VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

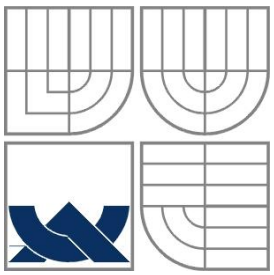
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

Systém pro správu slovníkových dat

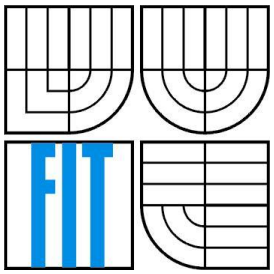
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE Petr Šimek
AUTHOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SYSTÉM PRO SPRÁVU SLOVNÍKOVÝCH DAT

DICTIONARY DATA MANAGEMENT SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Petr Šimek

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. RNDr. Pavel Smrž, Ph.D.

BRNO 2014

Abstrakt

Bakalářská práce se zabývá elektronickými slovníky, které jsou uloženy ve formátu Lexical Markup Framework. Jejich analýzou správnosti převodu z jiných formátů, sjednocování použité terminologie a hromadnou změnou dat. Pro tyto účely byly vytvořeny skripty v jazyce Python3.

Abstract

The bachelor's thesis deals with electronic dictionaries that are stored in the format Lexical Markup Framework. It deals with their analysis of the correctness of the conversion to other formats, the unification of used terminology and mass changes data. The scripts in Python3 were created for these purposes.

Klíčová slova

Elektronické slovníky, XML, transformace, XSLT, DTD, Lexical Markup Framework, Python3.

Keywords

Electronic dictionaries, XML, transform, XSLT, DTD, Lexical Markup Framework, Python3.

Citace

Šimek Petr: Systém pro správu a sjednocování jazykových slovníků, bakalářská práce, Brno, FIT VUT v Brně, 2014

Systém pro správu slovníkových dat

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.

……
Petr Šimek
20. května 2014

Poděkování

Rád bych poděkoval docentu Smržovi za cenné rady a vedení mé práce.

Obsah

Obsah 1
1 Úvod 3
2 XML 4
2.1 Syntaxe 4
2.2 Zpracování XML 5
2.2.1 DOM 6
2.2.2 SAX 6
2.3 Popis struktury 7
2.3.1 DTD 7
2.3.2 XML Schema 8
2.4 XSL 9
2.4.1 XSLT 9
2.5 XPath 10
3 Slovník 11
3.1 Typy slovníků 11
3.2 LMF 12
3.2.1 Data Category Registry 12
3.2.2 Základní balíček 13
3.2.3 Rozšíření základního balíčku 14
4 Návrh systému 15
4.1 Statistiky 15
4.2 Hromadné změny 16
4.3 Použité nástroje 17
4.3.1 GNU Aspell 17
4.3.2 Morfologický slovník a morfologický analyzátor pro češtinu 18
4.3.3 Modul SAX 18
4.3.4 DTDGenerator 18
4.3.5 Saxon 18
4.3.6 cProfile 18
5 Implementace 19
5.1 Ovládání 19
5.2 Statistiky 19
5.2.1 Kontrola slov 19
5.2.2 DTD schéma 20

5.2.3 Duplicity 22
5.2.4 Rozpoznání jazyků 22
5.2.5 Seznam použitých hodnot 23
5.2.6 Další statistiky 25
5.2.7 Chyba syntaxe 25
5.2.8 Vzorový soubor pro změnu 25
5.2.9 Zpracování XML dokumentu 26
5.3 Hromadné změny 26
5.3.1 SAX Generator 26
5.3.2 XSLT 27
6 Výsledky 32
6.1 Statistiky 32
6.1.1 DTD Schéma 33
6.1.2 Kontrola slov 34
6.1.3 Seznam použitých hodnot 34
6.1.4 Duplicity 34
6.2 Hromadné změny 35
6.2.1 SAX Generator 35
6.2.2 XSL 36
7 Závěr 37
8 Bibliografie 38

1 Úvod

Při převodu mezi různými formáty slovníků vznikají velmi často chyby, například může jít o špatně převedená slova nebo špatně napsaný převodník mezi formáty, který některé části nepřevede a podobně.

V této práci se zabývám systémem, který pomůže uživateli odhalit a opravit chyby ve slovnících uložených ve formátu Lexical Markup Framework. Pomocí analýzy slovníku uživatel získá potřebné informace, které mu pomohou rozhodnout o kvalitě převodu. Informace získané z analýzy nemusí sloužit pouze k odhalení chyb, ale mohou uživateli dát představu o vlastnostech slovníku jako je struktura, jazyk, počet klíčových slov, oblast použití klíčových slov a další informace. Systém uživateli dále dovoluje nalezené problémy opravit a sjednotit používanou terminologii u informací vyskytujících se ve slovnících.

2 XML

XML (Extensible Markup Language - rozšiřitelný značkovací jazyk) je podmnožinou staršího metajazyka SGML[1]. Specifikace jazyka XML je volně dostupná na stránkách konsorcia W3C, která formát spravuje[2]. Pomocí XML jsme schopni popsat význam dat, ale ne jejich vzhled jako u jiných jazyků, např. HTML.

Pro práci s formátem XML vzniklo velké množství knihoven pro spoustu různých programovacích jazyků, nástroje pro definici konkrétního formátu (DTD), nástroje pro navigaci v rámci dokumentu (Xpath), nástroje pro transformaci jednoho XML dokumentu do jiného (XSLT).

Hlavními výhodami XML oproti jiným formátům používaných pro přenos informací je nezávislost na konkrétní aplikaci nebo platformě, standardizace, poměrně malá velikost a podpora národních kódování[3].

2.1 Syntaxe

XML dokument je složen z prvků, které jsou do sebe navzájem vnořené. Prvky v textu se vyznačují pomocí tzv. značek. Většina prvků je v dokumentu vyznačena pomocí dvou značek, počáteční a ukončovací značky.

```
<Prvek_1> Obsah prvku</Prvek_1>
```

V případě, že prvek nemá žádný obsah, je ho možné zapsat zkrácenou formou.

```
<Prvek_1> Obsah <Prvek_2/> prvku</Prvek_1>
```

Syntaktická pravidla:

- Každý prvek musí mít ukončovací značku (</Prvek_1>) nebo musí být zapsán zkrácenou formou (<Prvek_2/>).
- Prvky se nesmí v dokumentu křížit.

```
< Prvek_1> Obsah < Prvek_2> prvku </ Prvek_1> </ Prvek_2>
```

- Žádny prvek nesmí obsahovat redefinování stejného názvu atributu.

```
< Prvek _1 id="1"  id="2" > Obsah prvku </ Prvek _1>
```

- Celý dokument musí být obsažen v jednom prvku.

```
<Root>
   < Prvek _1> Obsah prvku </ Prvek _1>
   < Prvek _2> Obsah prvku </ Prvek _2>
   < Prvek _3> Obsah prvku </ Prvek _3>
</Root>
```

- Pro vyjádření speciálních znaků uvnitř hodnoty atributu nebo obsahu prvku, používáme escape sekvence podle tabulky.

Původní hodnota	Nahrazení pomocí	Význam
<	<	menší než
>	>	vetší než
&	&	ampersand
‘	'	apostrof
“	"	uvozovka

- Další možnost pro vyjádření speciální znaků v obsahu prvku je uzavřít text do CDATA sekce.

```
< Prvek _1>
<![CDATA [ if a < 2 && a > 0] ]>
</ Prvek _1>
```

- Komentáře mohou být jednořádkové nebo víceřádkové.

```
<!—Komentář -->
<!—
    Komentář
-->
```

- V názvu prvku se rozlišují velká a malá písmena (case sensitive).

```
<PRVEK>
        < Prvek/>
 </PRVEK>
```

- Hodnoty atributů musí být ohraničeny pomocí apostrofů nebo uvozovek.

```
< Prvek _1 atribtu_1=“1“ atribut_2=‘42‘ />
```

Splňuje-li dokument všechna výše zmíněná pravidla, je syntakticky v pořádku a říkáme o něm, že je správně strukturován (well-formed)[3]. Tato vlastnost se nijak nevztahuje k samotnému obsahu nebo struktuře dokumentu.

Dokument je platný (valid) v případě, že splní definici dokumentu určenou v externím souboru (DTD)[4].

```
<!DOCTYPE LexicalResource SYSTEM "DTD_LMF_REV_16.dtd">
```

V případě, že program při zpracovávání XML dokument nalezne chybu, podle specifikace XML zastaví čtení dokumentu[2].

2.2 Zpracování XML

Pro zpracování XML dokumentu není nutné psát vlastní aplikaci nebo knihovnu, ale můžeme použít již existující XML parser.

XML parser je program (knihovna), který kontroluje syntaktickou správnost XML dokumentu. Některé parsery jsou schopné dokument validovat oproti schématu. Pomocí parseru se implementace čtení a zápisu XML dokumentu ve výsledné aplikace velmi zjednoduší. V aplikaci nám parser data zpřístupní v příjemnější podobě[3].

Existují standardizované rozhraní (API) pro práci s XML dokumentem a naštěstí většina tvůrců parserů tato rozhraní používá. Nejznámější a nejrozšířenější rozhraní jsou DOM a SAX.

2.2.1 DOM

DOM (Document object model) je standart konsorcia W3C[5]. Původně byl vytvořen jako standart pro nové webové prohlížeče, které podporují XML, aby používaly stejný objektový model pro přístup k dokumentům ze skriptovacích jazyků.

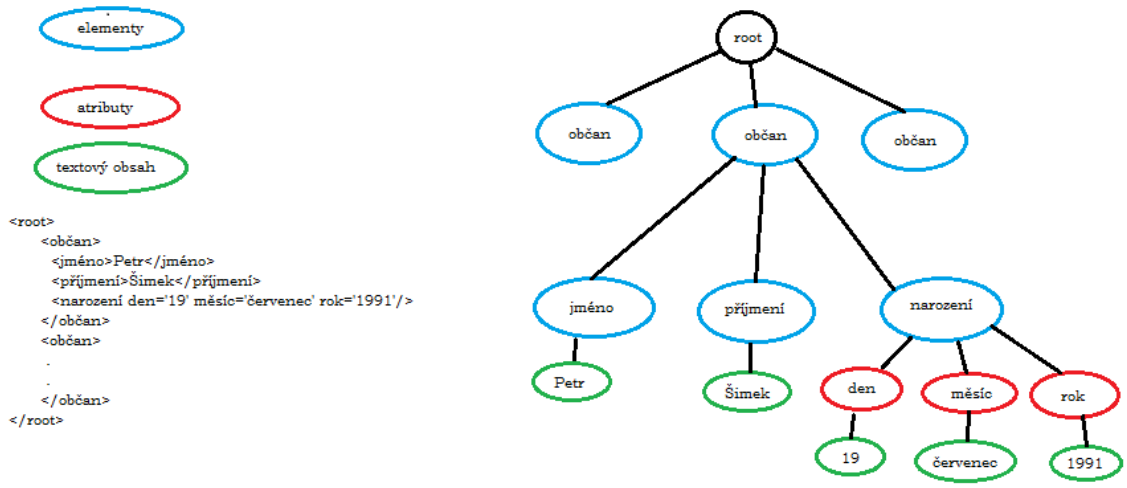
DOM, XML dokument reprezentuje jako stromovou hierarchickou strukturu, kde pro každý prvek je jeden uzel ve stromu. Tyto uzly mají dále odpovídající uzly z informací o atributech, komentářích, textovém obsahu a další. Tomuto způsobu se říká grove (Graph Representation Of property ValuEs).

