

UNIVERZITA KARLOVA V PRAZE
MATEMATICKO-FYZIKÁLNÍ FAKULTA

DIPLOMOVÁ PRÁCE



Matúš Kapusta

Normalizace dat ve fulltextovém systému

Katedra softwarového inženýrství
Vedoucí diplomové práce: Mgr. Jan Lánský, Ph.D.
Studijní program: Informatika, softwarové systémy

Rád by som poďakoval vedúcemu mojej diplomovej práce, Mgr. Janovi Lánskému, za cenné rady, konzultácie a námety, ktoré som vo svojej práci použil.

Diplomovú prácu chcem venovať mojej rodine, mojim kolegom a všetkým ľuďom, ktorí ma počas štúdia podporovali a motivovali.

Prehlasujem, že som svoju diplomovú prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím s požičiavaním práce.

V Prahe dňa 11.12.2009

Matúš Kapusta

Obsah

1	ÚVOD.....	5
1.1	VYMEDZENIE CIEĽOV.....	5
1.2	ŠTRUKTÚRA PRÁCE.....	6
1.3	SLOVNÍK POJMOV.....	7
2	XML CLEANER - NORMALIZÁCIA XML.....	8
2.1	ARCHITEKTÚRA RIEŠENIA.....	8
2.1.1	Organizácia balíčkov a tried.....	10
2.1.2	Vstup a nastavenia.....	11
2.1.3	Spracovanie vstupného súboru.....	14
2.1.4	Vyčistenie a úprava dát do výsledného modelu.....	17
2.1.5	Výsledný dátový model.....	19
2.1.6	Detekcia jazyka a integrácia do systému.....	21
2.1.7	Výstup programu.....	22
2.2	ZLE FORMOVANÉ XML A JEHO RIEŠENIA.....	23
2.2.1	Dobre formovaný dokument.....	23
2.2.2	Logická štruktúra dokumentu a elementy.....	24
2.2.3	Prológ - XML deklarácia a DTD.....	27
2.2.4	Elementy a atribúty.....	28
2.2.5	Textový obsah elementu.....	31
2.2.6	Komentáre.....	34
2.2.7	Procesná Inštrukcia.....	35
2.2.8	CDATA sekcia.....	36
2.2.9	Menné priestory.....	36
2.2.10	XML 1.0 vs. XML 1.1.....	39
2.3	ZÁVER A MOŽNÉ ROZŠÍRENIE FUNKČNOSTI.....	39
3	DETEKCIA JAZYKA.....	41
3.1	METÓDY A NÁSTROJE PRE DETEKCIU JAZYKA.....	41
3.1.1	Krátko slová a unikátne kombinácie písmen.....	42
3.1.2	Štatistická metóda.....	42
3.1.3	N-gramová metóda.....	43
3.1.4	Vyhodnotenie a porovnanie metód.....	43
3.2	N-GRAMOVA METÓDA.....	44
3.2.1	Vybudovanie referenčného modelu.....	44
3.2.2	Výpočet skóre.....	44
3.3	ARCHITEKTÚRA RIEŠENIA A IMPLEMENTÁCIA.....	46
3.3.1	Vybudovanie modelu.....	46
3.3.2	Detekcia jazyka.....	49
3.3.3	Organizácia tried a dátový model.....	50
4	TESTOVANIE A VÝSLEDKY.....	52
4.1	TESTOVANIE OPRAVY ZLE FORMOVANÉHO XML.....	52
4.1.1	Zdroje dát zle formovaného XML.....	52
4.1.2	Postup pri testovaní zle formovaného XML.....	53
4.1.3	Výsledky a záver testov normalizácie XML.....	53
4.2	TESTOVANIE IDENTIFIKÁCIE JAZYKA V TEXTE.....	53
4.2.1	Zdroje dát pre modely a vstupné súbory.....	54
4.2.2	Postup pri testovaní identifikácie jazyka.....	54
4.2.3	Výsledky a záver testovania identifikácie jazyka.....	54
4.3	TEST IDENTIFIKÁCIE JAZYKA HTML STRÁNOK.....	56
4.3.1	Zdroje dát HTML stránok.....	56
4.3.2	Postup pri testovaní identifikácie jazyka z HTML.....	56
4.3.3	Výsledky a záver testovania identifikácie jazyka z HTML.....	56
5	ZÁVER.....	58
6	REFERENCIE.....	59
7	OBSAH DVD.....	60

Názov práce: Normalizace dat ve fulltextovém systému

Autor: Matúš Kapusta

Katedra: Katedra softwarového inženýrství

Vedúci diplomovej práce: Mgr. Jan Lánský, Ph.D.

E-mail vedúceho: zizelevak@gmail.com

Abstrakt

Cieľom tejto práce je návrh a implementácia programu v jazyku Java pre spracovanie dát fulltextového systému do výslednej podoby dobre formovaného XML podľa špecifikácie XML 1.0. Vstupné dáta sú vo forme ľubovoľného zle formovaného XML súboru ako napríklad HTML dáta. Dôležitými kritériami pri úpravách vstupných dát sú zachovať vernosť štruktúry a obsahu vstupného súboru v čo najvyššej miere. Program kontroluje správnosť menných priestorov a umožňuje nahradenie špeciálnych HTML entít skutočnými Unicode znakmi. Výstupný súbor musí byť možné spracovať použitím štandardných parserov. Sekundárnym cieľom je preskúmať a implementovať vhodnú metódu pre identifikáciu jazyka textového obsahu výsledného XML dokumentu ako celku, obsahu jednotlivých elementov alebo jednoduchého textového súboru.

Kľúčové slová: XML, dobre formované, identifikácia jazyka, n-gram

Title: Fulltext system data normalization

Author: Matúš Kapusta

Department: Department of Software Engineering

Supervisor: Mgr. Jan Lánský, Ph.D.

Supervisor's e-mail: zizelevak@gmail.com

Abstract

The purpose of this thesis is to design and implement Java application to process data from full text system to well-formed XML data according to XML 1.0 specification. Input data are stored in XML files containing any kind of not well formed XML, typically HTML content. Major criteria are conservation of the structure and conservation of the text content as much as possible. Program verifies if namespaces are correct and allows replacement of special HTML entities by appropriate Unicode characters. The output file has to be correctly processed by standard XML parsers.

Secondary objective is to investigate and implement suitable method to identify language of the resulting output file as a whole, the content of individual elements or plain text file.

Key words: XML, well-formed, language identification, n-gram

1 Úvod

Internetové stránky obsahujú obrovské množstvo rôznych textových dát. Spracovať tieto dáta do fulltextového vyhľadávacieho systému vyžaduje tieto dáta zozbierať kategorizovať a indexovať. Jedným z takýchto vyhľadávacích nástrojov je aj fulltextový vyhľadávací nástroj *Egothor* [19]. Spracovanie dát tohto systému vyžaduje analýzu textového obsahu z formátu HTML. Napriek tomu, že HTML ako formát má jasnú špecifikáciu a pravidlá, tieto pravidlá sú málokedy rešpektované a dodržiavané. Je to zapríčinené nedôslednosťou ich tvorcov a podporované aj vysokou mierou tolerancie moderných internetových prehliadačov. HTML ako formát sa takisto vyvíja a mení. To sú niektoré z mnohých ďalších dôvodov prečo nie je priamočiary spôsob, ako zo samotného HTML dokumentu získať textový obsah stránok.

Hlavnou myšlienkou tejto práce je umožniť jednoduché štandardizované spracovanie obsahu HTML stránok bežnými nástrojmi. Ako najvhodnejšia sa javí cesta prevodu HTML do formátu XML nakoľko obidva tieto jazyky vychádzajú z metajazyka SGML (Standard Generalized Markup Language). XML je jasne definovaný štandardizovaný formát kde za splnenia určitých predpokladov je možné jeho obsah spracovať pomocou mnohých štandardných nástrojov. Základným predpokladom je vytvoriť dobre formovaný dokument podľa podmienok definovaných v špecifikácii XML. Práca sa ale nevymedzuje na spracovanie súboru, ktorého zdrojom je HTML, konečným cieľom by malo byť spracovanie akéhokoľvek zle formovaného XML dokumentu, spracovanie HTML dát bolo len prvotnou inšpiráciou.

Z HTML dát korektne prevedených do dobre formovaného XML je jednoduchým spôsobom možné získať ich pôvodný textový obsah a ďalej ho spracovať. Pre ďalšie spracovanie sa ponúka veľa možností, pre účely tejto práce bola zvolená kategorizácia obsahu podľa jazyka. K tomu je potrebné navrhnúť a preskúmať metódy pre tento účel vhodné.

1.1 Vymedzenie cieľov

Primárnym cieľom tejto práce je navrhnúť a implementovať program ktorý umožní spracovať akýkoľvek vstupný súbor takým spôsobom, aby výsledkom bol dobre formovaný XML dokument s možnosťou identifikovať jazyk jeho obsahu. Predpokladaným vstupom sú HTML dáta ale program musí vedieť spracovať akýkoľvek vstupný súbor.

Výsledné práca by malo spĺňať nasledujúce požiadavky:

- výstupný súbor je je dobre formovaný XML dokument podľa pravidiel najnovšej špecifikácie XML 1.0 [1], tým sa myslí najmä
 - rešpektovanie pravidiel gramatiky XML
 - zachovanie správneho hniezdenia elementov
 - oprava nesprávnych entitných referencií a nahradenie špeciálnych HTML entít patričnými znakmi Unicode pre následné spracovanie textového obsahu

- odstránenie opakovaných atribútov a priradenie hodnoty atribútom ktoré ich nemajú
 - kontrola platných znakov
- pri spracovaní a úpravách dát vstupného súboru je zachovaná maximálna možná vernosť pôvodnému dokumentu a to nielen pre jeho textový obsah ale aj štruktúru (mená a štruktúra elementov, atribúty, procesné inštrukcie, komentáre ...) tak, aby bol použiteľný nielen pre HTML ale opravu akéhokoľvek XML dokumentu
- vstupný súbor nemusí mať definované kódovanie a preto ho treba správne identifikovať
- výstupný súbor bude rešpektovať štandardy pre menné priestory definované v špecifikácii menných priestorov XML 1.0 [14]
- pre vybrané elementy alebo celý dokument je možné identifikovať jazyk ich obsahu a priradiť ho ako hodnotu atribútu `xml:lang` na daný element
- umožniť program „naučiť“ identifikovať nový jazyk

Ako jednoduchá kontrola toho, či je výsledný súbor dobre formovaný, poslúži úspešné spracovanie parserom SAX. Pre splnenie požiadavky identifikácie jazyka sa urobí krátky prehľad metód pre identifikáciu jazyka, zvolí sa vhodná metóda ktorá sa implementuje a otestuje na reálnych dátach.

1.2 Štruktúra práce

Práca je rozdelená do šiestich samostatných kapitol. Po úvode nasleduje kapitola zaoberajúca sa spracovaním zle formovaných XML dát. Prvá polovica tejto kapitoly obsahuje celkový náhľad na architektúru riešenia a popisuje postupy a návrh implementácie spracovania zle formovaného XML súboru. V druhej časti sú detailne analyzované všetky úpravy zle formovaného XML, ktoré program prevádza, spolu s príkladmi ilustrujúcimi jednotlivé situácie.

Nasledujúca kapitola sa zaoberá problematikou identifikácie jazyka. Prináša krátky prehľad rôznych metód a ich teoretické pozadie. Podrobne popisuje N-gramovú metódu identifikácie jazyka ako metódu zvolenú pre implementáciu v rámci tejto práce. Záverečná časť tejto kapitoly popisuje architektúru riešenia, návrh a postupy použité pri implementácii zvolenej metódy.

Štvrtá časť obsahuje popis testov ktoré boli v priebehu prác realizované spolu s interpretáciou ich výsledkov.

Záver, zhrnutie splnených cieľov a možný výhľad pre budúce rozšírenia práce cieľov sú obsahom piatej kapitoly.

Na konci práce je uvedený zoznam použitej literatúry a v prílohe popis obsahu priloženého DVD ktoré obsahuje implementáciu, užívateľskú príručku a všetky testovacie súbory použité pre účely tejto práce spolu s výsledkami ich spracovania.

1.3 Slovník pojmů

Nakoľko je táto práca v slovenskom jazyku, považujem za vhodné uviesť krátky slovník pojmov. Keďže takmer úplne všetky materiály a referencie z ktorých som čerpal boli v anglickom jazyku, nebolo vždy prirodzené zvoliť ten správny termín. Vychádzal som z termínov použitých knihe *XML v kostce - Pohotovú referenční příručka* [4].

- *start (end) tag* - počiatočná (koncová) značka
- *processing instruction* - procesná inštrukcia (česká literatúra využíva “zpracovací instrukce”)
- *namespace* - menný priestor
- *DTD (Document type definition)* - používam skratku DTD
- *DTD Internal subset* - interná podmnožina DTD
- *package* - balíček
- *unparsed entity* - nespracovaná entita
- *legal character* - platný znak
- *character data* - textový obsah

2 XML Cleaner - normalizácia XML

Predmetom tejto kapitoly je rozbor problematiky zle formovaného XML a návrh riešenia. Predpokladaným vstupom je zle formovaný XML súbor alebo adresár obsahujúci prevažne dáta získané zberom HTML, ktoré poskytuje projekt *Egothor* [19], nie je to však nutné. Program funguje pre akýkoľvek zle formovaný XML súbor. Výstupom programu je súbor alebo adresár obsahujúci dáta, ktoré sú dobre formované. Dôraz je kladený na zachovanie maximálnej vernosti obsahu pôvodného súboru s minimom nutných zmien.

V prvej časti je vysvetlená architektúra riešenia a návrh postupov využitých pri implementácii, to zahŕňa tieto oblasti:

- organizácia balíčkov tried
- spracovanie vstupu, popis možností a vstupných nastavení ktoré program umožňuje a implementuje
- popis spracovania vstupných súborov a vytvorenie dátového modelu nad vstupným súborom
- postup pri úprave dátového modelu pre potreby získania dobre formovaného XML
- integrácia rozpoznávania jazyka
- postup pri spracovaní výstupného súboru

V druhej časti sú analyzované jednotlivé situácie zle formovaného XML na konkrétnych prvkoch jazyka XML, ilustrované na príkladoch.

Táto kapitola nepopisuje postupy a procesy pri rozpoznávaní jazyka. Tieto postupy sú detailne vysvetlené v samostatnej časti práce.

2.1 Architektúra riešenia

Program je realizovaný ako Java aplikácia na platforme Java SE 6 a spúšťa sa z príkazového riadku. Vstupom aplikácie sú parametre z príkazového riadku a vstupné súbory, výstupom aplikácie je výstupný súbor. Alternatívne môže byť vstupom adresár so vstupnými súborami a výstupom rovnako adresár ktorý bude obsahovať spracované súbory zo vstupného adresára.

Celkový náhľad na proces ilustruje nasledujúci diagram:

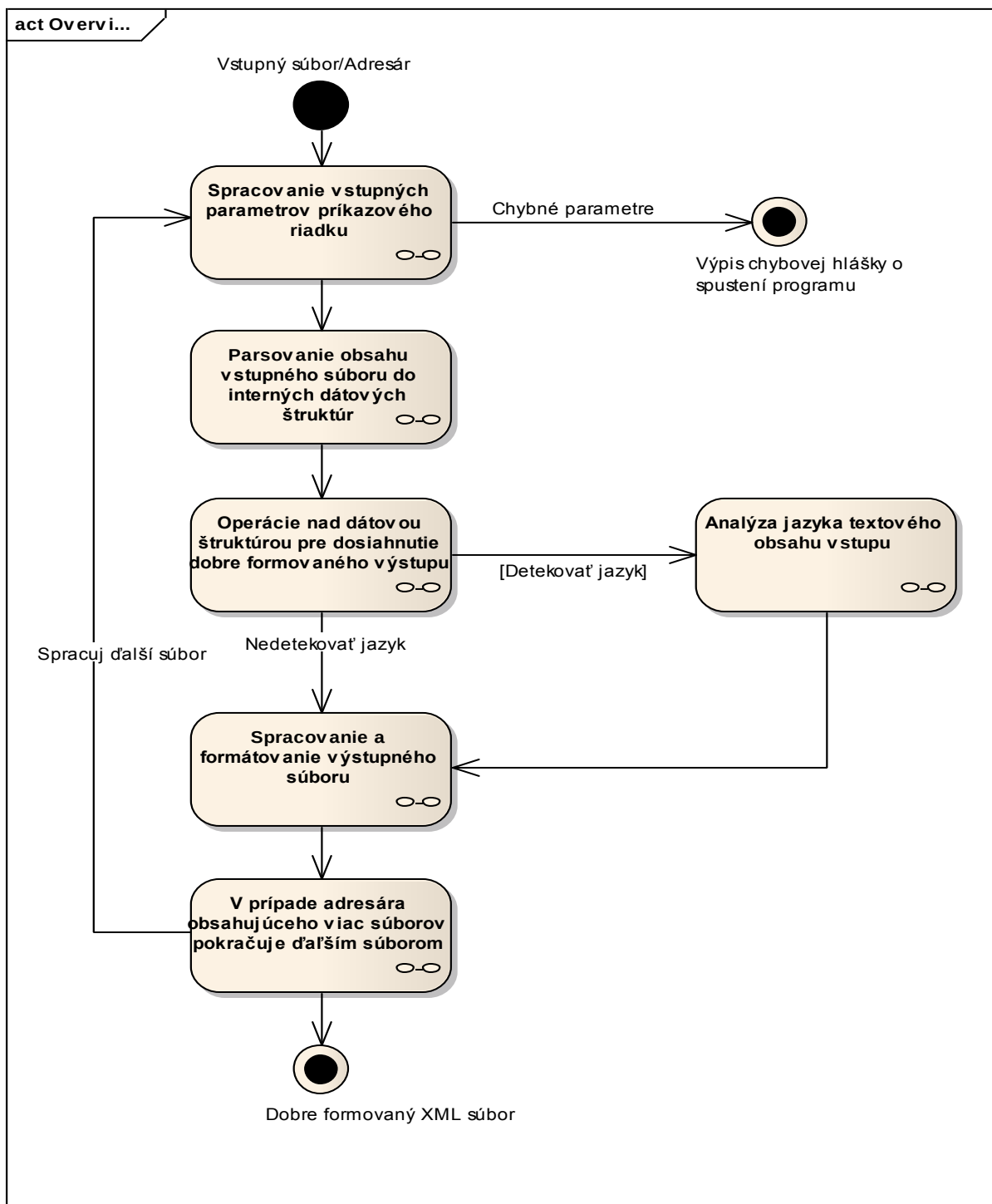


Diagram 1: Spracovanie súboru zle formovaného XML

V prvom kroku sú spracované vstupné parametre, následne sa spracuje priamo súbor zo vstupu alebo postupne jednotlivé súbory zo vstupného adresára. V prvom kroku sa obsah súboru prasaže a načíta do internej dátovej štruktúry. Potom sa obsah upraví aby vyhovoval všetkým podmienkam pre dobre formovaný dokument. Ak to vstup vyžaduje, pre textový obsah daných elementov sa vykoná detekcia jazyka. V závere sa vytvorí požadovaný formát výstupného súboru. Ak je na vstupe adresár, pokračuje sa spracovaním všetkých jeho súborov. Obsahom tejto kapitoly je tiež popis organizácie balíčkov a tried aplikácie a detailný popis spracovania od vstupného po výstupný súbor.

