VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

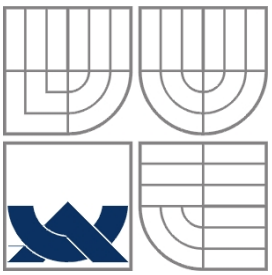
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

KNIHOVNA PRO ZPRACOVÁNÍ DOKUMENTŮ OPENXML

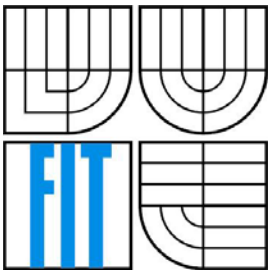
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. PETER BELICA
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

KNIHOVNA PRO ZPRACOVÁNÍ DOKUMENTŮ OPENXML

OPENXML DOCUMENT PROCESSING LIBRARY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETER BELICA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2009

Abstrakt

Práce popisuje typ dokumentů Office Open XML, pro které se rovněž používá označení OpenXML. Jsou vysvětleny základní vlastnosti těchto dokumentů a jejich struktura. Dále jsou přiblíženy i další technologie DOM a XHTML nezbytné pro realizaci. Je navržena knihovna, konzument, která čte tyto dokumenty, vytvoří pro ně DOM strukturu a následně ji vyexportuje do souboru XHTML.

Abstract

This thesis describes the type of documents named Office Open XML, which are commonly known as OpenXML. Basic properties of these documents as well as their structure are explained. There is also an introduction to other technologies such as DOM and XHTML beacause of their inevitability in realization. Program library is designed to read these documents, build a DOM structure for them and export it as a XHTML file.

Klíčová slova

Office Open XML, OpenXML, Open XML, ooxml, DOCX, DOM, XHTML, transformace

Keywords

Office Open XML, OpenXML, Open XML, ooxml, DOCX, DOM, XHTML, transformation

Citace

Belica Peter: Knihovna pro zpracování dokumentů OpenXML, diplomová práce, Brno, FIT VUT v Brně, 2009

Knižnica pre spracovanie dokumentov OpenXML

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Radka Burgeta, Ph.D..
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Peter Belica
26.5.2009

Poďakovanie

Ďakujem vedúcemu diplomovej práce Ing. Radkovi Burgetovi, Ph.D. za vedenie tohto projektu a cenné informácie.

Obsah

Obsah 1
1 Úvod 3
1.1 Popis kľúčových technológií 3
1.2 Analýza formátu Office Open XML a návrh transformácie 4
1.3 Implementácia knižnice 4
1.4 Slovník pojmov 4
1.5 Prílohy 4
2 Formát Office Open XML 5
2.1 Formát súboru 5
2.2 Adresárová štruktúra 6
3 Document Object Model 8
3.1 DOM Level 0 8
3.2 DOM Level 1 8
3.3 DOM Level 2 8
3.4 DOM Level 3 9
3.5 Štruktúra DOM 9
3.5.1 Node 10
3.5.2 Document 11
3.5.3 Element 11
3.6 DOM pre jazyk Java 11
3.7 Využitie DOM v knižnici transformácie dokumentu 12
4 XHTML 13
4.1 Základná štruktúra dokumentov 13
4.1.1 Odseky a textové bloky 14
4.1.2 Zoznamy 14
4.1.3 Odkazy v rámci dokumentu a odkazy na internetové stránky 14
4.2 Štýl dokumentu 15
4.3 Zhrnutie 15
5 Transformácia z Office Open XML do XHTML 16
5.1 Rozdielne možnosti transformácie 16
5.1.1 Priama transformácia 16
5.1.2 Transformácia s použitím XSLT 17
5.1.3 Porovnanie oboch možností transformácie 17
5.2 Vlastnosti pripravovanej knižnice 18

5.3 Návrh transformácie 18
5.4 Analýza Office Open XML dokumentu a tvorba DOM 19
5.4.1 Súbory textového dokumentu 20
5.4.2 Štruktúra obsahového dokumentu 20
5.4.3 Štýly a formátovanie 21
5.4.4 Odseky v DOCX dokumentoch 22
5.4.5 Zoznamy v DOCX dokumentoch 26
5.5 Exportovanie DOM do XHTML 27
6 Implementácia knižnice 28
6.1 Využitie XSLT 28
6.2 Flat OPC 29
6.3 Využitie externej knižnice s podporou DOCX 29
6.3.1 Knižnica POI-OpenXML4J 30
6.3.2 Knižnica docx4j 30
6.3.3 Priamy prístup k obsahu dokumentov 31
6.3.4 Porovnanie knižníc 31
6.4 Využité technológie pri transformácii 32
6.4.1 XPath 32
6.4.2 Java Reflection API 32
6.4.3 XSLT 33
6.4.4 Spracovávanie XML dokumentov v Jave 34
6.5 Postup transformácie dokumentu 36
6.6 Implementácie transformácie nadpisov 37
6.7 Implementácia transformácie zoznamov 38
6.7.1 Transformácia v časti XSLT 39
6.7.2 Transformácia v DOM štruktúre 40
6.8 Implementácia transformácie fontov 41
6.9 Transformácia odkazov 42
6.10 Transformácia vložených obrázkov 43
7 Záver 45
7.1 Využitie externých zdrojov 45
7.2 Námety na ďalší vývoj 46
Slovník pojmov 47
Literatúra 48
Zoznam obrázkov 50
Zoznam tabuliek 50
Zoznam príloh 51

1 Úvod

Téma diplomovej práce sa dotýka aktuálnej otázky kancelárskych balíkov a možnosti transformovania dokumentov vytvorených v proprietárnych prostrediach do otvoreného a často používaného formátu XHTML. Transformácia je konkrétne z formátu Office Open XML, zavedenom spoločnosťou Microsoft v rade kancelárskych produktov balíka Microsoft Office 2007, do prostredia internetových stránok a formátu XHTML. Office Open XML je otvorený štandard, ktorý je od novembra roka 2008 vedený ako norma ISO/IEC 29500:2008. XHTML je značkovací jazyk na tvorbu webových stránok, ktoré sú založené na formáte XML a rozširujú HTML postavené na SGML. Štruktúra oboch typov dokumentov, Office Open XML a aj XHTML, sa dá popísať pomocou jedného spoločného DOM, čo je hierarchický model dokumentov.

Prostredníctvom normy Office Open XML sú ukladané textové dokumenty, tabuľkové dokumenty a prezentácie. Pre transformáciu do výstupného XHTML má význam uvažovať iba prácu so vstupným textovým dokumentom, ktorý má v Office Open XML označenie DOCX a je svojou povahou zhodný s XHTML.

Cieľom práce je navrhnúť a implementovať knižnicu v programovacom jazyku Java, ktorá zo vstupného dokumentu DOCX vytvorí príslušnú reprezentáciu v XHTML. Tento transformovaný výstup uloží do príslušného súboru, alebo sprístupní DOM štruktúru, v prípade, že bude knižnica použitá v inej Java aplikácii.

1.1 Popis kľúčových technológií

V prvých kapitolách mojej práce som sa postupne venoval vysvetleniu jednotlivých kľúčových technológií, na ktorých je postavený návrh pripravovanej aplikácie. Keďže sa v prípade každej z použitých technológií jedná o rozsiahle systémy, nie sú tieto detailne popísané. Prehľad, ktorý je o každej popísaný je však dostatočný, aby si čitateľ vytvoril predstavu o základných princípoch a vlastnostiach tej ktorej technológie.

V kapitole Office Open XML sa venujem základnému vysvetleniu princípu ukladania dokumentov v tomto formáte. Venujem sa spôsobu uloženia jednotlivých častí v súbore, takisto aj jednotlivým typom dokumentov, ktoré sa dajú v tomto formáte uložiť.

Ďalšiu kapitolu venujem popisu rozhrania na reprezentáciu modelov dokumentov s názvom Document Object Model (DOM). Venujem sa nielen popisu jednotlivých úrovní DOM, ale aj popisu základných operácií nad takto popísanou štruktúrou, ktorá umožňuje vytváranie ale aj editovanie už existujúcich dokumentov.

V poslednej kapitole je rozobraný značkovací jazyk XHTML. Vzhľadom na to, že v tejto práci nie je priestor pre popis jednotlivých značiek, prípadne pre podrobné vysvetlenie princípov tvorby a prezentácie dokumentov v tomto formáte, je táto kapitola iba informatívneho charakteru.

1.2 Analýza formátu Office Open XML a návrh transformácie

V ďalšej kapitole je popísaná architektúra textových dokumentov DOCX formátu Office Open XML. Popísané sú aj jednotlivé kľúčové časti dokumentov s návrhom ich transformácie do XHTML.

V úvodnej časti kapitoly som popísal jednotlivé možnosti, ako aplikačne riešiť celý proces transformácie dokumentov. Tieto metódy sú zhrnuté v prehľadnej tabuľke spolu so stručným popisom výhod a nevýhod jednotlivých metód. V kapitole je ďalej uvedená analýza textových dokumentov DOCX. Navrhnutý je základný algoritmus prepisu dát do formátu XHTML.

1.3 Implementácia knižnice

Posledná kapitola práce je venovaná výberu metódy použitej na transformáciu dokumentov, z možností, ktoré boli popísané v predchádzajúcej časti. Pri implementácií som využil aj externú knižnicu docx4j, ktorá ponúka základnú podporu na prácu s DOCX dokumentmi. Táto knižnica je v kapitole popísaná spolu s ďalšími dostupnými knižnicami s podobnou funkčnosťou a ich vzájomným porovnaním.

Postupne sú vysvetlené jednotlivé časti procesu transformácie. Pri niektorých sa vyskytli problémy, ktoré si vyžadovali využitie ďalších technológií a častí jazyka Java. Tieto problémy, použité technológie a riešenia sú popísané na záver kapitoly.

1.4 Slovník pojmov

V texte diplomovej práce používam pojmy, ktoré nie sú kľúčové pre pochopenie technológie transformácie ale pochopenie ich významu uľahčuje orientáciu v obsahu práce. Tieto pojmy sú krátko vysvetlené v samostatnej kapitole na záver dokumentu.

1.5 Prílohy

V prílohách je ukážka testovacieho dokumentu vytvoreného v Microsoft Word 2007, ktorý bol použitý pri testovaní výslednej implementácie knižnice na operačných systémoch Microsoft Windows a Linux a príslušný vygenerovaný dokument XHTML.

2 Formát Office Open XML

OpenXML je jedným z možných označení súborového typu Office Open XML, pre ktorý sa používajú ďalšie názvy ako OOXML alebo Open XML. Tento formát súborov z produkcie spoločnosti Microsoft bol uvedený spolu s príchodom kancelárskeho balíka Microsoft Office 2007. Po zverejnení bol tento formát špecifikovaný ako norma ECMA 376 v prvej verzii v decembri roka 2006. Od novembra roka 2008 je tento formát súborov vedený ako ISO norma ISO/IEC 29500:2008, ktorá sa skladá zo štyroch častí s celkovým rozsahom viac ako 7000 strán.

Office Open XML je všeobecný súborový formát, ktorý sa využíva pre ukladanie rôznych typov kancelárskych súborov. Pre ukladanie textových dokumentov, ktoré sa doteraz v prostredí Microsoft Word ukladali v binárnej podobe ako súbory s príponou .doc, sa v tomto formáte ukladajú s príponou .docx. Podobne je to aj pre prezentácie z programu Microsoft PowerPoint, kde sa pre binárnu podobu súboru používa .ppt a pre nový typ .pptx. Taktiež aj pre tabuľkové súbory z programu Microsoft Excel, v binárnej podobe s príponou .xls, sa používa .xlsx. Pre každý typ súboru sa pridá na koniec označenia typu súboru znak x, ktorý označuje, že súbor je vo formáte Office Open XML a je príslušného typu kancelárskeho dokumentu.

So súbormi pracujú 2 typy programov. Jedným z nich sú konzumenti, ktorí vedia súbory čítať a zobrazovať, ale nevedia ich meniť a ukladať. Na druhej strane sú producenti, ktorí vytvárajú a editujú súbory. Samozrejme existujú programy, ktoré majú vlastnosti oboch typov programov. Ako príklad môže slúžiť program Microsoft Word 2007, ktorý je producentom a konzumentom súčasne.

2.1 Formát súboru

Súbor typu Office Open XML je fyzicky uložený, podľa konvencie Open Packaging Conventions (OPC), ako ZIP archív viacerých súborov v špecifikovanej adresárovej štruktúre. Takto uložený súbor sa nazýva balík. Architektúra balíka umožňuje okrem primárneho uloženia do podoby ZIP archívu uložiť obsah aj do databázy, pričom by takéto skladovanie umožnilo lepšie vyhľadávanie aj v databázových dotazoch. V tomto jednom archíve sú uložené XML súbory, ktoré tvoria obsah a vlastnosti celého súboru a multimediálne súbory ako aj obrázky a videá, ktoré sú použité v obsahu dokumentu. Obsah dokumentu je podľa typu tvorený jedným alebo viacerými XML súbormi. Pri dokumentoch programu Microsoft Word je obsah tvorený jedným XML súborom, ale v prípade rozdelenia dokumentu na logické časti, sekcie, sú tieto uložené v samostatnom XML súbore. Hlavičky, päty ale aj komentáre či poznámky sú uložené každé v samostatnom súbore. V prípade dokumentu programu Microsoft PowerPoint je každá strana dokumentu reprezentovaná jedným samostatným súborom. V dokumentoch programu Microsoft Excel tvorí každý jeden list samostatný

súbor. V takto rozloženom obsahu sú v rámci jedného balíka určené relácie medzi jednotlivými časťami.

Pre balíky generované programom Microsoft Word 2007 je potrebné vysvetliť jeho 3 hlavné časti. Prvky *časti dokumentu* je označenie pre sadu samostatných súborov tvoriacich obsah dokumentu. Prvky *typy obsahu* určujú v akom formáte sa ukladajú súbory vo vnútri balíku. Tieto informácie sú pre celkový dokument doplňujúce a využívajú ich aplikácie pracujúce s dokumentom, ako konzument či producent, na správne zaobchádzanie so súbormi podľa ich typu. Poslednú časť tvoria *prvky relácií*, ktoré definujú ako sa spájajú zdrojové a cieľové časti dokumentu do jedného výsledného obsahu. Tieto relácie sú v rámci balíku uložené v adresári */_rels/.rels*.

2.2 Adresárová štruktúra

Názvy adresárov a súborov vo vnútri balíka nie sú pevne definované normou Open Packaging Conventions, takže vhodné premenovanie odporúčanej štruktúry je možné a pokiaľ sa korektne zrealizuje, tak programy na prácu s týmito typmi súborov dokážu takýto zmenený súbor otvoriť a správne používať bez akýchkoľvek problémov. Popis jednotlivých súborov, ktoré sú dôležité pre fungovanie Office Open XML dokumentu, bude vysvetlený na základnej štruktúre, v ktorej sú tieto súbory uložené v prípade použitia programu Microsoft Word 2007.

- **[Content_Types].xml** – Popisuje typ súboru pre každý súbor v balíku.
- **Adresár _rels** – V adresári sa ukladajú časti popisujúce relácie.
- **Súbor .rels** – Popisuje relácie, ktoré tvoria základ štruktúry dokumentu. Zmeny štruktúry balíka je potrebné definovať v tomto súbore.
- **Adresár používateľských súborov** – Adresár s používateľskými XML súbormi.
- **Súbor item1.xml** – Obsahuje niektoré dáta zobrazované v dokumente. Toto je príklad používateľského XML súboru, ktorý je uložený v adresári popísanom v odrážke vyššie.
- **Adresár docProps** – Obsahuje súbory, ktoré špecifikujú vlastnosti aplikácie a dokumentu.
- **Súbor App.xml** – Obsahuje nastavenia aplikácie. Tieto nastavenia sú špecifické pre každú aplikáciu.
- **Súbor Core.xml** – Obsahuje nastavania dokumentu spoločné pre akúkoľvek aplikáciu pracujúcu s týmto balíkom.

V súbore *[Content_Types].xml* sú definované typy formátov častí dokumentu. Názov týchto formátov sa vytvára nasledovným spôsobom. Začína sa označením typu, zväčša *application* a pokračuje názvom poskytovateľa, ktorý je označený skratkou *vnd*. Ak názov formátu končí *+xml*, jedná sa o XML súbor. [10]

Nasledovné ukážky sú zo súboru *[Content_Types].xml* v balíku, ktorý pochádza z tejto práce. Nejedná sa o kompletný výpis zo súboru.

```
<Override PartName="/word/document.xml" ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.document.main+xml" />
```

Súbor uložený v balíku na ceste */word/document.xml*, tvorí hlavný obsah dokumentu. Tento súbor je všeobecný, netýka sa iba aplikácie Microsoft Word, preto je ako *vnd* označený *openxmlformats-officedocument*. Označenie *+xml* na konci značí, že je to XML súbor.

```
<Override PartName="/word/numbering.xml" ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.numbering+xml" />
```

V tomto súbore sú špecifikované číslovania zoznamov, použitých v dokumente. V sekcii dokumentu, kde sa používa číslovaný zoznam nie je uvedené konkrétne číslo odrážky, ale používa sa relácia práve na tento súbor, odkiaľ sa získa hodnota čísla.

Konkrétna štruktúra dokumentu, teda členenie na časti, stránky, odseky, bude uvedená v časti návrhu transformácie, kde bude rozoberanie štruktúry dokumentu konfrontované s vytváraním stromovej štruktúry DOM.

3 Document Object Model

DOM, skratka z anglického Document Object Model, je objektový model dokumentu. Je to rozhranie, ktoré je platformovo a jazykovo nezávislé a umožňuje prostredníctvom jeho využitia dynamický prístup k dokumentu s možnosťou upravovať jeho obsah, štruktúru a štýl. World Wide Web Consortium (W3C) špecifikuje DOM vo viacerých úrovniach. V nasledujúcich podkapitolách sú predstavené jednotlivé úrovne. [6].

