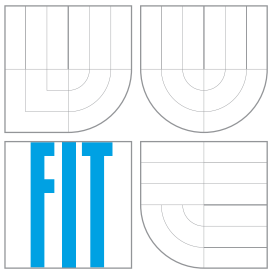
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SLOVNÍK PRO MOBILNÍ ZAŘÍZENÍ SE ZAMĚŘENÍM NA ANDROID

DICTIONARY FOR MOBILE DEVICES (IPAD, IPHONE, ANDROID)

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETER BOBA

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. RNDr. PAVEL SMRŽ, Ph.D.

BRNO 2013

Abstrakt

Tato práce se zabývá návrhem a implementací systému pro vytváření slovníkových databází, stejně tak návrhem a implementací aplikace pro systém Android, která zobrazuje slovníková data obsažená v těchto databázích. Zajímavou funkcionalitou aplikace je možnost pokročilého vyhledávání frází a vět. Slovníkové databáze jsou generovány ze zdrojových lexikálních dat, která jsou strukturována pomocí Lexical Markup Framework.

Abstract

This thesis deals with the design and implementation of a system for creating dictionary databases, as well as design and implementation of an Android application that displays the dictionary data contained in these databases. An interesting functionality of the application is the possibility of advanced searching for phrases and sentences. Dictionary databases are generated from a lexical data sources that are structured using Lexical Markup Framework.

Klíčová slova

Android, mobilní zařízení, databáze, vyhledávání, Lexical Markup Framework, slovník

Keywords

Android, mobile devices, database, search, Lexical Markup Framework, dictionary

Citace

Peter Boba: Slovník pro mobilní zařízení se zaměřením na Android, bakalářská práce, Brno, FIT VUT v Brně, 2013

Slovník pro mobilní zařízení se zaměřením na Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. RNDr. Pavla Smrže, Ph.D.

.....
Peter Boba
14. mája 2013

Poděkování

Chtěl bych poděkovat Doc. RNDr. Pavlu Smržovi, Ph.D. a Ing. Janu Kouřilovi za poskytnutí odborné pomoci při vypracovávaní bakalářské práce.

Obsah

Kapitola 1

Úvod

Slovníky sú v súčastnosti jeden z najužitočnejších zdrojov lexikálnych dát. **Machine-readable dictionary** (MRD) je elektronická forma slovníkov. Dáta obsiahnuté v tomto type slovníkov majú elektronickú podobu a môžu byť spracované alebo manipulované použitím počítača [6]. MRD sú masívne využívané aj v oblasti spracovania prirodzeného jazyka (NLP) [9].

S nástupom mobilných zariadení sa naskytá možnosť vytvárať aplikácie, ktoré dokážu prezentovať obsah týchto elektronických zdrojov slovníkových dát. Užívatelia, ktorí používajú tlačené, alebo elektronické formy slovníkov získavajú ďalšiu možnosť pre ich zobrazovanie.

Táto práca sa zaoberá vytvorením systému, ktorý umožňuje transformáciu zdrojových lexikálnych dát na slovníkovú databázu a následné zobrazenie týchto lexikálnych dát. Mobilné zariadenia sa líšia nielen architektúrou, ale aj použitým operačným systémom. V tejto práci sa zameriavam na vývoj mobilnej aplikácie pre systém **Android**. Dôraz bol pri vytváraní aplikácie kladený hlavne na umožnenie fulltextového vyhľadávania a rozšírenú podporu práce s frázami a vetami.

V kapitole 2 popisujem formát zdrojových lexikálnych dát. Nasleduje stručný prehľad existujúcich slovníkových aplikácií. V kapitole 4 sú uvedené požiadavky na výslednú aplikáciu. Zároveň v tejto kapitole popisujem proces návrhu jednotlivých súčastí vytváraného systému.

Popis implementácie vytváraného systému je obsahom kapitoly 5. V tejto sekcii sa zameriavam predovšetkým na technické prevedenie aplikačných súčastí a použitie technológií v praxi. V podkapitole 5.8 je uvedené porovnanie vytvorenej aplikácie **LMFDictionary** s podobnými existujúcimi riešeniami a popis testovania vytváranej aplikácie.

Prácu uzatvára kapitola 6 v ktorej sumarizujem celý proces riešenia a navrhujem ďalšie možnosti vylepšenia, alebo rozšírenia vytvorenej aplikácie.

Kapitola 2

Lexical markup framework

Cieľom mojej práce bolo implementovať slovníkovú aplikáciu pre systém Android. Slovníkové aplikácie sú založené na existujúcich lexikálnych dátach. Pre účely tejto práce som obdržal slovníkové dáta vo formáte Lexical markup framework (ďalej LMF). Popis tejto štruktúry môžeme nájsť v dokumente Language resource management – Lexical markup framework (LMF) [10]. Ďalší popis čerpá z tejto publikácie.

LMF je abstraktný metamodel, poskytujúci jednotnú štruktúru slúžiacu na konštrukciu výpočetných lexikónov. Na jeho vytvorení sa podieľali najmä Gil Francopoulo, Monte George a Nicoletta Calzolari. Poskytuje jednotné rozhranie pre použitie v rôznych aplikáciách a pre rôzne účely. LMF zahŕňa morfologické, syntaktické a semantické aspekty a najvyšším cieľom tohto modelu je vytvoriť modulárnu štruktúru, ktorá by umožňovala interoperabilitu všetkých aspektov elektronických lexikálnych zdrojov.

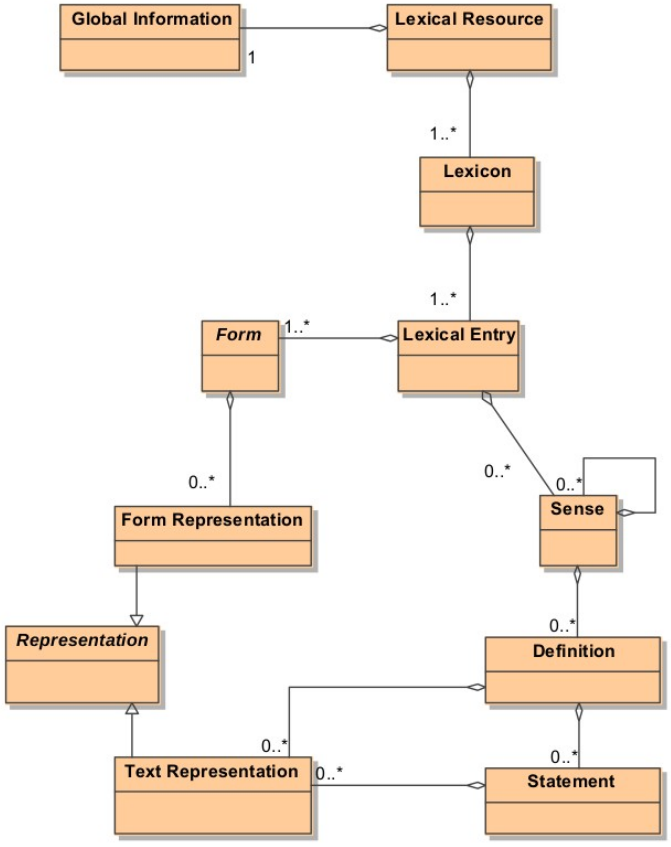
Model LMF je zostavený v súlade s Unified Modeling Language (UML). Používa jeho podmnožinu, ktorá je vhodná pre reprezentáciu lingvistických dát. Model je zložený z dvoch častí:

- LMF Core Package
- LMF Extensions

Pri vytváraní LMF lexikónov definujeme bázu v podobe LMF Core Package. Následne vyberieme voliteľné položky, medzi ktoré patria LMF Extensions a užívateľom definované dátové položky.

2.1 Lexical markup framework Core Package

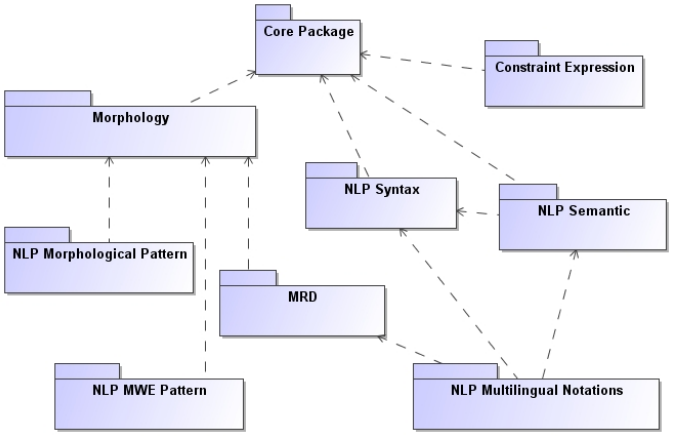
LMF Core Package tvorí základ pre tvorbu LMF modelov a rozšírení. Obsahuje základné triedy reprezentujúce jednotlivé súčasti slovníkových zdrojov. Jeho UML model zobrazuje obrázok 2.1.



Obr. 2.1: LMF core package (prevzaté z [10])

2.2 Lexical markup framework Extensions

LMF Extensions dopĺňajú základný LMF Core package. Ich výhodou je možnosť neustáleho rozširovania a prispôsobovania. Tieto rozšírenia zahŕňajú napríklad balíky pre podporu morfológie, syntaxe a sémantiky. Diagram niektorých rozšírení a ich väzby na LMF Core Package je zobrazený na obrázku 2.2.



Obr. 2.2: LMF extensions (prevzaté z [10])

2.3 Zdrojové slovníkové dáta a popis ich súčastí

Výpis 2.1: Ukážka časti vstupného XML súboru

```
<LexicalResource dtdVersion="16">
  <GlobalInformation>
    <feat att="languageCoding" val="ISO 639-3" />
  </GlobalInformation>
  <Lexicon>
    <feat att="language" val="eng" />
    <LexicalEntry id="e106567g">
      <feat att="partOfSpeech" val="noun" />
      <Lemma>
        <feat att="writtenForm" val="root" />
      </Lemma>
      <WordForm>
        <feat att="phoneticForm" val="ru:t" />
        <feat att="writtenForm" val="root" />
      </WordForm>
      <Sense>
        <feat att="senseNumber" val="1" />
        <Equivalent>
          <feat att="language" val="ces" />
          <feat att="gloss" val="rostliny" />
          <feat att="writtenForm" val="ko&#345;en" />
        </Equivalent>
        <Context>
          <TextRepresentation>
            <feat att="languageIdentifier" val="eng" />
            <feat att="text" val="Some trees have ..." />
          </TextRepresentation>
          <TextRepresentation>
            <feat att="languageIdentifier" val="ces" />
            <feat att="text" val="N&#283;kter&#233; ..." />
          </TextRepresentation>
        </Context>
        <SubjectField>
          <feat att="label" val="bot." />
        </SubjectField>
      </Sense>
    </LexicalEntry>
  </Lexicon>
</LexicalResource>
```

Výpis 2.1 zobrazuje ukážku časti zdrojových slovníkových dát, na ktorých je založená výsledná aplikácia. Výpis obsahuje reprezentáciu anglického termínu *root* pomocou LMF štruktúry. Popis základných prvkov uvádzam v nasledujúcom texte:

LexicalResource
: Táto trieda môže zahŕňať viacero lexikónov. Reprezentuje celý zdroj a môže sa vyskynúť len raz.

GlobalInformation
: Obsahuje informácie a atribúty, ktoré charakterizujú celý zdroj. Musí obsahovať jeden povinný atribút a to je jazykové kódovanie dokumentu (*languageCoding*).

Lexicon
: *Lexicon* je trieda, ktorá obsahuje lexikálne záznamy. Musí obsahovať aspoň jeden záznam.

