Zpravodaj Československého sdružení uživatelů TeXu

Zdeněk Wagner

Využití XML a LaTeXu při sazbě odborných knih

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 12 (2002), No. 3, 188–211

Persistent URL: http://dml.cz/dmlcz/149903

Využití XML a LaTeXu při sazbě odborných knih[1]

Zdeněk Wagner

Odborné knihy mají obvykle pevnější strukturu než beletrie. Tato struktura nebývá samoúčelná, ale často je důležitou nositelkou informace. Ač LaTeX jistým způsobem vede zkušené uživatele k užívání strukturního značkování, není zachování informace vždy triviální a vyžaduje programátorský způsob ke zpracování dokumentu. Jisté potíže vzniknou i při tvorbě neobvyklých rejstříků, kde se naráží na omezení programu MakeIndex. XML naproti tomu definuje strukturu dokumentu. Navíc přináší hotové nástroje pro různé typy zpracování. Pro znalce LaTeXu je pak přirozenou cestou využití XSLT a transformovaný text zpracovat LaTeXem jako sázecím strojem. Je demonstrována případová studie počínaje návrhem DTD přes transformační styl až po LaTeXová makra užitá k sazbě. Je provedeno srovnání se zpracováním obdobné knihy, kdy nebylo využito výhod XML.

Klíčová slova: XML, LaTeX, XSLT, DTD, validace, schéma

1. Úvod

Při zpracování dokumentů zjišťujeme, že jejich text je určitým způsobem členěn. Toto členění obvykle plní určitou funkci. Při tisku takového dokumentu pak musíme členění reprezentovat graficky.

Finální grafickou úpravu nemusíme vždy znát na počátku práce s dokumentem. S výhodou tedy využijeme nějaký sázecí systém, který dokáže separovat obsah a formu. Možností máme několik.

LaTeX (a vlastně i plain TeX) umožňuje definovat kontextové značky, jimž až dodatečně přidělíme vizuální podobu. Pokud se během zpracování změní požadavky na grafickou úpravu, nemusíme zasahovat do textu, ale upravujeme pouze balík podpůrných maker.

Na zachycení obsahu dokumentu nezávisle od formy jsou však primárně určeny SGML a XML. Výhodou je, že k těmto formátům existuje řada volně šiřitelných i komerčních nástrojů. Navíc lze dokument snadno přeformátovat do několika různých podob. V tomto příspěvku ukážeme, že použití XML je výhodné i v případě, kdy ze vstupního souboru generujeme pouze jediný výstup.

[1]Článek byl poprvé publikován ve sborníku Jan Kasprzak, Petr Sojka: SLT 2002, KONVOJ 2002, ISBN 80-7302-043-2, na stranách 27–48. Publikováno se souhlasem vydavatele a autorů.

2. Sazba XML souboru

XML definuje strukturu dokumentu, ale neurčuje vizuální vzhled. Ten lze také určit pomocí nástrojů XML, tzv. formátovacích objektů. Přestože již existují nezávislé nástroje, které provádějí sazbu přímo na základě formátovacích objektů, nejsou v typografické kvalitě schopny konkurovat TEXu.

V TEXovém světě již existují nástroje pro přímou sazbu XML, např. PassiveTEX [8] a ConTEXt. Jejich využitelnost však není univerzální. Může se stát, že z daného XML dokumentu chceme tisknout jen určité části, případně se mají tisknout v jiném pořadí, než v jakém se v původním souboru vyskytují. Tyto operace zvládne snadno XSLT. Zkušený uživatel TEXu či LATEXu si nyní zákonitě položí otázku, proč by po transformaci mělo následovat zpracování s využitím některého z výše uvedených nástrojů, když výstupem transformace může být stejně tak TEXový soubor a o jeho formátování se mohou postarat TEXová makra.

Při transformaci z XML do TEXu se můžeme rozhodnout, jakou formátovací práci provede transformační styl a co svěříme až TEXovým makrům. Prvním extrémem je, že při transformaci pomocí XSLT převedeme vše až na TEXové příkazy nízké úrovně. Opačným extrémem je, že veškerou práci ponecháme LATEXu. XSLT zpracuje pouze základní elementy a vše ostatní se bude formátovat s využitím LATEXového balíku KEYVAL, přičemž atributy jsou konvertovány na nepovinné parametry prostředí. Transformační styl pak může vypadat např. takto:

```
<?xml version="1.0" encoding='cp852'?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0"
                xmlns:saxon="http://icl.com/saxon"
                extension-element-prefixes="saxon">
<xsl:output indent="no" method="text" encoding="cp852"
            saxon:character-representation="native"/>

<xsl:strip-space elements="*"/>

<!-- Root -->
<xsl:template match="/">
  \documentclass{book}
  \usepackage{keyval}
  \usepackage{mujstyl}
  \begin{document}
    <xsl:apply-templates/>
  \end{document}
</xsl:template>
```

```
<!-- Texty -->
<xsl:template match="text()">
  <xsl:value-of select="."/>
</xsl:template>

<!-- Odstavec -->
<xsl:template match='p'>
  <xsl:apply-templates/>
  <xsl:text>&#10;&#10;</xsl:text>
</xsl:template>

<!-- Tučně -->
<xsl:template match='b'>
  <xsl:text>\textbf{</xsl:text>
  <xsl:apply-templates/>
  <xsl:text>}</xsl:text>
</xsl:template>

<!-- Kurzíva -->
<xsl:template match='i'>
  <xsl:text>\textit{</xsl:text>
  <xsl:apply-templates/>
  <xsl:text>}</xsl:text>
</xsl:template>

<!-- Ostatní -->
<xsl:template match='*'>
  <xsl:value-of select='concat("\begin{", name(), "}")'/>
  <xsl:if test='@*'>
    <xsl:text>[</xsl:text>
    <xsl:apply-templates select='@*'/>
    <xsl:text>]</xsl:text>
  </xsl:if>
  <xsl:apply-templates/>
  <xsl:value-of select='concat("\end{", name(), "}")'/>
</xsl:template>

<!-- Atributy -->
<xsl:template match='@*'>
  <xsl:if test='position()>1'>
    <xsl:text>,</xsl:text>
```

```
  </xsl:if>
  <xsl:value-of select='concat(name(), "=", .)'/>
</xsl:template>

</xsl:stylesheet>
```

Skutečně použitý styl pak bude někde mezi těmito dvěma extrémy. Co provede XSLT a co se ponechá LaTeXovým makrům, bude záležet především na schopnostech a zkušenostech každého programátora. Můžeme se ovšem také rozhodnout pro některý z běžně užívaných formátů XML a využít transformačních stylů, které jsou již hotové.