Rozhraní DOM obsahuje rozsáhlou paletu funkcí, které nám umožňují modifikovat jeho jednotlivé uzly, procházet v rámci stromu libovolným způsobem, mazat a přidávat uzly.

Na rozdíl od SAX nemusíme dokument procházet od začátku do konce, ale můžeme se v něm pohybovat dle naší potřeby. Proto se rozhraní DOM uplatní v aplikacích, které provádějí náročnější operace s dokumentem – editory, prohlížeče a formátovače[3].

Při použití na velkých XML dokumentech (100mb) je podstatně pomalejší a paměťově náročnější než SAX, protože se musí načíst celý XML dokument do paměti.

Příklad převedení XML dokumentu na DOM strom (Obrázek 1).



Obrázek 1 - XML dokument převeden na DOM strom

2.2.2 SAX

Rozhraní SAX (Simple API for XML) je založeno na řízení pomocí událostí (event-driven). V praxi to znamená, že si nadefinujete skupinu funkcí, které jsou volány při určitých událostech, například začátek prvku, konec prvku, obsah prvku, komentář apod. Nadefinovaným funkcím jsou poté předány potřebné parametry, jako název prvku a další.

Rozhraní SAX je dnes podporováno řadou parserů i přes skutečnost, že není definováno žádným standardem konsorcia W3C nebo jinou organizací. Rozhraní bylo vytvořeno skupinou programátorů z diskusní skupiny xml-dev a do jisté míry představuje standart[3].

Výhodou událostmi řízeného přístupu je rychlost zpracování a malá paměťová náročnost. Dokument se čte postupně, prvek za prvkem a proto ho není třeba celý načíst do paměti.

Nevýhodou je, že soubor se musí zpracovat v jednom průchodu a nelze se vracet k dříve zpracovaným částem.

Rozhraní SAX nejčastěji používáme pro zpracování velkých souborů, kvůli paměťovým nárokům, jaké by mělo rozhraní DOM.

2.3 Popis struktury

Jazyk XML nám dovoluje vytvářet si vlastní sady prvků, které použijeme v dokumentu. Proto je vhodné mít možnost jak popsat strukturu prvků, které použijeme v daném dokumentu. K tomuto účelu využíváme schémata. Pomocí těchto schémat jsme schopni definovat strukturu všech prvků v souboru přes povolené atributy, vnořené prvky až po typ dat, které se mohou požít.

Pomocí schéma jednoznačně definujeme strukturu dokumentu, a proto ho můžeme využít pro validaci dokumentu. Validace ověřuje shodu dokumentu XML se schématem, jestli dokument dodržuje všechny omezení definována ve schématu. Díky tomu můžeme vyvíjet aplikace, které již nemusí kontrolovat vstupní data a tím se nám zjednoduší vývoj aplikace.

2.3.1 DTD

DTD (Document Type Definition) [6] je poměrně starou technologií pro popis struktury XML dokumentu, ale i přesto se stále hojně používá. Je to díky výborné podpoře v mnoha různých aplikacích a parserech. Další technologie, které vytváří schémata, alespoň částečně vychází z DTD [3].

Jako největší nedostatek DTD schématu můžeme označit nulovou podporu pro jmenné prostory, nemožnost určit datový typ pro atribut a obsah prvků a samotný popis DTD schéma není uložen jako XML dokument.

Několik příkladů DTD schémat [7]:

1.
```
<!ELEMENT price (#PCDATA)>
<!ATTLIST price currency NMTOKEN #IMPLIED>
```
2.
```
<!ELEMENT product (number, name+, size?, color*)>
```
3.
```
<!ELEMENT el ((a | b)*, (c | d)?)>
```
4.
```
<!ELEMENT letter (#PCDATA | custName | prodName)*>
```

2.3.2 XML Schema

XML Schema Definition (XSD) byl vyvinut konsorciem W3C jako nástupce DTD a odstraňuje jeho největší nedostatky. Umožňuje definování datového typu pro atribut a obsahu prvků, používání jemných prostorů, dovoluje přesně určit počet a pořadí výskytu prvků.

V jednom dokumentu může kombinovat více schémat a díky tomu, že je objektově orientován, můžeme využívat například dědičnost a zapouzdření. Celý XSD dokument je zapsán ve formátu XML a díky tomu ho lze zpracovávat stejnými aplikacemi jako XML dokument (Zpracování XML2.2)[8].

Nevýhodou je složitá specifikace. Při ručním psaní schématu, bez aplikace pro tvorbu schématu, je velmi náročné napsat rozsáhlejší schéma.

Několik příkladů XML schema (význam stejný jako v části DTD 2.3.1)[7]:

1.
```
<xs:element name="price">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
    <xs:attribute name="currency" type="xs:NMTOKEN"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

2.
```
<xs:element name="product">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="number"/>
      <xs:element ref="name"  maxOccurs="unbounded"/>
      <xs:element ref="size"  minOccurs="0"/>
      <xs:element ref="color" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

3.
```
<xs:element name="el">
  <xs:complexType>
    <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="a"/>
        <xs:element ref="b"/>
      </xs:choice>
      <xs:choice minOccurs="0" maxOccurs="1">
        <xs:element ref="c"/>
        <xs:element ref="d"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.
```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="custName"/>
      <xs:element ref="prodName"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

2.4 XSL

XSL (eXtensible Stylesheet Language) je rodina jazyků umožňujících popsat, jak se mají XML soubory formátovat a převádět. XSL patří do rodiny standardů konsorcia W3C.

Při vzniku jazyka XSL, šlo pouze o vytvoření lepší varianty CSS pro definici vzhledu XML dokumentu. Umožňoval definovat vzhled jednotlivých prvků – způsob jejich zarovnání, velikost, styl písma apod. Kromě tohoto ho bylo možno použít i k takovým věcem, jako je automatické generování obsahu, číslování obrázků apod. Postupně se ukázalo, že XSL má sloužit ke dvěma poměrně odlišným věcem – k transformaci XML dokumentů a k definici vzhledu jejich formátování.

Během příprav standardu XSL z něj proto byla vyřazena jeho část sloužící k transformaci dokumentů, pro kterou se používá název XSLT (XSL Transformations). Pomocí XSLT lze vytvářet styly, které definují, jak se XML dokumenty mají převádět do formátu HTML, do XML dokumentů s jinou strukturou nebo do obyčejných textových souborů[9].

Druhé části XSL, která slouží k přesnému popisu vzhledu dokumentu, se nazývá XSL FO (formátovací objekty). K dispozici máme vše, co známe z CSS, navíc lze přesně určit layout stránky[10].

2.4.1 XSLT

XSLT je založeno na šablonách, vždy když se nalezne shoda pro daný prvek se šablonou, provede se transformace prvku popsaná v příslušné šabloně.

Příklad jak pomocí XSLT převedeme XML dokument z obrázku (Obrázek 1):

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  >
<xsl:output method='xml' encoding='utf-8' indent="yes"/>
<xsl:template match="*">
        <xsl:copy>
                <xsl:apply-templates select="*"/>
        </xsl:copy>
</xsl:template>
<xsl:template match="//narození">
   <xsl:element name="{translate(name(), ' ','_')}">
                <xsl:for-each select="@*">
                        <xsl:element name="{translate(name(), ' ','_')}">
                                <xsl:value-of select="."/>
                        </xsl:element>
                </xsl:for-each>
                <xsl:apply-templates select="*"/>
        </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

Výsledek:

```
<root>
```

```
<občan>
  <jméno />
  <příjmení />
  <narození>
    <den>19</den>
    <měsíc>červenec</měsíc>
    <rok>1991</rok>
  </narození>
</občan>
<občan>
 .
 .
</občan>
</root>
```

2.5 XPath

V XPath lze zapisovat jednoduché výrazy, které vybírají části XML dokumentu. XML dokument je přitom chápán jako stromová struktura, kde jsou jednotlivé prvky, atributy a text chápány jako uzly (viz Obrázek 1)[3].

Využití Xpath není limitováno pouze na XSLT a XPoint, ale využívá se všude tam, kde potřebujeme vyhledávat v XML dokumentu specifická data. Je to standart vydaný konsorciem W3C [11].

Výrazy v XPath jsou podobné zápisu cest ve struktuře adresářů. Kdybychom například chtěli získat počet občanů narozených v roce 1991, použijeme XPath výraz /občan/narození[@rok='1991']. Lomítko přitom odděluje jednu úroveň ve stromu. Znamená to, že položka musí být potomek občan a atribut rok musí obsahovat hodnotu 1991. Pokud nám na hloubce vnoření nezáleží, použijeme dvě lomítka bezprostředně za sebou – hledají se pak všichni potomci v libovolné úrovni stromu //narození[@rok='1991']. Pomocí @ se ve výrazu odkazujeme na atribut, výraz '//@rok' nám vyhledává atribut rok v jakémkoli prvku. V případě, že chceme zredukovat výběr, použijeme hranaté závorky, kam zapíšeme podmínku například [@rok='1991'], která nám zredukuje výběr jen na prvky, které mají atribut rok s hodnotou 1991.

K dispozici jsou dále funkce pro práci s řetězci, pro základní výpočty, pro pohyb po stromové struktuře a samozřejmě i logické spojky[11].

3 Slovník

Slovníky jsou uložené seznamy slovní zásoby v jednom nebo více jazycích, nejčastěji řazené v abecedním pořadí, spolu s informacemi o definici, výslovnosti, oblasti použití a další užitečné informace, které nám lépe vysvětlí smysl daného slova (výkladové slovníky) nebo to může být seznam slov v jednom jazyce s dalším významem v jiném jazyce (překladové slovníky). Sestavováním slovníků se zabývá lingvistická disciplína zvaná lexikografie.

Ve většině slovnících jsou slova zachycena pouze ve svém základním tvaru, tzv. lemmatu, přestože např. slovníky v rámci jazykových učebnic mohou obsahovat i nepravidelné tvary.

Slovníky se s historického hlediska nejčastěji vyskytují v knižní podobě. V poslední době s nástupem výpočetní techniky se stále častěji objevují slovníky v elektronické podobě. Slovníky, které jsou pouze převedeny z knižní podoby do elektronické a slovníky, které vznikají od základu znovu pouze v elektronické podobě[12]. Jedním z takovýchto projektů je Wikislovník[13].

