Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

# Diplomová práce

# Kontrola formálních pravidel u dokumentů vybraných textových procesorů

Plzeň 2015 Bc. Ondřej Kupilík

# Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 22. června 2015

Bc. Ondřej Kupilík

# Abstract


## Kontrola formálních pravidel u dokumentů vybraných textových procesorů

Diplomová práce se zabývá kontrolou formálních pravidel u dokumentů textových procesorů Microsoft Word, OpenOffice a LibreOffice. Cílem práce je analyzovat možnosti kontroly v dokumentech vybraných textových procesorů, definovat množinu kontrolovaných pravidel a vytvořit aplikaci, která bude množinu pravidel pro dané typy dokumentů kontrolovat. Pravidla kontrolovaná aplikací jsou definována ve vstupním XML souboru. Struktura XML souboru byla navržena v rámci diplomové práce a je popsána pomocí XSD schématu. Výstupem programu je XML soubor obsahující výsledky kontroly pravidel. Program byl otestován na sadě různých dokumentů.

## The Control of Rules in Documents of Chosen Word Processors

The diploma thesis deals with the control of rules in Microsoft Word, OpenOffice, and LibreOffice documents. The goal of the thesis is to analyse the possibilities of the control of rules in the documents of chosen word processors, to define a set of the rules, and to develop an application that will control the defined set of rules in the documents. The input of the program is an XML file that includes the rules required by the user. The structure of the XML file was designed within the framework of the thesis and the structure was described by XSD schema. The output of the application is the XML file containing the results of the control of rules. The testing of the program was performed on a set of various documents.

# Poděkování

Rád bych poděkoval vedoucímu diplomové práce Ing. Štěpánu Caisovi za cenné rady, věcné připomínky a za celkovou spolupráci při realizaci práce.

# Obsah

# 1 Úvod

Textové procesory patří mezi velmi často používané aplikace napříč různými operačními systémy. Mezi nejznámější patří Microsoft Word, OpenOffice či LibreOffice. Textové dokumenty ve formátech doc, docx nebo odt vytvořené v těchto procesorech jsou používány pro přenos či uchování textových informací a dat v pracovním, studijním i běžném životě.

Výše zmíněné textové procesory umožňují vytvářet dokumenty s různou strukturou, obsahem či formátováním. Textové dokumenty jsou často vytvářeny na základě konkrétního zadání, které přímo určuje, jak má struktura, obsah a formátování dokumentu vypadat. Příkladem může být zadání semestrální práce na vysoké škole, kde je studentům jasně stanoveno, jaké mají použít okraje stránky, jaký font s jakou velikostí má být použit v textu, zda má být použito číslování stran či jaké konkrétní kapitoly musí dokument obsahovat a podobně. Tyto formální požadavky je často nutné manuálně kontrolovat, což je časově náročné. Bylo by proto výhodné mít program, který by automatizovaným procesem dokázal tato pravidla kontrolovat.

Cílem práce je provést analýzu možností kontroly formálních pravidel u vybraných textových procesorů, dále definovat množinu pravidel, která se budou zaměřovat na obsah, strukturu a formátování dokumentu, a vytvořit aplikaci, jež bude kontrolu nadefinovaných pravidel provádět.

Prvním úkolem v rámci diplomové práce je seznámení se s možnostmi kontroly formálních pravidel. Je nutné analyzovat vlastnosti vybraných procesorů a dále na základě analýzy definovat množinu formálních pravidel, která bude výsledný program kontrolovat. Vzhledem k tomu, že vstupem aplikace má být XML soubor s definicemi jednotlivých pravidel, bude v rámci práce navržena struktura vstupního XML souboru. Jeho struktura bude popsána XSD schématem. Následně bude navržen a implementován program, jehož úkolem bude provádět kontrolu pravidel zadaných pomocí vstupního XML. Na závěr bude vytvořená aplikace otestována na sadě různých dokumentů.

Vývoj diplomové práce byl řízen pomocí webové aplikace Redmine, která je flexibilním nástrojem pro řízení projektů. Prostřednictvím tohoto nástroje probíhala velká část komunikace mezi autorem a vedoucím práce. Pro správu a verzování zdrojových kódů programu vytvářeného v rámci diplomové práce byl použit systém Subversion (SVN).

# 2 Možnosti kontroly formálních pravidel

V této kapitole bude popsáno, jaké jsou možnosti kontroly formálních pravidel v dokumentech vybraných textových procesorů. Budou představeny textové procesory, které byly pro tuto diplomovou práci vybrány, a nejpoužívanější formáty dokumentů, s nimiž tyto procesory pracují. Rovněž bude popsán výběr knihoven pro programovací jazyk Java, které umí dokumenty ve vybraných formátech zpracovat.

## 2.1 Vybrané textové procesory

Jako textové procesory jsou označovány počítačové programy, které slouží k vytváření, úpravě, formátování a tisku textových dokumentů.

Tato podkapitola představí jednotlivé textové procesory, s nimiž se v diplomové práci pracuje. Použití textových procesorů uvedených v této části bylo přímo určeno zadáním diplomové práce.

### 2.1.1 Microsoft Office Word

Jeden ze světově nejrozšířenějších textových procesorů je Microsoft Office Word (dále jen MS Word). Tento procesor je součástí kancelářského balíku Microsoft Office firmy Microsoft, který kromě Wordu obsahuje následující nástroje:

- **Excel** – tabulkový editor,
- **PowerPoint** – nástroj pro vytváření prezentací,
- **Outlook** – e-mailový klient,
- **Access** – databázový program,
- **OneNote** – nástroj pro organizaci poznámek,
- **Publisher** – aplikaci pro grafické publikování informací.

#### Historie

Historie MS Word sahá do roku 1983, kdy vznikl Word pro platformy DOS a Macintosh. První vydání Wordu (Word 1.0) pro Windows přišlo v roce 1989. Tento program měl pouze dvě nástrojové lišty. První verze celého balíku MS Office

pro operační systémy Windows vznikla v roce 1990 jako Office 1.0, který obsahoval Word 1.1, Excel 2.0 a PowerPoint 2.0. V letech 1989 až 2003 MS Word z hlediska vlastností a funkčnosti výrazně rostl. Například v roce 1995 byl vydán MS Word 1995, který již měl devět nástrojových lišt a poprvé obsahoval automatické funkce jako kontrola pravopisu či automatické opravy. Zvyšování počtu funkcí a vlastností si vybralo svou daň na verzi Word 2003, která měla 31 nástrojových lišt, 19 podoken úloh a různé kontextové, zkracovací, hierarchické a rozšiřující se nabídky. To všechno činilo pro uživatele program nepřehledný. Proto se od verze MS Word 2007 vývojáři snaží o zjednodušení ovládání produktu [1]. Po MS Word 2007 přišla ještě dvě vydání, MS Word 2010 a MS Word 2013.

Zásadní změnou mezi verzemi MS Word 2003 a 2007 byla změna formátu ukládání dokumentů. Do verze 2003 se používal formát doc. Jednalo se o balík binárních souborů, ve kterých byly uloženy všechny informace o dokumentu, například obsah nebo formátování dokumentu. Od verze 2007 začal být používám nový formát docx. Ten je na rozdíl od doc balíkem XML souborů, v nichž jsou uloženy informace, které byly dříve uchovávány v binárních souborech [2]. Detailnější popis jednotlivých formátů dokumentů je k dispozici v kapitole 2.2.

### 2.1.2 Apache OpenOffice Writer

OpenOffice Writer je součástí kancelářského balíku Apache OpenOffice. Balík vyvíjí The Apache Software Foundation a je vydáván pod Apache 2.0 Licence, což znamená, že může být zdarma používán pro jakékoliv účely, soukromé, komerční, vzdělávací i administrativní. Mezi uživateli je OpenOffice Writer široce rozšířen jako neplacená alternativa MS Word.

Kromě programu Writer balík obsahuje následující nástroje:

- **Calc** – tabulkový nástroj,
- **Impress** – program pro vytváření prezentací,
- **Draw** – grafický nástroj,
- **Base** – databázový program,
- **Math** – aplikaci pro práci s matematickými rovnicemi.

Writer podporuje OpenDocument formáty jako jsou odt, ott, oth a odm a rovněž dokumenty ve formátech MS Word doc a docx. Kromě nich je možné v programu Writer pracovat také s formáty jako xml, sxw, rtf, txt, csv a dalšími [3].

#### Historie

OpenOffice existuje od roku 2000, kdy Sun Microsystems převzala od firmy StarOffice její kancelářský balík StarOffice. První verze balíku pod názvem OpenOffice.org byla vydána v roce 2002 (OpenOffice.org 1.0). V roce 2010 Sun Microsystems převzala korporace Oracle. Někteří dosavadní vývojáři OpenOffice.org se obávali, že Oracle kancelářský balík zpoplatní a založili si vlastní společnost The Document Foundation, která vyvíjí balík LibreOffice, viz kapitola 2.1.3. V roce 2011 převzala kompletní produkt OpenOffice.org společnost Apache Software Foundation, jež balík vyvíjí pod názvem Apache OpenOffice dodnes. Nejnovější verzí kancelářského balíku je Apache OpenOffice 4.1.1 z roku 2014 [3].

### 2.1.3 LibreOffice Writer

LibreOffice Writer je textový editor, který je součástí kancelářského balíku LibreOffice. Tento balík vyvíjí společnost The Document Foundation. Stejně jako Apache OpenOffice je tento nástroj dostupný zdarma.

Vzhledem k tomu, že LibreOffice vychází z OpenOffice, obsahuje tento kancelářský balík stejné nástroje jako OpenOffice. Jediným program, který má LibreOffice navíc, je nástroj pro tvorbu grafů s názvem Charts.

LibreOffice Writer podporuje stejné formáty jako OpenOffice Writer.

#### Historie

Jak bylo zmíněno v předchozí kapitole 2.1.2, společnost The Document Foundation vznikla v roce 2010 oddělením od OpenOffice. První vydání balíku LibreOffice 3.3 proběhlo v roce 2011. Následovaly verze 3.4, 3.5 a 3.6. V roce 2013 byla vydána varianta LibreOffice 4.0, na kterou navázala aktuálně nejnovější verze 4.1 [4].

## 2.2 Formáty dokumentů

V kapitole 2.1 byly představeny textové procesory, se kterými se v diplomové práci pracuje. V této části práce budou popsány nejpoužívanější formáty textových dokumentů představených procesorů.

### 2.2.1 Formát DOC

Tento formát je spojen s verzemi MS Word 1997, 2000, 2002 a 2003. Patří do rodiny Office Binary File Formats, což znamená, že se jedná o balík binárních souborů, ve kterých jsou uchovány ve formě binárních dat informace o dokumentu. Tento způsob uložení textového dokumentu byl nahrazen ve verzi MS Word 2007 formátem docx, který používá Open XML File Formats a je popsán v kapitole 2.2.2.

Binární struktura dat znamená, že jsou vyjádřena jako skupiny hexadecimálních čísel, které interpretační program zpracuje a zobrazí přes své uživatelské rozhraní. V následujícím seznamu jsou uvedeny některé z binárních proudů (streamů) a úložišť (storage), které může dokument obsahovat [5]:

Dokument může obsahovat následující

- **WordDocument Stream** – Musí být v dokumentu pokaždé. Obsahuje text dokumentu a ostatní informace odkazované z dalších částí souboru.
- **Data Stream** – Nemá žádnou předdefinovanou strukturu. Obsahuje data odkazované v ostatních částech dokumentu.
- **Custom XML Data Storage** – Specifikuje jak uchovat kolekci XML fragmentů.
- **Summary Information Stream** – Obsahuje informace o dokumentu, například jméno autora, čas poslední úpravy a další.
- **Document Summary Information Stream** – Obsahuje informace o obsahu dokumentu, například celkový počet řádek a počet odstavců a další.
- **Macros Storage** – Jsou v něm uložena makra použitá v dokumentu.

### 2.2.2 Formát DOCX

Formát docx navazuje na doc od verze MS Word 2007. Patří do rodiny Office Open XML File Formats, která je následovníkem Office Binary File Formats. Dřívější uložení informací o dokumentu v binárních souborech bylo nahrazeno uložením do XML souborů. Soubor docx je tedy balíkem XML souborů, které obsahují všechna data popisující dokument.

Formát doc zůstal dostupný pro ukládání dokumentů i po zavedení docx, ovšem není již pro uložení dokumentů výchozím typem.

Podle [6] má Office Open XML File Formats následující přínosy:

- **Jednoduché propojení byznys informací s dokumenty** – umožňuje vytvoření dokumentů z rozličných zdrojů dat, zrychlení shromažďování dokumentů, data mining (dolování dat) a znovupoužití obsahu dokumentů.
- **Otevřenost** – Office XML Format je založen na XML a ZIP technologiích, takže je všeobecně přístupný.
- **Přenositelnost** – s XML standardem je výměna dat mezi byznys systémy a Office aplikacemi zjednodušená. Je možné změnit informace v dokumentu bez MS Office aplikace, pouze za použití standardních nástrojů pro manipulaci s XML.
- **Robustnost** – Office XML Format je navržen tak, aby byl robustnější než Binary File Format. Tím by měla být omezena možnost ztráty dat při poškození souboru.
- **Efektivita** – použití komprese pomocí ZIP technologie snižuje velikost souboru až o 75 % v porovnání s binárním formátem.
- **Bezpečnost** – dokumenty mohou být bezpečně sdíleny, protože uživatel může ze souboru odstranit citlivá data jako uživatelská jména, komentáře nebo cesty k souborům. Dále mají Word 2007, Excel 2007 a PowerPoint 2007 soubory výchozí nastavení, které určuje, že nemají spouštět vložený kód. To je výhodné například pokud uživatel dostane jeden z těchto dokumentů jako přílohu e-mailu. Nemusí se pak bát, že s jeho otevřením se spustí nějaký škodlivý program.
- **Zpětná kompatibilita** – starší typ souborů doc je plně kompatibilní s novým formátem docx.

Dokument typu docx je tedy ZIP balík, který může obsahovat následující soubory a adresáře [7], [8]:

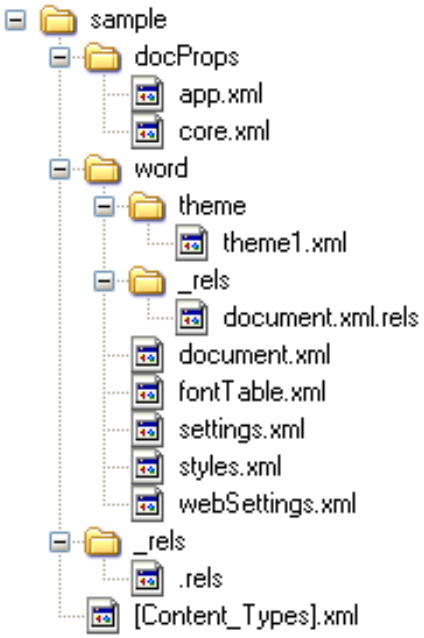
- **Soubor [Content_Types].xml** – popisuje typ obsahu každé části dokumentu. Typ hlavní části dokumentu vypadá následovně:

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

- **Adresář _rels** – jsou v něm uloženy vztahy mezi jednotlivými částmi dokumentu.
- **Soubor .rels** – popisuje vztahy ve struktuře dokumentu.
- **Adresář datastore** – obsahuje data dokumentu uložené v XML souborech.

- **Soubor item1.xml** – obsahuje určitá data použitá v dokumentu, například položky literatury.
- **Adresář docProps** – obsahuje informace o aplikaci, ve které byl dokument vytvořen.
- **Soubor App.xml** – obsahuje informace o specifických vlastnostech aplikace, ve které byl dokument vytvořen.
- **Soubor Core.xml** – obsahuje běžné informace o vlastnostech dokumentu, jako jsou čas vytvoření, jméno autora, jazyk nebo čas poslední modifikace.

Na obrázku 2.1 je struktura ukázkového dokumentu formátu docx.




Obrázek 2.1: Hierarchická struktura ukázkového DOCX dokumentu [7]

V adresáři word jsou XML soubory, které určují obsah a formátování celého dokumentu. Složka word tedy odpovídá adresáři datastore ze seznamu a může obsahovat tyto soubory [7], [8]:

- **Soubor document.xml** – je v něm uloženo tělo dokumentu. Zpravidla se jedná především o odstavce a tabulky.
- **Soubor styles.xml** – obsahuje definice všech stylů použitých v dokumentu. Definice stylu obsahuje font, jeho velikost, velikost řádkování, odkazy na příbuzné styly a další informace.

- **Soubor listDefs.xml** – obsahuje definice seznamů.
- **Soubor settings.xml** – obsahuje nastavení dokumentů.
- **Soubor header.xml** – obsahuje záhlaví dokumentu. Může jich být více, odlišují se pak číslicí na konci názvu souboru, například header1.xml, header2.xml atd.
- **Soubor footer.xml** – obsahuje zápatí dokumentu. Může jich být více, odlišují se pak číslicí na konci názvu souboru, například footer1.xml, footer2.xml atd.
- **Soubor footnotes.xml** – obsahuje poznámky pod čarou.
- **Soubor endnotes.xml** – obsahuje vysvětlivky.
- **Soubor comments.xml** – obsahuje komentáře k dokumentu.
- **Soubor fontTable.xml** – obsahuje definice všech fontů použitých v dokumentu.
- **Soubor webSettings.xml** – obsahuje nastavení pro web. Je v něm definováno, jak má být vyřešena situace, když je dokument uložen jako HTML.

Adresář může mít více částí, než je uvedeno v seznamu, nicméně položky vyjmenované v seznamu se ve složce vyskytují nejčastěji.

### document.xml

Soubor document.xml v sobě uchovává tělo dokumentu, bývá tedy zpravidla nejdůležitější, a proto zde bude v krátkosti představena jeho struktura.