3. XML, pro a proti

Viděli jsme, že oproti přímému vytvoření dokumentu v LaTeXu máme při použití XML jednu operaci navíc, a to transformaci. Měli bychom se tedy zamyslet, zda nám tato práce navíc přinese nějaký užitek.

Obvyklým argumentem pro použití XML je snadnost převedení dokumentu do několika různých tvarů (PDF, HTML, tištěná podoba...). XML má však jisté výhody i v případě, že požadujeme jediný výstup.

Jedna z výhod se projeví při sazbě odborných knih, které mají pevně danou strukturu. Struktura je zde též nositelkou informace a může sloužit pro následné zpracování. Předpokládáme totiž, že životnost knihy nekončí jejím vydáním, ale její obsah může v budoucnu sloužit k účelům, které v okamžiku sazby ještě neznáme. Můžeme namítnout, že i z LaTeXového souboru lze získat dodatečné informace, pokud byl vhodně kontextově značkován, ale vždy je to mnohem náročnější než využití XSLT.

V učebnici matematiky lze věty a důkazy značkovat způsobem:

```
<věta číslo='1'>
  <text>
    Sem přijde text věty.
  </text>
  <důkaz>
    Zde bude důkaz
  </důkaz>
</věta>
```

V budoucnu lze pak z knihy vytáhnout např. jen věty a jejich důkazy. Podobně můžeme využít strukturu:

```
<cvičení číslo='1'>
  <příklad>
    Zadání příkladu.
```

```
  </příklad>
  <řešení>
    Řešení, které bude uvedeno jinde.
  </řešení>
</cvičení>
```

Tak máme příklad i jeho řešení v dokumentu na stejném místě, což sníží pravděpodobnost zavlečení chyb. Pomocí XSLT snadno v hlavní části knihy vytiskneme jen zadání příkladu a v závěrečné kapitole jeho řešení.

Jinou aplikací je vytažení hypertextových odkazů z knihy, aby mohly být zveřejněny na WWW stránkách. Příklad najdete na WWW stránkách věnovaných knize *XML pro každého* [4].

Další výhoda se projeví, pokud v dokumentu potřebujeme velmi nestandardní rejstříky. Program MakeIndex má svá omezení. Někdy bychom potřebovali buď netriviální zásah do jeho zdrojového kódu, nebo pomocný preprocesor. Ukázka bude uvedena později v případové studii. Vzhledem k tomu, že XSLT umí abecední řazení, může být využití XSLT při tvorbě rejstříků snadnější.

XML narozdíl od LaTeXu umožňuje definovat strukturu dokumentu a ověřit jeho správnost. O metodách validace se podrobněji zmíníme později.

Při psaní českých textů v XML můžeme narazit na problém s nezlomitelnou mezerou za jednopísmennými neslabičnými předložkami. Řešení však není složité. Lze použít program VLNA či jeho ekvivalent a následně nahradit tildu entitou ` `.

4. Volba typu dokumentu

Na začátku návrhu dokumentu v XML musíme zvolit způsob jeho značkování. Velmi rozšířeným standardem, používaným v mnoha dokumentech, je DocBook [1]. Pro jeho použití v konkrétním případě se však nerozhodujeme podle toho, zda je dobrý či špatný, zda je používán zřídka či často, ale podle toho, zda podporuje struktury, které náš dokument obsahuje, případně, zda se dá snadno modifikovat, aby vyhověl našim značkovacím požadavkům. DocBook obsahuje základní elementy, které se používají ve všech dokumentech, a dále řadu dalších elementů potřebných pro různé technické dokumenty. DTD je navrženo tak, aby jej bylo možno poměrně snadno doplňovat a modifikovat. Přesto se v praxi setkáme s případy, kdy DocBook nabízí příliš mnoho elementů, které vůbec nepotřebujeme, a zcela chybí elementy, které jsou pro danou aplikaci naprosto nepostradatelné. Pak je vytvoření nového DTD nejpřirozenějším řešením.

5. DTD ano či ne?

Výhodou XML oproti SGML je, že pro zpracování dokumentu často nepotřebujeme DTD. Obvykle si vystačíme s definicemi různých entit, které můžeme vložit přímo do dokumentu. Ač je deklarace entit součástí DTD, není to takové DTD, jaké známe z SGML. Navrhujeme-li tedy strukturu dokumentu pro jedno použití, nemusíme se s vytvářením DTD zdržovat. Praxe však ukazuje, že DTD je užitečné i v těchto případech.

V první řadě si musíme uvědomit, že člověk při psaní dokumentu chybuje. Při zapisování tagu lze udělat překlep. Nemáme-li textový editor, který doplňuje ukončovací tagy, můžeme je zapomenout nebo v nich udělat překlep. Editor, který doplňuje uzavírací tagy párově k počátečním, je dobrý sluha, ale může být i zlým pánem. Předpokládejme, že místo `<emphasis>` namíšeme omylem `<emhpasis>`. „Chytrý“ editor nám pak na konec zvýrazněné fráze doplní `</emhpasis>`. Parser, který neprovádí validaci, žádnou chybu nezaznamená. Chyba se projeví až při zpracování dokumentu. XSLT procesor má obvykle defaultní pravidlo pro případ, že žádná explicitní šablona nebyla nalezena. Takto chybně zadaný element se tedy zpracuje „nějak“ a zvláště v případě rozsáhlých dokumentů a komplikovaných stylů nemusí být snadné odhalit příčinu, proč je část textu zpracována zcela jinak, než bylo zamýšleno. Diagnostické zprávy validujících parserů jsou sice někdy poněkud kryptické, ale mnohem rychleji vedou k určení zdroje chyby.

