem systému, se definují v manifestu pomocí elementu <uses-permission>, kde v atributu uvedeme název oprávnění. Android nabízí celou sadu možných oprávnění k různým službám a hardware přístroje. K našim účelům budeme potřebovat následující oprávnění, které takto vyjmenujeme v manifestu Androidmanifest.xml:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

K čemu tato přístupová oprávnění slouží? První z nich, ACCESS_FINE_LOCATION, dovolí naší aplikaci přistupovat k GPS modulu a snímat z něj souřadnice a další data – tedy k přesnému určování polohy. Další, ACCESS_COARSE_LOCATION, umožní aplikaci zjišťovat orientační polohu jinými prostředky, než je GPS, kterými je Wifi síť nebo mobilní síť. Třetí oprávnění, INTERNET, nám umožní připojení aplikace k internetu, což budeme později potřebovat pro načítání Google Maps nebo zjišťování adresy nasnímané pozice. Čtvrté potřebné oprávnění je VIBRATE, které aplikaci zpřístupní vibrační hardware přístroje, a ta pak bude moci uživatele notifikovat vibracemi. Poslední oprávnění je ACCESS_NETWORK_STATE, umožňující aplikaci zjišťovat dostupnost sítě. Tím, že tato oprávnění v manifestu vyjmenujeme, ještě samozřejmě neznamená, že aplikace tato oprávnění dostane. Pouze jsme tím řekli, která oprávnění potřebuje, a oprávnění budou reálně přidělena až po jejich potvrzení uživatelem během instalace aplikace. Pokud by je uživatel zamítnul, instalace aplikace se předčasně ukončí. Oprávnění ostatních aplikací pro

přístup k naší aplikaci není potřeba definovat, protože její služby nebudou ostatní aplikace využívat. Ale pokud bychom tak chtěli učinit, tak máme možnost obdobně vyjmenovat seznam oprávnění, které budeme po ostatních aplikacích vyžadovat, a to v elementu <permission>.

4.3.1.3 Lokalizace aplikace

I přes to, že vzorová aplikace bude mít uživatelské rozhraní pouze v jediném jazyce, a to v angličtině, tak si řekněme, jaký mechanizmus nabízí aplikační framework pro lokalizaci aplikací do různých jazyků. Když se řekne lokalizace aplikace, nemusí tím být myšleno jen zobrazování textů v jazyce uživatele, ale lokalizovány mohou být také grafické zdroje, zvuky, číselné formáty, peněžní měny atd. Aplikační framework používá principu oddělení všech těchto zmíněných zdrojů od aplikační logiky a prezentační vrstvy aplikace do souboru zdrojů, pojmenování zdrojů logickými jmény a používání logických jmen v kódu místo toho, abychom natvrdo uváděli řetězce či například obrázky, které se mají uživateli zobrazit. Mechanizmus je poměrně jednoduchý a vysvětlíme si jej na zdroji textových řetězců, avšak způsob lokalizace ostatních zdrojů je shodný.

Řekněme, že chceme, aby se název aplikace zobrazoval v jazyce, jaký je zvolen v zařízení Android. To znamená, že pokud uživatel používá anglické prostředí (locale), zobrazí se mu název aplikace jako MyWay Tracker. Náš požadavek ale může být, aby se aplikace v českém prostředí jmenovala MyWay Stopař. První věcí je externalizace řetězce názvu aplikace do souboru zdrojů, jehož výchozí podoba se nachází v souboru res/values/strings.xml. Externalizaci za nás již udělal ADT, takže pokud tento soubor zdrojů otevřeme, najdeme v něm řádek

```
<string name="app_name">MyWay Tracker</string>
```

který říká, že pro logický název řetězcového zdroje app_name, bude hodnota „MyWay Tracker“. V manifestu si pak připomeňme řádek, kde se logické jméno app_name nachází:

```
<application android:icon="@drawable/ic_launcher" android:label="@string/app_name">
```

Při spuštění aplikace dojde k nahrazení logického jména za reálných řetězec ze souboru strings.xml. Odkaz @string/app_name nebo jakýkoliv jiný ze souboru zdrojů můžeme používat i v jiných částech kódu, ať je to XML definice obrazovky, nebo Java kód. Pro používání zdrojů v Java kódu slouží generovaná třída R, kterou si popíšeme v kapitole 4.3.1.5. Jak nyní ale zobrazíme název aplikace lokalizovaný do češtiny? Jednoduše

vytvoříme další soubor strings.xml, tentokrát ale v adresáři res/values-cs a do něj zapíšeme řádek

```
<string name="app_name">MyWay Stopař</string>.
```

Název zdroje zůstává stejný – app_name. Jestliže bude na přístroji nastavena čeština, jejíž identifikátor locale je „cs“, načte se zdroj přednostně z adresáře res/values-cs a teprve v případě, že tam zdroj se jménem app_name nebude nalezen, prohledá se soubor výchozích zdrojů res/values/strings.xml. Z toho vyplývá jedna zajímavá vlastnost, že není nutné lokalizovat všechny zdroje, ale v lokalizovaných zdrojích res/values-XX mohou být jen podmnožiny všech zdrojů. Nutností ale je, aby ve výchozím souboru byla množina zdrojů kompletní. Podobným způsobem bychom mohli přidat například francouzštinu, vložením zdroje do souborů res/values-fr/strings.xml.

4.3.1.4 Grafické zdroje

Jak jsme si řekli v předchozí kapitole, různé zdroje, včetně grafických, máme možnost lokalizovat podle nastavení přístroje. Podobným způsobem se dají grafické zdroje zobrazovat v různých modifikacích na základě několika kritérií, mezi které patří například velikost rozlišení displeje přístroje, orientace obrazovky, použití dotykového/nedotykového displeje, zapnutí nočního režimu atp. Navíc se tato kritéria dají za dodržení pořadí předepsaného v dokumentaci kombinovat.

Výchozím adresářem pro grafické zdroje je adresář res/drawable. Nyní ale uvažujme případ, kdy bude aplikace používána na přístrojích s velice malým rozlišením displeje a naopak také na přístrojích s displejem o vysokém rozlišení. Jako grafický zdroj si pro názorný příklad zvolme ikonu aplikace. Jestliže bychom použili ikonu malého rozlišení a zobrazili ji na přístroji s vysokým rozlišením, bude vypadat velice nekvalitně. Naopak, jestliže použijeme ikonu vysokého rozlišení a zobrazíme ji na přístroji s nízkým rozlišením, bude se muset softwarově zmenšit za běhu aplikace, což zabere procesorový čas, a navíc zmenšovací algoritmus z důvodu urychlení nevyprodukuje kvalitní zmenšenou ikonu. Proto aplikační framework nabízí mechanizmus obdobný lokalizaci zdrojů i pro modifikaci výběru grafického zdroje na základě jiných kritérií. Problém s rozlišením displeje vyřešíme rozdělením adresáře res/drawable na tři, lišící se podle použití na různých rozlišeních. Android dělí rozlišení zdrojů na pět typů, a sice ldpi – rozlišení zhruba

120 dpi, mdpi – zhruba 160 dpi, hdpi – zhruba 240 dpi, xhdpi – velmi vysoké rozlišení okolo 320 dpi, nodpi – zdroj, který není potřeba škálovat pro různá rozlišení. Podle těchto typů pojmenováváme adresáře grafických zdrojů, ze kterých nakonec Android vybere pro daný přístroj ten nejvhodnější. Vzniknou nám tedy adresáře res/drawable-hdpi, res/drawable-mdpi, res/drawable-ldpi atd. Ikonu si pak kvalitně převedeme do daného rozlišení a uložíme do odpovídajícího adresáře. V kombinaci s kritériem lokalizace zdroje by mohly adresáře vypadat takto: res/drawable-cs-hdpi, res/drawable-cs-mdpi, res/drawable-fr-hdpi, res/drawable-fr-mdpi a dále. Předpokládejme, že jsme do těchto adresářů uložili obrázek ikony icon.png v odpovídajících rozlišeních. Referencovat pak budeme tento grafický zdroj v XML definicích vzhledu například takto:

```
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
      android:src="@drawable/icon" />
```

V odkazu není třeba nijak řešit rozlišení obrazovky nebo lokalizace, ale jednoduše uvedeme odkaz @drawable/icon bez modifikátorů a aplikační framework se sám postará o zobrazení správného zdroje.

Dalším kritériem může být například již zmíněná orientace displeje, kterou v názvu adresáře zdrojů zohledníme modifikátorem „port“ pro zobrazení na výšku, nebo „land“ pro zobrazení na šířku.

4.3.1.5 Třída R

Jestliže máme potřebné zdroje připraveny v adresáři res, můžeme se na ně v Java kódu jednoduše odkazovat pomocí speciální třídy R, kterou kompilátor automaticky generuje na základě obsahu adresáře res. Po kompilaci obsahuje třída R identifikátory ke všem zdrojům. Pro každý typ zdroje je ve třídě R vytvořena podtřída nazvaná podle daného typu zdroje. Například pro grafické zdroje, které jsou v adresáři drawable, se vygeneruje třída R.drawable. Pro každý zdroj se poté v těchto podtřídách vygenerují statické celočíselné konstanty, nazvané podle konkrétního zdroje.

Vývojář nikdy nemusí znát skutečné vygenerované číselné ID, protože se bude odkazovat pomocí statických konstant. Obdobným způsobem, jakým se vytvářely odkazy na zdroje v XML souborech, se vytváří struktura konstant v R třídě a jejích podtřídách. Pravidlo tvorby podtříd a konstant je následující:

```
[<název balíku>.]R.<typ zdroje>.<název zdroje>
```

To znamená, že pro naši ikonu, jejíž cesta je res/drawable-hdpi/icon.png (a v dalších adresářích drawable-xxxx), se vygeneruje v třídě R.drawable statická konstanta icon. V Java kódu se budeme na tento zdroj odkazovat jako R.drawable.icon. Název aplikace, na který jsme v kapitole 4.3.1.3 odkazovali v XML pomocí odkazu @string/app_name, budeme v Java kódu používat takto: R.string.app_name. Pro názornost si uveďme obsah vygenerované třídy R, která vznikla ihned po vytvoření nového projektu:

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Jak je ze zdrojového kódu vidět, tak kromě podtříd string a drawable byly vygenerovány ještě třídy attr a layout. První z nich, attr, nám poslouží, pokud budeme chtít v aplikaci deklarovat názvy jakýchkoliv atributů přehledně na jednom místě a kdekoliv v kódu na ně odkazovat – v podstatě se jedná o třídu libovolných konstant. Důležité je zmínit, že Android API obsahuje samo o sobě vlastní třídu R, která je ale v balíku android, takže se na ní poté odkazujeme uvedením plně kvalifikovaného názvu android.R.xxxx.yyyy. Ve třídě android.R.attr je definována celá řada atributů, které Android používá a vývojář je samozřejmě může používat také. Poslední podtřídou, která se nám ve třídě R vygenerovala, je layout. Ta obsahuje konstanty odkazující se na XML soubory definující rozložení jednotlivých obrazovek aplikace – v našem případě zatím jediné rozložení aktivity main.

Kromě zmíněných typů zdrojů existují ještě další, ze kterých v naší aplikaci využijeme jen některé. Zmiňme tedy ještě například typ R.color, pro definici barev, typ R.menu odkazující na XML soubory popisující strukturu menu, typ R.array, R.style, R.integer atd., odkazující na soubory v adresáři res/values.

Jakým konkrétním způsobem se v Java kódu reálně načte daný zdroj pomocí konstant R třídy bude zřejmé z implementace těla jednotlivých aktivit.

4.3.2 Databáze

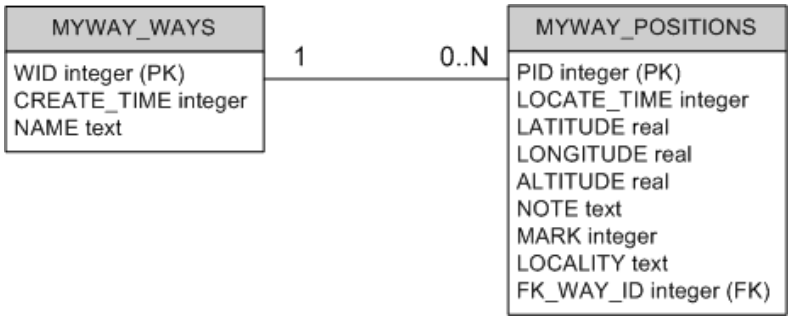
Operační systém Android v sobě integruje SQLite databázi pro strukturované a relačně založené ukládání aplikačních dat. Důležitým faktem je, že pokud aplikace databázi využívá, tak má vždy svou a tuto je možno používat z jakékoliv třídy aplikace. Jiné aplikace ale nemají šanci se napřímo do této databáze dotazovat. Aplikace může využívat i více databází, kdy každá musí mít jiné jméno, které nejen identifikuje databázi, ale také určuje název souboru, ve kterém budou data reálně uložena, protože SQLite je souborově založená databáze.

