VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POKROČILÁ RSS ČTEČKA PRO ANDROID OS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MAREK ŘEHULKA
AUTHOR

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POKROČILÁ RSS ČTEČKA PRO ANDROID OS

ADVANCED RSS READER FOR ANDROID OS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MAREK ŘEHULKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. IGOR SZŐKE, Ph.D.
SUPERVISOR

BRNO 2012

Abstrakt

Tato práce se zabývá vývojem pokročilé RSS čtečky pro mobilní platformu Android. Součástí práce je návrh, implementace a testování čtečky. Práce také popisuje problematiku vývoje aplikací pro mobilní zařízení se systémem Android a technologii syndikace obsahu. Výsledná aplikace je naprogramovaná v jazyce Java s použitím vývojového kitu Android SDK a pluginu ADT pro vývojové prostředí Eclipse. Aplikace implementuje vícevláknové zpracování, parsery JSON/XML formátů, vlastní adaptéry, SQLite databázi a uživatelské rozhraní založené na XML.

Abstract

This thesis deals with development of advanced RSS reader for Android mobile platform. The parts of the thesis are design, implementation and reader testing. The thesis also describe problematic of application development for mobile devices with Android system and technology of content syndication. Final application is programmed in JAVA language with usage of SDK Android development kit and ADT plug-in for Eclipse development environment. Application implements multi-thread processing, JSON/XML format parsers, custom adapters, SQLite database and user-friendly interface based on XML.

Klíčová slova

Android, aplikace, mobilní, platforma, Google, čtečka, RSS, ATOM, XML, JSON, syndikace, Java, AsyncTask, SQLite, GUI, ActionBar, grafické uživatelské rozhraní, vývoj

Keywords

Android, application, mobile, platform, Google, reader, RSS, ATOM, XML, JSON, syndication, Java, AsyncTask, SQLite, GUI, ActionBar, graphical user interface, development

Citace

Marek Řehulka: Pokročilá RSS čtečka pro Android OS, bakalářská práce, Brno, FIT VUT v Brně, 2012

Pokročilá RSS čtečka pro Android OS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Igora Szőke Ph.D.

. . . . . .
Marek Řehulka
18. května 2012

Poděkování

Děkuji Ing. Igoru Szőke Ph.D. za hodnotné rady, vstřícný přístup a odborné vedení během mé práce. Děkuji také svojí rodině, přátelům a především mamince a přítelkyni Jitce za skvělé studijní podmínky a maximální podporu.

© Marek Řehulka, 2012.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **3**

2 Platforma Android **4**
2.1 Architektura 4
2.2 Vývoj aplikací pro Android 6
2.3 Aplikační principy 8

3 Syndikace obsahu **15**
3.1 Možnosti využití 15
3.2 Formáty RSS a Atom pro syndikaci obsahu 16
3.3 Vyhledávání odběrů 17
3.4 Čtečky odebíraných zdrojů 18

4 Návrh RSS čtečky pro Android **19**
4.1 Průzkum existujících řešení 19
4.2 Návrh vlastního řešení a očekávaný přínos 20
4.3 Ukládání dat a nastavení 21
4.4 Uživatelské rozhraní 22

5 Implementace RSS čtečky pro Android **25**
5.1 Struktura projektu 25
5.2 Ukládání dat a nastavení 26
5.3 Vícevláknové zpracování 26
5.4 Vyhledávání odběrů 27
5.5 Ukládání a aktualizace odběrů 27
5.6 Načítání a aktualizace hodnocení popularity 27
5.7 Uživatelské rozhraní 29
5.8 Testování, kompatibilita a omezení 29

6 Závěr **30**

Seznam obrázků

2.1 Android Robot [11]. 4
2.2 Architektura platformy Android [24]. 6
2.3 AVD emulátor mobilních zařízení. 7
2.4 Životní cyklus aktivity [7]. 10
2.5 Životní cyklus služby [13]. 11

3.1 Ikona RSS. 18

4.1 ER diagram 22
4.2 Úvodní obrazovka aplikace 23
4.3 Výpis odebíraných článků 24
4.4 Detail článku 24
4.5 Vyhledávání a přidávání odběrů 24

Kapitola 1

Úvod

Od počátku svého vzniku urazily mobilní telefony neuvěřitelný kus cesty a dnes jsou z nich sofistikovaná zařízení vybavená nejmodernějšími technologiemi, vlastním operačním systémem a nepřeberným množstvím funkcí. Navzdory extrémně rychlému vývoji a rozšíření stále přetrvává jejich původní myšlenka zpříjemnit lidem život a ušetřit čas. V podobě mobilních aplikací se objevují další a další řešení usnadňující každodenní činnosti. Mezi tato řešení patří i takzvané RSS[1] čtečky neboli čtečky odběrů, které se poprvé objevily na stolních počítačích a později našly své místo i na mobilních telefonech.

Hlavním přínosem čteček je možnost na jednom místě soustředit články a informace z mnoha uživatelem preferovaných zpravodajských portálů, blogů, e–shopů a dalších internetových stránek. Toho je dosaženo pomocí technologie syndikace obsahu, kterou dnes běžně používá většina webů, především internetové magazíny, zmíněné blogy a zpravodajství. Bohužel zatím existuje pouze několik skutečně kvalitních RSS čteček pro mobilní zařízení, což byl také jeden z důvodů, proč začít pracovat na vlastní aplikaci.

Dalším důvodem byla motivace seznámit se s vývojem pro relativně novou populární platformu Android od firmy Google určenou pro mobilní zařízení. Android je otevřená platforma nabízející více než bohaté možnosti, podporu vývojářů, snadnou distribuci a zajímavé vyhlídky do budoucna, včetně potenciálně možného uplatnění v dopravních prostředcích [26].

Na základě předchozího zdůvodnění si tato bakalářská práce bere za cíl vytvořit pokročilou RSS čtečku pro Android. Výsledná čtečka by měla kombinovat osvědčené techniky, více usnadnit uživatelům práci s odběry a přinést nové užitečné funkce.

První část textu se zabývá platformou Android, její architekturou, vývojem, dostupnými nástroji a aplikačními principy pro tuto platformu. Následuje kapitola věnující se syndikaci obsahu, formátům pro syndikaci obsahu, vyhledávání odběrů a RSS čtečkám obecně. Na ni navazuje průzkum existujících aplikací a jejich zhodnocení. Dále jsou rozebrány různé možnosti řešení problému a vlastní návrh čtečky. Na základě návrhu pokračuje práce popisem implementace, kde jsou uvedeny použité nástroje, technologie a řešeny zásadní problémy při implementaci navrhované funkcionality, například parsování a ukládání dat, vícevláknové zpracování, vyhledávání odběrů, kompatibilita a omezení. Práci uzavírá zhodnocení dosažených výsledků včetně vlastního přínosu a nastínění možností budoucího vývoje.

[1] RSS – Really Simple Syndication

Kapitola 2

Platforma Android

Android je relativně nový fenomén – zařízení využívající tuto platformu se poprvé objevila na scéně koncem roku 2008 [26]. Je to propracovaná a otevřená mobilní platforma založená na linuxovém jádru [8]. Pojem Android nezahrnuje pouze samotný operační systém, ale kompletní softwarový set pro mobilní zařízení: operační systém, middleware[1]. a klíčové mobilní aplikace. Navíc přináší širokou podporu pro vývojáře v podobě SDK[2] a oficiálního ADT[3] pluginu pro vývoj v prostředí Eclipse. Všechny tyto jednotlivé komponenty a další nástroje obsažené v SDK budou probrány v následujících podkapitolách.

Android není určen pro konkrétní typ zařízení, ale je to platforma, kterou lze použít a přízpůsobit různému hardwaru [5]. Mobilní telefony jsou pro tohoto zeleného robota zatím hlavním hracím polem, ale v současné době se již nasazuje také v elektronických čtečkách, netboocích, tabletech, set-top boxech, dokonce i v pračkách a mikrovlnných troubách. Brzy se pravděpodobně začne běžně používat i jako palubní počítač automobilů, kde se v podobě amatérské instalace objevuje už dnes.

Obrázek 2.1: Android Robot [11].

2.1 Architektura

Předtím, než se pustíme do vývoje a aplikačních principů na Androidu, měli bychom se alespoň okrajově seznámit s jeho architekturou (viz obrázek 2.2). Architektura systému Android sestává z pěti následujících vrstev:

2.1.1 Linuxové jádro

Nejnižší a nejdůležitější vrstva mobilního systému Android používá linuxové jádro verze 2.6 pro základní systémové služby, jako je zabezpečení, správa paměti, řízení procesů, síťové služby, ovladače dotykového displeje, kamery, klávesnice a dalšího hardwaru. Linuxové jádro hraje také roli abstraktní vrstvy mezi hardwarem a ostatními vrstvami architektury [8].

[1] Middleware – Softwarová vrstva ležící mezi operačním systémem a aplikacemi
[2] SDK – Software Development Kit
[3] ADT – Android Development Tools

2.1.2 Běhové prostředí

Běhové prostředí umístěné nad linuxovým jádrem je tím hlavním, co dělá z Androidu více než jen mobilní implementaci Linuxu. Zahrnuje základní systémové knihovny a virtuální stroj Dalvik, jehož funkcí je vykonávání bytekódu, na kterém jsou postaveny vyšší vrstvy architektury [24].

Protože se aplikace pro Android většinou programují v Javě, můžete se domnívat, že budete potřebovat Java Virtual Machine (JVM), který je u standardních programů psaných v jazyce Java odpovědný za provádění bytekódu. JVM typicky poskytuje potřebné optimalizace, díky kterým Java dosahuje úrovně výkonnosti kompilovaných jazyků, jako jsou C a C++. Pro tyto účely Android vytvořil vlastní optimalizaci běhu zkompilovaných programů právě v podobě virtuálního stroje Dalvik, který je šitý na míru mobilním zařízením tak, aby aplikace mohly lépe čelit jejich výkonovým a dalším níže uvedeným omezením [22, 8].

2.1.3 Knihovny

Tato vrstva se také nachází přímo nad linuxovým jádrem a zahrnuje knihovny napsané v jazyce C/C++. Například systémovou knihovnu jazyka C založenou na Berkeley Software Distribution (BSD) vyladěnou (přibližně na polovinu svojí původní velikosti) pro použití na zařízeních postavených na linuxovém jádru. Dále knihovnu médií pro přehrávání a záznam audio a video formátů, správce přístupu k displeji zařízení, grafické knihovny včetně OpenGL pro 2D a 3D grafiku, SQLite pro relační databáze, SSL[4] a WebKit integrující webový prohlížeč a bezpečné připojení k internetu. Další nativní knihovny mohou postupně přibývat s novými verzemi systému Android [22, 8].