Druhou výhodu validace oceníme v okamžiku, kdy začneme programovat styl, jenž s dokumentem provádí složitější transformaci. DTD nám totiž naprosto jasně říká, jaké situace mohou nastat. Víme tudíž naprosto přesně, jaké případy musíme v transformačním stylu ošetřit, a čím se zabývat nemusíme, protože to validující parser nepřipustí. Autor ze své zkušenosti ví, že návrh transformačního stylu je snadnější, když si vezmeme DTD a postupně podle něj vytváříme jednotlivé šablony.

6. Alternativní metody validace

Je obecně známo, že DTD určuje pouze základní pravidla pro strukturu dokumentu. V praxi často potřebujeme podstatně přesnější vymezení typu a obsahu elementů i jejich atributů. Tento problém se snaží řešit *schéma*. Bohužel nástroje, které využívají schéma pro validaci, nejsou ještě dostatečně rozšířené. Navíc ani tak nejsou ošetřeny všechny možnosti, s nimiž se můžeme setkat. Nezbývá tedy jiná možnost než přistoupit k validaci vlastními prostředky. V principu je možné využít některou z knihoven pro práci s XML soubory a napsat si speciální validátor. Často je však vhodnější vytvořit validátor pomocí XSLT. Tento přístup je

využíván i v CML (Chemical Markup Language) v projektu SELFML (Standard Electronic File Format Markup Language) [5].

Vezměme si jako příklad XML soubor, který obsahuje výsledky matematického zpracování jistých fyzikálně chemických měření. Soubor odpovídá následujícímu DTD:

```
<!ELEMENT rset (result+)>

<!ELEMENT result (system,concentration,T,parameters,table,sigma)>

<!ELEMENT table (row+)>

<!ELEMENT row (cell+)>

<!ELEMENT cell (#PCDATA)>

<!ELEMENT system (#PCDATA)>

<!ELEMENT concentration (#PCDATA)>

<!ELEMENT T (#PCDATA)>
<!ATTLIST T unit  CDATA 'K'>

<!ELEMENT sigma (#PCDATA)>

<!ELEMENT parameters EMPTY>
<!ATTLIST parameters C  CDATA #REQUIRED
                     D  CDATA #REQUIRED>
```

Z DTD je patrné, že každý `<result>` obsahuje právě jednu tabulku. V DTD však nelze zapsat jednu podstatnou podmínku, kterou nepodporuje ani schéma. Tabulka totiž obsahuje obvykle jeden řádek, který má druhý sloupec prázdný. Takový řádek nás při dalším zpracování nebude zajímat. Podmínkou však je, aby všechny tabulky, po vynechání řádku s prázdným druhým sloupcem, měly identický počet řádků. Druhou podmínkou je, že hodnoty v prvním sloupci všech tabulek musí být shodné.

Čtenář by mohl namítnout, že jsme říkali, že soubor byl vytvořen počítačem. Validace by tudíž neměla být nutná, protože generované soubory jsou správné. Ve skutečnosti je však nepřeberná řada příčin, které vedou k tomu, že programově vytvořený soubor je vadný. Dokud není program dokonale otestován, není jisté, že v něm není nějaká chyba. Validace výsledného XML souboru je jedním z užitečných kroků při testování programu. A nesmíme ani zapomenout na to, že původní vstupní data vytvářel člověk. V hodnotách prvního sloupce tedy

mohou být překlepy[2] a také mohlo dojít k tomu, že se jeden řádek v některé tabulce ztratil. Podmínky tohoto typu neřeší ani schéma, ale k validaci lze použít následující styl:

```
<?xml version="1.0" encoding='cp852'?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0"
                xmlns:saxon="http://icl.com/saxon"
                extension-element-prefixes="saxon">
<xsl:output indent="no" method="text" encoding="cp852"
            saxon:character-representation="native"/>

<xsl:strip-space elements="*"/>

<!-- All elements -->
<xsl:template match="/|*">
  <xsl:apply-templates/>
</xsl:template>

<!-- Text contents of all elements -->
<xsl:template match="text()"/>

<!-- Result -->
<xsl:template match='result[position()>1]'>
  <xsl:apply-templates>
    <xsl:with-param name='pos'>
      <xsl:value-of select='position()'/>
    </xsl:with-param>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match='result'/>

<!-- Table -->
<xsl:template match='table'>
  <xsl:param name='pos'/>
  <xsl:if test=
         'not(count(row[not(cell[position()=2]="")])=
```

