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

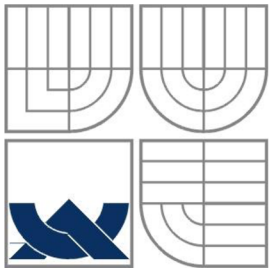
NÁSTROJ PRO ÚPRAVY A POROVNÁVÁNÍ SLOVNÍKŮ VE FORMÁTU LMF

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

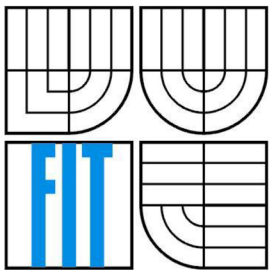
AUTOR PRÁCE
AUTHOR

MAREK FEŠAR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

NÁSTROJ PRO ÚPRAVY A POROVNÁVÁNÍ SLOVNÍKŮ VE FORMÁTU LMF

A TOOL FOR EDITING AND COMPARING LMF DICTIONARIES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Marek Fešar

VEDOUCÍ PRÁCE
SUPERVISOR

doc. RNDr. Pavel Smrž, Ph.D.

BRNO 2012

Zadání bakalářské práce/14104/2011/xfesar00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2011/2012

Zadání bakalářské práce

Řešitel: **Fešar Marek**
Obor: Informační technologie
Téma: **Nástroj pro úpravy a porovnávání slovníků ve formátu LMF**
A Tool for Editing and Comparing LMF Dictionaries
Kategorie: Databáze

Pokyny:

1. Seznamte se s databází Oracle Berkley XML a s nástroji pro vyhledávání (XQuery, XPath).
2. Prostudujte specifikaci normy LMF.
3. Na základě existujících řešení navrhněte a realizujte systém pro hromadnou editaci a porovnávání slovníků ve formátu LMF.
4. Diskutujte výhody a nevýhody daného řešení na příkladech ukázkových slovníků.
5. Vytvořte dokumentaci k realizovanému systému.

Literatura:

- podle doporučení vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Funkční prototyp řešení

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

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% 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: 1. listopadu 2011
Datum odevzdání: 16. května 2012

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 seznamuje čtenáře s možnostmi slovníkových formátů a zaměřuje se na principy a sémantiku uložení dat ve formátu Lexical Markup Framework. Zvláštní pozornost je věnována vazbě na registr datových kategorií, který specifikuje obsah elementů LMF.

V další části bude zmíněn databázový systém Oracle Berkeley pro ukládání slovníkových dat a další práci s nimi – adresování částí slovníku, získávání požadovaných informací a vzájemné porovnávání různých slovníků. Specifickou oblastí je hromadná změna informací ve slovnících obsažených. Pro tento účel se slovníková data podle druhu obsažené informace agregují a uživatel je může změnit.

Důraz je kladen také na usnadnění úprav slovníků pomocí systému nápovědy, který uživatele seznamuje s očekávanými vlastnostmi slovníku.

Abstract

The diploma thesis presents dictionary formats and focuses on principles and semantics of data storage using Lexical Markup Framework. Contains information on relationship between the framework and Data Category Registry, which is part of the ISO standard for further dictionary data specification.

After that, the Oracle Berkeley database system for XML document storage is mentioned followed by approaches for addressing documents and its portions. Methods for dictionary comparison are being described as well as grouping and changing dictionary content.

Help on the process of dictionary modification must be provided to the user in the way of showing expected features, relevant to the selected element.

Klíčová slova

Lexical Markup Framework, LMF, XML, XQuery, XPath, Berkeley DB, editor, slovník

Keywords

Lexical Markup Framework, LMF, XML, XQuery, XPath, Berkeley DB, editor, dictionary

Citace

Fešar Marek: Nástroj pro úpravy a porovnávání slovníků ve formátu LMF, bakalářská práce, Brno, FIT VUT v Brně, 2012

Nástroj pro úpravy a porovnávání slovníků ve formátu LMF

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doc. RNDr. Pavla Smrže, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Marek Fešar
31. 3. 2012

Poděkování

Rád bych poděkoval docentu Smržovi za pomoc při uskutečňování svého záměru v podobě realizace nástroje pro práci se slovníky.

Obsah

1 Úvod

Při zpracování nejrůznějších forem slovníků, ať již se jedná o překladové, výkladové či frazeologické slovníky, je často potřeba pracovat s velkým množstvím informací v těchto dílech obsažených a dosáhnout konzistentního zaznamenání stejného druhu informace napříč celým slovníkem. V současné době k tomu slouží nástroje pro psaní slovníků (Dictionary Writing System), které se však zaměřují zejména na zpracování jednotlivých hesel a jejich grafické podoby, než na slovník jako celek.

Ve slovnících může vyvstat potřeba změnit některou obsaženou informaci na více místech, na což se ve své práci zaměřuji. Mým cílem je vytvoření nástroje, který umožní uživateli hromadně měnit data obsažená v rozsáhlých slovnících formátu LMF. Půjde o seskupování stejného druhu informací, vyskytujících se napříč slovníkem, v závislosti na nadřazeném elementu.

Druhá část aplikace se zabývá porovnáváním slovníků navzájem a hledáním chybějících informací. Uplatňuje se více hledisek jako srovnání obsažených překladů u vícejazyčných slovníků či vyhledávání chybějících slov z definic ve výkladových slovnících.

Opomíjena není ani nápověda, která je uživateli poskytována v průběhu provádění hromadných změn ve slovníku. Záměrem je sdělit uživateli, jaký druh informace by měl být v tom kterém elementu zahrnut.

2 Koncepce uložení slovníku

Jak již bylo nastíněno v úvodu, ač pracujeme pouze s elektronickým vyjádřením slovníků, přicházíme do styku s více různými formáty. Zde nastává první obtíž v práci se slovníky – k formátům je potřeba nejen různě přistupovat, ale také interpretovat jejich obsah. Dále se vzájemně mezi sebou liší ve vyjadřovacích schopnostech.

Řekněme, že bychom navrhovali formát dvojjazyčného překladového slovníku. V nejzákladnější podobě bychom chtěli zachytit původní slovo a jeho ekvivalent ve druhém jazyce. Postačovalo by nám tedy na jeden řádek vždy zaznamenat slovo a za něj jeho překlad.

Takovýto jednoduchý slovník bychom uchovávali například jako prostý textový soubor, jenž by se snadno upravoval libovolným textovým editorem.

Co když budeme chtít do slovníku zanést nový druh informace, třeba v jakém kontextu se slovo obvykle používá? Na to náš formát zachycení informací nepamatuje, a zde bychom tedy naráželi na první obtíž (a to by nás ve vývoji tlačilo dále, snažili bychom se data lépe strukturovat, např. uvést kontext do závorky za překlad atp.).

2.1 Prostý textový formát

Prakticky používané formáty slovníků prošly vývojem do značné míry ovlivněným účelem použití. Na začátku si vystačily s výše užitým prostým textovým vyjádřením. Později přibyla potřeba informace lépe strukturovat.

2.1.1 GNU/FDL Anglicko-Český slovník

Českým zástupcem ukládání slovníkových dat v prostém textovém formátu je GNU/FDL Anglicko-Český slovník, jehož autorem je Milan Svoboda, alespoň co se formátu zachycených dat týká. Samotný obsah slovníku je utvářen více autory. Sám zakladatel formát popisuje následovně [1]:

```
anglické slovo [TAB] české slovo [TAB] poznámky [TAB] speciální poznámky
[TAB] jméno překladatele
```

Každý záznam je veden na separátním řádku. Má-li slovo více různých překladů, je tato skutečnost vyjádřena více řádky. Zde je patrná jistá redundance dat, neboť jedno anglické slovo, čítající například 5 českých ekvivalentů, bude ve slovníku uvedeno 5x, pokaždé ve spojitosti se svým ekvivalentem.

Do poznámky lze zachytit obor, pro který je daný překlad typický, nebo slovní druh (možnost kombinace více různých hodnot).

Speciální poznámka dovoluje podrobněji popsat výraz, například na úrovni výkladového slovníku. Autor dále uvádí, že pro výslovnosti anglických slov využívá separátního souboru ve tvaru každého řádku:

```
anglické slovo [TAB] výslovnost
```

Absence možnosti navázat konkrétní výslovnost ke konkrétnímu překladu může při tvorbě slovníku být znatelnou nevýhodou (např. slovo *tear* může podle výslovnosti mít význam trhat /t-air/

či plakat /t-ear/), dále rozdělení jednoho slovníku do více souborů vede k nutnosti data na sebe nějakým způsobem navázat při prohlížení i editaci, vytrácí se tedy jednoduchost prostého editování člověkem.

2.2 Strukturovaný textový formát

Strukturovaný textový formát přináší oproti prostému textovému větší variabilitu v obsažených informacích. Nabízí možnost vynechávat informace, které nejsou pro dané slovo relevantní (poznámka, oblast užití) nebo hierarchicky zanořovat, a tím například přiřadit více překladů jednomu heslu, bez potřeby heslo opakovat.

Oblíbeným prostředkem pro vyjádření hierarchie se stal značkovací jazyk XML, respektive určitá jeho podmnožina vymezená pomocí Definice Typu Dokumentu či XML Schema.

2.2.1 XML Dictionary Exchange Format

Formát určený pro výměnu slovníkových dat, založený na XML. Kromě toho však definuje v návrhu svého standardu [2] i další vlastnosti, jako je umístění každého slovníku do svého adresáře či doplnění slovníků o vlastní ikony.

Kořenový element slovníku (*xdxf*) musí obsahovat tři povinné atributy, a sice *lang_from, lang_to* a *format*. U prvních dvou se jedná o kódování jazyka v ISO 639-2, formát je buď *visual* (slovníky určené pro zobrazení slovníkovými programy, bez vynechávání mezer či konců řádků) nebo *logical* (slovníky bez grafického ztvárnění).

Uzly, které stojí za zmínku, jsou například *full_name* se jménem slovníku, *description* se stručným popisem, licencí či uvedením vlastníka autorských práv a *abbreviations* vymezující zkratky použité ve slovníku.

Obsah hesel je vždy obalen elementem pro článek (*ar)* obsahujícím heslo (*k*), slovní druh (*pos*), přepis výslovnosti (*tr*), definici samotného hesla a případné odkazy na jiná hesla (*kref*) nebo multimediální obsah (*rref;* například audio soubor s výslovností) V závislosti na definovaných zkratkách v úvodu slovníku lze využívat v jednotlivých heslech také odkazy na ně (*abr*). Nechybí ani značky určené pro příklady (*ex*) a komentáře autora (*co*).

Pro slovníky ve formátu *visual*, tedy určené přímo pro prezentaci bez dalších grafických úprav, je definována podmnožina XHTML značek. Jde o elementy pro text v dolním a horním indexu (*sub, sup*), kurzívu (*i*), tučné písmo (*b*), strojopis (*tt*), zvětšené a zmenšené písmo (*big, small*) a citaci (*blockquote*).

2.2.2 Lexical Markup Framework

Formát jazykových zdrojů LMF je ISO standard (24613:2008) pro zpracování přirozeného jazyka (NLP – Natural Language Processing) a strojově čitelný slovník (MRD – Machine Readable Dictionary). Právě standardizace dává tvůrcům aplikací záruku stálosti. Bohužel LMF ve snaze, aby vyhověl nejrůznějším formám jazykových zdrojů (slovníků), je ve své podstatě velmi široký.

Lexical Markup Framework jakožto aplikace XML je definován pomocí DTD, v současné chvíli ve verzi 16. Tato definuje přibližně 65 různých elementů s případnými atributy. LMF je modulární, obsahuje celkem 9 balíčků (jádro, morfologii, strojové zpracování (MRD), syntaxi, sémantiku, vícejazyčné notace, morfologické vzory, vzory pro víceslovné výrazy (MWE) a vyjádření omezení). Na rozdíl od předchozího formátu se nezabývá vizuální stránkou, ale pouze strukturou obsahu.