2.1.4 Aplikační rámec

Aplikační rámec je vrstva zahrnující klíčové stavební bloky pro vytváření aplikací na vysoké úrovni. Díky tomu, že Android poskytuje otevřenou platformu pro vývoj, dovoluje tak programátorům vytvářet velmi bohaté aplikace. Každá aplikace, včetně nativních, pracuje se stejným API[5], což umožňuje snadné opakované použití jednotlivých komponent. Jakákoliv aplikace tak může být používána jinou aplikací (v závislosti na bezpečnostních omezení frameworku) [22]. Například v našem konkrétním případě je možné uvnitř vlastní aplikace spouštět nativní službu pro sdílení vybraného obsahu. Základem všech aplikací pro Android je soubor služeb a systémů zahrnující:

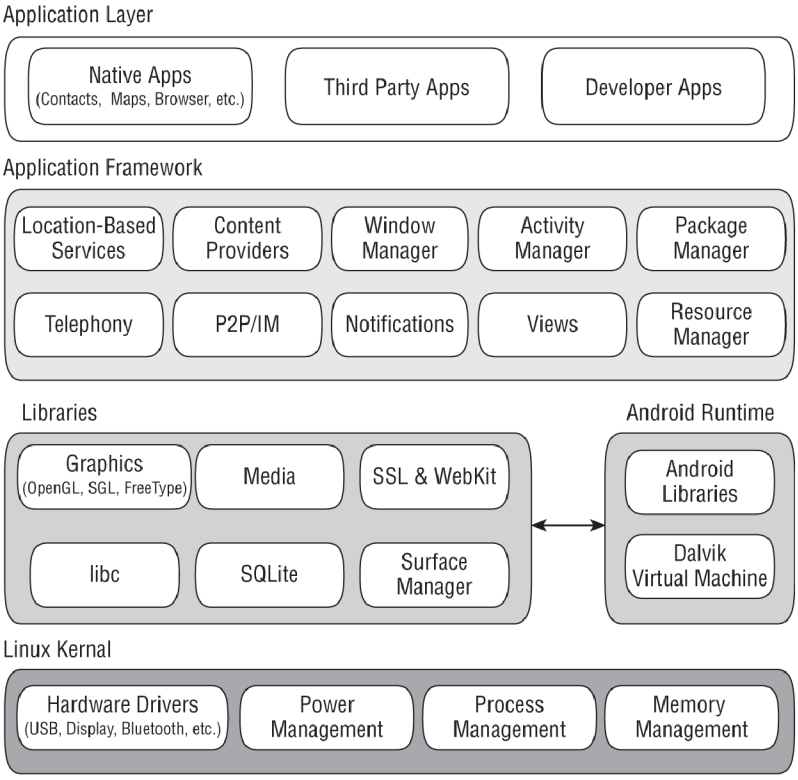
- **Systém Views (pohledů)** – Rozsáhlý balík předpřipravených prvků GUI[6], o kterém si řekneme něco více v podkapitole 2.3.1).
- **Dodavatele obsahu** – Zpřístupňují perzistentní data aplikací i operace nad těmito daty jiným aplikacím a naopak (viz 2.3.4).
- **Správce prostředků** – Poskytuje přístup k prostředkům, které nejsou součástí kódu aplikace, zejména lokalizované řetězce, grafika a deklarace GUI.
- **Správce upozornění** – Umožňuje, aby mohla libovolná aplikace zobrazovat vlastní upozornění ve stavovém panelu.

[4] SSL – Secure Sockets Layer
[5] API – Application Programming Interface
[6] GUI – Graphical User Interface

- **Správce aktivit** – Řídí životní cyklus aplikací a spravuje zásobník používaný pro navigaci mezi aplikacemi a jejich aktivitami.

2.1.5 Aplikace

Všechny aplikace pro Android spadají do nejvyšší vrstvy architektury a používají jednotný aplikační rámec. Součástí této vrstvy je také balík základních aplikací zahrnujících emailového klienta, program pro psaní zpráv, kalendář, mapy, internetový prohlížeč, adresář kontaktů a další. Každá z těchto nativních aplikací je napsána v programovacím jazyce Java [8, 24].



Obrázek 2.2: Architektura platformy Android [24].

2.2 Vývoj aplikací pro Android

Vývoj aplikací pro Android představuje zkušenost s velmi zajímavou technologií na rychle se rozvíjejícím trhu. Problém nastává ve chvíli, kdy si uvědomíme, že mobilní telefony mají malou obrazovku a ve srovnání s notebooky nebo stolními počítači i omezený výkon a jejich polohovací zařízení jsou nepřesná (velké prsty a displeje schopné reagovat na více dotyků najednou). Nejen z těchto důvodů se vývoj programů pro telefon liší od vývoje pro stolní počítače, webových stránek, nebo obslužných serverových procesů. Liší se jak vývojové nástroje, tak i aplikační principy a jednotlivé funkce programů podléhají mnoha různým omezením [26].

Programy pro Android je možné psát v běžně používaných programovacích jazycích, jako jsou Java nebo C/C++. Primárně používaným a podporovaným jazykem pro tuto

platformu je Java, ale pro programy jejichž odezva je kriticky důležitá je vhodné zvážit použití právě jazyka C/C++ [8], ale tato problematika není tématem bakalářské práce a nebudeme se jí dále zabývat. Nyní se zaměříme na dva zásadní prostředky pro vývoj androidích aplikací s použitím jazyka Java.

2.2.1 Vývojový kit Android SDK

Prvním krokem při vývoji na Androidu (v jazyku Java) je stažení a instalace startovacího balíku SDK z oficiálního webu pro vývojáře (viz [9, 10]). Čerstvě nainstalované SDK obsahuje pouze důležité základní programy. Další potřebné prostředky pro vývoj včetně jednotlivých verzí systému je potřeba si dle vlastních preferencí dodatečně nahrát pomocí SDK manažera. Ten se ve Windows nachází v kořenovém adresáři SDK pod názvem `SDK Setup.exe`, v Linux a OS X v adresáři `tools` jako soubor `android`.

Jednou z hlavních obtíží při vývoji pro Android je, že se musíte vyrovnat s několika podporovanými verzemi systému, které se můžou někdy i v zásadních oblastech lišit. V našem případě to byly mimo jiné oblasti: správa databázových kurzorů, použití horní navigační lišty (ActionBaru) i vícevláknové zpracování, ale o tom až později. Právě proto SDK nabízí k dispozici všechny dostupné platformy, které odpovídají produkčním platformám na skutečných zařízeních. Každá platforma zahrnuje dokumentaci, systémové knihovny, systémový obraz, ukázky kódu a jiné specifické zdroje pro danou platformu [5].

Obrázek 2.3: AVD emulátor mobilních zařízení.

S instalací SDK získáte i AVD emulátor mobilních zařízení (viz obrázek 2.3), což je nedocenitelný pomocník, který vývojářům podstatně usnadňuje a zpříjemňuje práci. Emulátor lze nakonfigurovat podle individuálních potřeb. Je možné měnit jak verze systému Android, tak simulovaný použitý hardware, tedy rozměry a rozlišení displeje, kapacitu paměti, velikost SD karty, parametry fotoaparátu atp. Emulátor obsahuje všechny části systému včetně nativních aplikací a zabudovaný webový prohlížeč jako skutečné mobilní zařízení. V emulátoru je také možnost otočit virtuální zařízení do horizontální polohy, čehož využijeme například při testování GUI a změn konfigurace viz dále. Těchto emulovaných zařízení je možné vytvořit několik současně. Jednotlivé instance jsou na sobě zcela nezávislé a umožňují tak snadnější ladění vyvíjených programů pro vybrané konfigurace a platformy.

Důležitou roli v SDK dále hraje i pokročilý debuger DDMS[7], který umí komunikovat jak s běžícími instancemi emulátoru (viz níže), tak s připojenými mobilními zařízeními. DDMS zvládá zobrazovat logy z celého systému a různé ladící informace. Umí simulovat různé akce, jako příjem zpráv a příchozí hovor, což využijeme především pro otestování životního cyklu aplikace viz následující kapitola. Dále SDK obsahuje program `adb`[8], s jehož pomocí lze mimo jiné na zařízení nahrávat soubory nebo aplikace, a program `aapt`[9] k vytváření a modifikaci aplikačních balíků. SDK zahrnuje i balíky Google API s knihovnami Google map, které standardně nejsou základní součástí platformy.

2.2.2 Plugin ADT pro Eclipse

ADT je oficiální plugin pro vývojové prostředí Eclipse, který umožňuje celé SDK pohodlně ovládat. Součástí tohoto pluginu je integrované snadné vytváření Android projektů, jejich překlad a instalace na připojená skutečná zařízení nebo instance emulátorů přímo z Eclipse. Další z vybraných předností ADT je propojení s pokročilým debugerem DDMS, editor XML[10], nástroj pro vytváření a editaci GUI a referenční dokumentace, která je přístupná i přejetím myši nad třídami, metodami nebo proměnnými v editoru.

Pokud ale nepatříte mezi fanoušky Eclipse a preferujete jiná IDE[11], jsou tu i další řešení třetích stran, například pro Intellij IDEA (více viz [18]).

2.3 Aplikační principy

Finální Android aplikace jsou zkompilované zdrojové soubory zabalené spolu s dalšími prostředky do ZIP balíku s koncovkou .apk[12]. V této podobě jsou distribuovány a instalovány na koncová zařízení. Pro každou spuštěnou aplikaci vytváří Android vlastní virtuální stroj Dalvik ve kterém běží nový Linuxový proces s jedním vláknem (tzv. „hlavním“ vláknem, nebo také vláknem UI). Operační systém Android je multiuživatelský linuxový systém, kde každá aplikace představuje jiného uživatele. Po instalaci na zařízení, žije každá aplikace ve vlastním sandboxu – jsou od sebe navzájem izolované a mají přísná omezení interakce s nativními komponentami zařízení a jeho hardwarem. Mohou ale používat záměry, přijímače vysílání, poskytovatele obsahu a připojení k internetu [24, 8, 5].

Jak už bylo zmíněno, jednou ze základních vlastností Androidu je možnost ve svém vlastním programu využít části jiných programů a naopak. Cílem tohoto řešení bylo zamezit opakovanému vytváření stejné funkcionality a ušetřit tak vývojářům čas. Proto byly aplikace navrženy jako balíky několika komponent, které lze samostatně používat.

Jednotlivé zdroje informací, které se věnují této problematice, uvádějí odlišné výčty nejdůležitějších komponent, prostředků a principů. Proto se budeme následně zabývat především stěžejními prvky pro vývoj vlastní aplikace, která je cílem bakalářské práce.

2.3.1 Views

Třída `android.view.View` je základní třídou pro všechny UI prvky v Android aplikacích. Existují tři základní typy views: `SurfaceView`, `ViewGroup` a `Wigdet`. První a nejdůležitější

[7] DDMS – Dalvik Debug Monitor Server
[8] adb – Android Debug Bridge
[9] aapt – Android Asset Packaging Tool
[10] XML – Extensible Markup Language
[11] IDE – Integrated Development Environment
[12] apk – Android Package

typ poskytuje hlavní vykreslovací plochu, o které nám stačí pro tuto práci vědět, že existuje, ale jinak s ní nepřijdeme dále do styku. Naopak s následujícími třídami budeme pracovat poměrně často. `ViewGroup` je abstrakce pro různé varianty kontejnerů (layoutů), které jsou určeny pro umístění dalších jim podřízených views a organizují jejich rozložení. Posledním typem jsou widgety, což jsou typické UI komponenty jako textové bloky, formulářové prvky, tlačítka, navigační menu, seznamy, obrázky atd.

2.3.2 Aktivity

Android aplikace se vždy skládají z jedné nebo více aktivit. Aktivity můžeme přirovnat k oknům či dialogům programů pro stolní počítač. Typická aktivita aplikace představuje jednu obrazovku uživatelského rozhraní složenou z pohledů (views), které jsou řízeny touto aktivitou. Aktivita může mít také podobu plovoucího okna nad jinou aktivitou [5, 6, 26].

