Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Konverzia DOCX na XHTML

diplomová práca

Bc. František Šmitala

Odbor: Informatika 9.2.1

Vedúci: RNDr. Richard Ostertág

Bratislava, 2009

Abstrakt

ŠMITALA, František: Konverzia DOCX na XHTML. [diplomová práca] - Fakulta matematiky, fyziky a informatiky Univerzity Komenského v Bratislave; Katedra informatiky. - Školiteľ: RNDr. Richard Ostertág - Bratislava 2009. 79 strán.

Diplomová práca sa zaoberá problematikou Office Open XML formátu. Má dve časti. V prvej teoretickej časti podáva popis formátu Office Open XML s dôrazom na DOCX dokumenty a v druhej praktickej časti sa zaoberá návrhom a implementáciou aplikácie na konverziu dokumentu vo formáte DOCX na dokument vo formáte XHTML. K diplomovej práci je priložená výsledná aplikácia na konverziu DOCX na XHTML.

Kľúčové slová: Office Open XML, konverzia formátov dokumentov, DOCX

Predhovor

Problém konverzie formátov dokumentov je tu v podstate od čias, keď tradičné papierové dokumenty boli nahradené dokumentami elektronickými. Rôzne formáty dokumentov môžu mať rôzne použitie, a preto je konverzia medzi nimi niekedy nevyhnutná. Typickým príkladom je publikovanie dokumentov na internete. Materiály, ktoré chceme publikovať sú len veľmi zriedka vo formáte na to určenom a to vo formáte XHTML. V súčasnosti najrozšírenejším nástrojom na prácu s dokumentami je kancelársky balík Microsoft Office. Jeho najnovšia verzia prišla s novým formátom na ukladanie dokumentov Office Open XML založeným na značkovacom jazyku XML. Konkrétne pre formátovaný text sa tento formát nazýva WordprocessingML a súbor v tomto formáte má príponu DOCX.

Táto diplomová práca je o konverzii tohto formátu na formát XHTML. Zahŕňa prehľad formátu Office Open XML, prehľad existujúcich riešení a nakoniec implementáciu vlastného riešenia. Naprogramovaná aplikácia má ambíciu prekonať existujúce riešenia hlavne v kvalite zdrojového kódu skonvertovaného dokumentu. Zároveň chce zachovať kvalitu prenosu formátovania z DOCX dokumentu na XHTML dokument aspoň na takej úrovni, ako konkurenčné riešenia.

Aplikácie je dostupná pre každého a je naprogramovaná tak, aby sa dali ľahko zapracovávať zmeny a vylepšenia. Výberom programovacieho jazyka Java sa zabezpečila použiteľnosť programu na akejkoľvek platforme.

Ďakujem svojmu vedúcemu diplomovej práce, RNDr. Richardovi Ostertágovi za cenné rady a pripomienky pri písaní tejto práce. Zároveň by som chcel poďakovať mojej priateľke a rodine za podporu.

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím citovaných zdrojov.

. . . . . .

Obsah

Kapitola 1

Úvod

Táto práca je určená hlavne pre začínajúcich vývojárov aplikácií na prácu s Office Open XML formátom. Aj keď sa prepokladá aspoň základná znalosť problematiky XML a XHTML, v prvej časti tejto kapitoly si vysvetlíme základné kľučové pojmy použité v tejto práci. V ďalšej časti tejto kapitoly prejdeme k definovaniu cieľa spolu s motiváciou. Na záver si ukážeme členenie tejto práce.

1.1 Základné pojmy

V tejto časti si zadefinujeme základné pojmy, ktoré sú potrebné na pochopenie tejto diplomovej práce. Pojmy sú vysvetľované nie všeobecne, ale v kontexte tejto práce.

CSS (Kaskádové štýly) Je to jazyk na popis spôsobu zobrazenia XHTML dokumentov. Umožňuje oddeliť vzhľad dokumentu od jeho štruktúry a obsahu.

DTD (Document Type Definition) Ide o jazyk určený na popis štruktúry XHTML dokumentu. Definuje, ktoré elementy môže dokument obsahovať, aké atribúty obsahujú jednotlivé elementy a podobne.

ECMA Je to medzinárodná nezisková štandardizačná organizácia pre informačné a komunikačné systémy.

HTML Ide o značkovací jazyk určený hlavne na reprezenáciu web stránok. Popisuje štruktúru textovej informácie tým, že v nej umožňuje označiť nadpisy, odstavce, zoznamy, odkazy, vložiť obrázky a podobne.

ISO Je to medzinárodná organizácia pre štandardizáciu zložená z predstaviteľov z rôznych národných štandardizačných organizácií. Zaoberá sa tvorbou medzinárodných noriem vo všetkých oblastiach.

menný priestor (namespace) V XML menné priestory zoskupujú elementy a atribúty za účelom ich jedinečnosti a jednoznačnosti v rámci celého XML. Definované sú pomocou URI a príslušnosť elementu resp. atribútu do menného priestoru býva označená prefixom. Napríklad element „w:b“ má prefix „w“. To, že ktorý menný priestor tento prefix označuje sa definuje atribútom xmlns:w.

URI Ide o reťazec znakov s definovanou štruktúrou používaný na identifikáciu zdroja informácií. Príkladom URI je napríklad URL adresa, alebo „urn:issn:1535-3613“, ...

W3C Je to medzinárodné konzorcium produkujúce štandardy pre World Wide Web.

XHTML (Extensible Hypertext Markup Language) Je to v podstate HTML založené na XML. Má prísnejšiu štruktúru ako HTML a vďaka tomu je automatizované spracovanie oveľa jednoduchšie a rýchlejšie. Všeobecne je považovaný za nástupcu HTML a postupne ho vytláča.

XML (Extensible Markup Language) Rozšíriteľný značkovací jazyk na tvorbu konkrétnych značkovacích jazykov pre rôzne účely (výmena dát, publikovanie dokumentov, ...).

XSLT (Extensible Stylesheet Language Transformations) Je to jazyk založený na XML určený na transformáciu XML dokumentov na iné XML dokumenty alebo akýkoľvek iný formát.

1.2 Motivácia

Vydanie nového formátu Office Open XML otvára hlavne vývojárom nové možnosti, keďže spoločnosť Microsoft zverejnila k tomuto formátu kompletnú dokumentáciu na voľné stiahnutie. Problémom však je fakt, že táto špecifikácia má viac ako 6 000 strán a je teda pre bežného vývojára nepoužiteľná. Existuje množstvo článkov, prehľadov a tutoriálov špecializovaných na tento formát a teda je si z čoho vyberať. Väčšinou sú to však materiály, ktoré buď poskytujú naozaj len základné informácie, alebo sú až príliš detailné a špecializované

na jednu oblasť. Neexistuje teda popis tohto formátu, ktorý by bol dostatočne obsiahly, aby bolo na jeho základe možné pracovať s dokumentami so štandardným obsahom, ale zároveň aby nebol príliš rozsiahly a nezaoberal sa zbytočnými vecami.

V súvislosti s formátmi dokumentov sa vyskytuje ešte jeden častý problém. Je ním konverzia medzi jednotlivými formátmi dokumentov. Na dokumenty publikované na internete existuje v podstate iba jeden rozumný formát a to HTML resp. XHTML. Tento formát je priamo podporovaný prehliadačmi. Aj keď existuje veľké množstvo konvertorov, všetky fungujú na princípe doslovného[1] prekladu. Problémom je tiež, že nie všetky DOCX štruktúry majú XHTML ekvivalent. Výsledkom konverzie tak je XHTML dokument, ktorý je síce vzľadom veľmi podobný na svoj DOCX originál, avšak za cenu objemného a neprehľadného zdrojového kódu.

Pri XHTML dokumentoch nám však ide o viac než len o to ako vyzerá. Tradičným problémom je rozdielne zobrazovanie toho istého dokumentu v rôznych prehliadačoch. Preto je vhodné pri vytváraní XHTML dokumentov dodržiavať štandardy a nerobiť ich zbytočne zložité (čo sa zdrojového kódu týka). Kvôli rýchlosti načítania a vyrenderovania obsahu je tiež dôležitá veľkosť dokumentu. Posledným aspektom v súvislosti s objemným a neprehľadným zdrojovým kôdom XHTML dokumentu je možnosť úprav a prispôsobovania. Predstavme si situáciu, keď na webovskú stránku s profesionálnym dizajnom potrebujeme pridať štruktúrovaný DOCX dokument. Máme na výber dve možnosti. Ručne skopírujeme iba text a postupne ho štruktúrujeme použitím XHTML tagov. Druhá možnosť je, že si dokument skonvertujeme a v jeho neprehľadnom zdrojovom kóde dokument pracne napojíme na naše existujúce CSS štýly.

Ukázali sme si teda dva problémy v súvislosti s DOCX formátom. A práve neexistujúca dokumentácia, ktorá by bola prehľadná, stručná a zároveň dostatočne silná a neexistujúci konvertovací nástroj z formátu Office Open XML na XHTML, v ktorom by kvalita zdrojového kódu bola dôležitejšia ako kvalita presnosti prenesenia vzhľadu, je dôvodom na vznik tejto diplomovej práce.

1.3 Cieľ práce

Táto diplomová práca má tri hlavné ciele. Sú to:

[1] Doslovný vo význame, že každý element v DOCX má nejaký ekvivalent v XHTML

- prehľad existujúcich riešení na konverziu DOCX na XHTML
- stručná dokumentácia k Office Open XML
- vytvorenie aplikácie na konverziu DOCX na XHTML

Primárnym cieľom je vytvorenie aplikácie na konverziu DOCX dokumentu na XHTML dokument. Pri tomto cieli sledujeme hlavne zachovanie štruktúry dokumentu, schopnosť vedieť pracovať so všetkými štandardnými elementami v bežných dokumentoch a možnosť voľby úrovne zachovania formátovania. Je zrejmé, že čím bude prenos formátovania presnejší, tým neprehľadnejší bude zdrojový kód XHTML výstupu. Ďalej chceme, aby bol výstup valídny XHTML dokument so Strict DTD. Tiež od výsledného programu očakávame primeranú rýchlosť a nenáročnosť na výpočtovú zložitosť. Nakoniec chceme, aby bol tento program riešený konzolovou aplikáciou. Samozrejme je potrebné vytvoriť aj grafické používateľské prostredie, ktoré na základe nastavení konverzie spustí konzolovú aplikáciu s patričnými parametrami. Týmto uľahčíme použitie programu aj iným aplikáciam.

Sekundárnym cieľom tejto diplomovej práce je vytvorenie stručnej a prehľadnej dokumentácie k formátu DOCX. Táto dokumentácia by mala obsahovať dostatok informácii na to, aby bol čitateľ schopný pracovať so štandardnými DOCX dokumentami. Zároveň chceme, aby v prípade zložitejšej konštrukcie dokumentu bol čitateľ schopný sám hľadať informácie z dostupnej oficiálnej dokumentácie.

1.4 Rozvrhnutie práce

Táto práca obsahuje dve logické časti. Prvá je teoretická a druhá praktická. Formálne sa delí na 5 kapitol.

Prvá kapitola je tento úvod. Sú v nej zadefinované základné pojmy, ktoré sú nevyhnutné na správne pochopenie problematiky. Tiež tu je vytýčený cieľ práce a motivácia k tejto práci.

V druhej kapitole je vypracovaná teoretická časť práce. Zahŕňa prehľad Office Open XML a hlavne WordprocessingML formátu na takej úrovni, aby bol čitateľ schopný na základe týchto informácii vytvárať alebo spracovávať bežné dokumenty v tomto formáte.

V tretej kapitole sú opísané všetky existujúce riešenia konverzie DOCX dokumentu na XHTML dokument.

Štvrtá kapitola obsahuje informácie o aplikácii, ktorá tvorí jadro tejto práce. Začneme špecifikáciou programu, potom prejdeme k návrhu a nakoniec k samotnej implementácii.

V piatej kapitole budeme program testovať. Porovnáme výsledky s existujúcimi riešeniami a zhodnotíme celkový prínos.

Kapitola 2

Office Open XML

Office Open XML (tiež známy ako OOXML, OpenXML alebo Open XML) je súborový formát reprezentujúci textové dokumenty, prezentácie a tabuľky. Súčasťou týchto dokumentov môžu byť tiež grafy, či iné grafické elementy alebo matematické vzorce. Formát vyvinula firma Microsoft a používa ho ako východzí súborový formát vo svojom kancelárskom balíku Microsoft Office 2007. Ide vlastne o niekoľko XML súborov a iných potrebných súborov vzájomne poprepájaných definovanými prepojeniami a spoločne zabalených v jednom ZIP archíve[1].

Tento formát prichádza v čase, keď už jeden štandardizovaný formát na ukladanie dokumentov existuje, a to OpenDocument (skrátené z OASIS Open Document Format for Office Applications). Existujú aj mnohé iné dôvody, pre ktoré je tento krok firmy Microsoft kritizovaný, ale porovnávanie týchto dvoch formátov je nad rámec tejto diplomovej práce. Faktom zostáva, že kancelárske balíky od firmy Microsoft majú dominantné postavenie na trhu a pôvodné binárne formáty používané v týchto balíkoch nemali verejne dostupnú špecifikáciu a teda akýkoľvek externý prístup k týmto súborom bol značne problematický. Aj keď neskôr Microsoft špecifikáciu k týmto binárnym formátom zverejnil, XML je predsa len pre vývojárov vhodnejší a prehľadnejší formát. Formáty súborov založené na formáte XML umožňujú vývojárom pristupovať ku konkrétnemu obsahu v rámci súborov bez toho, aby museli analyzovať celé dokumenty. Povedzme, že chceme vyextrahovať všetky obrázky z dokumentu. V prípade binárnych formátov musíme analyzovať celý dokument, v prípade XML dokumentu prečítame patričné prepojenia, kde nájdeme umiestnenie jednotlivých obrázkov.

[1] Porovnaj s [wik09]

Súčasťou kancelárskeho balíku Microsoft Office 2007 sú programy Word na vytváranie dokumentov s formátovaným textom, tabuľkový kalkulátor Excel a program PowerPoint, ktorý slúži na vytváranie prezentácii. Štandardne sú jednotlivé dokumenty odlíšené príponami. Dokumenty s formátovaným textom majú príponu „.docx“, tabuľky „.xlsx“ a nakoniec prezentácie „.pptx“. Preto pod pojmom „Konverzia DOCX na XHTML“ budeme v tejto práci rozumieť konverziu dokumentov vo formáte Office Open XML obsahujúcich formátovaný text na dokument vo formáte XHTML.

V prvej časti tejto kapitoly si popíšeme existujúce dokumentácie k Office Open XML a ukážeme si aké majú výhody a nevýhody. Na základe neuspokojivých záverov z tejto časti sme vytvorili túto kapitolu. V druhej časti si povieme základné informácie o histórii a vývoji Office Open XML formátu. Spomenieme všetky významné mílniky. V tretej časti prejdeme k štruktúre tohto formátu. Ukážeme si najprv logické rozdelenie Open Office XML na menšie značkovacie jazyky, kde každý z nich má nejakú špecifickú funkciu. Potom sa pozrieme bližšie na fyzickú reprezentáciu tohto formátu. V štvrtej časti sa začneme venovať základnej téme tejto diplomovej práce - WordprocessingML. Keďže je WordprocessingML značne rozsiahly, je nutné dopredu poznamenať, že popis jednotlivých elementov bude zredukovaný. Bude to však len do takej miery, aby podstata funkcie elementu bola dostatočne objasnená.

2.1 Existujúca dokumentácia k Office Open XML

Informácie o Office Open XML je možné čerpať z viacerých zdrojov. Rozlišujeme niekoľko typov. Prvým sú oficiálne špecifikácie štandardov ECMA a ISO. Potom sú to rôzne online tutoriály a príručky na internete. Poslednou kategóriou sú knihy. Keďže nie sú veľmi rozšírené, spomenieme si len jednu.

2.1.1 Špecifikácia ECMA-376

Špecifikácia k štandardu ECMA-376 je príliš rozsiahla na to, aby čitateľ v krátkom čase získal dostatočné vedomosti o Office Open XML. Táto špecifikácia je rozdelená na 5 častí:

1. **Fundamentals** - základy typu zoznam skratiek, konvencie v zápisoch a podobne.
2. **Open Packaging Conventions** - vnútorná fyzická štruktúra dokumentov
3. **Primer** - informatívny úvod do Office Open XML jazykov

4. **Markup Language Reference** - podrobná dokumentácia jednotlivých Office Open XML jazykov
5. **Markup Compatibility and Extensibility** - popisuje konvencie, ktoré umožňujú budúce vylepšenie a rozšírenie Office Open XML dokumentov

2.1.2 Špecifikácia ISO/IEC 29500:2008

Špecifikácia k druhému štandardu zaviedla použitie dvoch úrovní „Strict“ a „Transitional“. Kým „Transitional“ je veľmi podobná špecifikácií ECMA a jej úloha je hlavne spätná kompatibilita, v úrovni „Strict“ sú zapracované (takmer) všetky pripomienky vznesené počas schvaľovacieho procesu. Táto špecifikácia na 7 228 stranách sa člení na 4 časti:

1. **Fundamentals and Markup Language Reference** - podrobná dokumentácia
2. **Open Packaging Conventions** - vnútorná fyzická štruktúra dokumentov (obdobne ako v ECMA-376)
3. **Markup Compatibility and Extensibility** - možnosti rozšírenia (obdobne ako v ECMA-376)
4. **Transitional Migration Features** - táto časť obsahuje elementy a atribúty nespomenuté v časti 1, ktoré zachovávajú kompatibilitu s predošlími Microsoft Office aplikáciami. Nachádza sa tu aj zoznam rozdielov medzi ECMA-376 a ISO/IEC 29500:2008[2].

Na základe tejto špecifikácie vznikla špecifikácia ECMA-376 2. verzia[3]. Podobne ako pri ECMA-376 1. verzii aj táto verzia je príliš rozsiahla.

2.1.3 Online tutoriály a príručky, knihy

Veľa informácii o Office Open XML sa dá nájsť na rôznych špecializovaných fórach a blogoch. Bohužiaľ, tieto informácie nie sú kompletné a ťažko sa v spleti mnohých článkov orientuje. Príkladom zaujímavého článku je napríklad [Jon06], avšak nie je dostatočne detailný. Ďalšia kategória online príručiek sú rôzne prehľady, ako napríklad [Ehr06] alebo [Mic03]. Prvý z menovaných je zameraný hlavne na fyzickú štruktúru dokumentu a druhý je zase zastaralý[4]. Posledný zdroj, ktorý si popíšeme je [Kos08]. Ide o slidy, na ktorých je

[2] Rozdiely sú napísané na siedmich stranách.
[3] ECMA-376 2. verzia a ISO/IEC 29500:2008 sú obsahovo rovnaké.
[4] opisuje WordprocessingML ešte vo verzii z Office 2003 XML

síce prehľadne, ale až príliš stručne popísaný Office Open XML. Nakoniec si spomenieme knihu [LML04], ktorá je síce prehľadná, ale bohužiaľ zastaralá, keďže opisuje predchodcu Open Office XML - formát Office 2003 XML. Táto kniha je vhodná len na úplne základné informácie o tomto formáte.

2.2 Chronológia vývoja Open Office XML

Začiatky Office Open XML siahajú do roku 2000, kedy spoločnosť Microsoft v beta verzii kancelárskeho balíka Microsoft Office XP použila ako alternatívny súborový formát pre tabuľkové dokumenty formát založený na XML. Vtedy ho nazvala SpreadsheetML a tento formát je predchodcom toho dnešného. Neskôr v roku 2002 bol spolu s beta verziou Microsoft Office 2003 zverejnený formát na ukladanie textového dokumentu - WordprocessingML, ktorý je tiež predchodcom toho dnešného. Oba formáty potom vystupovali pod názvom Office 2003 XML. V roku 2004 Európska únia odporučila spoločnosti Microsoft zverejniť a štandardizovať špecifikáciu k tomuto novému formátu. Rok na to spoločnosť vyhlásila, že po úpravách sa chystá štandardizovať tento formát pod hlavičkou združenia ECMA International. To sa nakoniec v roku 2006 aj stalo a špecifikácia Office Open XML sa stala medzinárodným štandardom ECMA-376. Vzápätí sa tento formát stal prednastaveným v najnovšom kancelárskom balíku Microsoft Office 2007. Po zapracovaní rôznych pripomienok bola špecifikácia štandardizovaná Medzinárodnou organizáciou pre normalizáciu ako štandard ISO/IEC 29500:2008. Stalo sa tak v novembri 2008 a preto na rozdiel od ECMA štandardu, ISO štandard bude plne zapracovaný až v kancelárskom balíku v nadchádzajúcej verzii Microsoft Office 2010. Jeden z rodielov oproti ECMA štandardu je, že ISO definuje dve úrovne Office Open XML - Strict a Transitional. Strict verzia má prísnejšiu štruktúru (v pôvodnej verzii je veľa vecí, ktoré sa v Office Open XML dajú spraviť viacerými spôsobmi) oproti úrovni Transitional, ktorej úloha je hlavne spätná kompatibilita.