[2] Zdálo by se, že by bylo jednodušší, kdyby data pro první sloupec byla zadána pouze jednou. Z povahy měření a následného zpracování, jejichž vysvětlení je nad rámec tohoto příspěvku, však plyne, že by to způsobilo laborantce mnoho těžkostí a spíše by to vedlo k zavlečení dalších chyb.

```
          count(/rset/result[position()=$pos]/table/row
          [not(cell[position()=2]="")]))'>
    <xsl:text>Nesouhlasí počet řádků: </xsl:text>
    <xsl:value-of
         select='count(row[not(cell[position()=2]="")])'/>
    <xsl:text> != </xsl:text>
    <xsl:value-of
         select='count(/rset/result[position()=$pos]/table/row
                 [not(cell[position()=2]="")])'/>
  <xsl:text>&#10;</xsl:text>
  </xsl:if>
  <xsl:apply-templates>
    <xsl:with-param name='pos'>
      <xsl:value-of select='$pos'/>
    </xsl:with-param>
  </xsl:apply-templates>
</xsl:template>

<!-- Row -->
<xsl:template match='row'>
  <xsl:param name='pos'/>
  <xsl:variable name='rpos'>
    <xsl:value-of select='position()'/>
  </xsl:variable>
  <xsl:if test=
      'not(cell[1]=/rset/result[$pos]/table/row[$rpos]/cell[1])'>
    <xsl:text>Nesouhlasí hodnota v 1. sloupci: </xsl:text>
    <xsl:value-of select='cell[1]'/>
    <xsl:text> != </xsl:text>
    <xsl:value-of
         select='/rset/result[$pos]/table/row[$rpos]/cell[1]'/>
    <xsl:text>&#10;</xsl:text>
  </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

Ve stylu jsou zajímavé dvě věci. První z nich je potlačení výstupu textových uzlů. Druhou zvláštností jsou dvě šablony pro element `<result>`. Předpokládáme totiž, že tabulka v prvním výsledkovém bloku je správná, proto ji nekontrolujeme. Všechny ostatní musí souhlasit s první tabulkou.

Tím ovšem nejsou vyčerpány všechny možnosti, s nimiž se můžeme setkat. Některé chyby v XML souborech nelze odhalit žádným strojově vyjádřitelným algoritmem. K jejich nalezení nám ovšem zase pomůže XSLT. Kromě transformací se dá XSLT využít i jako dotazovací jazyk. Můžeme si tedy vyhledat a vypsat podezřelá místa a pak rozhodnout, zda je obsah správný či špatný. Konkrétní příklad bude uveden později.

7. Případová studie

Možnosti využití XML a LaTeXu jsou demonstrovány na knize *Vinohradský hřbitov včera & dnes* [6]. Její styl pro transformaci do LaTeXu má však více než 900 řádků a LaTeXová makra mají přes 250 řádků. Jejich plný kód zde proto nebude uveden a zaměříme se pouze na vyzdvižení hlavních výhod, které použití XML přineslo.

7.1. Pracovní cyklus

Kniha tohoto typu vzniká poměrně dlouho. Autor, který vzhledem ke svému věku počítač nepoužívá, zapisuje text obvykle na kartičky. Z této kartotéky písařka text knihy opíše. Navíc každý ze spoluautorů pracoval na jiné části (jež se však prolínají). Písařka musí tyto části vhodně spojit. Z toho pak plyne, že při prvních korekturách se provádí poměrně mnoho změn. Kniha je sázena dvousloupcově a do textu jsou vsazovány fotografie. Pravidla pro jejich umístění jsou dosti přísná a mechanismus plovoucích objektů nelze využít. Fotografie jsou proto vkládány až při poslední korektuře. Při jednotlivých fázích zpracování tedy dochází k posunu materiálu na jiné stránky. Odkazy uvnitř knihy musíme proto řešit nějakým automatem.

Odkazy na čísla stran se týkaly zejména rejstříků, jež byly dost nestandardní. Autoři je sice zpracovali ručně, ale opakované dopisování správných čísel stran by jistě vedlo k mnoha chybám. Jak se později ukáže, právě při tvorbě rejstříků se projevila jedna z výhod XML.

Text knihy přepisovala externí písařka a posílala po částech elektronickou poštou. Protože písařka nemá žádný nástroj pro práci s XML (text psala ve Wordu), nemá TeX ani Acrobat Reader, byl nakonec vytvořen i soukromý styl pro transformaci do HTML. Pokud bylo nutno, aby písařka něco rychle ověřila v původní kartotéce, mohla se podívat na privátní WWW stránky se zpracovanou částí knihy. Klikací verze v HTML také významně pomohla při korekturách.

7.2. Tvoříme DTD

K potřebě vytvoření DTD jsem dospěl hned na počátku práce, kdy písařka (která se při přepisování tohoto rukopisu teprve začínala učit XML) udělala

první chybu. Základní struktura byla zřejmá, ale postupem času se DTD mírně měnilo a rostlo.

Zmíněna kniha má dva úvody, hlavní část, dva rejstříky, seznam literatury a tiráž. Pouze hlavní část a rejstříky byly generovány z XML, zbývající části byly napsány přímo v LaTeXu. DTD bylo proto děláno jen pro část dokumentu.

V úvodu bychom se měli zmínit, že parser sice komentáře ignoruje, ale musí je přečíst, aby našel jejich konec. I když pro tagy použijeme pouze písmena anglické abecedy, musíme definovat použité kódování kvůli česky psaným komentářům. DTD je také XML soubor a pokud v něm píšeme česky, musíme v něm použít XML deklaraci. Protože DTD je poměrně krátké, uvedeme je celé:

```
<?xml version='1.0' encoding='cp852'?>
<!-- DTD pro hřbitov -->

<!-- Znakové entity -->
<!ENTITY otilde "&#245;">
<!ENTITY egrave "&#232;">

<!-- Parametrické entity -->
<!-- ================== -->

<!-- Obsah hrobů -->
<!ENTITY % attr
   "jmeno   CDATA #REQUIRED
    krestni CDATA #IMPLIED
    narozen CDATA #IMPLIED
    smrt    CDATA #IMPLIED
    titul   CDATA #IMPLIED
    prefix  CDATA #IMPLIED
    hodnost CDATA #IMPLIED">

<!-- Ano/Ne -->
<!ENTITY % ano "(ano|ne) 'ano'">

<!-- Struktura dokumentu,
     hřbitov má nejméně jedno oddělení -->
<!ELEMENT hrbitov (oddeleni|tex)+>

<!-- Oddělení jsou číslována,
     obvykle začínají plánkem a mají hroby -->
<!ELEMENT oddeleni (hrob|img|tex)+>
<!ATTLIST oddeleni cislo  CDATA #REQUIRED
```

```
                    planek CDATA #IMPLIED
                    extra  CDATA #IMPLIED
                    popis  (vedle|pres|sloupec)  'vedle'
                    space  CDATA #IMPLIED>

<!-- Obrázek, většinou fotografie -->
<!ELEMENT img (#PCDATA)>
<!ATTLIST img  height   CDATA #IMPLIED
               width    CDATA #IMPLIED
               scale    CDATA #IMPLIED>

<!-- Oddělení má hroby, většinou číslované,
     výjimečně obsahuje obrázky -->
<!ELEMENT hrob (#PCDATA|clovek|rodina|umelec|p|tex|em|img)*>
<!ATTLIST hrob cislo CDATA #IMPLIED>

<!-- Rodina má jenom lidi -->
<!ELEMENT rodina (clovek+)>
<!ATTLIST rodina odstavec %ano;>

<!-- Označíme, zda člověk patří do seznamu
     a zda se začíná nový odstavec -->
<!ELEMENT clovek (#PCDATA|hide)*>
<!ATTLIST clovek %attr;
                 tecka    %ano;
                 seznam   %ano;
                 odstavec %ano;>