2.1.1 Organizácia balíčkov a tried

Java triedy sú v aplikácii organizované do balíčkov ako ukazuje nasledujúci diagram usporiadania:

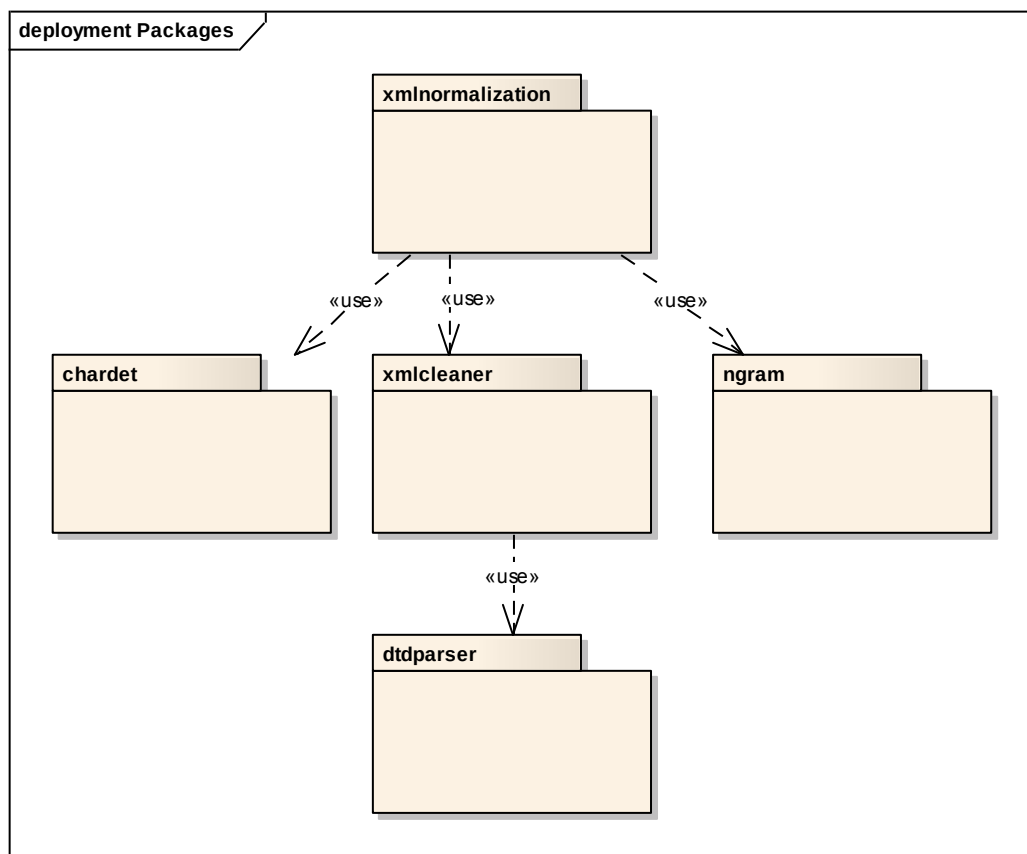


Diagram 2: Balíčky a triedy

Balíčky obsahujú triedy podľa ich určenia:

- **xmlnormalization** - základné triedy pre spracovanie vstupu a výstupu, integruje ostatnú funkcionality ako je spracovanie kódovania a rozpoznávanie jazyka
- **chardet** - triedy pre rozpoznanie a konverziu kódovania
- **xmldcleaner** - triedy pre spracovanie a úpravu XML dokumentu, základom pre zdrojový kód je projekt HtmlCleaner [5] ktorý je ale do značnej miery modifikovaný
- **dtdparser** - triedy pre spracovanie DTD sekcie obsahujú prebratý zdrojový kód z projektu M. Wutku [6]

Projekt využíva nasledovné knižnice:

- **jdom 1.1.1** - pre objektové spracovanie dobre formovaného výsledného XML modelu a na overenie či je výstupom dobre formovaný XML dokument [10]
- **log4j 1.2.15** - knižnica pre tvorbu systémových logov [11]
- **icu4j 4.0** - knižnica pre rozpoznávanie kódovania súboru [12]

Jadrom časti aplikácie pre normalizáciu XML sú triedy v balíku `xmldcleaner`. Tieto triedy vychádzajú zo zdrojových kódov projektu HtmlCleaner, ktorý je však zásadným spôsobom prepracovaný. Pre porovnanie je k dispozícii aj pôvodný kód projektu HtmlCleaner na DVD.

Prehľad jednotlivých zmien a modifikácií oproti projektu HtmlCleaner:

- Odstránenie častí kódu špecifických pre spracovanie HTML, nakoľko pôvodný program vychádza z predpokladanej štruktúry HTML dokumentu a cieľom tejto práce je spracovať akýkoľvek XML dokument
- Úplne prepracovanie menných priestorov v súlade so špecifikáciou pre menné priestory v XML 1.0 [14]
- Odstránenie zbytočných modifikácií pôvodného dokumentu nepotrebných pre získanie dobre formovaného dokumentu (cieľ: dobre formovaný dokument po spracovaní a výstupe by mal zostať rovnaký ako bol na vstupe)
- Pridanie spracovania pre komentáre a procesné entity
- Pridanie spracovania XML deklarácie
- Prepracovanie spracovania DTD sekcie
- Pridané spracovanie entít deklarovaných v internom DTD odkazovaných v textovom obsahu a v hodnotách atribútov
- Zmena validácie platných znakov, mien atribútov, mien elementov aby odpovedala pravidlám gramatiky XML 1.0 5. edície

2.1.2 Vstup a nastavenia

Aplikácia sa spúšťa z príkazového riadku ako java aplikácia príkazom "`java -jar xmlNorm.jar`". Vstupné parametre aplikácie:

- `src` - povinný parameter, meno vstupného súboru alebo adresára
- `dest` - meno výstupného súboru alebo adresára, pokiaľ nie je uvedené použije sa tvar `out.<vstupny_subor>`
- `incharset` - kódovanie vstupného súboru, pokiaľ nie je uvedené kódovanie sa detekuje automaticky
- `outputformat` - typ výstupného formátovania súboru, pokiaľ nie je uvedené, použije sa rovnaké nastavenie „original”
 - `original` - formát výstupného súboru je rovnaký ako formát vstupu
 - `compact` - formát výstupu je kompaktný, vynechajú sa všetky medzery, tabulátory a konce riadkov
 - `formatted` - výstup je formátovaný tak že každý vnorený element je odsadený tabulátorom
- `langtag` - meno tagu pre ktorý sa má vykonať detekcia jazyka, pokiaľ sa nepoužije detekcia jazyka sa nerobí, pri nastavení na hodnotu „<root>“ sa detekuje pre koreňový element a teda celý dokument bez ohľadu na meno koreňového elementu
- `modeldir` - adresár obsahujúci referenčné n-gramové modely jazykov pre identifikáciu jazyka
- `addroot` - určuje riešenie situácie v prípade, že dokument má viac koreňových elementov, východzie nastavenie je „ifmissing”
 - `ifmissing` - pridá koreňový element v prípade ak je to potrebné
 - `never` - nepridá koreňový element a použije ako koreňový element prvý element a jeho obsah, ostatné elementy sa nepoužijú
 - `always` - pridá koreňový element vždy aj pre dobre formovaný dokument

- `attrvalue` - určuje nastavenie hodnôt pre atribúty ktoré nemajú žiadnu hodnotu, východzie nastavenie je „self“
 - `self` - ako hodnota sa použije meno atribútu
 - `empty` - ako hodnota sa použije prázdny reťazec
 - `true` - ako hodnota sa použije „true“
- `ignorens` - umožňuje nastaviť ignorovanie menných priestorov, východzie nastavenie je „no“
 - `no` - menné priestory sú spracované a opravené podľa špecifikácie pre menné priestory [14]
 - `yes` - menné priestory sa neberú do úvahy a mená atribútov, elementov, entít a procesných inštrukcií sú spracované striktne podľa podmienok špecifikácie XML 1.0 [1]
- `ignoreddtd` - nastavuje ignorovanie DTD, východzie nastavenie je “no”
 - `no` - DTD sekcia sa použije v prípade ak je jej obsah dobre formovaný, entity z DTD sa zostanú zachované v hodnotách atribútov a textovom obsahu
 - `yes` - DTD sa vypustí a všetky entity budú nahradené ako nedefinované
- `ignorexmldecl` - nastavuje ignorovanie XML deklarácie, východzie nastavenie je “no”
 - `no` - XML deklarácia sa použije pokiaľ sa nachádza v pôvodnom dokumente
 - `yes` - XML deklarácia sa nepoužije
- `specialentity` - určuje či nahradiť špeciálne HTML entity znakmi podľa špecifikácie HTML 4.0 [9], východzie nastavenie je “yes”
 - `yes` - špeciálne entity sa nahradia znakmi (´ -> á)
 - `no` - špeciálne entity sa spracujú rovnako ako ostatné entity

Nasledujúci diagram ukazuje postup pri spracovaní vstupných parametrov a spustení spracovania súboru:

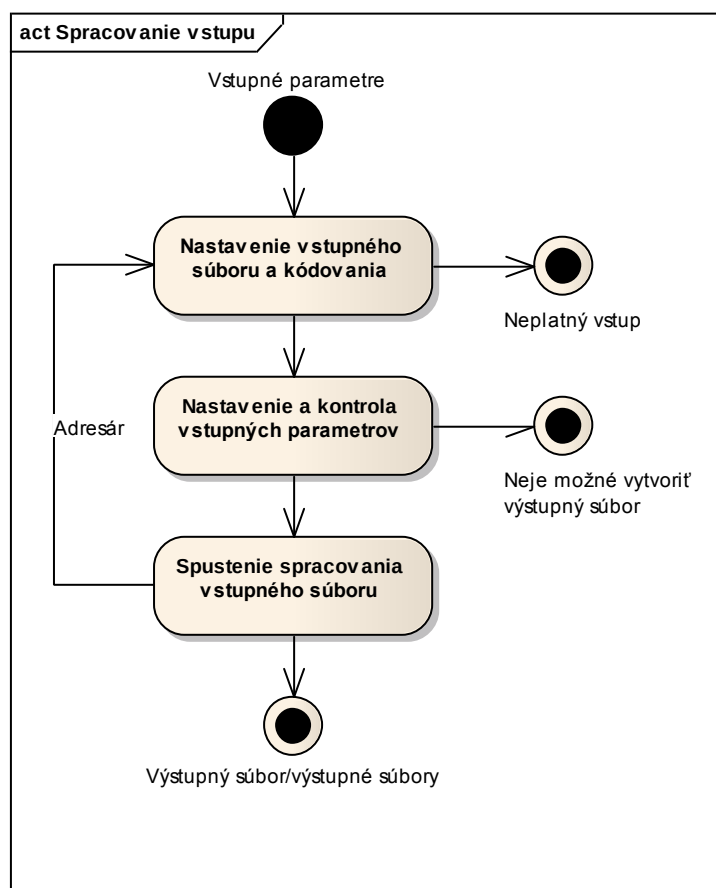


Diagram 3: Spracovanie vstupu

V prvom kroku sú spracované meno vstupného súboru alebo adresára a jeho kódovanie. V prípade neplatného mena vstupu sa program ukončí a vypíše štandardnú hlášku o neplatných vstupných parametroch. Ak nie je uvedené kódovanie, vykoná sa automatická detekcia kódovania vstupného súboru.

V druhom kroku sa nastaví hodnoty ostatných parametrov spracovania. Pokiaľ sú hodnoty vstupných parametrov neplatné alebo nedefinované, použijú sa východzie hodnoty parametrov. Ak je na vstupe adresár a nie je uvedený výstupný adresár, vytvorí sa automatický výstupný adresár „out“ ako podadresár vstupného adresára. Pokiaľ nie je možné vytvoriť výstupný súbor alebo adresár, program skončí výpisom chybovej hlášky.

V poslednom kroku sa spustí spracovanie vstupného súboru alebo prvého súboru zo vstupného adresára po jednotlivých znakoch. Pokiaľ je na vstupe adresár, tak po dokončení spracovania, sa spracuje ďalší súbor z adresára až kým nie sú spracované všetky súbory vstupného adresára.

2.1.3 Spracovanie vstupného súboru

Samotné parsovanie obsahu XML súboru po znakoch je realizované vo viacerých krokoch. Nasledujúci diagram zobrazuje postupnosť jednotlivých krokov parseru.

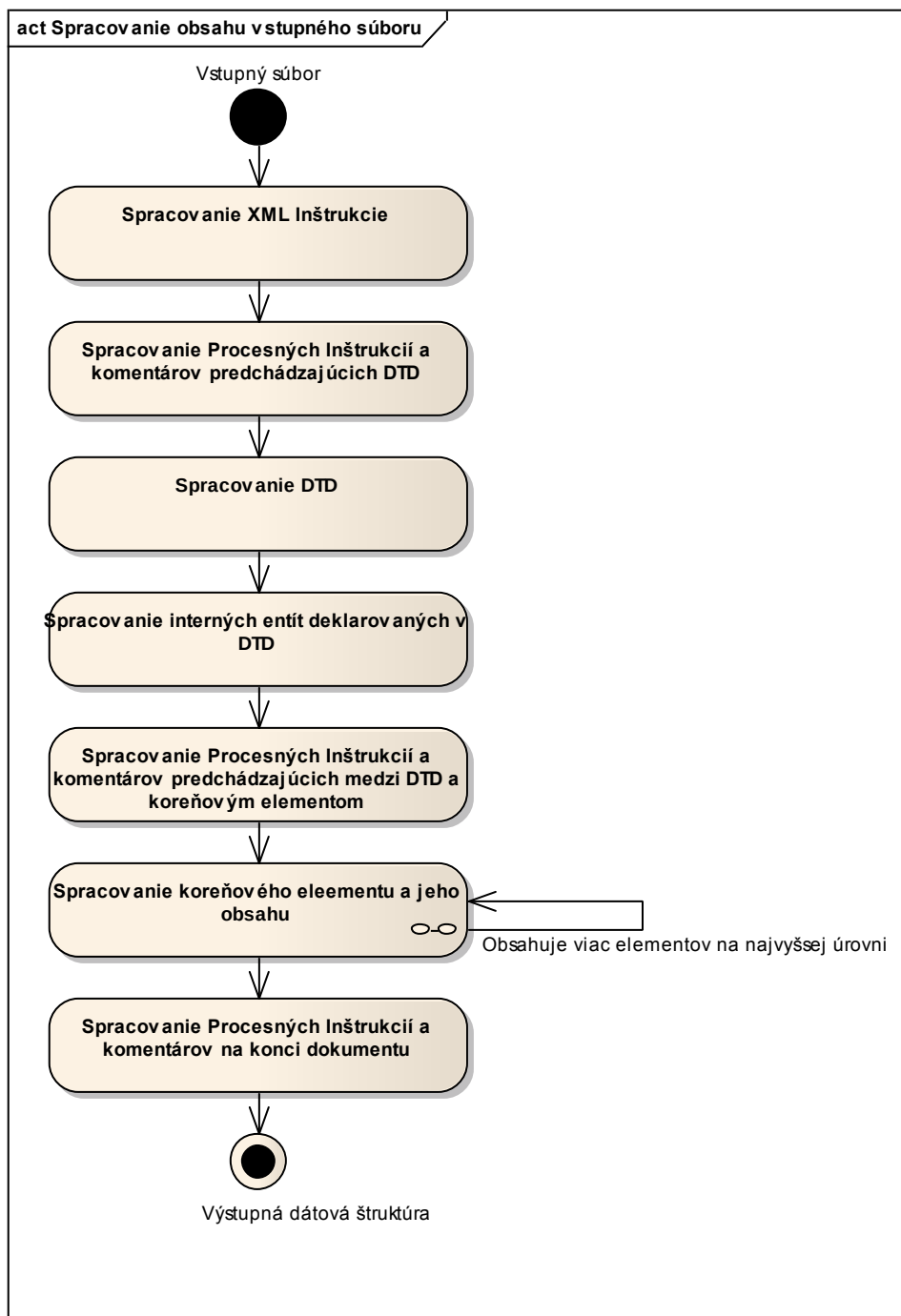


Diagram 4: Spracovanie obsahu vstupného súboru

V prvej fáze je spracovaný začiatok dokumentu, tá zahŕňa spracovanie obsahu pred koreňovým elementom, samozrejme za podmienky, že dokument okamžite nezačína koreňovým elementom. V druhej fáze je spracovaný obsah dokumentu.

2.1.3.1 Spracovanie XML deklarácie, DTD a entitných deklarácií

V prvom kroku sa parsuje XML deklarácia a prípadné procesné inštrukcie a komentáre ktoré predchádzajú DTD. Detaily o spôsobe spracovania obsahuje ďalšia kapitola. DTD sekcia sa parsuje ako celokm oprava DTD sekcie nieje predmetom tejto práce. Následne sa pomocou štandardného XML parseru [10] overí či je sekcia ako celok dobre formovaná. Ak áno uloží sa pre ďalšie použitie, inak sa vypustí. V prípade externej deklarácie DTD sa obsah externého súboru nekontroluje.

V druhom kroku sa DTD vyhodnotí obsah deklarovaných entít, v rámci internej podmnožiny DTD. Pre každú entitu sa vyhodnotí:

- či je parametrická alebo entitná
- jej konečná hodnota bez ohľadu na to či sa jedná o priamu alebo nepriamu deklaráciu cez ďalšiu entitu či už parametrickú alebo entitnú
- označí sa ak je deklarovaná ako externá alebo odkazuje na externú entitu
- označí sa ak odkazuje sama na seba či už priamo alebo nepriamo
- vyhodnotí sa či jej hodnota môže byť odkazovaná v hodnote atribútu
- vyhodnotí sa či jej hodnota môže byť odkazovaná v textovom obsahu

Všetky informácie o entitách sa uložia pre ďalšie spracovanie obsahu súboru, kde môžu byť entity odkazované v hodnotách atribútov a textovom obsahu elementov.

2.1.3.2 Spracovanie elementu a jeho obsahu

V druhej fáze sa po znakoch parsuje samotný obsah dokumentu - jednotlivé elementy a ich obsah. Druhá fáza sa spustí okamžite ako sa v dokumente vyskytne prvý element. Následne je akákoľvek XML deklarácia spracovaná rovnako ako zle formovaná procesná inštrukcia a DTD sekcia je ignorovaná.

Základná schéma spracovania elementov a ostatných konštrukcií jazyka XML obsiahnutých v elemente, ukazuje nasledujúci diagram:

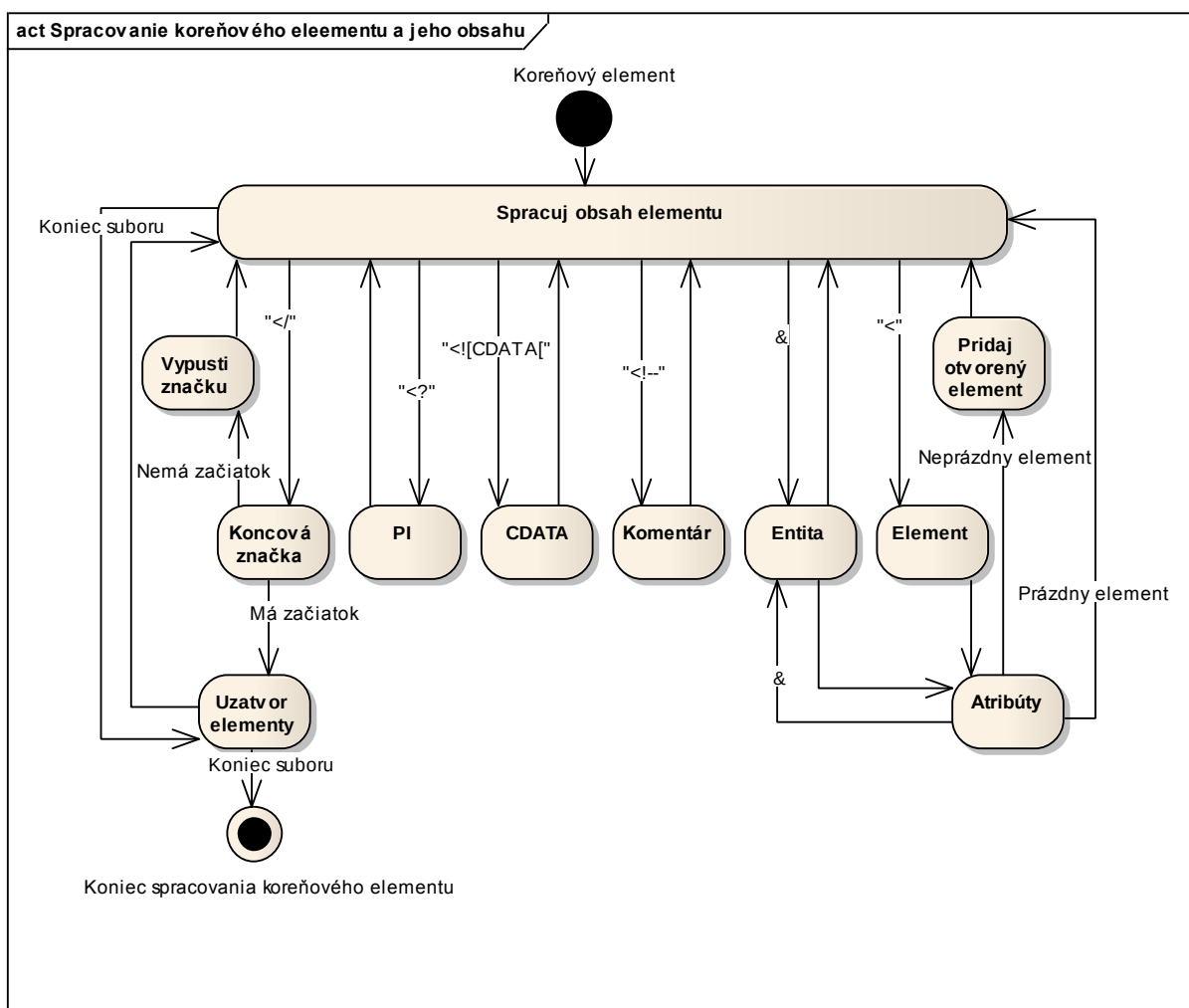


Diagram 5: Spracovanie elementu a jeho obsahu

Až kým nie je dosiahnutý koniec súboru je postupne po jednotlivých znakoch parsovaný obsah elementu. Postup spracovania obsahu elementu:

1. Za predpokladu úspešného načítania začiatkovej značky sa pridá otvorený element do zásobníka otvorených elementov, výnimkou je prázdny element - `<empty/>`
2. Spracujú sa atribúty elementu, v prípade výskytu entity v hodnote atribútu sa overí či je entita deklarovaná a použiteľná z pripraveného zoznamu vyhodnotených entít z DTD, ak je nedeklarovaná, externá alebo zle formovaná, spracuje sa ako nedefinovaná.
3. Spracuje sa na základe počiatkových znakov obsah elementu, ktorý sa s výnimkou koncovkej značky pridá do zoznamu obsahu elementu. Spracovanie podľa priorit
 - a. `</>` - koncová značka
 - b. `<?>` - procesná inštrukcia
 - c. `<![CDATA` - CDATA sekcia
 - d. `<!--` - komentár
 - e. `<>` - synovský element
 - f. znaky pokiaľ sú platné sa pridávajú ako textový obsah elementu, ukončenie spracovania textového obsahu v prípade výskytu vyššie uvedených reťazcov

4. V prípade spracovania textového obsahu sa môže vyskytnúť entitná referencia ktorá sa spracuje analogicky ako pri hodnote atribútu v bode 2.
5. Pri ukončení spracovania koncovej značky sa vyhľadá posledne pridaný otvorený element rovnakého mena. Tento element sa označí ako uzatvorený a zo zoznamu odstráni. Rovnako sa uzavru aj všetky otvorené elementy ktoré boli potomkom práve uzatvoreného elementu a odstránia sa zo zoznamu otvorených elementov.
6. Pri dosiahnutí konca súboru sa uzavru všetky ešte otvorené elementy pokiaľ existujú a zoznam otvorených elementov sa vyprázdni.