2.3 Štruktúra Office Open XML

Formát Office Open XML umožňuje ukladať tak formátovaný text, ako aj tabuľky či prezentácie. Pre každý z týchto typov existujú špecializované značkovacie jazyky v rámci Open Office XML. Tieto značkovacie jazyky sú nasledujúce:

WordprocessingML Slúži na reprezentáciu formátovaného textu. V tejto práci bude tomuto značkovaciemu jazyku venovaný najväčší priestor.

SpreadsheetML Reprezentuje tabuľky.

PresentationML Reprezentuje prezentácie.

Office Open XML obsahuje aj takzvané zdieľané značkovacie jazyky, ktoré sa môžu vyskytovať aj v rámci vyššie spomenutých jazykov:

DrawingML Značkovací jazyk reprezentujúci vektorovú grafiku. Pomocou tohto značkovacieho jazyka je možné renderovať grafický text (s rôznymi efektami), rôzne geometrické tvary, ale tiež grafické tabuľky či grafy (koláčové, stĺpcové, ...). Tomuto značkovaciemu jazyku sa budeme venovať ešte neskôr v samostatnej časti.

VML (Vector Markup Language) Tento značkovací jazyk slúži tiež na zobrazovanie vektorovej grafiky. Spoločnosť Microsoft sa rozhodla tento jazyk použiť aj napriek tomu, že konzorcium W3C odporučilo ako štandard na zobrazovanie vektorovej grafiky jazyk SVG (Scalable Vector Graphics).

Office MathML Šlúži na reprezentovanie matematických zápisov (vzorcov, výrazov...). Tento značkovací jazyk bol podrobený kritike, pretože na matematické vzorce už existuje štandardizovaný značkovací jazyk MathML. V skutočnosti sú tieto jazyky veľmi príbuzné a jediný zásadný rozdiel je v tom, že do Office MathML sa na rozdiel od MathML dajú zakomponovať elementy z WordprocessingML ako napríklad poznámky pod čiarou, komentáre, obrázky, či dokonca formátovanie.

V ďalšom texte si povieme bližšie informácie hlavne o WordprocessingML a čiastočne o DrawingML. Teda značkovacie jazyky, ktoré sa aktívne podieľajú na reprezentácii dokumentov vytváraných v programe Word z kancelárskeho balíka Microsoft Office 2007. Bližšie informácie o ostatných jazykoch sú nad rámec tejto diplomovej práce.

2.4 Open Packaging Conventions

Formát Office Open XML, ako už bolo spomenuté, obsahuje vzájomne poprepájané súbory zbalené do jedného ZIP archívu. Vnútorná štruktúra v tomto archíve je definovaná

na základe štandardu vyvinutého tiež spoločnosťou Microsoft s názvom Open Packaging Conventions (OPC).

2.4.1 Balík a časti v OPC

Balíkom v kontexte Open Packaging Conventions chápeme kontajner[5] obsahujúci časti dokumentu. Tieto časti môžu niesť nejakú informáciu o dokumente (text, štýly, komentáre, obrázok, fonty, ...) alebo môžu definovať prepojenia medzi týmito informáciami. Príkladom prepojenia je napríklad informácia o umiestnení obrázka v štruktúre balíka. Fyzicky je balík ZIP archív, kde každý súbor v ZIP archíve reprezentuje nejakú časť balíka. Je to vlastne elegantný spôsob, ako celý dokument a všetky jeho súčasti (obrázky, fonty, ...) uchovávať v jednom fyzickom súbore.

Každý typ dokumentu má inú súborovú štruktúru v rámci balíka. Niektoré konvencie sú však pre všetky rovnaké.

adresár _rels V tomto adresári sa nachádzajú súbory definujúce vzťahy medzi rôznymi časťami. Pre tieto súbory je vyhradená prípona „.rel“. Bližšie informácie o definovaní vzťahov si povieme v samostatnej časti.

súbor /_rels/.rels Súbor vzťahov /_rels/.rels definuje vzťahy celého balíku ku konkrétnym častiam. Príkladom takéhoto vzťahu je umiestnenie hlavného dokumentu.

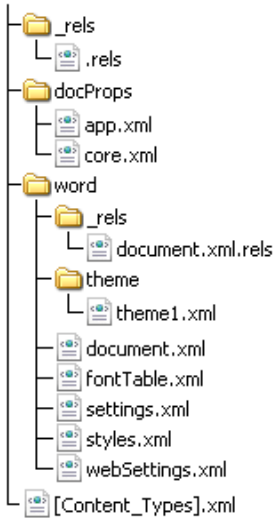
súbor [Content_Types].xml V tomto súbori sa definuje typ obsahu jednotlivých častí obsiahnutých v balíku. Podrobnejšie sa definícii typov obsahu budeme venovať neskôr v samostatnej časti.

Ako je vidno, povinných konvencií pri dodržiavaní štandardu OPC nie je veľa, avšak existujú i nepísané konvencie, ktoré sprehľadňujú celý dokument. Takáto konvencia napríklad je, že dokument aplikácie Microsoft Word má hlavný dokument uložený v adresári `/word/document.xml` a podobne. Ako taká typická štruktúra dokumentu aplikácie Microsoft Word vyzerá je možné vidieť na obrázku 2.1.

2.4.2 Prepojenia v OPC

Jednotlivé časti v dokumente môžu obsahovať referencie na iné časti v dokumente alebo aj mimo dokumentu (linky na web stránky a podobne). Tieto referencie v OPC nazývame

[5] Bohužiaľ, lepší preklad na anglické „container“ sa nedá nájsť. Myslíme na kontajner vo význame abstraktného balíka, ktorý obsahuje nejaké položky.



Obr. 2.1: Typická štruktúra WordprocessingML dokumentu

prepojenia. Definované sú v súbore uloženom v adresári `\_rels`, ktorý sa nachádza v tom istom adresári ako súbor, pre ktorý definujeme prepojenie. Navyše súbor, kde budú tieto prepojenia definované, bude mať také isté meno ako súbor, pre ktorý prepojenia definujeme, len bude mať pridanú príponu „.rel“. Napríklad súbor `/word/document.xml` bude mať prepojenia definované v súbore /word/_rels/document.xml.rel.

Súbor s prepojeniami je XML dokument s preddefinovanou štruktúrou. Hlavným koreňovým elementom je element **Relationships**. Potom nasleduje zoznam prepojení, každý v elemente **Relationship**. Atribúty tohto elementu sú nasledovné:

Id Atribút určuje jednoznačný identifikátor pre toto prepojenie. Hodnota musí byť jedinečná v rámci celého súboru s prepojeniami.

Type Atribút určuje typ prepojenia na základe URI. Zoznam najčastejšie používaných URI pre WordprocessingML môžeme vidieť v tabuľke 2.1.

Target Atribút určuje cieľový zdroj. Určený je na základe URI.

TargetMode Posledný atribút určuje či sa nachádza cieľový zdroj definovaný v atribúte Target vo vnútri balíka (vtedy má hodnotu „Internal“), alebo mimo balíka (hodnota „External“).

Teraz si ukážeme konkrétny príklad súboru s prepojeniami. V príklade 2.1 vidíme typický súbor /_rels/.rel pre WordprocessingML. Ako už bolo spomenuté, tento súbor je

Tabuľka 2.1: URI základných prepojení vo WordprocessingML dokumente

URI	**popis**
http://schemas.openxmlformats.org/officeDocument /2006/relationships/officeDocument	vzťah medzi balíkom a hlavným dokumentom
http://schemas.openxmlformats.org/officeDocument /2006/relationships/extended-properties	vzťah medzi balíkom a rozšírenými informáciami o dokumente (počet strán/riadkov/znakov, ...)
http://schemas.openxmlformats.org/package /2006/relationships/metadata/core-properties	vzťah medzi balíkom a prehľadom informácii o dokumente (názov a popis dokumentu, autor, dátum, ...)
http://schemas.openxmlformats.org/officeDocument /2006/relationships/styles	vzťah medzi dokumentom a externými štýlmi
http://schemas.openxmlformats.org/officeDocument /2006/relationships/numbering	vzťah medzi dokumentom a definíciou zoznamov
http://schemas.openxmlformats.org/officeDocument /2006/relationships/footnotes	vzťah medzi dokumentom a definíciou poznámok pod čiarou
http://schemas.openxmlformats.org/officeDocument /2006/relationships/endnotes	vzťah medzi dokumentom a definíciou poznámok na konci dokumentu
http://schemas.openxmlformats.org/officeDocument /2006/relationships/comments	vzťah medzi dokumentom a definíciou komentárov
http://schemas.openxmlformats.org/officeDocument /2006/relationships/theme	vzťah medzi dokumentom a témou
http://schemas.openxmlformats.org/officeDocument /2006/relationships/image	vzťah medzi dokumentom a obrázkom
http://schemas.openxmlformats.org/officeDocument /2006/relationships/hyperlink	vzťah medzi dokumentom a nejakým cieľom hyperlinky

Príklad 2.1 „.rel“ súbor vo WordprocessingML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships
    xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
    <Relationship Id="rId3"
        Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/
              extended-properties"
        Target="docProps/app.xml"/>
    <Relationship Id="rId2"
        Type="http://schemas.openxmlformats.org/package/2006/relationships/
              metadata/core-properties"
        Target="docProps/core.xml"/>
    <Relationship Id="rId1"
        Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/
              officeDocument"
        Target="word/document.xml"/>
</Relationships>
```

menšia výnimka, pretože definuje prepojenia k celému balíku a nie medzi časťami balíka. Súbor obsahuje tri prepojenia. Prvé prepojenie je podľa definície uvedenej v tabuľke 2.1 prepojenie na súbor s rozšírenými informáciami o dokumente, kde sa dá zistiť počet riadkov a iné sumáre bez potreby prechádzať celým dokumentom. Druhé prepojenie je

prepojenie tiež s informáciami o dokumente ale tento krát s menom autora, dátumom vytvorenia, poslednej zmeny a podobne. Posledné prepojenie definuje umiestnenie hlavného dokumentu.

2.4.3 Definícia typov obsahu v OPC

Definícia obsahu uloženého v jednotlivých častiach balíka sa nachádza v súbore `[Content_Types].xml`. Je to opäť XML dokument, ktorého štruktúru si teraz popíšeme. Koreňový element je **Types**. Tento element má potomkov element **Default** a element **Override**, v ktorých definujeme typy obsahu pre všetky súbory s danou príponou (Default) alebo pre konkrétne časti (Override).

Default element má atribúty Extension a ContentType. V prvom definujeme príponu, v druhom samotný typ obsahu. **Override** element má atribúty PartName a ContentType. Prvý z nich definuje časť balíka, pre ktorý definujeme obsah. Je v ňom uložená cesta v rámci balíka k danému súboru. Druhý parameter podobne ako pri prvom elemente definuje typ obsahu. Ak pre nejakú časť existuje aj **Override** aj **Default** element, použije sa typ definovaný v elemente **Override**.

Príklad 2.2 [Content_Types].xml súbor vo WordprocessingML dokumente

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Types xmlns="http://schemas.openxmlformats.org/package/2006/content-types">
    <Default
        Extension="rels"
        ContentType="application/vnd.openxmlformats-package.relationships+xml"/>
    <Default
        Extension="xml"
        ContentType="application/xml"/>
    <Override
        PartName="/word/document.xml"
        ContentType="application/vnd.openxmlformats-officedocument
                    .wordprocessingml.document.main+xml"/>
    <Override
        PartName="/word/styles.xml"
        ContentType="application/vnd.openxmlformats-officedocument
                    .wordprocessingml.styles+xml"/>
    <Override
        PartName="/docProps/app.xml"
        ContentType="application/vnd.openxmlformats-officedocument
                    .extended-properties+xml"/>
    <Override
        PartName="/docProps/core.xml"
        ContentType="application/vnd.openxmlformats-package
                    .core-properties+xml"/>
</Types>
```

Na príklade 2.2 vidíme jednoduchý dokument s definíciami obsahu vo WordprocessingML dokumente. Prvé dve definície určujú obsah súborov s príponou „.rels“ a „.xml“.

Ďalšia definícia určuje, že časť `/word/document.xml` obsahuje hlavný dokument. Ďalej vidíme, že súbor `/word/styles.xml` obsahuje štýly, `/docProps/app.xml` rozšírené informácie o dokumente a nakoniec `/docProps/core.xml` základné informácie o dokumente.

Prepojenia verzus definícia typov obsahu

Čitateľ si iste všimol náznak redundancie informácií o obsahu jednotlivých súborov v dokumente. Informácia o obsahu súboru je obsiahnutá v súbore s prepojeniami aj v súbore s typmi obsahu. Obe časti sú aj napriek tomu dôležité a to hneď z niekoľkých dôvodov. V prvom rade, informácia o obsahu v prepojeniach nie je úplná. Napríklad informácia o prepojení obrázka a nečíselného zoznamu nehovorí nič o formáte obrázka. Práve o tomto formáte hovorí typ obsahu súboru. Ďalším dôvodom je, že OPC neslúži len na definovanie WordprocessingML dokumentov, ale v podstate akéhokoľvek iného formátu. A `[Content\_Types].xml` súbor je podľa špecifikácie na rozdiel od definovania prepojení povinný.

2.5 WordprocessingML

V tejto časti si predstavíme WordprocessingML dokument. WordprocessingML dokument pozostáva z hlavného dokumentu[6] a jeho súčastí. V nich sa nachádzajú doplnkové informácie, texty, štýly a podobne. Teraz sa detailnejšie pozrieme na tento hlavný dokument, na jeho štruktúru a potom postupne spomenieme aj jeho ďalšie súčasti, ktoré sú s týmto hlavným dokumentom prepojené.

2.5.1 Jednoduchý text

Hlavný dokument je XML súbor, ktorý v štruktúre balíka tvorí najväčší podiel na výslednom dokumente. Skôr ako sa pozrieme detailnejšie na konkrétne XML elementy ukážeme si na príklade 2.3 jednoduchý dokument s textom „Hello world!“.

Koreňový element v hlavnom dokumente je **document**. Ten obsahuje element **body**, ktorý vymedzuje telo dokumentu. Telo dokumentu je členené na odstavce, ktoré obsahujú behy[7] a tie ešte obsahujú samotný text. Odstavce sú označené elementom **p**, beh elementom **r**. Behy definuje také úseky textu alebo iných komponentov, ktoré majú rov-

[6] Hlavný dokument je v štruktúre balíka umiestnený na základe prepojení, ktoré sme si definovali v časti 2.4.2.

[7] z anglického „run“

Príklad 2.3 Hlavný dokument vo WordprocessingML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
  <w:body>
    <w:p>
      <w:r>
        <w:t>Hello world!</w:t>
      </w:r>
    </w:p>
  </w:body>
</w:document>
```

naké formátovanie. Môže to byť napríklad niekoľko slov zobrazených hrubým písmom. Beh okrem zobraziteľného textu, ktorý je vždy v elemente **t**, môže obsahovat napríklad aj zalomenie riadku pomocou elementu **br**.

Poznámka. Treba si všimnúť, že meno každého elementu má prefix pred dvojbodkou. V príklade je to prefix „w“. Týmto je definovaný menný priestor daného elementu.

2.5.2 Formátovanie textu

Formátovanie textu v dokumente sa môže realizovať niekoľkými spôsobmi. Prvým z nich je využiť element **rPr**[8], ktorý definuje formátovanie pre daný beh alebo element **pPr**[9], ktorý definuje formátovanie pre daný odstavec. Ďalším spôsobom je definovať štýl, ktorý sa potom k danému behu alebo odstavcu pripojí využitím elementu **rStyle** resp. **pStyle**.

2.5.2.1 Formátovanie behu

Na formátovanie behu slúži už spomenutý element **rPr**, umiestnený na začiatku behu, ktorý chceme formátovať. Do tohto elementu potom môžme pridávať elementy, ktoré slúžia na formátovanie. Príklad 2.4 zobrazuje výsek dokumentu, v ktorom vypíšeme text „**Hello** world!“ („Hello“ hrubým písmom). Ako vidieť z príkladu, hrubé písmo dosiahneme použitím elementu **b**.

Poznámka. Štandardne sa medzere v texte na začiatku alebo na konci ignorujú. Nastavením atribútu *xml:space* na hodnotu „preserve“ zabezpečíme, že sa medzere ignorovať nebudú.

V tabuľke 2.2 je zoznam bežných formátovacích elementov a k nim popis, atribúty a ich význam.

[8] skratka z „run properties“ - vlastnosti behu
[9] skratka z „paragraph properties“ - vlastnosti odstavca

Príklad 2.4 Formátovanie behu vo WordprocessingML

```
...
<w:p>
    <w:r>
        <w:rPr>
            <w:b/>
        </w:rPr>
        <w:t>Hello</w:t>
    </w:r>
    <w:r>
        <w:t xml:space="preserve"> world!</w:t>
    </w:r>
</w:p>
...
```

Tabuľka 2.2: Zoznam formátovacích elementov pre beh

Element	**popis**	**atribúty**
b	hrubý font	*val* - on/off, prednastavený je on
i	kurzíva	*val* - on/off, prednastavený je on
u	podčiarknuté písmo	*val* - typ podčiarkuntia, napríklad „double“
strike	prečiarknuté písmo	*val* - on/off, prednastavený je on
vertAlign	vertikálne zarovnanie	*val* - typ vertikálneho zarovnania (superscript - horný index, subscript - dolný index, baseline - klasické zarovnanie v riadku)
color	farba textu	*themeColor* - špecifikuje farebnú tému definovanú v časti s témami *val* - RGB hodnota farby
sz	veľkosť fontu	*val* - veľkosť fontu v polbodoch
rFonts	meno fontu	*ascii*/*hAnsi* - meno fontu pre znaky ASCII rozsahu / ostatné znaky *asciiTheme*/*hAnsiTheme* - špecifikuje tému, kde je definovaný font
highlight	farba pozadia	*val* - farba pozadia (slovná hodnota, napr. red, blue, yellow, ...)
shd	farba pozadia[10]	*fill* - RGB hodnota farby *themeFill* - špecifikuje farebnú tému definovanú v časti s témami

2.5.2.2 Formátovanie odstavca

Analogicky k formátovaniu behu sa definuje formátovanie odstavca. Na rozdiel od behu sa však na odstavec viažu iné možnosti formátovania. Dá sa pracovať s odstavcom ako takým a nie so samotným textom. Na formátovanie odstavca teda slúži element **pPr**, ktorý musí byť umiestnený na začiatku odstavca, ktorý chceme formátovať. Možnosti formátovacích elementov je možné vidieť v tabuľke 2.3.

Element **pBdr** ešte môže obsahovať ďalšie elementy, ktoré definujú, jednotlivé strany

[10] Element **shd** má podobnú funkciu ako element **highlight**, ale umožňuje navyše nastaviť vlastnosti, ktoré sú nad rámec tejto diplomovej práce (napríklad pattern).

[11] 1440 twipsov = 1 palec = 96 pixelov

Tabuľka 2.3: Zoznam formátovacích elementov pre odstavec

Element	popis	atribúty
shd	farba pozadia	*fill* - RGB hodnota farby *themeFill* - špecifikuje farebnú tému definovanú v časti s témami
pBdr	rámček okolo odstavca	-
jc	zarovnanie textu v odstavci	*right*, *left*, *center*, *both* - v poradí vpravo, vľavo, na stred, rovnomerne rozmiestniť (justify)
ind	horizontálne odsadenie	*left* - odsadenie zľava *right* - odsadenie sprava (vyjadrené v twipsoch[11])
spacing	riadkovanie	*line* - výška riadku (vyjadrené v 240-tinách riadku) *before* - veľkosť medzery pred odstavcom *after* - veľkosť medzery za odstavcom (vyjadrené v twipsoch)

rámčeka. Sú to elementy **top** (vrchná strana), **right** (pravá strana), **bottom** (spodná strana) a **left** (ľavá strana). Každý z týchto elementov môže mať tieto atribúty: *color* (farba v hexadecimálnom RGB tvare), *val* (typ čiary, napríklad „dashed“, „single“, „solid“) a *sz* (hrúbka čiary - vyjadrené v osminách bodu).

Príklad 2.5 Formátovanie odstavca vo WordprocessingML