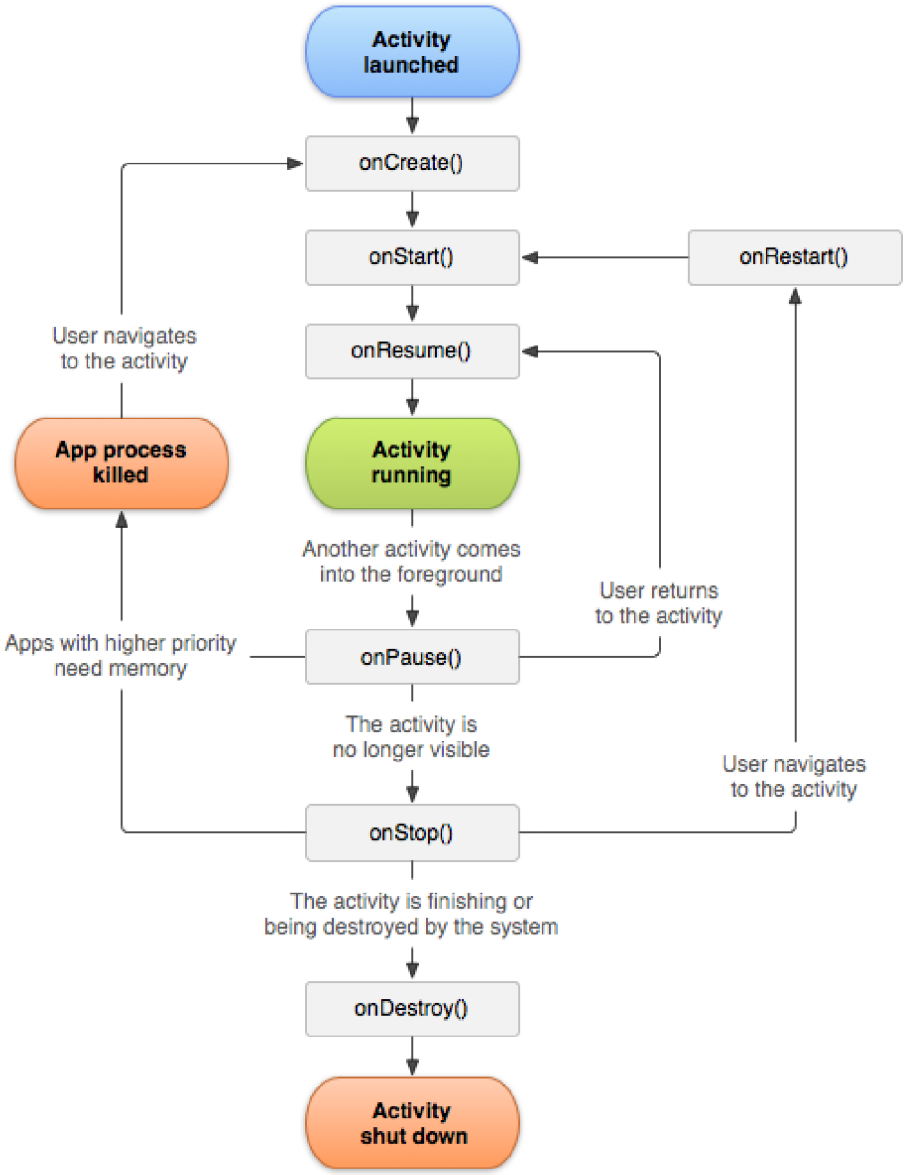
Kvůli zatím omezenému výkonu procesoru a velikosti paměti používá Android složitou správu zdrojů a zásobník aktivit (`stack`). `Stack` se snaží systém řídit tak, aby v danou chvíli byl pro uživatele nejdůležitější obsah na vrcholu zásobníku, čili aktivní a viditelný, a aby ostatní v konkrétní situaci méně důležité prvky zůstaly potlačeny na pozadí [5]. Každá aktivita je proto obecně vzato vždy v jednom z následujících stavů [26]:

- **Aktivní** – Běží v popředí a může mít uživatelský vstup.
- **Pozastavená** – Viditelná, ale část její obrazovky překrývá například nějaké upozornění a nemůže mít uživatelský vstup. K takové situaci dochází třeba při zobrazení dialogu pro příjem příchozího hovoru.
- **Zastavená** – Aktivita je skrytá za jinými aktivitami, nemůže nic přímo zobrazit, ale může s uživatelem komunikovat prostřednictvím upozornění.
- **Mrtvá** – Aktivita nebyla vůbec spuštěna, nebo byla násilně ukončena (kvůli nedostatku paměti atp.).

Android při přechodu mezi těmito stavy volá na základě životního cyklu aktivity (viz obrázek 2.4) následující callback metody [26, 5, 23]:

- `onCreate()` – Volána při vytvoření aktivity. V této metodě se standardně inicializuje aktivita včetně nastavení obsahu UI a provádí operace, které je třeba udělat na začátku a pouze jednou.
- `onStart()` – Zavolána, když se aktivita stává viditelnou pro uživatele.
- `onResume()` – Volána těsně před tím, než se aktivita přesune do popředí.
- `onPause()` – Volána, když se aktivita přesouvá do pozadí (před tím než je aktivita zcela pozastavena). Zde je dobré zrušit vše, co se provedlo v metodě `onResume()` (kupř. uvolnit jakékoliv prostředky s exkluzivním přístupem).
- `onStop()` – Volána ve chvíli, kdy se má aktivita zastavit.
- `onRestart()` – Zavolána, pokud byla předtím aktivita zastavena a restartuje se.

- `onDestroy()` – Volána při ukončování aktivity, pokud např. aktivita zavolala metodu `finish()` nebo systém kvůli nedostatku prostředků (paměť, CPU) aktivitu násilně ukončí. Proto se tato metoda hodí k čistému uvolnění prostředků. Je ale potřeba vědět, že pokud systém potřebuje urgentně uvolnit paměť, nemusí být před „zabitím“ aktivity volána vůbec.



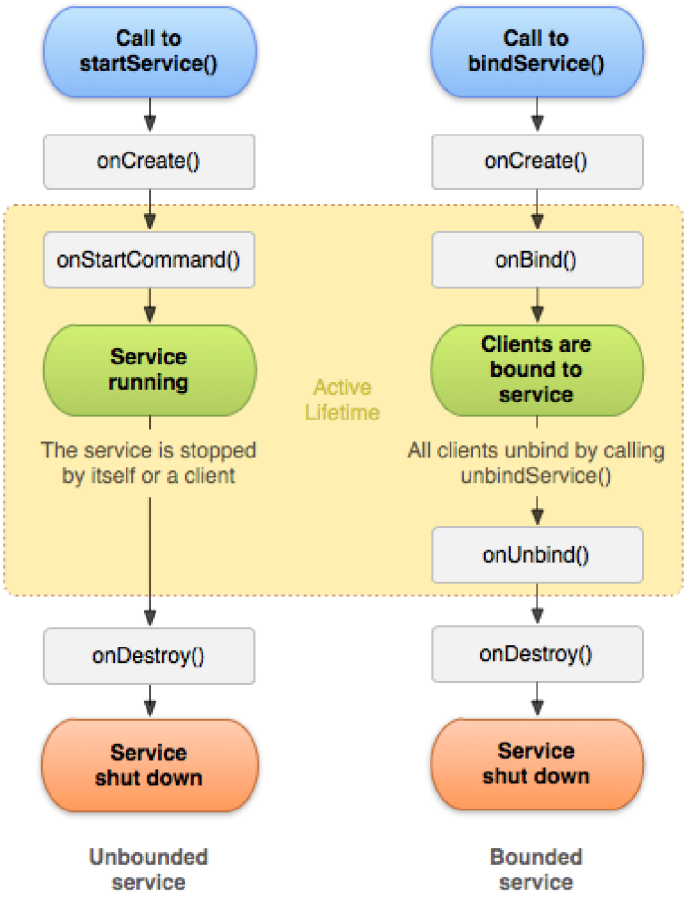
Obrázek 2.4: Životní cyklus aktivity [7].

Aktivity jsou v programu implementovány jako veřejné třídy odvozené od základních tříd `android.app.Activity` v případě základní aktivity, ale existují i třídy `ListActivity` a `ExpandableListActivity`, integrující přidanou funkcionalitu – především různé již defaultně nastylované prvky GUI. V případě `ListActivity` je to implementace `ListView`, což je jednoduchý seznam a např. u `ExpandableListActivity` je to seznam s podkategoriemi.

2.3.3 Služby

Služba je důležitá aplikační komponenta, která tvoří spolu s vlákny základní prostředek k multitaskingu na Adroidu. Služba neposkytuje žádné uživatelské rozhraní, má vlastní životní cyklus (viz obrázek 2.5) a je nezávislá na jakékoliv aktivitě. Používá se především pro provádění časově náročných operací a sdílení s nimi souvisejících informací. Služby mohou na rozdíl od aktivit běžet tak dlouho, jak je nutné nebo dokud není potřeba uvolnit paměť [26, 5, 13].

Např.: Akivita nějaké aplikace může spustit službu, která i když uživatel přejde do jiné aplikace, zůstává dále běžet na pozadí – typickou ukázkou je přehrávač hudby. Službu můžeme v naší výsledné aplikaci použít zejména k aktualizaci RSS zdroje, ukládání jednotlivých zpráv do databáze a načítání dodatečných informací k odebíraným článkům.



Obrázek 2.5: Životní cyklus služby [13].

Jednotlivé komponenty aplikace můžou služby spouštět, nebo se přivázat k již spuštěným službám a dokonce provádět meziprocesové komunikace (IPC). Pro jakoukoliv službu platí, že vždy může být spuštěna pouze jedna její kopie, i když ji využívá více aktivit (návrhový vzor Singleton). Ke službám je možné přistupovat následovně (viz také obrázek 2.5) [26, 13]:

- **Spuštění služby** – Službu lze spustit metodou `startService()`, pak je nutné službu manuálně ukončit pomocí jedné z ukončovacích metod (např. `stopService()`).

- **Přivázání služby** – Druhým a ve většině případů lepším způsobem je přivázání služby k aktivitě pomocí metody `bindService()`. Je možné se přivázat jak k již spuštěné službě, tak zavolat tuto metodu s konstantou `BIND_AUTO_CREATE`, která systému oznamuje, že pokud služba dosud neběží, má ji automaticky spustit. Přivázání služby má tu výhodu, že libovolná komponenta může přímo přistupovat k API služby. Jakmile přestane komponenta používat službu, typicky při přechodu z jedné aktivity na druhou, je nutné odvázat službu metodou unbindService(). Služba, která již není přivázaná k žádné komponentě se následně sama ukončí a není třeba ji explicitně ukončovat.

2.3.4 Dodavatelé obsahu

Tato komponenta v rámci bakalářské práce nebyla využita, ale jelikož je ve většině dostupných zdrojů uváděna mezi nejdůležitějšími, tak si je přiblížíme alespoň okrajově.

Dodavatele obsahu zajišťují úroveň abstrakce jakýchkoliv dat uložených v zařízení, která jsou přístupná více různým aplikacím. Architektura systému Android vás nepřímo navádí k tomu abyste zpřístupnili svá data i jiným aplikacím než pouze své vlastní. Toho lze dosáhnout právě vytvořením dodavatele obsahu, který vám současně poskytuje úplnou kontrolu nad způsobem přístupu k vašim datům [26].

2.3.5 Přijímače broadcastů

Broadcasty jsou na Androidu realizovány v podobě vysílání záměrů. Přijímače broadcastů umožňují nadefinovat požadovanou reakci při příjmu konkrétního záměru – např. zrušit provádění nějaké procedury, zavřít dialog s progressbarem nebo zobrazit upozornění ve stavovém panelu. Lze také přijímat systémové broadcasty, jako je nízký stav baterie, zapnutí bezdrátové sítě atp.

Je důležité zdůraznit, že tyto přijímače nejsou jediným způsobem pro reakci na záměry. Častějším řešením je nadefinování filtrů záměrů v souboru `AndroidManifest.xml`. Přijímače broadcastů se využívají především v případech, kdy chceme udělat nějakou operaci pouze za určitých okolností (pokud je při příjmu záměru Z hodnota nějaké proměnné ve vašem programu rovna X, tak spusť metodu M, jinak spusť metodu Y), nebo je potřeba provést něco uvnitř služby, a ne v aktivitě [26].

2.3.6 Záměry a filtry záměrů

Záměry (intents) zprostředkovávají komunikaci komponent a sdílení dat mezi nimi. Záměry samy o sobě nedělají žádnou práci, ale popisují něco, co se má udělat. Pokud komponenta chce provést nějakou akci, vytvoří záměr a pošle ho do systému. Systém pak tento záměr dekóduje a rozhodne, které další komponenty, aktivity, služby nebo přijímače vysílání jsou schopné požadovanou akci vykonat [5, 26]. Existují dva následující typy směrování záměrů:

- **Explicitní směrování** – Záměru je možné přímo zadat cíl pro který je určen, toho se využívá především při vytváření nových aktivit a podaktivit. Například když je potřeba z jedné aktivity spustit jinou aktivitu, vytvoří se nový záměr, kterému se sdělí, jaká aktivita se má spustit. S pomocí objektu typu `Bundle` lze ještě uložit do záměru data potřebná pro nově spouštěnou aktivitu a zavolá se metoda `startActivity`, které se jako parametr předá požadovaný záměr, nebo metoda `startActivityForResult`

pokud chceme aktivitu spustit jako podaktivitu. Podaktivity vytváříme především pokud potřebujeme znát nějaký výsledek činnosti spouštěné aktivity po jejím ukončení, ten potom získáme v metodě `onActivityResult`. Explicitní směrování se z důvodu bezpečnosti zásadně nedoporučuje používat pro komponenty mimo vlastní aplikaci.