Pro vytvoření a používaní SQLite databáze v Android aplikacích je doporučováno použít mechanizmus rozšíření třídy SQLiteOpenHelper, ve které za prvé definujeme verzi fyzického modelu pro případné rozšíření, a za druhé přetížíme metodu onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion), ve které máme možnost na základě verze modelu provést DDL skript a tím změnit databázové tabulky. Pro vytvoření samotné struktury tabulek je potřeba přetížit metodu onCreate(SQLiteDatabase db), ve které zavoláme DDL skripty pro vytvoření a případné naplnění potřebných tabulek. Tato metoda je volána pouze jednou, při prvotním vytváření databáze. Pro přístup k datům tato třída vystavuje dvě metody, getReadableDatabase(), sloužící jen pro čtení z databáze, a getWritableDatabase(), pro čtení a zápis do databáze. Obě metody při jejich prvním zavolání nejprve provedou onCreate (pouze pokud není databáze doposud vytvořena), onUpgrade a nakonec onOpen metodu. Od této chvíle je databáze umístěna do mezipaměti a další její zavolání již nebude časově náročné. Objekt SQLiteDatabase, vracený těmito dvěma metodami, zastupuje samotnou databázi a přímo nad ním provádíme běžné databázové operace CRUD (create, read, update, delete). Pro tento účel nabízí objekt databáze metody, z nichž některé si zde uveďme, protože jsou pro práci s SQLite databází nejdůležitější a některé z nich použijeme v naší aplikaci. Jedná se samozřejmě o metody insert(...), query(...), update(...) a delete(…), které odpovídají CRUD operacím, a dále metoda execSQL(String sqlQuery), kterou můžeme použít na jakýkoliv SQL nebo DDL dotaz/příkaz, který nevrací žádná data z databáze. Důležitá metoda, kterou použijeme v aplikaci, je rawQuery(String query, String[] args). Rozdíl mezi metodou query a rawQuery spočívá v tom, jakým způsobem se dotazujeme do databáze. Zatímco query očekává části dotazu jako jednotlivé argumenty – název tabulky, požadované

sloupce, podmínku, parametry podmínky, sloupce pro seskupení, filtr, řazení a limit, tak rawQuery má pouze dva argumenty. První je samotný dotaz jako řetězec a druhý může, ale nemusí obsahovat parametry pro zadaný dotaz jako pole řetězců. Dotaz je standardní SQL dotaz. Parametry jsou dosazeny místo znaků „?“ v řetězci dotazu, a to v pořadí, v jakém jsou v poli. Tento princip se také často využívá z důvodu bezpečnosti dotazování do databáze, protože do určité míry zabraňuje „SQL injection“ útokům.

Obě metody, query i rawQuery, vracejí data v iterovatelné struktuře reprezentované objektem Cursor. Tento objekt vystavuje všechny potřené metody pro procházení souboru dat (řádků) vráceného na základě dotazu a čtení jednotlivých sloupců.

Pro naši potřebu postačí dvě databázové tabulky, jedna pro ukládání vytvořených tras a druhá pro ukládání zjištěných pozic. Tento velice jednoduchý model je vyjádřen na následujícím diagramu:



Obrázek 5 - Databázové tabulky ukázkové aplikace, zdroj: archiv autora

Tabulka MYWAY_WAYS bude sloužit k ukládání tras a tabulka MYWAY_POSITIONS k ukládání pozic. Vztah je z diagramu zřejmý – pro každou trasu může být uloženo libovolné množství pozic, ale každá pozice se vždy vztahuje k právě jedné trase. Nyní si ještě popišme význam jednotlivých sloupců tabulek.

Pro tabulku tras je popis sloupců následující:

WID	Identifikátor – primární klíč
CREATE_TIME	Datum vytvoření trasy
NAME	Název trasy

Tabulka 3 - Popis sloupců tabulky tras

Tabulka pozic bude obsahovat následující sloupce:

PID	Identifikátor – primární klíč
LOCATE_TIME	Datum vytvoření trasy
LATITUDE	Zeměpisná šířka pozice
LONGITUDE	Zeměpisná délka pozice
ALTITUDE	Nadmořská výška pozice
NOTE	Uživatelova poznámka
MARK	Příznak pro označení pozice
LOCALITY	Adresa získaná z Internetu
FK_WAY_ID	Cizí klíč, odkazující na trasu do tabulky MYWAY_WAYS

Tabulka 4 - Popis sloupců tabulky pozic

Z diagramu tabulek, kde jsou vidět i datové typy sloupců, vyplývá jedna zvláštnost databáze SQLite, a sice že SQLite nezná datový typ datum. Místo něj existují tři způsoby, jak datum ukládat. Prvním způsobem je text, kdy se datum uloží ve formátu "YYYY-MM-DD HH:MM:SS.SSS", ale pokud s ním pak chceme dále logicky pracovat, je třeba jej parsovat a to má samozřejmě vliv na výkon. Dalším způsobem je ukládání jako reálné desetinné číslo vyjadřující počet uplynulých dní od 24. listopadu roku 4714 př.n.l. Pro nás je ale nejvýhodnější poslední způsob, uložení jako celočíselná hodnota vyjadřující počet milisekund uplynulých od půlnoci 1. 1. 1970. Tento způsob se pro nás hodí z jednoduchého důvodu – v jazyce Java z tohoto formátu vytvoříme typ Date pouhým předáním této hodnoty konstruktoru objektu Date a dále můžeme s datem pracovat běžným způsobem.

Nyní, když víme, jak budou vypadat potřebné tabulky, můžeme sestavit DDL příkazy pro jejich vytvoření a ty pak použít v metodě onCreate pomocné databázové třídy. Zdrojový kód pomocné třídy pro získávání databáze v naší ukázkové aplikace vypadá následovně:

```
public class DbHelper extends SQLiteOpenHelper {

   private static final int DATABASE_VERSION = 2;

        private static final String DATABASE_CREATE1 = "create table MYWAY_POSITIONS ( PID"
        + " integer primary key autoincrement, locate_time integer not null, latitude"
        + " real not null, longitude real not null, altitude real, note text, mark integer,"
        + " fk_way_id integer, locality text)";
        private static final String DATABASE_CREATE2 = "create table MYWAY_WAYS (WID"
        + " integer primary key autoincrement, create_time integer not null, name text not"
        + " null)";

   public DbHelper(Context context, String name, CursorFactory factory, int version) {
      super(context, name, factory, DATABASE_VERSION);
   }

   @Override
   public void onCreate(SQLiteDatabase db) {
      db.execSQL(DATABASE_CREATE1);
      db.execSQL(DATABASE_CREATE2);
   }

   @Override
   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
   }

}
```

Nyní máme připravenou třídu, která při prvním zavolání vytvoří databázi, poskytne nám k ní přístup pro čtení a zápis. Abychom ale udrželi kód přehledný, udržovatelný a jednoduše rozšiřitelný, tak potřebujeme oddělit databázovou vrstvu od aplikační logiky tak, aby v dalším kódu, jako jsou například třídy aktivit či služeb, jsme se nemuseli o databázové spojení vůbec starat a získávali data v podobě vhodných objektů. Pro dosažení tohoto návrhového vzoru je potřeba provést dvě věci. Nejprve vytvoříme dvě třídy objektů odpovídající tabulkám v databázi. Data vrácená v řádcích objektu Cursor se pak se budou mapovat na tyto objekty a s těmi se bude v aplikační logice reálně pracovat. Vznikne nám tedy objekt Way a objekt Position. Druhou věcí, kterou musíme zajistit, je vytvoření pomocné třídy, do které umístíme metody realizující samotná volání do databáze, ať už se jedná o dotazy či modifikace. Tato pomocná třída si bude v případě potřeby vytvářet spojení do databáze pomocí již námi vytvořené třídy DbHelper, ze které získá objekt databáze pro spouštění dotazů a metody nové pomocné třídy zastoupí všechny operace, které bude aplikace potřebovat nad databází provádět. Tyto třídy se také někdy nazývají DAO (Data Access Object) a zastřešují operace s objekty. V metodách se zrealizují SQL dotazy do databáze, provede se mapování vrácených řádků na objekty a ty se pak vrátí. Některé metody zase naopak přijmou objekt jako argument a na jeho základě sestaví databázový příkaz pro vložení/modifikaci nebo smazání dat v databázi a ten pak provedou. V aplikační logice nakonec použijeme jen DAO objekt, na kterém zavoláme potřebnou metodu, a nebudeme se starat o úkony spojené s databází samotnou. Jestliže

jsme pomocnou metodu pro práci s databází samotnou pojmenovali DbHelper, tak pomocnou třídu pro práci s daty pojmenujme například DataHelper. Nebudeme si zde uvádět kompletní zdrojový kód pomocné třídy DataHelper, ale pro názornost a ukázku jak provádět operace nad databází si zde ukažme alespoň dvě metody z této třídy. První poslouží k uložení pozice do databáze a druhá k vylistování seznamu všech tras.

```
public void insertPosition(Position position) {
   if (position == null) {
      return;
   }
   if (position.getLocateTime() == null) {
         position.setLocateTime(new Date());
   }
   position.setWayId(getActualWay());
   String QUERY = "INSERT INTO MYWAY_POSITIONS (locate_time, " +
         "latitude, longitude, altitude, note, mark, fk_way_id, " +
         "locality) VALUES ("+position.getLocateTime().getTime()+
         ", "+position.getLatitude()+
                     ", "+position.getLongitude()+
                     ", "+position.getAltitude()+
                     ", "+getQuotedOrNull(position.getNote())+
         ", "+position.getMark()+
                     ", "+position.getWayId()+", "+
         getQuotedOrNull(position.getLocality())+");";
   SQLiteDatabase db = dbHelper.getWritableDatabase();
   db.execSQL(QUERY);
   db.close();
}
public List<Way> getAllWays(boolean sortDesc) {
      String SORT = null;
      if (sortDesc) {
         SORT = "DESC";
      }
      else {
         SORT = "ASC";
      }
      List<Way> wayList = new ArrayList<Way>();
      String QUERY = "SELECT wid, create_time, name FROM myway_ways"+
            "ORDER BY wid "+SORT;
      SQLiteDatabase db = dbHelper.getReadableDatabase();
      Cursor cur = db.rawQuery(QUERY, null);
      cur.moveToFirst();
      while (!cur.isAfterLast()) {
         Way way = new Way();
         way.setId(cur.getInt(0));
         way.setCreateTime(new Date(cur.getLong(1)));
         way.setName(cur.getString(2));
         wayList.add(way);
         cur.moveToNext();
      }
      cur.close();
      db.close();
      return wayList;
   }
```

Metoda insertPosition(Position position) přijímá jako jediný argument objekt Position, ze kterého všechny vlastnosti přenese do databázového příkazu pro uložení dat a ten pak spustí nad databází. Druhá metoda getAllWays(boolean sortDesc) naopak žádná data neukládá, ale vybere z databáze všechny záznamy z tabulky MYWAY_WAYS, ty přemapuje na objekt Way a vrátí je v typovaném seznamu.

Tímto způsobem budeme implementovat všechny operace nad databází a jsme tím s databázovou vrstvou hotovi. Odteď máme připraveno vše, co bude potřeba v aplikační logice potřeba pro perzistenci dat.

4.3.3 GPS a jiné služby

Podobným způsobem, jako jsme implementovali databázovou vrstvu v naší aplikaci, vytvoříme službu, která bude poskytovat všechny požadované funkce pro zjišťování GPS polohy. Jelikož je požadováno, aby se poloha dala zjistit jednorázově anebo automaticky, tak by bylo nesmyslné zanášet kód pro zjištění polohy přímo do příslušných aktivit. Takto opět oddělíme třídy a metody, abychom je mohli volat z více míst v naší aplikaci a neduplikovali tak kód.

Zjišťování pozice

Nejprve si ale popišme část aplikačního frameworku Android, která má na starosti GPS lokaci. Nejdůležitější komponentou je systémová služba LocationManager. Objekt této služby získáme voláním metody getSystemService(Context.LOCATION_SERVICE) na aplikačním kontextu. LocationManager nabízí tři důležité věci:

1) Správce poskytovatelů polohy – zjištěná poloha může být získána od několika různých poskytovatelů – GPS poskytovatel, síťový/wifi poskytovatel; každý z poskytovatelů se liší především v přesnosti určení polohy a v rychlosti a dostupnosti zjištění polohy
2) Zapnutí/vypnutí registrace požadavků na automatické získávání polohy na základě zadaných kritérií
3) Zaregistrování akcí, které mají být spuštěny, jakmile jsou splněna určitá kritéria (například dosažení definované polohy v rámci jisté přesnosti)

V naší aplikaci použijeme oba dostupné poskytovatele polohy. Zatímco GPS poskytovatel (provider) bude výchozím, síťový bude uživatelem volitelný a umožní zjištění polohy i uvnitř objektů, kde GPS lokace není možná. V případě získání polohy pomocí síťového poskytovatele ale musí uživatel počítat s určitou nepřesností, protože

poloha odpovídá GPS souřadnici BTS stanice mobilního operátora, nebo místního wifi poskytovatele.

Způsob, jakým se LocationManager používá pro získání GPS souřadnic, vychází z požadavku, zda má být poloha aktuální, či jestli se uživatel spokojí s posledními známými daty. Pokud chceme jen rychle a bez čekání zjistit souřadnice, které daný poskytovatel naposledy zjistil, zavoláme na službě LocationManager metodu getLastKnownLocation(String provider) a prakticky ihned je nám vrácen objekt Location, obsahující všechny informace týkající se poslední známé polohy. My ale budeme používat způsob, kdy se vždy požaduje nová aktuální detekce polohy. Pro tento účel zvolíme volání metody requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener) na objektu třídy LocationManager. Prvním argumentem je název poskytovatele polohy, kterého chceme použít. Druhý argument říká, jak minimálně dlouhá by měla být doba od poslední detekce polohy, aby mohla být brána nově zjištěná poloha v potaz. Třetí argument je principiálně stejný jako předchozí, pouze místo minimálního času od poslední známé polohy pracuje s minimální vzdáleností, o kterou se musí nová poloha lišit od předchozí. Posledním argumentem je objekt typu LocationListener, jenž hraje důležitou roli ve chvíli, kdy nastane událost týkající se změny polohy, nebo změny stavu poskytovatele polohy. LocationListener musí implementovat následující rozhraní:

```
public abstract interface LocationListener
{
  public abstract void onLocationChanged(Location location);

  public abstract void onStatusChanged(String provider, int status, Bundle extras);

  public abstract void onProviderEnabled(String provider);

  public abstract void onProviderDisabled(String provider);
}
```

Za předpokladu, že máme objekt třídy implementující toto rozhraní, můžeme jej zaregistrovat ve službě LocationManager pomocí metody requestLocationUpdates. Při změnách polohy, které vyhovují kritériím definovaným argumenty metody requestLocationUpdates, zavolá LocationManager metodu onLocationChanged(Location location) na objektu typu LocationListener. V této metodě si tedy naprogramujeme akci, která se má provést při změně polohy, přičemž informace o poloze dostáváme v objektu Location. V případě naší aplikace se provede uložení polohy. Kromě metody

onLocationChanged je nutné ve třídě LocationListener implementovat další tři, které se zavolají při změně stavu poskytovatele polohy. Konkrétně metoda onStatusChanged bude volána, pokud se služba zjišťování polohy stane z nějakého důvodu dočasně nedostupnou. Zbývající dvě metody onProviderEnabled a onProviderDisabled se volají, jak už název napovídá, když dojde k úplnému vypnutí nebo zapnutí funkce zjišťování polohy uživatelem. Máme tak možnost reagovat na případy, kdy například během zjišťování polohy uživatel vypne GPS modul přístroje.