LexicalEntry
Reprezentuje *lexém*[1] v danom jazyku. Môže mať 0, alebo viac významov.

Lemma
Je podtrieda triedy *Form* a je používaná na určenie formy slova. Zvyčajne je odvodená zo slovotvorných prvkov (*root* a *stem*). Ako uvádza Payne[2], rozdiel medzi týmito termínami je veľmi jemný.

WordForm
Určuje tvar, ktorý môže *lexém* nadobúdať pri použití vo vete.

Sense
Je to trieda určujúca jeden z možných významov lexikálneho záznamu.

Equivalent
Týka sa hlavne viacjazyčných slovníkov. Predstavuje prekladový ekvivalent lexikálneho záznamu, ktorý vychádza z triedy *Lemma*.

Context
Predstavuje použitie formy slova v kontexte vety.

TextRepresentation
Je spojená s potomkami triedy *Sense*. Vyjadruje špecifický pravopis vetného spojenia a môže obsahovať ďalšie atribúty týkajúce sa tohto spojenia.

SubjectField
Určuje predovšetkým doménu použitia príslušného termínu.

[1] *lexém* je abstraktná jednotka, združujúca formy, ktoré zdieľajú spoločný význam [10].
[2] PAYNE, Thomas Edward. *Exploring language structure: a student's guide*. New York: Cambridge University Press, 2006, xxii, 367 p. ISBN 978-052-1671-507.

Kapitola 3

Prehľad existujúcich slovníkových aplikácií

V súčasnosti je na trhu veľké množstvo slovníkových aplikácií určených pre platformu Android. V nasledujúcom texte uvádzam prehľad niektorých z nich.

Aplikácie využívajú svoj vlastný slovníkový formát, alebo umožňujú načítanie už existujúcich formátov. Prehľad niektorých formátov uvádzam v podkapitole 3.1.

Všetky tieto aplikácie sú k dispozícií zdarma v službe Google Play [1]. Niektoré z nich sú k dispozícií aj v rozšírenej platenej verzii.

3.1 Slovníkové formáty

3.1.1 WordNet

WordNet je rozsiahla databáza angličtiny. Slová v tejto databáze sú organizované do skupín s podobným významom. Slová, ktoré majú medzi sebou synonymické vzťahy sú združované do setu (tzv. *synset*).

Každý slovný druh je v databáze organizovaný špecifickým spôsobom. Napríklad prídavné mená sú organizované ako páry antonym, zatiaľ čo podstatné mená môžu byť charakterizované ako „časť-celok“ (chair, furniture). Väčšina vzťahov spája slová, ktoré majú spoločný slovný druh. Čerpané z textu „*What is WordNet?*“[1].

3.1.2 StarDict

Slovníkový formát StarDict používa na vyjadrenie slovníkových dát viacero súborov. Niektoré z týchto súborov je možné komprimovať, aby sa zmenšila ich veľkosť.

Súbor .ifo
: Obsahuje základné informácie o slovníku ako názov, počet obsiahnutých slov.

Súbor .idx
: V tomto súbore sú obsiahnuté slová, podľa ktorých sa v slovníku vyhľadáva. Je treba špecifikovať posun dát (offset) a celkovú veľkosť slova.

[1] What is WordNet?. *About WordNet* [online]. © 2013 [cit. 2013-04-20]. Dostupné z: http://wordnet.princeton.edu/

Súbor .syn
: Súbor s koncovkou `.syn` je voliteľný a obsahuje informácie o synonymách slov. Každá položka tohto súboru obsahuje synonymum a identifikáciu pôvodného slova v `.syn` súbore.

Súbor .oft
: Súbor potrebný pre použitie medzipamäte (cache). Šetrí pamäť a urýchľuje načítavanie.

Súbor .clt
: Vzniká po zoradení slov funkciou *collate*. Obsahuje informácie o radení slov. Podpora od verzie StarDict-2.4.8.

Súbor .dict
: Obsahuje samotné slovníkové dáta. Každej položke predchádza identifikátor, ktorý udáva typ tejto položky. Môže ísť napríklad o význam slova, fonetickú reprezentáciu, *wav* súbor a iné.

Čerpané z textu *„StarDict_format“*[1].

3.1.3 DSL

Formát DSL využívajú napríklad slovníky ABBYY Lingvo. V úvode DSL slovníka je treba špecifikovať zdrojový a cieľový jazyk prekladu. Zázam (tzv. karta) v tomto slovníku sa skladá z hlavičky a tela. Hlavička obsahuje samotné slovo alebo frázu a telo obsahuje jednotlivé preklady a komentáre. Pred použitím slovníka vo formáte DSL v software ABBYY Lingvo, je potrebné previesť kompiláciu tohto slovníka. Čerpané z popisu *„DSL Dictionary Structure“*[2].

3.2 Slovníkové aplikácie

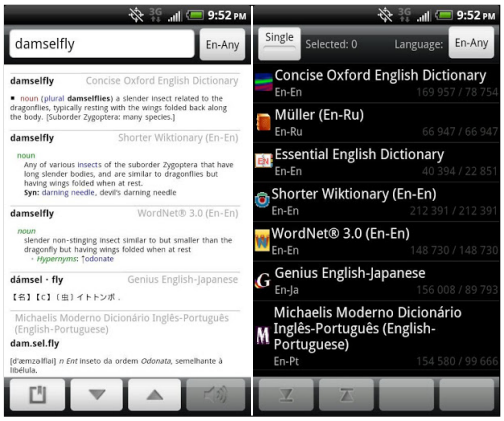
3.2.1 GoldenDict Free

Aplikácia GoldenDict Free umožňuje vkladanie slovníkov v rôznych formátoch. V súčasnosti podporuje formáty:

- Lingoes (.LD2)
- Babylon (.BGL)
- ABBYY Lingvo (.LSD, .DSL, .LSA, .DAT)
- StarDict
- Dictd
- Hunspell (.AFF, .DIC)

[1] StarDict_format. *Babiloo* [online]. 2007 [cit. 2013-04-20]. Dostupné z: http://code.google.com/p/babiloo/wiki/StarDict_format

[2] DSL Dictionary Structure. *Information Worker Solutions* [online]. [2012] [cit. 2013-04-20]. Dostupné z: http://informationworker.ru/lingvo.en/dsl_main_dlg.htm



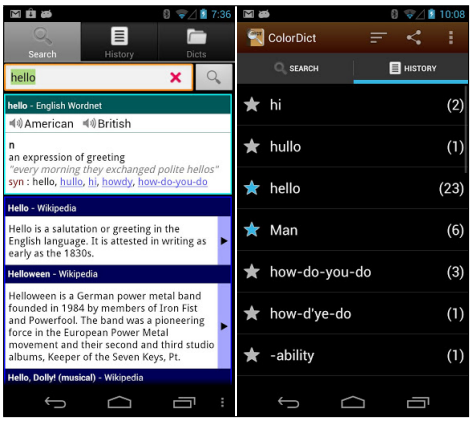
Obr. 3.1: GoldenDict Free

Medzi hlavné prednosti tejto aplikácie patrí možnosť vyhľadávania naraz vo viacerých slovníkoch. Slovník reaguje na vstup návrhmi vyhľadávania. Verzia zdarma umožňuje načítať 5 rôznych slovníkov.

3.2.2 ColorDict Dictionary Wikipedia

Táto slovníková aplikácia podporuje užívateľské slovníky vo formátoch WordNet a StarDict. Umožňuje vyhľadávať súčasne vo viacerých slovníkoch a aj v encyklopédií Wikipedia. Medzi ďalšie hlavné prednosti ColorDict patria:

- Hlasové vyhľadávanie
- Návrhy vyhľadávania
- Záznam histórie vyhľadávania
- Text-to-speech

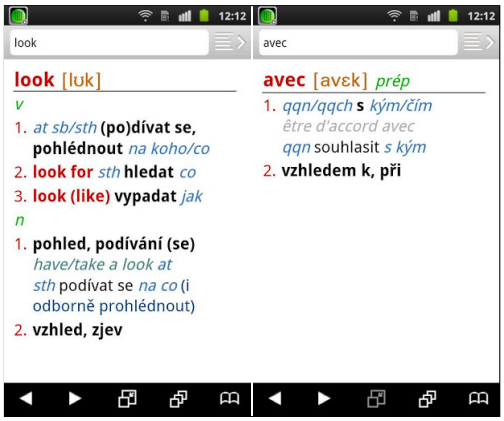


Obr. 3.2: ColorDict Dictionary Wikipedia

3.2.3 HandyLex 4 Czech

Aplikácia HandyLex 4 Czech používa vlastné slovníkové databázy (neumožňuje vkladať nové slovníky). Prekladá obojsmerne medzi základnou češtinou a jedným z jazykov: angličtina, francúzština, nemčina a španielčina.

Podporuje návrhy a históriu vyhľadávania. Podporuje automatické rozpoznanie jazyka a vyhľadanie podľa rôznych morfologických foriem.



Obr. 3.3: HandyLex 4 Czech

Informácie boli získané z popisov aplikácií, ktoré sú uvedené v službe *Google Play*[1].

[1]GOOGLE. *Google Play* [online]. © 2013 [cit. 2013-20-04]. Dostupné z: https://play.google.com/store

Kapitola 4

Návrh aplikácie

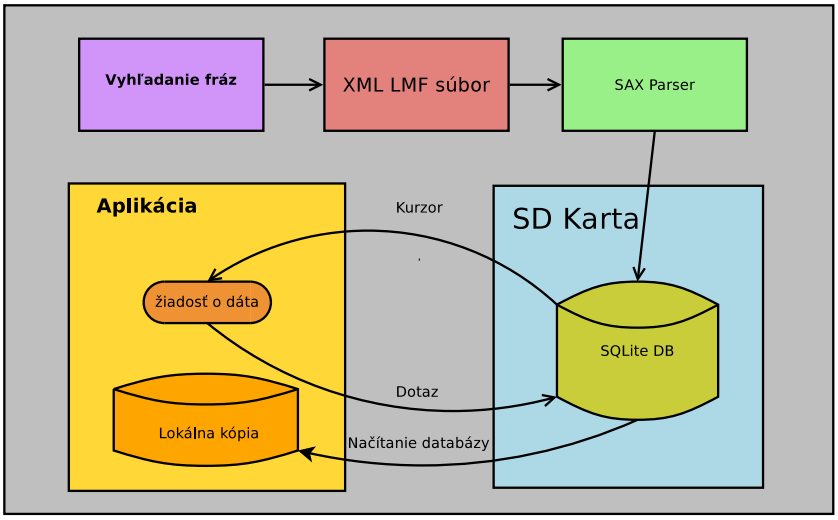
4.1 Požiadavky na aplikáciu

Výsledná slovníková aplikácia má spĺňať tieto požiadavky:

- Aplikácia je založená na zdrojových dátach vo formáte LMF.
- Je zobrazovaná nápoveda počas vyhľadávania (tzv. search suggestions).
- Aplikácia podporuje vyhľadávanie jednotlivých slov, ale aj celých viet.
- Je implementovaný mechanizmus, ktorý umožňuje znovupoužitie výsledkov vyhľadávania ako vstup nového vyhľadávania.
- Aplikácia podporuje vyhľadávanie fráz a viet.

4.2 Funkčné bloky aplikácie

Na základe požiadaviek som navrhol štruktúru aplikácie, ktorá je uvedená na obr. 4.1.



