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

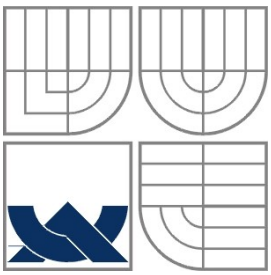
DATABÁZE XML PRO SPRÁVU SLOVNÍKOVÝCH DAT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

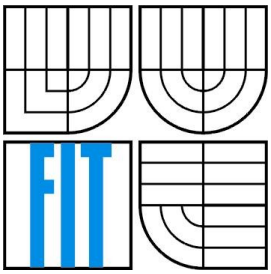
AUTOR PRÁCE
AUTHOR

Bc. MICHEL SAMIA

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DATABÁZE XML PRO SPRÁVU SLOVNÍKOVÝCH DAT

XML DATABASES FOR DICTIONARY DATA MANAGEMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Michel Samia

VEDOUCÍ PRÁCE
SUPERVISOR

doc. RNDr. Pavel Smrž, Ph.D.

BRNO 2011

Zadání diplomové práce/12950/2010/xsamia00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2010/2011

Zadání diplomové práce

Řešitel: **Samia Michel, Bc.**
Obor: Informační systémy
Téma: **Databáze XML pro správu slovníkových dat**
XML Databases for Dictionary Data Management
Kategorie: Databáze

Pokyny:

1. Seznamte se s nástroji pro práci s XML, XMLSchema atd.
2. Na základě existujících řešení navrhněte a realizujte systém pro správu slovníkových dat ve formátu XML.
3. Diskutujte výhody a nevýhody daného řešení na základě porovnání obsahu několika testovacích překladových a výkladových slovníků.
4. Vytvořte dokumentaci k realizovanému systému.

Literatura:

- podle dohody

Při obhajobě semestrální části diplomového projektu je požadováno:

- bez požadavků

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrž Pavel, doc. RNDr., Ph.D.**, UPGM FIT VUT
Datum zadání: 20. září 2010
Datum odevzdání: 25. května 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Práce se zabývá automatizací zpracování slovníkových dat, především těch ve formátech postavených na XML. Čtenář je nejprve seznámen s použitými pojmy lingvistiky a lexikografie, dále jsou představeny jednotlivé typy formátů lexikografických dat a konkrétní formáty a diskutovány jejich výhody a nevýhody. Z nich je vybrán na základě určitých kritérií formát LMF a pro něj navržena a realizována aplikace v Pythonu zaměřená především na inteligentní spojování více slovníků do jednoho.

Tato aplikace byla poté, co prošla všemi jednotkovými testy, použita také pro zpracování LMF slovníků, které jsou uloženy na školním serveru výzkumné skupiny pro zpracování přirozeného jazyka.

Na závěr jsou diskutovány výhody a nevýhody takto navržené a implementované aplikace a nastíněny možnosti dalšího použití a rozšiřování.

Abstract

The following diploma thesis deals with dictionary data processing, especially those in XML based formats. At first, the reader is acquainted with linguistic and lexicographical terms used in this work. Then particular lexicographical data format types and specific formats are introduced. Their advantages and disadvantages are discussed as well. According to previously set criteria, the LMF format has been chosen for design and implementation of Python application, which focuses especially on intelligent merging of more dictionaries into one.

After passing all unit tests, this application has been used for processing LMF dictionaries, located on the faculty server of the research group for natural language processing.

Finally, the advantages and disadvantages of this application are discussed and ways of further usage and extension are suggested.

Klíčová slova

lexikografie, XML, LMF, Stardict, OLIF, XDXF, Wordnet, Python, NLP, zpracování přirozeného jazyka, Data Categories, isocat, UML, RESTful, DOM, etree.

Keywords

lexicography, XML, LMF, Stardict, OLIF, XDXF, Wordnet, Python, NLP, natural language processing, Data Categories, isocat, UML, RESTful, DOM, etree.

Citace

Michel Samia: Databáze XML pro správu slovníkových dat, Brno, FIT VUT v Brně, 2011

Databáze XML pro správu slovníkových dat

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. RNDr. Pavla Smrže, Ph.D.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Michel Samia
13.3.2011

Poděkování

Rád bych poděkoval vedoucímu mé diplomové práce, panu doc. RNDr. Pavlu Smržovi, Ph.D., za odbornou pomoc při konzultacích.

Obsah

1 Úvod

Při zpracování přirozeného jazyka je často velkým problémem nedostupnost kvalitních, obsáhlých strukturovaných lexikografických dat. Přitom kvalita těchto dat výrazně ovlivňuje kvalitu nástrojů řešících jednotlivé úkoly, se kterými se v této oblasti setkáváme, například morfologickou analýzu, zjednoznačňování významů slov nebo strojový překlad.

Často máme k dispozici více lexikálních databází z různých zdrojů a naším cílem je tyto zdroje zanalyzovat a na základě provedené analýzy vytvořit jejich sjednocení, tedy spojit je. Tento úkol, ačkoli se na první pohled zdá být triviální, může být ve skutečnosti časově, a tedy i finančně, velmi náročný, neboť v datech z různých zdrojů mohou být stejné prvky označeny různě, hesla mohou mít různé definice, různý počet významů, a podobně. V neposlední řadě může být problémem, jsou-li jednotlivé zdroje v různých formátech.

1.1 Cíl práce

Cílem této práce je navrhnout a vytvořit program, který pomůže tento problém řešit, tedy co nejvíce zautomatizovat právě práci s lexikografickými zdroji, především jejich analýzu a spojování. Systém by měl být schopný sám zvládnout rozhodovat, které položky jsou dostatečně podobné na to, aby mohly být spojeny, a které mají naopak zůstat jako dva samostatné záznamy.

Dále by měl zvládat konverze atributů do společné podoby tak, aby na výsledném souboru dat bylo co nejméně poznat, že byl vytvořen z více různorodých zdrojů. V neposlední řadě by měl být navržen pokud možno co nejobecněji a nejmodulárněji, aby mohl být dále rozšiřován dle potřeby a požadavků uživatelů.

1.2 Průvodce kapitolami

Nejprve bylo třeba se seznámit s některými důležitými pojmy z oborů lingvistiky a lexikografie, neboť jejich definice se může dle úhlu pohledu na problematiku mírně lišit. O tomto pojednává kapitola 2. Dále bylo potřeba prozkoumat existující formáty pro uložení lexikografických dat a nástroje pro práci s nimi (kapitola 3). Tuto část jsem z velké části vyřešil v rámci semestrálního projektu. Na základě tohoto průzkumu pak bude vybrán meta-formát LMF (kapitola 3.7), pro nějž bude navržen a vytvořen nástroj nový, který umožní mimo jiné spojování více zdrojů (kapitola 4). Tento nástroj pak bude otestován, a to jednak umělými unit testy pokrývajícími většinu jeho funkčnosti a jednak na reálných datech uložených na školním serveru minerva1 (kapitola 5). V závěru pak zhodnotíme výsledky testů a klady a zápory takto navrženého systému, především s ohledem na další rozšiřitelnost (kapitola 6).

2 Definice pojmů

S oblastí zpracování přirozeného jazyka, počítačové lingvistiky a lexikografie se pojí spousta pojmů a jejich znalost je podmínkou nutnou pro jakoukoli práci v této oblasti. Proto si ty z nich, které jsou pro naši práci nejdůležitější, nyní blíže představíme.

2.1 Lexikografická databáze

Lexikografickou databází rozumíme data popisující slova, jejich významy, výslovnost, překlady a další důležité aspekty jednoho či více přirozených jazyků. V užším slova smyslu se může jednat také o lexikografickou databázi ovládanou systémem řízení báze dat.

Lexikografické databáze tvoří základ počítačových slovníků a také se používají jako součást komplexnějších systémů pracujících s přirozeným jazykem, například při získávání znalostí z textu či při strojovém překladu.

2.2 XML

XML je způsob ukládání strukturovaných informací pomocí značkování. Vznikl zjednodušením (t.j. zpřísněním) staršího standardu SGML. Výhodou XML je, že tvoří vhodný kompromis mezi možností snadného strojového zpracování a mezi čitelností lidmi. Pro zpracování dat v XML metaformátu vzniklo velké množství knihoven pro celou řadu programovacích jazyků. Dále existují nástroje pro specifikaci konkrétních formátů (DTD, XML Schema) a následnou validaci souborů dle těchto schemat, nástroje pro navigaci v rámci XML dokumentu (XPath), nástroje pro transformaci XML dokumentů na jiné dokumenty (XSLT).

2.3 XPath

XPath je, jak jsme již naznačili, nástroj (jazyk) pro výběr jednoho či více elementů a/nebo atributů z XML dokumentu. Umožňuje zadání cesty jak absolutně (tedy od kořenového uzlu), tak relativně od nějakého známého uzlu. Dále poskytuje možnost používat takzvané *wildcards*, čili při zadávání cesty symbolem '*' nahradit jeden či více elementů.

2.4 XSL Transformace

XSL transformace umožňují jednoduchým způsobem převádět XML dokument z jednoho XML schématu na jiné, či dokonce na ne-XML formát (třeba na obyčejný textový soubor). XSL Transformace funguje tak, že uživatel nejprve napíše takzvanou šablonu (soubor s koncovkou xsl), ve které specifikuje, kde se mají za značky ve jmenném prostoru 'xsl:' dosadit jednotlivé položky původního dokumentu a poté spustí XSLT procesor (na unix like systémech například xsltproc) a jako parametr mu předá výše uvedenou šablonu a vstupní soubor. XSLT procesor pak provede dané substituce a iterace (XSL je turingovsky úplný, umožňuje cykly, podmínky, a práci s proměnnými) a výsledkem je substituovaný soubor. XSL k výběru elementů a atributů používá právě výše zmíněný jazyk XPath.

2.5 XML databáze

XML databáze je databáze schopná načítat a vracet data ve formátu XML bez účasti jiného software (typicky middleware). Motivací pro vznik takovýchto databází může je například to, že import do databáze a export z ní se často provádí ve právě pomocí XML a jestliže použijeme XML databázi, odpadnou nám tyto dvě konverze (nebo se spíše částečně převedou na stranu systému řízení báze dat). XML databáze se dají rozdělit do dvou skupin dle stupně podpory práce s XML. *XML enabled databáze* jsou takové databáze, které si vnitřně ponechávají svoji relační architekturu a XML vstup pouze konvertují na řádky relační tabulky. Stejně tak při vracení výsledku jednotlivé řádky konvertují zpět na XML uzly. Naproti tomu n*ativní XML databáze* pracují přímo interně s XML uzly/dokumenty. Tyto uzly však nemusí být – a většinou ani nejsou – uloženy ve formě jednoho dlouhého textového řetězce, ale spíše jako stromová struktura. Pro urychlení vyhledávání v datech mohou, tak jako relační databáze, používat indexy. [WIKIPEDIA 2011a]

Jedním z populárních zástupců nativních XML databází je například projekt Sedna. Ta je psaná v C++, do databáze ukládá vztahy mezi všemi přímo příbuznými uzly (rodič, syn, sourozenci) a největší její výhodou je to, že sama dokáže na základě malého vzorku dat vygenerovat schéma, na základě nějž pak s daty efektivněji pracuje. [WIKIPEDIA 2011b]

2.6 Lexikografické pojmy

V této části si nadefinujeme celou řadu dalších pojmů z oboru lexikografie. *Lexikální jednotka* (anglicky *lexical entry*, *lexical item* nebo *lexical unit*) je základní jednotka slovní zásoby jazyka. Je to množina všech tvarů určitého slova nebo slovního spojení [WIKIPEDIA 2011c]. *Lexém* má velice podobný význam jako lexikální jednotka, ale na rozdíl od ní popisuje všechny tvary pouze jednoho slova, takže například slovní spojení by-the-way jsou tři lexémy, ale pouze jedna lexikální jednotka . Lexém je abstraktní jednotka, jeho konkrétní vyjádření se nazývá *lex*. Pokud má určité slovo více tvarů, obvykle v důsledku skloňování nebo časování, označují se tyto jednotlivé tvary jako *alolexy*

[WIKIPEDIA 2009d]. *Lemma* je pak základní alolex slova uváděný ve slovnících. *Lemmatizátor* je nástroj (program), který k danému řetězci reprezentujícímu určitý alolex najde jeho lemmata.