Snímání polohy a tím pádem volání metody onLocationChanged bude probíhat podle zadaných kritérií do té doby, než odstraníme registraci objektu LocationListener ve službě LocationManager. To se provede zavoláním metody removeUpdates(LocationListener listener).

Nyní, když víme, jak funguje mechanizmus detekce polohy v systému Android, tak si můžeme říci více o řešení v naší aplikaci. Důležitou třídu, kterou vytvoříme za účelem jednotné obsluhy systému detekce polohy, pojmenujeme PositionService a bude pro nás centrálním bodem pro zahájení zjišťování polohy. Do konstruktoru této třídy potřebujeme předat naší databázovou pomocnou třídu DataHelper a aplikační kontext, ze kterého získáme službu LocationManager. Pro uložení polohy vytvoříme metodu storeActualPosition(), ve které se provede inicializace objektu LocationListener a na základě nastavení uživatele se zaregistruje požadavek na aktualizaci polohy buď pouze pro GPS poskytovatele polohy (GPS provider), nebo i pro poskytovatele polohy ze sítě (network provider). Dále je potřeba vytvořit pomocnou třídu AutoLocator, které musíme vždy dávat vědět, v jakém stavu je průběh lokace. Tato pomocná třída se bude starat o aktualizace grafického prostředí aplikace – widgetu a notifikací. Navíc pomocí ní budeme spouštět periodickou úlohu pro automatické ukládání polohy. Způsob zobrazení notifikace je popsán na kódu v příloze IV. Další metody ve třídě PositionService se budou týkat jen formátování získaných souřadnic tak, aby se daly standardně zobrazit v aplikaci.

Již tedy víme, kde dojde k zaregistrování požadavků na aktualizaci polohy, ale ještě nám chybí nejzásadnější část kódu týkajícího se ukládání polohy, a tím je implementace metody onLocationChanged, která bude LocationManagerem zavolána na objektu třídy LocationListener ve chvíli, kdy poskytovatel polohy nabídne aktuální polohu. Jelikož se v této metodě provede několik zásadních příkazů, tak si ji zde uveďme celou. Komentáře k důležitým částem jsou přímo v kódu.

```
public synchronized void onLocationChanged(Location location) {
      // ziskame nastaveni aplikace
      SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(_context);
      // ukladame pouze polohu odlisnou od posledni, pokud si tak uzivatel nastavi
      if (prefs.getBoolean("onlyDistinct", true)) {
         if (lastPos != null && lastPos.getLatitude() == location.getLatitude() &&
         lastPos.getLongitude() == location.getLongitude()) {
            return;
         }
      }
      String provider = location.getProvider();
      // zrusime registraci zjistovani polohy
      _locationManager.removeUpdates(this);
      Position position = new Position();
      position.setLatitude(location.getLatitude());
      position.setLongitude(location.getLongitude());
      position.setLocateTime(new Date());
      position.setAltitude(location.getAltitude());
      // do pozice si poznamename, ktery provider nam polohu poskytl
      // tuto informaci pak budeme ukazovat uzivateli
      if (LocationManager.GPS_PROVIDER.equals(provider)) {
         position.setMark(1);
      }
      else if (LocationManager.NETWORK_PROVIDER.equals(provider)) {
         position.setMark(2);
      }
      else {
         position.setMark(0);
      }

      // zjistime lokalitu, pokud si tak uzivatel nastavi
      getLocality(position, prefs);
      // ulozime pozici do databaze
      _dataHelper.insertPosition(position);
      AutoLocator.setLocating(false);
      if (prefs.getBoolean("vibrate", false)) {
         Vibrator vibrator = (Vibrator)
         _context.getSystemService(Context.VIBRATOR_SERVICE);
         vibrator.vibrate(700);
      }
      // naplanujeme zmenu textu na widgetu
      scheduleButtonTextChange(_context);
      // zobrazime hlasku o ulozeni pozice
      Toast.makeText(_context, "Position saved", 4000).show();
   }
```

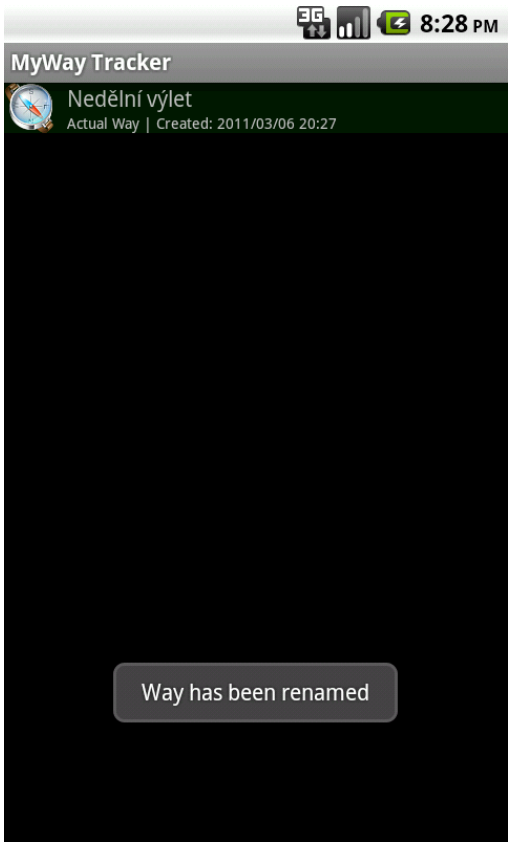
Pokud si shrneme akce, které se provedou ve výše uvedeném zdrojovém kódu, tak dojde k načtení uživatelských nastavení aplikace, vytvoří se objekt Position, do něhož se překopírují informace poskytnuté LocationProviderem a uloží se do něj i příznak, od kterého poskytovatele jsme informaci o poloze získali. Dále se získá a uloží do objektu Position adresa polohy, pokud si tak uživatel nastaví v nastavení. Pak se provede uložení objektu Position do databáze voláním na třídě DataHelper. Nakonec se uživatel notifikuje vibrací a textem na obrazovce a aktualizuje se text na widgetu.

Vibrační systém

Pro ovládání vibračního systému zařízení Android je potřeba z aplikačního kontextu vyzvednout systémovou službu VIBRATOR_SERVICE, která je vrácena jako instance třídy Vibrator. Vibraci přístroje pak spustíme zavoláním metody vibrate(long milliseconds) nebo vibrate(long[] pattern, int repeat). První pouze spustí vibraci na dobu určenou jediným argumentem. Druhá umožňuje číselným polem v prvním argumentu definovat sekvenci vibrací a mezer mezi nimi, a to střídavě, počínaje mezerou. Druhý argument dovoluje definovanou sekvenci provést libovolněkrát.

Notifikační textové hlášky

Operační systém Android nabízí vývojářům API pro jednoduchou notifikaci uživatele aplikace pomocí krátké textové vyskakovací zprávy na obrazovce. V balíku android.widget se nachází třída Toast, která nabízí metodu makeText(Context context, CharSequence text, int duration). Ve druhém argumentu této metody můžeme uvést libovolný text a ve třetím argumentu říci, jak dlouho se má uživateli tento text zobrazit.



Obrázek 6 - Textová notifikace Toast, zdroj: archiv autora

Výhodou této funkcionality je samozřejmě jednoduchost, ale také možnost zobrazit textovou notifikaci i ve chvíli, kdy není aktivní žádná obrazovka aplikace. V naší aplikaci takto máme možnost zobrazit informaci o tom, že byla automaticky uložena poloha i tehdy, když uživatel má aktivní jinou aplikaci nebo jen domovskou obrazovku operačního systému. Další výhodou také je, že uživatel není obtěžován žádným potvrzováním zobrazené zprávy.

Služba pro zjištění adresy dle polohy

V metodě onLocationChanged() objektu LocationListener jsme pro zjištěnou polohu volali metodu getLocality(…), abychom získali její adresu. Jakým způsobem se ale adresa reálně zjišťuje? Nejprve je vhodné zjistit, jestli je k dispozici připojení k Internetu, aby aplikace zbytečně nečekala na načtení adresy, pokud by byl poskytovatel internetového připojení vypnutý. Z kontextu aplikace získáme službu CONNECTIVITY_SERVICE.

```
ConnectivityManager conManager =
            (ConnectivityManager) _context.getSystemService(Context.CONNECTIVITY_SERVICE);
```

Zjišťování adresy pak uzavřeme do následující podmínky:

```
if (conManager.getActiveNetworkInfo() != null &&
               conManager.getActiveNetworkInfo().isAvailable()) {
```

Bude-li podmínka splněna, víme, že internetové připojení je k dispozici a máme možnost se dotazovat na adresu zjištěné polohy. K tomuto účelu slouží objekt Geocoder, k jehož instancování potřebujeme aplikační kontext a Locale.

```
Geocoder gc = new Geocoder(_context, Locale.getDefault());
```

Seznam adres na v dané lokalitě pak zjistíme takto:

```
List<Address> addresses = gc.getFromLocation(position.getLatitude(),
            position.getLongitude(), 1);
```

Pokud nás zajímá jen jedna adresa, vezmeme první prvek seznamu. Z něj je možné získat všechny informace týkající se adresy – ulici, město, PSČ, stát – které pomocí objektu Position uložíme do databáze.

4.3.4 Widget

V tuto chvíli máme připraveny služby pro práci s databází a pro zjišťování polohy. Proto nyní začněme realizovat části aplikace, se kterými uživatel bude pracovat. Zásadní částí aplikace bude widget.

Widget je miniaturní grafická komponenta aplikace, kterou je možné umístit na domácí obrazovku (homescreen) systému Android. Widgety jsou schopny získávat periodické aktualizace od své hostující aplikace, proto dělají domácí obrazovku interaktivní a dynamickou. Widget v naší aplikaci bude obnášet tři tlačítka, první pro jednorázové uložení polohy, druhé pro automatické periodické ukládání polohy a třetí bude spouštět hlavní aktivitu aplikace – seznam tras. Tlačítka budou vedle sebe, symbolizovány

ikonami a budou různě podbarvovány podle stavu. To znamená, že uživatel na první pohled pozná, zda je spuštěno automatické ukládání polohy, je li prostřední tlačítko (ikona) podbarveno modře. Neaktivní tlačítko bude podbarveno šedivou barvou. Při stisku se tlačítka na okamžik obarví červeně, aby uživatel bezpečně rozeznal, že tlačítko správně stiskl.

Tvorba widgetu v aplikaci Android zahrnuje tři nutné kroky:

1) Definici rozložení a vzhledu widgetu pomocí XML souboru v adresáři res/layout.

2) Vytvoření objektu AppWidgetProviderInfo, který se definuje v XML souboru a obsahuje metadata widgetu – odkaz na rozložení (layout), minimální šířku/výšku widgetu, obnovovací frekvenci.

3) Implementaci (rozšíření) třídy AppWidgetProvider umožňující programaticky reagovat na události vyvolané widgetem. Událostí může být akce na widgetu (např. kliknutí na tlačítko widgetu), přidání/odebrání widgetu na domácí obrazovce. Reakcí na přijatou událost může být volání služby v hostující aplikaci a/nebo změna grafického vzhledu widgetu (v našem případě podbarvení tlačítka při spuštění lokace).

Grafické rozložení

Začněme grafickým návrhem widgetu a určením jeho velikosti. Velikost widgetu je dána počtem buněk, které widget zabere na domovské obrazovce. Při definování widgetu se uvádí minimální šířka a výška v pixelech. Obvyklá domovská obrazovka operačního systému Android je tvořena maticí buněk o velikosti 4x4 buňky. Jelikož bude náš widget tvořen třemi tlačítky, tak se jako ideální velikost nabízí 3 buňky, vodorovně vedle sebe. Každá buňka bude přibližně odpovídat velikosti jednoho tlačítka. Abychom dodrželi maximální kompatibilitu s různými rozlišeními displejů, musí minimální velikost widgetu vycházet z nejhoršího případu, což je čtvercová buňka o délce hrany 74 pixelů. Minimální šířku widgetu pak spočítáme podle jednoduchého vzorce (počet buněk * 74) – 2. Dva

pixely se odčítají z důvodu zaokrouhlovací chyby při počítání s celočíselnou šířkou v pixelech. Widget tvořený třemi buňkami bude mít minimální šířku (3 * 74) – 2, tedy 220 bodů. Výška je jen jedna buňka, takže minimální výška widgetu bude 72 bodů. Rozložení (layout) definujeme v souboru widget.xml, který vytvoříme v adresáři res/layout. Princip je stejný jako u definice obrazovky. Jelikož se náš widget skládá ze třech tlačítek umístěných vedle sebe, zvolíme lineární rozložení s horizontální orientací.

```
<LinearLayout android:id="@+id/LinearLayout02"
   android:orientation="horizontal" android:gravity="center"
   android:layout_width="fill_parent" android:layout_height="fill_parent"
   xmlns:android="http://schemas.android.com/apk/res/android">
```

Uvnitř elementu vyjmenujeme objekty tlačítek.

```
   <Button android:layout_width="64dip" android:layout_marginRight="4dip"
      android:layout_height="64dip" android:id="@+id/button_locate_now"
      style="@style/Button_now"></Button>
   <Button android:visibility="gone" android:layout_width="64dip"
      android:layout_marginRight="4dip" android:layout_height="64dip"
      android:id="@+id/button_locate_now_act" style="@style/Button_now_act"></Button>
   <Button android:layout_width="64dip" android:layout_height="64dip"
      android:id="@+id/button_auto_location" style="@style/Button_auto"></Button>
   <Button android:visibility="gone" android:layout_width="64dip"
      android:layout_height="64dip" android:id="@+id/button_auto_location_act"
      style="@style/Button_auto_act"></Button>
   <Button android:layout_width="64dip" android:layout_marginLeft="4dip"
      android:layout_height="64dip" android:id="@+id/button_data"
      style="@style/Button_map"></Button>
```