- **Implicitní směrování** – Pokud cílový komponent nespecifikujete, musí systém sám určit, která komponenta se má spustit. Pro příjem záměrů, které nemají explicitně určený cíl je třeba u komponenty nadefinovat filtry záměrů. Systém na základě těchto filtrů určuje, která komponenta je nejvhodnější pro příjem záměru. Pokud nadefinované filtry odpovídají vysílanému záměru, je dané komponentě záměr předán. Filtry záměrů se definují v AndroidManifest.xml souboru, ze kterého jsou systému k dispozici. Na jeden záměr může reagovat i několik komponent současně [26].

2.3.7 Soubor Manifest

Všechny Android aplikace musí v kořenovém adresáři projektu obsahovat konfigurační soubor `AndroidManifest.xml`, který systému mimo jiné sděluje jaké komponenty vaše aplikace využívá, co pro svůj provoz potřebuje a co všechno umí. V manifestu je tedy nutné uvést všechny důležité informace – minimální podporovanou verzi API, verzi aplikace, ikonu a název aplikace, názvy jednotlivých komponent (aktivity, služby atd.), informaci o tom, která aktivita je startovací obrazovkou aplikace a různá oprávnění vyžadovaná pro správný provoz – povolení přístupu k internetu, práce s fotoaparátem atp. Nepovinné položky jsou např. nastavení podpory různých rozlišení displeje a povolení rotace obrazovky [6, 5, 26].

2.3.8 Asynchronní úlohy

Jak už bylo uvedeno (viz 2.3), každá aplikace na Androidu má vlastní UI vlákno. V tomto vláknu běží všechny komponenty aplikace, jako aktivity, služby, dodavatelé obsahu atp. Systém Android nedovoluje aktivitám blokovat uživatelské rozhraní přibližně déle než pět sekund, jakékoliv náročnější déletrvající operace běžící v uživatelském vlákně můžou skončit noční můrou pro mnoho vývojářů v podobě varování ANR – Activity Not Responding. Což často vede k frustraci uživatele a následnému odinstalování aplikace. Proto by se všechny časově i výkonově náročné úlohy měly provádět zásadně ve vláknech běžících na pozadí. Android disponuje spoustou prostředků pro správu vláken na pozadí a zároveň jim umožňuje bezpečně interagovat s uživatelským rozhraním. Od verze Android 1.5 se k těmto prostředkům řadí také třída `AsyncTask`, která se stala nejčastěji používanou koncepcí pro práci s vlákny na pozadí [26, 5].

Třída `AsyncTask` zahrnuje veškerou potřebnou funkcionalitu pro práci s vlákny na pozadí a jejich komunikaci s vláknem UI pomocí k tomu určených metod. Navíc Android takto vytvořená vlákna sám alokuje i odstraní. Instance třídy je generický typ, pro který je nutno specifikovat generika v tomto pořadí:

- Typ vstupních parametrů potřebných pro provedení požadovaného úkolu.
- Typ informace, která se předává v rámci úkolu a slouží jako indikátor průběhu.
- Návratový typ, předávaný níže uvedené obslužné funkci po skončení úkolu prováděného na pozadí.

Třída `AsyncTask` nabízí několik důležitých metod:

- `onPrepareProgress()` – Volá se před prováděním vlákna na pozadí.
- `doInBackground()` – Uvnitř této metody se definuje, co přesně se má provést ve vlákně na pozadí. Jako jediná se musí vždy přepsat.
- `onProgressUpdate()` – V těle metody `doInBackground()` je možné volat metodu onPublishProgress() jejíž parametrem je druhý výše uvedený generický typ. Volání metody onPublishProgress() spustí provádění metody `onProgressUpdate()`.
- `onPostExecute()` – Je zavolána po skončení provádění úlohy na pozadí.
- `onCancelled()` – Volá se místo metody onPostExecute() pokud byla úloha během provádění na pozadí přerušena.

Všechny metody kromě metody doInBackground(), která pracuje v novém vlákně na pozadí, pracují ve vlákně UI. Umožňují tak synchronizovaně komunikovat s komponentami aplikace. Žádná z těchto metod nesmí být v kódu volána, o jejich volání se implicitně stará třída `AsyncTask`. Provádění požadované činnosti instance třídy `AsyncTask` se spouští voláním její metody `execute()`. Objekt `AsyncTask` může být spuštěn vždy pouze jednou, pro každé spuštění je tedy nutné vytvořit novou instanci. Třída `AsyncTask` se často implementuje jako vnitřní třída aktivity, ale pokud potřebujeme provádět delší činnosti nezávisle na životním cyklu aktivity, implementuje se uvnitř služby.

2.3.9 Třída Application

Bázovou třídu `Application` lze použít pro udržování globálního stavu aplikace (sdílení netriviálních a neperzistentních dat). Objekt této třídy existuje pro každou aplikaci vždy pouze jeden a instancuje se ve chvíli, kdy je vytvořen nový proces pro aplikaci.

Oficiální referenční dokumentace (viz [12]) sice říká, že takové použití bázové třídy běžně není potřeba, protože je možné ji nahradit více modulárním návrhovým vzorem Singleton, naopak jiná odborná publikace (viz [5]) věnující se této problematice její použití vřele doporučuje, protože má velice dobře nadefinovaný vlastní životní cyklus. Díky tomu je pro mobilní aplikaci vhodnější než standardní implementace statického Singletonu, u kterého by mohlo na mobilních zařízeních snadno dojít k únikům paměti.

Současně se zde ale také uvádí, že pokud preferujete Singleton, tak zlatou střední cestou je jeho implementace s využitím té části zdrojového kódu bázové třídy `Application`, která zahrnuje ošetření událostí životního cyklu. Tak získáte to nejlepší z obou řešení [5].

Kapitola 3

Syndikace obsahu

Pod pojmem webová syndikace obsahu se skrývá technologie umožňující opětovné dodávání informací uvedených na internetu, která je založená na formátech RSS a Atom. Existuje dnes několik desítek milionů webů, které prostřednictvím této technologie poskytují svůj obsah nebo jeho část k dispozici [16]. V rámci této kapitoly se blíže seznámíme s využitím webové syndikace obsahu, jednotlivými formáty, vyhledáváním a čtením odběrů.

3.1 Možnosti využití

Jako první z možností využití je potřeba uvést **distribuci obsahu** určeného k odběrům, která stránkám takto nabízejícím svůj obsah přináší různé benefity (lepší výsledky ve vyhledávání, zvýšení návštěvnosti atp.) a odběratelům usnadňuje cestu ke kýženým informacím.

Obvyklým využitím syndikace obsahu je **umístění obsahu z cizího webu**, např. výpis výsledků sportovních utkání z více různých zdrojů na fanouškovských stránkách (blogu). Na tomto principu jsou také založeny tzv. agregátory článků, které shromažďují informace z vybraných oblastí (viz 3.3.1). Umístění kvalitního a relevantního cizího obsahu může navíc příznivě ovlivnit výsledky ve vyhledávačích, to je už ale záležitost spadající do oblasti optimalizace pro vyhledávače (SEO) a nebudeme se jí více věnovat.

Dalším nejčastějším použitím technologie je odebírání obsahu prostřednictvím **RSS čtečky**. Jak již bylo na začátku uvedeno (viz 1), největším přínosem čteček je možnost na jednom místě soustředit články a informace z mnoha odebíraných zdrojů. Daná problematika je stěžejní pro tuto práci, proto se čtečkami budeme podrobněji zabývat ve vlastní podkapitole (viz 3.4).

Zajímavým využitím webové syndikace jsou i služby nabízející **personalizované (domovské) stránky**, jako třeba iGoogle, Seznam, Netvibes, Genieo a další. Podstatou je, že si v podobě odběrů z RSS/Atom zdrojů uživatel nastaví libovolný obsah stránky, od oborů, o které se zajímá, přes počasí až po novinky z obchodu.

Obecně už méně známé aplikace této technologie představují některé **analytické a monitoringové nástroje**. Umožňují např. zjistit či pravidelně kontrolovat, jestli se nějaká informace, osoba, firma, událost atd. neobjevila někde na internetu. K tomu jim slouží objemné databáze odebíraných informačních zdrojů. Některé z těchto nástrojů umí také monitorovat a analyzovat dění na sociálních sítích prostřednictvím odběrů, které můžou dávat uživatelé těchto sítí k dispozici.

3.2 Formáty RSS a Atom pro syndikaci obsahu

Dříve, než se tyto formáty objevily, existovala celá řada jejich předchůdců, kteří se už ale nepoužívají, nebo se používají jen zřídka. RSS i Atom vychází ze značkovacího jazyka XML. Mají tedy obsah ve strukturované formě složené z několika elementů, některé elementy musí být v souboru uvedené vždy, jiné jsou volitelné. Syndikace obsahu už za svoji existenci stihla projít poměrně chaotickým vývojem, kdy jednotlivé formáty vznikaly i zanikaly a jedna verze střídala druhou. Tyto verze i formáty se bohužel v mnohém liší, což musíme při návrhu (viz kapitola 4) RSS čtečky mít na paměti.

3.2.1 RSS

RSS je zkratkou nejméně pro tři různá spojení, ovšem tím nejpoužívanějším je Realy Simple Syndication. První skutečnou verzi formátu RSS vyvinul Dan Libby ze společnosti Netscape v roce 1999 označovanou jako verzi 0.90, po ní se objevila první všeobecně známá verze 0.91. Během vývoje došlo ke zmatkům, které vyústily k rozdvojení standardu od verze 1.0. Jejím základem se stal XML-jazyk RDF (Resource Description Format), ten ji od předchozích verzí značně odlišoval. Po této verzi, kterou bylo pro řadu uživatelů velmi obtížné používat, se objevuje verze 0.92, která proto vychází z verze 0.91. Poté rychle následuje verze 0.93 a 0.94. Aby to bylo ještě trochu složitější, jako další byla konečně vydána verze 2.0, která ale vychází z verze 0.92. Verze RSS 2.0 je dnes jedním z nejpoužívanějších formátů pro syndikaci obsahu, po ní následuje Atom (viz dále) a v menším množství se můžeme stále setkat s RSS 0.91 a 1.0 [16]. U vyvíjené RSS čtečky se v rámci bakalářské práce počítá hlavně s podporou formátu RSS 2.0. Struktura XML souboru v tomto formátu může vypadat následovně:

```
<?xml version='1.0' encoding='UTF-8'?>
<rss version="2.0">
  <channel>
    <title>Název odběru</title>
    <link>http://www.strankaposkytujiciobsah.cz/rss/</link>
    <description>Informace o odebíraném zdroji</description>
    <language>cs</language>
    <pubDate>Mon, 10 Sep 2011 08:00:00 GMT</pubDate>
    <lastBuildDate>Tue, 15 May 2012 21:24:36 GMT</lastBuildDate>
    <item>
      <title>Název článku</title>
      <link>http://www.strankaposkytujiciobsah.cz/zprava.htm</link>
      <description>Zkrácený nebo celý obsah zprávy</description>
      <pubDate>Tue, 15 May 2012 21:24:36 GMT</pubDate>
    </item>
  </channel>
</rss>
```

Elementy a atributy formátu jsou ve většině případů samopopisné. Soubor vždy začíná informací o verzi XML a kořenovým elementem `rss`, který je povinný a informuje o verzi formátu RSS. Jeho přímým potomkem je element `channel`, který je také povinný a obaluje všechny ostatní elementy. Pro naši práci budou nejdůležitější přímí potomci elementu `channel` jako jsou: `pubDate`, který obsahuje datum poslední aktualizace ve formátu RFC 822, dále `title`, `link`, `description` a `item`. Kromě elementu `pubDate` jsou všechny tyto

elementy povinnou součástí formátu RSS 2.0. Element `item` představuje jednotlivé záznamy, například články zpravodajského portálu. Počet těchto záznamů je neomezený. U elementu `item` nás budou opět zajímat podelementy `title`, `link`, `description` a element `pubDate`, který v tomto případě značí datum vydání daného článku. Všechny tyto podelementy jsou nepovinné, ale alespoň `title` nebo `description` musí být vždy přítomen. Podrobnější popis je možné dohledat v oficiální specifikaci (viz [28]), ale pro naši potřebu jsou uvedené informace dostačující.