3.1 DOM Level 0

Úroveň 0 predstavuje v hierarchii vrstiev DOM tie rozhrania, ktoré boli ako prvé použité v prehliadačoch pri potrebe manipulovať s obsahom HTML súboru dynamicky. Na tejto vrstve nebola pevne definovaná jedna spoločná množina metód, ale pre každý prehliadač existovali samostatné, takže sa jednotlivé metódy líšili. Tieto rozhrania sa udržiavajú v hierarchii kvôli spätnej kompatibilite.

3.2 DOM Level 1

Na úrovni 1 je DOM rozdelený do dvoch častí: DOM Core a DOM HTML. Časť DOM Core umožňuje nízko úrovňový prístup ku štruktúre dokumentov prostredníctvom jednoduchého rozhrania. Špecifikované je aj rozšírené rozhranie na prístup k XML dokumentom. Časť DOM HTML poskytuje prídavné vysoko úrovňové rozhranie, ktoré spolu s rozhraním DOM Core vytvárajú vhodnejšie prostriedky na prácu s HTML dokumentmi. K dispozícii sú rozhrania Document, Node, Attr, Element a Text, ktoré sú určené na prácu s objektovými modelmi HTML a XML dokumentov.

3.3 DOM Level 2

Na úrovni 2 je v DOM špecifikovaných 6 častí: DOM2 Core, DOM2 Views, DOM2 Events, DOM2 Styles, DOM2 Traversal and Range a DOM2 HTML. Pre potreby pripravovanej knižnice je zaujímavá časť DOM2 Core, ktorá špecifikuje napríklad metódu `getElementById`. Takisto časť pracujúca s kaskádovými štýlmi.

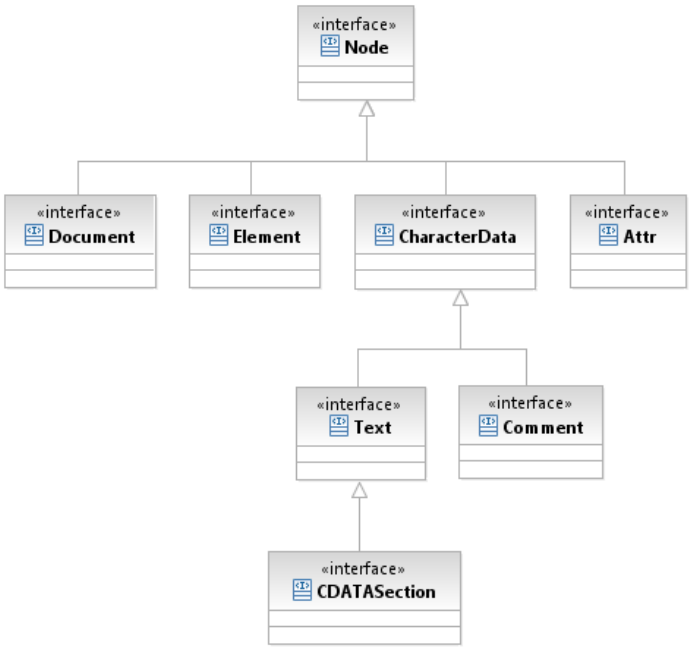
3.4 DOM Level 3

Na úrovni 3 špecifikácia prináša posledné doplnenie a to v podobe piatich častí: DOM3 Core, Load and Save, Validation, Events a XPath. Z tejto sady rozšírení nebude pre prácu knižnice potrebná žiadna časť.

3.5 Štruktúra DOM

DOM je reprezentovaný ako stromová hierarchia objektov rozhrania Node, ktoré môže implementovať ďalšie viac špecializované rozhrania. Niektoré objekty môžu mať podľa špecifikácie potomkov, zvyšné sú koncové objekty stromu. Základnú štruktúru, ktorá je v plnej špecifikácii väčšia, tvoria nasledovné rozhrania: Node, Document, Element, CharacterData, Attr. Rozhranie CharacterData je ďalej dedené ako Text a Comment. CDATASection je listové rozhranie, ktorého implementácia nemôže obsahovať ďalšie vetvy štruktúry. Okrem týchto typov objektov je v štruktúre zakomponovaných viac podporných rozhraní a typov:

- **DocumentType** - je rozhranie na určenie typu dokumentu. V tomto objekte sa vlastnosti zadávajú do zoznamu entít.
- **DocumentFragment** - je koreň stromovej štruktúry, podobne ako objekt typu Document, ktorý je ale určený na zrealizovanie operácii a následné vloženie do štruktúry objektu Document. DocumentFragment nie je možné použiť bez existencie rodičovského objektu typu Document.
- **NodeList** - je rozhranie pre objekt obsahujúci zoznam objektov typu Node. Toto rozhranie poskytuje možnosť prístupu k množine objektov a prípadne ku každému objektu pomocou číselného identifikátora v zozname.
- **NamedNodeMap** - je rozhranie pre objekt so zoznamom objektov typu Node, podobne ako NodeList, pričom ale v tomto zozname sa dá k jednotlivým objektom pristupovať na základe mena objektu.
- **DOMString** - je dátový typ na ukladanie textu v kódovaní UTF-16. Napriek tomu, že v norme je pri textových vstupoch použitý tento typ, v konkrétnej implementácii sa toto môže líšiť. Napríklad v jazyku Java sa používa dátový typ `java.lang.String`, pretože spĺňa požiadavky na kódovanie.



Obrázok 1 – Základná štruktúra DOM Level 2 Core

Z diagramu štruktúry je vidieť, že každý prvok stromovej štruktúry je zdedený z rozhrania Node. Rozhranie Document, ktoré je tiež zdedené z Node, tvorí koreň stromovej štruktúry dokumentu a z každého objektu typu Node, ktorý je v tejto stromovej štruktúre potom existuje referencia na tento koreň.

3.5.1 Node

Node je základným rozhraním, od ktorého dedia ďalšie rozhrania. Pre správu obsahu stromu obsahuje, okrem iných, tieto atribúty a operácie, ktoré sú zaujímavé z pohľadu priamej práce so štruktúrou dokumentu:

- boolean **hasChildNodes()** – operácia na zistenie potomkov objektu
- Node **parentNode** – referencia na rodičovský objekt
- NodeList **childNodes** – referencia na objekt zoznamu potomkov
- Node **firstChild** – referencia na prvého potomka
- Node **lastChild** – referencia na posledného potomka
- Node **previousSibling** – referencia na ľavého suseda (majú spoločný rodičovský objekt)
- Node **nextSibling** – referencia na pravého suseda (majú spoločný rodičovský objekt)
- Document **ownerDocument** – referencia na koreň stromovej štruktúry

Na úpravu DOM štruktúry slúžia napríklad nasledovné operácie:

- Node **insertBefore**(in **Node** newChild, in **Node** refChild) – vloží nový Node newChild pred Node refChild medzi svojimi potomkami.
- Node **replaceChild**(in **Node** newChild, in **Node** oldChild) – vymení Node newChild za Node oldChild medzi svojimi potomkami.
- Node **removeChild**(in **Node** oldChild) – odstráni Node oldChild zo svojich potomkov.
- Node **appendChild**(in **Node** newChild) – vloží nový Node newChild za posledný zo svojich potomkov.

3.5.2 Document

Document je rozhranie zdedené z Node, ktoré reprezentuje koreň celej štruktúry dokumentu. Rozhranie Node je doplnené o nasledujúce atribúty a operácie, pomocou ktorých je možné vytvárať nové objekty do štruktúry DOM, alebo definovať vlastnosti dokumentu:

- Element **createElement**(in **DOMString** tagName)
- Text **createTextNode**(in **DOMString** data)
- DocumentType **doctype** – referencia na objekt označujúci typ dokumentu
- Element **documentElement** – referencia na objekt typu Element, ktorý tvorí koreň štruktúry

3.5.3 Element

Element je rozhranie zdedené z Node a ako také reprezentuje práve element z HTML alebo XML dokumentu. Toto rozhranie umožňuje prístup ku svojim atribútom, nie len prostredníctvom rozhrania Node, kde sa dal získať iba objekt rozhrania NamedNodeMap so zoznamom atribútov, ale obsahuje aj operácie na pridávanie, mazanie a získavanie atribútov. Okrem toho obsahuje atribút *tagName*, ktorý určuje názov elementu. V prípade DOM2 HTML je tento názov tvorený názvom značiek z jazyka HTML ako sú DIV, P, IMG a ďalšie.
Zoznam niektorých atribútov a operácii rozhrania Element:

- DOMString **tagName** – názov objektu typu Element
- DOMString **getAttribute**(in **DOMString** name) – získa hodnotu atribútu s názvom name
- void **setAttribute**(in **DOMString** name, in **DOMString** value) – nastaví hodnotu value atribútu name.
- void **removeAttribute**(in **DOMString** name) – odstráni atribút s názvom name

3.6 DOM pre jazyk Java

W3C špecifikuje iba rozhrania DOM, ale neposkytuje žiadnu implementáciu. Je však pravidlom, že ak existuje implementácia DOM Level 2, tak táto obsahuje implementáciu DOM Level 1. V prostredí

jazyka Java existuje viacero implementácii rozhrania DOM. Staršia verzia jazyka J2SE 1.4 obsahuje implementáciu DOM Level 2, pri novšej J2SE 5.0 je implementované rozhranie DOM Level 3. Okrem týchto existujú aj ďalšie dostupné implementácie v postačujúcom rozsahu pre knižnicu transformácie. Ako príklad je možné uviesť knižnice dom4j[1] alebo jDom[2].

3.7 Využitie DOM v knižnici transformácie dokumentu

Na popis spracovávaného dokumentu v knižnici transformácie bude použitý DOM2 Core, ktorým je možné vytvoriť zodpovedajúci model súboru. Štruktúra by mala zodpovedať pôvodnému dokumentu na toľko, aby ale bolo možné z tejto istej štruktúry exportovať XHTML súbor. Problematickým miestom transformácie z Open Office XML dokumentu je uloženie prídavných multimediálnych súborov, keďže pre tieto neexistuje v DOM implementácia, ktorá by v sebe ukladala binárny obsah súborov. Ukladanie takýchto dát bude v DOM štruktúre teda uložené rovnakým spôsobom ako v HTML a teda bude prezentované príslušným elementom IMG pre obrázok alebo OBJECT s atribútom odkazu na príslušný súbor prostredníctvom URL.

[1] Dostupná na adrese http://www.dom4j.org

[2] Dostupná na adrese http://www.jdom.org

4 XHTML

XHTML je označenie značkovacieho jazyka, ktorý je určený na popis dokumentov, primárne určených na prezentovanie v podobe internetových stránok. Tento jazyk dopĺňa a upravuje základy jazykov HTML ale vychádza zo základov XML a nie SGML, ako je to v prípade jazykov HTML. XHTML je množina značiek, ktoré sa používajú na vytvorenie stromovej štruktúry dokumentu a obsah týchto značiek, podľa fungovania XML, tvorí obsah dokumentu. Značky musia byť navzájom správne zaobaľované a nesmú sa navzájom prekrižovať. Možné je iba zanáranie značiek, pričom v jednej môže byť zanorené ľubovoľné množstvo ďalších značiek.

Kontrolovaná je štruktúra značiek nie len podľa syntaktických ale aj sémantických pravidiel. Ako príklad je možné uviesť, že vytvorená tabuľka značkou `<table>` môže mať priamo v sebe zanorené iba značky riadku tabuľky `<tr>`, ale nemôže obsahovať priamo zanorený odsek so značkou `<p>`.

XHTML je samostatný súbor, v ktorom je obsah písaný v stromovej štruktúre pomocou značiek a ich zanárania. Štýl dokumentu, teda jeho výzor, je popísaný kaskádovými štýlmi, pričom tieto môžu ale nemusia byť priamo súčasťou súboru dokumentu. V prípade, že sú kaskádové štýly definované v samostatnom súbore, je v súbore dokumentu uvedená referencia pomocou špeciálnej značky na tento súbor a teda štýly budú na dokument aplikované. Podobný prístup je aj pri vkladaní multimediálnych súborov do dokumentu. Tieto nie sú nikdy súčasťou súboru dokumentu, ale sú vždy referencované ako externé súbory. Ako príklad je možné uviesť značku pre vloženie obrázka `<img src="<<referencia_súboru_obrázka>>" />`.

4.1 Základná štruktúra dokumentov

V jazyku XHTML je možné pomocou značiek vytvoriť štruktúrovaný textový dokument. Toto ale nie je jediný účel jazyka. Prvotným účelom bolo vytváranie jednoduchých dokumentov prezentovaných na Internete. Tomu bola prispôsobená aj základná množina značiek, pomocou ktorých sa dokumenty vytvárali. Postupom času a vývinu počítačových sietí dostal pôvodný jazyk HTML viac značiek a prísnu kontrolu zanárania, ktorá prišla zdedením XHTML z XML.

V dnešnej dobe, je XHTML často používaný s ďalšími technológiami ako JavaScript a DHTML pri tvorbe aplikácií, ktorých používateľské rozhranie je tvorené vo webovom prehliadači. Pri tomto využití jazyka ako popisovača dokumentov sú vidieť široké možnosti, ktoré prináša.

Základné využitie značiek XHTML jazyka, ktoré bude zaujímavé pre transformáciu čisto textových dokumentov z DOCX, je výrazne menšie ako v prípade tvorby moderných webových prezentácií. Niektoré základné štruktúry sú informatívne popísané v nasledujúcich podkapitolách.

4.1.1 Odseky a textové bloky

Všetky celistvé textové bloky písané v dokumentoch DOCX sú tvorené odsekmi. Tak ako v klasických tlačených publikáciách je teda aj v prípade elektronických dokumentov odsek základnou najmenšou celistvou textovou jednotkou. Odseky sú v XHTML tvorené značkou `<p>`. Pre túto značku existuje široká množina parametrov, ktoré nastavujú vlastnosti tohto elementu.

Jednotlivé odseky sa však môžu z pohľadu značkovanie deliť na menšie časti. Dôvod tohto delenia môže byť z vizuálneho pohľadu napríklad podčiarknutie niektorých slov, ktoré sú súčasťou jedného odseku. Pre menšie časti, ktoré majú spoločné nejaké vizuálne prvky, sa v XHTML používa značka `<span>`. Táto slúži pre všeobecné nastavanie vlastností textových reťazcov. Niektoré špecializované formátovania majú svoje vlastné značky ako `<b>` pre tučné písmo, `<u>` pre podčiarknutie, alebo `<i>` pre kurzívu.

4.1.2 Zoznamy

Zoznamy sú štandardnou súčasťou textových dokumentov. V základnom pohľade rozlišujeme zoznamy zoraďované a nezoraďované. V zoraďovaných sú jednotlivé prvky číslované, prípadne označené inou množinou znakov, ktoré majú definovanú postupnosť. Možnosti značenia sú teda číslovanie numerickými znakmi, rímskymi číslicami a abecedou. Pre nezoraďované zoznamy sa na označenie prvkov používajú vizuálne objekty ako štvorce, kruhy a pod.

Zo syntaktického hľadiska sú zoznamy realizované ako značky s označením OL pre zoraďované a UL pre nezoraďované zoznamy. V každom zo zoznamov sú ďalej povolené len značky `<li>`, ktoré predstavujú jednotlivé prvky zoznamov. Pre tieto je možné nastaviť štýl, akým majú byť označené, teda znaky, číslice alebo naopak vizuálne objekty ako kruhy, štvorce.

Dôležité je pri zoznamoch spomenúť možnosť zanárania `<ol>` prípade `<ul>` značiek navzájom. Zoznam je teda tvorený stromovou štruktúrou, kde uzly reprezentujú nové zoznamy a listy jednotlivé prvky zoznamov.

4.1.3 Odkazy v rámci dokumentu a odkazy na internetové stránky

Jeden zo základných stavebných kameňov internetu je prepájanie. Dôležité aj pri tvorbe internetových stránok je ich vzájomné prepájanie. Toto sa dá v jazyku XHTML docieliť prostredníctvom značky A. Pri tejto uvediem aj povinný parameter `href`, ktorého hodnota odkazuje na stránku, alebo prostredníctvom vhodné protokolu na akýkoľvek adresovateľný objekt na internete, prípadne na nejakú záložku v rámci dokumentu.

Záložky v dokumente sa, rovnako ako odkazy, tvoria prostredníctvom značky `<a>`, ale parameter `href` je vymenený za parameter `name`. Jeho hodnota tvorí identifikátor, prostredníctvom ktorého je možné presunúť sa na túto záložku.

4.2 Štýl dokumentu

Pri tvorbe kancelárskych dokumentov je okrem obsahu dôležitá aj vizuálna stránka dokumentu. V prípade dokumentov písaných v jazyku XHTML je prístup ku vizualizácii možný prostredníctvom Cascading Style Sheet (CSS), čo sú kaskádové štýly určené na použitie práve v jazykoch HTML a XHTML. Definovanie štýlov je možné priamo v súbore dokumentu v špeciálnom elemente hlavičky dokumentu `<style>`. Druhá možnosť je definovať štýly v samostatných súboroch, v jednom alebo viacerých. Ich vlastnosti sa môžu navzájom prepisovať a platné zostávajú naposledy definované.

Štýly sa dajú definovať na niekoľkých úrovniach, pričom aplikované štýly sú kombináciou viacerých definícii podľa ich hierarchie. Pri prepisovaní platí logické pravidlo zanárania, teda špecializovanejšie definovaný štýl prepíše vlastnosti definované všeobecnejšie.

Štýly sa na najvyššej úrovni dajú definovať pre jednotlivé značky. Takto definovaný štýl sa uplatní pre každý výskyt značky v dokumente, pokiaľ pre ten ktorý výskyt nebol tento štýl prepísaný iným. Takéto definovanie štýlov sa musí pripraviť v elemente `<style>`, prípadne v externom súbore.

Na ďalšej úrovni je možné definovať triedu štýlov. Táto je jednoznačne označená identifikátorom, prostredníctvom ktorého sa na túto triedu môže odkazovať takmer každá značka v dokumente. Rovnako ako predchádzajúca úroveň musí byť trieda štýlov definovaná v značke `<style>` alebo v externom súbore.

Posledným spôsobom ako definovať štýl je jeho definovanie priamo v značke, ktorá zaobaľuje obsah, ktorý chceme štýlom prispôsobiť. Takto definované vlastnosti platia len pre objekt, ktorý je zaobalený a prepisujú predtým definované vlastnosti, ktoré sa na túto značku vzťahovali.

