Vizualizace dat v relační databázi s využitím teorie formálních jazyků

Capeková, Zuzana
2006

Vizualizace dat v relační databázi s využitím teorie formálních jazyků

doktorand:
ING. ZUZANA CAPEKOVÁ
ÚI AVČR, Pod Vodárenskou věží 2, 182 07 Praha 8
FM TUL, Hálková 6, 461 17 Liberec 17
capekova@cs.cas.cz, zuzana.capekova@tul.cz

školitel:
ING. JÚLIUS ŠTULLER, CSC.
ÚI AVČR, Pod Vodárenskou věží 2, 182 07 Praha 8
stuller@cs.cas.cz

obor studia:
Technická kybernetika

Abstrakt

Článek se zabývá možností efektivního návrhu webových stránek pro vizualizaci dat v relačních databázích s využitím teorie formálních jazyků a překladů. Jazyk závorkových výrazů libovolné hloubky vnoření patří mezi bezkontextové jazyky. Jestliže omezíme hloubku vnoření závorek zvolenou konstantou, lze pro takový jazyk sestrojit regulární gramatiku, která jej generuje. Generování věty regulárního jazyka pomocí pravidel pravé regulární gramatiky se vyznačuje ”koncovou rekurzí” tj. větná forma obsahuje nejvýše jediný neterminální symbol, který je zároveň symbolem větné formy nejvíce vpravo. Výběr pravidla pro expanzi neterminálu je možné řídit informacemi získanými z relační databáze. Koncovou rekurzi lze snadno vyjádřit pomocí iterace. V dostatečně dlouhých větách všech bezkontextových jazyků lze na určitých místech zopakovat podřetězec dané věty tak, aby získaná věta patřila do jazyka. Kontextové jazyky podobnou vlastnost obecně nemají, nicméně i tam můžeme v izolovaných případech zopakovat na určitých místech části věty s identickým výsledkem. Pro získávání dat z relačních databází je možné definovat množinu SQL dotazů uspořádanou do stromu pomocí ”master-detail” referenčních vztahů. Na příkladu jazyka HTML ve spojení se závorkovou strukturou reprezentující strom dotazů konečné výšky je demonstrována možnost dynamického generování www stránek výhradně na základě deklarativní konfigurace s vyloučením programátorské práce.

1. Úvod

Současný rychlý rozvoj internetu klade zvýšené nároky na tvůrce webových stránek. Dnes již není možné spoléhat se pouze na statický webový obsah. Návrh www stránek v běžně užívaných systémech dynamického generování stránek (php, asp, jsp, Java-servlet apod.) však může úspěšně provádět jen odborník alespoň zhruba znalý problematiky imperativního a objektového programování. V této práci představujeme jeden ze způsobů, jak umožnit visualizaci dat z relační databáze pomocí dynamického generování www stránek výhradně na základě deklarativní konfigurace s možností dodatečně upravovat grafickou úpravu generovaných stránek s vyloučením programátorské práce. Dynamické generování HTML stránek můžeme považovat za speciální případ generování řetězců složených se symbolů alfanumerické abecedy. Zaměření této práce proto není omezeno čistě jen na generování HTML stránek, naopak s výjimkou konkrétních příkladů v HTML jsou všechny závěry práce aplikovatelné i na generování xml nebo rtf dokumentů. Prezentaci dat v podobě rtf dokumentu v kombinaci s široce rozšířeným editorem rtf dokumentů (Microsoft Word) lze též s výhodou využít jako alternativní systém tisku uživatelsky modifikovatelných tiskových sestav. Základním principem postupu prezentovaného v této práci je obohacení HTML (resp. rtf atd.) dokumentu o závorkovou strukturu reprezentující algoritmus získávání dat z relační databáze. S výhodou též využíváme vlastností jazyka HTML resp. rtf, zejména se jedná o možnost 0..n krát zopakovat ve vybraných větách jazyka určité podřetězce dané věty tak, že výsledná posloupnost symbolů zůstává větou jazyka. Dynamické generování dokumentů na základě uvedeného principu je možné výrazně zefektivnit, pakliže provedeme předzpracování dokumentu. Předzpracování dokumentu je možné provést s využitím teorie formálních překladů. Výsledkem předzpracování je binární vnitřní forma, která je vhodnější pro opakované generování dokumentu. Práce využívá výsledky teorie relačních databází [1, 5] a teorie formálních jazyků a