Každý XML dokument obsahuje kořenový element, kterým je zde *LexicalResource*. Jako jistou hlavičku lze chápat element *GlobalInformation*, který obsahuje pouze a jenom elementy *feat*, tedy vlastnosti (ty jsou hlavním nositelem informací v celém slovníku).

Následuje lexikon (*Lexicon)*, reprezentující slovník, kterých může být v dokumentu obsaženo více, například u překladového slovníku jeden pro každý jazyk, ze kterého se překládá. Každé heslo je vymezeno zapouzdřujícím elementem (*LexicalEntry)*, jehož povinnou součástí je právě jedno lemma (*Lemma)*. Běžně se vyskytuje ještě alternativní slovní tvar daného hesla (*WordForm)*, určený pro zaznamenání nepravidelných tvarů.

Další obsah *LexicalEntry* uzlu je už pak spíše věcí druhu slovníku, u překladových se využívá význam (*Sense)* se zanořeným překladem (*Equivalent)*, výkladové pak spíše budou obsahovat definice (*Definition)*. Elementy neobsahují informace přímo ve svých atributech, ale využívají k tomu vždy zanořené vlastnosti (*feat*), mající právě dva atributy – jméno vlastosti (*att*) a hodnotu (*val*).

2.3 Záznam dat

Popsané formáty slovníkových dat mohou být elektronicky zaznamenány různými způsoby, jejichž klady a zápory v této části krátce shrnu. Jedná se o záznam do textového souboru, relační databáze a XML databáze.

2.3.1 Textový soubor

Zřejmě nejjednodušší druh záznamu informací. K jeho vytváření není potřeba žádného speciálního nástroje, prakticky každý operační systém nabízí program pro psaní textových souborů. Vzniklý soubor je lidsky čitelný a bezproblémově přenositelný. Neexistuje žádná vazba na další součásti systému a programy, lze jej proto uložit na libovolné záznamové médium či poslat po síti dalším uživatelům. Textový soubor také potřebuje minimální režii pro své uložení. Vyjma struktur daných souborovým systémem nezabírá více místa, než kolik skutečně odpovídá uživatelem zadaným informacím.

Jeden z prvních záporů se projeví po zapsání většího množství informací. Soubor s rostoucí délkou znesnadňuje orientaci v zaznamenaných informacích a vyhledávání v něm se stává pomalejší. Je to dáno charakterem přístupu, kdy je třeba sekvenčně projít všechna obsažená data a v nich nalézt požadované heslo. Za nedostatek lze považovat také nutnost při každé změně zapsat celý soubor znovu.

2.3.2 Relační databáze

Druhým možným způsobem zachycení informací je jejich zápis do podoby relací (tabulek) v relační databázi. Zde se již nelze obejít bez dalších programů, zejména tedy bez samotné databáze (MySQL, PostgreSQL, Oracle Database) a dále bez nástroje, který umožní dotazování databáze.

Je nutné navrhnout, jakým způsobem informace uložit a vzájemně provázat v relacích. U prostého textového formátu si lze představit jedinou tabulku, formáty vycházející z XML budou mít tabulek více. Jde o proces návrhu schématu databáze, který zde dále nerozvádím. Slovník ve formě relačních tabulek nejde vytvořit bez specializovaného programu, který informace správně rozloží do tabulek. Nejedná se tedy o lidsky čitelný formát.

Informace v databázi mohou být indexovány na základě domény (sloupec tabulky), ve které se nachází. Díky tomu je vyhledávání rychlejší (nevyžaduje sekvenční průchod celou

tabulkou, pokud je hledaná informace v indexu). Je také možné přistoupit jen k části obsažených informací a jen tuto část změnit.

Za nevýhodu považuji obtížnější přenositelnost, neboť nelze uchopit soubor(y) databáze a pouze je uložit na vhodné médium, nýbrž je nutné informace z databáze pomocí dotazu získat a do jiné databáze je poté vložit. Režie uložení je vyšší, než u předchozího případu, protože ke každé relaci se váží metadata popisující danou relaci.

2.3.3 XML databáze

Na rozdíl od předchozí nepoužívá XML databáze relací, tudíž není potřeba informace „narovnávat“ do tabulek, pakliže jsou nějakým způsobem hierarchicky strukturovaná. Opět si nelze vystačit bez dalších programů (databáze, dotazovací nástroj). Odpadá zde nutnost návrhu schématu, protože databáze sama uloží strukturovaná data odpovídajícím způsobem (např. do stromové struktury). Z toho vyplývá jedna podmínka, a sice že informace je zaznamenána ve strukturovaném formátu vycházejícím z XML. Prostý textový formát by tedy bez úprav (přidání XML značek) byl nevhodný.

Zaznamenané informace nejsou lidsky čitelné. Po programové stránce je práce s daty jednodušší, neboť není potřeba navazovat na sebe různé relační tabulky. Informace je možné indexovat na úrovni různých XML uzlů a jejich atributů. Vyhledávání proto může využít index, pokud je hledaná informace v něm obsažena, místo procházení všech dat. K informacím není nutné přistupovat jako k celku (slovníku), ale lze získat jen jeho části.

V této podobě záznamu utrpí přenositelnost, protože ani zde nelze uchopit soubor(y) databáze a libovolně je zkopírovat. Je nutné se na přenášená data dotázat a následně je vložit do jiné databáze. I režie je vyšší oproti textovému souboru, protože se ke každému dokumentu ukládají metainformace.

3 Specifikace LMF

Po stručném úvodu do problematiky uložení slovníků se zaměřím na Lexical Markup Framework [4], s nímž aplikace pracuje.

3.1 Terminologie

Nejprve bych rád definoval několik pojmů, které v této práci užívám a v kontextu slovníků jsou všeobecně rozšířené a často se s nimi setkáme.

Lemma

Obvyklý tvar slova zvolený pro reprezentování lexému. V českém jazyce se typicky bude jednat o slovo v prvním pádě, jednotném čísle (*vůz, práce*), rodě mužském (*tvrdý, pěkný*). U sloves hovoříme o infinitivu – neurčitém slovesném způsobu, který v sobě nevyjadřuje žádnou konkrétní osobu, číslo, způsob ani čas (*chodit, volat, pracovat*).
U některých podstatných jmen číslo jednotné neexistuje, například u podstatných jmen látkových (*mouka, voda*).

Lexém

Abstraktní jazyková jednotka, obecně spojená s určitou množinou tvarů, jež definuje jeden společný význam. Příkladem budiž slovo *dovolená*, kteří přísluší do množiny tvarů *dovolenou* a *dovolené.*

Lexikální záznam

Kontejner (či obal, zapouzdření) pro jeden nebo více tvarů mající jeden a více významů. Popisuje celý lexém.

Lexikální zdroj

Neboli lexikální databáze, sestává z jednoho či několika lexikonů.

Lexikon

Zdroj pro daný jazyk utvářený lexikálními záznamy. Ve zvláštním případě (zpracování přirozeného jazyka) může obsahovat pouze specifickou podmnožinu jazyka určenou pro zamýšlenou oblast použití.

```
<LexicalResource dtdVersion="16"> <!-- lexikální zdroj -->
      …
      <Lexicon> <!-- první lexikon -->
            <LexicalEntry id="001">…</LexicalEntry> <!--lexikální záznam-->
            <LexicalEntry id="002">…</LexicalEntry> <!--lexikální záznam-->
      </Lexicon>
      <Lexicon> <!-- druhý lexikon -->
            <LexicalEntry id="101">…</LexicalEntry> <!--lexikální záznam-->
            <LexicalEntry id="102">…</LexicalEntry> <!--lexikální záznam-->
      </Lexicon>
</LexicalResource>
```

Strojově čitelný slovník

Někdy označován MRD (machine readable dictionary). Elektronický lexikální zdroj. Jedná se o slovník zachycený číslicovou formou určený pro náhradu tištěných slovníků.

Strojový překladový lexikon

Elektronický lexikální zdroj, ve kterém jsou jednotlivé lexikální záznamy zachyceny v jazyce zdrojovém a dále v jednom či více jazycích cílových (tj. do těchto jazyků slovník překládá). Obsahuje syntaktické, sémantické, či morfologické informace pro usnadnění automatického nebo poloautomatického zpracování lexémů během strojového překladu.

Morfologie

Popisuje skladbu slova z hlediska použitých předpon (*do*dělat, *vy*dělat) a přípon (cykl*ista*, cykl*istický*), méně často vpon (sp*ink*at).

Morfém

Nejmenší, sémanticky dále nedělitelná jazyková jednotka. (kořen, předpona, vpona, přípona, koncovka).

Morf

Konkrétní ztvárnění morfému. Posloupnost fonémů. Různé morfy mohou reprezentovat jeden morfém. Jedná se tedy o slovo, které užijeme k vyjádření významu (morfy *stroj* (kořen), *ov* (přípona), *ý* (koncovka) utváří slovo *strojový*).

Foném

Dále nedělitelná jednotka řeči po stránce zvukové, měnící význam slova (r*a*k, r*o*k).

Ortografie

Neboli pravopis, definuje způsob, jakým se zapisuje lexém v daném jazyce. Například podle starého německého pravopisu se zámek zapisuje *Schloß*, dle nového pak *Schloss*, sklizeň kávy *Kaffeernte* nově *Kaffeeernte* (bez vypuštění písmene při skládání).

Zpracování přirozeného jazyka

Běžně užívaná anglická zkratka NLP (natural language processing). Soubor technik, který umožňuje počítačové zpracování jazykových dat.

3.2 Konvence užívané slovníky ve formátu LMF

Všechny slovníky psané ve formátu LMF musí ukládat znaky v kódování Unicode, což zaručuje možnost reprezentovat znaky libovolných jazyků uvnitř jediného dokumentu. Typickými zástupci jsou UTF-8 či UTF-16.

Kódování jazyků

Pojem často zaměňovaný s kódováním znaků. Identifikuje jazyk (čeština, angličtina, němčina, …) pomocí dohodnutého označení – kódu. V LMF slovnících by se mělo jednat o kódování podle některého ze standardů ISO 639.

Kódování písem

Stejně jako jazyk, tak i různá písma mají své standardizované označení, kterému má být dána přednost před vlastními způsoby označení. Pakliže písmo není již dáno identifikátorem jazyka, mělo by se jednat o identifikátor vycházející z normy ISO 15924.

Data Category Registry

Jedná se o další ISO standard (v současnosti 12620:2009), který je ve své podstatě registrem [5] pro zaznamenávání lingvistických pojmů. Většina pojmů je v současnosti definována v angličtině, neboť původní verze standardu z roku 1999 počítala pouze s ní. Někdy je však k dispozici vazba i na francouzštinu, která byla přidána v roce 2009.

Účelem DCR je sjednotit výrazy obsažené ve slovnících, které vyjadřují stejnou skutečnost. Chceme-li tedy říci, že slovo je v určitém mluvnickém čísle, dočteme se u vlastnosti *grammaticalNumber* o doporučených hodnotách *dual, massNoun, massNumber*[1]*, plural, singular*. Někteří autoři slovníků mají bohužel tendenci vymýšlet si vlastní hodnoty či je alespoň zkracovat (*pl., p., sin., sg.*), což ztěžuje automatizovanou práci s takovýmito jazykovými zdroji.

Myšlenka DCR je tedy velmi rozumná – různé systémy (nejen LMF, DCR není na něj přímo navázáno) mohou využívat danou terminologii a tím dávat základ pro jistou kompatibilitu. Bohužel samotný registr je dosti obsáhlý, a tudíž není vždy lehké nalézt to, co by nalezeno být mělo – autoři pak mohou opět sáhnout k vlastní tvůrčí činnosti.

Jako ukázku zde uvedu atributy elementu *feat* s hodnotami *att=hint* a *val=zool*. Z atributu *val* vyplývá, že se jedná o oblast použití (zoologie), což by mělo být klasifikováno jako *att=domain*.

3.3 Model LMF

Koncepce Lexical Markup Frameworku je zachycena prostřednictvím UML (Unified Modeling Language), což je dnes všeobecně rozšířený způsob pro specifikaci a vizualizaci modelů. V případě LMF jde o diagram tříd se vzájemnými vazbami mezi nimi a množinu párů atribut-hodnota pro definici dat podléhajících DCR.

