UNIVERZITA KOMENSKÉHO V BRATISLAVE

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Pravidlá na kontrolu slovenskej gramatiky

BAKALÁRSKA PRÁCA

Študijný program: Aplikovaná informatika

Študijný odbor: 9.2.9 Aplikovaná informatika

Školiace pracovisko: Katedra aplikovanej informatiky

Školiteľ: Miloš, Šrámek, prof. Ing. PhD.

Bratislava 2010

Luboš Lehotský

Poďakovanie

Chcem sa poďakovať svojmu školiteľovi prof. Ing Milošovi Šrámekovi, PhD. za odbornú a ochotnú pomoc pri vypracovaní tejto práce. Moje veľké poďakovanie patrí aj Zdenkovi Podobnému, na ktorého som sa mohol vždy obrátiť s technickými otázkami. V neposlednom rade ďakujem aj všetkým, ktorí mi vytvorili priestor a podporovali ma pri vypracovaní práce. Rodine, priateľom…

Ďakujem

Čestné prehlásenie

Čestne vyhlasujem, že zadanie bakalárskej práce som vypracoval pod odborným vedením svojho školiteľa prof. Ing Miloša Šrámka, PhD. s použitím citovaných zdrojov.

V Bratislave dňa: Luboš Lehotský

Abstrakt

Názov práce: Pravidlá na kontrolu slovenskej gramatiky

Autor: Luboš Lehotský

Katedra: Katedra aplikovanej informatiky

Fakulta: Fakulta matematiky, fyziky a informatiky

Univerzita: Univerzita Komenského v Bratislave

Vedúci práce: doc. Ing. Miloš Šrámek PhD.

Rozsah: 39

Miesto, rok: Bratislava, 2010

Táto bakalárska práca sa zaoberá gramatickým korektorom LanguageTool. Opisuje jeho štruktúru, technickú architektúru a praktické použitie. Hlavným cieľom práce je vytvorenie pravidiel na kontrolu gramatiky slovenského jazyka. Okrem toho je jej ambíciou aj priblíženie aplikácie potenciálnym spolupracovníkom na tomto komunitnom open-source projekte. V tejto oblasti práca približuje zásady tvorby nových pravidiel, možnosti vývojára, či spôsob aktivácie pravidiel v používateľovej verzii programu OpenOffice a v neposlednom rade sprístupnenie novej verzie pravidiel širokej verejnosti. Okrem toho práca obsahuje aj príklady vytvorených pravidiel s popisom možných gramatických situácii, v ktorých úspešne aj neúspešne fungujú.

Kľúčové slová: LanguageTool, OpenOffice, gramatický korektor, tagset slovník

Abstract

This bachelor`s thesis deals with the grammar corrector LanguageTool. It describes its structure, technical architecture and practical use. The main aim of this paper is to create rules for checking the grammar of the Slovak language. Except of previous function, thesis's ambition is to bring the application closer to potential co-workers on this communitarian open-source project. In this area, the thesis tries to explain the principles of creating new rules, possibilities for developers and the method of ruler`s activation in the user version of OpenOffice. The last, but not the least point of this thesis is to make a new version of the rules available for the general public. In addition, the work includes examples of established rules with description of possible situations in which they operate successfully or unsuccessfully.

Keywords: LanguageTool, OpenOffice, grammar corrector, tagset dictionary

Obsah

Úvod

V dnešnej dobe existuje takmer v každej oblasti niekoľko druhov softvéru, medzi ktorým si môže bežný používateľ slobodne vybrať. Väčšina používateľov však často siaha po najznámejšej, no nie vždy najkvalitnejšej verzii, a to najmä v prípade, keď nie je finančne limitovaná. Snaha veľkých spoločností poskytnúť zákazníkovi kompletný rozsah softvéru, pre bežné použitie v oblasti grafických editorov, multimediálnych prehrávačov, či kancelárskych balíkov, postupne vytvára závislosť zákazníka na tvorcovi. Finančná motivácia je dnes pre veľké množstvo koncových používateľov jedinou cestou, ktorou sa dostávajú do kontaktu s produktmi s otvorenou licenciou. Ak teda autori tohto druhu aplikácii (zväčša vyvíjaného na báze opensource licencií) chcú oslovovať používateľov aj inou cestou ako je finančná výhodnosť, ostáva im možnosť ponúknuť kvalitnejšiu alternatívu ku korporatívnym produktom. Jedným z týchto produktov je aj kancelársky balík OpenOffice, ktorému sa úspešne darí získavať si nových používateľov na úkor najznámejšieho kancelárskeho softvéru MS Office od spoločnosti Microsoft. Dnes je OpenOffice jeho plnohodnotnou náhradou a množstvo zanietených spoluautorov pracuje na jeho ďalšom zdokonalení. OpenOffice obsahuje veľké množstvo prídavných modulov, ktoré rozširujú jeho možnosti, pričom jedným z nich je aj LanguageTool (LT). Ide o aplikáciu na kontrolu gramatiky pomocou preddefinovaných pravidiel. Dnes ponúka komplexnú podporu pre viaceré jazyky a zlepšenie stavu v oblasti slovenčiny bolo mojou motiváciou, prečo som sa rozhodol zvoliť si spoluprácu na tomto projekte ako súčasť mojej bakalárskej práce.

1. Ciele práce

Prvotným a hlavným cieľom mojej prace bolo vytvorenie sady pravidiel na kontrolu slovenskej gramatiky pre spomínaný program LanguageTool. Popisu časti z nich sa venujem v poslednej podkapitole práce, pričom uvádzam aj príklady, v ktorých sú tieto pravdilá účinné alebo neúčinné. Okrem toho si však moja práca kladie za cieľ aj oboznámenie sa so samotnou aplikáciu, jeho architektúrou, spôsobom kontroly a možnosťou použitia v praxi. Tejto oblasti je venovaná celá druhá kapitola. Keďže moja práca bola súčasťou väčšieho komunitného projektu a nie je v schopnostiach jednotlivca pokryť taký obsiahly problém ako je definícia a aplikácia všetkých jazykových problémov v jazyku natoľko morfologicky pestrom akým je slovenčina, posledným cieľom, ktorý je definovaný v mojej práci, bolo ozrejmenie základov tvorby pravidiel, ich testovania, kompilácie a možnosti začlenenia do novej verzie existujúceho programu. V tejto oblasti sa práca štylizuje do pozície akejsi príručky pre potencionálneho spoluautora, pričom sa daná oblasť vyskytuje v podkapitole 3.2.

2. Aplikácia LanguageTool

2.1. Charakteristika

Program LanguageTool bol vytvorený Danielom Naberom v jazyku Python ako jeho diplomová práca a neskôr bol pretransformovaný a rozšírený v jazyku Java. Daniel Naber je v súčasnosti aj hlavným správcom celej aplikácie. Okrem neho však konkrétne pravidlá pre jednotlivé jazyky spravujú lokálni vedúci.

LanguageTool bol vyvíjaný predovšetkým pre projekt OpenOffice.org, avšak môže byť použitý aj ako samostatná aplikácia alebo funguje na báze server-side procesu. Nová API korektúra programu OpenOffice od verzie 3.0 bola uvedená v úzkej spolupráci s autorom a umožňuje hlbokú integráciu LanguageTool do kancelárskeho balíka. Je potrebné zdôrazniť, že niektoré rozhodnutia v oblasti dizajnu projektu boli aplikované s hlavným ohľadom na praktické riešenia. Napríklad, do úvahy sa brali a berú len riešenia, ktoré môžu byť ľahko udržiavané a zavádzané po malých krokoch.

V októbri 2008 získal projekt LanguageTool ocenenie Gold Award spoločenstva pre inováciu OpenOffice (OpenOffice.org Community Innovation Program) sponzorovaný spoločnosťou Sun Microsystem Inc. Na stránke rozšírení programu OpenOffice bolo zahájených asi 145 tisíc unikátnych sťahovaní programu LanguageTool.

Podpora jazykov v súčasnosti

LT v súčasnosti podporuje 18 jazykov na rôznych úrovniach. Niekoľko nových balíkov je v počiatočnej fáze (slovenčina, slovinčina, švédčina, islandčina), zatiaľ čo niektoré sa už dostali na veľmi vysokú úroveň (poľský, taliansky, ruský, rumunský, francúzsky, holandský, nemecký a anglický).

2.2. Rozdelenie textu

LanguageTool napísaný v Jave implementuje nasledujúci pracovný postup. Vstupný text je rozdelený (segmentovaný) na jednotlivé vety a slová. Vety bývajú oddelené niektorými špecifickými interpunkčnými znamienkami, ktoré program rozpozná a následne rozdelí text na konkrétne vety. Keďže tento prístup nie je stopercentný je potrebné ošetriť možné výnimky ako napríklad to, že bodka, ktorá sa najčastejšie používa na oddelenie viet má v jazykoch aj iné funkcie a vyskytuje sa aj v zápise skratiek či dátumov. Ďalšie znaky

primárne určené na ukončenie vety ako „?“ alebo „!“, sa v prípade, že ide o priamu reč vyskytujú aj v strede a celá veta pokračuje ďalej. Na ošetrenie týchto prípadov je nutná podrobnejšia analýza okolia týchto znakov, ktorá je v LT implementovaná.

Oddeľovač slov *Tokenizer* jednoducho rozdelí vstupný reťazec na jednotlivé slová na základe rozpoznania medzery alebo interpunkcie (v niektorých jazykoch ako je napríklad holandčina, apostrof, ak sa nevyskytuje spolu s medzerou neoddeľuje 2 slová). Prístup k pomlčkám sa v jednotlivých jazykoch líši, v niektorých z nich je pomlčka považovaná za časť slova. Hoci takéto zaobchádzanie so vstupným textom je veľmi jednoduché, pracuje rýchlo a pre dané potreby je dostačujúce. Je vhodné a potrebné podotknúť, že architektúra programu umožňuje použiť iný tokenizer pre iný jazyk, ako je napríklad thajský, ktorý si vyžaduje použitie lexikónu pre rozdelenie textu na jednotlivé slová.

2.3. Morfologické analyzátory

Na ďalšie spracovanie textu, už rozdeleného na jednotlivé slová, slúži tzv. morfologický analyzátor. Ten priradí slovám slovné druhy a im prislúchajúce gramatické kategórie, ktoré reprezentuje špecifickými morfologickými značkami.

Samozrejme, že jednotlivé značky sú prideľované na základe daností konkrétneho jazyka. Niekedy analyzátor nedokáže rozpoznať dané slovo. Táto situácia sa vyskytuje najmä ak ide o slovo cudzieho pôvodu, prípadne slangový či argotový výraz. Takémuto slovu nemožno priradiť zodpovedajúcu značku. Vtedy môže analyzátor slovu priradiť prázdnu značku, ktorá len informuje ostatné nástroje spracovania textu, že dané slovo nemožno konkrétne určiť. Iným druhom prístupu je snaha analyzátora o pridelenie značky nerozpoznanému slovu na základe jeho okolia alebo jeho vlastností. Najefektívnejšou cestou je kombinácia oboch možností. LanguageTool v prípade neidentifikovaného slova používa prázdnu značku.