překladů [1, 2, 3]. Zápis gramatiky resp. grafické vyjádření konečných automatů je uvedeno v notaci podle [1, 2]. Přehled základních pojmů je uveden v [1, 2, 3].

2. Strom dotazů

Pro účely této práce definujeme strom dotazů nad množinou relací takto:

Definition 1 *N*

echť **R** je množina relací. Strom dotazů budiž orientovaný stromový graf ve kterém platí:

1. Hrany jsou orientovány ve směru od rodičovského uzlu k potomkům.
2. Každý uzel stromu je ohodnocen jedinou relací $r \in R$.
3. Každá hrana směřující od předka u_i ohodnoceného relací $r_i = \langle A_i, D_i, T_i \rangle$ k potomkovi u_j ohodnocenému relací $r_j = \langle A_j, D_j, T_j \rangle$je ohodnocena funkcí $f_j : T_i \times T_j \longrightarrow \{0, 1\}$. V souladu s rozšířenou definicí operace selekce v relační algebře podle[5], je funkce *f* tvořena logickým výrazem sestrojeným z logických spojek a porovnání hodnot atributů.

Stromový graf se vyznačuje m.j. tím, že počet jeho hran je vždy o jedna menší než počet uzlů a že od rodičovského uzlu k některému z přímých potomků vede vždy jen jedna jediná hrana. Můžeme proto označit uzly, hrany a jim přiřazené relace resp. funkce pomocí celých čísel takto:

- Uzly jsou označeny jednoznačně celými čísly *1..n*, kde *n* je počet uzlů grafu.
- Kořen stromu je označen číslem *1*.
- Hrana *h*, která vede od rodiče u_i k potomkovi u_j je označena číslem *j*.
- Symbolem r_i označíme relaci, kterou je ohodnocen uzel u_i.
- Symbolem f_j označíme funkci, kterou je ohodnocena hrana stromu h_j.
- Funkce f_j, kterou je ohodnocena hrana stromu h_j směřující od předka u_i k potomkovi u_j, určuje pro každou kombinaci prvků z relace r_i a r_j jejich obsahovou souvislost.
- Vstupem funkce je vždy jeden prvek z relace r_i a jeden prvek z relace r_j.
- Výstupem funkce je číslo *0*, pakliže vstupující prvky obsahově nesouvisí. V opačném případě funkce vrátí číslo *1*.

2.1. Traverzování stromu dotazu

Strom dotazů umožňuje zorganizovat pořadí zpracování prvků relací, kterými jsou ohodnoceny uzly stromu. Předpokládejme, že v průběhu traverzování je strom dotazů neměnný, tj. má konstantní množinu uzlů, množinu hran i incidenční zobrazení. Traverzování stromu a zpracování prvků relací nechť probíhá takto:

Pro každý prvek *t* relace r_i, omezené selekcí na podmínku $f_i = 1$, se navštíví všichni přímí následníci uzlu u_i ve stanoveném neměnném pořadí. Návštěva přímého potomka u_j je spojena s přechodem po hraně h_j ve směru její orientace a s provedením selekce na relaci r_j podle podmínky definované takto: $f_j(t, x) = 1$.

Prvek $t \in T_i$ představuje aktuálně zpracovávaný prvek *z* relace předka a je konstantní v průběhu provádění selekce, symbol *x* označuje po řadě jednotlivé prvky relace r_j, tj. platí $x \in T_j$. Pakliže je relace získaná selekcí prázdná, provede se návrat do rodičovského uzlu. V opačném případě se postup rekurzivně zopakuje.