V prípade korektného uzatvorenia elementu sa ešte spracujú prípadné komentáre a procesné inštrukcie a uložia ktoré sa podľa XML špecifikácie môžu nachádzať aj mimo elementov alebo na konci súboru po uzavretí koreňového elementu. Textový obsah a CDATA sekcie nachádzajúce sa mimo obsahu elementov sú ignorované a vo výstupnom súbore sa nepoužijú. Dátový model použitý pre spracovanie obsahu súboru je zobrazený v nasledujúcej kapitole. Detailné spracovanie jednotlivých chybových situácií pri spracovaní textového obsahu, počiatočných a koncových značiek elementov, atribútov a ich hodnôt, procesných inštrukcií, CDATA sekcií a komentárov je detailne rozobratý v kapitole 2.2.

2.1.4 Vyčistenie a úprava dát do výsledného modelu

Po ukončení spracovania obsahu súboru je potrebné získané dáta dodatočne spracovať aby vznikol dátový model z ktorého bude možné vytvoriť jednoduchou serializáciou požadovaný dobre formovaný XML dokument.

Dodatočné spracovanie obsahu, potrebné kroky a ich postupnosť zobrazuje nasledujúci diagram:

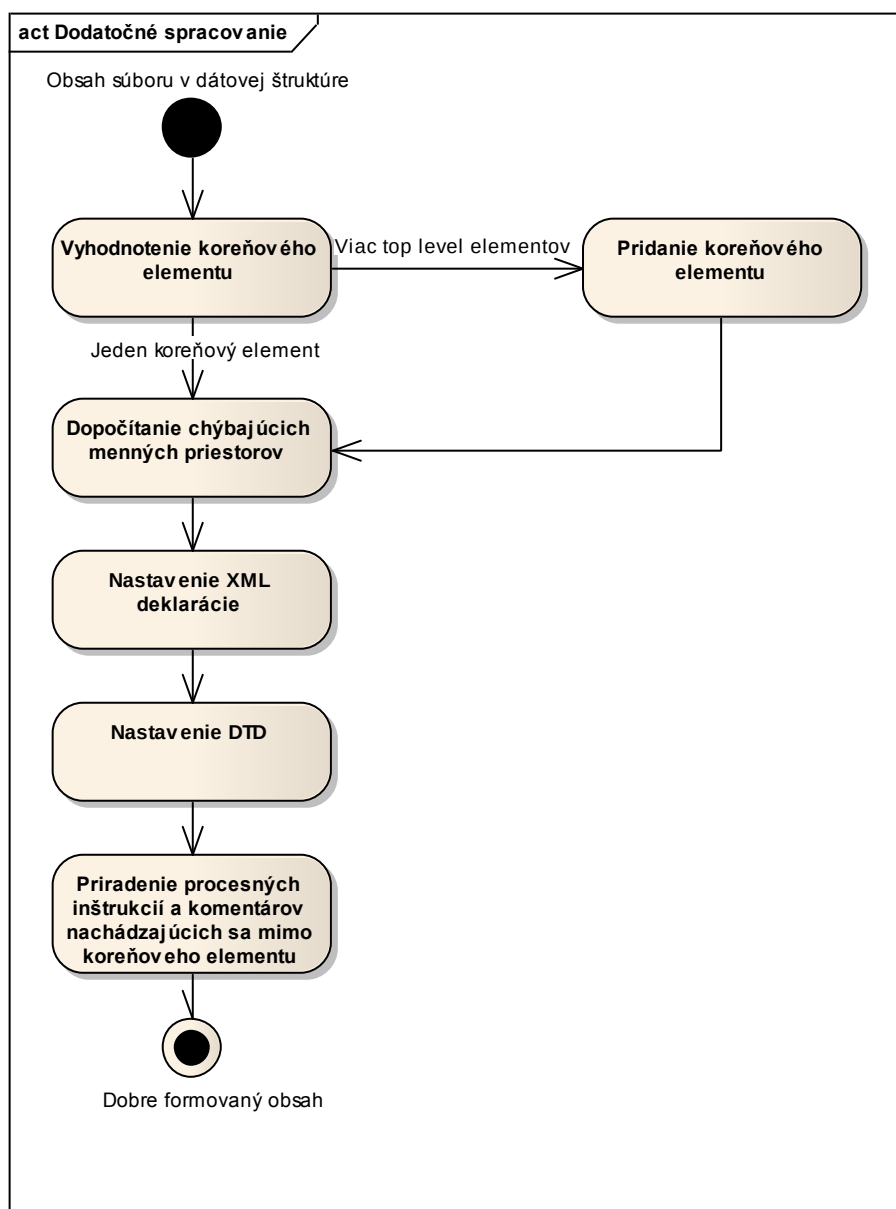


Diagram 6: Dodatočné úpravy

V prvom kroku sa skontroluje sa, či sa v spracovanom obsahu nachádza iba jeden koreňový element, na základe nastavenia vstupného parametru na hodnoty:

- `ifmissing` - pokiaľ je viac elementov na najvyššej úrovni, pridá sa nový koreňový element a všetky elementy sa nastavujú ako jeho potomkovia
- `never` - zoberie sa iba prvý nájdený element a jeho obsah sa použije ako obsah koreňového elementu
- `always` - vždy sa pridá koreňový element

Druhý krok - kontrola menných priestorov sa vykoná iba ak nie je vstupným parametrom nastavené ignorovanie menných priestorov (`ignorens=yes`). Pre všetky mená elementov a atribútov sa vyhodnotia menné priestory a v prípade, že nie sú deklarované tak ako určuje špecifikácia menných priestorov XML 1.0 [14], sú deklarácie menných priestorov pridané do

koreňového elementu. Detailný popis spracovania menných priestorov v kapitole 2.2.9 Menné priestory.

V treťom kroku sa podľa nastavenia vstupných parametrov `ignoredtd` a `ignorexmldecl` použije alebo zahodí XML deklarácia a DTD sekcia, pokiaľ sa nachádzali vo vstupnom súbore.

V poslednom kroku sa do modelu dokumentu pridajú spracované procesné inštrukcie a komentáre ktoré sa nachádzali mimo DTD a obsahu elementu. Výsledkom je naplnený dátový model pripravený na serializáciu do výstupného súboru.

2.1.5 Výsledný dátový model

Diagram tried zobrazuje objektový model aplikácie využitý pre spracovanie obsahu dokumentu. Po aplikácii všetkých predchádzajúcich krokov obsahuje dáta, z ktorých je serializáciou možné získať dobre formovaný XML dokument.

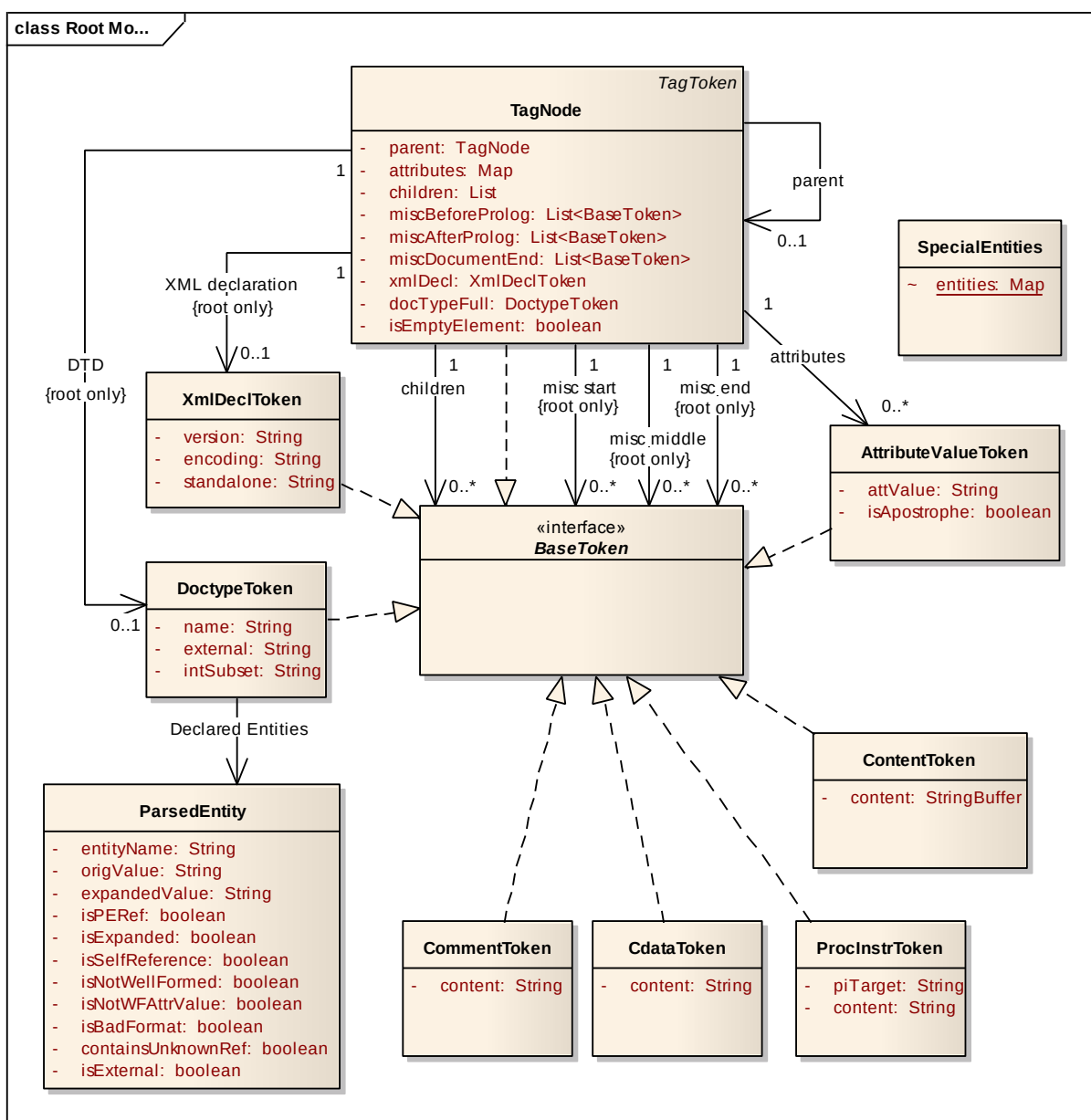


Diagram 7: Dátový model

TagNode - Trieda reprezentuje element. Obsahuje tieto atribúty:

- *parent* - odkaz na rodičovský element
- *attributes* - hash mapa atribútov elementu, kľúčom je meno atribútu
- *children* - zoznam reprezentujúci obsah elementu podľa poradia, môže obsahovať typy *TagToken*, *ContentToken*, *CommentToken*, *CdataToken*, *ProcInstrToken*
- *miscBeforeProlog*, *miscAfterProlog*, *miscDocumentEnd* (len koreňový element) - zoznamy komentárov a procesných inštrukcií nachádzajúcich sa mimo koreňového elementu
- *xmlDecl* (len koreňový element) - XML deklarácia
- *docTypeFull* (len koreňový element) - DTD sekcia
- *isEmptyElement* - príznak pre prázdny element

AttributeValueToken - Trieda obsahuje hodnotu atribútu.

- *attValue* - hodnota atribútu
- *isApostrophe* - príznak či je hodnota v úvodzovkách alebo apostrofoch

CdataToken, CommentToken, ProcInstrToken, ContentToken

Obsah CDATA sekcie, komentára, procesnej inštrukcie a textového obsahu elementu.

XmlDeclToken - Trieda reprezentujúca XML deklaráciu.

DoctypeToken - Trieda reprezentujúca DTD sekciu.

ParsedEntity - Trieda reprezentuje entity ktoré boli deklarované v DTD a využívajú sa pri spracovaní textového obsahu a hodnôt atribútov. Pre samotnú serializáciu už nemajú žiaden význam.

SpecialEntities - Statická trieda ktorá obsahuje mapovanie špeciálnych HTML entít na kódy znakov Unicode. Pre samotnú serializáciu už nemajú význam.

BaseToken - Interface ktorý implementujú všetky triedy, ktoré sú serializované do výstupného dokumentu.

2.1.6 Detekcia jazyka a integrácia do systému

Ak je na vstupne nastavený parameter `langtag`, prebehne detekcia jazyka. Výsledkom úspešnej detekcie jazyka je pridanie atribútu `xml:lang` s hodnotou ISO kódu detekovaného jazyka. Postup pri detekcii jazyka zobrazuje diagram:

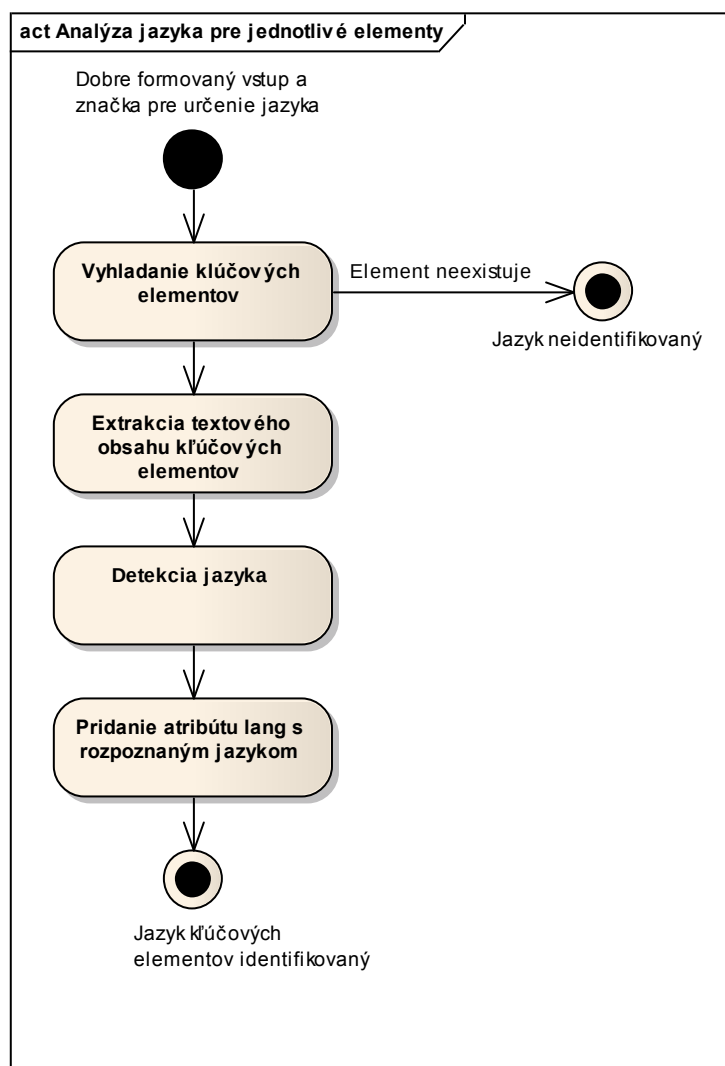


Diagram 8: Identifikácia jazyka pre element

V prvom kroku sa vyhľadávajú v dátovom modeli elementy, ktorých meno je uvedené ako hodnota vstupného parametru. V prípade nastavenia hodnoty na `<root>` sa identifikuje jazyk pre koreňový element.

V druhom kroku z daného elementu extrahuje textový obsah tohto elementu aj všetkých jeho potomkov (`ContentToken->content`). Pokiaľ je získaný obsah rozumne dlhý (25 znakov), spustí sa nad ním algoritmus pre detekciu jazyka. Samotný proces detekcie je detailne spracovaný v ďalšej časti práce v kapitole 3 Detekcia jazyka.

Pokiaľ bola detekcia úspešná, na daný element sa pridá nový atribút `xml:lang` ktorého hodnota sa nastaví na ISO kód detekovaného jazyka. Výsledná podoba bude potom vyzerat takto `<DOCUMENT num="734" xml:lang="cz">`.

2.1.7 Výstup programu

Každá trieda zo základného dátového modelu implementuje interface BaseToken ktorý obsahuje metódy pre serializáciu tried do výstupného textového reťazca. Postup pri serializácii dátového modelu je nasledovný:

1. Serializácia XML Deklarácie, pokiaľ ju model obsahuje
2. Serializácia Procesných Inštrukcií a Komentárov nachádzajúcich sa pred DTD
3. Serializácia DTD sekcie, pokiaľ ju model obsahuje
4. Serializácia Procesných Inštrukcií a Komentárov nachádzajúcich sa medzi DTD a koreňovým elementom
5. Serializácia koreňového elementu
6. Serializácia Procesných Inštrukcií a Komentárov nachádzajúcich sa na konci dokumentu

Na základe nastavenia parametru `outputformat` dochádza k formátovaniu výstupného súboru. Pri východnom nastavení sa formátovanie nemení. Vstupný aj výstupný súbor zostanú nezmenené.

Príklad 2171.xml

```
<breakfast_menu>
<food>
  <name>Belgian Waffles</name>
  <price>$5.95</price>
  <description>two of our famous Belgian Waffles</description>
  <calories>650</calories>
</food>
<food>
  <name>Strawberry Belgian Waffles</name>
  <price>$7.95</price>
  <description>light Belgian waffles</description>
  <calories>900</calories>
</food>
</breakfast_menu>
```

Pri nastavení formátovaného výstupu sú všetky vnorené elementy odsadené tabulátorom

Príklad 2172.xml

```
<breakfast_menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>two of our famous Belgian Waffles</description>
    <calories>650</calories>
  </food>
  <food>
    <name>Strawberry Belgian Waffles</name>
    <price>$7.95</price>
    <description>light Belgian waffles</description>
    <calories>900</calories>
  </food>
</breakfast_menu>
```

Pri nastavení kompaktného formátovania sa vypustia všetky tabulátory aj medzery vypustené.

```
<breakfast_menu>
<food>
<name>Belgian Waffles</name>
<price>$5.95</price>
<description>two of our famous Belgian Waffles</description>
<calories>650</calories></food>
<food>
<name>Strawberry Belgian Waffles</name>
<price>$7.95</price>
<description>light Belgian waffles</description>
<calories>900</calories></food></breakfast_menu>
```

2.2 Zle formované xml a jeho riešenia

Táto kapitola obsahuje detailný rozbor prípadov, ktoré nastávajú alebo môžu nastať a narušiť inak dobre formovaný dokument. Na začiatku je zhrnutie všetkých pravidiel ktoré dobre formovaný XML dokument musí dodržiavať a na ktoré sa druhá časť tejto kapitoly odvoláva. Druhá pre túto kapitolu kľúčová časť obsahuje rozbor jednotlivých prvkov jazyka XML s ako algoritmus opravuje jednotlivé chyby či už v gramatike alebo logickej štruktúre XML. Všetky prípady sú ilustrované na príkladoch spolu s ich riešením.

V závere tejto kapitoly je analyzovaná problematika dobre formovaného dokumentu z pohľadu menných priestorov a rozdiely medzi jednotlivými verziami edíciami XML.

2.2.1 Dobre formovaný dokument

Podľa najnovšej špecifikácie XML 1.0 [1] je XML dokument dobre formovaný vtedy keď:

1. Splňa pravidlá gramatiky dokumentu XML ako je uvedené v špecifikácii. Odkazy v práci na pravidlá gramatiky sú formou [G#], napríklad [G15] vyjadruje pravidlo 15 pre komentár
[15] Comment ::= '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
2. Splňa všetky podmienky dobre formovaného dokumentu
 - a. V internom DTD sa nesmú parametrické entitné referencie vyskytovať vo vnútri deklarácií, môžu sa vyskytnúť iba tam, kde sa vyskytujú samotné deklarácie – **PEs in Internal Subset** (ďalej odkazované ako [WFC1])
 - b. Externá sada DTD musí mať formát v súlade s pravidlom [G30] – **External Subset** [WFC2]
 - c. Obsah na ktorý odkazuje parametrická entita má formát v súlade s pravidlom [G31] - **PE Between Declarations** [WFC3]
 - d. Meno elementu v počiatočnej a koncovej značke sa musí zhodovať – **Element Type Match** [WFC4]
 - e. Meno atribútu v dobre formovanom XML dokumente sa v rámci počiatočnej značky elementu alebo prázdneho elementu nemôže vyskytnúť viac ako jeden krát - **Unique Att Spec** [WFC5]
 - f. Hodnoty atribútov nesmú obsahovať priame alebo nepriame entitné referencie na externé entity - **No External Entity References** [WFC6]
 - g. Hodnota atribútu ani nahradzujúci text na ktorý priamo alebo nepriamo odkazuje entitná referencia, nemôže obsahovať znak '<' - **No < in Attribute Values** [WFC7]

- h. Všetky znaky v dokumente musia byť platné znaky podľa [G2] - **Legal Character** [WFC8]
 - i. V dokumente ktorý obsahuje iba interné DTD bez parametrických entít alebo v standalone dokumente musí byť každá entitná referencia deklarovaná s výnimkou entít &, <, >, ', " - **Entity Declared** [WFC9]
 - j. Entitná referencia nesmie odkazovať na nespracovanú entitu, na túto môže odkazovať iba ak sa vyskytuje v hodnotách atribútov typu ENTITY alebo ENTITIES - **Parsed Entity** [WFC10]
 - k. Entita nemôže odkazovať rekurzívne na seba - **No Recursion** [WFC11]
 - l. Parametrická entita sa nesmie vyskytnúť mimo DTD - **In DTD** [WFC12]
3. Spracované entity na ktoré sa dokument odkazuje či už priamo alebo nepriamo byť rovnako dobre formované.