POS Značkovač (Pos Tagger)

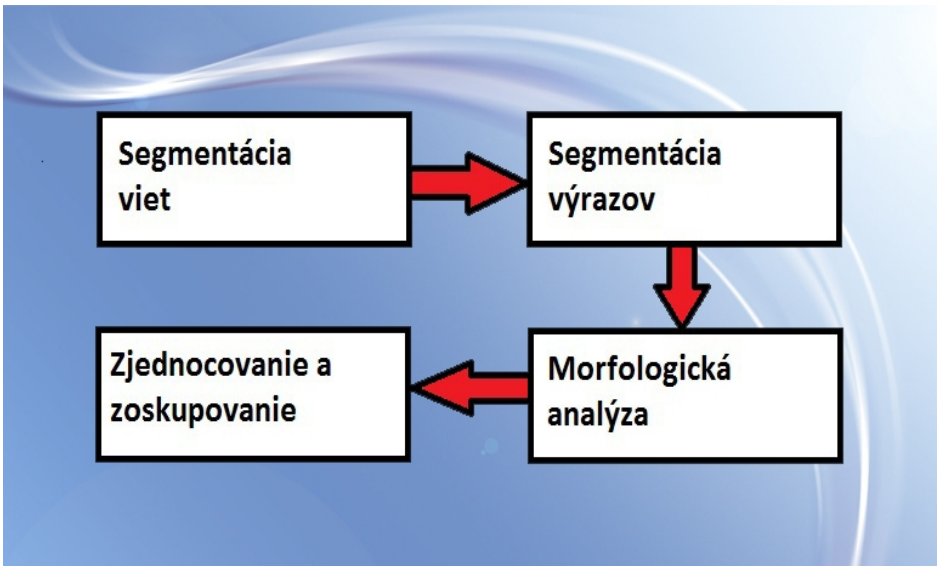
Part-of-speech značkovač je modul, ktorý postupne prechádza jednotlivé časti vety, ktoré sú oddelené interpunkciou alebo medzerami, a na základe zhody so slovníkovým výrazom (POS tagset slovník), im pridelí konkrétne morfologické značky.

Zdroje pre part-of-speech (POS tagset) slovníky pochádzajú z rôznych open-source projektov. Väčšina z nich je nejakým spôsobom odvodená z voľne dostupných výsledkov jazykovedného výskumu, čo je aj prípad slovenskej verzie. Dáta pre slovenský tagset slovník, použitý v LT, pochádzajú z Jazykovedného ústavu Ľudovíta Štúra Slovenskej akadémie vied. Ručne vytvorené slovníky programu na kontrolu gramatiky *ispell* zvyčajne vyjadrujúce morfologické vlastnosti slov, sú tiež relatívne ľahko prispôsobené pre potreby POS tagset slovníkov (základný tvar – lemma, je už definovaný v ispell slovníku).
Značky pridelené taggerom závisia od použitého lexikónu. Inými slovami, neexistuje žiadny všeobecný POS tagset slovník, ktorý by mohol byť zdieľaný viacerými jazykmi. Spôsob, akým sú prideľované konkrétne značky v slovenskom tagset slovníku, nájdeme na adrese http://korpus.juls.savba.sk/files/tagset-www.pdf.
Týmto riešením sa autori vyhli nutnosti konverzie jednotlivých jazykových systémov do spoločnej tagsetu. Popri POS značkách, ktoré sú založené a prideľované na základe jazykovej príslušnosti, existuje ešte niekoľko špeciálnych štrukturálnych značiek. Tie sú používané automaticky pre všetky jazyky. Slúžia na označenie začiatku (SEND_START) a konca vety (SEND_END), začiatku (PARA_START) a konca odseku (PARA_END) a tiež na označenie neznámeho slova (UNKNOWN). Napríklad, slovu „chlap“ priradí tagger použitím slovenského tagset slovníka nasledujúce značky: SSms1.

Disambiguator

Pre jazyky s bohatou morfológiou ako je slovenčina, je priradenie správnych slovných druhov zvyčajne jednoduchšie ako napríklad pre angličtinu, kde rovnaké slová môžu vystupovať v úlohe slovies, podstatných mien a prídavných mien. V slovenčine je takáto nejasnosť skôr zriedkavosťou. Túto nedokonalosť rieši zjednocovacie rozhranie, ktorým sa budeme zaoberať neskôr.

Disambiguator alebo zjednocovač dostáva na vstup zoznam slov interpretovaných morfologickými značkami. Jeho úlohou je odstránenie nesprávnych alebo zbytočných morfologických značiek. V slovenskom jazyku je najčastejšie potrebný pri zhode tvaru viacerých slovných druhov. Príkladom je slovo „vrchný“, ktoré môže byť interpretované ako prídavné, ale aj podstatné meno. Výsledkom jeho činnosti je kompletne pripravená zanalyzovaná veta, na ktorú je možno uplatniť konkrétne pravidlá.



Obrázok 1.1: Schéma architektúry LanguageTool

2.4. Prístupy na kontrolu gramatiky

V súčasnosti existujú tri spôsoby ako vykonať kontrolu gramatiky:

2.4.1. Kontrola na základe analýzy syntaxe

„Analyzátor sa v tomto prípade snaží k vete pristupovať z jazykovedného hľadiska a pokúša sa vytvoriť syntaktický strom. Ak sa to úspešne podarí, označí vetu za gramaticky korektnú. V prípade, že sa strom z nejakého dôvodu nepodarilo zostaviť, vyhlási vetu za chybnú. Výhodou tohto postupu je skutočnosť, že potenciálne môže zachytiť gramatické chyby, ktoré nie sú špecifické určitou postupnosťou slov vo vete. Na druhej strane vytvorenie úplného formálneho popisu určitého jazyka je veľmi náročná úloha.“ (Ličko, 2007, str. 10)

2.4.2. Kontrola založená na štatistike a pravdepodobnosti.

„Táto metóda priamo súvisí s použitím štatistických metód v morfologickej analýze. Vychádza z toho, že postupnosti určitých morfologických značiek sú veľmi časté. Naopak iné sú tak zriedkavé, a tak takéto postupnosti slov možno označiť ako chybu. Jej úspešnosť je zrejmá napríklad v angličtine a iných jazykoch, kde sa možno spoľahnúť na pevnú štruktúru vety a ustálené slovné spojenia.“ (Ličko, 2007, str. 10)

Štatistická korektúra môže mať pozitívny aj negatívny druh prístupu. Pozitívny prístup hľadá vzory časti reči alebo slov, ktoré boli často zaznamenané v chybovom slovníku.

Negatívne pristupujúci spôsob hľadá reťazce, ktoré spolu nemôžu navzájom vytvárať vzory slovných spojení alebo štatistická šanca na ich vytvorenie je veľmi nízka. Jazykové modely používané na dané účely musia byť komplexné a relatívne bez chýb.

Možnosť reálne použiteľnej kontroly gramatiky založenej na pozitívnom štatistickom prístupe si vyžaduje vybudovanie veľkého a reprezentatívneho súboru chybových textov (korpus) pre daný jazyk. Pre väčšinu jazykov boli vydané len menšie vzorky v rôznych obdobiach, nie však samostatný korpus. Je samozrejme vhodné, aby takýto chybový korpus vznikol. Je možné, že niektoré, na webe založené metódy samovzdelávania, by mohli byť na tento účel použité.

„V poslednej dobe bola vyvinutá nová metóda pre spracovanie voľne dostupných zdrojov, ako je napríklad revidovanie histórie Wikipédie a výpisy chýb z nej. Experimenty však čoskoro preukázali, že aj veľké revízie histórie obsahujú pomerne obmedzené informácie o chybách. Dôvodom je, že ani kompletný chybový korpus, ktorý vznikol z používania internetu nie je dostatočne komplexný na to, aby mohol v plnom rozsahu zastupovať súbor najčastejších chýb v jazyku. Napríklad, celá revízia poľskej Wikipédie v rozsahu niekoľkých gigabajtov, mala za výsledok len niekoľko sto dostatočne zdokumentovaných chýb, zväčša preklepov.“ (Milkowski, 2009, str. 2)

Inou možnosťou na vytvorenie programu na kontrolu gramatiky je negatívny štatistický prístup. Avšak na tento prístup je potrebný text bez chýb. Vzhľadom na obmedzenia, ktoré vyplývajú z autorských práv, národné jazykové korpusy (vyvíjané štátnymi inštitúciami predovšetkým s vedeckým účelom) sú len zriedka dostupné vo forme bežného textu, čo prakticky znamená, že ich nie je možné opätovne využiť na potreby strojového učenia. Z tohto dôvodu je potrebné vybudovať presný korpus z nechránených súborov. Navyše národné korpusy väčšinou nie sú presným odrazom súčasne používaného jazyka. Okrem toho overenie, či texty v národných korpusoch sú korektné, vyžaduje manuálnu kontrolu (tá je však nákladná) alebo existujúci rozsiahly program na kontrolu gramatiky (čím sa nám vytvoril začarovaný kruh). Obecne by táto metóda mohla byť vhodná pre jazyky, kde veľké sú k dispozícii veľké, korektné a voľne dostupné údaje.

„Jedným z týchto jazykov je angličtina, a to vďaka elektronickej knižnici ako je Projekt Gutenberg, kde tisíce dobrovoľníkov skutočne denne kontrolujú a „čistia“ dáta. Dáta pre ostatné jazyky však ešte daný projekt nezahŕňa. Táto metóda si teda vyžaduje príliš veľa práce, a to len na prípravu zdrojov.“(Milkowski, 2009, str. 3)

Posledným variantom využívajúcim štatistické metódy je ten, ktorý opakovane využíva na kontrolu frekvencie výskytu výrazov internet pomocou vyhľadávača (napr.: Google). Táto metóda nie je vhodná na každodennú kontrolu bežne používaných jazykových výrazov. Beží totiž veľmi pomaly môže ignorovať niektoré časté chyby, ktoré dokážu na svojou frekvenciou výskytu na internete prevýšiť niektoré výrazy, najmä ak ide o ojedinelé slová. Daná metóda je vhodná prevažne na skúmanie možných slovných spojení.

2.4.3. Kontrola založená na použití pravidiel.

„V tomto prípade sa slovám vo vete pridelia čo najpresnejšie morfologické značky. Potom sú na vety aplikované pravidlá. Na rozdiel od predchádzajúceho spôsobu, pravidlá sú vytvárané ručne a možno v nich vyjadriť skutočné gramatické pravidlá pre daný jazyk.“ (Ličko, 2007, str. 11)

Aj gramatický korektor založený na použití pravidiel má pozitívny a negatívny prístup. Pri negatívnom prístupe je potrebné zaviesť formálnu gramatiku vyjadrenú v pravidlách alebo vzoroch. Doteraz množstvu jazykov chýba kompletná formálna gramatika. Pozitívne pristupujúce metódy sú založené na intuitívnom poňatí gramatiky a štylistických chýb. V praxi používa ručne vybrané pravidlá na základe priemerného jazyka užívateľa, jeho intuície, jazykovej teórie či slovníkovej analýzy. Vďaka tomu môže pozitívne pristupujúci korektor gramatiky účinne poukazovať na najzávažnejšie a najčastejšie na vyskytujúce chyby. LanguageTool využíva práve túto metódu.

Jedným z dôvodov, prečo sú štatistické prístupy pre open-source projekty ako LT menej vhodné je, že pre neprofesionálov je udržiavanie štatisticky postavených zdrojov a zbieranie korektných slovníkových dát ťažšie, ako písanie deklaratívnych pravidiel. Rozhodnutie o nezaradení štatistických častí bolo jedným z hlavných pri vytváraní dizajnu systému, v zásade by však mohli byť štatistické nástroje rovnako účinné alebo dokonca lepšie ako systém založený na pravidlách. Keďže v súčasnosti chýba štruktúra, ktorá by podporila rozvoj štatistických metód, autori sa rozhodli nezaradiť ich. Toto by sa však mohlo v budúcnosti zmeniť, v prípade, ak bude k dispozícii dostatočne jednoduchý nástroj na použitie neprofesionálmi. V architektúre LanguageTool nie je nič, čo by zabránilo pridaniu štatistických zdrojov alebo formálnej gramatiky, ktoré by mohli byť jednoducho používané v schéme iných pravidiel.