<!ELEMENT hide (#PCDATA)>

<!ELEMENT umelec (#PCDATA)>
<!ATTLIST umelec %attr;>

<!-- Odstavec -->
<!ELEMENT p EMPTY>

<!-- Specifické příkazy pro TeX -->
<!ELEMENT tex (#PCDATA)>

<!-- Text zvýrazněný proložením -->
<!ELEMENT em (#PCDATA)>
```

Na začátku DTD si definujeme entity pro znaky, které nemáme v české abecedě, a parametrické entity, jež budeme dále v DTD využívat. Zbytek DTD je komentován, ale přesto potřebuje několik poznámek.

Členění do odstavců, jak je běžné v HTML i v DocBooku, by zde přidělalo práci a nebylo by účelné vzhledem k tomu, že odstavec zde není podstatnou stavební jednotkou textu. Proto používáme značku `<p/>` s významem *předěl odstavců*.

Element `<oddeleni>` má několik atributů. Text každého oddělení začíná plánkem a z prostorových důvodů se plánek sází jako perex s popisem buď umístěným vedle nebo přes plánek (když není obdélníkový a je v něm volné místo), jindy se sází přímo do sloupce. Umístění popisu je specifikováno atributem `popis`. Text oddělení nezačíná vždy na nové stránce. Prostředí MULTICOL může mít problémy, když perex je příliš vysoký. Řeší se to nepovinným parametrem, kdy tomuto prostředí sdělíme, že má přejít na novou stránku, je-li na stránce aktuální méně místa než zadaná hodnota. Tuto hodnotu zadáváme v atributu `space`. Atribut `extra` obsahuje volitelný LaTeXový kód, který ve výjimečných případech potřebujeme vložit.

Komentář si zaslouží elementy `<clovek>` a `<umelec>`. První z nich obsahuje údaje o pohřbené osobě. Protože jsou někteří pohřbení jen zmíněni, vyskytují se i uprostřed odstavce a nemají být uvedeni v rejstříku, máme pro tyto účely vytvořeny atributy. Obsahem elementu `<clovek>` je nějaká charakterizace, obvykle profesní (lékař, továrník, hudební skladatel apod.) příslušné osoby. Element `<umelec>` slouží k vyznačení autorů funerálního umění. Téměř vždy se v textu vyskytují ve standardní podobě, kterou vytvoříme programově, ale v některých případech je jméno uvedeno uprostřed věty v jiném než prvním pádě. Pak je požadovaný tvar uveden v obsahu tohoto elementu, ale ve většině případů je element `<umelec>` prázdný.

7.3. Pomocné nástroje

V části 6 jsme se zmínili o tom, že k validaci souboru lze využít XSLT. V předchozí části jsme uvedli, že element `<umelec>` je obvykle prázdný. Protože písařka občas omylem do tohoto elementu uzavřela text, který tam nepatří, bylo nutno najít nějaký systém, který by správnost elementu `<umelec>` ověřil. Proto byl vytvořen následující styl:

```
<?xml version="1.0" encoding="cp852"?>

<!--
Tento styl vypisuje umělce, kteří mají neprázdný obsah.
To je většinou špatně.
-->
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0"
                xmlns:saxon="http://icl.com/saxon"
                extension-element-prefixes="saxon">

<xsl:output indent="no" method="text" encoding="cp852"
            saxon:character-representation="native"/>

<xsl:strip-space elements='*'/>

<!-- Použij šablony na předky elementu <umelec> -->
<xsl:template match='/|hrbitov|oddeleni|hrob'>
  <xsl:apply-templates/>
</xsl:template>

<!-- Pokud je obsah elementu neprázdný,
     vypiš jej a uveď informace pro jeho nalezení -->
<xsl:template match='umelec'>
  <xsl:if test=".!=''">
    <xsl:value-of select='saxon:systemId()'/>
    <xsl:text> line </xsl:text>
    <xsl:value-of select='saxon:lineNumber()'/>
    <xsl:text>&#10;</xsl:text>
    <xsl:value-of select='@jmeno'/><xsl:text>, </xsl:text>
    <xsl:value-of select='@krestni'/><xsl:text>: </xsl:text>
    <xsl:value-of select='../../@cislo'/><xsl:text>/</xsl:text>
    <xsl:choose>
      <xsl:when test="../@cislo">
        <xsl:value-of select='../@cislo'/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>bez čísla --- </xsl:text>
        <xsl:value-of select='saxon:path()'/>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:text>&#10;</xsl:text>
    <xsl:value-of select='normalize-space(.)'/>
    <xsl:text>&#10;&#10;</xsl:text>
  </xsl:if>
</xsl:template>

<!-- Ignoruj všechno ostatní -->
```

```
<xsl:template match='*|@*|text()'/>

</xsl:stylesheet>
```

Styl využívá některá rozšíření XSLT procesoru Saxon [3]. Díky tomu nám sdělí přesné místo neprázdného elementu `<umelec>` a my se rozhodneme, zda je to správně, nebo zda to vyžaduje opravu.

Jiným typem chyby byl zapomenutý element `<clovek>`. Za určitých okolností totiž nemusel být prvním uzlem v elementu `<hrob>` a validace pomocí DTD pak takovou chybu neodhalí. Lze to však řešit pomocí XSLT:

```
<?xml version="1.0" encoding="cp852"?>

<!--
Tento styl vypisuje hroby, kde není žádný člověk.
-->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0"
                xmlns:saxon="http://icl.com/saxon"
                extension-element-prefixes="saxon">

<xsl:output indent="no" method="text" encoding="cp852"
            saxon:character-representation="native"/>

<xsl:strip-space elements='*'/>

<xsl:template match='/|*'>
  <xsl:apply-templates/>
</xsl:template>

<!-- Všechny elementy <hrob>, v nichž není
     žádný element <clovek> -->
<xsl:template match='hrob[count(.//clovek)=0]'>
  <xsl:value-of select='saxon:systemId()'/>
  <xsl:text> line </xsl:text>
  <xsl:value-of select='saxon:lineNumber()'/>
  <xsl:text>: </xsl:text>
  <xsl:value-of select='../@cislo'/><xsl:text>/</xsl:text>
  <xsl:choose>
    <xsl:when test="@cislo">
      <xsl:value-of select='@cislo'/>
    </xsl:when>
```