2.7 Pojmy morfologie

Morfém je nejmenší vydělitelná část slova, která je nositelem věcného nebo gramatického významu. Morfémy lze rozdělit na předpony (prefixy), kořeny, vpony (infixy), přípony, a koncovky. Souhrnné označení pro přípony a koncovky je sufix. Předpona, vpona, přípona a koncovka se souhrnně označují jako *afixy*. Konkrétní realizací morfémů jsou pak *morfy*.

2.8 Pojmy korpusové lingvistiky

Korpus je rozsáhlý strukturovaný soubor textů, určený pro účely statistické analýzy (například získání *frekvenčních seznamů* slov). Korpusy bývají opatřeny takzvanými *anotacemi*, což jsou jednak meta-informace jako autor, žánr nebo rok vytvoření, a jednak značky popisující do kterého slovního druhu (angl. *part-of-speech tags*), a jednak do kterého významu (*sémantické tagy*) daný lexém spadá. Může obsahovat i informace o dalších morfosyntaktických kategoriích, jako například pád, číslo, rod, osoba, způsob, rod, vid, typ (u zájmen, spojek) apod. Ty bývají označovány jako *morfosyntaktické tagy*.

Korpusy se mohou dělit na *synchronní* a *diachronní*. Synchronní obsahují vyvážený otisk jazyka pouze z krátkého časového období během něhož lze jazyk považovat za neměnný. Typicky se jedná o korpusy současných podob jazyků. Naproti tomu diachronní korpusy zachycují jazyk v různých vývojových fázích. Dále je lze dělit na jednojazyčné a vícejazyčné (paralelní).

2.9 Pojmy lexikální sémantiky

Synset (angl. též *synonym set* nebo *synonym ring*, česky *synonymická řada*) je množina lexémů, mající stejný, nebo velmi podobný význam (slova souznačná, synonyma). *Holonymum* k danému slovu je slovo popisující celek, jehož je dané slovo částí. Například holonymem slova okno je slovo dům. Opakem holonyma je *meronymum*, tedy slovo popisující část daného objektu. Okno je tedy meronymem ke slovu dům. *Hyponymum* daného slova, tedy slovo danému slovu podřazené, je slovo, jehož význam popisuje pouze některý typ objektů, popsaných daným slovem. Například kolie je hyponymem slova pes. Naopak *hypernymum* daného slova (též *hyperonymum*) neboli slovo danému slovu *nadřazené* je slovo nebo slovní spojení jehož význam popisuje nadmnožinu množiny objektů popsaných daným slovem. Například hypernymem ke slovu pes je psovitá šelma. *Antonymum* je slovo opačného významu, například antonymem k přídavnému jménu vysoký je nízký.

3 Formáty lexikografických dat

V současné době existuje a je používána již celá řada formátů pro ukládání lexikografických dat. V této kapitole se seznámíme s různými typy formátů a nejznámější reprezentanty jednotlivých typů si blíže představíme. Obecně se dají všechny formáty lexikografických dat rozdělit do tří velkých skupin (typů).

Nejstarším typem jsou jednoduché plain text formáty. Ty vznikly spontánně jakožto nejjednodušší způsob, jak na počítači ukládat strukturované informace. Známým zástupcem tohoto formátu je například formát Anglicko-Českého GNU FDL slovníku. Nevýhodou plaintext formátů je pomalý způsob vyhledávání - po rozparsování na řádky je v abecedně seřazeném slovníku bez dalších pomocných datových struktur časová složitost vyhledávání logaritmická, značeno O(log n). Dalším problémem tohoto typu formátu je (nebo spíše byla) jejich velikost.

Novějším typem jsou speciální binární formáty. Ty se snaží o minimalizaci přístupové doby pomocí vytváření indexu a/nebo velikosti souborů pomocí komprese. Mohou například používat strukturu zvanou trie. Jejich nevýhodou je nečitelnost bez implementace speciální čtečky a nemožnost úpravy jednoduchým textovým editorem.

A konečně třetí a nejnovější skupinou jsou formáty postavené na XML. U nich byl kladen důraz především na maximální strukturovanost, obecnost, čitelnost a škálovatelnost. Naopak rychlost přístupu k datům a úspornost místa v paměti již nebyly klíčové, neboť v době jejich vzniku byly paměti již přijatelně levné a jejich kapacita dostatečně vysoká. Koneckonců i u této kategorie lze problém objemu dat řešit kompresí, dokonce kompresní poměr může být u XML souborů ještě vyšší než u formátů předchozí kategorie. Z této skupiny si představíme formáty XDXF, OLIF a LMF.

3.1 Česko-Anglický GNU/FDL slovník

Jedná se o projekt Milana Svobody, externího zaměstnance ZČU v Plzni, jenž si dal za cíl vytvořit svobodný anglicko-český slovník. Data jsou uložena v jednoduchém textovém formátu a lze je stáhnout z [SVOBODA 2004a]. K tomuto slovníku existuje i spousta aplikací pro prohlížení a vyhledávání. [tamtéž]

Vlastní formát vypadá takto: každá výrazová dvojice je na samostatném řádku. Každý řádek je tvořen pěti poli oddělenými tabulátorem. První pole obsahuje anglický výraz, druhé obsahuje český výraz, třetí obsahuje poznámku, čtvrté speciální poznámku a poslední obsahuje jméno autora. Třetí pole, tedy poznámka, by měla obsahovat zkratu slovního druhu, rod, číslo (nepovinně - pokud neobsahuje, jde o singulár), tematickou oblast, a prioritu překladu. Speciální poznámka má sloužit k podrobnějšímu popisu významu, může obsahovat definici, jak ji známe z výkladových slovníků, a to až do délky 255 znaků. Nicméně toto použití není povinné a má spíš sloužit u slov, kde je potřeba upřesnit, o jaký význam jde. Speciální poznámka může obsahovat i odkazy na ostatní slova. [SVOBODA 2004b]

3.2 Stardict formát

Program Stardict patří mezi stálice na poli svobodných a open source slovníkových programů. Formát slovníků programu Stardict je jedním z binárních formátů optimalizovaných pro rychlé zobrazování, tedy zástupcem druhé skupiny našeho rozdělení. Je typickým příkladem jednoduchého formátu popisujícího spíše vzhled definic a překladů, než nejrůznější morfologické, syntaktické a sémantické informace. Každý slovník se skládá ze tří souborů, které mají stejný název, ale odlišnou koncovku. Soubor s koncovkou `ifo` je malý textový soubor obsahující metainformace jako např. autor, počet slov a podobně. Druhý soubor má koncovku `idx` a obsahuje trojice *(heslo, celková délka dat, offset)* jednotlivých hesel v slovníku. Poslední soubor má koncovku `dict` a obsahuje heslo, kód typu dat a data. Daty může být např. plaintext, HTML, mediawiki text, ale také třeba výslovnost ve formátu wav. Přesnou specifikaci formátu najdete v [GARCÍA 2007].

3.3 Wordnet

Wordnet je lexikální databáze anglického jazyka vyvinutá v Laboratoři kognitivní vědy Princetonské univerzity pod vedením profesora psychologie George A. Millera. Seskupuje slova to synsetů, obsahuje definice, příklady a především sémantické vztahy mezi synsety. Synsety jsou propojeny různými typy ukazatelů: nadřazenost (hypernyma) / podřazenost (hyponoma), část (meronyma) / celek (holonyma), opak (antonyma), slova odvozená.

Wordnet neobsahuje všechny slovní druhy, ale pouze slovní druhy z tzv. *otevřené třídy* (angl. *open word class*), tedy slovní druhy tvořené množinami slov, do kterých vývojem jazyka přibývají nová slova. Konkrétně jde tedy o podstatná jména, přídavná jména, slovesa a příslovce. Do otevřené třídy slovních druhů patří i citoslovce, ty však pro svoji povahu do Wordnetu samozřejmě zařazeny nejsou.

Zdrojová data jsou uložena v textových lexikografických souborech. Na každém řádku je jeden synset. Z hlediska našeho rozdělení se tak jedná o první skupinu formátů. U víceslovných lexémů se místo mezery používá podtržítko. Více se o formátu zdrojových dat dočtete v [WORDNET 2003a]. Ty jsou pak následovně zpracovány programem `grind`. Zdrojová data i program grind lze stáhnout z [WORDNET 2003b]. Výstupem grindu jsou pak databázové soubory `data.`*`pos`* a `index.`*`pos`*, kde *pos* = *'adj'* | *'adv'* | *'noun'* | *'verb'*, a `index.sense`. Tato výstupní data jsou v systému Debian dostupná v balíku *wordnet-base*.

Wordnet není jen samotná lexikální databáze, ale obsahuje také knihovnu v jazyce C pro práci s ní a konzolové (balík wordnet) i grafické (balík wordnet-gui) rozhraní. Dále lze pro přístup k výstupním datům Wordnetu využít skriptovací jazyk Python díky modulu wordnet balíku NLTK.

3.4 Extensible Dictionary Exchange Format

Extensible Dictionary eXchange Format, zkracováno jako XDXF, je formát slovníkových dat vzniklý za účelem převodu existujících slovníků do počítačové podoby tak, aby jednotlivé informace o slově nebyly uloženy pouze jako nějak graficky formátovaný řetězec (například pomocí HTML), ale aby byly zaznamenány strukturovaně na logické úrovni. Je založený na XML a pro jednotlivé tagy definuje i jejich sémantiku. [SIGNOV 2006]

Nyní si popíšeme základní značky obsažené v dokumentech ve formátu XDXF. Kořenovým tagem je `<xdxf>`. Ten povinně obsahuje atributy `format`, `lang_from` a `lang_to`. `format` může být buďto `logical`, nebo `visual`. Hodnota `logical` znamená, že obsahem jednotlivých „článků“, tedy úseků popisujících jeden lexém, jsou XDXF elementy popisující přímo jednotlivé lingvistické údaje pro to určenými značkami, například <pos> pro popis slovního druhu (z anglického part-of-speech). Oproti tomu `visual` říká, že články budou popsány „natvrdo“ malou podmnožinou jazyka XHTML. Pro vizuální popis XDXF povoluje konkrétně tyto tagy: <sup>, <sub>, <i>, <b>, <tt>, <big>, <small>, <blockquote>.

Atribut `lang_from` popisuje, ze kterého jazyka jsou lexikální záznamy popsané v daném souboru, kdežto `lang_to` říká, ve kterém jazyce jsou psány definice, příklady, překlady a další doplňující informace o lexému. Hodnotami `lang_from` i `lang_to` jsou třípísmenné kódy jazyků dle ISO 639-2/B.

Další důležitá značka, o které se zmíníme, je <ar>, ta popisuje jeden článek (z anglického *article*), což odpovídá jednomu lexému. <xdxf> musí obsahovat jeden nebo více článků. V každém z nich se pak musí nacházet právě jedna značka <head> popisující lemma (z anglického *headword*), a poté libovolný počet definic (značka <def>), nebo značek <m>. Značka <m> (z anglického *morphological derivative*) obsahuje převážně různé gramatické kategorie (např. značky <pos>, <gender>, <number>, <case>, <mood>, pro rod, číslo, pád, slovní druh a způsob, respektive). Definice se do sebe mohou vnořovat. Stejně tak <m> může obsahovat i další <m>, což může být vhodné, jestliže chceme nějakou vlastnost pro více tvarů:

```
<?xml version="1.0" encoding="UTF-8"?>
<xdxf format="logical" lang_from="ENG" lang_to="CZE">
 <ar>
 <head>be</head>
 <m>
  <m>
   <tense>present</tense>
   <mood>indicative</mood>
   <m>
    <number>singular</number>
    <m><person>1</person><k>am</k></m>
    <m><person>2</person><k>are</k></m>
    <m><person>3</person><k>is</k></m>
   </m>
   <m>
    <number>plural</number>
    <m><person>1</person><k>are</k></m>
    <m><person>2</person><k>are</k></m>
```

```
    <m><person>3</person><k>are</k></m>
   </m>
  </m>
  <m>
   <tense>past</tense>
   <mood>subjunctive</mood>
   <k>were</k>
  </m>
 </m>
 </ar>
</xdxf>
```

Zajímavou vlastností XDXF je možnost využití tagů <representations> a <represent>, pomocí kterých můžeme hodnoty gramatických kategorií uvádět i v jiném jazyce, než v angličtině.