```
...
<w:p>
    <w:pPr>
        <w:pBdr>
            <w:top w:val="single" w:sz="24" w:color="FF0000" />
            <w:bottom w:val="single" w:sz="24" w:color="FF0000" />
        </w:pBdr>
        <w:shd w:fill="0000F0" />
        <w:spacing w:line="480" />
        <w:ind w:left="720" />
    </w:pPr>
    <w:r>
        <w:t>Hello world!</w:t>
    </w:r>
</w:p>
...
```

V príklade 2.5 vidíme fragment hlavného dokumentu, v ktorom je zadefinovaný odstavec s textom „Hello world!“. Tento odstavec má nastavené modré pozadie a červené horné a dolné okraje s hrúbkou 4 pixely. Navyše je tento odstavec odsadený zľava o pol palca (48 pixelov) a má nastavené dvojité riadkovanie.

2.5.2.3 Formátovanie štýlmi

Štýly umožňujú vytvorenie množiny formátovacích elementov, ktoré je možné priradiť behu alebo odstavcu. Štýly zmenšujú veľkosť dokumentu a hlavne ho sprehľadňujú a zjednodušujú úpravy. Definujú sa v časti štýly[12] a referencujú sa na tom istom mieste, kde sa definuje formátovanie behu alebo odstavca. V elemente **rStyle** resp. **pStyle** cez atribút *val* určíme, ktorý štýl sa má použiť a ten potom platí v celom behu, resp. v celom odstavci.

Dokument s definíciami štýlov má koreňový element **styles**. Najdôležitejší potomkovia tohto elementu sú elementy **docDefaults** a **style**. V prvom z nich sa definujú prednastavené hodnoty platné na celý dokument (napríklad veľkosť fontu, názov fontu a podobne). Prednastavené hodnoty pre každý beh sa definujú v elemente **rPrDefaults** a pre každý odstavec v elemente **pPrDefaults**. Definovanie formátovania v týchto elementoch je už analogické k definovaniu formátovania v behoch resp. v odstavcoch.

V elementoch **style** sa zase definuje samotný štýl. Element má atribúty *type* a *styleId*. Prvý z nich definuje typ štýlu a druhý je jednoznačný textový identifikátor štýlu. Typy štýlov môžu byť:

- paragraph - štýl sa viaže na odstavce
- character - štýl sa viaže na jednotlivé znaky
- numbering - štýl sa viaže na číselné a nečíselné zoznamy
- table - štýl sa viaže na tabuľku

Štýly sú definované tak, že sa do elementu **style** vloží **rPr** element (definovanie štýlu pre beh) alebo **pPr** element (definovanie štýlu pre odstavec). Definovanie štýlov v týchto elementoch je už štandardné (tzn. vkladáme do nich formátovacie elementy). Ďalší element, ktorý sa môžu vyskytnúť v elemente **style** je **basedOn** element. Umožňuje cez atribút *val* nastaviť štýl, z ktorého má aktuálny štýl zdediť formátovacie elementy.

Použitie štýlov si teraz ukážeme na príklade 2.6. Prvý fragment dokumentu je definícia štýlov vo vnútri **styles** elementu v časti so štýlmi. Definujeme dva štýly (Italic - kurzíva a ItalicBold - hrubý font kurzívou), ktoré sa viažu na celý odstavec a kde druhý z nich dedí vlastnosti toho prvého. V druhom fragmente vidíme použitie štýlu v hlavnom dokumente. Výsledkom je odstavec, v ktorom je kurzívou a hrubým fontom napísaný text „Hello world!“.

[12] Kde sa štýly nachádzajú, zistíme z prepojení hlavného dokumentu, teda štandardne v `/word/\_rels/document.xml.rel`

Príklad 2.6 Príklad použitia štýlov vo WordprocessingML

```
súbor "/word/styles.xml":
...
<w:style w:type="paragraph" w:styleId="Italic">
    <w:rPr>
        <w:i/>
    </w:rPr>
</w:style>
<w:style w:type="paragraph" w:styleId="ItalicBold">
    <w:basedOn w:val="Italic"/>
    <w:rPr>
        <w:b/>
    </w:rPr>
</w:style>
...

súbor "/word/document.xml":
...
<w:p>
    <w:pPr>
        <w:pStyle w:val="ItalicBold" />
    </w:pPr>
    <w:r>
        <w:t>Hello World!</w:t>
    </w:r>
</w:p>
...
```

2.5.2.4 Hierarchia formátovania

Pri definovaní formátovania (či už priamo, alebo cez štýly) často dochádza k situácii, že na jeden fragment textu platí viac formátovacích vzájomne sa vylučujúcich elementov. Definovanie formátovania má však svoje pravidlá, na základe ktorých sa vždy dá jednoznačne určiť, aké formátovanie sa nakoniec použije. Funguje nasledovná hierarchia:

- Najprv sa použijú prednastavené štýly[13].
- Potom sa použijú štýly definované pre tabuľku.
- Ďalej sa použijú štýly definované pre číselné a nečíselné zoznamy.
- Potom nasledujú štýly definované pre odstavce.
- Potom nasledujú štýly definované pre beh.
- Nakoniec nasledujú formátovacie elementy priamo definované v odstavcoch a behoch.

Formátovaniu tabuľky a zoznamov sa budeme venovať v príslušných častiach.

2.5.3 Číselné a nečíselné zoznamy

Vo WordprocessingML sú zoznamy odstavce, ku ktorým je priradený štýl zoznamu. Čo odlišuje klasický odstavec od položky zoznamu je prítomnosť elementu **listPr** v elemente

[13] element **rPrDefaults**

pPr. V tomto elemente je definovaný štýl zoznamu a úroveň (level) aktuálnej položky. Tieto vlastnosti sú určené elementmi **ilfo** a **ilvl**. V prvom z nich sa cez atribút *val* definuje štýl zoznamu a v druhom taktiež cez atribút *val* úroveň vnorenia v zozname[14]. Definície štýlov zoznamu nájdeme na základe definície prepojení na hlavný dokument. Štandardne to však je v súbore `/word/numbering.xml`.

V dokumente so štýlmi zoznamov sa definujú najprv abstraktné štýly a až potom sa definujú konkrétne štýly, ktoré musia dediť vlastnosti jedného z abstraktných štýlov. Koreňový element v tomto dokumente je **numbering**.

Príklad 2.7 Príklad obrázkovej odrážky vo WordprocessingML

```
...
<w:numPicBullet w:numPicBulletId="0">
    <w:pict>
        <v:shape>
            <v:imagedata r:id="rId1" />
        </v:shape>
    </w:pict>
</w:numPicBullet>
...
```

Ak sa v odrážkach vyskytujú obrázky, tak tieto sa definujú v elemente **numPicBullet**. Atribútom *numPicBulletId* sa nastaví jedinečný identifikátor, ktorým sa neskôr referencuje. Obsahuje element **pict**, v ktorom je element **v:shape** a ten obsahuje element **v:imagedata**. V ňom sa cez atribút *r:id* nastavuje identifikátor obrázka definovaného v prepojeniach dokumentu so zoznamami (štandardne v súbore `/word/\_rels/numbering.xml.rels`). V príklade 2.7 vidíme definovanie obrázkovej odrážky. Obrázok s identifikátorom „rId1“ je potom definovaný v prepojeniach dokumentu so zoznamami.

Abstraktný štýl sa definuje elementom **abstractNum**. V ňom sa elementami **lvl** určujú vlastnosti jednotlivých úrovní zoznamu. Túto úroveň určíme atribútom *ilvl*[15]. Teraz si uvedieme zoznam elementov, ktoré definujú vlastnosti danej úrovne:

start Využíva sa pri číselných zoznamoch. Atribútom *val* určuje od ktorého čísla sa bude úroveň číslovať.

numFmt Určuje typ odrážky. Nastavuje sa atribútom *val* a môže nadobúdať napríklad tieto hodnoty:

- bullet - klasická nečíselná odrážka

[14] Zoznam môže byť vnorený v inom zozname.
[15] Najvyššia úroveň má *ilvl* 0, prvá vnorená 1, atď...

- decimal - číslo
- lowerLetter - malé písmená abecedy
- upperLetter - veľké písmená abecedy
- lowerRoman - malé rímske čísla
- upperRoman - veľké rímske čísla

lvlText Atribútom *val* určuje „text“ odrážky. Pri nečíselných zoznamoch určuje znak (alebo znaky) ktoré sa zobrazia ako odrážka. Pri číselných zoznamoch má špeciálny význam znak percenta nasledovaný číslom. Reprezentuje poradie položky pre úroveň vyjadrenú číslom za znakom percenta. Použitie tohoto atribútu môžeme vidieť v príklade 2.8.

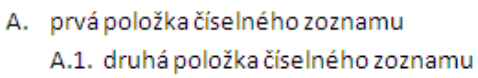
lvlPicBulletId - identifikátor skôr definovaného obrázka, ktorý sa má použiť ako odrážka

pPr - miesto pre formátovacie elementy vzťahujúce sa na odstavec (odsadenie)

rPr - miesto pre formátovacie elementy vzťahujúce sa na beh (font)

Definícia samotných štýlov sa nachádza v elementoch **num**. V atribúte *numId* sa potom definuje jednoznačný identifikátor, ktorým sa štýl číslovania použije v hlavnom dokumente. V elemente **num** sa potom nachádza element **abstractNumId**, ktorým sa určí, z ktorého abstraktného štýlu zoznamu sa majú zdediť vlastnosti. Môže ešte obsahovať element **lvlOverride**, v ktorom sa dajú predefinovať vlastnosti zoznamu pre úroveň definovanú v jeho atribúte *ilvl*. V ňom sa nachádzajú už štandardné elementy **lvl**.

Jednoduchá definícia štýlu číselného zoznamu a následné použitie vidíme v príklade 2.8. Prvý fragment je z dokumentu definujúceho štýly číslovania. Kvôli prehľadnosti sú definované len dve úrovne. Druhý fragment je priamo z hlavného dokumentu a demonštruje jednu položku zoznamu na úrovni 1 ($ilvl = 0$) a pod ňou je vnorený zoznam tiež s jednou položkou na úrovni 2 ($ilvl = 1$). Oba používajú štýl číslovania s id = 1. Ako vyzerá výstupný dokument môžeme vidieť na obrázku 2.2.



Obr. 2.2: Číselný zoznam

Príklad 2.8 Príklad číselného zoznamu vo WordprocessingML

```
súbor "/word/numbering.xml":
<w:numbering>
    <w:abstractNum w:abstractNumId="0">
        <w:lvl w:ilvl="0">
            <w:start w:val="1"/>
            <w:numFmt w:val="upperRoman"/>
            <w:lvlText w:val="%1."/>
            <w:pPr>
                <w:ind w:left="720"/>
            </w:pPr>
        </w:lvl>
        <w:lvl w:ilvl="1">
            <w:start w:val="1"/>
            <w:numFmt w:val="decimal"/>
            <w:lvlText w:val="%1.%2."/>
            <w:pPr>
                <w:ind w:left="720"/>
            </w:pPr>
        </w:lvl>
    </w:abstractNum>
    <w:num w:numId="1">
        <w:abstractNumId w:val="0"/>
    </w:num>
</w:numbering>

súbor "/word/document.xml":
...
<w:p>
    <w:pPr>
        <w:numPr>
            <w:ilvl w:val="0"/>
            <w:numId w:val="1"/>
        </w:numPr>
    </w:pPr>
    <w:r>
        <w:t>prvá položka číselného zoznamu</w:t>
    </w:r>
</w:p>
<w:p>
    <w:pPr>
        <w:numPr>
            <w:ilvl w:val="1"/>
            <w:numId w:val="1"/>
        </w:numPr>
    </w:pPr>
    <w:r>
        <w:t>druhá položka číselného zoznamu</w:t>
    </w:r>
</w:p>
...
```

2.5.4 Tabuľky

Tabuľky sa vo WordprocessingML označujú elementom **tbl** a vkladajú sa do elementu **body**. Element **tbl** môže obsahovať nasledujúce elementy:

- **tblPr** - vlastnosti tabuľky (bližšie informácie sú nad rámec tejto diplomovej práce)
- **tblGrid** - obsahuje pre každý stĺpec element **gridCol**, v ktorom sa atribútom *w* nastavuje jeho šírka

- **tr** - riadok tabuľky

Riadky potom obsahujú bunky tabuľky, ktoré reprezentuje element **tc**. Bunky obsahujú element **tcPr**, v ktorom sa definujú vlastnosti bunky, a nakoniec samotný obsah. Vlastnosti bunky sú napríklad šírka bunky (element **tcW**), definícia okrajov bunky (element **tcBorders**) alebo elementy ktoré majú na starosti spájanie buniek v tabuľke (tým sa budeme venovať neskôr). Obsah bunky je väčšinou odstavec textu, teda element **p**, ale môže to byť napríklad aj ďalšia vnorená tabuľka.

Spájanie buniek je relatívne neprehľadná záležitosť, pretože WordprocessingML umožňuje hneď niekoľko spôsobov. Prvý z nich je spájanie prostredníctvom elementov **vMerge** a **hMerge**. Prvý z nich umožňuje vertikálne spájanie buniek (riadky) a druhý horizontálne (stĺpce). Oba fungujú rovnakým spôsobom, že sa nastaví atribútu *val* hodnota „restart“ a tým sa určí prvá bunka v spojenej oblasti. Každá ďalšia bunka patriaca do tejto oblasti potom tento element s nastaveným atribútom *val* na „continue“ (resp. nenastaveným vôbec, pretože táto hodnota je prednastavená).
Ďalší spôsob spájania buniek je elementami **gridSpan** a **rowSpan**. Prvý slúži na spájanie buniek v riadku (teda spája stĺpce) a druhý na spájanie buniek v stĺpci (spája riadky). Cez atribút *val* sa celočíselnou hodnotou nastaví počet buniek, ktoré sa spoja. Ďalšie bunky, ktoré sa nachádzajú v takto definovanej spojenej oblasti, sa potom vynechávajú.

Príklad 2.9 Príklad tabuľky vo WordprocessingML

```
<w:tbl>
    <w:tblPr>
        ...
    </w:tblPr>
    <w:tblGrid>
        <w:gridCol w:w="4606"/>
        <w:gridCol w:w="4606"/>
    </w:tblGrid>
    <w:tr>
        <w:tc>
            <w:tcPr>
                <w:gridSpan w:val="2"/>
            </w:tcPr>
            <w:p><w:r><w:t>1</w:t></w:r></w:p>
        </w:tc>
    </w:tr>
    <w:tr>
        <w:tc>
            <w:p><w:r><w:t>2</w:t></w:r></w:p>
        </w:tc>
        <w:tc>
            <w:p><w:r><w:t>3</w:t></w:r></w:p>
        </w:tc>
    </w:tr>
</w:tbl>
```

Teraz si ukážeme jednoduchú tabuľku na príklade 2.9, kde vidíme fragment z hlavného dokumentu. Tabuľka má rozmery 2x2 a v prvom riadku sú spojené obe bunky. Táto spojená bunka obsahuje text „1“. Zvyšné bunky obsahujú text „2“ a „3“.

2.5.5 Poznámky pod čiarou

Poznámky pod čiarou sa vo WordprocessingML štandardne definujú v súbore `/word/footnotes.xml`[16] a referencujú sa potom v hlavnom dokumente. Dokument s definíciami poznámok má koreňový element **footnotes**, v ktorom sa nachádzajú elementy **footnote**. V nich sú definované samotné poznámky. Atribútom *id* elementu **footnote** sa nastavuje jednoznačný identifikátor, ktorým potom pristupujeme k poznámke v hlavnom dokumente. Element môže obsahovať element **footnotePr**, ktorým sa nastavujú rôzne vlastnosti (ich popis je však nad rámec tejto diplomovej práce). Obsahuje text samotnej poznámky a to väčšinou štandardne v elemente **p**.

Takto definované poznámky sa potom referencujú v hlavnom dokumente prostredníctvom elementu **footnoteReference** umiestneného v samostatnom behu (element **r**). Na definovanú poznámku sa prepojí prostredníctvom atribútu *id*, ktorý obsahuje hodnotu jednoznačného identifikátora poznámky.

V jednoduchom príklade 2.10 vidíme použitie poznámky pod čiarou za textom „Hello world!“ s textom poznámkou „toto je poznámka“. V prvom fragmente je dokument definujúci poznámky pod čiarou. Element **footnoteRef** reprezentuje číselnú hodnotu, ktorá definuje poznámku v texte. V druhom fragmente je použitie tejto poznámky v texte hlavného dokumentu.

2.5.6 Poznámky na konci dokumentu

S poznámkami na konci dokumentu sa pracuje presne rovnako ako s poznámkami pod čiarou. Jediný rozdiel je v tom, že všetky názvy „footnote“ treba nahradiť názvom „endnote“.

2.5.7 Hyperlinky

Hyperlinky vo WordprocessingML sa vkladajú pomocou prepojení. Najprv sa v dokumente s prepojeniami zadefinuje prepojenie na externý cieľ, teda sa do atribútu *Id* nastaví nejaký

[16] Umiestnenie určujú prepojenia hlavného dokumentu

Príklad 2.10 Príklad poznámky pod čiarou vo WordprocessingML

```
súbor "/word/footnotes.xml":
<w:footnotes>
    <w:footnote w:id="2">
        <w:p>
            <w:r>
                <w:footnoteRef />
            </w:r>
            <w:r>
                <w:t xml:space="preserve"> </w:t>
            </w:r>
            <w:r>
                <w:t>toto je poznámka</w:t>
            </w:r>
        </w:p>
    </w:footnote>
</w:footnotes>

súbor "/word/document.xml":
...
<w:p>
    <w:r>
        <w:t>Hello world!</w:t>
    </w:r>
    <w:r>
        <w:footnoteReference w:id="2" />
    </w:r>
</w:p>
...
```

jednoznačný identifikátor, do atribútu *Type* dá URI určujúce hyperlink, do *Target* sa dá cieľ odkazu a *TargetMode* sa nastaví na „External“.

Textový odkaz sa potom do hlavného dokumentu vkladá elementom **hyperlink** a atribútom *id* sa nastaví identifikátor odkazu. Tento element sa vkladá do elementu **p** a text v ňom sa definuje elementom **r**.

Keď chceme mať hyperlink z obrázka, namiesto elementu **hyperlink** použijeme element **hlinkClick** s rovnakými vlastnosťami. Rozdiel je len v umiestnení, keďže **hlinkClick** sa vkladá do elementu **docPr** v obrázku. Bližšie informácie o obrázkoch si uvedieme neskôr.

V príklade 2.11 si ukážeme definíciu a použitie jednoduchého odkazu na stránku http://www.foo.sk/. V prvom fragmente je časť dokumentu s definíciami prepojení[17] a v druhom použitie definovaného odkazu v hlavnom dokumente. Dokument vytvára hyperlink s textom „Toto je klikací odkaz.“ odkazujúcim na stránku http://www.foo.sk/.

[17] Prepojenia boli detailnejšie opísané v časti 2.4.2

Príklad 2.11 Príklad hyperlinky vo WordprocessingML

```
súbor "/word/_rels/document.xml.rels":
...
<Relationship
    Id="rId7"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/hyperlink"
    Target="http://www.foo.sk/" TargetMode="External"
/>
...

