Univerzita Komenského
Fakulta Matematiky, Fyziky a Informatiky
Katedra Informatiky

Wysiwyg editačné techniky XML dokumentov s použitím XSLT

Diplomová práca

Tomáš Studva **2007**

Wysiwyg editačné techniky XML dokumentov s použitím XSLT

Diplomová práca

Tomáš Studva

Univerzita Komenského v Bratislave
Fakulta Matematiky, Fyziky a Informatiky
Katedra informatiky

Informatika

Vedúci diplomovej práce
prof. RNDr. Branislav Rovan, PhD

bratislava máj 2007

Čestne prehlasujem, že túto diplomovú prácu som vypracoval samostatne len s použitím uvedenej literatúry a s odbornou pomocou vedúceho práce.

Bratislava, máj 2007 Tomáš Studva

Poďakovanie

Chcel by som poďakovať svojmu školiteľovi prof. RNDr. Branislavovi Rovanovi, PhD za vedenie, konzultácie a motiváciu a veľká vďaka patrí mojej rodine za podporu a strpenie.

Abstrakt

XML je dátový štandard pre širokú škálu dokumentov. Jeho rozšírenie spôsobilo veľký dopyt po všeobecných XML editoroch a XML editoroch pre konkrétne oblasti. Vzniklo mnoho editorov a ukázalo sa, že wysiwyg editori výrazne uľahčujú tvorbu dokumentov a sú žiadané. Tvorbu dokumentov uľahčuje aj separácia prezentácie dokumentu od dát dokumentu. To je osvedčený prístup tvorby dokumentov určených pre World Wide Web. K účelu separácie prezentácie od dát a transformácií XML dokumentov bol navrhnutý transformačný jazyk XSLT. Táto práca sa zaoberá editačnými technikami kombinujúcimi tieto výhody, teda wysiwyg a s využitím XSLT resp. inej transformácie. Najskôr prinášame krátky prehľad wysiwyg editačných techník a ďalej v práci analyzujeme techniku editácie pohľadu. Definujeme rozšírenie tejto techniky na wysiwyg editáciu pohľadu riadenej gramatikou a zisťujeme, že existujúce modely nie sú vhodné pre túto techniku. Navrhli sme teda nový model nazvaný TOS vhodný pre túto techniku a slovne sme popísali ako techniku realizovať. Model TOS nás ale sklamal svojou výrazovou silou, je menšia ako sme dúfali. Cieľom tejto práce bola realizácia techniky editácie pohľadu definovaného pomocou XSLT. To sme dosiahli obmedzením výrazovaj sily XSLT transformácie a navrhli sme algoritmy pre jednotlivé editačné operácie.

Klúčové slová: XML, XSLT, wysiwyg editácia, technika editácie dokumentu, technika editácie pohľadu, obojsmerná transformácia, spätná transformácia

Predhovor

Cieľom tejto práce bolo preskúmať a analyzovať poznatky z oblasti do ktorej zapadá, implementovať správu XSLT pohľadu pomocou čiastočnej XSLT transformácie a aplikovať techniku editácie pohľadu na XSLT.

Dosiahli sme prvý a tretí cieľ a v tom vidíme aj prínos práce. Preskúmaná oblasť nie je veľmi široká a existuje iba málo prác na túto tému. Oblasť vychádza z poznatkou získaných v databázovej komunite a v komunite zaoberajúcej sa výpočtom programov alebo transformácií. V práci predstavujeme dve známe základné techniky editácie a to techniku editácie dokumentu a techniku editácie pohľadu.

Druhú z techník sme rozšírili o riadenie editácie gramatikou a navrhli sme nový model TOS, v ktorom sme ju realizovali. Ako vieme, tak sme sa touto technikou zaoberali ako prvý a taktiež sme ju analyzovali pre XSLT. Druhý cieľ sa nám nepodarilo dosiahnuť z dôvodu veľkej časovej náročnosti úlohy. Myslíme si, že táto úloha by bola vhodná pre samostatnú diplomovú prácu. Tretí cieľ, aplikovať techniku editácie pohľadu na XSLT, sa nám podarilo čiastočne dosiahnuť. Navrhli sme obmedzenie XSLT a algoritmy pre jednotlivé editačné operácie. V prípade operácie vloženia, sme časť algoritmu neuviedli presne, a nechávame ju na budúce práce.

Vychádzali sme z referencovanej literatúry, z poznatkou z oblasti formálnych jazykov a z popisu technológií. Taktiež sme využili poznatky získané pri projekte Euromath2[15] a niekoľko-krát sme sa spoľahli na intuíciu.

Klúčové slová: XML, XSLT, wysiwyg editácia, technika editácie dokumentu, technika editácie pohľadu, obojsmerná transformácia, spätná transformácia

Obsah

Úvod **1**

1 Transformačný model XSLT/XPath **4**
1.1 XSLT 4
1.2 XPath 6

2 Wysiwyg XML editor s využitím transformácie **8**
2.1 Tvorba XML dokumentov 8
2.2 Wysiswyg xml editor s využitím transformácie 9
2.3 Technika editácia dokumentu 9
2.4 Technika editácie pohľadu 11
2.4.1 Obojsmerná transformácia 12

3 Editácia pohľadu riadená gramatikou **15**
3.1 Definícia problému 15
3.2 Realizácia techniky - vykonať operáciu a overiť či bola povolená 15
3.3 Preklad schémy do gramatiky pre pohľad 16
3.4 TOS - transformácia definovaná obojsmernými šablónami 16
3.4.1 Realizácia editácie TOS pohľadu riadenej gramatikou s výberom . . 20
3.4.2 Výrazová sila 21
3.4.3 Hodnotenie modelu 21

4 Editácia XSLT/XPath pohľadu **22**
4.1 Výrazové obmedzenie XSLT/XPath pre editáciu pohľadu 22
4.1.1 Obojsmernosť XPath 22
4.1.2 Obojsmernosť XSLT 23
4.2 Algoritmy pre editačné operácie na pohľade definovanom XSLT/XPath . . 24
4.2.1 Operácia zmazania na pohľade 24
4.2.2 Operácia modifikácie na pohľade 26
4.2.3 Operácia vkladania na pohľade 31

5 Porovnanie transformačných modelov/jazykov pre techniku editácie pohľadu **36**
5.1 Výrazová sila . . . 36
5.1.1 XSLT/XPath transformácie . . . 36
5.1.2 Transformácie definované obojsmernými šablónami . . . 36
5.1.3 Transformácie vo funkcionálnom jazyku X . . . 37
5.2 Podpora editácie pohľadu riadenou gramatikou s výberom . . . 37

6 Implementácia - projekt Euromath2 **38**
6.1 Popis . . . 38
6.1.1 Architektúra . . . 38
6.2 Návrh Controllera, všeobecného editora . . . 39
6.3 Dátové formáty editora . . . 39
6.3.1 MathML . . . 40

Záver **42**

Literatúra **42**

A XPath grammar **45**

B Implementácia **47**

Zoznam obrázkov

1 Prezentácia XML dokumentu pomocou XSLT štýlu 2

2.1 Technika editácie dokumentu 10
2.2 Technika editácie pohľadu 12

B.1 Návrch controllera v editore Euromath2 47
B.2 Výkonostný profil editora Euromath2 pre 50 stranový dokument 48

Úvod

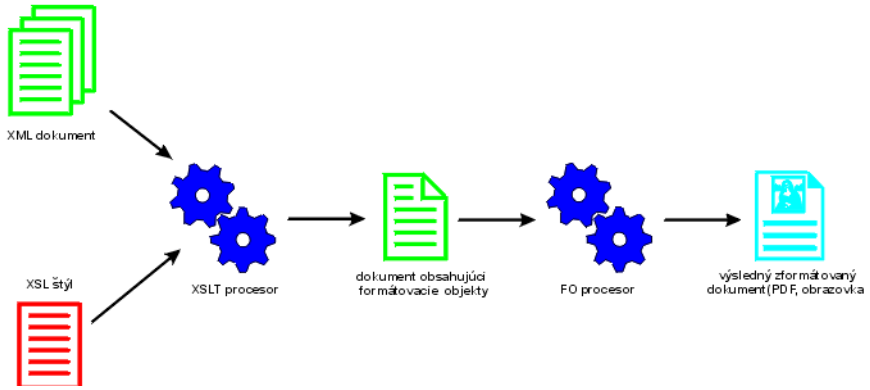
V tejto kapitole vysvetlíme čo nás k práci motivovalo, stručne prezentujeme oblasť do ktorej práca zapadá, ciele práce a popíšeme štruktúru práce.

Techniky tvorby dokumentov človekom v editoroch sú podľa nášho názoru ovplyvnené dátovým formátom dokumentu, používateľským rozhraním editora a doménou do ktorej dokument zapadá. Dátový formát je v našom prípade XML, prípadne XML s asociovanou gramatikou definujúcou jeho typ. Zaoberáme sa práve XML, pretože tvorí ucelenú rodinu technológií a v súčasnosti sa využíva v mnohých aplikáciách.

XML je vlastne rodina technológii a štandardov pre uskladňovanie a prácu s informáciami. Výhody XML sa nedajú zovšeobecniť, vždy záleží na konkrétnom prostredí nasadenia a požiadavkách, ale vlastnosti kôli ktorým sa XML používa sú hlavne: čitateľnosť, portabilita, jednoduchosť a rozšíriteľnosť. Do rodiny XML by som zaradil štandardy XML, XSLT, DTD, XML Schema, XPath, XPointer a nástroje pre ich podporu: validátory, XSLT processory, FO processory, parsery, autorské a vývojové prostredia a mnoho ďalších.

XML sa osvedčilo aj v oblasti tvorby a publikácií dokumentov a vznikli štandardné dátové formáty pre tento účel. Medzi ne patria štandardy XSL FO, MathML, GraphML, ChemML, XHTML .... Pri tvorbe dokumentov je rozumné oddeliť dáta od prezentácie. To sa dá dosiahnuť rozdelením obsahu XML dokumentu na dáta a prezentáciu dát(štýl) zvyčajne popísanú pomocou XSLT transformácie. Aplikovaním XSLT transformácie na XML obsahujúce dáta dostávame výsledný formátovaný dokument, ako vidieť na obrázku 1. Tento prístup je pre používateľa efektívny, lebo mu umožňuje znovu využiť prezentáciu na iné dáta a prezentovať dáta inou prezentáciou. Použitím XSLT transformácie v editore sa dá dosiahnuť aj zjednodušenie architektúry editora a využitie editora vo viacerých doménach, čím je určený druhý faktor ovplyvňujúci editačnú techniku. To znamená, že v tomto prípade musí mať editačná technika všeobecný základ.

Aj keď XML je dobre čitateľné, zložitosť štandardov znemožňuje ich hromadnému používaniu autormi dokumentov. Sú ťažko naučiteľné, rozsiahle a zle zapamätateľné. Wysiwyg editácia rieši tento problém - skrýva mnohé detaily xml pred používateľom. Podľa používateľského rozhrania delíme techniky tvorby dokumentov človekom na: priamu editáciu a wysiwyg (What You See Is What You Get) editáciu. Pri priamej editácii je dokument prezentovaný vo svojom natívnom dátovom formáte a teda používateľ ho musí podrobne poznať. Wysiwyg editácia prezentuje dokument vo vizuálne používateľsky prívetivej forme a preto nevyžaduje podrobne poznať dátový formát. Editácia je riadená cez editačné akcie v jazyku abstraktnom od dátového formátu. Kladie sa dôraz na použiteľnosť, naučiteľnosť,



Obrázok 1: Prezentácia XML dokumentu pomocou XSLT štýlu

dostupnosť. Skupina wysiwyg editačných techník sa delí na:

- wysiwyg, prezentuje dokument v jednej z jeho finálnych vizuálnych prezentácii, kladie sa dôraz na vzhľad a obsah, editačné akcie sú abstraktné od dátového formátu a sú definované nad vizuálnou prezentáciou dokumentu
- wysiwym (What You See Is What You Might), prezentuje dokument v používateľsky prívetivej forme, ale dôraz sa kladie na štruktúru a obsah dokumentu, editačné akcie nemusia byť abstraktné od dátového formátu dokumentu
- wygiwys (What You Get Is What You See), ako wysiwyg, ale editačné akcie a ich efekt je vizuálne zobrazený ešte pred ich použitím, editačné akcie sú kontextové

Tri črty XML, XSLT transformácia a wysiwyg editácia určujú charakter editora a editačnej techniky. Tento charakter má aj editor Euromath2, na ktorom sme pracovali. Pri jeho vývoji sa vyskytlol problém s pomalou odozvou používateľského rozhrania spôsobený vykonávaním XSLT transformácie po každej editačnej operácii. Riešením bolo vykonávať iba čiastočnú XSLT transformáciu, čo už bolo teoreticky preskúmané v prácach [1] a [2]. Oboznámenie sa s riešním a jeho následná implementácia bola motiváciou pre túto prácu.

Cieľom práce bolo preskúmať a analyzovať poznatky z oblasti do ktorej zapadá, implementovať správu XSLT pohľadu pomocou čiastočnej XSLT transformácie a aplikovať techniku editácie pohľadu[1] na XSLT.

V prvej kapitole popisujeme XSLT v praxi najviac používaný tranformačný jazyk pre XML dokumenty a jazyk XPath, ktorý XSLT využíva na navigáciu a adresáciu v XML dokumente. V druhej kapitole sa venujeme wysiwyg editorom a existujúcim wysiwyg editačným technikám s využitím transformácie. Techniky popisujeme, poukazujeme na ich problémy a predkladáme naše postrehy. V tretej kapitole definujeme techniku editácie pohľadu riadenej gramatikou a techniku editácie pohľadu riadenej gramatikou s výberom a

[1] touto technikou sa v práci zaoberáme neskôr

navrhujeme vlastný transformačný model nazvaný TOS a popisujeme ako druhú z techník v modeli TOS realizovať. V štvrtej kapitole navrhujeme algoritmy pre editácia XSLT/XPath pohľadu. V piatej kapitole porovnávame transformačné modely pre techniku editácie pohľadu. V poslednej kapitole sa venujeme návrhu editora Euromath2.

Kapitola 1

Transformačný model XSLT/XPath

V tejto kapitole popisujeme jazyky XSLT a XPath a model pre výpočet tranformácie XSLT/XPath. XSLT je jazyk na popis transformácií aXPath je jazyk na adresovanie v XML dokumentoch. Zaoberáme sa verziami XSLT 1.0 a XPath 1.0, aj keď počas našej práce bola definovaná verzia XSLT 2.0 a XPath 2.0. V práci spomíname a využívame aj iné transformačné modely, vždy však uvádzame referenciu na literatúru, kde sú popísané.