Již jsem nastínil, že LMF je rozděleno do kolekce balíčků, z nichž každý má svůj specifický význam. Pro všechny slovníky je určeno jádro (core package), které obsahuje elementární prvky. Užití dalších balíčků záleží na konkrétním určení slovníku.

3.3.1 Jádro LMF

Sestává z celkem jedenácti různých tříd (elementů, uzlů), které zde nyní podrobněji rozeberu. Pro získání představy o vzájemné vazbě slouží diagram tříd (Obrázek 3-1).

[1] massNoun – číslo hromadné (např. křoví, kamení, listí), massNumber – číslo pomnožné (např. kalhoty, varhany)

LexicalResource

Třída reprezentující lexikální zdroj jako celek. V celém slovníku se nachází právě jednou a zabaluje všechny ostatní elementy. V terminologii XML se pak dá hovořit o kořenovém (*root*) elementu.

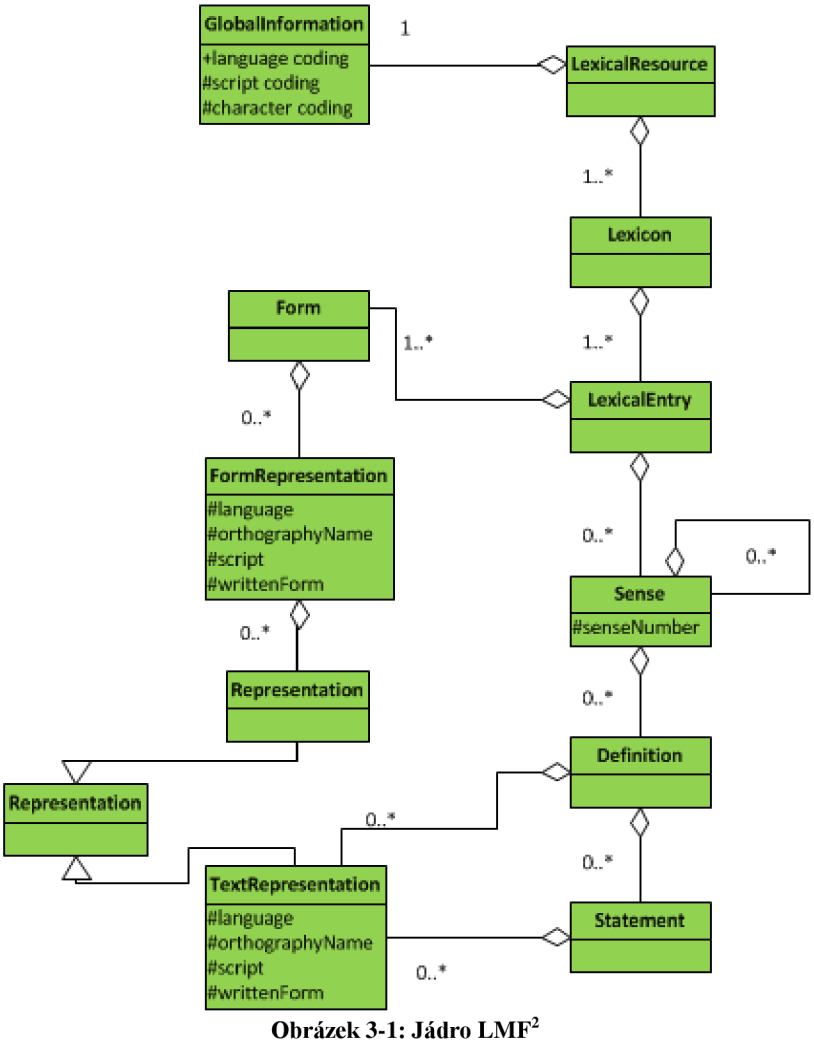
GlobalInformation

Obsahuje obecné informace o slovníku a vlastnosti platné pro celý slovník jakožto lexikální zdroj (*LexicalResource*). Tato třída obsahuje jeden povinný atribut a tím je *language coding*.

- **language coding** – určuje, který standard (typicky ISO norma) popisuje kódy názvů jazyků užitých v celém lexikálním zdroji

Následují dva volitelné atributy.

- **script coding** – specifikuje standard využitý pro popis kódování názvů písem užitých v celém lexikálním zdroji
- **character coding** – udává verzi Unicode, která je užita v celém lexikálním zdroji



Obrázek 3-1: Jádro LMF[2]

[2] Vytvořeno podle DTD [3] a specifikace LMF [4]

Lexicon

Ztvárňuje třídu obsahující všechny lexikální záznamy daného jazyka v rámci celého lexikálního zdroje (*LexicalResource*). Platná instance této třídy musí obsahovat jeden či více lexikálních záznamů (*LexicalEntry*).

LexicalEntry

Obsahuje jeden lexikální záznam slovníku, kterým se rozumí lexém v jazyce lexikonu. Jedná se o zapouzdření souvisejících tvarů a významů (třídy *Form* a *Sense*). Každý úplný lexikální záznam by měl obsahovat alespoň jednu formu, počet významů je volitelný.

Form

Abstraktní třída vyjadřující jeden lexém, morfologickou variantu lexému neb morf. Obsahem by měly být datové kategorie popisující vlastnosti dané slovní formy (lemma, výslovnost).

FormRepresentation

Třída představující jednu z možností zápisu lexému (*Form*). Zápis je v kódování Unicode a může se jednat nejen o samotný řetězec, nýbrž také o upřesnění v podobě specifického písma, jazyka či pravopisu (odlišného od zbytku slovníku – lexikonu).

- **language** – jazyk, kterým je zápis lexému proveden
- **script** – písmo, kterým je zápis lexému proveden
- **orthography name** – jméno pravopisu, ve kterém je lexém zapsán
- **written form** - lexém

Representation

Abstraktní třída obsahující řetězec v kódování Unicode s volitelným upřesněním písma, jazyka či pravopisu.

Sense

Představuje jeden význam lexikálního záznamu (*LexicalEntry*). Významy lze do sebe vzájemně zanořovat, čímž se vyjadřuje vyšší specifičnost daného významu.

- **sense number** – pořadové číslo významu slova v rámci elementu. Číslováno od jedné, netřeba užívat, pakliže význam je jen jeden

Definition

Zahrnuje slovní výklad vážící se k určitému významu (*Sense*). Užívá se zejména ve výkladových slovnících. Účelem je přiblížit čtenáři kontext pro lepší pochopení, není určeno pro strojové zpracování. Definicí může být i více, každá navíc může volitelně obsahovat neomezený počet textových vyjádření (*TextRepresentation*) pro zachycení definice ve více než jednom jazyce či písmu. Výklad nemusí být jen v jazyce/písmu lexikonu, nýbrž v kterémkoli jiném.

Statement

Popisuje význam slova, podobně jako výklad (*Definition*), nicméně v hlubším smyslu, nebo jako doplnění/upřesnění.

TextRepresentation

Vyjadřuje jedno z možných textových vyjádření. Obsahuje řetězec v kódování Unicode s volitelným upřesněním písma, jazyka či pravopisu.

- **language** – jazyk užitý ve vyjádření, musí respektovat *language coding* daného lexikálního zdroje
- **orthography name** – název pravopisu. Nepříliš běžné pro evropské jazyky, příkladem budiž *arabic unpointed* či *arabic pointed*
- **script** – písmo, kterým je vyjádření psáno, musí respektovat vlastnost *script coding* lexikálního zdroje
- **written form** – obsahuje samotné vyjádření

3.3.2 Balíček LMF pro morfologii

V aplikaci bude kromě jádra LMF využito také rozšíření v podobně elementů z balíčku pro morfologii (Obrázek 3-2) a strojově čitelný slovník (Obrázek 3-3). V poskytnutých slovnících se běžně vyskytují a z toho důvodu považuji za rozumné, aby si s nimi editor uměl při různých úpravách poradit.

Vysvětlím zde sémantiku těchto elementů.

Lemma

Třída Lemma je zděděna od třídy *Form*. Jedná se o jeden konkrétní slovní tvar, který byl vybrán, aby zastupoval celý lexikální záznam (*LexicalEntry*). Lexikální záznam bude obsahovat právě jednu instanci třídy Lemma. V českém jazyce se typicky jedná o sloveso ve tvaru infinitivu či podstatné jméno v prvním pádě jednotného čísla. Ve slovnících lze lemma považovat za jedno konkrétní heslo.

Lemma typicky obsahuje následující vlastnosti:

- **written form** – psaná podoba, řetězec ve formě Unicode
- **phonetic form** – výslovnost psané podoby, řetězec ve formě Unicode
- **geographical variant** – specifická forma užívaná v některém regionu
- **scheme** – vzor pro vytváření tvarů slova

WordForm

Slovní forma je opět zděděna od třídy *Form*. Ukazuje formu, v jaké se může vyskytovat lexém při použití v jazyce, typicky se jedná o ukázku nepravidelného skloňování.

Podobně jako Lemma, i slovní forma obsahuje vlastnosti, které se typicky objevují:

- **written form** – psaná podoba, řetězec ve formě Unicode
- **phonetic form** – výslovnost psané podoby, řetězec ve formě Unicode
- **hyphenation** – dělení slova, například na konci řádku, v souladu s gramatickými pravidly daného jazyka
- **grammatical number** – číslo, například u podstatného jména či slovesa, u jazyků, které ho rozlišují
- **grammatical gender** – rod, například rozlišení na mužský a ženský, v závislosti na jazyce
- **grammatical tense** – čas, typicky slovesný, udávající, k jakému období se forma slova vztahuje (minulost, současnost, budoucnost)
- **person** – osoba, například u podstatného jména či slovesa

Stem

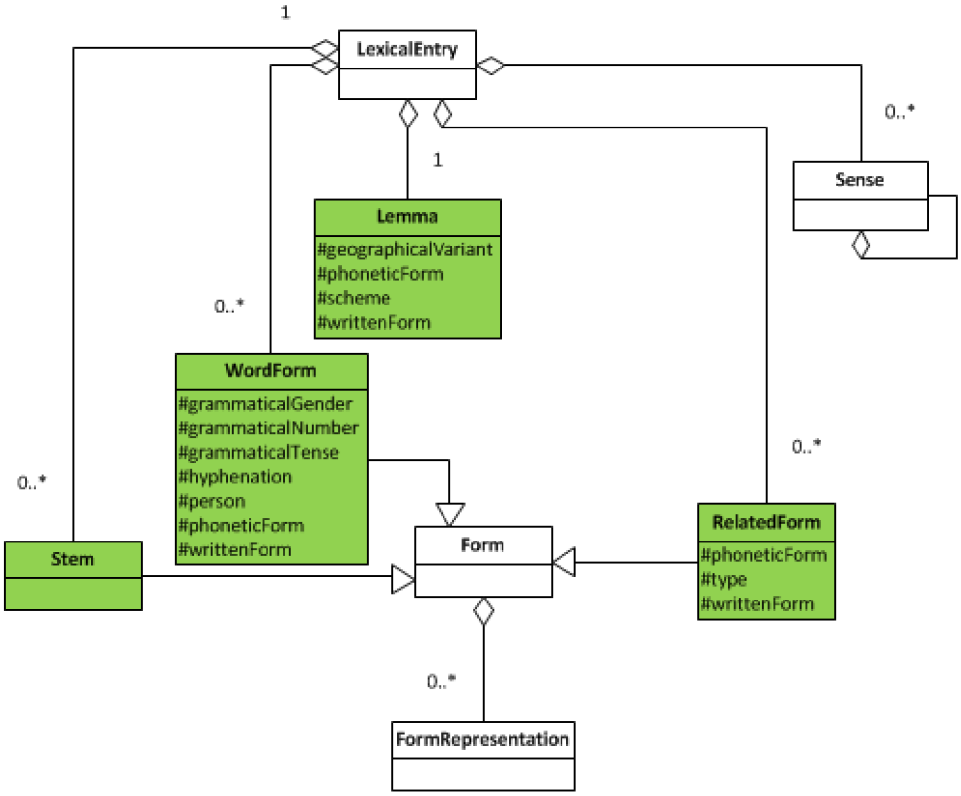
Další z podtříd odvozených od *Form*, reprezentuje jeden morf.