3.2.2 Atom

Standard Atom byl vyvinut skupinou vývojářů, včetně Sama Rubyho, jako náhrada za starší rozporuplné RSS. Oproti RSS je o něco složitější a o něco těžší na pochopení. Atom vylepšil RSS především přidáním podpory pro více jazyků, standardizací syntaxe a umožněním přidávat vlastní prvky. V roce 2003 vyšla první verze Atom 0.2, během téhož roku se objevila i verze 0.3 a po ní následovala verze 1.0. Tuto verzi přijala mezinárodní komise IETF jako standard definovaný v RFC 4287 [27]. Atom 1.0 navíc zahrnuje specifikaci publikačního protokolu určeného pro editaci blogů a podobných webových stránek, ten je obsahem RFC 5023 [15, 2, 16].

3.3 Vyhledávání odběrů

Abychom vůbec měli co odebírat, je potřeba vyhledat požadované informační kanály, které jsou k dispozici. U většiny stránek podporujících tuto technologii najdete někde v obsahu, často na začátku stránky, podobnou ikonu, jako je na obrázku 3.1. Ta vede na XML soubor (může jich být i více), který je určený k odběru. Některé lepší čtečky samy najdou tento soubor a nabídnou ho uživatelům automaticky. Další možností je vlastní vyhledávání zdrojů k odběru (často je i součástí čteček) například následujícími dostupnými prostředky:

3.3.1 Katalogy a agregátory

Katalogy a agregátory umožňují vyhledávat ve své databázi odběrů a odebíraných článků, např. Syndic8.com, Weblogy.cz, PrávěDnes.cz. Tato databáze ale neobsahuje všechny zdroje informací, které byste potenciálně mohli chtít odebírat. Bývá spíše výběrem z různých oborů a oblastí zájmů. Agregátory ani katalogy článků nemají většinou žádný svůj obsah, pouze dále reprodukují a zpracovávají obsah z RSS/Atom zdrojů. Často mají i přidané funkce pro hodnocení odběrů i různé analytické a monitoringové nástroje (viz 3.1).

3.3.2 Bing

Vyhledávač Bing od společnosti Microsoft umí hledat zdroje odběrů prostřednictvím příkazu `feed:` před hledaným termínem (klíčovým slovem, částí adresy, či celou adresou webu). Např. `feed: android developers` najde jako první položku RSS oficiálních stránek vývojářů pro tuto platformu.

3.3.3 Google

Google nemá vlastní vyhledávací příkaz pro vyfiltrování RSS/Atom zdrojů přímo z domovské stránky (resp. ze standardního vyhledávacího pole), místo toho ale nabízí pro tento účel pohodlné vestavěné API. Např. pomocí následujícího dotazu:

`https://ajax.googleapis.com/ajax/services/feed/find?v=1.0&q=HLEDANY_VYRAZ`

vrátí Google nalezené položky ve formátu JSON, které je možné dále dle potřeb zpracovávat. Pro přístup do tohoto API není potřeba žádná registrace ani vlastní účet na Google. Toto API přímo využívá i online čtečka od Google – Google Reader.

Obrázek 3.1: Ikona RSS.

3.4 Čtečky odebíraných zdrojů

Běžný uživatel internetu je dnes utopen ve spamu, reklamě a dalších nerelevantních informacích, které se na něj valí ze všech koutů sítě. Proto vznikly RSS čtečky – prostředek, který se snaží uživatelům zpříjemnit i urychlit cestu k požadovaným informacím a šetřit tak jejich čas. Od chvíle přihlášení k vybraným odběrům začíná uživatel čtečky přijímat automaticky aktualizované informace ze všech odebíraných zdrojů. Díky tomu např. může přímo z čtečky snadno sledovat veškeré novinky z oblasti jeho zájmů na oblíbených stránkách.

Množství ušetřeného času rychle stoupá spolu s počtem sledovaných zdrojů. Z toho důvodu jsou nejvděčnějšími uživateli čteček především novináři a redaktoři, kteří potřebují sledovat často i obrovské množství informačních kanálů, ale také ledajací odborníci, kteří chtějí mít přehled o novinkách v oboru.

Čtečky odběrů mají mnoho různých podob. Můžou být vestavěné uvnitř webového prohlížeče libovolného zařízení s připojením k internetu, nebo jsou to samostatné aplikace a programy pro stolní počítače i mobilní zařízení (od notebooků přes mobilní telefony až po tablety). Existují také online RSS čtečky, např. Google Reader. Pokročilejší čtečky nabízí různé užitečné funkce pro pohodlnou práci s odebíranými zdroji, jako třídění do kategorií dle preferencí, sdílení na sociálních sítích, filtrování, řazení, vyhledávání atp. Nově se objevují i aplikace, které se snaží nahradit denní tisk formou kompletně personalizovaného obsahu, např. Flipboard. Jsou to prakticky v základu RSS čtečky, ale se spoustou další funkcionality včetně přístupu na sociální sítě a propracovaného uživatelského rozhraní.

Někteří uživatelé můžou namítat, že RSS čtečky jsou „mrtvé“ a nahradily je sociální sítě v čele s Twitterem. Není tomu tak, ačkoliv je obsah z odebíraných zdrojů někdy totožný s obsahem publikovaným na sociálních sítích, ve většině případů to neplatí. Stále také existuje početná skupina lidí (používajících RSS čtečky) bez jediného účtu na sociálních sítích. A jako poslední důvod můžeme uvést, že ne každý uživatel sociální sítě chce mít svůj profil plný novinek, ačkoliv z oblasti jeho zájmů. To, jestli jsou lepší pro sledování dění na internetu RSS čtečky nebo sociální sítě či obojí, je čistě na preferencích uživatele, ale RSS čtečky „mrtvé“ rozhodně nejsou.

Kapitola 4

Návrh RSS čtečky pro Android

Než se pustíme do samotného návrhu vlastní RSS čtečky, seznámíme se s hotovými aplikacemi na trhu. Poté vymezíme cestu, kterou se bude ubírat vlastní vývoj, následně rozebereme možnosti řešení ukládání dat a navrhneme konceptuální model databáze. V závěru kapitoly se budeme zabývat podobou uživatelského rozhraní.

4.1 Průzkum existujících řešení

Na základě hodnocení uživatelů [14, 21, 20, 3, 19, 25] a vlastních praktických zkušeností můžeme mezi nejpovedenější dostupné RSS čtečky zařadit především **gReader**, oficiální mobilní aplikaci od Google – **Google Reader**, **Pulse**, **Taptu** a **NewsRob**. Každá z těchto aplikací se nějakým způsobem snaží odlišovat a nabízí různé pokročilé funkce od kategorizace odběrů přes ukládání článků až po práci se sociálními sítěmi a pokročilé možnosti synchronizace.

Aplikace Google Reader, NewsRob a gReader používají API online služby Google Reader. Díky tomu umožňují snadnou synchronizaci s touto online čtečkou odběrů, což je velice užitečná a často i vyžadovaná vlastnost u dnešních RSS čteček. Současně je ale uživatel nepřímo nucen používat tuto službu. To se ale každému nemusí líbit, nemluvě o uživatelích obávajících se potencionálně možného zneužití jejich soukromých informací tzv. „velkým bratrem“, za kterého je dnes Google často veřejností označován.

Oficiální mobilní aplikace Google Reader má nejjednodušší a nejpřívětivější uživatelské rozhraní, ale nenabízí tolik funkcí jako další uvedené čtečky. Aplikace Pulse má povedené inovativní a propracované uživatelské rozhraní, ale limituje počet odebíraných zdrojů, čímž bohužel značně klesá velikost cílové skupiny, která by měla zájem aplikaci používat. Nejkomplexnější řešení představují gReader a Taptu. Aplikace gReader je propracovaná RSS čtečka s velmi intuitivním rozhraním, pokročilými možnostmi kategorizace, rozsáhlou nabídkou doporučených zdrojů a je možné si ji zcela přizpůsobit pomocí mnoha uživatelských nastavení. Volitelných nastavení je ale tolik, že je poměrně těžké se v nich zorientovat. Taptu nabízí např.: plnou podporu sociálních sítí jako je Twitter, Facebook, LinkedIn, spoustu přednastavitelných odběrů, volitelné propojení s online službou Google Reader, inovativní rozhraní podobné Pulse, ale bez omezení počtu odběrů. To vše je bohužel částečně vykoupeno rychlostí aplikace a zobrazováním velkého množství nahuštěných informací na malé ploše, které má za následek horší přehlednost i ovladatelnost. NewsRob nepřináší ve srovnání s předchozími aplikacemi nic zásadního, spíše za nimi zaostává např. v propracovanosti uživatelského rozhraní.

Nebudeme zde podrobněji rozepisovat všechny výhody a nevýhody, ale po předchozím průzkumu a otestování dostupných aplikací můžeme konstatovat, že i přes velké množství dostupných řešení tu je stále místo pro další podobné produkty. Především z důvodu, že i u nejpoužívanějších aplikací se objevují různé nedostatky jako je limitování počtu zdrojů, vyžadovaná registrace, nepřehlednost, pomalá odezva, malá velikost dotykových ploch tlačítek a dalších aktivních prvků, irelevantní obsah atd. Zkrátka je a bude stále co zlepšovat. Mezi taková zlepšení můžeme zařadit např. různé kvalitativní nástroje pro hodnocení odebíraného obsahu, vylepšení uživatelského rozhraní atd.

4.2 Návrh vlastního řešení a očekávaný přínos

Vzhledem ke komplexnosti a propracovanosti dostupných čteček bylo potřeba postavit se k vývoji jiným způsobem, než jen vytvořit jednoduchý hotový produkt s omezenou funkcionalitou. Po zhodnocení trhu se stalo hlavní myšlenkou vyvíjet v rámci bakalářské práce platformu, která bude nabízet postupem času to nejlepší z výše uvedených aplikací a nejen to. Vlastní řešení by mělo uživateli umožňovat především snadné ovládání složité funkcionality, nabízet přehledné volby několika způsobů zobrazení a kategorizace obsahu, dále také možnost porovnávat odebírané zdroje podle určitého kvalitativního indikátoru. S postupem času také plnou podporu sociálních sítí, stylovatelné uživatelské rozhraní, možnost plné obousměrné synchronizace s Google Reader a mnoho dalšího. Z celkové navrhované funkcionality byla pro tuto bakalářskou práci s ohledem na její rozsah vybrána následující podmnožina, další možnosti rozšíření a vize budoucího vývoje budou uvedeny v závěru práce (viz 6).

Správa odebíraných zdrojů:

- Vyhledávání zdrojů podle přesné adresy i klíčových slov.
- Ukládání, aktualizace a mazání odebíraných zdrojů.
- Vytváření kategorií zdrojů.
- Editace názvu vytvořených kategorií.
- Mazání vytvořených kategorií.
- Hromadné označování všech článků ve zdrojích nebo kategoriích jako přečtených.

Zobrazování odebíraných zdrojů:

- Zobrazení informací o odebíraných zdrojích a kategoriích jako je název zdroje/kategorie a počet nepřečtených článků.
- Filtrování kategorií i zdrojů bez nepřečtených článků.
- Řazení kategorií a zdrojů podle preferencí uživatele.

Zobrazování odebíraných článků:

- Možnost prohlížet články podle jednotlivých zdrojů, ale i všech odebíraných článků ze všech zdrojů současně.
- Vizuální odlišení přečtených a nepřečtených článků.

- Filtrování přečtených a nepřečtených článků.
- Řazení článků podle data sestupně i vzestupně.
- Implementace určitého kvalitativního indikátoru u odebíraných článků a umožnění řazení podle tohoto indikátoru.
- Několik variant zobrazení výpisu článků, např. pouze titulek, nebo titulek, perex a datum, případně obrázky, různá rozložení prvků apod.
- Zobrazení náhledu zprávy s možností prohlédnutí celé zprávy v prohlížeči.
- Nabídka dalších článků z dané kategorie podle vybraných indikátorů u detailu článku.