3.1 Typy slovníků

Podle rozsahu můžeme slovníky dělit:

- slovníky malé, kapesní (do 10 000 hesel)
- slovníky střední (50 000 - 60 000 hesel)
- slovníky velké (nad 60 000 hesel)

Podle typu:

- slovníky výkladové, které jsou zpravidla napsány v jednom jazyce. U každého slova je možné nalézt informace, které nám pomohou určit význam, použití a případně další ekvivalenty daného slova. Tyto informace jsou napsány ve stejném jazyce.

  Lze je rozdělit:

  - slovníky současného jazyka
  - významové (též sémantické, výkladové) – s definicí každého hesla (v témže jazyku)
  - pravopisné
  - frazeologické (idiomatické)
  - slovníky synonym
  - slovníky rýmů
  - slovníky cizích slov

    - slovníky zkratek
    - slovníky dialektů
    - slovníky slangu
    - slovníky argotu
    - slovníky citátů
- slovníky jednotlivých historických období
- slovníky etymologické (s genezí každého hesla)
- slovníky popisující slovní zásobu profesních skupin
- speciální

- slovníky překladové (vícejazyčné, polyglotické), které slouží pro překlad z jednoho jazyka do druhého. Ke slovům z jednoho jazyka obsahují jeho překlad v druhém jazyce, často i s výslovností, komentářem, frázemi a příklady, nebo jinými doprovodnými informacemi. Některé větší překladové slovníky obsahují i druhou část, ve které jsou slova pro zpětný překlad z druhého jazyka do prvního. Tyto slovníky mohou být i specializované, například se omezovat jen na odborné termíny z některé oblasti.

3.2 LMF

Lexical Markup Framework je standard Mezinárodní organizace pro normalizaci ISO (International Organization for Standardization 24613:2008), pro zpracování přirozeného jazyka (Natural Language Processing) a strojově čitelné slovníkové zdroje (Machine-Readable Dictionary) založené na XML.

Hlavním cílem LMF je jednotný model pro vytváření a využívání elektronických slovníků, od těch nejmenších až po rozsáhlé slovníky, řídit výměnu dat mezi dvěma nebo více slovníky a usnadnit slučování velkého počtu různých slovníků do formy rozsáhlých globálních slovníků. Konečným cílem LMF je vytvořit modulární strukturu, která umožní skutečnou součinnost ve všech aspektech elektronických slovníkových zdrojů.

Slovníky LMF jsou reprezentovány pomocí kódování Unicode, názvy atributů jsou založeny na normě Data Category Registry[14].

LMF je rozděleno do kolekce balíčků, z nichž každý má svůj specifický význam. Pro všechny slovníky je určeno jádro, které obsahuje elementární prvky. Užití dalších balíčků záleží na konkrétním určení slovníku.

3.2.1 Data Category Registry

LMF je rozděleno do kolekce balíčků, z nichž každý má svůj specifický význam. Pro všechny slovníky je určeno jádro, které obsahuje elementární prvky. Užití dalších balíčků záleží

na konkrétním určení slovníku. Data Category Registry je standart ISO 12620:2009, který slouží pro registraci lingvistických termínů, které se mohou použít v různých systémech, a tím lze zajistit základní předpoklady pro kompatibilitu mezi různými systémy slovníků. Registr je dost obsáhlý a díky tomu není úplně nejlehčí nalézt výraz, který hledáme. Proto se často stává, že autoři vytváří svoje vlastní názvosloví a tím se výhoda registru snižuje[15].

Hlavním účelem registru je sjednotit používané výrazy ve slovnících, které vyjadřují stejnou informaci. Pokud chceme říci, že slovo je v určitém mluvnickém čísle, dočteme se v registru u vlastnosti grammaticalNumber, že doporučené hodnoty jsou dual, massNoun, massNumber1, plural, singular. Bohužel někteří autoři slovníku tyto hodnoty nedodržují a vytváří si svoje hodnoty nebo je zkracují např. plural => pl., singular => sin. [16].

Většina lingvistických pojmů uložená v registru je v angličtině, až od roku 2009 byla přidaná vazba na další jazyk, a to na francouzštinu.

3.2.2 Základní balíček

Základní LMF balíček je sestaven z 11 základních tříd, které slouží pro základní tvorbu LMF modelu[12].

- Lexical Resource – slouží jako kořenový uzel pro celý LMF slovník. Ve slovníku se nachází pouze jednou
- Global Information - obsahuje administrativní a obecné informace o celém slovníku, nejméně musí obsahovat atribut language coding
    - Language coding – specifikuje standart kódování názvu jazyků.
    - Script coding – specifikuje standart kódování názvu písma.
    - Character coding – verze Unicode.
- Lexicon - reprezentuje jeden slovník v souboru. Musí obsahovat jeden nebo více lexikálních záznamů
- Lexicon - reprezentuje jeden slovník v souboru. Musí obsahovat jeden nebo více lexikálních záznamů.
- Lexical Entry - reprezentuje jeden lexikální záznam ve slovníku. Jedná se o zapouzdření souvisejících tvarů a významů.
- Form - abstraktní třída vyjadřující jeden lexém, morfologickou variantu lexému nebo morfému.
- Form Representation - reprezentuje jednu z možností zápisu lexému, tam kde jich je více.
- Representation - abstraktní třída pro Unicode řetězec popisující specifický jazyk, písmo a pravopis.
- Sense - reprezentuje jeden z lexikálních významů a umožňuje významy hierarchicky vnořovat.

- Definition - představuje slovní popis významu. Usnadňuje uživatelům pochopit význam Lexical Entry.
- Statement – obsahuje výpravný popis a zpřesňuje nebo doplňuje obsah Definition.
- Text Representation - představuje jednu z možností textového obsahu. Obsahuje řetězec v kódování Unicode s volitelným upřesněním písma, jazyka či pravopisu.
    - language – jazyk užitý ve vyjádření.
    - orthography name – název pravopisu.
    - script – písmo, kterým je vyjádření psáno.
    - written form – obsahuje samotné vyjádření.

3.2.3 Rozšíření základního balíčku

Lexical Markup Framework obsahuje vedle základního balíčku také několik rozšíření [12].

- Lemma - reprezentuje slovní tvar vybraný podle konvence k označení lexikálního záznamu. Lexikální záznam obsahuje Lemma právě jednou. Příklad používaných vlastností
    - written form – psaná podoba
    - phonetic form – výslovnost
    - geographical variant – specifická forma užívaná v některém regionu
    - scheme – vzor pro vytváření tvarů slova
- WordForm - tvar lexému používaný ve větě nebo frázi. Příklad používaných vlastností:
    - written form – psaná podoba
    - phonetic form – výslovnost
    - hyphenation – dělení slova, například na konci řádku
    - grammatical number – číslo
    - grammatical gender – rod
    - grammatical tense – čas, typicky slovesný, udávající k jakému období se forma slova vztahuje
    - person – osoba
- RelatedForm – reprezentuje slovní tvar nebo morfém, který může souviset s lexikálním záznamem, ve kterém je obsažen. Jedná se například o slovo odvozené. Příklad používaných vlastností:
    - written form – psaná podoba
    - phonetic form – výslovnost
    - type – udává příslušnost k určité skupině věcí nebo osob majících podobné vlastnosti
- Equivalent - ve dvojjazyčném slovníku reprezentuje informaci o překladu hesla, které se nachází v uzlu Lemma. Překladů může být více, stejně tak nemusí být obsažen vůbec žádný, poté se jedná o jednojazyčný slovník.
- Context - reprezentuje kontext užití daného slova. Může být obsažena vícekrát nebo vůbec.

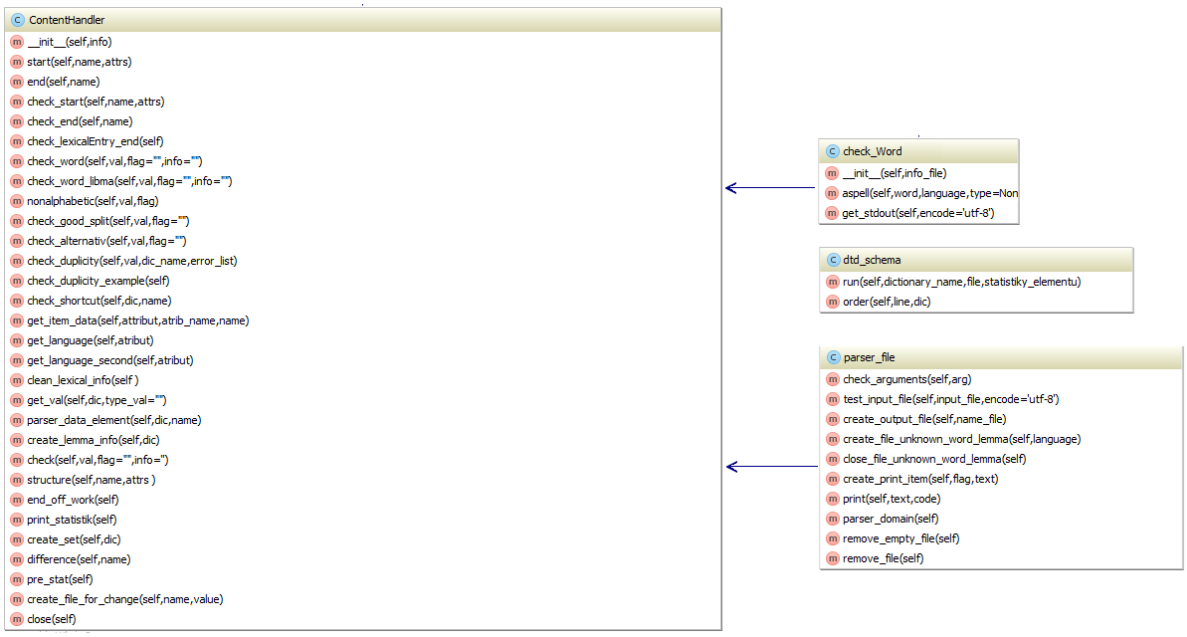
4 Návrh systému

Pro implementaci systému byl zvolen objektový styl programování v jazyce Python3. Systém byl rozdělen na dva samostatné logické celky. První celek slouží pro analýzu slovníků a jako výstup dostaneme statistiky daného slovníku. Druhý celek se zabývá hromadnými změnami slovníků, na výstupu dostaneme nový slovník s upravenou podobou.

4.1 Statistiky

Systém pro vytvoření statistik byl rozdělen do 4 tříd (Obrázek 2), kde každá třída provádí svoji specifickou činnost.

- Parser_file - třída, která zajistí zpracování vstupních parametrů, načtení a převedení vstupních dat z pomocných souborů (samotný slovník načítá lxml).
- Dtd_schema - třída, která má za úkol vytvořit DTD schéma, doplnit ho o počet výskytů značek a atributů ve slovníku.
- Check_Word - třída, která provádí vyhodnocení správnosti slov pomocí aplikací GNU Aspell a libma.
- ContentHandler - třída, která zpracovává informace získané ze slovníku. Obsahuje instanci třídy check_word a parser_file.