1.1 XSLT

XSLT je jazyk a model pre popis transformácií XML dokumentov. Je definovaný v špecifikácii [10] a je to štandard W3C. Je to funkcionálny jazyk založený na šablónach a XPath vzoroch. Vstupný dokument musí byť XML, výstupný môže byť ľubovoľný. Výpočet XSLT transfomácie spočíva v nájdení najvhodnejšej šáblóny pre koreň vstupného XML dokumentu a následne jej vyhodnotením. Šablóna obsahuje inštrukcie, ktoré sú postupne vyhodnocované. Pre potreby našej práce rozdelujeme inštrukcie do týchto kategórií[1]:

- všeobecné, pomocné a konfiguračné inštrukcie (xsl:stylesheet, xsl:output, xsl:attribute-set, xsl:key, xsl:namespace-alias, xsl:fallback, xsl:preserve-space, xsl:script, xsl:strip-space, xsl:decimal-format, xsl:document)
- inštrukcie pre modularizáciu (xsl:import, xsl:apply-imports, xsl:include),
- inštrukcie pre premenné (xsl:variable, xsl:param, xsl:with-param),
- riadiace inštrukcie (xsl:apply-templates, xsl:template, xsl:choose, xsl:for-each, xsl:if, xsl:otherwise, xsl:when, xsl:call-template, xsl:sort)
- inštrukcie generujúce výstup (xsl:element, xsl:attribute, xsl:text, xsl:processing-instruction, xsl:comment, xsl:value-of, xsl:copy, xsl:copy-of, xsl:message, xsl:number).

Riadiace inštrukcie riadia tok výpočtu transformácie a z nich dôležité [2] sú:

[1] toto rozdelenie sme prebrali z [1]

[2] ostatné riadiace inštrukcie sa dajú týmito nahradiť

- <xsl:template match="pattern"name="qname"priority="number"mode="qname» telo </xsl:template> - definuje šablónu transformácie
- <xsl:apply-templates select="node-set-exp"mode="qname» parametre a sort inštrukcie </xsl:apply-templates> - aplikuje šablóny na zvolenú množinu uzlov v select výraze
- <xsl:for-each select="node-set-exp» sort inštrukcie a inštrukcie pre vykonanie </xsl:for-each>, aplikuje inštrukcie pre vykonanie na zvolenú množinu uzlov v select výraze
- <xsl:if test="boolean-expr» telo </xsl:if> - ak platí podmienka, vykoná sa telo
- <xsl:sort select=expression ... > usporadúva množinu uzlov

V XSLT transformácii je výstup generovaný inštrukciami pre výstup, alebo je výstup zapísaný ako text priamo v šablónach. Priamo zapísaný text sa pri parsovaní kopíruje na výstup. Priamo zapísaný text môže obsahovať elementy, atribúty, textové dáta[3], vykonateľné inštrukcie, komentáre, entity a dá sa dosiahnúť rovnakého efektu pomocou <xsl: ... > inštrukcií generujúcich výstup. V práci využívame všetky výstupné inštrukcie no najviac tieto:

- <xsl:element .../> telo </xsl:element>, generuje element
- <xsl:attribute .../> hodnota </xsl:element>, generuje atribút
- <xsl:text .../> telo </xsl:text>, generuje statický text(okrem referencií na entity)
- <xsl:value-of select=/>, generuje dynamický text
- <xsl:copy-of select=/>, zkopíruje podstrom na výstup
- <xsl:copy/>, zkopíruje aktuálny uzol vstupného stromu na výstup

V XSLT transformáciách sa adresujú a získavajú hodnoty zo vstupného dokumentu pomocou XPath výrazov. XPath je jazyk využívaný aj v iných technológiách napr. v XQuery a je definovaný nezávisle na XSLT. Dvojica XSLT, XPath tvorí transformačný model, ktorý budeme označovať XSLT, prípadne XSLT/XPath.

Výrazová sila XSLT je rovnaká ako sila turingoveho stroja. Sila XSLT je skrytá v rekurzii dosiahnuteľnej pomocou šablón, premenných a parametrov, v podmienkach definovateľných cez XPath a vo funkciách definovaných v jazyku XSLT a v jazyku XPath. Funkcií definovaných v jazyku XPath je viac než 30 a XSLT špecifikácia definuje nasledovné, pričom umožňuje XSLT procesorom dodefinovať ďalšie:

- *Function: node-set current()* - vracia množinu uzlov s jediným prvkom - aktuálnym uzlom

[3] myslí sa text neobsahujúci ostatné uzly xml

- *Function: string generate-id(node-set?)* - vygeneruje reťazec unikátne identifikujúci uzol, ktorý je prvý v poradí podľa dokumentu z množiny node-set
- *Function: string format-number(number, string, string?)* - formátuje číslo podľa vzoru
- *Function: object system-property(string)* - vráti objekt reprezentujúci hodnotu systémovej vlastnosti určenej podľa mena
- *Function: node-set document(object, node-set?)* - umožňuje pristupovať k iným zdrojovým dokumentom ako je primárne spracovávaný
- *Function: node-set key(string, object)* - vráti elementy priradené k špecifikovanému kľúču
- *Function: boolean element-available(string)* - testuje dostupnosť inštrukcie podľa mena
- *Function: boolean function-available(string)* - testuje dostupnosť funkcie podľa mena
- *Function: string unparsed-entity-uri(string)* - vráti URI neparsovanej entity deklarovanej v zdrojovom dokumente

Druhá a tretia od konca je zbytočná pre výpočet a teda ich neuvažujeme. Neuvažujeme ani poslednú, lebo v práci sa entitám nevenujeme. na ostatné sa v práci odvolávame.

1.2 XPath

XPath[4] je jazyk na navigáciu a adresáciu v XML dokumentoch. Je definovaný v špecifikácii [11] a je to štandard W3C. Využíva sa v XSLT, ale je definovaný nezávisle. XML dokument je chápaný ako strom uzlov, pričom uzly sú rozlišované na koreňové, elementy, atribúty, textové uzly, vykonávateľné inštrukcie, komentáre a uzly menných priestorov. Pre každý uzol sa dá získať jeho textová hodnota, či už priamo alebo sa vypočíta. Základom adresácie je XPath cesta, podmienky a viac než 30 funkcií. XPath výraz adresuje objekt, ktorého typ je množina uzlov v XML dokumente, boolovská hodnota, číslo alebo reťazec. Vyhodnocovanie XPath výrazu vyžaduje kontext: kontextový uzol, pozíciu v kontexte a veľkosť kontextu, množinu definovaných premenných a ich hodnoty, knižnicu funkcií a množinu deklarovaných menných priestorov a ich asociovaných prefixov. Počas vyhodnocovania sa môže meniť kontextový uzol, pozícia v kontexte a veľkosť kontextu. XSLT tento kontext definuje a poskytuje pre vyhodnocovanie XPath výrazov.

Dôležitým XPath výrazom je cesta. Cesta adresuje množinu uzlov v závislosti od kontextového uzla. Cesta môže byť relatívna alebo absolútna. Skladá sa z úsekov oddelených '/' alebo '//', vyhodnocovaných z ľava do prava, každý adresujúci množinu uzlov. Prvý

[4] XML Path Language

úsek je vyhodnotený s kontextovým uzlom na množinu uzlov. S každým uzlom z tejto množiny, ako aktuálnym kontextovým uzlom, je vyhodnotený druhý úsek na množinu uzlov a zjednotenie týchto množín uzlov je výsledkom pre cestu tvorenú prvým a druhým úsekom. Podobne sa vyhodnotí nasledujúci tretí úsek. Výsledkom celej cesty je teda vyhodnotenie posledného úseku cesty nad množinou uzlov pre predposledný úsek. Absolútna cesta začína '/', čo adresuje koreňový uzol v dokumente, ktorý obsahuje kontextový uzol. Potom sa pokračuje vyhodotením relatívnej cesty.

Úsek cesty má tri časti: vzťah(axis), typ a meno uzla, nula alebo niekoľko predikátov. Syntax je axis::typ alebo meno uzla[predikát 1][predikát 2] ... [predikát n]. Výsledkom úseku cesty je množina uzlov, ktorá má požadovaný vzťah ku kontextovému uzlu, má typ alebo požadované meno, a postupne vyhovuje predikátom 1 až n. Povolené vzťahy sú *ancestor, ancestor-or-self, attribute, child, descendant, descendant-or-self, following, following-sibling, namespace, parent, preceding, preceding-sibling, self.* Predikát môže využívať pozíciu v množine uzlov, funckie, hodnoty premenných, booleovské hodnoty a operátory, reťazce, čísla a aritmetiku a ďalšie časti vstupného súboru adresovateľné pomocou cesty.

Jazyk XPath definuje 4 základné typy: číslo, reťazec, boolovskú hodnotu a množinu uzlov. Podporuje automatickú konverziu typov na číslo, reťazec, boolovskú hodnotu pomocou zabudovaných funkcií: string, number a boolean.

V XSLT sa XPath využíva na referencovanie množiny uzlov a na pattern matching. V XSLT sa teda dá v ľubovoľnom čase pristupovať k ľubovoľnej časti vstupného dokumentu, a vyhodnocovať zabudované funkcie. XSLT a XPath spolu tvoria výrazovo silný nástroj na transformácie dokumentov(stromov).

Kapitola 2

Wysiwyg XML editor s využitím transformácie

V tejto kapitole definujeme rozdelenie tvorby dokumentov podľa dostupnosti gramatiky, popisujeme návrh wysiswyg xml editora s využitím transformácie a definujeme dve základné techniky editácie, ktoré sú realizovateľné v popísanom editore. Pri technike editácie pohľadu sme hlbšie skúmali spätnú transformáciu a dospeli sme k inej podmienke obojsmernosti ako sa uvádza v referencovanej literatúre.

2.1 Tvorba XML dokumentov

Väčšina xml dokumentov má asociovanú gramatiku, ktorá definuje typ dokumentu. Podľa toho, či je gramatika známa pred samotnou tvorbou dokumentu, rozdeľujeme tvorbu dokumentu na tvorbu dokumentu od nuly a na tvorbu dokumentu podľa gramatiky.

Tvorba dokumentu od nuly

Pri tvorbe dokumentu od nuly je cieľom vytvoriť xml dokument a zároveň aj k nemu asociovanú gramatiku. Sem zaraďujeme aj prípad samotného návrhu gramatík, keď potrebujeme vytvoriť gramatiku pre nový typ dokumentu. Gramatika pre XML je typu DTD[13] alebo XML Schema[12].

Tvorba dokumentu od nuly je náročný proces a samotný proces sa podobá procesu tvorby softvéru. Prebieha cyklus analýza, design, testovanie. Je to iteračný proces a vyvíja sa dvojica gramatika, xml dáta.

Tvorba dokumentu podľa gramatiky

Pri tvorbe dokumentu podľa gramatiky je cieľom vytvoriť xml dokument patriaci do triedy definovanej k nemu asociovanou gramatikou (hovoríme, že je validný podľa gramatiky). To je prípad mnohých štandardných dátových typov: xhtml, mathml, XSL FO, ... V tomto

prípade existujú dva prístupy: dokument je validný po celý čas editácie alebo dokument nemusí byť validný celý čas.

2.2 Wysiswyg xml editor s využitím transformácie

Navrhli sme modulárnu architekúru wysiswyg xml editor s využitím transformácie založenú na princípe Model-View-Controller. Takúto architektúru má aj editor Euromath2 a je bližie popísaná v kapitole implementácia. Myslíme si, že je to dobrý návrch, keďže MVC návrhový vzor je široko používaný a modulárnosť je tiež pozitívna vlastnosť. Editor sa skladá zo 4 základných častí: časť spravujúca dokument a asociovanú gramatiku, časť vykonávajúca transformáciu a poskytujúca transformovaný dokument (ďalej označovaný pohľad), časť riadiaca editáciu a interakciu s používateľom a časť zobrazujúca pohľad. Editor môže obsahovať aj daľšie časti alebo sa odlišovať od predloženého rozdelenia, čo záleží na charaktere editora. Textové editory delia poslednú časť ešte na formatter a renderer. Editor je vlastne automat pracujúci nad konfiguráciou (dokument, gramatika, transformácia). Proces editácie sleduje cyklus: voľba editačnej operácie cez grafické používateľské rozhranie, vykonanie danej operácie - získanie novej konfigurácie a nakoniec zobrazenie pohľadu.

Wysiwyg xml editor môže umožňovať editáciu iba dokumentu, čo je prípad aj editora Euromath 2, alebo môže umožňovať editovať aj ostatné časti konfigurácie, t.j. gramatiku a transformáciu.

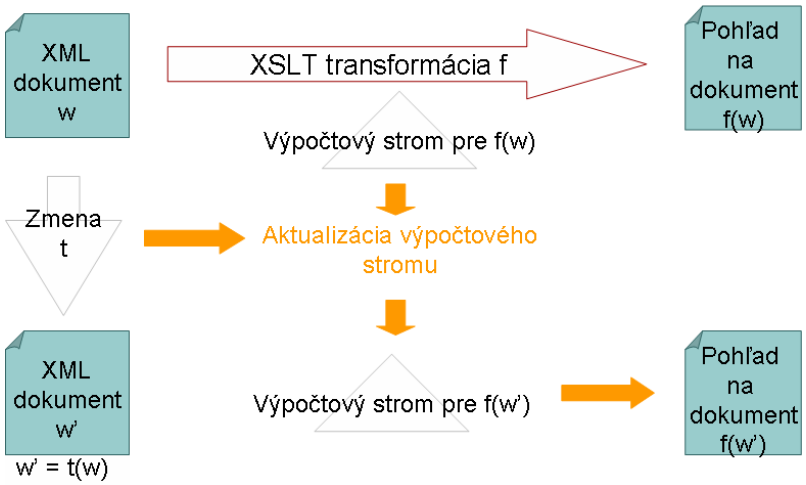
Editori daľej delíme podľa miesta vykonávania editačnej operáci a podľa spôsobu získavania novej konfigurácie (dokument, gramatika, transformácia). Sú dve základné techniky editácie riešiace tieto otázky: technika editácia dokumentu a technika editácia pohľadu.

2.3 Technika editácia dokumentu

Pri tejto technike sú editačné operácie vykonávané v dokumente. Operácie sú kontextové, poskytované gramatikou. Pohľad je synchronizovaný s dokumentom po každej vykonanej editačnej operácii. Následne je pohľad zobrazený. Proces je zobrazený na obrázku 2.1. Za hlávné problémy pri tejto technike považujeme: synchronizáciu pohľadu s dokumentom a mapovanie grafického rozhrania do dokumentu.

Synchronizácia pohľadu s dokumentom môže byť realizovaná vykonaním XSLT transformácie na modifikovanom dokumente, čím dostávame nový aktuálny pohľad. Je to jednoduché riešenie, ale neefektívne, pretože vykonávame celú transformáciu vždy nanovo a teda aj formátovanie a zobrazenie[1], aj keď to nie je nutné. Toto jednoduché riešenie bolo implementované v projekte Euromath 2, čo sa ukázalo pomalé aj na najrýchlejších osobných počítačoch. Čas potrebný na vykonanie celej XSLT transformácie transformácie na veľkom(cca 50 strán) dokumente bol viac ako 1 sekunda a pre niektoré transformácie

[1] aj keď sa vykonáva vždy celá transformácia, dajú sa počítať diferencie medzi dvojicou pohľadov, a tak umožniť vykonať inkrementálne formátovanie a renderovanie dokumentu



Obrázok 2.1: Technika editácie dokumentu

presahoval 4 sekundy. Čas výpočtu XSLT transformácií je $\Omega(n)$, ak n je veľkosť vstupu. Každý editor musí dostatočne rýchlo vykonávať editačné operácie a dostatočne rýchlo na ne reagovať. Preto treba pri návrhu wysiwyg XML editora s použitím transformácie dbať na efektivitu a čo v tomto prípade znamená vykonať transformáciu iba čiastočne. Okrem vykonania transformácie ešte treba vykonať formátovanie a zobrazenie. To je druhý náročný proces. Ako vidieť na obrázku B.2 výkonostného profilu aplikácie Euromath 2, väčšina času sa vykonávalo zobrazenie (formátovanie + vykreslenie) nového pohľadu. Tento profil bol získaný pri editácii 50 stranového dokumentu, ale bez XSLT transformácie.