súbor "/word/document.xml":
...
<w:p>
    <w:hyperlink r:id="rId5">
        <w:r>
            <w:t>Toto je klikací odkaz.</w:t>
        </w:r>
    </w:hyperlink>
</w:p>
...
```

2.5.8 Komentáre

Komentáre sa vo WordprocessingML definujú štandardne v dokumente `/word/comments.xml`[18]. Koreňový element je **comments**. Ten obsahuje elementy **comment**, v ktorých sú definované samotné komentáre. Atribútom *id* sa určí jednoznačný identifikátor, atribútom *author* sa určí autor poznámky, atribút *date* určuje dátum vytvorenia a nakoniec atribút *initials* určuje inicálky autora (hodnota sa použije na označenie komentára). Môže obsahovať akýkoľvek blokový element (tzn. **p**, **tbl**). Špeciálny význam má element **annotationRef**, ktorý reprezentuje označenie poznámky v texte.

V hlavom dokumente sa potom poznámky referencujú elementami **commentRangeStart** a **commentRangeEnd**, ktoré určujú začiatok a koniec komentovanej časti. Nachádzajú sa v elemente **p** a medzi nimi môže byť prakticky akýkoľvek obsah. Atribútom *id* sa určí identifikátor komentára, ktorému chceme začiatok a koniec v dokumente určiť. Nakoniec elementom **commentReference** definujeme vloženie komentára do dokumentu. Atribútom *id* sa určí identifikátor komentára, ktorý chceme do dokumentu vložiť. Tento element by sa mal nachádzať v elemente **r**.

Teraz si ukážeme použitie komentárov na príklade 2.12. V prvom fragmente vidím definíciu komentárov. Zadefinovali sme jeden komentár s identifikátorom „0“, autorom „Frantisek Smitala“ a s iniciálkami „FS“. Text komentára je „Toto je komentár“. V druhom fragmente je časť hlavného dokumentu, kde na text „Tento text bude okomentovaný.“

[18] Umiestnenie určujú prepojenia hlavného dokumentu

Príklad 2.12 Príklad komentárov vo WordprocessingML

```
súbor "/word/comments.xml":
<w:comments>
    <w:comment w:id="0" w:author="Frantisek Smitala"
    w:date="2009-02-20T10:13:00Z" w:initials="FS">
        <w:p>
            <w:r>
                <w:annotationRef />
            </w:r>
            <w:r>
                <w:t>Toto je komentár</w:t>
            </w:r>
        </w:p>
    </w:comment>
</w:comments>

súbor "/word/document.xml":
...
<w:p>
    <w:commentRangeStart w:id="0" />
    <w:r>
        <w:t>Tento text bude okomentovaný.</w:t>
    </w:r>
    <w:commentRangeEnd w:id="0" />
    <w:r>
        <w:commentReference w:id="0" />
    </w:r>
</w:p>
...
```

aplikujeme pred tým definovaný komentár.

2.5.9 Obrázky

Obrázky sa do dokumentu vkladajú pomocou elementu **drawing**, ktorý sa nachádza v elemente **r**. Do tohto elementu sa potom vkladajú elementy značkovacieho jazyka DrawingML. Prvý takýto element určuje, že akým spôsobom sa obrázok vkladá do textu. Môže to byť element **wp:inline** alebo **wp:anchor**. Prvý z nich určuje, že sa obrázok vloží do riadku s textom. Druhý z nich určuje, že obrázok bude ukotvený priamo v texte. Spôsob obtekania textu okolo obrázka bude definovaný v potomkoch tohto elementu. Teraz si ukážeme zoznam možných potomkov týchto elementov:

- **wp:extent** - Atribútmi *cx* a *cy* určuje rozmer zobrazovaného obrázka. Udáva sa v jednotkách EMU[19].
- **wp:docPr** - Obsahuje rôzne vlastnosti obrázka, ale hlavne môže obsahovať element **hlinkClick**, ku ktorému sú bližšie informácie v časti 2.5.7.
- **wp:positionH** - Určuje horizontálnu pozíciu obrázka.

[19] 360000 EMU = 1 cm

- **wp:positionV** - Určuje vertikálnu pozíciu obrázka.
- **wp:wrapSquare** - Jeho výskyt definuje spôsob obtekania textu, v tomto prípade bude text obtekať obrázok tak, že sa okolo neho vytvorí (virtuálny) obdĺžnik.
- **wp:wrapTopAndBottom** - Text bude obrázok obtekať zvrchu a zospodu, ale nie po bokoch.
- **a:graphic** - Tento element obsahuje informácie o obrázku. Bližšie informácie si uvedieme neskôr.

Elementy **wp:positionH** a **wp:positionV** určujú kde bude obrázok ukotvený. Oba obsahujú atribúty *relativeFrom*, ktorý určuje, že vzhľadom na čo sa bude obrázok kotviť[20]. Potom obsahujú element **wp:posOffset** alebo **wp:align**. V prvom z nich sa definuje posun obrázka horizontálne a vertikálne v jednotkách EMU cez textový obsah elementu. V druhom z nich sa definuje zarovnanie obrázka cez textový obsah elementu hodnotami „right“ (vpravo), „left“ (vľavo) alebo „center“ (na stred).

Element **a:graphic** obsahuje len element **a:graphicData**. V jeho atribúte *uri* sa zadefinuje URI, ktorá definuje obsah tohto elementu. V našom prípade je to obrázok - `http://schemas.openxmlformats.org/drawingml/2006/picture`. Potomkom tohto elementu je potom už samotný obrázok, ktorý je definovaný elementom **pic:pic**. Element **pic:pic** môže obsahovať viac elementov, my sa budeme venovať iba dvom z nich. Prvý z nich je element **pic:blipFill** a druhý **pic:spPr**.

Element **pic:blipFill** určuje spôsob akým sa obrázok vyplní. Môže obsahovať elementy **a:blip** a **a:stretch**. Prvý z nich definuje prostredníctvom atribútu *r:embed* jednoznačný identifikátor obrázka definovaného v prepojeniach. Druhý z nich definuje, že sa obrázok má roztiahnuť aby vyplnil celý definovaný priestor. Obsahuje jeden element **a:fillRect**. Tento definuje na akej ploche z definovaného priestoru sa má obrázok roztiahnuť. Pomocou atribútov *b*, *l*, *r*, *t* môžme percentuálne vymedziť túto plochu. V danom poradí určujú percentuálne odsadenie zdola, zlava, sprava a zhora. Keď sa vynechajú atribúty, obrázok sa natiahne na celý definovaný priestor.

Element **pic:spPr** určuje vlastnosti obrázka. My sa budeme zaoberat jedným jeho potomkom - elementom **a:ln**. Tento určuje orámovanie obrázka. Atribútom *w* nastavíme hrúbku čiary v EMU jednotkách. Môže obsahovať element **a:solidFill** alebo **a:prstDash**. Prvý z nich definuje plnú výplň orámovania a farba výplne sa definuje v jeho potomkoch.

[20] napríklad „paragraph“ - odstavec, „column“ - stĺpec, „margin“ - okraj...

Druhý element definuje prerušovanú čiaru orámovania. Farba výplne môže byt definovaná dvoma spôsobmi. Prvý spôsob definovania farby je elementom **a:srgbClr**. V jeho atribúte *val* je 6 miestny hexadecimálny zápis RGB hodnoty farby. Druhý spôsob je cez element **a:schemeClr**, ktorým cez atribút *val* nastavíme farbu pomocou farebnej schémy definovanej v dokumente s témamy. Definícia prerušovaného rámčeka sa robí elementom **a:prstDash**, v ktorom nastavením atribútu *val* určíme spôsob prerušovania. Tieto spôsoby môžu byt „dash“ - čiarkovane, „dot“ - bodkovane a nakoniec „solid“ - súvislá čiara.

Príklad 2.13 Príklad obrázkov vo WordprocessingML

```
súbor "/word/_rels/document.xml.rels":
<Relationships>
    <Relationship
        Id="rId8"
        Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image"
        Target="media/obrazok.jpeg" />
</Relationships>

súbor "/word/document.xml":
...
<w:drawing>
    <wp:anchor>
        <wp:positionH relativeFrom="column">
            <wp:posOffset>2540</wp:posOffset>
        </wp:positionH>
        <wp:positionV relativeFrom="paragraph">
            <wp:posOffset>2540</wp:posOffset>
        </wp:positionV>
        <wp:extent cx="2755900" cy="2066925" />
        <wp:wrapSquare />
        <a:graphic>
            <a:graphicData uri="http://schemas.openxmlformats.org/drawingml/2006/picture">
                <pic:pic>
                    <pic:blipFill>
                        <a:blip r:embed="rId8" />
                        <a:stretch>
                            <a:fillRect />
                        </a:stretch>
                    </pic:blipFill>
                    <pic:spPr>
                        <a:ln w="38100">
                            <a:solidFill>
                                <a:srgbClr val="00FF00" />
                            </a:solidFill>
                            <a:prstDash val="dash" />
                        </a:ln>
                    </pic:spPr>
                </pic:pic>
            </a:graphicData>
        </a:graphic>
    </wp:anchor>
</w:drawing>
...
```

Teraz si na príklade 2.13 ukážeme vloženie obrázka do dokumentu. Obrázok chceme ukotvit v texte a chceme aby text obtekal okolo obrázka. Ďalej chceme, aby bol okolo

obrázka rámček z prerušovanej čiary zelenej farby. V prvom fragmente v príklade vidíme definíciu obrázka v prepojeniach. V druhom fragmente je samotné vloženie obrázka do dokumentu. Je dôležité poznamenat, že pre jednoduchosť boli odstránené niektoré časti dokumentu, ktoré neboli nevyhnutné na demonštráciu obrázkov vo WordprocessingML.

Poznámka. Výskyt prefixov elementov, ktoré určujú menné priestory v tejto časti je zámerný, aby sme sa vyhli nedorozumeniam. Kým v predchádzajúcich častiach sme pracovali iba s jedným menným priestorom, tu ich je hneď niekoľko. Zoznam menných priestorov použitých v tejto kapitole sa nachádza v tabuľke 2.4.

Tabuľka 2.4: Zoznam menných priestorov

prefix	**URI**
w	http://schemas.openxmlformats.org/wordprocessingml/2006/main
wp	http://schemas.openxmlformats.org/drawingml/2006 /wordprocessingDrawing
a	http://schemas.openxmlformats.org/drawingml/2006/main
pic	http://schemas.openxmlformats.org/drawingml/2006/picture
r	http://schemas.openxmlformats.org/officeDocument/2006 /relationships
v	urn:schemas-microsoft-com:vml

2.5.10 Témy

Viackrát sme sa v tejto kapitole odvolávali na časť dokumentu s témami. Teraz si ukážeme, akú štruktúru má tento dokument. Koreňovým elementom je **theme**, ktorý v atribúte *name* určuje názov danej témy. Pod týmto elementom sa nachádza element **themeElements**. V ňom sa nachádzajú napríklad elementy **clrScheme** alebo **fontScheme**. V prvom sa definujú farebné schémy, v druhom schémy pre fonty. My sa budeme venovať iba farbám. Element **clrScheme** obsahuje atribút *name*, kde sa definuje meno pre danú schému. Obsahuje elementy definujúce jednotlivé farebné schémy. Na meno týchto elementov sa odvolávame pri použití farebnej schémy v hlavnom dokumente. V týchto elementoch sa nachádza element definujúci farbu, teda najčastejšie elementom **a:srgbClr**, ktorý sme si definovali v časti 2.5.9. Analogicky k farebným schémam sa definujú schémy s fontami v elemente **fontScheme**.

Teraz si na príklade 2.14 ukážeme definovanie a použitie farebných schém. V prvom fragmente vidíme definovanie tém a farebnej schémy *accent1* na RGB hodnotu 4F81BD.

Príklad 2.14 Príklad použitia farebnej schémy vo WordprocessingML

```
súbor "/word/theme/theme1.xml":
<a:theme name="Office Theme">
    <a:themeElements>
        <a:clrScheme name="Office">
            ...
            <a:accent1>
                <a:srgbClr val="4F81BD" />
            </a:accent1>
            ...
        </a:clrScheme>
        ...
    </a:themeElements>
    ...
</a:theme>