RelatedForm

Zděděna od třídy *Form*. Obsahuje slovní tvar nebo morf, který nějakým způsobem souvisí s lexikálním záznamem, ve kterém je obsažen. Jedná se například o slovo odvozené.

Vlastnostmi, které se typicky nacházejí v této entitě, jsou následující:

- **written form** – psaná podoba, řetězec ve formě Unicode
- **phonetic form** – výslovnost psané podoby, řetězec ve formě Unicode
- **type** – udává příslušnost k určité skupině věcí nebo osob majících podobné vlastnosti



Obrázek 3-2: Balíček pro morfologii LMF [3] [4]

3.3.3 Balíček LMF pro strojově čitelný slovník (MRD)

Účelem balíčku pro strojově čitelný slovník je poskytnout formát pro uložení jedno- a dvojjazyčných slovníků, které lze dále využít v automatizovaných překladačích, či v systémech přímo zpracovávajících přirozený jazyk. Opět zde vysvětlím sémantiku použitých elementů, jejichž vazby ilustruje diagram (Obrázek 3-3).

Equivalent

Ve dvojjazyčném slovníku ponese informaci o překladu hesla, které se nachází v uzlu *Lemma*. Překladů může být více, stejně tak nemusí být obsažen vůbec žádný, pakliže se jedná o jednojazyčný slovník. Smí obsahovat elementy *TextRepresentation* pro vyjádření jednoho kontextu v různých písmech či s různým pravopisem.

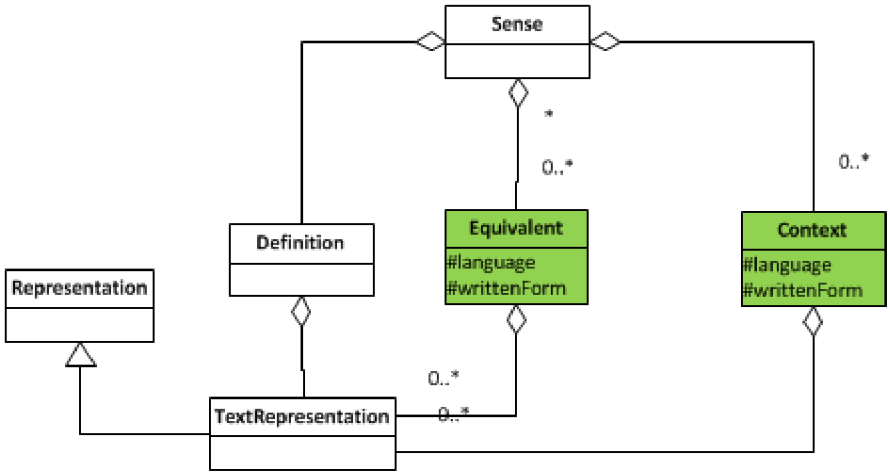
Typicky obsaženými vlastnostmi budou *written form*, která již byla vysvětlována, a také *language*.

- **language** – jazyk, ve kterém je překlad uveden. Musí být některá z hodnot určená vlastností *language coding* daného lexikálního zdroje.

Context

Třída znázorňující kontext užití daného slova. Může být obsažena vícekrát, či vůbec. Kromě přímo obsaženého kontextu lze využít zanořených elementů *TextRepresentation* pro vyjádření jednoho kontextu v různých písmech či s různým pravopisem.

Za vlastnosti, často se vyskytující v tomto elementu, lze považovat *written form* a *language*, které byly vysvětleny.



Obrázek 3-3: Balíček pro strojově čitelný slovník LMF[3] [4]

4 Oracle Berkeley XML DB

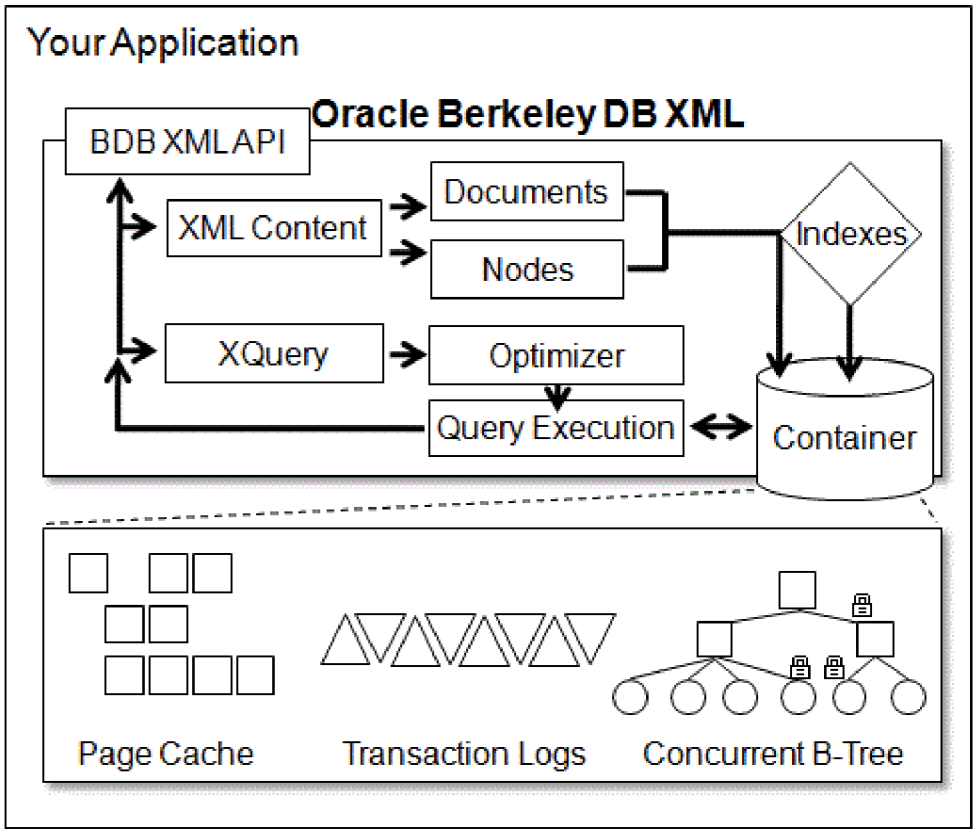
Systém řízení báze dat napsaný v jazyce C/C++, v současnosti vyvíjený firmou Oracle, původem však dílo University of California, Berkeley. Jedná se o knihovnu, kterou lze přidat k takřka libovolné aplikaci, neboť jsou k dispozici programová rozhraní pro C++, Java, Perl, Python, PHP, Tcl, Ruby a další. Nabízí rychlý přístup k uloženým dokumentům, možnost automatického nebo nastavitelného indexování, automatickou obnovu po chybě či transakční zpracování. Databáze pracuje v prostředí operačních systému Windows i Linux.

Databáze podporuje možnost dotazovat se nad jedním či více dokumenty současně, což je vhodné pro práci s více slovníky. Kromě XQuery ve verzi 1.0 (dotazovací jazyk pro XML dokumenty bez podpory jejich aktualizace) je k dispozici také XQuery Update 1.0, což je nestandardizovaná verze aktualizačních funkcí pro XML dokumenty.

Na ilustraci (Obrázek 4-1) lze vidět architekturu databáze. Dolní část zachycuje fyzické uložení dat, které je realizováno prostřednictvím B-stromu s podporou souběžného přístupu. Transakční žurnál nabízí možnost zotavení po chybách, například při nekorektním ukončení aplikace.

XML obsah lze zaznamenat jako celý dokument bez rozdělení na jednotlivé uzly, nicméně pro možnost adresování a vyhledávání je vhodnější jej nechat načíst, rozdělit a uložit po uzlech. Indexy vážící se ke každému dokumentu jsou uloženy přímo s ním v jediném souboru, který se nazývá kontejner. Jeden takový kontejner pak může obsahovat jeden či více XML dokumentů.

To vše je zapouzdřeno do API ve zvoleném programovacím jazyce, které se linkuje jako knihovna.



Obrázek 4-1: Oracle Berkeley DB XML [6]

4.1 XPath

S uložením slovníků v databázi velice úzce souvisí XML Path Language [7], což je jazyk, s jehož pomocí je možné adresovat určité části XML dokumentu. Nemusí se tedy jednat o jeden konkrétní uzel, nýbrž o množinu splňující zadaná pravidla (například uzly, které jsou potomkem zadaného uzlu).

XPath je standard organizace W3C [8], dá se tedy očekávat stálost v jeho podpoře a dalším vývoji, stejně tak kompatibilita napříč XML databázemi. Takto bude možné zaměnit systém řízení báze dat za jiný, aniž by celá aplikace začala postrádat funkčnost.

Příkladem XPath výrazu (adresy) budiž následující:

```
LexicalEntry/Lemma/feat[att=writtenForm]
```

Tímto dosáhneme selekce všech lemmat ve slovníku, respektive vlastnosti zachycující jejich psanou podobu. Obdobně by šlo adresovat výslovnost (*phoneticForm*) a jiné.

4.2 XQuery

XQuery se dá svým způsobem považovat za evoluci XML Path Language, neboť využívá jeho možnosti a dovádí je o úroveň dále. Jedná se o funkcionální programovací jazyk, jehož význam pro XML lze přirovnat k Structured Query Language (SQL) pro relační databáze.

Pomocí XQuery se dají vybírat části XML dokumentu a provádět jejich modifikace, například transformace uzlů před prohlížením či seřazení vrácených uzlů. Lze dokonce vystavět celý nový dokument jen za použití XQuery, a to díky funkcím pro dynamickou tvorbu uzlů.

Jako ukázku použití bych zde zvolil jeden z pokročilejších výrazů, takzvaný FLWOR (vzniklo ze slov FOR, LET, WHERE, ORDER BY a RETURN), který je do značné míry obdobou již jmenovaného SQL.

```
for $lexEntry in
doc("dbxml://slovnik.dbxml/dictionary_main")/LexicalResource/Lexicon/Lexic
alEntry
order by $lexEntry/Lemma/feat[att=writtenForm]/@val descending
return $i
```

For cyklus postupně prochází všechny elementy *LexicalEntry* obsažené ve slovníku a po seřazení dle lemmatu je navrací.

5 Analýza problému

Formuluji několik základních podproblémů, kterým se dále budu věnovat. Potřebujeme načíst XML soubor se slovníkem a vhodným způsobem jej uložit, k čemuž využívám XML databázi Oracle Berkeley.

Jedním druhem práce se slovníky je jejich porovnávání, kdy je záměrem zobrazit vzájemné rozdíly mezi dvěma různými slovníky, například za účelem doplnění hesel z prvního slovníku do druhého či pro ověření, že obrácená verze překladového slovníku obsahuje všechna hesla užitá v překladech.

Dalším nárokem na práci je možnost hromadně měnit informace ve slovníku obsažené. Oproti individuální úpravě hesel klade tento způsob editace jiné požadavky na přístup k datům, neboť dochází ke slučování většího množství čtených dat.

Ve výsledku má uživatel možnost upravený slovník opět uložit do XML souboru.

5.1 Ukládání slovníků

Vhodnou otázkou je, proč vlastně slovníky ukládat do databáze. Přímá práce s XML souborem na disku je možná alternativa, kterou nelze zcela zavrhnout. Nicméně soubor nabízí pouze sekvenční přístup, a pakliže máme potřebu číst jen určitou jeho část (pro zobrazení náhledu hesla), musíme nějakým způsobem tuto část nejprve nalézt, což se neobejde bez postupného rozdělení a načtení celého souboru do paměti.

Dále je po provedení modifikace vyžadován zpětný zápis celého souboru. Vzhledem k tomu, že slovníky dosahují velikosti i nad 100MB, není jejich čtení a zápis z časového hlediska okamžitě proveditelná úloha a pokud chceme zobrazit jen část slovníku, nevyplatí se čekat na zpracování celého souboru.