3.5 Open Lexicon Interchange Format

Open Lexicon Interchange Format, zkráceně OLIF, je otevřený formát pro výměnu lexikálních dat vzniklý a užívaný především v korporátní sféře. Podobně jako XDXF je založen na XML. Jedná se o formát, který se snaží vyčerpávajícím způsobem standardizovat způsob zaznamenání co nejvíce jazykových jevů. Nevyhýbá se ani tak konkrétním věcem, jako explicitní výčet vzorů ohýbání ohebných slov pro celou řadu jazyků. Hlavním sponzorem projektu OLIF je známá německá firma SAP a jeho vývoj započal již v devadesátých letech minulého století. Na oficiálních stránkách projektu OLIF je k dispozici ke stažení schema ve formátech DTD i XSD. [OLIF 2008a]

Mezi základní značky tohoto formátu patří <olif>, <header>, <body>, <entry>, <mono>, <language>, <ptOfSpeech>, <subjField> a další. <olif> je kořenová značka každého dokumentu, <head> obsahuje metainformace o celém souboru, jako například datum vzniku, počet záznamů (značka <entryCount>), velikost souboru v bajtech (značka <byteCount>, atribut Unit může ovlivnit jednotku – kb, mb...), vlastníka (značka <owner>) apod. Body pak obsahuje jednotlivé záznamy (značka <entry>). V každém entry pak mohou být například různé monolinguální informace jako slovní druh (značka <ptOfSpeech>), jazyk (značka <language>), obor (značka <subjField>) a podobně. Následuje malá ukázka.

```
<?xml version="1.0"?>
<!DOCTYPE olif SYSTEM "olif.dtd">
<olif OlifVersion="2.0 (July 2001)">
   <header CreaTool="XML spy" CreatToolVersion="3.5" OrigFormat="SYSTRAN dictionaries"
AdminLang="en" CreaDate="01-09-11" CreaId="lr">
      <fileDesc>
         <fileName>EN_FR_ES</fileName>
         <fileId>1</fileId>
         <fileExtent>
            <entryCount>21</entryCount>
            <termCount>21</termCount>
            <byteCount Unit="kb">18</byteCount>
         </fileExtent>
      </fileDesc>
      <publStmt>
         <distributor>
            <name>SYSTRAN</name>
            <address>1 rue du Cimetierre</address>
         </distributor>
         <owner>
            <name>SYSTRAN</name>
            <address>1 rue du Cimetierre</address>
         </owner>
         <availability Region="OLIF2 Consortium" PubStatus="restricted">for OLIF2
consortiumuse</availability>
         <date DateValue="ISO 8601">20010911T</date>
      </publStmt>
      <contentInfo>
         <quotMarkInfo QuotMarkRet="all" QuotMarkForm="unknown">un</quotMarkInfo>
         <langIdUse>region_standard</langIdUse>
      </contentInfo>
   </header>
   <body>
      <!--Monolingual description used in transfer entries -->
      <!--ENTRY 1 smooth-->
      <entry>
         <mono MonoUserId="" MonoUniversalId="">
            <keyDC KeyDCUserId="" KeyDCUniversalId="">
               <canForm>smooth</canForm>
               <language>EN</language>
               <ptOfSpeech>adj</ptOfSpeech>
               <subjField>general</subjField>
               <semReading/>
            </keyDC>
            <monoDC>
               <monoMorph>
                  <morphStruct/>
                  <number>sg</number>
                  <gender>m</gender>
                  <inflection>high</inflection>
               </monoMorph>
            </monoDC>
         </mono>
      </entry>
   </body>
</olif>
```

3.6 Lexical Markup Framework

Lexical Markup Framework (dále jen LMF) je ISO standard (ISO 24613:2008) pro ukládání a přenos lexikografických databází založený na formátu XML a na modelovacích technikách UML. Byl vyvinut týmem desítek lexikografů a pokrývá širokou řadu lingvistických aspektů od morfologie přes syntaxi až po sémantiku. Jedná se o poměrně obecný a komplexní systém pro vytváření konkrétnějších formátů, tedy o meta-formát. Obdobně jako XML je meta-formátem pro konkrétní aplikace s přesně specifikovaným DTD, případně XML Schematem, LMF definuje velice obecné DTD, které je potřeba pro další použití poněkud upřesnit.

3.6.1 Rozdělení informací mezi třídy a datové kategorie

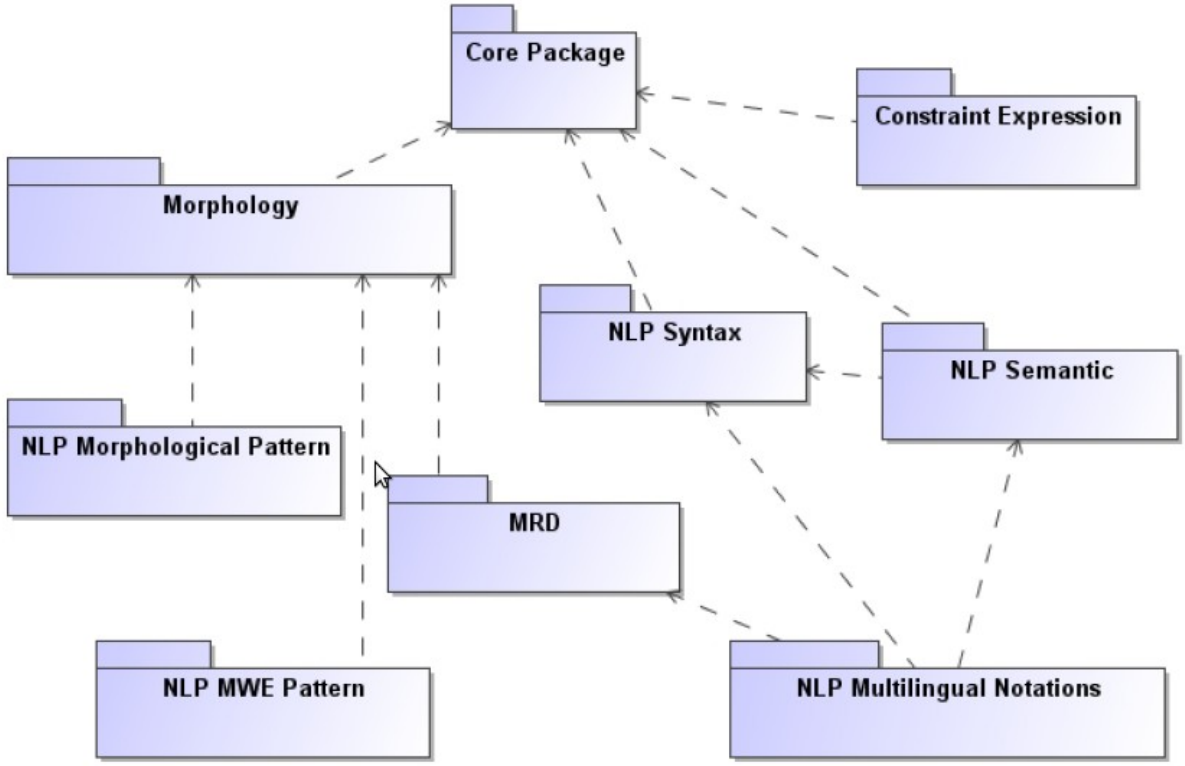
Při návrhu formátu si byli jeho autoři vědomi toho, že v lingvistice a v lexikografii neexistuje konečná množina druhů informací, které bychom chtěli o jednotlivých částech slovní zásoby zaznamenávat, neboť pro různé jazyky a pro různá použití je potřeba ukládat různé, často velmi specifické informace. Vytvoření takto uceleného souboru druhů informací by tedy nebylo v silách autorů tohoto formátu. Proto přišli s myšlenkou, že standard bude definovat pouze způsob uložení těch nejzákladnějších a nejobecnějších konceptů společných pro všechny jazyky a vztahů mezi nimi, jako je například lexikon, lexikální jednotka, význam, definice, překlad, a podobně. Těmto typům objektů pak odpovídají třídy, které při reprezentaci v podobě XML odpovídají jednotlivým značkám. Vztahům mezi značkami pak odpovídají XML atributy těchto značek.

Všechny ostatní specifičtější informace (slovní druh, pád, číslo, rod, četnost...) zprostředkovávají takzvané datové kategorie (anglicky data categories), v reprezentaci XML realizované značkami <feat> - z anglického *feature* - vztahujícími se k určité třídě pevně zadefinované standardem LMF. Značka <feat> vždy obsahuje atributy „att“ a „val“ (z anglického attribute a value) a reprezentuje vlastnosti dané instance třídy, ve které je obsažena. Na rozdíl od LMF tříd není definice sémantiky datových kategorií součástí standardu LMF (s výjimkou kategorie „language_coding“). Standard pouze odkazuje na externí databázi těchto kategorií, zvanou Data Category Registry (dále jen DCR) [ISOCAT 2009]. Ta pak obsahuje popis všech kategorií včetně autora, seznamu hodnot (jde-li o konečnou množinu), popisu významu v několika jazycích a různých dalších metadat. Koncept datových kategorií je detailně popsán v příbuzném standardu ISO 12620. Z DCR je možné načítat data pomocí protokolu zvaného RESTful, což je protokol podobným protokolům XML-RPC a JSON. Standard doporučuje používat datové kategorie uvedené v DCR a v případě použití vlastních kategorií doporučuje tyto nahrát zpět do DCR.

3.6.2 Koncept balíčků

Formát byl navržen na základě modelovacích konvencí UML definovaných skupinou OMG. To se projevilo především použitím konceptu „balíčků“ - souborů značek popisujících určitou oblast lexikografie. Jedná se konkrétně o tyto balíčky: Core, Morphology, MRD (Machine Readable Dictionary),

Syntax, Semantics, Multilingual Notations, Morphological Patterns, WME Patterns a Constraint Expression. Jednotlivé balíčky si nyní dále podrobněji popíšeme.



Obrázek 1: Závislostní vztahy mezi rozšiřujícími balíčky formátu LMF (převzato z [FRANCOPOULO a kol. 2008])

Tvůrce konkrétního LMF slovníku si pak může vybrat, které z těchto balíčků chce používat a které nikoli. Musí však při tom dodržet určitá omezení, neboť některé balíčky závisí na jiných balíčcích a jejich použití bez použití závisejícího balíčku by bylo nemožné nebo by nedávalo smysl. Závislosti balíčků reprezentuje následující obrázek.Základní balíček (core)
Základní balíček je jediný balíček, který musí být obsažen v každém souboru splňujícím standard LMF, neboť obsahuje některé prvky, které jsou nezbytné pro každou lexikografickou databázi. Konkrétně obsahuje tyto prvky: LexicalResource, GlobalInformation, Lexicon, LexicalEntry, Sense, Definition, Statement a TextRepresentation.

LexicalResource je kořenovým tagem každého LMF dokumentu a musí obsahovat právě jedno GlobalInformation a jeden nebo více Lexiconů. GlobalInformation obsahuje meta-informace vztahující se k celé databázi, například způsob kódování jazyků a písem. Způsob kódování jazyků je jako jediná z položek GlobalInfo povinná. Lexicon popisuje slovní zásobu jednoho konkrétního jazyka. Lexicon musí obsahovat minimálně jedno LexicalEntry. LexicalEntry obsahuje informace o jednom konkrétním lexému nebo lexikální jednotce. Každé LexicalEntry může obsahovat libovolný počet objektů Sense popisujících jednotlivé významy daného lexému. Sense pak může obsahovat libovolný počet značek Definition popisujících definice těchto významů. Každá definice pak může obsahovat ještě libovolný počet značek Statement, které obsahují popis upřesňující nebo doplňující definici. De-

finition i Statement mohou obsahovat libovolný počet značek TextRepresentation, což umožňuje psát definice a upřesnění v jazycích a písmech odlišných od jazyka lexému, který popisujeme.

Balíček Morphology

Cílem tohoto balíčku je umožnit detailní explicitní výčtový popis morfologie (tvarosloví) jednotlivých lexikálních jednotek. Tento balíček sice není povinný, ale ve skutečnosti existuje jen velmi málo slovníků, které jej nevyužívají. Důvodem je především to, že jednou ze značek v něm obsažených je značka Lemma, tedy slovníkový tvar. Standard celkem logicky stanovuje, že jeden lexém (značka LexicalEntry) nemůže obsahovat více lemmat. Dále obsahuje značku WordForm, která tvoří rámec pro ukládání různých tvarů stejného lexému, značku Stem, umožňující popis jednotlivých podčástí určitého alolexu, tedy morfů. Dále je zde obsažena značka RelatedForm pro popis souvisejících tvarů vzniklých například odvozením a značky ListOfComponents a Component pro popis víceslovných spojení (anglicky *Multi-word expression*). Jsou to taková slovní spojení, jejichž význam je odlišný od významu jednotlivých slov, tedy porušující princip kompozicionality. Příkladem může být třeba slovní spojení „chytat lelky“.