4.3 Zhrnutie

V mojej práci nie je priestor pre presný popis jazyka na definovanie štýlov v XHTML dokumente a nie je to ani jej úlohou. Jedná sa však o komplexný jazyk, ktorý od svojej prvej verzie prešiel mnohými zmenami a v čase písania tejto správy sa dostáva do praxe revolučná verzia CSS3. Problémom nie len pri internetových stránkach ale aj pri textových dokumentoch je prezentácia XHTML a CSS kódu v jednotlivých prehliadačoch. Tento problém vzniká z dôvodu rozdielnej implementácie špecifikácie týchto jazykov. Z tohto dôvodu nie je možné v niektorých špeciálnych prípadoch očakávať zhodné zobrazenie výsledku vo všetkých prehliadačoch. Podobne ako to platí aj pri zobrazení zdrojového DOCX dokumentu v rôznych textových editoroch.

5 Transformácia z Office Open XML do XHTML

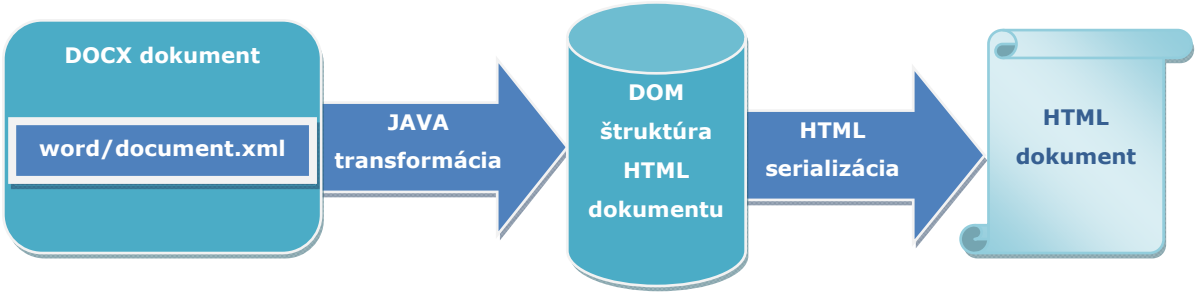
Zadanie práce sa týka transformácie dokumentov z Office Open XML do XHTML, pričom táto transformácia musí viesť do vytvorenia DOM štruktúry, z ktorej sa následne vyexportuje XHTML súbor. Office Open XML je norma na popis kancelárskych typov dokumentov ako sú dokumenty programu Microsoft Word, prezentácie z programu Microsoft PowerPoint a na koniec tabuľkové dokumenty z programu Microsoft Excel. Transformácia do XHTML má však praktický význam iba z dokumentov DOCX, ktoré sú v rámci Office Open XML písané podľa špecifikácie WordprocessingML. Tieto sú totiž svojim zameraním podobné a nahraditeľné s XHTML.

5.1 Rozdielne možnosti transformácie

Transformáciu je možné riešiť dvoma odlišnými spôsobmi. V nasledujúcich dvoch podkapitolách vysvetlím hlavnú myšlienku oboch spôsobov transformácie. Obe vedú najskôr na transformáciu do DOM štruktúry a až následne do jej perzistencie do HTML súboru. Na záver bude podkapitola s porovnaním výhod a nevýhod oboch metód transformácie.

5.1.1 Priama transformácia

Priamy spôsob transformácie spočíva v zostrojení sady prepisovacích pravidiel v programovacom jazyku Java, pre ktoré bude na vstupe XML súbor s obsahovou časťou dokumentu. Postupným prechádzaním tohto dokumentu, či už použitím parseru založenom na princípe DOM parseru alebo SAX parseru sa postupne pre každý obsahový element vstupného dokumentu vytvorí príslušný výstupný element v DOM štruktúre. Dôležité je, že z obsahového dokumentu DOCX súboru sú robené referencie do ďalších XML súborov s nastaveniami. Pri tejto transformácii je potrebné spracovávať popri obsahovom dokumente aj ostatné doplňujúce XML súbory a vyhľadávať v nich časti, ktoré sú referencované z hlavného obsahového dokumentu.



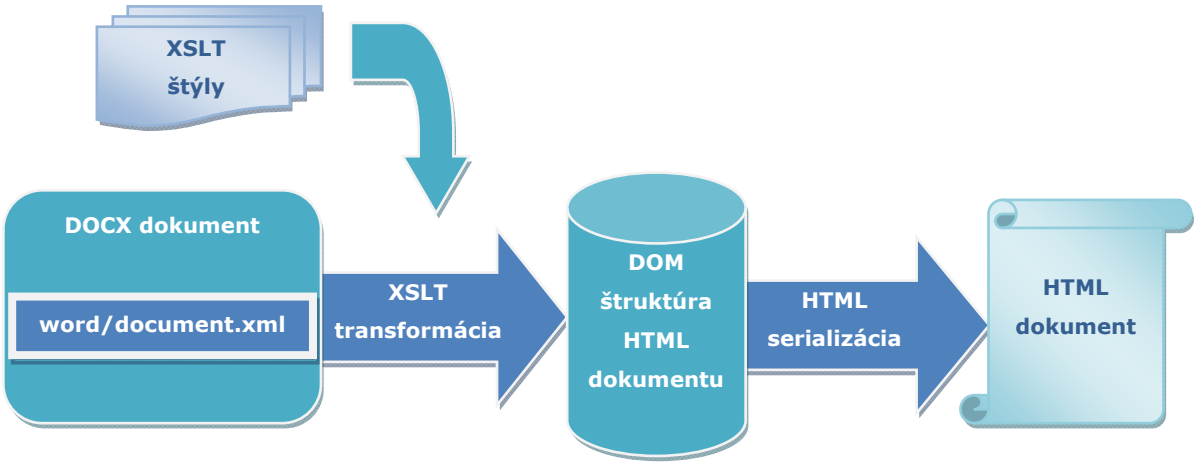
Obrázok 2 – Základná štruktúra priamej transformácie

Takýto spôsob spracovávania má priamo v Java kóde uložené pravidlá na prepisovanie. Toto môže byť z hľadiska doplňania transformácie o ďalšie časti, prípadne jej zmenu, nevýhodné. Za jednoznačnú výhodu takéhoto riešenia sa dá považovať použitie malého počtu prídavných knižníc a potreba znalosti menšieho počtu technológii.

5.1.2 Transformácia s použitím XSLT

Druhou možnosťou ako spracovávať DOCX dokument a transformovať ho do HTML je využitie technológie XSLT. Tento spôsob vyžaduje rovnako prepisovacie pravidlá, podobne ako v predchádzajúcom prípade. Rozdiel však spočíva v tom, že tieto sú uložené v podobe XML súboru v tvare XSLT štýlov. Rovnako ako v prípade priamej transformácie, aj v tejto je potrebné brať ohľad na dáta uložené v prídavných XML súboroch, ktoré sú referencované z hlavného obsahové dokumentu.

Výhodou tohto riešenia je nepochybne oddelenie Java kódu a procesora transformácie od prepisovacích pravidiel, ktoré sú uvedené v samostatnom riešení. Toto so sebou prináša väčšiu prehľadnosť a jednoduchšie pokračovanie v ďalších úpravách transformácie. Na opačnej strane, medzi nevýhody tohto riešenia patrí väčšie množstvo použitých technológií a spolu s tým spojený väčší počet používaných knižníc.



Obrázok 3 – Základná štruktúra transformácie s použitím XSLT

5.1.3 Porovnanie oboch možností transformácie

Obidve možnosti transformácie majú spoločný záver transformácie DOCX dokumentu v podobe serializácie DOM štruktúry už transformovaného dokumentu do výstupného HTML súboru. Rozdielny spôsob samotnej transformácie prináša pre oba spôsoby určité výhody a nevýhody, ktoré sú zhrnuté v nasledovnej tabuľke.

	Priama transformácia	Transformácia s použitím XSLT
Výhody	Menší počet použitých technológií a knižníc	Oddelenie prepisovacích pravidiel od Java kódu a procesora transformácie
Nevýhody	Všetky prepisovacie pravidlá sú v Java kóde a teda komplikovanejšie dopĺňanie, prípadne prerábanie transformácie	Väčšie množstvo použitých knižníc a technológií

Tabuľka 1 Prehľad možností transformácie

5.2 Vlastnosti pripravovanej knižnice

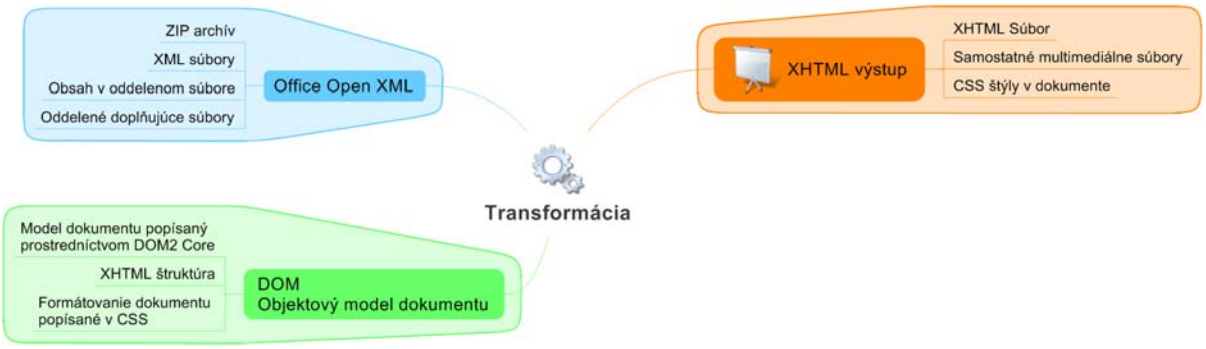
Riešenie transformácie je pripravované ako knižnica v programovacom jazyku Java. Spracovanie v podobe knižnice so sebou prináša niektoré vlastnosti, ako napríklad absenciu akéhokoľvek používateľského rozhrania. Aplikačné rozhranie knižnice bude umožňovať prácu s knižnicou v dvoch možných variantoch. Jeden variant je integrovanie knižnice do inej Java aplikácie a následné spustenie transformácie z Java kódu tejto aplikácie. Druhý variant počíta s možnosťou využívať knižnicu ako samostatnú aplikáciu a to prostredníctvom spustenia priamo z príkazového riadku.

Ďalšou vlastnosťou knižnice je možnosť mať v sebe integrované všetky potrebné knižnice tretích strán a v tom prípade bude knižnica jeden samostatný jar súbor, ktorý nebude potrebovať explicitne pridať do svojho classpath žiadne ďalšie knižnice. Iná možnosť je mať v knižnici uložené iba samostatné zdrojové kódy transformácie a všetky potrebné knižnice explicitne pridávať do classpath spúšťanej aplikácie. Výhoda vloženia knižníc priamo do knižnice transformácie je v stabilnosti, teda nebudú potrebné žiadne zmeny v použitých knižniciach a transformácia bude fungovať aj samostatne. Nevýhoda spočíva v možnej duplicite niektorých knižníc tretích strán, ktoré budú raz používané pre samostatnú aplikáciu a druhý krát ako súčasť knižnice transformácie.

Aplikačné rozhranie knižnice umožní zrealizovať proces transformácie ako jeden proces, teda vytvorí DOM a vzápätí ho exportuje do XHTML výstupu. Druhý spôsob je, že najskôr vytvorí DOM a vráti referenciu na túto štruktúru a až potom umožní, ako druhý krok, vyexportovať DOM do XHTML, pričom zrealizovanie druhého kroku nie je nutné.

5.3 Návrh transformácie

Celý proces transformácie sa skladá z 2 hlavných, po sebe idúcich procesov. V prvom procese sa zo vstupného Office Open XML súboru vytvorí DOM, podľa špecifikácie W3C. Druhý proces spočíva vo vytvorení XHTML dokumentu z DOM štruktúry, ktorá bola vytvorená v prvom procese. Ako je vidieť, druhý proces je závislý na prvom, takže v prípade, že sa nepodarí vytvoriť zo vstupného dokumentu model, nie je možné zrealizovať exportovanie do XHTML.



Obrázok 4 - Informatívny pohľad na transformáciu z Office Open XML do XHTML

Súčasťou celej transformácie sú ešte doplňujúce procesy, ktoré nepatria striktne ani do jednej časti, ale sú potrebné a využívané v oboch. Sem patrí proces spracovania multimediálnych súčastí vstupného dokumentu, ktoré sa nachádzajú v balíku dokumentu, ale pri výstupe budú ako samostatné súbory v príslušnom adresári výstupného XHTML súboru.

5.4 Analýza Office Open XML dokumentu a tvorba DOM

Táto časť je najzložitejšou v celom procese transformácie, pretože je potrebné implementovať značnú logiku v spôsobe, akým je kódovaný vstupný dokument. Ako som písal v časti, kde je rozoberané fungovanie Office Open XML, súbor na vstupe je komprimovaný archív, zvaný balík, ktorý vo svojej vnútornej štruktúre obsahuje niekoľko XML súborov. Prvým krokom je teda rozbalenie archívu a spracovanie prvého XML súboru, kde sú uložené, okrem iného, referencie na ďalšie časti obsahu a súbory špecifikujúce vlastnosti dokumentu.

V prípade, že sa v spracovávanom dokumente nachádzajú multimediálne dáta, ktoré sú uložené v samostatných súboroch v rámci balíku, budú tieto skopírované do výstupného adresára.
Postupným čítaním dokumentu v jazyku XML sa vytvárajú v DOM štruktúre, ktorá je stromová, nové elementy. Keďže je vstupný dokument popisovaný viacerými XML súbormi, ktoré sú postavené na princípe stromovej štruktúry, tvorba DOM by mala byť priamočiara. Takýto prístup je použiteľný pre väčšinu prvkov používaných v rámci Office Open XML WordprocessingML dokumentov.

Pri prechádzaní kódu dokumentu sú však miesta, kedy sa princíp zanárania poruší, nie na syntaktickej úrovni ale na sémantickej. Ako príklad je možné uviesť komentáre písané k nejakej časti.

```
<w:commentRangeStart w:id="3" />
<w:r w:rsidR="00FF0615" w:rsidRPr="0060190B">
<w:t>Architektúra balíka, umožňuje okrem primárneho uloženia do podoby ZIP archívu uložiť obsah aj do databázy, pričom vyhľadávanie ...</w:t>
</w:r>
<w:commentRangeEnd w:id="3" />
```

Takto používané značkovanie si vynucuje použitie stavového automatu pri riešení transformácie, na pamätanie, aký príkaz bol použitý a ako sa má ďalej spracovávať obsah medzi príkazmi.

5.4.1 Súbory textového dokumentu

Wordprocessing Markup Language (WordprocessingML) je jazyk z Office Open XML určený pre textové dokumenty DOCX. Štruktúra vnútorných súborov, z ktorých sa tieto dokumenty skladajú, je zhodná so všeobecnou štruktúrou popísanou v OPC. Tieto textové dokumenty DOCX sú práve tie, ktoré vieme transformovať do DOM a následne do XHTML.

Jednotlivé dokumenty v adresárovej štruktúre DOCX sa zo syntaktického hľadiska rozdeľujú do 2 skupín podľa typu ich obsahu. Pomocou vzájomných referencií sú obsahy z týchto súborov kombinované do výslednej interpretácie. Jednu skupinu tvoria súbory vlastností (z anglického properties), medzi ktoré patrí napríklad súbor so štýlmi, súbor s číslovaním zoznamov a podobne. Druhá skupina je tvorená obsahovými súbormi (z anglického stories), čo sú takzvané dejové línie. Sú to súbory s hlavným obsahom, prípadne sekcie, komentáre, záhlavia a podobne.

5.4.2 Štruktúra obsahového dokumentu

Dokument začína elementom `w:document`, v ktorom sa nachádza jeden element w:body. Na tomto mieste je vidieť analógiu medzi štruktúrou HTML zdrojového kódu so zdrojovým kódom OpenXML súboru. Ako jeden z možných potomkov elementu `w:document` je `w:background`. Tento prvok umožňuje nastavovať farbu pozadia dokumentu, prípadne nastavenie obrázka do pozadia. Nastavovanie pozadia dokumentu sa robí prostredníctvom ďalších elementov, ktoré sú súčasťou jazyka DrawingML, ktorý je súčasťou normy Office Open XML.

Ďalej sa obsah dokumentu, konkrétne elementu `w:body`, delí na štruktúrované prvky a odseky. Nasledujúca ukážka zobrazuje takéto členenie dokumentu:

```
<w:document>
  <w:body>
    <w:p>
    </w:p>
  <w:body />
<w:document />
```

5.4.3 Štýly a formátovanie

Štýly je možné v dokumente Office Open XML definovať 2 spôsobmi veľmi podobne, ako v prípade jazyka XHTML a CSS. V prípade, že sú triedy štýlov definované pre celý dokument, sú uložené v samostatnom súbore */word/styles.xml*.

5.4.3.1 Vopred definované triedy štýlov

Definícia týchto štýlov sa nachádza v samostatnom súbore */word/styles.xml* a bežne sa používa ako dopredu definovaná množina formátovacích pravidiel. Ukážku definovania štýlov predvediem na definovaní formátovania odsekov, ktoré sa používajú pre reprezentáciu nadpisov najvyššej úrovne.

```
<w:style w:type="paragraph" w:styleId="Heading1">
        <w:name w:val="heading 1"/>
        <w:basedOn w:val="Normal"/>
        <w:next w:val="Odstavecprvn"/>
        <w:qFormat/>
        <w:rsid w:val="00A13294"/>
        <w:pPr>
                <w:keepNext/>
                <w:numPr>
                        <w:numId w:val="1"/>
                </w:numPr>
                <w:spacing w:after="240"/>
                <w:outlineLvl w:val="0"/>
        </w:pPr>
        <w:rPr>
                <w:rFonts w:cs="Arial"/>
                <w:b/>
                <w:bCs/>
                <w:kern w:val="32"/>
                <w:sz w:val="48"/>
                <w:szCs w:val="32"/>
        </w:rPr>
        </w:style>
```

Takáto definícia obsahuje okrem elementov opisujúcich formátovanie príslušného elementu obsahu aj správanie sa tohto elementu k ďalším obsahovým elementom. Napríklad v predchádzajúcej ukážke definovania štýlu nadpisu prvej úrovne je vidieť, že prostredníctvom elementu `w:next` je možné definovať formátovanie nasledujúceho odseku. Pre transformáciu sú zaujímavé vlastnosti napríklad použitý font, veľkosť písma či farba písma.