Práce s odebíranými články:

- Možnost ponechat přečtený článek jako nepřečtený.
- Sdílení článku s přáteli.
- Přidávání článku mezi oblíbené položky.
- Skupinové označování článků jako přečtených.

Integrace:

- Propojení s vestavěným nástrojem pro sdílení.
- Propojení s akcí (viz 2.3.6) související s načtením souboru RSS zdroje v internetovém prohlížeči.
- Uložení uživatelských preferencí nastaveného zobrazení a obnovení tohoto nastavení po startu aplikace.

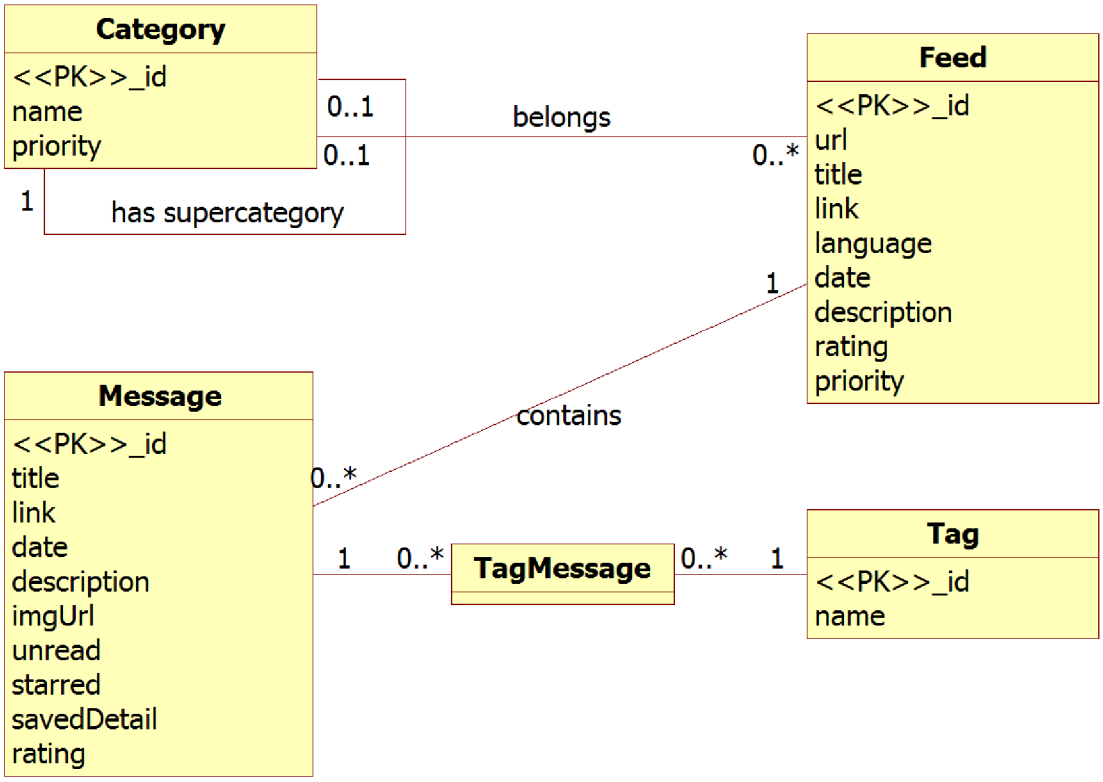
4.3 Ukládání dat a nastavení

Je zřejmé, že data aplikace bude potřeba někam ukládat. Zde stojí za zvážení, zda je lepší ukládat data do XML souboru, nebo do relační databáze. Z navrhované funkcionality můžeme odvodit, že se nad daty bude často provádět mnoho netriviálních operací, pokud k tomu přičteme výkonová omezení mobilních zařízení, dojdeme k závěru, že pro tento účel je nejvhodnější použít databázi. Přímo součástí Androidu je databáze SQLite, kterou Android implementuje celou a v rámci SDK (viz 2.2.1) nabízí různé vestavěné funkce pro práci s touto databází.

Pro ujasnění a přesné definování toho, co je vše potřeba ukládat jako perzistentní data, byl v rámci návrhu vytvořen ER [1] diagram neboli entitně-vztahový diagram, který můžeme vidět na obrázku 4.1. Všechny entitní množiny zde mají jako primární klíč atribut `_id`, jelikož je vyžadován u potřebných komponent jako jsou různé adaptéry pro zobrazování obsahu z databáze, bez kterých bychom se těžko obešli. Atributy entitních množin byly určeny na základě průzkumu osvědčených řešení, specifikace formátů pro odebírání webového obsahu (viz 3.2 a [28]) a navrhované funkcionality. V návrhu vidíme, že každý zdroj může být odebírán pouze jednou a tento zdroj může, ale nemusí patřit do určité kategorie.

[1] ER – Entity-Relationship

Atribut `starred` bude ukládat informaci, zda byl článek označen jako oblíbený, atribut `rating` uchovává hodnotu určitého kvalitativního indikátoru, atributy `url`, `link`, `title`, `date`, `description`, `language` odpovídají jednotlivým obsahovým elementům načítaným z odebíraných zdrojů. Entitní množina `Tag`, vazební entitní množina `TagMassege` a atributy `savedDetail`, `imgUrl`, `language` a `priority` nebudou v rámci této bakalářské práce použity, ale s ohledem na budoucí rozšiřování aplikace bylo potřeba s nimi počítat už při návrhu databáze.



Obrázek 4.1: ER diagram

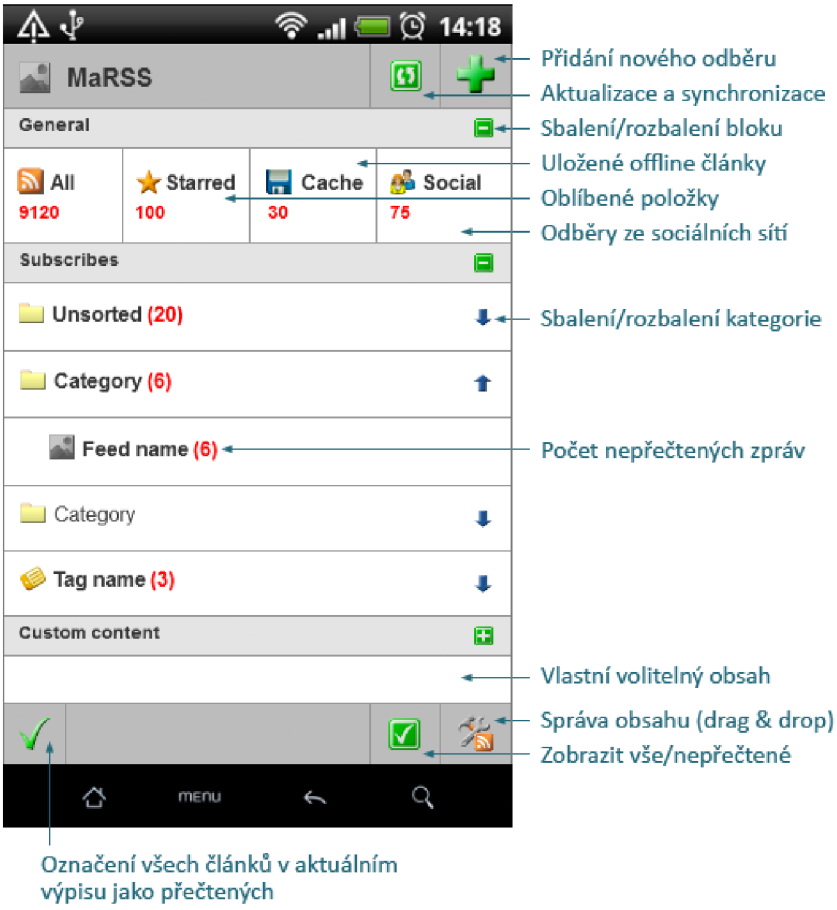
Aby bylo možné po opětovném startu aplikace obnovit její předchozí stav a při přechodu mezi aktivitami respektovat uživatelské předvolby, bude také potřeba zaznamenávat informace jako jsou poslední nastavená zobrazení, filtrování, řazení apod. Pro tento účel nabízí Android opět vlastní řešení v podobě takzvaných preferencí, které toto ukládání usnadňují a můžeme je tedy použít i pro naši RSS čtečku.

4.4 Uživatelské rozhraní

Jak jsme již uvedli, uživatelské rozhraní by mělo poskytovat snadné ovládání složité funkcionality, nabízet přehledné volby několika způsobů zobrazení, kategorizace a řazení obsahu. Úvodní obrazovka aplikace (viz obrázek 4.2) by měla zahrnovat určitou modularitu obsahu a logické uspořádání jednotlivých prvků, to platí i pro ostatní sekce. Současně by neměla chybět informace o aktuálně nepřečtených zdrojích a tlačítko pro aktualizaci/synchronizaci odběrů. Užitečnou vlastností je také možnost skrývat/zobrazovat jednotlivé hlavní obsahové bloky, aby bylo na obrazovce více místa pro ostatní informace. Jedním z nejdůležitějších

prvků je tlačítko pro přesun do sekce s vyhledáváním a přidáváním odběrů. Dále by hlavní sekce měla zahrnovat kategorie odebíraných zdrojů a možnost filtrovat přečtené/nepřečtené články. Do budoucna se počítá s mechanismem drag & drop správy obsahu a prostorem pro vlastní vybraný obsah. Na obrázku 4.3 je jedna z variant výpisu odebíraných článků, těchto variant bude ve výsledné aplikaci více a bude možné nadefinovat, jaké informace se mají ve výpisu objevovat. Důležité je zde snadno dostupné filtrování a řazení ve spodní části obrazovky. Předposlední obrázek 4.4 ukazuje budoucí podobu rozložení sekce detailu článku, mimo odebírané informace obsahuje i užitečná akční tlačítka a nabídku dalších článků z dané kategorie. Samozřejmostí je možnost prohlédnout si celou odkazovanou stránku v prohlížeči. Na posledním obrázku 4.5 je návrh sekce pro vyhledávání a přidávání zdrojů z internetových stránek nebo ze sociálních sítí či předpřipraveného katalogu odběrů z různých oblastí.

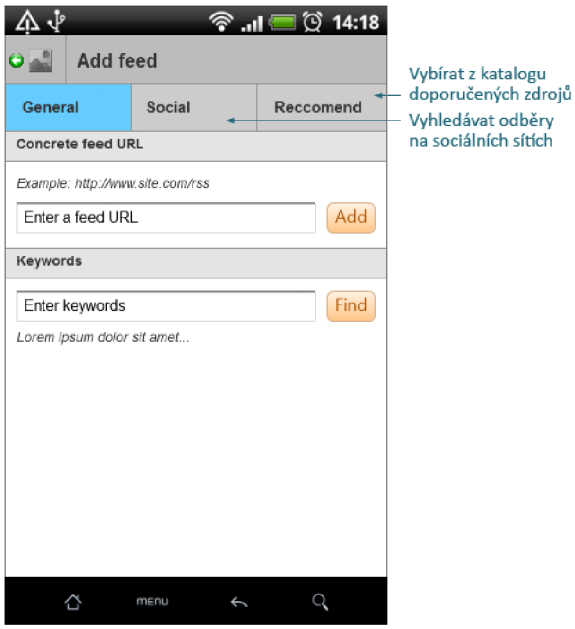
Pro návrh uživatelského rozhraní byla použita časově omezená verze profesionálního programu Axure RP, což je nástroj pro tvorbu drátěných modelů tzv. wireframes a interaktivních prototypů (více na stránkách výrobce viz [1]).



Obrázek 4.2: Úvodní obrazovka aplikace

Obrázek 4.3: Výpis odebíraných článků

Obrázek 4.4: Detail článku



Obrázek 4.5: Vyhledávání a přidávání odběrů

Kapitola 5

Implementace RSS čtečky pro Android

Na základě informací uvedených v kapitole 2.2 byl pro implementaci zvolen jazyk Java spolu s použitím vývojového kitu Android SDK (viz 2.2.1) a pluginu ADT (viz 2.2.2) pro IDE Eclipse. Aplikace mimo jiné implementuje vícevláknové zpracování, parsery JSON[1]/XML[2] formátů, vlastní adaptéry, komunikaci mezi komponentami prostřednictvím zásílání záměrů a jejich obsluhou uvnitř přijímačů broadcastů (viz 2.3.5). Dále také SQLite databázi a uživatelské rozhraní založené na XML. Obsahem následujícího textu bude popis stěžejních částí implementace RSS čtečky.