súbor "/word/document.xml":
...
<w:p>
    <w:r>
        <w:rPr>
            <w:color w:themeColor="accent1" />
        </w:rPr>
        <w:t>Hello world!</w:t>
    </w:r>
</w:p>
...
```

V druhom fragmente z hlavného dokumentu vidíme použitie tejto farebnej schémy ako farbu textu „Hello world!“.

Kapitola 3

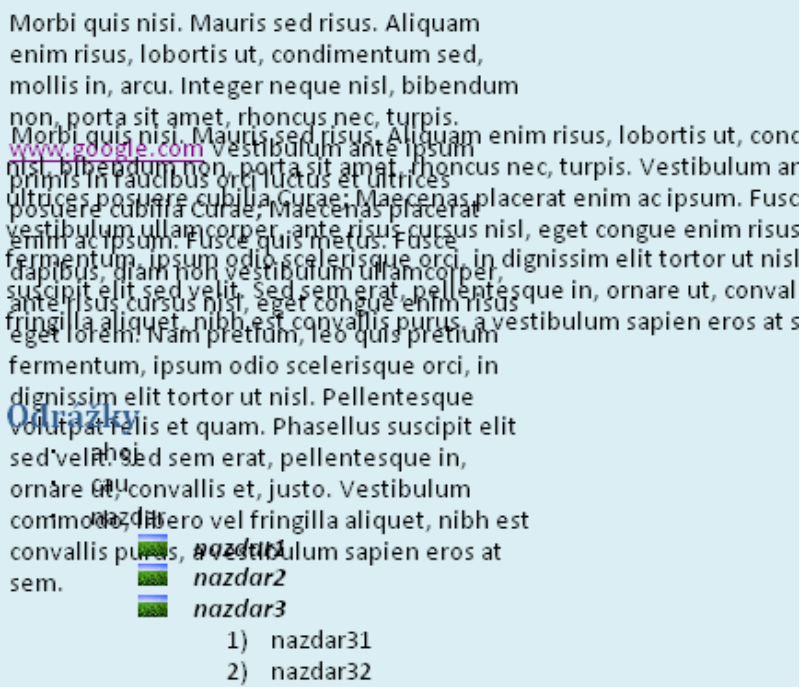
Existujúce riešenia

V tejto kapitole sa bližšie pozrieme na existujúce riešenia konverzie dokumentov vo formáte DOCX na dokumenty vo formáte XHTML. Ukážeme si súčasné možnosti takejto konverzie a tiež poukážeme na výhody a nevýhody jednotlivých riešení.

3.1 Export z Microsoft Office 2007

Prvá z možností konverzie DOCX dokumentu na XHTML dokument je priamo v aplikácii Microsoft Word 2007. Prvá nevýhoda tohto riešenia spočíva v tom, že balík Microsoft Office 2007 nie je práve najlacnejší softvér. Preto je nepravdepodobné, že by si ho niekto, kto túto aplikáciu nevlastní, kupoval len kvôli konverzii DOCX na XHTML. Ďalším problémom je veľmi neprehľadný zdrojový kód a nezachovávanie štruktúry. Výstupný dokument napríklad nezachoval zoznam, ale namiesto toho CSS štýlmi upravil klasické odstavce aby zoznam pripomínali. Taktiež po vizuálnej stránke Microsoft Office zaostáva, keď XHTML výstup sa inak zobrazuje v rôznych prehliadačoch, ako je možné vidieť na obrázku 3.1.

Word umožňuje dve možnosti exportu. Prvá z nich by mala za cenu objemnosti a neprehľadnosti zdrojového kódu ponúkať takmer totožný vizuálny výstup ako originál dokumentu. Druhá možnosť (filtrovaná) sa snaží neplytvať miestom, čo sa odrazí na menšej vizuálnej zhode s originálom. Bohužiaľ aj prvá aj druhá verzia má zdrojový kód neprehľadný a zbytočne veľký a tiež pri oboch verziách sa vyskytuje problém so zobrazením niektorých častí dokumentu. Konkrétne problémy si čitateľ môže pozrieť na priloženom CD médiu. V prehliadači Microsoft Internet Explorer je výstup nefiltrovanej verzie na dobrej úrovni, v ostatných prehliadačoch je v podstate nepoužiteľný.



Obr. 3.1: XHTML dokument skonvertovaný programom Microsoft Word 2007 zobrazený v prehliadači Firefox

3.2 Export z Open Office 3.0

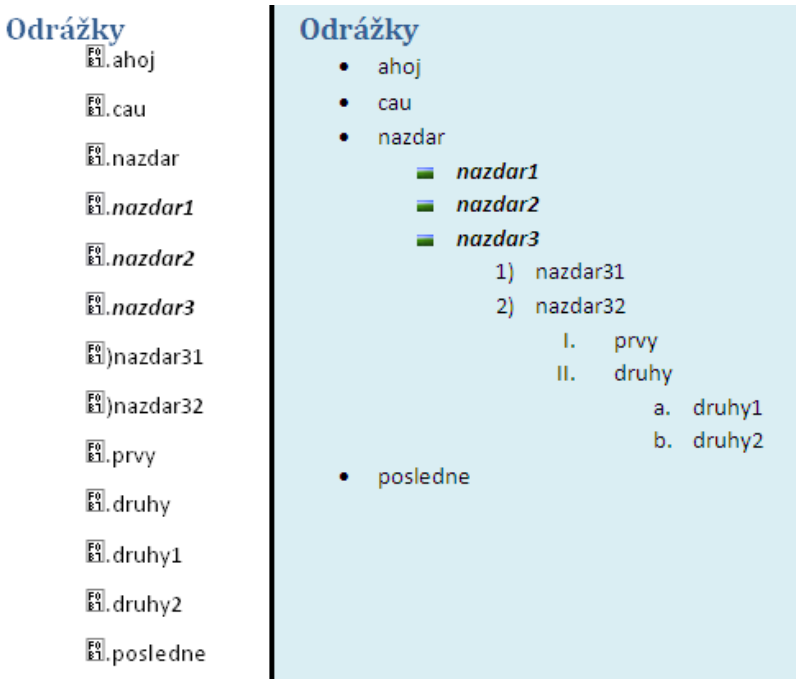
Ďalšia možnosť exportu, tentoraz už zadarmo, je prostredníctvom kancelárskeho balíka Open Office. Program umožňuje exportovať aj do HTML aj do XHTML. Ako prvé čo si používateľ všimne je fakt, že už len samotný DOCX dokument Open Office nezobrazí správne. Preto sa netreba čudovať, keď aj XHTML, resp. HTML export nie je vizuálne veľmi podobný na svoju DOCX predlohu.

Zdrojový kód XHTML dokumentu je veľmi neprehľadný a mal obrázky uložené priamo v sebe[1], čo je niekedy veľmi nepraktické a hlavne neštandardné. Výsledný dokument tiež neprešiel testom validity. Oproti HTML exportu sa na svoj DOCX originál podobá oveľa menej.

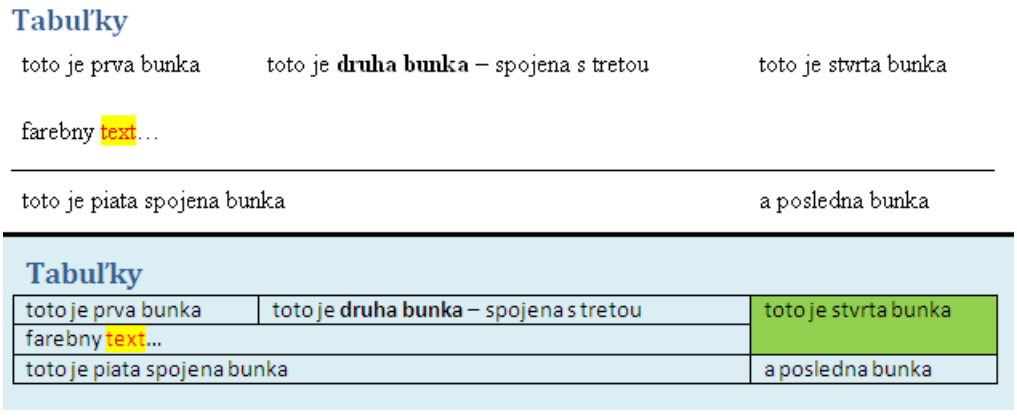
Prehľadnosť zdrojového kódu HTML dokumentu je na relatívne slušnej úrovni. Z vizuálnej stránky je tento export lepší ako spomenutý XHTML dokument, stále má ale problémy napríklad s tabuľkami, s farbou pozadia a iné. Všetky obrázky konvertuje do GIF formátu, čo je tiež niekedy výrazným problémom. GIF formát nepodporuje viac ako 256 farieb a tiež je veľmi neefektívny pri ukladaní fotografií.

[1] Obrázky kódované v base64 a sú vkladané do *src* atribútu elementu **img**.

Na obrázku 3.2 je porovnanie číselného zoznamu v XHTML dokumente, ktorý je výstupom konverzie DOCX dokumentu programom Open Office 3.0, a originálneho číselného zoznamu DOCX dokumentu. Na ďalšom obrázku je porovnanie tej istej tabuľky v HTML dokumente a v pôvodnom DOCX dokumente.



Obr. 3.2: Vľavo: XHTML dokument, vpravo: pôvodný DOCX dokument.



Obr. 3.3: Hore: HTML dokument, dole: pôvodný DOCX dokument.

3.3 Plugin pre prehliadač Mozilla Firefox

Ďalšou možnosťou je nainštalovať si do prehliadača Mozilla Firefox plugin na zobrazovanie DOCX dokumentov. Tento plugin funguje tak, že zdrojový DOCX dokument skonvertuje na XHTML a potom otvorí. Bohužiaľ, toto riešenie je opäť nedostačujúce. Zdrojový kód tohto výstupu je veľmi neprehľadný a tiež nezachováva štruktúru dokumentu. Podobne ako Microsoft Office 2007 aj tento konvertor pretransformoval zoznam na odstavce, ktoré sú CSS štýlmi upravené aby vyzerali ako zoznam. CSS štýly priamo v zdrojovom kóde XHTML dokumentu sú tiež neprehľadné a hlavne zbytočne objemné. Čo sa týka podobnosti XHTML dokumentu na pôvodný DOCX dokument je na tom toto riešenie relatívne dobre. Trocha je problém s obrázkami ukotvenými priamo v texte (ktoré XHTML dokument dal do riadku) a tiež s odrážkami zoznamu, ktoré vôbec neboli zobrazené.

3.4 Online konverzia

Možnost konverzie online službou na internete ponúka viac webovských stránok. Väčšinou však ide o platené služby. Prevádzkovatelia týchto služieb využívajú fakt, že keď potrebujeme otvoriť dokument vo formáte DOCX a nevlastníme Microsoft Office 2007, tak je lacnejšie využiť ich službu, ako si tento produkt kupovať. Implicitne ani staršia verzia Microsoft Office 2003 nepodporuje formát DOCX. Prevádzkovatelia využívajú to, že väčšina používateľov nie je vždy technicky zdatná a teda nemá prístup k informáciam o bezplatných riešeniach. Navyše, konverziu DOCX na XHTML berú skôr ako doplnkovú službu[2] a teda nie je predpoklad, že by XHTML výstup bol na vysokej úrovni.

Medzi týmito online službami je však jedna výnimka. Je ňou konvertor **DOCX Convert Office 2007** (`http://www.docx-converter.com/`), ktorý je zadarmo. Je to však len online implementácia konvertora používaného v Open Office. Jediným rozdielom je verzia. Kým v časti 3.2 sme písali o verzii 3.0 na platforme Windows, táto online implementácia je vo verzii 2.3 na platforme Linux. Problémy sú tam teda podobné. Avšak až na problém s obrázkom, ktorý je nesprávne umiestnený, je staršia verzia oproti novšej lepšia v zachovaní dizajnu pôvodného DOCX dokumentu.

[2] Najrozšírenejšia je konverzia do staršieho DOC formátu.

3.5 GMail náhľad dokumentov

Posledná možnosť je dosť nepraktická. Služba GMail v doručenej pošte ponúka v prípade prílohy vo formáte DOCX možnosť zobraziť HTML náhľad. Teda keď chceme spraviť konverziu z DOCX dokumentu na HTML, tak najprv musíme mať GMail účet, potom musíme poslať na tento účet email a ako prílohu pridať DOCX dokument a potom si ho v doručenej pošte zobraziť. Výsledný HTML dokument má síce relatívne jednoduchý zdrojový kód, avšak nepoužíva štýly a všetky formátovania robí HTML tagmi a elementami a tiež podporuje len základné možnosti formátovania. Vôbec nepodporuje obrázky. Podobnosť s pôvodným DOCX dokumentom je teda len približná. Okrem toho, tento náhľad obsahuje vložený automatický odstavec s informáciou o tom, že dokument je len náhľad bez obrázkov. Keď chceme teda tento výstup použiť, je nutné tento odstavec ručne zmazať.

3.6 Zhrnutie

V tejto časti zhrnieme jednotlivé možnosti konverzie do tabuľky 3.1. Budeme pri tom sledovať tieto vlastnosti exportovaných HTML resp. XHTML dokumentov zo vzorového DOCX dokumentu[3]:

1. Formát dokumentu - HTML/XHTML.
2. Či je dokument valídny - Hodnoty áno/nie a v zátvorke počet chýb[4].
3. Úroveň zachovania formátovania - teda či je výsledný dokument podobný na svoj DOCX originál. Hodnotíme číselnou hodnotou od 1 do 5, kde 1 je najlepšia úroveň a 5 je najhoršia úroveň. Do úvahy sa berie zobrazenie v prehliadači Firefox 3.0, Opera 9.6 a Internet Explorer 7.
4. Úroveň zachovania štruktúry - teda či zoznamy sú v HTML resp. XHTML tagoch pre zoznamy, nadpisy v tagoch pre nadpisy a podobne. Hodnotíme číselnou hodnotou od 1 do 5, kde 1 je najlepšia úroveň a 5 je najhoršia úroveň.
5. Prehľadnosť zdrojového kódu. Hodnotíme číselnou hodnotou od 1 do 5, kde 1 je najlepšia úroveň a 5 je najhoršia úroveň.
6. Veľkosť HTML/XHTML dokumentu spolu s CSS štýlmi udaná v kB.
7. Celková veľkosť exportu udaná v kB.

[3]Referenčný dokument DOCX podobne ako aj všetky HTML resp. XHTML exporty sa nachádzajú v prílohe na CD médiu.

[4]Dokumenty boli testované oficiálnym validátorom na stránke konzorcia W3C.

Tabuľka 3.1: Zhrnutie kvality XHTML resp. HTML exportov

riešenie	**formát**	**validita**	**formátovanie**	**štruktúra**	**prehľadnosť**	**veľkosť bez obrázkov**	**celková veľkosť**
MS Word	XHTML	nie (372)	3	3	5	74.4	507
MS Word (filtrovaný)	HTML	nie (5)	1	3	3	21.8	418
Open Office (HTML)	HTML	nie (9)	3	1	2	12.9	74.2
Open Office (XHTML)	XHTML	nie (26)	4	1	5	52.6[5]	52.6
Firefox plugin	XHTML	nie (8)	4	5	4	29.1	87.7
DOCX Convert Office 2007	HTML	nie (18)	2	2	2	16.8	414
GMail	HTML	nie (9)	4	1	1	9.49	9.57

[5] V tejto veľkosti je zarátaná aj veľkosť obrázkov, kedže sú priamo v XHTML dokumente.

Kapitola 4

Aplikácia Office Open XML convert

Teraz nasleduje popis realizovania praktickej časti tejto práce - tvorby programu na konverziu DOCX dokumentov na XHTML dokumenty, ktorý dostal názov **Office Open XML convert**, resp. skrátene **OOXML convert**. Aplikáciu budeme tvoriť v troch etapách. Prvá bude špecifikácia, teda zhrnieme si všetky vlastnosti, ktoré od aplikácie očakávame. V ďalšej časti na základe požadovaných vlastností vytvoríme návrh aplikácie a vyberieme si prostriedky, pomocou ktorých aplikáciu vytvoríme. Nakoniec si popíšeme samotnú implementáciu.

4.1 Špecifikácia aplikácie

Očakávané vlastnosti programu sme už čiastočne definovali v časti 1.3. Teraz ich postupne rozpíšeme a pridáme ďalšie.

Najdôležitejšia vlastnosť programu je **schopnosť zachovávať štruktúru**[1] **bežných dokumentov**. Viackrát sa v predošlom texte vyskytol pojem bežného či štandardného dokumentu. Teraz si ukážeme, ktoré súčasti dokumentu považujeme za bežné. V prvom rade to budú nadpisy a odstavce. Tieto definujú základnú štruktúru a sú teda nevyhnutné. Pri nadpisoch požadujeme možnosť zvoliť si k jednotlivým štýlom nadpisov v DOCX dokumente úroveň nadpisov v XHTML dokumente. Ďalej sú to zoznamy, kde chceme aby sa zachoval aj typ zoznamu, teda či je číselný alebo nečíselný. Samozrejmosťou by malo byť

[1] Pod štruktúrou myslíme rozdelenie na nadpisy, odstavce, zoznamy a podobne.

zvládnutie zachovania vnorených zoznamov. Poznámky pod čiarou a na konci dokumentu tiež budeme považovať za súčasť bežných dokumentov. Ďalej chceme zachovávať tabuľky a základnú štruktúru tabuľky. Tým myslíme hlavne spájanie buniek. Chceme tiež zachovať obrázky a ich pozíciu v texte a nakoniec aj hyperlinky na externé zdroje. Hyperlinkom nemusí byť len text, ale tiež obrázok.

Ďalšia vlastnosť programu je **zachovanie formátovania**. Teraz si špecifikujeme, aké formátovanie chceme vo výslednom XHTML dokumente zachovávať. Pri texte všeobecne chceme zachovávať hlavne to, či je text hrubým fontom, kurzívou alebo je podčiarknutý či prečiarknutý, ďalej samotný font, farbu a veľkosť písma, farbu pozadia a nakoniec horné a dolné indexy. Pri odstavcoch chceme zachovať výšku riadku, odsadenie odstavca zo všetkých strán, rámčeky okolo odstavcov a tiež štýl zarovnania textu. Nakoniec chceme prenášať obrázky a základné vlastnosti obrázka ako sú rozmery, orámovanie a štýl orámovania.

Chceme aby sa dala **nastaviť úroveň prenášania formátovania**. Mali by existovať tri rôzne úrovne, kde v prvej sa nebude prenášať žiadne formátovanie, v druhej iba základné, ako sú napríklad farba fontu, hrúbka fontu, podčiarknutie a prečiarknutie, horný a dolný index. Nakoniec v poslednej úrovni chceme prenášať všetky vyššie spomenuté formátovania.

Ďalšia podstatná vlastnosť je, aby bol XHTML výstup **valídnym XHTML dokumentom a aby mal Strict DTD**. Dôvod prečo má výstupný dokument Strict DTD je hlavne prísnejšie rozdelenie dizajnu a formátovania dokumentu od samotného obsahu a štruktúry dokumentu. Týmto si zabezpečíme maximálnu možnú prehľadnosť. V prípade, že má čitateľ záujem o bližšie dôvody výberu formátu, je dobré prečítať si [Dow00], [Joh05] a [Web03].

Ďalšia vlastnosť je primeraná **rýchlosť konverzie**. Teda je potrebné všetky výpočtové kroky zoptimalizovať.

Tiež chceme, aby bola aplikácia **použiteľná aj inými aplikáciami** a teda aby bola konzolová. Pomocou spúšťacích parametrov nastavíme požadované parametre konverzie. Kvôli jednoduchosti použitia treba tiež spraviť nejaké používateľské rozhranie, kde sa budú môcť vizuálne nastaviť jednotlivé parametre konverzie a po tomto kroku aplikácia spustí s požadovanými parametrami vyššie spomenutú konzolovú aplikáciu.

Ďalšia vlastnosť by mala byť možnosť **lokálne v dokumente zmeniť nastavenia**

vstupných parametrov. Napríklad chceme, aby sa dalo nastaviť, že sa polke dokumentu bude prenášať iba základné formátovanie, ale v druhej polke rozšírené. Malo by sa to dať robiť komentármi v dokumente, ktoré budú mať presne definovanú štruktúru.

Nakoniec požadujeme možnosť **prepojiť XHTML výstup na existujúce CSS štýly**. Malo by sa dať kontrolovať, či sa CSS štýly vytvoria nanovo, alebo sa jednotlivé štýly pridajú do existujúceho dopredu určeného CSS súboru. Posledná možnosť by mala byť menovité prepojenie štýlov na existujúce štýly v dopredu určenom CSS súbore. Teda napríklad, keď máme stránku s existujúcimi štýlmi a do nej chceme pridať nejaký DOCX dokument, tak len určíme, ktorá trieda má na starosti hrubý font, alebo kurzívu a podobne. XHTML výstup bude v tomto prípade prepojený na tieto existujúce štýly.

4.2 Návrh aplikácie

V druhej časti tvorby softvéru sa budeme venovať návrhu aplikácie. Začneme výberom programovacieho jazyka, potom sa budeme zaoberať voľbou pomocných knižníc a nakoniec si zanalyzujeme metódy riešenia problému.

4.2.1 Programovací jazyk

V prvom rade si bolo treba vybrať programovací jazyk. Prvá alternatíva bola použiť .NET a napríklad jazyk C# a druhá zase Java. Výhoda prvej alternatívy je hlavne predpoklad, že na prácu s Office Open XML bude obsahovať kvalitnejšie knižnice, keďže aj .NET aj Office Open XML pochádza od tej istej spoločnosti. Nakoniec ale prevážil fakt, že Java je multiplatformová, a teda neviazanosť na konkrétny operačný systém umožní väčšie rozšírenie aplikácie.

4.2.2 Pomocné knižnice

Ďalším problémom je voľba pomocných knižníc. K dispozícii máme hneď niekoľko možností. Vo všeobecnosti môžme tieto možnosti rozdeliť na knižnice na prácu s Office Open XML a knižnice na prácu s XML.

Do prvej kategórie spadajú knižnice **OpenXML4J** a **docx4j**. Obe knižnice vytvárajú objektovú reprezentáciu Office Open XML dokumentu. Toto by nám uľahčilo prácu, keďže by sme nemuseli potrebné súčasti dokumentu parsovať. Problémom však je fakt, že uvedené

knižnice sú až príliš zložité a na naše účely by ich použitie bolo zbytočné. Projekty tohoto typu majú až príliš veľký záber (aj keď v skutočnosti potrebujeme využiť len časť z nich) a potom klesá ich výkon. Jeden z dôvodov prečo je priamy prístup k XML lepší je aj to, že potom je možné na tomto programe demonštrovať štruktúru Office Open XML opísanú v kapitole 2.

V druhej kategórii sú knižnice na prácu s XML. Použitie takejto knižnice nie je nevyhnutné, keďže aj samotná Java ponúka hneď niekoľko štandardných knižníc na parsovanie XML dokumentov, ale práca so špecializovanou knižnicou je jednoduchšia. Hlavným kritériom výberu bola rýchlosť a jednoduchosť použitia. Voľba nakoniec padla na knižnicu **JDOM**[2]. Táto knižnica ponúka možnosť čítať a zapisovať XML dokumenty a to rýchlo a narába úsporne s pamäťou.

4.2.3 Metódy riešenia problému

Ďalším krokom bolo zvolenie si metódy riešenia problému. Prvá alternatíva bola použiť XSLT techniku a druhá programovo preparsovať obsah Office Open XML. Aj keď je XSLT určené práve na takéto transformácie, niektoré veci by sa v ňom robili oveľa zložitejšie[3] ako v nejakom vyššom programovacom jazyku. Naším cieľom je spraviť hlavne kvalitný export, čo by sa pomocou XSLT veľmi ťažko dosahovalo. Preto bude aplikácia naprogramovaná tak, že sa preparsujú potrebné XML dokumenty obsiahnuté v DOCX dokumente a spraví sa XHTML výstup.

4.3 Implementácia aplikácie

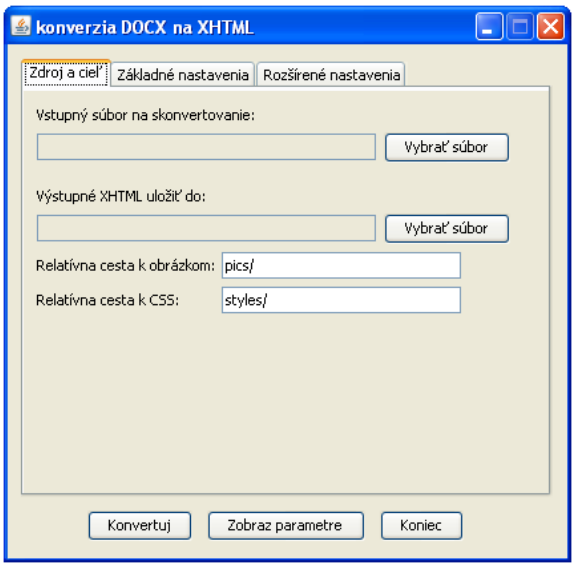
Aplikácia bude mať dve časti. Prvá bude grafické rozhranie, kde si používateľ na štandardných komponentoch nastaví požadované vlastnosti konverzie. Druhá časť bude konzolová aplikácia zabezpečujúca samotnú konverziu. Popis implementácie si rozdelíme tiež na dve časti. V prvej si popíšeme grafické rozhranie, ktoré bude slúžiť aj ako používateľská príručka a v druhej časti si popíšeme postup implementácie konzolovej aplikácie.

[2] Ak má čitateľ záujem o zaujímavý článok s porovnaním existujúcich knižníc na prácu s XML pre Javu, je dobré si prečítať [xml07]

[3] Napríklad optimalizovanie použitia **span** elementov o ktorom si povieme v časti 4.3.2.5.

4.3.1 Grafické používateľské rozhranie

Vychádzajúc zo špecifikácie je implementácia grafického používateľského rozhrania už jednoduchá. Jednotlivé nastavenia sme rozdelili na 3 karty, aby sa v množstve nastavení ľahšie orientovalo.

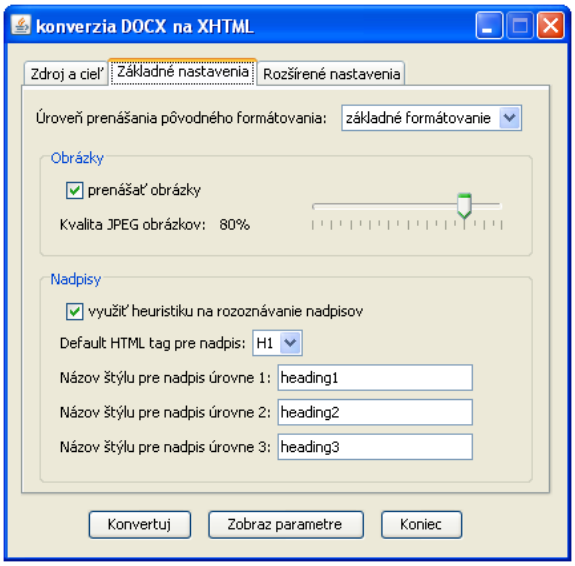


Obr. 4.1: Screenshot grafického používateľského rozhrania - karta *Zdroj a cieľ*

Na obrázku 4.1 vidíme prvú kartu rozhrania. Vyberajú sa na nej najdôležitejšie a povinné položky - cesta k DOCX dokumentu, ktorý chceme skonvertovať a cesta k XHTML dokumentu, kam sa má uložiť výstup. Cesta k obrázkom a k CSS súboru sa určuje relatívne k umiestneniu XHTML dokumentu a zabezpečuje prehľadnosť výstupu.

Na obrázku 4.2 sa nachádza screenshot z druhej karty rozhrania, ktorá ponúka základné možnosti nastavenia. Najdôležitejšia položka je výberovník na nastavenie úrovne prenášania formátovania z originálneho DOCX dokumentu. Obsahuje tri hodnoty s nasledujúcim významom:

- **žiadne formátovanie** - Určuje, že sa CSS štýly nebudú vôbec vytvárať. Exportuje sa iba čistá štruktúra dokumentu bez formátovania.
- **základné formátovanie** - CSS štýly sa budú vytvárať, ale iba na základné formátovanie: hrubý font, kurzíva, podčiarknutie, prečiarknutie, horný a dolný index a farba písma.



Obr. 4.2: Screenshot grafického používateľského rozhrania - karta *Základné nastavenia*

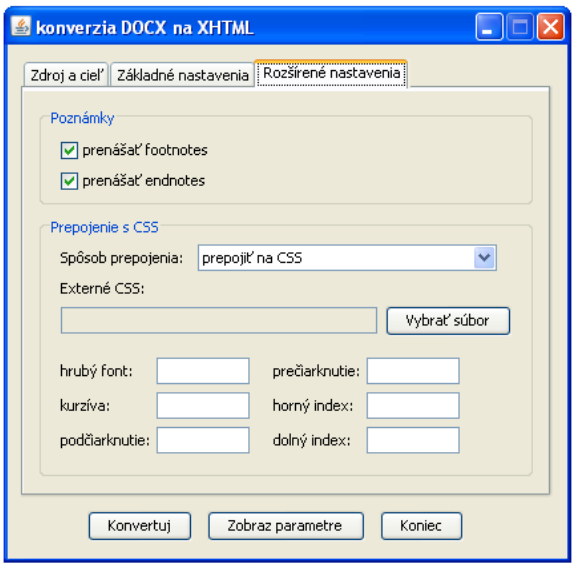
- **rozšírené formátovanie** - Budú sa prenášať všetky pre program známe formátovania. Rozsah je definovaný v časti 4.1 na mieste kde sa píše o zachovávaní formátovania.

V časti obrázky môžeme určiť, či ich chceme prenášať do výsledného dokumentu a ak áno, tak môžeme určiť úroveň kompresie JPEG obrázkov[4]. Keďže Microsoft Word ukladá originálne obrázky a zmeny na obrázkoch (zmena veľkosti a podobne...) sa uchovajú len ako informácie, tak sa po konverzii obrázky musia spracovávať a v prípade formátu JPEG sa budú nanovo komprimovať. Platí, že čím vyššia kvalita, tým zaberá obrázok viac miesta na disku a opačne.

Posledná možnosť na tejto karte sú nastavenia nadpisov. Keďže v DOCX dokumentoch nie sú nadpisy explicitne určené, treba na zisťovanie nadpisov používať heuristiku. Viac sa o fungovaní tejto heuristiky je možné dočítať v časti 4.3.2. Heuristiku môžeme zapnúť alebo vypnúť príslušným zaškrtávacím políčkom na tejto karte. Hneď pod touto voľbou sa nachádza textové pole „Default HTML tag pre nadpis“, ktoré určuje, aký XHTML element bude reprezentovať heuristikou rozpoznané nadpisy. Najbežnejší spôsob ako sa označujú nadpisy v DOCX dokumente sú štýly. Ak názvy týchto štýlov poznáme a chceme ich priradiť jednotlivým tagom na nadpisy v XHTML, tak je to možné spraviť v textových

[4]Tento parameter sa použije len v prípade, že dokument obsahuje JPEG obrázok. Pre iné formáty sa kompresia nenastavuje.

poliach umiestnených na tejto karte.



Obr. 4.3: Screenshot grafického používateľského rozhrania - karta *Rozšírené nastavenia*

Na poslednej karte, ktorú vidíme na obrázku 4.3 môžeme nastaviť to, či chceme prenášať poznámky pod čiarou (footnotes) a poznámky na konci dokumentu (endnotes). Keďže vo webovskom dokumente nie sú strany, poznámky pod čiarou aj na konci dokumentu nie sú vo výslednom XHTML dokumente rozlišované. Ďalej si na tejto karte pomocou výberovníka (combo box) môžeme nastaviť spôsob prepojenia štýlov na existujúci CSS súbor. Máme k dizpozícii tieto hodnoty:

- **žiadne prepojenie** - Táto voľba určuje, že sa vytvorí nový CSS súbor a neprihliada sa na žiadne existujúce štýly.
- **štýly pridať do CSS** - Týmto je určené, že do výstupného CSS súboru sa najprv skopíruje existujúci CSS súbor a potom sa pridajú všetky ostatné vlastné štýly. V tomto prípade treba daný CSS súbor špecifikovať v poli „Externé CSS“.
- **prepojiť na CSS** - Táto voľba určí, že do výstupného CSS súboru sa skopíruje existujúci CSS súbor a nepridajú sa žiadne iné vlastné štýly. Textové polia „hrubý font“, „kurzíva“, … atď. určujú názov štýlu pre dané formátovanie v existujúcom CSS súbore (ak neexistujú, stačí nechať prázdne pole).

Keďže sa snažíme používateľom čo najviac uľahčiť prácu, pri výbere súborov (DOCX,

XHTML, CSS) sa zapamätáva cesta a pri budúcom výbere sa automaticky nastaví aktuálny adresár na ten použitý naposledy. Táto cesta sa zapamätáva do súboru s lokálnymi nastaveniami programu, ktorý sa automaticky vytvorí v domovskom adresári.

Po vyplnení všetkých potrebných polí, sa tlačidlom „Konvertuj“ na základe vyplnených nastavení pripravia parametre, s ktorými sa spustí konzolová aplikácia. Jej štandardný aj chybový výstup sa zobrazí v novom okne. V prípade, že si neželáme spustiť konverziu, ale chceme zistiť, aké sú parametre na spustenie konzolovej aplikácie s aktuálnymi nastaveniami, tak musíme kliknúť na talčidlo „Zobraz parametre“.

4.3.2 Konzolová aplikácia

Teraz nasleduje popis programu na konverziu DOCX dokumentu na XHTML dokument. Začneme najprv veľmi stručným popisom pomocnej knižnice **JDOM** na prácu s XML dokumentami. Potom prejdeme ku samotnej konverzii a ukážeme si ako prebieha.

4.3.2.1 Knižnica JDOM

JDOM je open source knižnica optimalizovaná pre Javu určená na manipuláciu s XML dokumentami. V našom projekte budeme pracovať s týmito balíkmi obsiahnutých v **JDOM**:

- **org.jdom** - Balík obsahujúci triedy na reprezentáciu XML dokumentov. Je to napríklad trieda *Document*, *Element*, *Namespace*, *Attribute*... Už samotný názov napovedá účelu.
- **org.jdom.input** - Balík obsahujúci triedy, ktoré vytvárajú XML dokument. Najdôležitejšia trieda, ktorá je použitá aj v aplikácii je trieda *SAXBuilder*. Načúvaním prichádzajúcim SAX udalostiam vytvára zodpovedajúci dokument[5].
- **org.jdom.output** - Balík obsahujúci triedy na publikovanie dokumentu. Trieda XMLOutputter transformuje vnútornú reprezentáciu dokumentu na nejaký výstup. My ju budeme používať na XHTML výstup a keďže ako jediná bola pre potreby aplikácie upravená, ešte o nej budeme písať.

Na manipuláciu s XML dokumentom budeme používať jednoduché mechanizmy. Ukážeme si ich na príklade 4.1. V prvej časti vidíme vytvorenie jednoduchého dokumentu s koreňovým elementom **root**, ktorý má jedného potomka **child**. V druhej časti postupne

[5]Viac informácií o SAX metóde prechádzania XML dokumentom je možné získať prečítaním [sax09].

Príklad 4.1 Parsovanie dokumentu