```
    <xsl:otherwise>
      <xsl:text>bez čísla</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:text>&#10;</xsl:text>
</xsl:template>

<!-- Ignoruj atributy a text -->
<xsl:template match='@*|text()'/>

</xsl:stylesheet>
```

Podobný nástroj byl vytvořen pro nalezení místa, kde se vyskytuje konkrétní zemřelý či konkrétní umělec. Vzhledem k tomu, že vstupních souborů bylo 22, ukázal se tento nástroj jako velmi užitečný.

XSLT našlo uplatnění i při fakturaci. Bylo nutno zjistit počet snímků, které byly nově fotografovány (byly v adresářích „1“ a „2“), počet fotografií z autorova archívu, které byly pouze skenovány (byly v adresáři „3“) a počet plánků. Tyto údaje zjistil následující styl:

```
<?xml version="1.0" encoding="cp852"?>

<!--
Tento styl zjišťuje počet fotek v jednotlivých adresářích
a~počet plánků
-->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0"
                xmlns:saxon="http://icl.com/saxon"
                extension-element-prefixes="saxon">

<xsl:output indent="no" method="text" encoding="cp852"
            saxon:character-representation="native"/>

<xsl:strip-space elements='*'/>

<!-- Celý výpočet dělá tato šablona -->
<xsl:template match='/'>
  <xsl:text>Adresář 1:    </xsl:text>
    <xsl:value-of select=
    'count(//img[substring-before(normalize-space(.),"/")="1"])'/>
  <xsl:text>&#10;</xsl:text>
```

```
  <xsl:text>Adresář 2:    </xsl:text>
    <xsl:value-of select=
    'count(//img[substring-before(normalize-space(.),"/")="2"])'/>
  <xsl:text>&#10;</xsl:text>
  <xsl:text>Adresář 3:    </xsl:text>
    <xsl:value-of select=
    'count(//img[substring-before(normalize-space(.),"/")="3"])'/>
  <xsl:text>&#10;</xsl:text>
  <xsl:text>Fotky celkem: </xsl:text>
    <xsl:value-of select='count(//img)'/>
  <xsl:text>&#10;</xsl:text>
  <xsl:text>Počet plánků: </xsl:text>
    <xsl:value-of select='count(//oddeleni[@planek])'/>
  <xsl:text>&#10;</xsl:text>
</xsl:template>

<!-- Ignoruj všechno ostatní -->
<xsl:template match='*|@*|text()'/>

</xsl:stylesheet>
```

7.4. Tvorba rejstříků

Kniha obsahuje jmenný rejstřík pohřbených osob a rejstřík tvůrců funerálního umění. Ve jmenném rejstříku se uvádí jméno člověka, jeho profesní charakterizace, datum narození a smrti a číslo stránky. Je to více údajů, než se v jiných knihách zpracovává programem MakeIndex, ale v zásadě by tento prolém byl standardními LaTeXovými prostředky řešitelný. Náročnější je rejstřík tvůrců funerálního umění. Ten obsahuje příjmení a jméno umělce, jeho datum narození a případně úmrtí a číslo stránky. Potíž je v tom, že v textu se nevyskytují vždy všechny údaje. Písařka jména umělců nezná a není schopna všechny údaje do elementů `<umelec>` doplňovat. Navíc by tím stoupl počet neprázdných elementů `<umelec>` v dokumentu a validace by pak byla obtížnější. Při sazbě knihy *Olšanské umění, jeho tvůrci a doba* [9] to bylo řešeno pomocným perlovským skriptem a následným tříděním programem CSR [7]. Nyní bylo doplňování údajů řešeno výhradně v XSLT. Navíc bylo třídění podle normy implementováno do procesoru Saxon [10]. Uvádíme jen základní šablonu, která provádí třídění. Šablony pro standardizovaný tisk některých částí uvedeny nejsou.

```
<!-- Šablona pro výpis rejstříku umělců -->
<xsl:template name='umelci'>
  <xsl:variable name='jmeno' saxon:assignable='yes'/>
  <xsl:variable name='krestni' saxon:assignable='yes'/>
```

```
<xsl:variable name='narozen' saxon:assignable='yes'/>
<xsl:variable name='smrt' saxon:assignable='yes'/>
<xsl:variable name='odkaz' saxon:assignable='yes'/>
<xsl:variable name='novy' saxon:assignable='yes'/>
<xsl:variable name='fn'>
  <xsl:value-of select='$dir'/>
  <xsl:text>umelci.tex</xsl:text>
</xsl:variable>
<xsl:text>\input{</xsl:text>
<xsl:value-of select='$fn'/>
<xsl:text>}&#10;</xsl:text>
<xsl:document indent="no" method="text"
              encoding="cp852" href="{$fn}"
              saxon:character-representation="native">
  <xsl:text>\typeout{Tvůrci funerálního umění</xsl:text>
  <xsl:text> File: </xsl:text>
  <xsl:value-of select='saxon:systemId()'/>
  <xsl:text> line </xsl:text>
  <xsl:value-of select='saxon:lineNumber()'/>
  <xsl:text>}&#10;</xsl:text>
  <xsl:text>\begin{umelci}&#10;</xsl:text>
  <!-- Setřídění všech umělců -->
  <xsl:for-each select='//umelec'>
    <xsl:sort select='@jmeno' lang='cs'/>
    <xsl:sort select='@krestni' lang='cs'/>
    <!-- Zjišťujeme, zda je to nový umělec,
         data jsou dána jen někdy -->
    <xsl:if test="$jmeno!=''">
      <xsl:if test='@jmeno!=$jmeno'>
        <saxon:assign name='novy' select='1'/>
      </xsl:if>
      <xsl:if test='$krestni'>
        <xsl:if test='$krestni!=@krestni'>
          <saxon:assign name='novy' select='1'/>
        </xsl:if>
      </xsl:if>
      <xsl:if test='$narozen'>
        <xsl:if test='$narozen!=@narozen'>
          <saxon:assign name='novy' select='1'/>
        </xsl:if>
      </xsl:if>
      <xsl:if test='$smrt'>
```