2.2.2 Logická štruktúra dokumentu a elementy

Z pravidiel [G1] vyplývajú isté požiadavky na celkovú logickú štruktúru dobre formovaného XML dokumentu:

- Dobre formovaný XML dokument musí obsahovať jeden alebo viacej elementov [G1]
- Dobre formovaný XML dokument musí obsahovať práve jeden koreňový element [G1]
- Každý nekoreňový element dobre formovaného XML dokumentu musí mať počiatočnú aj koncovú značku v rámci obsahu jedného rodičovského elementu (inak povedané – elementy sú správne zahniezdené jeden do druhého), vyplýva to z pravidiel gramatiky [G1], [G39] a [G43] a pravidla [WFC4]
- Musí spĺňať podmienku pre štruktúru dokumentu, kde dokument začína prológom pokračuje vlastným obsahom (koreňovým elementom) [G1]

2.2.2.1 Chýbajúce elementy

Porušenie prvej podmienky nastane vtedy ak dokument neobsahuje vôbec žiadne elementy, ale môže obsahovať XML deklaráciu, DTD a prípadne komentáre alebo procesné inštrukcie.

Príklad 2221.xml

```
<?xml version="1.0"?>
<!-- start of the document -->
<!DOCTYPE doc [
<!ENTITY ent1 "text">
<!ENTITY ent2 SYSTEM "001.ent">
]>
Some content text
<!-- end of the document -->
```

Výsledok po normalizácii:

```
<?xml version="1.0"?>
<!-- start of the document -->
<!DOCTYPE doc [
<!ENTITY ent1 "text">
<!ENTITY ent2 SYSTEM "001.ent">
]>
<!-- end of the document -->
<DOCUMENT name="test.xml">
</DOCUMENT>
```

Pre vyriešenie tohto problému sa obsah dokumentu ktorý nasleduje po prológu, zabalí do prednastaveného elementu a s atribútom obsahujúcim meno vstupného súboru <DOCUMENT name="file_name">, jedná sa málo pravdepodobnú situáciu, nakoľko sa nepredpokladá práca s XML dokumentmi, ktoré by neobsahovali žiadne elementy. Ako je vidieť na príklade pôvodné korektné komentáre zostanú na pôvodnom mieste, nepovolený textový obsah sa odstráni.

2.2.2.2 Viacnásobný výskyt koreňového elementu

Viacnásobný výskyt elementov na najvyššej - koreňovej úrovni, je porušením druhej podmienky. Môže nastať nasledovná situácia:

Príklad 2222.xml

```
<?xml version="1.0"?>
<!DOCTYPE doc [
<!ENTITY ent1 "text">
]>
<root1>text1</root1>
<!-- missing root -->
<root2>text2</root2>
```

Výsledok po normalizácii:

```
<?xml version="1.0"?>
<!DOCTYPE doc [
<!ENTITY ent1 "text">
]>
<DOCUMENT name="test.xml">
<root1>text1</root1>
<!-- missing root -->
<root2>text2</root2></DOCUMENT>
```

Riešenie v prípade nesplnenia druhej podmienky závisí od nastavenia parametru `addroot` na vstupe programu. Príklad ukazuje implicitné nastavenie tohto parametru. V tomto prípade sa situácia rieši automatickým pridaním koreňového elementu a atribútu s názvom vstupného súboru, rovnako ako v predchádzajúcom prípade. Existujúce elementy sa pridajú ako deti nového elementu. V prípade nastavenia vstupného parametru `addroot` na hodnotu `never`, sa prvý nájdený element použije ako koreňový, ostatné nasledujúce elementy na najvyššej úrovni sa odstránia. Komentáre alebo procesné inštrukcie zostanú zachované na pôvodnom mieste. V tejto konfigurácii bude výsledkom vyššie uvedeného príkladu:

```
<?xml version="1.0"?>
<!DOCTYPE doc [
<!ENTITY ent1 "text">
]>
<root1>text1</root1>
<!-- missing root -->
```

V prípade nastavenia parametru na hodnotu `always` sa pridá koreňový element s atribútom vždy aj keď je obsah dokumentu korektný.

2.2.2.3 Nesprávne hniezdenie elementov

Tretia podmienka sa týka nesprávneho hniezdenia elementov. Takáto situácia môže nastať napríklad v prípade zlého uzavretia elementov v prípade HTML tabuľky:

Príklad 2223.xml

```
<table>
<tr><td>1.1</td><td>1.2</tr></td>
<tr><td>2.1<td></td>2.2</td></tr>
</table>
```

Výsledok po normalizácii:

```
<table>
<tr><td>1.1</td><td>1.2</td></tr>
<tr><td>2.1<td></td>2.2</td></tr>
</table>
```

Riešením tejto situácie je uzavretie každého otvoreného elementu, vždy pred uzavretím jeho rodičovského elementu. V prípade existencie samostatnej koncovkej značky, sa značka vypustí. Na príklade v prvom riadku tabuľky je druhý stĺpec tabuľky uzavretý značkou `</td>` až potom ako je uzatvorený riadok značkou `</tr>`. Preto sa pridá koncový znak `</td>` pred koncovú značku rodičovského elementu. Následne koncový znak `</tr>` sa považuje za zbytočný lebo nemá počiatkovú značku, preto je odstránený.

V druhom riadku je chybné poradie uzatvorenia a otvorenia stĺpca, nakoľko výsledkom tejto chyby je dobre formované XML, riadok zostane nezmenený. Ďalej sa problematikou elementov zaoberá kapitola 2.2.4 Zlá logická štruktúra.

2.2.2.4 Zlá logická štruktúra

Pre splnenie štvrtej podmienky o správnej štruktúre dokumentu je potrebné v správne spracovať prológ dokumentu – XML deklaráciu, DTD a prípadné komentáre či procesné inštrukcie. K detailnému spracovaniu prológu viac v nasledujúcej kapitole 2.2.3. Možné komplikácie ktoré môžu nastať:

- Výskyt čohokoľvek nepovolených štruktúr pred XML deklaráciou
- Výskyt nepovolených iných štruktúr ako sú komentáre alebo procesné inštrukcie medzi deklaráciou a DTD, medzi DTD a koreňovým elementom alebo na konci dokumentu
- DTD alebo XML deklarácia sa nachádzajú až za prvým elementom
- Vyskytuje sa viac DTD sekcií

Prvé dva prípady ilustruje nasledujúci príklad:

Príklad 2224.xml

```
<?pil aaa?>
<!-- start of the document -->
<?xml version="1.0" encoding="UTF-8"?>
Invalid Text
<!DOCTYPE doc [
<!ENTITY ent1 "text">
<!ENTITY ent2 SYSTEM "001.ent">
]>
<![CDATA[ bad cdata ]]>
<DOCUMENT num="2224">
Some content
</DOCUMENT>
&ent1;
<!-- end of the document -->
```

Výsledok po normalizácii:

```
<?xml version="1.0" encoding="UTF-8"?>
<?pil aaa?>
<!-- start of the document -->
<!DOCTYPE doc [
<!ENTITY ent1 "text">
<!ENTITY ent2 SYSTEM "001.ent">
]>
<DOCUMENT num="2224">
Some content
</DOCUMENT>
<!-- end of the document -->
```

Pokiaľ sa pred XML deklaráciou vyskytuje komentár (alebo procesná inštrukcia), presunú sa až za XML deklaráciu pred DTD. Prípadný textový obsah, entita, CDATA sekcia alebo iný neplatný obsah, ktorý sa nesmie nachádzať nikde mimo koreňového elementu, sú vypustené.

Ďalším prípadom, keď sa koreňový alebo iný element nachádzajú už pred DTD sekciou:

Príklad 2225.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- start of the document -->
<DOCUMENT num="2225">
Some content
&ent1;
</DOCUMENT>
<!DOCTYPE doc [
<!ENTITY ent1 "text">
<!ENTITY ent2 SYSTEM "001.ent">
]>
<!-- end of the document -->
```

Výsledok po normalizácii:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- start of the document -->
<DOCUMENT num="2225">
Some content
&ent1;
</DOCUMENT>
<!-- end of the document -->
```

Riešením tejto chyby je vypustenie celej DTD sekcie. Prípadné entitné referencie deklarované vo vypustenej DTD sekcii sa v dokumente spracujú rovnakým spôsobom ako nedeclarované. Rovnako sa DTD sekcia vypustí aj v prípade ak sa nachádza ako obsah elementu.

Alternatívnou možnosť, ktorá nie je implementovaná, by bola presunúť DTD sekciu za XML deklaráciu a ďalej postupovať štandardným spôsobom, alebo v prípade výskytu DTD sekcie v obsahu takúto sekciu uzavrieť do sekcie CDATA aby sa nestratila pôvodná informácia. V prípade výskytu viacerých dobre formovaných DTD sekcií v jednom dokumente sa použije iba prvá DTD sekcia a to len v prípade ak je správne umiestnená.

2.2.3 Prológ - XML deklarácia a DTD

Prológ XML dokumentu tvorí XML deklarácia, DTD sekcia a prípadné komentáre alebo procesné inštrukcie, ktoré sa však nesmú nachádzať pred XML deklaráciou. Algoritmus rozlišuje nasledovné situácie:

- Správna logická štruktúra prológu [G22]
- Správny formát XML deklarácie pokiaľ sa v dokumente vyskytuje podľa [G23], [G24], [G32] a [G80]
- Správny formát DTD sekcie podľa [G28]

2.2.3.1 XML Deklarácia

Za existenciu XML deklarácie algoritmus považuje výskyt sekvencie znakov ‘<?xml’ bez ohľadu na veľkosť písmen, pred začiatkom DTD sekcie alebo koreňového elementu aj v prípade že deklarácia nie je dobre formovaná. Následne sa aplikujú tieto pravidlá:

- Iné atribúty ako version, standalone a encoding sú ignorované
- Pokiaľ nie je uvedená verzia XML automaticky sa použije “version=“1.0””
- Pokiaľ je verzia uvedená a je iná ako 1.0 alebo 1.1 použije sa “version=“1.0””
- Pokiaľ je uvedený atribút standalone iný ako “yes” alebo “no”, bez ohľadu na veľkosť písmen použije sa “standalone=“yes””
- Pokiaľ je uvedený atribút encoding použije sa aj vo výstupnom súbore
- Hodnota atribútu encoding pokiaľ je uvedený a nie je špecifikované výstupné kódovanie súboru sa nastaví “UTF-8”, čo je aj štandardné kódovanie výstupného súboru
- Pokiaľ je na vstupe uvedené kódovanie výstupného súboru a zároveň sa vo vstupnom súbore vyskytuje XML deklarácia nastaví sa hodnota atribútu encoding na túto hodnotu
- Hodnota atribútu nie je validovaná na existenciu danej znakovnej sady

Niektoré z možností ktoré môžu nastať ilustruje nasledujúci príklad:

Príklad 2231.xml:

```
<?xml version="1.2" standalone="yes" encode="Big5"?>
<?xml version="1.0" standalone="yes" encoding="UTF-8"?>
<DOCUMENT num="2231">
</DOCUMENT>
```

Výsledok po normalizácii:

```
<?xml version="1.0" standalone="yes"?>
<?_xml version="1.0" standalone="yes" encoding="UTF-8"?>
<DOCUMENT num="2231">
</DOCUMENT>
```

Vyššie uvedená ukážka tiež ukazuje riešenie pokiaľ sa vyskytuje XML deklarácie na inom mieste v dokumente ako pred začiatkom DTD či koreňového elementu, prípadne sa vyskytuje viackrát. Potom je spracovaná rovnakým spôsobom ako procesná inštrukcia, úvodná sekvencia sa nahradí sekvenciou “<?_xml”, čím vznikne dobre formovaná procesná inštrukcia. Ak sa XML deklarácia nenachádza na začiatku dokumentu, použijú sa pravidlá z predchádzajúcej kapitoly 2.2.2.4 Zlá logická štruktúra.

2.2.3.2 DTD sekcia

Samotná oprava DTD sekcie nebola predmetom tejto práce, preto sa na DTD program pozerá ako na celok. DTD sekcia je parserom načítaná podľa pravidiel gramatiky [G28]. Následne sa pomocou parseru SAX z knižnice jdom [10] overí, či je DTD sekcia ako celok dobre formovaná. V prípade, že tomu tak nie je, DTD sekcia sa ako celok vypustí. Ak dobre formovaná je, vyhodnotí a spracuje sa podľa pravidiel z predchádzajúcej kapitoly 2.2.2.4 Zlá logická štruktúra. Externé entitné referencie rovnako ako externé deklarácie DTD sa aj v prípade platnej sekcie nevalidujú.

Vzhľadom k tomu, že platné DTD môže obsahovať internú podmnožinu v ktorej môžu byť deklarované interné entity, na ktoré by sa mohli odkazovať entitné referencie v rámci dokumentu, je potrebné ich vyhodnotiť pre ďalšie spracovanie. Hodnota každej internej entity sa vyhodnotí, či už je deklarovaná priamo alebo nepriamo až na konečnú hodnotu – nahradzujúci text. Pre každú entitu sa ďalej tieto parametre:

- Porušenie pravidla odkazovania na samú seba [WFC11], nakoľko entita ktorá odkazuje sama na seba a nie je odkazovaná v dokumente, nebráni DTD aby bola dobre formovaná
- Možnosť odkazovať na entitu v atribútoch
- Možnosť odkazovať na entitu v textovom obsahu elementu
- Vyhodnotenie externej entity – entita odkazuje priamo alebo nepriamo na externú alebo nespracovanú entitu

Hodnoty a vlastnosti entít sa následne využijú pri vyhodnotení obsahu pri entitných referenciách v hodnotách atribútov a textového obsahu elementu.

2.2.4 Elementy a atribúty

Z definície dobre formovaného XML, podľa XML špecifikácie [1], musí každý dokument spĺňať tieto vstupné podmienky dotýkajúce sa elementov a atribútov:

- Rovnaký počiatočný a koncový tag podľa [WFC4].
- Atribúty sa neopakujú podľa [WFC5].
- Hodnoty atribútov neobsahujú referencie na externé entity [WFC6], rovnako ako priame či nepriame referencie na nedeklarované entity.
- Hodnota atribútu ani nahradzujúci text odkazovaný cez referenciu nemôže obsahovať znak ‘<’ [WFC7]

- Formát elementu musí spĺňať pravidlá pre element [G39] a následné definície formátu pre prázdny element [G44], počiatočný tag [G40], koncový tag [G42], atribúty [G41] a hodnoty atribútov [G10].

2.2.4.1 Počiatočné a koncové značky v elementoch

Pre splnenie prvej podmienky [WFC4] algoritmus rozpoznáva nasledovné prípady:

- Neprázdny element nie je uzavretý koncovým tagom
- Existuje koncový tag uzatvárajúci element ku ktorému nemá počiatočný tag
- Špeciálny prípad predchádzajúcich, koncový tag nájde k sebe počiatočný otvorený tag, ktorý sa zhoduje až na veľkosť písmen

Nasledujúci príklad ilustruje normalizáciu v prípade porušenia prvej a druhej podmienky.

Príklad 2241.xml:

```
<DOCUMENT num="2241">
<CaSe repeated="val1" other="123"
repeated="value2"> Content1
</cAsE >
<start>
Content2
</end>
</DOCUMENT>
```

Výstup po normalizácii:

```
<DOCUMENT num="2241">
<CaSe repeated="val1" other="123">
Content1
</CaSe>
<start>
Content2
</start></DOCUMENT>
```

Ako je ilustrované na príklade normalizácie, neprázdny neuzavretý element `<start>` sa automaticky uzavrie, presne pred uzavretím jeho rodičovského elementu. V prípade viacerých neuzavretých elementov sa všetky elementy uzavru postupne, tak aby bola splnená podmienka hniezdenia.

V prípade výskytu koncového tagu bez párového počiatočného tagu - `</end>`, sa koncový tag jednoducho odstráni. Kombinácia predchádzajúcej a tejto opravy sa aplikuje aj v prípade nesprávneho zahniezdenia elementov. Ak nastane zhoda počiatočného a koncového tagu až na veľkosť písmen - `<CaSe>` a `</cAsE >`, sa veľkosť písmen v koncovom tagu upraví podľa počiatočného tagu. Na rozdiel od aplikácie predchádzajúcich prípadov, tak koncovému tagu zostáva pôvodné umiestnenie v dokumente.

2.2.4.2 Atribúty a hodnoty atribútov

Pre splnenie druhej podmienky o viacnásobnom výskyte atribútu v rámci elementu , sa vyberie vždy iba prvý výskyt atribútu v danom elemente, ostatné výskyty atribútu s rovnakým menom sa úplne vypustia bez ohľadu na ich hodnoty.

Splnenie tretej a štvrtej podmienky sa dotýka hodnôt atribútov. Rozlišujeme nasledujúce prípady ktoré môžu nastať:

- Hodnota atribútu nie je v úvodzovkách alebo v apostrofoch
- Hodnota atribútu obsahuje hodnotu '`<`' alebo obsahuje entitnú referenciu ktorá na takúto hodnotu priamo alebo nepriamo odkazuje
- Hodnota atribútu obsahuje priamu alebo nepriamu referenciu na externú entitu
- Hodnota atribútu obsahuje nedeklarovanú entitnú referenciu, nepriamu referenciu na nedeklarovanú entitu alebo znak '`&`' za ktorým nenasleduje referencia

Príklad ukazuje rôzne možnosti ktoré môžu nastať a ich riešenia:

Príklad 2242.xml

```
<!DOCTYPE doc [  
<!ENTITY ent1 "text">  
<!ENTITY ent2 "aa&ent3;zz">  
<!ENTITY ent3 SYSTEM "001.ent">  
<!ENTITY ent4 "1 &ent5; 2">  
<!ENTITY ent5 "<">  
<!ENTITY % pent '!ENTITY ent6  
"&ent1;">' >  
%pent;  
>  
<DOCUMENT num="2242">  
<a att1="val<" att2=val1/>  
<b att2="val2&">  
<c att3="&ent1;">  
<d att4="&ent2;">  
<e att5="&ent3;">  
<f att6="&ent4;">  
<f att7="&ent5;">  
<h att8="aa&ent6;">  
</DOCUMENT>
```

Výstup po normalizácii:

```
<!DOCTYPE doc [  
<!ENTITY ent1 "text">  
<!ENTITY ent2 "aa&ent3;zz">  
<!ENTITY ent3 SYSTEM "001.ent">  
<!ENTITY ent4 "1 &ent5; 2">  
<!ENTITY ent5 "<">  
<!ENTITY % pent '!ENTITY ent6  
"&ent1;">' >  
%pent;  
>  
<DOCUMENT num="2242">  
<a att1="val&lt;1" att2="val1"/>  
<b att2="val2&amp;">  
<c att3="&ent1;">  
<d att4="&amp;ent2;">  
<e att5="&amp;ent3;">  
<f att6="&amp;ent4;">  
<f att7="&amp;ent5;">  
<h att8="aa&ent6;">  
</DOCUMENT>
```

V prípade, že hodnota obsahuje priamo zakázaný znak ‘<’ alebo hodnota atribútu nie je uzavretá v úvodzovkách či apostrofoch, ako `<a att1="val<" att2=val1/>` znak je nahradený odpovedajúcou entitou `<`, hodnota ktorá nie je uzavretá v úvodzovkách sa vloží do úvodzoviek. Za koniec hodnoty atribúty sa v takom prípade považuje prvá medzera, znaky ‘<’ a ‘>’ a sekvencia ‘/’.

Ak hodnota obsahuje znak ‘&’, ktorý nie je súčasťou entitnej referencie `<b att2="val2&">`, potom tento znak je nahradený odpovedajúcou entitou `&`. V situácii, keď referencia v hodnote atribútu odkazuje priamo `<e att5="&ent3;">` alebo nepriamo `<d att4="&ent2;">` na externú prípadne nedeklarovanú entitu je referencia zrušená, znak ‘&’ sa nahradí entitou `&`. Rovnakým spôsobom sa postupuje aj v prípade keď referencia odkazuje na priamo `<g att7="&ent5;">` alebo nepriamo `<f att6="&ent4;">` na entitu ktorej obsahom je zakázaný znak.

V prípadoch, kde je entita správne deklarovaná, či už priamo `<c att3="&ent1;">` alebo nepriamo komplikovaným spôsobom cez referenciu `<h att8="aa&ent6;">`, na entitu deklarovanú cez parametrickú entitu `% pent`, nedôjde k žiadnej zmene v pôvodnom vstupe.