Ukázkový soubor tvořený pouze prvky definovanými v základním balíčku a v balíčku Morphology může vypadat například takto:

```
<LexicalResource dtdVersion="15">

    <GlobalInformation>
        <feat att="languageCoding" val="ISO 639-3"/>
    </GlobalInformation>
    <Lexicon>
        <feat att="language" val="eng"/>
        <LexicalEntry>
            <feat att="partOfSpeech" val="commonNoun"/>
            <Lemma>
                <feat att="writtenForm" val="clergyman"/>
            </Lemma>
            <WordForm>
                 <feat att="writtenForm" val="clergyman"/>
                 <feat att="grammaticalNumber" val="singular"/>
            </WordForm>
            <WordForm>
                <feat att="writtenForm" val="clergymen"/>
                <feat att="grammaticalNumber" val="plural"/>
            </WordForm>
        </LexicalEntry>
    </Lexicon>
</LexicalResource>
```

Balíček Machine Readable Dictionary

Jak již název napovídá, cílem tohoto balíčku je poskytnout prostředky pro tvorbu strojem čitelných slovníků ať již pro podporu překladatelů, nebo za účelem využití v komplexních NLP systémech (například pro extrakci pojmenovaných entit). Toto rozšíření patří k těm jednodušším a obsahuje pouze

tři značky: Equivalent, Context a SubjectField. Equivalent se používá ve dvojjazyčných, tedy překladových slovnících k označení protějšku v cílovém jazyce. Jeden význam (Sense) může obsahovat libovolný počet ekvivalentů. Context uvádí slovo s jeho typickými větnými „sousedy“, například pro slovo „práce“ ve významu literární dílo by mohl být uveden kontext „diplomová práce“ nebo pro slovo „platit“ ve významu finančně se vyrovnávat by mohly být kontexty „platit kreditní kartou“ a „platit v hotovosti“. Opět platí, že jeden význam může obsahovat více takovýchto ukázek kontextu. SubjectField pak určuje oblast, do které pojem spadá. Příkladem takovéto oblasti může být třeba chemie pro slovo oxidovat (ve svém původním významu). I SubjectField může být, poněkud nelogicky, obsažen libovolněkrát u jednoho významu.

Balíček Syntax

Cílem tohoto balíčku je popsat syntaktické vazby lexému k jiným lexémům, nikoli však obecný popis gramatiky daného jazyka. Základem je značka SyntacticBehaviour, jež popisuje jedno z možných „chování“ daného lexému v daném jazyce, typicky odkazem na některou z instancí tříd SubcategorizationFrame nebo SubcategorizationFrameSet. Ty umožňují sdílet popis určitého syntaktického chování pro všechny lexémy, které se syntakticky „chovají stejně“. Příkladem různých syntaktických chování (dokonce stejného lexému) může být chování lexému popsaného lemmatem „vařit“ ve větách „Karel vaří čaj.“ a „Voda vaří“. Typické použití je tedy například k označení, zda sloveso vyžaduje předmět, například sloveso „hodit“ předmět vyžaduje, zatímco sloveso „existovat“ nikoli.

Balíček Semantics

Účelem tohoto rozšíření je podrobněji popsat významy lexikálních jednotek jednoho jazyka a vztahy mezi nimi. Klíčovou značkou je zde Synset, což je množina sdílených významů v rámci jednoho jazyka. Jednotlivé značky Sense různých lexikálních jednotek se pak mohou odkazovat právě na tyto značky a sdílet tak například definice a překlady. Pro popis vzájemných vztahů mezi významy, jako je například synonymie, antonymie, holonymie, meronymie, hyperonymie, hyponymie a podobně je zde značka SenseRelation a pro popis vztahů mezi synsety značka SynsetRelation. První z nich je vhodné použít, nechceme-li synsety vůbec ukládat. V případě, že používáme synsety, je vhodnější použít SynsetRelation a mít tak vztah mezi významy sdílený napříč lexikálními jednotkami. Dále zde stojí za zmínku značka SenseExample pro uvedení konkrétního příkladu daného významu. Například pro význam „nevládní organizace“ by jeho SenseExample mohlo být například „Amnesty International“. Tato značka není povinná. A jako poslední zajímavou značku tohoto rozšíření si zde uvedeme MonolingualExternalRef, která se také váže ke značce *Sense* a obsahuje odkaz na externí systém pro popis významových konceptů, například na ontologii. I tato značka je nepovinná.

Balíček Multilingual notations

Cílem tohoto rozšíření je propojit informace mezi více lexikony různých jazyků. Jedná se konkrétněji například o propojení významů mezi jazyky nebo o propojení „syntaktických chování“. K propojení významů slouží značka SenseAxis. Propojení může být upřesněno značkou SenseAxisRelation a podmíněno podmínkami uvedenými ve značkách SourceTest a TargetTest. Dále je zde opět možnost odkazovat jednotlivé „osy významů“ na externí systémy pro popis konceptů značkou InterlingualExternalRef podobně jako jsme mohli odkazovat jednotlivé významy jednoho jazyka značkou MonolingualExternalRef. Jako poslední zmíníme značku TransferAxis, jež umožňuje propojit značky SyntacticBehaviour napříč jazyky. Následuje malá ukázka použití značek SenseAxis a SenseAxisRelation:

```
<SenseAxis id="SA1" senses="fra.riviere1 eng.river1">
    <SenseAxisRelation targets="SA2">
        <feat att="label" val="more general"/>
    </SenseAxisRelation>
</SenseAxis>
<SenseAxis id="SA2" senses="fra.fleuve1"/>
```

Balíček Morphological Patterns

Balíček slouží k popisu pravidel určujících postup, jak vytvářet různé nové tvary různých lexémů. Tato pravidla mohou popisovat ohýbání (tedy skloňování jmen a časování sloves), tvorbu slov odvozených (žák → žákovský) a tvorbu složenin (černobílý). To tvůrci lexikonu umožní popsat všechny tvary jednotlivých lemmat mnohem snadněji a za pomocí nesrovnatelně méně objemu dat. Balíček popisuje celkem pět různých způsobů popisu pravidel pro ohýbání, jeden pro odvozování a dva pro skládání. Jejich popisu věnuje standard téměř dvacet stránek, a proto zájemce pouze odkážeme na [FRANCOPOULO a kol. 2008].

Balíček NLP Multiword Expression Patterns

Balíček je určen pro popis vzorů, pomocí kterých lze tvořit víceslovná spojení. Jako příklad takového „parametrického“ slovního spojení můžeme uvést „věšet *někomu* bulíky na nos“. Parametrem je zde *komu*. K tomuto parametru můžeme definovat i určitá omezení. Například, že ten *někdo* musí být člověk.

Balíček Constraint Expression

Tento balíček umožňuje definovat omezení pro hodnoty atributů (vlastností) jednotlivých tříd (značek) – typicky vyjádřené značkou feat s atributy att a val nesoucími jméno vlastnosti a hodnotu dané vlastnosti respektive. Takže například pro slovník obsahující francouzská slova můžeme použít omezení vlastnosti gramaticalGender pouze na hodnoty *masculine* a *feminine*.

3.7 Výběr nejvhodnějšího formátu pro návrh a implementaci manažeru

Našimi požadavky pro výběr nejvhodnějšího formátu pro další zpracování budou strukturovanost, univerzálnost, rozšiřitelnost, přehlednost, zpracovatelnost s podporou již existujících standardních knihoven a status (návrh, předschválený návrh, schválený standard apod.). Naopak méně důležité pro nás bude, jestli se v tomto formátu vyskytuje velké množství slovníků, neboť toto kritérium by velice diskriminovalo formáty nové. Koneckonců převedení již existujícího slovníku do jiného formátu není neřešitelný problém a v rámci výzkumné skupiny zpracování přirozeného jazyka na naší fakultě již několik takových konvertorů vzniklo (viz kapitola 5.2).

Všechny formáty spadající do první skupiny, tedy skupiny jednoduchých textových formátů, můžeme rovnou vyřadit, neboť nesplňují požadavek na strukturovanost, rozšiřitelnost a univerzálnost. Stardict formát je poměrně pružný, neboť umožňuje definovat nové datové položky, nicméně nesplňuje požadavek na zpracovatelnost s podporou již existujících standardních knihoven a navíc téměř vůbec nedefinuje, jak by se měly zaznamenávat jednotlivé lexikografické položky, ale zaměřuje se více na možnosti vzhledu hesel. To je velice nevhodné pro možnosti dalšího automatického zpracování. Formát Wordnetu, tak jak je definován oficiálních stránkách projektu, neodpovídá prakticky žádnému z našich kritérií.

Jak se tedy zdá, pro naše účely poslouží nejlépe formáty třetí skupiny, tedy ty založené na XML. Všechny z nich automaticky splňují strukturovanost a zpracovatelnost s podporou existujícch standardních knihoven. Bylo třeba se tedy rozhodovat mezi formáty XDXF, OLIF a LMF. Co se týče rozsahu druhů uložitelných informací, jedná se o poměrně dost podobné formáty. Všechny nabízejí možnost uložit metainformace o souboru, všechny umožňují ukládat běžné gramatické kategorie slov jako například slovní druh, rod, číslo, vzor a podobně. LMF navíc umožňuje definovat vlastní atributy a i předepisuje, jak tyto vlastní atributy sdílet s ostatními (Data Category Registry). Co se týče statusu, tak XDXF je dodnes (19. 5. 2011) ve stádiu konceptu. Přesto však pro něj vznikají slovníky, především pro mobilní aplikace. Formát OLIF dospěl již do verze 2.1.1. Nejdále se však dostal formát LMF, který byl schválen jako standard ISO. Jeho specifikace je dostupná na oficiálních stránkách formátu i včetně vysvětlujících textů a ukázek. Pro formát LMF také mluví to, že na serveru výzkumné skupiny zpracování přirozeného jazyka vzniklo v rámci jiných projektů poměrně velké množství souborů v tomto formátu, převážně převodem z různých jiných více či méně strukturovaných formátu, a je potřeba tyto lexikony vhodně spojit. Formátem, pro který navrhneme naši aplikaci, bude tedy nakonec Lexical Markup Framework.

4 Specifikace požadavků, analýza, návrh a implementace

4.1 Specifikace požadavků

Nyní si popíšeme požadavky, které byly stanoveny pro vývoj naší aplikace. Základním požadavkem bylo automatické spojování více slovníků a získávání statistik o slovnících. Dalším z požadavků bylo konzolové i grafické uživatelské rozhraní, neboť konzolové lze využít při automatizaci úloh skriptováním a grafické se zase lépe hodí pro méně zkušeného uživatele. Dále jsme chtěli, aby šlo pracovat jak s obyčejnými LMF dokumenty, tak i dokumenty komprimovanými. V neposlední řadě jsem chtěl, aby klíčové funkce aplikace byly pokryty testy.

4.2 Analýza

Ve fázi analýzy bylo potřeba vybrat vhodný programovací jazyk a knihovnu pro grafické uživatelské prostředí (dále jen GUI). Jako programovací jazyk jsem si zvolil Python, a to hned z několika důvodů. Za prvé je jeho referenční implementace multiplatformní, takže uživatel není nucen používat nějaký konkrétní operační systém. Dále je dostatečně vysokoúrovňový, takže není problém pracovat například s datovým typem unicode nebo s formátem XML přímo pomocí zabudovaných knihoven a funkcí. V neposlední řadě také hrála roli jeho přehlednost a jednoduchost (z pohledu uživatele). Jako knihovnu pro GUI jsem si vybral PyGTK, opět pro svou jednoduchost a velké rozšíření napříč platformami.

4.3 Návrh

Ve fázi návrhu bylo potřeba program rozdělit do tříd tak, aby byl co nejlogičtější, nejpřehlednější a nejrozšiřitelnější, a vymezit vzájemné vztahy mezi objekty těchto tříd. Dále bylo mým požadavkem, aby byla logika aplikace co nejvíce oddělena od uživatelského rozhraní, tedy aby většina tříd nepřistupovala přímo k uživatelskému rozhraní, ale aby pouze používaly objekt odkazující se na konkrétní UI. To nám umožní snadno podporovat jak textové, tak grafické rozhraní a v případě grafického rozhraní snadno přidat podporu i pro jiné než námi vybrané PyGTK. V neposlední řadě bylo třeba se rozhodnout pro určitý typ zpracování XML souborů.