5.2 Porovnávání slovníků

Jak již bylo nastíněno, účelem je zobrazit uživateli rozdíly mezi dvěma slovníky srozumitelnou formou, přičemž řešení si poradí i s velkými slovníky (nad 100MB). Práce s takto rozsáhlým souborem dat formátu XML je paměťově náročná a při samotné implementaci musí být hospodárně nakládáno se zdroji, aby nedocházelo k nedostatkům v množství alokovatelné paměti[3].

Porovnávání překladů hesel

První formou porovnávání je zjišťování, zda dané dva slovníky obsahují pro stejná hesla shodné překlady. Jinými slovy, zda se v jednom ze slovníku nenachází některé navíc, či naopak chybějí. Využití pro takovouto funkci lze najít například při srovnávání konkurenčních slovníků od různých autorů, kdy se uživatel (autor slovníku) snaží doplnit své dílo o překlady, které se ve druhém slovníku nacházejí.

Z technického hlediska by se jedná o operaci spojení (join) nad dvěma XML dokumenty, přičemž vazebním prvkem jsou hesla (lemmata). Vezmou se překlady z prvního slovníku, z nichž

[3] Zvolený implementační jazyk Java, respektive jeho virtuální stroj, dovoluje využít nejvýše 1400MB operační paměti na 32bitovém systému Windows (přesná hodnota se může na různých počítačích lišit), který je používán pro vývoj apliakce.

se odeberou ty, které jsou obsažené i ve druhém slovníku. Zbylé překlady z prvního slovníku se vypíší uživateli. Záměnou pořadí porovnávaných slovníků lze snadno získat opačnou verzi.

Hluboké porovnávání hesel

Někdy nám nejde o pouhé vypsání překladů, ale zajímají nás i další informace, jež se k nim váží. Slovníky často obsahují další slovní formy, zejména u nepravidelně skloňovaných slov (sloves, podstatných jmen, …). Každý význam, pro který má dané heslo překlad, může být navíc doplněn příkladem nebo vysvětlením (definicí).

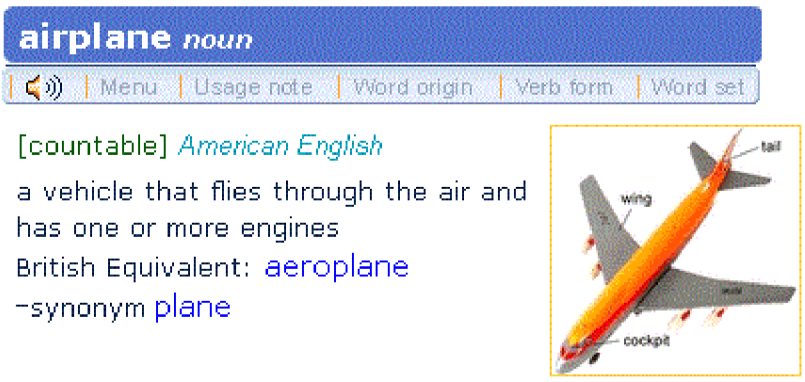
Cílem je tedy opět provést operaci spojení nad slovníky. Tentokráte však množina informací braných v potaz bude rozsáhlejší a strukturovaná, nepůjde o pouhý pár slovo-překlad. Ve výsledku je uživateli vypsán rozdíl obou slovníků.

Vyhledávání slov z definic

Pro výkladové slovníky je zpravidla významné, aby sama formulace, kterou je některý pojem vysvětlován, byla ve slovníku také dohledatelná, tedy aby sestávala ze slov, která se ve slovníku nacházejí. Takto to dělá například slovník současné angličtiny Longman [9] ve své verzi na CD (Obrázek 5-1, poklepáním na libovolné slovo z definice jej lze vyhledávat). Úkolem tedy je vzít obsahy definic zadaného slovníku a tyto definice rozdělit na jednotlivá hesla (slova). Slova poté jsou vyhledána ve slovníku, aby se ověřilo, že „slovník je schopen vysvětlit sám sebe“ a neopírá se tak o slova, která by čtenáři nemusela být srozumitelná.

Popsaný případ však vychází z praxe všeobecných výkladových slovníků, bez specifického zaměření. Je pochopitelné, že u anglického technického výkladového slovníku nejsou běžná podstatná jména čtenáři vysvětlována. Volitelně proto jde zadat jiný slovník, ve kterém se definice vyhledávají. Obsah definic technického slovníku tak je srovnáván s hesly všeobecného slovníku.

Při tomto způsobu hledání je vhodné, aby uživatel disponoval možností zahrnout do uvažovaných hesel (lemmat) také jejich nepravidelné slovní tvary.

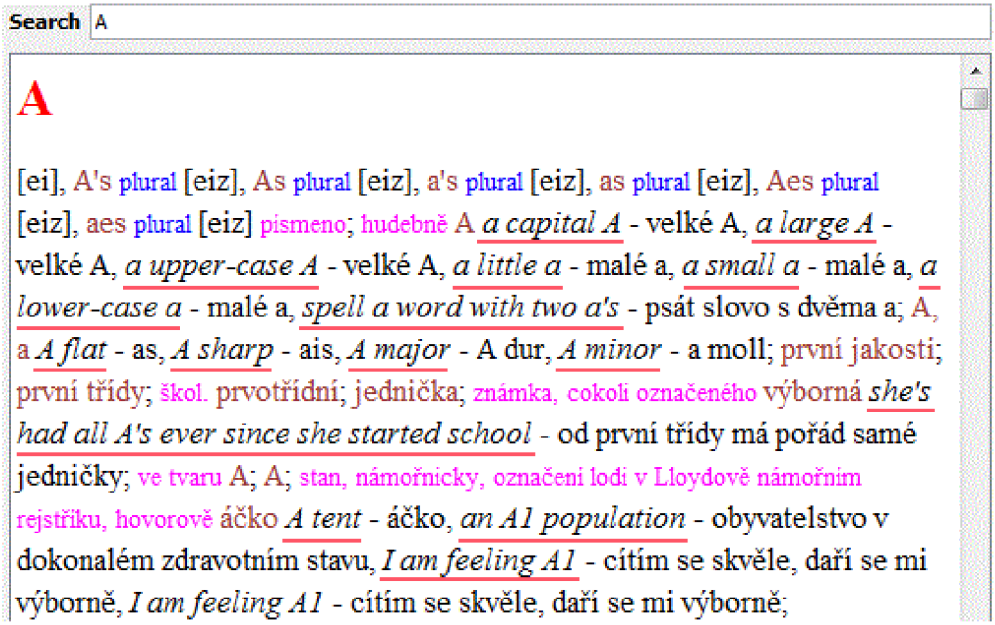


Obrázek 5-1: Ukázka z výkladového slovníku Longman [9]

Vyhledávání slov z příkladů

Ještě o něco specifičtější než předchozí případ jsou příklady užití ve slovníku. Nezávisle na tom, zda je slovník překladový či výkladový, může obsahovat příklady (v jazyce překládaném či v jazyce překladu). Uživatel si tedy vybere, který jazyk je pro něj relevantní a buď nad slovníkem samým, či nad jiným zadaným, provede vyhledání hesel (případně i slovních tvarů).

Obrázek (Obrázek 5-2) ilustruje možné příklady k heslu *A* (červeně podtržené) v anglicko-českém překladovém slovníku. Jedná se o slovník bez specifického zaměření, měl by tedy být schopen vysvětlit většinu ze slov, která jsou v něm pro příklady použita.

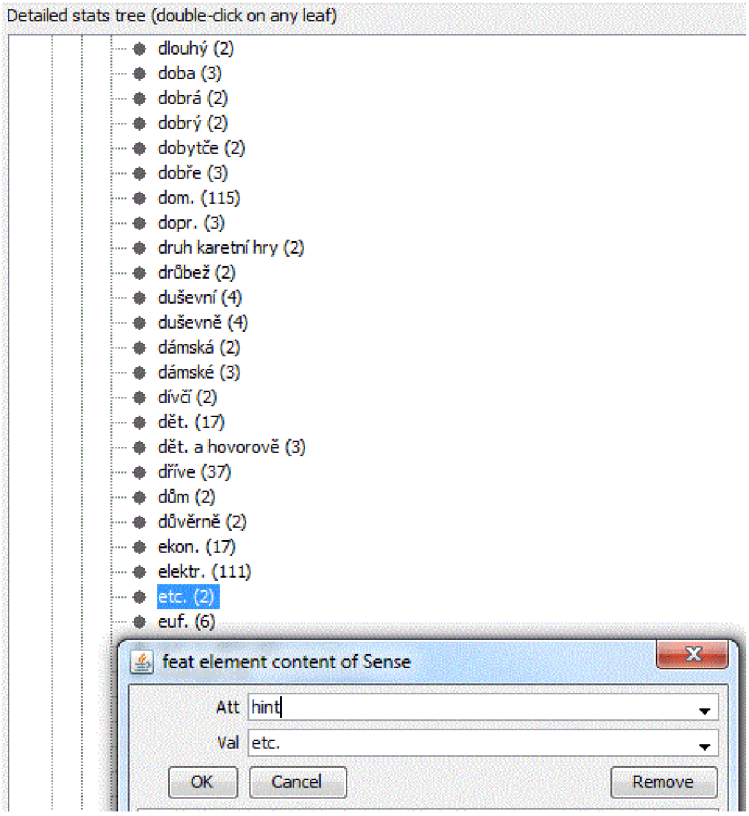


Obrázek 5-2: Příklady užití hesla *A* v anglicko-českém slovníku

5.3 Hromadná změna

Hromadná změna informací obsažených ve slovnících je poměrně náročná na databázi, neboť je potřeba vytvářet agregovaná data – seskupovat různé uzly či vlastnosti obsažené ve slovnících. Je nezbytné si také jednoznačně zapamatovat, která data byla agregována z kterých uzlů, aby bylo možné zpětně modifikovat údaje obsažené v databázi.

V neposlední řadě se snažím dosáhnout uživatelsky přívětivého rozhraní, které vhodným způsobem (ve formě stromu) znázorňuje výše uvedené. Rozhraní zobrazuje uživateli náhled konkrétního lexikálního záznamu, pokud se dostane až na jeho úroveň. Stejně tak lze obsah lexikálního záznamu upravovat, ať už za účelem opravy obsažených informací, či přidáním nových.