Na první pohled je zvláštní, že máme ve widgetu tři tlačítka, ale v rozložení jich definujeme pět. Ale povšimněme si, že druhé a čtvrté tlačítko má nastaven atribut visibility na hodnotu „gone“, která způsobí, že tlačítko je skryté. Toto je způsob, jak realizovat změnu podbarvení tlačítka po určitou dobu vykonávání nějaké akce. První dvě tlačítka budou vykonávat akce jednorázového uložení pozice, což zabere kratší chvíli a opakovaného automatického ukládání pozice – to bude trvat, dokud akci uživatel manuálně nevypne opětovným stisknutím tlačítka. Aktivní, a k němu skryté tlačítko má rozdílný atribut style, čímž se odliší jejich vzhled. Stiskem tlačítka programaticky přesuneme hodnotu „gone“ atributu visibility ze skrytého tlačítka na aktivní a tím dojde k jejich prohození. Ještě si doplňme význam zbývajících použitých atributů. Atribut Layout_width definuje šířku grafického elementu (tlačítka), Layout_height obdobně jeho výšku. Atributem Layout_marginRight říkáme, jak velký bude okraj na dané straně elementu (napravo od tlačítka vznikne mezera široká 4 body). Hodnota atributu style odkazuje do souboru res/values/themes.xml, ve kterém budeme definovat vzhled samotných tlačítek, viz XML fragment níže:

```
<style name="Button_now" parent="@android:style/Widget.Button">
     <item name="android:gravity">center_vertical|center_horizontal</item>
     <item name="android:background">@drawable/cust_button_now</item>
     <item name="android:focusable">true</item>
     <item name="android:clickable">true</item>
 </style>

 <style name="CustomButtonNow" parent="android:style/Theme.NoTitleBar">
     <item name="android:buttonStyle">@style/Button_now</item>
 </style>
```

Definice stylů umožňuje vytvořit vzhled určité grafické komponenty a aplikovat ho libovolněkrát na komponenty pomocí odkazu. Docílíme tak odstranění duplicit a oddělení definice grafických parametrů od rozložení pohledu. Styl Button_now říká, že grafický prvek, na který se styl aplikuje, bude vertikálně i horizontálně centrovaný, bude mít na pozadí obrázek cust_button_now. Parametrem focusable říkáme, že je možné prvek vybrat i jinak než kliknutím, například trackballem přístroje. Poslední parametr clickable udělá grafický prvek aktivním.

Takto navržený widget má následující podobu:

Zdroj: archiv autora

Pokud je aktivní automatické ukládání polohy, bude prostřední tlačítko podbarveno modře, dokud uživatel autolokaci nevypne opětovným stisknutím prostředního tlačítka. Pokud zároveň dochází k reálnému načítání polohy a ukládání do databáze, podbarví se i tlačítko první tak, jak je vidět na následujícím výřezu:

Zdroj: archiv autora

Metadata

Metadata widgetu se obvykle definují pomocí XML souboru, ve kterém se uvádí jediný element <appwidget-provider>. V tomto elementu nastavíme základní vlastnosti widgetu, jako je jeho velikost, obnovovací frekvence, název a rozložení.

Velikost widgetu jsme si již určili v sekci Grafické rozložení, takže zde hodnoty minimální šířky a minimální výšky nastavíme do atributů minWidth a minHeight. Název widgetu bude stejný jako název aplikace – MyWay Tracker. Dále nastavíme rozložení

widgetu, které již máme připravené a definované v souboru res/layout/widget.xml, proto stačí uvést odkaz na zdroj @layout/widget do atributu initialLayout. Ještě nám zbývá nastavit obnovovací frekvenci widgetu. Obnovovací frekvence udává, jak často se bude provádět obnova widgetu a volání metody onUpdate(...) na implementující třídě widgetu AppWidgetProvider. Někdy může být žádoucí aktualizovat widget častěji, například u widgetu přehrávače audio souborů, který musí stále ukazovat aktuální data o stavu přehrávání skladby atd. Časté obnovování ale přináší nevýhodu spočívající ve větší spotřebě energie přístroje, proto je vhodnější, pokud je to možné, snížit obnovovací frekvenci na co nejnižší hodnotu. Pokud bude docházet k obnovení widgetu jednou za hodinu, je dopad na spotřebu energie prakticky zanedbatelný. V naší aplikaci nemusíme obnovování widgetu řešit, protože uložení polohy je jednorázová událost a co se týká automatického ukládání polohy, tak použijeme komponentu časovač (Timer) a při každé akci provedeme obnovu widgetu manuálně, ve smyslu volání metody pro obnovení widgetu programaticky. Proto nastavíme vysokou hodnotu časové periody mezi obnoveními.

Konfigurace metadat bude uložena v souboru res/xml/widget_provider.xml a její výsledná podoba pro aplikaci MyWay Tracker je následující:

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
   android:initialLayout="@layout/widget" android:minWidth="220dp"
   android:minHeight="72dp" android:label="MyWay Tracker"
   android:updatePeriodMillis="12000000" />
```

Implementace třídy AppWidgetProvider

Logika widgetu a reakce na události, to vše je popsáno metodami této třídy. AppWidgetProvider je potomkem třídy BroadcastReceiver, a tudíž je schopna odchytávat události vysílané widgetem a reagovat na ně. Mezi standardní události patří update, enable, disable, delete. Update je přijímán ve chvíli, kdy uživatel přidá widget na domovskou obrazovku, nebo když nastane doba obnovy stanovená periodou obnovení. Často je metoda onUpdate používána k základnímu nastavení, spouštění služeb nebo definování obsluhy událostí. Metody onEnabled a onDisabled obsluhují odpovídající události enable a disable, které jsou generovány při vytvoření první instance widgetu, respektive při odstranění poslední instance widgetu. Pokud uživatel přidá widget na domovskou obrazovku vícekrát, je událost enable vyslána pouze jednou, proto se metoda onEnabled hodí pro kód, u kterého požadujeme, aby byl vykonán v rámci aplikace pouze jednou. Oproti metodě

onDisabled, která se zavolá v rámci několika instancí widgetu pouze při odstranění posledního, metoda onDeleted je volána při odstranění každé instance widgetu.

Třída AppWidgetProvider nabízí ještě jednu metodu, která je specifická tím, že je volána při každé události. Jedná se o metodu onReceive(Context, Intent) a není nutné ji pro widget implementovat, pokud nechceme reagovat na události jiné, než které jsme si vyjmenovali výše. V našem případě ale bude tato metoda velice důležitá, protože widget nabízí aktivní tlačítka, jejichž stisknutí vyvolá naši definovanou událost a my potřebujeme tuto událost odchytit, identifikovat a reagovat na ni. Jak víme z grafického návrhu, widget obsahuje tři tlačítka. Poslední tlačítko spouští hlavní aktivitu aplikace MyWay Tracker, takže není potřeba vysílat a odchytávat nějakou událost, framework nám jednoduše dovolí nastavit, že kliknutím na toto tlačítko se spouští aktivita. Ostatní dvě tlačítka ale musí být schopné spouštět akci na pozadí, v hostující aplikaci, proto také budou po stisknutí vysílat událost, kterou v metodě onReceive obdržíme, rozpoznáme a danou akci spustíme. Pro práci s událostmi použijeme tři třídy objektů. První je třída Intent, zastupující samotnou událost. Té nastavíme název akce, kterou událost přináší, a podle tohoto názvu také událost identifikujeme po jejím odchycení. Objekt třídy Intent může být konstruktorem vytvořený pro specifickou komponentu, což v našem případě bude buď aktivita, nebo widget samotný. Dalším objektem je PendingIntent, pomocí kterého zaregistrujeme událost k určitému typu akce. PendingIntent nabízí tři statické metody pro vytvoření typu akce, getActivity(...), getBroadcast(...) a getService(...). Podle názvu metod už je zřejmé, pro jaký typ akce událost registrujeme. Je to buď spouštění aktivity, což použijeme pro poslední tlačítko, nebo šíření události systémem, anebo spuštění služby. První dvě tlačítka budou šířit událost systémem, takže použijeme getBroadcast. Posledním objektem, který potřebujeme k registraci události je RemoteView. Ten zastupuje uživatelský pohled widgetu a událost kliknutí je na něm potřeba zaregistrovat voláním metody setOnClickPendingIntent(ViewId, PendingIntent). Opět si zde pro názornost uveďme alespoň část zdrojového kódu metody onUpdate, konkrétně registraci události spuštění autolokace, která bude šířena systémem a události pro spuštění hlavní aktivity.

```
Intent intentAutolocation = new Intent(context, MyWayWidget.class);
intentAutolocation.setAction(MyWayWidget.ACTION_WIDGET_AUTOLOCATION);
PendingIntent pendingIntentAutolocation = PendingIntent.getBroadcast(context, 0,
intentAutolocation, 0);
views.setOnClickPendingIntent(R.id.button_auto_location, pendingIntentAutolocation);

Intent intentData = new Intent(context, DataActivity.class);
intentData.setAction("android.intent.action.MAIN");
PendingIntent pendingIntentData = PendingIntent.getActivity(context, 0, intentData, 0);
views.setOnClickPendingIntent(R.id.button_data, pendingIntentData);
```

Takto se nám po přidání instance widgetu zaregistrují potřebné události na jednotlivá tlačítka. Nyní je musíme v metodě onReceive odchytit a zareagovat na ně. Následující část kódu ukazuje, jakým způsobem v metodě onReceive identifikujeme událost a spustíme následnou akci.

```
if (intent.getAction().equals(ACTION_WIDGET_AUTOLOCATION)) {
   DataHelper dh = new DataHelper(context);
   if (dh.getActualWay() == 0) {
      Toast.makeText(context, "First, create a new Way", 6000).show();
   } else {
      if (AutoLocator.isAutolocationRunning()) {
         Toast.makeText(context, "Tracking stopped", 4000).show();
         AutoLocator.stopAutolocation(context);
      } else {
         SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(context);
         Toast.makeText(context, "Tracking started", 4000).show();
         AutoLocator.startAutolocation(context, new DataHelper(context), new
         Integer(prefs.getString("prefPeriodKey", "60")));
      }
   }
}
```

Objekt intent se do metody onReceive dostane jako argument. První podmínka zkoumá, zda je událost daného typu. Pokud ano, provede se inicializace databázové pomocné třídy. Dále se ověří, zda má uživatel vytvořenu alespoň jednu trasu, jinak by nebylo možné ukládat pozice. Jestliže trasu vytvořenu doposud nemá, zobrazí se textová hláška, že je nejprve potřeba trasu vytvořit. Pokud trasu vytvořenu má, provede se buďto zapnutí autolokace, nebo vypnutí, v závislosti na předchozím stavu, který je držen v objektu AutoLocator. Pokud se autolokace spouští, zjistí se nejprve uživatelem nastavená perioda pro automatické ukládání pozice a ta se předá do metody startAutolocation(...). Obdobně budeme detekovat i druhou událost pro jednorázové uložení polohy.

Nyní máme kompletně vytvořen widget, který je připraven pro vložení na domovskou obrazovku přístroje. Avšak zbývá provést ještě jednu operaci, aby aplikace o widgetu vůbec věděla, a aby sama o sobě od widgetu očekávala nějaké události. Touto operací je definice přijímače (receiveru) v manifestu aplikace. Do souboru AndroidManifest.xml tedy vložíme následující kus XML kódu:

```
<receiver android:name="cz.byteworks.android.myway.widget.MyWayWidget"
         android:label="MyWay Tracker">
      <intent-filter>
         <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
      </intent-filter>
      <meta-data android:name="android.appwidget.provider"
      android:resource="@xml/widget_provider" />
</receiver>
```

Tímto říkáme, že objekt MyWayWidget bude sloužit jako přijímač událostí pro akce typu APPWIDGET_UPDATE. V metadatech provedeme odkazem @xml/widget_provider

napojení na zdroj definující vzhled a rozložení (Pohled – View) widgetu. Nyní již máme přichystáno vše, aby mohl být widget použit a zároveň aby aktivně reagoval na události kliknutí na tlačítko.

4.3.5 Aktivity

4.3.5.1 Hlavní aktivita

Vstupním bodem pro ukázkovou aplikaci je widget, ale ten slouží pouze jako ovladač a indikátor stavu aplikace. Pokud chceme na obrazovce zobrazovat okna aplikace samotné, musíme definovat příslušné aktivity. Aplikace se bude rozkládat mezi několik aktivit, respektive obrazovek. V této kapitole si ukážeme, jakým způsobem se realizuje hlavní aktivita aplikace a tím se seznámíme s vytvářením aktivit obecně.

Začněme u souboru AndroidManifest.xml, ve kterém je nutné všechny aplikační aktivity zaregistrovat. Jednoduše do elementu application vložíme následující kód:

```
<activity android:name="DataActivity">

</activity>
```

Takto máme zaregistrovánu aktivitu DataActivity, která bude sloužit k zobrazování vytvořených tras a hlavního menu aplikace a zároveň bude hlavní aktivitou. Název aktivity také určuje název třídy implementující logiku aktivity. Aby aktivita DataActivity byla operačním systémem Android brána jako hlavní aktivita, musíme ji tuto vlastnost dodat. To učiníme elementem <intent-filter>, který dané komponentě přiřazuje určité schopnosti. Předchozí XML zápis obohatíme takto:

```
<activity android:name="DataActivity">
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

Element action s názvem android.intent.action.MAIN říká, že tato aktivita bude hlavní aktivitou aplikace a její vstupním bodem. S tím souvisí druhý element, category, s názvem android.intent.category.LAUNCHER. Ten totiž navíc aktivitu zařadí mezi ty, jenž se dají spustit z hlavního menu aplikací systému Android, tzv. Launcheru. V podstatě to není nic jiného, než přidání zástupce (ikony) do seznamu aplikací ke spuštění, které máme na daném zařízení.