5.4.3.2 Transformácia vopred definovaných štýlov

Transformácia štýlov, ktoré sú vopred definované, je realizovaná vytvorením nového elementu `<style>` v DOM štruktúre ako potomka elementu s názvom HTML. Tento element bude v časti exportu do XHTML použitý ako HTML element `<style>`, kde budú jazykom CSS popísané triedy štýlov. Keďže referencia do súboru *styles.xml* na príslušnú triedu štýlov je realizovaná pomocou unikátneho názvu, tento bude ponechaný aj naďalej ako identifikátor. Postupným prechádzaním súboru *styles.xml* sa pre každú triedu štýlov pridá do elementu `<style>` textový blok, ktorý toto formátovanie popisuje v už spomínanom jazyku CSS. V rámci DOM štruktúry takto transformovaného dokumentu nie je možné sa priamo referenciou dostať na konkrétne definovanie formátovania. Toto je spôsobené tým, že všetky definície formátovania sú uložené v jednom elemente `<style>`, kde sú už vedené iba ako neformátovaný text. Fungovanie referencie sa preukáže až pri vyexportovaní do XHTML, kde prehliadač správne použije referenciu medzi príslušným obsahovým blokom a konkrétnou definíciou formátovania v elemente `<style>`.

5.4.3.3 Štýly definované s obsahovým elementom a ich transformácia

Definícia štýlov uložená priamo vo vlastnostiach obsahového elementu dokumentu je analogická ako pri predchádzajúcej kategórii štýlov. Rozdielne bude iba spracovanie týchto štýlov, kde upravený prístup spočíva v transformácii, kedy sa pre príslušný element v DOM štruktúre pridá atribút `style`. V tomto atribúte bude uložená definícia formátovania v jazyku CSS. Atribút `style` je v jazyku XHTML povolený pre každý element, ktorý vytvára obsah dokumentu. Pri zobrazení výsledného dokumentu tak príde k aplikovaniu aj týchto štýlov.

5.4.3.4 Poradie aplikovania štýlov

Pre každý element dokumentu OpenXML, ako aj element dokumentu v XHTML, je možné definovať štýly viacerými spôsobmi a to aj súčasne. V prvom kroku sú použité prednastavené štýly aplikácie, ktorá s dokumentom pracuje. V druhom kroku sa aplikujú vopred definované štýly zo súboru */word/styles.xml*, prípadne po transformácii z elementu `<style>` v XHTML. Ako posledné sa aplikujú štýly definované priamo v obsahovom elemente. Prerývanie týchto definícií je riešené spôsobom prepisovania, teda posledne definovaný štýl prepíše tie pravidlá z pôvodného formátovania, ktorá má zadefinované.

5.4.4 Odseky v DOCX dokumentoch

Základné členenie obsahu dokumentu je na odseky, ktoré sú blokové. Odseky sú označované ako `w:p` a ich obsah tvorí jeden blok. Prvým potomkom je najčastejšie `w:pPr`, čo je označenie vlastností spoločných pre celý odsek. Medzi vlastnosti, ktoré sa vnútri vyskytujú patrí:

- zarovnávanie značené `w:jc`
- orámovanie `w:pBdr` s možnosťami nastaviť zvlášť orámovanie pre každú stenu.
- nastavenie odkazovaného štýlu pomocou `w:pStyle`, kde sa ako argument `w:val` použije názov štýlu, ktorý sa má aplikovať
- vertikálne zarovnanie obsahu v rámci odseku pomocou `w:textAlignment`. Táto možnosť sa využije napríklad v prípade, že je v jednom odseku použitých niekoľko rôznych veľkostí textu.

Nasledujúca ukážka zobrazuje základné členenie odseku. V ukážke je jednoznačne vidieť oddelenie nastavovania vlastností odseku a obsahu.

```
<w:p>
        <w:pPr>
                <w:jc w:val="center" />
                <w:pBdr>
                        <w:top w:val="single" w:sz="24" w:space="1" w:color="FF0000" />
                        <w:left w:val="single" w:sz="24" w:space="4" w:color="FF0000" />
                        <w:bottom w:val="single" w:sz="24" w:space="1" w:color="FF0000" />
                        <w:right w:val="single" w:sz="24" w:space="4" w:color="FF0000" />
                        <w:between w:val="single" w:sz="48" w:space="1" w:color="4D5D2C" />
                </w:pBdr>
        </w:pPr>
        <w:r><w:rPr><w:i /></w:rPr>
                <w:t xml:space="preserve">The quick brown fox jumped ...</w:t>
        </w:r>
</w:p>
```

Po nastavení vlastností odseku sa použije element spustenie (z anglického run) `w:r`, v ktorom je obsah odseku. Pre každé spustenie v odseku je možné definovať nastavenia obsahu ako je napr. `w:b` čo je označenie tučného písma, w:i pre kurzívu a w:u pre podčiarknuté písmo. Takisto je možné nastaviť orámovanie a farbu. Nastavenie týchto vlastností je rovnaké v celej časti `w:r`.

Obsahom spustenia je textový prvok `w:t`, ktorý obsahuje iba text bez akéhokoľvek kódu. Pre `w:t` vieme nastaviť jeho správanie sa voči bielim znakom, ktoré sú pred a/alebo po súvislom texte. S parametrom `w:t xml:space` nastaveným na hodnotu `preserve` sa medzery považujú za plnohodnotné znaky textového reťazca. To znamená, že sa pri nastavenej hodnote `preserve` pre výraz `<w:t xml:space="preserve"> medzery </w:t>` berú do úvahy aj 3 medzery, ktoré sú pred a za slovom „medzery“.

Každý element z `w:p` a `w:r` obsahuje atribúty, ktorých názvy majú tvar `rsid*`. Tieto atribúty sú dôležité pre producenta dokumentov na označovanie spoločných častí dokumentu, ktoré boli tvorené v rovnakom behu editácie (z anglického editing session).

- **rsidDel** – revízny identifikátor pre zmazanie odseku
- **rsidP** – revízny identifikátor pre vlastnosti odseku
- **rsidR** – revízny identifikátor pre odseku
- **rsidRDefault** - spoločný identifikátor pre všetky `w:r` elementy v rámci daného odseku, ktoré nemajú definovaný svoj vlastný `rsidR` atribút
- **rsidRPr** - revízny identifikátor pre glyph formátovanie odseku

Tieto atribúty spôsobujú rozdeľovanie obsahu na časti podľa toho, ako používateľ, producent, vytváral prípadne editoval obsah. Z pohľadu konzumenta, kam sa radí aj navrhovaná knižnica, je toto členenie zbytočné a pri transformácii ho treba odstrániť.

5.4.4.1 Transformácia odsekov

Vytváranie DOM štruktúry z takýchto elementov je nasledovné. Pre každý odsek, teda element `w:p`, sa vytvorí v DOM element `<p>`, čo reprezentuje v HTML blokový textový element.

5.4.4.2 Štýly odseku

Pokiaľ má daný odsek potomka `w:pPr`, bude potrebné pre tento odsek definovať štýly. Najjednoduchším spôsobom ako je možné celému odseku nastaviť jednu spoločnú sadu štýlov je využitím referencie do preddefinovaných štýlov, ktoré sú uložené v samostatnom súbore */word/styles.xml*. Keďže názov, podľa ktorého je vytvorená referencia v OpenXML súbore, je unikátny, bude tento použitý aj pri vytváraní referencie pre výstupný dokument. Pre nový element odseku bude nastavený atribút `class` na názov tejto referencie. Ďalšie nastavenia, ktoré nie sú realizované prednastavenými štýlmi, musia byť spravené pomocou atribútu `style` pre element odseku. Postupným prechádzaním elementmi `w:wpPr` bude pre každý element podľa jeho typu prevádzaný jeho obsah do jazyka CSS. Takto nastavené štýly budú platné pri exportovaní iba v časti jedného odseku.

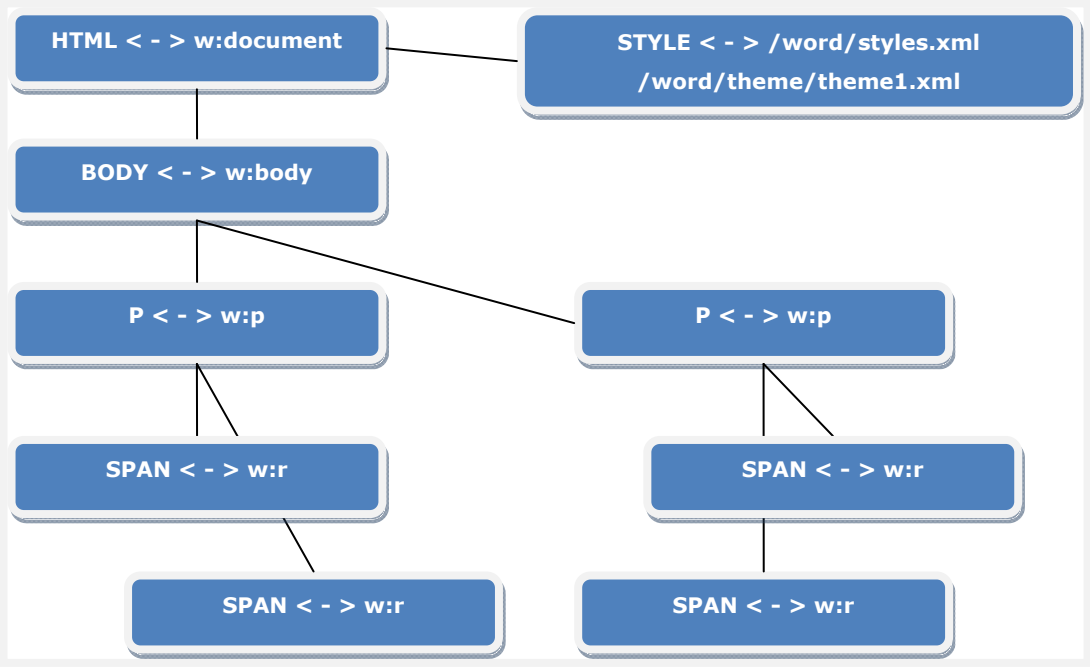
5.4.4.3 Redukovanie w:r elementov

Pre každý odsek `w:p` a vnorené `w:r` elementy sú používané atribúty `rsid*`, podľa ktorých sa súvislý text rozdeľuje do menších častí. Pri transformácii je potrebné tieto časti, ktoré sú rozdelené iba na základe týchto atribútov, spojiť do väčších celkov. Nasleduje ukážka segmentovania obsahu odseku do `w:r` elementov.

```
<w:p w:rsidR="008E18AC" w:rsidRPr="008E7E35" w:rsidRDefault="008E18AC">
        <w:r w:rsidRPr="008E7E35">
                <w:t xml:space="preserve">Téma diplomovej práce sa dotýka aktuálnej</w:t>
        </w:r>
        <w:r w:rsidR="00623241" w:rsidRPr="008E7E35">
                <w:t>otázky</w:t>
        </w:r>
        <w:r w:rsidRPr="008E7E35">
                <w:t xml:space="preserve">kancelárskych balíkov a možnosti
                transformovania</w:t>
        </w:r>
```

Pri takýchto rozdeleniach sa budú časti `w:r` spájať do jedného elementu, ktorý bude mať iba jeden textový element `w:t`. Pre prvý výskyt `w:t` transformácia vytvorí nový element odseku `<span>`, čo v HTML reprezentuje riadkový blok určený pre text. Pokiaľ sa v ďalších elementoch `w:r` nenachádza explicitné nastavenie štýlov pomocou `w:rPr`, potomkovia týchto elementov, teda textové elementy `w:t`, sa budú spájať do jedného prechádzajúceho `<span>` elementu. Týmto postupom sa odstráni, pre konzumenta, zbytočné segmentovanie častí.

V prípade, že bude pre niektorý `w:r` element definovaný element `w:rPr`, bude pre tento element v DOM štruktúre vytvorený nový element `<span>` spolu s atribútmi `class`, pre referenciu na vopred definovaný štýl, a atribút `style`, pre definovanie štýlov z nasledujúcich potomkov. Tento proces je analogický ako pri štýloch odsekov, ale aplikujú sa na element `<span>` v DOM štruktúre a nie na element `<p>`.



Obrázok 5 – Ilustračná schéma DOM štruktúry dokumentu tvoreného odsekmi

DOM štruktúra, ktorá sa takýmto procesom vytvorí, bude principiálne dodržiavať schému zobrazenú na predchádzajúcom obrázku. Zobrazená schéma nie je úplná a konečná, ale je možné ju rozširovať pridávaním doplňujúcich elementov ako sú napríklad prechod na nový riadok alebo horizontálna čiara.

5.4.5 Zoznamy v DOCX dokumentoch

Zoznamy sú v dokumentoch Office Open XML riešené ako za sebou idúce odseky, teda elementy typu `w:p`, ktoré ale majú vo svojich vlastnostiach element označujúci príslušnosť odseku do zoznamu. Na tejto úrovni, teda v obsahu odseku, nie je vôbec jasné, či sa jedná o číslovaný zoznam, alebo o zoznam bez poradia. Označenie zoznamu je tvorené elementom `w:numPr`, ktorý obsahuje dvoch dôležitých potomkov. Prvý z nich je element `w:ilvl`, ktorý má iba jeden atribút `w:val`. Tento element reprezentuje úroveň, na ktorej sa daný odsek v zozname nachádza. Jednoúrovňové zoznamy majú pre každú svoju jednu odrážku, číslovanú alebo nie, jeden odsek, pre ktorý je nastavená hodnota `w:val` vo `w:ilvl` na 0. Pri členitejších zoznamoch je táto hodnota 1, 2 ... podľa zanorenia odrážky v stromovej štruktúre zoznamu. Druhý element `w:numId` je znovu iba s jedným atribútom `w:val`, ktorého hodnota robí referenciu do súboru */word/numbering.xml* na element inštancie zoznamu. Nasledujúca ukážka zobrazuje jeden prvok záznamu. Na tejto ukážke je viditeľné, že nie je jasné či sa jedná o číslovaný zoznam alebo nie.

```
<w:p w:rsidR="0055532C" w:rsidRPr="001E6165" w:rsidRDefault="0055532C">
<w:pPr>
        <w:pStyle w:val="ListParagraph" />
        <w:numPr>
                <w:ilvl w:val="0" />
                <w:numId w:val="7" />
        </w:numPr>
</w:pPr>
<w:r w:rsidRPr="001E6165">
        <w:t xml:space="preserve">NamedNodeMap ... </w:t>
</w:r>
</w:p>
```

Inštancia zoznamu je element, ktorý na základe identifikátora priradí definíciu zoznamu. V tejto definícii je potom špecifikované, či sa jedná o zoznam číslovaný alebo nečíslovaný. V skutočnosti je každý zoznam chápaný ako číslovaný, ale namiesto čísel sa priradia každému prvku ľubovoľné symboly poskytované editorom dokumentu.

5.4.5.1 Transformácia zoznamov

Transformácia zoznamov je realizovaná nasledovným spôsobom. Pre každý odsek, ktorý sa spracováva sa skontroluje príslušnosť do zoznamu. V prípade, že sa takáto príslušnosť vyskytne, tak

pokiaľ je aktuálne vytváraný nejaký zoznam, skontroluje sa na základe elementu `w:numId` príslušnosť odseku do tohto zoznamu. V prípade, že sa táto príslušnosť potvrdí, pokračuje sa ďalším prvkom zoznamu. Ak nie, je potrebné predchádzajúci zoznam ukončiť a vytvoriť nový. Podľa referencie získanej cez inštanciu zoznamu sa zistí o aký typ zoznamu sa jedná a podľa toho sa použije nový typ elementu `<ul>` pre nečíslované zoznamy a `<ol>` pre číslované. Každý záznam zoznamu, hoci je vo vstupnom dokumente ako odsek v DOM štruktúre, bude realizovaný ako element `<li>`. Pre tento element sa dajú rovnako ako pre odsek nastaviť atribúty `class` a `style`.

5.5 Exportovanie DOM do XHTML

Poslednou časťou transformácie je exportovanie naplnenej DOM štruktúry do XHTML súboru. Tento krok procesu transformácie nie je komplikovaný, pretože vygenerovaná DOM štruktúra zodpovedá definícii XHTML.

Na začiatku exportovania je potrebné pridať element hlavičky `<head>` a presunúť do neho element `<style>`. Potrebné je do `<head>` elementu doplniť meta informácie ako napríklad kódovanie alebo autora dokumentu, či dátum vytvorenia.

Postup pri exportovaní spočíva ďalej už iba v postupnom prechádzaní DOM štruktúry do hĺbky, pričom pre každý element je realizovaný zápis do súboru v podobe:

1. Otvárací príkaz elementu vypíše < ako prvý znak, nasledovaný výpisom `tagName` elementu.
2. Pokiaľ element obsahuje atribúty, tieto sa vypíšu spôsobom `<názov atribútu>=“<hodnota atribútu>“`.
3. Ukončí sa znakom >.
4. Pokiaľ element nie je koncový a obsahuje ďalšie elementy, tieto sa rekurzívne spracujú rovnakým algoritmom na tomto mieste.
5. Pokiaľ sa jedná o element, ktorý je koncový a obsahuje textový odkaz, tento sa bez úpravy vypíše vo vnútri.
6. Uzatvárací príkaz element vypíše `</` ako prvé dva znaky nasledované výpisom `tagName` elementu ukončené znakom >.

V prípade spracovávania ďalších doplňujúcich elementov vytváraných v časti transformácie je potrebné, aby boli vytvorené s ohľadom na transformáciu a mali už v DOM štruktúre sémantiku príkazov jazyka XHTML. Druhý variant je upraviť algoritmus exportu tak, aby sa vo výstupnom dokumente tento tvar dodržal.

Na exportovanie, teda serializovanie DOM, existujú pre prostredie Java rôzne knižnice, ako napríklad knižnica Xalan[3].