Obr. 4.1: Funkčné bloky aplikácie

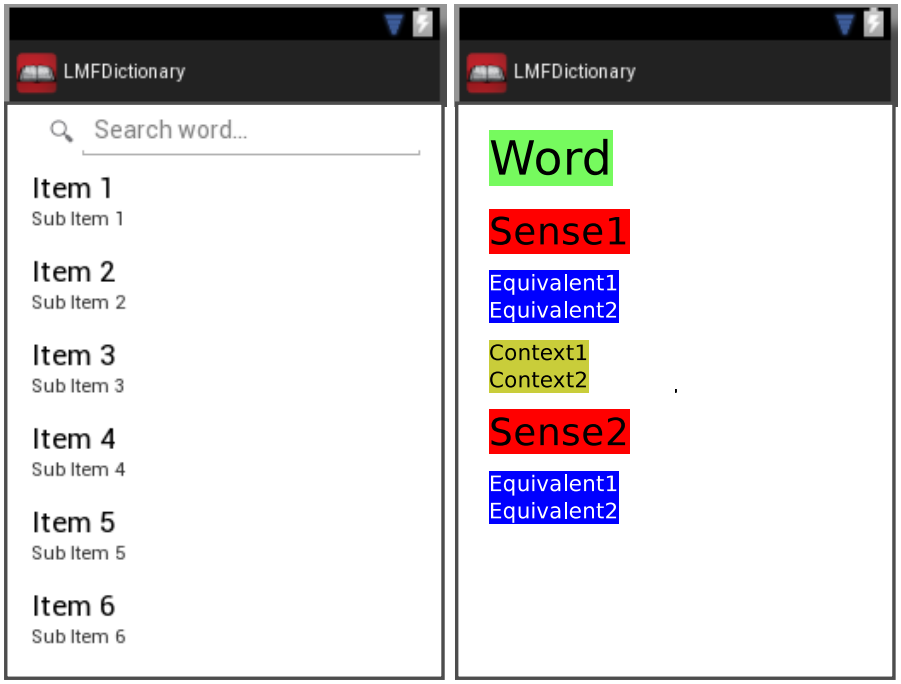
Obr. 4.1 opisuje funkčné bloky aplikácie. Prvou operáciou, ktorá prebieha nad vstupnými dátami je vyhľadávanie fráz. Túto činnosť vykonáva skript vytvorený v jazyku *Python*. Tento skript je opísaný v sekcii 5.1.

Nasleduje syntaktická analýza vstupného súboru (*parsing*) a vytváranie databázy. Rozhodol som sa použiť databázu *SQLite* [4] a rozšírenie FTS [5], ktoré popisujem v sekcii 5.5. Na tento účel som vytvoril program *LMFDictToDB*. Tento program je napísaný v programovacom jazyku *Java* a jeho činnosť popisujem v sekcii 5.2.

Výstupný databázový súbor, ktorý vytvoril program *LMFDictToDB* sa uloží na SD kartu zariadenia s operačným systémom Android. Pri prvom spustení je táto databáza načítaná a vzniká lokálna kópia v pamäťovej oblasti aplikácie, konkrétne v priečinku `databases`.

Aplikácia podľa potreby pristupuje k tejto databáze a formuluje svoje požiadavky na dáta (dotazy). Dáta, ktoré spĺňajú špecifikácie požiadavky sú navrátené vo forme databázového kurzora.

4.3 Návrh grafického rozhrania



Obr. 4.2: Koncept grafického rozhrania aplikácie

Na obr. 4.2 je zobrazený koncept grafického rozhrania aplikácie. V ľavej časti je obrazovka zariadenia s vyhľadávacím poľom a prvkom *ListView*. ListView je prvok, ktorý umožňuje zobrazovanie a vertikálny posun dátových položiek [3]. Toto zobrazenie umožňuje vyhľadávanie slovníkových dát a priame zobrazovanie výsledkov vyhľadávania v ListView.

V pravej časti obr. 4.2 je načrtnutý koncept zobrazovania výsledkov vyhľadávania. Na túto obrazovku sa užívateľ dostane po výbere konkrétnej položky zo zoznamu vyhľadaných slov. Funkčnosť tohto zobrazenia bude zabezpečovať prvok *ScrollView* [3] a viacero položiek prvku *TextView* [3].

4.4 Návrh databázy

Aplikácia musí pracovať s veľkým množstvom informácií. Uchovanie týchto informácií vo vstupnom XML súbore a priame vyhľadávanie v tomto súbore nie je vhodné riešenie. Neposkytuje totiž vhodnú funkcionalitu pre prístup k dátam.

Rozhodol som sa transformovať tieto vstupné dáta do databázy. Zvolil som reprezentáciu databázou *SQLite* [4]. Medzi dôvody použitia práve tohto riešenia patria:

- Má vlastnosti ACID.
- Nevyžaduje konfiguráciu (úvodné nastavenie a administráciu).
- Základom je štandard ANSI SQL92.
- Celá databáza je uložená v jednom súbore.
- Databáza SQLite je priamo podporovaná operačným systémom Android [2].

Databázu SQLite som zvolil aj z dôvodu podpory fulltextového vyhľadávania. Vo výpise 4.1 je uvedené porovnanie rýchlosti vyhľadávania v tabuľke používajúcej rozšírenie FTS3 a v klasickej tabuľke. Porovnávané sú tabuľky s rovnakým obsahom. Porovnanie prebehlo pomocou príkazov `LIKE` (bežná tabuľka) a `MATCH` (FTS3 tabuľka).

Výpis 4.1: Porovnanie rýchlosti tabuľky používajúcej rozšírenie FTS3 a bežnej tabuľky

```
1 CREATE VIRTUAL TABLE data1 USING fts3(content TEXT); /*FTS3 tabuľka*/
2 CREATE TABLE data2(content TEXT);                    /*Bežná tabuľka*/
3 SELECT count(*) FROM data1 WHERE content MATCH 'linux';   /* 0.03 s*/
4 SELECT count(*) FROM data2 WHERE content LIKE '%linux%';  /* 22.5 s*/
```

Čerpané z [4].

4.4.1 Štruktúra databázy

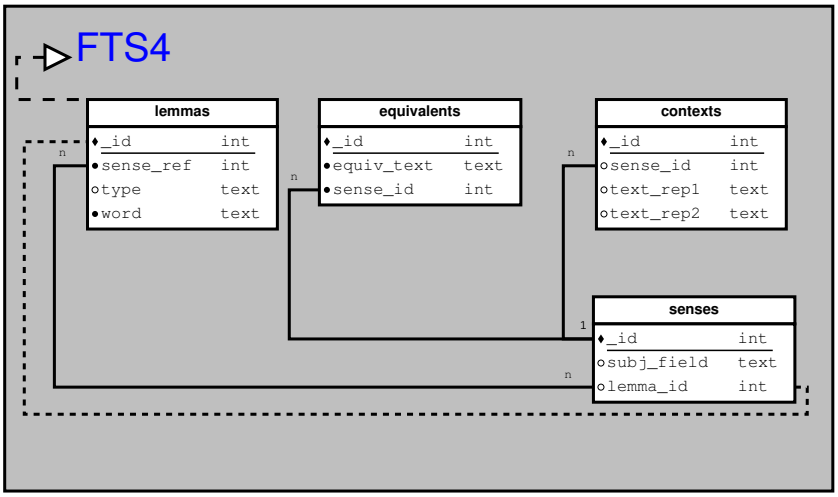
Databázu tvoria 4 základné tabuľky – sú to tabuľky `lemmas`, `senses`, `contexts` a `equvivalents` (pozri obr. 4.3).

Tabuľka `lemmas`
: Obsahuje polia, ktoré sú nevyhnutné pre presnú identifikáciu vyhľadávaného termínu/frázy. Táto tabuľka používa rozšírenie FTS4, ktoré popisujem v podkapitole 5.5.

Tabuľka `senses`
: Slúži ako spojovací článok medzi vyhľadávanou položkou a jej cudzojazyčnými ekvivalentami, prípadne použitím v kontexte. Obsahuje pole `subj_field`, ktoré vyjadruje oblasť použitia daného významu.

Tabuľka `equivalents`
: Polia tejto tabuľky charakterizujú cudzojazyčné ekvivalenty vyhľadávaných slov. Pole `sense_id` vyjadruje, ku ktorému významu daný ekvivalent patrí.

Tabuľka `contexts`
: Obsahuje polia, ktoré definujú použitie vyhľadávanej položky v kontexte vety. Polia `text_rep1` a `text_rep2` vyjadrujú použitie slova v kontexte vety dvoch rôznych jazykov. Podľa obsahu týchto položiek sa dá aj vyhľadávať, pretože tento obsah sa ukladá do tabuľky `lemmas`.



Obr. 4.3: ER diagram navrhovanej databázy

Prvotný návrh databázy nezahŕňal pole `sense_ref`. Pôvodná väzba je vyjadrená prerušovanou čiarou na obr. 4.3. Väzba primárneho kľúča `_id` tabuľky `lemmas` a cudzieho kľúča `lemma_id` tabuľky `senses` nepodporovala výskyt viacerých termínov/fráz ktoré by zdieľali spoločný význam.

Nový prístup (väzba *n* ku *n* položiek `sense_ref` a `lemma_id`) zabezpečuje, že v databáze sa môžu vyskynúť viaceré položky tabuľky `lemmas`, ktoré obsahujú referenciu na jeden spoločný význam v podobe položky poľa `sense_ref`.

Pri vytváraní databázy je vhodné vytvoriť aj tabuľku `android_metadata`, ktorá definuje *locale*[1] (súbor parametrov na identifikáciu jazyka, štátu apod.).

[1]Locale. *Wikipedia* [online]. [cit. 2013-20-04]. Dostupné z: http://cs.wikipedia.org/wiki/Locale

Kapitola 5

Implementácia a vyhodnotenie

V tejto kapitole postupne vysvetlím jednotlivé fázy implementácie aplikácie. Aplikáciu som vyvíjal pod operačným systémom Linux, konkrétne na linuxovej distribúcií Fedora 18 „Spherical Cow“.

Na vyhľadávanie a značenie fráz slúži skript napísaný v programovacom jazyku Python verzie 2.7.3. Pri analýze vstupného XML súboru a vytváraní databázy som používal programovací jazyk Java SE 7 Update 9.

5.1 Vyhľadávanie a značenie fráz

Z dôvodu potreby detekovania fráz v kontexte viet som vytvoril skript v jazyku Python, ktorú túto činnosť vykonáva. Na vstupe tohto skriptu je slovník vo formáte XML. Následne prebieha syntaktická analýza obsahu slovníka metódou *iterparse*. Táto metóda je veľmi podobná spracovaniu pomocou syntaktického analyzátora SAX (pozri podkapitola 5.2.1).

Nájdené frázy v položkách *Lemma* sa ukladajú do zoznamu fráz. Nájdené kontexty viet sa uložia do zoznamu kontextov. Po naplnení týchto zoznamov nasleduje fáza vyhľadávania fráz v kontextoch viet. Prechádzaním zoznamov a použitím operátoru *in* sa zistí, či sa práve spracovávaná fráza vyskytuje v práve spracovávanom kontexte vety.

Pre vyhľadávanie anglických fráz som vytvoril optimalizovaný skript. Z kontextu vety je obtiažne rozlíšiť, či sa jedná o frázu. Napríklad fráza „*the floor*“ - snemovňa, zasadacia sieň parlamentu apod. nie je použitá v slovnom spojení „*standing on the floor*“. Preto frázy, ktoré obsahujú slovo „*the*“, prípadne „*The*“ nevyznačujem.

Následnou aplikáciou metódy `subn` (pozri výpis 5.1) sa nájdené frázy označia. Zvolil som označovanie pomocou ostrých zátvoriek (znakov < a >).

Po ukončení vyznačovania fráz prebieha vkladanie upravených fráz do pôvodného textu a uloženie výstupného súboru.

