Technická univerzita v Liberci, Fakulta mechatroniky a mezioborových inženýrských studií

Prezentace XML dat telefonního seznamu školy

(ročníkový projekt)

Zadavatel: RNDr. Pavel Satrapa, Ph.D.
Katedra aplikované informatiky (KAI)

Vypracoval: Miroslav Mrázek

Zadání:
1. Vytvořit zdrojová XML data telefonního seznamu
2. Vytvořit XSLT stylu pro transformaci XML na HTML
3. Vytvořit XSL-FO styl pro zformátování XML dat do PDF souboru nebo k tisku.

Abstrakt:

Cílem této práce je seznámit čtenáře se základy značkovacího jazyka XML, s možnostmi formátování XML dat a s volně dostupným software, který toto formátování zajišťuje.

V praktické části jsou dopodrobna popsané dva skripty pro převod zdrojových XML dat do formátů (X)HTML a PDF.

V závěru je zachycen současný stav softwarové podpory pro transformaci XML dat do zformátované podoby.

Úvod:

Téma zpracování XML dat se dnes dostává do popředí zájmu profesionálů i odborné veřejnosti, v poslední době vzniká mnoho tajemných X-technologií (XML, XHTML, XSLT, XSL-FO) a tato práce si klade za cíl podat čtenáři ucelený přehled celé této rodiny technologií.

Poznámka:

Tento projekt byl zpracováván v prostředí Windows a z toho plynou i použitá kódování souborů. Vzhledem k tomu, že technologie XML je poměrně mladá a některé použité nástroje se teprve učí používat jazyky s diakritikou, budu se problému kódování češtiny věnovat podrobněji.

I. Teoretická část

I.1. Stručný úvod do značkovacích jazyků

V dnešní době je kladen čím dál větší důraz na popis u ukládání dat, informací je stále více a přestává být v lidských silách všechny pojmout. Zpracování těchto informací elektronicky sice velmi zvyšuje efektivitu, ale bez vhodného formátu pro popis dat se nemohou realizovat dostatečně silné a efektivní funkce pro vyhledávání, třídění, editaci, publikaci a v neposlední řadě i opětovné sdílení..

Jak už z předchozího odstavce vyplývá, čím více dat s vysokým informačním obsahem, tím větší důraz je kladen na použité technologie. Pokud jsme si na vlastním disku vystačili s .txt a .doc soubory v adresářové struktuře, na informačních serverech a na internetu obecně je na způsob a formát uložení dat kladen nejvyšší důraz.

Jednou z osvědčených možností uložení a přístupu k datům je uložit tato do **databáze**, dnes v drtivé většině databáze relační nebo objektově relační. Tyto technologie již vyspěly natolik, že většina profesionálních i amatérských projektů je postavena na nich. Vzhledem k existenci velkého množství relačních DB si každý může vybrat tu nejvhodnější pro jeho práci. Některé (např. MySQL) jsou k dispozici i zcela zdarma.

Základní myšlenkou databázových systémů je **znovuvyužitelnost** dat. Máme-li tedy databázi odběratelů pro aplikaci, která vyplňuje a rozesílá faktury, můžeme si z dat v databázi vytvořit například telefonní seznam odběratelů. Bude to jiná aplikace, ale využívat bude stejná data, která jí zprostředkuje řídící databázový systém. Pokud by aplikace FAKTURY používala vlastní formát pro uložení dat, musela by ho dopodrobna znát i aplikace TELEFONY. Řídící databázový systém zajišťuje veškerou agendu ohledně databáze. Obstarává zabezpečení,

vyhledávání, spojování a seskupování dat, a navíc tyto služby poskytuje síťově.
Data se v relační databázi ukládají do relací. Teoreticky ne zcela přesné, ale výstižnější je označení tabulka. U relace, na rozdíl od tabulky, nezáleží na pořadí řádků a je možné používat relační algebru při popisu dat..
Zdálo by se tedy, že hledat další možnosti správy dat je zbytečné, ale i relační databáze mají své nevýhody. Jednou z nich je fakt, že data jsou uložena v jednom nečitelném souboru. Každá aplikace tedy musí používat databázový systém, aby se dostala k datům. Také při poškození datového souboru je jeho obnova věcí profesionálů. Do relací se špatně ukládají některé typy dat a databáze neřeší problém exportu a výměny dat mezi aplikacemi.

Z výše uvedených důvodů vznikly tzv. **značkovací jazyky**, tedy způsob zápisu dodatečných informací do textového dokumentu. Pro zpracování mnoha typů dat není zkrátka zapotřebí matematická síla relační algebry, některé informace se navíc do relací špatně zapisují. Jedná se o převážně textová data, jako INI soubory, články, knížky, data se stromovou strukturou, objekty s různým počtem atributů..
Značky, označované také jako TAGy, mohou nést obecně jakékoliv informace, většinou formátovací, nebo data popisující. Při popisu formátovacích stylů se do tagů zapisuje i kód nějakého jazyka.

První abstraktní značkovací jazyk *(**GML** - generalized markup language)* vytvořili pánové Charles Goldfarb, Edward Mosher a Raymond Lorie. Jazyk byl určen k popisu konkrétních značek, které se pak mohly zapisovat do dokumentu; dokumenty používající stejnou sadu značek pak byly navzájem kompatibilní a bylo možné je zpracovávat jednotně. Tento princip se používá dodnes, samotné GML však bylo příliš obecné.

Společnosti ANSI a GCA pak v rámci projektu ODA *(Open Dokument Architecture)* vyvinuly v 80.letech jazyk **SGML** *(Standart GML)*, který vyl roku 1986 přijat jako norma ISO 8879.
SGML je otevřeným standardem, nezávislým na platformě, nebo aplikacích. Soubory SGML jsou ukládány jako text ASCII, což zajišťuje jejich použitelnost prakticky na libovolné počítačovém zařízení. Význam a přínos SGML se objevuje ve chvíli, kdy je dokument označen příslušnými značkovacími příkazy. Definováním struktury a vnitřních vztahů v do té doby nestrukturovaných datech vznikla informace, které se otevírají zcela nové možnosti zpracování, publikování a opakovaného používáni. SGML databáze může obsahovat tisíce označkovaných dokumentů, které se následně formátují stejným stylem do požadované výstupní podoby. Tím stylovacím jazykem je v případě SGML metajazyk DSSSL *(Dokument Style Semantics and Specification Language)*.
Všechny následné značkovací jazyky jsou psány v SGML ((X)HTML, XML, XSL, DocBook, aj.)
SGML se však neomezilo jen na popis textových dat, specializované značkovací jazyky umožňují popsat vektorovou grafiku, matematické či chemické vzorce a jistě spoustu dalších dat. Ve formátu SGML je údajně popsána i dokumentace k letadlu Boeing 747.