V případě kořene u_1 se postupuje obdobně s tím rozdílem, že neproběhne selekce na relaci r_1 a po ukončení probírky všech prvků relace r_1 algoritmus skončí.

Uvedený postup umožňuje zpracovat (např.vypsat) všechny prvky relací přiřazených následníkům ("detail") obsahově související s právě zpracovávaným prvkem relace přiřazené rodičovskému uzlu ("master"). Až poté dojde ke zpracování dalšího prvku relace přiřazené rodičovskému uzlu. V okamžiku provedení selekce je jisté, že relace na úrovni rodičovského uzlu není prázdná, tj. selekční kritérium má zajištěny korektní vstupní hodnoty.

2.2. Vyjádření stromu dotazů pomocí jazyka závorkových výrazů

Pro vyjádření stromu dotazů pomocí jazyka závorkových výrazů můžeme využít atributované terminální symboly otevírací a uzavírací závorky a syntaxi jazyka formálně vyjádřit pomocí atributové gramatiky. Atributová gramatika je čtveřice *AG=(G,A,V,F)*, kde *G=(N,T,P,S)* je základní gramatika atributové gramatiky, *A* je množina atributů, *V* je zobrazení přiřazující každému symbolu z $N \bigcup T$ množinu jeho atributů, *F* je množina sémantických pravidel [2].

Předpokládejme, že strom dotazů obsahuje vždy alespoň jeden uzel. Pak můžeme sestrojit atributovou gramatiku takto:

1. Základní gramatika G = (N, T, P, S)
   $T = \{[,]\}$
   $N = \{S, Z\}$
   $P = \{S \longrightarrow [Z, Z \longrightarrow SZ, Z \longrightarrow]\}$
2. Množina atributů
   $A = \{node\_number\}$
3. Zobrazení V
   $V = \{([, \{node\_number\}), (], \{node\_number\}), (S, \{\}), (Z, \{\})\}$
4. Množina F - prázdná
   $F = \{\}$

Převod stromu dotazů na větu uvedeného jazyka provedeme rekurzivním průchodem přes uzly stromu. Při první návštěvě uzlu zapisujeme otevírací závorku s atributem nastaveným na číslo uzlu, pak rekurzivně projdeme potomky a před návratem k rodiči zapíšeme uzavírající závorku s atributem nastaveným na číslo uzlu. Takto získaný výraz označme pro účely tohoto textu symbolem "**VSD**". Uvedené vyjádření stromu dotazů není jednoznačné v tom smyslu, že pro kompletní zpětný převod je třeba ještě doplnit převodní zobrazení, které na základě čísla uzlu určí uzlu příslušející relaci a hraně (přicházející od rodiče) přiřadí příslušející funkci. Dále je pro tvar **VSD** výrazu podstatné pořadí procházení potomků. Neméně důležitý je převod zadaného řetězce na strom dotazů resp. kontrola zadaného atributovaného řetězce a zjištění, zda odpovídá zadanému stromu dotazů. Tento úkol řeší lexikální a syntaktická analýza. Snadno ověříme, že základní gramatika námi sestrojené atributové gramatiky je bezkontextová a navíc LL(1). Proto můžeme snadno sestrojit deterministický syntaktický analyzátor pomocí techniky rekurzivního sestupu [1, 2].

3. Jazyk závorkových výrazů

Zabývejme se nyní izolovaně jazykem závorkových výrazů *L*, který je obecně bezkontextový. Vyznačuje se existencí dvou symbolů v abecedě – levé a pravé závorky. V každé větě jazyka je celkově stejný počet pravých a levých závorek a navíc v libovolném prefixu věty je počet levých závorek větší nebo roven počtu pravých závorek. Jazyk závorek je např. generován gramatikou $G = (N, T, P, S)$

- $T = \{[,]\}$

- $N = \{S, Z\}$
- $P = \{S \longrightarrow \varepsilon, S \longrightarrow ZS, Z \longrightarrow [S]\}$

Uvedená gramatika *G* generuje nekonečný jazyk $L(G) = L$.