Výpis 5.1: Ukážka regulárneho výrazu a použitia metódy subn

```
1 #vzor vyjadrený regulárnym výrazom
2 pattern=re.compile(r'(?<=\b)(?<!<)(%s)(?!>)(?=\b)'%i)
3 #vzor náhrady
4 replacement=(r'<%s>'%i)
5 #použitie metódy subn
6 mytuple=re.subn(pattern,replacement,j,count=0,flags=0)
```

5.2 Generovanie databázy SQLite

Na generovanie databázy SQLite som vytvoril samostatný program v jazyku Java. Vytvoreniu tohto programu predchádzali pokusy o vytváranie databázy priamo v Android aplikácií, ktoré sa neskôr ukázali ako málo výkonné. Program, ktorý vytvára slovníkovú aplikáciu sa skladá zo 4 tried:

- **Trieda LMFDictParser** - Reprezentuje syntaktický analyzátor. Táto trieda sa stará aj o vkladanie potrebných položiek do vytváranej databázy.
- **Trieda LMFDatabase** - Reprezentuje vytváranú databázu. Obsahuje definíciu databázových tabuliek a indexov.
- **Trieda ProcessFile** - Trieda obsahuje metódu `main` a inicializuje potrebné súčasti.
- **Trieda LMFConstants** - Obsahuje definíciu konštánt, ktoré sú zdieľané viacerými triedami.

5.2.1 Syntaktická analýza vstupných dát

Na syntaktickú analýzu vstupných dát som zvolil **Simple API for XML (SAX)** syntaktický analyzátor [7]. Tento analyzátor používa *event-based API*, ktoré reaguje na výskyt udalostí - typicky na výskyt začiatočných a ukončovacích XML značiek. Výhoda SAX analyzátora je jeho nízka pamäťová náročnosť. Nepotrebuje v pamäti uchovávať celú stromovú štruktúru dokumentu (oproti DOM analyzátoru). Z tohto dôvodu môže dosiahnuť vačšie rýchlosti spracovania dokumentov.

Vo výpise 5.2 je uvedený spôsob vytvorenia a inicializácie SAX analyzátoru a metódy, ktoré reagujú na výskyt začiatočných a koncových XML značiek. Pri vyvolaní metódy, ktorá reaguje na začiatočnú značku rozlišujem typ značky a inkrementujem globálne premenné, ktoré vyjadrujú počet výskytov konkrétnej značky. V prípade potreby získam atribút alebo obsah XML elementu.

Výpis 5.2: Inicializácia analyzátora a metódy reagujúce na výskyt XML značiek

```
1  SAXParserFactory saxFactory = SAXParserFactory.newInstance();
2  //nová instancia analyzátoru
3  SAXParser saxParser = saxFactory.newSAXParser();
4  //inicializácia XMLReader
5  XMLReader myXmlReader = saxParser.getXMLReader();
6  //priradenia XML handler
7  LMFDictParser myXMLHandler = new LMFDictParser();
8  //priradenie content handler
9  myXmlReader.setContentHandler(myXMLHandler);
10 //syntaktická analýza XML
11 myXmlReader.parse(LMFConstants.PATH_TO_LMFXML);
12 /*************************************************************/
13 /*metóda, ktorá reaguje na výskyt začiatočnej XML značky*/
14 public void startElement(String uri, String localName, String qName,
15       Attributes attributes) throws SAXException {...}
16 /*metóda, ktorá reaguje na výskyt koncovej XML značky*/
17 public void endElement(String uri, String localName, String qName)
18       throws SAXException {...}
```

5.2.2 Vkladanie položiek do databázy

Vkladanie položiek do databázy prebieha súčasne so syntaktickou analýzou vstupného súboru. Pre pripojenie k databáze používam ovládač JDBC (SQLite JDBC driver).

Vytváranie databázy nie je kritická operácia, pri výskyte problému sa dá zopakovať. Nastavenie prepínača `synchronous` na hodnotu 0 výrazne urýchli vkladanie položiek do databázy. Ak je prepínač nastavený na hodnotu `FULL`, pri vkladaní prebieha overovanie, či sa práve vkladaná položka úspešne vložila do databázy[1].

Pri vkladaní položiek, ktorých dáta môžu obsahovať apostrofy (znak ') sa môže vyskytnúť chyba pri pokuse o vloženie. Preto je treba tento problém ošetriť zdvojením týchto apostrofov.

Vkladanie položky musí byť umiestnené vo vnútri *try - catch* bloku. Vo výpise 5.3 je uvedené nastavenie prepínača synchronous, nahradzovanie apostrofov a vloženie položky tabuľky *lemmas* do vytváranej databázy.

Výpis 5.3: Vkladanie položiek do databázy

```
1  //nastavenie prepínača synchronous
2   stmt.execute("PRAGMA synchronous=0;");
3  //vloženie položky do databázy
4   stmt.executeUpdate(
5        "insert into lemmas(_id, sense_ref, type, word) values('"
6        + lemmaId
7        + "','"
8        + sense_ref
9        + "','"
10       + typeVal
11       + "','"
12       +
13 // zdvojenie apostrofov pomocou metódy replaceAll
14 attributes.getValue(secondAtt).replaceAll("'", "''") + "');");
```

5.2.3 Vytvorenie databázových indexov

Databázové indexy urýchľujú získavanie záznamov z databázy. V dokumentácií SQLite [4] je uvedené odporúčanie vytvárať indexy pre každý stĺpec tvoriaci cudzí kľúč. Pre optimalizáciu rýchlosti vyhľadávania vytváram v databáze 3 indexy:

- **index esenseindex** pre cudzí kľúč v tabuľke *equvivalents*
- **index lemmaindex** pre cudzí kľúč v tabuľke *senses*
- **index csenseindex** pre cudzí kľúč v tabuľke *contexts*

5.2.4 Zobrazenie vytvorenej databázy

V priebehu navrhovania a vytvárania databázy je často potrebné overiť, akým spôsobom sú dáta v databáze uložené, prípadne, či uložené dáta spĺňajú požiadavky. Na tento účel som používal 2 nástroje - konzolový program **sqlite3** a **SQLite Database Browser**. Tieto nástroje umožňujú pristupovať k vytvorenej databáze a zobrazovať jej obsah.

[1] SQLite Documentation *PRAGMA Statements* [online]. 2013 [cit. 2013-04-25]. Dostupné z: http://www.sqlite.org/pragma.html

sqlite3

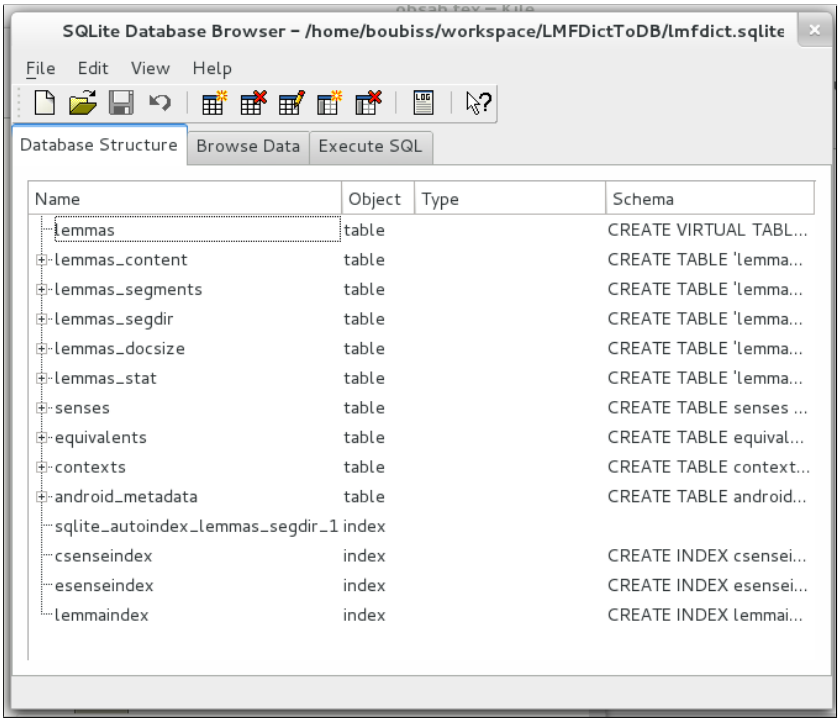
Konzolový program, ktorý umožňuje vykonávať príkazy nad SQLite databázou. Databázu treba pred zadávaním dotazov pripojiť príkazom *attach*. Vo výpise 5.4 je uvedená ukážka rozhrania, pripojenie databázy a príkaz nad tabuľkou *lemmas*.

Výpis 5.4: Ukážka rozhrania konzolového programu sqlite3

```
sqlite> attach 'lmfdict.sqlite' as 'db';
sqlite> select * from lemmas where word match 'root' limit 10;
2573|2570|adjective|aerial roots
4087|4085|verb|people alienated from their roots
5782|5780|verb|Roots anchor the plant in the earth.
20549|20549|noun|brace root
20782|20776|noun|root and branch
23878|23878|noun|buttress root
24116|24116|noun|cabbage root fly
27470|27469|noun|celery root
28589|28589|noun|characteristic root
32445|32445|noun|club root
```

SQLite Database Browser

Tento program obsahuje grafické rozhranie, ktoré poskytuje veľmi prehľadné zobrazovanie práve otvorenej databázy. Umožňuje vykonávať príkazy nad jednotlivými tabuľkami databázy a taktiež vytvárať nové tabuľky. Program vo verzii 2.0b1 ale neumožňuje vykonávať príkazy nad tabuľkami s rozšírením FTS4. Na obrázku 5.1 je ukážka grafického rozhrania tohto programu a zobrazenie súčastí virtuálnej tabuľky *lemmas*.



Obr. 5.1: SQLite3 browser

5.3 Android SDK

Android Software development kit (SDK) poskytuje nástroje, ktoré sú nevyhnutné na vývoj aplikácií pre platformu Android. V nasledujúcom texte čerpám z poznatkov získaných z knihy *Pro Android 2* [8] a z webovej stránky *Android Developers* [3]. Pri vytváraní slovníkovej aplikácie **LMFDictionary** som používal vývojové prostredie Eclipse so zásuvným modulom Android developer tools (ADT). Na webovej stránke Android SDK [1] je možné získať balík ADT Bundle, ktorý obsahuje:

- Prostredie Eclipse a ADT zásuvný modul
- Android SDK Tools
- Android Platform-tools
- Aktuálnu platformu Android
- Aktuálny snímok systému Android pre účely emulátoru

5.4 Štruktúra aplikácie

Štruktúra každej Android aplikácie je veľmi podobná. Zložka `src` obsahuje zdrojové súbory aplikácie. Táto zložka má štruktúru, ktorá vychádza zo spôsobu organizovania zdrojových súborov do balíkov ako v prípade jazyka Java.

Hlavný balík slovníkovej aplikácie *LMFDictionary* má názov „`com.vut.fit.knot.LMFDictionary`“. V štruktúre sú zahrnuté knižnice potrebné pre beh aplikácie (`android.jar`, `android-support-v4.jar`).

Zložka `src` v aplikácií LMFDictionary obsahuje 5 aplikačných tried:

- **LMFContentProvider.java** - Reprezentuje poskytovateľa obsahu.
- **LMFDatabaseConnector.java** - Poskytuje spojenie aplikácie s databázou.
- **Messages.java** - Pomocná trieda, zahŕňa funkcionalitu pre externalizáciu reťazcov.
- **ResultActivity.java** - Aktivita pre zobrazenie výsledkov vyhľadávania.
- **SearchActivity.java** - Aktivita ktorá realizuje proces vyhľadávania.