2.5. Použitie v praxi

Program LanguageTool je voľne k dispozícii na adrese: http://www.languagetool.org/download/LanguageTool-1.0.0.oxt. V prípade, ak už máme nainštalovaný kancelársky balík OpenOffice a používame operačný systém MS Windows, stačí kliknúť na ikonu a program sa sám nainštaluje ako zásuvný modul kancelárskeho balíka. V prípade, ak používame operačný systém Unix, v niektorej jeho distribúcii a jednoduché kliknutie na inštaláciu nepostačuje, je potrebné spustiť program OpenOffice a zvoliť menu `Nástroje/Správa rozšírení`. Následne kliknime na tlačidlo Pridať a zvolíme daný .oxt súbor. Po inštalácii je potrebné OpenOffice reštartovať. LanguageTool je však možné spustiť aj ako samostatnú aplikáciu s vlastným grafickým rozhraním, a to buď kliknutím na súbor LanguageToolGUI.jar (ten získame rozbalením .oxt) alebo cez príkaz: `java -jar LanguageToolGUI.jar`.

2.5.1 Samostatná aplikácia

Spomínaná verzia LanguageTool ako samostatnej aplikácie poskytuje používateľovi možnosť korekcie gramatiky bez potreby inštalácie akéhokoľvek ďalšieho softvéru. Výhodou samostatnej aplikácie je taktiež intuitívne ovládanie a kompaktnosť a v neposlednom rade aj neprítomnosť obmedzení týkajúcich sa rozhrania kancelárskeho balíka. Hlavnou nevýhodou je však strata príslušnosti k známemu kancelárskemu balíku a s tým spojená nižšia možnosť jeho distribúcie a uplatnenia. Samotný LT využíva dva druhy pravidiel : XML a Java. Z hľadiska vývoja XML pravidiel je nespornou výhodou samostatnej verzie aj to, že na začlenenie nových pravidiel stačí prepísať externý XML súbor, nachádzajúci sa v priečinku `rules/xx` kde „xx“ označuje skratku konkrétneho jazyka, v prípade slovenčiny je to teda súbor `rules/sk/grammar.xml`. Takto jednoducho prepísaný súbor je už aplikáciou priamo používaný a nové pravidlá sú okamžite k dispozícii. Pokiaľ ide o Java pravidlá, na pridanie novej verzie je nutná úprava príslušného java súboru alebo vytvorenie nového súboru (ak ide o nové unikátne pravidlo) a následná kompilácia. Podrobným popisom programu z pohľadu vývojára sa budem zaoberať neskôr v kapitole Zásady tvorby.

2.5.2. Súčasť balíka OpenOffice

OpenOffice.org je voľne dostupný a šíriteľný kancelársky balík, ktorý je preložený do viac ako 30-tich jazykov a dostupný na všetkých hlavných počítačových platformách (Microsoft Windows, Mac OS X, GNU/Linux, Solaris). Dnes ho používajú desiatky miliónov používateľov na celom svete. OpenOffice je kompatibilný s mnohými iným kancelárskymi balíkmi. Medzi ne patri aj najrozšírenejší balík od firmy Microsoft, Microsoft Office a tak je v ňom možné pracovať aj so súbormi, ktoré boli vytvorené spomínaným programom. Dokumenty vytvorené v rámci balíka OpenOffice sa dajú uložiť v OpenDocument formáte, ktorý je novým medzinárodným ISO štandardom pre kancelárske balíky. Inštalácia jednotlivých častí balíka je jednoduchá a intuitívna, prebieha prebieha podľa zvyklostí v danom operačnom systéme. Keďže je vyvíjaný ako Open Source program, je ho možné používať a voľne šíriť zadarmo. Od verzie 2.0 obsahuje balík 6 súčastí a to : textový editor – Writer, tabuľkový procesor – Calc, kresliaci nástroj – Draw, program na tvorbu prezentácii – Impress, program na prácu s databázou – Base a program na spracovanie matematických vzorcov a operácii – Math.

Vďaka možnosti použiť LanguageTool ako zásuvného modul OpenOffice sa predovšetkým zvýšila šanca na jeho využitie, ktorá bude pravdepodobne rásť vzhľadom na stúpajúcu popularitu Open Source kancelárskeho balíka. Používateľ LanguageTool tak získava možnosť využiť prostriedky OpenOffice a prvotný používateľ OpenOffice zasa ďalšiu možnosť skvalitnenia služieb balíka. Obmedzujúcim z hľadiska LanguageTool je, že rozhranie kancelárskeho balíka nemusí implementovať všetky možnosti LT. V prípade, keď používateľ chce striktne využiť len služby gramatického korektora, nie je spojenie s kancelárskym balíkom vhodné, pretože núti používateľa k inštalácii iného, v tomto prípade nie nevyhnutného softvéru, čím môže dôjsť k nežiaducemu ovplyvneniu operačného systému, či už využitým miesta na disku alebo zvýšením prevádzkových nárokov. Z hľadiska vývoja a testovania nových pravidiel nie je pre potenciálneho vývojára spojenie s kancelárskym balíkom nevyhnutnosťou, existujú však aj objektívne príčiny prečo je daná forma využitia LaguageTool vhodná. Bližšie v kapitole Zásady tvorby.

3. Vývoj slovenskej verzie

3.1. Východiská

Slovenčina ako jeden so slovanských jazykov predstavuje jednu z veľkých výziev pre programy na kontrolu gramatiky. Väčšina korektorov bola totiž vyvinutá pre anglický jazyk. Hlavnou črtou slovenského jazyka je predovšetkým bohatá morfológia. Inou špecifickou črtou slovenčiny je aj pomerne voľné poradie slov vo vetách a taktiež nedostatok pomôcok, ktoré by mohli bližšie špecifikovať jednotlivé slovné druhy. V dnešnej podobe sa tvar a vlastnosti ohybných slovných druhov určuje len pomocnou morfologických informácii ako sú číslo, rod či pád a napríklad pri podstatných a prídavných menách absentuje príslušnosť k určitému vzoru.

Vzhľadom na spomenuté znaky, softvér na spracovanie prirodzeného slovenského jazyka, ak má zvládať aj označovanie fráz, musí obsahovať aj značkovač alebo slovník so základnými formami slov, ktoré dokážu rozpoznať všetky skloňované formy a časti fráz. Tieto však nesmú byť definované ako nesúvisiace zložky, ktoré sú pomerne vhodné a úspešné pri gramatickej korektúre anglického jazyka. Pre správnu analýzu je výhodou disponovať slovníkom obsahujúcim gramatické kategórie a možné vzájomné väzby slov.

„Najdôležitejšie zmeny v LanguageTool boli vykonané z dôvodu špecifických požiadaviek pri vývoji pravidiel pre poľský jazyk. Jedinou výnimkou je francúzština, ktorá vyžadovala pridanie zjednocovacieho rozhrania. Disambiguation (zjednocovač) bol zavedený s cieľom umožniť usmerňovanie francúzskych pravidiel, ktoré vyvíjala Agnes Souque na základe súboru pripravených Myriam Lechelt pre open-source program Gramadoir. Zavedenie ďalších jazykov už nevyžaduje veľké zmeny v kóde aj keď jednotliví správcovia pravidiel navrhli veľa možných úprav. Užitočné návrhy dostali autori hlavne od autora holandských pravidiel Ruud Baarsa.“ (Milkowski, 2009, str. 4)

Ako bolo spomenuté v predchádzajúcej časti, hlboké analýzy a úplné lingvistické analýzy nie sú potrebné pre úspešné korektúry. Oveľa jednoduchšie heuristické pravidlá na základe častých a stabilných foriem jazykových chýb pracujú veľmi dobre. Slovenčinu a iné slovanské jazyky možno považovať ako osobitne komplikované prípady, ďalšie jazyky by sa mali do existujúcej formy programu jednoducho zapracovať. Štruktúra programu zatiaľ nebola použitá pre neeurópske jazyky, takže je predčasné hovoriť, či LanguageTool môže

zahrnúť akýkoľvek dnes používaný jazyk, avšak autori dúfajú, že pre absolútnu väčšinu jazykov to možné bude. (Milkowski, 2009)

3.2. Tvorba pravidiel

Dnes LanguageTool využíva na tvorbu pravidiel dva jazyky, a to XML a Java. Java pravidlá sú definované v samostatných súboroch s koncovkou .java. Ide vlastne o jednotlivé triedy, ktoré používa hlavná časť programu. Niektoré Java pravidlá sú spoločné pre všetky jazyky, keďže poukazujú na všeobecné štylistické chyby, ako je napríklad dvojitá medzera či malé písmeno na začiatku vety. Drvivá väčšina pravidiel je však špecifikovaná ako vzor bežného jazyka v XML formáte. Na jednej strane, pravidlá vyvíjané v jazyku Java poskytujú väčšiu flexibilitu, vyžadujú si však hlbšie znalosti v oblasti programovania. Na druhej strane, deklaratívne XML pravidlá, vyvíjané najmä v prostredí špecializovaných XML editorov, poskytujú neporovnateľne lepšie možnosti na používateľské úpravy. Jedným z cieľov projektu bolo aj rozšíriť komunitu pracujúcu na pravidlách a tak sa autori projektu rozhodli pre obohatenie funkčnosti najmä v rozmedzí deklaratívnych pravidiel. Toto konštrukčné riešenie sa ukázalo ako veľmi vhodné: niektoré jazyky podporujúce LanguageTool disponujú stovkami pravidiel napísaných v XML a nemá ani jedno Java pravidlo (francúzsky a holandský jazyk).

Všetky XML pravidlá sú používané modulmi vytvorenými v Jave, ktoré porovnávajú jednotlivé neprázdne časti vstupného textu s konkrétnymi vzormi. Celý proces prebieha slovo po slove. V prípade potreby sa uplatňujú konkrétne základné formy, regulárne výrazy či konkrétne slovné spojenia. Ak sa chce autor vyhnúť zhode slova a vzoru, môže definovať výnimku, ktorú systém následne ako vyhovujúcu neidentifikuje. Zhoda slova a príslušného vzoru deklarovaného v XML pravidle je v súčasnej dobe založená na naivnom lineárnom prehľadávaní štruktúry prvkov v zozname slov konkrétnej vety. Tvorcovia LanguageTool nechcú zavádzať predčasnú optimalizáciu pred dokončením funkčnosti XML pravidiel. Zložitejšie prehľadávacie postupy sú obvykle efektívnejšie pri dlhších vstupných reťazcoch. V prirodzených jazykoch sa však vety s dĺžkou nad 30 slov objavujú zriedka. Porovnávanie textu a vzorov dnes používaným spôsobom je zdá sa uspokojivé, pokiaľ ide o čas, a to aj v prípade práce s tisíckami pravidiel (ako je to pre francúzštinu). Schopnosť preskočiť určité vstupné časti textu, ktorou sa budeme neskôr bližšie zaoberať, znemožňuje niektoré jednoduché optimalizácie naivného vyhľadávacieho

algoritmu, ako je napríklad kontrola prvého a posledného elementu. V nasledujúcich podkapitolách sa pokúsime priblížiť problematiku vývoja pravidiel z pohľadu autora tak, aby prípadne mohla táto časť práce slúžiť ako možný návod pre potenciálneho pokračovateľa či spolupracovníka v tomto komunitnom projekte.