Symbolem $h(s)$ označme maximální hloubku vnoření závorek ve větě $s \in L(G)$, přičemž hloubku vnoření závorky definujeme obvyklým rekurzivním předpisem $h = h\_parent + 1$ hloubka závorky nikde nevnořené je 1, $h(\varepsilon) = 0$. Hloubka vnoření závorek je v každé izolované větě omezena, neboť věta jazyka má vždy konečný počet symbolů. Pro libovolné celé číslo $n \in N$ však vždy existuje věta $s \in L(G)$, taková že $h(s) = n$.

Zabývejme se nyní jazykem $\{L2 = s : s \in L \wedge h(s) \leq k\}$, kde $k \in N$ e zvolená konstanta. Jazyk *L2* je tvořen podmnožinou řetězců jazyka *L*, kde je maximální hloubka vnoření závorek $h(s)$ všech řetězců $s \in L2$ omezena zvolenou konstantou $k \in N$. Jazyk L2 patří mezi regulární jazyky a můžeme pro něj sestrojit konečný automat, který jej přijímá, resp. regulární gramatiku, která jej generuje.

I v případě, že povolíme více typů závorek, tj. uvažujeme více než jednu levou závorku a ke každé levé závorce přiřadíme jí odpovídající pravou závorku, dostaneme při konstantou omezené hloubce vnoření regulární jazyk popsatelný např. konečným automatem.

Vzpomeňme nyní na atributovaný závorkový výraz **VSD** z kapitoly 2.2, který jsme použili jako jednu z možností pro vyjádření stromu dotazů. Jelikož má konečnou hloubku, má smysl zabývat se sestrojením konečného automatu resp. regulární gramatiky generující jazyk, jehož věty jsou odvozeny od **VSD** výrazu konečným (i nulovým) počtem opakování závorky na daném místě v těsném sousedství s ostatními výskyty. Atribut "node_number" považujme nyní za typové označení závorek. Hledaný jazyk je tvořen řetězci nad abecedou $A = \{$"[1″, "1]″, "[2″, "2]″... "[n″, "n]″$\}$, kde hodnoty atributu se staly součástí symbolu abecedy. Dále požadujeme, aby závorku příslušející uzlu u_j (vyskytuje se ve výrazu jen jednou) bylo možno libovolně-krát zopakovat na daném místě v rámci závorky příslušející rodiči uzlu u_j.

Konečný automat přijímající uvedený jazyk sestrojíme na základě znalosti struktury stromu dotazů a požadovaného řazení potomků uzlů stromu, obojí můžeme jednoznačně určit z **VSD**. Požadovaný deterministický konečný automat MSD=(Q, T, δ, q_0, F) sestrojíme takto:

1. Množina vnitřních stavů Q:
   $Q = \{q\_0, q\_left\_1, q\_right\_1, ..., q\_left\_n, q\_right\_n\}$
   Každému uzlu stromu dotazů příslušejí dva stavy – "left" a "righ". Navíc je zde stav q_0, který je počátečním stavem automatu.

2. Vstupní abeceda T:
   $T = A = \{$"[1″, "1]″, "[2″, "2]″... "[n″, "n]″$\}$

3. Přechodové zobrazení δ:
   $\delta(q\_0,$ "[1″$) = q\_left\_1$
   $\delta(q\_left\_i,$ "i]″$) = q\_right\_i$ pro $i \in 1..n$
   $\delta(q\_right\_i,$ "[i″$) = q\_left\_i$ pro $i \in 1..n$
   $\delta(q\_left\_i,$ "[j″$) = q\_left\_j$, je- li uzel u_j přímým potomkem uzlu u_i
   $\delta(q\_right\_j,$ "i]″$) = q\_right\_i$ je- li uzel u_j přímým potomkem uzlu u_i
   Nakonec vyřešíme přechody mezi potomky stejného rodiče: Potomky jednoho rodiče seřadíme v pořadí podle výskytu ve **VSD** výrazu a z pravého stavu předchůdce vedeme přechod do levého stavu všech následníků, přechod spojíme se čtením symbolu levé závorky s číslem příslušného následníka "[descendant_number".