4.3.1 Rozdělení do tříd

Rozdělení do tříd se stane velice intuitivním ve chvíli, kdy si uvědomíme, že jednotlivé konkrétní XML značky libovolného LMF dokumentu v podstatě popisují jednotlivé konkrétní objekty. Představíme-li si pak LMF soubor jako databázi objektů, jedná se pak ve skutečnosti o objektově relační mapování. Pro každou významnou značku tedy vytvoříme jednu třídu – ideálně pak poděděnou od nějaké naší abstraktní třídy, která bude obsahovat metody společné pro všechny potomky (nebo alespoň jejich deklarace). Toto rozhodnutí vytvářet pro jednotlivé značky uživatelské třídy je naprosto klíčové pro škálovatelnost. V první iteraci vývoje, tedy v jakémsi prototypu, jsem zkoušel provádět operace spojování přímo nad objekty DOM stromu bez kompletního parsování jednotlivých informací do členských proměnných vlastních tříd, což se neosvědčilo, neboť to vedlo k velice neudržovatelnému a jen těžko rozšiřitelnému kódu a i triviální algoritmy nad jednotlivými položkami bylo komplikované naimplementovat.

4.3.2 Vztahy mezi objekty tříd

Nyní již v tomto směru zbývají pouze pouze dvě závažná rozhodnutí. Prvním z nich je reprezentace vztahu mezi značkou a vnořenou značkou. Tomuto konceptu se v objektově orientovaném programování nejvíce blíží vztah agregace, tedy obsažení. Vnořené značky $b_1 \ldots b_n$ značky *a* tedy budou členskými objekty objektu odpovídajícího elementu *a*.

Druhým rozhodnutím je pak způsob reprezentace typu atributu IDREF, tedy reference na jiný element, obsahující atribut typu ID. V jazyce C++ by tomuto zcela jistě odpovídal typ ukazatel, případně reference. U té bychom dokonce měli implicitně zajištěno, že se v době inicializace musí odkazovat na existující objekt. V Pythonu můžeme využít velice příjemné vlastnosti operátoru přiřazení, a to té, že v případě přiřazení existujícího objektu do nové proměnné se typicky vytvoří pouze právě reference na již existující objekt. Čestnou výjimku tvoří objekty, které jsou instancí třídy označené dekorátorem *immutable*, například objekty typu string. Výchozí typ uživatelských tříd je však *mutable*, čehož budeme v této souvislosti také využívat.

4.3.3 Způsob parsování XML souborů

Obecně existují dva základní způsoby práce s XML soubory. Prvním z nich je SAX (z anglického *Simple API for XML*), což je událostmi řízené parsování „za letu“. To znamená, že uživatel SAX knihovny/balíku nadefinuje „callbacky“, jež se budou volat ve chvíli, kdy parser narazí na nějakou určitou značku, nebo konec značky, a spustí parsování. Parser pak jen v definovaných okamžicích volá uživatelem definované funkce a vše ostatní je již na programátorovi. Výhodou tohoto způsobu práce s XML je paměťová nenáročnost. To se může hodit například jestliže potřebujeme zpracovávat větší soubor, než kolik je k dispozici operační paměti. Nevýhodou je pak pracnost psaní jednotlivých metod, neboť si musíme stále uchovávat kontext, v jaké části dokumentu zrovna jsme (jeden element

se může vyskytovat uvnitř různých rozdílných elementů). Z toho pak také plyne určitá nepřehlednost výsledného kódu.

Druhou metodou práce s XML je Document Object Model, dále jen DOM. Scénář použití DOM vypadá přibližně tak, že uživatel DOM knihovny/balíku předloží řetězec nebo textový soubor a knihovna mu vrátí kompletně vyparsovaný XML strom. XML strom je strom, jehož uzly jsou instancemi některé z tříd poděděné od typu Node. Těmito třídami jsou Document, Element, Attr, Comment, Entity, EntityReference, ProcessingInstruction, DocumentType, CDATASection a Notation [CHAMPION et al. 1998]. Kromě toho umožňuje voláním příslušných metod upravovat již existující DOM strom (to se využívá například v ECMAscriptu při práci s obsahem HTML stránky), nebo dokonce sestavit zbrusu nový DOM strom. Nevýhodou tohoto způsobu je však nutnost načíst celý dokument do paměti, což může být pro velké dokumenty pomalé (může dojít ke swapování obsahu paměti na disk), nicméně s dostatečně velkým odkládacím prostorem nikoli nemožné.

Právě pro jeho univerzalitu a vzhledem k tomu, že jako jeden z požadavků na aplikaci jsem si stanovil rozšiřitelnost, jsem si vybral jako hlavní nástroj pro práci s XML metodu DOM. Bral jsem mimo jiné v úvahu, že velikost operační paměti pracovních stanic a serverů se v čase zvětšuje téměř exponenciálně a také to, že se jedná o aplikaci, kde rozhodující není rychlost zpracování dané operace, ale především její kvalita. V neposlední řadě můžeme na obhajobu tohoto rozhodnutí říci to, že například při spojování většího množství slovníků roste celková velikost výsledného souboru pomaleji, než součet velikostí jednotlivých vstupů, neboť tyto slovníky jsou typicky velmi duplicitní. Konkrétně jsem si vybral nejprve standardní balíček v Pythonu pro práci s DOM s názvem `xml.dom.minidom`. Ten je dostupný ve všech verzích Pythonu série 2.x a vyšší. Při jeho používání se však ukázalo, že jeho paměťová náročnost je opravdu obrovská (viz kapitola 5.2), a proto jsem časem přešel na `xml.etree.cElementTree`, neboť je implementován jako rozšíření psané v C++ a je o několik řádů paměťově i časově výhodnější. `xml.etree.cElementTree` se stal součástí standardní distribuce Pythonu ve verzi 2.5 a jeho API je velice podobné API `xml.dom.minidom`, avšak o něco pohodlnější a navíc umožňuje dotazování pomocí jazyka XPath. Nakonec jsem objevil knihovnu `lxml`, která je zpětně zcela kompatibilní s `xml.etree.cElementTree` a navíc obsahuje metodu pro přehledné formátování XML při konverzi zpět na text. Tato knihovna je dokonce i o něco rychlejší než standardní pythoní implementace `xml.etree.cElementTree`.

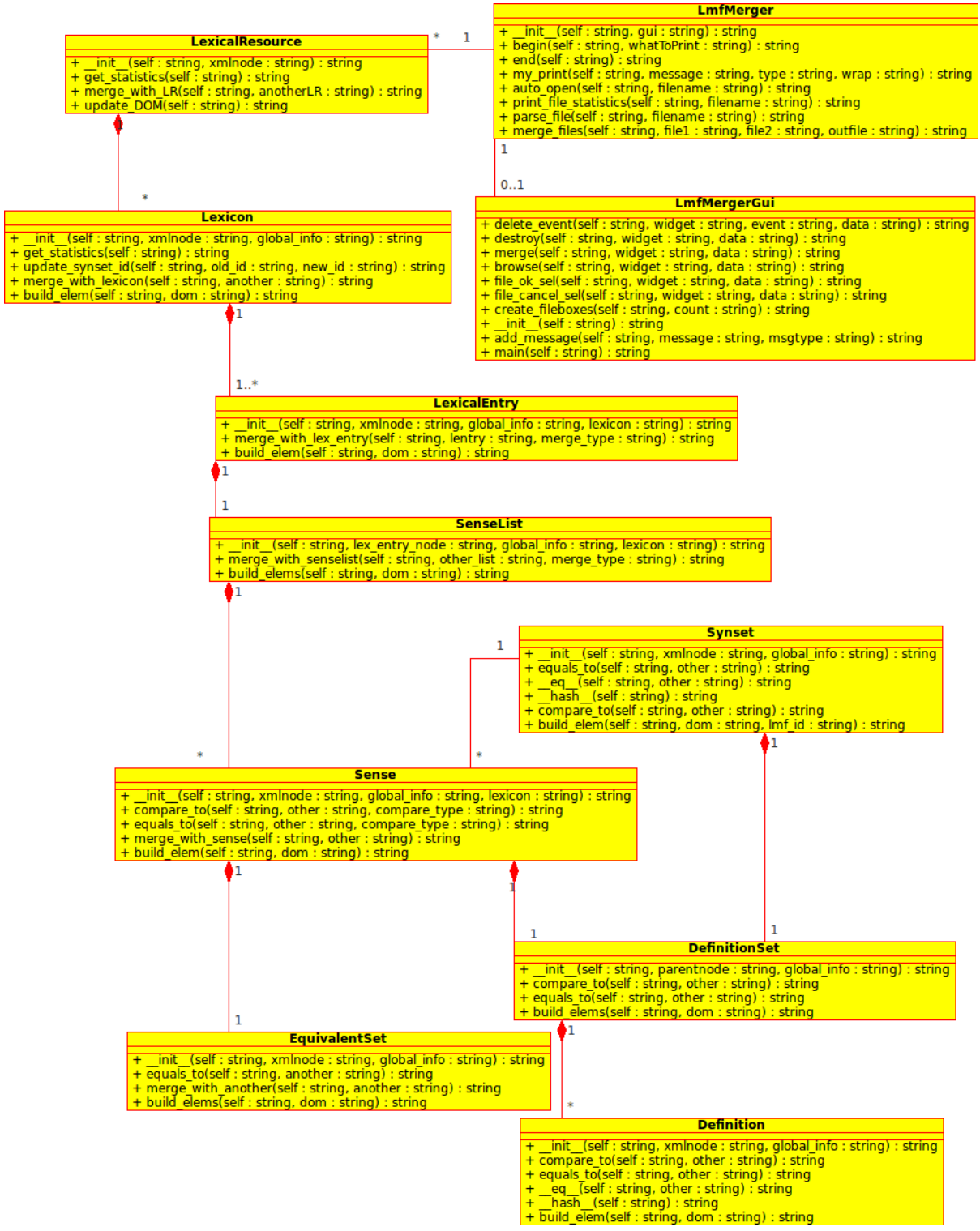
4.4 Implementace

V této části si popíšeme jednotlivé moduly a třídy a seznámíme se s problémy, které bylo nutné řešit v době implementace. Ještě předtím se však seznámíme se stylem psaní kódu, který jsem se rozhodl používat a nebude ani chybět jeho zdůvodnění.

4.4.1 Styl psaní kódu

Každý programátor má sice svůj styl psaní kódu, jakýsi svůj „rukopis“, nicméně v praxi se ukázalo, že je výhodné v rámci organizace, nebo alespoň v rámci jednoho projektu, dodržovat určitý standard pro psaní zdrojových textů, takzvaný *coding style*. Tato aplikace je sice momentálně (letní semestr 2011) „one-man-show“, nicméně vzhledem k její open source licenci a hostování na serveru Google Code je zde určitý potenciál pro to, že by se časem na vývoji mohlo podílet více vývojářů. Ani další úpravy ze strany členů Výzkumné skupiny zpracování přirozeného jazyka nejsou vyloučeny. Proto bylo i v rámci tohoto projektu potřeba si zvolit vhodný styl zápisu, který bude důvěrně známý i budoucím vývojářům, a ten následně dodržovat.

Při výběru coding style jsem se řídil doporučením tvůrců Pythonu popsaným v [ROSUM a kol.]. Tento výběr není samozřejmostí, neboť například ve firmě, ve které pracuji, používáme pro Python poněkud odlišný standard. Dle standardu tvůrců Pythonu tedy odsazuji čtyřmi mezerami, snažím se nepřekračovat délku řádku 79 znaků, udržuji pevný počet dvou prázdných řádků k oddělení metod, dodržuji umístění mezer mezi výrazy, komentáře píši anglicky a celými větami, používám dokumentační komentáře (tzv. *docstringy*). Dále soubory, funkce a metody pojmenovávám malými písmeny s podtržítky, kdežto třídy pak pomocí tzv. *CamelCase* (tedy první písmena slov velká). Názvy všech identifikátorů, stejně jako komentáře, píši zásadně anglicky.



Obrázek 2: Diagram tříd

4.4.2 Třída LmfMerger

Jedná se o třídu, která tvoří jádro aplikace. Obsahuje mimo jiné metody `merge_files` a `print_file_statistics`. Její konstruktor má parametr `ui`, čímž lze předat referenci na objekt ui, který obsahuje metody pro práci s s uživatelským rozhraním. Získání statistik o souborech, ani spojení dvou souborů nedělá sama, ale využívá k tomu metod instancí třídy LexicalResource (viz kapitola 4.4.4). Metody `merge_files` a `print_file_statistics` přijímají jako parametr jméno souboru (jména souborů) a na základě nich pomocí metody `parse_file` vytvoří instance třídy LexicalResource. Metoda `parse_file` volá metodu `auto_open`, která abstrahuje rozdíl mezi komprimovanými a nekomprimovanými soubory, neboť podle koncovky určí, zda soubor otevřít normálně, a nebo pomocí balíčku `gzip`.