Kořenovým elementem souboru je w:document, který má potomka w:body, jež reprezentuje tělo dokumentu. Tělo dokumentu obsahuje buď element w:p pro odstavec, nebo w:sectPr pro sekci, nebo w:tbl pro tabulku.

Příklad souboru document.xml pro ukázkový docx, který obsahuje pouze text „Hello World!“:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document>
  <w:body>
    <w:p w:rsidR="00681FD7" w:rsidRDefault="00CA6612">
      <w:r>
        <w:t>Hello world!</w:t>
      </w:r>
    </w:p>
```

```
    <w:sectPr w:rsidR="00681FD7" w:rsidSect="001B6127">
      <w:pgSz w:w="11906" w:h="16838"/>
      <w:pgMar w:top="1417" w:right="1417" w:bottom="1417"
      w:left="1417" w:header="708" w:footer="708" w:gutter="0"/>
      <w:cols w:space="708"/>
      <w:docGrid w:linePitch="360"/>
    </w:sectPr>
  </w:body>
</w:document>
```

Ze souboru byly odstraněny definice jmenných prostorů z elementu w:document kvůli větší přehlednosti. Text odstavce bývá rozdělen do jednoho nebo více takzvaných runs, které reprezentuje element w:r. Tyto runs obsahují textové elementy w:t. V sekci jsou nadefinovány rozměry stránky v elementu w:pgSz a okraje stránky v elementu w:pgMar.

### 2.2.3 Formát ODT

Formát pro ukládání textových dokumentů odt je součástí Open Document Format (dále jen ODF), který vyvíjí společnost Organization for the Advancement of Structured Information Standards (dále jen OASIS). ODF je souborový formát pro dokumenty vytvořené kancelářskými aplikacemi. ODF je založen na XML.

Informace o ODF dokumentu (obsah, formátování) jsou tedy uchovávány v XML souborech, které jsou uloženy v JAR archivu. JAR soubor je komprimovaný ZIP archiv, proto může uživatel použít jakýkoliv standardní nástroj pro rozbalení ZIP archivu a prohlédnout si přímo XML soubory s obsahem a formátováním ODF dokumentu.

Světově nejpoužívanějšími rozšířeními ODF jsou následující formáty [10]:

- **.odt** – textové procesory,
- **.ods** – tabulkové procesory,
- **.odp** – prezentace,
- **.odb** – databáze,
- **.odg** – grafika,
- **.odf** – matematické rovnice.

Všechna rozšíření ODF mají v JAR archivu následující strukturu [9]:

- **Soubor mimetype** – obsahuje jeden řádek textu, který určuje MIME (Multi-Purpose Internet Mail Extensions). Textový dokument odt má následující MIME typ: application/vnd.oasis.opendocument.text

- **Soubor content.xml** – obsah dokumentu.
- **Soubor styles.xml** – obsahuje informace o stylech použitých v dokumentu. Obsah a styl (prezentace) jsou odděleny do dvou různých souborů za účelem zvýšení flexibility.
- **Soubor meta.xml** – meta-informace o obsahu dokumentu, například jméno autora, čas poslední úpravy, čas vzniku atd.
- **Soubor settings.xml** – obsahuje specifické informace o aplikaci, ve které dokument vznikl. U textového dokumentu je to například stanovení, jestli mají být záhlaví a zápatí zobrazena nebo ne.
- **Adresář META-INF** – obsahuje soubor manifest.xml.
  - **Soubor manifest.xml** – jsou v něm meta-informace o JAR archivu. Obsahuje seznam všech ostatních souborů z JAR archivu. Tento soubor musí v JAR archivu být pro otevření dokumentu v OpenOffice.org. Není to manifest používaný v programovacím jazyce Java.
- **Adresář Pictures** – obsahuje obrázky použité v dokumentu. Některé textové procesory tuto složku vytvoří pouze v případě, kdy dokument obrázky skutečně obsahuje, některé ji naopak vytvoří pokaždé.
- **Adresář Thumbnails** – obsahuje PNG soubor s náhledem dokumentu.

### content.xml

Soubor content.xml v sobě uchovává obsah dokumentu, což je zpravidla ta nejdůležitější část dokumentu, a proto zde bude představena jeho struktura.

U všech typů ODF má content.xml následující strukturu:

```
<office:document-content namespace declarations
office:version="1.0"
office:class="document type">
  <office:scripts/>
  <office:font-face-decls>
    <!-- font specifications -->
  </office:font-decls>
  <office:styles>
    <office:automatic-styles>
      <!-- style information -->
    </office:automatic-styles>
  </office:styles>
```

```
  <office:body>
    <office:documentType>
      <!-- actual content here -->
    </office:documentType>
  </office:body>
</office:document-content>
```

Ukázka struktury souboru byla převzata z [9].

Kořenovým elementem je office:document, v němž jsou definovány všechny jmenné prostory, které jsou použity v dokumentu. Potomek office:scripts se v souboru vyskytuje pouze v případě, že byla v dokumentu použita makra. Potomek office:font-face-decls popisuje fonty použité v dokumentu. Stejná informace je k dispozici v souboru styles.xml. V elementu office:automatic-styles jsou definice automatických stylů, které obsahují odkazy na styly v souboru styles.xml. Poslední potomkem je element w:body, jenž je povinný a nejdůležitější, protože obsahuje tělo dokumentu [9].

U odt dokumentů je prvním potomkem body elementu office:text, který určuje, že se jedná o textový dokument. Potomky tohoto textového elementu bývají nejčastěji text:p, jenž definuje odstavec, text:h, jenž definuje záhlaví, a table:table, který definuje tabulku.

## 2.3 Knihovny pro zpracování dokumentů

Pro realizaci programu, který bude kontrolovat formální pravidla dle zadání, byl zvolen programovací jazyk Java. Bylo tedy zapotřebí najít a vybrat knihovny pro tento jazyk, které umí formáty představené v 2.2 načíst a zpracovat.

Během hledání a analyzování knihoven se podařilo nalézt jen jednu, která by byla schopná zpracovat všechny formáty, ta ovšem byla placená. Ostatní knihovny uměly pracovat vždy pouze s jedním formátem. Je to celkem pochopitelné vzhledem k rozdílům ve struktuře jednotlivých formátů. Bylo tedy nutné najít zvlášť knihovnu pro každý formát.

Při hledání knihoven jich bylo vyzkoušeno šest:

- **Apache POI** – zahrnuje dvě knihovny.
  - **HWPF** – umí zpracovat doc dokumenty.
  - **XWPF** – umí zpracovat docx dokumenty.
- **DOCX4J** – umí zpracovat docx.
- **ODFDOM** – zpracovává OpenDocument formát.
- **Aspose.Words** – dokázala by zpracovat všechny požadované formáty, ale je placená, což se pro použití v rámci diplomové práce nehodí.
- **jOpenDocument** – umí zpracovat OpenDocument formát.

Již po otestování na několika základních dokumentech bylo jasné, které knihovny budou pro tento projekt vhodné. Ty budou popsány v následujících dvou kapitolách.

### 2.3.1 Apache POI

Celý název knihovny zní Apache POI - the Java API for Microsoft Documents. Jak název naznačuje, knihovna obsahuje komponenty pro zpracování všech typů dokumentů Microsoft Office. Vyvíjí ji Apache Software Foundation [11].

Pro tuto diplomovou práci byla knihovna Apache POI vybrána, protože obsahuje dvě komponenty, které umí zpracovat dokumenty z programu MS Word. Jedná se o komponenty HWPF pro doc dokumenty a XWPF pro docx dokumenty. Výhodou knihovny je, že obě komponenty, ačkoliv každá pracuje s jiným formátem dokumentu, mají podobné API. Je tak snazší používat dvě komponenty, které mají podobná API, než používat úplně odlišné knihovny. Další výhodou knihovny byly přehledné návody pro práci s komponentami a podrobná dokumentace k API.

Nejnovější plnou verzí knihovny je Apache POI 3.11.

V následujících dvou částech budou popsány jednotlivé komponenty knihovny pro zpracování dokumentů z MS Word.

### HWPF

Základní třídy pro práci s komponentou HWPF jsou:

- **HWPFDocument** – jedná se o hlavní reprezentaci načteného dokumentu. Je součástí balíku org.apache.poi.hwpf.
- **WordExtractor** – poskytuje metody pro základní průchod textu dokumentu. Je součástí balíku org.apache.poi.hwpf.extractor. Jako parametr konstruktoru se zadává objekt HWPFDocument, ze kterého si třída získá potřebný textový obsah. Používá se pro průchod odstavců dokumentu.
- **Range** – hlavní třída objektového modelu HWPF. Je součástí balíku org.apache.poi.hwpf.usermodel. Z objektu HWPFDocument se získá pomocí metody getRange(). Dá se přes ní přistupovat k odstavcům, tabulkám, sekcím a celému obsahu dokumentu. Rovněž může být použita pro vkládání nového textu do dokumentu.

Důležitými balíky jsou org.apache.poi.hwpf.model a org.apache.poi.hwpf.usermodel, které obsahují třídy pro reprezentaci dílčích částí dokumentu. Tyto balíky obsahují například třídu Paragraph reprezentující odstavec, Section reprezentující sekci, Table reprezentující tabulku, CharacterRun reprezentující část textu odstavce či StyleSheet reprezentující styly dokumentu.

### XWPF

Základní třídy pro práci s komponentou XWPF jsou:

- **XWPFDocument** – jedná se o hlavní reprezentaci načteného dokumentu. Je součástí balíku org.apache.poi.xwpf.usermodel.
- **CTDocument1** – jde o nízkoúrovňovou reprezentaci dokumentu. Jedná se o odkaz na kořenový element souboru document.xml popsaný v 2.2.2. Objekt XWPFDocument vrací instanci této třídy metodou getDocument(). Není přímo součástí komponenty, patří do balíku org.openxmlformats.schemas.wordprocessingml.x2006.main. Příklad použití této třídy je uveden pod tímto seznamem.
- **POIXMLProperties** – obsahuje souhrnné informace o dokumentu, jako jsou počet stran nebo počet slov. Je součástí balíku org.apache.poi.

Příklad získání informace o pravém a levém okraji stránky pomocí CTDocument1:

```
CTDocument1 documentCt = xwpfDoc.getDocument();
CTBody body = documentCt.getBody();
CTSectPr section = body.getSectPr();
CTPageMar margin = section.getPgMar();
double left = margin.getLeft().doubleValue();
double right = margin.getRight().doubleValue();
```

Uvedený příklad přímo kopíruje strukturu souboru document.xml uvedenou v kapitole 2.2.2.

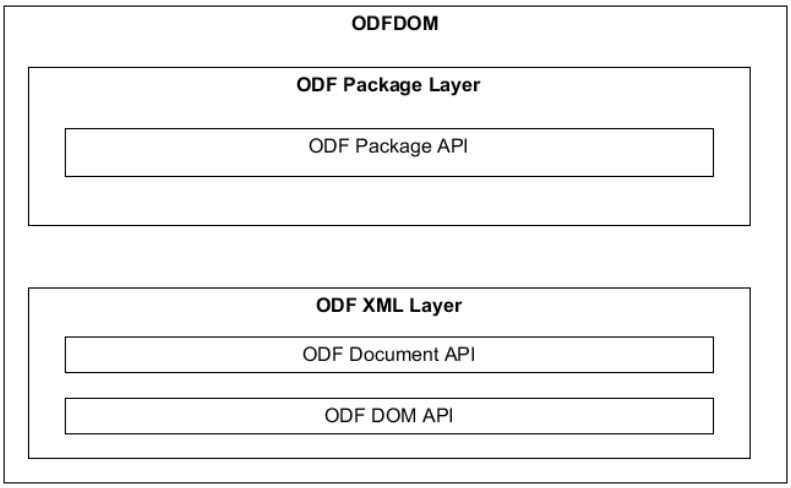
Nejdůležitějším balíkem komponenty je org.apache.poi.xwpf.usermodel. Ten obsahuje důležité třídy pro reprezentaci dílčích částí dokumentu. Jedná se například o třídu XWPFParagraph reprezentující odstavec, XWPFTable reprezentující tabulku, XWPFHeader reprezentující záhlaví či XWPFFooter reprezentující zápatí.

### 2.3.2 ODFDOM

Knihovna ODFDOM je součástí Apache ODF Toolkit a vyvíjí ji Apache Software Foundation. Existuje její vysokoúrovňová abstrakce nazvaná Simple Java API for ODF (dále jen Simple API), která bude v diplomové práci použita zároveň s ODFDOM.

Tato knihovna byla pro realizaci diplomové práci vybrána kvůli přehlednému API, dostupným vzorovým příkladům a podrobné dokumentaci.

Cílem ODFDOM je poskytnout jednoduché API pro čtení, vytváření a úpravu ODF dokumentů. Proto je API rozděleno do dvou vrstev, jak je znázorněno na obrázku 2.2.




Obrázek 2.2: Vrstvy ODFDOM knihovny

Popis jednotlivých vrstev knihovny [12]:

- **ODF Package Layer** – poskytuje přístup ke všem zdrojům v archivu ODF dokumentu, jako jsou XML soubory nebo obrázky. Jeho úkolem je rozbalit (v případě načítání) či zabalit (v případ ukládání) soubory do ZIP archivu, který reprezentuje ODF dokument.
- **ODF XML Layer** – poskytuje přístup ke všem vlastnostem dokumentu, jako je formátování, obsah atd.
  - **ODF DOM API** – nízkoúrovňový přístup. Pracuje přímo s XML soubory z ODF archivu pomocí W3C DOM API (Document Object Model). Pro každý XML element a atribut, který se v ODF archivu vyskytuje, je definována třída. Toto API kompletně pokrývá celou ODF specifikaci.
  - **ODF Document API** – vysokoúrovňový přístup. Jeho použití by mělo být přehlednější, protože se snaží oddělit uživatele od detailů XML implementace, se kterými se přímo pracuje v ODF DOM API.

#### Simple API

Tato komponenta je vysokoúrovňovou abstrakcí knihovny ODFDOM. Nahrazuje vrstvu ODF Document API, má širší možnosti využití. Její výhodou je jednoduché použití na základních operacích s dokumentem. Jelikož jde o abstrakci ODFDOM, je možné používat obě komponenty společně. Tam kde vysokoúrovňový přístup Simple API nestačí, dá se použít nízkoúrovňový přístup ODFDOM. Díky tomu je knihovna ODFDOM v kombinaci s komponentou Simple API velice užitečným nástrojem, s jehož pomocí se dá získat téměř jakákoliv informace o dokumentu nebo provést téměř jakákoliv jeho úprava.

Základní třídy pro práci s knihovnou Simple API jsou:

- **TextDocument** – jedná se o základní reprezentaci odt dokumentu.
- **DocumentStatistic** – uchovává informace o dokumentu, jako jsou počet stran nebo počet slov.
- **OdfContentDom** – tato třída patří do ODF DOM API a reprezentuje stromovou strukturu souboru content.xml vytvořenou pomocí DOM. Objekt třídy TextDocument vrací instanci této třídy metodou getContentDom(). Používá se v situacích, kdy je potřeba použít nízkoúrovňový přístup.
- **OdfStylesDom** – tato třída patří do ODF DOM API a reprezentuje stromovou strukturu souboru styles.xml vytvořenou pomocí DOM. Objekt třídy TextDocument vrací instanci této třídy metodou getStylesDom().

Pro použití knihovny Simple API je zapotřebí naimportovat následující tři JAR archivy:

- **Simple ODF 0.6.6** – balík se Simple API.
- **ODFDOM 0.8.7** – balík s ODFDOM.
- **Apache Xerces 2.9.1 nebo vyšší** – balík s knihovnou Xerces, která umí zpracovávat XML dokumenty.

# 3 Množina kontrolovaných pravidel

V této kapitole bude představena množina formálních pravidel, která bude výsledný program kontrolovat. Formální pravidla jsou taková pravidla, která se zabývají kontrolou obsahu, struktury a formátování dokumentu.

Sestavení množiny kontrolovaných pravidel předcházela důkladná analýza knihoven vybraných v kapitole 2.3. Množina pravidel totiž musí být pro všechny tři formáty dokumentů shodná. Bylo tedy zapotřebí zanalyzovat, jaké vlastnosti dokumentu se dají jednotlivými knihovnami zjistit a otestovat. Výsledkem analýzy a diskuze s vedoucím diplomové práce o jeho požadavcích ohledně kontroly pravidel bylo sestavení finální množiny s jedenácti formálními pravidly, která budou v této kapitole společně s jejich parametry představena. Rovněž bude popsáno, jak se budou pravidla pomocí jednotlivých knihoven kontrolovat. U některých pravidel bude popis kontroly doplněn ukázkovým úsekem kódu. U pravidel, kde ukázka kódu chybí, je důvodem úspora místa, protože se jednalo o delší části kódu.

## 3.1 Počet slov

Pravidlo bude kontrolovat počet slov v celém dokumentu.

#### Parametr limit

Celočíselný údaj, který je jediným parametrem tohoto pravidla. V rámci kontroly pravidla bude porovnán počet slov v dokumentu se zadaným limitem. Uživatel bude programem informován, zdali došlo k překročení limitu či byl splněn. Parametr musí být zadán právě jednou.

#### Kontrola u DOC

Z hlavní reprezentace doc dokumentu, objektu třídy HWPFDocument, se získá instance třídy SummaryInformation, která má v sobě uchovanou informaci o počtu slov. Část kódu, jež zjišťuje počet slov dokumentu, je uvedena zde:

```
SummaryInformation summary = hwpfDoc.getSummaryInformation();
int wordCount = summary.getWordCount();
```

#### Kontrola u DOCX

Z hlavní reprezentace docx dokumentu, objektu třídy XWPFDocument, se získá instance třídy POIXMLProperties, která v sobě uchovává objekt třídy ExtendedProperties. Tento objekt v sobě má uloženou informaci o počtu slov. Část kódu, jež zjišťuje počet slov dokumentu, je uvedena zde:

```
POIXMLProperties prop = xwpfDoc.getProperties();
int wordsCount = prop.getExtendedProperties().getWords();
```

#### Kontrola u ODT

Z hlavní reprezentace odt dokumentu, objektu třídy TextDocument, se získá instance třídy Meta, která v sobě uchovává meta-informace o dokumentu. Z této instance se získá objekt třídy DocumentStatistic, jenž v sobě má uloženou informaci o počtu slov. Část kódu, která zjišťuje počet slov dokumentu, je uvedena zde:

```
Meta meta = odtDoc.getOfficeMetadata();
DocumentStatistic stat = meta.getDocumentStatistic();
int wordCount = stat.getWordCount();
```

## 3.2 Počet stran

Pravidlo bude kontrolovat počet stran celého dokumentu.

#### Parametr limit

Celočíselný údaj, který je jediným parametrem tohoto pravidla. V rámci kontroly pravidla bude porovnán počet stran dokumentu se zadaným limitem. Uživatel bude programem informován, zdali došlo k překročení limitu či byl splněn. Parametr musí být zadán právě jednou.

#### Kontrola u DOC

Z hlavní reprezentace doc dokumentu, objektu třídy HWPFDocument, se získá instance třídy SummaryInformation, která má v sobě uchovanou informaci o počtu stran. Část kódu, jež zjišťuje počet stran dokumentu, je uvedena zde:

```
SummaryInformation summary = hwpfDoc.getSummaryInformation();
int pageCount = summary.getPageCount();
```

#### Kontrola u DOCX

Z hlavní reprezentace docx dokumentu, objektu třídy XWPFDocument, se získá instance třídy POIXMLProperties, která v sobě uchovává objekt třídy ExtendedProperties. Tento objekt v sobě má uloženou informaci o počtu stran. Část kódu, jež zjišťuje počet stran dokumentu, je uvedena zde:

```
POIXMLProperties prop = xwpfDoc.getProperties();
int pageCount = prop.getExtendedProperties().getPages();
```

#### Kontrola u ODT

Z hlavní reprezentace odt dokumentu, objektu třídy TextDocument, se získá instance třídy Meta, která v sobě uchovává meta-informace o dokumentu. Z této instance se získá objekt třídy DocumentStatistic, jenž v sobě má uloženou informaci o počtu stran. Část kódu, která zjišťuje počet stran dokumentu, je uvedena zde:

```
Meta meta = odtDoc.getOfficeMetadata();
DocumentStatistic stat = meta.getDocumentStatistic();
int pageCount = stat.getPageCount();
```

## 3.3 Velikost řádkování

Pravidlo bude kontrolovat velikost řádkování. Kontrola řádkování bude provedena pro každý odstavec dokumentu.

#### Parametr value

Reálné číslo, které je jediným parametrem tohoto pravidla. V rámci kontroly pravidla bude u každého odstavce porovnáno zjištěné řádkování se zadaným parametrem value. U každého odstavce bude uživatel informován, zdali zjištěné řádkování zadaný parametr překročilo či nikoliv. Parametr musí být zadán právě jednou.

#### Kontrola u DOC

Každý odstavce reprezentuje objekt třídy Paragraph. Tato třída má metodu getLineSpacing(), která vrací řetězec s informacemi o řádkování. Zpracováním tohoto řetězce se získá hodnota řádkování daného odstavce.

#### Kontrola u DOCX

Řádkování je zjišťováno přímo z XML souboru document.xml. Pro každý odstavec se získá jeho element s názvem w:p z XML souboru a je testováno, zdali se mezi potomky vyskytuje element w:spacing s atributem w:line. Pokud se tento element mezi potomky vyskytuje, je hodnota řádkování načtena z něj. Pokud element mezi potomky není, je hodnota řádkování získána ze stylu odstavce.

#### Kontrola u ODT

U každého odstavce je zjišťováno, jaké má nastaveny instance OdfStyleProperty, což je třída uchovávající informace o stylu odstavců. Pokud má odstavec nastaven OdfStyleProperty se jménem fo:line-height, je řádkování rovno hodnotě této instance.

## 3.4 Okraje stránky

Pravidlo bude kontrolovat okraje stránky dokumentu. Kontrola bude vždy provedena pro všechny čtyři strany – horní, dolní, pravý a levý okraj. Pokud je dokument rozdělen na oddíly, které mají odlišné okraje stran, budou zkontrolovány všechny oddíly.

#### Parametr value

Reálné číslo, které je jediným parametrem tohoto pravidla. V rámci kontroly pravidla bude u každého okraje porovnána zjištěná hodnota se zadaným parametrem value. U každé strany bude uživatel informován, zdali okraj zadaný parametr překročil či nikoliv. Parametr musí být zadán právě jednou.

#### Kontrola u DOC

Informace o okrajích stránky je uchována v objektu třídy Section. K sekcím

se přistupuje přes hlavní třídu objektového modelu knihovny HWPF, což je třída Range. Objekt této třídy se získá z hlavní reprezentace dokumentu, z třídy HWPFDocument. Pokud dokument obsahuje více objektů třídy Section, jsou zkontrolovány okraje ve všech těchto objektech. Část kódu, která zjišťuje okraje stran dokumentu, je uvedena zde:

```
Range range = hwpfDoc.getRange();
Section sec = range.getSection(0);
double left = sec.getMarginLeft();
double right = sec.getMarginRight();
double top = sec.getMarginTop();
double bottom = sec.getMarginBottom();
```

### Kontrola u DOCX

K informaci o okrajích stránky se přistupuje přes třídu CTDocument1, což je odkaz na kořenový element XML souboru document.xml. Z něj se získá postupně odkaz na tělo dokumentu, poté na sekci a z ní na okraje stránky. Pokud dokument obsahuje více sekcí, jsou zkontrolovány okraje ve všech. Část kódu, která zjišťuje okraje stran dokumentu, je uvedena zde:

```
CTDocument1 documentCt = xwpfDoc.getDocument();
CTBody body = documentCt.getBody();
CTSectPr section = body.getSectPr();
CTPageMar margin = section.getPgMar();
double left = margin.getLeft().doubleValue();
double right = margin.getRight().doubleValue();
double top = margin.getTop().doubleValue();
double bottom = margin.getBottom().doubleValue();
```

### Kontrola u ODT

Informace o okrajích stránky je uložena v souboru styles.xml. Přistupuje se k ní tedy přes objekt třídy OdfStylesDom. V souboru se najde element style:page-layout, jenž má potomka style:page-layout-properties. Pokud je v dokumentu více elementů style:page-layout jsou zkontrolovány okraje pro všechny tyto elementy. A tento element má atributy, které reprezentují jednotlivé okraje stránky.

## 3.5 Uvedení jména

Pravidlo bude kontrolovat, zdali bylo v zápatí či záhlaví dokumentu uvedené konkrétní jméno či více jmen.

#### Parametr text

Textový parametr, kterým uživatel zadá, jaké jméno či jména má program v dokumentu hledat. U každého jména zadaného jako parametr bude uživatel informován, zdali bylo v záhlaví nebo zápatí nalezeno. Pravidlo musí mít zadaný alespoň jeden parametr.

#### Kontrola u DOC

Z hlavní reprezentace dokumentu, objektu třídy HWPFDocument, se metodou getHeaderStoryRange() získá objekt třídy Range, který v sobě má informace o záhlaví i zápatí dokumentu. Z něj se získá textový obsah záhlaví a zápatí, a ten se porovnává s hledanými jmény.

#### Kontrola u DOCX

Hlavní reprezentace dokumentu, objekt třídy XWPFDocument, má metody getHeaderList() a getFooterList(), které vrací seznam objektů XWPFHeader a XWPFFooter, jež reprezentují všechna záhlaví a zápatí dokumentu. Ze seznamů se získá textový obsah záhlaví a zápatí a ten se porovná s hledanými jmény.

#### Kontrola u ODT

Hlavní reprezentace dokumentu, objekt třídy TextDocument, má metody getHeader() a getFooter(), které vracejí instance tříd Header a Footer, což jsou reprezentace záhlaví a zápatí dokumentu. Z těchto instancí se získá textový obsah záhlaví a zápatí a ten se porovná s hledanými jmény.

## 3.6 Uvedení data

Pravidlo bude kontrolovat, zdali bylo v zápatí či záhlaví uvedeno datum. Uživatel si určí pomocí parametru, jaké formáty data má pravidlo kontrolovat.

#### Parametr format

Tímto parametrem uživatel určí, jaké formáty data má pravidlo hledat. Parametr může mít následující hodnoty:

- dd.MM.yyyy,
- dd-MM-yyyy,
- dd/MM/yyyy,
- yyyy/MM/dd,
- yyyy-MM-dd,

kde dd určuje dny, MM měsíce a yyyy roky.
Parametr musí být zadán alespoň jeden, maximálně může být zadán až pětkrát, protože požadovaných formátů data je pět. U každého parametru bude uživatel informován, zdali byl datum v tomto formátu nalezen či nikoliv. Rovněž budou vypsána konkrétní nalezená data, která zadané formáty splňují.

#### Kontrola

Kontrola uvedení data v záhlaví a zápatí bude u jednotlivých formátů dokumentů probíhat stejně, jako bylo popsáno v kapitole 3.5.

## 3.7 Použitý font

Pravidlo bude kontrolovat, jaký font byl použit v jednotlivých odstavcích. Rovněž bude zjišťovat, jakou mají jednotlivé odstavce velikost fontu.

#### Parametr requiredFont

Jedná se o parametr, který reprezentuje požadovaný font. Obsahuje další dva parametry, name a size. První je řetězec a reprezentuje název požadovaného fontu a druhý je reálné číslo, které určuje požadovano velikost fontu. U každého odstavce

bude uživatel programem informován, zdali použitý font a jeho velikost patří mezi požadované či nikoliv. Pravidlo musí mít zadaný alespoň jeden tento parametr.

#### Kontrola u DOC

Text odstavce je uložen v objektu třídy CharacterRun. Tento objekt má metody getFontSize() a getFontName(), které vrací velikost fontu a jeho název.

#### Kontrola u DOCX

Text odstavce je uložen v objektu třídy XWPFRun. Tento objekt má metody getFontSize() a getFontFamily(), které vrací velikost fontu a jeho název. V případě, že odstavec přebírá font a jeho velikost od stylu, který má nastavený, vrací metody 0, respektive null. V takovém případě se font a velikost odstavce nastaví podle jeho stylu.

#### Kontrola u ODT

U každého odstavce je zjišťováno, jaké má nastaveny instance OdfStyleProperty, což je třída uchovávající informace o stylu odstavců. Pokud má odstavec nastaven OdfStyleProperty se jménem fo:font-size a style:font-name, jsou velikost fontu a jeho název rovny hodnotám těchto instancí. Pokud tyto instance nemá odstavec nastaveny, nastaví se font a jeho velikost podle stylu odstavce.

## 3.8 Použití konkrétního nadpisu

Pravidlo bude kontrolovat, zdali je v dokumentu použit nadpis či nadpisy požadované uživatelem.

#### Parametr text

Textový parametr, kterým uživatel zadá, jaké nadpisy má program v dokumentu hledat. Uživatele bude program u každého hledaného nadpisu informovat, zdali se v dokumentu vyskytuje či nikoliv. Pravidlo musí mít zadaný alespoň jeden parametr.

#### Kontrola u DOC

U objektu odstavce Paragraph lze pomocí metody getStyleIndex() získat index stylu, který má odstavec nastavený. Z hlavní reprezentace dokumentu, objektu třídy HWPFDocument, se metodou getStyleSheet() získá objekt třídy StyleSheet, který zahrnuje všechny styly použité v dokumentu. Mezi těmito styly se najde ten, jenž má shodný index s indexem stylu odstavce, zjistí se jeho jméno a otestuje se, zdali jméno obsahuje slova Nadpis nebo Heading, která reprezentují styly pro nadpisy. Pokud ano, jedná se o nadpis a jeho text může být porovnán s požadovanými nadpisy.

#### Kontrola u DOCX

Z objektu odstavce Paragraph se dá získat název jeho stylu pomocí metody getStyle(). Název stylu odstavce je poté zkontrolován stejně jako je popsáno u kontroly doc.

#### Kontrola u ODT

Z objektu odstavce Paragraph se dá získat název jeho stylu pomocí metody getStyleName(). Název stylu je poté zkontrolován stejně, jako je popsáno u kontroly doc.

## 3.9 Použití odkazů na literaturu

Pravidlo bude kontrolovat, zda je v dokumentu vygenerovaný seznam použité literatury, a kolik tento seznam obsahuje položek. Také bude kontrolovat, zdali je v dokumentu nadpis, který odpovídá uživatelem požadovanému názvu sekce s literaturou.

#### Parametr text

Textový parametr, který má reprezentovat název sekce s literaturou. Program tedy bude procházet nadpisy dokumentu a bude ověřovat, zdali se mezi nimi nachází požadovaný název sekce. U každého požadovaného názvu bude uživatel informován, zdali se v dokumentu nachází nebo ne. Pravidlo musí mít zadaný alespoň jeden textový parametr.

#### Parametr limit

Celočíselný parametr, který může uživatel zadat jen jednou. Jedná se o limit vygenerovaných počtů odkazů na literaturu. Zjištěný počet položek literatury program porovná s limitem a informuje uživatele o tom, zdali byl limit dosažen či nikoliv.

#### Kontrola u DOC

Kontrola existence nadpisu s konkrétním názvem proběhne stejně jako je popsáno v 3.8. Pro zjištění počtu položek v seznamu literatury se prochází jednotlivé odstavce a kontroluje se jejich styl podobně jako v 3.8. Pokud je název stylu odstavce Bibliografie, je čítač položek inkrementován, protože se jedná o vygenerovaný odkaz na literaturu.

#### Kontrola u DOCX

V XML souboru document.xml je vygenerovaný seznam literatury uložen v elementu sdt. Je tedy zapotřebí procházet všechny potomky tohoto uzlu a najít potomka s názvem sekce s literaturou a tento název poté otestovat s požadovanými názvy. Položky literatury jsou uloženy v tabulce. Je tedy zapotřebí tuto tabulku mezi potomky uzlu sdt najít a zjistit počet jejích řádků, který je roven počtu odkazů na literaturu.

#### Kontrola u ODT

Vygenerovaný seznam literatury je v dokumentu content.xml uložen v elementu text:bibliography. Mezi jeho potomky je nutné nalézt element s názvem text:index. Ten má potomka text:index-title, který obsahuje název sekce s literaturou a dále má potomky text:p reprezentující odstavce, které obsahují jednotlivé položky literatury. Počet těchto odstavců je tedy roven počtu odkazů na literaturu.

## 3.10 Použití konkrétního textu

Pravidlo bude kontrolovat, zdali se v dokumentu vyskytuje určitý úsek či úseky textu požadované uživatelem.

#### Parametr text

Textový parametr, kterým uživatel určí, jaký konkrétní text má program v dokumentu hledat. U každého hledaného úseku textu bude uživatel informován, zdali se text v dokumentu vyskytuje či nikoliv. Pravidlo musí mít zadaný alespoň jeden parametr.

#### Kontrola u DOC

Text jednotlivých odstavců se získá pomocí objektu třídy WordExtractor. Tato třída má metodu getParagraphText(), která vrací pole řetězců, kde jeden řetězec z pole je textový obsah jednoho odstavce. Takto se získá kompletní textový obsah dokumentu, jenž pak může být porovnáván se zadanými parametry.

#### Kontrola u DOCX

Třída reprezentující odstavec Paragraph má metodu getText(), jež vrací textový obsah odstavce. Pomocí této metody se získá kompletní textový obsah dokumentu, který pak může být porovnáván se zadanými parametry.

#### Kontrola u ODT

Třída reprezentující odstavec Paragraph má metodu getTextContent(), jež vrací textový obsah daného odstavce. Pomocí této metody se získá kompletní textový obsah dokumentu, který pak může být porovnáván se zadanými parametry.

## 3.11 Použití číslování stran

Pravidlo bude kontrolovat, zdali se v záhlaví nebo zápatí dokumentu vyskytuje číslování stran. Toto pravidlo nemá žádný parametr. Uživatel bude program informován, zdali bylo číslování v záhlaví nebo zápatí nalezeno či nikoliv.

#### Kontrola u DOC a DOCX

Kontrola obsahu záhlaví a zápatí probíhá stejně jako je popsáno v kapitole 3.5. V textu záhlaví a zápatí je hledán řetězec PAGE * MERGEFORMAT, který u obou formátů označuje použití číslování stránek.

#### Kontrola u ODT

V XML souboru je hledán element text:page-number, který značí, že v záhlaví nebo zápatí bylo použito číslování stránek.

# 4 XML soubor pro zadání pravidel

V kapitole 3 byla představena množina formálních pravidel, která bude finální program vytvořený v rámci této diplomové práce kontrolovat. Program nebude pokaždé kontrolovat všechna pravidla, ale pouze ta, jejichž kontrolu bude uživatel požadovat. K zadání uživatelem požadovaných pravidel a nastavení jejich parametrů bude sloužit XML soubor. Struktura XML souboru a způsob definování jednotlivých požadovaných pravidel budou popsány v této kapitole. Výstupem programu bude rovněž XML soubor, který vznikne úpravou vstupního XML.

Aby výsledný program vytvořený v rámci této diplomové práce mohl XML soubory zpracovávat, bylo nutné najít vhodný XML parser pro programovací jazyk Java. Analýze parserů a výběru konkrétního nástroje se rovněž věnuje tato kapitola.

V této kapitole bude také popsána validace vstupního XML souboru prostřednictvím XML schématu.

## 4.1 Struktura XML souborů

Program bude pracovat se dvěma XML soubory. Jeden bude vstupní a bude sloužit pro zadávání pravidel. Druhý bude výstupní, program bude prostřednictvím tohoto souboru informovat uživatele o výsledcích kontroly pravidel. V této kapitole budou představeny elementy použité ve vstupním a výstupním XML souboru a jejich vzájemné propojení.

Hlavními elementy obou XML souborů jsou:

- **rules** – jedná se o kořenový element XML souboru. Zahrnuje všechna zadaná pravidla. Jeho potomci jsou elementy rule.
- **rule** – element reprezentující jednotlivá pravidla. Je vždy potomkem kořenového elementu rules. Má vždy jen jednoho potomka, který definuje, o jaké konkrétní pravidlo se jedná.

### 4.1.1 Vstupní soubor

V této části bude popsáno, jak se ve vstupním XML souboru definují jednotlivá pravidla v rámci elementu rule. U každého pravidla bude uveden název elementu, který pravidlo deklaruje, elementy pro nastavení parametrů pravidla a konkrétní příklad definice pravidla.

### Počet slov

Element, který definuje toto pravidlo, se jmenuje words. Může mít jednoho potomka, který definuje parametr pravidla. Toto pravidlo má celočíselný parametr, jenž reprezentuje element limit. Element musí být uveden právě jednou. Příklad definice pravidla:

```
<rule>
    <words>
      <limit>1100</limit>
    </words>
</rule>
```

### Počet stran

Element, který definuje toto pravidlo, má jméno pages. Musí mít jednoho potomka, který definuje parametr pravidla. Jedná se o celočíselný parametr, jenž reprezentuje element limit. Element musí být uveden právě jednou. Příklad definice pravidla:

```
<rule>
    <pages>
      <limit>3</limit>
    </pages>
</rule>
```

### Velikost řádkování

Element, který definuje toto pravidlo, má jméno lines. Musí mít jednoho potomka, jenž definuje parametr pravidla. Jedná se o reálné číslo, které reprezentuje element s názvem value. Parametr musí být uveden právě jednou. Příklad definice pravidla:

```
<rule>
    <lines>
      <value>1.1</value>
    </lines>
</rule>
```

### Okraje stránky

Element, který definuje toto pravidlo, se jmenuje margin. Musí mít jednoho potomka, jenž definuje parametr pravidla. Jedná se o reálné číslo, které reprezentuje element s názvem value. Parametr musí být uveden právě jednou. Příklad definice pravidla:

```
<rule>
    <margin>
      <value>1.5</value>
    </margin>
</rule>
```

### Uvedení jména

Element, jenž definuje toto pravidlo, má jméno name. Může mít několik potomků, kteří definují parametr pravidla. Jedná se o textové parametry, které reprezentuje element s názvem text. Tento element obsahuje řetězec, jenž určuje hodnotu jednoho parametru. Pravidlo musí mít vždy alespoň jeden parametr. Příklad definice pravidla:

```
<rule>
    <name>
      <text>Ondřej Kupilík</text>
      <text>Martin Majer</text>
      <text>Marek Pučelík</text>
    </name>
</rule>
```

### Uvedení data

Element definující toto pravidlo, se jmenuje date. Může mít několik potomků. Jedná se pokaždé o element format, jenž reprezentuje parametr pravidla. Element obsahuje jeden z předem definovaných řetězců, které definují, jaké formáty data má pravidlo kontrolovat. Pravidlo musí mít vždy alespoň jeden parametr. Příklad definice pravidla:

```
<rule>
    <date>
      <format>dd/MM/yyyy</format>
```

```
      <format>dd.MM.yyyy</format>
      <format>yyyy/MM/dd</format>
      <format>yyyy-MM-dd</format>
    </date>
</rule>
```

### Použití konkrétního textu

Toto pravidlo definuje element textContent, jenž může mít několik potomků. Tito potomci musí být elementy text, které obsahují řetězec a reprezentují parametry pravidla. Pravidlo musí obsahovat minimálně jeden textový element. Příklad definice pravidla:

```
<rule>
    <textContent>
      <text>volejbal s dlouhými výměnami</text>
      <text>Vedlejší činností úseku je dále ještě
      prodej olejů</text>
      <text>zkusit spustit App.java a pokud se vypise neco
      jako</text>
    </textContent>
  </rule>
```

### Použitý font

Kontrolu tohoto pravidla definuje element s názvem font, jehož potomky jsou elementy requiredFont. Tento element musí být zadán alespoň jednou a má vždy dva potomky, elementy name a size. První obsahuje řetězec, který reprezentuje jméno požadovaného fontu. Druhý obsahuje reálné číslo, které určuje požadovanou velikost fontu. Příklad definice pravidla:

```
<rule>
<font>
 <requiredFont>
  <name>Times New Roman</name>
  <size>12</size>
 </requiredFont>
 <requiredFont>
  <name>Calibri</name>
  <size>11</size>
```

```
 </requiredFont>
 <requiredFont>
  <name>Times New Roman</name>
  <size>13</size>
 </requiredFont>
</font>
</rule>
```

### Použití konkrétního nadpisu

Element, který definuje toto pravidlo, se nazývá heading. Element má potomky, kteří určují parametr pravidla. Jedná se o textové parametry, které reprezentuje element s názvem text. Tento element obsahuje řetězec, jenž určuje hodnotu parametru. Pravidlo musí mít zadaný vždy alespoň jeden parametr. Příklad definice pravidla:

```
<rule>
    <heading>
      <text>Úvod</text>
      <text>Toleranční mez</text>
      <text>Unikátní zeměpisný útvar (popis)</text>
      <text>Uživatelská dokumentace</text>
    </heading>
</rule>
```

### Použití odkazů na literaturu

Toto pravidlo definuje element s názvem literature. Potomci elementu mohou být dvojího typu. Buď se jedná o elementy text, nebo limit. První reprezentuje textový parametr pravidla. Musí být zadán alespoň jeden. Element s názvem limit reprezentuje celočíselný parametr pravidla a musí být zadán právě jednou. Příklad definice pravidla:

```
<rule>
    <literature>
      <text>Literatura</text>
      <text>Reference</text>
      <text>Bibliografie</text>
      <limit>2</limit>
    </literature>
</rule>
```

#### Použití číslování stran

Element, který toto pravidlo definuje, se jmenuje pageNumber. Jelikož pravidlo nemá žádný parametr, jedná se o prázdný element. Definice tohoto pravidla vypadá následovně:

```
<rule>
    <pageNumber/>
</rule>
```

### 4.1.2 Výstupní soubor

Struktura výstupního XML bude vycházet ze vstupního souboru, výstupní dokument totiž vznikne jeho úpravou.

Elementy definující pravidla a jejich parametry ve vstupním souboru, zůstanou zachovány i ve výstupu. Pouze budou těmto elementům přidány atributy nebo potomci podle výsledků kontroly zadaných pravidel. Vzhledem k tomu, že struktura jednotlivých pravidel je odlišná, liší se i výstupy jednotlivých pravidel. V následující části budou proto popsány výstupní elementy pro všechna pravidla.

#### Počet slov

Elementu words bude přidán ve výstupním souboru potomek count. Ten bude obsahovat počet slov nalezených v kontrolovaném dokumentu. Dále přibude elementu limit atribut s názvem exceeded. Ten má hodnotu true v případě, že počet slov v dokumentu překročil zadaný limit. Pokud k překročení nedošlo, má hodnotu false. Příklad výstupu:

```
<rule>
    <words>
      <limit exceeded="true">1100</limit>
      <count>3436</count>
    </words>
</rule>
```

#### Počet stran

V tomto případě proběhne úprava elementu stejně jako u předchozího pravidla. Bude přidán potomek count s počtem stran dokumentu a podle toho, zdali tento počet překročí zadaný limit, bude atribut exceeded u elementu limit nastaven na true nebo false. Příklad výstupu:

```
<rule>
    <pages>
      <limit exceeded="false">3</limit>
      <count>1</count>
    </pages>
</rule>
```

### Velikost řádkování

Kontrolu velikosti řádkování provádí program u každého odstavce dokumentu. Ve výstupním souboru bude tedy každý odstavec reprezentován elementem paragraphLineSpacing, který je potomkem elementu lines. Obsahem elementu je velikost řádkování daného odstavce. Dále má element dva atributy, excedeed a text. První z nich určuje, zdali velikost řádkování daného odstavce překročila zadaný limit či nikoliv. Druhý atribut obsahuje prvních pět slov odstavce, aby bylo jasné, o který odstavec dokumentu se jedná.

```
<rule>
    <lines>
      <value>1.1</value>
      <paragraphLineSpacing excedeed="true" text="Máte
      k dispozici cca 20...">1.15</paragraphLineSpacing>
      <paragraphLineSpacing excedeed="true" text="Capital
      Market Line (CML)...">1.15</paragraphLineSpacing>
      <paragraphLineSpacing excedeed="false" text="Aby
      se mohlo přejít k...">1.1</paragraphLineSpacing>
    </lines>
</rule>
```

### Okraje stránky

Rodičovskému elementu margin budou přidáni potomci, elementy section, které reprezentují jednotlivé oddíly dokumentu. Element section má vždy čtyři potomky, elementy left, right, top a bottom, které reprezentují levý, pravý, horní a dolní okraj stránky v daném oddílu. Každý element obsahuje hodnotu okraje stránky v centimetrech. Zdali velikost jednotlivých okrajů překročila požadovanou hodnotu, určuje atribut exceeded, který mají všechny elementy. Pokud k překročení došlo, má atribut hodnotu true, pokud ne, je roven false. Příklad výstupu:

```
<rule>
    <margin>
```

```
      <value>1.5</value>
      <section number="1">
        <left exceeded="true">2.0</left>
        <right exceeded="true">2.0</right>
        <top exceeded="true">2.0</top>
        <bottom exceeded="true">2.0</bottom>
      </section>
      <section number="2">
        <left exceeded="true">2.499</left>
        <right exceeded="true">2.0</right>
        <top exceeded="true">2.0</top>
        <bottom exceeded="true">2.0</bottom>
      </section>
    </margin>
  </rule>
```

### Uvedení jména

Každému hledanému jménu, které reprezentuje element text, bude přidán atribut found. Pokud bylo jméno záhlaví dokumentu nalezeno, má atribut hodnotu true, pokud ne, je hodnota false. Příklad výstupu:

```
<rule>
    <name>
      <text found="false">Ondřej Kupilík</text>
      <text found="true">Martin Majer</text>
      <text found="false">Marek Pučelík</text>
    </name>
</rule>
```

### Uvedení data

Ke každému elementu format, jenž definuje požadovaný formát data je přidán atribut found, který má hodnotu true, pokud je datum v daném formátu v záhlaví dokumentu nalezeno, jinak je roven hodnotě false. Nalezený datum ve správném formátu je do výstupu přidán jako hodnota elementu foundDate, který je potomkem elementu date. Příklad výstupu:

```
<rule>
    <date>
```

```
      <format found="false">dd/MM/yyyy</format>
      <format found="true">dd.MM.yyyy</format>
      <format found="false">yyyy/MM/dd</format>
      <format found="false">yyyy-MM-dd</format>
      <foundDate>21.12.2015</foundDate>
    </date>
</rule>
```

### Použití konkrétního textu

Ke každému elementu text reprezentujícímu hledaný text je přidán atribut found. Ten je roven true, pokud byl konkrétní text v dokumentu nalezen, jinak má hodnotu false. Příklad výstupu:

```
<rule>
    <textContent>
      <text found="true">volejbal s dlouhými výměnami</text>
      <text found="true">Vedlejší činností úseku je dále ještě
      prodej olejů</text>
      <text found="false">zkusit spustit App.java a pokud se
      vypise neco jako</text>
    </textContent>
</rule>
```

### Použitý font

Kontrolu použitého fontu a jeho velikosti provádí program pro každý odstavec dokumentu. Jednotlivé odstavce reprezentují elementy paragraphFont, které jsou přidávány jako potomci elementu font. Obsahují text složený z velikosti fontu a jeho názvu. Mají dva atributy – text obsahuje prvních pět slov odstavce a correct má hodnotu true, pokud patří font a jeho velikost odstavce mezi požadované, jinak je rovny false. Příklad výstupu:

```
<rule>
<font>
      <requiredFont>
        <name>Times New Roman</name>
        <size>12</size>
      </requiredFont>
      <requiredFont>
```

```
        <name>Calibri</name>
        <size>11</size>
      </requiredFont>
      <requiredFont>
        <name>Source Sans Pro Light</name>
        <size>12</size>
      </requiredFont>
      <paragraphFont correct="false"
      text="Poptávkový dokument...">40.0 Source Sans Pro
      Semibold2</paragraphFont>
      <paragraphFont correct="true"
      text="ZČU je jedinou univerzitou sídlící...">12.0 Source
      Sans Pro Light</paragraphFont>
      <paragraphFont correct="false"
      text="Zpracovatel dokumentu...">15.9 Source Sans Pro
      Semibold</paragraphFont>
      <paragraphFont correct="true"
      text="Shrnutí projektu...">11.0 Calibri
      </paragraphFont>
</font>
</rule>
```

### Použití konkrétního nadpisu

Ke každému elementu text reprezentujícímu hledaný nadpis je přidán atribut found. Ten je roven true, pokud byl nadpis v dokumentu nalezen, jinak má hodnotu false. Příklad výstupu:

```
<rule>
    <heading>
      <text found="false">Úvod</text>
      <text found="true">Toleranční mez</text>
      <text found="false">Unikátní zeměpisný útvar (popis)</text>
      <text found="false">Uživatelská dokumentace</text>
    </heading>
</rule>
```

### Použití odkazů na literaturu

Elementům s názvem text, které reprezentují možné názvy sekce s literaturou, bude přidán atribut found. Ten bude mít hodnotu true, pokud byl požadovaný

název v dokumentu nalezen, nebo bude roven false, pokud název nalezen nebyl. Elementu literature je přidán potomek count, jenž obsahuje počet položek seznamu literatury nalezeného v dokumentu. A elementu limit je přidán atribut reached, který má hodnotu true, pokud je počet položek větší nebo roven zadanému limitu, v opačném případě je roven false. Příklad výstupu:

```
<rule>
    <literature>
      <text found="false">Literatura</text>
      <text found="false">Reference</text>
      <text found="true">Bibliografie</text>
      <limit reached="true">2</limit>
      <count>3</count>
    </literature>
</rule>
```

#### Použití číslování stran

Elementu pageNumber je přidán atribut found. Ten má hodnotu true, pokud má dokument očíslované stránky, jinak je roven false. Příklad výstupu:

```
<rule>
   <pageNumber found="true"/>
</rule>
```

## 4.2 XML parser

V této části diplomové práce bude provedena analýza nástrojů, pomocí nichž se dají v programovacím jazyce Java zpracovávat XML soubory. Takovéto nástroje se nazývají parsery, protože provádí takzvané parsování XML dokumentu, tedy jeho rozdělení na jednotlivé části [13].

### 4.2.1 Parsery

V této podkapitole budou stručně zanalyzovány dostupné XML parsery pro programovací jazyk Java. Na základě této analýzy bude vybrán parser, který bude použit při realizaci diplomové práce.

Podle způsobu zpracování XML dokumentu se rozeznávají dva přístupy [13]:

- **Proudové čtení** – parser postupně čte XML dokument a pro každou uceleno část vyvolá událost.

  - **Push parsery** – čtení XML probíhá automaticky, po přečtení určitého úseku dokumentu parser generuje události, na které programátor reaguje.
  - **Pull parsery** – čtení XML probíhá na žádost programátora. Ten generuje události a parser na ně reaguje tím, že vrátí požadovaný úsek dokumentu.

- **Práce se stromovou reprezentací dokumentu** – celý XML dokument se načte do stromové struktury v paměti.

V následující části budou ve stručnosti představeny čtyři XML parsery pro jazyk Java. Autor práce vybíral právě z těchto čtyř parserů, jelikož s nimi má osobní zkušenost z předmětu KIV/JXT (Java a XML objektové technologie) a práce s nimi je podrobně popsána v [13].

#### SAX

Celý název parseru je Simple API for XML. Jedná se o push parser, XML dokument tedy zpracovává proudově. Čte XML dokument odshora dolů, pro každou ucelenou část je vytvořena událost, ke které musí programátor vytvořit obsluhu.

Tento parser je vhodné použít v případě, že se jedná o rozsáhlý XML dokument a XML nemá hluboké zanoření elementů. Výhodou je rychlost zpracování a nízká spotřeba paměti. Nevýhodou je sekvenční zpracování, parser neumožňuje se při průchodu dokumentu vracet. Parser rovněž není vhodný pro vytváření či modifikaci dokumentů [13], [15], [16].

#### StAX

Celý název parseru je Streaming API for XML. Jedná se o pull parser. Rozdíl oproti SAX je ten, že dokument je čten na žádost, průchod dokumentu tak kontroluje programátor. Dalším rozdílem je možnost modifikace a vytváření dokumentů. StAX je nevalidující parser, což znamená, že neumí kontrolovat strukturu XML dokumentu podle XSD a DTD souborů (viz kapitola 4.3.2). Výhodou tohoto parseru je vysoká rychlost zpracování, nízká paměťová náročnost a již zmíněná možnost vytváření a modifikace XML dokumentů [13], [15], [16].

#### DOM

Celý název je Document Object Model. Parser pracuje se stromovou reprezentací dokumentu. Parser načte celý dokument do paměti, kde vytvoří jeho stromo-

vou objektovou reprezentaci, v níž každému elementu odpovídá jeden uzel stromu [13], [14].

Narozdíl od parseru SAX není dokument procházen od začátku do konce, ale programátor se může v dokumentu pohybovat dle své potřeby. Je vhodný rovněž pro modifikaci dokumentu. Nevýhodou je malá rychlost a velká paměťová náročnost načítání, což může být problémem u velkých dokumentů [13], [14], [15], [16].

#### JAXB

Celý název je Java Architecture for XML Binding. Využívá XSD schématu, ve kterém je nadefinována struktura XML dokumentu (viz kapitola 4.3.2). JAXB umožňuje využít informací o XML dokumentu z XSD souboru k automatickému vygenerování odpovídajících třídy v Javě. Při operacích unmarshal (čtení XML dokumentu) a marshal (zápis XML dokumentu) se poté pracuje s objekty vygenerovaných tříd. Použití tohoto nástroje je vhodné v případech, kdy známe dopředu schéma XML, potřebujeme XML načítat a ve velkém rozsahu modifikovat, nebo XML dokument není příliš rozsáhlý a informace v něm jsou hodně strukturované [13].

### 4.2.2 Vybraný parser

Pro vytvoření programu, který bude splňovat zadání této diplomové práce, je zapotřebí použití XML parseru, který bude mít následující vlastnosti:

- Možnost XML dokument číst i modifikovat. Vstupní XML dokument s definicí pravidel totiž program modifikuje na výstupní soubor s výsledky.
- Možnost validovat proti XSD schématu. Takto validován bude vstupní XML dokument s definicí pravidel.

Nástroj SAX sice umožňuje XML dokument validovat proti XSD schématu, ale není vhodný pro modifikaci dokumentu. Parser StAX umožňuje úpravu XML, ale není validující, takže také nesplňuje jednu z podmínek. Nástroj JAXB obě podmínky splňuje. Ale vzhledem k tomu, že struktura vstupního dokumentu je poměrně jednoduchá, nebylo použití JAXB vyhodnoceno jako potřebné. Vybrán byl tedy parser DOM, který splňuje obě podmínky. To, že je tento parser pomalejší a paměťově náročnější, není vzhledem malému rozsahu vstupního i výstupního dokumentu významné. Velkou výhodou vybraného nástroje je poměrně snadná modifikace vstupního dokumentu na výstupní.

## 4.3 Validace XML souboru

V této kapitole budou popsány možnosti kontroly struktury XML dokumentu.

### 4.3.1 Správně strukturovaný XML dokument

Nejnižší úrovní validace je kontrola, zdali je dokument správně strukturovaný (well-formed). Takový dokument splňuje následující pravidla syntaxe jazyka XML [13].

- Musí existovat právě jeden kořenový element.
- Každá počáteční značka má odpovídající ukončovací značku.
- Elementy se nesmějí navzájem překrývat (křížit).
- Hodnoty atributů musejí být uzavřeny v uvozovkách nebo apostrofech.
- Elementy nesmějí mít stejně pojmenované atributy.
- Komentáře nesmějí být vnořené a ani uvnitř značek.
- Ve znakových datech nejsou znaky < a &.

Tyto základní pravidla jsou kontrolována každým XML parserem. Pokud je dokument splňuje, je správně strukturovaný. Dále může být dokument označen za validní. Struktura takového dokumentu musí splňovat výše uvedená pravidla a rovněž musí odpovídat struktuře definované XML schématem viz kapitola 4.3.2 [14], [17].

### 4.3.2 W3C XML Schema

Schémata se používají pro formální definici datových formátů založených na XML. Tato definice je jednoznačná a jasně určuje, jak mohou dokumenty vypadat. Pomocí schématu se stanovuje, jaké elementy a atributy je možné v XML použít a jaké jsou mezi nimi při použití vzájemné vztahy [13], [14], [18].

K popisu XML schémat se používají schémové jazyky, kterých existuje několik – například DTD (Document Type Definition), W3C XML Schema, Relax NG nebo Schematron [14], [18]. V této diplomové práci bude pro validaci vstupního souboru s definicí pravidel použit jazyk W3C XML Schema. Použití tohoto jazyka bylo přímo stanoveno zadáním diplomové práce.

Jazyk W3C XML Schema je rovněž označován zkratkou XSD (XML Schema Definition), což je také nejčastější přípona souborů se schématy definovanými tímto jazykem. Autorem tohoto jazyka je konsorcium W3C. Tento jazyk dodržuje konvenci XML. V následující části kapitoly budou stručně představeny základní vlastnosti XSD.

Jazyk se používá k následujícím definicím [19]:

- Definice elementů a atributů, které se mohou objevit v dokumentu.
- Určení, které elementy jsou potomky.
- Stanovení pořadí a počtu potomků.
- Stanovení, zdali jsou elementy prázdné či mohou obsahovat text.
- Stanovení datových typů elementů a atributů.
- Definice výchozích a pevných hodnot pro elementy a atributy.

Kořenovým elementem schématu je xs:schema. Hlavním principem a zároveň velkou výhodou XSD je možnost definování vlastních datových typů. Ty mohou být dvojího typu [13], [20]:

- **Jednoduché** – element xs:simpleType. Odvozuje se od základních datových typů jako jsou string, date, decimal a další.
- **Složené** – element xs:complexType. Sestavuje se z jednoduchých datových typů.

K odvození jednoduchého datového typu se nejčastěji používá restrikce – element xs:restriction. Pomocí restrikce lze mimo jiné omezit délku řetězce, stanovit výčet přípustných hodnot, omezit číselný rozsah nebo omezit počet desetinných míst [13], [20].

Komplexní datové typy se skládají z elementů a atributů. Elementy se definují pomocí xs:element a atributy pomocí xs:attribute. Pro každý element v komplexním typu lze určit pořadí výskytu a počet opakování. Pro určení pořadí lze elementy v komplexním typu obalit pomocí xs:sequence, xs:all nebo xs:choice. Při použití první varianty musí být dodrženo pořadí elementů, v jakém byly ve schématu uvedeny. Druhá varianta umožňuje uvádět elementy v libovolném pořadí. Třetí možnost stanovuje, že může být uveden pouze jeden elementů definovaných uvnitř xs:choice. Počet výskytů daného elementu se stanovuje pomocí atributů minOccurs a maxOccurs, kde první atribut určuje minimální počet výskytů a druhý maximální počet výskytů [13], [20].

Základní přístupy k tvorbě schématu jsou následující [18], [20]:

- **Matrjóška** – analogie s ruskou panenkou. Definice všech atributů a elementů jsou zanořeny do jednoho globálního elementu.
- **Salámová kolečka** – opak přístupu matrjóška. Všechny elementy jsou definovány jako globální. Schéma je sestaveno pomocí referencí (odkazů).
- **Slepý Benátčan** – každý element a atribut má definovaný datový typ. Elementy jsou pak deklarovány lokálně pomocí těchto typů. Globálně definovaný element je jen jeden, jedná se o kořenový element dokumentu. Tato metoda je ve většině případů nejvhodnější volbou. Její nevýhodou je velikost schématu a s tím spojená větší pracnost.

V této kapitole byly stručně popsány především ty konstrukce jazyka XSD, které budou použity v kapitole 4.3.3, při vytváření XSD souboru pro kontrolu vstupního souboru programu.

### 4.3.3 XSD soubor

V této kapitole bude popsán XSD soubor, jenž byl vytvořen v rámci diplomové práce a pomocí něhož je validován vstupní soubor programu s definicí pravidel.

Pro sestavení schématu byla zvolena metoda slepého Benátčana, která byla představena v kapitole 4.3.2. Tato metoda byla zvolena, protože je nejpřehlednější a autor práce ji používal již v minulosti. V souladu s tímto přístupem byly nadefinovány datové typy pro všechny elementy použité ve vstupním souboru.

Jednoduché datové typy byly deklarovány pro elementy, které obsahují základní datové typy. Jedná se o elementy, jež reprezentují parametry pravidel:

- **formatType** – obsahuje základní datový typ string (textový řetězec). Deklaruje výčet hodnot, které může element format obsahovat.
- **limitType** – obsahuje základní datový typ nonNegativeInteger (nezáporné celé číslo). Jiná omezení nemá.
- **textType** – obsahuje základní datový typ string (textový řetězec). Jiná omezení nemá.
- **valueType** – obsahuje základní datový typ float (reálné číslo). Jiná omezení nemá.

Pro elementy, které do sebe mají vnořené další elementy, byly deklarovány složené (komplexní) datové typy. Jedná se o elementy, jež reprezentují jednotlivá pravidla:

- **dateType** – datový typ elementu date, jenž reprezentuje pravidlo kontrolující uvedení data. Datový typ obsahuje konstrukci sequence, do které je vložen element format. Ten je datového typu formatType a vyskytuje se v sekvenci minimálně jednou.
- **fontType** – datový typ elementu font, který reprezentuje pravidlo pro kontrolu použitého fontu. Obsahuje konstrukci sequence, do níž jsou vloženy elementy requiredFont. Ty je datového typu requiredFontType a musí se v sekvenci vyskytovat alespoň jednou.
- **headingType** – datový typ elementu heading, jenž reprezentuje pravidlo pro kontrolu použití konkrétního nadpisu. Obsahuje konstrukci sequence, do které je vložen element text datového typu textType. Element se v sekvenci vyskytuje minimálně jednou.

- **linesType** – datový typ elementu lines, který reprezentuje pravidlo pro kontrolu velikosti řádkování. Obsahuje konstrukci sequence, do níž je vložen element value. Ten je datového typu valueType a musí být použit právě jednou.

- **literatureType** – datový typ elementu literature, který reprezentuje pravidlo kontrolující použití odkazů na literaturu. Je zde použito skládání pomocí sequence. Do sekvence jsou vloženy elementy text datového typu textType a limit typu limitType. První element se musí v sekvenci vyskytovat minimálně jednou. Druhý element musí být použit právě jednou. Kvůli použití sequence musí být dodrženo pořadí elementů, nejdříve jsou uvedeny všechny text elementy, limit je uveden až po nich.

- **marginType** – datový typ elementu margin, který reprezentuje pravidlo pro kontrolu okrajů stránky. Obsahuje konstrukci sequence, do níž je vložen element value. Ten je datového typu valueType a musí být použit právě jednou.

- **nameType** – datový typ elementu name, jenž reprezentuje pravidlo kontrolující použití konkrétního jména v záhlaví dokumentu. Obsahuje konstrukci sequence, do které je vložen element text datového typu textType. Element se v sekvenci vyskytuje minimálně jednou.

- **pagesType** – datový typ elementu pages, který reprezentuje pravidlo kontrolující počet stránek. Obsahuje konstrukci sequence, do níž je vložen element limit. Ten je datového typu limitType a musí být použit právě jednou.

- **textContentType** – datový typ elementu textContent, jenž reprezentuje pravidlo pro kontrolu použití konkrétního textu v dokumentu. Obsahuje konstrukci sequence, do které je vložen element text datového typu textType. Element se v sekvenci vyskytuje minimálně jednou.

- **requiredFontType** – datový typ elementu requiredFont, jenž je použit v rámci fontType pro definici požadovaných fontů. Obsahuje konstrukci all, do které jsou vloženy elementy name typu textType a size typu valueType. Oba elementy musí být použity právě jednou a vzhledem k použití all nezáleží na jejich pořadí.

- **wordsType** – datový typ elementu words, jenž reprezentuje pravidlo kontrolující počet slov v dokumentu. Obsahuje konstrukci sequence, do které je vložen element limit. Ten je datového typu limitType a musí být použit právě jednou.

Pro element rule, jenž obaluje každé pravidlo, byl vytvořen složený datový typ ruleType. Ten obsahuje konstrukci choice, ve které jsou elementy odpovídající jednotlivým pravidlům. Znamená to tedy, že element rule vždy obsahuje pouze

jednoho potomka, který reprezentuje jedno z pravidel. Definice datového typu ruleType:

```
<xs:complexType name="ruleType">
    <xs:choice>
      <xs:element name="words" type="wordsType"/>
      <xs:element name="pages" type="pagesType"/>
      <xs:element name="lines" type="linesType"/>
      <xs:element name="margin" type="marginType"/>
      <xs:element name="name" type="nameType"/>
      <xs:element name="date" type="dateType"/>
      <xs:element name="font" type="fontType"/>
      <xs:element name="literature" type="literatureType"/>
      <xs:element name="heading" type="headingType"/>
      <xs:element name="textContent" type="textContentType"/>
      <xs:element name="pageNumber"/>
    </xs:choice>
</xs:complexType>
```

Kořenovým elementem dokumentu je rules. Pro tento element je definován složený datový typ rulesType, jenž obsahuje konstrukci sequence. Do ní je vložen element rule typu ruleType, který je tedy potomkem kořenového elementu a může se v dokumentu vyskytovat maximálně jedenáctkrát. Kořenový element má jako jediný samostatnou deklaraci mimo datové typy:

```
<xs:complexType name="rulesType">
    <xs:sequence>
      <xs:element name ="rule" type="ruleType" minOccurs="1"
      maxOccurs="11"/>
    </xs:sequence>
</xs:complexType>