5.1 Struktura projektu

V rámci projektu nese aplikace kódové označení `MaRSS`. Pro lepší přehlednost a logické členění zdrojových souborů byly vytvořeny následující balíky tříd:

- `android.marss` – Hlavní balík aplikace, který obsahuje jednotlivé aktivity (viz 2.3.2) včetně úvodní obrazovky (třída `MainActivity`) a vzájemně související další základní komponenty aplikace, např.: adaptéry, přijímače broadcastů (viz 2.3.5) a abstraktní třídy pro efektivnější komunikaci s dalšími komponentami. Mezi základní komponenty aplikace, patří také vlastní podtřída bázové třídy Application (viz 2.3.9). Tato třída slouží jako určitá alternativa za Singleton objekt a zprostředkovává všem ostatním komponentám přístup ke správci databáze, kontrolu konektivity a nastavuje určité signalizační proměnné tzv. flagy, které používá pro ukládání dočasných informací o stavu aplikace.

- `android.marss.data` – Balík zahrnující veškerou potřebnou funkcionalitu pro práci s databází včetně definicí tabulek a jednotlivých tříd DAO[3] objektů.

- `android.marss.services` – Obsahem tohoto balíku jsou třídy služeb (viz 2.3.3), které zprostředkovávají ukládání a aktualizaci RSS zdrojů i načítání a aktualizaci hodnocení popularity jednotlivých odebíraných článků. Blíže se jimi budeme zabývat v samostatných podkapitolách 5.5 a 5.6.

[1] JSON – JavaScript Object Notation
[2] XML – Extensible Markup Language
[3] DAO – Data Access Object

- `android.marss.vos` – Tento balík zahrnuje všechny samostatné třídy objektů takzvaných Value Objects (VOs) nebo také Plain Old Java Objets (POJOs) určených pro transport dat.
- `android.marss.xml` – Součástí posledního balíku je rozhraní zprostředkovávající parsování RSS zdrojů přes implementaci `XML Pull Parseru` ve třídě `XmlPullFeedParser`.

5.2 Ukládání dat a nastavení

Pro ukládání perzistentních dat byla použita databáze SQLite v kombinaci se záznamem uživatelských nastavení pomocí vestavěného mechanismu tzv. preferencí, které Android pro tento účel nabízí. Databáze implementuje navrhovaný ER diagram (viz 4.1) v podobě tabulek tříd: `MessageTable`, `FeedTable`, `CategoryTable`, `TagTable` a `TagMessageTable`.

Veškeré potřebné operace nad daty jsou ostatním komponentám dostupné přes rozhraní `DataManager`, které implementuje třída `DataManagerImpl`. Třída `DataManagerImpl` má přímý přístup k DAO objektům, na jejichž základe staví vlastní funkcionalitu potřebnou pro jednotlivé části aplikace. Náročnější databázové operace jsou vloženy do transakcí, které zajišťují jejich atomické provedení, nebo vrácení procesu do původního stavu tzv. rollback.

V rámci aplikace bylo potřeba ukládat předvolby řazení, filtrování a nastavené typy zobrazení. Jednotlivé hodnoty těchto předvoleb se ukládají a načítají pomocí instance `SharedPreferences` a metod, které její API nabízí, např. `getSharedPreferences`, `edit` a různé settery a gettery.

5.3 Vícevláknové zpracování

Pro realizaci vícevláknového zpracování je využito kombinace několika podtříd abstraktní třídy `AsyncTask` (viz. 2.3.8). V některých případech jsou podtřídy implementované ve službách, aby se zamezilo jejich ukončení po opuštění aktivity. Vzhledem k požadovanému velkému množství současně spouštěných úloh pro potřeby ukládání i aktualizace odebíraných zdrojů a načítání dodatečných informací byla převzata původní oficiální implementace třídy `AsyncTask` z vyšší verze SDK a poupravena pro potřeby aplikace. Jedním z hlavních důvodů tohoto kroku byla standardně u nižších API nepodporovaná metoda `onCancell`, bez které bychom se těžko obešli, a také bylo vhodné vyhnout se redundantnímu nastavování parametrů pro každou asynchronní úlohu.

Jednotlivá vlákna komunikují s ostatními komponentami aplikace většinou prostřednictvím vysílání záměrů. Po jejich zachycení s pomocí přijímačů broadcastu je vyvolaná odpovídající reakce na daný typ záměru, např.: zastavení progress baru, deaktivace dialogu, upozornění o dokončení nějakého úkolu apod.

Při implementaci asynchronních úloh bylo třeba zjistit, jaký je optimální počet maximálně spuštěných vláken. Z počátku se totiž automaticky vytvářelo až tři sta samostatných vláken, což samozřejmě nebylo v pořádku. Následně byly algoritmy přepracované tak, aby bylo možné nastavovat počty vláken pomocí konstant `MAX_CONCURRENT_UPDATE_FEED_TASKS` a `MAX_CONCURRENT_COUNT_RATING_TASKS`, které jsou umístěny spolu s dalšími systémovými konstantami ve třídě `Constants` balíku `android.marss`. Na základě testování byly hodnoty nastaveny na maximálně dvacet vláken pro úlohy související s načítáním popularity na sociálních sítích (viz 5.6) a deset vláken pro ukládání a aktualizaci zdrojů k odběru.

5.4 Vyhledávání odběrů

Vyhledávání odběrů je postaveno na získávání informací z dostupného Google API zmíněného v podkapitole 3.3.3. Ve srovnání s ostatními řešeními nebo vlastní implementací podobné funkcionality, bylo použití tohoto API nejlepší možnou volbou i přes určité nedostatky, jako je např. nemožnost odstínění odebíraných zdrojů podle jazykových nastavení. Tento problém by bylo možné řešit vlastní implementací, ale ta nebyla v rámci rozsahu bakalářské práce realizována.

Po zadání klíčových slov nebo adresy zdroje do vyhledávacího pole uvnitř aktivity `NewFeedActivity` je dotaz zpracován ve vlákně na pozadí podtřídou `FindFeedsTask` s použitím Google API, které vrátí výsledek hledání v podobě JSON souboru. Ten je rozparsován do transportního objektu `FoundFeed`, v této podobě je předán vlastní implementaci adaptéru `FoundFeedsAdapter` a s jeho pomocí zobrazen uživateli. Během parsování dat ve vlákně na pozadí se také pomocí databázového dotazu kontroluje zda-li již není vyhledaný zdroj odebírán. Pokud ano, bude v adaptéru vizuálně odlišen a nebude možné jeho opakované přidání. Po kliknutí na vyhledaný zdroj je uživatel dotázán na kategorii, do které má být zdroj uložen, kde je jako defaultní nastavena kategorie s nezařazenými položkami. V návrhu databáze je uvedeno, že zdroje nemusí mít žádnou kategorii, ale v rámci snadnější implementace a lepší přehlednosti je hned při prvním vytvoření databáze vytvořena také kategorie pro nezařazené položky, která tak usnadňuje vypisování položek v objektu `mAdapter` na hlavní obrazovce aplikace (`MainActivity`).

5.5 Ukládání a aktualizace odběrů

Po zvolení kategorie pro uložení odběru je prostřednictvím API přivázané služby (viz 2.3.3) `UpdateFeedsService` volána metoda `LoadFeedTask`, která provádí ve vláknu na pozadí rozparsování vybraného zdroje pomocí implementace XML Pull Feed parseru. Parser zkontroluje verzi zdroje a podle ní volá odpovídající větev pro rozparsování požadovaných dat z elementů do transportního objektu `ParsedFeed`, který se posléze předá příslušné metodě správce dat a uloží se do databáze.

Aktualizace zdrojů se spouští akčním tlačítkem v horní části obrazovky a způsobí opět zavolání metody přivázané služby. V tomto případě metody `LauncherUpdateFeedsTasks`, která je spouštěcím mechanismem řetězových událostí, jejichž výsledkem je aktualizování všech odebíraných zdrojů pomocí několika propojených potomků třídy `AsyncTask`.

5.6 Načítání a aktualizace hodnocení popularity

V rámci návrhu (viz 4) bylo uvedeno, že současným RSS čtečkám často chybí nějaká kvalitativní funkce, která by mohla uživatele informovat o míře zajímavosti daného článku. Po prozkoumání různých možností se jako vhodné řešení jevilo získávat údaje o počtu sdílení, komentářů atd. daného článku na sociálních sítích. Tyto informace bylo možné načítat buď zvlášť z jednotlivých dostupných API vybraných sociálních sítí, nebo použít nějakého existujícího nástroje s jednoduchým aplikačním rozhraním, který by dělal přesně tuto činnost a výsledek nabízel jako jeden soubor např. ve formátu JSON. Právě takový nástroj představuje služba **Shared Count** (viz [4]), která např. pomocí příkazu:

```
http://api.sharedcount.com/?url=http://www.fit.vutbr.cz/
```

vrátí následující výstup v JS0N formátu:

{"StumbleUpon": 3, "Reddit": 0, "Facebook": {"commentsbox_count": 0, "click_count": 5, "total_count": 216, "comment_count": 61, "like_count": 40, "share_count": 115}, "Delicious": 0, "GooglePlusOne": 4, "Buzz": 0, "Twitter": 36, "Diggs": 0, "Pinterest": 0, "LinkedIn": 0}

Ve výsledku můžeme najít informace o sdílení na nejznámějších sociálních sítích, v případě sítě Facebook i údaj o počtu komentářů, označení položky jako oblíbené apod. Pokud vycházíme z toho, že když uživatele internetu něco zaujme, tak to chce dále sdělit svým přátelům a kontaktům např. přes sociální sít, je možné jednotlivá data ze služby Shared Count považovat za určité poměrně relevantní indikátory popularity internetové stránky. Vzhledem k tomu, že množství uživatelů sociálních sítí stále narůstá, získává tato informace čím dál více na významu. Pro uživatele by tedy mohlo být přínosné podle takového indikátoru články řadit a nebo uživatelům nabízet nejpopulárnější články v dané kategorii.

Veškerou tuto funkcionalitu zprostředkovává služba `CounterRatingService` (viz 2.3.3). Ta používá instance třídy `ExecutorUpdateRating` pro každou samostatnou úlohu načítání a aktualizace popularity prováděnou ve vlastním vláknu na pozadí (viz. 2.3.8 a 5.3). Pokud je doba od poslední aktualizace popularity všech článků větší než hodnota konstanty `TIMEOUT_FLAG_RATING_ALL`, v tuto chvíli časovač v metodě `startUpdateTimeout` posílá záměr, který zachytí přijímač `RefreshReceiver` a nastaví hodnotu flagu `hasUpdatedAllOk` na `false`. Každá aktivita po přivázání ke službě kontroluje tuto hodnotu. Pokud je `false`, volá metodu `LauncherUpdateRatingAll` přes API přivázané služby, která spouští mechanismus pro aktualizaci hodnocení popularity. Podobný mechanismus je také volán ve chvíli nově načtených článků, ukládání zdrojů, či opravy článků, u kterých došlo při načítání k chybě a mají hodnotu popularity `-1`. Zjednodušený algoritmus daného mechanismu vypadá následovně:

1. Určí se počet všech článků k aktualizaci.

2. Jsou vytvořeny odpovídající instance tříd `ExecutorUpdateRating` a spuštěno provádění aktualizace metodou `executeTask`, která pro každý článek iterativně vytváří novou úlohu k jeho ohodnocení a hlídá počet těchto úloh.

3. Uvnitř aktualizačních úloh je pomocí volání metody `CountRating` zjištěna cílová adresa pokud existuje přesměrování. Dále jsou regulárními výrazy odstraněny případně nalezené UTM[4] tagy, používané pro monitorování chování návštěvníků stránek, které by způsobily chybné ohodnocení.