3.2.1. XML v skratke

XML znamená eXtensible Markup Language, v preklade rozšíriteľný značkovací jazyk. Bol vyvinutý a štandardizovaný konzorciom W3C ako pokračovanie jazyka SGML a HTML. Umožňuje jednoduché vytváranie konkrétnych značkovacích jazykov na rôzne účely a široké spektrum rôznych typov údajov. Jazyk je určený predovšetkým na výmenu údajov medzi aplikáciami a na publikovanie dokumentov. (XML slovník pojmov, 19.5.2010)

Základným znakom XML dokumentu je jeho stromová štruktúra s jedným koreňovým prvkom (root element). Samotný dokument sa skladá niekoľkých vzájomne prepojených prvkov, ktoré sú zapísané pomocou významových značiek – tagov.

Tag je značka, ktorá umožňuje štruktúrovanie dokumentu. Tagy sú uzatvorené do ostých zátvoriek, pričom rozlišujeme začiatočný a koncový tag.

Element je považovaný za základný prvok XML dokumentu. Elementy sú ohraničené tagmi, na rozdiel od HTML sú striktne vyžadované začiatočné aj koncové tagy. Elementy musia byť korektne uzatvorené a musia byť úplne vnorené do iného elementu.

„Atribúty definujú dodatočné vlastnosti elementu a zapisujú sa vždy v počiatočnej značke element. Jeden element môže mať viacero atribútov.

Komentáre môžu byť súčasťou dokumentu, zapisujú sa do zátvoriek < – – a – – >. Používajú sa na vloženie akejkoľvek poznámky. Tiež môžu slúžiť na skrytie časti textu. Pri spracovaní dokumentu sú ignorované.“ (Valentová, 2006, str. 16)

Príklad jednoduchého zápisu v XML:

```
<?xml version="1.0"?>
<!DOCTYPE PARTS SYSTEM "parts.dtd">
<?xml-stylesheet type="text/css" href="xmlpartsstyle.css"?>
<PARTS>
   <TITLE>Computer Parts</TITLE>
   <PART>
      <ITEM>Motherboard</ITEM>
      <MANUFACTURER>ASUS</MANUFACTURER>
      <MODEL>P3B-F</MODEL>
      <COST> 123.00</COST>
   </PART>
</PARTS>
```

XML je v podstate súbor pravidiel tvorby textových formátov, ktoré umožňujú štruktúrovať dáta. XML takto uľahčuje počítaču tvoriť, čítať a zapisovať dáta a zaistiť jednoznačnosť štruktúry dát. Za jeho hlavné výhody možno považovať rozšíriteľnosť, nezávislosť na platforme a podporu lokalizácie. Taktiež plne vyhovuje štandardu Unicode. Textový formát, v ktorom XML zapisuje štruktúrované dáta na disk, uľahčuje vývojárom ladenie programu a umožňuje ľudom nahliadnuť na dáta v textovom editore. Tieto výhody prinášajú so sebou zvýšené nároky na veľkosť súboru. Kompresné programy ako zip alebo gzip však dokážu túto nevýhodu zmierniť.

XML dovoľuje definovať nový formát kombináciou a opätovným použitím iných formátov. Pri ich kombinovaní si však musíme dávať pozor na možnosť definovania rovnakých názvov elementov alebo atribútov. Voľbou XML za základ projektu získa autor prístup k obsiahlej skupine nástrojov a odborníkov so skúsenosťami v tejto technológii. Voľba XML je niečo podobné ako keď si zvolíme SQL pre databázu: musíme síce vytvoriť vlastnú databázu a programy na jej obsluhu, ale existuje mnoho nástrojov a ľudí, ktorí nám môžu pomôcť. Keďže XML je k dispozícii bez obmedzujúcich licenčných podmienok, môžeme vytvoriť vlastný software pracujúci x XML bez toho, aby sme niečo niekomu platili. Rastúca podpora a veľký počet vývojárov mám navyše dáva istotu, že sa nemusíme viazať k jedinému výrobcovi softvéru. Táto a ale aj predchádzajúce spomenuté vlastnosti jazyka XML viedla autorov LanguageTool práve k použitiu XML na tvorbu väčšiny pravidiel, s ktorými pracuje architektúra samotného projektu naprogramovaného v jazyku Java.

3.2.2. Zásady tvorby pravidiel

Základný recept na tvorbu pravidiel načrtol už samotný autor LanguageTool Daniel Naber takto:

> „1. Nájdi a konkretizuj chybu, ktorou ešte nie je ošetrená v gramatickom korektori.
> 2. Použi chybné slová alebo slová s celého kontextu na tvorbu pravidla
> 3. Ak je to možné zovšeobecni pravidlo. Použi POS tagové značky slova namiesto konkrétnych slov.
> 4. Skontroluj či sa vzor pravidla môže vyskytovať aj v správnej vete. V prípade veľkého množstvo falošným poplachov zruš krok 3 a skús to znova.“ (Naber, 2003, str. 32)

Samotný vývoj pravidiel pre slovenskú verziu je už len otázkou konkretizácie jazykového problému, ktorý autor plánuje ošetriť, a jeho implementácie v XML alebo Jave. Všetky potrebné slovníky či jednotlivé Java triedy pracujúce so súbormi slovenských pravidiel sú už totiž pripravené.

3.2.3. Súbor grammar.xml

Potenciálny vývojár tak môže siahnuť po súbore `grammar.xml`. V ňom sa už nachádzajú pravidlá vytvorené inými autormi. Samotný súbor sa nachádza v priečinku `rules/sk/`. Tento súbor môže používateľ upravovať, prípadne si vytvoriť vlastný `grammar.xml`, a tak môže pracovať s pomerne malým súborom pravidiel, bez ohľadu na počet už preddefinovaných pravidiel. Taktiež môže nový autor pravidiel porovnať účinnosť rôznych prístupov k jednému jazykovému problému. Poslednou zreteľnou výhodou použitia vlastného súboru s vlastnými pravidlami je to, že samotná kontrola gramatiky prebehne pri menšom súbore rýchlejšie, hlavne v prípade, ak pôvodný už obsahuje niekoľko stoviek pravidiel.

Základnú štruktúru XML súboru sa pokúsime priblížiť sa konkrétnom prípade, pričom nasledujúci kód je kompletný súbor grammar.xml s jedným pravidlom.

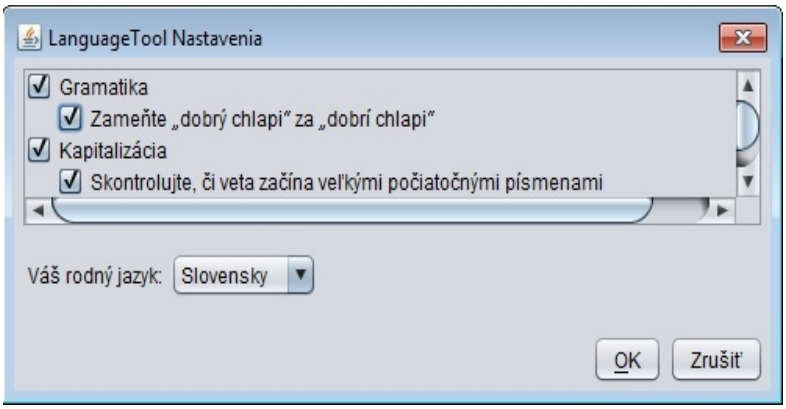
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="../print.xsl" ?>
<?xml-stylesheet type="text/css" href="../rules.css"
title="Easy editing stylesheet" ?>
<rules lang="sk" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="../rules.xsd">
  <category name="Typografia">
    <rulegroup id="POMLCKA_SPOJOVNIK_1" name="Zameňte spojovník pomlčkou
            1">
      <rule>
        <pattern>
          <token>-</token>
        </pattern>
        <message>Podľa pravidiel slovenského pravopisu spojovník nemá byť oddelený
                 medzerou.
                 Použite namiesto neho pomlčku (<suggestion>–</suggestion>), alebo
                 odstráňte medzery.
        </message>
        <short>Typografická chyba</short>
        <example correction="–" type="incorrect">Toto je test <marker>-</marker> alebo aj
           nie.  </example>
        <example type="correct">Toto je test – alebo aj nie.</example>
      </rule>
    </rulegroup>
  </category>
</rules>
```

Prvé štyri riadky sú hlavička XML dokumentu a upresňujú kódovanie a spôsob zobrazenia dokumentu internetovým prehliadačom. Hlavným elementom celého súboru je element `rules`, pričom v jeho atribútoch je konkretizovaná lokalizácia jazyka, v našom prípade

`lang="sk"`. Ďalšie atribúty elementu nie sú pre vývojára príliš podstatné, zhodujú sa vo všetkých podporovaných jazykoch a tak nie je vhodné experimentovať s nimi.

Prvým elementom, pri ktorom sa vývojár rozhoduje medzi viacerými možnosťami je `category`. V dnešnej podobe LangugeTool je pre slovenčinu vytvorených niekoľko kategórii pravidiel: gramatika, kapitalizacia, typografia, rôzne, predložky a číslovky. V našom prípade dané pravidlo patrí do kategórie „Typografia“. Samotný element `category` slúži na sprehľadnenie pravidiel pri zobrazení XML dokumentu vo webovom prehliadači ale najmä pri konfigurácii pravidiel v samotnom OpenOffice, ktoré je dostupné cez menu `Nástroje/Gramatika – LanguageTool/Konfigurácia`. Nové kategórie možno pridať jednoducho použitím elementu `category` s unikátnym atribútom `name`.

Jazykovým problémom ako takým sa zaoberá element `rulegroup`. Jeho atribút `id` slúži len na identifikáciu a vyžaduje si ho architektúra systému. Atribút `name` má aj praktické použitie, jeho text sa zobrazí tak pri GUI verzii ako aj pri použití v LangageTool v OpenOffice, konkrétne pri konfigurácii pravidiel používateľom, preto by mal čo najlepšie opisovať aký problém pravidlo, či skupina pravidiel riešia.



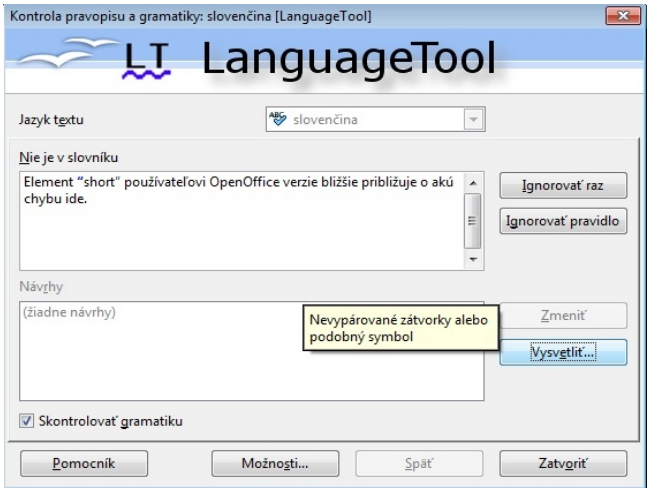
Obrázok 3.1: Konfigurácia pravidiel so zobrazením mien a kategórii pravidiel

Element `rule` ohraničuje jedno konkrétne pravidlo, v jednom rulegroup ich môže byť viacero. V praxi táto situácia nastáva ak je potrebné ošetriť niekoľko rôznych podôb rovnakého jazykového problému. V tomto prípade však bolo pravidlo len jedno.

Element `pattern` umožňuje systému porovnávanie vstupného textu so vzorom, ktorý definuje autor pravidla. Vzor sa môže skladať s viacerých častí, tie konkretizujú elementy

`token`, pričom ich počet určuje z koľkých častí sa vzor skladá. Jeden token prislúcha buď jednému slovu alebo ľubovoľnej interpunkcii. Medzery sú ignorované, avšak prípade, keď autor chce do vzoru zakomponovať medzeru je možné použiť atribút tokenu `spacebefore="yes"`, ktorý určuje či sa pred možnou časťou textu má nachádzať medzera. Element `token` má ešte niekoľko možných atribútov, ktorými sa budem zaoberať neskôr.

Element `message` ohraničuje text, ktorý sa vypíše používateľovi pri nájdení chyby. Ak ide o samostatnú aplikáciu, text sa zobrazí automaticky, v prípade prídavného modulu OpenOffice sa text zobrazí v menu kontroly gramatiky pri voľbe tlačidla „Vysvetliť…“.



Obrázok 3.2: Dialógové okno kontroly gramatiky V OpenOffice so zobrazením textu elementu message

Znaky, prípadne text v elemente `suggestion` symbolizujú korektnú formu textu. `Suggestion` musí byť súčasťou elementu `message`. Použije sa ako návrh na oprav, pričom celý element `message` sa použije pri vysvetlení chyby.

Element `short` používateľovi OpenOffice verzie bližšie približuje o akú chybu ide. Najvhodnejšou verziou tohto elementu je spojenie kategórie a mena chyby, hoci v tomto prípade bola použitá len kategória.

Posledný použitý element `example` sa využíva len pri zobrazení XML súboru v internetovom prehliadači. Popisuje nesprávne prípady riešenia jazykového problému. Správny príklad riešenia je výsostne orientačný a nezobrazuje sa ani v prehliadači.

Teraz sa pokúsime zamerať len na nevyhnutné časti pravidla. Všetky nepovinné elementy a atribúty, ktoré slúžia na k zlepšeniu orientácie konečného používateľa vynecháme.

Absolútne najjednoduchšou formou je nasledujúca:

```
<pattern>
  <token>-</token>
</pattern>
<message>
  <suggestion>–</suggestion>
</message>
```

V tomto prípade je vzorom len spojovník, pričom ho v návrhu správnej gramatickej formy nahrádzame pomlčkou.

O niečo zložitejší postup ilustruje nasledujúci príklad:

```
<pattern >
  <token>.</token>
  <token>.</token>
  <token>.</token>
</pattern>
<message>
   <suggestion>…</suggestion>
</message>
```

V tomto prípade nahrádzame vzor zložený z troch blokov textu jediným. Tri bodky teda nahradíme trojbodkou.

V nasledujúcich pravidlách nebudeme uvádzať celý kód, ale obmedzíme ho len na kľúčové časti, hoci spomenieme aj nepovinné elementy aatribúty. Nasledujúce pravidlo upozorňuje na nesprávne použité úvodzovky po skratke „tzv“. Zaujímavým atribútom tohto pravidla je atribút `regexp`, symbolizuje zápis hľadaného tokenu pomocou regulárnych výrazov. Tie hovoria o viacerým možných hľadaných tvaroch konkrétneho slova, či v tomto prípade interpunkcie. V praxi sa používa najmä symbol „|”, ktorý predstavuje alebo. Viac o regulárnych výrazoch nájdete na adrese http://www.regularnivyrazy.info/regularni-vyrazy-zaklady.html. V prípade, že ide o viaceré rôzne tvary interpunkcie je možné hľadané znaky zapísať v hranatých zátvorkách alebo ich oddeliť znakom „|“. Takto teda môžeme zabezpečiť zhodu napríklad s výrazmi: tzv. „prikladným“ ale aj tzv. "prikladným"

```
<pattern mark_from="2">
   <token>tzv</token>
   <token>.</token>
   <token regexp="yes">[„"]</token>
   <token skip="-1" ></token>
   <token regexp="yes" >["“]</token>
 </pattern>
```

Ešte zaujímavejším atribútom, s ktorým sme sa doteraz nestretli, je atribút tokenu `skip`. Používa sa na preskočenie tých častí vstupného textu, ktoré nevyhovujú deklarovanému vzoru, pričom vzorové slovo sa následne porovná s nasledujúcim slovom vstupného textu.

Týmto spôsobom je možné správne rozpoznať slovné spojenia či časti textu, v ktorých sa nachádza aj nekonkretizované slovo. Hodnota atribútu `skip` je číslo, označujúce koľko slov sa má pri porovnávaní vstupného a vzorového testu preskočiť. V prípade, že je hodnota záporná, znamená to, že vzorový token, nasledujúci po tokene s atribútom `skip` je porovnávaný so všetkými slovami vety, ktoré nasledujú po poslednom úspešne porovnanom slove.

V tomto prípade je pre nás zaujímavý aj atribút vzoru `mark_from="2"`. Ten určuje, od ktorého slova bude program značkovať vstupný text. Vzor môže mať aj atribút `mark_to`, ktorý určuje posledný značkovaný token. Štandardná hodnota 0 prislúcha označeniu všetkých častí vstupného textu, ktorý je porovnávaný so vzorom. Hodnota -1 zasa určí, že posledné slovo alebo interpunkcia sa nebude značkovať. Podobne to platí aj pre iné záporné hodnoty. Značkovanie za základe týchto atribútov je vlastne len označenie chybnej časti vety. V praxi sa iné hodnoty ako 0 používajú v prípadoch, keď hľadáme frázu zloženú s viacerých slov, pričom chybná je len jej určitá časť. Príklad: Hľadáme frázu *tzv. „príklad"*, pričom správna forma má podobu *tzv. príklad*. Atribút `mark_from="2"` zabezpečí, že sa bude nahrádzať len časť vzoru po prvých dvoch tokenoch (*tzv.*).

Príklad mierne komplikovanejšieho pravidla, v ktorom sa skúma viacero slov, sleduje použitie slova „jedny" s pomnožnými podstatnými menami.

```
<pattern  mark_from="0">
 <token>jedni</token>
 <token postag_regexp="yes" postag="S.(i|f|n)(p|o)."></token>
</pattern>
<message>Slovo „jedni" má v spojení s podstatnými menami tvar
 <suggestion>jedny
  <match no="2">
  </match>
 </suggestion>.
```

Pre nás zaujímavým atribútom je atribút tokenu `postag_regexp="yes"`. Ten hovorí o tom, že hľadané slovo v tagset slovníku je zadané pomocou regulárneho výrazu – teda viacerými rôznymi postupnosťami znakov. Hľadaný výraz je v našom prípade zadaný atribútom `postag="S.(i|f|n)(p|o)."`. Ten symbolizuje podstatné meno (prvý znak – S) s ľubovoľnou paradigmou (druhý znak – .) mužského neživotného alebo ženského alebo stredného rodu (tretí znak – i|f|n), množného alebo neurčitého čísla (štvrtý znak – p|o), v ľubovoľnom páde (piaty znak – .). Prislúchajúce znakové sady sa líšia

vzhľadom na slovný druh, ktorý určujú. Podrobný opis prideľovania znakov slovným druhom a ich gramatickým kategóriám nájdeme na adrese http://korpus.juls.savba.sk/files/tagset-www.pdf.

Posledný element z tohto pravidla, ktorým sa budeme zaoberať je `match` a jeho atribúty. Atribút `no="2"` označuje poradie slova, ktoré chceme zahrnúť do správneho návrhu. Tentoraz to bolo druhé slovo. Ďalšie atribúty určujúce príslušné slovo majú rovnaké vlastnosti a úlohy ako v elemente `token`.

Iné atribúty jednotlivých elementov uplatňuje nasledujúce pravidlo. To slúži na správne použitie veľkých písmen v názvoch pamätných dní, konkrétne v prípade Dňa matiek či Dňa zeme.

```
<pattern case_sensitive="yes" mark_from="0">
  <token><exception postag="SENT_START"></exception></token>
  <token regexp="yes">deň|dňa|dňu|dni|dňom|dní|dňoch|dňami
    <exception case_conversion="startupper"></exception></token>
  <token regexp="yes">matiek|zeme|Zeme</token>
</pattern>
<message>Pri pamätných dňoch použite veľké začiatočné písmeno:
<suggestion>
  <match no="1"></match>
  <match case_conversion="startupper" no="2"></match>
  <match no="3"></match>
</suggestion>.
</message>
```

Prvým novým atribútom je `case_sensitive` určujúci dôležitosť veľkosti písmen v deklarovanom vzore. Oveľa zaujímavejšou novinkou je element `exception`. Ten totiž patrí medzi veľmi často používané a slúži na deklarovanie možností, v ktorých nechceme s daným slovom pracovať, hoci spĺňa požadované vlastnosti. Atribút `postag="SENT_START"` určuje, že táto konkrétna výnimka platí pre začiatočné slová vety. V tomto prípade sme použili prvý token na ošetrenie prípadu, keď by veta aj slovné spojenie začínali nesprávnym malým písmenom a spôsobovali by dve rôzne chybové hlášky s rovnakým riešením. Zbytočne by tak nastala nejednoznačná situácia.

Posledným doteraz nespomenutým atribútom je `case_conversion`, s možnými hodnotami `startupper` a `startlower` prislúchajúcim jednotlivým veľkostiam začiatočných písmen. Ten zabezpečí jednoduchým spôsobom správnu veľkosť prvého písmena tokenu.

Jednoduchým príkladom ilustrujúcim ďalšie možné atribúty tokenov je pravidlo upozorňujúce na správne tvar slova xkrát, kde x je číslovka.

```
<pattern  mark_from="0">
   <token postag_regexp="yes" postag="N.*"></token>
   <token spacebefore="yes">krát</token>
</pattern
```

Tvar hodnoty atribútu `postag="N.*"` symbolizuje číslovku s ľubovoľnými vlastnosťami. Zápis `.*` teda nahrádza `....`, keďže číslovky sú charakterizované znakovou sadou s dĺžkou 5. Atribút `spacebefore` označuje medzeru, ktorá sa musí nachádzať pred slovom „krát“. Inou možnosťou na zápis tejto podmienky by bolo pridanie prázdneho tokenu medzi už existujúce.

Nasleduje pravidlo, ktoré upozorňuje na nesprávne použitie opytovacieho zámena „koho“ na mieste, kde je správne použiť privlastňovacie zámeno „čia“.

```
<pattern  mark_from="0">
  <token skip="-1">koho</token>
  <token postag_regexp="yes" postag="S.f.1">
    <exception  scope="previous">,</exception>
  </token>
</pattern>
```

Na tomto príklade z pohľadu vývoja pravidiel zaujímavý atribút výnimky `scope="previous"`. Základným nastaveným atribútu je `current` a poslednou možnou hodnotou je `next`. Použitie `scope` je užitočné v prípade potreby identifikácie rôznych výnimiek, ktoré by mohli viesť k falošným poplachom. V našom prípade sa jeho použitím podarilo odstrániť možné poplachy, v ktorých sa podstatné meno ženského rodu v nominatíve nachádza za čiarkou a teda je zjavné, že zámeno koho sa k nemu nevzťahuje. Iný spôsob ako vylúčiť tieto prípady by bol omnoho komplikovanejší.

Pre ilustráciu nasledujúcich možností poslúžia časti kódu, ktoré nie sú použité v žiadnom slovenskom pravidle. Je však pravdepodobné, že pre tvorbu nových verzii LT sú nápomocné.

```
<patterm case_sensitive="yes">
  <unify feature="case_sensitivity" type="startupper">
    <token/>
    <token>Boca</token>
  </unify>
</patter>
```

Takto deklarovanému vzoru vyhovujú všetky slovné spojenia, kde prvé slovo začína veľkým písmenom. Keďže prvý token je prázdny, zahŕňa všetky slová, či interpunkcie. V prípade, že chceme definovať pravidlo pre slovné spojenia obsahujúce slová s rovnakými začiatočnými písmenami nezávisle na ich skutočnej veľkosti, stačí keď zo zápisu vynecháme `type` a v tomto konkrétnom prípade zmeníme druhý token za prázdny. Trochu komplikovanejšiu verziu symbolizuje nasledujúci kód:

```
<patterm case_sensitive="yes">
  <unify negate="yes" feature="case_sensitivity" type="startupper">
    <token/>
    <token regexp="yes">[bB]oca</token>
  </unify>
```

```
</patter>
```

Tento vzor sa nezhoduje s výrazmi „Vyšná Boca“, ale s výrazmi „vyšná boca“, „vyšná Boca“ a „Vyšná boca“. Atribút `negate` určuje, že postupnosť slov nemôže obsahovať len slová s určitou veľkosťou prvého písmena. V tomto prípade teda vyhovuje akákoľvek séria slov, kde nie sú všetky prvé písmená veľké. Pretože druhý token obsahuje regulárny výraz, podmienkam vyhovuje obe verzie, „boca“ aj „Boca“.

Posledným nástrojom, s ktorým sa oboznámime je zahrnutie preskočenej časti vety do návrhu. V praxi sa používa v prípadoch, kde sa medzi hľadanými slovami môže vyskytovať jeden, či viac výrazov. Tie je potrebné zahrnúť aj do návrhu správneho riešenia, inak by som mohli pri použitý korektora v rámci OpenOffice stratiť. Nasledujúce pravidlo skúma nesprávne použitie slova „mimo“ v spojení s prídavným menom v 4. páde. Kľúčovým atribútom je `include_skipped="all"`, ten zabezpečí spomínanú celistvosť návrhu správneho riešenia.

```
<pattern  mark_from="0">
  <token skip="-1">mimo</token>
  <token postag_regexp="yes" postag="A...4."></token>
</pattern>
<message>Predložku „mimo“ nahraďte výrazom
  <suggestion>okrem
    <match no="2" postag_regexp="yes" postag="A...4." include_skipped="all"/>
  </suggestion>.
</message>
```

3.2.4 Pravidlá v jazyku Java

Možným doplnením odporúčania Daniela Nabera z úvodu predchádzajúcej kapitoly je azda rozšírenie testovania nového pravidla v rôznych situáciách (bod 4) a taktiež rozhodnutie, v akom vývojom prostredí pravidlo implementovať. Do procesu rozhodovania v tejto otázke vstupuje viacero faktorov. Do úvahy je potrebné zobrať programátorské schopnosti tvorcu, náročnejší proces pridania nového pravidla do lokálnej verzie LanguageTool, či už v podobe samostatnej aplikácie alebo súčasti OpenOffice a v neposlednom rade možnosť úprav širokým spektrom potenciálnych používateľov – zväčša programátorských laikov. Z tohto zoznamu objektívnych faktov nám ako jasná voľba pripadá XML a je realitou, že aj samotní autori či správcovia jednotlivých jazykových pravidiel preferujú túto možnosť, pričom priznávajú, že v XML je možné implementovať takmer všetky doterajšie pravidlá. Jedným z dôvodov prečo sa rozhodli aj pre JAVA formu je to, že viaceré pravidlá sa vyskytujú vo všetkých jazykoch, a tak je pohodlnejšie ich zrealizovanie v samostatných .java súboroch ako v každom XML súbore prislúchajúcom konkrétnemu jazyku. Navyše,

tieto základné pravidlá sú skryté pred bežným používateľom. Z praktického hľadiska sa teda, pri zanedbaní objektívnych výhod XML formy, javí ako jediná výhoda JAVA prostredia efektivita z pohľadu práce tvorcu samotných pravidiel. Z mojich praktických skúseností môžem povedať, že ideálnych prípadom na možné java pravidlo je také, ktoré obsahuje čo možno najviac samostatných foriem. Príkladom vhodným na prepis z XML je pravidlo, ktoré kontroluje správne použitie úvodzoviek. Jeho autor Zdenko Podobný na implementáciu musel totiž použiť až 4 rôzne formy (4 samostatné elementy rule v jednom Rulegroup). Z programátorského hľadiska je teda efektívnejšou formou využitie java nástrojov ako viacnásobné opakovanie veľmi podobného XML zápisu.

V prípade, že sa autor rozhodne pre JAVU je proces tvorby pravidla oveľa zložitejší. Autor sa totiž v prvom rade musí dostať ku kompletným zdrojovým kódom aplikácie, nestačí mu teda len bežne dostupná .oxt či .zip verzia, ktorá obsahuje všetky XML pravidlá. Keďže však ide o open-source, k CVS verzii sa možno dostať použitím nasledujúcich príkazov:

cvs -d:pserver:anonymous@languagetool.cvs.sourceforge.net:/cvsroot/languagetool login

cvs -z3 -d:pserver:anonymous@languagetool.cvs.sourceforge.net:/cvsroot/languagetool co -P JLanguageTool

Na ich správne vykonanie je potrebný špecializovaný CVS software. V prostredí Windows odporúčam WinCVS. V operačných systémoch založených na Unix stačí nainštalovať nástroj CVS a príkazy zadať do príkazového riadku.

Po stiahnutí kompletnej verzie má vývojár možnosť zasahovať do všetkých jej častí a jednotlivých nástrojov. Dôležitým z pohľadu tvorby slovenských pravidiel je priestor v adresári

`JLanguageTool\src\java\de\danielnaber\languagetool\rules\sk`.

Tu sa nachádzajú všetky pravidlá vytvorené v JAVE určené výhradne pre slovenský jazyk. Nasledujúci postup je už zrejmý. Je potrebné vytvoriť nový súbor formátu .java. V ňom samotnom je treba vytvoriť niekoľko nevyhnutých metód. V prvom rade je potrebné pridať na začiatok súboru nasledujúci riadok:

```
package de.danielnaber.languagetool.rules.sk;
```

Ten zabezpečí správnu príslušnosť k slovenčine. Okrem neho musí súbor obsahovať aj hlavnú triedu:

```
public class PokusRule extends SlovakRule { }
```

so štyrmi kľúčovými metódami:

```
public PokusRule(final ResourceBundle messages) {
  if (messages != null) {
    super.setCategory(new
    Category(messages.getString("category_misc")));
    }
 setDefaultOff();
}
```

ktorá táto zabezpečí konštrukciu pravidla a príslušnosť ku konkrétnej kategórii,

```
public final String getId() {
  return "SK_POKUS";
}
```

ktorá je nevyhnutná pre aktiváciu pravidla a o ktorej bude ešte zmienka

```
public final String getDescription() {
 return "Názvy obcí, v ktorých je „Ves"";
}
```

ktorá zobrazuje sa pri konfigurácii pravidiel samotným používateľom s atribútom name v XML a napokon

```
public final RuleMatch[] match(final AnalyzedSentence text) {
...
}
```

ktorá určuje vzor pravidla a spôsob detekcie chyby. Slúži taktiež na návrh správneho riešenia. Keďže je pomerne obsiahla, nebudeme jej obsah konkretizovať ale odporúčame preskúmanie všetkých definovaných java pravidiel na ilustráciu možností implementácie

Po úspešnom vytvorení spomenutých metód je potrebné pravidlo pridať medzi ostatné slovenčinou používané pravidlá. Na tento účel je potrebné zmeniť obsah súboru Slovak.java, ktorý sa nachádza v adresári:

`JlanguageTool\src\java\de\danielnaber\languagetool\language.`

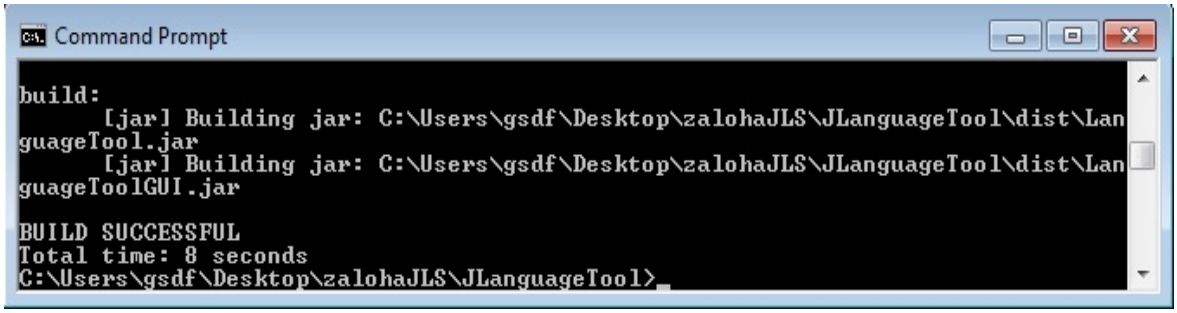
V súbore samotnom stačí pridať identifikátor (SK_POKUS) súboru s novým pravidlom do metódy:

```
public Set<String> getRelevantRuleIDs() {
  Set<String> ids = new HashSet<String>();
  .add("COMMA_PARENTHESIS_WHITESPACE");
  ids.add("DOUBLE_PUNCTUATION");
  ids.add("UNPAIRED_BRACKETS");
  ids.add("UPPERCASE_SENTENCE_START");
  ids.add("WORD_REPEAT_RULE");
  ids.add("WHITESPACE_RULE");
  ids.add("SK_COMPOUNDS");
  ids.add("SK_POKUS");
  return ids;
}
```

Po uložení oboch súborov je pravidlo úspešne deklarované. Na jeho aktiváciu či testovanie sú však potrebné ďalšie kroky.

3.2.5. Kompilácia

Takto pripravené pravidlá je ešte nutné zapracovať do novovytvorenej .oxt verzie. Tá vznikne kompiláciou všetkých zdrojových súborov. Na tento účel poslúži softvérová aplikácia Apache Ant, Ktorý je voľne dostupný na adrese: http://ant.apache.org/bindownload.cgi. Samotná inštalácia nie je práve triviálna, a tak pre používateľom linuxových systémov odporúčam riadiť sa nasledujúcim manuálom: http://ant.apache.org/manual/install.html. V operačných systémoch Windows je to obdobne zložité, je potrebných aj niekoľko úprav systémových nastavení. Na internetovom portáli youtube však existuje veľmi intuitívny tutoriál: http://www.youtube.com/watch?v=XJmndRfb1TU.Po úspešnom rozbehnutí softvéru Ant nasledujú v kompilácii ďalšie kroky. V termináli `command line` sa presunieme do prečinka `JLanguageTool` a novú verziu vytvoríme cez príkaz `ant build`. Úspešná kompilácia pod systémom Windows vyzerá asi takto:

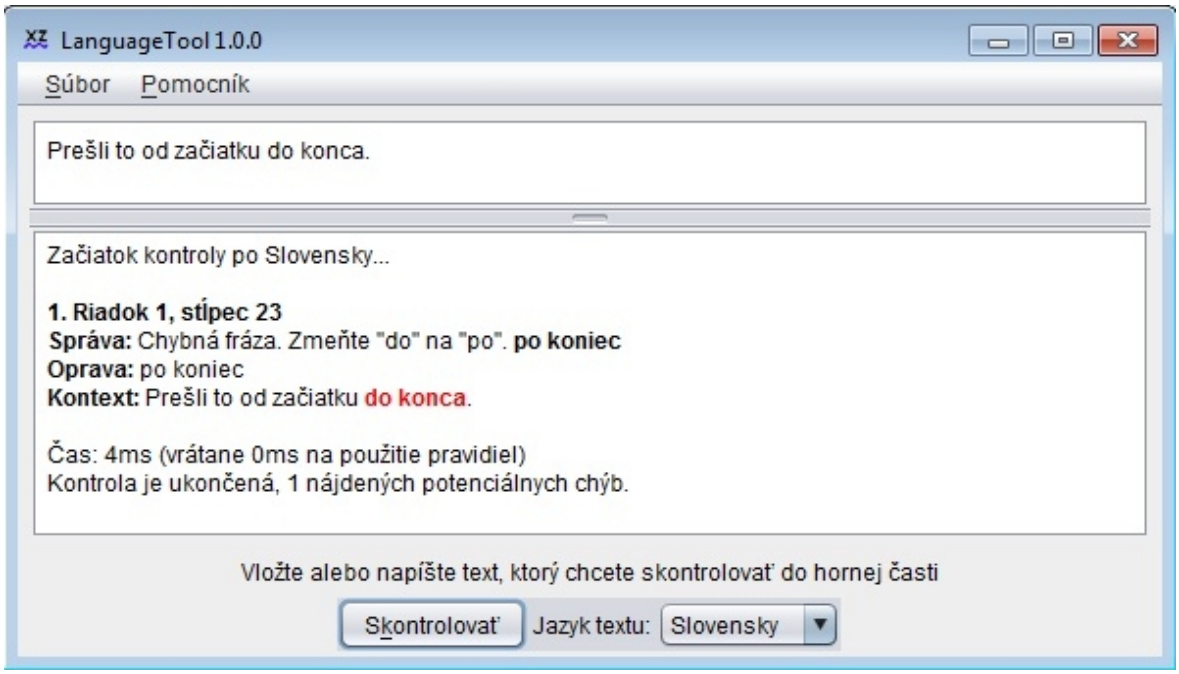


Obrázok 3.3: Kompilácia vo Windows použitím príkazu ant build

Jej výsledkom je niekoľko súborov a priečinkov, ktoré sa objavia v priečinku `JLanguageTool\dist`. Z nich je možné vytvoriť inštalačný oxt balík. Stačí do archívu zabaliť prikázom zip priečinky `images`, `META-INF`, `resource` a `rules` a taktiež všetky samostatné súbory z adresára. Následne zmeniť koncovku súboru na .oxt. Takto pripravený inštalačný balík je náhradou voľne dostupného z internetu. Jeho inštalácia je teda úplne rovnaká, ako tá s ktorou sme sa už zaoberali. Po jej inštalácii môžeme modifikovanú a vylepšenú verziu používať ako súčasť OpenOffice alebo v samostatnom GUI prostredí.

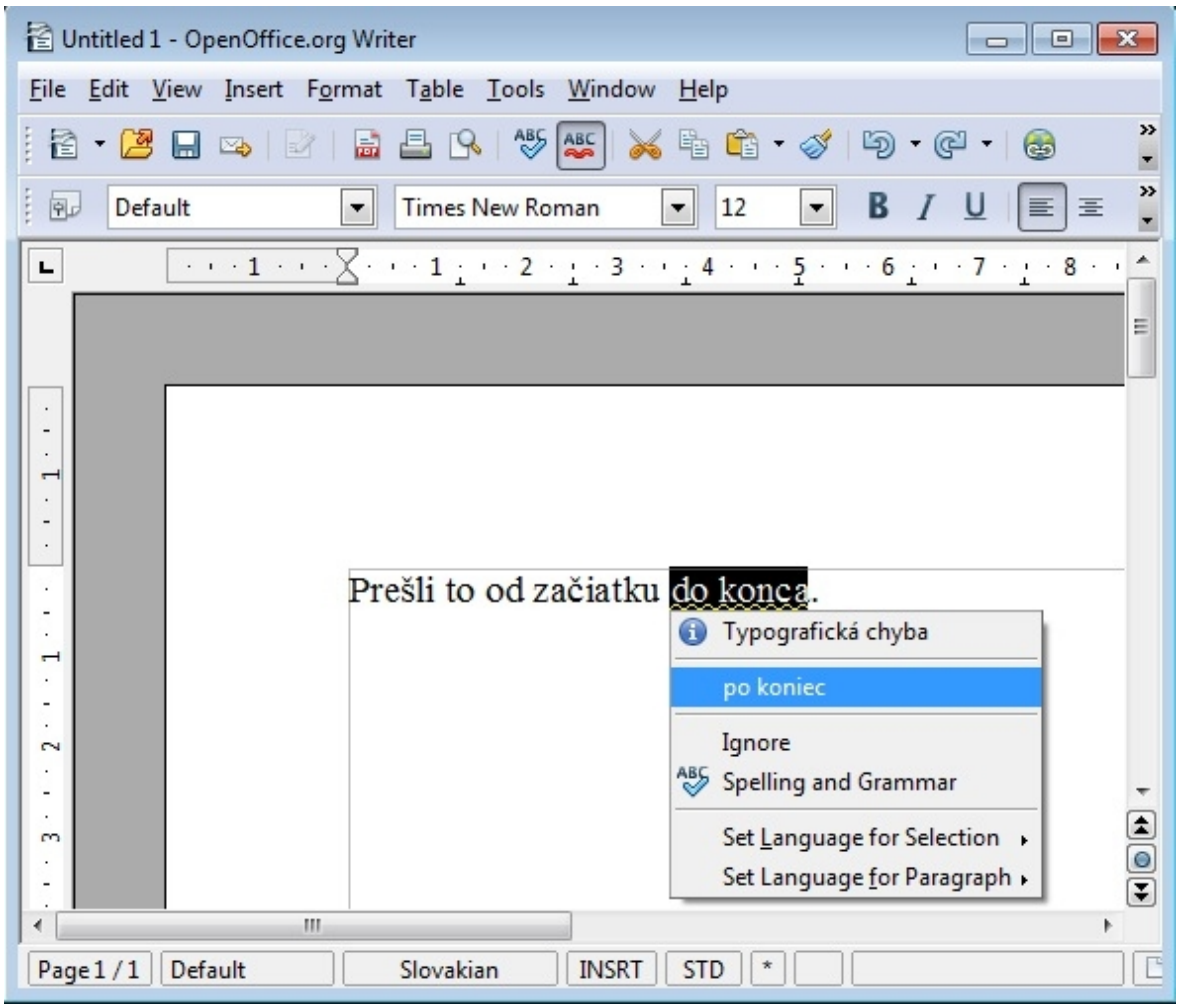
3.2.6. Testovanie

Prvotné testovanie novovytvoreného pravidla spravidla prebieha v jednoduchom grafickom prostredí, ktoré dokáže odhaliť prípadné nepresnosti, či už v návrhu správneho riešenia alebo v označení chybnej časti vety. Obrázok ilustruje príklad tohto jednoduchého testu:



Obrázok 3.4: Testovanie pravidla pomocou LanguageToolGUI.jar

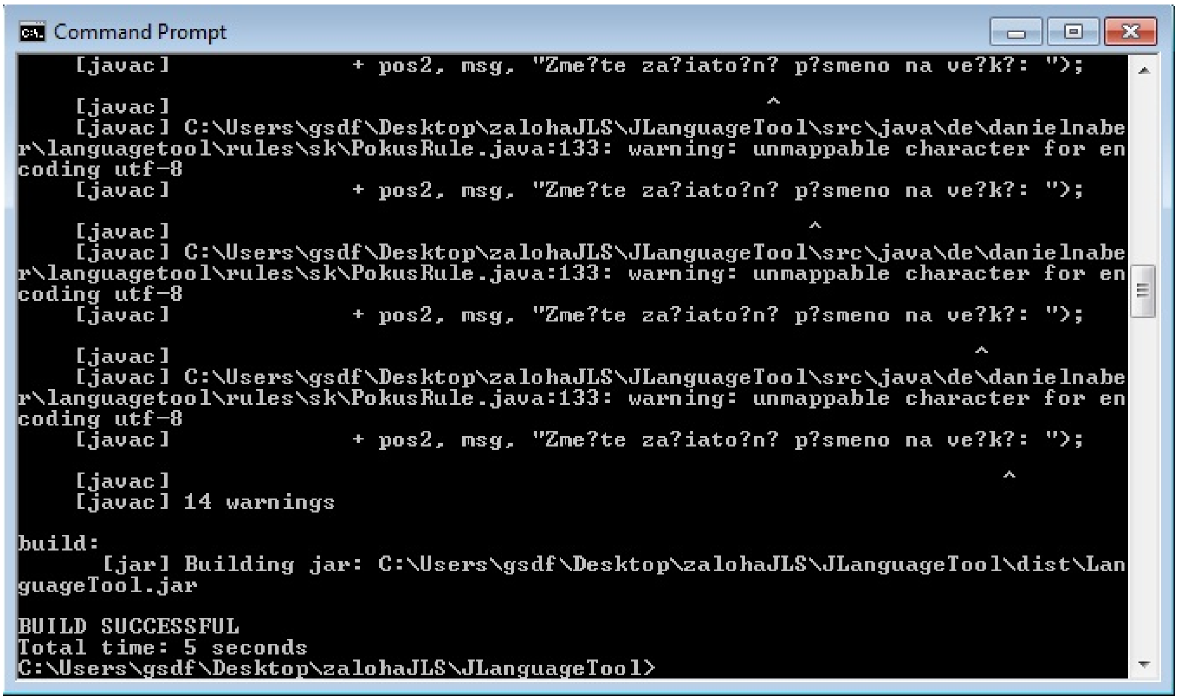
Nespornou výhodou tejto formy testovania pravidiel na rôznych tvaroch pravdepodobne chybnej časti je jej rýchlosť a v prípade tvorby XML pravidiel aj okamžitá dostupnosť bez nutnosti vytvorenia a inštalácie úplne novej .oxt verzie. Tento spôsob však nedokáže preveriť všetky možnosti, ktoré dobre definované pravidlo má poskytovať používateľovi. Tie je potrebné otestovať v prostredí OpenOffice. Najdôležitejšie je otestovať správne nahradenie chybného textu ponúkaným riešením. Niekedy totiž GUI interpretuje na pohľad správny návrh ale pri testovaní v OpenOffice sa objaví chyba. Často pritom ide o medzeru navyše alebo inú chybu vo vzťahu k interpunkcii, ktorú v grafickom prostredí samotného LangugeTool nepostrehneme. Ponuka alternatívneho textu, ktorá sa zobrazí po kliknutím pravým tlačidlom myši a chybnú časť textu je na obrázku.



Obrázok 3.5: Dialógové okno v OpenOffice zobrazené po kliknutí pravým tlačidlom na chybnú časť textu.

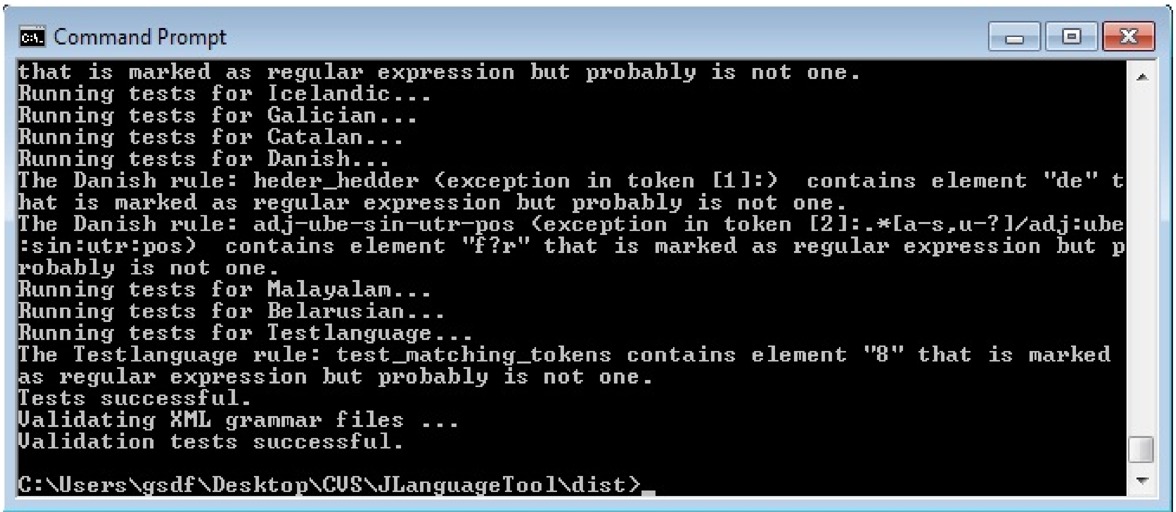
Inou možnosťou na zobrazenie možných chýb je spustenie kontroly cez ponuku `Nástroje – LanguageTool – Kontrola dokumentu`. Táto možnosť taktiež obsahuje možnosť jednoduchého nahradenie pôvodného textu jeho korektnou formou.

Ďalšou možnosťou testovania, tentoraz skôr formálnej štruktúry ako samotnej účinnosti korekcie je aj kompilácia pomocou `ant build`. V nej sa totiž zobrazujú chyby a varovania v jednotlivých pravidlách spolu so stručným popisom. Príkladom je táto screenshot:



Obrázok 3.6: Kompilácia cez ant aj s výpisom warningov

Posledný spôsob na testovanie súborov s pravidlami bol primárne na to aj vyvinutý. Ide o script `testrules.bat` alebo `testrules.sh` v závislosti od operačného systému. Jeho efektivita oproti použitiu `ant build` je vyššia v prípade testovania XML pravidiel. Ponúka však pomerne ťažko pochopiteľné označenie nevyhovujúcich častí pravidiel. Úspešný test by skončiť asi takto:



Obrázok 3.7: Výpis po testovaní pravidiel cez testovací script

Po úspešnom zbehnutí všetkých spomenutých testov je nová verzia pravidiel pripravená na jej začlenenie do existujúceho balíka dostupného na oficiálnej stránke projektu.

3.2.7. Aktivácia v súčasnej verzii LT

Za najlepšiu cestu, ako dostať svoje pravidlá k širokej užívateľskej verejnosti, považujeme kontaktovať správcu slovenskej verzie LanguageTool Zdenka Podobného. Stránka všetkých projektov, ktorým sa v súčasnosti venuje je http://www.sk-spell.sk.cx/. Alternatívou je možnosť zapojiť sa priamo do vývoja LanguageTool a dať o sebe vedieť na internetovom fóre http://sourceforge.net/mailarchive/forum.php?forum_name=languagetool-devel, kde sa stretáva väčšina spolupracovníkov, ktorí riešia možné problémy, či budúci vývoj. Fórum je voľne dostupné na čítanie. V prípade záujmu o aktívnu komunikáciu je nutné sa zaregistrovať na adrese https://lists.sourceforge.net/lists/listinfo/languagetool-devel. Na tomto fóre často radia aj hlavní predstavitelia projektu ako Daniel Naber a Marcin Milkowski, pričom práve druhý spomenutý má pod patronátom vývoj pravidiel v jednotlivých jazykoch. Po úspešnom začlenení sa do tejto komunity je možné pravidlá vyvíjať pomocou CVS verzie a nové formy posielať priamo na server, kde sa nachádzajú všetky doterajšie vývojárske verzie programu. Táto možnosť je pravdepodobne výhodná v prípade vývoja časti samotnej architektúry LanguageTool alebo implementácie väčšieho počtu pravidiel v JAVE. Ak však ide o XML pravidlá, tu považujeme sprístupnenie novej verzie cez Zdenka Podobného za prospešnejšie, najmä v prípade nového spolupracovníka.

3.3. Popis vytvorených pravidiel

V tejto časti sa budeme zaoberať časťou z vytvorených pravidiel. Pre účely jednoduchšieho testovania a tiež prepisu pravidiel do textovej podoby sme vytvorili súbor `konverzia.xsl`. Vďaka nemu sa vo webovom prehliadači zobrazia len určité časti XML kódu s jednoduchými nadpismi. Na jeho použitie stačí v súbore grammar.xml nahradiť druhý riadok nasledujúcim kódom: `<?xml-stylesheet type="text/xsl" href="../konverzia.xsl" ?>`.