Obrázek 2 Diagram tříd systému statistik

4.2 Hromadné změny

Systém pro hromadné změny je rozdělen na dvě části. První, která změny provádí pomocí XMLGeneratoru (Obrázek 3) a druhá, která změny provádí pomocí XSLT šablon (Obrázek 4).

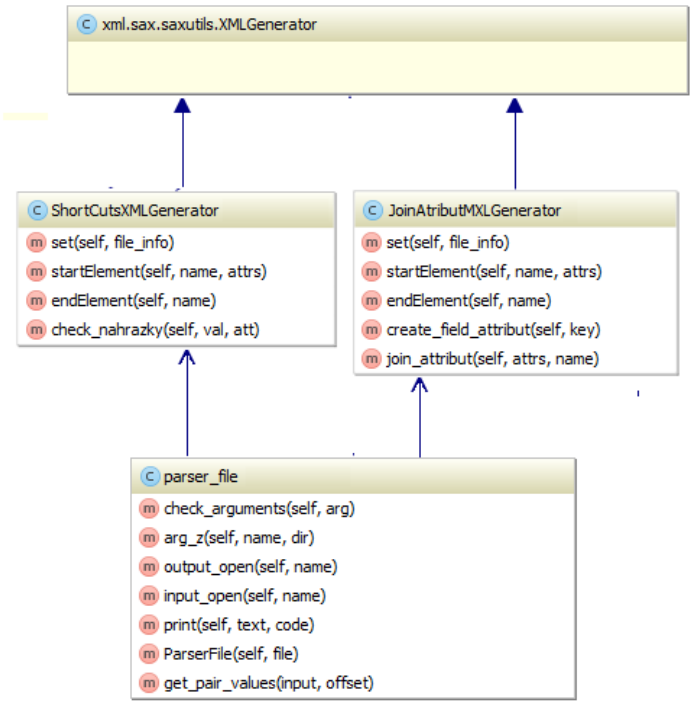
Obě části využívají instanci třídy parser_file, která zajistí zpracování vstupních parametrů, načtení a převedení vstupních dat z pomocných souborů.

Část, která provádí změny pomocí XMLGenerator, je rozdělena do 2 tříd.

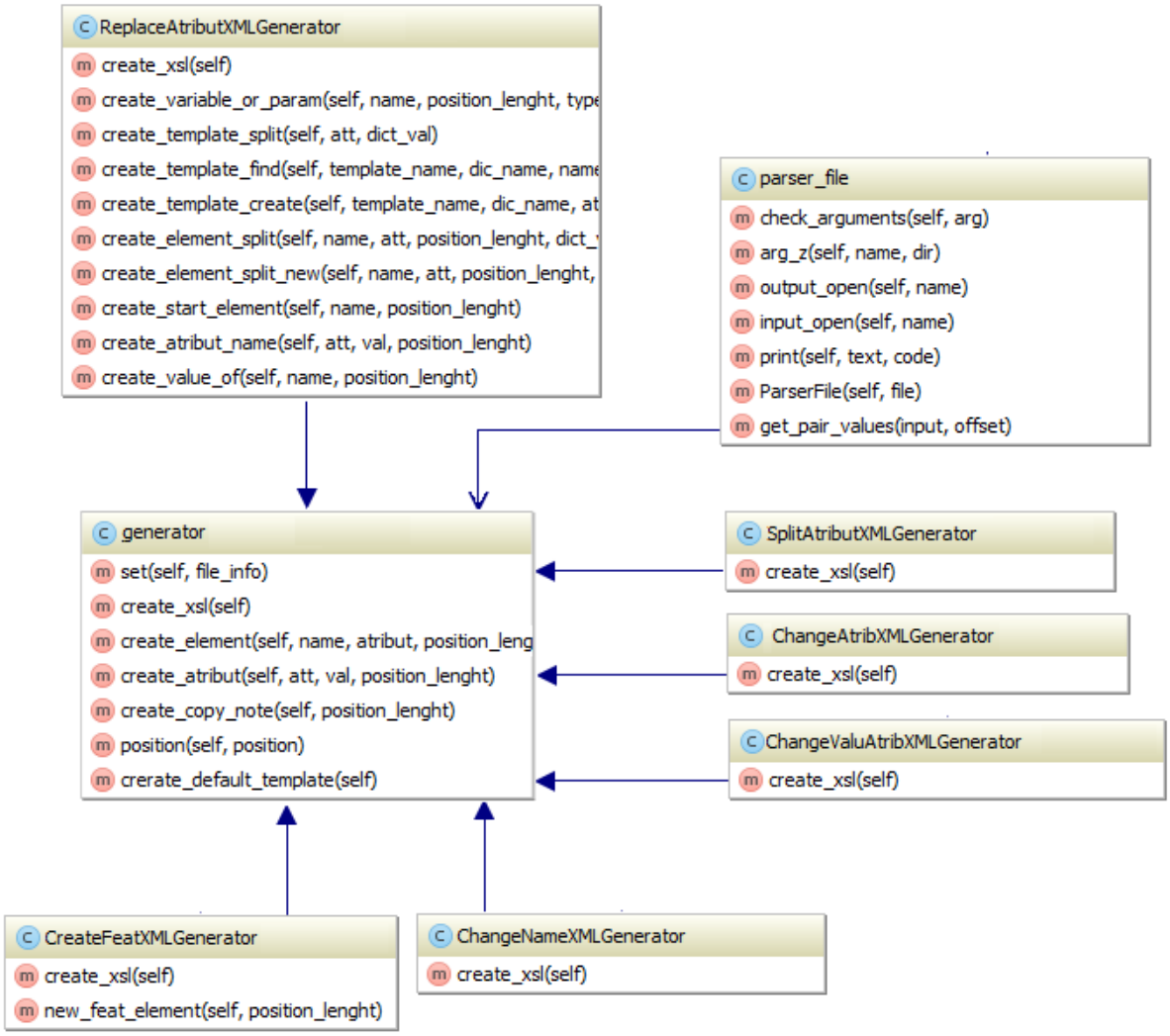
- ShortCutsXMLGenerator - třída, která provádí nahrazování zástupek.
- JoinAtributMXLGenerator - třída, která provádí sloučení informací v jednotlivých značkách.

Provádění změn pomocí XSLT šablon je rozděleno do 7 tříd, kdy třída generator obsahuje operace, které využijeme při každém vytváření šablony. Další třídy dědí od třídy generator a přidávají funkce specifické pro jejich úlohu.

- SplitAtributXMLGenerator - třída, která provádí rozdělení informace.
- ChangeAtribXMLGenerator - třída, která provádí nahrazení atributu att ve značce feat.
- ReplaceAtributXMLGenerator - třída, která provádí nahrazení podle seznamu pravidel.
- ChangeValuAtribXMLGenerator - třída, která provádí nahrazení atributu val ve značce feat.
- CreateFeatXMLGenerator - třída, která z atributu vytvoří značku feat.
- ChangeNameXMLGenerator - třída, která provádí změnu názvu značky.



Obrázek 3 Diagram tříd systému hromadných změn pomocí XMLGeneratoru



Obrázek 4 - Diagram tříd systému hromadných změn pomocí XSLT

4.3 Použité nástroje

4.3.1 GNU Aspell

GNU Aspell je Open Source aplikace, která slouží pro kontrolu pravopisu. GNU Aspell může být využíván jako knihovna pro rozšíření jiných aplikací nebo jako samostatná aplikace.[13] Lze ji využít i pro kontrolu slov, jestli ve specifikovaném jazyce opravdu existují. Na serveru minerva1 jsou uloženy slovníky pro jazyky angličtina, ruština, slovenština, latina, němčina, francouzština a čeština. Český jazyk není využíván, ale je kontrolován pomocí MA 4.3.2.

4.3.2 Morfologický slovník a morfologický analyzátor pro češtinu

Je zaměřen na podrobnou analýzu slov v českém jazyce. Lze získat informace o slovním druhu, pádu, číslo a další informace.[14] Podobně jako GNU Aspell, lze využít jako knihovnu (libma) pro další aplikace nebo jako samostatnou aplikaci MA. Pomocí MA je kontrolován pouze český jazyk.

4.3.3 Modul SAX

4.3.3.1 XMLGenerator

Z knihovny xml je použito rozšíření XMLGenerator, které zajistí funkci parseru pro čtení a vyhodnocení souboru, tak i následného vytvoření souboru XML.

4.3.3.2 Parser

Pro parsování XML souboru je použit parser z knihovny lxml. Má podobné vlastnosti jako parser z knihovny xml (2.2.2) v Python3, ale dosahuje mnohem vyšší rychlosti ve zpracování souborů.[15] Knihovna je distribuována pod licencí BDS.

4.3.4 DTDGenerator

Aplikace slouží pro vytvoření DTD (Document Type Definition 2.3.1) na základě vstupního souboru XML. DTDGenerator je napsán v jazyce Java a pro parsování XML souboru využívá SAX (2.2.2). Aplikace je distribuována pod licencí Mozilla Public Version 1.0.[16]

4.3.5 Saxon

V mém případě je využívána distribuce Saxon-HE (Home edition), která je distribuována pod licencí Mozilla Public Version 1.0.[17] Aplikace je použita pro transformaci XML souboru, pomocí XSLT šablony.

4.3.6 cProfile

Modul cProfile slouží pro měření času spuštěného programu [18]. Modul neměří pouze celkový čas, který byl program spuštěn, ale i čas strávený v jednotlivých funkcích. Dále nám počítá i počet volání jednotlivých funkcí a čas, který byl potřeba v průměru na jedno volání funkce. Díky těmto informacím můžeme snadno zjistit, kde se nám vyplatí optimalizovat kód.

5 Implementace

5.1 Ovládání

Ovládání aplikace je pouze konzolové. Zpracování atributu, které jsou předány při spuštění programu, je prováděno pomocí knihovny argparse.[19] Tato knihovna nabízí řadu výhodných funkcí, jako automatické generování nápovědy, hlídání zadaných povinných atributů a řadu dalších nastavení, které se využijí při konzolovém ovládání.

5.2 Statistiky

Jednotlivá vyhodnocování získaných informací se vždy provádí po načtení celé značky. V tomto případě se značka feat používá pouze jako nositel informace o nadřazené značce, jako například značky Lemma, Equivalen, Sense atd..

V dalším textu již nebudu zmiňovat, že informace jsou převážně vždy uloženy ve značce feat, název informace je uložen v atributu att a hodnota této informace je uložena v atributu val.

Je pouze pár výjimek, kdy je informace uložena přímo ve značce. Jedná se o atribut id, který se vyskytuje u značek LexicalEntry, Sense jako identifikátory záznamu ve slovníku a SenseRelation, kde slouží pro uložení id příbuzného významu.

5.2.1 Kontrola slov

Kontrolu slov je možné provést dvěma způsoby. Záleží na tom, jak přesnou kontrolu chceme provést. V případě použití GNU Aspell / libma budou všechna slova zkontrolována, jestli existují v daném jazyce. V případě hledáni neodpovídajících znaků, hledáme pouze znaky, které by se neměly vyskytovat v daném jazyce.