Obrázek 5-3: Hromadná změna slovníku

Ukázka (Obrázek 5-3) znázorňuje změnu elementů *feat* obsahujících dvojici *(hint, etc.)*, které jsou potomky uzlů *Sense*. Prostřednictvím dialogu lze všechny tyto elementy změnit, v tomto případě například přeložit *etc.* na české *atd.*

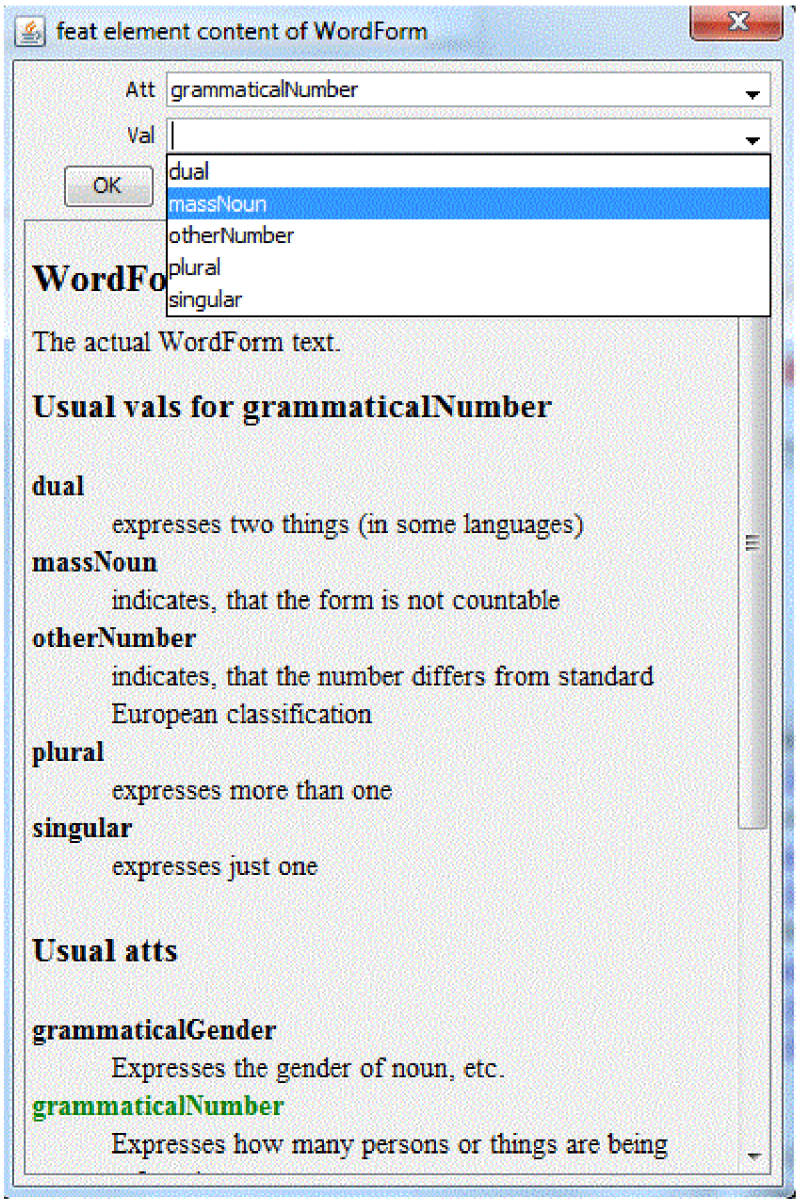
5.3.1 Nápověda

Lexical Markup Framework je rozsáhlá značkovací podmnožina XML a jeho vazba na registr datových kategorií (DCR) uživateli příliš neusnadňuje práci na slovníku. Zejména u vlastností, zaznamenaných v uzlech *feat*, můžeme být často na pochybách, jaká hodnota by měla být obsažena. Z tohoto důvodu aplikace poskytuje nápovědu při editaci LMF elementů.

V podstatě jde o část informací obsažených v DCR (označované jako Data Category Selection, tedy výběr informací) zanesených do aplikace a zobrazovaných relevantně při editaci určitého uzlu.

Tím by měly být eliminovány případy, kdy autoři slovníků vytvářejí z vlastní invence nové názvy vlastností, ačkoli již existují jiné, se stejnou sémantikou. Samotný obsah nápovědy je načítán z XML souboru, proto je možné jej podle potřeb dále upravovat.

Obrázek (Obrázek 5-4) zachycuje skutečnou podobu nápovědy v aplikaci. Ke každému uzlu zvlášť je k dispozici soubor atributů elementů *feat*, ke kterým mohou být definovány výčtem očekávané hodnoty.



Obrázek 5-4: Nápověda k uzlu WordForm

6 Návrh a popis implementace

Pro implementaci samotného nástroje jsem si zvolil jazyk Java [10]. Důvodem je přenositelnost výsledného zdrojového kódu, který díky běhu ve virtuálním stroji není závislý na architektuře počítače a v současné chvíli se nabízí virtuální stroje pro Windows i Linux, na kterých budu aplikaci vyvíjet (existují však i pro Solaris, či OS X).

Další předností je objektově orientovaný přístup k tvorbě programu, který si Java vynucuje, a jenž zvyšuje přehlednost i modifikovatelnost aplikace.

Nejprve jsem se zabýval konzolovou částí, která zahrnuje porovnávání slovníků. Zde si lze vystačit s výstupem ve formě textu. Zhotovil jsem však i grafické rozhraní, jehož účelem je nabídnout větší přitažlivost softwaru a snazší ovládání širší skupině uživatelů. Hromadná změna slovníků si pak bez grafického rozhraní přirozeně nevystačí.

V popisu používám odkazování na `balík.třída`, čímž upřesňuji, kde danou třídu ve zdrojových kódech čtenář najde.

6.1 Základ aplikace

Základní součástí, ze které se program spouští, je `Console.Main` a s ním úzce související `Console.Params`. V těchto třídách se provádí rozpoznání a uložení zvolených parametrů pro další řízení běhu aplikace.
Vytváří se zde také instance `dbxml.DictionaryManager2`, což je hlavní třída řídící uložení slovníků do databáze a jejich čtení.

6.2 Ukládání slovníků

`DictionaryManager2` obsahuje metody pro nakládání se slovníky, tedy zadávání souborů, ze kterých se má číst do databáze, či do kterých má databáze exportovat. Hlavní práci obstarávají metody `AddLmfDictionary`, `SaveLmfDictionary` a `DeleteLmDictionary`.

6.2.1 Přidávání

Metoda pro přidávání vytvoří nejprve nový kontejner v Berkeley XML databázi. V průběhu implementace bylo vyzkoušeno, že ukládat všechny slovníky do jednoho kontejneru není výhodné. Důvodem je, že s rostoucí velikostí kontejneru se zpomaluje práce s ním, což při porovnávání velkých objemů dat ve slovnících nepříjemně zpomaluje práci s aplikací.

Příčinou zřejmě je, že kontejner má sdílený index pro všechny dokumenty. Všechny XML soubory v jednom kontejneru mají jednotné nastavení indexování, mohou tedy mít i více indexů, podle kterých se vyhledává, nicméně pro každý index je jedna datová struktura sdílená mezi všemi dokumenty.

Berkeley XML databáze si sama zpracovává dokument pomocí SAX parseru, což je velmi výhodné z hlediska paměťové náročnosti. Byly vyzkoušeny i jiné XML databáze (eXist [11], Xindice

[12]), které užívaly DOM, známý svou paměťovou náročností. 60MB slovník se pak projevil jako 800MB zabrané operační paměti, nad 100MB bylo téměř vyloučeno slovník do databáze přidat[4].

Po přidání naroste slovník až na trojnásobek své původní velikosti (porovnávána velikost XML souboru vůči výslednému kontejneru).

6.2.2 Indexování

Po vytvoření nového kontejneru (a s tím spojeného souboru na disku) se nastaví vlastnosti indexování. V současné chvíli to je indexování elementů a atributů elementů podle jejich řetězcového obsahu na porovnávání na rovnost.

Databáze nabízí vcelku široké možnosti nastavení indexů. Lze například indexovat každý uzel v cestě (XPath výrazu), nebo jen koncové (listové) uzly. Pro naše slovníky se zdá vhodnější indexování všech, neboť častěji se vybírá celý jeden záznam (*LexicalEntry*) slovníku, než jeden koncový uzel.

Zahrnuty jsou jak elementy, tak atributy, neboť při vyhledávání jednoho konkrétního hesla se tato informace nachází jako atribut elementu *feat*, zatímco při zmíněném čtení celého jednoho záznamu se atribut nepotřebuje. Indexovat lze i metadata, která se ke každému XML dokumentu automaticky váží.

Dále se nabízí nastavení porovnávání klíče a to na přítomnost, rovnost či výskyt podřetězce. První uvedené je vhodné, pokud je vyhledáván nějaký atribut bez ohledu na jeho hodnotu, druhé, pokud chcete atribut s danou konkrétní hodnotou, a poslední je podobné předchozímu s tím rozdílem, že rychle nachází uzly, když je zadána jen část obsažené hodnoty atributu.

Databáze umožňuje přiřadit jednotlivým vyhledávacím klíčům také určitou syntaktickou podobu, což však shledávám méně užitečným pro slovníky. Pakliže v atributu například očekáváte číslo s pohyblivou řádovou čárkou, booleovskou hodnotu, čas, nebo datum, můžete to explicitně vyjádřit, pro účely slovníku specifikuji jen obecně řetězec (*string*) bez hlubší syntaktické vazby.

Nastavení indexu je vhodné provést před přidáním slovníku a dále ho neměnit, neboť následné reindexování si může vyžádat několik minut.

6.2.3 Ukládání a odstranění

Databáze nabízí prostředky pro přímé uložení struktury slovníku do XML souboru, není tedy třeba explicitně celý slovník vybírat nějakým XQuery dotazem a ručně zapisovat do souboru.

Odstranění slovníku z databáze lze považovat za nejrychlejší operaci. Se slovníkem se automaticky odstraní i veškerá metadata s ním svázaná.

6.3 Porovnávání slovníků

Porovnávání slovníků je zajišťováno třídou `dbxml.DictionaryManager2`, která obsahuje další potřebné metody pro získávání informací ze slovníků.

[4] Testování bylo prováděno na Windows 7 Professional x86 s 2GHz dvoujádrovým procesorem s JDK 1.6.0 Update 24.

6.3.1 Porovnání překladů hesel

Nejprve jsem se pokoušel implementovat porovnání pomocí jediného XQuery dotazu, tak jak bych to dělal například v relační databázi pomocí SQL. Napsal jsem tedy dotaz v následujícím tvaru, který vzápětí vysvětlím:

```
for $i in
doc('dbxml:/slovnik1/dictionary_main')/LexicalResource/Lexicon/LexicalEntr
y/Sense
let $b :=
doc('dbxml:/slovnik2/dictionary_main')/LexicalResource/Lexicon/LexicalEntr
y
let $z := $b[Lemma/feat[@att='writtenForm'][@val=distinct-
values($i/../Lemma/feat[@att='writtenForm']/@val)]]

where count($z) > 0 and exists($i/Equivalent) and not(distinct-
values($i/Equivalent/feat[@att='writtenForm']/@val) = distinct-
values($z/Sense/Equivalent/feat[@att='writtenForm']/@val))
return ($i/../Lemma, $i)
```

V první části dotazu se do proměnné *i* dosadí uzly *Sense*, obsahující překlad slova. Proměnná *b* poté vybere všechny uzly *LexicalEntry* z druhého slovníku a následně je naváže prostřednictvím jejich lemmatu (join using) jeden na druhý (proměnná *z*). V poslední části se poté vyberou pouze ty uzly, které nemají shodnou psanou podobu překladu v uzlu *Sense/Equivalent* a dojde k jejich navrácení uživateli.

Je tedy na samotném systému řízení báze dat, aby si poradil s navázáním jednotlivých lemmat na sebe (operace join) a celý dotaz úspěšně provedl. Toto se v praxi bohužel ukázalo jako neefektivní řešení, neboť SŘBD v XML databázi zpracovává dotaz pomalu.

Několikanásobně rychlejším řešením, na základě testování (7.2), se ukázalo být takové, kdy jsou nejprve z obou slovníků načtena všechna lemmata pomocí dotazu

```
for $i in distinct-values(doc('dbxml:/slovnik1/dictionary_main')/
LexicalResource/Lexicon/LexicalEntry[Lemma/feat[@att='writtenForm'][@val=$
var]]/Sense/Equivalent/feat[@att='writtenForm']/@val)
order by $i
return $i
```

a následně se postupně porovnávají překlady z prvního slovníku s překlady z druhého. Zaznamenané rozdíly mezi slovníky jsou na závěr vypsány na standardní výstup.

6.3.2 Úplné porovnávání překladů hesel

Úplné porovnávání vyžaduje komparaci celých uzlů *Sense* patřících danému lexikálnímu záznamu. Na začátku byla opět snaha zajistit celou operaci jedním XQuery dotazem, což by představovalo elegantní řešení problému.

Zde je však operace o něco složitější a o to sofistikovanější bude dotaz, který ji zajistí. Výsledný dotaz mající realizovat úplné porovnání slovníků nabyl této podoby:

```
for $i in doc('dbxml:/
slovnik1/dictionary_main')/LexicalResource/Lexicon/LexicalEntry
```

```
let $b := doc('dbxml:/
slovnik2/dictionary_main')/LexicalResource/Lexicon/LexicalEntry
let $z := $b[Lemma/feat[@att='writtenForm'][@val=distinct-
values($i/Lemma/feat[@att='writtenForm']/@val)]]
let $senses := $i/Sense
let $other_senses := $z/Sense

where count($z) > 0 and not (some $one in $z satisfies deep-equal($one,
$i))
return (
for $x in $senses
where   not(some $y in $other_senses satisfies deep-equal($y,$x))
return ($i/Lemma, $x)
)
```

Uvedený dotaz vybírá z prvního slovníku do proměnné *i* všechny lexikální záznamy (uzly *LexicalEntry*) a totéž činí u druhého slovníku do proměnné *b*. Následně provede jejich vzájemné navázání, kdy proměnná *z* bude obsahovat takové uzly druhého slovníku, které odpovídají aktuálnímu lemmatu lexikálního záznamu prvního z nich.