2.2.4.3 Syntax elementov a atribútov

Pre splnenie piatej podmienky dotýkajúcej sa gramatiky, algoritmus rieši nasledovné špecifické situácie:

- Neplatný prvý znak na začiatku alebo v tele mena elementu
- Neplatný znak na začiatku alebo v tele mena atribútu
- Atribút bez priradenej hodnoty
- Atribút nasledujúci iný atribút bez medzery

Nasledujúci príklad ilustruje možné situácie ktoré môžu nastať:

```
<DOCUMENT num="2343">
<.a^.a$>
content
</.a^.a>
<bbb c^c="val1"dd="val2" eee/>
</DOCUMENT>
```

```
<DOCUMENT num="2243">
<_a_.a>
content
</_a_.a>
<bbb c_c="val1" dd="val2"
eee="eee"/>
</DOCUMENT>
```

Ako je vidieť v prípade mena elementu `<.a^.a$>` sa neplatné znaky nahradia podtržítkom, pričom sa rozlišuje prvý znak a ostatné znaky ktoré sa validujú rozdielnym spôsobom ako špecifikuje pravidlo [G5]. V prípade že sa neplatný znak nachádza na konci, nenahradzuje sa, ale vypustí. Koncová značka sa následne upraví rovnako ako počiatočná. V prípade neplatných znakov pri menách atribútov sa postupuje rovnakým spôsobom ako pri menách elementov.

Pokiaľ nemá atribút priradenú žiadnu hodnotu (eee) , jedná sa typicky o boolean atribút ktorý sa môže v minimalizovanej forme vyskytovať napríklad v HTML. Pre takýto prípad algoritmus implicitne nastaví hodnotu atribútu rovnakú ako meno atribútu. Vstupným parametrom "attrvalue" je možné toto implicitné chovanie zmeniť, vzhľadom na parameter sa hodnota nastaví buď na "true" (true) alebo na prázdnu hodnotu "" (empty). Pokiaľ atribút nasleduje iný atribút bez požadovanej medzery (atribút dd v elemente `<bbb c^c="val1"dd="val2" eee/>`), medzera sa automaticky pridá.

2.2.5 Textový obsah elementu

Pre obsah elementov je nutné dodržať niekoľko pravidiel. Medzi najčastejšie problémy najmä pokiaľ je zdrojovým dokumentom HTML patrí výskyt nedeklarovaných entít, špeciálnych entít a výskyt zakázaných znakov alebo neplatných znakov. Algoritmus rieši nasledovné situácie:

- V obsahu sa vyskytuje nedeklarovaná entita [WFC9] s výnimkou preddefinovaných entít, číselných entít odkazujúcich na konkrétny znak alebo špeciálnych HTML entít
- V obsahu sa vyskytuje priama alebo nepriama referencia na nespracovanú entitu [WFC10]
- Neobsahuje referenciu na entitu ktorá priamo alebo nepriamo odkazuje na samú seba [WFC11]
- Nesmie sa vyskytnúť znak '&' [G14] mimo referencie na entitu
- Nesmie sa vyskytnúť sekvencia znakov označujúcich koniec CDATA sekcie ']]>' [G14]
- Nesmie sa vyskytnúť znak '<' [G14] mimo toho že značí začiatok elementu
- Znak vyskytujúci sa v obsahu musí byť platným znakom
- Text sa nesmie vyskytovať mimo obsahu elementu

Nasledujúci príklad ilustruje uvedené možné situácie a ich riešenia:

Príklad 2251.xml

```
<!DOCTYPE doc [  
<!ENTITY unpr "&pic;" >  
<!ENTITY pic SYSTEM "img.gif"  
NDATA gif >  
<!ENTITY self "&self;" >  
<DOCUMENT num="2251">  
<a attr1="special">  
l&eacute;k  
</a>  
<b attr2="parsed">  
&unpr;&self;  
</b>  
<c attr3="ampersand">  
J&T bank  
</c>  
<d attr5="cdataEnd">  
cdata]]>  
</d>  
<g attr6="invChar">  
ASCII25 &#25;  
</g>  
<f attr7="ok">  
&#39;&gt;  
</f>  
</DOCUMENT>
```

Výstup po normalizácii:

```
<!DOCTYPE doc [  
<!ENTITY unpr "&pic;" >  
<!ENTITY pic SYSTEM "img.gif"  
NDATA gif >  
<DOCUMENT num="2251">  
<a attr1="special">  
lék  
</a>  
<b attr2="parsed">  
&unpr;&self;  
</b>  
<c attr3="ampersand">  
J&T bank  
</c>  
<d attr5="cdataEnd">  
cdata]]>  
</d>  
<g attr6="invChar">  
ASCII25 &#25;  
</g>  
<f attr7="ok">  
&#39;&gt;  
</f>  
</DOCUMENT>
```

Ako je vidieť na príklade, použije sa v prípade nedeklarovanej, nespracovanej entity, entity odkazujúcej na seba alebo nesprávneho použitia amersandu jeho náhrada ampersandu za preddefinovanú entitu `&`. Výnimkou je možnosť nahradenia špeciálnej entity definovanej v štandarde HTML 4.0 [9] odpovedajúcim Unicode znakom, pre zachovanie možnosti ďalej pracovať z obsahom napr. vyhľadávať v texte. Príkladom je entita malého dlhého písmena „e“ `é`.

V prípade výskytu sekvencie znakov ukončujúcich štandardne sekciu CDATA sa nahradí iba znak `'>'` za odpovedajúcu entitu `'>'`, rovnako aj tu je možné použiť odpovedajúcu numerickú entitu.

V prípade neplatného znaku ACSII 25 (v priloženom súbore na je použitý reálny neplatný znak ASCII 25 END OF MEDIA tu ho nebolo možné reprodukovať) alebo jeho numerickej entity sa použije jeho náhrada v tvare `&#kod_znaku;`. Pokiaľ sú nedeklarované entity použité správne, ako v prípade `'>`, , pôvodný obsah je ponechaný.

V prípade výskytu znaku `'<'` v znakovnej časti nedochádza k jednoduchému nahradeniu preddefinovanou entitou nakoľko sa v tomto prípade môže jednáť o začiatok elementu ktorý nie je korektné uzavrieť znakom `'>'`. Nasledujúci príklad ukazuje situácie ktoré môžu v takomto prípade nastať:

Príklad 2252.xml

```
<DOCUMENT num="2252">
<a attr1="invStartTag">
Text <tag
</a>
<b attr2="invStartTagAtt">
Text <tag attr='123'
</b>
<c attr2="invStartTag2">
Text <tag attr attr2
</c>
<d attr4="invStartTag3">
Text <tag<d/>
<e attr5="ok">
Text< Text
</e>
</DOCUMENT>
```

Výsledok po normalizácii

```
<DOCUMENT num="2252">
<a attr1="invStartTag">
Text <tag></tag>
</a>
<b attr2="invStartTagAtt">
Text <tag attr='123'></tag>
</b>
<c attr2="invStartTag2">
Text <tag attr="attr"
attr2="attr2"></tag>
</c>
<d attr4="invStartTag3">
Text &lt;tag<d/>
<e attr5="ok">
Text&lt;Text
</e>
</DOCUMENT>
```

Ako ukazuje príklad v prípade výskytu znaku ‘<’ v prvom elemente, za ktorým nasleduje reťazec akceptovateľný ako meno elementu nasledovaný medzerou, je tento reťazec vyhodnotený ako začiatok elementu, na príklade je k nemu pridané aj uzatvorenie pre splnenie podmienky [WFC4].

V druhom elemente sa jedná o podobný prípad kde navyše za medzerou nenasleduje ďalší element ale textový reťazec potenciálny atribút s hodnotou, výsledkom normalizácie je podobne ako v predchádzajúcom prípade vytvorenie nového elementu a následne jeho uzavretie v rámci rodičovského elementu.

Tretí prípad ilustruje možnosť keď za potenciálnym menom elementu nasledujú ďalšie textové reťazce oddelené medzerou. Až do výskytu ďalšieho elementu sa to vyhodnotí ako element s atribútmi ktoré nemajú priradené hodnoty. Následne sa atribútom priradia hodnoty rovnakým spôsobom ako pri hodnotách štandardných atribútov na základe vstupného parametru, element sa uzavrie rovnako ako v predchádzajúcich prípadoch.

V štvrtom prípade nastáva situácia kedy je text nachádzajúci sa za znakom ‘<’ ihneď nasledovaný začiatkom iného elementu, v takomto prípade sa predpokladá, že sa nejde o element a preto je znak ‘<’ nahradený odpovedajúcou preddefinovanou entitou <.

V piatom prípade kde je znak ‘<’ nasledovaný medzerou prípadne iným znakom ekvivalentným medzere [G3], sa tento znak nahradí odpovedajúcou entitou rovnakým spôsobom ako v predchádzajúcom prípade.

Posledným prípadom je situácia kedy sa textový obsah nachádza mimo obsahu koreňového elementu. Ako ukazuje nasledujúci prípad môžu nastať tri situácie:

Príklad 2253.xml

```
<!-- start of the document -->
Text Before Doctype
<!DOCTYPE doc [
<!ENTITY ent1 "text">
]>
Text Out of Element
<DOCUMENT num="2252">
TextContent
</DOCUMENT>
Text&ent1;
<!-- end of the document -->
```

Výsledok po normalizácii:

```
<!-- start of the document -->
<!DOCTYPE doc [
<!ENTITY ent1 "text">
]>
<DOCUMENT num="2252">
TextContent
</DOCUMENT>
<!-- start of the document -->
```

Ako ilustruje príklad textový obsah sa môže nachádzať pred prológom, medzi prológom a koreňovým elementom alebo na konci za koncovou značkou koreňového elementu. Vo všetkých prípadoch sa postupuje rovnako a textový obsah sa odstráni. V tomto prípade sa do budúcnosti ponúka aj alternatívna možnosť takýto text zakomentovať aby sa zachoval pôvodný obsah kompletný. Táto možnosť však nie je implementovaná.

2.2.6 Komentáre

Komentár podľa špecifikácie je definovaný pravidlom [G15] a ďalšími pravidlami pre logickú štruktúru dokumentu. Z toho plynú nasledovné podmienky:

- Komentár nesmie začínať sekvenciou znakov “<!--”
- Komentár nesmie končiť sekvenciou znakov “-->”
- Komentár nesmie obsahovať dve pomlčky po sebe
- Komentár nesmie obsahovať neplatné znaky
- Komentár sa nesmie nachádzať pred XML deklaráciou
- Komentár sa v rámci elementu môže nachádzať len vo vnútri obsahu elementu

Príklad ilustruje možné narušenia dobrej formovanosti dokumentu:

Príklad 2261.xml

```
<!-- comment1 -->
<?xml version="1.0"?>
<!DOCTYPE doc [
<!ENTITY ent1 "text">
<!-- comment2 -->
]>
<DOCUMENT num="2324">
<a attr="1" <!--comment3 -->
attr="2">
</a>
<!-- com--ment4 -->
<!--_ comment5 -->
<!-- comment6 --->
</DOCUMENT>
<!-- comment7 -->
```

Výsledok po normalizácii

```
<?xml version="1.0"?>
<!-- comment1 -->
<!DOCTYPE doc [
<!ENTITY ent1 "text">
<!-- comment2 -->
]>
<DOCUMENT num="2324">
<a attr="1"><!-- comment3 -->
attr="2">>
</a>
<!-- com_ ment4 -->
<!--_ comment5 -->
<!-- comment6 _-->
</DOCUMENT>
<!-- comment7 -->
```

Komentár ktorý sa nachádza pred XML deklaráciou je presunutý za deklaráciu a zostáva zachovaný. Komentár vo vnútri DTD sekcie zostáva zachovaný (DTD je dobre formovaná) v pôvodnom stave, rovnako ako komentár na konci dokumentu.

V komentároch obsahujúcich zakázané sekvencie znakov, sú nevhodné pomlčky nahradené podtržkami. V prípade že by komentár obsahoval zakázané znaky, sú znaky nahradené rovnakým spôsobom ako v textovej kapitole 2.2.5 Textový obsah elementu.

V prípade, že by sa komentár nachádzal priamo medzi atribútmi v počiatočnej značke elementu, potom sa počiatočná značka elementu začatím komentára považuje za uzavretú, komentár sa presunie do obsahu elementu a ostatné atribúty sa rovnako presunú do obsahu elementu.

2.2.7 Procesná Inštrukcia

Procesná inštrukcia je podľa špecifikácie definovaná pravidlami [G16] a [G17] a ďalšími pravidlami pre logickú štruktúru dokumentu. Z toho plynú nasledovné podmienky:

- Cieľ procesnej inštrukcie musí mať platné meno [G5] rovnako ako napr. meno atribútu
- Cieľ procesnej inštrukcie nesmie byť "xml" bez ohľadu na veľkosť písmen
- Obsah procesnej inštrukcie nesmie obsahovať sekvenciu znakov ">"
- Procesná inštrukcia nesmie obsahovať neplatné znaky
- Procesná inštrukcia sa nesmie nachádzať pred XML deklaráciou
- Procesná inštrukcia sa v rámci elementu môže nachádzať len vo vnútri obsahu elementu

Príklad ilustruje možné narušenia dobrej formovanosti dokumentu procesnou inštrukciou:

Príklad 2271.xml

```
<?pi1 text ?>
<?xml version="1.0"?>
<!DOCTYPE doc [
<!ENTITY ent1 "text">
<?pi2 text ?>
]>
<DOCUMENT num="2324">
<a attr="1"><? pi3 text ?>
attr="2">
</a>
<? pi4 text ?>
<?xml text ?>
<?pi6 text <?pi7 ?> text?>
</DOCUMENT>
<?.a^.a$ text ?>
```

Výsledok po normalizácii:

```
<?xml version="1.0"?>
<?pi1 text ?>
<!DOCTYPE doc [
<!ENTITY ent1 "text">
<?pi2 text ?>
]>
<DOCUMENT num="2324">
<a attr="1"><?pi3 text ?>
attr="2">>
</a>
<?pi4 text ?>
<?_xml text ?>
<?pi6 text <?pi7 ?> text?>>
</DOCUMENT>
<?aa text ?>
```

Procesná inštrukcia ktorá sa nachádza pred XML deklaráciou, je presunutá za deklaráciu a zostáva zachovaná rovnako ako procesná inštrukcia vo vnútri dobre formovanej DTD sekcie. Ak cieľ procesnej inštrukcie je "xml" a nejedná sa o XML deklaráciu nahradí sa cieľom "_xml". V prípade že cieľ obsahuje neplatné znaky <?.a^.a\$ text ?>, neplatné znaky sa vypustia, ak je medzi začiatkom a cieľom procesnej inštrukcie medzera, vypustí sa.

V prípade že by procesná inštrukcia obsahovala zakázané znaky, sú znaky nahradené rovnakým spôsobom ako v textovej sekcii, kapitola 2.2.5. Ak nastane situácia, že by sa procesná inštrukcia nachádzala priamo medzi atribútmi v počiatočnej značke elementu, spracovanie je identické ako v prípade komentáru v kapitole 2.2.6.

V situácii keď sa procesná inštrukcia nesprávne uzatvára, alebo sú dve procesné inštrukcie do seba zahniezdené `<?pi6 text <?pi7 ?> text?>`, sa ukončí procesná inštrukcia prvým uzatvorením. Zbytok `“text?>“` sa buď použije ako textový obsah elementu, alebo sa vypustí, ak sa nachádza procesná inštrukcia mimo elementu.

2.2.8 CDATA sekcia

CDATA sekcia je definovaná pravidlami [G18] - [G21]. Algoritmus opravuje nasledovné špeciálne prípady:

- CDATA sekcia na nachádza mimo obsahu koreňového elementu
- CDATA sekcia začína malými písmenami
- CDATA sekcia obsahuje zakázané znaky

Príklad 2281.xml

```
<?xml version="1.0"?>
<![CDATA[ <b attr1="aa"></b> ]]>
<DOCUMENT num="2281">
<![CDATA[ <b attr1="aa"></b> ]]>
</DOCUMENT>
```

Výsledok po normalizácii

```
<?xml version="1.0"?>
<DOCUMENT num="2281">
<![CDATA[ <b attr1="aa"></b> ]]>
</DOCUMENT>
```

Ak sa CDATA sekcia nachádza mimo koreňového elementu je z dokumentu vypustená. V prípade že začína malými písmenami je opravená, v ostatných prípadoch je považovaná za štandardný text. Ak by procesná inštrukcia obsahovala zakázané znaky, sú znaky nahradené rovnakým spôsobom ako v textovej sekcii 2.2.5.

2.2.9 Menné priestory

Menné priestory v XML 1.0 definuje dokument Namespaces in XML 1.0, najnovšia je druhá edícia špecifikácie z 16. Augusta 2006 [14]. Podmienky kladené na menné priestory nie sú podmienkami nutnými pre dobre formovaný XML dokument. Napriek tomu program opravuje dokument, aby bol dobre formovaný aj z pohľadu menných priestorov, je to vhodné pre následné jednoduchšie spracovanie výstupu štandardným parserom SAX [10]. Ignorovanie menných priestorov je možné nastaviť vstupným parametrom `ignorenamespaces` na `yes`, v takom prípade bude program postupovať čisto iba podľa podmienok základnej XML špecifikácie. Základné podmienky pre splnenie dobrej formovanosti z pohľadu menných priestorov:

- menný priestor `"xmlns"` sa nesmie použiť inak ako na deklaráciu iných menných priestorov a nesmie byť použitý v menách elementov
- rezervovaný menný priestor `"xml"` nesmie byť deklarovaný inak ako `“http://www.w3.org/XML/1998/namespace“`
- mená atribútov v súlade s pravidlom gramatiky [14] - [G15] a neobsahujú viac ako jednu dvojbodku
- mená elementov v súlade s pravidlami gramatiky [14] - [G12-G14] a neobsahujú viac ako jednu dvojbodku
- pri použití menného priestoru pre meno atribútu alebo elementu musí byť tento menný priestor deklarovaný v rovnakom alebo nadradenom elemente
- mená entít a cieľov procesných inštrukcií neobsahujú dvojbodku

- lokálne časti atribútov v rámci rovnakého menného priestoru v jednom elemente nie sú duplicitné

V nasledujúcich podkapitolách budú jednotlivé prípady rozobraté spolu s riešeniami situácií ktoré narušujú podmienky špecifikácie menných priestorov.

2.2.9.1 Deklarácia menného priestoru a rezervovaný menný priestor

Menný priestor sa deklaruje podľa pravidiel gramatiky pre menné priestory [14] - [G1-G6]. Z pravidiel gramatiky vyplývajú tieto konkrétne podmienky:

- Hodnota deklarácie menného priestoru nemôže byť prázdny reťazec
- Nie je možné deklarovať namespace "xml" nesmie inak ako "http://www.w3.org/XML/1998/namespace"
- Nie je možné deklarovať namespace "xmlns"
- Deklarácia menného priestoru musí byť platné meno podľa pravidla [G14] - [G4] a teda nesmie obsahovať dvojbodku
- Deklarácie sa nesmú opakovať (rovnako ako atribúty)

Jednotlivé prípady a ich opravy na príkladoch:

Príklad 2291.xml

```
<a xmlns:a=" " >
<b xmlns:xmlns="ns1">
<c xmlns:xml="ns2">
<d xmlns:b:c="ns2">
<e xmlns:d="ns4" xmlns:d="ns5">
</e></d></c></b></a>
```

Výsledok po normalizácii:

```
<a xmlns:a="xmlns:a">
<b>
<c xmlns:xml="http://
www.w3.org/XML/1998/namespace">
<d xmlns:b="ns2">
<e xmlns:d="ns4">
</e></d></c></b></a>
```

Príklad ilustruje riešenia jednotlivých prípadov nesprávnych deklarácií. V prípade prázdnej deklarácie, sa ako hodnota použije znovu meno atribútu deklarácie. Ak je deklarovaný menný priestor xmlns vypustí sa, pri deklarovaní menného priestoru xml sa nahradí jediným povoleným priestorom podľa deklarácie. Pri výskyte dvojbodky v mene atribútu deklarácie sa atribút sprava skráti tak aby meno bolo platné. Opakované deklarácie sa vypustia, zostáva iba prvá deklarácia.

2.2.9.2 Elementy a menné priestory

Podľa pravidiel pre menné priestory musí byť v prípade použitia v elemente musia byť splnené tieto podmienky:

- menný priestor použitý v názve elementu musí byť deklarovaný v tomto alebo v nadradenom elemente
- nesmie byť použitý v menný priestor „xmlns“
- element nesmie v mene obsahovať viac ako jednu dvojbodku

Príklad 2292.xml

```
<x:a xmlns:x="ns1">
<xmlns:b >
<y:c>
<x:d:e xmlns:y="ns2">
</d></y:c></xmlns:b></x:a>
```

Výsledok po normalizácii:

```
<x:a xmlns:x="ns1" xmlns:y="y">
<b>
<y:c>
<x:d xmlns:y="ns2">
</d></y:c></b></x:a>
```

Ako ukazuje príklad v prípade chýbajúcej deklarácie menného priestoru „y“, deklarácia je doplnená do koreňového elementu. Ak je použitý v mene elementu menný priestor „xmlns“, použije sa meno atribútu bez menného priestoru, v prípade že obsahuje viac ako jednu dvojbodku, meno sa sprava skráti tak aby bolo platné.