<xs:element name="rules" type="rulesType"/>
```

# 5 Návrh a implementace programu

Tato kapitola se zabývá vytvořením programu, který bude kontrolovat nadefinovaná pravidla v textových dokumentech. Nejdříve budou představeny požadované funkce programu, poté programátorské prostředky, které autor při realizaci aplikace použil. Dále bude popsán návrh a architektura programu a v poslední části této kapitoly bude představena samotná implementace aplikace.

## 5.1 Požadované funkce programu

Před sestavením návrhu architektury řešení je vhodné shrnout, jaké funkce má mít program, který bude výstupem této diplomové práce. Jedná se o následující funkce:

- Načítání a zpracování textových dokumentů ve formátu doc, docx, odt.
- Načtení XML souboru, jehož prostřednictvím uživatel programu zadává, jaká pravidla chce kontrolovat. Struktura XML souboru je popsána v kapitole 4.1.1.
- Validace vstupního XML souboru proti XSD souboru, který byl představen v kapitole 4.3.3.
- Kontrola pravidel nadefinovaných v kapitole 3. Výstup kontroly pravidel musí být jednotný pro všechny tři formáty textových dokumentů.
- Modifikace vstupního XML souboru na výstupní soubor, který bude obsahovat informace o výsledcích kontroly zadaných pravidel.

## 5.2 Programátorské prostředky

K vytvoření programu byl použit programovací jazyk Java, konkrétně verze Java SE 7. Tento jazyk byl k realizaci aplikace vybrán vzhledem k tomu, že autor práce má s vytvářením aplikací v Javě největší zkušenosti v porovnání s ostatními programovacími jazyky.

Účelem výsledného programu by měla být kontrola textových dokumentů ve formátech doc, docx a odt. Bylo tedy nutné nalézt knihovny pro programovací jazyk Java, které umějí dokumenty v těchto formátech zpracovávat. Výběr knihoven byl proveden v kapitole 2.3. Vybranými nástroji jsou ODFDOM pro zpracování odt a Apache POI, jenž zahrnuje knihovny HWPF pro zpracování doc dokumentů a XWPF pro zpracování docx.

Dále má program načítat a modifikovat XML soubor s definicí pravidel. Pro tyto účely bylo zapotřebí nalézt vhodný XML parser pro programovací jazyk Java. Analýza parserů byla provedena v kapitole 4.2 a vybraným nástrojem je DOM.

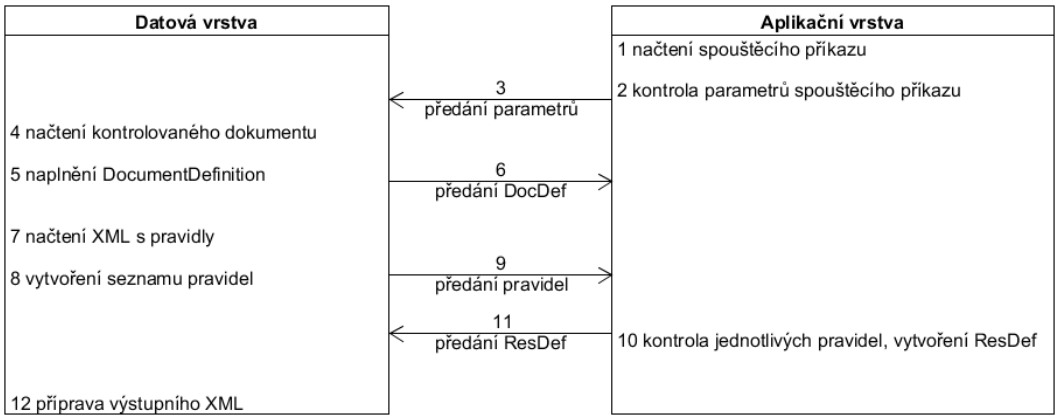
Program byl vytvářen ve vývojovém prostředí Eclipse IDE for Java Developers, konkrétně ve verzi Luna 4.4.1.

## 5.3 Návrh řešení

Architektura programu se bude dělit na dvě vrstvy, datovou a aplikační. Datová vrstva bude obstarávat načítání, zpracování a zápis do souborů. Aplikační vrstva bude celý program spouštět a řídit jeho chod. Jejím hlavním úkolem bude kontrola jednotlivých pravidel.

### 5.3.1 Dílčí činnosti vrstev

Pro účely návrhu architektury byl chod programu rozdělen na třináct dílčích činností, které byly podle své podstaty rozděleny mezi aplikační a datovou vrstvu. Aktivity týkající se přímo práce se soubory byly zařazeny do datové vrstvy, obslužné činnosti byly přiřazeny do aplikační vrstvy. Rozdělení aktivit je znázorněno na obrázku 5.1.




Obrázek 5.1: Rozdělení činností programu mezi vrstvy architektury

Jednotlivé činnosti uvedené na obrázku 5.1 jsou specifikovány v následujícím seznamu. Pořadí aktivit v seznamu odpovídá jejich pořadí v chodu programu:

1. **Načtení spouštěcího příkazu** – činnost aplikační vrstvy. Načítá ze zadaného příkazu parametry.

2. **Kontrola parametrů spouštěcího příkazu** – činnost aplikační vrstvy. Kontroluje načtené parametry, zdali byly zadány ve správném formátu a se správnými přepínači. Po dokončení této aktivity zná program název

a umístění kontrolovaného souboru, vstupního XML souboru s definicí pravidel a výstupního XML souboru. Dále také ze zadaných parametrů zjistí, má-li probíhat validace proti XSD.

3. **Předání parametrů** – přenos informací z aplikační vrstvy do datové. Načtené parametry obsahují informace o všech souborech, se kterými program pracuje, proto musí dojít k předání těchto dat z aplikační do datové vrstvy.

4. **Načtení kontrolovaného dokumentu** – činnost datové vrstvy. V předchozí aktivitě získá datová vrstva informaci o názvu a umístění kontrolovaného dokumentu, proto je dalším krokem jeho načtení.

5. **Vytvoření DocumentDefinition** – činnost datové vrstvy. Třída DocumentDefinition je součástí aplikační vrstvy a je blíže popsána v kapitole 5.3.2. K vytvoření objektu této třídy a jeho naplnění však dojde v datové vrstvě, protože při naplnění objektu se pracuje přímo s kontrolovaným dokumentem. Tento objekt pak obsahuje veškeré, pro program potřebné informace o načteném dokumentu, jedná se tak o jeho reprezentaci.

6. **Předání DocumentDefinition** – přenos informací z datové vrstvy do aplikační. K naplnění a vytvoření reprezentace dokumentu, tedy objektu DocumentDefinition, dojde v datové vrstvě. Dále tuto instanci už zpracovává aplikační vrstva, která ji potřebuje ke kontrole pravidel. Proto je objekt předán z datové vrstvy do aplikační.

7. **Načtení XML souboru s pravidly** – činnost datové vrstvy. Při předání parametrů získá datová vrstva mimo jiné informaci o názvu a umístění vstupního XML souboru s definicí pravidel. V tomto kroku datová vrstva vstupní soubor načítá.

8. **Vytvoření seznamu pravidel** – činnost datové vrstvy. Krok, ve kterém dojde k zpracování načteného vstupního XML souboru. Bude vytvořen seznam objektů rozhraní IRule (viz kapitola 5.3.2), kde každá položka seznamu bude reprezentovat jedno pravidlo načtené ze vstupního souboru.

9. **Předání pravidel** – přenos informací z datové vrstvy do aplikační. Seznam pravidel sestavený v předchozí činnosti na základě zpracování vstupního souboru bude předán aplikační vrstvě. Ta jej potřebuje pro provedení kontroly jednotlivých pravidel.

10. **Kontrola jednotlivých pravidel, vytvoření ResultDefinition** – činnost aplikační vrstvy. Na základě předaného seznamu pravidel a reprezentace kontrolovaného dokumentu, objektu DocumentDefinition, provede aplikační vrstva kontrolu jednotlivých pravidel ze seznamu. Výsledky kontroly se ukládají do objektu třídy ResultDefinition (viz kapitola 5.3.2).

11. **Předání ResultDefinition** – přenos informací z aplikační vrstvy do datové. Výsledky kontroly jednotlivých zadaných pravidel je nutné předat datové vrstvě, protože ta bude vytvářet výstupní XML soubor, v němž budou výsledky kontroly shrnuty.

12. **Příprava výstupního XML** – činnost datové vrstvy. Na základě výsledků kontroly pravidel, které jsou reprezentovány objektem třídy ResultDefinition a které byly datové vrstvě předány v předchozím kroku, modifikuje vrstva vstupní XML soubor. Výsledkem je výstupní XML soubor, jenž obsahuje všechna původní data a navíc jsou do něj přidány elementy a atributy, které uživateli poskytnou informaci o výsledcích kontroly.

13. **Uložení výstupního XML** – činnost datové vrstvy. Vrstva uloží výstupní XML soubor vytvořený v předchozím kroku na požadované místo. Umístění a název souboru jsou datové vrstvě předány v kroku číslo 3.

Jak byla provedena implementace jednotlivých činností je popsáno v kapitole 5.4.

### 5.3.2 Klíčová rozhraní a třídy

Za klíčová rozhraní a třídy programu jsou považovány ty, které jsou používány pro komunikaci mezi jednotlivými vrstvami architektury a ty, které obstarávají načítání a zpracování kontrolovaných dokumentů. Cílem této kapitoly není popsat implementaci těchto tříd, to bude provedeno v kapitole 5.4, ale představit princip návrhu řešení a úlohy jednotlivých rozhraní a tříd v navržené architektuře.

#### IDocumentLoader

Rozhraní, které patří do datové vrstvy. Implementují ho třídy, jejichž úkolem bude načítání kontrolovaného dokumentu. Vzhledem k tomu, že kontrolované dokumenty mohou mít tři různé formáty, půjde o tři třídy. Načítání jednotlivých formátů kontrolovaného dokumentu probíhá odlišným způsobem, jsou při nich používány různé knihovny. Úkolem rozhraní je sjednotit používání těchto tříd, kdy každá načítá jiný formát dokumentu odlišným způsobem, ale jejich použití bude díky implementaci stejného rozhraní jednotné.

#### DocumentDefinition

Třída, která patří do aplikační vrstvy. Jedná se o reprezentaci kontrolovaného dokumentu. Obsahuje informace potřebné ke kontrole všech nadefinovaných pravidel (viz kapitola 3). Vytvoření a naplnění objektu má na starosti datová vrstva programu, protože podkladem k těmto činnostem jsou data z kontrolovaného dokumentu. Tento objekt je poté předán aplikační vrstvě, která informace z něj používá k provedení kontroly jednotlivých pravidel.

#### ResultDefinition

Třída, která patří do aplikační vrstvy. Reprezentuje výsledky kontroly pravidel. K vytvoření a naplnění objektu této třídy dochází při kontrole pravidel v aplikační vrstvě. Tento objekt je poté předán datové vrstvě, která informace z něj použije při modifikaci vstupního XML souboru na výstupní.

#### IDocumentManager

Rozhraní, které patří do datové vrstvy. Třídy, které toto rozhraní implementují, mají za úkol vytvořit a naplnit objekt třídy DocumentDefinition, tedy vytvořit reprezentaci kontrolovaného dokumentu. Stejně jako u tříd implementujících rozhraní IDocumentLoader i zde probíhá vytvoření reprezentace dokumentu pro jednotlivé formáty odlišným způsobem a za použití různých knihoven. Opět toto rozhraní implementují tři třídy, každá má na starosti jeden formát dokumentu. Cílem rozhraní je sjednotit používání těchto tříd, ačkoliv každá třída pracuje odlišným způsobem.

#### IRule

Rozhraní, které patří do aplikační vrstvy. Zajišťuje jednotné volání metod, které mají na starosti kontrolu požadovaných pravidel. Implementují ho třídy reprezentující jednotlivá pravidla. Objekty těchto tříd jsou vytvářeny v datové vrstvě při zpracování vstupního XML. Seznam těchto objektů je poté z datové vrstvy předán aplikační vrstvě, která provádí kontrolu pravidel nad reprezentací dokumentu, instancí DocumentDefinition.

#### XMLLoader

Třída, která patří do datové vrstvy. Jejím úkolem je zpracování vstupního XML dokumentu tak, aby pro každé zadané pravidlo vznikl objekt odpovídající třídy, jež implementuje rozhraní IRule. Vytvoří se tak seznam objektů IRule, který je následně předán aplikační vrstvě.

## 5.4 Implementace

V této kapitole bude popsána implementace řešení navrženého v kapitole 5.1. Budou zde podrobně rozebrány jednotlivé vrstvy architektury programu.

### 5.4.1 Aplikační vrstva

Aplikační vrstva provádí následující činnosti:

- Spouští program.
- Kontroluje zadané argumenty programu.
- Předává datové vrstvě zpracované argumenty.
- Přebírá od datové vrstvy reprezentaci kontrolovaného dokumentu, objekt třídy DocumentDefinition, a seznam načtených pravidel.
- Pravidla kontroluje a výsledky kontroly ukládá do objektu třídy ResultDefinition.
- Předává objekt třídy ResultDefinition datové vrstvě.
- Vypisuje do konzole informativní hlášku o ukončení kontroly a uložení výsledků do výstupního souboru.

V další části kapitoly budou představeny jednotlivé balíky aplikační vrstvy. U každého balíku budou uvedeny nejdůležitější třídy a rozhraní a jejich metody.

#### Balík main

Obsahuje pouze třídu DPMain s metodou main, jež celý program spouští. Jedinou aktivitou, která se v metodě provádí je vytvoření objektu třídy ArgsParser, jemuž se předá pole argumentů programu.

### Balík control

Obsahuje následující třídy:

- **ArgsParser** – třída, jejímž úkolem je kontrola vstupních argumentů programu. Má jednu metodu – controlParams. Ta je volána z konstruktoru třídy a vrací datový typ boolean. Pokud jsou argumenty zadány správně vrací metoda true a informace načtené z argumentů jsou uloženy do instance třídy InputParams, jinak vrátí false. V případě, že kontrola argumentů proběhne správně, vytvoří se v konstruktoru objekt třídy RuleController, kterému se předává instance třídy InputParams.
- **InputParams** – třída, která slouží pouze k uchování argumentů programu. Pro každý atribut třídy jsou nastaveny odpovídající getry a setry.
- **RuleController** – třída, která řídí běh celého programu. Klíčovými metodami třídy jsou:
  - **findFileType** – zjistí, jaký formát má kontrolovaný dokument. Tuto informaci uloží do lokální proměnné třídy s názvem fileType.
  - **loadDocument** – podle formátu kontrolovaného dokumentu vytvoří buď objekt třídy DOCManager, DOCXManager, nebo ODTManager. Z tohoto objektu je získána zavoláním metody getDocumentDefinition instance třídy DocumentDefinition. Ta je uložena do lokální proměnné třídy nazvané docDef.
  - **controlRules** – získá z datové vrstvy seznam načtených pravidel. Poté seznam prochází a nad každým pravidlem zavolá metody control a summary. První z nich provede kontrolu pravidla a druhá uloží výsledky kontroly do objektu třídy ResultDefinition. Tento objekt uchovávající výsledky kontroly všech pravidel bude následně předán datové vrstvě, konkrétně objektu třídy DOMOutput.

### Balík rules

Tento balík obsahuje třídy, které reprezentují jednotlivá pravidla nadefinovaná v kapitole 3. Každá z těchto tříd implementuje rozhraní IRule a zároveň je potomkem třídy AbstractRule. Balík obsahuje následující třídy a rozhraní:

- **AbstractRule** – abstraktní třída, jejímiž potomky jsou třídy implementující jednotlivá pravidla. Třída má čtyři lokální proměnné, které reprezentují možné parametry pravidel. Ke každé proměnné jsou nastaveny odpovídající getry a setry.

- **IRule** – rozhraní, které implementují třídy reprezentující jednotlivá pravidla. Obsahuje celkem devět metod, z nichž nejzásadnější jsou dvě:
  - **control** – jediným parametrem metody je instance třídy DocumentDefinition. Úkolem této metody je získat z DocumentDefinition informace potřebné pro dané pravidlo a provést tak kontrolu pravidla.
  - **summary** – má jeden parametr, instanci třídy ResultDefinition. Objekt stejné třídy je i návratovou hodnotou metody. Metoda do objektu, který má zadaný jako parametr, přidá informace o výsledcích kontroly a tento objekt následně vrací.
- **DateRule** – provádí kontrolu uvedení data v záhlaví a zápatí dokumentu. Má tři lokální proměnné – objekt třídy DateUtil, seznam řetězců findingList a mapu findingMap, kde klíče jsou požadované formáty data a hodnota je datový typ boolean, implicitně nastavená na false. V metodě control jsou z instance třídy DocumentDefinition získány všechna záhlaví a zápatí dokumentu a pomocí DateUtil je kontrolováno, zdali v nich není uveden datum v jednom z požadovaných formátů. Pokud je takový datum nalezen, je přidán do seznamu findingList a v mapě findingMap je odpovídajícímu formátu data nastavena hodnota true. V metodě summary je vytvořena instance DateRuleResults, do ní jsou uloženy findingList a findingMap a celá instance je vložena do ResultDefinition.
- **Font** – třída, která reprezentuje požadovaný font. Uchovává v sobě jméno fontu a jeho velikost.
- **FontRule** – provádí kontrolu použitého fontu a jeho velikosti v jednotlivých odstavcích dokumentu. Má čtyři lokální proměnné – seznam požadovaných fontů fonts a mapy fontMap, correctFontMap a sizeMap. U všech map jsou klíče řetězce, u correctFontMap jsou hodnoty datového typu boolean, u fontMap jsou hodnotami řetězce a u sizeMap jde o datový typ double. V metodě control je z objektu DocumentDefinition získán seznam instancí ParagraphDefinition reprezentujících odstavce dokumentu. Při procházení odstavců je do každé mapy přidán jako klíč řetězec obsahující prvních pět slov odstavce. Do fontMap je jako hodnota vložen název fontu použitého v odstavci, do sizeMap velikost tohoto fontu a do correctFontMap se vloží true, pokud nalezený font a jeho velikost odpovídají některému z požadovaných fontů, v opačném případě je vložena hodnota false. V metodě summary je vytvořen objekt FontResultDefinition, do kterého jsou vloženy všechny tři mapy. Celá instance je poté uložena do ResultDefinition.
- **HeadingRule** – provádí kontrolu použití konkrétního nadpisu v dokumentu. Má jednu lokální proměnnou – mapu findingMap, kde klíče jsou hledané nadpisy a hodnotou je datový typ boolean, implicitně je nastavená

na false. V metodě control je z instance třídy DocumentDefinition získán seznam všech nadpisů v dokumentu. V tomto seznamu jsou hledány požadované nadpisy. Pokud je požadovaný nadpis nalezen, je v mapě findingMap klíči odpovídajícímu tomuto nadpisu nastavena hodnota true. V metodě summary je poté výsledná mapa uložena do instance ResultDefinition.

- **LinesRule** – provádí kontrolu velikosti řádkování jednotlivých odstavců dokumentu. Má dvě lokální proměnné – mapy paragraphMap, jejíž klíč je datového typu String a hodnota double, a excedeedMap, jejíž klíč je datového typu String a hodnota boolean. V metodě control je z instance DocumentDefinition získán seznam objektů ParagraphDefinition, které reprezentují odstavce dokumentu. Tento seznam je v metodě procházen a u každého odstavce je do obou map přidán stejný klíč, první čtyři slova odstavce. S tímto klíčem je do paragraphMap přidána jako hodnota velikost řádkování daného odstavce a do excedeedMap je vložena hodnota true, pokud velikost řádkování překročila požadovaný limit, nebo false, pokud je řádkování menší nebo rovno limitu. V metodě summary je vytvořena instance LinesRuleResults, do ní jsou uloženy obě mapy a celá instance je vložena do ResultDefinition.

- **LiteratureRule** – provádí kontrolu použití odkazů na literaturu v dokumentu. Má dvě lokální proměnné – bibiCounts je datového typu integer a findingMap je mapa, kde klíče jsou hledané názvy sekce s literaturou a hodnota je datový typ boolean, implicitně nastavená na false. V metodě control je z instance DocumentDefinition získán seznam všech nadpisů dokumentu, následný postup hledání požadovaných nadpisů je stejný jako v HeadingRule. Počet odkazů na literaturu se získá z DocumentDefinition a je uložen do proměnné bibiCounts. V metodě summary je vytvořena instance LiteratureRuleResults. Do ní jsou uloženy obě proměnné a celá instance je vložena do ResultDefinition.

- **MarginRule** – provádí kontrolu okrajů stránky. Má jednu lokální proměnnou – seznam margins, který obsahuje objekty MarginDefinition. V metodě control je seznam naplněn zavoláním metody getMargins nad objektem třídy DocumentDefinition. V metodě summary jsou postupně procházeny položky seznamu margins. Pro každý objekt MarginDefinition se vytvoří instance MarginRuleResults, do které se nastaví jednotlivé okraje z MarginDefinition a informace o překročení či dodržení požadovaného limitu. Každý takovýto objekt je vložen do seznamu a tento seznam instancí MarginRuleResults je přidán do ResultDefinition.

- **NameRule** – provádí kontrolu použití konkrétních jmen v záhlaví a zápatí dokumentu. Má jednu lokální proměnnou, mapu findingMap, kde klíče jsou hledaná jména a hodnota je datového typu boolean, implicitně nastavená na false. V metodě control jsou z instance třídy DocumentDefinition získány

seznamy se všemi řetězci ze záhlaví a zápatí dokumentu. Ty jsou postupně procházeny a pokud se řetězec shoduje s požadovaným jménem, je klíči odpovídající danému jménu ve findingMap nastavena hodnota true. V metodě summary je mapa vložena do ResultDefinition.

- **PageNumberRule** – provádí kontrolu použití číslování stránek. Má jednu lokální proměnnou s názvem pageNumbers, která je datového typu boolean. Hodnota proměnné je nastavena v metodě control zavoláním getru nad objektem DocumentDefinition, který vrátí true, pokud dokument má očíslované stránky, nebo false, pokud ne. V metodě summary je pageNumbers vložena do ResultDefinition.

- **PagesRule** – provádí kontrolu počtu stran dokumentu. Má jednu lokální celočíselnou proměnnou pageCount. Počet stran je získán v metodě control z objektu DocumentDefinition a je uložen do proměnné. V metodě summary je zjištěný počet stran porovnán se zadaným limitem. Proměnná s počtem stran a informace o překročení či dodržení limitu je vložena do ResultDefinition.

- **TextContentRule** – provádí kontrolu použití konkrétního textu v dokumentu. Má dvě lokální proměnné – seznam firstWords, který obsahuje první slova všech hledaných řetězců a mapu findingMap, kde klíče jsou hledané řetězce a hodnota je datového typu boolean a je implicitně nastavena na false. V metodě control je z instance DocumentDefinition získán seznam všech odstavců dokumentu. U každého odstavce je kontrolováno, jestli neobsahuje některé ze slov ve firstWords. Pokud je takové slovo nalezeno, je zavolána metoda containWord, jež zjišťuje, zdali řetězce následující po nalezeném slově dávají dohromady hledaný text. Pokud tomu tak je, v mapě findingMap je odpovídajícímu textu nastavena hodnota true. V metodě summary je mapa vložena do ResultDefinition.

- **WordsRule** – provádí kontrolu počtu slov v dokumentu. Má jednu lokální celočíselnou proměnnou wordsCount. Počet slov je získán v metodě control z objektu DocumentDefinition a je uložen do proměnné. V metodě summary je zjištěný počet slov porovnán se zadaným limitem. Proměnná s počtem slov a informace o překročení či dodržení limitu je vložena do ResultDefinition.

### Balík definitions

Tento balík obsahuje třídy, které v programu reprezentují kontrolovaný dokument. Jde o následující třídy:

- **DocumentDefinition** – třída, která obsahuje objekty ostatních tříd z tohoto balíku. Jedná se o jednu z klíčových tříd celého programu, přímo se s ní pracuje při kontrole pravidel. Její objekt totiž reprezentuje kontrolovaný dokument. Obsahuje informace o počtu stran, počtu slov, o použití číslování stránek. Dále má v sobě uložen seznam ParagraphDefinition, který reprezentuje odstavce dokumentu, seznam HeadingDefinition reprezentující všechny nadpisy, seznam MarginDefinition, jenž reprezentuje okraje stránek dokumentu, a seznam StyleDefinition, což jsou reprezentace stylů použitých v dokumentu. Třída DocumentDefinition ještě obsahuje informaci o počtu nalezených odkazů na literaturu, objekt HeaderDefinition, který reprezentuje záhlaví dokumentu, a instanci FooterDefinition, v níž jsou informace o zápatí. Vytvořením a naplněním objektu této třídy tak vznikne instance, která obsahuje pro program všechny potřebné informace o kontrolovaném dokumentu.

- **FooterDefinition** – třída uchovávající informace o zápatí dokumentu. Má v sobě uložen seznam datového typu String, který obsahuje všechny řetězce použité v zápatí dokumentu.

- **HeaderDefinition** – třída uchovávající informace o záhlaví dokumentu. Má v sobě uložen seznam datového typu String, který obsahuje všechny řetězce použité v záhlaví dokumentu.

- **HeadingDefinition** – třída, která reprezentuje nadpis. Uchovává text nadpisu, název jeho fontu a velikost písma.

- **MarginDefinition** – třída, která uchovává informace o okrajích dokumentu. Jsou v ní uloženy rozměry pravého, levého, horního i dolního okraje stránky v centimetrech.

- **ParagraphDefinition** – třída reprezentující odstavec. Je v ní uložen text odstavce, počet slov odstavce, velikost řádkování odstavce, použité fonty a jejich velikosti.

- **StyleDefinition** – třída, která uchovává informace o stylu použitém v dokumentu. Je v ní uloženo jméno stylu, použitý font a jeho velikosti, velikost řádkování a odkazy na rodičovské styly.

### Balík results

V tomto balíku jsou uloženy třídy, které reprezentují výsledky kontroly pravidel. Tyto třídy slouží pouze k uchování informací o kontrole, jedinými metodami ve třídách jsou tak getry a setry k proměnným tříd. Všechny třídy balíku jsou uvedeny v následujícím seznamu:

- **CountResults** – třída, která uchovává výsledky pravidel kontrolujících počet slov a stránek dokumentu. Má dvě lokální proměnné – celočíselnou count a exceeded datového typu boolean. První reprezentuje zjištěný počet a druhá určuje, zdali tento počet překročil požadovaný limit či nikoliv.