[3] Dostupná na adrese http://xml.apache.org/xalan-j/

6 Implementácia knižnice

Pri popise možností ako realizovať transformáciu dokumentov Office Open XML do DOM štruktúry XHTML dokumentu prichádzali do úvahy dva spôsoby. Jedným bola transformácie prostredníctvom prepisovacích pravidiel, ktoré sú súčasťou Java kódu aplikácie. Druhý spôsob bol použitie XSLT súboru ako sady prepisovacích pravidiel.

6.1 Využitie XSLT

Pre implementáciu transformácie som sa rozhodol využiť možnosť XSLT. Nie však spôsobom vytvárania vlastných pravidiel a vlastného XSLT súboru.

Na stránkach Open Source Project Community[4] je vedený projekt pod názvom PowerTools for Open XML[5]. Tento projekt je v jazyku C# a rozširuje možnosti PowerShell operačných systémov Microsoft Windows. Okrem iných vlastností obsahuje aj možnosť transformovať textové dokumenty Open Office XML do XHTML prostredníctvom XSLT transformácie. Celý tento projekt, teda aj príslušný XSLT súbor, ktorý obsahuje pravidlá potrebné na transformáciu, je vydaný pod licenciou Microsoft Public License (Ms-PL)[6]. To umožňuje použiť tento XSLT súbor aj pri ďalšom samostatnom vývoji.

Nevýhoda použitia XSLT súboru na transformáciu DOCX dokumentov spočíva v odkazoch na vlastnosti jednotlivých častí, ktoré sú uložené v samostatných súboroch v rámci OPC dokumentu. Existujú dva rôzne prístupy ako riešiť tento problém. Jedna možnosť je načítať do pamäte obsah ostatných doplňujúcich dokumentov a pri spracovávaní hlavného obsahového dokumentu každú potrebnú vlastnosť hľadať v týchto spracovaných dokumentoch uložených v pamäti. Alternatíva uloženia dokumentov do pamäte je využívať vhodným spôsobom DOM parser.

Druhý spôsob, ktorým je riešená aj transformácia v aplikácii PowerTools for Open XML, je vytvorenie takzvaného Flat OPC súboru. OPC definuje uloženie viacerých súborov v ZIP archíve a teda viac súborov reprezentujúcich jeden dokument vystupuje ako jeden spoločný. Flat OPC je označenie dokumentu, v ktorom sú tieto časti pospájané do jedného spoločného súboru.

[4] Open Source Project Community je server zameraný na podporu open source projektov na adrese http://www.codeplex.com

[5] Dostupný na adrese http://powertools.codeplex.com

[6] Licencia je dostupná na adrese http://powertools.codeplex.com/license

6.2 Flat OPC

Flat OPC je označenie vygenerovaného dokumentu v ktorom sú za sebou pospájané jednotlivé časti OPC dokumentu. Nejedná sa o oficiálnu normu OPC, ale je to formát používaný pri transformáciách DOCX dokumentu do XHTML prostredníctvom XSLT.

Princíp spočíva v uložení jednotlivých častí DOCX dokumentu, ktoré tvoria vlastné XML súbory v rámci DOCX dokumentu do jedného spoločného XML súboru. Dokumenty sú uložené v tomto spoločnom dokumente za sebou. Pre každú vloženú časť je definovaná pôvodná cesta súboru a jej označenie v rámci OPC. Nasledujúci fragment poskytuje krátku ukážku z vygenerovaného Flat OPC dokumentu.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<?mso-application progid="Word.Document"?>
<pkg:package xmlns:pkg="http://schemas.microsoft.com/office/2006/xmlPackage">
<pkg:part pkg:name="/docProps/app.xml" pkg:contentType="application/vnd.openxmlformats-
officedocument.extended-properties+xml">
<pkg:xmlData>
        <Properties xmlns="http://schemas.openxmlformats.org/officeDocument/2006/extended-
        properties"
        xmlns:vt="http://schemas.openxmlformats.org/officeDocument/2006/docPropsVTypes">
                <Template>Normal.dotm</Template>
                <TotalTime>1</TotalTime>
        </Properties>
```

Treba pripomenúť, že Flat OPC formát nie je identický s formátom, ktorý bol používaný v staršej verzii Microsoft Office produktoch pod názvom „Word 2003 XML Document“. Tieto formáty majú navzájom výrazne odlišnú schému. [1]

6.3 Využitie externej knižnice s podporou DOCX

Transformáciu, ktorá prevedie DOCX dokument do výstupného XHTML súboru som realizoval prostredníctvom XSLT transformácie, ktorá prevedie vstupný Flat OPC súbor na výstupný XHTML. Potrebné však bolo vyriešiť vytvorenie Flat OPC dokumentu a pripraviť obslužné programy, ktoré sú nevyhnutné na sémanticky správnu transformáciu. Transformácia prostredníctvom čistej XSLT transformácie môže totiž generovať iba jednoduché zoznamy, ktoré nie sú založené na zanáraní a elementoch `<ol>` prípadne `<li>`, viac v kapitole 6.7 Implementácia transformácie zoznamov.

Oproti .NET prostrediu, ktoré obsahuje pre prácu s Office Open XML dokumentmi samostatný framework „Open XML Format SDK 2.0“, chýba v prostredí jazyka Java priama podpora pre spracovanie týchto dokumentov. Existuje viacero knižníc tretích strán, ktoré poskytujú túto

podporu, ale v mnohých prípadoch ide o rozpracované pokusy, ktoré nepokrývajú všetky základné možnosti práce s dokumentmi DOCX.

6.3.1 Knižnica POI-OpenXML4J

Jedna z knižníc, ktorá má pomerne dobre spracovanú prácu s OPC je knižnica s názvom POI OpenXML4J[7]. Táto knižnica vznikla pod názvom openxml4j a jej obsahom bola iba funkcionalita pre operácie nad OPC. Poskytovala podporu pre vytáranie archívu Office Open XML dokumentov a prístup k jednotlivým jej častiam. V tejto knižnici však absentovala akákoľvek ďalšia podpora pre konkrétny typ dokumentov ako DOCX, XSLX a podobne. Neskôr prešla knižnica pod Apache POI, čo je Java API pre prácu s Microsoft Office dokumentmi. Momentálne je teda openxml4j vedená ako súčasť Apache POI a je charakterizovaná ako čistá Java implementácia Open Packaging Conventions (OPC).

6.3.2 Knižnica docx4j

Knižnica, ktorú som sa rozhodol použiť pre podporu riešenia transformácie, je knižnica s názvom docx4j[8]. Táto knižnica poskytuje operácie na prácu s OPC. Poskytuje aj podporu pre prácu s obsahom DOCX dokumentov. Od poslednej finálnej verzie knižnice v2.1.0[9], ktorá je dostupná od novembra roka 2008, je súčasťou aj transformácia DOCX dokumentu do XHTML práve prostredníctvom XSLT súboru, ktorý je súčasťou aplikácie PowerTools for Open XML. Prínos tejto knižnice v oblasti transformácie je skonštruovanie Flat OPC dokumentu zo vstupného DOCX súboru. Využitie tejto knižnice v mojej práci bude práve pre vytvorenie tohto vstupného dokumentu pre XSLT transformáciu a ďalej využitie podpory, pre prístup do ďalších častí OPC dokumentu.

Zaujímavý prístup použitý v tejto knižnici je v práci s XML dokumentmi. Pre štandardnú prácu ako je čítanie, zápis a modifikácia XML sa používajú parsre, napríklad DOM parser alebo SAX parser. Pokiaľ je dopredu známa schéma dokumentov, ktoré sa budú spracovávať, tak ako v prípade dokumentov DOCX, kde je schéma pevne daná normou ISO-IEC 29500, je možné použiť JAXB, opísaný v kapitole 6.4.4 Spracovávanie XML dokumentov v Jave, ktorý je súčasťou Java API. Triedy vygenerované JAXB kompilátorom, potrebné pre prácu so spracovávaným dokumentom sú súčasťou tejto knižnice.

[7] Dostupná na adrese http://poi.apache.org/oxml4j/index.html

[8] Dostupná na adrese http://dev.plutext.org/blog/category/docx4j/

[9] Verzia je dostupná od 11. novembra 2008 na adrese http://dev.plutext.org/blog/2008/11/11/docx4j-v210-released/

6.3.3 Priamy prístup k obsahu dokumentov

Iný prístup k transformácii DOCX dokumentov je programový kód založený na princípe SAX parser. Takýto príklad je známy pod názvom Open XML to XHTML Translator (Java-based), ktorého autorom je Yoko Ikeda zo spoločnosti Toshiba. Java API obsahuje rozhranie na prácu so ZIP súbormi, takže je jednoduché rozbaliť Office Open XML dokument a ďalej pracovať s jednotlivými časťami. Pri transformácii hlavného obsahového dokumentu sa pre každú doplňujúcu vlastnosť musí volať SAX parser nad príslušným doplňujúcim XML súborom. V ukážke[10], ktorá bola sprístupnená na serveri zameranom na vývoj aplikácií pracujúcich s Open Office XML[11], je pripravená implementácia na transformáciu tabuliek, zoznam a odsekov. Avšak tento prístup nebol dokončený do podoby komplexnejšej knižnice a absentuje akákoľvek dokumentácia alebo popis riešenia.

6.3.4 Porovnanie knižníc

Pre potreby transformácie DOCX dokumentov som skúmal a vyskúšal spomenuté knižnice. V nasledujúcej tabuľke je prehľad výhod a nevýhod jednotlivých knižníc, medzi ktorými som sa rozhodoval.

	Výhody	Nevýhody
POI-OpenXML4J	Knižnica vedená pod správou vývojovej skupiny Apache. Knižnica je súčasťou frameworku Apache POI.	Knižnica neobsahuje žiadnu podporu pre DOCX dokumenty. Podpora je len pre OPC.
Docx4j	Knižnica poskytuje úplnú podporu pre spracovania DOCX dokumentov spolu s možnosťou exportovania do HTML založenom na XSLT	Knižnica je pomerne obsiahla a primárne určená na spoluprácu pri vytváraní DOCX dokumentov. Exportovanie do HTML je len okrajovou súčasťou.
Priama transformácia, Yoko Ikeda	Nejedná sa priamo o knižnicu a teda obsahuje len programovací kód na transformáciu, čo je prehľadnejšie	Generovanie výstupu je závislé iba na pravidlách v programovacom kóde, čo v prípade akejkoľvek zmeny znamená zásah do zdrojového kódu

Tabuľka 2 Porovnanie výhod a nevýhod jednotlivých knižníc

[10] Ukážka je dostupná na adrese http://openxmldeveloper.org/articles/OpenXMLtoXHTMLinJava.aspx

[11] OpenXML Developer, http://openxmldeveloper.org

6.4 Využité technológie pri transformácii

Pri realizovaní samotnej transformácie dokumentu som použil okrem Java Standard Edition a príslušných knižníc niekoľko doplňujúcich technológii, ktoré v niektorých častiach transformácie iba uľahčili prácu s danou problematikou a ich použitie bolo dobrovoľné, v niektorých prípadoch bolo ich použitie nutné.

6.4.1 XPath

XML Path Language (XPath) je technológia používaná pri práci s XML dokumentmi, pri ich vytváraní alebo editovaní. World Wide Web Consortium vedie túto technológiu od roku 1999.

XPath je dotazovací jazyk určený na vyhľadávanie elementov v DOM štruktúrach, napríklad v XML súbore. Môže byť tiež použitý na zisťovanie a výpočet hodnôt ako sú textové reťazce, čísla alebo logické premenné.[2]

XPath je v mojej práci využité na jednoduché a priamočiare nájdenie všetkých elementov s názvom OL, ktoré predstavujú kontajnery pre prvky zoznamov. S použitím XPath bolo nájdenie prvkov podstatne jednoduchšie, nakoľko stačilo nad DOM štruktúrou vytvoriť objekt XPathExpression a spustiť jeho funkčnosť. Funkcia spracovávajúca XPath vrátila zoznam elementov, pre ktorý už stačilo spustiť príslušnú logiku. Nasledujúca ukážka zobrazuje využitie XPath v jazyku Java na vyhľadanie všetkých zoznamov.