I přes ohromnou dokumentační sílu se jazyk SGML příliš nerozšířil mezi prostý počítačový lid, důvodem byla jeho komplexnost a složitost. Nicméně jedna aplikace tohoto jazyka se ujala v nevídaném rozsahu, a to jazyk **HTML** *(HyperText ML)*. Byly i doby, kdy si prostý uživatel pod pojmem Internet představil jen webové stránky.
Přestože HTML byl navržen jako univerzální formát pro prezentaci dat, časem „zdegeneroval" na formát popisující hlavně jak se mají části dokumentu zobrazit, nikoliv co znamenají. Typickou vlastností HTML prohlížečů je tolerance vůči chybám, aplikace se snaží zobrazit i špatně popsanou stránku, chybné tagy a atributy ignorovat, nebo se

snažit domyslet, co měl autor na mysli. Z toho ale plynulo i rozdílné zobrazení v jednotlivých prohlížečích.
Jednoduchost a rozšířenost HTML tak pomohla naplnit Internet obrovským množstvím dat, která se však bohužel velice složitě elektronicky dále zpracovávají a probírají. Data se dají prohledávat jen fulltextově, prohlížeč nezná jejich význam.
Všechny tyto okolnosti vedly k hledání nového formátu, který by byl jednodušší než SGML, a přitom umožnil oddělit data od jejich vzhledu a přiřadit jim význam. Tedy ukládat *informace* místo holých *dat*.

Tímto formátem se stal **XML** *(eXtensible ML)*, potomek SGML, který si zachovává svojí rozšiřitelnost, ale je výrazně jednodušší a je určen k *popisu* převážně textových dat a k *přenosu* dat mezi aplikacemi.
Tento metajazyk umožňuje definovat značky a strukturu dokumentu *(**DTD** – Data Type Definition, Definice Typu Dat)*, dokumenty vyhovující stejnému DTD se následně mohou zformátovat do finální podoby pomocí tzv. stylovacích jazyků *(SSL - StyleSheet Languages)*.

I.2 Oddělení obsahu od formy – vznik stylových jazyků

Jedním z hlavní úkolů popisovaných technologií je důsledné oddělení obsahu od formy. Datové dokumenty tak mají velmi vysokou informační hodnotu, přidružený styl popisuje zas jen formát dat, takže změna vzhledu všech dokumentů s odpovídajícím DTD vyžaduje jen změnu v jednom textovém souboru.

Mezi nejstarší patří **FOSI** *(Formatting Output Specification Instance)*, jehož autorem je americké ministerstvo obrany. FOSI je podporováno mnoha SGML aplikacemi. Výhoda je, že FOSI styl je sám o sobě SGML dokumentem, takže pro jeho editování lze využívat stávající nástroje. I tato myšlenka provází XML technologie dodnes. FOSI je však pouze deklarativní, nelze v něm provádět žádné výpočty nebo přeskupování dokumentu pro potřeby výstupu.

Nástupcem FOSI byl jazyk **DSSSL** *(Document Style Semantics and Specification Language)*, který byl v roce 1995 přijat za ISO normu. Je to kompletní programovací jazyk. Kromě specifikování vzhledu jednotlivých elementů tak můžeme provádět různé úpravy dokumentu před jeho výsledným zformátováním. Typickým příkladem je například generování obsahu dokumentu nebo změna struktury a pořadí elementů před jejich vytištěním.

Další z jazyků pro popis stylu vznikl pro potřeby jazyka HTML, používá však i pro formátovaní XML. Jedná se o **CSS** *(*Cascading Stylesheets*)*. Kaskádové styly měly být nástrojem umožňující psaní čistějšího HTML kódu. Stejně jako FOSI, neumělo CSS transformaci dokumentu, bylo však určeno především k oddělení popisu stylu. Celá sada stránek tak může být zformátovaná jednotně, změna vzhledu je záležitostí zásahu do jediného souboru.

V mnoha aplikacích je však vhodné, můžeme-li před zobrazením, či vytištěním, dokument nějak modifikovat. V tomto případě své uplatnění nalezne jazyk **XSL** *(eXtensible Stylesheet Language)*. Jazyk XSL, vznikl až speciálně pro potřeby jazyka XML. Pro zápis stylu se využívá jazyk XML, takže lze použít běžné XML editory. XSL umožňuje dvě základní operace: transformovat jeden XML dokument do jiného XML dokumentu a specifikovat formátovací vlastnosti jednotlivých částí dokumentu.

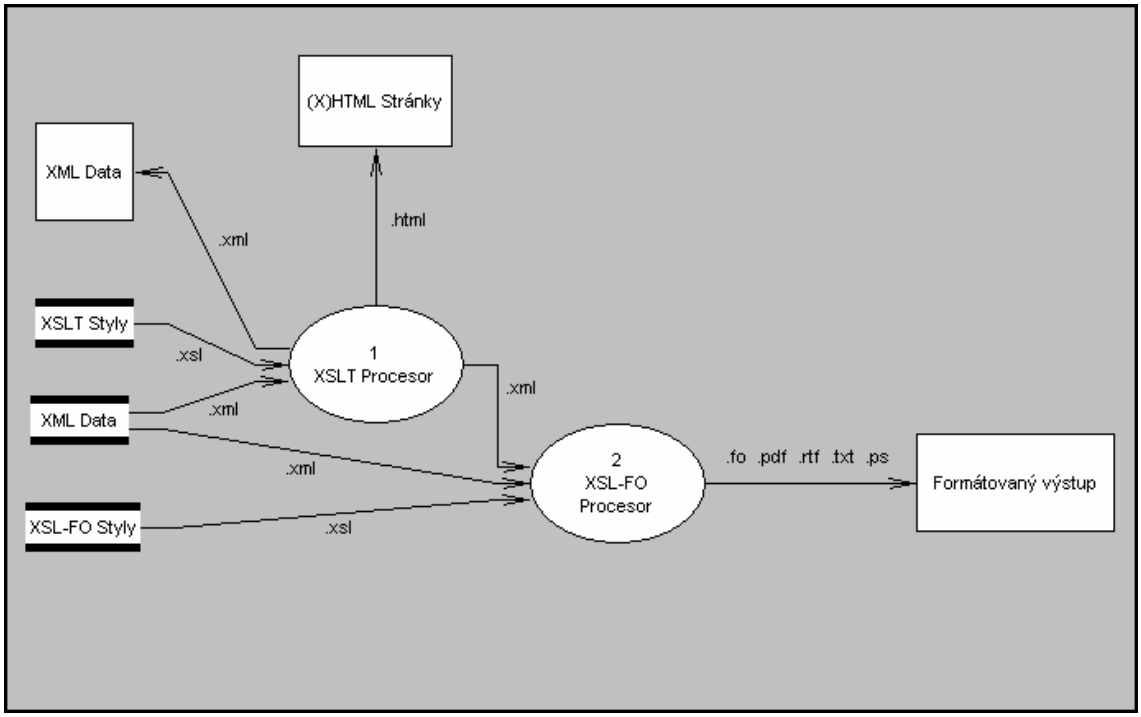
Transformační jazyk se označuje jako **XSLT** (*XSL Transformations*) a je určen k popisu převodu XML dokumentu do jiného, tedy i do XHTML. Umožňuje generovat obsah, třídit a seskupovat data, zajišťuje převod na jinou DTD.