```
// vytvaranie dokumentu
Document doc = new Document();
Element e = new Element("root");
e.addContent(new Element("child"));
doc.addContent(e);

// citanie dokumentu
Element root = doc.getRootElement();
Element child = root.getChild("child");
// ...
List listElements = child.getChildren();
```

prečítame tieto elementy. Predpokladáme, že na mieste označenom tromi bodkami pridáme do elementu **child** niekoľko potomkov. Potom v poslednom riadku všetkých týchto potomkov prečítame do List štruktúry. List môžeme prechádzať klasickým iterátorom.

Ak má čitateľ záujem o bližšie informácie o **JDOM** knižnici, vhodné doplnkové čítanie je [Hun02] a [Hun07].

4.3.2.2 Priebeh konverzie DOCX na XHTML

Keďže je program na to aby sme si ho podrobne opísali až príliš rozsiahly, ukážeme si len základnú myšlienku algoritmu a podrobnejšie si prejdeme len dôležitejšie a programátorsky zaujímavejšie miesta. Konverzia začína tým, že sa vytvorí inštancia hlavnej triedy **OOXMLconvert**. Tá potom zabezpečí vykonanie týchto krokov:

1. vytvorí inštanciu triedy **OOXMLparams**, ktorá spracuje vstupné parametre - stručný popis jednotlivých vstupných parametrov vidíme v tabuľke 4.1
2. rozbalí obsah DOCX dokumentu do dočasného adresára
3. vytvorí inštanciu triedy **OOXMLpreserve**, ktorá obsahuje informácie o tom, aké formátovanie sa na základe zvolenej úrovne prenáša do výsledného XHTML dokumentu
4. vytvorí inštanciu triedy **OOXMLrels**, ktorá prečíta vzťahy v dokumente
5. vytvorí inštanciu triedy **OOXMLtheme**, ktorá prečíta definíciu tém v dokumente
6. vytvorí inštanciu triedy **OOXMLstyles**, ktorá prečíta definíciu štýlov v dokumente
7. vytvorí inštanciu triedy **OOXMLnumberings**, ktorá prečíta definíciu zoznamov v dokumente
8. vytvorí inštanciu triedy **OOXMLnotes**, ktorá prečíta definíciu poznámok pod čiarou a na konci dokumentu

Tabuľka 4.1: Zoznam vstupných parametrov konzolovej aplikácie

param	popis	hodnota
-i	vstupný DOCX subor na skonvertovanie	súbor
-o	výstupný XHTML súbor	súbor
-co	relatívna cesta od XHTML súboru k adresáru, kde sa uložia CSS	súbor
-io	relatívna cesta od XHTML súboru k adresáru, kde sa uložia obrázky	súbor
-fl	úroveň zachovania formátovania	0 - žiadne formátovanie 1 - základné formátovanie 2 - rozšírené formátovanie
-pi	určuje, či sa majú prenášať obrázky	(y/n)
-jq	kvalita jpeg kompresie použitej na obrázkoch	(od 1 do 100)
-hh	určuje, či sa majú heuristikou rozoznávať nadpisy	(y/n)
-hs1	názov štýlu nadpisu úrovne 1	názov štýlu
-hs2	názov štýlu nadpisu úrovne 2	názov štýlu
-hs3	názov štýlu nadpisu úrovne 3	názov štýlu
-ht	HTML tag pre nadpis rozpoznaný heuristikou	názov tagu
-pf	určuje, či sa prenášajú poznámky pod čiarou	(y/n)
-pe	určuje, či sa prenášajú poznámky na konci dokumentu	(y/n)
-ct	typ prepojenia na CSS súbor	0 - žiadne prepojenie 1 - štýly sa pridajú do existujúceho CSS 2 - prepojiť na CSS a nevytvoriť nové
-ci	vstupný CSS súbor	súbor
-cssb	určuje CSS štýl pre hrubý font	názov štýlu
-cssi	určuje CSS štýl pre kurzívu	názov štýlu
-cssu	určuje CSS štýl pre podčiarknutie	názov štýlu
-csss	určuje CSS štýl pre prečiarknutie	názov štýlu
-cssui	určuje CSS štýl pre horný index	názov štýlu
-cssli	určuje CSS štýl pre dolný index	názov štýlu

9. vytvorí inštanciu triedy **OOXMLcomments**, ktorá prečíta definíciu komentárov v dokumente
10. preparsuje hlavný XML dokument pomocou triedy **OOXMLparser**
11. vzápätí získaný XHTML výstup zapíše do súboru pomocou triedy **XHTMLoutput**
12. vytvorené CSS štýly pomocou triedy **CSSoutput** zapíše do súboru

Základ celej aplikácie tvorí trieda OOXMLparser, preto si o tejto triede bližšie povieme v samostatnej časti.

4.3.2.3 Trieda OOXMLparser

Hlavná metóda triedy **OOXMLparser** je *parse*, ktorá prejde daným dokumentom a vráti XHTML dokument. Pri navrhovaní tejto metódy existovali dve verzie. Prvá verzia, pracovne nazvaná rekurzívna, pre každý potomok aktuálneho elementu zavolala rekurzívne metódu *parse* a všetky elementy sa teda spracovávali v jednej centrálnej metóde. Výhoda tohoto prístupu boli menej prísne požiadavky na štruktúru dokumentu, keďže si spomínaná metóda vedela dať rady aj so zle umiestneným elementom. Nevýhoda bola neprehľadnosť a tým aj vyššia náchylnosť na chyby. Druhá verzia, pracovne nazvaná lineárna, pre každý typ elementu zavolá metódu, ktorá daný element spracuje. Na názov tejto metódy existuje konvencia a síce, že má tvar *parseElementXXX*, kde XXX je názov daného elementu. Kvôli väčšej prehľadnosti sú elementy obrázka a tabuľky spracovávané v samostatných triedach **ElementPicture** a **ElementTable**. Pri tomto prístupe je dopredu dané aké elementy v danom elemente očakávame a to sa nakoniec ukázalo aj ako chcené. Preto je do aplikácie zapracovaná táto druhá verzia.

Príklad 4.2 Použitie rPr premennej triedy OOXMLparser

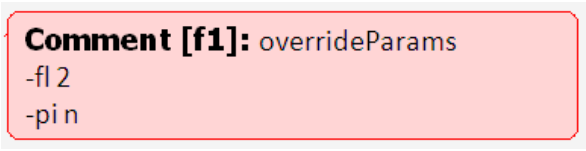
```
fragment z metódy parseElementRPr:
...
if (rPrChild.getName().trim().equals("i")) {
    rPr.put("i", rPrChild.getAttributeValue("val", Global.wns, "on"));
}
...

fragmenty z metódy parseElementR:
...
String stylesClasses = "";
if (Global.ooxmlPreserve.italic && rPr.containsKey("i") && rPr.get("i").equals("on")) {
    stylesClasses += " " + Global.ooxmlParams.cssI;
    Global.cssOutput.addClass("." + Global.ooxmlParams.cssI,"font-style: italic;");
}
...
runEl.setAttribute("class", stylesClasses);
paragraphEl.addContent((Element)runEl.clone());
...
```

V tejto triede sú veľmi dôležité premenné *rPr* a *pPr*. Obe triedy sú dátového typu *LinkedHashMap<String,String>*. Ide vlastne o mapovanie vlastností behu, resp. odstavca. Ako sa tieto premenné používajú si ukážeme na príklade 4.2. V prvom fragmente vidíme zisťovanie, aký element obsahuje **rPr** element. Ak ide o **i** element (kurzíva), tak do *rPr* pridáme o tom záznam. Metóda getAttributeValue vráti hodnotu atribútu *val*. Global.wns je definovaný menný priestor, v ktorom je daný atribút. Menné priestory sú definované v triede Global (nachádzajú sa v nej všetky premenné, konštatny a inštancie tried potrebné

pre celý systém). V druhom fragmente vidíme výsek z metódy parseElementR. Výsek je z časti, keď sme už zanalyzovali obsah tohoto elementu a teraz mu nastavujeme štýly. V zobrazenej podmienke najprv zistíme, či formátovanie kurzívou prenášame do výsledného dokumentu, potom či dané formátovanie aktuálny beh obsahuje a keď toto spĺňa tak do reťazca *stylesClasses* pridáme názov triedy v CSS pre kurzívu a do výsledných CSS štýlov pridáme štýl na písmo kurzívou. Nakoniec aktuálnej XHTML reprezentácii behu (*runEl*) nastavíme atribút *class* na triedy pridané do *stylesClasses*. Potom pridáme klon *runEl* do XHTML reprezentácie aktuálneho odstavca.

Teraz si ukážeme ako vlastne funguje spomínaná heuristika pri určovaní nadpisov v dokumente. Nadpisom bude odstavec, v ktorom je jediný beh s textom napísaným hrubým fontom a veľkosť fontu je väčšia ako vo zvyšku dokumentu. Obsah tohoto odstavca sa pridá do XHTML elementu pre nadpis[6].

Ďalšia oblasť, ktorá si zaslúži pozoronosť je riešenie problému zo špecifikácie, kde sme chceli aby sa dali parametre konverzie dočasne meniť. Slúžiť na to majú komentáre s presne definovanou štruktúrou. Úsek, na ktorom chceme dočasne použiť iné parametre konverzie označíme komentárom, ktorý má na prvom riadku text „overrideParams“. V každom ďalšom riadku komentára môžeme nastaviť parametre konverzie, ktoré sa dočasne zmenia, so syntaxou takou istou ako pri parametroch na vstupe. Každý jeden parameter treba dať na nový riadok. Výskyt takéhoto komentára v dokumente spôsobí, že sa v triede OOXMLparams vytvorí záloha aktuálnych parametrov a parametre nastavené v tomto komentári sa prepíšu. Keď komentár skončí, tak sa obnovia parametre zo zálohy. Príklad na obrázku



Obr. 4.4: Komentár dočasne meniaci parametre konverzie dokumentu

4.4 ukazuje ako by vyzeral komentár, ktorým chceme dočasne zmeniť parameter určujúci úroveň zachovania formátovania na hodnotu „2“ (rozšírené formátovanie) a paramter určujúci či chceme prenášať obrázky do výsledného dokumentu na hodnotu „n“ (neprenášať obrázky).

[6] Úroveň tohoto nadpisu sme nastavili vstupnými parametrami.

4.3.2.4 Trieda ElementTable

Ďalšia trieda, ktorú je dôležité spomenúť je trieda **ElementTable**. Konštruktor tejto triedy dostane ako parameter XML element obsahujúci tabuľku v dokumente. Verejnou metódou *getXHTMLTable* trieda preparsuje tabuľku a vráti XHTML výstup. Pri vytváraní tabuľky však narážame na niekoľko problémov.

Prvý z nich je fakt, že bunka tabuľky môže obsahovať akékoľvek elementy a teda napríklad aj ďalšiu tabuľku. Riešenie tohoto problému je relatívne jednoduché, keďže stačí upraviť metódu *parse* v triede **OOXMLparser**, aby zvládala preparsovať nielen celý dokument, ale aj vnorený element. Keď budeme parsovať obsah bunky, tak vytvoríme novú inštanciu triedy **OOXMLparser** a spustíme metódu *parse*, kde ako parameter pošleme element obsahujúci telo bunky.

Ďalším problémom je fakt, že spôsob spájania buniek funguje vo WordprocessingML inak ako v XHTML. Existujú na to dva spôsoby. V XHTML dokumente sa bunky spájajú atribútmi *colspan* a *rowspan*, ktoré fungujú na tom istom princípe ako spájanie buniek v DOCX dokumente pomocou elementov **rowSpan** a **gridSpan**[7]. Avšak pri parsovaní buniek spojených elementom **vMerge** resp. **hMerge**, ak neoznačujú začínajúci spojený úsek, tak potom tieto bunky do XHTML dokumentu neprenášame, len inkrementujeme *colspan* resp. *rowspan* atribút. Práve na túto inkrementáciu si pri parsovaní musíme pamätať aktuálny stĺpec a riadok. Navyše si musíme pamätať pre každý stĺpec, či v ňom náhodou nie je začatý spojený úsek. Pre riadky si to pamätať nemusíme, keďže parsovanie prebieha po riadkoch a teda nám stačí iba informácia o tom, či v aktuálnom riadku nie je začatý spojený úsek.

Obr. 4.5: Tabuľka so spojenými bunkami

Uvažujme príklad na obrázku 4.5. Keď na spojenie buniek v prvom stĺpci použijeme

[7] Bližšie informácie o spájaní buniek je možné nájsť v časti 2.5.4.

rowSpan a na spojenie buniek v poslednom stĺpci použijeme **vMerge**, tak vnútorná reprezentácia elementami **tc** je zobrazená červenými bunkami. V tomto prípade by nám nefungovalo jednoduché inkrementačné[8] počítadlo pre aktuálny stĺpec. Keď sa parsovaním budeme nachádzať v bunke v poslednom stĺpci, v druhom riadku, tak toto počítadlo by malo hodnotu 3, pritom by malo mať hodnotu 4. My však musíme zistiť v akom stĺpci sme, aby sme mohli inkrementovať atribút *rowspan* bunky v poslednom stĺpci v prvom riadku na XHTML výstupe. Preto si naimplementujeme počítadlo, ktoré bude vedieť správne narábať s číslom stĺpcov. Využijeme pri tom pomocné pole definované takto: `boolean mergedCells[][]`. Pre každú bunku v riadku *r* a v stĺpci *s* nastavíme hodnotu `mergedCells[s][r]` na *true* ak je táto bunka v spojenej oblasti buniek metódou **rowSpan** alebo **gridSpan**, ale nie je prvá a na *false* inak. Hodnoty `mergedCells[][]` nastavíme najprv všetky na *false* a pri spracovaní **rowSpan** alebo **gridSpan** patričné bunky nastavíme na *true*. V príklade 4.3 vidíme telo funkcie ktorá vracia poradie stĺpca

Príklad 4.3 Metóda getNextCol()