```
XPathExpression expr = xpath.compile("//ol[@hierarchy]");
Object result = expr.evaluate(doc, XPathConstants.NODESET);
```

6.4.2 Java Reflection API

Java okrem klasického prístupu k aplikačným objektom v Java runtime, umožňuje aj špeciálny prístup prostredníctvom Java Reflection API. Tento prístup umožňuje prezerať a meniť chovanie objektov počas ich životného cyklu v runtime.

Java Reflection API pozostáva z triedy `java.lang.Class` a tried z balíka `java.lang.reflect` ako Field, Method, Constructor, Array alebo Modifier. Tieto triedy reprezentujú zodpovedajúce rozhranie a prvky definovania tried. Prostredníctvom nich je možný prístup ku všetkým častiam objektu cez klasické Java API a teda realizovať ľubovoľné operácie ako napríklad zistenie rodičovskej triedy objektu, zistenie triedy objektu prostredníctvom konštrukcie `instance of` a podobne. Možný je prístup ku všetkým metódam triedy objektu, prípadne k jeho členským premenným.

Java Reflecttion API sa využíva napríklad na dopĺňanie syntaxe písaného kódu, alebo v aplikačných serveroch pri prechode z názvu tried v konfiguračnom súbore ako *web.xml* k inštancii triedy Servlet, ktorý spracováva požiadavky.[4]

Využitie Java Reflection API v mojej práci je spôsobené nevhodným navrhnutím rozhrania triedy knižnice docx4j a prístupu k súkromnej členskej premennej. Konkrétny prípad použitia je pri transformácii zoznamov, kde v rozhraní pre získavanie hodnôt nastavenia prvkov zoznamu nie je priamy prístup k typu formátovania prvku. K dispozícii je formátovací reťazec, pomocou ktorého sa odvodí zobrazované očíslovanie prvku a hodnota číslovania. Typ formátovania, ako je popísaný v špecifikácii Open Office XML je v danej triede prístupný len ako súkromná členská premenná s dostupnými súkromnými prístupovými metódami. Vzhľadom na to, že je celá knižnica postavená na tom, že docx4j iba využíva, ale priamo ju nemodifikuje, je využitie Java Reflection API jednou z najjednoduchších možností ako získať príslušnú hodnotu. Ukážka zobrazuje využitie Java Reflection API pre zavolanie súkromnej metódy.

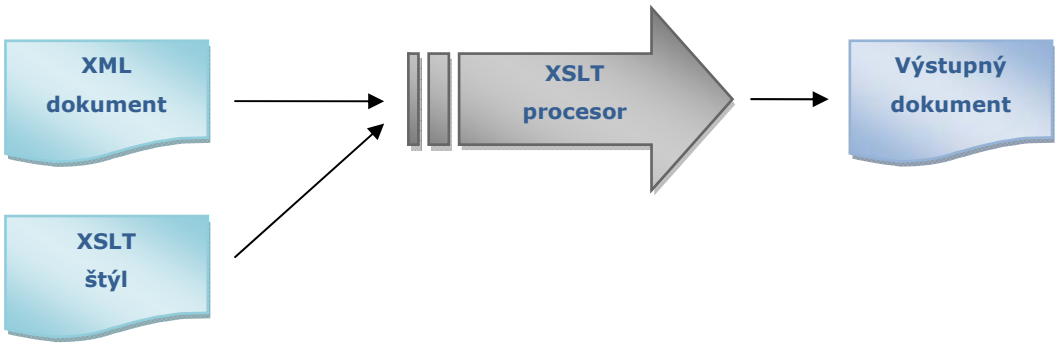
```
Class<?> c = ListLevel.class;
Object t = level;
try {
        Method m = c.getDeclaredMethod("getNumFmt", new Class[0]);
        m.setAccessible(true);
        Object o = m.invoke(t, new Object[0]);
        if (o instanceof NumberFormat) {
                return (NumberFormat) o;
        }
}
```

Potrebné je upozorniť, že v prípade využívanie transformačnej knižnice v prostredí jazyka Java, ktoré bude nastavené podobne, ako v prípade mnohých webových kontajnerov, kde je zakázané využívanie Java Reflection API, bude takéto získavanie hodnoty vytvárať výnimky v behu programu. V takomto prípade je nevyhnutné pristúpiť k úprave externej knižnice docx4j a upraviť príslušnú triedu a jej metódy ako verejne prístupné.

6.4.3 XSLT

Extensible Stylesheet Language Transformations (XSLT) je jazyk založený na XML používaný na transformáciu XML dokumentov do iných jazykov, avšak tiež založených na XML. Dôvod prečo sa zaoberať transformáciou XML dokument je nasledovný. Značkovací jazyk XML slúži k ukladaniu štruktúrovaných dát pomocou prvkov a atribútov. Často však nie je potrebné dáta iba ukladať ale ich aj vhodným spôsobom reprezentovať. Dokument vo formáte XML obsahuje iba informácie o logickom usporiadaní dát ale žiadne informácie o tom, ako majú byť údaje reprezentované. Z tohto dôvodu je nutné pomocou nejakého mechanizmu zmeniť dokumenty vo formáte XML do inej zobraziteľnej podoby.[7]

Štandard XSLT slúži k transformácii zdrojových dokumentov XML, prípadne iba ich častí, na výstupné dokumenty XML prostredníctvom transformačných pravidiel, ktoré sú uvedené v dokumente vo formáte XML, ktorý sa nazýva XSLT štýl (XSLT stylesheet).[7]



Obrázok 6 – Schéma transformácie XML dokumentov prostredníctvom XSLT

6.4.4 Spracovávanie XML dokumentov v Jave

Extensible Markup Language (XML) a Java technológia je častá kombinácia na pomoc vývojárom pri výmene dát a programov. Z tohto dôvodu sa z XML stal uznávaný štandard na výmenu dát medzi rôznorodými systémami, pričom Java technológia poskytuje platformu na budovanie prenositeľných aplikácií. Táto kombinácia je obzvlášť častá pri poskytovaní webových služieb.

Zaujímavý je ale práve spôsob, ako pristupovať k XML dokumentom a ako ich používať v programovacom jazyku Java. Jeden klasický a najtypickejší spôsob je použitie programového kódu parser založených na Simple API for XML (SAX)[12] alebo Document Object Model (DOM)[13]. Oba tieto spôsoby sú zahrnuté pod spoločným Java API for XML Processing (JAXP)[14]. V programovom kóde jazyka Java je možné využívať SAX a DOM parser prostredníctvom JAXP API. To znamená, že spracovávaný XML dokument je načítaný a rozdelený do samostatných logických častí. Takto spracovaný obsah je potom prístupný v aplikácii. [16]

6.4.4.1 SAX Parser

V prípade využitia SAX parser je situácia trochu odlišná. Parser začne spracovávať začiatok dokumentu a sekvenčne spracováva každú časť dokumentu až po jeho koniec. Pri tomto prístupe sa žiadne dáta neukladajú do pamäte. Aplikácia môže vykonávať úlohy nad dátami vtedy, keď sú aktuálne načítané a sprístupnené v aplikácii, ale nemôže vykonávať žiadne operácia v pamäti, ktoré by manipulovali s dokumentom.

6.4.4.2 DOM Parser

V prípade využitia prístupu DOM parser sa vytvorí stromová DOM štruktúra, ktorá reprezentuje obsah spracovávaného XML dokumentu. Tento model dokumentu je uložený v pamäti aplikácie, čo

[12] Bližšie definovaný na adrese http://www.saxproject.org/

[13] Bližšie definovaný na adrese http://www.w3.org/DOM/

[14] Technológia je súčasťou Java API, dostupná na adrese https://jaxp.dev.java.net/

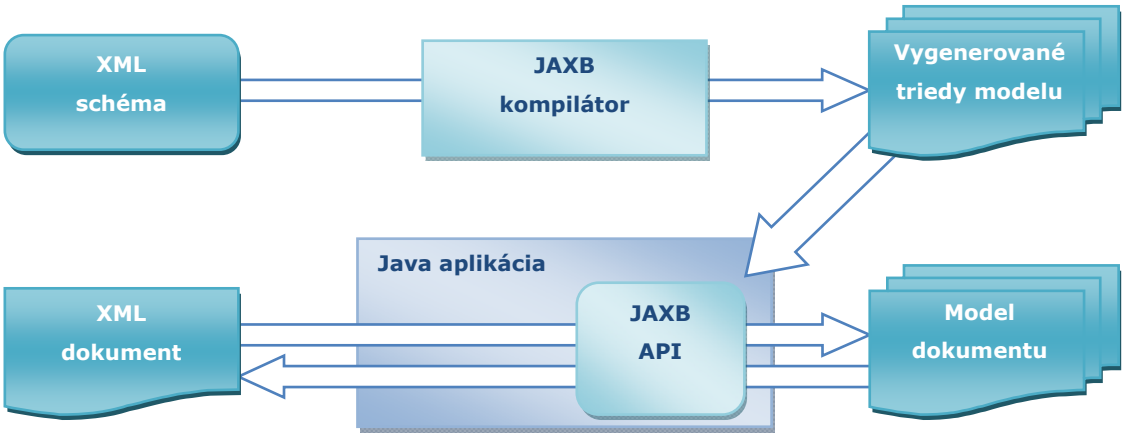
umožňuje ľubovoľný prístup k akýmkoľvek dátam z dokumentu. Takisto je možné meniť obsah a štruktúru tejto reprezentácie dokumentu.

6.4.4.3 JAXB

Vývojom webových služieb sa viac času venovalo aj možnostiam spracovávania XML dokumentov. Nový prístup, ktorý je súčasťou Java API je Java Architecture for XML Binding (JAXB)[15]. Princíp JAXB spočíva vo vytvorení tried modelu dokumentu z vopred známej schémy XML dokumentov a následného vytvorenia inštancie modelu týchto tried z načítaného vstupného dokumentu.

JAXB API je definované v balíku `javax.xml.bind` a tvorí ho skupina rozhraní prostredníctvom ktorej je možné v Java aplikácii pracovať s vygenerovanými triedami tvoriacimi model dokumentu. Ústredným bodom JAXB API je JAXBContext, ktorý tvorí vstupný bod v spracovávaní zdrojového dokumentu. Poskytuje abstrakciu na prácu so spracovávaným XML dokumentom prostredníctvom troch hlavných operácií definovaných samostatnými rozhraniami. [17]

- **Demaršalovanie** – proces deserializácie XML dát do modelu dokumentu v Jave. Tento krok môže prípadne obsahovať aj validáciu vstupného XML dokumentu počas demaršalovania.
- **Maršalovanie** – proces serializácie, kedy sa z pripraveného modelu v Jave vytvára spätne príslušná XML reprezentácia dokumentu
- **Validácia** – kontrola spracovávaného dokumentu prostredníctvom objektového grafu v pamäti



Obrázok 7 – Orientačná schéma JAXB [16]

Postup spracovávania XML dokumentov prostredníctvom tejto technológie, ktorá je použitá aj knižnicou docx4j je nasledovný. V prvej časti, pri vývoji knižnice, bolo potrebné zo schém Office Open XML vygenerovať triedy modelu. Tieto je možné editovať a upravovať ich vlastnosti, ako to bolo v príklade zoznamov a ich vlastností ich formátovania. Tieto vygenerované a upravené triedy sú súčasťou balíka docx4j.

[15] Definovaný na stránkach Java na adrese
http://java.sun.com/webservices/technologies/index.jsp#Core_Web_Services

V druhej časti, ktorá už prebieha počas behu aplikácie, ktorá využíva docx4j, sa vstupný dokument spracuje a vytvorí sa jeho model. Toto spracovanie zodpovedá procesu demaršalovania, tak ako je definovaný v JAXB. Pri demaršalovaní je možné spraviť aj validáciu vstupných údajov, aby vytvorený model bol korektný.

6.5 Postup transformácie dokumentu

Transformácia zo vstupného DOCX súboru do výstupného XHTML sa skladá z viacerých častí. Tieto časti na seba priamo nadväzujú a sú navzájom závislé. Na transformáciu využívam knižnicu docx4j spolu s ďalšími knižnicami, ktoré sú potrebné pre jej fungovanie.

V prvom kroku transformácie je potrebné zo súboru DOCX, ktorý je vo formáte OPC vytvoriť Flat OPC dokument. Túto funkčnosť obsahuje knižnica docx4j a ponúka ďalej na prácu už spojený XML dokument. V implementácii výslednej transformačnej knižnice je možné pomocou prepínača nastaviť generovanie tohto XML dokumentu do súboru pre prípadné ďalšie účely.

V druhom kroku transformácie sa z tohto Flat OPC dokumentu vytvorí pomocou JAXB model dokumentu do ktorého sú mapované jednotlivé entity XML súborov. Ďalšie doplňujúce XML súbory z OPC balíka, ktoré nie sú súčasťou vygenerovaného Flat OPC sú taktiež spracovávané prostredníctvom JAXB. Takto pripravený model je reprezentáciou pôvodného DOCX súboru a je možné cez neho pristupovať k jednotlivým jeho hodnotám a častiam.

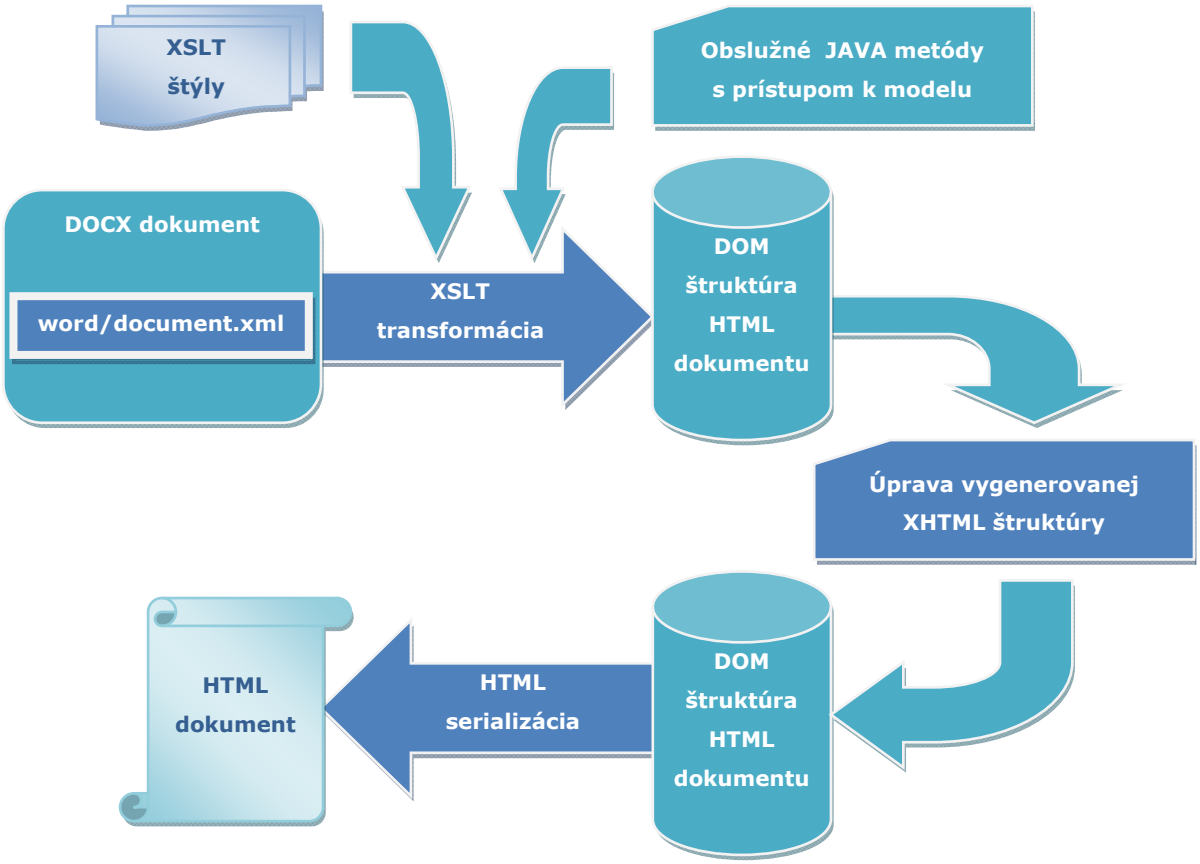
V treťom kroku transformácie je pre takto spracovaný vstupný DOCX súbor spustená XSLT transformácia. Rozhranie transformácie v Jave je súčasťou Java API. Implementáciu tohto rozhrania však treba do aplikácia pridať z externých zdrojov. Jedna z doplňujúcich knižníc, ktoré sú súčasťou docx4j knižnice, je implementácia XSLT Procesora Xalan v modifikovanej verzii s označením 2.7.0. Tá umožňuje v štýloch XSLT súboru viacnásobné definovanie premenných, volanie statických metód Java tried a podobne. Tieto vlastnosti sú dôležité, nakoľko je potrebné pri transformácii volať obslužné podprogramy v Jave.

Vstupom tejto transformácie je Flat OPC dokument, ku ktorému je do transformácie pomocou parametra pridaná aj referencia na model spracovávaného dokumentu. Výstup transformácie je oproti klasickému výstupu do dátového prúdu urobený do DOM modelu. Toto je potrebné pre dokončenie transformácie, ktorá si vyžaduje ešte úpravu transformovaného výstupu.

Štvrtý krok transformácie spočíva v dokončení transformačného algoritmu v podobe úpravy DOM štruktúry. Pri tejto úprave sa vykonáva presunutie niektorých HTML elementov najmä v prípade spracovania zoznamov.

Posledným krokom transformácie je exportovanie vygenerovanej DOM štruktúry dokumentu do XHTML súboru. V prípade že používateľ nevyužíva túto knižnicu ako samostatný transformačný nástroj, ale je súčasťou inej Java aplikácie, je možné proces ukončiť pred spustením tohto posledného

kroku. Vtedy je na výstupe transformácie iba DOM štruktúra stransformovaného súboru. Inak je týmto krokom štandardne ukončený proces transformácie.



Obrázok 8 – Ilustračná schéma procesu transformácie

6.6 Implementácie transformácie nadpisov

Nadpisy, ktoré sú známe v XHTML ako text, ktorý je obalený značkami H1, H2 až H6 podľa úrovne nadpisu, sa v dokumente DOCX vyskytujú ako samostatné obyčajné odseky. Z pohľadu syntaxe ich nič neodlišuje od štandardných odsekov. Pre ich identifikovanie je potrebné spoliehať sa na použité štýly, ktoré sú aplikované na celý odsek.

Transformácia samotného obsahu nadpisu je závislá na zistení použitého štýlu pre formátovanie daného odseku. Názov štýlu, tak ako je prezentovaný v textových editoroch, je však iba lokalizovaný názov, ktorý sa môže meniť použitím rôznych jazykových verzií editorov. V dokumente *word/styles.xml*, ktorý je účasťou DOCX súboru a definuje použité štýly, sú definované výnimky v podobe kľúčových slov, ktoré nemôžu identifikovať žiadne používateľské štýly. Medzi tieto výnimky patrí aj pomenovanie štýlov nadpisov. Na nasledujúcej ukážke je zobrazené definovanie výnimiek pre prvé 2 úrovne nadpisov.

```
<w:lsdException w:name="heading 1" w:semiHidden="0" w:uiPriority="9"
w:unhideWhenUsed="0" w:qFormat="1" />
<w:lsdException w:name="heading 2" w:uiPriority="9" w:qFormat="1" />
```

Tieto názvy, konkrétne `heading 1` a postupne až `heading 9`, sú unikátne a majú rovnaký význam v každom dokumente DOXC. Označujú nadpisy v dokumentoch a spolu pokrývajú 9 úrovní nadpisov.

Pri každom odseku v obsahovej časti je definovaný použitý štýl podľa jeho identifikátora. Tento identifikátor sa, napríklad pri použití editora Microsoft Word 2007, generuje na základe použitej lokalizovanej verzie editora. Ako príklad uvediem testovací dokument použitý na overenie správnosti transformovanie nadpisov vytvorený v českej verzii. Na nasledujúcej ukážke je zobrazené uplatnenie štýlu nadpisu prostredníctvom lokalizovaného identifikátora.

```
<w:p w:rsidR="00CE179F" w:rsidRDefault="00CE179F" w:rsidP="00CE179F">
<w:pPr><w:pStyle w:val="Nadpis3" /></w:pPr>
<w:r><w:t>Pokusný nadpis 3</w:t>
</w:p>
```

V takomto prípade je potrebné pri transformácii prostredníctvom XSLT využiť možnosti použitej implementácie XSLT procesora a volať statické metódy v jazyku Java, ktoré majú prístup k celému modelu DOCX dokumentu. Pre použitý identifikátor sa prostredníctvom Java kódu získa hodnota pôvodného štýlu, ktorý ak je zhodný s jedným z reťazcov `heading 1` až `heading 9`, tak reprezentuj štýl nadpisu. Vzhľadom na to, že v XHTML je definovaných iba 6 úrovní nadpisov, prekladá sa z DOCX dokumentu iba prvých 6 úrovní. Volané Java metódy najskôr vrátia logickú hodnotu, či daný odsek reprezentuje nadpis a v prípade kladnej odpovede sa volaním druhej metódy získa úroveň nadpisu. Spracovanie úrovne nadpisu do príslušnej XHTML značky je už ďalej realizované prostredníctvom XSLT. Táto úprava nadpisov si vyžiadala prepísanie pôvodného spracovávania odsekov v XSLT a vytvorenie nového XSLT štýlu, ktorý presúva spracovávanie pôvodných odsekov až za zisťovanie prítomnosti nadpisov.

6.7 Implementácia transformácie zoznamov

Zoznamy sú v dokumentoch DOCX riešené ako obyčajné odseky `w:p` . Jednou z vlastností odsekov je, že sa nesmú navzájom zanárať. Na základe tejto vlastnosti sú teda zoznamy riešené ako jednoúrovňové odseky s referenciou na vlastnosti zoznamov, ako je opísané v kapitole 5.4.5 Zoznamy v DOCX dokumentoch.

Implementácia transformácie takto riešeného jednoduchého zoznamu do viacúrovňového zoznamu so zanáraním vyžaduje použitie stavového systému, ako na to bolo poukázané v kapitole 5.4 Analýza Office Open XML dokumentu a tvorba DOM.

Pôvodné riešenie transformácie zoznamov v používanom XSLT súbore pracuje bez použitia HTML značiek `<ol>` alebo `<ul>`. Každý jeden odsek, ktorý v DOCX reprezentuje jeden prvok zoznamu, je vo výstupnom HTML formáte reprezentovaný ako HTML odsek `<p>`. Odsunutie, ktoré vizuálne pôsobí ako využívanie zoznamov, je získané prostredníctvom parametru `ilvl`.

Nasledujúca ukážka zobrazuje zdrojový kód záznamu v dokumentoch Office Open XML.

```
<w:p w:rsidR="00DB4B02" w:rsidRPr="008248C4" w:rsidRDefault="00DB4B02"
w:rsidP="00DB4B02">
<w:pPr>
        <w:pStyle w:val="ListParagraph" />
        <w:numPr>
                <w:ilvl w:val="0" /><w:numId w:val="5" />
        </w:numPr>
</w:pPr>
<w:r w:rsidRPr="008248C4"><w:t>Položka</w:t></w:r>
</w:p>
<w:p w:rsidR="00DB4B02" w:rsidRDefault="00DB4B02" w:rsidP="00DB4B02">
<w:pPr>
        <w:pStyle w:val="ListParagraph" />
        <w:numPr>
                <w:ilvl w:val="1" /><w:numId w:val="5" />
        </w:numPr>
</w:pPr>
<w:r><w:t>Položka</w:t></w:r>
</w:p>
```

Pôvodná transformácia použitá v XSLT bola síce syntakticky správna z pohľadu jazyka HTML, ale generovaný výstup bol zo sémantického pohľadu na dokument nekorektný a bolo by komplikované z neho následnou analýzou správne získavať obsah zoznamov.

Nakoľko nie je možné sa pri spracovávaní pomocou XSLT vrátiť naspäť k predchádzajúcemu spracovávanému prvku, aby sa dal prípadne ďalší prvok zanoriť do predchádzajúceho, bolo potrebné transformáciu rozdeliť na 2 časti.

6.7.1 Transformácia v časti XSLT

V prvej časti, ktorú som realizoval na úrovni XSLT transformácie, som pre každý prvok zoznamu v DOCX súbore nechal vygenerovať samostatný jednoprvkový zoznam. Vygenerovaný html kód pre predchádzajúcu ukážku kódu bol nasledovný.