Správa pohľadu pomocou čiastočnej transformácie (Incremental view maintanance) je efektívne riešenie spomenutého problému. Bola preskúmaná pre výrazovo obmedzené XSLT/XPath transformácie v [1], [2]. Na začiatku sa vykoná celá XSLT transformácia, čím získame pohľad a zároveň zaznamenávame výpočet vo výpočtovom strome XT (XT-tree). Uzol výpočtového stromu má referenciu na XSLT výraz z ktorého vznikol, referenciu na kontextový uzol zo vstupného dokumentu a referenciu na výstup, ak ide o uzol generujúci výstup. Proces správy pohľadu má tri kroky:

- identifikácia XPath výrazov ovplyvnených(nutných znovu vyhodnotiť) cestou definujúcou editačnú operáciu a lokalizácia kontextového uzla vo vstupnom dokumente
- znovu-vyhodnotenie časti XSLT transformácie na nájdenom kontextovom uzle a aktualizácia výpočtového stromu
- aktualizácia výstupu z aktualizovaného výpočtového stromu

Popíšeme ešte bližšie XT strom, pretože ho budeme využívať neskôr pri návrhu algoritmov. XT strom sa skladá z uzlov typu:

- XT-template je vytvorený pre xsl:template inštrukciu a obsahuje referenciu na kontextový uzol

- XT-param je vytvorený pre xsl:param alebo xsl:variable a obsahuje hodnotu danej premennej
- XT-if je vytvorený pre xsl:if a obsahuje deti, ak bola podmienka splnená
- XT-node-set je vytvorený pre xsl:apply-templates a obsahuje referencie na množinu XT-template uzlov
- XT-view je vytvorený pre inštrukcie pre výstup

Mapovanie grafického rozhrania sme v editore Euromath2 riešili pridaním identifikátorov do dokumentu ako atribúty, zachovaním týchto identifikátorov počas transformácie a následne zachovanie a priradenie prislúchajúcim častiam zobrazeného pohľadu. Identifikátory sme generovali prehľadávaním do hĺbky pri načítaní dokumentu. Po vykonaní editačnej operácie sme identifikátory nemenili, iba pre vložené uzly sme použili ako identifikátor najmenšie voľné číslo. Zachovať identifikátory počas XSLT transformácie si vyžaduje upraviť XSLT procesor alebo pri použití neupraveného XSLT procesora upraviť transformáciu. My sme upravovali transformáciu a pre zložité transformácie to bola pracná a náročná úloha.

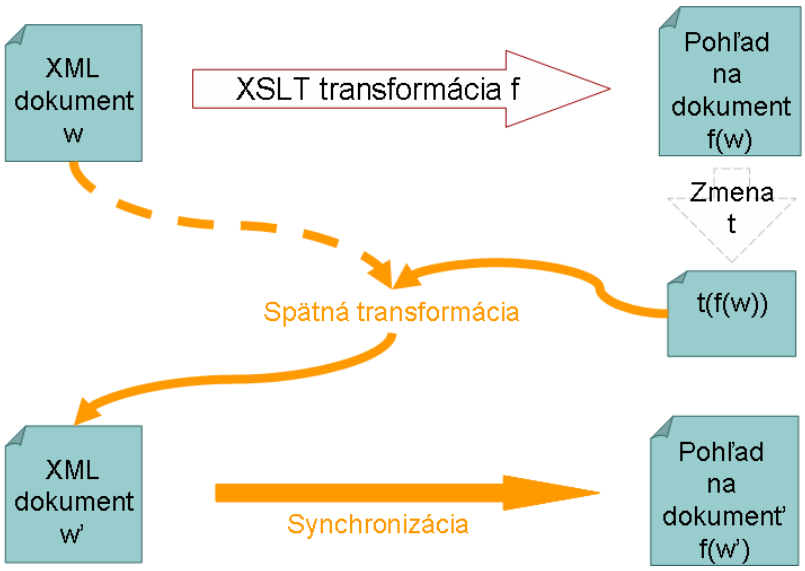
XT strom sám definuje mapovanie medzi dokumentom a pohľadom a preto pri jeho použití nie je nutné upravovať transformáciu. Na mapovanie z pohľadu do grafického rozhrania by sme v prípade použitia XT stromu tiež využili identifikátory.

Technika editácie dokumentu umožňuje bez problémov tvorbu dokumentu od nuly ako aj tvorbu dokumentu podľa gramatiky. Bližšie sme sa technike editácie dokumentu nevenovali, ale myslíme si, že je to veľmi sľubná technika pre wysiwyg štýl editácie. Efektívnu implementáciu tejto techniky s v editore Euromath2 prenechávame budúcim prácam, chceme ale upozorniť, že táto úloha vyžaduje veľa času a je náročná.

2.4 Technika editácie pohľadu

Pri tejto technike sú editačné operácie vykonávané v pohľade. Prechod do novej konfigurácie je nasledovný: dokument je synchronizovaný s modifikovaným pohľadom a následne pohľad znovu synchronizovaný s dokumentom, z dôvodu možných existujúcich závislostí v pohľade. Táto dvojstupňová synchronizácia môže byť vykonaná aj v jednom kroku. Proces je zobrazený na obrázku. 2.2.

Táto technika bola preskúmaná pre funkcionálny transformačný jazyk X, ktorý bol využitý pri návrhu wysiwyg editora [4]. X je výrazozvo silný funkcionálny transformačný jazyk a umožňuje vyjadriť väčšinu používaných transformácií. X je v editore prekladaný do *injektívneho* obojsmerného jazyka Inv [3], ktorý automaticky definuje spätnú transformáciu. Neinjektívnosť niektorých funkcií z jazyka X je ošetrená pri preklade skopírovaním vstupu k výstupu a teda všetky funkcie $f : S \to V$ vyjadriteľné v X sú po preklade v Inv interpretované ako $f' : S \to (S \times V)$. Jazyk Inv definuje operácie nad pármi, zoznamami, stromami. Obsahuje aj nesurjektívnu funciu - duplikáciu, podmienky, a rekurzívne



Obrázok 2.2: Technika editácie pohľadu

definície funkcií. Obsahuje kombinátory na skladanie funkcií a umožňuje point-free štýl programovania. Pre každú funkciu v jazyku Inv, je definovaná opačná funkcia. Opačné funkcie majú dodefinovanú sémantiku na rozšírenom obore hodnôt, aby umožňovali editáciu. Nevýhodou tohoto jazyka je nutnosť pri každej editačnej operácii vykonať spätnú transformáciu na celom pohľade a následne aj transformáciu na celom aktualizovanom dokumente. To je operácia rozsahu $O(n + m)$, čo je neefektívne. My sme zadefinovali transformačný jazyk a model TOS umožňujúci robiť synchronizáciu efektívnejšie. Model je prezentovaný v tretej kapitole.

2.4.1 Obojsmerná transformácia

V tejto časti bližšie skúmame spätnú transformáciu a následne definujeme podmienku obojsmernosti. Prichádzame k inej podmienke ako v [3] a podmienku v [3] považujeme za nesprávnu.

Cieľom obojsmernej transformácie je možnosť získať modifikovaný zdrojový dokument po modifikácii pohľadu. Obojsmerná transformácia je dvojica f, g, kde f je transformácia a g je spätná transformácia ku f. Pozrieme na charakter dvojice f, g odhliadnuc od použitého modelu pre ich špecifikáciu a výpočet.

Majme transformáciu f, dokument d, pohľad $v = f(d)$ a modifikovaný pohľad v'. Chceme získať dokument d'. Ak modifikovaný pohľad v' patrí do oboru hodnôt f a f je injektívna, potom existuje inverzná funkcia k f a vyžadujeme aby $d' = f^{-1}(v') = g(v')$. Ak modifikovaný pohľad v' patrí do oboru hodnôt f a f nie je injektívna, je viacero prislúchajúcich d' a nevieme ktoré zvoliť. V praxi sa používajú aj neinjektívne transformácie, ktoré strácajú informácie, napr. utriedenie uzlov v strome, alebo selektory. K funkcii $f = sort$

môžeme definovať opačnú funkciu ako $g = sort$, čo umožní editáciu. Príkladom selektora je funkcia $first(a, b) = a$, alebo výrazy XPath. Môžeme k nim definovať opačnú funkciu, ak g dostane ďalší argument - dokument d, pre funkciu first: $first_g((a, b), c) = (c, b)$. My nevyžadujeme, aby f bolo injektívne a g definuje ako funkciu $g : (D \times V) \rightarrow D$.

Definícia 1. *Základné editačné operácie sú:*

$$insert(d, path, index, tree) \rightarrow d',\ kde\ d, tree, d' \in D, path \in P, index \in Int$$

$$delete(d, path) \rightarrow d',\ kde\ d, d' \in D, path \in P$$

$$modify(d, path, newValue) \rightarrow d',\ kde\ d, d' \in D, path \in P, newValue \in String$$

Operácia insert vkladá strom tree do dokumentu d. Ak uzol na ceste path je element n, tak strom tree sa vloží ako dieťa s indexom index pod uzol n. Ak uzol na ceste path je textový uzol n, tak strom tree sa vloží za znak s indexom index − 1, to znamená, že sa textový uzol n rozpadne na dva v mieste znaku index a strom tree sa vloží medzi ne zavesením pod rodiča n. Operácia delete vymaže podstrom v dokumente d určený cestou path. Operácia modify modifikuje uzol na ceste path na hodnotu newValue. V prípadoch, ktoré neboli spomenuté sú operácie neplatné, invalidné.

Modifikovaný pohľad v' nepatrí do oboru hodnôt f, ak pohľad obsahuje funkčné závislosti, t.j. f nie je surjektívna a editor na pohľade vykonáva základné editačné operácie (vkladanie, zmazanie, modifikácie). Príklad funkčnej závislosti je duplikácia alebo podmienka. Ak teda v' nie je obrazom žiadneho dokumentu, bolo by rozumné nájsť dokument d', ktorý zachoval vykonanú editačnú operáciu a zároveň sa minimálne líšil od d. Mal by zachovať editačnú operáciu, pretože používateľ ju chce vykonať, ale nie vždy sa to dá, viď príklad v jazyku xslt.

Príklad, XSLT transformácia f.

```
<xsl:stylesheet \ldots >
    <xsl:template match="/">
        <xsl:choose>
            <xsl:when test="count(//section) > 10">
                Dokument obsahuje viac ako 10 sekcií.
            </xsl:when>
            <xsl:otherwise>
                Dokument obsahuje menej/práve ako 10 sekcií.
            </xsl:otherwise>
        </xsl:choose>
    </xsl:template>
</xsl:stylesheet>
```

Obor hodnôt f = {"Dokument obsahuje viac ako 10 sekcií", "Dokument obsahuje menej ako/práve 10 sekcií"}. Ak dokument obsahuje 20 sekcií, tak pohľad v = "Dokument obsahuje viac ako 10 sekcií". Modifikácia v na v' = "Článok obsahuje viac ako 10 sekcií"bez

modifikácie f nie je možná. To pre používateľa predstavuje problém, ak nepozná podrobne f, resp. obor hodnôt f.

Ako teda bolo spomenuté, sú dve základné otázky dizajnu obojsmernej transformácie (f, g): povoliť neinjektívnosť f a povoliť nesurjektívnosť f. My nevyžadujeme, aby f bola surjekcia a ani injekcia. Obojsmernú transformáciu definujeme tak, aby sme mohli vykonať synchronizáciu pohľadu s dokumentom po modifikácii.

Definícia 2. *Hovoríme, že transformácia $T = (f, g)$, kde f a g sú funkcie $f : D \to V$, $g : (D \times V) \to D$, D je množina všetkých dokumentov, V je množina všetkých pohľadov, je obojsmerná ak platia nasledovné podmienky:*

$$g(d, f(d))) = d, \text{ kde } d \in D$$

$$g(d', f(d')) = d', \text{ kde } d' = g(d, v), d \in D, v \in V$$

Funkciu g nazývame spätná transformácia, resp. opačná k f.

Prvá podmienka hovorí že, ak pohľad $f(d)$ na dokument d nie je modifikovaný, tak g vráti pôvodný dokument, je silnejšia ako v [3] a myslíme si, že je nesprávne ju zoslabiť. Druhá podmienka obojsmernosti zaručuje, že po spomenutej dvojstupňovej synchronizácii, je pohľad obrazom dokumentu, a teda už nie je nutná ďalšia synchronizácia. Táto druhá podmienka je prebratá z [3] a nie je nutné, aby bola silnejšia, v silnejšej verzii je prezentovaná v [7].

Kapitola 3

Editácia pohľadu riadená gramatikou

V tejto kapitole rozširujeme techniku editácie pohľadu na tvorbu dokumentu podľa gramatiky. Predkladáme riešenia pre realizáciu takejto techniky, ale nenašli sme model, kde by sme vedeli techniku ľahko realizovať aj keď sme obmedzili jeho výrazovú silu. Preto sme zadefinovali vlastný model a jazyk, kde sa nám to podarilo.

3.1 Definícia problému

V [18] bolo ukázané, že ak máme xml dokument D validný podľa schémy(gramatiky) a základnú modifikačnú operáciu, tak vieme efektívne zistiť či daná schéma povoľuje danú operáciu. Naviac vieme používateľovi poskytnúť na výber z povolených operácií v danom mieste dokumentu, čo sme implementovali v editore Euromat2. Túto techniku nie je možné priamočiaro aplikovať na techniku editácie pohľadu. Pri technike editácie pohľadu sa operácie vykonávajú na pohľade, pre ktorý však nemáme gramatiku.

Definícia 3. *Nech d je xml dokument validný podľa asociovanej gramatiky G, $T = (f, g)$ je obojsmerná transformácia a v je pohľad získaný aplikáciou f na d, $v = f(d)$. Editácia pohľadu riadená gramatikou je technika, ktorá povoľuje na pohľade v iba množinu modifikačných operácií M takých, ktoré po vykonaní na v a premietnutí do dokumentu d, zachovajú dokument d validný. T.j. ak $m \in M, m(v) = v'$ a $d' = g(v')$, tak d' je validný dokument podľa G.*

3.2 Realizácia techniky - vykonať operáciu a overiť či bola povolená

Najjednoduchším riešením je vykonať operáciu na pohľade, a modifikovaný dokument získaný spätnou transformáciou zvalidovať. Je to vhodné riešenie pre editáciu pohľadu riadenú gramatikou pre väčšinu transformačných modelov [1]. V praxi je ale pre používateľa

[1] toto riešenie sa dá aplikovať aj na funkcionálny model pre jazyk X

náročné zvoliť editačnú operáciu na pohľade tak, aby zachovala po spätnej transformácii dokument validný. Vyžaduje to znalosť gramatiky pre dokument a aj znalosť použitej transformácie. Pre zložité transformácie je to až nemožné. Preto je potrebné poskytnúť používateľovi množinu povoľovaných operácií v danom mieste pohľadu. Pre operáciu vloženia, je to zoznam vložiteľných typov uzlov a ich hodnôt, pre modifikáciu zoznam hodnôt a pre vymazávanie, sú to závislosti v pohľade. Uvedomujeme si, že tento problém je niekedy neriešiteľný[2], lebo jeho riešenie je podobné výpočtu oboru hodnôt funkcie. Vieme ale, že obmedzením výrazovaj sily jazyka, môžeme dospieť k riešeniu. Ďaľším problémom je ako prezentovať množinu povolených hodnôt - reťazcov. Výrazovo silnou reprezentáciou by bol regulárny výraz, ale my budeme uvažovať iba dva prípady: konečnú množinu hodnôt popísanú vymenovaním prvkov a nekonečnú množinu hodnôt reprezentovanú *.