Dalším krokem je návrh grafického rozložení obrazovky představující uživatelské rozhraní dané aktivity. Jak už jsme naznačili, naše hlavní aktivita bude primárně zobrazovat seznam tras, které si uživatel vytvořil, sekundárně pak bude nabízet kontextové menu pro každou trasu a hlavní menu aplikace pro vytvoření nové trasy a vstup do nastavení. Začněme tím, že v adresáři res/layout vytvoříme nový XML soubor data_activity.xml. Jako hlavní rozložení zvolíme lineární, které je nejjednodušší a grafické prvky se v něm logicky skládají za sebou. My zde budeme mít pouze jediný grafický prvek – seznam tras. Proto do lineárního rozložení vložíme pohled ListView. Rozložení komponent na obrazovce aktivity je tedy definováno následujícím XML:

```
<LinearLayout android:id="@+id/DataTableLayout"
   android:layout_width="fill_parent" android:layout_height="fill_parent"
   xmlns:android="http://schemas.android.com/apk/res/android">
         <ListView xmlns:android=http://schemas.android.com/apk/res/android
         android:id="@+id/dataListView" android:layout_width="fill_parent"
         android:layout_height="wrap_content">
   </ListView>
</LinearLayout>
```

Identifikátor rozložení je DataTableLayout a identifikátor komponenty seznamu dataListView. Co potřebujeme dále, je grafické rozložení položky seznamu. To už bude mírně složitější, protože položka seznamu se bude skládat z obrázku v levé části a tabulky o dvou řádcích v části pravé. Řádky tabulky budou navíc obsahovat textové pohledy. Rozložení položky seznamu by mělo ve výsledku vypadat přibližně takto:

obrázek	Název trasy
	Informace o trase, datum a čas vytvoření

Jako rozložení položky zvolíme opět lineární, obsahující pohled pro zobrazování obrázků ImageView a komponentu pro tabulkové rozložení TableLayout. Tyto dvě části se nám díky lineárnímu rozložení zobrazí vedle sebe tak, jak potřebujeme. Nyní musíme tabulkové rozložení odpovídající pravé části položky seznamu rozdělit do dvou řádků vyplněných textem tak, jak je požadováno. K tomu se používá komponenta TableRow. Pro Zobrazení textu slouží pohled TextView. Poskládání rozložení je zřejmé z následujícího XML, které uložíme do souboru res/layout/data_item.xml.

```
<LinearLayout android:id="@+id/dataRow"
   android:layout_width="fill_parent" android:layout_height="fill_parent"
   xmlns:android="http://schemas.android.com/apk/res/android">
   <ImageView android:id="@+id/dataImage" android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:paddingRight="8dip"></ImageView>
   <TableLayout android:id="@+id/dataTable"
      android:layout_width="wrap_content" android:layout_height="wrap_content">
      <TableRow android:id="@+id/dataTableRow1"
         android:layout_width="wrap_content" android:layout_height="wrap_content">
```

```
            <TextView android:id="@+id/dataText1" android:layout_width="wrap_content"
               android:layout_height="wrap_content"></TextView>
         </TableRow>
         <TableRow android:id="@+id/dataTableRow2"
            android:layout_width="wrap_content" android:layout_height="wrap_content">
            <TextView android:id="@+id/dataText2" android:layout_width="wrap_content"
               android:layout_height="wrap_content" android:textColor="#C1C1C1"
               android:textSize="9dip"></TextView>
         </TableRow>
      </TableLayout>
</LinearLayout>
```

Celé rozložení položky seznamu bude v aplikaci pod identifikátorem dataRow, obrázek pak pod dataImage, tabulka dataTable a její řádky dataTableRow1 a dataTableRow2. Pro naplnění textových polí se budeme odkazovat na identifikátory dataText1 a dataText2. Některým komponentám nastavíme určité vlastnosti. Tak například velikost písma spodního řádku tabulky chceme menší než je výchozí, proto nastavíme pro dataText2 atribut textSize na 9 bodů a změníme i barvu písma. Všechny povolené atributy pro dané komponenty nalezneme vždy v dokumentaci API (17).

Nyní musíme pro definovanou grafiku naprogramovat potřebnou logiku chování a plnění položek daty. V balíku cz.byteworks.android.myway.activities vytvoříme třídu DataActivity (v našem případě odpovídá názvu aktivity v manifestu) rozšiřující třídu android.app.Activity. Ve třídě DataActivity budeme implementovat několik metod, počínaje nejdůležitější – onCreate. Ta je volána jako první při spuštění aktivity a mimo jiné v ní říkáme, která definice grafického rozložení se má pro aktivitu použít. Grafické rozložení (layout) vybereme metodou setContentView(R.layout.data_activity); jejímž argumentem je odkaz na celkové rozložení obrazovky.

Pro plnění dat seznamu nabízí aplikační framework systému Android rozhraní ListAdapter. V naší aplikaci rozšíříme abstraktní implementaci tohoto rozšíření – android.widget.ArrayAdapter. Vytvoříme tedy třídu cz.byteworks.android.myway.adapter.DataArrayAdapter rozšiřující ArrayAdapter a naimplementujeme metodu getView(int position, View convertView, ViewGroup parent). Tato metoda vrací objekt View odpovídající položce seznamu na pozici dané argumentem position. Proto je potřeba do konstruktoru adapteru předat celý seznam, který bude zobrazován anebo se v této metodě na každou položku zvlášť dotazovat do databáze. Jelikož neočekáváme příliš velká kvanta dat, zvolíme možnost první a předáme seznam (objekt třídy ArrayList) v konstruktoru třídy DataArrayAdapter. V těle metody getView už pak pouze poskládáme programaticky komponenty a naplníme daty v požadovaném formátu. Celou metodu naleznete v příloze I.

Zbývá nám vrátit se k třídě DataActivity a naimplementovat potřebné metody. Jako první sestavíme tělo metody onCreate. Její funkce v této aktivitě spočívá v nastavení celkového rozložení obrazovky, načtení dat z databáze do seznamu, inicializace adaptéru a nastavení obsluhy kliknutí na položku seznamu. Celou metodu ukazuje následující zdokumentovaný kód.

```
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // nastavime celkove rozlozeni obrazovky
      setContentView(R.layout.data);
      // ziskame pohled graficke komponenty seznamu
      lv = (ListView) findViewById(R.id.dataListView);
      // z databaze nacitame seznam tras
      DataHelper dh = new DataHelper(this);
      wayList = dh.getAllWays(true);
      // inicializujeme adapter a predavame seznam polozek
      dataAdapter = new DataArrayAdapter<Way>(this, R.layout.data_item, wayList,
      getLayoutInflater());
      lv.setAdapter(dataAdapter); // nastavujeme adpater seznamu
      // pridame obsluhu kliknuti
      lv.setOnItemClickListener(new OnItemClickListener() {
         // metoda se provede pri kliknuti na polozku seznamu
         public void onItemClick(AdapterView<?> arg0, View view, int arg2, long arg3) {
            // zobrazime kontextove menu k dane polozce
            view.showContextMenu();
         }
      });
      // zaregistrujeme komponentu k pouzivani kont. menu
      registerForContextMenu(lv);
}
```

Nyní jsme již schopni zobrazit obrazovku aktivity, která ale kromě seznamu tras nemá žádnou další funkci. Proto je potřeba naimplementovat další metody, týkající se kontextového menu jednotlivých položek a nakonec hlavního menu aplikace.

Nejprve se soustřeďme na kontextové menu položek seznamu. Menu bude mít čtyři položky – Open map pro zobrazení trasy na mapě, Open way pro zobrazení seznamu uložených poloh, Rename way pro přejmenování trasy a Delete way pro smazání trasy. Tuto strukturu si nadefinujeme v souboru res/menu/way_context_menu.xml a poté ji použijeme jako zdroj pro vytvoření menu. Struktura menu v XML je tato:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/openMap" android:title="@string/context_menu_open_map"></item>
   <item android:id="@+id/openWay" android:title="@string/context_menu_open_way"></item>
   <item android:id="@+id/renameWay" android:title="@string/context_menu_rename_way"></item>
   <item android:id="@+id/deleteWay" android:title="@string/context_menu_delete_way"></item>
</menu>
```

V XML jsou pomocí elementu <item> vyjmenovány položky menu, kterým přiřazujeme určité ID a textový řetězec, jenž bude titulkem položky menu. Zdroj poté načteme v metodě onCreateContextMenu třídy DataActivity pomocí objektu MenuInflater.

```
MenuInflater inflater = getMenuInflater();
inflater.inflate(R.menu.way_context_menu, menu);
```

Od této chvíle se bude po dlouhém stisknutí položky seznamu – trasy – zobrazovat kontextové menu v takové podobě, jaké jsme si definovali. Akci, kterou spouští položka

menu, určíme v metodě onContextItemSelected(MenuItem item). Argument metody, objekt MenuItem, v sobě přináší informaci o tom, která položka menu byla vybrána a kterého záznamu v seznamu se kontextové menu týkalo. Díky tomu jsme schopni určit trasu a požadovanou akci. Celou metodu ze třídy DataActivity je možné nalézt v příloze II.

Obdobným způsobem vytvoříme i hlavní menu aplikace, které se bude uživateli zobrazovat, pokud na obrazovce hlavní aktivity stiskne tlačítko Menu svého přístroje. Jak již bylo řečeno dříve, hlavní menu bude obsahovat položku pro vytvoření nové trasy, položku pro přechod na nastavení aplikace (preferenční aktivitu) a nakonec položku pro zobrazení dialogového okna s informacemi o aplikaci a vývojáři. Tímto způsobem si tedy ukážeme, jak lze pro libovolnou aktivitu definovat menu. Opět si nejprve vytvořme strukturu menu jako zdroj v XML.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/menu_add_new_path" android:icon="@drawable/ic_menu_add"
      android:title="@string/menu_add_new_path" />
   <item android:id="@+id/menu_prefs" android:icon="@drawable/ic_menu_preferences"
      android:title="@string/menu_prefs" />
   <item android:id="@+id/menu_about" android:icon="@drawable/ic_menu_info_details"
      android:title="@string/menu_about" />
</menu>
```

Jediný rozdíl oproti kontextovému menu je ten, že v hlavním menu je potřeba zobrazovat i ikonu položky. Ve třídě DataActivity menu zaregistrujeme následující metodou, kterou takto přetížíme.

```
public boolean onCreateOptionsMenu(Menu menu) {
      MenuInflater mi = getMenuInflater();
      mi.inflate(R.menu.data_menu, menu);
      return true;
}
```

Obsluha menu je realizována metodou onOptionsItemSelected, kde opět využijeme objekt MenuItem.

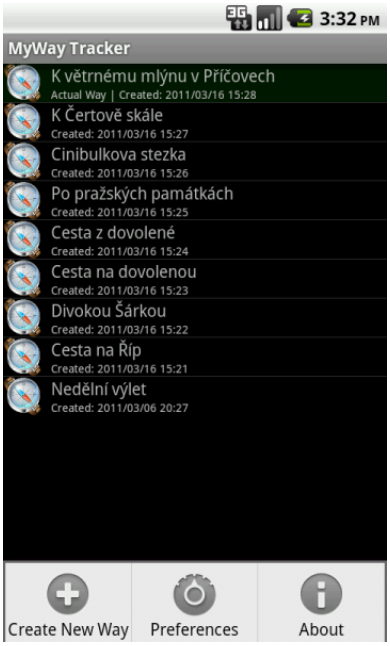
```
public boolean onOptionsItemSelected(MenuItem item) {
      switch (item.getItemId()) {
      case R.id.menu_add_new_path:
         // volame dialog pro vytvoreni nove trasy
         menuPopupNewPath();
         break;
      case R.id.menu_prefs:
         // spoustime preferencni aktivitu
         Intent intent = new Intent(this, Preferences.class);
         startActivity(intent);
         break;
      case R.id.menu_about:
         // zobrazime dialog s info o aplikaci
         createAboutDialog();
         break;
      }
      return true;
}
```

Pro úplnost si ještě uveďme výstřižek zdrojového kódu, kterým zajistíme zobrazení dialogového okna, které je v aplikacích velice často používané pro rozličné uživatelské vstupy, informativní texty nebo pro potvrzovací dotazy. Dialogové okno s informacemi o aplikaci zobrazíme voláním jednoduchého kódu:

```
private void createAboutDialog(String text) {
      TextView message = new TextView(this);
      message.setText(text);

      new AlertDialog.Builder(this).setTitle("About").setCancelable(true)
            .setIcon(R.drawable.ic_launcher)
            .setPositiveButton("Close", null).setView(message).create()
            .show();
}
```

Vytvořením instance objektu třídy AlertDialog.Builder a zavoláním metody show() na tomto objektu se dialogové okno ihned uživateli zobrazí.

Výsledkem implementace naší první a zároveň hlavní aktivity je následující obrazovka, kde můžeme námi vytvořený seznam tras v požadovaném tvaru a ve spodní části hlavní menu aplikace.



Obrázek 7 - Obrazovka hlavní aktivity, zdroj: archiv autora

Zcela obdobným způsobem je v aplikaci implementována aktivita a obrazovka pro zobrazování detailu trasy. Na detailu trasy opět bude seznam, tentokrát ale nebudeme načítat trasy ale uložené polohy. Pomocí kontextového menu se nabídnou obdobné

akce s položkami seznamu, včetně přechodu na mapovou aktivitu, kterou si popíšeme později.

4.3.5.2 Preferenční aktivita

Preferenční aktivitou budeme chápat obrazovku, kde si uživatel má možnost konfigurovat chování aplikace. Výhodou aplikačního frameworku operačního systému Android je jednoduchost způsobu, jakým se dá preferenční aktivita do aplikace implementovat. Vývojář není nucen obstarávat úložiště nastavených preferencí a zajišťovat jeho konzistenci, protože vše za něj udělá právě aplikační framework.