```
<ol hierarchy="0" style=" list-style-type: decimal">
        <li class="ListParagraph-H"><span style=" color:#FF0000; ">Položka</span></li>
</ol>
<ol hierarchy="1" style=" list-style-type: lower-alpha">
        <li class="ListParagraph-H">Položka</li>
</ol>
```

Pri spracovávaní každého odseku zo vstupného DOCX dokumentu sa skontroluje, či sa jedná o odsek patriaci zoznamu a v prípade kladného zistenia sa pri spracovávaní vynechá výstupné obaľovanie odseku do HTML značiek `<p>`. Namiesto nich sa spracovávaný odsek obalí do značiek `<ol>`, pre ktoré sa ešte pridáva jeden doplňujúci atribút, ktorý určí zanorenie v zozname. Toto je potrebné pre druhú časť transformácie zoznamov, ktorá sa už musí realizovať nad DOM štruktúrou.

Parameter zanorenia sa vo výslednom zozname realizuje podobne, ako je to v zdrojovom DOCX dokumente. Pôvodne navrhovaná transformácia pracovala iba s využitím 3 hodnôt pridaného parametra `hierarchy` a to konkrétne: `stay`, `lower`, `higher`. Hodnota `stay` mala vyjadrovať, že prvok záznamu je v rovnakom zozname ako prvok z predchádzajúceho odseku. Hodnota `lower` mala vyjadrovať zanorenie prvku v zozname do novej nižšej úrovne zoznamu. Naopak `higher` vynorenie z nižšej úrovne zoznamu do predchádzajúceho zoznamu vyššej úrovne. Takéto využitie pridaného parametra `hierarchy` však povoľovalo korektné správanie zoznamov iba pri zanáraní alebo vynáraní vždy iba o jednu úroveň. Preto som musel pôvodný návrh upraviť a namiesto pôvodnej 3 stupňovej hierarchie použiť numerickú hodnotu úrovne zoznamu. Takto pripravený sa vstupný zoznam skladá z jednoduchého zoznamu, ktorého prvkami sú HTML zoznamy s pridanou informáciou o plánovanom zanorení.

6.7.2 Transformácia v DOM štruktúre

Druhá časť transformácie zoznamov spočíva v upravení transformovaného HTML kódu. Vo vygenerovanej DOM štruktúre sa vyhľadajú všetky zoznamy a spracovávajú sa postupne. Na vyhľadanie všetkých elementov DOM štruktúry som namiesto sekvenčného prehľadávania použil technológiu XPath, bližšie popísanú v kapitole 6.4.1 XPath, využívanú pri práci s dokumentmi XML. Každý jeden zoznam sa skladá z práve jedného prvku a z pridanej informácie o zanorení. Na základe tohto je možné v následnom spracovávaní vytvárať stromovú štruktúru zoznamu. Oproti pôvodnému návrhu s 3 hodnotovým parametrom `hierarchy` je umožnené pracovať so zanorením o ľubovoľný počet úrovní.

6.7.2.1 Prvok zoznamu na rovnakej úrovni

Pre zoznam, ktorý je označený atribútom rovnakej úrovne ako predchádzajúci prvok, sa v DOM štruktúre nájde tento predchádzajúci zoznam a do množiny jeho prvkov sa pridá prvok z práve spracovávaného zoznamu.

6.7.2.2 Prvok zoznamu na nižšej úrovni

Pre zoznam, ktorý je označený atribútom nižšej úrovne ako predchádzajúci prvok sa v DOM štruktúre nájde tento predchádzajúci zoznam a do jeho množiny prvkov sa pridá potrebný počet zanorených nových zoznamov tak, aby po pridaní aktuálne spracovávaného zoznamu do naposledy pridaného, odpovedalo celkové zanorenie definovanej hodnote v parametri.

6.7.2.3 Prvok zoznamu na vyššej úrovni

Pre zoznam, ktorý je označený atribútom vyššej úrovne ako predchádzajúci prvok sa v DOM štruktúre nájde tento predchádzajúci zoznam a postupne sa v ňom zanára po príslušnú úroveň. Do tejto úrovne sa následne pridá prvok z aktuálne spracovávaného zoznamu. Podstatné pri zanáraní, ktoré začína v rodičovskom zozname je, že zanorenie o úroveň nižšie sa robí v prvku zoznamu, ktorý bol na danej úrovni pridaný ako posledný.

6.8 Implementácia transformácie fontov

Pri transformácií jednotlivých prvkov DOCX dokumentu do príslušných značiek XHTML a odpovedajúcej štruktúry je potrebné sa okrem syntaktických a sémantických vlastností XHTML venovať aj vizuálnej stránke transformovaných častí. Medzi dôležité vizuálne prvky patrí nastavenie vhodného fontu pre CSS štýly transformovaných prvkov. Na rozdiel od textových editorov, ktoré sú pri výbere fontov vo výhode, pretože majú znalosť o všetkých nainštalovaných fontoch v systéme, pri transformácií je potrebné vychádzať z faktu, že XHTML formát textového dokumentu má byť univerzálny a prenositeľný.

Pre každý CSS štýl sa použitý font definuje pomocou parametra `font-face`, prípadne `font-family`. Oba parametre však umožňujú nastavenie hodnoty v podobe zoznamu viacerých názvov fontov. Takto je možné okrem pôvodného názvu fontu, ktorý je definovaný na príslušnom mieste v DOCX dokumente, definovať alternatívy v podobe ďalších fontov, prípadne uviesť rodinu fontov do ktorej patrí.

V CSS štýloch je definovaných 5 rodín fontov, pričom každý bežne používaný font patrí do niektorej z týchto skupín. V nasledujúcom zoznamu sú popísané jednotlivé rodiny fontov aj s príkladmi, ktoré tieto rodiny reprezentujú. [12]

- **sans-serif** – je označenie bezpätkových proporcionálnych fontov, ako príklad uvediem známy Arial, Helvetica alebo v novom Microsoft Office 2007 často používaný Calibri
- **serif** – je označenie pätkových proporcionálnych fontov, typickými zástupcami sú Times New Roman, Georgia alebo z Microsoft Office 2007 Cambria
- **cursiva** – je označenie kurzívnych fontov, ktoré majú napodobovať rukou písané písmo, zástupcami tejto rodiny sú napríklad Caflisch Script, Sanvito

- **fantasy** – je označenie dekoratívnych fontov, teda fonty pre voľne písaný text. Zástupcami tejto rodiny sú Comic Sans, Critter
- **monospace** – označuje rodinu neproporcionálnych fontov, ktoré napodobujú písmo z písacích strojov. Zástupcami tejto rodiny sú Courier, Prestige

Takto je možné v prípade, že font používaný v dokumente nie je na príslušnom operačnom systéme k dispozícii, zachovať aspoň rodinu fontov, do ktorých tento nedostupný font patrí. Vzhľadom na to, že v prostredí jazyka Java neexistuje univerzálny systém na zisťovanie príslušnosti fontu do tej ktorej rodiny fontov, vytvoril som pre potreby transformácie konfiguračný súbor, v ktorom sú tieto príslušnosti definované.

V procese zápisu fontu do CSS štýlu, ktorý je realizovaný v XSLT, sa pre získanie generického fontu, teda názvu rodiny fontu, zavolá statická Java metóda, ktorá dohľadá v konfiguračnom súbore danú rodinu fontov. V prípade, že sa daný font v zozname nenachádza, použije sa dopredu definovaný font určený práve pre tento prípad.

6.9 Transformácia odkazov

V textových dokumentoch je často využívaná možnosť používať odkazy, podobne ako sa využívajú v prípade internetových stránok. V dokumentoch DOCX je možné vytvoriť odkazy na externé dokumenty, na ktoré sa odkazuje prostredníctvom príslušného protokolu. Možné je však vytvárať aj odkazy na miesta v samotnom dokumente. V tomto prípade sa využíva na odkazovanie identifikácia jednotlivých prvkov dokumentu pomocou súboru relácii definovaného v OPC.

Nasledujúca ukážka zobrazuje definovanie odkazu na element v rámci dokumentu. Zobrazený fragment je výberom z elementu odseku. Na ukážke chcem demonštrovať definovanie cieľa odkazu pomocou identifikátora `w:anchor` v elemente `w:hyperlink`.

```
<w:hyperlink w:anchor="_Toc224194034" w:history="1">
<w:r w:rsidR="00644214" w:rsidRPr="00433A36">
        <w:rPr><w:rStyle w:val="Hyperlink" /><w:noProof /></w:rPr>
        <w:t>Odstavce</w:t>
</w:r>
</w:hyperlink>
```

Pri implementácii odkazov, som vychádzal z existujúceho riešenia, ktoré bolo použité v knižnici docx4j, avšak v prostredí jazyka Java bolo nefunkčné. Pre názov linky bolo potrebné získať identifikátor elementu v dokumente a vytvoriť zodpovedajúci XHTML element na vytvorenie odkazu. Na pomenovanie odkazu som použil rovnaký identifikátor ako je použitý v zdrojom DOCX dokumente. Tento bol upravený pre jazyk XHTML takže výsledný odkaz je tvorený elementom `<a>` s parametrom `href` s hodnotou `#identifikátor`.

V nasledujúcej ukáže je zobrazený cieľ odkazu, ktorý je definovaný ako odsek. Vo vlastnostiach odseku je uvedený rovnaký identifikátor ako pri definovaní odkazu. V tomto prípade sa v XHTML vygeneruje element `<a>` ale s parametrom `name` s hodnotou identifikátora.

```
<w:p w:rsidR="00DB4B02" w:rsidRDefault="00DB4B02" w:rsidP="00DB4B02">
        <w:pPr><w:pStyle w:val="Heading2" /></w:pPr>
        <w:bookmarkStart w:id="2" w:name="_Toc224194034" />
…
</w:p>
```

6.10 Transformácia vložených obrázkov

Obrázky sú neoddeliteľnou súčasťou textových dokumentov. Ich transformovanie v rámci súborov DOCX však prináša niektoré problémy, ktoré sa pri iných prvkoch nevyskytujú. Takýmto problémom je, že okrem vytvorenia samotného XHTML elementu `<img>` je potrebné obrázok uložiť ako samostatný súbor.

V adresárovej štruktúre OPC súboru, sú všetky multimediálne objekty uložené v samostatnom adresári */word/media*. V tomto adresári sa okrem obrázkov ukladajú aj ostatné multimediálne objekty, avšak pre transformáciu sú zaujímavé práve obrázkové súbory.

Obrázky sú, tak ako všetky prvky textového dokumentu DOCX, uložené v samostatnom odseku. Vnoreným elementom označujúcim, že v odseku je obrázok je `w:drawing`. V nasledujúcom fragmente uvádzam základný princíp vkladania obrázku do dokumentu.

```
<w:drawing><wp:inline distT="0" distB="0" distL="0" distR="0">
<wp:docPr id="9" name="Picture 8" descr="Garden.jpg" />
<a:graphic xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main">
        <a:graphicData uri="http://schemas.openxmlformats.org/drawingml/2006/picture">
        <pic:pic xmlns:pic="http://schemas.openxmlformats.org/drawingml/2006/picture">
                <pic:nvPicPr>
                        <pic:cNvPr id="0" name="Garden.jpg" /><pic:cNvPicPr />
                </pic:nvPicPr>
                <pic:blipFill>
                        <a:blip r:embed="rId10" cstate="print" />
                </pic:blipFill>
        </pic:pic>
        </a:graphicData>
        </a:graphic>
</wp:inline></w:drawing>
```

V prípade, že sa v obsahu dokumentu nachádzajú obrázky, sú tieto pri samotnej transformácii skopírované do pracovného adresára. Tento adresár je tvorený názvom vstupného DOCX dokumentu, ktorý je zreťazený s výrazom *files*. Takýto pracovný adresár má názov podobný ako je štandardné pri ukladaní webových stránok prostredníctvom prehliadačov.

Podľa hodnoty uloženej v parametri `r:embed` je možné odkázať sa do súboru referencií, kde je pre daný obrázok uložený názov súboru, pod akým je uložený v adresári */word/media*, ktorý zodpovedá obrázku. Cesta, ktorá sa uloží v HTML kóde ako cesta k obrázku sa však generuje v podobe absolútnej URI. Takto vygenerovaný obsah je potom neprenositeľný na iný systém. Potrebné je teda na základe absolútnej cesty obrázku a absolútnej cesty pracovného adresára, do ktorého sa tieto obrázky kopírujú, použiť v HTML parametri SRC iba relatívnu cestu.

Pri exportovaní obrázkov je značnou výhodou, že táto funkcionalita je už obsiahnutá v knižnici docx4j, takže samotné fyzické kopírovanie súborov a získavanie potrebných parametrov z príslušných XML súborov je súčasťou pripravenej triedy na spracovávanie grafických prvkov.

7 Záver

Cieľom diplomovej práce bolo naštudovať potrebné technológie a na ich základe vytvoriť knižnicu v programovacom jazyku Java. Táto zo vstupného dokumentu vo formáte Office Open XML vytvorí prostredníctvom transformácie DOM štruktúru tohto dokumentu vo formáte XHTML, ktorý potom umožní transformovať do príslušného fyzického súboru.

V úvodných kapitolách mojej práce som sa venoval popisu kľúčových technológií, na ktorých je postavená analýza a návrh transformačnej knižnice. V prvej, z týchto kapitol, sa venujem základnému členeniu Office Open XML dokumentov a ich štruktúre. V tejto kapitole sa však ešte nevenujem presnej analýze dokumentov. V ďalších 2 kapitolách som postupne popísal princíp objektového modelu dokumentov DOM a základy značkovacieho jazyka XHTML. V popise DOM som uviedol jeho základné operácie určené na manipuláciu, ktoré sú neskôr využívané pri implementácii knižnice. Pri popise značkovacieho jazyka XHTML som sa zameral iba na niektoré základné zobrazovacie prvky ako sú napríklad odseky, nadpisy a zoznamy.

V časti analýzy rozoberám základné vlastnosti a štruktúru Office Open XML dokumentov, pričom sa sústreďujem výhradne na podmnožinu textových dokumentov, ktoré sú popisované jazykom WordprocessingML. Tieto dokumenty sú svojím určením podobné ako webové stránky. Z tohto dôvodu je aj knižnica na transformáciu zameraná práve na ne. Popísané sú tiež 2 možné spôsoby, ako realizovať samotnú transformáciu dokumentov a ich vzájomné porovnanie.

Po analýze DOCX dokumentov, čo sú textové dokumenty Office Open XML, som vybral možnosť realizovať implementáciu knižnice založenú na XSLT transformácii. V kapitole venovanej implementácii som popísal postup realizácie procesu transformácie, výber vhodných knižníc na prácu so vstupným súborom a technológie, ktoré boli použité pri riešení jednotlivých častí procesu.

7.1 Využitie externých zdrojov

Vo fáze analýzy, kde som sa začal zaoberať možnosťami ako transformovať DOCX dokumenty, som sa stretol s viacerými hotovými riešeniami exportovania vstupného textového dokumentu vo formáte Office Open XML do XHTML. Tieto boli realizované v jazyku C# a určené pre operačný systém Microsoft Windows, či už v podobe samostatných skriptov pre príkazový riadok, alebo v podobe prídavnej funkcionality prehliadača. Spoločným prvkom týchto transformácii je použitie rovnakého XSLT súboru na transformáciu, ktorý je voľne šíriteľný a pochádza z produkcie spoločnosti Microsoft.

V samotnej časti implementácie som použil práve tento dostupný XSLT súbor a dynamicky sa vyvíjajúcu knižnicu docx4j, ktorá je primárne určená na editovanie a vytváranie DOCX dokumentov. V tejto knižnici je pripravená podpora na exportovanie dokumentov do XHTML,

realizovaná prostredníctvom XSLT transformácie. Z dôvodu vývoja tejto knižnice bolo však exportovanie zo začiatku nestabilné a s mnohými chybami počas behu programu. Moja práca bola teda spočiatku zameraná primárne na preštudovanie fungovania knižnice a až následne bolo možné túto knižnicu plnohodnotne využiť pre potreby transformácie. Podstatnou nevýhodou pôvodného XSLT súboru bolo zaniknutie sémantického významu niektorých značiek jazyka XHTML vo vygenerovanom dokumente. Z tohto dôvodu bolo potrebné rozšíriť možnosti pôvodného XSLT súboru a vytvoriť aj príslušné obslužné metódy v jazyku Java, pomocou ktorých sa vygenerovaný dokument viac podobá dokumentom vytvoreným ako webové stránky. Táto sémantika jednotlivých značiek je dôležitá napríklad pri prípadnej analýze textu v dokumentoch.

7.2 Námety na ďalší vývoj

Použitie rovnakého XSLT súboru, ktorý sa používa vo viacerých transformačných aplikáciách, aj keď určených pre inú vývojovú platformu, so sebou prináša možnosť ďalšieho kooperatívneho vývoja transformácie, v dolaďovaní a prípadnom rozširovaní funkcionality. Taktiež skutočnosť, že knižnica docx4j, ktorá je pre knižnicu transformácie kľúčová, sa využíva iba ako samostatná programová časť, ktorá nebola nijak upravovaná, so sebou prináša možnosť sledovať ďalší vývoj tohto sľubného produktu a využiť jeho novšie verzie v implementovanej knižnici.

Slovník pojmov

Pojem	Vysvetlenie pojmu
Nightly build	Označenie verzie programu, ktorý je stále v procese vývoja, ale už je v spustiteľnej podobe.
Parser, DOM parser, SAX parser	Spôsoby spracovávania XML dokumentov. Popísané sú v kapitole 6.4.4 Spracovávanie XML dokumentov v Jave
Java Runtime, Java Runtime Environment (JRE)	Prostredie, ktoré sa skladá z virtuálneho stroja, v ktorom sú spúšťané Java programy a Java API
Classpath	Množina knižníc (referencie na uložené knižnice), ktoré sú potrebné pre spustenie Java aplikácie
JAR súbor	Aplikácia v programovacom jazyku Java v podobe knižnice. Jedná sa o archív vo formáte ZIP, v ktorom sú uložené skompilované triedy a potrebné prídavné vlastnosti
DOCX dokument	Súbor reprezentujúci textový dokument v Office Open XML
Framework, Softvérový framework	Softvérový framework je množina programových kódov alebo knižníc, ktoré tvoria softvérovú podporu na riešenie celej triedy aplikácií. Nejedná sa o knižnicu, ktorá poskytuje len jednu sadu funkcionality.
Maršalovanie, demaršalovanie	Procesy podobné serializácii, ktoré slúžia na využívanie JAXB modelu pre zdrojový XML dokument.

Literatúra

[1] White, Eric. The Flat OPC Format . [Online] 29. 9 2008. [cit. 7. 4 2009.] http://blogs.msdn.com/ericwhite/archive/2008/09/29/the-flat-opc-format.aspx.

[2] Tidwell, Doug. *XSLT.* místo neznámé : O'Reilly, 2001. ISBN 10: 0-596-00053-7.

[3] Rábová, Zdena, a další. Užitečné rady pro psaní odborného textu. [Online] +. 12 2008. [cit. 22. 5 2009.] http://www.fit.vutbr.cz/info/statnice/psani_textu.html.cs.

[4] Olsen, Russ. Reflections on Java Reflection. *O'Reilly Media.* [Online] 15. 3 2007. [cit. 7. 4 2009.] http://www.onjava.com/pub/a/onjava/2007/03/15/reflections-on-java-reflection.html.

[5] Nic, Miloslav a Jirat Jiri. XPath Tutorial. *ZVON.* [Online] [cit. 7. 4 2009.] http://zvon.org/xxl/XPathTutorial/General/examples.html.

[6] Jeka, Jürgen. DOM Levels. [Online] 13. 5 2009. [cit. 7. 4 2009.] https://developer.mozilla.org/en/DOM_Levels.

[7] Hruška, Tomáš, Kroulík, Jan a Techet, Jiří. Internetové aplikace (WAP) III. část XML, XML schémata, XPath, XSLT. [Online] 2 2007. [cit. 7. 4 2009.] https://www.fit.vutbr.cz/study/courses/WAP/private/opory/OporaWAP3XMLXPathXQueryXSLT.pdf.

[8] Hrazdil, Jiří. XHTML™ 1.0: Rozšířitelný hypertextový značkovací jazyk. [Online] 1. 8 2002. [cit. 20. 12 2009.] http://www.zralog.cz/translate/TR/REC-xhtml1-20020801/Overview.php.

[9] Hors, Le Arnaud, a další. Document Object Model (DOM) Level 2 Core Specification Version 1.0. *W3C Recommendation.* [Online] 13. 11 2000. [cit. 20. 12 2009.] http://www.w3.org/TR/DOM-Level-2-Core/.

[10] Ehrli, Erika. Walkthrough: Word 2007 XML Format. [Online] 6 2006. [cit. 20. 12 2009.] http://msdn.microsoft.com/en-us/library/bb266220.aspx.

[11] Clark, James a DeRose, Steve. XML Path Language (XPath) Version 1.0. *W3C Recommendation.* [Online] 16. 11 1999. [cit. 7. 4 2009.] http://www.w3.org/TR/xpath.

[12] Bos, Bert. Web Style Sheets CSS tips & tricks. *W3C.* [Online] 15. 5 2009. [cit. 20. 5 2009.] http://www.w3.org/Style/Examples/007/fonts#font-family.

[13] The Apache Software Foundation. *POI-OpenXML4J - Java API To Access Office Open XML documents.* [Online] [cit. 7. 4 2009.] http://poi.apache.org/oxml4j/.

[14] International Standards for Business, Government and Society. Freely Available Standards. [Online] 14. 05 2009. [cit. 20. 12 2008.] http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html.

[15] CSS (10.) - Písmo V. (Rodina písma - úvod). [Online] 27. 9 2004. [cit. 7. 4 2009.] http://www.linuxsoft.cz/article.php?id_article=418.

[16] Ort, Ed a Mehta, Bhakti. Java Architecture for XML Binding (JAXB). *Sun Developer Network.* [Online] č 2003. [cit. 7. 4 2009.] http://java.sun.com/developer/technicalArticles/WebServices/jaxb/.

[17] Kawaguchi, Kohsuke. The JAXB API. [Online] 8. 1 2003. [CIT. 7. 4 2009.] http://www.xml.com/pub/a/2003/01/08/jaxb-api.html.

Zoznam obrázkov

Obrázok 1 – Základná štruktúra DOM Level 2 Core 10
Obrázok 2 – Základná štruktúra priamej transformácie 16
Obrázok 3 – Základná štruktúra transformácie s použitím XSLT 17
Obrázok 4 - Informatívny pohľad na transformáciu z Office Open XML do XHTML 19
Obrázok 5 – Ilustračná schéma DOM štruktúry dokumentu tvoreného odsekmi 25
Obrázok 6 – Schéma transformácie XML dokumentov prostredníctvom XSLT 34
Obrázok 7 – Orientačná schéma JAXB [16] 35
Obrázok 8 – Ilustračná schéma procesu transformácie 37

Zoznam tabuliek

Tabuľka 1 Prehľad možností transformácie 18
Tabuľka 2 Porovnanie výhod a nevýhod jednotlivých knižníc 31

Zoznam príloh

Príloha 1. Používateľská príručka
Príloha 2. Obsah dátového média
Príloha 3. Ukážka testového dokumentu vo formáte DOCX
Príloha 4. Ukážka vygenerovaného HTML dokumentu
Príloha 5. Dátové médium - CD

Používateľská príručka

Na spustenie knižnice je potrebné mať na cieľovom počítači nainštalovaný Java Runtime Environment. Z používateľského hľadiska je možné knižnicu využívať dvomi spôsobmi, ako samostatnú aplikáciu spúšťanú priamo z príkazového riadku alebo ako súčasť inej Java aplikácie.

Knižnica transformácie ako samostatná aplikácia

Spustenie transformácie ako samostatnej aplikácie je potrebné realizovať prostredníctvom volania spúšťacej sekvencie Java aplikácií pre daný JRE. V operačných systémoch Microsoft Windows pri štandardnom nainštalovaní prostredia Java je možné transformáciu spustiť nasledovne.