Kontrola slov se provádí ve všech značkách s informací o writtenForm a text. U jiných informací se tato kontrola neprovádí.

5.2.1.1 GNU Aspell / libma

V první fázi implementace byla kontrola řešena pomocí programu GNU Aspell/libma a při každém nalezení informace writtenForm a text byla provedena kontrola. Komunikace s programem aspell se prováděla pomocí knihovny subprocess [20].

Toto řešení mělo výhodu v tom, že slova byla vyhodnocována okamžitě při zpracování jednotlivých značek. Díky tomu se daly velmi lehce vypsat další doplňující informace, jako například značka, ve které se nachází daný znak, ID lexému, celá hodnota informace, jestli se nejedná jen o název, kde bylo použito české slovo např. ČR1 Radiožurnál a další užitečné informace. Tyto

informace mohly být použity k určení, jestli je opravdu problém ve slovníku, bez další potřeby studovat původní slovník.

Toto řešení, ale mělo bohužel zásadní rychlostní problém. Doba, kterou bylo potřeba pro zpracování slovníku o velikosti 150 mb, se pohybovala na hranici 6 minut. Problém byl v příliš časté komunikaci pomocí subprocess.

V další fázi bylo přistoupeno k jinému způsobu zpracování, všechny slova jsou uložena do dočasného souboru. Soubory jsou vždy rozděleny podle jazyků. V okamžiku, kdy je celý XML slovník zpracován, je přistoupeno ke zpracování vytvořených dočasných souboru. Soubory jsou podle jazyku předány příslušnému programu.

Pro češtinu se vždy používá libma, pro jazyky, jako je francouzština, němčina, angličtina, latina, ruština a slovenština, je použit GNU Aspell. V případě, kdy je nalezen jazyk, který nejsem schopen zkontrolovat, je celý soubor přesunut do výsledné složky se statistikami. Pro další kontrolu, kterou může provést sám uživatel.

V tomto případě se rozlišují nejen nalezená slova podle jazyků, ale i podle umístění. Klíčová slova, která jsou uložena ve značce Lemma, jsou v samostatném souboru. Pouze u klíčových slov je zachováno vypisování gramatické informace.

5.2.1.2 Hledání neodpovídajících znaků

Kontrola hledá ve slovech znaky, který do dané abecedy nepatří. Hledání se provádí pomocí regulárních výrazů.

U této metody je velká pravděpodobnost, že nebudou odhalena slova, která jsou napsána jiným jazykem, než je uvedeno v jazykové informaci v dané značce. Například slovo pes, bude vyhodnoceno ve většině jazyků jako správné, protože se skládá pouze ze základních znaků ASCII. Těchto příkladů je možné nalézt celá řada, hlavně mezi jazyky psané latinkou.

Tento způsob kontroly slov bude dosahovat nejlepších výsledků mezi jazyky, které používají velmi odlišnou abecedu, například mezi češtinou a ruštinou.

Jazyky, které jsou podporovány: čeština, francouzština, němčina, angličtina, latinka, ruština a slovenština.

5.2.2 DTD schéma

DTD schéma se vytváří pomocí programu DTDGenerator (4.3.4). DTDGenerator vytvoří standardní schéma, které můžeme použít pro validaci s jiným XML souborem.

Toto schéma nejprve upravím, tak aby se vypisovalo v pořadí výskytu ve slovníku, tato úprava se týká značek i atributů.

Dále vytvořím upravené DTD schéma, které má navíc ještě počet výskytů atributů ve značce (vypočítávám jen z hodnot atributu att), počet výskytu značky v celém slovníku a počet výskytů zanořených značek ve značce.

Příklad slovníku s jedním klíčovým slovem:

```
<LexicalResource dtdVersion='16'>
    <GlobalInformation>
        <feat att='languageCoding' val='ISO 639-3'/>
    </GlobalInformation>
    <Lexicon>
        <feat att='language' val='ces'/>
        <LexicalEntry id='e1g'>
            <feat att='type' val='phrase'/>
            <Lemma>
                <feat att='writtenForm' val='a to'/>
                <feat att='writtenForm' val='a sice'/>
            </Lemma>
            <WordForm>
                <feat att='writtenForm' val='a to'/>
                <feat att='writtenForm' val='a sice'/>
            </WordForm>
            <Sense>
                <feat att='senseNumber' val='1'/>
                <Equivalent>
                    <feat att='language' val='eng'/>
                    <feat att='gloss' val='jmenovitě'/>
                    <feat att='writtenForm' val='namely'/>
                    <feat att='writtenForm' val='that is'/>
                </Equivalent>
            </Sense>
        </LexicalEntry>
    </Lexicon>
</LexicalResource>
```

Příklad výstupního DTD schematu:

```
<!ELEMENT LexicalResource(1) (  GlobalInformation (1)| Lexicon (1))* >
<!ATTLIST LexicalResource dtdVersion NMTOKEN #REQUIRED >
<!ELEMENT GlobalInformation(1) (  feat (1))* >
<!ELEMENT feat(12) EMPTY >
<!ATTLIST feat att ( languageCoding(1) | language(2) | type(1) | writtenForm(6) | senseNumber(1) | gloss(1) )>
<!ATTLIST feat val CDATA #REQUIRED >
<!ELEMENT Lexicon(1) (  feat (1), LexicalEntry (1)) >
<!ELEMENT LexicalEntry(1) (  feat (1), Lemma (1), WordForm (1), Sense (1)) >
<!ATTLIST LexicalEntry id NMTOKEN #REQUIRED >
<!ELEMENT Lemma(1) (  feat+ (2)) >
<!ELEMENT WordForm(1) (  feat+ (2)) >
<!ELEMENT Sense(1) (  feat (1), Equivalent (1)) >
<!ELEMENT Equivalent(1) (  feat+ (4)) >
```

Pro vysvětlení, co je myšleno počtem výskytů zanořených značek, si uvedeme příklad. U značky Sense je zjištěno, že se ve slovníku vyskytuje pouze jednou, <!ELEMENT Sense(1)‘, a celkově obsahuje jednu zanořenou značku feat a Equivalen.

Tento proces patří k nejzdlouhavějším procesům celého systému. U každé značky, kterou ve slovníku najdu, musím nejprve zjistit, jestli je to první nález nebo již další. Musím pokaždé inkrementovat počítadlo jak počtu celkového výskytu ve slovníku, tak počtu zanořených značek a upravit počet výskytu atributů.

V případě rozsáhlejších slovníků, kde není problém mít přes 3 miliony značek ve slovníku, jen funkce na zpracování výskytů, si v původním návrhu zabrala kolem 20% celkového času.

Nejprve byla použita běžná konstrukce:

```
if  značka|atribut in slovník:
        inkrementuj
else:
        přidej do slovníku
```

V prvních optimalizacích jsem se zaměřil na úpravy použitých konstrukcí if else v krajních hodnotách a vyřešil jsem to pomocí úpravy datových struktur. Stále bylo dosaženo jen velmi malého časového zlepšení, proto jsem přistoupil k jinému řešení.

Pomocí odchytávání výjimky, přesněji výjimky KeyError, která vznikne při použití klíče (atribut, název značky) ve slovníku, který takový klíč neobsahuje. Vzhledem k tomu, že se běžně ve slovnících vyskytuje okolo 10 různých značek a 8 různých druhů atributu att, výjimky se nebudou příliš často vyvolávat. Tímto řešením jsem dosáhl zrychlení funkce přibližně o 35%.

5.2.3 Duplicity

Ve slovníku XML vyhledávám duplicitní záznamy u tří typů značek. Ve značce Lemma hledám duplicity klíčových slov, ve značce wordForm vyhledávám duplicity jiných tvarů klíčového slova a ve značce equivalent vyhledávám duplicitní překlad klíčového slova.

V případě značky Lemma, vyhledávám duplicity v rámci celého slovníku, u značek wordForm a equivalent, hledám duplicity vždy jen v rámci jednoho záznamu slovníku (LexicalEntry).

U značek wordForm a equivalent převedu všechny její informace do textového řetězce, seřazeny abecedně podle názvu informace. Ve formátu „název informace: hodnoty informace“.

U vyhledávání duplicitních klíčových slov, je rozdíl pouze v informacích, které použiji. Použiji všechny informace uložené ve značce Lemma a k těm to informacím, přidám ještě informace uložené přímo v LexicalEntry jako jsou type, partOfSpeech atd. .

5.2.4 Rozpoznání jazyků

Formát LMF využívá značku GlobalInformation, pro informaci o použitých jazycích ve slovníku. Podle použitých informací language, fromLanguage a toLanguage jsem schopen vyhodnotit, jaký jazyk je považován jako defaultní.

V případě, kdy se v nějaké značce nevyskytuje informace, podle které lze vyhodnotit, v jakém jazyce jsou informace zapsány, je použit defaultní jazyk. Takovéto vyhodnocení se nejvíce používá u značek, napsaných v jazyce slovníku, kde se zpravidla nepíše jazyková informace.

V případě, kdy je špatně uveden jazyk, ve kterém je daná značka zaspána, dojde ke špatnému vyhodnocení kontroly slov (5.2.1).

Když není uvedena značka GlobalInformation, jako defaultní jazyk, bude použit první nalezený název jazyka.

5.2.5 Seznam použitých hodnot

Defaultně se vytváří seznam použitých hodnot pro informace partOfSpeech, type, grammar, domain, hint, gloss, register. Dále je možné informace, pro které vytvářím seznam použitých hodnot, rozšířit pomocí souboru.

Pro každou informaci je vytvořen samostatný soubor, kde je vždy vypsán seznam všech použitých hodnot. Tento seznam je vytvořen sestupně od nejvíce se vyskytující se hodnoty po nejméně se vyskytující hodnotu. Dále je uvedeno, kolikrát se hodnota vyskytla v dané značce a v jakém jazyce se použila.

Každá nalezená hodnota je dále rozdělena pomocí těchto znaků ,;\s()/\| na jednotlivé části. Až z těchto částí se provádí výpočet výskytů a dalších statistik. Stejným způsobem je zpracován i vstupní soubor se zadanými hodnotami. Tento proces byl zaveden pro zjednodušení zadávání povolených hodnot do souboru. Pokud bych tento způsob nezavedl, muselo by se vypisovat velké množství povolených značek do souboru.

Dále se velmi často objevují případy, kdy se dvě hodnoty liší pouze tečkou na konci. Proto se při zpracování vstupního souboru vytvoří vždy dvě hodnoty, jedna původní a druhá, která bude opačně k původní s nebo bez tečky na konci. Ve výsledném seznamu použitých hodnot, se mohou objevy obě varianty, ale nebudou hlášeny jako špatně použité, nebo že nebyly nalezeny v povolených hodnotách.

V případě, kdy bude zadán soubor s povolenými hodnotami pro danou informaci, vygenerují se další statistiky v případě, že bude zadáno pouze jméno informace. Bez dalších povolených hodnot se další statistiky vytvářet nebudou.