4. Počáteční stav je q_0

5. Množina koncových stavů F:
   $F = \{q\_0, q\_prava\_1\}$
   Koncové stavy představují uzavření závorky příslušející kořeni stromu dotazů resp. přijetí prázdného řetězec znamenající nulový výskyt závorky přiřazené kořeni stromu dotazů.

Výraz **VSD** označující strom dotazů můžeme tedy převést na konečný automat a posléze na regulární gramatiku generující jazyk požadovaných vlastností.
S výhodou nyní užijeme skutečnosti, že atributovaný závorkový výraz **VSD** nepopisuje strom dotazů jednoznačně. Může existovat více stromů dotazů se stejným vyjádřením pomocí **VSD** výrazu - stromy se shodným **VSD** výrazem se mohou lišit v přiřazení relací uzlům nebo v definici funkcí přiřazených hranám. Zkusme nyní sestrojit algoritmus pro generování vět jazyka **L(MSD)**, vznik věty jazyka bude při tom řízen informacemi získanými z traversování stromu dotazů. Na základě znalosti **MSD** automatu sestrojíme nejprve pravou regulární gramatiku postupem uvedeným v [1]. Gramatika takto sestavená velice názorně kopíruje strukturu zdrojového konečného automatu.

Generování věty v pravé regulární gramatice probíhá jako posloupnost přímých derivací z počátečního symbolu gramatiky. Přímá derivace v pravé regulární gramatice znamená nahrazení jediného neterminálu X větné formy pravou stranou pravidla, jehož levou stranu tvoří právě neterminál X. Pravidel se stejným neterminálem na levé straně může být v pravé regulární gramatice několik, proto je třeba umět se při generování věty rozhodnout pro jedno určité pravidlo.

Při expanzi neterminálů vytvořeným z "levých" stavů (q_left_i), jsou na výběr pravé strany obsahující po řadě neterminály vytvořené z "levých" stavů potomků uzlu u_i a "pravého" stavu uzlu u_i. Analogicky při expanzi neterminálů vytvořeným z"pravých" stavů (q_right_i), jsou na výběr pravé strany obsahující po řadě neterminály vytvořené z "levích" stavů následníků uzlu u_i v "sourozenecké řadě", "levého" stavu uzlu u_i a "pravého" stavu rodičovského uzlu.

Výběrovým kritériem budiž existence resp. neexistence dosud nezpracovaných prvků v relaci příslušející některému ze zainteresovaných uzlů. Konkrétní postup budiž tento:

Při expanzi neterminálu q_left_i e již znám právě zpracovávaný prvek *t* relace r_i. Proto je možné určit selekci na realce potomků uzlu u_i.

Příslušné pravé strany pravidel představují přechod ke generování obsahu na základě potomků uzlu u_i resp. přechod na další prvek relace r_i. Pravidlo obsahující na pravé straně neterminál q_left_j se užije v případě, že výsledek selekce na relaci u_i není prázdná relace. V opačném případě se provede přechod na další pravidlo v pořadí určeném **VSD** výrazem a testování probíhá stejně. Při expanzi neterminálu q_right_i e nejprve označí prvek *t* za zpracovaný a testuje existence dalšího dosud nezpracovaného prvku v relaci r_i. Pakliže takový prvek existuje, použije se pravidlo q_right_i $\longrightarrow$ "[i"q_left_i, jinak se ve známém pořadí zkoumají pravidla vedoucí k levým závorkám sourozenců, pravidlo vedoucí k pravé závorce předka se užije v případě, že ostatní možnosti nelze užít.