Postupne prejdeme niekoľko pravidiel, pričom sa sústredíme na názov pravidla a príklady situácii, v ktorých sú účinné.

Dajte čiarku za citoslovce

Nesprávny text: Joj vyhrali sme.

Správny text: Joj, vyhrali sme.

Nedokonalosť tohto pravidla závisí predovšetkým od nekonečnej rozmanitosti citosloviec, ktoré môže používateľ testovať. Je teda pochopiteľné, že tagset slovník nie je schopný zachytiť všetky. Vďaka tomu patria medzi úspešné prípady s nasledujúcimi citoslovcami: tralala, fíha, bác, bums, dokelu, ahoj, cveng, plesk. Neúspech pravidlo dosiahne pri použití: hurá, ech, jej …

Zmeňte x na znak násobenia ×

Nesprávny text: Rozlíšenie obrazovky 800x600

Správny text: Rozlíšenie obrazovky 800×600

Nesprávny text: Rozlíšenie obrazovky 800x600x400

Správny text: Rozlíšenie obrazovky 800×600×400

Úspešnosť tohto pravidla je podmienená počtom čísel. Správne upozorní na chybu v prípade výskytu 2 či 3 čísel. Ak je však aplikované na početnejšie reťazce, stáva sa neúčinným.

„Cítiť sa“ sa spája s 1. pádom

Nesprávny text: Cítim sa užitočným.

Správny text: Cítim sa užitočný.

Problém tohto pravidla je v nepokrytí všetkých tvarov odvodených od slova cítiť. Úspešne pokrýva len niektoré: cítiť, cítim, cítia, cítime, cítiš, cíti, cítite. Iné, aj keď pravdepodobne gramaticky menej vhodné formy pokryť nedokáže. Rovnako neúspešné je v prípadoch, keď sa pred hľadaným prídavným menom v 7. páde nachádzajú ešte nejaké výrazy: Cítim sa veľmi užitočným.

Zámeno koho nahraďte zámenom čí

Nesprávny text: Nevedel, koho dom kúpil.

Správny text: Nevedel, čí dom kúpil.

Nesprávny text: Nevedel, koho je tá noha.

Správny text: Nevedel, čia je tá noha.

Nesprávny text: Nevedel, koho auto si požičal.

Správny text: Nevedel, čie auto si požičal.

Ide o podobný prípad ako predchádzajúci, taktiež nedokáže odhaliť chybu ak sa medzi zámenom „koho“ a podstatným menom v prvom páde nachádzajú iné výrazy. Túto nedokonalosť sa sme sa nepokúšali riešiť hlavne z dôvodu veľkej pravdepodobnosti

výskytu podstatných mien v prvom páde, a teda relatívne značnému riziku falošných poplachov. Príklad: Nevedel, koho bol ten dom, ktorý kúpil.

Druhým rokom študovať je nespisovné

Nesprávny text: Chcel už druhým rokom študovať politológiu.

Správny text: Chcel už druhý rok študovať politológiu.

V tomto prípade pravidlo dobre zachytáva všetky hľadané možnosti. Negatívom je však prípadná náchylnosť k falošným poplachom. Toto riziko sme však považovali za prijateľné. Falošný poplach spôsobujú všetky výrazy zložené z číslovky v 7. páde, podstatného mena v 7. páde a ľubovoľného slovesa. Teoreticky tak chybu spôsobí aj výraz: „tretím perom písať“. Tu sa však prejaví jedna z hlavných výhod LanguageTool, a to možnosť pravidlo ignorovať, poprípade dokonca zakázať kontrolu týmto pravidlom.

Pravidlá, ktorými sme sa zaoberali slúžia len na ilustráciu možných nedokonalostí, keďže opisom všetkých vytvorených pravidiel sa práca s pochopiteľných dôvodom zaoberať nemôže. Nepresnostiam sa však počas vývoja nedá stopercentne vyhnúť a autor musí často zvážiť mieru rizika možných nedokonalostí či falošných poplachov.

Záver

Vďaka nástrojom na vývoj pravidiel v XML, s ktorými sme sa oboznámili, je možné pomerne presne a účinne ošetriť veľké množstvo jazykových nepresností vyskytujúcich sa v bežnej textovej komunikácii. Existuje však mnoho prípadov, kde aj napriek veľkej snahe dnes dostupné prostriedky nedokážu vývojárovi vytvoriť prostredie, v ktorom by dokázal presne aplikovať pravidlo na akýkoľvek jazykový problém. Za jeden z najväčších nedostatkov považujem nemožnosť identifikovať vetné členy vo vete a taktiež absenciu možnosti priradiť podstatné mená k ich vzoru. Úspešným vyriešením týchto nedostatkov by sa sprístupnil pomerne veľký priestor na vývoj ďalších pravidiel na kontrolu slovenskej gramatiky.

V rámci tejto práce sa mi podarilo navrhnúť základnú množinu 50 pravidiel na kontrolu slovenskej gramatiky pre program LanguageTool. Súčasťou práce je ich popis a prípady úspešného použitia, pričom aplikácia ponúka každému používateľovi možnosť vypnúť kontrolu určitým pravidlom. Na vytvorenie pravidiel som použil externý XML súbor a tiež som jedno vzorové pravidlo implementoval s jazyku JAVA. Celý postup práce a charakterizácia štruktúry autorskej komunity spolu tvoria potencionálny študijný materiál pre ďalších záujemcov o pokračovanie v projekte. Vytvoril som stylesheet na konverziu XML súboru do html, ktorý je nápomocný pri tvorbe základných testovacích textov.
Práca má ambíciu spopularizovať samotný program LanguageTool a taktiež aj kancelársky balík OpenOffice. V budúcnosti by sa mohla stať pomôckou pre všetkých, ktorí sa chcú zúčastniť tvorby slovenskej verzie. Pre slovenčinu je potrebné vytvorenie ďalších pravidiel tak, aby sa jej podpora mohla porovnávať s iným slovanským a morfologicky rozmanitým jazykom, ktorým je poľština a jej viac ako 1000 pravidiel. Okrem toho je potrebné zlepšiť segmentáciu slovenských výrazov. Osobne som odhodlaný naďalej pokračovať vo vývoji v oblasti pravidiel a dúfam, že moja bakalárska práca prispeje k rýchlosti a kvalite tohto procesu.

Použitá literatúra

LIČKO, J. 2007. J*ednoduchý gramatický korektor pro češtinu do OpenOffice:* Bakalárska práca. Brno: Vysoké technické učení v Brně, 2007. 27 s.

NABER, D. 2003. *A rule-based style and grammar checker*: Diplomová práca. Bielefeld: Universität Bielefled. 2003. 57 s.

KAČALA, J. A kol.: 2000 *Pravidlá slovenského pravopisu*. 3. vyd. Bratislava : Veda, 2000.

Gábriš, M. *Slovenský jazyk do vrecka.* [online]. Radošina: [cit. 2010.04.12.] Dostupné na internete: http://www.slovencina.vselico.com/priamarec.html

Špireng, P. 2006. *Morfologická anotácia textov Slovenského národného korpusu.* [online]. Košice: Dostupné na internete: http://korpus.juls.savba.sk/files/tagset-www.pdf

Jazykovedný ústav Ľudovíta Štúra SAV. *Slovenské slovníky.* [online]. Bratislava: SAV 2010.04.22. Dostupné na internete: http://slovniky.juls.savba.sk/?d=sss

Soursceforge. *LanguageTool development.* [online]. Dostupné na internete: http://sourceforge.net/mailarchive/forum.php?forum_name=languagetool-devel

Milkovski, M. *Developing a grammar checker.* Varšava: 2008. 29 s.

LanguageTool wiki [online]. 2010.5.16. Dostupné na internete: http://languagetool.wikidot.com/

Valentová, Z. 2006. *Tvorba a publikácia matematicekj terminológie pomocou XML:* **Diplomová práca. Bratislava: Univerzita Komenského, 2007. 54 s.**