Medzi ďalšie zložky patria:

- **gen** - obsahuje vygenerovaný kód, ktorý umožňuje prístup k zdrojom (popisy rozloženia prvkov, bitmapy atď.).
- **assets** - zložka pre ukladanie voliteľných súborov/zložiek.
- **bin** - obsahuje vygenerované súbory (napríklad `.class` súbory), ktoré sú potrebné pre vytvorenie spustiteľného `.apk` súboru.
- **res** - obsahuje aplikačné prostriedky. Každý prostriedok má priradený identifikátor.

[1] http://developer.android.com/sdk/index.html

Zložka `res` obsahuje podzložky:

- **drawable** - zložka obsahujúca grafické prvky a bitmapy.
- **layout** - obsahuje súbory, ktoré definujú rozloženie prvkov aplikácie.
- **menu** - súbory definujúce vzhľad a obsah aplikačného menu.
- **values** - obsahuje rôzne programátorom definované hodnoty ako externalizované reťazce, alebo definíciu aplikačných štýlov.
- **xml** - zložka pre uloženie xml súborov, ktoré sú využité v aplikácií.

5.4.1 Kľúčové komponenty aplikácie

View (pohľad) tvorí užívateľské rozhranie aplikácie. Môže sa jednať o tlačidlo, textové pole apod. Pohľady sú hierarchické a obsahujú informácie o spôsobe vykreslenia.

Activity (aktivita) väčšinou reprezentuje jednu „obrazovku“ aplikácie a môže obsahovať niekoľko *Views*. Aktivity majú svoj životný cyklus, ktorý uvádzam v prílohe A. Aktivita, ktorá neobsahuje žiaden *View* sa nazýva Service (služba).

Intent je prostriedok, ktorý realizuje komunikáciu medzi aplikačnými komponentami. Intent je možné použiť napríklad na zahájenie služby, alebo spustenie aktivity. Intent nemusí byť iniciovaný aplikáciou, je používaný aj systémom, ktorý ním informuje aplikácie o rôznych udalostiach (príchodzí hovor).

Content Provider (poskytovateľ obsahu) je štandardný mechanizmus zdieľania dát medzi aplikáciami. Môžeme ho použiť aj s cieľom poskytovania dát len v rámci vytváranej aplikácie. Content provider zapúzdruje dáta a vytvára jednotný prístup k dátam.

5.4.2 Android Manifest

Android manifest je XML súbor, ktorý obsahuje dôležité informácie potrebné pre beh aplikácie. Manifest obsahuje informácie ako názov aplikácie, balíka, podporované verzie SDK, údaje o aktivitách, poskytovateľoch obsahu apod.

Je potrebné zvoliť minimálnu a cieľovú verziu Android SDK. Ako minimálnu som zvolil Android SDK verzie 11 (Android 3.0.x Honeycomb). Túto verziu som zvolil z dôvodu použitia prvku `SearchWidget`, ktorý nie je podporovaný v predchádzajúcich verziách. Za cieľovú SDK som zvolil momentálne aktuálnu Android SDK verzie 17 (Android Jelly Bean).

Manifest obsahuje povolenia, ktoré aplikácia vyžaduje pre svoj beh. V aplikácií používam len povolenie pre zápis do externého pamäťového priestoru.

Manifest ďalej popisuje konfiguráciu aktivít, ktoré tvoria aplikáciu. Popisuje ich spôsoby spustenia, *intenty* na ktoré tieto aktivity reagujú - tzv. **intent-filter**.

Pokiaľ v aplikácií využívame poskytovateľa obsahu, je potrebné tento prvok zaregistrovať v súbore `AndroidManifest.xml`. V prípade mojej aplikácie je to poskytovateľ „**LMFContentProvider**“.

5.4.3 Konfigurácia vyhľadávania

Search widget je prvok, ktorý reaguje na vstupné udalosti, doručuje vstupy aplikácií a realizuje návrhy vyhľadávania (*search suggestions*). Pre účely vyhľadávania v Android aplikácií je potrebné definovať konfiguráciu vyhľadávania (*searchable configuration*). Je to XML súbor, ktorý definuje správanie prvku *search widget*.

Konfigurácia vyhľadávania aplikácie LMFDictionary definuje tieto hlavné nastavenia:

- Poskytovateľa obsahu aplikácie pre účely návrhov vyhľadávania.
- Akciu, ktorú bude prevádzať *intent*, ktorý sa vytvorí po výbere návrhu vyhľadávania a dáta, ktoré bude tento *intent* obsahovať.
- Nastavenie minimálneo počtu znakov potrebných na vyvolanie návrhu vyhľadávania.
- Použitie globálneho vyhľadávania.
- Aktiváciu hlasového vyhľadávania.

5.5 Rozšírenie full-text search

V tejto podkapitole vychádzam z dokumentácie SQLite [5]. Full-text search (FTS) je modul virtuálnych tabuliek databázy SQLite. Tento modul umožňuje fulltextové vyhľadávanie podobne ako napríklad internetový vyhľadávač Google. SQLite podporuje 2 verzie modulov - FTS3 a FTS4. V aplikácií LMFDictionary používam rozšírenie FTS4 pretože:

- FTS4 oproti FTS3 obsahuje optimalizáciu, ktorá môže výrazne zvýšiť rýchlosť dotazov.
- Virtuálna tabuľka ktorá používa FTS4 môže zaberať viac pamäte ako ekvivalentná tabuľka, ktorá používa FTS3. Tabuľka s FTS4 ale poskytuje možnosti komprimovania, ktoré zmenšuje veľkosť tabuľky a tým aj použitie diskového priestoru.
- FTS4 je inováciou modulu FTS3. Pre nové verzie aplikácií je doporučené používať rozšírenie FTS4.

5.5.1 Vytvorenie FTS tabuliek

Tabuľky s rozšírením FTS4 sa vytvárajú podobne ako klasické tabuľky. Vo výpise 5.5 uvádzam zápis na vytvorenie virtuálnej databázovej tabuľky *lemmas*.

Výpis 5.5: Vytvorenie virtuálnej tabuľky lemmas

```
1 CREATE VIRTUAL TABLE lemmas USING fts4 ("_id integer primary key,
2               sense_ref integer not null,
3                         type text,
4                    word text not null,
5                    tokenize = porter);
```

Použitie rozšírenia FTS4 špecifikujem použitím kľúčového slova `USING`. Kľúčové slovo `tokenize` je použité na určenie prvku *tokenizer* pozri podkapitolu 5.5.2.

Vytvorenie virtuálnej tabuľky *lemmas* zahŕňa vytvorenie 5 reálnych tabuliek (pozri obrázok 5.1). Tieto tabuľky sa nazývajú tzv. „**shadow tables**“.

Tabuľka lemmas_content obsahuje „čisté“ (užívateľské) dáta, ktoré boli vložené do tabuľky.

Tabuľka lemmas_segments slúži na uloženie fulltextového indexu (pre potreby rýchleho vyhľadávania). Obsahuje informácie o umiestnení v B-strome[1].

Tabuľka lemmas_segdir obsahuje jeden B-strom pre každý riadok virtuálnej tabuľky.

Tabuľka lemmas_docsize a lemmas_stat obsahujú pomocné informácie a vytvárajú sa len v prípade použitia FTS4.

5.5.2 FTS tokenizer

Tokenizer je set pravidiel, podľa ktorých sa transformujú dotazované termíny pri fulltextovom dotaze [5]. SQLite obsahuje implicitne 3 možnosti nastavenia týchto pravidiel:

simple - transformuje vyhľadávací dotaz na sériu malých písmen, takže napríklad pri zadaní dotazu na termín „rOOt“ sa tento termín transformuje na „root“. Vo verzii aplikácie, ktorá slúži na preklad z českého jazyka na anglický používam tento typ.

porter - používa *porter* algoritmus na redukciu vyhľadávaného výrazu na slovný základ. Je použiteľný pre anglické termíny. Používam ho vo verzii databázy, ktorá slúži na preklad z anglického do českého jazyka.

ICU tokenizer - umožňuje nastaviť *locale* podľa ktorého sa budú upravovať vyhľávané termíny. Súčasná verzia triedy SQLite v systéme Android nie je skompilovaná s touto možnosťou a tak som *ICU tokenizer* nemohol použiť.

vlastný tokenizer - SQLite umožňuje registráciu a použitie vlastnej implementácie *tokenizer* pravidiel napísaných v programovacom jazyku C.

5.6 Prístup k databáze

Prístup k databáze zabezpečuje instancia poskytovateľa obsahu **LMFContentProvider**. Táto instancia dotazuje instanciu databázového konektora **LMFDatabaseConnector** a volá metódy tejto instancie podľa zadaného dopytu.

5.6.1 Databázový konektor

Databázový konektor zapúzdruje databázu, s ktorou pracuje aplikácia. Obsahuje metódy, ktoré priamo dotazujú databázu. Konektor v aplikácii *LMFDictionary* má v konštruktore definované operácie, ktoré v prípade absencie databázy pre potreby aplikácie skopírujú túto databázu z SD karty do lokálneho aplikačného priestoru (zložky `databases`).

V prípade, že aplikácia už obsahuje databázu v zložke `databases`, pri spustení aplikácie sa táto skutočnosť zistí pomocou metódy `checkDataBase()` a databáza sa znovu nekopíruje.

Databázový konektor *LMFDatabaseConnector* obsahuje 8 metód, pomocou ktorých je možné vykonávať dopyt po dátach z databázy. Vo výpise 5.6 je uvedená metóda `searchLemmas`.

[1] Comer, D.: Ubiquitous B-tree. *ACM Computing Surveys (CSUR)*, ročník 11, č. 2, 1979: s. 121–137.

Výpis 5.6: Metóda triedy LMFDatabaseConnector searchLemmas

```
public Cursor searchLemmas(String query, String[] columns) {
String selection = LEMMAS_WORD + " MATCH ?";
String[] selectionArgs = new String[] { query + "*" };
SQLiteQueryBuilder builder = new SQLiteQueryBuilder();
builder.setTables(tableLemmas);
builder.setProjectionMap(mapLemmas);
String limits = null;
String order = "LENGTH (" +(LEMMAS_WORD) + ") ASC";
return query(selection,selectionArgs,columns,builder,order,limits);
}
```

Metóda `searchLemmas`, ktorej telo je uvedené na výpise 5.6 vyhľadáva termíny v databázovej tabuľke *lemmas*. Na vyhľadanie používa kľúčové slovo `MATCH` ktoré umožňuje fulltextové vyhľadávanie vo virtuálnej tabuľke používajúcej rozšírenie FTS4.

K pôvodnému dotazu metóda pridáva znak „*“. Pridanie tohto znaku spôsobí, že zadanému slovu budú vyhovovať všetky termíny, ktoré obsahujú toto slovo ako prefix. Napríklad dotazu „`MATCH` 'roo*'“ bude vyhovovať slovo „root“ ale aj „rooster“, alebo „room“.

Na vytvorenie databázového SQL dotazu slúži instancia triedy `SQLiteQueryBuilder`. Je potrebné nastaviť, nad ktorou tabuľkou databázy sa má dotaz vykonať a taktiež zvoliť tzv. **projekčnú mapu**.

Projekčná mapa a ďalšie parametre

Metóda `searchLemmas` používa projekčnú mapu `mapLemmas`. Vo výpise 5.7 je uvedený príklad projekčnej mapy pre stĺpce tabuľky *equivalents* .