Definícia 4. *Editácia pohľadu riadená gramatikou s výberom je technika editácie pohľadu riadenej gramatikou rozšírená o výber editačnej operácie dopredu.*

3.3 Preklad schémy do gramatiky pre pohľad

Editačné operácie sú vykonávané v pohľade v a XML schému(gramatiku) G máme pre dokument d. Jedným z riešení je preklad schémy G do gramatiky pre pohľad v. Ak by preložená gramatika bola regulárneho typu (resp. typu XML Schema alebo DTD), tak by sme vedeli získať množinu vložiteľných elementov v danom mieste a tiež efektívne overiť, či je pohľad validný, napr. po operácii vymazania. Ako sa ukázalo, výsledná gramatika bude kontextová už pri umožnení duplikácie ako transformácie (príkladom je $w \rightarrow ww$). Intuícia nám vravela, že toto nie je správna cesta vedúca k riešeniu a tak sme sa po nej nevydali.

3.4 TOS - transformácia definovaná obojsmernými šablónami

Predchádzajúce riešenia editácie pohľadu riadeného gramatikou s výberom sú neefektívne alebo sa nám nedzali vhodné. Problémom sa zdá byť priveľká výrazová sila transformácie, ktorá nám nedovoľuje efektívne vypočítať množinu povolených operácií, a zložitosť použitého modelu pre výpočet transformácie. Ak je jazyk výrazovo silný, je nutné dynamicky(transformácia sa musí simulovať) počítať pre dané miesto v pohľade povolené hodnoty pre modifikáciu alebo vkladanie. To je prípad funkcionálneho jazyka X.

My sme sa pokúšali nájsť model a neskôr zadefinovať model, kde by tieto informácie boli ľahko prečítateľné priamo zo zápisu transformácie a prípadne z mapovania vstupného dokumentu na výstupný, získaného počas výpočtu transformácie. Uvažovali sme o modeloch popísaných v [19] a to: syntaxou riadené prekladové schémy, atribútové gramatiky, stromovo-transformačné gramatiky a zostupné stromové prekladače, ale nezdali sa nám

[2]napríklad pre triedu jazykova definovanú turingovým strojom

vhodné. Potom sme sa pokúšali realizovať túto myšlienku pre funkcionálny jazyk podobný jazyku X, keďže jazyk X bol narvhnutý pre techniku editácie pohľadu. Myšlienkou úpravy modelu bolo najprv aplikovať selektory na vstupnú časť dokumentu a následne na zvolenej vypočítať obojsmernú funkciu. Pokus bol neúspešný a nepomohlo ani obmedzenie výrazovej sily použitých funkcií.

Neskôr sme navrhli model nazvaný TOS[3], kde operácia vloženia riadená gramatikou je jednoducho realizovateľná. TOS taktiež umožňuje editáciu pohľadu riadenú gramatikou s výberom. Snažili sme sa zachovať čo najväčšiu výrazovú silu tohoto modelu, avšak výrazová sila modelu je omnoho menšia ako sme požadovali. Tento model teraz prezentujeme. Jeho návrh vychádza zo šablón použitých v XSLT, ktoré robíme obojsmernými a meníme model vyhodnocovania takéhoto jazyka. Taktiež preberáme zápis z jazyka XSLT.

Príklad obojsmernej šablóny:

```
<xsl:template matchin="pattern na vstupe" matchout="pattern na výstupe">
    (f,g) nad patternami
</xsl:template>
```

Pattern umožňuje navigáciu vo vstupnom a výstupnom strome. Preberá syntax z XPath popisu cesty a zjednotenie pre zjednodušenie zápisu [4]. Patternu vyhovuje množina uzlov v strome [5]. Od XPath vyhodnocovania patternu sa uvažovaný pattern líši, že nevyžaduje počiatočný kontext. To znamená, že povoľuje iba absolútne cesty, ktoré začínajú '/' alebo '//'. Taktiež sme obmedzili navigáciu v strome a to iba v smere k potomkom. Takže ak postupujeme po popísanej ceste, v strome stále klesáme.

```
$Pattern ::= AbsolutePath | AbsolutePath '|' Pattern$
AbsolutePath ::= '/' RelativePath? || '//' RelativePath?
RelativePath ::= Step '/' RelativePath | Step '//' RelativePath | Step
Step ::= AxisSpecifier NodeTest | AbbreviatedStep
AbbreviatedStep ::= '.'
AxisSpecifier ::= AxisName '::' | AbbreviatedAxisSpecifier
AbbreviatedAxisSpecifier ::= '@'?
AxisName ::= 'attribute' | 'child' | 'descendant'
             | 'descendant-or-self' | 'self'
NodeTest ::= NameTest | NodeType '(' ')'
             | 'processing-instruction' '(' Literal ')'
NameTest ::= '*' | NCName ':' '*' | QName
NodeType ::= 'comment' | 'text' | 'processing-instruction' | 'node'
```

Obojsmerná šablóna sa môže aplikovať, ak matchin pattern vyhovuje uzlu x vo vstupnom strome a matchout pattern vyhovuje uzlu y vo výstupnom strome. Potom funkcia f sa aplikuje na zoznam $x, x.parent, x.parent.parent, \ldots, root$. Výstup f sa zapíše ako dieťa pod uzol y. Funckie f a g vracajú strom zapísaný ako v XSLT kôli jednoduchosti a prehladnosti zápisu, teda syntax funkcií je čiastočne prebratá z XSLT. Funkcie f a g spĺňajú podmienku obojsmernosti.

[3] TOS = transformácia obojsmerných šablón

[4] znak '|'reprezentuje zjednotenie

[5] neuvažujeme namespace uzly

Transformácia t pozostáva z množiny takýchto šablón $t = \{t_1, t_2, t_3, \ldots\}$. Výpočet transformácie pre vstupný dokument d pozostáva z prechodu vstupným dokumentom do šírky a vyhodnocovaním šablón, ktorých pattern vyhovuje aktuálne skúmanému uzlu. Je to efektívne, lebo prechádzaním súrodencov postupne za sebou, cesta k nim sa nemení. Ďalej takýto prechod je nutný pre model z dôvodu, neschopnoti modelu špecifikovať pozíciu vo výstupnom súbore v rámci súrodencov. Tu je ešte priestor na ďalšiu prácu a výskum, ako špecifikovať navigáciu vo výstupnom strome, aby podporovala operáciu vkladania, ako náš model. Nasledujúci príklad je transformácia generujúca obsah z článku zapísaný v HTML. Funkcie g v príklade neuvádzame, ale existujú.

Príklad - transformácia generujúca obsah

```
Vstupné XML:
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook V4.1//EN">
<article>
  <articleinfo>
    <title>An example article</title>
    <author>
      <firstname>Your first name</firstname>
      <surname>Your surname</surname>
      <affiliation>
        <address><email>foo@example.com</email></address>
      </affiliation>
    </author>
    <copyright>
      <year>2000</year>
      <holder>Copyright string here</holder>
    </copyright>
    <abstract>
      <para>If your article has an abstract then it should go here.</para>
    </abstract>
  </articleinfo>
  <sect1>
    <title>My first section</title>
    <para>This is the first section in my article.</para>
  </sect1>
  <sect1>
    <title>My second section</title>
    <para>This is the second section in my article.</para>
  </sect1>
</article>

Transformácia:
<xsl:template matchin="/article" matchout="/">
    f(List l):
    <html>
        <head>
            <title>
```

```
            </title>
        </head>
        <body>
            <h1>
            </h1>
        </body>
    </html>
    g:
</xsl:template>

<xsl:template matchin="/article/articleinfo/title/child::text()"
    matchout="/html/head/title | /html/body/h1">
    f(List l):
    l.first
g:
</xsl:template>

<xsl:template matchin="/article/sect1/title" matchout="/html/body">
     f(List l):
     <h2>
        l.first
    </h2>
g:
</xsl:template>

Výstupný dokument:
<html>
    <head>
        <title>
            An example article
        </title>
    </head>
    <body>
        <h1>
            An example article
        </h1>
        <h2>
            My first section
        </h2>
        <h2>
            My second section
        </h2>
    </body>
</html>
```

3.4.1 Realizácia editácie TOS pohľadu riadenej gramatikou s výberom

Pre vykonanie operácií riadených gramatikou na pohľade využijeme pomocnú štruktúru, bude obsahovať mapovanie medzi uzlami vstupného dokumentu, použitými šablónami a uzlami výstupného dokumentu.

Operácia zmazania

Operácia zmazania uzla n v pohľade môže byť jednoducho vykonaná. Stačí nájsť príslušnú šablónu asociovanú s uzlom n a uzol u zo vstupného dokumentu asociovaný s nájdenou šablónou. Potom vymazaním uzla u a výstupom šablóny [6] dosiahneme požadovanú operáciu. Následne je ešte nutné vymazať výstupy ostatných šablón asociovaných s u. Tento proces môže byť pre používateľa veľmi zprehladnený, ak výstupy zo šablóny pre operáciu zmazania budú tvoriť celky. To môže byť zvýraznené vizuálnym spôsobom napríklad pomocou farieb.

Operácia modifikácie

Obojsmerné šablóny generujú v mnohých prípadoch celé stromy, a nie jednoduché hodnoty ako reťazec, tak operácia modifikácie s výberom, bude poskytovať množinu stromov, ktoré môžu byť vo výstupe nahradené. Opäť takúto operáciu zprehľadníme vytvorením celkov v pohľade. Nahraditeľné stromy môžu byť postupne generované ako nimi používateľ listuje.

Operácia modifikácie celku bude spočívať v nájdení asociovanej šablóny a uzla u zo vstupného dokumentu. Potom pre každý uzol na ceste koreň u, získame z gramatiky množinu hodnôt. Ak je niektorá množina nekonečná, podporujeme operáciu modifikácie pohľadu bez výberu, a pomocou g môžeme aktualizovať dokument a vykonať synchronizáciu. Ak ani jedna množina hodnôt nie je nekonečná, tak postupným aplikovaním f získame množinu stromov, na ktoré je možné strom z pohľadu modifikovať. Po výbere používateľom, použijeme funkciu g na aktualizáciu dokumentu a vykonáme synchronizáciu.

Operácia vkladania

Ak chceme vložiť uzol na niektoré miesto v pohľade, tak zistíme ktoré zo šablón majú matchout pattern vyhovujúci danému miestu. Potom pre každú takú šablónu nájdeme miesto vo vstupnom dokumente, kde by mohla byť aplikovaná. Miesto vo vstupnom dokumente, rodiča, nájdeme pomocou matchin patternu a pozíciu pre vloženie dieťaťa určíme zo sekvenčého charakteru vykonávania transformácie. Pomocou matchin patternu určíme aj podmienku na uzol, ktorý má byť vložený. Podmienka je tvorená typom a prípadne menom uzla z patternu. Potom pomocou gramatiky pre dokument vieme určiť asociované pravidlá v danom kontexte a použiť prístup z modifikácie na získanie vložiteľných stromov v pohľade. Po zvolení stromu používateľom, sa použije g, čím pridáme uzol u do dokumentu, vykonáme na ňom šablóny a od tohoto miesta pokračujeme ako keby v normálnom výpočte transformácie. Pokračovať vo výpočte je nutné, lebo tým, že sme zmenili(rozšírili) pohľad, môže matchout pattern niektorej zo šablón aplikovaných po u vyhovovať rozšírenému pohľadu.

[6] uzol n nemusí byť koreňom výstupu šablóny

3.4.2 Výrazová sila

Výrazová sila modelu a jazyka je menšia ako výrazová sila XSLT, keďže umožňuje referencovať iba uzly na ceste k vrcholu. Umožňuje však napríklad duplikáciu alebo nepreniesť informácie z dokumentu do pohľadu.

3.4.3 Hodnotenie modelu

S modelom a jazykom nie sme celkom spokojný, lebo má menšiu výrazovú silu ako sme požadovali a nerieši systémovo problém operácie modifikácie. Na druhej strane poskytuje závislosti v pohľade, ktoré sa dajú používateľovi prezentovať a prezentovali sme návod, ako by sa dali vypočítať vložiteľné stromy v pohľade.

Kapitola 4

Editácia XSLT/XPath pohľadu

V tejto kapitole navrhujeme techniku editácie pohľadu založenú na XSLT/XPath transformácii. Techniku navrhujeme pre výrazovo obmedzené XSLT/XPath transformácie podľa podmienky obojsmernosti. Ďalej navrhujeme algoritmy potrebné na vykonanie základných editačných operácií na pohľade definovanom XSLT/XPath transformáciou. Pri týchto algoritmoch sa opierame o dátovú štruktúru, výpočtový strom XT, popísanú v [2]. Výpočtový strom XT zaznamenáva mapovanie medzi XSLT inštrukciami, vstupným a výstupným dokumentom. Pomocou neho, vstupného dokumentu a pohľadu definujeme spätnú transformáciu, prekladajúcu operáciu z pohľadu do operácie v dokumente. Preloženú operáciu na dokumente vykonáme a ďalej postupujeme ako v prípade techniky editácie dokumentu - aktualizujeme XT strom a aktualizujeme pohľad. Navrhli sme aj primitívny algoritmus pre výpočet čiastočného oboru hodnôt pre dvojicu gramatika, transformácia. Ten využijeme pre výber modifikačnej operácie používateľom.

4.1 Výrazové obmedzenie XSLT/XPath pre editáciu pohľadu

Aby sme umožnili editácia XSLT/XPath pohľadu definujeme výrazové obmedzenie XSLT/XPath transformácie.

4.1.1 Obojsmernosť XPath

XPath počíta aj neobojsmerné funkcie a preto je nutné výrazové obmedzenie. Definujeme rozdelenie podľa podmienky obojsmernosti. Pre obojsmerné XPath výrazy sa dá dodefinovať spätná funkcia mapujúca hodnoty výrazu na argumenty výrazu. Pre jednosmerné výrazy sme spätnú funkcia nenašli, a teda nepodporujú editáciu pohľadu.

XPath výrazy podľa syntaktickej analýzy delíme na obojsmerné a jednosmerné. Do obojsmerných XPath výrazov nezaradujeme boolovskú logiku mimo predikátov a teda nemá zmysel ponechať tam výnimky[1] ako *not, true, false*. Preto celú boolovskú logiku a boolovké typy mimo predikátov považujeme za jednosmerné. V prípade aritmetiky je to obdobné, a preto aritmetiku

[1] obojsmerné funkcie

a čísla mimo predikátov považujeme za jednosmerné [2].