Snadno zjistíme, že při řízení expanze neterminálů výše navrženým způsobem kopíruje generování věty jazyka **L(MSD)** traverzování stromu dotazů. Operační složitost je tedy dána nároky na traverzování plus nároky na generování věty. **VSD** výraz při tom m.j. určuje pořadí procházení potomků resp. pořadí kontroly použitelnosti pravidel pro expanzi příslušného neterminálu. Pakliže takto navrženému algoritmu necháme zpracovat jiný strom dotazů pro daný **VSD** výraz, vygeneruje algoritmus obecně jinou větu jazyka **L(MSD)**. Pořadí provádění kontrol a expanzí je neměnné pro všechny povolené vstupní stromy dotazů. Proto je můžeme s výhodou zakódovat např. do posloupnosti operací, která se provede v okamžiku generování výstupu.

Na závěr prozkoumejme ještě možnost umístit mezi závorky vstupního **VSD** výrazu nějaké pro vlastní generování "balastní" symboly. Závorku pak zopakujeme na daném místě včetně jejího obsahu. Úprava **MSD** automatu je relativně jednoduchá – přibudou lineární úseky stavů a přechodů generující dané konstantní řetězce výplňových symbolů, **MSD** automat zůstane deterministický. Algoritmus generování vět jazyka

L(MSD) zůstane též v podstatě nezměněn, žádné dodatečné rozhodování není potřeba.

3.1. Grafické formátování informací

Nyní se zaměříme blíže na řetězce terminálních symbolů umístěných mezi závorkami VSD výrazu. V dostatečně dlouhých větách všech bezkontextových jazyků lze na určitých místech zopakovat podřetězec dané věty tak, aby získaná věta patřila do jazyka [3]. Kontextové jazyky podobnou vlastnost obecně nemají, nicméně i tam můžeme v izolovaných případech zopakovat na určitých místech části věty s identickým výsledkem. Na problematiku můžeme nahlédnout i z druhé strany, tj. existuje možnost vložit atributované závorky **VSD** výrazu do věty jiného jazyka tak, aby repetice obsahu závorky na daném místě a následné vypuštění **VSD** závorek neporušilo příslušnost věty k jazyku. Toho můžeme s výhodou využít např. ve spojení s jazykem HTML takto:

1. HTML dokument obohatíme o atributované závorky.
2. Pomocí stromu dotazu modifikujeme generování vstupní věty.
3. Z výstupné věty odstraníme atributované závorky, čímž se z ní stane opět věta jazyka HTML.

Jediné úskalí postupu je v bodu 1. – rozmístění atributovaných závorek po původní větě jazyka nemůže být zcela libovolné. Rozmístování závorek po původní větě jazyka však může být přenecháno k tomu určenému editoru. Jako příklad již existující implementace uveďme jazyk RTF a editor Microsoft Word – Microsoft Word obsahuje funkci “Vložit záložku”, která vloží atributovanou pravou a levou závorku požadovaných vlastností. Úplně postačuje funkce umožňující korektní vložení izolovaných závorek, vzájemný vztah závorek se vyřeší v rámci předzpracování dokumentu.

4. Realizace generátoru dokumentů

Doposud jsme probrali jen základní “motor” generátoru dokumentů. Od reálného exportního systému požadujeme ještě výstup hodnot atributů prvků relací do výsledného textu. V průběhu traverzování stromu dotazů jsou při průchodu uzlu u_i známy a konstantní aktuálně zpracovávané prvky relací všech předchůdců uzlu u_i až ke kořeni stromu. Je proto přirozené umožnit zápis hodnot atributů prvku *t* relace r_i v rámci celé závorky příslušející uzlu u_i ve **VSD** výrazu. Definujme nyní pojem “**SCH (SD)**”.

Definition 2 *S*

chéma **SCH(SD)** stromu dotazů **SD** je stromový graf se stejnou množinou uzlů *a* hran *a* se shodným incidenčním zobrazením jako zdrojový strom dotazů **SD**. Uzly schématu jsou ohodnoceny schémata relací, hrany schématu nejsou ohodnoceny.