Formátovací model **XSL-FO** vychází z DSSSL a je poměrně komplexní. Protože se jedná o velmi mladou technologii, má zatím jen malou softwarovou podporu. Pomocí formátovacích objektů je možné definovat typograficky správný a na platformě nezávislý vzhled dokumentu, procesor FO pak převede dokument do libovolného formátu. Taková je tedy idea, reálně dnes formátovací procesory převádějí většinou jen do .pdf a .rtf, nebo .ps formátu pro tiskárnu. Převod do HTML zatím žádný FO procesor neumožňuje, HTML se z XML generuje pomocí XSLT.

Z dalšího diagramu by mělo být zřejmé, jak se v celém procesu formátování XML souborů pohybují data.

XSLT procesor nám může generovat jakýkoliv výstup v XML formátu, tedy i XHTML stránky, buď již naformátované, nebo připravené na použití spolu CSS. Jeden soubor může projít XSLT procesorem i vícekrát, jeden skript vybere data pro další zpracování, druhý je převede na požadované DTD, jiný zajistí XHTML výstup..

Procesor formátovacích objektů převádí XML pomocí XSL do FO formátu, ze kterého pak generuje požadovaný výstup.



I.3. Softwarová podpora - co všechno budeme potřebovat

Cílem této práce je vytvoření formátovacích stylů pro data telefonního seznamu fakulty. K tomu je zapotřebí alespoň základní programové vybavení, konkrétně textový editor, XSLT procesor a XML-FO procesor.

Zmiňovat **textový procesor** není tak zbytečné, jak to zní. Ten pro práci s XML musí zvládat UTF-8 kódování. XML dokumenty totiž mají být co nejméně závislé na čemkoliv, různé kódování znaků by vnášelo chaos tam, kde se XML snaží o pořádek a jednotnost. UTF je kódování s proměnnou délkou znaků, standardní ASCII znaky se kódují do jednoho byte (shodně jako ASCII, je tedy zpětně kompatibilní), ostatní znaky zabírají 2 až 8 (16 pro UTF-16) byte, speciální české znaky jsou kódovány do 2 B.
S UTF umí pracovat i Notepad ve WinXP, interní prohlížeč v Total Commanderu a celá řada specializovaných editorů pro programátory.
Já jsem použil UltraEdit32 v demonstračním režimu (program je šířen jako shareware, ke stažení je k dispozici například na Slunečnici www.slunečnice.cz). Tento editor zvládá editovat dokumenty v různých znakových sadách a převádět dokumenty mezi ISO, Windows a UTF kódováním.

K převodu XML dokumentu na HTML stránku je nutný **XSLT procesor**. Těch je k dispozici hned několik, nejpoužívanější je Saxon (http://saxon.sourceforge.net/) a Sablotron (původně český projekt Šablotron), který je integrován do PHP (http://cz2.php.net/xslt). Já jsem si zvolil kvůli jednoduchosti instalace (de-facto žádná) a malé velikosti procesor LibXSLT (http://xmlsoft.org/XSLT/), který je obsažen i na přiloženém CD. S kódováním tu žádný problém nenastává, celý proces probíhá v UTF, výstupem je opět soubor v UTF.

Největší problém je sehnat kvalitní **XSL-FO procesor**. Formátovací objekty jsou tak mladá technologie, že veškerý software pro ně se teprve vyvíjí. Vzniká jich několik (UFO, XEP, XSL Formatter), ale jen jediný je nekomerční a zároveň nezávislý na OS, a to FOP (FO Processor) (http://xml.apache.org/fop/).

FOP si navíc klade za cíl širokou podporu výstupních formátů (PDF, PCL, PS, SVG, XML, Print, AWT, MIF a TXT). Primárně je však zaměřen na .pdf, a v tuto chvíli dobře zvládá jenom ten.

Další problém nastává opět s češtinou, tentokrát již ne s kódováním, ale s českými fonty. FOP v poslední verzi (0.20.5) už zvládá práci s externími fonty, bohužel vygenerovaný .pdf soubor obsahuje chyby, které s použitím vestavěných fontů nenastávají. Ostatní procesory jsou na tom s podporou externích fontů podobně. Zkoušel jsem použít i XEP, ale ten zvládá české fonty jen zobrazit (a poslat na tiskárnu), pro převod do .pdf je použít neumí.

II. Vlastní práce

II.1. Nastudování zadané problematiky

Tento bod bych označil za stěžejní v celém projektu. Jak již bylo zmíněno, XML technologie jsou teprve v začátcích a některé ještě nebyly standardizovány. Studijních materiálů tedy není mnoho, a v češtině jich je ještě méně. V této kapitole se tedy budu věnovat tomu kde čerpat informace..

Vzhledem k okolnostem je asi jasné, že papírové knihy jsou naprosto nevyhovující, situace se rok od roku mění. A protože XML je počítačová záležitost, úzce související s internetem, bude nutné hledat studijní materiály tam..

Zdroje v češtině:

Nejobsáhlejším zdrojem u nás, jsou WWW stránky Jiřího Koska (http://www.kosek.cz/index.html), na kterých jsou materiály, popisující základy XML, DTD, XSLT a v rámci potřeb pro DocBook i XSL-FO. Rozhodně je to v současné době nejlepší startovací místo.

Pokud jsme již vstřebali základní XML kašičku, můžeme začít tvořit první skripty a pokusná data. V této fázi nejspíše oceníme slajdy, ve kterých jsou popsány nejdůležitější konstrukce (http://badame.vse.cz/izi238/prednasky.html), autorem je opět Jiří Kosek.

Další informace můžeme získat z článků na odborných serverech, např. Softwarové noviny, Root, nebo Živě, většinou však obsahují méně než články J. Koska. Obsahují však diskuze k článkům, ve kterých se dají nalézt odpovědi na konkrétní otázky.