4.4.3 Třídy LmfManagerGui a LmfMergerThread

Třída LmfMergerGui zajišťuje vše ohledně grafického uživatelského rozhraní. Nyní podporuje pouze knihovnu widgetů PyGTK, ale při implementaci jsem se snažil, aby stejné API šlo použít i pro jiné knihovny. Protože některé operace při spojování mohou trvat delší dobu a po tu dobu musí být aplikace stále připravena na příkazy uživatele, používá třídu LmfMergerThread, jejíž název již napovídá, že půjde o třídu reprezentující vlákno. Třída LmfMergerThread se stará o vlastní výpočetně náročné operace (vytváří instanci třídy LmfMerger a volá její metody) a poskytuje zpětnou vazbu uživatelskému rozhraní v podobě takzvaných „callbacků“, čili funkcí, které se zavolají, dojde – li k určité události. V tom spočívá oddělení uživatelského rozhraní. Třída LmfMergerThread dostane pouze parametr gui, jehož typ nekontroluje ani netestuje, pouze volá jeho metody. Jestliže tedy parametr gui nebude instancí LmfMergerGui, ale třeba nějaké hypotetické třídy LmfMergerQtGui obsahující uživatelské rozhraní implementované widgety z knihovny QT, avšak bude obsahovat všechny potřebné metody, třída LmfMergerThread ji bude používat stejným způsobem.

4.4.4 Třída LexicalResource

Tato třída se stará o práci s celým LMF souborem. Parametrem konstruktoru je kořenový element v reprezentaci DOM. Při konstrukci objektu dané třídy se postupně načítá vybraný obsah objektového modelu dokumentu do členských proměnných. Načítání jednotlivých elementů však nedělá sama, ale zpracování některých elementů deleguje na třídy k tomu určené (vytváří a ukládá si jejich instance). Konkrétně se jedná o třídu Lexicon. Připomeňme, že jeden soubor může obsahovat více lexikonů. Ještě před lexikony však načítá GlobalInfo, jež pak předává konstruktorům lexikonů. To je důležité například pro správné vyparsování kódů jazyků k následnému převedení na společný formát. Jako společný formát jsem vybral ISO 639-2/T, neboť obsahuje kódy 464 jazyků včetně jazyků umělých (Esperanto, Ido apod.) a mrtvých (např. latina). Lexikony si pak ukládá do asociativního pole (v Pythonu zvaného dictionary) dle kódu jazyka jimi popisovaného.

Statistiky jednotlivých lexikonů opět nepočítá přímo, ale deleguje je na příslušnou metodu třídy Lexicon a jednotlivé výsledky již pouze agreguje (např. průměrný počet lexikálních záznamů na lexikon, celkový počet lexikálních záznamů a podobně).

Spojování s jinou instancí této třídy řeší postupným spojováním jednotlivých lexikonů. Obsahuje-li již lexikon se stejným jazykovým kódem, zavolá na něm metodu pro obohacení o jiný lexikon, jinak si jej normálně přidá. Opět tu využíváme toho, že nedojde ke kopírování celé instance třídy Lexicon do asociativního pole, ale pouze k přiřazení reference na tento objekt.

V neposlední řadě obsahuje tato třída metodu `update_dom` pro obrácenou transformaci, než jakou jsme prováděli v konstruktoru, tedy převod naší interní reprezentace dokumentu zpět do reprezentace DOM pomocí vytváření elementů a atributů. Tento převod opět z velké části deleguje na objekty jiných tříd v sobě obsažené a pouze spojuje jejich výstupy.

4.4.5 Třída Lexicon

Třída Lexicon se zabývá prací s lexikony, tedy množinami lexémů, synsetů a dalších přidružených informací popisujících jeden konkrétní jazyk. Opět obsahuje konstruktor, tentokrát ve formě __init__(self, xmlnode, global_info), který uloží DOM podstromu elementu xmlnode do členských proměnných dané instance třídy. Konkrétně se jedná o jazyk, synsety a obsahy značek LexicalEntry, tedy lexémy. Jazyk dekóduje na základě hodnoty proměnné `language_coding` schované v asociativním poli `global_info` a převede si jej do reprezentace ISO639-2/T. Načítání lexémů deleguje na třídu LexicalEntry. Její instance pak ukládá do asociativního pole asociativních polí, nejprve dle slovního druhu, poté podle psané formy lemmatu. Takže přístup k lexému reprezentovanému lemmatem „book“ ve významu „kniha“ uloženému v instanci třídy Lexicon pojmenované `lexicon1` by vypadal následovně:

```
lexicon1.lex_entries['noun']['book']
```

Tento způsob uložení jsem si vybral, protože při spojování dvou lexémů, přesněji řečeno při doplňování chybějících významů druhého lexému do prvního, je vhodné postupovat pouze v rámci slovního druhu (dva významy nemohou být shodné, jestliže se neshodují slovním druhem). Dále, jestliže je daný lexikon obsahuje, načítá konstruktor synsety. Ukládá je opět do asociativního pole dle jejich ID a opět jejich načítání deleguje na konstruktor třídy Synset. Poslední zajímavá věc, která se provádí v konstruktoru této třídy, je pokus o detekci, zda se jedná o slovník výkladový, a nebo překladový, případně jejich kombinaci. To využijeme opět ve fázi spojování slovníků.

Dále tato třída obsahuje metodu pro získání statistik o daném lexikonu. Do statistik lexikonu patří například počet lexémů pro jednotlivé slovní druhy. Poté obsahuje metodu pro spojení s obsahem jiné instance této třídy `merge_with_another`. Ta se skládá ze dvou hlavních činností – spojení synsetů a spojení lexikálních záznamů.. Implementace první z nich by byla poměrně triviální, kdybychom nemuseli brát v úvahu jednak kolize identifikátorů synsetů, tedy případy, kdy stejný identifikátor synsetu ve dvou slovnících popisuje jiný význam, a jednak, že dva různé identifikátory mohou v různých souborech popisovat obsahově stejný synset. Oba tyto případy je potřeba jak detekovat, tak i napravit. Pro řešení obou těchto problémů bylo potřeba přidat třídě Synset (viz níže) me-

todu pro porovnání s jiným synsetem na základě jejich obsahu. Algoritmus přidávání synsetů zdrojového lexikonu do cílového lexikonu pak vypadá následovně. Procházíme postupně všechny synsety zdrojového lexikonu a pro každý děláme tyto kroky:

1. Postupně porovnáme se všemi synsety cílového lexikonu a uložíme si, zdali jsme našli (obsahovou) shodu. Shodou zde nemyslíme pouze úplnou rovnost obsahu, ale i podobnost převyšující určitou hranici.
2. Jestliže ano, přiřadíme synsetu ve zdrojovém lexikonu identifikátor shodného synsetu slovníku cílového a ve zdrojovém lexikonu aktualizujeme všechny odkazy původní identifikátor na novou hodnotu.
3. Jestliže jsme žádnou shodu dle obsahu nenašli, ověříme, zda se již identifikátor synsetu ze zdrojového slovníku nepoužívá. Jestliže ne, můžeme do cílového slovníku rovnou uložit zkoumaný synset. Jestliže ano, je třeba nejprve vygenerovat pro daný synset nový vhodný identifikátor, aktualizovat všechny odkazy ve zdrojovém lexikonu a teprve poté synset vložit do cílového slovníku.

Teprve nyní, když máme vyřešenou synchronizaci identifikátorů synsetů, můžeme začít přidávat jednotlivé lexikální záznamy. Jejich spojování je jednodušší v tom, že nemusíme spojovat a ukládat identifikátory, neboť žádné z námi vybraných rozšíření je nepoužívá. V případě pozdějšího použití by byla implementace jejich spojování prakticky identická. Způsob přidávání vlastních lexikálních jednotek z jednoho lexikonu do druhého je také podobná jako přidávání synsetů, dokonce ještě jednodušší.

1. Procházíme postupně přes všechny slovní druhy zdrojového slovníku.
2. Nevyskytuje-li se daný slovní druh v cílovém slovníku, přidáme tam všechny lexikální jednotky daného slovního druhu. Tato operace je v Pythonu opět levná, poněvadž se obsah tohoto pole vůbec nekopíruje, pouze se uloží reference.
3. Je-li daný slovní druh i v cílovém slovníku (toto je mnohem častější varianta než předchozí), procházíme jednotlivé lexikální jednotky tohoto slovního druhu ve zdrojovém slovníku a:
    1. Nenachází-li se v cílovém slovníku lexikální jednotka se stejnou hodnotou *writtenForm*, rovnou ji tam přidáme.
    2. Nachází-li se tam, zavoláme na tomto objektu jeho metodu `merge_with_another`. Tím veškeré detaily spojování informací k této lexikální jednotce delegujeme na třídu LexicalEntry

I tato třída obsahuje metodu `build_elem` pro konverzi svých členských proměnných (tedy převážně asociativních polí synsetů a lexikálních jednotek) do uzlů DOM stromu, která je pak volána ze stejnojmenné metody třídy LexicalResource a která ke své činnosti využívá stejnojmenné metody tříd LexicalEntry a Synset. Těm předává jako parametr uzel rodiče, ke kterému se mají vytvořené elementy připojit.

4.4.6 Třída LexicalEntry

Objekty této třídy reprezentují jednotlivé lexikální jednotky daného jazyka popsané stejnojmennou značkou v XML reprezentaci metaformátu LMF. Třída v konstruktoru obdrží odpovídající uzel DOM stromu, sadu globálních nastavení global_info a referenci na lexicon, do kterého patří. Ze synovských elementů se pak pokusí vybrat především slovní druh, lemma a jednotlivé významy. Pro načtení jednotlivých významů použije konstruktor třídy SenseList.

Pro spojení svého obsahu s obsahem jiné instance obsahuje tato třída opět metodu `merge_with_another`, která již však tentokrát neobsahuje jako parametr pouze zdrojový objekt, ze kterého se budou data přidávat, ale i proměnnou `merge_type` udávající, zda se má spojovat na základě obsahu překladů (hodnota BY_EQUIVALENT), a nebo na základě výkladu (hodnota BY_DEFINITION). Pro toto řešení jsem se rozhodl, neboť to, že zrovna tyto dvě instance obsahují nebo neobsahují definici respektive překlad, nemusí vypovídat nic o tom, zda se jedná o slovník výkladový respektive překladový. Vlastní spojování pak spočívá pouze v delegování této činnosti na stejnojmennou metodu třídy SenseList (včetně parametru merge_type). Zde by se mohlo zdát, že celá třída SenseList je zbytečná a tuto činnost by mohla vykonávat rovnou třída LexicalEntry. Zde je ale potřeba zmínit, že s přibývající funkčností třídy LexicalEntry by se tato stala nepřehlednou (dle standardu může obsahovat kromě značek Sense dalších sedm různých typů objektů) a tato technika nám umožní udržet složitost této třídy stále pod kontrolou.

Ani zde nechybí metoda `build_elem`, jež vyparsované a sjednocené informace (lemma, slovní druh a významy) převede zpět do podoby DOM elementu LexicalEntry a patřičných synovských elementů.

4.4.7 Třída SenseList

Instance této třídy obsahuje množinu významů jednoho konkrétního lexému nebo lexikální jednotky. V konstruktoru obdrží uzel DOM stromu odpovídající elementu LexicalEntry, pole nastavení GlobalInfo a referenci na lexikon, ve kterém se popisovaná lexikální jednotka nachází. Pro daný uzel LexicalEntry najde všechny synovské elementy Sense, postupně na ně volá konstruktor třídy Sense a vytvořené objekty ukládá do členské proměnné (seznamu).

Dále obsahuje metodu `merge_with_senselist` pro spojení s jiným seznamem významů. Ta jako parametr obdrží nejen zdrojový seznam, ale i typ spojování popsaný u třídy LexicalEntry. Zde v závislosti na předaném typu spojování dochází k následujícímu:

1. V případě varianty BY_DEFINITION se každý význam zdrojového seznamu postupně porovná s významy cílového seznamu, dokud není nalezena shoda dostatečně vysoká podobnost. K tomu se volá metoda třídy Sense nazvaná `compare_to`. Nebyla-li nalezena žádná shoda či vysoká podobnost, přidá se význam do cílového seznamu významů.
2. V případě vyrianty BY_EQUIVALENT je situace velmi podobná, liší se pouze tím, že jestliže byla shoda nalezena, dochází ke spojení obsahu významu ze zdrojového seznamu s významem z cílového seznamu. Je tomu tak proto, že u definic pracujeme jen s

jednou definicí, a tudíž je-li nalezena shoda, není již dále co spojovat, zatímco u překladů může být shoda v jednom cílovém jazyce a pak chceme do cílového významu doplnit i překlady do případných dalších jazyků.