Výpis 5.7: Projekčná mapa pre tabuľku equivalents

```
private static HashMap<String, String> EquivsMap() {
HashMap<String, String> map = new HashMap<String, String>();
map.put(EQUIVS_ID, "_id AS " + EQUIVS_ID);
map.put(EQUIVS_TEXT, "equiv_text AS " + EQUIVS_TEXT);
map.put(EQUIVS_SID, "sense_id AS " + EQUIVS_SID);
return map;
}
```

Poskytovateľ obsahu funguje ako prostredník medzi abstraktnými a reálnymi stĺpcami ktoré definuje databáza. Aby poskytovateľ mohol pracovať s abstraktnými stĺpcami a zároveň reálnymi stĺpcami, je v triede *LMFDatabaseConnector* definovaný prvok *projekčná mapa* (projection map). Projekčná mapa mapuje názvy stĺpcov, ktoré používa poskytovateľ obsahu na názvy stĺpcov použité v databáze [8].

Implementačne je projekčná mapa instancia triedy `HashMap`, ktorá obsahuje dvojice reťazcov. Trieda LMFDatabaseConnector obsahuje 4 projekčné mapy pre 4 databázové tabuľky (*lemmas, senses, contexts, equivalents*).

Parameter `limits` určuje maximálny počet položiek navrátených databázovým kurzorom. V tele metódy `searchLemmas` je tento parameter nastavený na hodnotu `null`. To znamená, že počet položiek vrátených v kurzore nie je limitovaný.

Parameter `order` určuje, akým spôsobom budú položky vrátené kurzorom zoradené. Kurzor vrátený metódou `searchLemmas` radí položky podľa počtu znakov reťazca.

Metóda `query` pracuje priamo s databázou a podľa kritérií, ktoré jej boli predané ako parametre vracia vyhovujúce riadky tabuliek databázy v podobe databázového kurzora.

5.6.2 Poskytovanie obsahu databázy

Pri žiadosti o databázový obsah je dotazovaná instancia triedy *LMFContentProvider*. Táto trieda obsahuje metódy, ktorých úlohou je získať databázový kurzor. Tieto metódy volajú metódy databázového konektora a určujú, ktoré polia tabuliek má výsledný kurzor obsahovať.

Uri matcher
Pri žiadosti o databázové dáta je treba rozlíšiť, o aké dáta má aplikácia (užívateľ) záujem. Túto úlohu vykonáva instancia triedy `UriMatcher`. Najskôr je potrebné zaregistrovať Uri (Uniform resource identifier), ktoré budú presne indentifikovať metódy pre výber databázových polí. Príklad takejto Uri je uvedený vo výpise 5.8.

Výpis 5.8: Príklad Uri identifikujúcej metódu na vyhľadanie v tabuľke lemmas

```
1 /*vyhľadávanie podľa slova*/
2 content://com.vut.fit.knot.LMFDictionary.LMFContentProvider/lmfdict
3 /*vyhľadávanie podľa _id*/
4 content://com.vut.fit.knot.LMFDictionary.LMFContentProvider/lmfdict/#
```

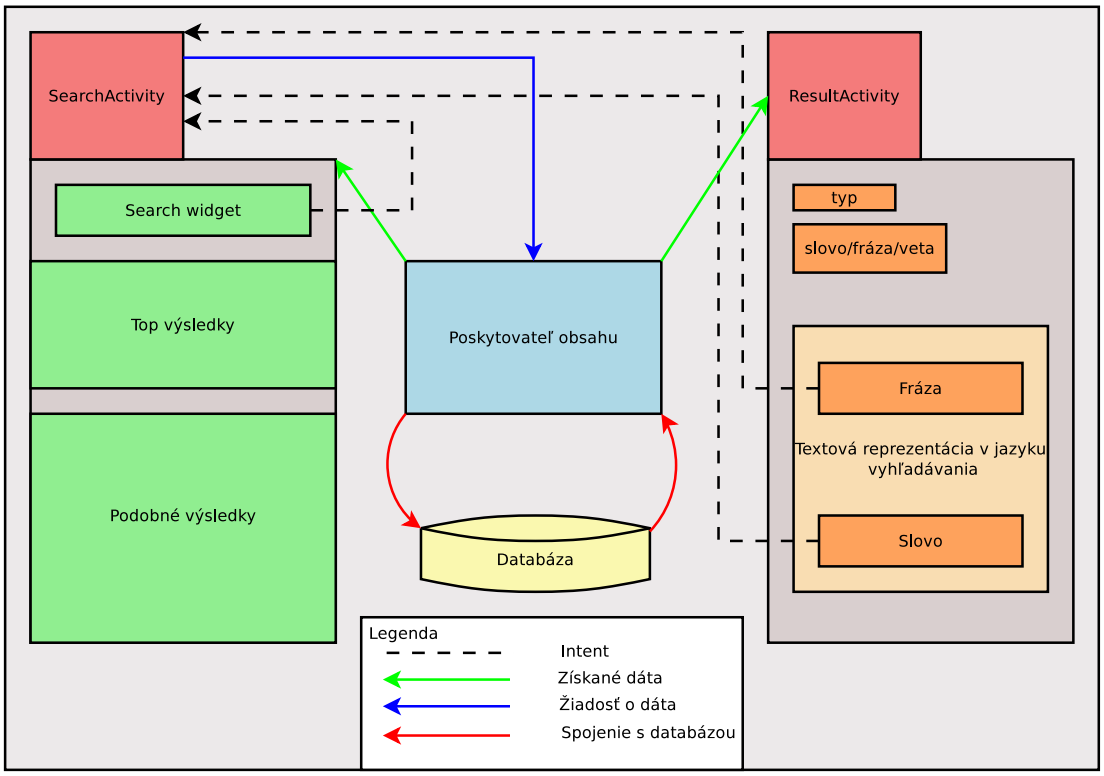
Uri uvedená na riadku č. 2 vo výpise 5.8 slúži na výber riadkov tabuľky *lemmas* podľa slova, ktoré bolo predané metóde `query` poskytovateľa obsahu. Uri uvedená na riadku č. 4 vo výpise 5.8 slúži na výber riadkov tabuľky *lemmas* podľa identifikačného čísla, ktoré obsahuje posledný úsek Uri (reprezentovaný znakom „#“). Príkazom `switch (sURIMatcher.match(uri))` sa určí typ Uri a na základe tohto typu sa prevedie metóda triedy *LMFContentProvider*, ktorá je určená touto Uri. Trieda *LMFContentProvider* obsahuje aj hlavičky metód `insert`, `update` a `delete`, ale tieto metódy nie sú používané. Ich prípadné použitie vyvolá výnimku `UnsupportedOperationException()`.

Metóda getType()
Poskytovateľ obsahu musí implementovať metódu `getType()`, ktorá vracia reťazec reprezentujúci typ **MIME**[1] pre danú Uri. Úlohou tejto metódy je rozlíšiť, či daná Uri slúži na získanie jedného, alebo kolekcie záznamov.

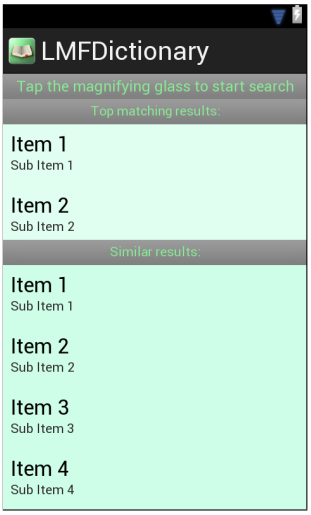
5.7 Hlavné aktivity aplikácie

Aplikácia LMFDictionary obsahuje 2 hlavné aktivity. Aktivitu `SearchActivity` a aktivitu `ResultActivity`. `SearchActivity` vykonáva funkciu vyhľadávania a ResultActivity slúži na zobrazovanie výsledkov vyhľadávania. Na diagrame 5.2 je uvedený popis týchto aktivít.

[1] FREED, Ned; BORENSTEIN, Nathaniel. Multipurpose internet mail extensions (MIME) part one: Format of internet message bodies. 1996.



Obr. 5.2: Diagram hlavných aktivít aplikácie



Obr. 5.3: Pohľad activity_search

5.7.1 Vyhľadávanie v slovníku

Aktivita `SearchActivity` slúži na vyhľadávanie slov v slovníku. V metóde `onCreate` sa nastaví pohľad `activity_search` pomocou metódy `setContentView`. Tento pohľad obsahuje prvky grafického rozhrania, ktoré užívateľ uvidí po spustení aplikácie. Na obrázku 5.3 je zobrazenie tohto pohľadu.

Pohľad activity_search
Pohľad sa skladá z dvoch hlavných častí. Vo vrchnej časti pohľadu je `ListView`, ktorý slúži na zobrazovanie výsledkov, ktoré sú najviac podobné so zadaným dopytom. Spodný `ListView` slúži na zobrazovanie ďalších podobných výsledkov.

Aplikačné menu
Menu aplikácie LMFDictionary je transformované na prvok *SearchView*, ktorý je po rozbalení obsiahnutý v aplikačnej lište akcií (*action bar*). Popis tohto prvku je uvedený v súbore `activity_search.xml` v zložke `menu`.

Metóda handleIntent
Táto metóda reaguje na instancie triedy `Intent`, ktoré sú zaslané aktivite `SearchActivity`. Na obrázku 5.2 sú tieto instancie triedy `Intent` modelované prerušovanou čiarou smerujúcou do triedy `SearchActivity`. Rozlišuje akcie `ACTION_VIEW` a `ACTION_SEARCH`.

`ACTION_VIEW` - intent s touto akciou sa v aplikácii LMFDictionary vytvára v 4 prípadoch.

1. V prípade výberu položky z návrhov vyhľadávania.
2. V prípade výberu slova v aktivite `ResultActivity`.
3. V prípade výberu frázy v aktivite `ResultActivity`.
4. V prípade výberu položky z prvku `ListView` v aktivite `SearchActivity`.

`ACTION_SEARCH` - intent s touto akciou sa vytvára pri vyvolaní vyhľadávania po zadaní dopytu a stisknutí tlačidla klávesnice určeného na spustenie vyhľadania (napr. `go`). Metóda `handleIntent` tento Intent zachytí a vyvolá metódu `populateResults()`.

Výpis vyhovujúcich slov do ListView
Metóda `populateResults` získa databázový kurzor a vypíše obsah tohto kurzora do príslušného objektu triedy `ListView`. Na tento účel sa v metóde vytvorí instancia triedy `SimpleCursorAdapter`. Následne sa tento adaptér priradí instancii triedy `ListView` a tým sa stane obsah databázového kurzora obsahom instancie `ListView`.

5.7.2 Zobrazovanie výsledkov vyhľadávania

Zobrazenie výsledov vyhľadávania prebieha v aktivite `ResultActivity`. Pri spustení tejto aktivity sa v metóde `onCreate` vytvorí samostatné vlákno, v ktorom sa vyvolá metóda `loadData`. Zároveň sa vytvorí *ProgressDialog*, ktorý upozorňuje na načítavanie dát a beží po celú dobu vykonávania tela metódy `loadData`.

Metóda loadData
Táto metóda získava relevantné databázové dáta a stará sa o ich dynamické vykresľovanie do užívateľského rozhrania. Beží v samostatnom vlákne. Postupne získava databázové kurzory. Z návrhu databázy (pozri podkapitolu 4.4) vyplýva, že pri znalosti základného *lemma* môžeme postupným prechodom získať všetky potrebné informácie, ktoré sa týkajú tohto *lemma*.

Vieme, že viaceré položky tabuľky *lemmas* môžu obsahovať rovnaký odkaz na význam (`sense_ref`). Ak sa chceme dostať k základnému slovu, ktoré chceme zobraziť vo výsledkoch, musíme získať kurzor so všetkými záznamami, ktoré majú rovnaký *sense_ref* a následne vybrať prvú položku tohto kurzoru.