Přímo o formátovacích objektech jsem žádný článek nenašel, ale tato problematika se týká i jazyka DocBook (univerzální DTD pro popis textových dokumentů typu článek nebo kniha, na který existují kvalitní XSL styly pro zformátování do prezentačního formátu), o kterém se již něco najít dá. Kromě článků J. Koska jsem našel

popsané základy formátovacích objektů např. v této práci (http://nb.vse.cz/~zelenyj/it380/eseje/xdroj03/xslfo.htm).
Srovnání několika XSL-FO procesorů od Pavla Žampacha je na (http://www.volny.cz/zampach/dbk/fo_pdf.html)

Zdroje v angličtině:

Chceme-li podrobnější informace o vybraných technologiích, musíme navštívit zahraniční servery.
Patronát nad XML převzalo konsorcium W3C (http://www.w3.org/), takže oficiální informace k jednotlivým formátům nalezneme tam.
Problematice XML se také věnuje společnost Apache Group (http://xml.apache.org/).
Na těchto stránkách nalezneme kompletní reference zmiňovaných jazyků.
Pokud však narazíme na problém, se kterým si nevíme rady, zbývá Google a pokusit se najít odpověď v nepřeberné pokladnici diskusních fór z celého světa. Málokdy se stane, že problém, na který jsme narazili, ještě nikdo na internetu nezmiňuje..

II.2. Vytvoření zdrojových XML dat

Tento projekt vzniká společně s diplomovou prací, která řeší správu a zpracování telefonního seznamu univerzity. Data jsou uložená v relační databázi a zpracovávaná pomocí PHP. Pro převod do XML jsem použil částečně PHP skript, finální úpravy jsem provedl ručně.
Při ukládání výsledku do souboru nastal problém s češtinou, PHP sice obsahuje funkce pro převod do UTF-8, bohužel však pouze z kódování ISO 8859-1, tedy bez podpory češtiny. Vyřešil jsem to tak, že jsem soubor uložil ve Windows kódování a soubor pak převedl na UTF pomocí UltraEditu.

XML dokument je sestavován podle následující DTD:

```
<?xml version="1.0"?>
<!DOCTYPE telefony [
 <!ELEMENT seznam (pracoviste | funkce | zamestnanec)*>
 <!ELEMENT pracoviste (pracoviste | funkce | zamestnanec)*>
 <!ELEMENT funkce (pracoviste | funkce | zamestnanec)*>
 <!ELEMENT zamestnanec (titul1?, jmeno?, jmeno2?, prijmeni,
     titul2?, telefony, email?)>
  <!ELEMENT titul1 (#PCDATA)>
  <!ELEMENT jmeno (#PCDATA)>
  <!ELEMENT jmeno2 (#PCDATA)>
  <!ELEMENT prijmeni (#PCDATA)>
  <!ELEMENT titul2 (#PCDATA)>
  <!ELEMENT telefony (#PCDATA)>
  <!ELEMENT email (#PCDATA)>

 <!ATTLIST seznam name PCDATA #REQUIRED>
 <!ATTLIST pracoviste nazev PCDATA #REQUIRED
              zkratka PCDATA #REQUIRED
              strom PCDATA #REQUIRED>
 <!ATTLIST funkce nazev PCDATA #REQUIRED>
 <!ATTLIST zamestnanec id PCDATA #REQUIRED>
]>
```

II.3. Vytvoření XSLT stylu pro HTML výstup

hlavička souboru

```
<?xml version="1.0"  encoding="UTF-8"?>
```

hlavička stylu

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="xml" encoding="windows-1250" omit-xml-
declaration="yes"/>
```

definice vlastních entit pro vložení mezery a konce řádku v HTML kódu

```
<!DOCTYPE xsl:stylesheet [
<!ENTITY nbsp " ">   <!-- (nedelitelna) mezera -->
<!ENTITY eol "<xsl:text>&#xA;</xsl:text>">   <!-- konec radku -->
]>
```

definice klíčů pro řazení a seskupování (viz http://www.kosek.cz/xml/xslt/index.html, kapitola 8.1)

```
<xsl:key name="zamestnanci" match="//zamestnanec" use="@id"/>
<xsl:key name="zkratky" match="//pracoviste" use="zamestnanec/@id |
funkce/zamestnanec/@id"/>
```

pak už jen nadpis a velice důležitý příkaz <xsl:apply-templates/>, který zajistí volání šablon pro vnořené uzly.

Nakonec výsledného dokumentu zařadíme ještě jmenný seznam všech zaměstnanců:

```
   <div class="nadpis">Abecední seznam:</div>
```

V rejstříku nás zajímá vždy jen 1. výskyt dané osoby; kontroluje se podle atributu zamestnanec@id, který je jedinečný

```
   <xsl:for-each select="//zamestnanec[generate-id(.) = generace
   id(key('zamestnanci',./@id)[1])]">

   <xsl:sort select="prijmeni" lang="iso-8859-2"/>
        <xsl:sort select="jmeno"/> <!-- seradime vysledky -->
```

Výsledky ještě seřadíme podle abecedy a zapíšeme do výsledného dokumentu.

```
   <div class="zamestnanec">
<xsl:if test="prijmeni != ''">
<span class="prijmeni"><xsl:value-of select="prijmeni"/></span></xsl:if>
<span class="jmeno">
<xsl:if test="jmeno != ''"> <xsl:value-of select="jmeno"/></xsl:if>
<xsl:if test="jmeno2 != ''"> <xsl:value-of select="jmeno2"/></xsl:if>
<xsl:if test="titul1 != ''">, <xsl:value-of select="titul1"/></xsl:if>
<xsl:if test="titul2 != ''">, <xsl:value-of select="titul2"/></xsl:if>
```

Pomocí následující konstrukce zjistíme zkratky pracovišť, na kterých dotyčný zaměstnanec pracuje (vyhledají se jen pracoviště z aktuálního seznamu, nikoliv z celé databáze!)

```
    ( <xsl:for-each select="key('zkratky', ./@id)">
   <xsl:value-of select="@zkratka"/> </xsl:for-each>)</span>
   <xsl:if test="telefony != ''"><span class="telefony"><xsl:value-of
select="telefony"/></span></xsl:if>
  </div>
  </xsl:for-each>
```

Pomocí šablon pro vnořené uzly převedeme do XHTML i ostatní elementy:

```
  <!-- Sablona pro element <pracoviste>: -->
  <xsl:template match="pracoviste[count(.//zamestnanec) > 0]">
```

Do výpisu nechceme zařazovat větve, které nemají žádného zaměstnance, pracoviště očíslujeme (víceúrovňové číslování zpřehlední výsledky).