Jednosmerné XPath výrazy sú výrazy obsahujúce tieto funkcie: */, and, or, !=, =, <, <=, >, >=, +, -, div, mod a last, position, count, concat, contains, string-length, boolean, lang, sum, floor, ceiling, round, not, true, false.*

Obojsmerné výrazy sú teda tvorené referenciami obojsmerných premenných, reťazcami, cestami, predikátmi a volaniami nasledovných funkcií: *id(string), local-name, namespace-uri, name, string(node-set), starts-with, substring-before, substring-after, substring*[3]. Referencie obojsmerných premenných sú také, ktoré v definícii hodnoty pomocou <xsl:param> alebo <xsl:variable> využili iba obojsmerné XPath výrazy. Predikáty sme povolili v obojsmerných výrazoch a môžu obsahovať ľubovoľné(jednosmerné aj obojsmerné) výrazy. K spomenutým obojsmerným funkciám sa dajú dodefinovať ich spätné časti. Zaujímavým príkladom sú tieto funkcie:

- id

$$f : id(string) \rightarrow node\text{-}set$$

$$g : id^o(idValue, nodes) \rightarrow string :$$

$$idValue\ if(\forall node \in nodes \,|\, node.id = idValue)\ else\ error$$

- string definujeme, iba ak node-set.first je typu textový uzol

$$f : string(node\text{-}set) \rightarrow string :$$

$$nodes.first.value\ if nodes.first.type = textNode$$

opačná funkcia g zmení hodnotu prvého uzla na string

$$g : string^o(string, node\text{-}set) \rightarrow node\text{-}set$$

$$string^o(value, nodes) = nodes.copy[nodes.first.value = value]$$

4.1.2 Obojsmernosť XSLT

Analogicky ako v prípade jazyka XPath, je nutné aj výrazové obmedzenie jazyka XSLT. Toto obmedzenie sme navrhli podľa obmedzenia jazyka XPath a podľa charakteru XSLT inštrukcií a funkcií. Nedá sa ale tvrdiť o rozdelení inštrukcií podľa podmienky obojsmernosti, pretože inštrukcie sú nevynímateľné z kontextu a teda by takéto rozdelenie bolo chybné.

Všeobecné, pomocné a konfiguračné inštrukcie povoľujeme, ale sa nimi v algoritmoch nezaoberáme, keďže neovplyvňujú[4] výpočet transformácie. Inštrukcie pre modularizáciu nie sú potrebné, lebo existuje normálny tvar transformácií, kde nie sú.

Všetky riadiace inštrukcie a inštrukcie pre premené považujeme za obojsmerné, ak pracujú s obojsmernými XPath výrazmi. Všetky inštrukcie generujúce výstup okrem xsl:number považujeme za obojsmerné, ak pracujú s obojsmernými XPath výrazmi. Inštrukciu xsl:message neuvažujeme, lebo nie je potrebná.

[2] čísla budeme automaticky považovať za reťazec

[3] funkcie *id(string), string(node-set)* má typ argumentu s zúžený oproti [11]

[4] resp. ovplyvňujú ale zvláštnym spôsobom

Nasledujúce z funkcií definovaných v XSLT považujeme za obojsmerné a zaraďujeme ich medzi obojsmerné XPath funckie: current a generate-id. Zvyšné nie sú potrebné a format-number nie je obojsmerná.

O toto rozdelenie sa opierame v nasledujúcich algoritmoch. Algoritmy sú navrhnuté pre obojsmerné inštrukcie, ale všímajú si iba inštrukcie zaujímavé pre danú operáciu a umožňujúce ju. Výstup generujúce inštrukcie uvažujeme aj v prípade, ak generujú pevný výstup a teda nie sú obojsmerné funkcie, napríklad <xsl:text> abcdef </xsl:text>.

4.2 Algoritmy pre editačné operácie na pohľade definovanom XSLT/XPath

V tejto časti prezentujeme algoritmy, ktoré sme navrhli. Sú písané v pseudokóde a sú popísané komentárom. V algoritmoch používame niekoľko nepresných vyjadrení a preto ich najprv ozrejmíme:

XT uzol je typu výraz nehovorí o samotnom type XT uzla, ale o type inštrukcie s uzlom asociovanej

XPathVal je funkcia vyhodnocujúca XPath výraz klasickým spôsob definovaným v špecifikácii XPath [11]

Taktiež predpokladáme pre všetky algoritmy:

- normálne trary XSLT transformácií, kde pre výstup sú použité iba výstup generujúce inštrukcie, inštrukcia xsl:text generuje iba pevne zadanú hodnotu a z riadiacich inštrukcií sú použité iba: xsl:if, xsl:apply-templates, xsl:for-aech, xsl:template, xsl:sort.
- modifikácia dokumentu d je vykonaná na kópii, a teda všetky hodnoty načítavané z dokumentu d počas vykonávania algoritmov sú pôvodné. Kópia je iba virtuálny dokument, tvorený zaznamenanými editačnými operáciami na nej a pôvodným dokumentom d. Operácie sú zaznamenané vo forme zoznamu v tvare základných editačných operácií. V ľubovoľnom momente vie kópia poskytnúť svoj obsah - dokument d'[5] a aj zoznam operácií.
- po každej vykonanej celej spätnej transformácii, by v editore bola virtuálna kópia zvalidovaná asociovanou gramatikou.
- vyhodnotenie pomocou XPathVal sa dá vykonať efektívnejšie, ak sú údaje zaznamenávane naviac v XT strome pri každej aktualizácii stromu, avšak zvyšuje to pamäťové nároky

4.2.1 Operácia zmazania na pohľade

Chceme vykonať operáciu zmazania na pohľade v.

$$delete(v, path) \rightarrow v', kdev, v' \in V, path \in P$$

[5] dokument d' sa musí vypočítať, ale pristupuje sa k nemu zriedkavo, ak vôbec

Treba nájsť a vymazať prislúchajúce časti v dokumente d také, aby sa dosiahlo požadovanej operácie a prípadne ďalších zmien v pohľade. Cesta *path* určuje podstrom vo v, označme koreňový vrchol tohoto podstromu n. Pomocou XT stromu, vieme ktorá výstupná inštrukcia/vrchol generuje n, označme tento vrchol x_n. Požadovanú operáciu zmazania dosiahneme, ak modifikujeme dokument d tak, že vykonaním inštrukcie x_n sa nevygeneruje n, resp. že sa x_n ani nevykoná. To zaručí algoritmus 1. Algoritmus hľadá inštrukciu v XT strome, ktorej asociovaný vstup z d generuje podstrom v pohľade obsahujúci n a následne zmaže prislúchajúcu časť z d. Algoritmus sa pokúša zmazať čo najmenej, postupom v strome XT zdola nahor. Algoritmus určite skončí, v najhoršom prípade ak sa dostane do koreňa XT stromu. V takom prípade sa zmaže celý dokument, čo môže byť nečakané alebo nechcené používateľom, ale túto operáciu potom môže klasicky vrátiť pomocou undo vykonávaným nad dokumentom.

Algorithm 1 Algoritmus pre spätnú transformáciu pre operáciu zmazania na pohľade

Require: dokument d reprezentovaný stromom; XSLT transformácia t, XT strom xt pre t, d; pohľad $v = t(d)$ reprezentovaný stromom; cesta *path* v pohľade definujúca vrchol, ktorý má byť zmazaný a pre operáciu zmazania textu naviac index *from* a index *to*

Ensure: d' - modifikovaný dokument d, zaručujúci operáciu zmazania

1: $n \leftarrow$ vrchol na ceste *path* v pohľade v
2: $x_n \leftarrow$ vrchol v XT strome xt generujúci n
3: **while** *true* **do**
4: **if** x_n je typu <xsl:element>, <xsl:attribute>, <xsl:text>, <xsl:processing-instruction>, <xsl:comment>, <xsl:number> alebo <xsl:copy> **then**
5: $x_n \leftarrow x_n.parent$
6: **else if** x_n je <xsl:copy-of> **then**
7: v d zmaž prislúchajúcu časť ku n
8: **return**
9: **else if** x_n je <xsl:value-of> **then**
10: {n je textový uzol}
11: v d zmaž prislúchajúcu časť ku n, *from* a *to*
12: **return**
13: **else if** x_n je typu <xsl:if> alebo <xsl:choose> **then**
14: $x_n \leftarrow x_n.parent$
15: **else if** x_n je <xsl:for-each> alebo <xsl:apply-templates> **then**
16: {inštrukcia pracuje s množinou uzlov z d}
17: v d zmaž uzol z množiny pre inštrukciu použitý na vygenerovanie n
18: **return**
19: **else if** x_n je <xsl:template> **then**
20: $x_n \leftarrow x_n.parent$
21: **else**
22: $x_n \leftarrow x_n.parent$ {ľubovoľná iná XSLT inštrukcia, preskočíme ju}
23: **end if**
24: **end while**

Algoritmus využíva inštrukcie spomenuté v jeho jednotlivých vetvách podmienok s výnimkou neinjektívnych inštrukcií xsl:if a xsl:choose, lebo ich nevieme využiť. Algotimus vie ošetriť všetky typy inštrukcií a to ich prípadným preskočením na rodiča.

4.2.2 Operácia modifikácie na pohľade

Operácia modifikácie v technike editácie pohľadu riadenou gramatikou s výberom má dva kroky. Za prvé, poskytnutie čiastočného oboru hodnôt používateľovi. Za druhé vykonanie samotnej operácie modifikácie.

Najprv sa pozrieme bližšie na sémantiku operácie modifikácie a následne analyzujeme ako vznikajú textové hodnoty v XSLT.

Operácia modifikácie na pohľade v má nasledovný tvar.

$$modify(v, path, newValue) \rightarrow v', \text{ kde } v, v' \in V, path \in P, newValue \in String$$

Je to operácia modifikujúca textový reťazec. V závislosti na type uzla určeného cestou *path* má operácia rôzny význam:

- element - neuvažujeme z dôvodu nezvyklosti operácie, logicky by takáto operácia mohla mať význam, lebo meno elementu môže byť dynamicky generovaný
- atribút - modikuje sa textová hodnota atribútu (modifikáciu mena atribútu neuvažujeme z dôvodu nezvyklosti operácie)
- text - najčastejšie využitie, modikuje sa textová hodnota
- vykonateľná inštrukcia - modikuje sa textová hodnota inštrukcie (modifikáciu mena inštrukcie neuvažujeme z dôvodu nezvyklosti operácie)
- komentár - modikuje sa textová hodnota
- referencia na entitu - tento prípad neuvažujeme

Textové reťazce pomocou XSLT môžu vzniknúť nasledovnými spôsobmi:

- sú pevne zadané - sú zapísané priamo v XSLT transformácii alebo cez <xsl:text>, v tomto prípade nie sú modifikovateľné
- skopírovaním - <xsl:copy>, <xsl:copy-of> inštrukciami
- vyhodnotením výrazu - <xsl:value-of> inštrukciou alebo výraz v kučeravej zátvorke alebo <xsl:number>
- kombináciou(konkatenáciou) predošlých

Uvažujme normálny tvar XSLT transformácie, kde pevne zadané textové reťazce sú zadané pomocou <xsl:text> inštrukcie. Výrazmi v kučeravej zátvorke sa nezaoberáme pri operácii modifikácie, kedže ich je možné použiť iba pre špecifikáciu mena elementu, atribútu alebo vykonateľnej inštrukcie. Prípad keď je použitá inštrukcia <xsl:number> nie je obojsmerný, čiže nepodporuje modifikáciu. Teda textový reťazec uvažovaný pre operáciu modifikácie s daným výrazovým obmedením a normálnym tvarom vzniká iba pomocou inštrukcií: <xsl:text>, <xsl:copy>, <xsl:copy-of>, <xsl:value-of>.

Výpočet čiastočného oboru hodnôt

Ako už bolo spomenuté, je vhodné poskytnúť používateľovi množinu povolených hodnôt v pohľade vzhľadom na transformáciu a gramatiku. Predpokladajme, že pre textové reťazce a atribúty nachádzajúce sa v dokumente d poznáme podľa asociovanej gramatiky obor hodnôt. Rozlišujeme iba dva prípady. Ak je konečný nech je zapísaný cez množinu - "{ ... }", v prípade nekonečného oboru nech ho reprezentuje "{*}". Tento zápis budeme používať aj pre výstup algoritmov.

Algoritmus 2 vypočítava podmnožinu oboru hodnôt pre spomenuté XSLT inštrukcie, ktoré sú obojsmerné, ale uvažuje aj neobojsmernú funkciu konkatenáciu reťazcov [6]. Využíva pritom algoritmus 3 na získanie množiny hodnôt pre XPath výraz. Uvažovaný XPath výraz musí byť obojsmerný a nesmie obsahovať predikáty.

[6] iba v jednoznačnom prípade

Algorithm 2 Algoritmus XSLTValueSet pre získanie podmnožiny povolených hodnôt pre operáciu modifikácie na pohľade

Require: dokument d reprezentovaný stromom; XSLT transformácia t, XT strom xt pre t, d; pohľad $v = t(d)$ reprezentovaný stromom; cesta *path* v pohľade definujúca vrchol, ktorý má byť modifikovaný, pre každý vrchol $z \in d$ množinu M_z povolených hodnôt asociovanou gramatikou (ak je konečná, tak reťazec "{ ...}", inak "{*}")

Ensure: M - podmnožina množiny všetkých možných prípustných hodnôt gramatikou a transformáciou, na ktorú môže byť vrchol určený cestou *path* modifikovaný v tvare $\{\}\{\} \ldots \{\}$

$n \leftarrow$ vrchol na ceste *path* v pohľade v
if n je atribút **then**
    nájdi prislúchajúce vrcholy $x_{n1}, x_{n2}, \ldots x_{nk}$ v XT strome xt generujúce hodnotu atribútu n
else
    {n je komentár, vykonateľná inštrukcia alebo textový uzol}
    nájdi prislúchajúce vrcholy $x_{n1}, x_{n2}, \ldots x_{nk}$ v XT strome xt generujúce textovú hodnotu n
end if
if $x_{n1}, x_{n2}, \ldots x_{nk}$ obsahuje viac ako 1 uzol typu <xsl:copy>, <xsl:copy-of> alebo <xsl:value-of> **then**
    return NotModifiable {z dôvodu neobojsmernosti funkciu konkatenácie reťazcov}
end if
for $i = 1$ to k **do**
    if x_{ni} je typu <xsl:text> **then**
        {hodnota je premenlivá, ale nevieme zistiť množinu hodnôt}
        $r_i \leftarrow$ {"vyhodnotený obsah inštrukcie x_{ni}"}
    else if x_{ni} je typu <xsl:copy> alebo <xsl:copy-of> **then**
        $z \leftarrow$ vrchol z d asociovaný s x_{ni}
        $r_i \leftarrow M_z$ {M_z - množina povolených hodnôt pre vrchol z}
    else if x_{ni} je typu <xsl:value-of> **then**
        expr $\leftarrow$ výraz select reprezentovaný abstraktným syntaktickým stromom {<xsl:value-of> inštrukcia pracuje so select XPath výrazom}
        $r_i \leftarrow XPathValueSet(expr)$
        if r_i je reťazec alebo číslo **then**
            return $string(r_i)$ {pretypovanie na reťazec}
        else
            return $r_i.first.second$ {}r_i je množina uzlov, vráti hodnotu prvého uzla - pretypovanie na reťazec
        end if
    else if x_{ni} je typu <xsl:number> **then**
        {hodnota je premenlivá, ale nevieme zistiť množinu hodnôt}
        $r_i \leftarrow$ {"vyhodnotený obsah inštrukcie x_{ni}"}
    end if
end for
return $r_1 \oplus r_2 \oplus \ldots \oplus r_k$