```
    <xsl:if test='$smrt!=@smrt'>
      <saxon:assign name='novy' select='1'/>
    </xsl:if>
  </xsl:if>
  <xsl:if test='$novy'>
    <!-- Vypsání jména a vymazání proměnných -->
    <xsl:call-template name='umelec-v-rejstriku'>
      <xsl:with-param name='jmeno' select='$jmeno'/>
      <xsl:with-param name='krestni' select='$krestni'/>
      <xsl:with-param name='narozen' select='$narozen'/>
      <xsl:with-param name='smrt' select='$smrt'/>
      <xsl:with-param name='odkaz' select='$odkaz'/>
    </xsl:call-template>
    <saxon:assign name='krestni'/>
    <saxon:assign name='narozen'/>
    <saxon:assign name='smrt'/>
    <saxon:assign name='odkaz'/>
    <saxon:assign name='novy'/>
  </xsl:if>
</xsl:if>
<!-- Definice hodnot proměnných -->
<saxon:assign name='jmeno'>
  <xsl:value-of select='@jmeno'/>
</saxon:assign>
<xsl:if test='@krestni'>
  <saxon:assign name='krestni'>
    <xsl:value-of select='@krestni'/>
  </saxon:assign>
</xsl:if>
<xsl:if test='@narozen'>
  <saxon:assign name='narozen'>
    <xsl:value-of select='@narozen'/>
  </saxon:assign>
</xsl:if>
<xsl:if test='@smrt'>
  <saxon:assign name='smrt'>
    <xsl:value-of select='@smrt'/>
  </saxon:assign>
</xsl:if>
<!-- Vytvoření odkazu: id -->
<saxon:assign name='odkaz'>
  <xsl:value-of select='$odkaz'/><xsl:text> </xsl:text>
```

```
        <xsl:value-of select='generate-id()'/>
      </saxon:assign>
    </xsl:for-each>
    <!-- Konec cyklu, takže musíme vypsat posledního umělce -->
    <xsl:if test="$jmeno!=''">
      <xsl:call-template name='umelec-v-rejstriku'>
        <xsl:with-param name='jmeno' select='$jmeno'/>
        <xsl:with-param name='krestni' select='$krestni'/>
        <xsl:with-param name='narozen' select='$narozen'/>
        <xsl:with-param name='smrt' select='$smrt'/>
        <xsl:with-param name='odkaz' select='$odkaz'/>
      </xsl:call-template>
      <saxon:assign name='krestni'/>
      <saxon:assign name='narozen'/>
      <saxon:assign name='smrt'/>
      <saxon:assign name='odkaz'/>
    </xsl:if>
    <xsl:text>\end{umelci}&#10;</xsl:text>
  </xsl:document>
</xsl:template>

<!-- Výpis umělce v rejstříku -->
<xsl:template name='umelec-v-rejstriku'>
  <xsl:param name='jmeno'/>
  <xsl:param name='krestni'/>
  <xsl:param name='narozen'/>
  <xsl:param name='smrt'/>
  <xsl:param name='odkaz'/>
  <!-- Proměnná pro tisk iniciál -->
  <xsl:variable name='init'>
    <xsl:value-of select='ibs:cs-initial($jmeno)'/>
  </xsl:variable>
  <xsl:if test='$init!=$initial'>
    <xsl:text>\uminic </xsl:text>
    <xsl:value-of select='$init'/>
    <xsl:text>&#10;</xsl:text>
    <saxon:assign name='initial'>
      <xsl:value-of select='$init'/>
    </saxon:assign>
  </xsl:if>
  <!-- Tisk jména a dat -->
  <xsl:text>\umelec{</xsl:text>
```

```
  <xsl:call-template name='str-rep'>
    <xsl:with-param name='string'>
      <xsl:value-of select='$jmeno'/>
      <xsl:if test="$krestni!=''">
        <xsl:text> </xsl:text><xsl:value-of select='$krestni'/>
      </xsl:if>
      <xsl:call-template name='roky'>
        <xsl:with-param name='narozen'>
          <xsl:value-of select='$narozen'/>
        </xsl:with-param>
        <xsl:with-param name='smrt'>
          <xsl:value-of select='$smrt'/>
        </xsl:with-param>
      </xsl:call-template>
    </xsl:with-param>
  </xsl:call-template>
  <xsl:text>}\umpage </xsl:text>
  <!-- Rozeber odkazy, oddělovač je definován
       v LaTeXovém stylu -->
  <xsl:for-each select='saxon:tokenize($odkaz)'>
    <!-- Třídění není použitelné! -->
    <xsl:text>{</xsl:text>
    <xsl:value-of select='.'/>
    <xsl:text>}</xsl:text>
  </xsl:for-each>
  <!-- Konec řádku -->
  <xsl:text>\umpage&#10;</xsl:text>
</xsl:template>
```

Zbývá ještě vyřešit odkaz na správnou stránku. Používáme-li MakeIndex, máme v nesetříděném souboru čísla stránek uvedena. Pomocí XSLT však rejstřík vytváříme ještě před zahájením sazby a čísla stránek tudíž neznáme. Vytvoříme tedy jednoznačný identifikátor funkcí `generate-id{}`. Při sazbě textu jej použijeme jako jméno návěští v makru `\label`, v rejstříku jej použijeme v makru `\ref`. LaTeXovou část rejstříku tvůrců funerálního umění pak vysázíme těmito makry:

```
% Rejstřík umělců

\newenvironment{umelci}{\clearpage \let\Mezera\space
  \begin{multicols}{2}[\NadpisOddeleni
        {Tvůrci funerálního umění}]}%
  {\end{multicols}}
```

```
\def\uminic #1 {\par\vb\textbf{#1}\par\nobreak}

\def\umelec #1{\par
  \umpagetoks{}\def\delim{~\dots~\ignorespaces}\noindent
  \hangafter 1 \hangindent\parindent
  \rightskip 0mm plus 2em \relax #1}