Metóda `loadData` vykresľuje získané položky do užívateľského rozhrania pomocou metódy `appendView` (pozri odstavec „Metóda appendView“). O transformáciu vykresľovaných slov na hypertextové odkazy sa stará trieda Linkify (pozri odstavec „Vytváranie hypertextových odkazov“).

Metóda appendView

Táto metóda na základe parametrov pridáva požadovaný prvok do špecifikovaného prvku užívateľského rozhrania. Ako bolo spomenuté, metóda `loadData` beží v samostatnom vlákne. Pridávanie prvkov do užívateľského rozhrania je možné len z hlavného vlákna aplikácie (UI Thread). Preto táto metóda využíva volanie `runOnUiThread`, ktoré zabezpečí vykonávanie operácie v hlavnom vlákne aplikácie.

Vytváranie hypertextových odkazov

Vytváranie hypertextových odkazov je riešené využitím funkcionality triedy *Linkify*. Je potrebné pripraviť 2 vzory reprezentované regulárnymi výrazmi. Tieto vzory uvádzam vo výpise 5.9. Pomocou týchto vzorov sa vo vkladanej položke najskôr vyhľadajú frázy a následne slová.

Z nájdených fráz a slov sa vytvoria hypertextové odkazy, ktoré reagujú na stisk vytvorením objektu Intent so schémami uvedenými vo výpise 5.9. Tento Intent následne spracuje aktivita *SearchActivity*. Aby sa z fráz odstránili ostré zátvorky, ktoré su použité na ich vyznačenie, vytváram instanciu triedy *TransformFilter*, ktorú predávam ako argument metóde `addLinks`.

Výpis 5.9: Parametre metódy addLinks

```
1 //vzor frázy
2 Pattern phrasePattern = Pattern.compile("<(.+?)>");
3 //schéma Uri pre frázy
4 String phraseScheme = "search://phrase/";
5 //vzor slova
6 Pattern wordPattern = Pattern.compile("(\\p{L}{2,})");
7 //schéma Uri pre slová
8 String wordScheme = "search://word/";
```

Definícia štýlov a farieb

Vzhľad statických prvkov užívateľského rozhrania aplikácie *LMFDictionary* definujú XML súbory, ktoré sú uložené v zložke `res/layout`. V aktivite *ResultActivity* je ale potrebné pridávať prvky dynamicky. Na určenie rozloženia obsahu v prvku *TextView* používam objekt triedy *LayoutParams*. Ďalšie vlastnosti určujem šablónami. Šablóny pre prvky, ktoré dynamicky vkladá aktivita *ResultActivity* osahujú definície štýlov. Štýly sú uložené v zložke `res/styles.xml` a pre každý prvok obsahujú hodnoty ako veľkosť a farbu písma, rez písma apod. Farba prvkov je definovaná ako odkaz do súboru `colors.xml`. V súbore `colors.xml` sú položky definované názvom a priradenou hexadecimálnou hodnotou danej farby.

Takto definované šablóny sa pri vytváraní prvku aplikujú pomocou metódy `inflate`.

Hypertextové odkazy v aktivite *ResultActivity* majú v definícií štýlu priradený v položke `android:background` prvok typu *selector*. *Selector* zabezpečuje zmenu farby pozadia prvku *TextView* v prípade označenia tohto prvku.

5.8 Vyhodnotenie a testovanie aplikácie

Pomocou skriptu *LMFDictPhraseFinder.py* som vytvoril 2 XML súbory s vyznačenými frázami. Tieto súbory uvádzam v tabuľke 5.1 (`en-czlmf_marked.xml` a `cz-enlmf_marked.xml`). Z týchto súborov som vytvoril 3 druhy databáz obsahujúce slovníkové dáta. Databázové súbory `lmfdict_en-cz.sqlite` a `lmfdict_cz-en.sqlite` obsahujú dáta pre jednosmerný preklad (anglicko-český a česko-anglický). Veľkosti týchto databáz a počet obsiahnutých slov sú taktiež uvedené v tabuľke 5.1.

Súbor `lmfdict_cz-en_full.sqlite` obsahuje dáta umožňujúce obojsmerný preklad (anglicko-český a česko-anglický). Pri použití tohto súboru je ale aplikácia málo responzívna a preto doporučujem použiť slovníky s menšou velkosťou.

Počet slov a veľkosť databázy
Počet slov v databázach, ktoré používa slovník *LMFDictionary* je signifikantný (pozri tabuľka 5.1). Napríklad v popise slovníkov aplikácie *HandyLex 4 Czech* je uvedené, že počet hesiel obsiahnutých v anglicko-českom slovníku je 4000. Ďalšiu predstavu o veľkosti databázy môžeme získať pri porovnaní s veľkosťou anglicko-českého slovníka vo formáte *StarDict* dostupného na webovej stránke Michala Čihaře – Slovníky pro StarDict[1]. Veľkosť komprimovaného StarDict slovníka je 4.1 MB a veľkosť databázy SQLite použitej v aplikácii LMFDictionary je 54.7 MB (pozri tabuľka 5.1).

Nevýhodou môže byť väčšia veľkosť takto vytvorených slovníkových databáz. Pamäťový priestor nových Android zariadení sa ale neustále zväčšuje, takže uloženie väčšieho množstva dát prestáva byť problém.

Tabuľka 5.1: Vytvorené súbory

Názov súboru	Veľkosť súboru	Počet slov fráz a viet
`en-czlmf_unmarked.xml`	152.7 MB	-
`cz-enlmf_unmarked.xml`	143.1 MB	-
`en-czlmf_marked.xml`	159.9 MB	-
`cz-enlmf_marked.xml`	149.1 MB	-
`lmfdict_en-cz.sqlite`	54.7 MB	209 167
`lmfdict_cz-en.sqlite`	56.0 MB	229 757
`lmfdict_cz-en_full.sqlite`	107.2 MB	725 654

5.8.1 Porovnanie aplikácie LMFDictionary s existujúcimi slovníkmi

V tabuľkách 5.2 a 5.3 uvádzam porovnanie vytváranej aplikácie *LMFDictionary* s ostatnými aplikáciami (pozri kapitola 3). Aplikácie boli vyberané na základe podobnosti s vytváranou aplikáciou.

[1] http://cs.cihar.com/software/slovnik/

Tabuľka 5.2: Porovnanie vyhľadávania

Názov aplikácie	**História vyhľ.**	**Vyhľ. fráz / viet**	**Hlasové vyhľ.**
LMFDictionary	×	✓ / ✓	✓
GoldenDict Free	✓	✓ / ×	✓
ColorDict Dictionary Wikipedia	✓	✓ / ×	✓
HandyLex 4 Czech	✓	✓ / ×	✓

Tabuľka 5.3: Využitie slovníkov

Názov aplikácie	**SD karta**	**Podpora viacerých slovníkov súčasne**
LMFDictionary	✓	×
GoldenDict Free	✓	✓
ColorDict Dictionary Wikipedia	✓	✓
HandyLex 4 Czech	×	×

Najväčšou výhodou vytvoreného riešenia je rozšírená podpora vyhľadávania fráz a rozsiahlosť slovníkovej databázy. Aplikácia nepodporuje vyhľadávanie vo viacerých slovníkových databázach súčasne a nemá implementované udržiavanie histórie vyhľadávania.

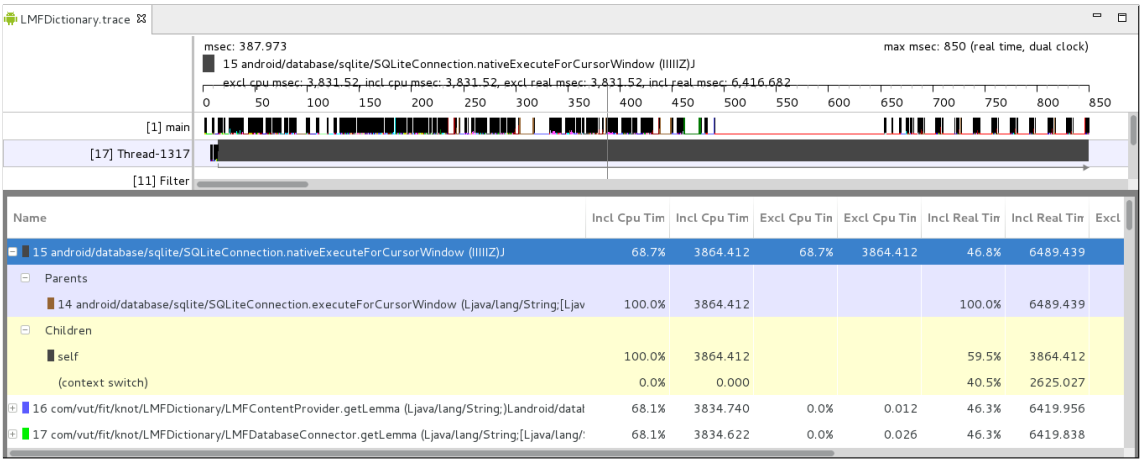
5.8.2 Ladenie a testovanie aplikácie

Aplikácia *LMFDictionary* bola testovaná priebežne pri vytváraní. Android SDK obsahuje emulátor Android zariadenia. Neskôr bola aplikácia testovaná na porte systému Android – virtuálnom stroji *Android-x86*. Následne som aplikáciu testoval na fyzickom zariadení *Samsung GT-I9000* s verziou Android systému 4.2.1.

Pri testovaní funkcionality som používal ladiaci nástroj obsiahnutý vo vývojovom prostredí Eclipse a taktiež **Dalvik Debug Monitor Server** (DDMS). DDMS obsahuje informácie o procesoch a vláknach spustených na zariadení, využití pamäte apod.

Pri ladení rýchlosti zobrazovania výsledkov v aktivite *ResultActivity* som používal nástroj **Traceview**, ktorý poskytuje prehľad o rýchlosti vykonania operácií. Ukážka zobrazenia súboru `LMFDictionary.trace` pomocou nástroja Traceview vo vývojovom prostredí Eclipse je uvedená na obrázku 5.4.

V priebehu ladenia som často potreboval zasiahnuť do súborového systému zariadenia. Na tento účel som využíval spojenie so zariadením pomocou konzolového nástroja **ADB Shell**.



Obr. 5.4: Zobrazenie trace súboru nástrojom Traceview

Rýchlosť vyhľadávania

Rýchlosť vyhľadávania v aplikácii *LMFDictionary* je pomalšia ako v podobných slovníkových aplikáciách. Nižšia rýchlosť vyhľadávania je spôsobená použitím veľkého objemu zdrojových dát. Ak bolo spomenuté v podkapitole 5.2.3 pre zvýšenie výkonnosti databázy boli vytvorené databázové indexy.

Ďalšími optimalizáciami by sme prevdepodobne mohli dosiahnuť vyšších rýchlostí prístupu k databáze. Spomalenie môže spôsobovať dynamické pridávanie prvkov v aktivite *ResultActivity*.

Nižšia rýchlosť sa ale netýka návrhov vyhľadávania a zobrazení výsledkov. Tieto operácie sa zdali užívateľom pri testovaní plynulé. Pri prechode na zobrazenie výsledkov je dlhšia čakacia doba riešená zobrazením instancie triedy *ProgressDialog*. Užívatelia majú pocit, že aplikácia „pracuje“.

Kapitola 6

Záver

Cieľom tejto práce bolo vytvoriť slovníkovú aplikáciu pre systém Android. V rámci riešenia som vytvoril nielen aplikáciu pre zobrazovanie slovníkových dát, ale aj súčasti umožňujúce vytváranie slovníkových databáz.