Na základě znalosti schématu stromu dotazů je možné určit všechny možné tvary **VSD** výrazu a ověřit rozmístění atributů mezi závorkami **VSD** výrazu. Kompletní atributová gramatika definující syntaxi rozeznávanou syntaktickým analyzátorem vznikne jednoduchým rozšířením gramatiky pro **VSD** výraz o atributovaný symbol *a*“atributu relace” a atributovaný symbol *p*“vmezeřený text”. Na jejím základě lze sestrojit syntaktický analyzátor a prekladač vstupního textu do vhodné binární vnitřní formy jak je uvedeno např. [2]. Schéma stromu dotazů musí být zadáno spolu se vstupním textem. Překladač jej využije při ověření umístění symbolů značících atributy relací v rámci příslušných závorek atd. Interpret vnitřní formy (vlastní generátor dokumentů) je možné realizovat s využitím poznatků z kapitoly 2 a 3 tohoto textu, hodnoty atributů je třeba v okamžiku zápisu konvertovat do textové podoby.

4.1. Příklad užití ve spojení s jazykem HTML

Předpokládejme, že máme k dispozici implementovaný překladač a implementovaný interpret vnitřní formy. Lexikální analyzátor v překladači nechť předpokládá zápis atributovaných závorek pomocí lexikálních elementů ve tvaru `<%BS_nodeid%>` pro otevírací závorku a `<%BE_nodeid%>` pro uzavírací závorku. Zápis označení atributů se předpokládá ve tvaru `<%variableid%>`. Předpokládejme nyní, že chceme zobrazit izolovanou tabulku v určitém grafickém formátování, data získáme z relační databáze v podobě jediné relace – strom dotazů má pouze kořenový uzel.

Example 1 *V*

zorový dokument obohacený o závorky a atributy bude vypadat např. takto:

```
<html><body> <table>
    <tr><td>First name</td><td>Last name</td></tr>
    <%BS_1%>
    <tr><td><%txtfirstname%></td><td><%txtlastname%></td></tr>
    <%BE_1%>
</table> </body></html>
```

Případná změna resp. doplnění formátování může probíhat bez zásahu programátora s např. pomocí vhodného HTML editoru.

Example 2 *V*

ýsledný dokument bez závorek a s nahrazenými atributy bude vypadat např. takto:

```
<html> <body> <table>
    <tr><td>First name</td><td>Last name</td></tr>
    <tr><td>Zuzana</td><td>Capekova</td></tr>
<tr><td>Iva</td><td>Zlamalova</td></tr> </table> </body></html>
```

5. Závěr

Použití uvedeného postupu není omezeno jen na HTML jazyk. Postup lze aplikovat i ve spojení s jazykem xml, rtf apod. Editace vzhledu dokumentů může být přenechána zcela neškoleným uživatelům. Např. v případě jazyka rtf je možné pro změnu vzhledu dokumentu přímo použít aplikaci MS Word (funkce ”vložit záložku” vytvoří závorky požadovaných vlastností), v důsledku čehož není nutné uživatele přeškolovat na specifická vývojová prostředí, návrháře reportů apod. Předzpracování vzorového dokumentu do vhodné binární vnitřní formy navíc zvyšuje efektivitu metody a minimalizuje výpočetní i paměťové nároky na generování dokumentu.

Literatura

[1] Melichar, B. “Jazyky a překlady”, Vydavatelství ČVUT 1999.

[2] Müller, K. “Programovací jazyky”, Vydavatelství ČVUT 2002.

[3] Mareš, J. “Jazyky, gramatiky a automaty”, Vydavatelství ČVUT 2004,

[4] Stuller, J. “Přednášky z databázových systémů”, `http://www.cs.cas/ stuller`.

[5] Maier, D. “The theory of relational databases”, Pitman 1983.