Funkcí *deep-equal* se provede porovnání vždy dvou korespondujících uzlů, a pakliže je nalezena neshoda, je tato vypsána. Funkce pro úplné (hluboké) porovnání nemusí být dostupná v každé implementaci XQuery, proto může být potřeba u některých databází si ji definovat vlastními prostředky.

Popsaná komparace na základě jediného dotazu byla podrobena testování (7.3), při kterém nevykazovala příliš uspokojivé výsledky v oblasti rychlosti provedení. Proto byla vyzkoušena varianta, kdy jsou z prvního i druhého slovníku vybrány všechny lexikální záznamy, a ty následně porovnávány klasickými prostředky, tedy parsováním uzlu *LexicalEntry* a následným zjištěním rozdílu mezi prvním a druhým z nich.

Druhý přístup je paměťově náročnější, neboť načítá z obou slovníků všechny záznamy do operační paměti počítače, ale vykazuje znatelně rychlejší provedení. Vzhledem k dostatku paměti a nedostatku času většiny uživatelů jsem proto zvolil druhou implementaci jako vhodnější.

6.3.3 Vyhledávání slov z definic

Vyhledávání slov z definic, která nejsou obsažena jako hesla ve slovníku (s případným zahnutím slovních tvarů – *WordForm*) probíhá v několika krocích. Nejprve jsou vybrány všechny definice, které se v lexikálním zdroji nacházejí. Následně se načtou všechna hesla (lemmata) ze zadaného slovníku, což může být slovník totožný. Poté se sekvenčně procházejí definice, přičemž každá je rozdělena na znaku mezery. Provede se odstranění speciálních znaků – `'('`, `')'`, `','`, `';'`, `'.'`, `'!'`, `'?'`, `'%'`, `'['`, `']'`, `'+'`, `'-'`, `'/'`, `'…'`, `':'`, které mohly roztržením vět zůstat na konci slov. Zde se dbá na to, aby nezůstala ve slově nepárová otevírací kulatá závorka, pakliže byla odtržena uzavírací. Pokud máme například slovo „minut(y)“, nedojde k odtržení koncové závorky. Mohlo by být vhodné odtrhnout celou část slova v závorkách, nicméně jsem se přiklonil k řešení v podobě ponechání části slova.

Nyní, když máme definici rozdělenou na jednotlivá slova, jsou tato hledána v heslech slovníku. Hesla jsou ze slovníku načtena do kolekce v podobě množiny založené na rozptýlených položkách (*HashSet*). Volitelně se načtou obdobným způsobem tvary slov, pokud uživatel vyžaduje jejich zahrnutí do vyhledávání.

6.3.4 Vyhledávání slov z příkladů

Operace podobná práci s definicemi, kdy je nejprve potřeba načíst všechny příklady ze slovníku, ty následně rozdělit na jednotlivá slova a s těmito slovy pracovat. Opět se odstraňují nežádoucí speciální znaky a výsledek se porovnává s hesly či slovními formami ve slovníku zadaném uživatelem.

6.3.5 Alternativy

Za účelem dosažení úspory času porovnávání jsem experimentoval s uložením slovníku do více než jednoho kontejneru. Pomocí vlastního algoritmu jsem jej nejprve rozdělil na více dílů, přičemž každý obsahoval stejný (nebo lišící se o jedničku) počet lexikálních záznamů. Vycházel jsem z poznatku, že menší kontejner má méně zaplněný svůj index a tím dokáže i rychleji přistupovat k datům v něm obsaženým.

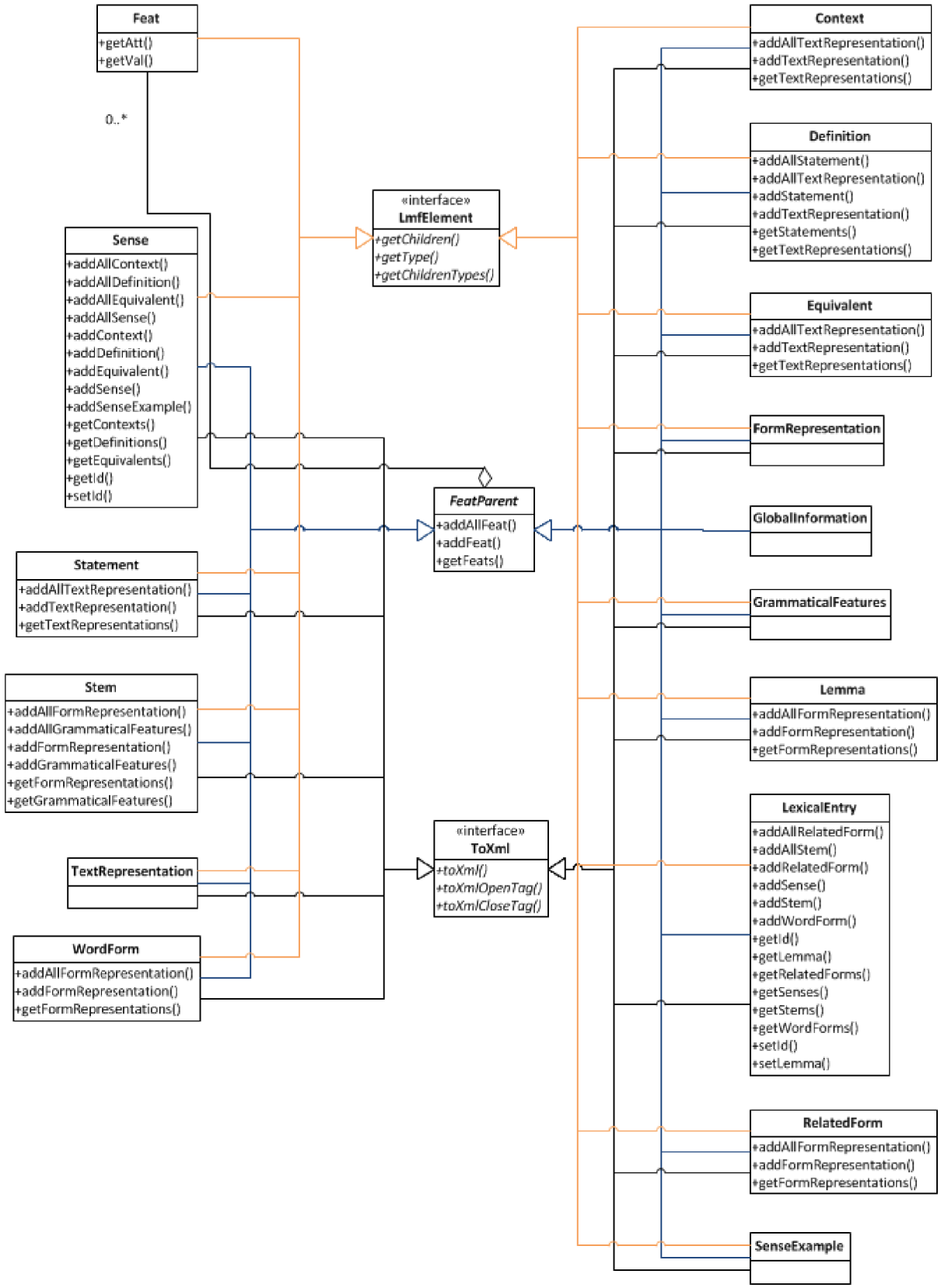
Tento předpoklad se však ukázal jako lichý, neboť zrychlení bylo jen nepatrné. Důvodem zřejmě bylo, že do určité velikosti dokumentu v kontejneru (100MB) funguje vyhledávání hesel velice rychle. Za hlavní nevýhodu považuji, že při editaci slovníku se musí určovat, ve které části se daný lexikální záznam nachází, což znesnadňuje následnou implementaci.

Další variantou bylo vícevláknové zpracování. Na základě abecedního rozdělení hesel mělo jedno vlákno porovnat část slovníku a další vlákno další část. Opět se jednalo z hlediska programátorského přístupu o složitější řešení, jehož úskalím se zdál být jeden pevný disk, ze kterého vlákna četla slovník, a proto dosažené zrychlení nebylo pozorovatelné.

6.4 Hromadná změna slovníků

Specifickou úlohou nástroje je poskytnout možnost hromadně měnit informace obsažené ve slovnících. Na rozdíl od selektivní editace jednotlivých uzlů se zde pracuje s velkým množstvím uzlů, potažmo dat, která je třeba ze slovníku získávat způsobem dostatečně rychlým pro pohodlnou práci uživatele.

Při této editaci nevystačíme s pouhými XQuery dotazy, kterými budeme z databáze získávat lexikální záznamy. Důvodem je zejména fakt, že potřebujeme získat agregovaná data ze slovníku, tedy kolikrát se ten který element či vlastnost ve slovníku nachází. Při takovém dotazu by se pokaždé musel prohledávat celý slovník, což by uživatele zdržovalo. Nepomohou zde ani indexy, které jsem vysvětloval v úvodu, neboť by musel být indexován téměř každý uzel slovníku, což by vytvořilo výsledný index značného objemu a malého výkonu. Namísto toho jsem vytvořil vlastní datové struktury (Obrázek 6-1) reprezentující jednotlivé LMF entity, které se snaží s co možná nejmenší paměťovou náročností pojmout data obsažená ve slovníku.



Obrázek 6-1: Datové struktury pro načtení LMF slovníku

6.4.1 Datové struktury

S ohledem na velikost a přehlednost diagramu byly vynechány některé operace. Jde zejména o metody *getOrdinalNo()* a *setOrdinalNo()*, či *getParent()* a *setParent()*, sloužící k jednoznačnému specifikování vazby mezi elementy. Pomocí pořadových čísel v rámci jednoho rodiče lze jednoznačně identifikovat v XPath výrazu jakýkoli uzel.

6.4.2 Naplnění datových struktur

Pro načtení dat bude sloužit jediný XQuery dotaz, který postupně načte celý lexikon (uzel *Lexicon*) jednoho lexikálního zdroje. Rozsah jednoho lexikonu byl zvolen z toho důvodu, že obsahuje informace vztahující se k jednomu jazyku v rámci lexikálního zdroje. Dalším důvodem je paměťová náročnost, kdy je snaha načítat logicky použitelný celek, ale zároveň nenačítat zbytečně mnoho dat.

Jedná se o jednoduchý dotaz, kde *i* značí pořadové číslo lexikonu:

```
doc('dbxml:/slovnik/dictionary_main')/LexicalResource/Lexicon[i]/LexicalEn
try
```

Postupnou iterací přes výsledek budou vraceny jednotlivé lexikální záznamy. Vzhledem k tomu, že se sekvenčně procházejí úplně všechny záznamy, je operace poměrně rychlá a nenáročná pro systém řízení báze dat. Nicméně velké množství čtených dat zvyšuje nároky na záznamové médium, kterým je pevný disk, a proto by tato operace mohla být urychlena použitím disku na bázi flash pamětí (SSD), namísto klasického.

6.4.3 Výpočet statistik

Statistiky spravuje třída `stats.LightweightStats`. Název dává tušit, že je kladen důraz na nízkou paměťovou náročnost. Pro vznik se nejprve vyžádají z databáze lexikální záznamy daného lexikonu, přičemž jejich transformaci do mnou navržené datové struktury provádí třída `lmf.Parser`.