Vytvoření preferenční aktivity sestává ze dvou kroků. Prvním z nich je definice struktury proměnných, které si bude moci uživatel nastavovat. To se provádí pomocí zdroje v XML souboru, podobně jako tomu bylo u menu. Součástí definice je typ preference (výběr ze seznamu, zaškrtávací políčko, textové pole, atd.), který je určen použitým elementem. V atributech elementu pak dále definujeme název, popisek, klíč, pod kterým preferenci načteme v aplikační logice a nakonec obvykle výchozí hodnotu. Výchozí hodnota je dané preferenci nastavena po nainstalování aplikace a trvá do té doby, dokud ji uživatel poprvé nezmění. Aplikační framework nabízí ještě další atributy pro definici preferencí, ale my si v naší aplikaci vystačíme s těmito a lze i říci, že jsou dostačující pro většinu aplikací obecně. Jakmile je struktura definována, nezbývá než vytvořit aktivitu jako takovou, což obnáší implementaci třídy dědící z třídy PreferenceActivity a její zaregistrování v aplikačním manifestu AndroidManifest.xml.

Naše aplikace bude potřebovat celkem pět preferencí. První bude typu seznam a nabídne uživateli možnost vybrat si z několika předdefinovaných velikostí periody pro automatické ukládání polohy. Ostatní budou jen zaškrtávací pole, takže v Java kódu budou reprezentovány hodnotou boolean. Jedná se o zapnutí/vypnutí vibrací při uložení polohy, zapnutí/vypnutí ukládání lokality (adresy) polohy, zapnutí/vypnutí používání určování polohy pomocí sítě a nakonec zapnutí/vypnutí možnosti ukládat pouze polohy rozdílné od poslední uložené. Strukturu budeme definovat v souboru res/xml/preferences.xml. Pro názornost si uveďme jeho obsah v příloze III.

Třída Preferences, která od ostatních tříd implementujících logiku aktivit liší pouze předkem, který není třída Activity ale PreferenceActivity. Jediné co je v ní pro naše účely

potřeba implementovat je metoda onCreate, a to velice jednoduchým způsobem. Pouze v ní zavoláme metodu addPreferencesFromResource(int resId) a jako argument předáme identifikátor zdroje, ve kterém máme strukturu preferenční obrazovky.

```
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      addPreferencesFromResource(R.xml.preferences);
}
```

Nakonec aktivitu zaregistrujeme v manifestu přidáním následujícího řádku:

```
<activity android:name="Preferences" />
```

Od této chvíle je možné preferenční aktivitu v aplikaci spustit vybráním příslušné položky v hlavním menu. Hodnoty nastavené uživatelem se v kódu získají pomocí objektu SharedPrefs, který je dostupný přes volání statické metody getDefaultSharedPreferences(Context context) objektu PreferenceManager. Objekt SharedPrefs poskytuje metody pro načtení nastavených hodnot pro různé typy preferencí (boolean, String, atd.), kterým předáme klíč a výchozí hodnotu, jenž se použije, pokud uživatel preferenci doposud nenastavil a preference nemá v XML definovánu svou výchozí hodnotu. Příkladem může být získání uživatelovi preference týkající se používání sítě pro určování polohy:

```
PreferenceManager.getDefaultSharedPreferences(ctx).prefs.getBoolean("useNetwork", false);
```

Uživatelská nastavení přetrvávají v aplikaci po dobu, po kterou je aplikace na přístroji nainstalována. Smažou se jen v případě, když aplikaci uživatel odinstaluje, nebo smaže uživatelská data aplikace ve správci aplikací.

4.3.5.3 Mapová aktivita

Poslední částí naší ukázkové aplikace je aktivita, díky níž budeme uživateli zobrazovat jeho trasy a uložené polohy na mapě. Operační systém Android používá pochopitelně mapové podklady z aplikace Google Maps a zpřístupňuje je vlastním API, které ale není součástí aplikačního frameworku, nýbrž externí knihovnou Google API. Klíčové třídy zaobalující rozhraní Google Maps jsou umístěny v balíku com.google.android.maps a odstiňují vývojáře od nutnosti obstarávat stahování, ukládání a vykreslování map a ovládacích prvků mapy. Pro definici rozložení obrazovky je v tomto balíku připravená třída MapView a pro implementaci třídy mapové aktivity se používá třída MapActivity.

Jak již bylo řečeno, pro vývoj aplikace používající mapy potřebujeme kromě standardního Android SDK ještě navíc přídavek Google API. Další věcí, bez které není možné mapy načítat, je zaregistrování otisku certifikátu, který použijeme pro podepsání aplikace, na stránkách Google Code (18). Společnost Google si tím zajistí souhlas s podmínkami používání rozhraní Google Maps API. Během vývoje a testování aplikace na emulátoru (AVD) je automaticky používán debug certifikát, jehož umístění v ADT zjistíme v nastavení Android prostředí (Windows > Prefs > Android > Build). Otisk certifikátu se získá pomocí nástroje keytool, jenž je součástí každé distribuce Java. Vývojový debug certifikát má alias „androiddebugkey“ a hesla k úložišti a klíči jsou obě „android“. Spustíme tedy tento příkaz

```
$ keytool -list -alias androiddebugkey \
-keystore <path_to_debug_keystore>.keystore \
-storepass android -keypass android
```

a vypíše se otisk certifikátu. Ten následně zadáme na stránce Google Code určené k registraci služby Google Maps API a obratem získáme klíč k rozhraní Google Maps (API Key). Tento klíč je nutné zadat v definici rozložení mapové aktivity, jinak není možné mapové API používat. Atribut, do kterého se klíč zadává, bude vidět v XML souboru rozložení mapové aktivity. Jakmile budeme aplikaci sestavovat pro distribuci, musíme totéž provést s vlastním certifikátem, který použijme pro podepsání výsledného APK souboru.

Dříve než začneme se samotnou mapovou aktivitou, shrňme si, jaké funkce po ni budeme požadovat. V první řadě je to zobrazení trasy spojením uložených poloh výraznými spojnicemi. Každý bod (poloha) na dané trase musí být vyznačen, a pokud na něj uživatel klikne, zobrazí se datum a čas uložení polohy. Dále požadujeme standardní mapové funkce jako je posuv mapy, oddalování, přibližování a přepínání na družicový pohled.

Opět začneme definicí rozložení aktivity, které uložíme do souboru res/layout/map_activity.xml. Hlavním typem rozložení bude lineární, do kterého vložíme jediný pohled MapView.

```
<com.google.android.maps.MapView
   android:id="@+id/mapview"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:clickable="true"
   android:apiKey="0k4YR1ogKpLJ7Xs4bVpOaypm9tdT7kjZ67cGFRmn983" />
```

Oproti pohledům jiného typu není tento nijak zvlášť odlišný, pouze uvedeme navíc dva atributy, a to clickable, kterým řekneme, že mapa bude interaktivní a apiKey, do kterého zadáme získaný klíč k Google Maps API. S tímto si v naší aplikaci vystačíme a není potřeba definovat nic jiného. Mnohem složitější práce nás čeká v implementaci třídy aktivity a třídy pro překryvnou vrstvu.

Nejprve si vytvořme třídu aktivity MyWayMapActivity a učiňme ji potomkem třídy MapActivity. Pro prosté zobrazení mapy na obrazovce aktivity by nám stačilo zavolat v metodě onCreate metodu setContentView(R.layout.map_activity), čímž bychom načetli zdroj rozložení, který jsme výše definovali a mapa by se nám na obrazovce opravdu načetla. My ale požadujeme další funkce a zobrazení bodů a spojnic. Grafické prvky jako jsou body, jejich spojnice, ikony atd. jsou skládány do jakýchsi vrstev, které pokládáme na mapu. Proto je možné vytvořit zvlášť vrstvu bodů samotných, dále vrstvu spojnic těchto bodů a do další vrstvy přidat například popisky bodů. Díky tomu pak máme možnost jednotlivé vrstvy vypínat a zapínat a tím měnit detail zobrazovaných informací na mapě.

V naší aplikaci nepotřebujeme oddělovat spojnice od bodů a budeme vždy zobrazovat uložené polohy dohromady s jejich spojnicemi, proto použijeme nejjednodušší způsob, jak spojnice realizovat. Poslouží nám k tomu objekt ItemizedOverlay, jenž představuje vrstvu, na kterou je možné přidat libovolné množství objektů. Výhodou je, že při vykreslování se volá metoda draw, kterou si můžeme přetížit a naprogramovat v ní vykreslení spojnice. Princip spočívá v tom, že každou aktuální polohu a předchozí polohu předáme konstruktorem do třídy MyWayItemizedOverlay, která je potomkem třídy ItemizedOverlay a v metodě draw sestavíme spojnici mezi těmito body. Pokud se přeiteruje celá množina uložených poloh v chronologickém pořadí, vznikne nám vrstva spojnic tvořící celou trasu. Další požadavek je, aby se kliknutím na uložený bod (polohu) zobrazilo dialogové okno s datem a časem uložení. I toto lze realizovat v objektu třídy MyWayItemizedOverlay, a sice přetížením metody onTap. Cokoliv se vloží do této metody, tak bude voláno a provedeno ve chvíli, kdy se na vykreslený objekt klikne. Metody draw a onTap jsou uvedeny v příloze IV. Nyní si ukažme, jakým způsobem budeme jednotlivé body do vrstev vkládat.

Vše se bude odehrávat v metodě onCreate třídy implementující mapovou aktivitu. Pomocí databázové pomocné třídy si vytáhneme do seznamu všechny polohy patřící do trasy, kterou chceme vykreslit. Následně je potřeba na mapě zapnout ovládací tlačítka pro

přibližování a oddalování a nastavit výchozí přiblížení. K tomu nám poslouží následující část kódu:

```
mapView = (MapView) findViewById(R.id.mapview);
mapView.setBuiltInZoomControls(true);
mapView.getController().setZoom(17);
```

Dále si inicializujeme obrázek, který se použije jako značka ukazující uloženou polohu. Obvykle se pro tyto účely používá obrázek špendlíku.

```
Drawable drawablePin = this.getResources().getDrawable(R.drawable.mappin);
```

Následně budeme iterovat přes všechny polohy, které jsme z databáze získali. Pro každý bod vytvoříme objekt MyWayItemizedOverlay, do kterého tento bod vložíme společně s bodem předchozím, aby bylo možné vytvořit spojnici. Po načtení mapy se zobrazí všechny body a spojnice mezi nimi.

K práci s mapami si ještě uveďme dvě funkce, které se často využívají, a my je v naší aplikaci budeme vyžadovat. První funkcí je přepínání běžné a satelitní mapy. V mapové aktivitě si musíme držet příznak, který druh mapy je zrovna zobrazován a jakmile uživatel pomocí menu přepne zobrazení mapy, tak zavoláme metodu setSatellite(boolean b), kde booleovskou hodnotou rozhodneme, zda se zobrazí satelitní pohled nebo běžná mapa. Tato metoda se nachází v instanci třídy MapView. Druhou funkcí, která stojí za zmínění je umístění středu mapového pohledu na libovolnou pozici. To využijeme pro posun pohledu na začátek, respektive na konec zobrazované trasy. Pohledem se dá posouvat pomocí kontroleru, jenž získáme z objektu třídy MapView. Na kontroleru je pro to určená metoda animateTo(GeoPoint point). Když uživatel vybere v menu mapy položku pro zobrazení začátku trasy, zavoláme následující příkaz:

```
mapView.getController().animateTo(startPoint);
```

Nyní máme vše připraveno, abychom mohli zobrazit naše trasy na mapě. Do manifestu zaregistrujeme připravenou aktivitu záznamem

```
<activity android:name="MyWayMapActivity" />
```

a ucelíme tím naší aplikaci.

4.4 Grafické standardy aplikací Android

4.4.1 Ikony

Ačkoli je vývojář Android aplikací, co se týká grafického návrhu, svobodný a operační systém mu neklade prakticky žádná omezení, tak je doporučováno dodržovat určité standardy vycházející ze snahy udržet vzhled obrazovek aplikací, menu, ikon, notifikací a dalších grafických prvků přibližně ve stejném stylu. Standardy nejsou nijak závazné a ve skutečnosti se jedná spíše o doporučení, avšak pokud jsou v rámci možností dodržovány, můžeme tím docílit profesionálnějšího vzhledu aplikací a zároveň lepší uživatelské použitelnosti. S vývojem operačního systému Android se vyvíjejí i doporučení pro grafický návrh aplikací, proto nemá smysl uvádět všechny doporučované parametry, jako jsou velikosti okrajů, odsazení, rozměry ikon atd., nýbrž obecně aplikovatelná doporučení týkající se stylu, barevnosti a použití. V této kapitole se zaměříme na ikony, které mohou být několikerého druhu, proto si je rozdělme na ikony spouštěcí (launcher icon), ikony menu a ikony na stavovém panelu.

Spouštěcí ikony

Spouštěcí ikona reprezentuje aplikaci ve spouštěči aplikací (launcher) anebo na ploše domácí obrazovky, pokud si na ni přidáme zástupce této aplikace. Zároveň se také objevuje ve správci aplikací a v některých případech reprezentuje aplikaci ve službě Android Market. Ve verzích Android 1.6 a starších byl zavedený standard spouštěcích ikon zobrazovaných v trojrozměrné perspektivě, což mělo vytvářet dojem, že se uživatel dívá na prostorovou ikonu pod určitým úhlem.

Obrázek 8 - 3D ikona v Android 1.6 (19)

Od verze Android 2.0 jsou doporučovány ikony s pohledem zepředu, bez trojrozměrné perspektivy. Stejný styl ikon se používá například na přístrojích iPhone.

Obrázek 9 - Spouštěcí ikony v systému Android 2.0 a vyšším (19)

Spouštěcí ikony by měly svým výrazem jasně symbolizovat svou aplikaci. Rozhodně není vhodné, aby byly překomplikované a obsahovaly velké množství textu. Je třeba myslet na to, že se budou zobrazovat na přístrojích s různě velkými displeji a rozlišeními, takže to, co je čitelné na jednom displeji, nemusí být čitelné na jiném. Zažitým standardem je vržený stín ikony směrem dolů. Pokud ikona obsahuje určitý objekt jako motiv, je vhodné, aby tento motiv byl zarámovaný ve čtverci se zaoblenými rohy. Barva výplně čtverce je obvykle gradientní a tvoří mírný přechod odstínu od světlé nahoře do tmavší dole.