```
  <div class="pracoviste"><xsl:number level="multiple"/> - <xsl:value-of
select="@nazev"/> (<xsl:value-of select="@zkratka"/>)</div>
  <xsl:apply-templates/>
 </xsl:template>


<!-- Sablona pro element <funkce>: -->
<xsl:template match="funkce">
  <div class="funkce"><xsl:value-of select="@nazev"/>:</div>
    <xsl:apply-templates/>
</xsl:template>
```

Šablona pro vypsání každého zaměstnance; protože XSLT ani XHTML nepodporuje žádný objekt pro vložení teček, musíme (kvůli přehlednosti) nastavit podtržení celého řádku v přiloženém CSS stylu

```
<!-- Sablona pro element <zamestnanec>: -->
<xsl:template match="zamestnanec">
  <div class="zamestnanec">
  <xsl:if test="prijmeni != ''"><span class="prijmeni"><xsl:value-of
select="prijmeni"/></span></xsl:if>
<span class="jmeno">
<xsl:if test="jmeno != ''"> <xsl:value-of select="jmeno"/></xsl:if>
<xsl:if test="jmeno2 != ''"> <xsl:value-of select="jmeno2"/></xsl:if>
<xsl:if test="titul1 != ''">, <xsl:value-of select="titul1"/> </xsl:if>
<xsl:if test="titul2 != ''">, <xsl:value-of select="titul2"/>
</xsl:if></span><span class="telefony">
<xsl:if test="telefony != ''"><xsl:value-of
select="telefony"/></xsl:if></span></div>
</xsl:template>

A to je vse..
</xsl:stylesheet>
```

Styl byl vytvořen v textovém editoru UltraEdit32.
Zformátovanou HTML stránku si můžete prohlédnout na přiloženém CD, jedná se o soubor `seznam.html.original` Tento soubor není *well-formed*, protože použitý XSLT styl je určen pro vytvoření XHTML fragmentu, který bude dynamicky vložen do stránky. Chybí tedy všechny hlavičky i určení typu dokumentu. Tyto náležitosti jsou obsaženy v souboru `sablona.txt`. Tu jsem aplikoval ručně a vznikl výsledný soubor `seznam.html`.
Formátování zajišťuje soubor kaskádových stylů (stylxml.css), ve kterém je definován vzhled použitých elementů a jejich tříd (atribut *class* formátovaných elementů)
Tento styl zajišťuje kromě kosmetických úprav i sloupcovou úpravu telefonních čísel a vykreslení řádků, pro lepší orientaci mezi výsledky:

```
span.popis {font-weight: bold; color: black}
span.jmeno {color: navy}
span.prijmeni {font-weight: bold; color: navy}
```

Odsazení telefonních čísel pomocí absolutního pozicování:

```
span.telefony {font-style: italic; position: absolute; left: 400px}
span.email {font-style: italic; color: navy}

div.pracoviste {font-weight: bold; text-transform:uppercase; color:
green; line-height: 1.5}

div.funkce {font-style: italic; color: black; line-height: 1.2;
position: relative; left: 15px}

div.nadpis {font-style: italic; font-weight: bold; font-size: large;
text-decoration: underline; line-height: 2.0}
```

Podtržení každého řádku výsledků - atribut *border-bottom* :

```
div.zamestnanec {border-bottom: 1px solid gray}
```

II.4. Vytvoření XSL-FO stylu pro výstup do .pdf

soubor je opět XML dokument s příslušnou hlavičkou

```
<?xml version="1.0" encoding="UTF-8"?>
```

v hlavičce stylu definujeme dva jmenné prostory podle dvou různých DTD, a to standardní XSLT jako :xsl a formátovací objekty :fo

```
<xsl:stylesheet version="1.1"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format" exclude-result-prefixes="fo">
  <xsl:output method="xml" version="1.0" omit-xml-declaration="no"
indent="yes"/>
```

definice klíčů pro řazení a seskupování, stejne jako v minulém stylu

```
<xsl:key name="zamestnanci" match="//zamestnanec" use="@id"/>
<xsl:key name="zkratky" match="//pracoviste" use="zamestnanec/@id |
funkce/zamestnanec/@id"/>
```

v šabloně kořenového elementu definujeme vzhled celého dokumentu,

```
<!-- Sablona pro korenovy element: -->

  <xsl:template match="/">
    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
      <fo:layout-master-set>
```

tedy rozměry stránky,

```
        <fo:simple-page-master master-name="mojeA5" page-height="210mm"
page-width="150mm" margin-top="1cm" margin-bottom="1cm" margin-left="1cm"
margin-right="1cm">
```

a rozložení stránky do regionů, v našem případě dvou, hlavní pro text

```
          <fo:region-body margin-bottom="0.5cm"/>
```

a pod ním pak řádek pro číslo stránky

```
          <fo:region-after extent="0.3cm"/>
        </fo:simple-page-master>
      </fo:layout-master-set>
```

následující kód slouží k očíslování stránek do příslušného regionu

```
      <fo:page-sequence master-reference="mojeA5" initial-page-number="1">
        <fo:static-content flow-name="xsl-region-after">
          <fo:block text-align="center" font-size="7pt" font-style="italic">
            -<fo:page-number/>-
          </fo:block>
        </fo:static-content>
```

pak už přijde jen nadpis

```
   <fo:flow flow-name="xsl-region-body" font-family="CourierNew"
     font-size="8pt">
  <fo:block text-align="center" text-decoration="underline" font-size="10pt"
font-weight="bold" space-after="4mm"><xsl:value-of
select="seznam/@name"/></fo:block>
```

a příkaz pro zformátování vnořených elemenů

```
        <fo:table table-layout="fixed">
          <fo:table-column column-width="130mm"/>
           <fo:table-body>
            <xsl:apply-templates/>
           </fo:table-body>
          </fo:table>
```

Přidáme abecední seznam,

```
       <fo:table table-layout="fixed">
        <fo:table-column column-width="130mm"/>
         <fo:table-body>
         <fo:table-row keep-with-previous="always">
           <fo:table-cell>
           <fo:block text-align="center" text-decoration="underline" font-
size="10pt" font-weight="bold" space-after="4mm">Abecední seznam:</fo:block>
            </fo:table-cell>
           </fo:table-row>
```

generovaný pomocí konstrukce FOR-EACH a klíče *zaměstnanci,* aby se předešlo vícenásobnému výpisu zaměstnanců na více pozicích.