Ke konverzi spojených dat zpět do DOM stromu je zde metoda `build_elems`, která vrátí seznam XML elementů Sense naplněných obsahem patřičných objektů. K vytvoření těchto elementů volá metodu `build_elem` těchto objektů.

4.4.8 Třída Sense

Tato třída se stará o práci s jedním významem určité lexikální jednotky. V konstruktoru obdrží XML uzel elementu Sense odpovídající určitému významu, pole globálních nastavení a referenci na lexikon, ve kterém se nachází. Tato reference sem „probublala“ skrz volání konstruktorů tříd LexicalEntry a SenseList a nyní konečně najde své uplatnění. Značka Sense v případě výkladového slovníku a při použití rozšiřujícího balíčku Semantics nemusí obsahovat definice přímo, ale může odkazovat na značku Synset, která je obsahuje. Proto i při načítání definic musíme zkontrolovat, zda nemá daná instance Sense připojen synset a načíst pak definice z něj. Synsety jsou však uloženy v instanci třídy Lexicon, které je daná lexikální jednotka a její významy součástí, a proto si jej musíme do konstruktoru takto poslat. Dále, jestliže je dokument obsahuje, načítá také protějšky významů z jiných jazyků v podobě elementů Equivalent. Definice i ekvivalenty načítá voláním konstruktorů tříd DefinitionSet a EquivalentSet respektive.

Pro porovnávání míry podobnosti jsou zde metody `compare_to` a `equals_to`. První z nich vrací míru podobnosti daného významu s významem v argumentu a druhá pouze vrací, zda mají či nemají stejný obsah. Ke své činnosti využívají metody `equals_to` a `compare_to` tříd EquivalentSet a Definition. Dále je zde metoda `merge_with_another` pro spojení s obsahem jiné instance třídy sense, pochopitelně se stejným nebo podobným významem. Rozhodnutí o podobnosti se děje ve stejnojmenné metodě třídy SenseList na základě použití zmíněných metod třídy Sense `equals_to` a `compare_to`.

Opět nechybí metoda `build_elem` pro vytvoření patřičných uzlů reprezentace DOM, tedy o vytvoření elementu Sense a buď jeho případné naplnění atributem synset_id, nebo elementy Definition. Dále doplní případné značky Equivalent. K těmto činnostem používá metodu `build_elems` tříd DefinitionSet a EquivalentSet.

4.4.9 Třída Synset

Tato třída slouží k reprezentaci a zpracování obsahu stejnojmenného elementu z rozšíření Semantics. V konstruktoru dojde k načtení všech definic pomocí volání konstruktoru třídy DefinitionSet a k uložení identifikátoru synsetu (atribut id). Dále obsahuje metodu pro zjištění shody obsahu `equals_to` a metodu `compare_to` prozjištění míry podobnosti. Obě ke své činnosti využívají stejnojmennou metodu třídy DefinitionSet. A konečně zde nechybí metoda `build_elem` pro vytvoření elementu Synset včetně všech jeho atributů a podelementů.

4.4.10 Třída DefinitionSet

Třída slouží k uložení, porovnávání a spojování seznamu definic uložených u jednoho konkrétního významu (tj. elementu Sense) nebo synsetu. V konstruktoru třída obdrží element, ve kterém jsou obsaženy definice (tedy Sense nebo Synset) a ty vyparsuje voláním konstruktoru třídy Definition a poukládá do množiny. Dále obsahuje metodu `compare_to`, která pracuje tak, že pro každou definici daného množiny definic z cílového souboru najde definici z množiny definic zdrojového souboru, která je jí nejpodobnější (k tomu používá stejnojmennou metodu třídy Definition) a zapamatuje si tuto podobnost. Následně zapamatované podobnosti zprůměruje a vrátí jako celkovou podobnost. Jedná se tedy vlastně o aritetický průměr maxim podobností.

4.4.11 Třída Definition

Tato třída se stará o práci s definicemi, tedy o načtení definice z objektového modelu XML dokumentu, o případné porovnání (zjištění míry podobnosti) s jinou definicí a o opětovný převod do uzlu reprezentace DOM. Jedna instance třídy Definition popisuje právě jednu definici. V konstruktoru se do členské proměnné text uloží obsah stejnojmenné „feature“. Porovnání probíhá tak, že se obě definice rozdělí na slova a z těchto dvou množin slov se vypočte průnik (jazyk Python pro tuto činnost poskytuje přímo kontejner `set` a operátor průniku „&“). Výsledná podobnost se pak spočte jako podíl počtu společných slov průměrným počtem slov první a druhé definice.

$$similarity = \frac{|WORDS1 \cap WORDS2|}{\frac{|WORDS1| + |WORDS2|}{2}}$$

Protože nechceme odstraňovat pouze podobné definice při spojování, ale i identické definice při načítání, definuje tato třída také metody `__eq__a __hash__`, které zajistí, že při přidávání do množiny se přidají pouze instance, jejichž text se v množině ještě nevyskytuje. Opět nechybí metoda build_elem, která obsah dané instance převede zpět do uzlu reprezentace DOM.

4.4.12 Třída EquivalentSet

Tato třída je zodpovědná za práci s množinami překladů popsaných značkou Equivalent z rozšiřujícího balíčku Machine Readable Dictionary. V konstruktoru třída obdrží uzel reprezentace DOM odpovídající elementu Sense a asociativní pole globálních nastavení lexikálního zdroje global_info. Konstruktor najde všechny poduzly odpovídající značce Equiv a pro každý z nich načte jazyk a psanou podobu překladu. Při načítání jazyka použije položku language_encoding z globálního nastavení pro správné dekódování cílového jazyka překladu. Tento jazyk pak překóduje do ISO639-3. Takto získanou dvojici (jazyk, ekvivalent v daném jazyce) uloží do množiny překladů. Konkrétně pro uložené těchto dvojic používám datový typ NamedTuple z balíku collections. Díky tomu se nemusím starat o správnou implementaci metod `__eq__a __hash__` pro správnou funkci množinových operací nad množinami těchto dvojic.

Dále je zde metoda equals_to pro porovnání, zda se jedná o stejný překlad, tedy pravděpodobně o stejný význam. Její implementace je založena na předpokladu, že shoduje-li se text překladu alespoň do jednoho jazyka pro dané dva EquivalentSety (a tedy i pro dva Sense), shodují se pak i tyto významy. Pozná-li se při spojování významů na základě výše zmíněné funkce shoda dvou významů a je-li tedy potřeba připojit zbylé překlady ze zdrojového významu do cílového, použije se metoda merge_with_another třídy EquivalentSet, která právě využívá snadnosti aplikace množinových operátorů na výše zmíněné dvojice a pouze provede operaci sjednocení těchto množin.

Opět ani zde nechybí metoda build_elem pro zpětný převod obsahu instance do elementů reprezentace DOM.

4.4.13 Modul language_coding.py

Všechny výše uvedené třídy byly uloženy v samostatných stejnojmenných modulech, tedy souborech, pouze s tou změnou, že název souboru je vždy složen z malých písmen a slova jsou oddělena podtržítky, tak jak je definováno v [ROSUM a kol.]. Tento modul je jedním ze dvou modulů, který neobsahuje stejnojmennou třídu, ale pouze soubor pomocných užitečných funkcí.

language_coding.py se konkrétně stará o převod mezi různými kódováními jazyka. Ke své činnosti používá balíček pycountry [THEUNE 2010], který sice není součástí standardní knihovny Pythonu, nicméně je dostupný ke stažení z oficiálního katalogu (převážně) volně dostupných balíčků pro Python zvaném PyPI (z anglického Python Package Index) a také jako balíčky v repozitářích distribucí Debian [SIP 2011] a Ubuntu [CANONICAL 2011]. Použití tohoto balíku již implikuje množinu podporovaných kódování. Těmi jsou ISO 639-1, ISO 639-2 a anglický název daného jazyka. Kromě různých kódování jazyků poskytuje také kódování písem dle ISO 15924, kódování měn dle ISO 4217 a kódování států a jejich regionů dle ISO 3166 a ISO 3166-2 respektive.

4.4.14 Modul lmf_tools.py

Tento modul obsahuje pomocné funkce, které usnadňují implementaci většiny výše uvedených tříd reprezentujících elementy dokumentu ve formátu LMF. První funkcí, kterou si zde popíšeme, je funkce get_new_id. Funkce se používá při řešení kolize identifikátorů k nalezení identifikátoru, který bude obsahovat všechny znaky z původního identifikátoru, ale přesto nebude kolidovat s žádným jiným. Funkce tento problém řeší tak, že zkouší za původní identifikátor doplnit řetězec „_coll“ (jako *collision*) a celé číslo počínaje jedničkou. Zjistí-li, že takový identifikátor již existuje (málo pravděpodobné, avšak nikoli nemožné), inkrementuje dané číslo o jedničku a opět ověří, zda tento identifikátor neexistuje. Toto provádí tak dlouho, dokud nenarazí na neobsazený identifikátor.

Další potřebnou funkcí je zde funkce get_words, která zajišťuje jednoduchou tokenizaci, tedy rozdělení textu na slova. Ta je potřeba například při určování podobnosti definic. Funkce postupuje tak, že nejprve nahradí všechny nealfanumerické znaky mezerou (bez tohoto kroku by slova, vedle kterých se vyskytuje například tečka, čárka nebo závorka, obsahovala i tyto speciální znaky), poté

vzniklý řetězec převede na malá písmena (velikost písmen zde nehraje prakticky žádnou roli) a nakonec text rozdělí na tokeny (v podstatě slova) tak, že tokenem bude vše, co se nachází mezi dvěma sousedními „bílými znaky“ a samo není bílý znak (v Pythonu metoda `string.split()`).

5 Testování

Součástí vývoje každého softwarového projektu by zcela jistě mělo být i testování a ani náš projekt nebude výjimkou. V této kapitole si přiblížíme nejprve unit testy, které se snaží pokrýt veškerou důležitou funkcionalitu našeho programu, a dále se seznámíme s průběhem a výsledky testování na reálných datech.

5.1 Pokrytí aplikace jednotkovými testy

Jednotkovým testem (anglicky *unit test*) rozumíme program nebo skript, jehož úkolem je ověřit funkčnost nějaké malé a pokud možno samostatné jednotky zdrojového kódu. Typicky se jedná o test třídy, ideálně pak všech jejích metod. Vzhledem k tomu, že naše aplikace je vyvíjena v Pythonu, jenž neposkytuje možnost statické kontroly datových typů vracených jednotlivými funkcemi, neboť se jedná o dynamicky typovaný jazyk, je obzvláště důležité otestovat co nejvíce případů, a především, jak se daná část kódu zachová, nedostane-li očekávanou hodnotu. Dalším dobrým důvodem psaní těchto testů je snadné ověření, že změna existujícího kódu či doplnění kódu nového nezpůsobily nefunkčnost některých jiných již existujících funkcionalit. Jestliže pak tedy po spuštění všech testů není zaznamenána žádná chyba nebo nesrovnalost, můžeme garantovat jistou minimální funkčnost našeho projektu.

V rámci testování mé aplikace jsem se však rozhodl ne pro čistě jednotkové testy, ale spíše pro jakousi kombinaci *grey box testingu* a *fuctional testingu*. *Grey box testing* znamená, že máme určitou představu o „vnitřnostech“ naší aplikace, abychom věděli, na co se máme přibližně zaměřit, avšak na rozdíl od *white box testingu* netestujeme „jednotlivé řádky kódu“, ale spíše z pohledu konečného uživatele. *Functional testing* pak znamená, že testujeme pouze funkcionalitu a žádné jiné vlastnosti softwaru (např. rychlost, bezpečnost, přístupnost a podobně). Později se však ukázalo, že testování rychlosti a paměťové náročnosti je také užitečné, a proto bylo do sady testů také doplněno.