2.2.9.3 Atribúty a menné priestory

V prípade mien atribútov ktoré nie sú deklaráciami menných priestorov, platia nasledovné podmienky:

- menný priestor musí byť deklarovaný rovnako ako v prípade elementu
- atribúty po aplikovaní menných priestorov sa nesmú opakovat', viac túto situáciu objasňuje príklad ukazuje príklad
- atribút nesmie obsahovať viac ako jednu dvojbodku

Príklad 2293.xml

```
<a xmlns:x="ns1">
<b xmlns:y="ns1">
<c z:attr1="val1">
<d x:attr="val2" y:attr="val3">
<e x:attr2:aaa="val4">
</e> </d></c></b></a>
```

Výsledok po normalizácii:

```
<a xmlns:x="ns1" xmlns:z="z">
<b xmlns:y="ns1">
<c z:attr1="val1">
<d x:attr="val2">
<e x:attr2="val4">
</e> </d></c></b></a>
```

Nedeklarovaný menný priestor „z“ sa dodatočne doplní do koreňového elementu rovnako ako v predchádzajúcom prípade. Rovnakým orezaním z pravej strany ako v prípade mena elementu sa opraví aj meno atribútu obsahujúce viac ako jednu dvojbodku.

V prípade elementu `<d x:attr="val2" y:attr="val3">` je podľa štandardných požiadaviek na atribúty všetko v poriadku, nakoľko sa ich mená odlišujú (sú v rôznych menných priestoroch). V prípade aplikácie hodnôt menných priestorov sa situácia mení. Nakoľko sú obidva menné priestory „x“ aj „y“ deklarované v nadradených elementoch s rovnakou hodnotou, podľa podmienok špecifikácie [14] a sa ich mená považujú za identické. Preto zostáva zachovaný len prvý atribút a druhý sa vypustí.

2.2.9.4 Entity, procesné inštrukcie a menné priestory

Pre entity a procesné inštrukcie vzhľadom na menné priestory platí:

- Cieľ procesnej inštrukcie neobsahuje dvojbodku
- Meno entitnej referencie v textovom obsahu alebo hodnote atribútu neobsahuje dvojbodku

Príklad 2294.xml

```
<!DOCTYPE doc [
<!ENTITY x:ent1 "val1" >
]>
<a xmlns:x="ns1">
&x:ent1;
<b x:attr1="&x:ent1;">
<?x:pi1 aaa?>
</b></a>
```

Výsledok po normalizácii:

```
<!DOCTYPE doc [
<!ENTITY x:ent1 "val1" >
]>
<a xmlns:x="ns1">
&x:ent1;
<b x:attr1="&x:ent1;">
<?xpi1 aaa?>
</b></a>
```

Cieľ procesnej inštrukcie zostal rovnaký akurát dvojbodka je vypustená ako neplatný znak. V prípade použitia entitnej referencie obsahujúcej dvojbodku v hodnote atribútu alebo v textovom obsahu, je entita spracovaná rovnako ako v prípade že by nebola deklarovaná.

2.2.10 XML 1.0 vs. XML 1.1

V súčasnosti sú platné dve verzie XML 1.0 a 1.1. Verzia 1.0 má päť edícií pôvodná špecifikácia je z roku 1998 a najnovšia edícia bola publikovaná 26. Novembra 2008. XML 1.1 [3] vyšlo v roku 2004 a v rovnaký deň ako tretia edícia XML 1.0 [2] a posledná druhá edícia je zo 16. Augusta 2006. XML 1.1 bola v dobe vydania výrazne menej reštriktívna ako XML 1.0. Pôvodným zámerom vydania XML 1.1 bola najmä podpora rozšírenej znakovkej sady a tým podpora znakov ktoré XML 1.0 v danej dobe nepodporovala [7]. Príkladom môže byť definícia mena atribútu alebo elementu [G4] a [G5] :

XML 1.0 4. edícia 29. September 2006

```
[4] NameChar ::= Letter | Digit | '.' | '-' | '_' | ':'  
                | CombiningChar | Extender  
[5] Name      ::= (Letter | '_' | ':') (NameChar)*
```

XML 1.1 2. edícia 29. September 2006

```
[4] NameStartChar ::= ":" | [A-Z] | "_" | [a-z] | [#xC0-#xD6] |  
                    [#xD8-#xF6] | [#xF8-#x2FF] | [#x370-#x37D] |  
                    [#x37F-#x1FFF] | [#x200C-#x200D] | [#x2070-  
                    #x218F] | [#x2C00-#x2FEF] | [#x3001-#xD7FF] |  
                    [#xF900-#xFDCF] | [#xFDF0-#xFFFF] | [#x10000-  
                    #xEFFFF]  
[4a] NameChar      ::= NameStartChar | "-" | "." | [0-9] | #xB7 |  
                    [#x0300-#x036F] | [#x203F-#x2040]  
[5] Name           ::= NameStartChar (NameChar)*
```

XML 1.0 je postavená na základe Unicode 2.0 kdežto XML 1.1 vychádza zo špecifikácie Unicode 3.2. Týmto umožňuje napríklad využívať v menách elementov a atribútov aj také znaky ako sú znaky Fenickej abecedy (#x10900 - #x 1091F) alebo súčasného mongolského písma, ktoré boli pridané až od verzie Unicode 3.2 [8].

Všetky tieto rozdiely sa vydaním piatej edície XML 1.0 zmazali, nakoľko piata edícia prebrala definíciu mien atribútov a elementov z verzie XML 1.1. Napriek tomu aj v najnovších verziách ešte pretrvávajú tieto rozdiely:

1. XML 1.1 umožňuje odkazovať cez entitné referencie na znaky kontrolnej sady (napr. #xC) ktoré definuje v pravidle [G2a], v prípade XML 1.0 by sa jednalo o porušenie pravidla [WFC8], až na výnimku uvedenú v ďalšom pravidle.
2. XML 1.0 umožňuje použitie kontrolných znakov [#x7F-#x84] a [#x86-#x9F] priamo, XML 1.1 na ne musí odkazovať podľa predchádzajúceho pravidla.

Vzhľadom k tomu, že aktuálna verzia XML je XML 1.0 5. edícia, všetky validácie sú robené vzhľadom k tejto špecifikácii.

2.3 Záver a možné rozšírenie funkčnosti

Výsledkom spracovania podľa všetkých vyššie uvedených by mal byť dobre formovaný súbor. Napriek tomu existujú ešte možnosti, ktoré by z praktického hľadiska funkčnosť programu výrazným spôsobom rozšírili. Za predpokladu, že vstupný súbor obsahuje dáta z HTML by bolo vhodné pridať špeciálnu možnosť spracovania niektorých elementov. Toto

by sa uplatnilo najmä v prípade elementov `<style>` a `<script>`, ktorých obsah nie je vhodné opravovať ale bolo by vhodnejšie ho uzavrieť do CDATA sekcie alebo komentáru, aby sa vyhlo ich spracovaniu ako textového obsahu. Pridať možnosť parametrizovať mená elementov ktorých obsah by sa spracoval ako CDATA sekcia alebo komentár, by bolo vhodným rozšírením aplikácie do budúcnosti. Ďalším možným rozšírením by bolo spracovanie externých DTD a externých entít, ktoré by umožnilo zachovať deklarované externé entitné referencie v pôvodnej podobe.

Ako ďalšia fáza vývoja sa ponúka možnosť kontrolovať validitu dokumentu DTD, aby výsledný dokument bol nielen dobre formovaný ale aj validný.

3 Detekcia jazyka

Základnou motiváciou identifikovať jazyk XML dokumentu vychádza z predpokladu, že na vstupe je súbor obsahujúci obsahy internetových stránok neznámeho jazyka. Pri vyhľadávaní v takomto obsahu je potom možné použiť kritérium jazyka ako obmedzujúcu podmienku. Pri vstupe je možné zadať identifikáciu pre dokument ako celok (jeho koreňový element), alebo špecifikovať meno elementu pre ktorý má byť identifikácia vykonaná. Možnosť zadať meno elementu ako vstup vychádza s predpokladaného vstupného súboru ktorý obsahuje viac dokumentov tvare ako súbory z projektu Egothor [19]:

```
<EOGTHOR>
<DOCUMENT num="1">
  <html>
    Web page 1 content ...
  </html>
</DOCUMENT>
<DOCUMENT num="2">
  <html>
    Web page 2 content ...
  </html>
</DOCUMENT>
.
.
</EOGTHOR>
```

V tomto prípade sa ako vstup pre identifikáciu nastaví hodnota parametru langtag na „DOCUMENT“.

Sekundárnym cieľom bolo vytvoriť platformu ktorá umožní pridať do zoznamu rozpoznávaných jazykov nový jazyk, jednoduchým a užívateľský prívetivým spôsobom, bez znalostí a potreby čokoľvek programovať.

3.1 Metódy a nástroje pre detekciu jazyka

Pre určenie najvhodnejšej metódy bolo potrebné z niekoľkých kritérií.

- zdroj obsahu textov - v tomto prípade obsah WWW
- dĺžka vstupného textu - nehrá rolu, očakávaný rozsah minimálne 50 znakov
- podporované jazyky - akékoľvek vrátane čínštiny
- vstupné kódovanie - predpokladá sa konverzia do UTF-8 a následná identifikácia
- možnosť jednoducho trénovať model pre nový jazyk na webových stránkach známeho jazyka
- otvorený zdrojový kód a jednoduchá implementácia v jazyku Java
-

Pri analýze možných riešení som uvažoval aj využití nejakej vhodnej knižnice, ale v dobe vývoja tejto časti práce som nenašiel žiadne uspokojivé riešenie v jazyku Java s otvoreným zdrojovým kódom s možnosťou pridávania nových jazykov. Riešenia na ktoré som pri prieskume narazil:

- patentované riešenie firmy XEROX - **Language Identifier 1.5**

- riešenie spoločnosti Lingua v jazyku Perl - <http://www.lingua-systems.com/language-identifier/Lingua-Lid-Perl-extension/>
- program v jazyku Python od Diamira Cavara - <http://ling.unizd.hr/~dcavar/LID/>
- online riešenie Google - <http://www.google.com/uds/samples/language/detect.html>

V súčasnosti je k dispozícii riešenie podobné môjmu riešeniu v jazyku Java od Cedrica Champeau http://www.jroller.com/melix/entry/nlp_in_java_a_language. Nakoľko v dobe programovania tejto časti práce nebolo žiadne vhodné riešenie k dispozícii rozhodol som sa implementovať vlastné riešenie.

Na začiatku bolo potrebné rozhodnúť sa ktorú metódu pre identifikáciu jazyka zvoliť pre implementáciu, aby bola vhodná pre tento projekt. Predtým ako som dospel ku konečnému rozhodnutiu, podrobil som krátkej analýze štyri metódy ktorých popis je voľne dostupný a publikovaný na internete.

3.1.1 Krátke slová a unikátne kombinácie písmen

Táto metóda vychádza z článku prezentovaného G. Grefenstettom na konferencii v roku 1995 [15]. Metóda krátkych slov najprv pre každý jazyk vytvorí model z dostatočne veľkého objemu známeho textu. Pre každý jazyk sa zaznamená početný výskyt slov v dĺžke do päť znakov, ktoré sa vyskytujú aspoň tri krát a uloží sa do databáze. Slová zo vstupného textu sa porovnávajú proti modelu pre každý jazyk. Podľa výskytu slov a ich početnosti sa pre každý jazyk spočíta výsledné skóre. Najvyššie skóre by mal dosiahnuť jazyk v ktorom je text napísaný. Výhodou tejto metódy relatívna jednoduchosť na implementáciu a výpočtová nenáročnosť. Veľkou nevýhodou tejto metódy je že požaduje dostatočne dlhý vstupný text aby získala dostatočne veľa slov úspešnú identifikáciu. Rovnako metóda nie je vhodná pre ideografické znaky ktoré používa napríklad čínština.

Podobným variantom tohto riešenia je vytváranie modelu z častých kombinácií písmen ktoré sú unikátne pre každý jazyk. Príklady:

- der, die - nemčina
- the, ery - angličtina
- eux - francúzština
- ní, ře - čeština
- ie, st - slovenčina

Táto metóda rovnako požaduje dostatočne dlhý vstupný text a jej výsledky sú pre niektoré podobné jazyky nie veľmi presné.

3.1.2 Štatistická metóda

Ďalšia metóda pochádza od *T. Dunninga* [16]. Pre každý jazyk sa spočíta na tréningových dátach pravdepodobnosť výskytu jednotlivých postupností znakov (najčastejšie v dĺžke 2-3 znaky) na dostatočne veľkom objeme tréningových dát. Následne sa pre daný vstupný text a jeho postupnosti znakov zoberú pravdepodobnosti výskytu v jednotlivých model. Model jazyka s najlepším výsledkom by mal identifikovať jazyk.

Podľa *T. Dunninga* model vykazuje dobré výsledky už pri veľmi krátkych anglických a španielskych textoch - 92% pri dĺžke vstupu 20 znakov a veľkosti tréningových dát 500k znakov.

3.1.3 N-gramová metóda

N-gramová metóda je založená na podobnom princípe ako metóda štatistická. Vychádza z diela *W.B. Canvara* [17]. N-gram je postupnosť znakov s dĺžkou N . Vychádza z myšlienky Zipfovo zákona ktorý hovorí že početnosť výskytu udalosti je inverzne proporcionálna jeho poradiu. To znamená že niektoré N-gramy sa vyskytujú v danom jazyku oveľa častejšie ako iné. Graf ukazuje vzťah medzi frekvenciou výskytu a poradím podľa Zipfovho zákona.

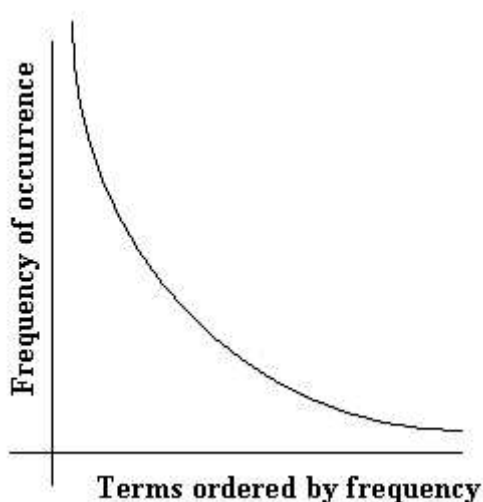


Diagram 9: Zipfov zákon

Podobne ako pri ostatných metódach je najprv vybudovaný model početností N-gramov pre daný jazyk. V druhom kroku vypočíta poradie výskytu N-gramov vo vstupnom texte a porovnaním k modelu sa získa skóre. Jazyk textu by mal získať najvyššie skóre. Detailné vysvetlenie tejto metódy je v kapitole 3.2 N-gramova .

3.1.4 Vyhodnotenie a porovnanie metód

Vyššie spomenuté metódy zahŕňajú len niektoré z mnohých metód. Medzi ďalšie patrí Kompresná metóda na princípe PPM (Prediction by partial matching) algoritmu alebo Kukuiho metóda ktorá je variantom štatistickej metódy, ktorá je vznikla ako alternatíva pre ázijské jazyky.

Pre moju prácu som sa rozhodol implementovať N-gramovú metódu nakoľko jej implementácie je relatívne jednoduchá, nevyžaduje žiadnu znalosť identifikovaného jazyka a je dostatočne úspešná už aj pri krátkych vzorkách vstupného textu. Jej hlavnou nevýhodou bola v danej dobe bola nevhodnosť pre ázijské jazyky využívajúce multi-byte kódovania, to však s príchodom kódovania Unicode ktoré sa využíva v jazyku Java stráca na význame.

3.2 N-gramova metóda

Ako som už bolo uvedené v úvode kapitoly, N-gramová metóda vychádza zo štatistického výskytu N-gramov v danom jazyku. N-gramy sú postupnosti N-znakov v danom slove. Ako príklad uvažujem slovo ŠKOLA. Dostaneme takýto rozpad na N-gramy:

- 1-gramy: Š, K, O, L A
- 2-gramy: _Š, ŠK, KO, OL, LA
- 3-gramy: _ŠK, ŠKO, KOL, OLA, LA_
- 4-gramy: _ŠKO, ŠKOL, KOLA, OLA_
- 5-gramy: _ŠKOL, ŠKOLA, KOLA_
-

Ako je vidieť na príklade, začiatok aj koniec slova je nahradený podtržítkom a počíta sa za jeden znak.

3.2.1 Vybudovanie referenčného modelu

V prvej fáze je potrebné postaviť model pre každý jazyk, ktorý by mal byť identifikovaný. Na začiatku je rovnako potrebné sa rozhodnúť s ako veľkými N-gramami sa bude pracovať. Je zrejmé, že veľké N-gramy ($N > 5$) nie je potrebné brať do úvahy, nakoľko ich početnosť je veľmi nízka. Pre účely tejto práce som sa rozhodol využiť N-gramy o veľkosti 2 - 4. Ešte pred spracovaním textu do modelu N-gramov sa všetky medzery, interpunkcia a konce riadkov vypustia a miesto nich sa použije podtržítka, nakoľko tieto znaky nemajú pre identifikáciu jazyka význam.

Aj keď volíme veľkosť N-gramu 1-4, počet N-gramov je veľmi vysoký. Napríklad pri vstupnej vzorke veľkosti 10MB pre katalánčinu je nájdených 137827 rôznych N-gramov veľkosti 2-4. Preto pre profil jazyka postačí uložiť iba najčastejšie sa vyskytujúce N-gramy. Podľa W. Canvara [17] pre úspešnú identifikáciu postačí 300 najčastejšie sa vyskytujúcich N-gramov bez ohľadu na ich typ, pri väčšom množstve dát by vstupné dáta mohli byť ovplyvnené kontextom dokumentu. Týmto spôsobom je potrebné spočítať model pre každý jazyk. Profilom každého jazyka teda bude poradie 300 najčastejšie sa vyskytujúcich N-gramov.

3.2.2 Výpočet skóre

Pri spracovaní vstupného textu sa spočíta rovnaký N-gramový profil (poradie najčastejších 300 N-gramov) pre vstupný text. Pre úspešnú identifikáciu už stačí len porovnať získaný profil s profilmi jednotlivých jazykov. Pre výpočet vzdialenosti dvoch profilov je použitý veľmi jednoduchý algoritmus kde pod pojmom vzdialenosť rozumieme rozdiel poradia rovnakých N-gramov v rámci porovnávaných profilov. Viac ukazuje obrázok prebraný od W. Canvara [17]:

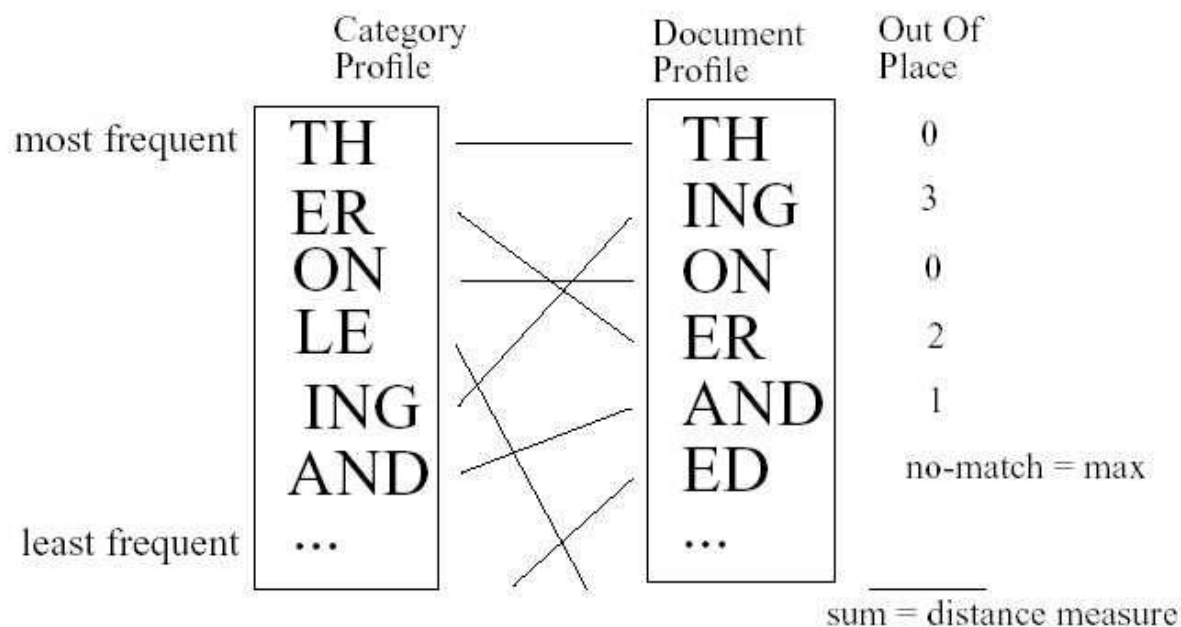


Diagram 10: Výpočet vzdialenosti dvoch profilov W. Canvar [17]

V prípade že sa N-gram z profilu nenachádza vo vzorke je mu pripísaná maximálna vzdialenosť, ktorá má hodnotu počtu N-gramov v referenčnom profile (300). Sumou vzdialenosti získame vzdialenosť vstupného a referenčného profilu pre každý jazyk. Profil ktorý ma najnižšiu vzdialenosť s najväčšou pravdepodobnosťou určuje jazyk vstupného textu.