V případě, kdy se zadají hodnoty pro srovnání do souboru, bude provedeno srovnání nalezených hodnot ve slovníku s hodnotami zadanými v souboru. Budou se vypisovat takové hodnoty, které nebudou nalezeny v souboru a hodnoty, které se nacházejí v seznamu povolených hodnot u jiných informací.

Soubor s povolenými hodnotami má následující podobu:

```
Source tag: název informace
Název hodnoty v původním jazyce ‘tabulátor‘ název hodnoty v jiném jazyce
...
```

Tento soubor je možné po menších úpravách použít pro hromadné změny (5.3.2.7) a naopak lze použít soubor pro hromadné změny pro kontrolu výskytu hodnot bez úpravy.

Příklad souboru s povolenými hodnotami:

```
Source tag: partOfSpeech:
adjective	přídavné jméno
adv	adverb (příslovce)
3.sg	3. osoba jedn. č.
particle

Source tag: gloss
hotovo	done
to bychom	it would
```

Část slovníku:

```
<LexicalEntry id='e3g'>
		<feat att='partOfSpeech' val='particle'/>
		<Lemma>
			<feat att='writtenForm' val='a'/>
		</Lemma>
		<WordForm>
			<feat att='writtenForm' val='a'/>
		</WordForm>
		<Sense>
			<feat att='senseNumber' val='1'/>
			<Equivalent>
				<feat att='language' val='eng'/>
				<feat att='writtenForm' val='and'/>
			</Equivalent>
			<Context>
				<TextRepresentation>
					<feat att='languageIdentifier' val='eng'/>
					<feat att='gloss' val='to bychom měli'/>
					<feat att='gloss' val='hotovo'/>
					<feat att='gloss' val='příslovce'/>
					<feat att='text' val='And that‘s it!, So that‘s that!, Done!, And bob‘s your uncle.'/>
				</TextRepresentation>
			</Context>
		</Sense>
	</LexicalEntry>
```

Výstupní soubor pro gloss:

```
Statistiky pro gloss

Byly nalezeny hodnoty, které se mají vyskytovat ve značce partOfSpeech
příslovce                       výskyt v jazycích       eng


Hodnoty, které jsou navíc oproti definovaným v souboru
měli                            výskyt v jazycích       eng
příslovce                       výskyt v jazycích       eng


bychom                          [1, {'TextRepresentation': 1}, {'eng'}]
to                      [1, {'TextRepresentation': 1}, {'eng'}]
příslovce                       [1, {'TextRepresentation': 1}, {'eng'}]
měli                            [1, {'TextRepresentation': 1}, {'eng'}]
hotovo                          [1, {'TextRepresentation': 1}, {'eng'}]
```

5.2.6 Další statistiky

Při každém spuštění systému bez doplňujících přepínačů se automaticky vytváří soubor se souhrnnými statistikami, kde jsou informace o počtu klíčových slov ve slovníku, počet chyb, které byly zaznamenány, výpis všech názvů jazyků ve slovníku, nejvíce se opakující hodnoty z informací domain, hint a partOfSpeech, celkový počet nalezených duplicitních informací a celkový počet nalezených nealfabetických slov.

Dále vytváří soubory, které obsahují informace o příkladech použitých alternativ, jak se zapisuje výslovnost, soubor se seznamem nealfabetických slov, hodnoty vhodné pro rozdělení a výpis nalezených zástupek.

Zástupky se vypisují do dvou různých souborů podle typu zástupky. Zástupky složené pouze z teček se vypisují do jednoho souboru a zástupky, které obsahuji i jiné znaky kromě teček, se vypisují do druhého souboru.

5.2.7 Chyba syntaxe

V případě, že vstupní XML slovník bude obsahovat chybnou syntaxi, do souboru se souhrnnými statistikami se zapíše informace o vzniklé chybě a všechny soubory mimo soubor statistics budou smazány.

5.2.8 Vzorový soubor pro změnu

S vytvořením statistik lze vytvořit i soubor, který poslouží jako základ pro hromadné změny. Vytvoří se soubor (5.3.2.7), který bude obsahovat pouze původní hodnoty a na uživateli je doplnění hodnot, za které se mají změnit.

5.2.9 Zpracování XML dokumentu

Zpracování XML dokumentu je prováděno pomocí rozhraní SAX, které nemá velké paměťové nároky. Vzhledem k velikosti slovníků, kde není neobvyklé mít slovník přes 100mb, je rozhraní SAX nejlepší možnou variantou.

Zpracování souboru v případě vytváření statistiky, jsem v první fázi prováděl pomocí SAX parser (2.2.2), z knihovny xml, které je běžně dostupné u všech distribucí Python. Po dalším průzkumu a testech jsem zjistil, že řešení, které bylo použito, nebylo příliš rychlé ve srovnání s variantou, která je použita v knihovně lxml[15] a proto jsem systém převedl na SAX parser používaný v knihovně lxml.

Největší problém, s kterým jsem se při zpracování XML souboru setkal, byl ve špatném nahrazování syntaktických značek [21]. Pro tyto případy byl implementován skript, který nahrazoval syntaktické značky a správně vypočítával uvozovky v atributech.

Skript jednoduše načítá řádek po řádku celý soubor. Pomocí regulárních výrazů nahrazuje špatně uložené hodnoty atributů, které obsahují syntaktické značky a přitom vytváří syntakticky správný soubor.

5.3 Hromadné změny

5.3.1 SAX Generator

5.3.1.1 Nahrazení zástupek

Zástupka se používá v textu jako nahrazení klíčového slova. Může mít různou podobu. Zástupky, které jsem schopen nalézt:

- jedno písmeno a tečka
- posloupnost tří a více teček
- písmeno pomlčka písmeno

Zástupky hledám v hodnotách informací text, note, label a context. Zástupky nahrazuji za klíčová slova, která jsou uložena ve značce Lemma v informaci writtenForm a vždy nahrazuji jen v informacích, které jsou uloženy ve stejném jazyce jako klíčové slovo.

V případě, že Lemma obsahuje více writtenForm, jsem schopen správně určit, za kterou hodnotu writtenForm zástupku nahradit pouze v případě, že mají různé počáteční písmeno. V případě, že mají počáteční písmeno stejné a naleznu zástupku určenou k nahrazení, vypíši do souboru číslo

řádku, kde se tato zástupka nachází, aby uživatel později mohl sám rozhodnout, jak zástupku nahradí. V případě, že nebude použita značka Lemma, bude vypsáno varování do souboru.

O každé úspěšně nahrazené zástupce je vytvořen záznam do souboru s informací, na kterém řádku se zástupka vyskytuje, původní podoba zástupky a za jaké klíčové slovo byla nahrazena.

5.3.1.2 Sloučení informací

Ve slovnících je více podob, jak jsou hodnoty informací uloženy. Některé slovníky mají informaci pro danou značku uloženou vždy v jednom výskytu informace a pouze oddělené nějakým znakem, např. středníkem. Další využívají více výskytů dané informace v jedné značce. Sloučení informací lze využít v případě, kdy chceme mít slovníky stejně uloženy.

Sloučení informací se provádí pro značkou feat a vždy se slučují značky na základě hodnoty atributu att (název informace). Lze specifikovat, ve kterých značkách se má provádět sloučení informací, pro které typy informací a jakým znakem se mají spojované hodnoty oddělit. Defaultní hodnota na oddělení jednotlivých hodnot je čárka.

5.3.1.3 Zpracování XML dokumentu

SAXGenerator provádí dvě základní činnosti. První je stejná jako u SAX parser, provede parsování XML souboru, tak abychom mohli vyhodnotit, jestli danou značku budeme chtít i ve výstupním souboru a provést případnou změnu její podoby. Druhá činnost je generování XML souboru, podle námi zadaných hodnot.

SAXGenerator je použit pouze u nahrazení zástupek a sloučení značek. Řešení není příliš rychlé, ale bohužel tyto dva případy nebylo možné přepsat do úpravy pomocí XSLT.

5.3.2 XSLT

5.3.2.1 Šablona

Při těchto editacích se vždy ukládá i šablona XSLT, kterou je možné dále editovat. Je mnohem jednodušší si nechat vygenerovat tuto šablonu a poté ji upravit.

Toho lze využít například v případě, že budeme chtít, aby se hodnoty, které nebudou nalezeny v našem souboru (5.3.2.7) nekopírovaly do výsledného XML souboru. Takovýchto různých modifikací je celá řada a proto je vhodné, aby uživatel mohl provádět editace XSLT šablony.

Proto je přidána podpora pro samotné vygenerování XSLT šablony, načtení šablony a její spuštění bez přegenerování.

5.3.2.2 Rozdělení informace

Proces opačný ke sloučení informací (5.3.1.2). Lze zadat, ve které značce se má provádět editace a jaký typ informace se má rozdělit. Rozdělení je defaultně nastaveno na čárky, v případě, že budeme chtít rozdělit informaci podle jiného znaku, je ho možné zadat.

5.3.2.3 Vytvoření informace z atributu

U některých značek se vyskytuje informace přímo v atributu místo ve značce feat. V těchto případech, abychom sjednotili celý soubor, můžeme použít tuto editaci. Značka, z které se budou přesouvat atributy, můžeme určit pomocí Xpath (2.5). Je možné buď zadat, které atributy se mají přesunout do značek feat nebo v případě, že žádný nezadáme, budou se přesouvat všechny nalezené atributy v dané značce.

Značka feat se z atributu vytvoří tak, že název atributu bude použit jako hodnota atributu att ve značce feat a hodnota atributu bude použita jako hodnota atributu val ve značce feat.

Ve výsledném souboru se již atribut nebude vyskytovat, bude již pouze značka feat.

5.3.2.4 Změna názvu značky

Přejmenování stávající značky. V případě, že budeme chtít přejmenovat jen určité značky, můžeme použít k adresování značky Xpath (2.5).

Využít tento druh editace můžeme například v případě, kdy je značka ve slovníku zapsána víckrát s různou podobou.

Ve slovníku z dictform06 ameroget-LMF je značka feat vytvořena v odlišné podobě od ostatních slovníků a to, že běžně je značka zapsána jako feat, ale zde je použito Feat.

5.3.2.5 Změna hodnoty atributu att

Slouží pro přejmenování hodnoty v atributu att. Značku lze adresovat pomocí Xpath, ale vždy jen značku, která obsahuje atribut att.

5.3.2.6 Změna hodnoty atributu val

Slouží pro jednoduchou změnu hodnoty atributu val, v případě kdy chceme určit i ve kterých značkách se tato změna má provést. Změnu lze provést vždy jen pro jednu hodnotu.

Je to jednoduší způsob, než vytvářet seznam pravidel (5.3.2.7), ale velmi omezený. Proto je tato editace spíše vhodná, když chceme nahradit pouze velmi omezenou skupinu hodnot, například sjednotit názvy jazyků ve všech slovnících.