5.1.1 Způsob psaní a spouštění testů

Pro hromadné spouštění všech testovacích případů využívám jednoduchý skript `runtests.sh` napsaný v Bashi, umístěný v adresáři `test`. Ten postupně prochází všechny podadresáře adresáře `test` začínající řetězcem `test-` a v něm buďto spustí testem definovaný skript `./runtest.sh` (způsob vyhodnocení správnosti výstupu programu je pak již plně v rukou daného skriptu), nebo spustí spojení souborů `file1.xml` a `file2.xml` a výsledek uloží do souboru `merged.xml`. Ten pak porovná s již existujícím a člověkem ověřeným souborem `merged_correct.xml`. Nejsou-li tyto soubory stejné, vypíše rozdíl a ukončí testování.

5.1.2 Jednotlivé testy

Nyní si popíšeme jednotlivé testovací případy, které se vyskytují v našem adresáři pro testy.

test-01-pos

Cílem tohoto testu bylo ověřit hned několik věcí. Jednak, že se při spojování dvou lexikonů stejného jazyka správně oddělují významy stejného lemmatu dle jednotlivých slovních druhů, dále, že dva odlišné významy stejného slovního druhu stejného lemmatu zůstanou odlišné a konečně, že nevznikají duplicitní významy v případě shody v lemmatu, slovním druhu a významu. Dále test ověřuje, že se jazyk slovníku správně převání na společné kódování jazyka, neboť v prvním souboru je jazyk v kódování ISO 639-1, kdežto ve druhém v kódování ISO 639-2.

test-02-translational

Předchozí test ověřoval správnost spojování slovníků výkladových, Tento se pro změnu zaměřuje na stejné jevy u slovníků překladových. Testuje se zde především porovnávání významů za základě překladu a obecně správnost implementace rozšíření LMF pro strojově čitelné slovníky (MRD), konkrétněji pak implementace třídy Equivalent pro práci s překlady.

test-03-synsets

Tento test se věnuje kontrole správnosti implementace spojování synsetů a řešení kolizí identifikátorů, ke kterým při tomto spojování dochází. První soubor obsahuje dvě lexikální jednotky, „quick“ a „fast“, jejichž významy se odkazují na synset s identifikátorem „SS1“ a s definicí „moving to a long distance in a short time“. Druhý soubor obsahuje lexikální jednotku „rapid“ odkazující se na synset s identifikátorem „SS2“ s velmi podobnou definicí „moving to long distances in a short time“. Cílem je ověřit, že program vyhodnotí synset SS1 z prvního souboru a synset SS2 z druhého souboru za totožný, zachová jim společný identifikátor a nastaví odkazy na všech tří lexikálních jednotek na synsety tak, aby byly konzistentní s výsledným identifikátorem spojeného synsetu.

test-05-statistics

Tento test ověřuje, zda se správně počítají a vypisují statistické informace o jednotlivých souborech.

test-06-gzip

Test ověřuje, zda je program schopen správně otevřít i komprimovaný soubor. Jedná se v podstatě o kopii předešlého testu s tím rozdílem, že vstupní soubor je zkomprimován metodou gzip.

5.2 Testování na reálných datech

Na školním serveru minerva1 vyhrazeném pro výzkumnou skupinu zpracování přirozeného jazyka se v adresáři projektů nachází adresáře dicts2lmf, dicts2lmf2, dicts2lmf3 a dicts2lmf4, ve kterých se nacházejí LMF slovníky vzniklé z konverzí z různých papírových a elektronických formátů v rámci projektů jiných studentů. Ty si nyní blíže představíme.

dicts2lmf

Jedná se o adresář projektu, jehož cílem byl převod různých komerčních slovníků z formátu OLIF do LMF. Autory jsou Milan Bartoš a Jakub Kadlas a Martin Skalický.

dicts2lmf2

V tomto adresáři se nacházejí soubory ve formátu AC a skripty sloužící pro převod do LMF. Autorem skriptů je Ing. Kamil Dudka a konvertor je postaven na technologii SAX.

dicts2lmf3

Tento adresář obsahuje další soubory ve formátu AC a skripty v jazyce Perl pro převod do LMF. Autorem skriptů je Filip Kocina.

dicts2lmf4

Cílem projektu nacházejícím se v tomto adresáři je převod slovníků z binárního formátu EX do LMF. Autorem skriptů je student Peter Pač.

Teprve při testování na těchto reálných slovnících jsem narazil na některé problémy, které se v mých jednotkových testech nevyskytovaly. Mezi ně patřila například již zmíněná časová a především paměťová složitost balíčku `xml.dom.minidom`, jež vedla k jeho nahrazení balíčkem `xml.etree.cElementTree` a nakonec knihovnou lxml. Dalším problémem bylo, že některé soubory uložené na školním serveru měly špatně uvedené languageCoding. Namísto kódu jazyka dle ISO 639 obsahovaly název znakové sady. To jsem vyřešil autodetekcí na základě délky kódu jazyka.

test-07-real_data_translational

Tento test je prvním z testů, který používá opravdová data vytvořená na fakultě. Testuje především časovou náročnost zpracování souborů o velikosti řádově megabajtů (nekomprimovaných). Konkrétně se jedná o tyto soubory:

- dicts2lmf2/out/ac_zem_xxx_lmf.xml
- dicts2lmf2/out/ac_obecny_xxx_lmf.xml

6 Závěr

Cílem mé diplomové práce bylo vytvořit aplikaci pro spojování slovníků ve formátu LMF a zjišťování informací o daných slovnících a tuto aplikaci otestovat. Tento cíl se mi podařilo splnit. Výsledná aplikace je schopna sama určit zda daný objekt spojit s jiným objektem, a nebo spíše objekt přidat, tak jak je. Objekty jsou zde míněny lexikony, lexikální jednotky, významy, překlady, definice a synonymické řady. Aplikace je schopna pracovat jak se LMF soubory komprimovanými, tak s nekomprimovanými. Dále je schopna částečně odstraňovat duplicity i v rámci jednoho souboru. Aplikace je navržena a naimplementována tak, aby mohla být dále rozšiřována, neboť se sestává spíše z více menších a nezávislých tříd a modulů než mála monolitických celků. Kromě toho jsem vytvořil jednoduchý nástroj pro převod LMF slovníků do XHTML a plain textu pomocí XSL transformací a programu html2txt.

Na aplikaci plánuji dále pracovat. Plánovanými rozšířeními jsou například prohlížení slovníku včetně vyhledávání, doplnění plně automatického režimu o režim s podporou uživatele (dojde ke dramatickému zvýšení přesnosti), podpora dalších LMF balíčků a značek, online komunikace s Data Category Registry pomocí RESTful API, přidání konfiguračního souboru ukládání nastavení (nyní jsou přímo v aplikaci jako konstanty na začátku jednotlivých modulů), možnost výběru způsobu řazení a podobně.

Aplikace je hostována na serveru Google Code [SAMIA 2010] a licencována pod svobodnou licencí GNU GPL, což jí poskytuje možnost dalšího kolaborativního vývoje – jak ze strany studentů FIT, tak třetími stranami. Odkaz na moji aplikaci byl také uveřejněn na oficiálních stránkách formátu Lexical Markup Framework [FRANCOPOULO 2008].

Literatura

[CANONICAL 2011] Canonical, ltd. *Ubuntu Packages Search* [online]. [cit. 2011-05-11]. Ubuntu -- Package Search Results -- python-pycountry. Dostupné z WWW: <http://packages.ubuntu.com/search?keywords=python-pycountry&searchon=names&suite=all&section=all>

[CHAMPION et al. 1998] CHAMPION, Mike; BYRNE, Steve; NICOL Gavin; WOOD, Lauren. *World Wide Web Consorcium (W3C)* [online]. 1998 [cit. 2011-05-16]. Document Object Model (Core) Level 1. Dostupné z WWW: <http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/level-one-core.html>

[FRANCOPOULO 2008] FRANCOPOULO, Gil. *Lexical Markup Framework* [online]. 2008 [cit. 2011-05-12]. Main page. Dostupné z WWW: <http://lexicalmarkupframework.org/>

[FRANCOPOULO a kol. 2008] FRANCOPOULO, Gil; GEORGE, Monte. *Lexical Markup Framework* [online]. 2008 [cit. 2011-02-23]. Language resource management — Lexical markup framework (LMF). Dostupné z WWW: <http://www.tagmatica.fr/lmf/iso_tc37_sc4_n453_rev16_FDIS_24613_LMF.pdf>

[GARCÍA 2007] Ivan García. *Babiloo - Google Code* [online]. 2007 [cit. 2011-04-16]. Format for StarDict dictionary files. Dostupné z WWW: <http://code.google.com/p/babiloo/wiki/StarDict_format>

[ISOCAT 2009] Max Planck institute. [online]. 2009 [cit. 2011-05-11]. Data Category Registry. Dostupné z WWW: <http://www.isocat.org>

[OLIF 2008a] OLIF Consorcium. *OLIF - Home* [online]. 2008 [cit. 2011-05-15]. OLIF Home. Dostupné z WWW: <http://www.olif.net/>

[ROSUM a kol.] ROSUM, Guido van; WARSHAW, Barry. *Python Programming Language - Official Website* [online]. 2001 [cit. 2011-03-27]. PEP 8 -- Style Guide for Python Code. Dostupné z WWW: <http://www.python.org/dev/peps/pep-0008/>

[SAMIA 2010] SAMIA, Michel, Bc. *Google Code* [online]. 2011 [cit. 2011-05-12]. LMF merger. Dostupné z WWW: <http://code.google.com/p/lmf-merger/>

[SIGNOV 2006] SIGNOV, Sergej. *XDXF.revdanica.com* [online]. 2006 [cit. 2011-05-16]. XDXF Manual. Dostupné z WWW: <http://xdxf.revdanica.com/drafts/logical/05a/XDXF_manual.html>

[SIP 2011] Software in the Public Interest, Inc. *Debian packages* [online]. 2011 [cit. 2011-05-11]. Debian -- Details of package python-pycountry in squeeze. Dostupné z WWW: <http://packages.debian.org/squeeze/python-pycountry>

[SVOBODA 2004a] SVOBODA, Milan. *GNU/FDL Anglicko - Český slovník* [online]. 2004 [cit. 2009-12-04]. GNU/FDL Anglicko-Český slovník - stažení. Dostupné z WWW: <http://slovnik.zcu.cz/download.php>

[SVOBODA 2004b] SVOBODA, Milan. *GNU/FDL Anglicko - Český slovník* [online]. 2004 [cit. 2009-12-04]. GNU/FDL Anglicko-Český slovník - formát dat. Dostupné z WWW: <http://slovnik.zcu.cz/format.php>

[THEUNE 2010] THEUNE, Cristian. *Python Package Index* [online]. 2010 [cit. 2011-05-18]. pycountry 0.12.1. Dostupné z WWW: <http://pypi.python.org/pypi/pycountry/>

[WIKIPEDIA 2009d] Přispěvatelé Wikipedie. *Wikipedie: Otevřená encyklopedie* [online]. 2009 [cit. 2009-06-20]. Lexém. Dostupné z WWW: <http://cs.wikipedia.org/w/index.php?title=Lex%C3%A9m&oldid=4087421>

[WIKIPEDIA 2011a] Wikipedia contributors. *Wikipedia, The Free Encyclopedia.* [online]. 2011 [cit. 2011-05-17]. XML database. Dostupné z WWW: <http://en.wikipedia.org/w/index.php?title=XML_database&oldid=426227862>

[WIKIPEDIA 2011b] Wikipedia contributors. *Wikipedia, The Free Encyclopedia.* [online]. 2011 [cit.]. Sedna (database). Dostupné z WWW: <http://en.wikipedia.org/w/index.php?title=Sedna_(database)&oldid=391165718>

[WIKIPEDIA 2011c] Přispěvatelé Wikipedie. *Wikipedia, The Free Encyclopedia* [online]. 2011 [cit.]. Lexical item. Dostupné z WWW: <http://en.wikipedia.org/w/index.php?title=Lexical_item&oldid=425418551>

[WORDNET 2003a] Wordnet authors. *WordNet: An Electronic Lexical Database* [online]. 2003 [cit. 2009-11-30]. WordNet lexicographer files. Dostupné z WWW: <http://wordnet.princeton.edu/wordnet/man/wninput.5WN.html>

[WORDNET 2003b] Wordnet Authors. *WordNet: An Electronic Lexical Database* [online]. 2003 [cit. 2009-11-30]. Grind and input data. Dostupné z WWW: <http://wordnetcode.princeton.edu/3.0/WNgrind-3.0.tar.gz>

Seznam příloh

Příloha 1: CD s implementovanou aplikací a datovými soubory