Výsledky ktoré získal W. Canvar ukazujú pozoruhodné výsledky už pri veľmi malej množine vstupných dát. Výsledky ukazuje tabuľka:

Article Length	≤ 300	≤ 300	≤ 300	≤ 300	300	300	300	300
Profile Length	100	200	300	400	100	200	300	400
Newsgroup								
australia	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
brazil	70.0	80.0	90.0	90.0	91.3	91.3	95.6	95.7
britain	96.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0
canada	96.9	100.0	100.0	100.0	100.0	99.6*	100.0	100.0
celtic	100.0	100.0	100.0	100.0	99.7	100.0	100.0	100.0
france	90.0	95.0	100.0	*95.0	99.6	99.6	*99.2	99.6
germany	100.0	100.0	100.0	100.0	98.9	100.0	100.0	100.0
italy	88.2	100.0	100.0	100.0	91.6	99.3	99.6	100.0
latinamerica	91.3	95.7	*91.3	95.7	97.5	100.0	*99.5	*99.0
mexico	90.6	100.0	100.0	100.0	94.9	99.1	100.0	*99.5
netherlands	92.3	96.2	96.2	96.2	96.2	99.0	100.0	100.0
poland	93.3	93.3	100.0	100.0	100.0	100.0	100.0	100.0
portugal	100.0	100.0	100.0	100.0	86.8	97.6	100	100
spain	81.5	96.3	100.0	100.0	90.7	98.9	98.9	99.45
Overall	92.9	97.6	98.6	98.3	97.2	99.5	99.8	99.8

Diagram 11: Výsledky identifikácie, W. Canvar [17]

Z výsledkov vyplýva, že kvalita a pravdepodobnosť úspechu identifikácie závisí na kvalite a veľkosti referenčných dát a veľkosti vstupného textu. Medzi hlavné výhody tohto prístupu patrí vysoká úspešnosť už aj pre veľmi krátky vstupný text, nezanedbateľným aspektom je tiež možnosť trénovať model bez akejkol'vek znalosti jazyka. Dostupné referenčné dáta pre každý jazyk je možné jednoduchým spôsobom získať napríklad z Wikipédie.

3.3 Architektúra riešenia a implementácia

Implementácia N-Gramovej metódy by sa dala rozdeliť do dvoch oddelených procesov. V prvej fáze je potrebné vytvoriť kvalitný model pre každý jazyk a umožniť užívateľovi pridávať modely pre ďalšie jazyky.

V druhej fáze bolo potrebné implementovať algoritmus pre získanie N-gramov zo vstupného súboru ich porovnanie so získanými profilmi a výpočet skóre.

3.3.1 Vybudovanie modelu

Základnou požiadavkou pre úspešnú identifikáciu je získať kvalitné dáta pre tréning modelu - získanie N-Gramových profilov. Ideálnym riešením je získať dáta ktoré by boli náhodným spôsobom zozbierané z internetových stránok v danom jazyku. Ideálny zdroj je k dispozícii pre nekomerčné účely na stránkach *Univerzity v Lipsku* [18]. Tento zdroj obsahuje 100 tisíc fráz pre 16 európskych jazykov zozbieraných z internetu extrahovaných do čistého textu, čo je ideálny podklad pre vytvorenie profilov pre jednotlivé jazyky. Týmto spôsobom boli získané modely pre tieto jazyky. Zoznam kódov pre získané jazyky ktorý sa používa aj v XML:

- | | | |
|-----------------|-------------------|----------------|
| • de - nemecký | • fr - francúzsky | • no - nórsky |
| • dk - dánsky | • it - taliansky | • se - švédsky |
| • ee - estónsky | • jp - japonský | • tr - turecký |
| • en - anglický | • kr - kórejský | |
| • fi - fínsky | • nl - holandský | |

Nakoľko tento zoznam neobsahuje český a slovenský jazyk a cieľom bolo vytvorenie univerzálneho nástroja ktorý by dokázal pridať akýkoľvek ďalší jazyk, bolo by vhodné mať možnosť vytvárať modely nielen z textu ale aj z internetových stránok. Za týmto účelom bol vytvorený jednoduchý skript ktorý pre daný kód jazyka sťahuje náhodné stránky z lokálnej Wikipédie. Takto získané dáta už môžu slúžiť ako podklad získania modelu pre akýkoľvek svetový jazyk, ktorý má dostatok stránok na Wikipédii.

Postup použitý pri vytvorení modelu z referenčných dát ukazuje nasledujúci diagram:

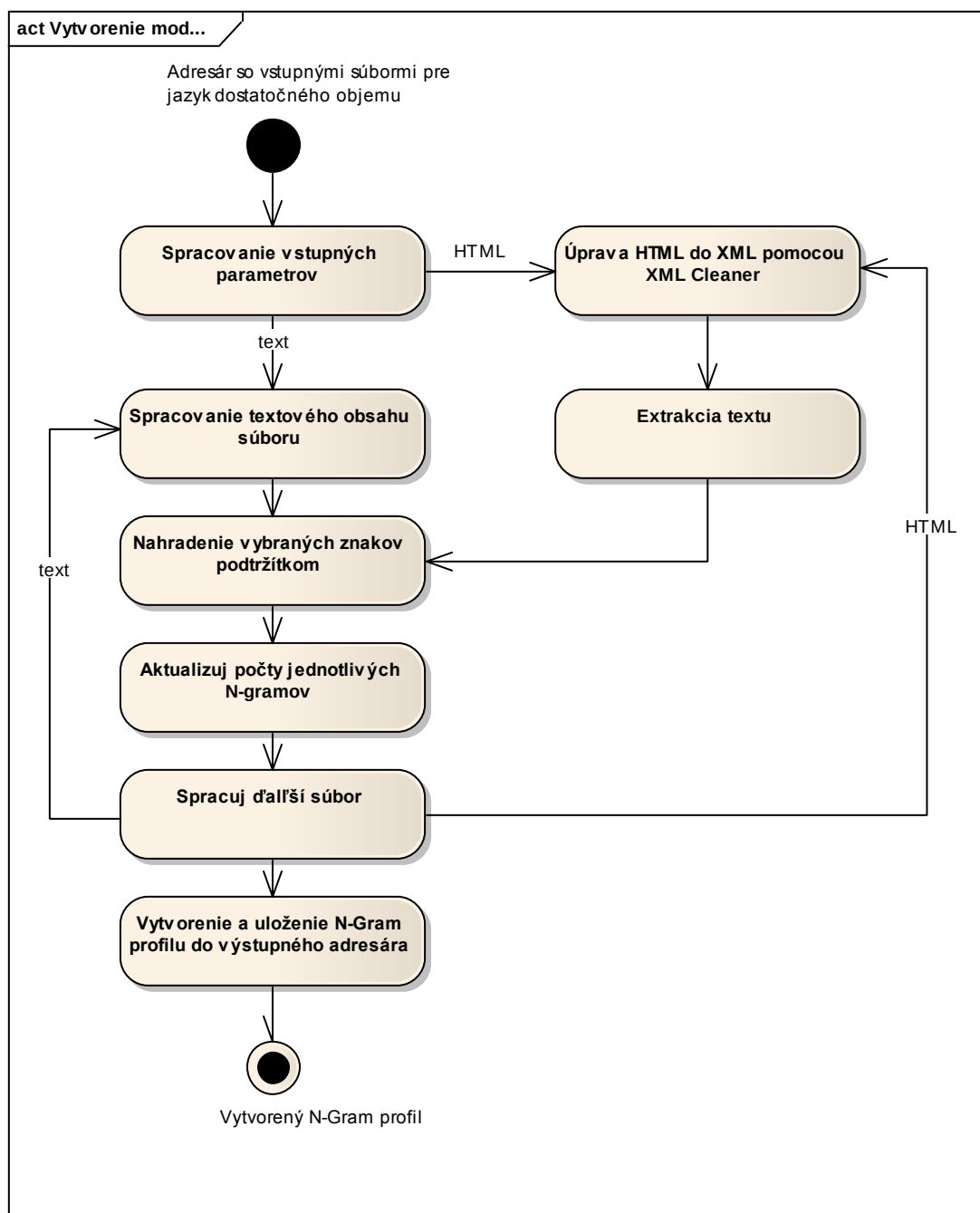


Diagram 12: Vytvorenie referenčného n-gramového modelu

V prvom kroku sa spracujú vstupné parametre pre budovanie modelu. Nutné vstupné parametre sú:

- kód jazyka pre ktorý sa model buduje (napr. sk, cz, it ..)
- vstupný adresár v ktorom sa nachádzajú súbory pre spracovanie
- výstupný adresár kam sa uloží model v tvare <kod_jazyka>.ngram
- typ vstupu - html/text

V prípade, že vstupné dáta sú vo formáte HTML, použijem získanie dát vlastný program pre prevod HTML do dobre formovaného XML. Z dobre formovaného XML získať textové dáta je už triviálna záležitosť. Pozorovaním získaných dát z Wikipédie je potrebné ešte dáta ďalej

upraviť. Preto sa pre extrakciu textu vynechá obsah odkazov - elementov <a>, nakoľko práve v obsahu odkazov sa opakujú stále tie isté dáta, prípadne sa tam heslá často krát ktoré nie sú vhodné pre budovanie modelu.

V ďalšom kroku je nutné zaradiť všetky znaky ktoré nie sú písmená podtržítom (medzery, koniec riadku, čísla, interpunkcia ...). V prípade výskytu sekvencie takýchto znakov je použité len jedno podtržítko pre celú sekvenciu aby nevznikli N-gramy obsahujúce podtržítka. V tejto fáze sa pristúpi k získaniu N-Gramov z textu. Každý N-gram je uložený v hash mape, kde kľúčom je N-gram a hodnota je počet výskytov. Pri novom N-Grame sa pridá nový záznam s počtom jedna, inak sa len inkrementuje počet.

Po spracovaní všetkých súborov získame hash mapu všetkých N-Gramov a ich počtov zo všetkých súborov v adresári. Počet N-gramov pre bežný európsky jazyk sa pohybuje medzi 100 až 200 tisícmi. K vytvoreniu profilu pre daný jazyk sa vytvorí usporiadaný zoznam podľa počtu výskytov z ktorého sa zoberie prvých 1000 prvkov.

V poslednom kroku sa získaný profil uloží do výstupného súboru. Každý výstupný súbor má 1000 riadkov. Príklad ukazuje prvých 20 riadkov z profilu spočítaného pre český jazyk.

Príklad cz.ngram

e_	0.02097703	35519
a_	0.019271417	32631
í_	0.012715911	21531
st	0.0117940055	19970
_p	0.011781013	19948
o_	0.011680023	19777
_s	0.011175072	18922
ov	0.010917576	18486
_v	0.009455285	16010
y_	0.009226137	15622
en	0.00897691	15200
ro	0.008953877	15161
u_	0.008900134	15070
ní	0.008335534	14114
_n	0.008313091	14076
na	0.007995946	13539
i_	0.0076516345	12956
te	0.0074106753	12548
ce	0.0071248314	12064
to	0.007044512	11928

Ku každému n-gramu je uvedená štatistická pravdepodobnosť výskytu vo vzorke a počet výskytov. Tieto údaje slúžia len pre kontrolu a ďalej sa nespracovávajú.

3.3.2 Detekcia jazyka

Pre úspešné vyhodnotenie jazyka vstupného textového súboru alebo textu je potrebné porovnať získať z textu jeho profil a pre každý jazyk spočítať skóre ktoré je definované ako vzdialenosť dvoch profilov. Postup spracovania zobrazuje nasledujúci diagram:

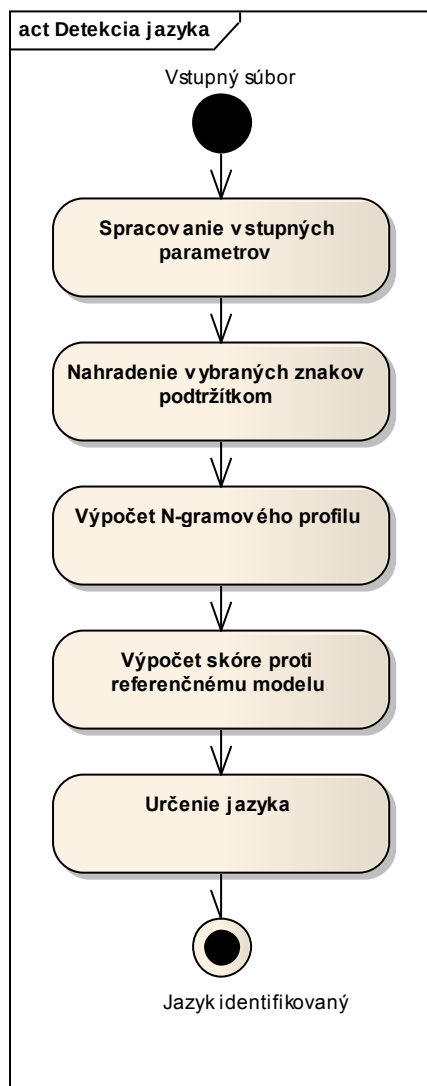


Diagram 13: Identifikácia jazyka

V prvom kroku je potrebné spracovať vstupné parametre. Ako vstup je očakávaný čistý text bez HTML značiek. Potrebné vstupné parametre sú:

- vstupný súbor
- výstupný adresár kde sú ložené modelové profily jednotlivých jazykov
-

V druhom kroku sa text vstupného súboru upraví rovnakým spôsobom ako pri vytváraní modelu - sekvencie iných znakov ako sú písmena sa nahradia podtržítom. Z tohto modelu sa spočíta N-gramový model vstupného textu - maximálne 1000 najčastejšie sa vyskytujúcich N-gramov.

V nasledujúcom kroku sa sa pre každý referenčný jazyk a jeho profil vypočíta skóre daného jazyka. Výpočet skóre pre dva N-gramové modely je detailne popísaný v kapitole 3.2 N-gramova . Jazyk ktorého profil získa v porovnaní so vstupným profilom najvyššie skóre sa považuje za jazyk vstupného súboru alebo textu. Návratovou hodnotou je kód daného jazyka.

3.3.3 Organizácia tried a dátový model

Usporiadanie tried v balíčku ngram znázorňuje diagram:

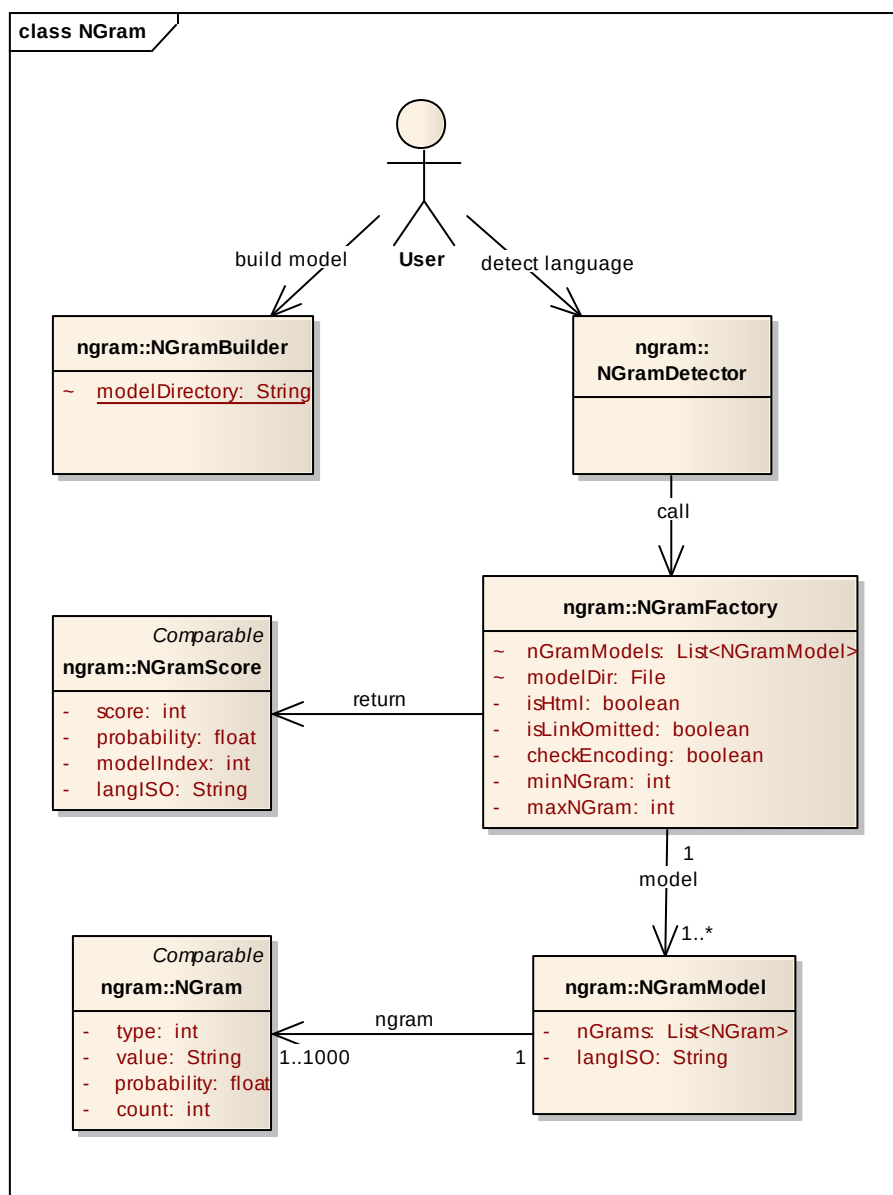


Diagram 14: Dátový model

NGramBuilder

Trieda sa používa pre vybudovanie modelu pre nový jazyk. Obsahuje metódy ktoré je možné volať priamo z príkazového riadku za účelom pridania modelu pre nový jazyk. Podporuje budovanie modelu z textových dát alebo z HTML.

NGramDetector

Trieda sa používa pre detekciu jazyka, vstupom pre volanie z príkazového riadku je adresár s uloženými modelmi a vstupný text. Výstup je ISO kód detekovaného jazyka.

NGramFactory

Trieda ktorá pri inicializácii nahrá profily pre všetky jazyky zo súborov. Obsahuje metódy ktoré umožnia porovnanie vstupu a vrátiť výsledné skóre - vzdialenosť vstupného textu od jednotlivých modelov.

NGramScore

Trieda ktorá pre daný jazyk obsahuje jeho skóre. Rovnako je možné spočítať skóre na základe pravdepodobností.

NGramModel

Obsahuje zoznam N-gramov pre každý jazyk

Ngram

Reprezentuje jeden N-Gram. Obsahuje okrem iných hodnôt aj počet výskytov a pravdepodobnosť, ktoré sa v algoritme nepoužívajú.

4 Testovanie a výsledky

Pre overenie výsledkov funkčnosti algoritmu boli vykonané tri sady testov. Prvá sada sa zameriava na testovanie opráv zle formovaných súborov a overenie či po spracovaní sú súbory dobre formované. Druhá sada testov je podobná ako je Canvarov test v kapitole 3.2 N-Gramová metóda. Overuje úspešnosť identifikácie jazyka na obyčajnom texte pre rôzne modely jazykov a rôzne dlhé vstupy. Posledná sada testov kombinuje obidva prístupy a testuje úspešnosť identifikácie XML súborov obsahujúcich HTML stránky. Cieľom testov je aj overiť spotrebu systémových prostriedkov a rýchlosť spracovania.

Všetky vstupné aj výstupné súbory použité v testoch sú dispozícii na DVD v adresári „data“. Testovanie aplikácie prebiehalo na PC platforme s parametrami:

- OS - Windows 7 Ultimate
- CPU - C2D T9500 2.5GHz
- RAM - 3GB
- Java - 1.6.0_06
- HDD 250GB 5400 ot/min

4.1 Testovanie opravy zle formovaného XML

Cieľom prvého testu je overiť či program pre úpravu normalizáciu XML generuje z pokazených súborov dobre formované súbory. Ako vstupné dáta budú použité testovacie sady konzorcia W3C, príklady obsahujúce HTML a vlastné ilustračné príklady použité v tejto práci. Všetky vstupné aj výsledné súbory sú k dispozícii na priloženom DVD. Tento test by mal overiť aj výkonnosť a spotrebu systémových prostriedkov pri spracovaní väčšieho množstva dát.

4.1.1 Zdroje dát zle formovaného XML

Hlavným zdrojom dát zle formovaného XML pre testovanie sú testovacie sady priamo od konzorcia W3C - *Extensible Markup Language (XML) Conformance Test Suites* [13]. Z týchto sád som vybral tie ktoré obsahujú zle formované XML. Spolu je to 1200 súborov. Podľa pôvodu dát zoznam obsahuje tieto sady:

- James Clark "XMLTEST" (186 súborov)
- IBM tests (732)
- SUN-written testcases (57)
- Edinburgh University (62)
- NIST/OASIS test suite (349)

Ani jeden z týchto súborov neobsahuje dobre formované XML.