5.3.2.7 Editace na základě pravidel

Při editaci na základě pravidel je důležité mít správně vytvořený soubor. Jednotlivé hodnoty, které jsou nalezeny ve slovníku, jsou vždy rozděleny na menší části na základě znaků ,;\s()/\| , stejně jako u Seznam použitých hodnot. Na rozdíl od statistik, se zde již vstupní soubor neupravuje.

Je vhodné si nechat vygenerovat vzorový soubor (5.2.8), který obsahuje všechny hodnoty, v podobě jaké je bude program dále vyhledávat a nahrazovat.

Soubor s pravidly slouží pro dvě různé úpravy a vždy můžeme využít obě úpravy naráz nebo jen jednu z nich. Pravidla jsou aplikovány na značky feat podle zadané hodnoty atributu att, shoda na hodnotu je hledána v atributu val.

První úprava spočívá v nahrazování hodnot v rámci jedné značky. Podle nalezené shody může být nahrazena část hodnoty, celá hodnota nebo vůbec žádná část hodnoty atributu val. Nevytváří se zde žádná nová značka.

Druhá úprava vytváří nové značky podle pravidel. Hodnota značky se skládá jen z hodnot, které byly rozpoznány. Hodnoty, které nebyly nalezeny v seznamech, nebudou dále propagovány do nové značky.

Soubor s pravidly má jednoduchou strukturu, nejprve určíme, kterou hodnotu atributu att chceme upravovat pomocí pravidel:

Source tag: název hodnoty atributu

Poté můžeme určit hodnoty, které se budou měnit:

allowed values:
původní hodnota 'tabulátor' nová hodnota
původní hodnota 'tabulátor' nová hodnota

Určit nové značky, které vzniknout v případě, že nalezneme shodu:

change to: název atributu att pro novou značku 1
původní hodnota 'tabulátor' nová hodnota
původní hodnota 'tabulátor' nová hodnota
…
change to: název atributu att pro novou značku 2
původní hodnota 'tabulátor' nová hodnota
….

Příklad souboru s pravidly:

```
Source tag: domain
allowed values:
AmE	americky
zkr.	zkratka
ling.	lingvistika
change to: origin
AmE	American English
zkr.	shortcut
ling.	linguistics

<!—Není nutné zadávat allowed values -->
Source tag: gloss
change to: origin
poznatky	knowledge
```

Část slovníku:

```
<LexicalEntry id='e34g'>
        <feat att='partOfSpeech' val='noun'/>
        <Lemma>
            <feat att='writtenForm' val='abeceda'/>
        </Lemma>
        <Sense>
            <Equivalent>
                <feat att='language' val='eng'/>
                <feat att='gloss' val='soustava psaných znaků'/>
                <feat att='writtenForm' val='alphabet'/>
            </Equivalent>
            <Context>
                <TextRepresentation>
                    <feat att='languageIdentifier' val='ces'/>
                    <feat att='domain' val='ling.'/>
                    <feat att='text' val='mezinárodní fonetická abeceda'/>
                </TextRepresentation>
                <TextRepresentation>
                    <feat att='languageIdentifier' val='eng'/>
                    <feat att='domain' val='zkr.'/>
                    <feat att='text' val='international phonetic alphabet'/>
                </TextRepresentation>
            </Context>
            <Equivalent>
                <feat att='language' val='eng'/>
                <feat att='gloss' val='základní poznatky'/>
                <feat att='writtenForm' val='ABC'/>
            </Equivalent>
        </Sense>
    </LexicalEntry>
```

Výsledný soubor:

```
            <LexicalEntry id="e34g">
        <feat att="partOfSpeech" val="noun"/>
        <Lemma>
            <feat att="writtenForm" val="abeceda"/>
        </Lemma>
        <Sense>
            <Equivalent>
                <feat att="language" val="eng"/>
                <feat att="gloss" val="soustava psaných znaků "/>
                <feat att="writtenForm" val="alphabet"/>
            </Equivalent>
            <Context>
                <TextRepresentation>
                    <feat att="languageIdentifier" val="ces"/>
                    <feat att="domain" val="lingvistika "/>
                      <feat att="origin" val="linguistics "/>
                    <feat att="text" val="mezinárodní fonetická abeceda"/>
```

```
                    </TextRepresentation>
                    <TextRepresentation>
                        <feat att="languageIdentifier" val="eng"/>
                        <feat att="domain" val="zkratka "/>
                          <feat att="origin" val="shortcut "/>
                        <feat att="text" val="international phonetic alphabet"/>
                    </TextRepresentation>
                </Context>
                <Equivalent>
                    <feat att="language" val="eng"/>
                    <feat att="gloss" val="základní poznatky "/>

        <!—Hodnota základní není uvedena v seznamu pravidel, proto se zde převedla jen hodnota poznatky -->
                    <feat att="origin" val="knowledge "/>
                    <feat att="writtenForm" val="ABC"/>
                </Equivalent>
            </Sense>
    </LexicalEntry>
```

5.3.2.8 Zpracování XML dokumentu

V první fázi vývoje systému pro hromadné změny, byly všechny úpravy prováděny pomocí SAXGenerator. Tento způsob nebyl příliš efektivní z časového hlediska.

Proto byl systém přepsán do podoby, kdy můj systém vygeneruje pouze XSLT šablonu, kterou poté zpracuje Saxon (4.3.5). Tímto řešením se rychlost provádění hromadných změn nad XML dokumentem zrychlila až o 80%.

6 Výsledky

Při vývoji systému byly testovány slovníky uložené na školním serveru minerva1 ve složce dicts2lmf a slovníky uložené ve složce ocr2lmf. Samotné testování probíhalo opět na školních serverech minerva1 a knot1.

Během testování bylo přegenerováno 189 slovníků, kde 9 mělo syntaktickou chybu ve vytvořených atributech. Po opravě pomocí skriptu, který byl vytvořen pro nahrazení syntaktických značek a správné nahrazení uvozovek v atributu (5.2), všech 9 souborů bylo v pořádku zpracováno.

Opravené slovníky jsou uloženy na serveru minerva1 ve složce dict_profile_and_unify/profile/new_dictionary.

Při testování byl vytvořen soubor pro kontrolu použitých hodnot v informacích (5.2.5). Pro informace PartOfSpecha, type a domain byl vytvořen seznam povolených hodnot.

U značek hint a gloss byl problém, protože se vyskytují ve velkém množství, každá značka má okolo 200 000 různých nálezů. Proto bylo potřeba povolené hodnoty z toho seznamu vygenerovat automaticky. Nejprve jsem provedl rozdělení značek podle toho, v jakém jazyce se použily v daném slovníku. V případě, že se použily ve více jazycích, hodnotu jsem dále nezkoumal. V případě použití v českém jazyce použil jsem libma a v ostatní jazycích byl použit program GNU Aspell. Tímto řešením jsem získal 28 000 povolených značek hint a 90 000 povolených značek gloss.

6.1 Statistiky

Měření času byla prováděna pomocí modulu cProfile (4.3.6). Měření byla provedena na školním serveru minerva1 na následujících 5 slovnících:

- ocr2lmf/slovnik_nespisovne_cestiny/lmf_dicts/snc-lmf.xml'
- dicts2lmf/dictform01/corrected/encz-lmf.xml
- dicts2lmf/dictform01/corrected/sksp-lmf.xml
- dicts2lmf/dictform02/lmf_dicts/encz_hh-lmf2.xml
- dicts2lmf/dictform03/corrected/zkraceny-lmf.xml

V případě kdy se program spustí s modulem cProfile, dojde k prodloužení času potřebného pro vytvoření statistik. Čas se navýší přibližně o 15%, oproti spuštění bez modulu cProfile. Časy, které budou udávány u jednotlivých funkcí, budou brány z modulu cProfile. Čas, který byl potřeba k proběhnutí celého programu, bude měřen při spuštění bez cProfile.

V následující tabulce (Tabulka 1) je souhrn rychlostních testů provedených pro část systému, která se zabývá statistikami. Nejprve jsou uvedeny 2 kontroly (pouze čas funkcí, které danou kontrolu

provádí), které v celém systému zabírají nejdelší dobu (DTD schéma a Kontrola slov) a srovnání vytvořených statistik v případě se všemi přepínači (Aspell, seznam použitých, vyhledávat duplicity atd.) se statistikami, které budou obsahovat pouze základ (základní statistiky (5.2.6) a slova se budou kontrolovat pomocí hledání neodpovídajících znaků).

V případě kontroly slov pomocí GNU Aspell/libma, je nejprve pozorován očekávaný nárůst doby potřebné pro zpracování větších slovníků, až pro slovník snc-lmf se čas potřebný pro zpracování zvýšil skokově. Tento skokový nárůst je způsoben tím, že všechna slova jsou kontrolována pouze pomocí libma(ma), libma bohužel nedosahuje takové rychlosti jako aspell a tím je způsobeno toto skokové zvýšení času. Na druhou stranu libma poskytuje mnohem rozsáhlejší slovníky pro kontrolu slov než aspell.

Rychlost provedené kontroly pomocí hledání neodpovídajících znaků je mnohem rychlejší, ale kontrola nalezla pouhou 1/3 špatných slov oproti GNU Aspell/libma.

Název slovníku	Velikost (mb)	DTD schéma	Aspell/libma	Neodpovídající znaky	Všechny přepínače (s)	Bez přepínačů (s)
snc-lmf	322	32,1	63,5	3,2	139	34
encz-lmf	152	21,9	16,5	2,3	79	30
encz_hh-lmf2	115	15,4	11,3	1,3	58	20
sksp- lmf	52	7,9	4,7	0,9	24	9
zkraceny-lmf	26	3,6	2,3	0,3	14	5

Tabulka 1 Rychlostní testy statistik

6.1.1 DTD Schéma

Pomocí vytvořených schémat byly odhaleny nedostatky v řadě slovníků, které nebyly při vývoji aplikace úplně očekávané. Ve slovnících ocr2lmf velmi často chybí značka Lemma, to způsobí, že vyhledávání duplicit je chybné. Výpis informací je nepoužitelný, protože je velmi těžké dohledat k jakému LexicalEntry se chyba stahuje, ve výpisu chybí jak název klíčového slova, který očekávám ve značce Lemma, tak i atribut ID ve značce LexicalEntry, které nepoužívají.

V začátcích testování byl nalezen slovník, který dokonce kombinoval různé zápisy značky feat a Feat, než byl tento problém odhalen pomocí DTD schéma způsoboval v systému velký problém. Tento slovník byl již v době psaní textu bakalářské práce opraven, a proto bohužel nemohu uvést jeho název.

6.1.2 Kontrola slov

U této kontroly velmi záleží na tom, jestli je správně použita informace o jazyku, ve kterém jsou informace wordform a text uloženy. U slovníku dictform02/czen-lmf byl nalezen problém, kdy autor chybně uvádí u klíčových slov, že byla uložena v anglickém jazyce. Poté jsou všechna klíčová slova vyhodnocena jako špatná.