```
     <xsl:for-each select="//zamestnanec[generate-id(.) = generate-
id(key('zamestnanci',./@id)[1])]">
```

Výsledky seřadíme podle abecedy

```
<xsl:sort select="prijmeni" lang="utf-8"/>
        <xsl:sort select="jmeno"/>
```

A vypíšeme do tabulky (v tabulce se dá nastavit nezalamování odstavců – viz níže)

```
   <fo:table-row keep-with-previous="always">
   <fo:table-cell>
   <fo:block text-align-last="justify">
   <xsl:if test="prijmeni != ''"><xsl:value-of select="prijmeni"/></xsl:if>
   <xsl:if test="jmeno != ''"> <xsl:value-of select="jmeno"/></xsl:if>
   <xsl:if test="jmeno2 != ''"> <xsl:value-of select="jmeno2"/>
         </xsl:if>
   <xsl:if test="titul1 != ''">, <xsl:value-of select="titul1"/>
         </xsl:if>
   <xsl:if test="titul2 != ''">, <xsl:value-of select="titul2"/>
         </xsl:if>
   <xsl:for-each select="key('zkratky', ./@id)"> 
         (<xsl:value-of select="@zkratka"/>)</xsl:for-each>
   <fo:leader leader-pattern="dots"/>
   <xsl:value-of select="telefony"/>
   </fo:block>
   </fo:table-cell>
   </fo:table-row>
   </xsl:for-each>
   </fo:table-body>
   </fo:table>
```

Na samotný konec dokumentu přidáme ještě obsah,

```
<fo:block text-align="center" text-decoration="underline" font-size="10pt"
font-weight="bold" space-after="4mm"  break-before="page">Obsah:</fo:block>
          <xsl:for-each select="//pracoviste">
            <fo:block text-align-last="justify" font-weight="bold">
```

unikátní ID nám tentokrát jednoduše zajistí FO funkce *generace-id,*

```
              <fo:basic-link internal-destination="{generate-id(.)}">
                <xsl:value-of select="@nazev"/>
              </fo:basic-link>
```

pro vytečkování mezery použijeme FO element *leader*

```
              <fo:leader leader-pattern="dots"/>
```

zjištění čísla stránky pomocí našeho ID zajistí tento řádek

```
              <fo:page-number-citation ref-id="{generate-id(.)}"/>
            </fo:block>
```

a celé to zopakujeme pro ostatní elementy, které chceme v obsahu mít, tentokrát jen nebudeme používat tučné písmo a text o 1 znak odsadíme

```
          <xsl:for-each select="funkce">
            <fo:block start-indent="1em" text-align-last="justify">
            <fo:basic-link internal-destination="{generate-id(.)}">
             <xsl:value-of select="@nazev"/>
            </fo:basic-link>
          <fo:leader leader-pattern="dots"/>
          <fo:page-number-citation ref-id="{generate-id(.)}"/>
             </fo:block>
             </xsl:for-each>
           </xsl:for-each>


        </fo:flow>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>
```

V šablonách pro vnořené objekty musíme vyřešit mimo jiné i problém stránkování, není pěkné, pokud nám procesor vloží konec stránky mezi nadpis a příslušný text, v našem případě je stránkování nastaveno tak, aby celý blok FUNKCE byl nedělitelný.

Řízení stránkování se provádí nastavením atributů *keep-with-next* a *keep-with-previous* na hodnotu *always.*

Ačkoliv podle W3C specifika má být tento atribut dostupný i element *fo:block*, dostupné procesory zvládají pracovat s tímto atributem jen u řádků tabulky. Abychom se o tuto formátovací možnost nepřipravili, musíme tedy umístit PRACOVISTE, FUNKCE i ZAMESTNANCE do tabulky s jednou buňkou na každém řádku, a u každého nadpisu nastavit *keep-with-next=*"*always*"

```
<!-- Sablony pro vnorene elementy: -->

  <xsl:template match="pracoviste">
    <fo:block>
      <fo:table table-layout="fixed">
        <fo:table-column column-width="130mm"/>
        <fo:table-body>
          <fo:table-row  keep-with-next="always">
            <fo:table-cell>
              <fo:block font-weight="bold" font-size="10pt" space-
before="3mm" id="{generate-id(.)}">
                <xsl:value-of select="@nazev"/> (<xsl:value-of
select="@zkratka"/>)
              </fo:block>
            </fo:table-cell>
          </fo:table-row>
          <xsl:apply-templates/>
        </fo:table-body>
      </fo:table>
    </fo:block>
  </xsl:template>

        .

        .

        .
```

Pokud budeme chtít stejným způsobem řešit i nezalamování jednotlivých ZAMESTNANCU, zjistíme, že to nefunguje. Celý seznam na sebe totiž navazuje, není jednoduchý způsob, jak atribut u posledního ZAMESTNANCE nenastavovat, bloky se tedy nemohou zalamovat ani mezi sebou a dokument je nestránkován, jako bychom nic neměnili. Dlouho mi trvalo, než jsem zjistil, že řešením je použít u řádků se zaměstnanci atribut *keep-with-previous,* text je pak spojen jen mezi sebou a nadpisem a stránkování funguje dle očekávání.