Druhou sadou sú dáta z projektu *Egothor* [19]. Táto sada obsahuje dáta zozbierané z internetových stránok českých, slovinských a anglických. Spolu ide o 30 súborov. Každý

súbor obsahuje 1000 náhodných internetových stránok. Súbor je v tvare ako je uvedené na začiatku kapitoly 3 Detekcia jazyka. Ani jeden súbor neobsahuje dobre formované XML. Posledný test je vykonaný na príkladoch ktoré sú k dispozícii ako ilustračné príklady v kapitole 2.2 Zle formované xml a jeho riešenia.

4.1.2 Postup pri testovaní zle formovaného XML

Cieľom tohto testu je overiť či výsledkom opravy všetkých vstupných súborov bude dobre formované XML. Každá sada sa nahrá do vlastného adresára a spustí sa program XML Cleaner pre každý adresár.

Pre druhú sadu dát z projektu Egothor sa overuje aj časová a pamäťová náročnosť spracovania, nakoľko sa jedná o relatívne veľké súbory. Pri tomto teste sa kontroluje aj správnosť menných priestorov. Overenie správnosti sa vykoná spracovaním výstupného súboru pomocou štandardného parseru SAX z knižnice *jdom* [10] vybudovaním JDOM objektového modelu. Keďže tento parser overuje aj externé entity a referencie, je nahrávanie externých entít a DTD vypnuté. V prípade že parser zlyhá vyhodí *JDOMException* a na štandardný výstup sa vypíše hláška o zle formovanom XML.

Pre vizuálnu kontrolu spracovania bol použitý nástroj XML Spy 2007 ktorý rovnako obsahuje nástroj pre kontrolu dobre formovaného XML.

4.1.3 Výsledky a záver testov normalizácie XML

Z prvého testu sú všetky súbory okrem desiatich súborov dobre formované. Vo všetkých prípadoch zle formovaných výstupných súborov sa vyskytla tá istá chyba, zle formované meno elementu alebo atribútu. Dochádza tu k paradoxnej situácii, pretože mená súborov obsahujú okrajovo používané Unicode znaky. Podľa špecifikácie by ale mali byť správne. Jedná sa napríklad o znaky `#x036F`, `#x0300`, `#xFFFD`, `#x017F` a iné, ktoré sú ale podľa pravidiel gramatiky [G4] a [G4a] piatej edície špecifikácie XML správne a v mene atribútu aj elementu povolené.

Výsledné súbory z druhej sady sú všetky dobre formované. Najčastejšou komplikáciou je spracovanie elementu `script` ktorý obsahuje najčastejšie Javascript, ktorý často krát nie je dobre formovaný textový obsah elementu. Medzi ďalšie časté chyby patrí použitie HTML entít, atribúty bez hodnôt, nesprávne deklarované menné priestory a použitie zakázaných znakov v atribútoch.

Časová náročnosť spracovania 30 súborov v celkovej veľkosti asi 550MB asi 290s. Pamäťová náročnosť spracovania je až 850MB nakoľko štruktúra niektorých súborov je veľmi komplexná.

4.2 Testovanie identifikácie jazyka v texte

Pre testovanie identifikácie sa pokúsim urobiť podobný test ako bol urobil W. B. Canvar a jeho výsledky sú v kapitole 3.2 N-gramova . Predpoklad je, že výsledky by mali byť minimálne rovnako kvalitné, nakoľko som mal k dispozícii dostatočné množstvo dát pre vybudovanie modelu.

Zaujímavé bude porovnať dáta podobných jazykov, obzvlášť sa sústredím na porovnanie češtiny a slovenčiny a pre komplexný obraz tiež severské jazyky ktoré ku ktorým existuje dostatočné množstvo dát pre model - švédčina, nórčina a dánčina. Podobne ako Canvar sa pokúsím overiť či je lepšie použiť 300 alebo viacej najčastejšie používaných N-gramov pre model daného jazyka. Ako referenčnú hodnotu som použil 1000 N-gramov.

Testy by mali rovnako ukázať aká je zmysluplná minimálna dĺžka vstupného textu pre identifikáciu jazyka. Predpokladám, že 500 znakov by malo stačiť pre takmer 100% úspešnosť.

4.2.1 Zdroje dát pre modely a vstupné súbory

Ako zdroje dát pre výpočet modelov sú použité dáta získané z Univerzity v Lipsku [18]. Pre každý jazyk bol vytvorený model z objemu dát približne 10M znakov. Výnimkou je čeština a slovenčina, kde tieto dáta neboli k dispozícii. Jednou možnosťou bolo použiť dáta získané z wikipédie. Nakoľko som ale chcel mať rovnaké referenčné dáta rozhodol som sa použiť dostatočne veľký dlhý voľne dostupný rovnaký text preložený do oboch jazykov, preto som modely natrénoval na texte Starého Zákona z Biblie ktorý je voľne k dispozícii v oboch prekladoch a má asi 3M znakov. Preto môžu byť výsledky oproti ostatným jazykom trochu nadhodnotené nakoľko text neobsahuje žiadne cudzojazyčné kusy textov.

Ako vstupné dáta som použil krátke náhodne vybrané úseky zo zdrojových textov v požadovanej veľkosti.

4.2.2 Postup pri testovaní identifikácie jazyka

Aby sa porovnal vplyv veľkosti modelu, tak sú pre každý jazyk použité dva modely - jeden obsahuje 300 a druhý 1000 najčastejšie sa vyskytujúcich N-gramov v danom jazyku získaných z rovnakých zdrojových dát. Pre zistenie vhodnej dĺžky textu pre identifikáciu bol test opakovaný pre štyri rôzne dlhé vstupy 25, 50, 100 a 500 znakov.

Za dostatočný počet vstupov pre elimináciu rozptylu bol test pre každý jazyk prevedený 300 krát na náhodnom vstupe danej dĺžky. Pre rôzne veľkosti modelu je použitý rovnaký vstup. Zvolená veľkosť N-gramov je 2-4.

4.2.3 Výsledky a záver testovania identifikácie jazyka

Výsledky testovania identifikácie jazyka na čistom texte nepochádzajúcom z HTML zobrazuje nasledujúca tabuľka:

Model	300	300	300	300	1000	1000	1000	1000	
Vstup	25	50	100	500	25	50	100	500	
cz	82,0	89,7	97,3	100,0	90,3	97,3	99,7	100,0	94,5
de	82,3	98,0	99,3	100,0	87,7	98,7	99,7	100,0	95,7
dk	65,3	73,3	84,7	99,7	72,3	82,0	93,0	99,7	83,8
ee	87,0	97,0	100,0	100,0	93,3	98,3	100,0	100,0	97,0
en	83,3	97,3	100,0	100,0	89,0	99,0	100,0	100,0	96,1
fi	89,3	97,7	100,0	100,0	95,3	98,3	100,0	100,0	97,6
fr	89,0	98,0	100,0	100,0	93,3	99,7	100,0	100,0	97,5
it	89,7	98,3	100,0	100,0	94,3	100,0	100,0	100,0	97,8
jp	98,3	99,7	100,0	100,0	98,7	99,7	100,0	100,0	99,5
kr	100,0	99,7	100,0	100,0	100,0	99,7	100,0	100,0	99,9
nl	77,3	91,0	98,0	100,0	84,3	94,7	99,0	100,0	93,0
no	60,0	69,7	79,7	98,7	60,3	86,0	90,3	100,0	80,6
se	57,3	75,3	90,7	99,7	73,0	86,0	96,0	100,0	84,8
sk	78,0	93,0	96,0	100,0	89,7	98,7	98,7	100,0	94,3
tr	88,7	95,0	99,7	100,0	91,3	96,3	99,7	100,0	96,3
	81,8	91,5	96,4	99,9	87,5	95,6	98,4	100,0	93,9

Diagram 15: Výsledky identifikácie jazyka

Získané výsledky sú podobné tým ktoré získal Canvar pri svojom teste. Nepotvrdila sa ale hypotéza použiť iba 300 najčastejšie sa vyskytujúcich N-gramov nakoľko výsledky sú preukázateľne lepšie v prípade použitia modelu s 1000 N-gramami. V prípade dĺžky vstupu 50 znakov je chybovosť polovičná

Výsledky ukazujú, že v prípade monolingválneho textu nemá význam testovať dlhšie vstupy ako 500 znakov keďže pri tejto dĺžke je úspešnosť prakticky 100% pre všetky pozorované jazyky. Ďalším pozorovaným výsledkom je vysoká úspešnosť identifikácie pri jazykoch ktoré sa od seba dostatočne odlišujú a to aj pri veľmi krátkych vstupoch. S výnimkou severských jazykov a holandčiny prakticky u všetkých jazykov dosahuje úspešnosť pri modeli veľkosti 1000 úspešnosť okolo 90% už pri dĺžke vstupu 25 znakov a to aj v prípade češtiny a slovenčiny.

Podrobnejšia analýza výsledkov skóre v priebehu testu pre jazyky s nižšou mierou identifikácie ukazuje na vysokú podobnosť severských jazykov, vzorka v nórskom jazyku je v prípade zlej identifikácie takmer vždy identifikovaná ako dánčina alebo švédčina a opačne. Pozorovaním získaných skóre pre český a slovenský jazyk sa ukazuje, že 90% nesprávne identifikovaných českých textov je ako jazyk priradená slovenčina a opačne. Toto platí pre vzorky o dĺžke aspoň 50 znakov. Pri textoch dĺžky 25 znakov hrá značnú rolu výskyt cudzojazyčného textu.

Časová náročnosť identifikácie pre 100 vzoriek vstupného textu o dĺžke 500 znakov je v priemernom prípade asi 13s pre 15 jazykov. Časová náročnosť identifikácie pre jeden súbor veľkosti 10MB je asi 10s, rovnako pre 15 jazykov. Spotreba pamäte v tomto prípade je asi 70MB. Pri bežiacom procese je vyťažené len jedno jadro na 100% nakoľko program nie je optimalizovaný pre multiprocessing.

4.3 Test identifikácie jazyka HTML stránok

Cieľom tohto testu je zistiť ako funguje identifikácia jazyka na dátach získaných z náhodných HTML stránok a komplexne otestovať fungovanie programu ako celku. Pred identifikáciu je potrebné extrahovať text z HTML stránok. Preto je v prvej fáze potrebné previesť HTML do dobre formovaného XML a následne z neho extrahovať text ktorý bude predmetom identifikácie.

4.3.1 Zdroje dát HTML stránok

Referenčný model jazykov pre výpočty N-Gramových profilov je identický ako v predchádzajúcom teste. Počet jazykov je tiež 15. Ako vstupný súbor je použitý zoznam 1000 náhodných internetových stránok s doménou .cz. Súbor je v tvare ako je uvedené na začiatku kapitoly 3 Detekcia jazyka. Ako element pre identifikáciu je preto zvolený <DOCUMENT>.

4.3.2 Postup pri testovaní identifikácie jazyka z HTML

Pre súbor obsahujúci 1000 HTML stránok - každá stránka je zabalená v elemente <DOCUMENT> sa spustí program XmlCleaner s parametrami `src=inputFile langtag=DOCUMENT dest=outputFile ngrammodel=modelDir`. Cieľom je zistiť úspešnosť identifikácie, rýchlosť spracovania a podrobnejšie overiť príčiny prípadného zlyhania identifikácie. Predpoklad je že stránky by mali byť v českom jazyku.

4.3.3 Výsledky a záver testovania identifikácie jazyka z HTML

Celkové výsledky identifikácie zobrazuje nasledujúca tabuľka:

jazyk	cz	en	sk	de	iné	Spolu
počet	846	87	29	16	22	1000
pomer	84,60%	8,70%	2,90%	1,60%	2,20%	100,00%

Diagram 16: Výsledky identifikácie jazyka HTML stránok

Z 1000 stránok z domény CZ bolo 84.6% identifikovaných ako stránky v českom jazyku. Z ostatných stránok je asi polovica identifikovaná ako anglická, z ostatných jazykov ešte výraznejšie počty dosahuje nemčina a slovenčina ostatné jazyky sú zastúpené v zanedbateľnom množstve. Výsledkov v inom jazyku ako českom je predmetom ďalšieho rozboru, nakoľko úroveň úspešnej identifikácie na úrovni maximálne 84% je nízka. Výsledky rozboru zachytáva ďalšia tabuľka:

dôvodo výsledku	iný jazyk ako cz - ok	iný jazyk ako cz - ok	iný jazyk ako cz - ok	cz text bez diakritiky	viacjaz. Text	viacjaz. Text	chybný výsledok	chybný výsledok	Spolu
výsledok	en	sk	iny	sk	en	iný	sk	iný	
počet	59	10	6	29	28	6	4	12	154
pomer	38,31%	6,49%	3,90%	18,83%	18,18%	3,90%	2,60%	7,79%	100,00%

Diagram 17: Rozbor chybových výsledkov pri HTML stránkach

Zo 154 chybných výsledkov takmer polovica identifikovaná správne nakoľko zdrojové stránky neboli v českom jazyku. V pätine prípadov zlyhala identifikácia nakoľko stránka bola viacjazyčná väčšinou s prevahou inojazyčného textu. Čisto chybne identifikovaných českých

stránok je 23%. Najčastejšou príčinou zlej identifikácie je vstup bez diakritiky ktorý je identifikovaný ako slovenčina, ďalšou častou príčinou su stránky obsahujúce nezmyselne krátke texty.

Z uvedených výsledkov vyplýva celková úspešnosť identifikácie v tomto teste okolo 92-93% a pri zmysluplne dlhých stránkach ktoré nie sú vo viacerých jazykoch sa úspešnosť pohybuje okolo 97-98%. Pre úplnosť je potrebné dodať, že chybovosť môže byť o trochu vyššia, nakoľko, chybne identifikované stránky sa v minimálnom množstve mohli vyskytnúť aj na stránkach ktoré boli identifikované ako české.

Časová náročnosť spracovania súboru obsahujúceho 1000 HTML stránok celkovej veľkosti asi 25MB je 463s. Z toho 13s trvá normalizácia do XML a ostatok pripadá na identifikáciu jazyka. Pri počte 1000 dokumentov to pre jeden dokument vychádza priemerne na 450ms. SW a HW konfigurácia pre test je identická ako v minulom teste. Program počas svojho behu spotrebuje asi 400MB pamäte, relatívne vysoká spotreba pamäte je daná tým že pre celý súbor je pri normalizácii XML vytvorený jeden objektový model.

Tento test bol realizovaný dvakrát, po prvom pokuse bola urobená optimalizácia pre vyradenie obsahu elementu `<script>` a `<style>` z identifikácie jazyka. Nakoľko obsah týchto elementov nie je text stránky ale väčšinou JavaScript čo môže viesť k identifikácii stránky ako anglickej.

5 Záver

Hlavným cieľom tejto práce bolo vytvoriť program ktorý umožní spracovať akýkoľvek XML súbor takým spôsobom aby bol ďalej použiteľný. Podľa vykonaných testov, výsledný program vytvára dobre formované výstupy, z tohto pohľadu sa cieľ dá považovať za splnený.

Ďalšou požiadavkou bolo zachovanie vernosti štruktúry a obsahu pôvodného súboru. Aby bolo možné overiť nakoľko je aj tento cieľ splnený, bolo by vhodné a potrebné nasadiť program do reálneho použitia, keďže pri testovaní referenčnej vzorky súborov nie je vždy možné s istotou povedať nakoľko je tento cieľ splnený. Z tohto dôvodu by pre ďalší vývoj systému bolo vhodné implementovať funkcionality ktorá by počas úpravy vstupných dát zaznamenala všetky zmeny ktoré na vstupnom súbore vykoná, aby si to v prípade potreby mohol užívateľ skontrolovať a prípadne ručne opraviť.

Požiadavka umožniť spracovanie textového obsahu vstupného bola splnená v prípade HTML dát iba čiastočne. Ako bolo naznačené v závere druhej kapitoly, za predpokladu, že vstupný súbor obsahuje dáta z HTML by bolo vhodné pridať špeciálnu možnosť spracovania elementov `<style>` a `<script>`, ktorých obsah nie je vhodné použiť, ale bolo by vhodnejšie ho uzavrieť do CDATA sekcie alebo komentáru. Tým by sa vyhlo ich spracovaniu ako textového obsahu. Preto pridať možnosť parametrizovať mená elementov ktorých obsah by sa spracoval ako CDATA sekcia alebo komentár, by bolo vhodné a potrebné rozšírenie pre budúci vývoj.

Ako ďalší evolučný krok by bolo prirodzené implementovať kontrolu validity oproti DTD, takým spôsobom aby výsledný súbor bol nielen dobre formovaný ale aj validný.

V druhej časti práce sa podarilo úspešne implementovať identifikáciu jazyka s úrovnou úspešnosti cez 90% a to nielen na textovom súbore ale aj v rámci XML dokumentu pochádzajúceho z HTML, čím sa dá tento cieľ považovať za splnený. Nižšia úspešnosť bola dosahovaná iba v prípade škandinávskych jazykov ktorých podobnosť je veľmi vysoká. Do budúcnosti by bolo vhodné aby identifikácia okrem výsledného najpravdepodobnejšieho jazyku bola schopná zistiť aj nakoľko je výsledná identifikácia kvalitná, podobne ako to robí API spoločnosti Google.

Celá aplikácia je otvorená ďalšiemu vývoju či už v oblasti čistenia XML dokumentov alebo identifikácie jazyka. V celom projekte je braný ohľad na použitie otvorených technológií nezávislých na platforme alebo konkrétnom produkte.

6 Referencie

- [1] W3C, Extensible Markup Language (XML) 1.0 (Fifth Edition)
<http://www.w3.org/TR/REC-xml/>
- [2] W3C, Extensible Markup Language (XML) 1.0 (Third Edition)
<http://www.w3.org/TR/2004/REC-xml-20040204/>
- [3] W3C, Extensible Markup Language (XML) 1.0 (Second Edition)
<http://www.w3.org/TR/xml11/>
- [4] *Elliotte Rusty Harold, W. Scott Means*, XML v kostce, Pohotová referenční příručka, Computer press 2002
- [5] V. Nikic, HTML Cleaner project, <http://htmlcleaner.sourceforge.net/>
- [6] M. Wutka, DTDParser project, <http://www.wutka.com/dtdparser.html>
- [7] *Elliotte Rusty Harold*, Effective XML 2003
- [8] *Unicode Standard Annex #28*, <http://unicode.org/reports/tr28/tr28-3.html>
- [9] W3C, Character entity references in HTML 4,
<http://www.w3.org/TR/html4/sgml/entities.html>
- [10] *JDOM Project*, <http://www.jdom.org/>
- [11] *Log4j project*, <http://logging.apache.org/log4j/1.2/>
- [12] *ICU - International Components for Unicode*, <http://site.icu-project.org/>
- [13] W3C, Extensible Markup Language (XML) Conformance Test Suites,
<http://www.w3.org/XML/Test/>
- [14] W3C, Namespaces in XML 1.0 (Third Edition), <http://www.w3.org/TR/xml-names/>
- [15] *Greg Grefenstette*, Comparing Two Language Identification Schemes 1995
<http://www.xrce.xerox.com/Research-Development/Publications/1995-012>
- [16] *Ted Dunning*, Statistical Identification of Language, 1994
- [17] *William B. Cavnar and John M. Trenkle*, N-gram based categorization
- [18] *Universität Leipzig*, Corppus, <http://corpora.uni-leipzig.de/>
- [19] *Leo Galamboš a kol*, <http://www.egothor.org/>

7 Obsah DVD

K tejto práci je priložené DVD, ktoré obsahuje implementáciu programu Xml Cleaner popísaného v kapitole 2. DVD obsahuje aj ďalšie samostatné programy pre identifikáciu jazyka textu a pridávanie modelov pre nové jazyky. Pre ich spustenie je potrebné mať nainštalované prostredie Java Runtime environment (JRE) vo verzii 1.6.0 (je možné stiahnuť na stránkach <http://java.sun.com>).

Adresár **doc** obsahuje:

- text diplomovej práce vo formátoch PDF a DOC
- **užívateľskú príručku** programu v súbore **UserGuide.pdf**
- podadresár **javadoc** obsahuje programátorskú dokumentáciu k aplikácii.

Adresár **src** obsahuje zdrojový kód celej aplikácie.

Inštalácia spočíva v prekopírovaní adresára **programs** ktorý obshuje dva podadresáre:

- **xmlcleaner** obsahuje aplikáciu pre spracovanie XML súborov
- **ngramident** obsauhje programy pre identifikáciu jazyka textového súboru a pridanie nového jazyka.

Návod na použitie a spustenie programov je súčasťou **užívateľskej príručky**.

Adresár **ngram** obsahuje dva podadresáre:

- **model** n-gramové modely využívané pre identifikáciu jazyka v dvoch veľkostiach - 300 a 1000 uložených v separátnych adresároch
- **resources** obsahuje dáta z ktorých boli modely jednotlivých jazykov vytvorené, tieto náhodné útržky týchto súborov boli použité v testoch v kapitole 4.2 Testovanie identifikácie jazyka v texte

Adresár **testfiles** obsahuje súbory ktoré boli použité pri testoch. Podľa jednotlivých podadresárov:

- **examples** - príklady použité v texte práce
- **htmlang** - súbor použitý v teste v kapitole 4.3 Test identifikácie jazyka HTML stránok
- **notwftest** - obsahuje sady súborov použité v testoch v kapitole 4.1 Testovanie opravy zle formovaného XML
- **xml_korpus** - obsahuje dáta získané z projektu EGOTHOR použité v testoch v kapitole 4.1 Testovanie opravy zle formovaného XML