```
private int getNextCol() {
    ...
    col++;
    while ((col<cCols) && (mergedCells[col][row])) {
        col++;
    }
    if (col >= cCols) {
        // sme v novom riadku
        col = 0;
        row++;
        if (row >= cRows) {
            throw new Exception("V tabuľke sa nenachádza ďalší sĺpec!");
        }
    }
    return col;
    ...
}
```

ďalšej bunky v poradí. *col* a *row* označujú aktuálne spracovávanú bunku, *cCols* a *cRows* označujú počet stĺpcov, resp. počet riadkov v tabuľke.

4.3.2.5 Trieda XHTMLoutput

Trieda XHTMLoutput má na starosti prípravu XHTML výstupu. Trieda pracuje s už predpripraveným *body* elementom, ktorý je vlastne výstupom preparsovania dokumentu. Tento element sa ešte rôznymi spôsobmi upravuje. V prvom rade sa spoja elementy **span**,

[8] Počítadlo, ktoré by sa pre každú novú bunku v stĺpci zväčšilo o jedna.

ktoré sú v dokumente hneď vedľa seba a majú rovnaké CSS štýly. Ďalšou úpravou, ktorou dosiahneme zníženie veľkosti výstupu je vloženie obsahu **span** elementu do rodičovského elementu, ak je to možné. Napríklad môžme vložiť obsah **span** elementu do rodičovského elementu **p**, ak **p** neobsahuje žiadny iný element.

Ďalšia dôležitá funkcia triedy XHTMLoutput je transformácia objektovej reprezentácie dokumentu na textovú, ktorú môžme zapísať do súboru. Na to slúži trieda **XMLOutputter** z knižnice **JDOM**. Táto trieda obsahuje niekoľko predpripravených možností formátovania výsledného textu (odsadenie elementov, nové riadky a podobne). Môžme si nastaviť, či chceme aby bol vo výslednom dokumente každý vnorený element odsadený a na novom riadku, alebo chceme aby bol celý dokument v jednom riadku. My by sme ale potrebovali niečo medzi tým. Keby sme mali celý dokument v jednom riadku, tak by v ňom boli dodatočné úpravy veľmi zložité. Keby sme však mali dokument pekne formátovaný, tak zase na niektorých miestach vo výslednom dokumente by boli nadbytočné medzery. Napríklad XHTML kód v príklade 4.4 by sa v prehliadači zobrazil namiesto súvislého textu „Popocatepetl“ s rôznou farbou textu, nesúvislý text s medzerami medzi jednotlivými fragmentmi slov. Je to preto, lebo nový riadok berie prehliadač ako znak a interpretuje ho ako medzeru.

Príklad 4.4 Príklad XHTML dokumentu