```
<xsl:template match="zamestnanec">
    <fo:table-row keep-with-previous="always">
      <fo:table-cell>
        <fo:block text-align-last="justify">
        <xsl:if test="prijmeni != ''"><xsl:value-of
select="prijmeni"/></xsl:if>
        <xsl:if test="jmeno != ''"> <xsl:value-of
select="jmeno"/></xsl:if>
        <xsl:if test="jmeno2 != ''"> <xsl:value-of
select="jmeno2"/></xsl:if>
        <xsl:if test="titul1 != ''">, <xsl:value-of
select="titul1"/></xsl:if>
        <xsl:if test="titul2 != ''">, <xsl:value-of
select="titul2"/></xsl:if>
        <xsl:if test="../@abc != ''"> (<xsl:value-of
select="pzkratka"/>)</xsl:if>
        <fo:leader leader-pattern="dots"/>
        <xsl:value-of select="telefony"/>
       </fo:block>
      </fo:table-cell>
    </fo:table-row>
  </xsl:template>
   </xsl:stylesheet>
```

Výsledný dokument je na doprovodném CD v souboru seznam.pdf. Písmo *Courier* jsem použil proto, že s jeho českou verzí generoval FOP méně chyb než s jinými českými fonty.

III. Závěrečné zhodnocení

Hlavním cílem bylo nastudování problematiky XML obecně a XSL zvláště. Dále pak vytvořit styly pro převod zdrojového XML souboru do nejpoužívanějších publikačních formátů HTML a PDF.

Myslím, že se vytyčené cíle podařilo splnit. Na základě předchozího samostatného studia jsem vytvořil dva XSL styly, XML2HTML a XML2FO, které pomocí různých technologií (XSLT a XSL-FO) generují podobné výsledky. K výběru dat pro další zpracování je v obou - případech použit jazyk XPath.

Tento projekt připravuje nástroje, které budou dále využité při tvorbě diplomové práce, tedy internetové aplikace pro správu a zprostředkování dat telefonního seznamu školy.

Výsledky práce jsou obsaženy na přiloženém CD.
Adresář *software* obsahuje veškeré použité programy, *XML2HTML* a *XML2FO* obsahují XML data, příslušné styly, vhodný procesor a dávkový soubor, zajišťující převod dokumentu data.xml na seznam.html (.pdf). Bohužel, FOP s českými fonty generuje v .pdf souboru chyby(obzvláště ve velkých souborech – seznamFM.pdf), zbavil jsem XML diakritiky a nechal je přeložit pomocí integrovaného fontu. Výsledný soubor (seznamFM-BD.pdf) je nejenom bezchybný, ale také velmi malý oproti dokumentu s českými znaky.

IV. Použité informační zdroje

Články Jiřího Koska (http://www.kosek.cz/)

Seriál o XML pro Softwarové noviny:

http://www.kosek.cz/clanky/swn-xml/index.html

XML schémata:

http://www.kosek.cz/xml/schema/

XSLT v příkladech:

http://www.kosek.cz/xml/xslt/index.html/

Podpora češtiny pro FOP:

http://www.kosek.cz/sw/fop/index.html

Slajdy k přednáškám:

http://badame.vse.cz/izi238/prednasky.html

Formátovací objekty XSL:

http://nb.vse.cz/~zelenyj/it380/eseje/xdroj03/xslfo.htm

Srovnání dostupných FO procesorů:

http://www.volny.cz/zampach/dbk/fo_pdf.html

W3C Technical Reports and Publications:

http://www.w3.org/TR/

Apache Group XML Project:

http://xml.apache.org/

V. Přílohy

V.1. Ukázka XML dat

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

<seznam name="Telefonní seznam TUL podle pracovišť">

 <pracoviste nazev="fakulta architektury" zkratka="FA"
strom="/VSLIB/FA">

 <pracoviste nazev="Katedra architektury – architektonická kancelář"
zkratka="ARK" strom="/VSLIB/FA/ARK">

<funkce nazev="zaměstnanec">

 <zamestnanec id="18">
  <titul1>Ing. arch.</titul1>
  <jmeno>Vladimír</jmeno>
  <prijmeni>Balda</prijmeni>
  <telefony>3593, 3480 TEL/FAX</telefony>
  <email>vladimir.balda@vslib.cz</email>
 </zamestnanec>

 <zamestnanec id="730">
  <titul1>Ing. arch.</titul1>
  <jmeno>Marie</jmeno>
  <prijmeni>Procházková</prijmeni>
  <telefony>3480 TEL/FAX</telefony>
  <email>marie prochazkova@vslib.cz</email>
 </zamestnanec>

 <zamestnanec id="875">
  <titul1>Ing. arch.</titul1>
  <jmeno>Martin</jmeno>
  <prijmeni>Šaml</prijmeni>
  <telefony>3593, 3480 TEL/FAX</telefony>
  <email>martin.saml@vslib.cz</email>
 </zamestnanec>

 <zamestnanec id="910">
  <titul1>Ing. arch.</titul1>
  <jmeno>Vojtìch</jmeno>
  <prijmeni>Šrut</prijmeni>
  <telefony>3480 TEL/FAX</telefony>
 </zamestnanec>

</funkce>
.
.
.
```

Atribut pracoviště@strom slouží k jednoznačné identifikaci pracoviště (pro potřeby XSLT stylu tedy nahrazuje pracoviste@id), pří přípravě XML dokumentu má však klíčovou roli pro navigaci ve stromové struktuře pracovišť.

V.2. Styl XML2HTML

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

<!-- Definice pouzitych entit: -->
<!DOCTYPE xsl:stylesheet [
<!ENTITY nbsp " ">   <!-- (nedelitelna) mezera -->
<!ENTITY eol "<xsl:text>&#xA;</xsl:text>">   <!-- konec radku (v XHTML kodu,
ne v prohlizeci) -->
]>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="utf-8" />

<xsl:key name="zamestnanci" match="//zamestnanec" use="prijmeni"/>
<xsl:key name="zkratky" match="//pracoviste" use="zamestnanec/prijmeni |
funkce/zamestnanec/prijmeni"/>

<!-- Sablona pro korenovy element <seznam>: -->
<xsl:template match="/">
  <link rel="stylesheet" type="text/css" href="stylxml.css">
  <div class="nadpis"><xsl:value-of select="seznam/@name"/></div>&eol;
      <xsl:apply-templates/><br /><br />&eol;

      <div class="nadpis">Abecední seznam:</div>&eol;
      <xsl:for-each select="//zamestnanec[generate-id(.) = generate-
id(key('zamestnanci',./prijmeni)[1])]">
        <xsl:sort select="prijmeni" lang="cs"/>
        <xsl:sort select="jmeno"/>

  <xsl:if test="prijmeni != ''"><span class="prijmeni"><xsl:value-of
select="prijmeni"/></span></xsl:if>
  <xsl:if test="jmeno != ''"> <span class="jmeno"><xsl:value-of
select="jmeno"/></span></xsl:if>
  <xsl:if test="jmeno2 != ''"> <span class="jmeno"><xsl:value-of
select="jmeno2"/></span></xsl:if>
  <xsl:if test="titul1 != ''">, <span class="jmeno"><xsl:value-of
select="titul1"/></span></xsl:if>
  <xsl:if test="titul2 != ''">, <span class="jmeno"><xsl:value-of
select="titul2"/></span></xsl:if>

  <span class="jmeno"> ( <xsl:for-each select="key('zkratky',
./prijmeni)">
  <xsl:value-of select="@zkratka"/> 
  </xsl:for-each>)</span>

  <xsl:if test="telefony != ''"><span class="telefony"><xsl:value-of
select="telefony"/></span></xsl:if><br />&eol;

      </xsl:for-each>

</xsl:template>
```

```
<!-- Sablona pro element <pracoviste>: -->
<xsl:template match="pracoviste[count(.//zamestnanec) > 0]">   <!-- Do
vypisu nezarazovat vetve, ktere nemaji zadneho zamestnance -->
  <a href="#{@nazev}" /><div class="pracoviste"><xsl:number
level="multiple"/> - <xsl:value-of select="@nazev"/></div>&eol;   <!--
Viceurovnove cislovani zprehledni vysledek -->
      <xsl:apply-templates/>
</xsl:template>

<!-- Sablona pro element <funkce>: -->
<xsl:template match="funkce">
  <a href="#{@nazev}" /><div class="funkce"><xsl:value-of
select="@nazev"/>:</div>&eol;
      <xsl:apply-templates/>
</xsl:template>


<!-- Sablona pro element <zamestnanec>: -->
<xsl:template match="zamestnanec">
  <xsl:if test="prijmeni != ''"><span class="prijmeni"><xsl:value-of
select="prijmeni"/></span></xsl:if>
  <xsl:if test="jmeno != ''"> <span class="jmeno"><xsl:value-of
select="jmeno"/></span></xsl:if>
  <xsl:if test="jmeno2 != ''"> <span class="jmeno"><xsl:value-of
select="jmeno2"/></span></xsl:if>
  <xsl:if test="titul1 != ''">, <span class="jmeno"><xsl:value-of
select="titul1"/></span></xsl:if>
  <xsl:if test="titul2 != ''">, <span class="jmeno"><xsl:value-of
select="titul2"/></span></xsl:if>
  <xsl:if test="telefony != ''"><span class="telefony"><xsl:value-of
select="telefony"/></span></xsl:if><br />&eol;
</xsl:template>