Algorithm 3 Algoritmus XPathValueSet pre získanie podmnožiny povolených hodnôt pre XPath výraz

Require: dokument d reprezentovaný stromom; XSLT transformácia t, XT strom xt pre t, d; pohľad $v = t(d)$ reprezentovaný stromom; pre každý vrchol z z d množinu M_z povolených hodnôt asociovanou gramatikou (ak je konečná, tak reťazec "{ ... }", inak "{*}"), XPath výraz *expr*

Ensure: Ak sa XPath výraz *expr* vyhodnotí cez XPathVal na

- množinu uzlov - zoznam dvojíc, uzol a s ním asociovaná podmnožina množiny všetkých možných prípustných hodnôt gramatikou a transformáciou pre daný uzol
- textový reťazec - podmnožina množiny všetkých možných prípustných hodnôt gramatikou a transformáciou

1: $ep \leftarrow expr$ {ep je ukazovateľ v strome expr}
2: **if** $ep \in \{|, and, or, !=, =, <, >, <=, >=, +, -, div, mod, s\}$ **then**
3: {hodnota je premenlivá, ale nevieme zistiť množinu hodnôt}
4: **return** {"XPathVal(ep)"}
5: **else if** ep ukazuje na referenciu na premennú x **then**
6: vstrome XT nájdi XT-param uzol x_param viažuci premennú x
7: **return** XPathValueSet(x_param.expr)
8: **else if** ep ukazuje na reťazec s **then**
9: **return** $\{s\}$
10: **else if** ep je obojsmerná XPath funkcia f s argumentami $a_1, a_2, \ldots a_l$ **then**
11: **for** $i = 1$ to l **do**
12: $A_i \leftarrow$ XPathValueSet(a_i)
13: **end for**
14: **if** $\exists i, A_i$ je $*$ **then**
15: **return** "*"
16: **else**
17: **return** $\{s | s = f(b_1, b_2, \ldots b_l), b_i \in A_i$
18: **end if**
19: **else if** ep ukazuje na cestu obsahujúcu predikáty **then**
20: {hodnota je premenlivá, ale nevieme zistiť množinu hodnôt}
21: $nodes \leftarrow$ XPathVal(ep)
22: $result \leftarrow$ prázny zoznam
23: **for all** $z \in nodes$ **do**
24: $result.Add((z, z.value))$
25: **end for**
26: **return** $result$
27: **else if** ep ukazuje na cestu neobsahujúcu predikáty **then**
28: $nodes \leftarrow$ XPathVal(ep)
29: $result \leftarrow$ prázny zoznam
30: **for all** $z \in nodes$ **do**
31: $result.Add((z, M_z))$
32: **end for**
33: **return** $result$
34: **end if**

Spätná transformácia

Definujeme spätnú transformáciu, ktorá prekladá operáciu modifikácie pohľadu na operáciu modifikácie dokumentu. Túto spätnú transformáciu sme zapísali pomocou algoritmu 4. Algoritmus v prípade hodnoty mino podporovaného oboru hodnôt, vyhlási chybu ErrorValue. V prípade zistenia, že hodnota nie je modifikovateľná týmto prístupom, vyhlási NotModifiable. Algoritmus uvažuje iba xslt inštrukcie <xsl:copy>, <xsl:copy-of> a <xsl:value-of>. V prípade, ak reťazec vzniká pomocou inštrukcie <xsl:text> je táto hodnota nemenná, lebo uvažujeme v tele inštrukcie iba pevnú hodnotu.

Algorithm 4 Algoritmus BidiXSLTModify pre spätnú transformáciu pre operáciu modifikácie na pohľade

Require: dokument d reprezentovaný stromom; XSLT transformácia t, XT strom xt pre t, d; pohľad $v = t(d)$ reprezentovaný stromom; cesta $path$ v pohľade definujúca vrchol, ktorý má byť modifikovaný a hodnota $value$ na ktorú má byť modifikovaný

Ensure: d' - modifikovaný dokument d, zaručujúci operáciu modifikácie, alebo chyba ErrorValue ak $value$ je mimo podporovaného oboru hodnôt, alebo NotModifiable

1: $n \leftarrow$ vrchol na ceste $path$ v pohľade v
2: **if** n je atribút **then**
3: nájdi prislúchajúce vrcholy $x_{n1}, x_{n2}, \ldots x_{nk}$ v XT strome xt generujúce hodnotu atribútu n
4: **else**
n je komentár alebo vykonateľná inštrukcia alebo textový uzol nájdi prislúchajúce vrcholy $x_{n1}, x_{n2}, \ldots x_{nk}$ v XT strome xt generujúce textovú hodnotu n
5: 6: **end if**
7: **if** $x_{n1}, x_{n2}, \ldots x_{nk}$ obsahuje viac ako 1 uzol typu <xsl:copy>, <xsl:copy-of> alebo <xsl:value-of> **then**
8: **return** NotModifiable {z dôvodu neobojsmernosti funkciu konkatenácie reťazcov}
9: **end if**
10: $value$ rozdeľ na hodnoty $v_1, v_2, \ldots v_k$ prislúchajúce k $x_{n1}, x_{n2}, \ldots x_{nk}$ {existuje práve 1 také rozdelenie}
11: **for** $i = 1$ to k **do**
12: **if** x_{ni} je typu <xsl:copy> alebo <xsl:copy-of> **then**
13: $z \leftarrow$ vrchol z d asociovaný s x_{ni}
14: $z.value = v_i$
15: **else if** x_{ni} je typu <xsl:value-of> **then**
16: $expr \leftarrow$ výraz select reprezentovaný abstraktným syntaktickým stromom {<xsl:value-of> inštrukcia pracuje so select XPath výrazom}
17: $BidiXPathModify(expr, value)$
18: **end if**
19: **end for**

Algorithm 5 Algoritmus BidiXPathModify pre získanie podmnožiny povolených hodnôt pre XPath výraz

Require: dokument d reprezentovaný stromom; XSLT transformácia t, XT strom xt pre t, d; pohľad $v = t(d)$ reprezentovaný stromom; XPath výraz $expr$ a hodnota $value$, ktorá má byť použitá pre modifikáciu $expr$

Ensure: d' - modifikovaný dokument d, zaručujúci operáciu modifikácie, alebo chyba ErrorValue ak $value$ je mimo podporovaného oboru hodnôt, alebo NotModifiable

1: $ep \leftarrow expr$ {ep je ukazovateľ v strome expr}
2: **if** $ep \in \{|, and, or, !=, =, <, >, <=, >=, +, -, div, mod, s\}$ **then**
3: {hodnota je premenlivá, ale nie je modifikovateľná, lebo nie je injektívna}
4: **return** NotModifiable
5: **else if** ep ukazuje na referenciu na premennú x **then**
6: vstrome XT nájdi XT-param uzol x_param viažuci premennú x
7: BidiXPathModify(x_param.expr, $value$)
8: **else if** ep ukazuje na reťazec s a $s! = value$ **then**
9: **return** ErrorValue
10: **else if** ep je obojsmerná XPath funkcia f s l argumentami a so spätnou funkciou g **then**
11: $a_1, a_2, \ldots a_l \leftarrow g(XPathVal(ep), value)$
12: **for** $i = 1$ to l **do**
13: $BidiXPathModify(ep.child(i), a_i)$
14: **end for**
15: **else if** ep ukazuje na cestu **then** {môže obsahovať aj predikáty}
16: $nodes \leftarrow$ XPathVal(ep)
17: $nodes = value$ {value musí byť typu node-set }
18: **end if**

4.2.3 Operácia vkladania na pohľade

Operácia vkladania v technike editácie pohľadu riadenou gramatikou s výberom má dva kroky. Za prvé, poskytnutie čiastočného oboru hodnôt používateľovi. Za druhé vykonanie samotnej operácie vloženia.

Najprv sa pozrieme bližšie na sémantiku operácie vloženia. Následne analyzujeme ako môže vzniknúť nový uzol operáciou vloženia v pohľade definovanom XSLT transformáciou. Nakoniec navrhneme algoritmus pre spätnú transformáciu pre operáciu vkladania na pohľade definovanom XSLT.

Nepodarilo sa nám navrhnúť riešenie pre výpočet čiastočného oboru hodnôt pre operáciu vkladania na pohľade. Tento problém je komplexný a jeho riešenie je zložitejšie ako samotná spätná transformácia pre operáciu vkladania. Preto sme sa tomuto problému nevenujeme a nechali sme ho otvorený.

Operácia vloženia na pohľade v má nasledovný tvar.

$$insert(d, path, index, tree) \rightarrow d', \text{ kde } d, tree, d' \in D, path \in P, index \in Int$$

Je to operácia vkladajúca strom uzlov do pohľadu na miesto určené cestou $path, index$. My sme navrhli postup pre jednoduchšiu operáciu vloženia, vkladajúcu jeden uzol, nasledovného tvaru.

$$insert(d, path, index, node) \rightarrow d', \text{ kde } d, node \in Nodes d' \in D, path \in P, index \in Int$$

Samozrejme uvažujeme intuitívnu sémantiku operácie vloženia. Podľa typu uzla, ktorý vkladáme, vyžadujeme nasledovné informácie získané z *node*:

- element - kvalifikované meno elementu
- atribút - kvalifikované meno atribútu a hodnotu atribútu
- textový uzol - textovú hodnotu uzla
- vykonateľná inštrukcia - meno inštrukcie a dáta
- komentár - text komontáru
- entita - tento prípad neuvažujeme

Analýza vzniku nového uzla v pohľade

Nech je daný XML dokument d a XSLT transformácia t. Pohľad v je získaný vykonaním transformácie t, pričom budujeme aj výpočtový strom XT xt. Podľa argumentov operácie *path* a *index* určujúcich pozíciu pre vloženie nájdeme v pohľade dva uzly p a q:

- ak uzol určený cestou *path* je textový uzol x, tak $p = q = n$
- ak uzol určený cestou *path* je element n, tak $p = n.child(index - 1)$ a $q = n.child(index)$ [7]

V strome XT k týmto uzlom p a q nájdeme uzly x_p a x_q a teda aj XSLT inštrukcie pomocou ktorých uzly p a q vznikly. V prípade textových uzlov ošetrujeme prípady spájania textových uzlov XSLT transformáciou. Ak uzol určený cestou *path* je

- textový uzol n, vkladáme do uzla n[8] na pozíciu *index*, tak x_p je uzol generujúci znak na pozícii $index - 1$ v uzle n a x_q je uzol generujúci znak na pozícii *index* v uzle n
- je element, vkladáme uzol medzi p a q, tak x_p je uzol asociovaný s p[9] a x_q je uzol asociovaný s q[10]

Aby vznikol uzol na požadovanej pozícii, tak by musel byť vygenerovaný XSLT transformáciou po vykonaní[11] inštrukcie x_p a pred vykonaním inštrukcie x_q. Jedinou výnimkou je prípad ak x_p resp. x_q je inštrukcia xsl:copy-of a vtedy *node* môže vzniknúť aj z x_p resp. x_q. Ak sa $x_p = x_q$, tak $x_p = x_q$ je typu xsl:copy-of. Medzi x_p a x_q v strome XT v smere výpočtu transformácie[12] môže byť veľa inštrukcií, ale žiadna generujúca výstup. Naším cieľom je modifikovať dokument d, tak

[7] v špeciálnych prípadoch: ak $index == 0$, tak $p = n$; ak $index == n.childCount$, nerozvádzame
[8] samozrejme, že uzlo vkladáme ako dieťa uzla $n.parent$
[9] ak p je textový uzol, tak x_p je uzol generujúci jeho poslednú časť
[10] ak q je textový uzol, tak x_q je uzol generujúci jeho prvú časť
[11] treba si uvedomiť že x_p, x_q sú výstup generujúce inštrukcie
[12] výpočet transformácie v strome postupuje podľa prehladávania do hĺbky

aby výpočet medzi x_p a x_q vygeneroval uzol *node*. Dosiahnuť tento cieľ nám pomôžu inštrukcie typu: *xsl:copy-of, xsl:apply-templates, xsl:for-each, xsl:if, xsl:choose a xsl:value-of*. Sú to riadiace inštrukcie, ktoré môžu na základe zmien v dokumente d pozmeniť výpočet a inštrukcie, ktorých výstup môže byť pozmenený podľa modifikácií v d. V inštrukciách povolíme iba obojsmerné XPath výrazy bez predikátov[13]a bez funkcií. Teda nevieme využiť inštrukcie xsl:if a xsl:choose, keďže využívajú podmienky. Spätná transformácia založená na zvyšných štyroch inštrukciách pre operáciu modifikácie pohľadu je popísaná algoritmom 6.

Spätná transformácia

Navrhli sme algoritmus pre spätnú transformáciu pre operáciu vkladania na pohľade. Tento algoritmus naväzuje na predchádzajúcu analýzu a pracuje už s daným kontextom. Algoritmus hodnotíme ako jednostupňový v zmysle, že sa snaží pozmeniť výpočet najviac do hĺbky 1 a to v prípade inštrukcií xsl:apply-templates a xsl:for-each. Dá sa diskutovať o správnosti algoritmu. Problémom je, že v niektorých prípadoch, keď sa modifikuje dokument a algoritmus skočí, nedosiahne sa požadovanej operácie. Je to spôsobené zmenou výpočtu ešte pred uzlom x_p. V iných prípadoch algoritmus môže dosiahnuť požadovanej operácie, čo ale môže spôsobiť zmeny v pohľade nežiadúce pre používateľa. V takom prípade to neznamená nesprávnosť algoritmu a používateľ môže operáciu vrátiť pomocou undo vykonávaným nad dokumentom.

Algoritmus pracuje v dvoch fázach. V oboch fázach špeciálne ošetruje prípad, keď sa preskúmava hraničný uzol x_p alebo x_q. V prvej fáze sa algoritmus snaží dosianuť operáciu, pozmenením dokumentu tak, aby niektorá výstupná inštrukcia pôvodne negenerujúca nič, vygenerovala uzol *node*. Do druhej fázi sa dostáva, ak prvá bola neúspešná. V druhej fáze sa algoritmus snaží dosianuť operáciu, pozmenením dokumentu tak, aby niektorá riadiaca inštrukcia xsl:apply-templates alebo xsl:for-each bola aplikovaná na ešte jeden uzol na **správnom mieste**. Potom sleduje výstup a ak sa vygeneruje požadovaný uzol, skončí. Navrhli sme ho ako dvojfázový, pretože prvá fáza výpočet nepredlžuje narozdiel od druhej fázy. V druhej fáze sa môže výpočet dostať dokonca do rekurzie.