- **DateRuleResults** – třída, která reprezentuje výsledky pravidla kontrolujícího použití data. Má dvě lokální proměnné – seznam řetězců findingList a mapu findingMap, kde klíče jsou řetězce a hodnota je datového typu boolean. V seznamu jsou uložena nalezená data odpovídající požadovaným formátům. V mapě jsou klíče požadované formáty a hodnoty jsou informace, zdali byl daný formát data v dokumentu použit či nikoliv.
- **FontRuleResults** – třída, která reprezentuje výsledky pravidla pro kontrolu fontu použitého v jednotlivých odstavcích dokumentu. Má tři lokální proměnné – mapy fontMap, correctFontMap a sizeMap. Všechny mapy mají jako klíče řetězce, které obsahují prvních pět slov odstavce. Hodnotami u fontMap jsou řetězce s názvem fontu a u sizeMap jsou to velikosti fontu. V correctFontMap jsou hodnoty rovny true, pokud nalezený font a jeho velikost odpovídají některému z požadovaných fontů, v opačném případě jsou rovny false.
- **LinesRuleResults** – třída, která reprezentuje výsledky pravidla pro kontrolu řádkování odstavců. Má dvě lokální proměnné – konkrétně mapy findingMap a exceededMap. První mapa obsahuje jako klíče první pět slov odstavce a jako hodnotu velikost řádkování v daném odstavci. Ve druhé mapě je klíč stejný jako u první a hodnotou je datový typ boolean. Hodnota je nastavena na true, pokud velikost řádkování odstavce překročila požadovaný limit, jinak je false.
- **LiteratureResults** – třída, která reprezentuje výsledky pravidla kontrolujícího použití odkazů na literaturu. Má tři lokální proměnné – mapu findingMap, celočíselnou proměnnou s názvem bibiCounts a reached datového typu boolean. Klíče mapy jsou řetězce s hledanými názvy sekce s literaturou a hodnota je datového typu boolean. Je nastavena na true, pokud byl název sekce nalezen, v opačném případě je rovna false. Proměnná bibiCounts uchovává počet položek seznamu literatury a reached je rovna true, pokud počet položek dosáhl požadovaného limitu, jinak je hodnota nastavena na false.

- **MapResults** – třída, jež reprezentuje výsledky pravidel, které kontrolují výskyt konkrétního jména, textu a nadpisu v dokumentu. Má jednu lokální proměnnou, mapu findingMap. Klíče mapy jsou hledané řetězce, hodnota je datového typu boolean a je rovna true, pokud byl daný řetězec nalezen, jinak je nastavena na false.
- **MarginRuleResults** – třída, která reprezentuje výsledky pravidla kontrolujícího okraje stránky. Obsahuje osm proměnných, pro každý okraj stránky má dvě proměnné – jedna datového typu double a druhá boolean. V první je uložena velikost okraje, ve druhé informace o tom, zdali velikost okraje překročila požadovaný limit či nikoliv.
- **ResultsDefinition** – třída, která shrnuje informace o výsledcích kontroly všech zadaných pravidel. Obsahuje objekty všech ostatních tříd z tohoto balíku. Objekt této třídy je výstupem kontroly pravidel, aplikační vrstva tento objekt předává datové, jež na základě informací z něj modifikuje vstupní soubor na výstupní.

#### Balík util

Tento balík obsahuje pouze třídu DateUtil. Její objekt se používá ve třídě DateRule, jeho úkolem je totiž zjistit, zdali jednotlivá slova ze záhlaví a zápatí dokumentu nejsou datem v požadovaném formátu. Klíčová je metoda controlDate, která má jako parametr kontrolovaný řetězec. Její návratovou hodnotou je datový typ boolean. Metoda vrací true, pokud je kontrolované slovo datem v požadovaném formátu, jinak vrací false.

### 5.4.2 Datová vrstva

Datová vrstva provádí následující činnosti:

- Přebírá od aplikační vrstvy argumenty programu.
- Načítá kontrolovaný dokument.
- Zpracuje kontrolovaný dokument – vytvoří a naplní objekt třídy DocumentDefinition.
- Předává instanci DocumentDefinition aplikační vrstvě.
- Načítá vstupní XML soubor s definicí pravidel.
- Zpracuje vstupní XML soubor – vytvoří seznam objektů IRule, které reprezentují jednotlivá zadaná pravidla.

- Předává aplikační vrstvě seznam se zadanými pravidly.
- Přebírá od aplikační vrstvy výsledky kontroly v objektu třídy ResultDefinition.
- Podle výsledků kontroly modifikuje vstupní XML soubor na výstupní.
- Uloží výstupní XML soubor na požadované místo.

V další části kapitoly budou představeny jednotlivé balíky datové vrstvy. U každého balíku budou uvedeny nejdůležitější třídy a rozhraní a jejich metody.

#### Balík documentLoader

Balík obsahující jednu abstraktní třídu a dvě rozhraní:

- **AbstractManager** – abstraktní třída, jejímiž potomky jsou třídy, které obsluhují zpracování kontrolovaných dokumentů. Nejdůležitější proměnnou je objekt DocumentDefinition, jenž je v jednotlivých potomcích třídy naplněn informacemi o načteném dokumentu.
- **IDocumentLoader** – rozhraní, které implementují třídy načítající kontrolovaný dokument.
- **IDocumentManager** – rozhraní, které implementují třídy zpracovávající kontrolovaný dokument. Rozhraní má šest metod, nejdůležitější je getDocumentDefinition, jež vrací vytvořenou reprezentaci kontrolovaného dokumentu, objekt DocumentDefinition.

#### Balík documentLoader.doc

Balík, který má na starosti načtení a zpracování dokumentu ve formátu doc. Obsahuje dvě třídy:

- **DOCInput** – třída implementující rozhraní IDocumentLoader. Jejím cílem je načtení dokumentu, tedy vytvoření objektu třídy HWPFDocument, která patří do knihovny HWPF, jež zpracovává doc dokumenty.
- **DOCManager** – třída, která je potomkem AbstractManager a implementuje IDocumentManager. Zpracovává objekt třídy HWPFDocument, který byl vytvořen třídou DOCInput. Naplňuje objekt třídy DocumentDefinition informacemi, které jsou podkladem pro kontrolu všech pravidel. Jak konkrétně probíhá zpracování knihovnou HWPF je popsáno v kapitole 3, kde je u každého pravidla uveden postup kontroly v odstavci Kontrola u DOC.

#### Balík documentLoader.docx

Balík, jehož úkolem je načtení a zpracování dokumentu ve formátu docx. Obsahuje dvě třídy:

- **DOCXInput** – třída, která implementuje rozhraní IDocumentLoader. Jejím cílem je načtení dokumentu, tedy vytvoření objektu třídy XWPFDocument. Ta patří do knihovny XWPF, jež zpracovává docx dokumenty.
- **DOCXManager** – třída, která je potomkem AbstractManager a implementuje IDocumentManager. Zpracovává objekt třídy XWPFDocument, jenž byl vytvořen třídou DOCXInput. Naplňuje objekt třídy DocumentDefinition informacemi, které jsou podkladem pro kontrolu všech pravidel. Jak konkrétně probíhá zpracování knihovnou XWPF je popsáno v kapitole 3, kde je u každého pravidla uveden postup kontroly v odstavci Kontrola u DOCX.

#### Balík documentLoader.odt

Úkolem tohoto balíku je načtení a zpracování dokumentu ve formátu odt. Obsahuje dvě třídy:

- **ODTInput** – třída implementující rozhraní IDocumentLoader. Jejím cílem je načtení dokumentu, tedy vytvoření objektu třídy TextDocument, která patří do knihovny ODFDOM, jež zpracovává odt dokumenty.
- **ODTManager** – třída, která je potomkem AbstractManager a implementuje IDocumentManager. Zpracovává objekt třídy TextDocument, jenž byl vytvořen třídou ODTInput. Naplňuje objekt třídy DocumentDefinition informacemi, které jsou podkladem pro kontrolu všech pravidel. Jak konkrétně probíhá zpracování knihovnou ODFDOM je popsáno v kapitole 3, kde je u každého pravidla uveden postup kontroly v odstavci Kontrola u ODT.

#### Balík xml

Balík, který má na starosti veškeré operace s XML soubory, které se v programu provádí. Obsahuje následující třídy:

- **DOMInput** – třída, která načítá vstupní XML dokument prostřednictvím knihovny DOM. Při načítání provádí rovněž validaci proti XSD souboru. Třída má dvě lokální proměnné – loader je objekt třídy XMLLoader a doc je objekt rozhraní Document. První z nich provádí zpracování načteného

XML souboru, druhá je reprezentací XML dokumentu v knihovně DOM – právě do tohoto objektu je vstupní soubor načten. Načítání souboru probíhá v metodě read. Ta pomocí prostředků knihovny DOM dokument načte a následně nad ním zavolá metodu xsdValidation, která provede validaci proti XSD. Pokud XML soubor není vyhodnocen jako validní, program vyhodí výjimku a vypíše její zprávu. Pokud validace proběhne bez problémů, je nad proměnnou loader zavolána metoda loadRules, jejímž parametrem je proměnná doc s načteným XML. Tato metoda provede zpracování pravidel zadaných ve vstupním souboru a vytvoří seznam objektů IRule.

- **DOMOutput** – třída, která provádí modifikaci vstupního XML souboru na výstupní. Třída má tři lokální proměnné – doc je objektem rozhraní Document, results je instance ResultDefiniton a outputFile je řetězec s názvem souboru. Proměnná doc reprezentuje vstupní soubor, results uchovává výsledky kontroly pravidel. Na základě informací z této proměnné probíhá úprava souboru. V metodě editDocument jsou procházeny elementy rule nalezené ve vstupním souboru. Podle názvu potomka tohoto elementu je zjištěno, o jaké konkrétní pravidlo se jedná a na základě toho je zavolána odpovídající metoda, která daný element upraví. Jak probíhá úprava jednotlivých pravidel, je popsáno v kapitole 4.1.2. Po provedení modifikace je zavolána metoda createOutput, která vytvoří výstupní XML soubor a uloží jej na požadované místo.

- **RuleSet** – třída, která pouze uchovává statické konstanty datového typu String. V těchto konstantách jsou uloženy názvy jednotlivých elementů s pravidly. Jsou používány ve třídách XMLLoader a DOMOutput.

- **XMLLoader** – třída, která zpracovává vstupní XML soubor. Má jednu lokální proměnnou ruleList, seznam objektů rozhraní IRule, jenž je v této třídě naplněn na základě dat ze vstupního souboru. Zpracování XML řídí metoda loadRules, která má jeden parametr, a to objekt rozhraní Document z knihovny DOM. V této metodě jsou procházeny elementy rule z XML. Podle názvu potomka tohoto elementu je zavolána metoda vytvářející objekt odpovídající zadanému pravidlu. Každá z těchto metod vytvoří objekt třídy, jež implementuje rozhraní IRule, a vloží jej do ruleList.

### 5.4.3 Argumenty programu

Program je distribuován ve formátu jar. Spuštění se provádí z příkazové řádky pomocí dávkového bat souboru. Program může být spuštěn se čtyřmi argumenty, z nichž jeden je povinný a tři jsou volitelné. Každý argument musí být uveden přepínačem. Přepínače parametrů programu jsou vypsány v následujícím seznamu:

- **-f** – musí být použit při každém spuštění, uvádí povinný parametr, kterým je umístění a název kontrolovaného souboru.
- **-i** – jeho použití je nepovinné, uvádí totiž volitelný parametr, kterým je umístění a název vstupního XML souboru s pravidly. Pokud argument není uveden, zpracuje program pravidla nadefinovaná v ukázkovém souboru rules.xml, který je přímo uložen v jar balíku s programem ve složce resources.
- **-o** – jeho použití je nepovinné, uvádí totiž volitelný parametr, kterým je umístění a název výstupního XML souboru. V případě, že argument není uveden, je soubor s výsledky uložen pod názvem output.xml do adresáře, z něhož je program spouštěn.
- **-noxsd** – volitelný parametr. Je jako jediný používán samostatně. Pokud ho uživatel použije, nebude při načítání vstupního XML provedena validace proti XSD schématu. V případě, že uživatel tento přepínač nepoužije, je validace provedena proti XSD souboru popsanému v kapitole 4.3.3. Ten je přímo uložen v jar balíku s programem ve složce resources pod názvem rules.xsd.

# 6 Testování

Závěrečnou částí diplomové práce bylo dle zadání otestování vytvořeného programu na sadě různých dokumentů. Testovací množina dokumentů a průběh samotného testování je popsán v této kapitole.

## 6.1 Testovací množina

Ještě před sestavením množiny testovacích dokumentů byly stanoveny podmínky, které musí dokumenty v této sadě splňovat:

- V množině musí být zastoupeny soubory vytvořené ve všech třech textových procesorech představených v kapitole 2.1. Musí se tedy jednat o soubory ve formátech doc, docx nebo odt.
- Každý formát by měl v množině mít přibližně stejné zastoupení.
- Množina musí obsahovat minimálně 50 dokumentů.
- Dokumenty v sadě musí být z různých zdrojů, od různých autorů.

Dokumenty pro testování byly získány z mnoha různých zdrojů a svým obsahem a strukturou pokrývají široké spektrum použití textových procesorů (semestrální, bakalářské, diplomové práce atd.) Výsledná množina dokumentů obsahuje 75 dokumentů, přičemž docx má zastoupení 31 dokumentů, doc 23 dokumentů a odt 21 dokumentů.

Všechny soubory množiny jsou na přiloženém CD v adresáři test_files.

## 6.2 Nalezené problémy

Vytvořený program byl postupně spouštěn nad všemi dokumenty z testovací množiny představené v kapitole 6.1. U každého dokumentu byla otestována všechna pravidla nadefinovaná v kapitole 3.

U každého testovaného dokumentu byl obsah vstupního XML upravován tak, aby byl na každém dokumentu program otestován s různými konfiguracemi pravidel. Po každém spuštění programu nad testovaným dokumentem byl manuálně zkontrolován výstupní XML soubor.

Ačkoliv se autor snažil program testovat průběžně po každé větší úpravě kódu, byly během testování nalezeny situace, kdy program na zadané vstupy nereagoval správně. Problémy, na které autor práce při testování narazil jsou popsány v následující části kapitoly. Vzhledem k tomu, že každý formát dokumentu je zpracováván odlišně, týkala se nalezená chyba v programu zpravidla jen jednoho

formátu dokumentu. Proto jsou problémy rozčleněny do třech kategorií podle formátů dokumentu.

### Chyby u DOC

Dokumentů ve formátu doc bylo testováno dvacet tři. Při testování těchto dokumentů byly nalezeny následující nedostatky:

- Program při kontrole použití číslování stránek nepoznal, když byly stránky číslovány římsky.
- U kontroly použití data a jmen v záhlaví a zápatí program špatně rozpoznával data a jména, za kterými následoval konec řádky.
- Program nepřesně zjišťoval řádkování, které mělo nastavenou přesnou šířku v bodech.
- Program vypisoval okraje stránky pouze prvního oddílu dokumentu. Pokud byl dokument rozdělen na více oddílů s různými okraji stran, program okraje dalších oddílů nevypsal. Tato chyba se vyskytovala u všech formátů dokumentu.

Všechny nalezené chyby programu byly opraveny. Sadu testovacích dokumentů formátu doc zpracoval program pro provedení oprav správně.

### Chyby u DOCX

Dokumentů ve formátu docx bylo testováno třicet jedna. Při testování těchto dokumentů byly nalezeny následující nedostatky:

- Při kontrole použití číslování stran program nepoznal, když byly stránky číslovány římsky.
- Program špatně vyhodnocoval font a jeho velikost, pokud nebyly explicitně určeny pro styl, ale byly přebírány z rodičovského stylu.
- Při hledání jmen a dat v záhlaví způsobovalo problémy použití tabulátoru mezi jednotlivými jmény či daty.
- Počítání položek seznamu literatury nefungovalo přesně.
- Pokud odstavec neměl přímo nastavené řádkování, ale jeho hodnotu přebíral z použitého stylu, program takové řádkování nedokázal správně zpracovat.

Všechny nalezené chyby programu byly opraveny. Sada testovacích dokumentů formátu docx byla pro provedení oprav zpracována programem správně.

### Chyby u ODT

Dokumentů ve formátu odt bylo testováno dvacet jedna. Při testování těchto dokumentů byly nalezeny následující nedostatky:

- Kontrola použití jména a data v záhlaví a zápatí neprobíhala po slovech, ale po řádcích. Pokud byla dvě jména nebo data na jednom řádku, program našel jen jedno.
- Dvě mezery v textu po sobě byly spojovány do elementu text:s, který nedokázala knihovna ODFDOM zpracovat.
- Program nenačítal text v položkách seznamů. Knihovna ODFDOM totiž seznamy nezpracovává jako odstavce.

Všechny nalezené chyby programu byly opraveny. Sada testovacích dokumentů formátu odt byla po odstranění nedostatků zpracována programem správně.

# 7 Závěr

Při realizaci diplomové práce byly splněny všechny body zadání. Hlavním výstupem práce je aplikace, která byla vytvořena v programovacím jazyce Java a která kontroluje formální pravidla v dokumentech textových procesorů Microsoft Word, OpenOffice a LibreOffice. Diplomová práce je rozčleněna do sedmi kapitol. Samotnou realizací práce se zabývají kapitoly 2 až 6, ve kterých jsou postupně popsána řešení jednotlivých bodů zadání.

V teoretické části práce byla provedena analýza možností kontroly formálních pravidel u dokumentů vybraných textových procesorů. V rámci analýzy byly představeny jednotlivé procesory a formáty jejich dokumentů. Dále byly analyzovány knihovny pro jazyk Java, které umějí tyto dokument zpracovávat.

V rámci praktické části diplomové práce byl navržen a implementován program, který se zabývá kontrolou formálních pravidel v dokumentech vybraných textových procesorů. V první fázi byla nadefinována jednotlivá formální pravidla, která bude výsledná aplikace kontrolovat. Celkem bylo nadefinováno jedenáct pravidel.

Vstupem aplikace je XML soubor, jenž obsahuje definice formálních pravidel, která požaduje uživatel zkontrolovat. Navržení vhodné struktury XML souboru a její popis pomocí XSD schématu bylo další fází praktické části práce.

V závěrečné fázi byl sestaven návrh programu a na jeho základě byla realizována samotná implementace. Při navrhování aplikace byly jednotlivé činnosti programu rozděleny mezi aplikační a datovou vrstvu. Aplikace byla navržena tak, aby ji bylo možné snadno rozšířit o kontrolu nových pravidel. Výsledný program byl otestován na sadě sedmdesáti pěti dokumentů z různých zdrojů. Veškeré nedostatky nalezené v rámci testování byly opraveny.

Program by bylo možné dále rozšířit o grafické uživatelské rozhraní, které by usnadnilo jeho spouštění. Uživatelské rozhraní by mohlo také obsahovat funkcionalitu pro jednoduché vytváření vstupního XML souboru s definicemi pravidel, kde by si uživatel mohl nadefinovat požadovaná pravidla, aniž by musel manuálně vytvářet XML. Návrh aplikace byl proveden tak, aby byla snadno rozšiřitelná o kontrolu nových pravidel, vylepšením tak může být přidání kontroly dalších pravidel.

# Literatura


[1] MURRAY, K. – MILLHOLLON, M. – MELTON, B. *Mistrovství v Microsoft Office Word 2007*. 1. vydání. Computer Press, 2008. ISBN 978-80-251-2051-4.

[2] BUKOVJAN, A. *OpenXML dokument Wordu 2007 a jeho vnitřní struktura*. Plzeň, 2011. 57 s. Bakalářská práce na Pedagogické fakultě Západočeské univerzity na katedře výpočetní a didaktické techniky. Vedoucí bakalářské práce Mgr. Tomáš Jakeš.

[3] *Apache OpenOffice About*. URL: <http://www.openoffice.org/about/> [cit. 19.4.2015].

[4] THE DOCUMENT FOUNDATION. *LibreOffice 4.0 Getting Started Guide*. 2013. Dostupné z: <http://www.libreoffice.org/get-help/documentation/>

[5] *[MS-DOC]: Word (.doc) Binary File Format*. URL: <https://msdn.microsoft.com/en-us/library/office/cc313153(v=office.12).aspx>. [cit. 20.4.2015].

[6] *Introducing the Office (2007) Open XML File Formats*. URL: <https://msdn.microsoft.com/en-us/library/aa338205%28v=office.12%29.aspx> [cit. 20.4.2015].

[7] *Walkthrough: Word 2007 XML Format*. URL: <https://msdn.microsoft.com/en-us/library/bb266220%28v=office.12%29.aspx>. [cit. 20.4.2015].

[8] *Office Open XML – Anatomy of an OOXML WordProcessingML File*. URL: <http://officeopenxml.com/anatomyofOOXML.php> [cit. 21.4.2015].

[9] EISENBERG, D. J. *OASIS OpenDocument Essentials*. Lulu Enterprises Incorporated, 2006. 303 s. ISBN 9781411668324. Dostupné z: https://books.google.cz/books?id=-AasoVVqNU8C

[10] *What is the Open Document Format (ODF)? – Definition from Techopedia*. URL: <http://www.techopedia.com/definition/24432/open-document-format-odf>. [cit. 21.4.2015].

[11] *Apache POI - the Java API for Microsoft Documents*. URL: <https://poi.apache.org/>. [cit. 22.4.2015].

[12] *ODFDOM - the Open Document API*. URL: <https://incubator.apache.org/odftoolkit/odfdom/>. [cit. 22.4.2015].

[13] HEROUT, Pavel *Java a XML*. První vydání. České Budějovice: nakladatelství KOPP, 2007. ISBN 978-80-7232-307-4.

[14] KOSEK, Jiří *XML pro každého: podrobný průvodce*. První vydání. Praha: Grada Publishing, 2000. ISBN 80-7169-860-1.

[15] *Java XML Tutorial*. URL: <http://www.tutorialspoint.com/java_xml/index.htm>. [cit. 1.6.2015].

[16] *Java & XML Tool Overview — tutorials.jenkov.com*. URL: <http://tutorials.jenkov.com/java-xml/overview.html>. [cit. 1.6.2015].

[17] *XML Document Types*. URL: <http://www.w3schools.com/xml/xml_doctypes.asp>. [cit. 2.6.2015].

[18] BALON, M. *Generátor XML souborů řízený XSD schématem*. Plzeň, 2014. 80 s. Diplomová práce na Fakultě aplikovaných věd Západočeské univerzity na katedře informatiky a výpočetní techniky. Vedoucí diplomové práce doc. Ing. Pavel Herout Ph.D.

[19] *XML Schema Tutorial*. URL: <http://www.w3schools.com/schema/default.asp>. [cit. 2.6.2015].

[20] *XML schémata* [online]. 2014. URL: <http://www.kosek.cz/xml/schema/wxs.html>. [cit. 2.6.2015].

# Seznam zkratek

| | |
|---|---|
| MS | Microsoft |
| DOS | Diskový operační systém |
| DOC | Binární formát dokumentu programu Microsoft Word |
| DOCX | Na XML založený formát dokumentu programu Microsoft Word |
| ODT | Open Document Format pro textové procesory |
| XML | Extensible Markup Language |
| SXW | StarOffice Writer Document |
| RTF | Rich Text Format |
| TXT | Textový soubor |
| CSV | Comma-separated values |
| ZIP | Souborový formát pro kompresi a komprimaci dat |
| HTML | HyperText Markup Language |
| OASIS | Organization for the Advancement of Structured Information Standards |
| ODF | OASIS Open Document Format |
| JAR | Java Archive |
| MIME | Multi-Purpose Internet Mail Extensions |
| PNG | Portable Network Graphics |
| API | Application Programming Interface |
| DOM | Ústav pro informace ve vzdělávání |
| Apache POI | The Java API for Microsoft Documents |
| HWPF | Součást Apache POI pro formát DOC |
| XWPF | Součást Apache POI pro formát DOCX |

| | |
|---|---|
| ODFDOM | The Open Document API |
| SAX | Simple API for XML |
| StAX | Streaming API for XML |
| DOM | Document Object Model |
| JAXB | Java Architecture for XML Binding |
| XSD | XML Schema Definition |
| W3C | World Wide Web Consortium |
| DTD | Document Type Denition |
| BAT | Formát dávkového souboru pro Microsoft Windows |
| SVN | Subversion |

# Přílohy

## Uživatelská příručka

Tato příloha obsahuje návod pro uživatele k používání programu.

### Spuštění programu

Jedná se o konzolový program, je tedy spouštěn z příkazové řádky. Program je uložen v jar balíku s názvem DP_Kupilik.ja a je k němu vytvořen dávkový soubor run.bat.

Spuštění programu se provádí z příkazové řádky prostřednictvím dávkového souboru. Je zapotřebí mít dávkový soubor ve stejném adresáři jako jar s aplikací. Program může mít až čtyři argumenty, které se uvádí pomocí přepínačů:

- **-f** – musí být použit při každém spuštění, uvádí povinný parametr, kterým je umístění a název kontrolovaného souboru.
- **-i** – jeho použití je nepovinné, uvádí totiž volitelný parametr, kterým je umístění a název vstupního XML souboru s pravidly. Pokud argument není uveden, zpracuje program pravidla nadefinovaná v ukázkovém XML souboru, který je přímo uložen v jar balíku.
- **-o** – jeho použití je nepovinné, uvádí totiž volitelný parametr, kterým je umístění a název výstupního XML souboru. V případě, že argument není uveden, je soubor s výsledky uložen pod názvem sf output.xml do adresáře, ze kterého je program spouštěn.
- **-noxsd** – volitelný parametr. Je jako jediný používán samostatně. Pokud ho uživatel použije, nebude při načítání vstupního XML provedena validace proti XSD schématu. V případě, že uživatel tento přepínač nepoužije, je validace proti XSD provedena. XSD schéma je přímo uloženo v jar balíku.

Příklad spuštění programu, kdy se kontroluje dokument s názvem example.doc, pravidla jsou nadefinovaná v souboru myRules.xml, výstup je uložen do output.xml a je provedena validace proti XSD:

```
run.bat -f example.doc -i myRules.xml
```

Příklad spuštění programu, kdy se kontroluje dokument s názvem example.doc, pravidla jsou nadefinovaná ve výchozím souboru rules.xml v balíku jar, výstup je uložen do myOutput.xml a není provedena validace proti XSD:

```
run.bat -f example.doc -o myOutput.xml -noxsd
```

Pokud budou argumenty programu zadány špatně, kontrola pravidel nebude spuštěna a program vypíše informaci o chybně zadaném příkazu. Program je poté nutné spustit znovu se správnými parametry.

## Vstupní soubor

Ještě před spuštěním programu je zapotřebí si připravit vstupní soubor. Jedná se o XML soubor, jehož prostřednictvím uživatel určuje, jaká pravidla chce nad dokumentem textového procesoru zkontrolovat. Struktura vstupního XML souboru je detailně popsána v kapitole 4.1.1. Zde je uveden příklad XML vstupu, který obsahuje definice všech pravidel, které program umí zkontrolovat. U každé definice pravidla je uveden komentář, který vysvětluje, co daná definice znamená:

```
<?xml version="1.0" encoding="utf-8"?>

<rules xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="rules3.xsd">

  <!-- pravidlo na kontrolu poctu slov -->
  <!-- parametr limit stanovuje maximalni pocet slov -->
  <rule>
    <words>
      <limit>1100</limit>
    </words>
  </rule>

  <!-- pravidlo na kontrolu poctu stran -->
  <!-- parametr limit stanovuje maximalni pocet stran -->
  <rule>
    <pages>
      <limit>3</limit>
    </pages>
  </rule>

  <!-- pravidlo na kontrolu radkovani -->
  <!-- parametr value stanovuje maximalni velikost radkovani -->
  <rule>
    <lines>
```

```
   <value>1.1</value>
  </lines>
</rule>

<!-- pravidlo na kontrolu okraju stranky -->
<!-- parametr value stanovuje maximalni velikost okraju -->
<rule>
  <margin>
    <value>1.5</value>
  </margin>
</rule>

<!-- pravidlo na kontrolu pouziti konkretniho jmena -->
<!-- parametry jsou hledana jmena -->
<rule>
 <name>
    <text>Ondřej Kupilík</text>
    <text>Martin Majer</text>
    <text>Marek Pučelík</text>
  </name>
</rule>

<!-- pravidlo na kontrolu pouziti data -->
<!-- parametry jsou hledane formaty data -->
<rule>
  <date>
    <format>dd/MM/yyyy</format>
    <format>dd.MM.yyyy</format>
    <format>dd-MM-yyyy</format>
    <format>yyyy/MM/dd</format>
    <format>yyyy-MM-dd</format>
  </date>
</rule>

<!-- pravidlo na kontrolu pouziteho fontu -->
<!-- parametry jsou pozadovane fonty -->
<!-- name je jmeno pozadovaneho fontu -->
<!-- size je velikost pozadovaneho fontu -->
<rule>
  <font>
    <requiredFont>
  <name>Times New Roman</name>
  <size>12</size>
```

```
</requiredFont>
<requiredFont>
  <name>Calibri</name>
  <size>11</size>
</requiredFont>
<requiredFont>
<name>Source Sans Pro Light</name>
<size>12</size>
</requiredFont>
  </font>
</rule>

<!-- pravidlo na kontrolu pouziti odkazu na literaturu -->
<!-- parametry text jsou pozadovane nadpisy sekce -->
<!-- limit stanovuje minimalni pocet polozek literatury -->
<rule>
  <literature>
    <text>Literatura</text>
    <text>Reference</text>
    <text>Bibliografie</text>
    <limit>2</limit>
  </literature>
</rule>

<!-- pravidlo na kontrolu pouziti konkretniho nadpisu -->
<!-- parametry jsou hledane nadpisy -->
<rule>
  <heading>
    <text>Úvod</text>
    <text>Toleranční mez</text>
    <text>Unikátní zeměpisný útvar (popis)</text>
    <text>Uživatelská dokumentace</text>
  </heading>
</rule>

<!-- pravidlo na kontrolu pouziti konkretniho textu -->
<!-- parametry jsou hledane retezce -->
<rule>
  <textContent>
    <text>Lorem ipsum</text>
    <text>Ahoj nazdar</text>
    <text>Hledaný text</text>
   </textContent>
```

```
  </rule>

  <!-- pravidlo na kontrolu pouziti cislovani stranek -->
  <rule>
    <pageNumber/>
  </rule>

</rules>
```

# Seznam obrázků



# Obsah CD

Přiložené CD obsahuje tyto adresáře:

- src – obsahuje soubory se všemi zdrojovými kódy aplikace.
- jar – obsahuje soubor DP_Kupilik.jar, což je spustitelná verze programu. Dále je v něm dávkový soubor run.bat, pomocí něhož se program spouští, a soubor READ_ME.txt s návodem ke spuštění.
- text – v souboru DP_Kupilik.pdf je uložena finální verze diplomové práce. V adresáři text_src se nachází zdrojové soubory textu, které byly vytvořeny v programu TeX.
- test_files – obsahuje všechny dokumenty z testovací množiny.