Vo svojej práci som vychádzal zo zdrojových dát vo formáte LMF. Navrhol som a implementoval skript v programovacom jazyku Python, ktorý slúži na vyznačovanie fráz v zdrojovom súbore. Taktiež som navrhol slovníkovú databázu a systém, ktorý ukladá vybrané položky zo zdrojového XML súboru do tejto databázy.

Hlavnou časťou práce bola implementácia aplikácie pre systém Android zobrazujúcej slovníkové dáta. Výsledným produktom je aplikácia **LMFDictionary**, ktorá umožňuje načítavanie vytvorených slovníkových databáz vo formáte SQLite a zadávanie dotazov nad týmito databázami. Aplikácia disponuje fulltextovým vyhľadávaním, vyhľadávaním fráz a viet. Práve schopnosť vyhľadávania viet obsiahnutých vo vstupnom slovníku túto aplikáciu odlišuje od konkurencie. Aplikácia ďalej umožňuje znovupoužitie výsledkov vyhľadávania vo forme nového dotazu pomocou hypertextových odkazov a podporuje hlasové vyhľadávanie.

Za povšimnutie stojí aj objem dát, s ktorými aplikácia **LMFDictionary** pracuje. Pri použití anglicko-českej verzie prekladového slovníka má SQLite databáza veľkosť 56.0 MB a obsahuje 229 757 slov vrátane fráz a viet (pozri tabuľka 5.1).

Vytvoril som niekoľko verzií slovníkových databáz (pozri podkapitola 5.8). Veľkou výhodou vyššie spomenutého riešenia je možnosť vytvárania ľubovoľných verzí slovníkov. Spôsob vyhotovenia je kompatibilný s rôznymi jazykovými variáciami. Nutosťou je ale zachovanie rovnakého formátu vstupných súborov.

V prípade odlišného formátu je tu možnosť úpravy aplikácie **LMFDictToDB** tak, aby vyhovovala novému formátu.

6.1 Možnosti vylepšenia a ďalšieho vývoja projektu

Ako bolo spomenuté, riešenie umožňuje vytvoriť ďalšie verzie slovníkov a tým výrazne rozšíriť možnosti aplikácie. V tabuľkách 5.2 a 5.3 je uvedené porovnanie aplikácie **LMFDictionary** s podobnými existujúcimi aplikáciami. Je možné doplniť podobnú funkcionalitu akú poskytujú konkurenčné aplikácie. Ide napríklad o históriu vyhľadávania, možnosť používania viacerých slovníkov súčasne alebo dopĺňanie užívateľom definovaných slov do slovníkových databáz.

V súčasnom stave je implementované provizórne načítavanie slovníkovej databázy z SD karty. Aplikácia očakáva výskyt databázového súboru `lmfdict.sqlite` v zložke SD karty

`LMFDatabase`. Bolo by prínosné toto riešenie reimplementovať tak, aby mal užívateľ možnosť výberu slovníkových databáz, ktoré sa nachádzajú na SD karte zariadenia.

V prípade dizajnu aplikácie **LMFDictionary** som sa inšpiroval existujúcimi slovníkovými aplikáciami (pozri prílohu B). Myslím si, že pri vývoji mobilnej aplikácie je dôležitá práca v tíme. O grafické rozhranie, voľbu celkového dizajnu a farieb by sa mal starať profesionálny grafik. Preto by som v rámci zlepšenia vizuálneho vnemu a pôsobenia na užívateľa zvážil celkovú rekonštrukciu vzhľadu aplikácie profesionálom.

Ďalšou zaujímavou inováciou by bolo pridať do aplikácie možnosť prekladu užívateľom definovaných viet, podobne ako napríklad vo webovej aplikácií Google Translator. Možné je aj integrovanie vyhľadávania v populárnych online webových slovníkoch a službách (Merriam-Webster, Wikipedia).

Literatúra

[1] Aplikácie pre Android v aplikácii Google Play. [online], cit. 2013-04-20.
URL https://play.google.com/store

[2] Storage Options. [online], cit. 2013-04-23.
URL http://developer.android.com/guide/topics/data/data-storage.html

[3] Android Developers. [online], cit. 2013-04-25.
URL http://developer.android.com/index.html

[4] SQLite Documentation. [online], cit. 2013-04-25.
URL http://www.sqlite.org/docs.html

[5] SQLite FTS3 and FTS4 Extensions. [online], cit. 2013-04-29.
URL http://www.sqlite.org/fts3.html

[6] Brown, K.: *Encyclopedia of Language and Linguistics, 14-Volume Set*. Elsevier Science, 2005.

[7] Harold, E. R.: *Processing XML with Java: a guide to SAX, DOM, JDOM, JAXP, and TrAX*. Addison-Wesley Professional, 2003.

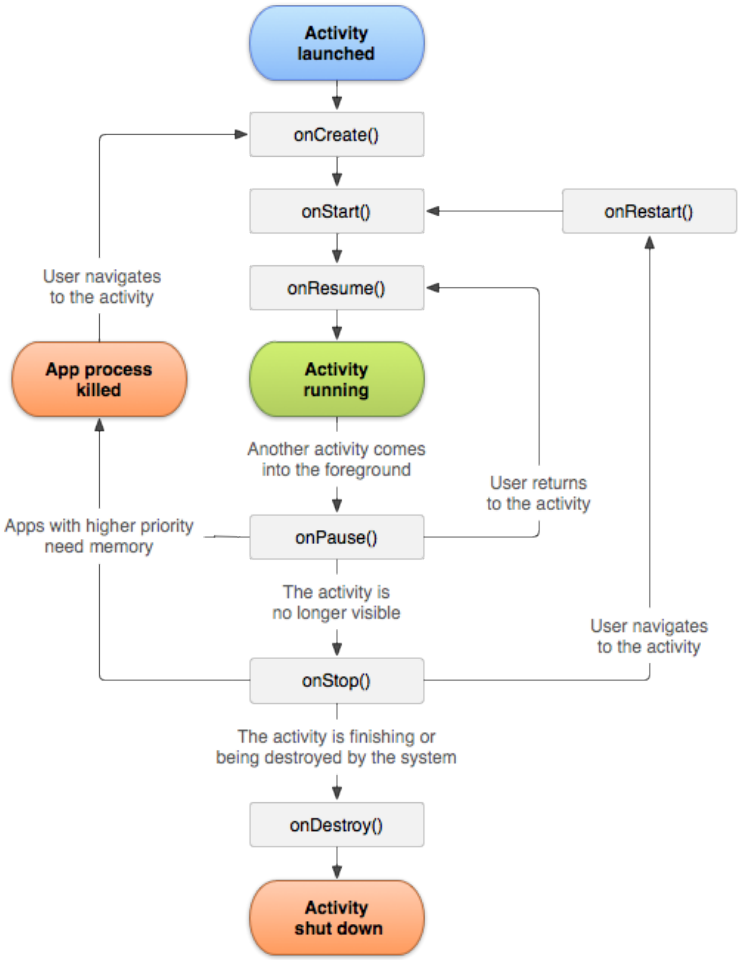
[8] Hashimi, S.; Komatineni, S.; MacLean, D.: *Pro Android 2*. 2, Apress, 2010.

[9] Ide, N.; Veronis, J.: Extracting knowledge bases from machine-readable dictionaries: Have we wasted our time. In *KB&KS Workshop*, 1993, s. 257–266.

[10] ISO: Language resource management – Lexical markup framework (LMF). ISO 24613, International Organization for Standardization, Geneva, Switzerland, 2008.

Dodatok A

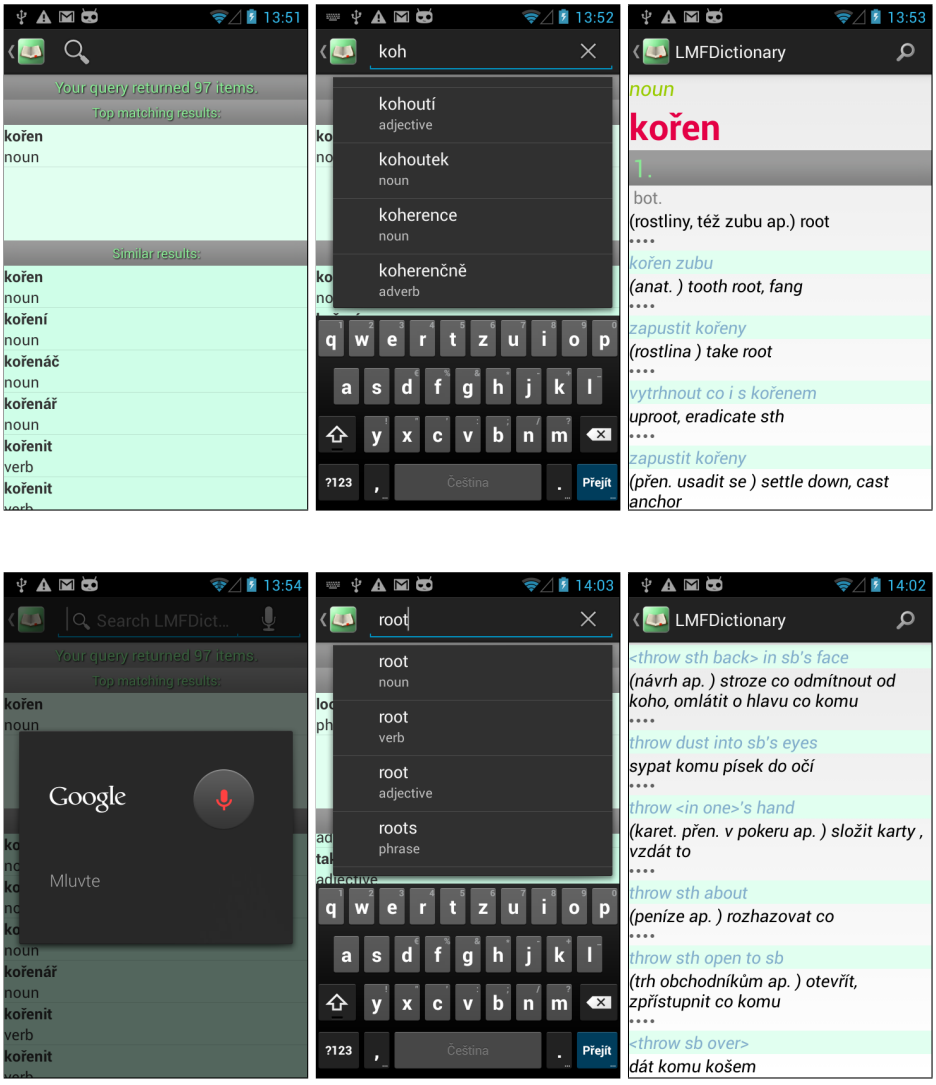
Životný cyklus Android aktivít



Obr. A.1: Životný cyklus Android aktivity

Dodatok B

Snímky obrazovky aplikácie



Obr. B.1: Snímky obrazoviek aplikácie LMFDictionary

Dodatok C

Obsah CD

Tabuľka C.1: Obsah priloženého CD

`Android_app`	Obsahuje zdrojové súbory Android aplikácie *LMFDictionary*.
`DB_app/phrase_finder`	Obsahuje zdrojový súbor skriptu pre vyznačovanie fráz `LMFDictPhraseFinder.py`.
`DB_app/LMFDictToDB`	Obsahuje zdrojové súbory programu pre vytváranie slovníkových databáz.
`README`	Manuál popisujúci použitie programových súčastí.
`source_LMF_XML`	Zdrojové slovníkové dáta vo formáte XML.
`sqlite_dicts`	Vytvorené slovníky vo formáte SQLite.
`thesis_text`	Obsahuje zdrojové súbory technickej správy a technickú správu vo formáte pdf.