Metóda next uzla v strome XT vráti nasledujúci uzol v smere prehľadávania do hĺbky, ak ešte nebol navštívený. BidiXPathInsert(expr, node, position) je funkcia modifikujúca dokument d tak, aby sa XPath výraz expr vyhodnotil na o jeden viac uzol typu node na pozícii position. Expr je obojsmerný XPath výraz neobsahujúci predikáty ani funkcie. Node je buď uzol alebo ANY. Position je START, END, číslo, alebo ANY. Funkcia BidiXPathInsert vráti TRUE ak bola úspešná a dokument d bol modifikovaný, inak vráti FALSE. Keďže expr je obojsmerný XPath výraz neobsahujúci predikáty ani funkcie, tak ho tvoria referencie obojsmerných premenných, reťazce a cesty. Funkciu sme detailne nenavrhli, a jej presný návrh nechávame na budúce práce. Myslíme si, že jej čo najlepšia implementácia, si vyžaduje rozlišovať jednotlivé prípady axix a zvoliť správnu stratégiu modifikácií, aby bola efektívna a dosiahla požadovaného cieľa, ak je to možné. My uvažujeme naivnú stratégiu, nedosahujúcu cieľ vždy keď je to možné. Funkcia rozlišuje dva prípady. V prvom prípade ak sa *expr* nad dokumentom d vyhodnotí na neprázdnu množinu uzlov, BidiXPathInsert pridá uzol do dokumentu d, tak aby bol na požadovanej pozícii a vyhovoval poslednému úseku cesty podľa vyhodnotenia *expr* po predposlednný úsek. V druhom prípade funkcia pracuje po úsekoch cesty. Pre úseky 1, 2, ... k-1 cesty dĺžky k zabezpečí, aby sa

[13]lebo nevieme akú modifikáciu treba na d. vykonať, aby podmienka (neinjektívna funkcia) začala resp. prestala platiť

vyhodnotili aspoň na jeden uzol, ak sa tak nedeje. Dosiahneme to postupne pre prvý úsek, potom pre druhý úsek, ..., až pre úsek k-1 vložením uzla vyhovujúceho danému úseku do dokumentu d'. Pre posledný úsek cesty zabezpečí, aby sa vyhodnotil na uzol *node* a to tak, že ho vloží na správne miesto do dokumentu d'.

Algorithm 6 Algoritmus BidiXSLTInsert pre spätnú transformáciu pre operáciu vkladania na pohľade

Require: dokument d reprezentovaný stromom; XSLT transformácia t, XT strom xt pre t, d; pohľad $v = t(d)$ reprezentovaný stromom; uzly x_p a x_q v strome XT a uzol *node*, ktorý má byť vložený v pohľade medzi výstup uzlov x_p a x_q

Ensure: vráti TRUE a modifikuje dokument d na d' zaručujúci operáciu modifikácie, alebo FALSE

1: **if** $x_q = x_p$ **then** {triviálny prípad inštrukcie copy-of}
2: vlož *node* na správne miesto {jednoduchá operácia, keďže copy-of je identita}
3: **return true**
4: **end if**
5: {fáza 1}
6: $calc \leftarrow x_p$ {calc je referencia na uzol v strome XT, s ktorým algoritmus pracuje}
7: **while** $calc! = x_q.next$ **do**
8: *position* $\leftarrow$ *START* ak $calc == x_p$, *END* ak $calc == x_q$, *ANY* inak
9: **if** *calc* je typu <xsl:copy-of select=ëxpr"> a BidiXPathInsert(expr, *node*, *position*) **then**
10: **return true**
11: **else if** *calc* je typu <xsl:value-of select=ëxpr"> a *node* je textový uzol a BidiXPathInsert(expr, *node*, *position*) **then**
12: **return true**
13: **end if**
14: $calc \leftarrow calc.next$
15: **end while**
16: {fáza 2}
17: $countFlowInstr \leftarrow$ počet riadiacich inštrukcií medzi x_p a x_q
18: **for** $i = 1$ to $countFlowInstr$ **do**
19: $calc \leftarrow$ i-tu riadiacu inštrukciu medzi x_p a x_q
20: **if** *calc* je typu <xsl:apply-templates select=ëxpr», <xsl:for-each select=ëxpr» **then**
21: $d.mark$
22: $child_with_p \leftarrow$ dieťa *calc* obsahujúce v podstrome x_p
23: $position \leftarrow calc.childIndex(child_with_p) + 1$
24: **if** BidiXPathInsert(expr, ANY, *position*) **then**
25: *output* $\leftarrow$ vyhodnoť inštrukciu *calc* pre uzol s indexom *position* vyhovujúci expr nad dokumentom d'
26: **if** *output* == *node* **then**
27: **return true**
28: **else**
29: $d.undo$ {operácia nebola úspešná, a tak modifikácie sú vrátené späť}
30: **end if**
31: **end if**
32: **end if**
33: **end for**

Kapitola 5

Porovnanie transformačných modelov/jazykov pre techniku editácie pohľadu

V tejto kapitole porovnávame transformačné modely navrhnuté alebo upravené pre techniku editácie pohľadu. Porovnávame obojsmerné XSLT/XPath transformácie s transformáciami vo funkcionálnom jazyku X a s transformáciami definovanými obojsmernými šablónami. Ako kritéria pre porovnanie sme zvolili výrazovú silu transformácie a podporu editácie pohľadu riadenú gramatikou s výberom.

5.1 Výrazová sila

Formálne porovnanie modelov nie je cieľom tejto práce, bolo by veľmi rozsiahle a nemáme aparát na takéto porovnanie. Preto modely porovnávame iba podľa kritérií adresovateľnosti vo vstupnom dokumente, schopnosti vygenerovať uzol na ľubovoľnom mieste vo výstupnom dokumente a množiny funkcií použiteľných pre výpočet transformácie. Kritériom adresovateľnosti vo vstupnom dokument rozumieme, množinu uzlov zo vstupného dokumentu ktoré model dokáže vyžiť(adresovať) pre vygenerovanie uzla vo výstupnom dokumente.

5.1.1 XSLT/XPath transformácie

Obojsmerné XSLT/XPath transformácie bez predikátov a funkcií podporujú všetky editačné operácie na pohľade. Takýto model vie pristupovať k ľubovoľnej časti vstupného dokumentu. Z charakteru generovania výstupu v poradí dokumentu resp. do hĺbky, dokáže vygenerovať uzol na ľubovoľnom mieste vo výstupnom dokumente. Nedisponuje však funkciami, pri operácii vkladania. Pre operácie zmazania a modifikácie, obsahuje všetky obojsmerné funkcie a aj predikáty.

5.1.2 Transformácie definované obojsmernými šablónami

Transformácie popísané obojsmernými šablónami podporujú všetky editačné operácie na pohľade. Model vie využiť jednu cestu od koreňa k uzlu vo vstupnom dokumente pre vygenerovanie uzla vo

Obojsmerné šablóny	XSLT/XPath	jaxyk X
plná podpora	čiastočná podpora	?

Tabuľka 5.1: Podpora techniky editácie pohľadu riadenej gramatikou s výberom

výstupnom dokumente. Z charakteru generovania výstupu podľa prehľadávania do hĺbky na vstupnom dokumente a adresovateľnosti vo vstupnom a výstupnom dokumente iba podľa štruktúry a to iba podľa jednej cesty vyplýva, že nedokáže zmeniť poradie susedných uzlov vo vstupnom dokumente. Teda nedokáže implementovať jednoduchú funkciu akou je výmena dvoch susedných uzlov vo vstupnom dokumente. Model disponuje ľubovoľnými obojsmernými funkciami.

5.1.3 Transformácie vo funkcionálnom jazyku X

Transformácie v jazyku X podporujú všetky editačné operácie na pohľade. Jazyk umožňuje adresovať celý dokument pre vygenerovanie uzla vo výstupnom dokumente. Model umožňuje vygenerovať uzol na ľubovoľnom mieste vo výstupnom dokumente. Jazyk disponuje ľubovoľnými obojsmernými funkciami.

5.2 Podpora editácie pohľadu riadenou gramatikou s výberom

Nevieme povedať ani o jednom modely, že by nepodporoval editáciu pohľadu riadenú gramatikou s výberom. Vieme iba povedať na základe prezentovaných algoritmov, ktoré modely takúto editáciu podporujú. Pod podporou tejto techniky rozumieme prípad, ak gramatika ohraničuje povolené hodnoty na konečné množiny. Za podporu tejto techniky však nepovažujeme, ak sa vykoná transformácia na a všetkých dokumentoch [1], a tak sa získa obor hodnôt. Preľad je v tabuľke 5.1.

[1] to je úplne nezmyselný prípad, lebo je to výpočtovo náročne a v praxi gramatiky pre xml dokumenty definujú nekonečné jazyky

Kapitola 6

Implementácia - projekt Euromath2

Projekt Euromath2 bol motiváciou pre túto prácu. Cieľom projektu je vytvoriť wysiwyg XML editor hlavne na matematické, ale aj technické a formálne dokumenty z rôznych oblastí. Editácia má byť kontextovo riadená, to znamená, že používateľovi sú prezentované editačné operácie platné v danom mieste (kontexte). Kvalita dokumentu musí byť profesionálna aspoň pri tlači. Celý editor má byť technologicky pokročilý. Editor sme nenavrhovali celý, ale nadväzovali sme na prácu iných ľudí. Dospeli sme pripráci k niektorím dôležitým postrehom a vyriešili sme niekoľko netriviálnych problémov pri návrhu editora.

6.1 Popis

Medzi základnú funkcionalitu/požiadavky partí: editácia formátovaného textu, editácia matematiky, všeobecnosť, rozšíriteľnosť na ďalšie oblasti, vnáranie obsahu rôzneho typu do seba (napr. matematika vo formátovanom texte), profesionálny vzhľad dokumentu pri tlači, kontextové používateľské rozhranie umožňujúce rýchlu a prehľadnú prácu, multi-platformovosť.

6.1.1 Architektúra

Zúčastnili sme sa na návrhu architektúry editora, ktorú v krátkosti predstavíme. Architekrúra editora je založená na architektonickom vzore Model-View-Controller a drží sa modulárneho dizajnu. Editor ja založený na technike editácie dokumentu. Skladá sa zo 4 základných častí:

1. Model - Časť zaoberajúca sa validitou dokumentu podľa gramatiky, hlavne poskytujúca kontextové editačné operácie
2. Model – transformácia dokumentu pomocou XSLT riešiaca: transformovanie, mapovanie a delenie dokumentu podľa namespace-ov pre ďalšie spracovanie
3. Controller – časť riadiaca editáciu a GUI, všeobecná, pre text a pre rôzne iné domény
4. View – procesor schopný spracovať dokument v danom namespace a vykresliť ho. Je to napojiteľná časť funkcionality.

Vytvorením nezávislého modulu Controller, sme dosiahli flexibilitu Controllera a jeho jednoduchú rozšíriteľnosť. Na to sme nadviazali v podrobnoejšom návrhu.

6.2 Návrh Controllera, všeobecného editora

My sme podrobne navrhovali a implementovali tretiu časť editora - controller, nazývanú všeobecný editor. Controller definuje používateľské rozhranie a vykonáva editačné operácie. Tento controller využíva nižšiu vrstvu Model a jeho implementácia je v Jave a je založená na SWT[1] + JFace a GEF[2]. Je dostatočne všeobecný, aby bol použitý s ľubovoľným XML dokumentom a na druhej strane umožňuje špecializáciu pre konkrétnu doménu(príkladom je TextEditor) alebo jeden typ dokumentu(príkladom je FOEditor).

- **IEditor** je rozhranie controllera pre pre časť dokumentu, ktorá je z jedného namespace (domény). Spolupracuje s IRenderer-om, ktorý sa stará o grafickú reprezentáciu(renderovanie a aj formátovanie/layout) časti dokumentu a Modelom, ktorý mu poskytuje editačné operácie nad dokumentom. IEditor pracuje v dvoch fázach. V prvej inicializačnej fáze je jeho úlohov vytvoriť a inicializovať potrebné triedy na efektívnu modifikáciu dokumentu. V druhej fáze IEditor riadi editáciu a synchronizuje sa prostredníctvom triedy EditorSite s inými editormi. Po každej vykonanej editačnej operácii ním samým alebo iným IEditorom sa vygeneruje nový pohľad[3] a prebehne aktualizácia všetkých IRendererov a následne IEditorov. Takto dosiahneme aktuálnu grafických reprezentáciu celého dokumentu a k nej aktualizovaný Controller.
- **EditorSite** konštruuje hierarchiu IEditor-ov pre rozčlenený dokument podľa namespace-ov a pre hierarchiu IRendererov, ktorý dostáva od GENE-data providera a rozmiesňuje ich podľa pozícií získaných od IRendererov. Synchronizuje seleckie, editačné nástroje vytvorených IEditorov a niektoré prebiehajúce editačné akcie vyššej úrovne akou je napríklad akcia drag-and-drop.
- IEditor je rozhranie, sú dve všeobecné implementácie tohto rozhrania: XMLEditor a TextEditor, pre konkrétne domény (namespace) sa registruje aký IEditor sa má použiť, pričom môže sa využiť aj priamo jeden z nich
- **XMLEditor** je najvšeobecnejší IEditor postavený na GEF-e a umožňuje: selekcie elementov, editáciu pomocou klávesnice a cez menu(Toolbar a kontextové menu), clipboard, undo/redo, zoom.
- **TextEditor** je IEditor pre text. Dedí a rozširuje XMLEditor. Pridáva nástroj(TextTool) na selekcie textu a na písanie textu pomocou klávesnice.
- Document Outline je stromové zobrazenie dokumentu synchronizované so zvyškom editora

Návrh editora je zobrazený aj na obrázku B.1.

6.3 Dátové formáty editora

Wysiwyg editor je aplikácia k okamžitému použitiu a preto má definovať základnú sadu typov dokumentu pre ktorú je určený. Pre každý takýto základný typ dokumentu má obsahovať šablónu

[1] Standard widget toolkit
[2] Graphical Editing Framework
[3] v aktuálnej implementácii vykonaním celej transformácie

pre vytvorenie novej inštancie dokumentu, špecializované používateľské rozhranie a metodiku integrácie a využitia s ostatnými typmi dokumentov. V editore Euromath2 sme sa ešte nerozhodli pre základné dátové formáty, ktoré bude používateľ editovať, ale zvolili sme MathML[9] a XSL FO[14] ako prezentačné dátové formáty. XSL FO dokumenty majú profesionálnu formu a vzhľad na obrazovke aj pri tlači. MathML je jazyk pre matematické texty. Tieto dva formáty môžu byť skombinované a teda prezentovať textové dokumenty obsahujúce matematiku. Dátových formátov budeme podporovať viac, ale ako základný zvolíme používateľmi najobľúbenejší. Pomocou XSLT transformácie ich budeme transformovať do XSL FO + MathML pohľadu a teda stačí, aby editor Euromath2 obsahoval IRenderer-y pre tieto dva jazyky.

6.3.1 MathML

V editore Euromath 2 sme zvolili ako formát pre matematické texty a výrazy jazyk MathML. Je to jazyk vyvinutý W3C pre prezentáciu matematikých textov vo World Wide Web a je špecifikovaný v [9]. Matematické texty sa vyznačujú zložitou dvojrozmernou notáciou a veľkým počtom špecializovaných symbolov notácie. Je veľmi vhodné matematické texty editovať a prezentovať wysiswyg, práve kôli zložitosti zápisu. S týmto zámerom bolo MathML navrhnuté - pre wysiwyg editáciu, aj keď je to textový formát čitateľný človekom.