V případě, že kontrolu slov provedeme pomocí GNU Aspell/libma, získáme v průměru 16% špatně napsaných klíčových slov. Pokud stejné slovníky zkontrolujeme pomocí hledání neodpovídajících znaků, získáme výsledek pouze okolo 5% špatně napsaných klíčových slov. Proto bych tuto variantu kontroly doporučil pouze v případě, že v systému nemáme nainstalovány programy pro kontrolu pomocí GNU Aspell/libma.

Všechen text, který kontroluje aspell, se automaticky zbaví všech nealfabetických znaků. Tato úprava je v převážné většině případů vhodná, protože nechceme, aby slova byla vyhodnocená jako chybná jen proto, že mají na konci otazník nebo jiný znak, ale v některých případech nám to může skrýt problém, například pokud se nerozpozná znak při převodu z jiných formátů a bude nahrazen otazníkem. Libma tuto vlastnost nemá, v jejím případě i slovo ‘pes!‘ je vyhodnoceno jako chyba, a proto jsem přistoupil k tomu, že před kontrolou slov nahradím alespoň základní sadu znaků ‘.,-?!_:;<>()"{}'‘ za mezeru.

6.1.3 Seznam použitých hodnot

Pomocí této kontroly bylo odhaleno chybné použití názvu informace, například ve slovníku dictform08/encz_technicky2_LMF je naprosto špatně použitá informace partOfSpeech, v informaci se vyskytují hodnoty jako archeologie, biologie, hedvábí, japonský a řada dalších hodnot, které spadají pod jinou informaci.

Přiložený seznam, který jsem z části doplnil, pokrývá informace domain, type a PartOfSpech u většiny slovníků ze složky dicts2lmf, nerozpoznané hodnoty jsou většinou zařazeny pod špatné informace nebo jde o špatně napsanou hodnotu. U ostatní informace jako gloss, hint a další informace, které se spíše vyskytují ve slovnících ze složky ocr2lmf, je pokrytí slabší a průměrný počet nerozpoznaných hodnot informací je 40%, z toho 10% je rozpoznáno jako hodnota, která se má použít v jiné informaci.

6.1.4 Duplicity

Při vyhledávání duplicit byl nalezen problém v nedostatečném používání informací u klíčových slov. V řadě případů nalezených duplicitních klíčových slov se nejednalo o duplicitu, ale pouze o klíčová slova s jiným významem, ale tato informace není uvedena u klíčových slov (pomocí značky feat).

6.2 Hromadné změny

Měření byla provedena na školním serveru knot1. Slovníky jsou opět brány z úložiště na serveru minerva1. Byly použity následující slovníky:

- ocr2lmf/slovnik_nespisovne_cestiny/lmf_dicts/snc-lmf.xml'
- dicts2lmf/dictform01/corrected/encz-lmf.xml
- dicts2lmf/dictform01/corrected/sksp-lmf.xml
- dicts2lmf/dictform01/corrected/ssc-lmf.xml
- dicts2lmf/dictform02/lmf_dicts/encz_hh-lmf2.xml
- dicts2lmf/dictform03/corrected/zkraceny-lmf.xml

6.2.1 SAX Generator

6.2.1.1 Nahrazení zástupek

Jak je vidět v následující tabulce, na rychlost nahrazení zástupek nemá vliv počet (ne)nahrazených zástupek, ale velikost souboru. To je způsobeno použitím SAX Generatoru, který je pomalý. Bohužel jiné řešení se nezdálo dostatečně přesné a spolehlivé.

Název slovníku	Velikost (mb)	Čas (s)	Nahrazeno	Nenahrazeno
snc-lmf	322	158	7	16
encz-lmf	152	89	2423	324
gecz-lmf	135	82	90	3650
sksp- lmf	52	33	28	2
ssc-lmf	34	22	185	3420

Tabulka 2 Nahrazování zástupků, rychlost a úspěšnost

Při nahrazení zástupků narážím na problém, že se objevují různé zkratky i v textu například r. = rok nebo výpustky, které jsou v textu, ale nejsou myšleny jako zástupky. Například v případě, kdy mám klíčové slovo rak, v informaci text je hodnota ,,Nejdéle žijící rak se narodil v r. 1972‘‘, po nahrazení zástupky ‚r.‘ vznikne nelogická věta, ,,Nejdéle žijící rak se narodil v rak 1972‘‘. Takových špatně vyhodnocených výrazů je celá řada, proto je generován soubor s informací, jaké zástupky byly nahrazeny, aby uživatel mohl sám rozhodnout, zda nahrazení proběhlo úspěšně, nebo došlo k chybě.

Možnost nahradit zástupku je jen v případě, že hodnota je uložena ve stejném jazyce jako klíčové slovo. Pro ostatní hodnoty, které jsou zapsány v jiném jazyce než klíčové slovo, nemám žádnou možnost, jak zástupku nahradit. Tím se počet nahrazených zástupek velmi snižuje.

6.2.2 XSL

V následující tabulce (Tabulka 3) je souhrn rychlostních testů provedených pro rozdělení informace writtenForm ve všech značkách feat, vytvoření informace z atributu id ve značce LexicalEntry a editace na základě pravidel.

Název slovníku	Velikost (mb)	Rozdělení informace	Vytvoření informace z atributu	Editace na základě pravidel
snc-lmf	322	41	33	45
encz-lmf	152	21	18	25
encz_hh-lmf2	115	16	13	21
sksp- lmf	52	9	9	12
zkraceny-lmf	26	7	6	11

Tabulka 3 Rychlostní test hromadných editaci pomocí XSLT

7 Závěr

Při odhalování chyb ve slovních se program projevil jako užitečný pomocník a z vygenerovaných statistik není problém vyhodnotit kvalitu převodu a případně navrhnout změny, které můžeme realizovat pomocí programu pro editace.

V této fázi ještě není program plně využitelný pro sjednocení všech slovníků, které jsou uloženy na školních serverech. Stále chybí zpracovat soubor, který by obsahoval všechny hodnoty používané ve slovnících a jejich jednotná podoba, kterou budeme využívat pro sjednocení.

V další fázi vývoje by bylo vhodné vyvinout grafické uživatelské rozhraní pro jednodušší ovládání a grafické výstupy ze statistik, které by uživateli usnadnili interpretaci výsledků. Interpretace výsledků by bylo vhodné rozšířit nejen v rámci jednoho slovníku, ale vytvořit výsledky, které by srovnávaly výsledky z více slovníků.

Dále je potřeba zapracovat na inteligentnějším nahrazení zástupek. Do budoucna by se mohla využít sémantická analýza, jestli je slovo do dané pasáže textu vhodné. Tímto přístupem bychom opět mohli snížit počet špatně nahrazených zástupek.

8 Bibliografie

1. **Bos, Bert.** XML in 10 points. *w3schools.com.* [Online] W3C, 25. 1 2011. [Citace: 2. 5 2014.] http://www.w3.org/XML/1999/XML-in-10-points.

2. Extensible Markup Language (XML) 1.0 (Fifth Edition). *w3schools.com.* [Online] W3C, 26. 11 2008. [Citace: 2. 5 2014.] http://www.w3.org/TR/xml/.

3. **Kosek, Jiří.** *XML pro každého, podrobný průvodce.* Praha : Grada Publishing, spol. s.r.o., 2000.

4. **Walsh, Norman.** A Technical Introduction to XML. [Online] O'Reilly Media, Inc., 3. 10 1998. [Citace: 20. 4 2014.] http://www.xml.com/pub/a/98/10/guide0.html.

5. DOM. *w3school.org.* [Online] [Citace: 4. 5 2014.] http://www.w3.org/DOM/.

6. XML DTD. *w3schools.com.* [Online] W3C. [Citace: 3. 5 2014.] http://www.w3schools.com/xml/xml_dtd.asp.

7. **Walmsley, Priscilla.** *Definitive XML Schema, 2nd Edition.* New Jersey : Prentice-Hall, 2012 září. ISBN: 0132886723.

8. XML Schema. *w3school.org.* [Online] [Citace: 2. 5 2014.] http://www.w3.org/TR/NOTE-xml-schema-req.

9. XSLT. *w3school.org.* [Online] 16. 11 1999. [Citace: 4. 5 2014.] http://www.w3.org/TR/xslt.

10. XSL. *w3school.org.* [Online] 5. 12 2006. [Citace: 4. 5 2014.] http://www.w3.org/TR/xsl/.

11. XPath. *w3school.org.* [Online] 16. 11 1999. [Citace: 4. 5 2014.] http://www.w3.org/TR/xpath/.

12. *lexicalmarkupframework.org.* [Online] [Citace: 5. 5 2014.] http://www.lexicalmarkupframework.org/.

13. **Atkinson, Kevin.** Aspell.net. [Online] 2004. [Citace: 20. 4 2014.] http://aspell.net/.

14. KNOT Wiki. [Online] 16. 10 2010. [Citace: 20. 4 2014.] http://knot.fit.vutbr.cz/wiki/index.php/Morfologick%C3%BD_slovn%C3%ADk_a_morfologick%C3%BD_analyz%C3%A1tor_pro_%C4%8De%C5%A1tinu.

15. LXML. [Online] 18. 4 2014. [Citace: 20. 4 2014.] http://lxml.de/.

16. **Kay, Michael.** saxon XMLGenerator. [Online] 18. 9 2001. [Citace: 20. 4 2014.] http://saxon.sourceforge.net/dtdgen.html.

17. **Kay, Michael H.** Saxon. [Online] 21. 6 2013. [Citace: 20. 4 2014.] http://saxon.sourceforge.net/.

18. The Python Profilers. *Python.org.* [Online] [Citace: 27. 4 2014.] https://docs.python.org/2/library/profile.html#module-cProfile.

19. **Lekhonkhobe, Tshepang.** Argparse. [Online] 18. 3 2014. [Citace: 20. 4 2014.] https://docs.python.org/dev/library/argparse.html.

20. Subprocess. [Online] [Citace: 22. 4 2014.] https://docs.python.org/2/library/subprocess.html.

21. Xml. *Wikipedia.* [Online] 12. 3 2014. [Citace: 20. 4 2014.]
http://en.wikipedia.org/wiki/Xml#Escaping.
22. Dictionary. *wikipedia.org.* [Online] [Citace: 4. 5 2014.] http://en.wikipedia.org/wiki/Dictionary.
23. *Wiktionary.org.* [Online] [Citace: 5. 5 2014.] http://www.wiktionary.org/.
24. Lexical Markup Framework. *wikipedia.org.* [Online] [Citace: 5. 5 2014.]
http://en.wikipedia.org/wiki/Lexical_Markup_Framework.
25. ISO 12620. *wikipedia.org.* [Online] [Citace: 6. 5 2014.]
http://en.wikipedia.org/wiki/Data_Category_Registry.
26. Data Category Registry. *ISOcat.org.* [Online] [Citace: 6. 5 2014.] http://www.isocat.org/.

Seznam příloh

Přiložené CD obsahuje:

- Zdrojové kódy
- Písemnou zprávu ve formátu PDF
- Skript pro opravu chyb špatně nahrazených syntaktických značek