```
...
    <p>
        <span style="color: red;">Popo</span>
        <span style="color: green;">cate</span>
        <span style="color: blue;">petl</span>
    </p>
...
```

Chceme preto upraviť výstup tak, aby elementy boli odsadené, ale nech sa obsah elementu **p** vypíše do riadku. Na tento účel musíme upraviť triedu **XMLOutputter** a pridať podmienku, že keď sa vnoríme do elementu **p**, tak prestaneme výstup formátovať.

4.3.3 Problémy pri konverzii

V tejto časti si povieme o niektorých problémoch, na ktoré sme narazili počas konverzie dokumentu. Nie každá štruktúra v DOCX dokumente má ekvivalent v XHTML dokumente a preto konvertovaný súbor nebude nikdy úplne rovnaký ako jeho DOCX predloha.

Prvým príkladom takejto nekompatibility sú odrážky. Napríklad v DOCX dokumente

sa dá nastaviť začiatok číslovania na určitú hodnotu. Toto by sa dalo robiť aj v XHTML dokumente atribútom *start* v elemente **ol**. Bohužiaľ takúto možnosť nemáme so Strict DTD. Ďalším problémom je fakt, že v DOCX dokumente môže prvá odrážka začať v podstate v akejkoľvek úrovni, naproti tomu v XHTML dokumente, musí byť prvá odrážka v prvej úrovni. Nakoniec je problém aj v samotných odrážkach. WordprocessingML definuje odrážky ako text. Nastaví sa font a znak, ktorý sa použije ako odrážka. Vďaka fontom ako je napríklad *Symbol*, je na výber mnoho zaujímavých znakov ako fajky, šípky a mnoho iných. XHTML dokumenty, resp. CSS štýly takéto nastavenia odrážok nepodporujú.

Ďalším problémom sú obrázky. V DOCX dokumentoch sa im dá nastaviť, že text okolo nich ich má obtekať a tiež, že sa majú posunúť o niekoľko pixelov napríklad smerom doprava. Text potom bude obrázok obtekať na novej súradnici. V XHTML a CSS to ale funguje tak, že keď obrázok posunieme relatívne k jeho pôvodnej polohe, tak sa síce posunie, ale text ho obteká na pôvodných súradniciach.

Posledný problém, o ktorom si povieme, sú tabulátory. Tabulátor nemá v XHTML dokumente obdobu. Faktom však je, že mnoho dokumentov sa vytvára práve ich použitím. Tabulátory možnosťami pripomínajú tabuľku, preto v budúcnosti, keď sa tento problém bude riešiť v programe Office Open XML convert, tak to bude pravdepodobne cestou transformácie tabulátorového rozvrhnutia dokumentu na tabuľkové.

Problémov, ktoré spôsobujú nekompatibilitu DOCX dokumentov a XHTML dokumentov je samozrejme viac, ale vyššie spomenuté problémy sú z nich najčastejšie. Ich riešenie je však už nad rámec tejto diplomovej práce.

Kapitola 5

Výsledky

V tejto kapitole sa budeme venovať výsledkom, ktoré aplikácia Office Open XML convert (ďalej ho budeme označovať skratkou OOXML convert) dosahuje. Tieto výsledky budeme porovnávať s výsledkami konvertorov spomínaných v kapitole 3. Kapitola bude rozdelená na niekoľko častí. V prvej časti si opíšeme okolnosti, za akých sme dokumenty konvertovali. To znamená hlavne ako sme dokumenty vyberali, aké nastavenia sme použili, aké kritéria budeme sledovať a podobne. V ďalšej časti si porovnáme výsledky konverzie jednotlivých riešení.

5.1 Príprava testovania konverzie

Testovanie konverzie bude prebiehať na platforme Windows a výstupné XHTML resp. HTML dokumenty sa budú otvárať v prehliadačoch Mozilla Firefox 3, Opera 9 a Internet Explorer 7. Počas testovania výstupov konverzie jednotlivých dokumentov budeme sledovať tie isté vlastnosti ako v časti 3.6, teda hlavne verný prenos formátovania, zachovanie štruktúry, prehľadnosť kódu a s tým súvisiaca veľkosť výsledného dokumentu. Pripravili sme si niekoľko DOCX dokumentov, na ktorých budeme konvertory testovať. Výber sme sa snažili spraviť tak, aby bola zachovaná najväčšia možná objektívnosť testovania. Preto okrem jedného dokumentu, ktorý bol vytvorený za účelom demonštrácie sily programu OOXML convert, sú všetky dokumenty z rôznych nezávislých zdrojov na internete a podobne.

5.2 Testovanie konverzie

V tejto časti si prejdeme jednotlivé DOCX dokumenty a ich skonvertované HTML resp. XHTML ekvivalenty. Najprv začneme vlastnosťami XHTML výstupu programu OOXML convert. Ak sme narazili na nejaké problémy, tak si ich detailnejšie zanalyzujeme. Pri ostatných riešeniach si len poukážeme na klady a zápory. Program OOXML convert, keď nie je povedané inak, budeme spúšťať s tromi rôznymi verziami vstupných parametrov. Prvá verzia je nastavená tak, aby bola zachovaná maximálna možná vizuálna zhoda s originálnym dokumentom. Vstupné parametre pre túto verziu sú: *-i "subor.docx" -o "subor.html" -co "styles/" -io "pics/" -fl 2 -pi y -jq 80 -hh n -hs1 "heading1" -hs2 "heading2" -hs3 "heading3" -pf y -pe y -ct 0*. Druhá verzia vstupných parametrov zachováva vo výslednom dokumente len základné formátovanie. Vstupné parametre pre túto verziu sú: *-i "subor.docx" -o "subor.html" -co "styles/" -io "pics/" -fl 1 -pi y -jq 80 -hh n -hs1 "heading1" -hs2 "heading2" -hs3 "heading3" -pf y -pe y -ct 0*. A nakoniec v poslednej verzii, kde nechceme prenášať žiadne formátovanie, spúšťame program s týmito parametrami: *-i "subor.docx" -o "subor.html" -co "styles/" -io "pics/" -fl 0 -pi y -jq 80 -hh n -hs1 "heading1" -hs2 "heading2" -hs3 "heading3" -pf n -pe n -ct 0*.

5.2.1 Vzorový dokument k OOXML convert

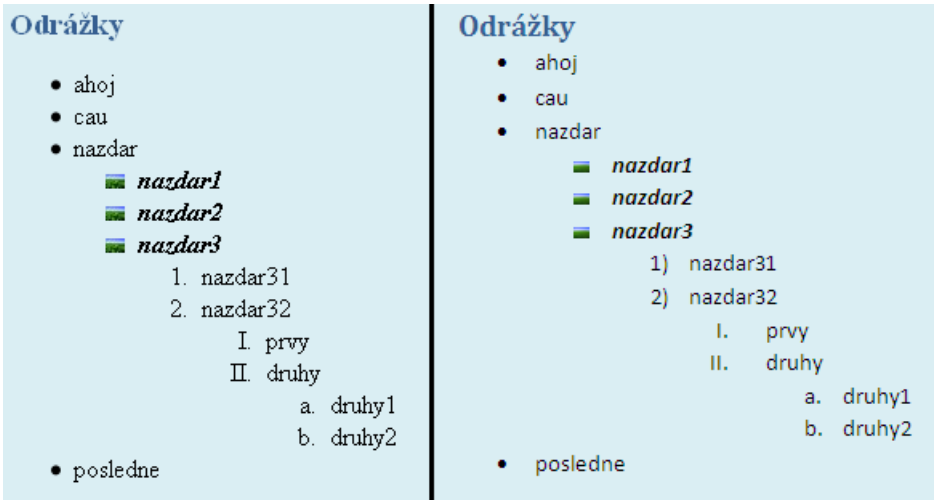
Prvým analyzovaným dokumentom bude ten, na ktorom bola demonštrovaná sila programu OOXML convert. Dokument obsahuje všetky bežné súčasti, ako sme si ich definovali v časti 4.1. Keďže sme sa už venovali tomu, ako si s týmto dokumentom poradili ostatné programy, opíšeme si len výstup programu OOXML convert a potom si všetky vlastnosti jednotlivých riešení zhrnieme do tabuľky.

OOXML convert

Keďže sa jedná o dokument, ktorý bol na mieru vytvorený práve pre tento program, nie je prekvapením, že kvalita výstupu je na vysokej úrovni. Najprv si však prejdime nastavenia programu. Program sme na vzorovom vstupe spustili celkovo trikrát, zakaždým s vyššie definovanými verziami parametrov.

Prvá verzia je pochopiteľne vo všetkým oblastiach na vysokej úrovni. Čo sa týka prenosu formátovania, tak je to, až na nejaké detaily (pozri obrázok 5.1), prakticky úplná kópia pôvodného dokumentu. Zachováva tiež štruktúru všetkých elementov. Jediné, čo by

sme tomuto výstupu mohli vytknúť sú trocha neprehľadné definície atribútov *class*, keďže každé formátovanie má samostatný štýl. Toto by mohla byť výzva do budúcna, aby sa nejakým spôsobom spájali štýly dohromady tam, kde to má zmysel (napríklad, keď sa v dokumente veľakrát vyskytuje tá istá kombinácia tried). Aj napriek tomu sa tento výstup zaradil medzi najlepšie. Ako jediný z konvertorov tento program vytvára plne validné XHTML dokumenty so Strict DTD.



Obr. 5.1: Vľavo: XHTML dokument, vpravo: pôvodný DOCX dokument.

Druhá verzia má oproti prvej tú výhodu, že má zdrojový kód prehľadnejší, ale za cenu menšej presnosti prenosu formátovania. Preto je výsledný dokument vizuálne podobný na svoj originál len základnými črtami ako je farba písma, podčiarknutie, hrubý font a podobne.

Tretia verzia nepoužíva žiadne štýly, preto sa na svoj originál podobá len štruktúrou. Táto verzia je určená tým, ktorí potrebujú zachovať štruktúru dokumentu, ale nezáleží im na formátovaní. Výsledkom je veľmi prehľadný zdrojový kód, ktorý sa ľahko upravuje.

Ostatné konvertory

Pozri kapitolu 3.

Zhrnutie

Teraz si výsledky všetkých konvertorov môžme pozrieť v tabuľke 5.1. Ide o tú istú tabuľku, akú sme si ukázali v časti 3.6, len tentokrát je v nej aj program OOXML convert. Zeleným

písmom sme označili najlepšie hodnoty v rámci stĺpca a červeným písmom najhoršie.

Tabuľka 5.1: Zhrnutie kvality XHTML resp. HTML exportov na prvom vstupnom dokumente

riešenie	**formát**	**validita**	**formátovanie**	**štruktúra**	**prehľadnosť**	**veľkosť bez obrázkov**	**celková veľkosť**
OOXML convert 1	XHTML	áno (0)	1	1	2	10.7	33.1
OOXML convert 2	XHTML	áno (0)	3	1	1	8.32	29.3
OOXML convert 3	XHTML	áno (0)	4	1	1	6.51	27.5
MS Word	XHTML	nie (372)	3	3	5	74.4	507
MS Word (filtrovaný)	HTML	nie (5)	1	3	3	21.8	418
Open Office (HTML)	HTML	nie (9)	3	1	2	12.9	74.2
Open Office (XHTML)	XHTML	nie (26)	4	1	5	52.6[1]	52.6
Firefox plugin	XHTML	nie (8)	4	5	4	29.1	87.7
DOCX Convert Office 2007	HTML	nie (18)	2	2	2	16.8	414
GMail	HTML	nie (9)	4	1	1	9.49	9.57

Vidíme teda, že najlepším konvertorom (v zachovaní formátovania) pre prvý vstup je OOXML convert s prvou verziou nastavenia parametrov. V prípade, že si ako kritérium dáme najkompaktnejšie riešenie, tak najlepším by bol opäť konvertor OOXML convert, tentokrát s treťou verziou nastavenia parametrov. Najhoršie obstáli v takmer všetkých sledovaných vlastnostiach konvertor, ktorý je súčasťou programu Microsoft Word 2007 a plugin do prehliadača Mozilla Firefox.

5.2.2 Chybný dokument

Ďalší DOCX dokument bol pôvodne vybraný na demonštráciu nedostatkov programu OOXML convert a na zistenie ako sa s týmito problémami vysporiadajú ostatné konvertory. Dokument je príloha k dokumentácii o XSLT transformácii XML dokumentu na DOCX dokument. Obsahuje niekoľko chýb, čo sa nakoniec ukázalo ako výhoda. Môžme sa teda pozrieť na to, ako si poradia jednotlivé konvertory s chybným DOCX dokumentom. Chyba spočíva napríklad v tom, že nie všetky použité štýly v dokumente sú definované

[1] V tejto veľkosti je zarátaná aj veľkosť obrázkov, kedže sú priamo v XHTML dokumente.

alebo že sa element **br** nachádza v elemente **p**, kde by sa podľa špecifikácie nachádzať nemal.

OOXML convert

Program OOXML convert nemal žiadne problémy s tým, že v dokumente sú chyby. Spomenutý zle umiestnený element **br** bol vo všetkých verziách parametrov odignorovaný, chýbajúce štýly výstup tiež výraznejšie neovplyvnili.

Tretia verzia parametrov spravila presne to, čo sa od nej očakávalo, teda zachovala len štruktúru a bez formátovania. V druhej verzii tiež nenastal žiadny problém, program pracoval podľa očakávaní. V tretej verzii už treba spomenúť jeden problém. Ide o problém spomenutý v časti 4.3.3 - tabulátory. OOXML convert nateraz tabulátory ingoruje, ale na tomto konkrétnom príklade to nie je až takým problémom. Chaos v štýloch tiež spôsobil, že tabuľka v texte nie je správne naformátovaná.

Ostatné konvertory

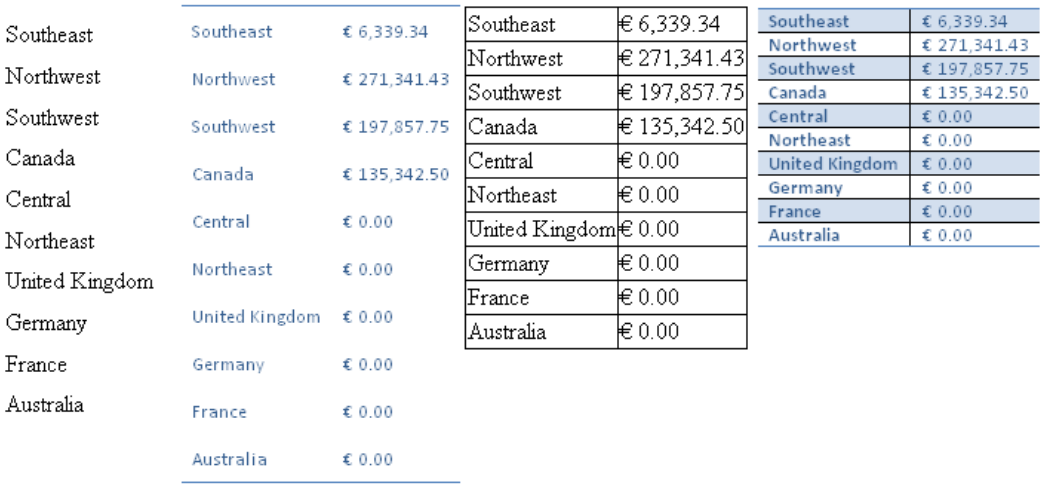
Výstup z **Open Office** a z **GMail** náhľadu sa nepodarilo získať. Open Office uvedený dokument ani neotvoril, GMail vypísal chybovú hlášku: „Príloha sa nedá zobraziť ako HTML“. Paradoxne, **DOCX Convert Office 2007**, ktorý je tiež založený na Open Office, nemal s týmto dokumentom také veľké problémy. So zachovaním formátovania a štruktúry to však bolo horšie. Časť tabuľky nebola vôbec zobrazená a chýba tiež formátovanie tabuľky. Tabulátory boli vynechané.

Plugin pre prehliadač Firefox tiež vynechal tabulátory, ale tabuľku zachoval a je tu tiež náznak pôvodného formátovania tabuľky. Potom sú tu ešte drobné chyby ako napríklad chýbajúca čiara pod menom a podobne.

Obidva výstupy z **Microsoft Office 2007** sú v zachovaní formátovania na vysokej úrovni. Tradičným problémom je veľkosť a neprehľadnosť zdrojového kódu. Na obrázku 5.2 je možné vidieť štyri rôzne interpretácie tej istej tabuľky.

Zhrnutie

Na tomto vstupe konvertory preukázali svoju schopnosť narábať s nepredvídateľnými problémami. Všetky informácie si teraz zhrnieme do tabuľky 5.2. Najhoršie a najlepšie hodnoty sú označené červenou, resp. zelenou farbou.



Southeast	Southeast	€ 6,339.34	Southeast	€ 6,339.34	Southeast	€ 6,339.34
Northwest	Northwest	€ 271,341.43	Northwest	€ 271,341.43	Northwest	€ 271,341.43
Southwest	Southwest	€ 197,857.75	Southwest	€ 197,857.75	Southwest	€ 197,857.75
Canada	Canada	€ 135,342.50	Canada	€ 135,342.50	Canada	€ 135,342.50
Central	Central	€ 0.00	Central	€ 0.00	Central	€ 0.00
Northeast	Northeast	€ 0.00	Northeast	€ 0.00	Northeast	€ 0.00
United Kingdom	United Kingdom	€ 0.00	United Kingdom	€ 0.00	United Kingdom	€ 0.00
Germany	Germany	€ 0.00	Germany	€ 0.00	Germany	€ 0.00
France	France	€ 0.00	France	€ 0.00	France	€ 0.00
Australia	Australia	€ 0.00	Australia	€ 0.00	Australia	€ 0.00

Obr. 5.2: Export tej istej tabuľky v poradí v: DOCX Convert Office 2007, plugin Firefox, OOXML convert (2. verzia parametrov), Microsoft Office 2007

Tabuľka 5.2: Zhrnutie kvality XHTML resp. HTML exportov na druhom vstupnom dokumente

riešenie	**formát**	**validita**	**formátovanie**	**štruktúra**	**prehľadnosť**	**veľkosť bez obrázkov**	**celková veľkosť**
OOXML convert 1	XHTML	áno (0)	2	1	2	23.1	38.8
OOXML convert 2	XHTML	áno (0)	3	1	1	20.0	35.7
OOXML convert 3	XHTML	áno (0)	4	1	1	17.8	33.5
MS Word	XHTML	nie (1612)	1	2	5	185	205
MS Word (filtrovaný)	HTML	nie (20)	1	2	4	106	119
Open Office (HTML)	n/a	n/a	5	5	5	0	0
Open Office (XHTML)	n/a	n/a	5	5	5	0	0
Firefox plugin	XHTML	nie (37)	2	2	4	129	136
DOCX Convert Office 2007	HTML	nie (18)	5	3	2	42.1	49
GMail	n/a	n/a	5	5	5	0	0

Z tabuľky vidíme, že najlepším konvertorom na tomto vstupe, keď berieme ako hlavné kritérium zachovanie formátovania, je Microsoft Office 2007. Hneď však treba dodať, že zdrojový kód oboch verzií je veľmi neprehľadný a preto je objektívne prvenstvo na tomto vstupe viac než diskutabilné. Keď berieme do úvahy aj prehľadnosť, tak sa potom víťazom stávajú exporty z programu OOXML convert s parametrami v prvej a druhej verzii.

Naopak najhoršie obstáli konvertory, ktoré dokument ani neotvorili a z tých čo vytvorili aspoň nejaký výstup je najhorší DOCX Convert Office 2007.

5.2.3 Veľký a neprehľadný dokument

Ďalší v poradí bude veľký a neprehľadný verejný dokument stiahnutý z internetu - výročná správa. Takéto dokumenty bývajú zväčša skúškou ohňom, keďže tvorcovia len málokedy myslia na to, aby mal dokument aj nejakú štruktúru. Konkrétne v tomto dokumente sú zaujímavosti typu odsadzovanie medzerami, vo veľkom sa využívajú už spomínané tabulátory (na niektorých miestach nie práve najvhodnejšie), druhá úroveň odrážok je tu niekedy robená tak, že je vytvorený nový zoznam a ten je viac odsadený... teda zachovanie štruktúry v tomto dokumente nebude veľmi prísne brané, keďže štruktúru ani veľmi nemá. Hlavne nás bude zaujímať ako vyzerá výstup v prehliadači a tiež možnosť úprav, teda jednoduchosť zdrojového kódu.

OOXML convert

XHTML výstup programu OOXML convert je už tradične valídny, prehľadný a hlavne ľahko upraviteľný. Čo sa týka prenesenia formátovania, tak v prvej verzii vstupných parametrov je výstup na slušnej úrovni, ale miestami zaostáva. Takým miestom sú napríklad tabuľky, alebo niektoré odrážky. Druhá verzia parametrov aj napriek tomu, že prenáša iba základné formátovanie má tiež kvalitný výstup, aj keď sa na pôvodný dokument podobá skôr len štruktúrou ako formátovaním. Nakoniec tretia verzia vyniká jednoduchosťou zdrojového kódu a je teda určená hlavne na dodatočné úpravy. Na originál sa podobá len vzdialene.

Ostatné konvertory

Microsoft Office 2007 drží prvenstvo v prenášaní formátovania. Až na nejaké detaily pri odrážkach je to naozaj verná kópia. Horšie je to so zdrojovým kódom. Štruktúry ako napríklad zoznamy sú prekladané do odsekov textu, ktoré majú také formátovanie, aby ako zoznamy vyzerali.

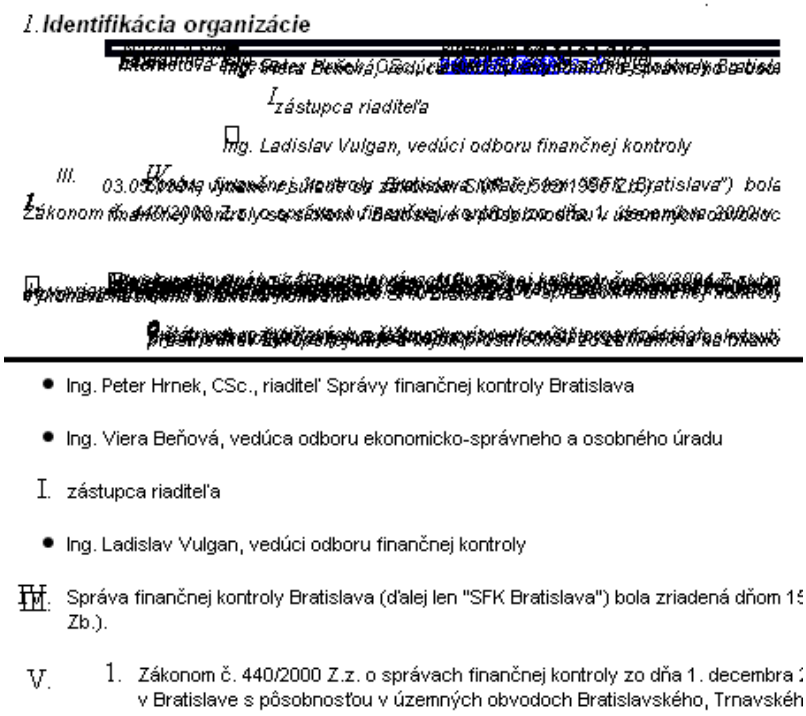
XHTML výstup z **Open Office** je v zachovávaní formátovania na veľmi nízkej úrovni. V prehliadači Opera bola vyrenderovaná spleť neznámych znakov a v ostatných sa tiež vyskytujú neznáme znaky a nepresnosti vo formátovaní. HTML verzia je na tom trocha

lepšie. Miestami je na úrovni OOXML convert, na iných miestach ale dochádza k úplnému rozbitiu štruktúry (napríklad pri niektorých tabuľkách). Zdrojový kód HTML dokumentu je však na celkom dobrej úrovni.

Plugin pre prehliadač Firefox má výstup tiež na nízkej úrovni. Nezachováva ani štruktúru. Svetlou výnikou sú tabuľky, ktoré majú formátovanie takmer ako pôvodný dokument. Zdrojový kód je tiež celkom kvalitný, kazia ho jedine CSS štýly, ktoré sú priamo v každom elemente.

GMail má celkom prehľadné formátovanie, ale na pôvodný dokument sa až tak moc nepodobá. Zachovanie štruktúry a tiež relatívne prehľadný zdrojový kód je silná stránka GMail náhľadu.

Nakoniec konvertor **DOCX Convert Office 2007**. Jeho silnejšou stránkou je relatívne dobrý prenos formátovania a tiež celkom dobré zachovávanie štruktúry. V prehľadnosti patrí k priemeru.



Obr. 5.3: Rôzne chyby v zobrazení, hore: Open Office - XHTML verzia v prehliadači Opera, dole: Open Office - HTML verzia

Zhrnutie

Tento vstup testoval schopnosť poradiť si s veľmi zlým formátovaním. V tabuľke 5.2 vidíme prehľadné výsledky jednotlivých konvertorov na tomto vstupe.

Tabuľka 5.3: Zhrnutie kvality XHTML resp. HTML exportov na treťom vstupnom dokumente

riešenie	**formát**	**validita**	**formátovanie**	**štruktúra**	**prehľadnosť**	**veľkosť bez obrázkov**	**celková veľkosť**
OOXML convert 1	XHTML	áno (0)	2	1	2	176	176
OOXML convert 2	XHTML	áno (0)	3	1	1	137	137
OOXML convert 3	XHTML	áno (0)	3	1	1	127	127
MS Word	XHTML	nie (4573)	1	3	5	699	845
MS Word (filtrovaný)	HTML	nie (8)	1	3	4	421	440
Open Office (HTML)	HTML	nie (120)	3	2	2	317	317
Open Office (XHTML)	XHTML	nie (29)	4	2	3	270	270
Firefox plugin	XHTML	nie (4)	3	3	2	662	662
DOCX Convert Office 2007	HTML	nie (201)	2	2	3	527	527
GMail	HTML	nie (251)	3	1	2	202	202

Z tabuľky vidíme, že aj na tomto vstupe pokračujú typické výsledky, keď Microsoft Office 2007 vedie v prenášaní formátovania a OOXML convert vedie v kvalite zdrojového kódu a v prenášaní štruktúry. Treba tiež spomenúť, že Microsoft Office 2007 ako jediný z konvertorov vyexportoval aj obrázok organizačnej štruktúry, ktorý bol v DOCX dokumente vložený ako objekt *Organization Chart*. Najhoršie obstál XHTML výstup z programu Open Office.

5.2.4 Bežný dokument z internetu

Posledným z testovacej sady dokumentov je dokument o Office Open XML publikovaný na internete. Ide o dokument, ktorý má bežné formátovanie, obsahuje odstavce, nejaké odkazy, nadpisy, zoznamy a podobne. Tento dokument má za úlohu overiť použitie týchto konvertorov v bežných úlohách.

OOXML convert

Vzhľadom na to, že ide o relatívne jednoduché formátovanie, rozdiel medzi jednotlivými verziami vstupných parametrov nie sú až také veľké, ako na iných príkladoch. Aj keď je zachovávanie formátovania veľmi kvalitné, problémom sú opäť tabuľky, pretože sa im zobrazujú okraje, čo by sa však nemali.

Ostatné konvertory

Microsoft Office 2007 má obe verzie výstupov takmer úplne totožné. Výnimočne je aj zdrojový kód, na pomery tejto aplikácie, vcelku prehľadný. Celkom dobré zachovanie formátovania má okrem aplikácie Microsoft Office 2007 aj **GMail náhľad**. Ostatné riešenia v tomto mierne zaostávajú.

Zhrnutie

Keďže ide o veľmi jednoduchý dokument, budeme ho hodnotiť prísnejšie. V tabuľke 5.3 sú zhrnuté vlastnosti jednotlivých konvertorov na tomto vstupe.

Tabuľka 5.4: Zhrnutie kvality XHTML resp. HTML exportov na štvrtom vstupnom dokumente

riešenie	**formát**	**validita**	**formátovanie**	**štruktúra**	**prehľadnosť**	**veľkosť bez obrázkov**	**celková veľkosť**
OOXML convert 1	XHTML	áno (0)	2	1	2	112	150
OOXML convert 2	XHTML	áno (0)	2	1	1	83.3	121
OOXML convert 3	XHTML	áno (0)	3	1	1	76.7	114
MS Word	XHTML	nie (1047)	1	2	4	266	289
MS Word (filtrovaný)	HTML	nie (27)	1	2	3	152	175
Open Office (HTML)	HTML	nie (26)	2	1	2	148	174
Open Office (XHTML)	XHTML	nie (37)	4	4	4	142	142
Firefox plugin	XHTML	nie (51)	3	2	3	248	271
DOCX Convert Office 2007	HTML	nie (13)	2	1	2	170	193
GMail	HTML	nie (170)	3	1	2	116	116

Opäť sme dostali podobné výsledky ako doposiaľ. Na prvých priečkach Microsoft Word

2007 a OOXML convert, každý ale v inej kategórii a najhoršie opäť dopadol Open Office s XHTML verziou.

5.3 Vyhodnotenie výsledkov

Teraz si zhrnieme doterajšie výsledky v jednej tabuľke. Hodnotenia od 1 do 5 sprieme-rujeme aritmetickým priemerom. Veľkosti súborov a počet chýb z validácie sčítame (bez tých výsledkov, kde sa niektoré konvertory ani nespustili). Takto dostaneme tabuľku 5.5.

Tabuľka 5.5: Zhrnutie kvality XHTML resp. HTML exportov na všetkých vstupných dokumentoch

riešenie	**formát**	**počet chýb**	**formátovanie**	**štruktúra**	**prehľadnosť**	**veľkosť bez obrázkov**	**celková veľkosť**
OOXML convert 1	XHTML	0	1.75	1	2	299	359
OOXML convert 2	XHTML	0	2.75	1	1	229	287
OOXML convert 3	XHTML	0	3.5	1	1	210	269
MS Word	XHTML	5992	1.5	2.5	4.75	1039	1641
MS Word (filtrovaný)	HTML	40	1	2.5	3.5	594.8	1033
Open Office (HTML)	HTML	155	3.25	2.25	2.75	478	565
Open Office (XHTML)	XHTML	92	4.25	3	4.25	465	465
Firefox plugin	XHTML	63	3	3	3.25	939	1021
DOCX Convert Office 2007	HTML	232	2.75	2	2.25	714	1134
GMail	HTML	430	3.75	2	2.5	328	328

Celkové poradie sa určiť nedá, pretože v každej oblasti je iný víťaz. V prenášaní formátovania je najlepší Microsoft Word 2007, v kvalite výstupného zdrojového kódu dokumentu je zase najlepší OOXML convert. OOXML convert zároveň celkom dobre prenáša formátovanie (je druhý v poradí v tomto kritériu), preto sa zdá, že by mohol byť z uvedených riešení objektívne najlepší.

Prednosti programu **OOXML convert** sú okrem valídneho kvalitného výstupu aj variabilita nastavení a multiplatformové použitie. Naopak nevýhody sú horšia podpora formátovania tabuliek a nepodporovanie tabulátorov.

Hlavná výhoda použitia **Microsoft Office 2007** je kvalitné zachovanie formátovania

dokumentu a hlavná nevýhoda je, že tento výstupný dokument je príliš veľký a neprehľadný a je teda prakticky nemožné ho ďalej upravovať. Nezanedbateľná nevýhoda je tiež, že táto aplikácia nie je zadarmo.

Z tohto nám teda vyplýva, že ak nám ide o striktné zachovanie formátovania, tak najlepšia voľba na konvertor z DOCX na XHTML by bola Microsoft Office 2007. V prípade, že síce chceme vernú kópiu pôvodného DOCX dokumentu, ale záleží nám aj na ďalšej použiteľnosti a prispôsobiteľnosti, tak je lepšou voľbou OOXML convert.

Záver

Cieľom tejto práce bolo vytvoriť stručnú a prehľadnú dokumentáciu k formátu Office Open XML. Táto dokumentácia by mala obsahovať dostatok informácií na to, aby na jej základe čitateľ nielen porozumel danej problematike, ale aby tiež vedel tieto informácie prakticky používať. Tento cieľ sa nám podarilo splniť v 2. kapitole. Neskôr sme popísali (spolu s príkladmi) existujúce riešenia konverzie DOCX dokumentov na XHTML dokumenty, čím sme naplnili náš ďalší cieľ - spraviť prehľad existujúcich riešení.

Nakoniec sme naprogramovali funkčnú konzolovú aplikáciu v jazyku Java slúžiacu na konverziu DOCX na XHTML a tiež grafické používateľské rozhranie. Táto aplikácia bola vzápätí podrobená testovaniu na náhodne vybraných DOCX dokumentoch spolu s ostatnými existujúcimi riešeniami. Na základe výsledkov sme spravili prehľad kvality XHTML resp. HTML exportov, v ktorom naša aplikácia vo väčšine sledovaných vlastností uspela najlepšie. Týmto sme splnili náš hlavný cieľ - naprogramovať aplikáciu na konverziu DOCX na XHTML, kde nám záleží hlavne na kvalite zdrojového kódu, ale tiež aj na zachovaní formátovania pôvodného dokumentu.

Konverzia tak komplexného formátu, akým je Office Open XML, je vysoko nad rámec jednej diplomovej práce. Preto hlavne v časti, kde sme aplikáciu testovali, uvádzame niekoľko nápadov na jej vylepšenie. Ide hlavne o rozšírenie funkcionality konverzie na nové elementy v dokumente. Využitie by mala napríklad podpora tabulátorov, viacerých stĺpcov v texte či pokročilejšie heuristiky na zjednodušenie a sprehľadnenie zdrojového kódu XHTML dokumentu. Veľký potenciál vidíme aj vo webovskej aplikácii, ktorá by slúžila ako grafické používateľské rozhranie pre našu aplikáciu.

Zoznam použitej literatúry

[Dow00] Thomas Dowling. Validating html. 2000. Available from: `http://gold.ohiolink.edu/tdowling/validation.html`.

[ECM06] ECMA. *Standard ECMA-376, Office Open XML File Formats, 1st Edition*, 2006.

[Ehr06] Erika Ehrli. Walkthrough: Word 2007 xml format. 2006. Available from: `http://msdn.microsoft.com/en-us/library/ms771890.aspx`.

[Hun02] Jason Hunter. Jdom and xml parsing. *Oracle Magazine*, 2002.

[Hun07] Jason Hunter. Jdom documentation. 2007. Available from: `http://www.jdom.org/downloads/docs.html`.

[ISO08] ISO. *International Standard ISO/IEC 29500:2008, Information technology – Document description and processing languages - Office Open XML file formats*, 2008.

[Joh05] Roger Johansson. Transitional vs. strict markup. 2005. Available from: `http://24ways.org/2005/transitional-vs-strict-markup`.

[Jon06] Brian Jones. Introduction to word documents. 2006. Available from: `http://blogs.msdn.com/brian_jones/archive/2006/02/02/523469.aspx`.

[Kos08] Jirka Kosek. Office open xml, seminář o novém formátu souborů. 2008. Available from: `http://www.kosek.cz/xml/2008ooxml/`.

[LML04] Evan Lenz, Mary McRae, and Simon St. Laurent. *Office 2003 XML: Integrating Office with the Rest of the World.* O'Reilly, 2004.

[MDHY06] Sanjay Kumar Madhva, Kulkarni D.V., Srinidhi H.S., and Pujari Y. Creating word document in office open xml format using java. 2006. Available from: `http://openxmldeveloper.org/articles/JavaWordProcessingML.aspx`.

[Mic03] Microsoft. *Overview of WordprocessingML*, 2003. Available from: `http://rep.oio.dk/Microsoft.com/officeschemas/wordprocessingml_article.htm`.

[sax09] Simple api for xml. 2009. Available from: `http://en.wikipedia.org/wiki/Simple_API_for_XML`.

[Web03] The Web Standards Project. *HTML Versus XHTML*, 2003. Available from: `http://www.webstandards.org/learn/articles/askw3c/oct2003/`.

[wik09] Office open xml. 2009. Available from: `http://en.wikipedia.org/wiki/Office_Open_XML`.

[xml07] Open source xml parsers in java. 2007. Available from: `http://java-source.net/open-source/xml-parsers`.

Prílohy

Zoznam príloh:

1. CD médium

Obsah CD média

Priložené CD má nasledovnú štruktúru:

- **vysledky** - v tomto adresári sa nachádzajú testovacie sady DOCX dokumentov a tiež HTML resp. XHTML výstupy jednotlivých konvertorov
- **JavaDoc** - v tomto adresári sa nachádza popis tried, metód a premenných použitých v aplikácii Office Open XML convert
- **diplomova praca** - v tomto adresári sa nachádza elektronická verzia tejto práce
- **OOXML convert** - v tomto adresári sa nachádza program Office Open XML convert spolu so zdrojovými súbormi