V technike editácie pohľadu by sme pre matematiku v pohľade zvolili používateľsky zrozumitelnejší jazyk ako je MathML, čím by sme umožnili jednoduchšiu implementáciu užívateľského prostredia. Ak by sme potrebovali editovať dokument obsahujúci MathML, tak by sme transformovali MathML do zvoleného vhodnejšieho formátu v pohľade.

V technike editácie dokumentu, v Euromath-e 2, sme MathML zvolili ako formát(prezentačný) v pohľade, hlavne z dôvodu ucelennosti formátu, štandardnosti a podpory renderovacích nástrojov. V dokumente je v tomto prípade vhodné použiť už spomínaný používateľsky zrozumitelnejší jazyk ako MathML.

Popísané dva prípady riešia otázku náročnosti pamätať si elementy jazyka MathML, keďže väčšina sú akronymi a taktiež nevhodnej štruktúry jazyka. Obe riešenia sú založené na transformácii, čím je umožnené použiť jednoduchšie používateľské rozhranie.

So zložitosťou matematických textou súvisí aj problém správnosti editovaného textu. Editovanú matematiku je nutné udržiavať validnú a vedieť ju validovať, tak aby bola syntakticky správna. Gramatiky Schema a DTD pre XML nedokážu zaručiť správnu syntax MathML, príkladom sú dátové typy (napr. pozitívne čísla) a kontextovosť (napr. prvé dieťa také, druhé také). Teda je nutné implementovať editačné operácie v editore na základe silnejšej gramatiky. Dá sa využiť gramatika definovaná W3C [8] a pouzívať parser na validáciu. Controller editora by mal povolovať iba operácie validné podľa danej gramatiky, napr. pri vložení ľavej zátvorky vloží aj pravú (zátvorka v MathML nie je reprezentovaná elementom).

Príklad MathML, vidieť názvy elementov - akronymi, zátvorky obalené samostatne elementom, veľkú hĺbku dokumentu:

```
<m:math display="block"><m:math display="block">
    <m:mi>x</m:mi>
    <m:mo>=</m:mo>
    <m:mrow>
    <m:mo>[</m:mo>
        <m:mn>32000</m:mn>
```

```
        <m:mo>(</m:mo>
        <m:mn>1</m:mn>
        <m:mo>+</m:mo>
        <m:mfrac linethickness="2">
            <m:mrow>
                <m:mfrac>
                    <m:mn>25</m:mn>
                    <m:mn>2</m:mn>
                </m:mfrac>
            </m:mrow>
            <m:mn>100</m:mn>
        </m:mfrac>
        <m:mo>)</m:mo>
        <m:mo>]</m:mo>
    </m:mrow>
</m:math>
```

$$x = [32000(1 + \frac{\frac{25}{2}}{100})]$$

Záver

V práci sme prezentovali najčastejšie wysiwyg editačné techniky s použitím XSLT alebo inej transformácie. Zaoberali sme sa hlavne technikou editácie pohľadu, analyzovali sme ju a navrhli sme jej realizáciu pre pohľad definovaný XSLT transformáciou. Taktiež sme definovali jej rozšírenú verziu, keď je editácia riadená gramatikou. Podľa dostupnej literatúry si myslíme, že sme sa tejto technike venovali ako prvý. Chceli sme ju realizovať pre nejaký existujúci transformačný jazyk, to sa nám však nepodarilo. Realizovali sme ju teda pre jazyk a model TOS, ktorý sme navrhli. Na záver sme priniesli porovanie modelov pre techniku editácie pohľadu a predstavili sme architektúru a časť návrhu editora Euromath2. Myslíme si, že editor disponuje veľmi dobrou architektúrov a a návrhom.

Ďalšia práca môže byť venovaná podrobnému návrhu algoritmu BidiXPathInsert alebo analýze a realizácii techniky editácie pohľadu riadenej gramatikou. Pri analýze techniky vidíme konkrétne úlohy v pokuse realizovať techniku nami nepreštudovaným modelom alebo rožírením prípadne úpravou modelu TOS. Taktiež implementácia čiastočnej XSLT transformácie pre projekt Euromath2 je výzvou a zaujímavým problémom. Budúcim prácam v tejto oblasti želáme veľa úspechov.

Literatúra

L. Villard and N. Layaida, *An incremental XSLT transformation processor for XML document manipulation.* (Proceedings of the 9th international conference on World Wide Web, pages 474-485, 2002)

Makoto Onizuka, Fong Yee Chan, Ryusuke Michigami and Takashi Honishi, *Incremental Maintenance for Materialized XPath/XSLT Views* (In Proceedings of the 14th international conference on World Wide Web, 671-681, 2005)

Shin-Cheng Mu, Zhenjiang Hu and Masato Takeichi, *An algebraic approach to bi-directional updating.* (Second Asian Symposium, APLAS 2004, Taipei, Taiwan, November 4-6, 2004)

Shin-Cheng Mu, Zhenjiang Hu and Masato Takeichi, *A programmable editor for developing structured documents based on bidirectional transformations.* (Higher-Order and Symbolic Computation, 1, 1-31 ,2006)

Shin-Cheng Mu, Zhenjiang Hu and Masato Takeichi, *Bidirectionalising HaXML.* (The Third Workshop on Programmable Structured Documents, Yokohama, Japan, January 26-28, 2005)

Shin-Cheng Mu, Zhenjiang Hu and Masato Takeichi, *An Injective Language for Reversible Computation* (The Second Workshop on Programmable Structured Documents, pages 15-35, 2004)

Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt, *A Language for BiDirectional Tree Transformations* (University of Pennsylvania CIS Dept. Technical Report, MS-CIS-03-08, August, 2003)

W3C, *MathML Content Markup Validation Grammar.* (http://www.w3.org/TR/MathML2/appendixb.html)

W3C, *Mathematical Markup Language (MathML) Version 2.0 (Second Edition)* (http://www.w3.org/TR/MathML2/)

W3C, *XSL Transformations (XSLT) Version 1.0.* (http://www.w3.org/TR/xslt)

W3C, *XML Path Language (XPath) Version 1.0* (http://www.w3.org/TR/xpath)

W3C, *XMLSchema specifications and development* (http://www.w3.org/XML/Schemadev)

W3schools, *W3schools DTD tutorial* (http://www.w3schools.com/dtd/default.asp)

W3C, *Extensible Stylesheet Language (XSL) Version 1.1* (http://www.w3.org/TR/xsl/)

FMFI UK, *Euromath2 editor projekt* (http://sourceforge.net/projects/euromath2, http://euromath2.sourceforge.net/)

Geert Jan Bex, Sebastian Maneth, Frank Neven, *A Formal Model for an Expressive Fragment of XSLT* (Computational Logic - CL 2000: First International Conference, London, UK, July 2000)

Makoto Murata, Dongwon Lee, Murali Mani, *Taxonomy of XML Schema Languages Using Formal Language Theory* (Extreme Markup Languages 2001® (Montréal, Québec))

Martin Vyšný, *Wysiwyg XML modification driven by schema* (Fakulta matematiky, fyzika a informatiky UK, diplomová práca, 2004)

Jana Dvořáková, *Transformácie XML dokumentov* (Fakulta matematiky, fyzika a informatiky UK, diplomová práca, 2004)

Witold Charatonik, Zoltan Fülöp, Werner Kuich, Markus Lohrey, Jun Matsuda, Aart Middeldorp, Hitoshi Ohsaki, P. K. Manivannan, Masahiko Sakai, Helmut Seidl, Stephan Tobies, Ralf Treinen, Thomas Uribe, Sandor Vágvölgyi, Kumar N. Verma, Toshiyuki Yamada, *Tree Automata Techniques and Applications* (A full version, October, 1rst 2002, http://www.grappa.univ-lille3.fr/tata/)

Dodatok A

XPath grammar

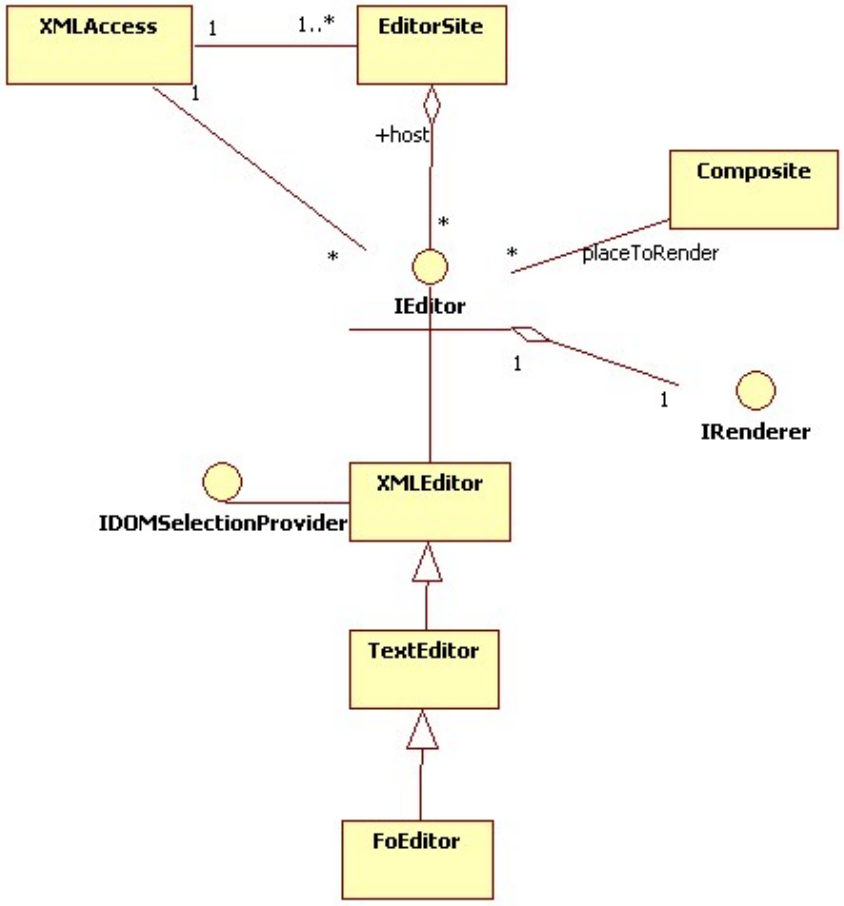
```
Expr ::= OrExpr
OrExpr ::= AndExpr | OrExpr 'or' AndExpr
AndExpr ::= EqualityExpr | AndExpr 'and' EqualityExpr
EqualityExpr ::= RelationalExpr | EqualityExpr '=' RelationalExpr
                 | EqualityExpr '!=' RelationalExpr
RelationalExpr ::= AdditiveExpr
                   | RelationalExpr '<' AdditiveExpr
                   | RelationalExpr '>' AdditiveExpr
                   | RelationalExpr '<=' AdditiveExpr
                   | RelationalExpr '>=' AdditiveExpr
AdditiveExpr ::= MultiplicativeExpr
                 | AdditiveExpr '+' MultiplicativeExpr
                 | AdditiveExpr '-' MultiplicativeExpr
MultiplicativeExpr ::= UnaryExpr
                       | MultiplicativeExpr MultiplyOperator UnaryExpr
                       | MultiplicativeExpr 'div' UnaryExpr
                       | MultiplicativeExpr 'mod' UnaryExpr
MultiplyOperator ::= '*'
UnaryExpr ::= UnionExpr | '-' UnaryExpr
UnionExpr ::= PathExpr | UnionExpr '|' PathExpr
\\ CESTA
PathExpr ::= LocationPath | FilterExpr
             | FilterExpr '/' RelativeLocationPath
             | FilterExpr '//' RelativeLocationPath
\\ADRESNÁ ČASŤ CESTY
LocationPath ::= RelativeLocationPath | AbsoluteLocationPath
AbsoluteLocationPath ::= '/' RelativeLocationPath?
                          | AbbreviatedAbsoluteLocationPath
AbbreviatedAbsoluteLocationPath ::= '//' RelativeLocationPath
RelativeLocationPath ::= Step | RelativeLocationPath '/' Step
                         | AbbreviatedRelativeLocationPath
AbbreviatedRelativeLocationPath ::= RelativeLocationPath '//' Step
```

```
Step ::= AxisSpecifier NodeTest Predicate* | AbbreviatedStep
AbbreviatedStep ::= '.' | '..'
AxisSpecifier ::= AxisName '::' | AbbreviatedAxisSpecifier
AbbreviatedAxisSpecifier ::= '@'?
AxisName ::= 'ancestor' | 'ancestor-or-self' | 'attribute' | 'child' | 'descendant'
             | 'descendant-or-self' | 'following' | 'following-sibling' | 'namespace'
             | 'parent' | 'preceding' | 'preceding-sibling' | 'self'
NodeTest ::= NameTest | NodeType '(' ')'
             | 'processing-instruction' '(' Literal ')'
NameTest ::= '*' | NCName ':' '*' | QName
NodeType ::= 'comment' | 'text' | 'processing-instruction' | 'node'
\\ FILTER
FilterExpr ::= PrimaryExpr | FilterExpr Predicate
\\ PREDIKÁT
Predicate ::= '[' PredicateExpr ']'
PredicateExpr ::= Expr
\\ PRIMÁRNY VÝRAZ
PrimaryExpr ::= VariableReference | '(' Expr ')' | Literal
                | Number | FunctionCall
VariableReference ::= '$' QName
Literal ::= '"' [^"]* '"' | "'" [^']* "'"
Number ::= Digits ('.' Digits?)? | '.' Digits
Digits ::= [0-9]+
FunctionCall ::= FunctionName '(' ( Argument ( ',' Argument )* )? ')'
FunctionName ::= QName - NodeType
Argument ::= Expr


ExprToken ::= '(' | ')' | '[' | ']' | '.' | '..' | '@' | ',' | '::'
                    | NameTest | NodeType | Operator | FunctionName
                    | AxisName | Literal | Number | VariableReference
Operator ::= OperatorName | MultiplyOperator | '/' | '//' | '|'
           | '+' | '-' | '=' | '!=' | '<' | '<=' | '>' | '>='
OperatorName ::= 'and' | 'or' | 'mod' | 'div'
ExprWhitespace ::= S
```

Dodatok B

Implementácia



Obrázok B.1: Návrch controllera v editore Euromath2

Obrázok B.2: Výkonostný profil editora Euromath2 pre 50 stranový dokument

Výkonostný profil formátovania a renderovania 50 stranového dokumentu v editore Euromat2. V profile je zvýraznená časť vyžadujúca väčšinu času výpočtu a je vidieť, že je to práve formátovanie a vykreslovanie.

Pre lepšiu orientáciu v profile, uvádzame vysvetlenie jednotlivých stĺpcov tabuľky.

Base Time Čas potrebný na vykonanie metódy(jej všetkých volaní), bez času na vykonanie metód zavolaných z tejto metódy

Average base time Priemerný čas na vykonanie 1 volania metódy

Cumulative base time Celkový čas potrebný na vykonanie metódy (jej všetkých volaní), vrátane času na vykonanie metód zavolaných z tejto metódy

Calls Počet volaní metódy