Postupným procházením výsledků dotazu se čtou jednotlivé lexikální záznamy a jsou zpracovávány do tříd popsaných na obrázku (Obrázek 6-1) a vzájemně provázány, čímž vzniká stromová struktura popisující celek. Všechny třídy obsahují jen nezbytné množství dalších objektů, neboť při čtení rozsáhlejších slovníků vznikají těchto objektů desetitisíce a je potřeba, aby se do paměti vešly všechny.

Při zpracování lexikálních záznamů se zároveň prochází jejich stromová struktura a vyhledávají se vlastnosti, jejichž hodnoty či výskyt se agregují. Díky tomu je možné uživateli zobrazit počty výskytů různých elementů a také počty a hodnoty vlastností v rámci těchto elementů. Cílem je poskytnout náhled na všechny uzly a vlastnosti slovníku v takové formě, kdy lze přímo vybrat některou vlastnost a tu jednoduše editovat.

Po změně vlastnosti se musí provést aktualizace mnou vytvořené datové struktury, následně tedy i aktualizace statistiky a zároveň uskutečnit zpětný zápis do XML databáze, aby úprava byla trvalého charakteru. Právě zpětný zápis změněné vlastnosti do databáze je z časového hlediska náročná operace – přistupuje se k relativně pomalému pevnému disku, proto bylo důležité nalézt co možná nejrychlejší dotaz.

7 Testování

Pozornost je věnována době provádění různých akcí spjatých s prací aplikace. Cílem je přiblížit čtenáři časovou náročnost operací a také poskytnout možnost srovnání s případnou vlastní implementací. Na základě některých testů byly upraveny způsoby provádění operací v aplikaci.

7.1 Přidávání slovníků

Předmětem testu je čas potřebný pro přidání slovníku do databáze, zjištění, zda je čas přímo úměrný velikosti slovníku, a jaká bude výsledná velikost kontejneru se slovníkem oproti samotnému XML dokumentu.

K testování užívám slovník *encz_docx.xml*, což je překladový anglicko-český slovník čítající 103022 hesel a zabírající 123209kB ve formě XML souboru. Pro potřeby testu byla v textovém editoru vytvořena verze s poloviční počtem hesel – 51511, 61328kB a dále s 5588 hesly, 6258kB.

7.1.1 Výsledky

Tabulka 7-1: Výkonnost přidávání slovníků

Počet hesel	Velikost XML [kB]	Čas přidávání [ms]	Velikost kontejneru [kB]	Čas / heslo [ms]	Velikost kont. / heslo [kB]
5588	6258	5306	20088	0,949534717	3,594846099
51511	61328	50880	194376	0,987750189	3,773485275
103022	123209	101884	383680	0,988953816	3,724253072

Z tabulky je vidět, že čas přidávání jednoho hesla slovníku do kontejneru je prakticky stejný pro všechny velikosti slovníků, stejně tak zabraná velikost paměti na disku odpovídá počtu hesel ve slovníku. Velikost kontejneru oproti čistému XML souboru na disku je pak přibližně trojnásobná.

7.2 Rychlost porovnávání překladů

Testována je různá implementace operace porovnávání dvou slovníků, kdy je požadavkem vypsat všechny překlady z prvního slovníku, které se nenacházejí pro dané heslo ve druhém slovníku. Nejprve je celá operace realizována jedním XQuery dotazem, který na sebe vzájemně naváže dva slovníky podle hesla (lemmatu) a následně vybere ty překlady, které se nacházejí jen v prvním z nich.

Poté je stejná operace rozdělena do více dotazů. Vyberou se hesla z prvního slovníku, z druhého a vzápětí se iterací budou hledat překlady vždy pro dané heslo. Překlady se okamžitě porovnají a případná neshoda se zaznamená. Na závěr budou rozdíly zapsány na výstup.

Poslední metodou porovnávání je vypsání překladů do souboru a následné zpracování v programech sort a comm.

K testování jsou využity tři překladové slovníky – anglicko-český, francouzsko-český a španělsko-český, čítající 51511, 28935 respektive 36106 hesel. U prvního z nich se jedná o zkrácenou (poloviční) verzi slovníku. Od každého ze slovníků byla vytvořena mírně upravená verze s odstraněnými či pozměněnými překlady. V tabulce jsou zaznamenány časy porovnávání.

7.2.1 Výsledky

Tabulka 7-2: Čas porovnávání

Slovník			Čas porovnávání			
Název	Počet hesel	Velikost XML [kB]	jedním dotazem [ms]	[min]	více dotazy [ms]	[min]
encz_docx	51511	61328	3217224	53,62	105568	1,76
frcz_docx	28935	35466	1059697	17,66	42994	0,72
spcz_docx	36106	44643	2481154	41,35	60446	1,01

7-3: Čas porovnávání (sort, comm)

Slovník			Čas porovnávání				
Název	Počet hesel	Velikost XML [kB]	výpis překladů [ms]	sort [ms]	comm [ms]	celkem [ms]	celkem [min]
encz_docx	51511	61328	100534	1939	1125	103598	1,73
frcz_docx	28935	35466	38232	1210	511	39953	0,67
spcz_docx	36106	44643	56230	2107	1017	59354	0,99

Porovnávání jediným XQuery dotazem je pomalé a v praxi nepoužitelné. Daleko rychlejší se jeví druhá varianta, která byla nakonec v aplikaci zvolena. Poslední varianta je sice nejrychlejší, ale oproti druhé jen nepatrně, navíc činí aplikaci závislou na dalších nainstalovaných aplikacích.

7.3 Rychlost úplného porovnávání

Úkolem testu je srovnat rychlost úplného (hlubokého) porovnávání lexikálních záznamů dvou slovníků. Použit je XQuery dotaz zvládající celou operaci najednou a vícekrokový postup, kdy se nejprve načtou celé lexikální záznamy z obou slovníků a ty se následně v operační paměti porovnávají. K tomu se využije rozdílu mezi významy (*Sense*) vždy dvou korespondujících lexikálních záznamů.

K testování byly užity překladové slovníky z předchozího testu, tentokrát z nich však není brána jen informace o českém ekvivalentu hesla, nýbrž se berou v potaz všechny informace. Každý ze slovníků byl cíleně upraven tak, aby obsahoval rozdíly oproti originálu. V tabulce jsou zaznamenány časy porovnávání.

7.3.1 Výsledky

Tabulka 7-4: Čas úplného porovnávání

Slovník			Čas porovnávání			
Název	Počet hesel	Velikost XML [kB]	jedním dotazem [ms]	[min]	více dotazy [ms]	[min]
encz_docx	51511	61328	1010690	16,84	132419	2,21
frcz_docx	28935	35466	300459	5,01	56461	0,94
spcz_docx	36106	44643	499254	8,32	78342	1,31

Z výsledků je patrné, že druhá implementace dosahuje znatelně lepších výkonů co do rychlosti porovnání. Na základě testování byla proto upravena implementace metody pro porovnávání tak, aby využívala druhou, tedy rychlejší variantu. Trošku paradoxně proběhlo hluboké porovnání rychleji, než srovnávání jednotlivých překladů. Jako příčinu vidím menší počet spojování záznamů, kdy namísto navazování jednoho překladu (uzel *Equivalent*) po druhém se rovnou srovnává celý uzel *Sense*.

7.4 Vytváření statistik

Tvorba statistik je časově náročná operace, proto je snaha ji co možná nejvíce urychlit. Původně se výpočet statistik prováděl pouze v jednom vlákně, nicméně ve snaze najít rychlejší řešení jsem otestoval možnost výpočtu statistik vícevláknově.

Do testování byly zahrnuty slovníky Diderot a překladové anglicko-český, francouzsko-český a španělsko-český. Vzhledem k tomu, že testovací stroj disponuje pouze dvěma jádry pro výpočet (bez podpory zpracování více vláken na jednom jádře), dá se operace hodnotit jako dvouvláknová. Nicméně zamýšlený princip by šel rozvést i do více paralelních vláken.

7.4.1 Výsledky

Slovník			Čas vytváření statistik			
Název	Počet hesel	Velikost XML [kB]	jednovláknově [ms]	[min]	vícevláknově [ms]	[min]
diderot	90982	48412	91315	1,52	58791	0,98
encz_docx	51511	61328	76114	1,27	45120	0,75
frcz_docx	28935	35466	41068	0,68	26707	0,45
spcz_docx	36106	44643	48932	0,82	33949	0,57

Z naměřených časů usuzuji, že je vhodnější použít ve výsledné implementaci vícevláknový přístup, neboť nechat uživatele čekat bezmála o minutu déle na vytvoření statistik může značně snížit komfort při práci s aplikací.

8 Závěr

Vzniklý nástroj považuji za přínosný nejen v oblasti hromadné úpravy slovníků, což je poměrně specifická oblast, náročná na zpracování velkých objemů dat, oproti individuální modifikaci záznamů, ale také jako pomůcku ke zjišťování rozdílů mezi různými verzemi slovníků. Podle zadaného kritéria lze mezi sebou slovníky porovnávat a zjišťovat tak jejich nedostatky v obsažených informacích.

Aplikace poskytuje uživateli nápovědu při práci se slovníkem. Jde o zobrazování vhodných vlastností, které se typicky v modifikovaném elementu vyskytují. Pokud je to relevantní, lze z nabízeného výčtu vybrat konkrétní hodnotu.

Realizovaná aplikace by mohla být dále vyvíjena směrem rozšiřování zahrnutých LMF balíčků, které jsem zde užil tři – jádro LMF, balíček pro morfologii a balíček pro strojově čitelný slovník (MRD), a stejně tak v oblasti poskytování ještě širší nápovědy uživateli. Data Category Registry je svou podstatou velmi rozsáhlý a aplikaci by prospělo, kdyby z něj obsahovala co nejširší část.

Literatura

[1] SVOBODA, Milan. *GNU/FDL Anglicko-Český slovník* [online]. 2001-2004 [citováno 2012-01-26]. Dostupné z: <http://slovnik.zcu.cz/format.php>.

[2] SIGNOV, Sergej. *XDXF standand; Draft 028* [online]. 2012 [citováno 2012-03-28]. Dostupné z: <http://xdxf.revdanica.com/drafts/visual/028/XDXF-draft-028.txt>.

[3] International Organization for Standardization. *LMF DTD v. 16* [online]. 2010 [citováno 2012-01-26]. Dostupné z: <http://www.tagmatica.fr/lmf/DTD_LMF_REV_16.dtd>.

[4] International Organization for Standardization. *Language resource management — Lexical markup framework* [online]. 3/2008 [citováno 2012-02-09]. Dostupné z: <http://www.tagmatica.fr/lmf/iso_tc37_sc4_n453_rev16_FDIS_24613_LMF.pdf>.

[5] ISOcat. *ISOcat – Web interface* [online]. 2010 [citováno 2012-02-09]. Dostupné z: <http://www.isocat.org/interface/index.html>.

[6] Oracle Corporation. *Oracle Berkeley DB XML* [online]. 2011 [citováno 2012-02-12]. Dostupné z: <http://www.oracle.com/technetwork/database/berkeleydb/overview/index-083851.html>.

[7] BRADLEY, Neil. *XML: kompletní průvodce.* Vyd. 1. Praha: Grada, 2000, 537 s. ISBN 80-716-9949-7.

[8] W3C. *XML Path Language (XPath)* [online]. 1999 [citováno 2012-02-09]. Dostupné z: <http://www.w3.org/TR/xpath>.

[9] BULLON, Stephen. *Longman Dictionary of Contemporary English: The living dictionary.* Vyd. 3. Essex: Pearson Education Limited, 2003, 1949 s. ISBN 05-827-7646-5.

[10] DARWIN, Ian F. *Java: kuchařka programátora.* Vyd. 1. Brno: Computer Press, 2006, 798 s. ISBN 80-251-0944-5.

[11] MEIER, Wolfgang. *eXist-db Open Source Native XML Database* [online]. 2000 [citováno 2012-03-28]. Dostupné z: <http://exist-db.org/exist/index.xml>.

[12] The Apache Software Foundation. *Apache Xindice* [online]. 2001 [citováno 2012-03-28]. Dostupné z: <http://xml.apache.org/xindice/>.

Seznam příloh

Příloha 1. CD s aplikací a dokumentací