```
java –jar transformation.jar [–f] docxFilename
```

Presná definícia spustenia knižnice je nasledovná.

```
OpenXML document processing library
Usage:
        transformation.jar [-f] docxFilename
Options:
        -f              optional, use for saving generated flatOPC document to flatopc.xml file
        docxFilename    required, source file name
Examples:
        java -jar transformation.jar test.docx
        java -jar transformation.jar -f test.docx
```

Použitie knižnice transformácie ako súčasť inej Java aplikácie

Pre použitie knižnice v inej aplikácii je potrebné pridať knižnicu *transformation.jar* do classpath spúšťanej aplikácie. Programové využite knižnice je už ďalej závislé na vývojárovi. Knižnica poskytuje nasledovné rozhranie na svoje spustenie.

```
public DOMTransformation(boolean crateFlatOPC, File sourceFile, File resultFile, File resultFolder) { ... }

public void transform() throws TransformationNotPerformedException  { … }


public Node getTransformedDOM() throws TransformationNotPerformedException,
TransformationFailuredException  { ... }
```

Fragment programovacieho kódu v ktorom je ukázané volanie knižnice spolu s ošetrením výnimiek, ktoré môžu vzniknúť pri neúspešnej transformácii vstupného dokumentu.

```
try {
        DOMTransformation t = new DOMTransformation(true, new File("sourceDocxFile.docx"),
                                        new File("outputFile.html"), new File("outputFile-files"));
        t.transform();
        t.getTransformedDOM();
} catch (TransformationNotPerformedException e) {
        e.printStackTrace();
} catch (TransformationFailuredException e) {
        e.printStackTrace();
}
```

Obsah dátového média

Názov súboru	Popis súboru
./readme	Úvodný súbor obsahu dátového média
./abstrakt	Abstrakt práce v českom jazyku
./abstract	Abstrakt práce v anglickom jazyku
./diplomova_praca.doc	Dokument diplomovej práce v zdrojovom formáte
./diplomova_praca.pdf	Dokument diplomovej práce
./pouzivatelska_prirucka.pdf	Používateľská príručka
./transformation.jar	Spustiteľná verzia knižnice
./transformation.zip	Exportovaný projekt so zdrojovým kódom
./test1.docx	Testovací dokument 1
./test2.docx	Testovací dokument 2

Testovací dokument

Table of Contents

Testovací dokument 1
Rôzne typy zoznamov 1
Odstavce 1
Tabuľky 2

Rôzne typy zoznamov

1) Položka1
2) Položka2
    a) Položka3
        i) Položka4
        ii) Položka5
            ✓ Položka6
            ✓ Položka7
        iii) 8
    b) Položka9
        i) Položka10
        ii) Položka11
    c) 12
3) Položka13
    a) Položka14
    b) Položka15
    c) 16
4) Položka17

Odstavce

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla lacinia erat ac arcu. Nulla vestibulum urna suscipit diam. **Suspendisse** potenti. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque scelerisque facilisis neque. *Aliquam* consectetur eros. Morbi accumsan pretium dui. In ipsum. Aliquam sagittis leo nec sapien. Vivamus sollicitudin facilisis velit.

Etiam egestas iaculis sapien. Donec iaculis aliquam tellus. Aliquam imperdiet. Aenean erat metus, sollicitudin et, malesuada et, feugiat vitae, nibh. ~~Vivamus~~ eros ipsum, rutrum a, commodo nec, lobortis at, $_{neque}$. Sed ^{lacinia}. Donec ullamcorper mauris vitae augue. Donec nec risus sit amet nisl cursus

ultrices. ***Vivamus euismod magna.*** Mauris feugiat orci eu nulla. Proin pretium lorem vitae lectus.

Nulla vel massa vitae ipsum cursus rutrum. In semper luctus diam. Vestibulum malesuada. Aliquam velit arcu, bibendum ac, tempor id, volutpat vitae, neque. Etiam id metus. Morbi facilisis hendrerit purus. Nam sagittis aliquam nisl. Mauris vel urna eget lectus dignissim tincidunt. Fusce vel sem nec felis suscipit vulputate. Praesent eget nibh eget risus facilisis mattis.

Mauris iaculis pulvinar lorem. Suspendisse orci nulla, lobortis eu, consectetur eu, volutpat ut, felis. Cras pellentesque lorem. Vivamus consequat porttitor massa. Sed fringilla tempus nisi. Maecenas non nulla. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nullam vitae turpis at felis viverra aliquam. Nunc venenatis dolor. Sed diam nunc, volutpat eu, eleifend eget, accumsan mattis, magna.

- Fusce varius purus vitae massa.
  a) Integer dignissim tincidunt ligula.
  b) Cras quis augue ut nulla rutrum sagittis.
     i) Aliquam vitae dui id urna hendrerit iaculis.
     ii) Morbi tincidunt varius nunc.
- Mauris eu leo nec mauris eleifend rutrum.
- Sed nec massa non est posuere rutrum.
  a) aaa

Pellentesque odio. Cras scelerisque massa nec ante. Vestibulum nulla purus, scelerisque in, vestibulum quis, dignissim id, libero. Pellentesque posuere neque ut nunc. Vestibulum luctus venenatis nibh. Ut posuere sem et leo.

> Praesent ac purus. Curabitur magna. Integer ac mi. Ut non libero. Quisque interdum ultrices tellus. Pellentesque consectetur, arcu vitae ullamcorper adipiscing, libero libero lacinia tellus, non feugiat nunc purus ut quam. Suspendisse potenti. Curabitur eget justo quis mi lobortis vehicula. Integer tincidunt, lacus ac vehicula imperdiet, nibh ante sagittis dui, in gravida neque nisi ac justo. Donec dapibus ultricies erat. Nunc elit. Nulla facilisi. Aliquam in eros ut orci luctus tincidunt. Donec quis leo non nulla vestibulum pulvinar. Vestibulum sed risus. Suspendisse potenti. Maecenas cursus lacinia diam.

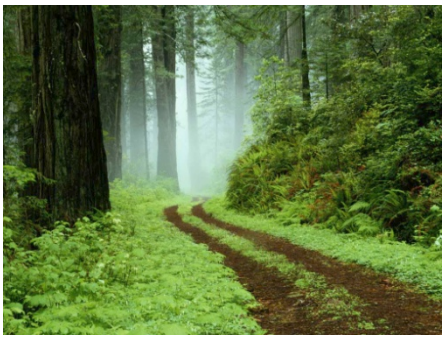
Tabuľky

SPOJENÉ BUNKY TABUĽKY

RIADOK 1	H+	H+	H+	H+	H+
RIADOK 2					

Vložené obrázky

Obrázok 1



Obrázok 2

Obrázok 3

Hlavicka – h3

Nulla pretium dignissim lectus. Sed vel velit et est vestibulum bibendum. Nullam massa erat, semper a, molestie quis, faucibus eu, sapien. Sed lacinia, tellus id feugiat consectetur, massa odio laoreet purus, ac varius leo tortor a leo. In magna.

Hlavicka – h4

Curabitur ultricies. Morbi leo tellus, volutpat et, accumsan eu, ornare sed, est. Donec tincidunt orci ac justo. Praesent arcu. Morbi lobortis commodo urna. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Suspendisse potenti.

Externý Obrázok

Obrázok bol vložený prostredníctvom vloženia cesty
http://www.fit.vutbr.cz/common/img/fit_logo_en.gif v dialógovom okne na pridanie obrázka.

Testovací dokument

Table of Contents

Testovací dokument 1

Rôzne typy zoznamov 1

Odstavce 1

Tabuľky 2

Rôzne typy zoznamov

1. Položka1
2. Položka2
    a. Položka3
        i. Položka4
        ii. Položka5
            - Položka6
            - Položka7
        iii. 8
    b. Položka9
        i. Položka10
        ii. Položka11
    c. 12
3. Položka13
    a. Položka14
    b. Položka15
    c. 16
4. Položka17

Odstavce

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla lacinia erat ac arcu. Nulla vestibulum urna suscipit diam. **Suspendisse** potenti. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Quisque scelerisque facilisis neque.

Aliquam consectetur eros. Morbi accumsan pretium dui. In ipsum. Aliquam sagittis leo nec sapien. Vivamus sollicitudin facilisis velit.

Etiam egestas iaculis sapien. Donec iaculis aliquam tellus. Aliquam imperdiet. Aenean erat metus, sollicitudin et, malesuada et, feugiat vitae, nibh. Vivamus eros ipsum, rutrum a, commodo nec, lobortis at, neque. Sed lacinia. Donec ullamcorper mauris vitae augue. Donec nec risus sit amet nisl cursus ultrices. ***Vivamus euismod magna.*** Mauris feugiat orci eu nulla. Proin pretium lorem vitae lectus.

Nulla vel massa vitae ipsum cursus rutrum. In semper luctus diam. Vestibulum malesuada. Aliquam velit arcu, bibendum ac, tempor id, volutpat vitae, neque. Etiam id metus. Morbi facilisis hendrerit

purus. Nam sagittis aliquam nisl. Mauris vel urna eget lectus dignissim tincidunt. Fusce vel sem nec felis suscipit vulputate. Praesent eget nibh eget risus facilisis mattis. Mauris iaculis pulvinar lorem. Suspendisse orci nulla, lobortis eu, consectetur eu, volutpat ut, felis. Cras pellentesque lorem. Vivamus consequat porttitor massa. Sed fringilla tempus nisi. Maecenas non nulla. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nullam vitae turpis at felis viverra aliquam. Nunc venenatis dolor. Sed diam nunc, volutpat eu, eleifend eget, accumsan mattis, magna.

- Fusce varius purus vitae massa.
    a. Integer dignissim tincidunt ligula.
    b. Cras quis augue ut nulla rutrum sagittis.
        i. Aliquam vitae dui id urna hendrerit iaculis.
        ii. Morbi tincidunt varius nunc.
- Mauris eu leo nec mauris eleifend rutrum.
- Sed nec massa non est posuere rutrum.
    a. aaa

Pellentesque odio. Cras scelerisque massa nec ante. Vestibulum nulla purus, scelerisque in, vestibulum quis, dignissim id, libero. Pellentesque posuere neque ut nunc. Vestibulum luctus venenatis nibh. Ut posuere sem et leo.

> Praesent ac purus. Curabitur magna. Integer ac mi. Ut non libero. Quisque interdum ultrices tellus. Pellentesque consectetur, arcu vitae ullamcorper adipiscing, libero libero lacinia tellus, non feugiat nunc purus ut quam. Suspendisse potenti. Curabitur eget justo quis mi lobortis vehicula. Integer tincidunt, lacus ac vehicula imperdiet, nibh ante sagittis dui, in gravida neque nisi ac justo. Donec dapibus ultricies erat. Nunc elit. Nulla facilisi. Aliquam in eros ut orci luctus tincidunt. Donec quis leo non nulla vestibulum pulvinar. Vestibulum sed risus. Suspendisse potenti. Maecenas cursus lacinia diam.

Tabuľky

SPOJENÉ BUNKY TABUĽKY

RIADOK 1	H+	H+	H+	H+	H+
RIADOK 2					

Vložené obrázky

Obrázok 1

Obrázok 2

Obrázok 3

Hlavicka – h3

Nulla pretium dignissim lectus. Sed vel velit et est vestibulum bibendum. Nullam massa erat, semper a, molestie quis, faucibus eu, sapien. Sed lacinia, tellus id feugiat consectetur, massa odio laoreet purus, ac varius leo tortor a leo. In magna.

Hlavicka – h4

Curabitur ultricies. Morbi leo tellus, volutpat et, accumsan eu, ornare sed, est. Donec tincidunt orci ac justo. Praesent arcu. Morbi lobortis commodo urna. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Suspendisse potenti.

Externý Obrázok

Obrázok bol vložený prostredníctvom vloženia cesty http://www.fit.vutbr.cz/common/img/fit_logo_en.gif v dialógovom okne na pridanie obrázka.