Ikony menu

Menu, které uživatel v aplikaci vyvolá stiskem tlačítka Menu, obsahuje položky doprovázené ikonami v odstínech šedi. Vývojář nemá možnost do menu přidat ikony barevné, protože operační systém Android je stejně vždy převede do odstínů šedi. Výsledná ikona musí být tvořena PNG souborem bez pozadí. Je obvyklé, že motiv ikony má vržený vnitřní stín a působí dojmem, že je vyříznutý nebo vytlačený.

Obrázek 10 - Příklad menu ikon (20)

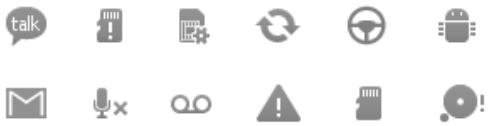
Distribuce Android obsahuje velké množství použitelných ikon pro menu a ve většině případů si vývojáři vyberou vhodnou ikonu odtud a nemusí navrhovat vlastní. Nedoporučuje se ale odkazovat se na tyto ikony napřímo do Android SDK, ale použít vlastní kopii. Důvodem jsou změny ve verzích systému Android a na různých přístrojích by se pak mohly zobrazovat různé ikony pro stejnou položku menu.

Ikony na stavovém panelu

Tento druh ikon se zobrazuje na horním panelu a slouží k notifikacím uživatele nebo k zobrazení stavu aplikace či služby. Obdobně jako u ikon menu jsou standardně v odstínech šedi a vzhledem k jejich malé velikosti by měly být co nejjednodušší a neměly by obsahovat složitý text. Doporučení týkající se barev je ale poměrně často porušováno a spousta aplikací zobrazuje notifikační ikony barevné,

což v tomto případě systém Android dovoluje. U těchto ikon je obzvlášť důležité otestovat čitelnost na displeji s malým rozlišením a vyvarovat se tak nečitelnosti ikony. Finální obrázek ikony musí být uložen jako PNG soubor s průhledným pozadím.

Do verze Android 2.2 bylo platné doporučení umisťovat motiv ikony na tmavý gradientní čtverec. Od verze 2.3 již toto doporučení neplatí a naopak ikony by žádné pozadí mít neměly a měl by je tvořit pouze motiv samotný. Na následujícím obrázku je ukázka standardních ikon z verze systému Android 2.3.



Obrázek 11 - Ikony pro použití na stavovém panelu (21)

4.4.2 Widget design

Ve starších verzích systému Android se počítalo s určitou standardizací vzhledu widgetů, aby domácí obrazovka uživatele vypadala jednotně. Proto byly definovány vzhledy rámečků, jejich zaoblení, velikosti okrajů apod. Jak ale docházelo k růstu obliby widgetů a tím logicky k nárůstu nabízených widgetů, tak se od standardů a doporučení upouštělo a dnes existují widgety rozličných tvarů, barev a velikostí. Jak již bylo řečeno v kapitole 4.3.4, widget vždy na domovské obrazovce přístroje zabírá několik buněk, na které je domovská obrazovka rozdělena. Může jej tvořit jedna jediná buňka, nebo spojité spojení několika buněk, které ale dohromady musí tvořit buďto čtverec, nebo obdélník. Widget, jenž by zabíral spojení buněk do tvaru písmene L nebo podobně není možný.

Poměrně často jsou widgety na ploše poloprůhledné, což je činí atraktivnějšími, ale je potřeba při návrhu dbát na to, aby při použité rozmanité tapetě plochy nebyl obsah widgetu rušen a netrpěla jeho čitelnost. Pro jednotnost celého vizuálního dojmu systému Android je vhodné, aby widget vrhal mírný stín a působil tak plasticky.

Distribuce vývojového prostředí systému Android obsahuje několik pomocných nástrojů pro práci s aplikační grafikou. Jejich zásluhou je možné například vyrábět tlačítka s variabilní velikostí pomocí 9-patch metody (22).

4.5 Distribuce aplikace

Pokud chce vývojář kompletní a otestovanou aplikaci vydat do produkce a nabídnout jí co největšímu počtu uživatelů, nejvhodnější metodou je publikace ve službě Android Market. Drtivá většina uživatelů vyhledává aplikace pro Android přímo z mobilních přístrojů prostřednictvím této služby a nezáleží na tom, zda se jedná o aplikace placené, zdarma, nebo vydělávající autorovi zobrazovanou reklamou. V této kapitole si popíšeme postup, kterým se aplikace publikuje veřejnosti prostřednictví právě Android Marketu.

Jak již bylo zmíněno dříve, každá aplikace musí být podepsána platným certifikátem, jinak ji není možné na operační systém Android nainstalovat. Není však nutné, aby certifikát byl vydaný certifikační autoritou. Stačí, když si vývojář vygeneruje vlastní certifikát a pouze on bude vlastnit jeho privátní klíč. Systém Android tak identifikuje autora aplikace a umožní vytvářet důvěryhodné relace mezi aplikacemi. Postup podepisování aplikace sestává z několika kroků. Předpokládejme, že hotovou aplikaci máme již vyexportovánu do souboru APK. Jako první je nutné vygenerovat vlastní certifikát pomocí nástroje keytool. Doporučuje se minimální doba validity certifikátu 25 let. Nástrojem jarsigner a vygenerovaným certifikátem následně podepíšeme APK soubor a nakonec ještě provedeme optimalizaci pro systém Android nástrojem zipalign, který je součástí Android SDK. Celý tento postup je také možné nahradit funkcí exportu aplikace ve vývojovém prostředí ADT, která vývojáře vyzve k vybrání platného certifikátu, nebo nabídne možnost vytvořit nový. Výsledkem je v obou případech podepsaný a optimalizovaná APK soubor, který je možné instalovat na zařízení Android a publikovat ve službě Android Market.

Pro samotnou publikaci aplikace ve službě Android Market musí být vývojář registrovaný na této službě svým Google účtem a mít zaplacený jednorázový poplatek, který Google vyžaduje pouze jednou. Po zaplacení poplatku je zpřístupněna konzole pro

administraci aplikací. Zde je možné aplikace přidávat, povyšovat jejich verze, nebo je ze služby odebírat. Pokud vývojář zvolí možnost přidat novou aplikaci, tak je mu zobrazen formulář, kde je potřeba vybrat a nahrát na server samotný APK instalační soubor aplikace, dále nahrát minimálně dva snímky obrazovky z aplikace, dále je nutné uvést titulek aplikace, popis, typ a kategorii, druh licence, lokaci, pro kterou je aplikace určena (EU, USA, atd.) a nakonec jsou vyžadovány kontaktní informace na vývojáře nebo osobu zodpovědnou za aplikaci. Volitelně je možné nahrát další grafiku k aplikaci, do které patří ikona ve vysokém rozlišení, propagační obrázek a propagační video. Tuto grafiku Google používá k propagaci aplikací ve svých internetových reklamách a dalších reklamních kanálech. Pokud chceme aplikaci prodávat, musíme uvést identifikační číslo plátce daně. Tato funkcionalita není zatím dostupná ve všech zemích.

Po odsouhlasení licenčních podmínek proběhne samotná publikace, během které si služba Android Market z APK souboru zjistí další potřebné informace, jako je například název aplikace, balík aplikace, ikona, verzi aplikace a minimální potřebná verze operačního systému Android. Pokud vše proběhne v pořádku, je během chvíle aplikace dostupná široké veřejnosti ke stažení.

V konzoli služby Android Market se vývojářům zobrazují statistiky stažení aplikace, včetně údaje o aktivních instalacích, což je reálný počet uživatelů, kteří si aplikaci ponechali na svém přístroji nainstalovánu. Kromě těchto údajů je možné prohlížet další pokročilé statistiky v podobě tabulek a grafů zobrazující údaje o verzích systému Android nebo typech přístrojů, na kterých je aplikace nainstalována. Je k dispozici také geografická statistika uvádějící, ve kterých zemích je aplikace nejvíce stahována.

Důležitou informací pro vývojáře je zpětná vazba od uživatelů. Ti mají možnost aplikaci hodnotit kladnými body v rozmezí 1 až 5, přidávat komentáře a nahlašovat zjištěné chyby. Komentáře a hodnocení se zobrazuje i ostatním uživatelům, kteří se na jejich základě mohou rozhodovat, jestli si aplikaci stáhnou a nainstalují. Chybová hlášení se zobrazují v konzoli pouze vývojáři. Mezi zpětnou vazbu se také řadí anketa, která se volitelně zobrazuje uživatelům během odstraňování aplikace z přístroje. V této anketě je uživatel dotazován na důvod odstranění. Vývojář tak má šanci reagovat na případnou nespokojenost uživatelů a zabránit vylepšením aplikace jejich úbytku.

5 Výsledky a diskuze

Výsledkem této práce je v první části teoretický souhrn týkající se operačního systému Android obecně, od historie systému až po aktuální podobu, dále popis jeho architektury, aplikačního frameworku, virtuálního běhového prostředí a srovnání Androidu s konkurenčními operačními systémy. V části druhé byl realizován a popsán postup implementace reálné aplikace využívající pokročilejší funkce Android API. Na této implementaci bylo ukázáno, jak se nastavuje vývojové prostředí a emulátor zařízení se systémem Android pro testování a ladění aplikací. Následně se čtenář seznámil s vývojem aplikace na různých vrstvách, počínaje databázovými službami, přes obsluhu GPS, tvorbu menu a uživatelského prostředí aktivit aplikace, návrh widgetu, až po načítání dat a mapových podkladů z Internetu a jejich zobrazování uživateli. Následně byly popsány doporučení pro grafický návrh některých grafických prvků používaných v aplikacích a nakonec byl rozebrán proces publikace kompletní aplikace do služby Android Market.

Z teoretického rozboru, který jsme provedli v kapitole 3, vyplývá skutečnost, že operační systém Google Android se v poměrně krátké době své existence stal velice vyspělou mobilní platformou, jež se právě stává lídrem na svém trhu a poráží svou konkurenci, co se týká počtu prodaných chytrých telefonů s tímto systémem. Otázkou, na kterou zatím nelze odpovědět, zůstává, zda operační systém Android zvítězí nebo se alespoň udrží v předních příčkách na trhu s tablety, které se stávají stále populárnějšími, přičemž lze očekávat, že se jedná o dlouhodobější trend a tablety budou nahrazovat mininotebooky. Skutečnost, že uživatelé si budou vybírat při koupi spíše dotykové tablety místo notebooků, znamená, že se Android stane konkurencí i pro operační systémy stolních počítačů, konkrétně pro Microsoft Windows a Apple Mac OS.

Práce je zaměřena především na vývoj aplikací pro platformu Android, proto je implementační část stěžejní a pro názornost se v textu objevují části zdrojového kódu. Z fáze vývoje aplikace vyplývají dva důležité poznatky. První se týká požadavků na dovednosti vývojáře Android aplikací, který by měl být nejen pokročilejší Java programátor, ale měl by se dobře orientovat i v XML kódu a ideálně také ve vývojovém prostředí Eclipse. Pro vývoj na nižší úrovni systému pomocí nativního rozhraní je navíc nutná znalost jazyka C. Druhý poznatek se týká samotného stylu programování a struktury

aplikace, která je odlišná od struktury běžných návrhových vzorů. Logika aplikací je z velké části řízena systémem pomocí šířených událostí, které aplikace přijímá, pokud je vyžaduje, a následně na ně reaguje. Aplikace platformy Android dělá ještě více odlišné od běžných aplikací přístup k obsluze vzdálených grafických rozhraní, kterými jsou widgety. Práce s nimi má do určité míry omezení daná tím, že jsou tvořeny prvky nacházející se mimo samotnou aplikaci a není triviální udržovat stav aplikace a widgetu v konzistentním stavu. Na druhou stranu, vývojáři začínající s vývojem pro systém Android budou zcela jistě příjemně překvapeni některými funkcemi, které usnadní a zpřehlední jejich práci. Mezi ně mimo jiné patří podpora pro nastavení uživatelských preferencí aplikace, správa datového obsahu, lokalizace aplikace, jednoduchá škálovatelnost detailu grafických prvků v závislosti na možnostech cílového zařízení, a nakonec publikace aplikace pro širokou veřejnost.

Pokud diskuzi zaměříme na vývoj aplikací pro Android v České republice, tak zjistíme, že existuje již celá řada společností, které se buď zcela, nebo částečně na Android zaměřují a realizují produkci aplikací na zakázku. Pro menší vývojáře je v době vzniku této práce situace o něco složitější, protože služba Android Market neumožňuje prodej aplikací z našeho území. Vývojáři jsou tak nuceni si vybrat jiné služby, přes které budou své aplikace prodávat, nebo vydělávat na aplikaci jiným způsobem než je prodej, což obvykle znamená vkládání reklamních bannerů do svých aplikací. Česká komunita vývojářů pro platformu Android je široká, což dokazuje v první řadě fakt, že na Android Marketu nalezneme velké množství aplikací od českých vývojářů, a v druhé řadě existence hojně navštěvovaných českých diskusních fór a komunitních webů se zaměřením na Google Android.

Závěr

V práci byl komplexně popsán operační systém Android tak, aby s ním byl čtenář seznámen jak z pohledu pokročilého uživatele s povědomím o jeho architektuře, historii a použití, tak i z pohledu vývojáře, který implementací ukázkové aplikace pojme značnou část Android API. Stěžejní částí práce jsou kompletní a funkční zdrojové kódy této ukázkové aplikace, jejichž důležité části jsou v textu uvedeny a popisovány. Čtenář se také dozvěděl o vývojovém prostředí, které je k distribuci Android API dodáváno. V závěru implementační části byly zmíněny doporučení týkající se grafického vzhledu uživatelského prostředí aplikací a popsali jsme způsob distribuce aplikace pomocí služby Android Market.

Díky širokým možnostem Android API ukázková aplikace aspiruje na rozšíření o dodatečné funkcionality, které by zahrnuly další části aplikačního frameworku. Do budoucna by se zcela jistě nabízela kapitola věnující se vývoji aplikací pro dotykové tablety s novou verzí Android 3.0 a vyšší, která nabízí nové možnosti. Na základě diskutovaných výsledků lze konstatovat, že prací byly pojaty všechny vytyčené cíle stanovené tak, aby konečná práce měla jistou teoretickou komplexnost a zároveň i praktickou hloubku. V průběhu psaní práce vznikla funkční a použitelná aplikace, která je již nyní dostupná přes službu Android Market a jejíž zdrojové kódy jsou součástí této práce.

Seznam použitých zdrojů

1. **Schmidt, Eric.** Google CEO: Moore's Law to Transform Smartphones Into World's Dominant Communications Platform. *Editor & Publisher.* [Online] 2010. http://www.editorandpublisher.com/Departments/Online/google-ceo-moores-law-to-transform-smartphones-into-worlds-dominant-communications-platform-63185-.aspx.

2. **Gartner.** Gartner Says Android to Become No. 2 Worldwide Mobile Operating System in 2010 and Challenge Symbian for No. 1 Position by 2014. *Gartner newsroom.* [Online] Gartner, 2010. http://www.gartner.com/it/page.jsp?id=1434613.

3. **Elgin, Ben.** Google Buys Android for Its Mobile Arsenal. *Bloomberg Businessweek.* [Online] 2005. http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm.

4. **Hill, Simon.** Bright Hub. *History of Android: First Applications, Prototypes & Other Events.* [Online] 2010. http://www.brighthub.com/mobile/google-android/articles/18260.aspx.

5. **Wikipedia.** Android (Operační systém). *Wikipedia.* [Online] 2011. http://cs.wikipedia.org/wiki/Android_(opera%C4%8Dn%C3%AD_syst%C3%A9m).

6. **Android Developers.** What is Android? *android.com.* [Online] 2010. http://developer.android.com/guide/basics/what-is-android.html.

7. **Ehringer, David.** davidehringer.com. [Online] 2010. http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf.

8. **Meier, Reto.** *Professional Android Application Development.* Indianapolis : Wrox, 2008. ISBN 978-0470344712.

9. **Android Developers.** Application Fundamentals. *android.com.* [Online] 2011. http://developer.android.com/guide/topics/fundamentals.html.

10. —. Content Providers. *android.com.* [Online] 2011. http://developer.android.com/guide/topics/providers/content-providers.html.

11. **Juniper Networks.** Juniper Networks. *Juniper Global Threat Center.* [Online] 2010. http://globalthreatcenter.com/.

12. **Apple.** iOS Overview. *iOS Reference Library.* [Online] 2011. http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/URL_iPhone_OS_Overview/index.html#//apple_ref/doc/uid/TP40007592.

13. **Wikipedia.** Windows Embedded CE 6.0. *Wikipedia.* [Online] 2011. http://en.wikipedia.org/wiki/Windows_Embedded_CE_6.0.

14. —. Windows Phone 7. *Wikipedia.* [Online] 2011. http://en.wikipedia.org/wiki/Windows_Phone_7.

15. —. Qt Framework. *Wikipedia.* [Online] 2010. http://en.wikipedia.org/wiki/Qt_(framework).

16. —. Symbian. *Wikipedia.* [Online] 2011. http://en.wikipedia.org/wiki/Symbian.

17. **BlackBerry.** Blackberry Architecture. *blackberry.com.* [Online] 2011. http://us.blackberry.com/ataglance/solutions/architecture.jsp.

18. **Wikipedia.** BlackBerry. *Wikipedia.* [Online] 2011. http://en.wikipedia.org/wiki/BlackBerry.

19. **Android Developers.** The AndroidManifest.xml file. *Android Developers.* [Online] 2011. http://developer.android.com/guide/topics/manifest/manifest-intro.html.

20. —. *Reference.* [Online] 2011. http://developer.android.com/reference/packages.html.

21. **Google.** Android Maps API Key Signup. *Google Code.* [Online] 2011. http://code.google.com/intl/cs-CZ/android/maps-api-signup.html.

22. **Android Developers.** Icon Design Guidelines - Launcher. *android.com.* [Online] 2011. http://developer.android.com/guide/practices/ui_guidelines/icon_design_launcher.html.

23. —. Icon design Guidelines - Menu. *android.com.* [Online] 2011. http://developer.android.com/guide/practices/ui_guidelines/icon_design_menu.html.

24. —. Icon Design Guidelines - Status Bar Icons. *android.com.* [Online] 2011. http://developer.android.com/guide/practices/ui_guidelines/icon_design_status_bar.html.

25. —. Tools. *android.com.* [Online] 2011. http://developer.android.com/guide/developing/tools/draw9patch.html.

26. **Wikipedia.** Android (operating system). *Wikipedia.* [Online] 2011. http://en.wikipedia.org/wiki/Android_%28operating_system%29.

Přílohy

A. Zdrojové kódy

I. Metoda getView adaptéru DataArrayAdapter

```
public View getView(int position, View convertView, ViewGroup parent) {
	LinearLayout ll = (LinearLayout) infl.inflate(R.layout.data_item, parent, false);
	// podle pozice-indexu si vyzvedneme objekt trasy
	Way way = (Way) list.get(position);


	if (way.getId() == actualWayId) {
	   ll.setBackgroundColor(Color.argb(100, 0, 68, 0));
	}
	// plnime polozky pohledu a tabulky
	ImageView image = (ImageView) ll.findViewById(R.id.dataImage);
	image.setImageDrawable(ctx.getResources().getDrawable(R.drawable.compass));
	TableLayout tbl = (TableLayout) ll.findViewById(R.id.dataTable);
	TableRow tr1 = (TableRow) tbl.findViewById(R.id.dataTableRow1);
	TableRow tr2 = (TableRow) tbl.findViewById(R.id.dataTableRow2);
	TextView tv1 = (TextView) tr1.findViewById(R.id.dataText1);
	TextView tv2 = (TextView) tr2.findViewById(R.id.dataText2);
	tv1.setText(way.getName());
	// datum musime naformatovat
	SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd HH:mm");
	String info = "Created: "+sdf.format(way.getCreateTime());
	if (way.getId() == actualWayId) {
	   info = "Actual Way | "+info;
	}
	tv2.setText(info);
	return ll;
}
```

II. Metoda onContextItemSelected

```
public boolean onContextItemSelected(MenuItem item) {
	// z objektu MenuItem si vytahneme objekt AdapterContextMenuInfo
	AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
	// z info objektu ziskame index polozky seznamu, takze vime, na kterou polozku se kliklo
	Way way = wayList.get(info.position);
	    if
	    (item.getTitle().toString().equals(getResources().getString(R.string.context_menu_open_map)))
	    {
	        // spoustime mapovou aktivitu
	        Intent intent = new Intent(this, MyWayMapActivity.class);
	        intent.putExtra("wayObj", way);
	        startActivity(intent);
	    }
	    if
	    (item.getTitle().toString().equals(getResources().getString(R.string.context_menu_open_way)))
	    {
	        // spoustime aktivitu s detailem trasy
	        Intent intent = new Intent(this, MyWayDetailActivity.class);
	        intent.putExtra("wayObj", way);
	        startActivity(intent);
	    }
	    if
	    (item.getTitle().toString().equals(getResources().getString(R.string.context_menu_rename_way))
	    ) {
	        // zobrazime dialog pro zadani noveho jmena trasy
	        menuPopupRenamePath(way.getName(), way.getId());
	    }
```

```
            if
            (item.getTitle().toString().equals(getResources().getString(R.string.context_menu_delete_way))
            ) {
                // zobrazime dotaz na potvrzeni smazani trasy
                menuPopupDeletePath(way.getName(), way.getId());
        }

        return true;
}
```

III. Struktura souboru preference.xml

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
   <PreferenceCategory android:title="Preferences">
         <ListPreference android:entryValues="@array/prefPeriodValues"
         android:summary="Time period for tracking"
         android:entries="@array/prefPeriodEntries" android:title="Auto Location Period"
         android:key="prefPeriodKey" android:defaultValue="60"></ListPreference>
         <CheckBoxPreference android:title="Vibrate" android:summary="Vibrate when position
         is saved" android:key="vibrate" android:defaultValue="false"/>
         <CheckBoxPreference android:title="Save locality" android:summary="Save address of
         the position (uses internet connection)" android:key="locality"
         android:defaultValue="true"/>
         <CheckBoxPreference android:title="Use network for localization"
         android:summary="Cellular network will be used to locate in addition to GPS (may
         be inaccurate)" android:key="useNetwork" android:defaultValue="false"/>
         <CheckBoxPreference android:title="Save only distinct positions"
         android:summary="The position will be stored only when differ from previous one"
         android:key="onlyDistinct" android:defaultValue="true"/>
   </PreferenceCategory>
</PreferenceScreen>
```

IV. Vytvoření notifikace

```
String ns = Context.NOTIFICATION_SERVICE;
// vyzvedneme si notifikacniho manazera
NotificationManager mNotificationManager = (NotificationManager) ctx.getSystemService(ns);
// urcime ikonu notifikace
int icon = R.drawable.ic_stat_notification;
CharSequence tickerText = "Tracking is enabled";
long when = System.currentTimeMillis();
// zalozime novou notifikaci s textem a ikonou, platna odted
Notification notification = new Notification(icon, tickerText, when);
CharSequence contentTitle = "MyWay Tracker";
CharSequence contentText = "Tracking is enabled";
// po kliknuti na notifikaci se spusti hl. aktivita
Intent notificationIntent = new Intent(ctx, DataActivity.class);
PendingIntent contentIntent = PendingIntent.getActivity(ctx, 0, notificationIntent, 0);
notification.setLatestEventInfo(ctx, contentTitle, contentText, contentIntent);
// notifikaci nelze smazat dokud je aktivni automat. lokace
notification.flags = Notification.FLAG_NO_CLEAR;
// notifikujeme
mNotificationManager.notify(MYWAY_TRACKER_NOTIF_ID, notification);
```

V. Metody onTap a draw z třídy MyWayItemizedOverlay

```
@Override
   protected boolean onTap(int index) {
      OverlayItem item = mOverlays.get(index);
      // vytvoreni instance dialogu
```

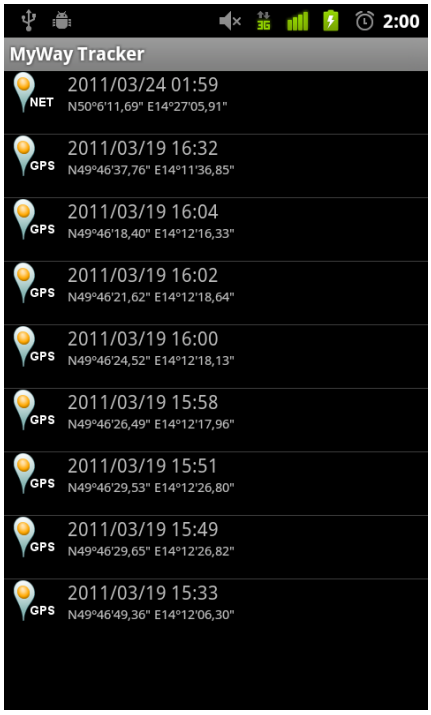
```
        AlertDialog.Builder dialog = new AlertDialog.Builder(mContext);
        dialog.setTitle(item.getTitle());
        dialog.setMessage(item.getSnippet());
        // na dialog pridame potvryovaci tlacitko
        dialog.setNegativeButton("Ok", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
            }
        });
        // yobrayime dialog
        dialog.show();
        return true;
    }

    @Override
    public void draw(Canvas canvas, MapView mapView, boolean shadow) {
        super.draw(canvas, mapView, shadow);
        // vytvorime objekt kresby a body pro zaneseni
        Paint paint = new Paint();
        Point screenCoords = new Point();
        Point screenCoords1 = new Point();

        mapView.getProjection().toPixels(curPoint, screenCoords1);
        int x2 = screenCoords1.x;
        int y2 = screenCoords1.y;
        if (prePoint != null) {
            mapView.getProjection().toPixels(prePoint, screenCoords);
            int x1 = screenCoords.x;
            int y1 = screenCoords.y;
            // nastavime vlastnosti cary
            paint.setDither(true);
            paint.setAlpha(150);
            paint.setAntiAlias(true);
            paint.setStyle(Paint.Style.FILL_AND_STROKE);
            paint.setStrokeJoin(Paint.Join.ROUND);
            paint.setStrokeCap(Paint.Cap.ROUND);
            paint.setStrokeWidth(3);
            paint.setColor(Color.RED);
            // kreslime
            canvas.drawLine(x1, y1, x2, y2, paint);
        }
    }
```

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // ziskame objekt trasy a vsechny jeji pozice
        Way way = (Way) getIntent().getExtras().getSerializable("wayObj");
        DataHelper dh = new DataHelper(this);
        List<Position> positionList = dh.getAllPositions(way.getId(), false);
        setContentView(R.layout.map);
        // ziskame objekt pohledu mapy
        mapView = (MapView) findViewById(R.id.mapview);
        mapView.setBuiltInZoomControls(true);
        mapView.getController().setZoom(17);
        // obrazek bodu
        Drawable drawablePin = this.getResources().getDrawable(R.drawable.mappin);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd HH:mm");
        GeoPoint prePoint = null;
        GeoPoint curPoint = null;
        int index = 0;
        // iterujeme pres vsechny body trasy
        for (Position pos : positionList) {
            // ulozime si aktualni bod
            prePoint = curPoint;
            // inicializujeme novy aktualni bod
            curPoint = new GeoPoint(new Double(pos.getLatitude()*1000000).intValue(), new
            Double(pos.getLongitude()*1000000).intValue());
            // instancujeme vrstvu
            MyWayItemizedOverlay overlay = new MyWayItemizedOverlay(prePoint, curPoint, this,
            mapView);
            // bod zaobalime do objektu OverlayItem a jako popis predame cas lokace
            OverlayItem overlayitem = new OverlayItem(curPoint, pos.getLocateTime());
            overlay.addOverlay(overlayitem);
```

```
        // pridame vrstvu do mapy
        mapView.getOverlays().add(overlay);
        index++;
    }
}
```

B. Obrázky

VI. Obrazovka – seznam uložených poloh



VII. Obrazovka – Trasa na mapě