4. Takto upravená adresa článku je vložena do dotazu volajícího API služby Shared Count.

5. Výsledný soubor JSON je rozparsován a jednotlivé hodnoty sečteny, protože uvedené počty výskytu článků na sociálních sítích mají stejnou váhu a nás zajímá především celková míra popularity. Pokud dojde při parsování k chybě, je cyklus opakován tolikrát, kolik uvádí konstanta `MAX_RETRY_JSONPARSING`.

6. Součet hodnot je uložen do databáze, a pokud byla daná aktualizační úloha poslední v pořadí, je pomocí vyslaného záměru aktuálně spuštěná aktivita informována o dokončení prováděné operace a jsou nastaveny odpovídající flagy v objektu `Application`.

[4]UTM – Urchin Tracking Module (více viz [17])

V průběhu může také dojít k různým chybám způsobeným ztrátou spojení, nevalidní adresou, odezvou služby Shared Count atd. V těchto případech je vyslán záměr informující o chybě a aktivity, jejichž součástí jsou implementace přijímačů broadcastů, tento záměr zachytí, provedou potřebná opatření a informují uživatele o chybě.

Při implementaci byl objeven současně i problém s rychlostí aktualizace popularity velkého množství položek způsobený delší odezvou služby Shared Count. Ta je zapříčiněna pomalým vyhodnocením dotazu samotnými API sociálních sítí, které Shared Count používá. Dané tvrzení bylo ověřeno vlastní implementací metody, která zpracovávala jednotlivé údaje přímo z API sledovaných sociálních sítí. Problém byl v rámci aplikace částečně vyřešen načítáním hodnocení pouze u nepřečtených článků a omezením maximálního počtu článku, pro které se operace provede, na hodnotu konstanty `MAX_MESSAGES_FOR_COUNT_RATING`. Další možná budoucí řešení jsou uvedena v závěrečné kapitole 6.

5.7 Uživatelské rozhraní

Uživatelské rozhraní je implementováno s pomocí všech dostupných prostředků, které Android nabízí, např: views (viz 2.3.1), různé kontejnery a nabídky menu definované ve vlastních XML souborech. Uživatelské rozhraní zahrnuje také adaptéry pro zobrazovaní informací z databáze a styly definované v souboru `styles.xml`. Všechny textové řetězce, které jsou součástí uživatelského rozhraní jsou uloženy odděleně v souboru `strings.xml`, aby bylo možné do budoucna připravit další jazykové mutace.

Pro uživatelské rozhraní bylo důležité zajistit zpětnou kompatibilitu widgetu (viz 2.3.1) `ActionBar`, který je podporovaný až od jedenácté verze API. Pro tento účel byl jako nejlepší kandidát zvolen `SherlockActionBar` Jamese Whartona. `SherlockActionBar` je rozšíření oficiální knihovny pro zpětnou kompatibilitu tohoto widgetu, které umožňuje podstatně snadnější implementaci a nastavení než standardní řešení. `SherlockActionBar` má podobu samostatného Android projektu, který je nastavený jako knihovna přivázaná k aplikaci RSS čtečky (více viz [29]).

5.8 Testování, kompatibilita a omezení

V průběhu celého procesu implementace byly prototypy aplikace opakovaně podrobovány testování prostřednictvím mobilního emulátoru (viz 2.2.1) a na reálném zařízení Samsung Galaxy ACE a HTC 3D. Byly testovány platformy Android verze 2.2, 2.3, 3.0, 4.0. V průběhu testování bylo objeveno několik problémů vycházejících z odlišností jednotlivých verzí SDK. Objevené problémy byly následně odstraněny a aplikace by měla být na těchto platformách funkční. Minimální požadovaná verze pro spuštění aplikace je tedy SDK 8, kromě toho vyžaduje aplikace pouze oprávnění pro přístup k internetu.

Při zhodnocení celkové složitosti řešení bylo upuštěno od některých původně plánovaných funkcí, jako je podpora více formátů než pouze RSS 2.0 ačkoliv parser umí jednotlivé verze identifikovat dále už neimplementuje jejich parsování. Chybí také propojení s akcí související s otevřením souboru RSS v internetovém prohlížeči a možnost zobrazení obrázků u odebíraných článků. Dalším již výše uvedeným omezením je limitování počtu zpráv, pro které se načítá informace o popularitě na sociálních sítích kvůli délce trvání této akce. Možnosti budoucího vylepšení současného stavu aplikace a její podrobnější zhodnocení budou součástí následující kapitoly.

Kapitola 6

Závěr

V průběhu práce se objevovalo množství komplikací, které bylo nutné řešit. Mezi zásadní problematiku patřilo především správné pochopení životního cyklu aplikačních komponent systému Android uvedených v podkapitole 2.3. Velkou část práce zahrnovala mimo jiné implementace vícevláknového zpracování (viz 2.3.8) početných déletrvajících operací, implementace databáze SQlite (viz 5.2) a návrh s implementací uživatelského rozhraní včetně dialogů a upozornění (viz 4.4 a 5.7). Důležité také bylo naučit se používat dostupné prostředky pro vývoj aplikací pro Android z podkapitoly 2.2.

K vypracování bakalářské práce bylo potřeba nastudovat objemné množství informací uvedených v kapitolách 2 a 3, ale současně se zabývat i průzkumem existujících řešení (viz 4.1). Vlastní původní záměr, připravit v rámci bakalářské práce hotový produkt určený k distribuci, se na základě zhodnocení předchozích dvou kroků změnil, protože vytvořit plně konkurenceschopnou komplexní aplikaci by bylo nad rámec rozsahu této práce. Proto se další postup ubíral cestou k vytvoření pokročilé vývojové platformy RSS čtečky, která implementuje funkcionalitu navrženou v kapitole 4, vyjímaje omezení uvedené v podkapitole 5.8.

Poměrně zásadním omezením aplikace je limitování počtu zpráv, pro které se načítá informace o popularitě na sociálních sítích, kvůli délce trvání této akce. Na což se přišlo během implementace funkcionality (viz 5.6). Tento problém by bylo možné do budoucna minimalizovat například ukládáním uživatelských dat na server, který by v plánovaných intervalech zpracovával, načítal a shromažďoval požadované informace tak, aby byly uživatelům aplikace okamžitě k dispozici.

Ačkoliv se nepodařilo naplnit vlastní vizi výsledné aplikace, byly jednotlivé body bakalářské práce splněny. Implementovaná RSS čtečka nabízí zajímavé funkce, jako je například již uvedené načítání popularity odebíraných článků na sociálních sítích, uživatelské filtrování, hromadné operace s odebíranými články, vyhledávání odběrů podle klíčových slov a několik variant zobrazení výpisu odebíraných článků. Postupem času se počítá s rozšířením aplikace o další užitečné funkce, odstraněním zjištěných nedostatků a uvedením finálního produktu k distribuci.

Mezi plánovaná budoucí rozšíření můžeme zařadit podporu více formátů pro odebírání obsahu, částečnou inteligenci v podobě indexování a automatické kategorizace odebíraných zdrojů, sledování návyků a historie chování uživatele a na tomto základě nabízení relevantního obsahu. Dále také plná podpora sociálních sítí, nebo i lepší integrace do mobilního prohlížeče a synchronizace s online RSS čtečkou Google Reader.

Literatura

[1] Axure Software Solutions, I.: Design Interactive Wireframes, Prototypes, and Specifications with Axure RP [online]. 2012 [cit. 2012-05-07].
URL http://www.axure.com

[2] Ayers, D.; Watt, A.: *Beginning RSS and Atom Programming*. Wiley Publishing, Inc., 2005, ISBN 978-0-7645-7916-5.

[3] Baquilta, J.: Best RSS Reader Apps for Android [online]. 2011-12-31 [cit. 2012-05-07].
URL
http://www.androidauthority.com/best-rss-reader-apps-for-android-40454

[4] Carmon, Y.: Shared Count. [cit. 2012-05-15].
URL http://sharedcount.com

[5] Collins, C.; Galpin, M. D.; Käppler, M.: *Android in Practice*.
Manning Publications Co., 2012, ISBN 978-19-351-8292-4.

[6] Felker, D.; Dobbs, J.: *Android Application Development for Dummies*.
Wiley Publishing, Inc, 2011, ISBN 978-0-470-77018-4.

[7] Google Inc.: Activities [online]. 2012 [cit. 2012-02-05].
URL
http://developer.android.com/guide/topics/fundamentals/activities.html

[8] Google Inc.: Application Fundamentals [online]. 2012 [cit. 2012-02-05].
URL http://developer.android.com/guide/basics/what-is-android.html

[9] Google Inc.: Android Developers [online]. 2012 [cit. 2012-03-25].
URL http://developer.android.com

[10] Google Inc.: Installing the SDK [online]. 2012 [cit. 2012-04-25].
URL http://developer.android.com/sdk/installing.html

[11] Google Inc.: Android Brand Assets, Icons, and Guidelines [online]. 2012 [cit. 2012-05-04].
URL http://www.android.com/developers/branding.html

[12] Google Inc.: Application [online]. 2012 [cit. 2012-05-05].
URL
http://developer.android.com/reference/android/app/Application.html

[13] Google Inc.: Service [online]. 2012 [cit. 2012-05-05].
URL http://developer.android.com/reference/android/app/Service.html

[14] Google Inc.: Google Play: Noviny a časopisy [online]. 2012 [cit. 2012-05-07].
URL https://play.google.com/store/apps/category/NEWS_AND_MAGAZINES?feature=category-nav

[15] Gregorio, J.; de hÓra, B.: The Atom Publishing Protocol [online]. 2007 [cit. 2012-05-05].
URL http://www.ietf.org/rfc/rfc5023.txt

[16] Holzner, S.; Šindelář, J.: *RSS: automatické doručování obsahu vašich WWW stránek*. Computer Press, a.s., 2007, ISBN 978-80-251-1479-7.

[17] Inc., G.: Urchin Tracking Module (UTM). [cit. 2012-05-16].
URL
http://support.google.com/urchin45/bin/answer.py?hl=en&answer=28307

[18] JetBrains s.r.o.: IntelliJ IDEA - Free IDE for Google Android [online]. 2012 [cit. 2012-04-25].
URL http://http://www.jetbrains.com/idea/features/google_android.html

[19] Kameka, A.: The Best RSS Android app with Google Reader is... [Droid vs Droid] [online]. 2011-08-25 [cit. 2012-05-07].
URL http://androinica.com/2010/08/the-best-rss-android-app-with-google-reader-is-droid-vs-droid

[20] Kannan, S.: Top 10 Minimal and Elegant Android News Reader Apps [online]. 2011-06-04 [cit. 2012-05-07].
URL http://android.appstorm.net/roundups/lifestyle-roundups/top-10-minimal-and-elegant-android-news-reader-apps

[21] Kilián, K.: SvětAndroida doporučuje: RSS čtečky a agregátory článků [online]. 2011-06-09 [cit. 2012-05-07].
URL http://www.svetandroida.cz/svetandroda-doporucuje-rss-ctecky-a-agregatory-clanku-201106

[22] Komatineni, S.; MacLean, D.; Hashimi, S. Y.: *Pro Android 3*. Apress, 2011, ISBN 978-1-4302-3222-3.

[23] Kypta, T.: Vyvíjíme pro Android – úvod [online]. 2011-3-16 [cit. 2012-05-03].
URL http://www.abclinuxu.cz/clanky/vyvijime-pro-android-uvod

[24] Meier, R.: *Professional Android Application Development*. Wiley Publishing, Inc., 2009, ISBN 978-0-470-34471-2.

[25] Moor, C.: Top 5 Android RSS readers [online]. 2011-08-03 [cit. 2012-05-07].
URL http://www.talkandroid.com/8805-top-5-android-rss-readers

[26] Murphy, M. L.: *Android 2: průvodce programováním mobilních aplikací*. Computer Press, a.s., 2011, ISBN 978-80-251-3194-7.

[27] Nottingham, M.; Sayre, R.: The Atom Syndication Format [online]. 2005 [cit. 2012-05-05].
URL http://www.ietf.org/rfc/rfc4287.txt

[28] Software, U.: RSS 2.0 Specification [online]. 2002-08-19 [cit. 2012-04-04].
URL http://www.rssboard.org/rss-2-0

[29] Wharton, J.: ActionBarSherlock [online]. ©2012 [cit. 2012-04-14].
URL http://actionbarsherlock.com