\def\umprint #1.{\let\next\umprint
  \ifx \relax #1\let\next\par
  \else
    \delim #1\def\delim{, \ignorespaces}\fi \next}

\newtoks\umpagetoks

\def\umpage #1{\let\next\umpage
  \ifx \umpage #1\def\next{\expandafter
                \umprint\the\umpagetoks\relax.}\else
    \expandafter\ifx\csname r@#1\endcsname\relax
      \@latex@warning{Reference `#1' on page \thepage
                      \space undefined}\else
      \expandafter\expandafter\expandafter \edef
      \expandafter\expandafter\expandafter \test
      \expandafter\expandafter\expandafter
      {\expandafter\expandafter\expandafter
           \@secondoftwo\csname r@#1\endcsname}%
      \ifcat$\the\umpagetoks$\umpagetoks\expandafter{\test.}\else
        \def\next{\expandafter \expandafter \expandafter
             \umsortstart \expandafter \test
             \expandafter .\the\umpagetoks \relax.}\fi \fi \fi
  \next}

\def\umsortstart{\umpagetoks{}\umsort}
\def\umsort #1.#2.{\ifx \relax #2%
    \umpagetoks\expandafter
      {\the\umpagetoks #1.}\let\next\umpage\else
      \ifnum #1>#2
        \umpagetoks\expandafter{\the\umpagetoks #2.}%
        \def\next{\umsort #1.}\else
        \let\next\umsortend
        \ifnum #1=#2
          \umpagetoks\expandafter{\the\umpagetoks #2.}\else
```

```
          \umpagetoks\expandafter{\the\umpagetoks #1.#2.}%
        \fi
      \fi
    \fi \next}

\def\umsortend #1\relax.{\umpagetoks
    \expandafter{\the\umpagetoks #1}\umpage}
```

V původní verzi procesoru Saxon bylo zajištěno, že abecedně seřazené identifikátory získané funkcí `generate-id()` odpovídaly jejich pořadí v dokumentu. Později však z důvodu zvýšení rychlosti zpracování od toho bylo upuštěno [2]. V rejstříku ovšem chceme čísla stránek setříděná. Může se též stát, že je některý umělec na téže stránce uveden dvakrát. V rejstříku však nechceme duplikovaná čísla. Dosahujeme toho cviky s token registrem `\umpagetoks`. Analýzu příslušných maker ponecháváme čtenářům za domácí cvičení.

8. Závěr

Při sazbě knihy, použité v případové studii, se prokázalo, že XML může významně pomoci i tehdy, požadujeme-li pouze jediný výstupní formát. Navíc došlo k nasazení XML v čistě komerční oblasti, kde za špatné rozhodnutí platíme skutečnými penězi. Zde však XML pomohlo k časové, a tím i finanční úspoře. Současně byl vyvrácen mýtus, že běžná sekretářka není schopna psát dokumenty v LaTeXu, natož v XML. Nelze pochopitelně předpokládat, že si sekretářka naprogramuje LaTeXový makrobalík, transformační styl pro XSLT či formátovací objekty, ale pokud jí někdo potřebné nástroje dodá, pak se je naučí používat stejně, jako se naučila naklikat typ a velikost písma, zarovnání odstavce a jiné volby v nejrůznějších dialogových oknech kancelářského textového editoru. V tomto případě stačila patnáctiminutová instruktáž o základech XML a když si odmyslíme běžné drobné překlepy, napsala písařka celý text bez chyb.

9. Poděkování

Chtěl bych poděkovat Jiřímu Koskovi za přehledně psanou knihu *XML pro každého*, bez jejíhož přečtení bych asi podrobnější knihy o XML nepochopil, i za jeho cenné rady, které mi při práci na knize *Olšanské umění, jeho tvůrci a doba*, jež byla mým prvním větším projektem v XML, ochotně poskytoval. Dále děkuji Michaele Knapové, která, ač není odbornicí v informačních technologiích a nemá žádný specializovaný editor pro práci s XML, přepsala celý text z rukopisu v podstatě bezchybně.

Reference

[1] **DocBook web page**, http://www.oasis-open.org/docbook/
[2] Kay M. H.: Soukromé sdělení.
[3] Kay M. H.: **XSLT procesor Saxon**, http://users.iclway.co.uk/mhkay/
[4] Kosek J.: **XML pro každého**. Grada Publishing, 2000. http://www.kosek.cz/xml/
[5] Murray-Rust P., Rzepa H., Jirát J., Nic M., Wagner Z.: **Chemical Markup Language**, http://sourceforge.net/projects/cml/
[6] Neckářová L., Vanoušek A., Wagner J.: **Vinohradský hřbitov včera & dnes**. Správa pražských hřbitovů, Praha 2002. (Publikace je určena pro vnitřní potřebu členů Klubu Za starou Prahu a spolku Svatobor.)
[7] Olšák P.: **Program csr (Czech Sort) – abecední řazení podle normy**. Zpravodaj Československého sdružení uživatelů TEXu, **4** (3), 126–139 (1994). http://math.feld.cvut.cz/olsak/
[8] Rahtz S.: **PassiveTEX**, http://users.ox.ac.uk/~rahtz/passivetex/
[9] Vanoušek A: **Olšanské umění, jeho tvůrci a doba**. Správa pražských hřbitovů, Praha 2000. (Publikace je určena pro vnitřní potřebu členů Klubu Za starou Prahu a spolku Svatobor.)
[10] Wagner Z.: **Nástroje pro práci s XML/XSLT**, http://www.icebearsoft.cz/icebearsoft.euweb.cz/xml/

Summary: Exploatation of XML and LaTeX in Scientific Book Publishing

Scientific books usually have rigorous structure than texts in humanitites. This structure conveys information. Even though LaTeX enforces some kind of structured markup, nontrivial macroprogramming is needed in bigger applications. Some problems are caused during creation of indices with MakeIndex. Usage of XML strictly enforces document structure; one can take advantage of many existing tools for XML processing. Using LaTeX for the typesetting of XML transformed by XSLT is a natural choice. The approach is described on a case study that illustrates the whole publishing process.

Zdeněk Wagner
Ice Bear Soft
Email: wagner@cesnet.cz
Web: icebearsoft.euweb.cz