</xsl:stylesheet>
```

V.3. Styl XML2FO

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.1"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format" exclude-result-prefixes="fo">
  <xsl:output method="xml" version="1.0" omit-xml-declaration="no"
indent="yes"/>

<!-- definice klicu pro razeni a seskupovani -->
<xsl:key name="zamestnanci" match="//zamestnanec" use="@id"/>
<xsl:key name="zkratky" match="//pracoviste" use="zamestnanec/@id |
funkce/zamestnanec/@id"/>

 <!-- Sablona pro korenovy element: -->

  <xsl:template match="/">
    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
      <fo:layout-master-set>
        <fo:simple-page-master master-name="mojeA5" page-height="210mm"
page-width="150mm" margin-top="1cm" margin-bottom="1cm" margin-left="1cm"
margin-right="1cm">
          <fo:region-body margin-bottom="0.5cm"/>
          <fo:region-after extent="0.3cm"/>
        </fo:simple-page-master>
      </fo:layout-master-set>

      <fo:page-sequence master-reference="mojeA5" initial-page-number="1">
        <fo:static-content flow-name="xsl-region-after">
          <fo:block text-align="center" font-size="7pt" font-style="italic">
            -<fo:page-number/>-
          </fo:block>
        </fo:static-content>

        <fo:flow flow-name="xsl-region-body" font-family="CourierNew" font-
size="8pt">
          <fo:block text-align="center" text-decoration="underline" font-
size="10pt" font-weight="bold" space-after="4mm"><xsl:value-of
select="seznam/@name"/></fo:block>
           <fo:table table-layout="fixed">
           <fo:table-column column-width="130mm"/>
            <fo:table-body>
             <xsl:apply-templates/>
            </fo:table-body>
           </fo:table>

           <fo:table table-layout="fixed">
           <fo:table-column column-width="130mm"/>
            <fo:table-body>
             <fo:table-row keep-with-previous="always">
               <fo:table-cell>
               <fo:block text-align="center" text-decoration="underline"
font-size="10pt" font-weight="bold" space-after="4mm">Abecední
seznam:</fo:block>
              </fo:table-cell>
             </fo:table-row>
```

```
            <xsl:for-each select="//zamestnanec[generate-id(.) = generate-
id(key('zamestnanci',./@id)[1])]">  <!-- v rejstriku nas zajima vzdy jen 1.
vyskyt dane osoby; kontroluje se podle atributu @id, ktery je jedinecny -->
              <xsl:sort select="prijmeni" lang="utf-8"/>  <!-- seradime
vysledky podle prijmeni a jmena -->
              <xsl:sort select="jmeno"/>

               <fo:table-row keep-with-previous="always">
                 <fo:table-cell>
                <fo:block text-align-last="justify">
                <xsl:if test="prijmeni != ''"><xsl:value-of
select="prijmeni"/></xsl:if>
                <xsl:if test="jmeno != ''"> <xsl:value-of
select="jmeno"/></xsl:if>
                <xsl:if test="jmeno2 != ''"> <xsl:value-of
select="jmeno2"/></xsl:if>
                <xsl:if test="titul1 != ''">, <xsl:value-of
select="titul1"/></xsl:if>
                <xsl:if test="titul2 != ''">, <xsl:value-of
select="titul2"/></xsl:if>
                <xsl:for-each select="key('zkratky',
./@id)"> (<xsl:value-of select="@zkratka"/>)</xsl:for-each>
                <fo:leader leader-pattern="dots"/>
                <xsl:value-of select="telefony"/>
               </fo:block>
              </fo:table-cell>
             </fo:table-row>

              </xsl:for-each>
             </fo:table-body>
            </fo:table>

          <fo:block text-align="center" text-decoration="underline" font-
size="10pt" font-weight="bold" space-after="4mm"  break-
before="page">Obsah:</fo:block>
          <xsl:for-each select="//pracoviste">
            <fo:block text-align-last="justify" font-weight="bold">
              <fo:basic-link internal-destination="{generate-id(.)}">
                <xsl:value-of select="@nazev"/>
              </fo:basic-link>
              <fo:leader leader-pattern="dots"/>
              <fo:page-number-citation ref-id="{generate-id(.)}"/>
            </fo:block>
            <xsl:for-each select="funkce">
              <fo:block start-indent="1em" text-align-last="justify">
           <fo:basic-link internal-destination="{generate-id(.)}">
            <xsl:value-of select="@nazev"/>
           </fo:basic-link>
          <fo:leader leader-pattern="dots"/>
          <fo:page-number-citation ref-id="{generate-id(.)}"/>
             </fo:block>
             </xsl:for-each>
          </xsl:for-each>
        </fo:flow>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>
```

```
 <!-- Sablony pro vnorene elementy: -->

  <xsl:template match="pracoviste">
          <fo:table-row  keep-with-next="always">
            <fo:table-cell>
              <fo:block font-weight="bold" font-size="10pt" space-
before="3mm" id="{generate-id(.)}">
                <xsl:value-of select="@nazev"/> (<xsl:value-of
select="@zkratka"/>)
              </fo:block>
            </fo:table-cell>
          </fo:table-row>
          <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="funkce">
    <fo:table-row  keep-with-next="always">
      <fo:table-cell>
       <fo:block font-style="italic" font-weight="bold" space-before="2mm"
id="{generate-id(.)}">
        <xsl:value-of select="@nazev"/>:
       </fo:block>
      </fo:table-cell>
    </fo:table-row>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="zamestnanec">
    <fo:table-row keep-with-previous="always">
      <fo:table-cell>
        <fo:block text-align-last="justify">
        <xsl:if test="prijmeni != ''"><xsl:value-of
select="prijmeni"/></xsl:if>
        <xsl:if test="jmeno != ''"> <xsl:value-of
select="jmeno"/></xsl:if>
        <xsl:if test="jmeno2 != ''"> <xsl:value-of
select="jmeno2"/></xsl:if>
        <xsl:if test="titul1 != ''">, <xsl:value-of
select="titul1"/></xsl:if>
        <xsl:if test="titul2 != ''">, <xsl:value-of
select="titul2"/></xsl:if>
        <xsl:if test="../@abc != ''"> (<xsl:value-of
select="pzkratka"/>)</xsl:if>
        <fo:leader leader-pattern="dots"/>
        <xsl:value-of select="telefony"/>
       </fo:block>
      </fo:table-cell>
    </fo:table-row>
  </xsl:template>

</xsl:stylesheet>
```

VI. Obsah: