UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH
PRÍRODOVEDECKÁ FAKULTA
ÚSTAV INFORMATIKY

DIPLOMOVÁ PRÁCA

Školský rok 2006 / 2007

JAROSLAV ŠEBEŠ

UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH
PRÍRODOVEDECKÁ FAKULTA
ÚSTAV INFORMATIKY

DIPLOMOVÁ PRÁCA

VYUŽITIE XML PRI TVORBE INTERNETOVÝCH APLIKÁCIÍ

Školský rok 2006 / 2007 JAROSLAV ŠEBEŠ

Analytický list

Autor:	Jaroslav Šebeš
Názov práce:	Využitie XML pri tvorbe internetových aplikácií
Jazyk práce:	Slovenský
Typ práce:	Diplomová práca
Počet strán:	
Nadobúdaný titul:	Magister
Univerzita:	Univerzita Pavla Jozefa Šafárika
Fakulta:	Prírodovedecká fakulta
Ústav:	Ústav informatiky
Študijný odbor:	Geografia - informatika
Študiijný program:	Magisterský študijný program
Sídlo univerzity:	Košice
Vedúci záv. práce:	RNDr. Jozef Studenovský, CSc
Dátum odovzdania:	30. 4. 2007
Dátum obhajoby:	28.5 – 1.6 2007
Kľúčové slová:	XML, internet, MVC, PHP
Názov práce v AJ:	Using XML in internet applications
Kľúčové slová v AJ:	XML, internet, MVC, PHP

Čestné prehlásenie

Vyhlasujem, že som celú diplomovú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice, 30. 4. 2007-05-09

. . . . . .

Vlastnoručný podpis

Abstrakt

V práci sú vysvetlené základné princípy, na ktorých je postavený jazyk XML. Pozornosť je venovaná rozdeleniu XML dokumentov podľa charakteru dát. V ďalšej časti za zaoberám prezentáciou XML dokumentov s využitím kaskádových štýlov. Poukazujem na niektoré výhody a nedostatky použitia CSS. Ďalej sa zaoberám spracovaním XML dokumentov pomocou jazyka XSLT. Použitie je vysvetlené na jednoduchých príkladoch. Zložitejší príklad sa zaoberá riešením tranzitívneho uzáveru pomocou XML a XSLT. Najväčšia časť práce je venovaná architektúre internetových aplikácií. Vysvetľujem princíp modelu MVC a jeho jednotlivé vrstvy. V závere práce ja návrh riešenie MVC frameworku s využitím XML a XSLT. Obslužná logika frameworku je napísaná v jazyku PHP.

Abstract

The aim of this work is explain basic principles, which are used by XML language. I divide XML documents according to character of their data. In next part, I deal with presentation of XML documents by using of cascading stylesheets. I point on some facilities and minuses of CSS usability. Next, I go into processing of XML docs through the XSLT language. Usage of it is explained on simple examples. More complicated example proceed transitive lock solution using XML and XSLT. The most part of this work I pay attention to architecture of internet applications. I explain principle of MVC model and its layers. The end of this work is application of MVC framework using XML and XSLT. Business logic is written in PHP language.

Obsah

Zoznam obrázkov

Úvod

Webové aplikácie sú v súčasnosti najprogresívnejšie sa rozvíjajúcou oblasťou v IT priemysle. Rozvoj webu úzko súvisí so značkovacími jazykmi. Od počiatku webu to bol jazyk HTML založený na definícii SGML. V súčasnosti je to najmä XHTML, ktorý je implementáciou jazyk XML. XML je definovaný ako štandard a jeho vývoj zastrešuje organizácia W3 (World Wide Web Consorcium). Členmi W3 prakticky všetci najzvýznamnejší hráči v IT oblasti. Táto mohutná podpora dáva vývojarom webových aplikácii istotu, že svoje intelektuálne úsilie vkladajú do správnych technológií.

Existuje veľa prístupov k návrhu a tvorbe webových aplikácií. Dalo by sa povedať, že koľko programátorov, toľko prístupov. Postupom času sa sa presadzuje návrhový vzor MVC (Model –View – Controller). Tento návrhový vzor má svoje základy mimo prostredia webu a je aplikovateľný aj na iné typy aplikácií.

V mojej práci sa zaoberám využitím XML pri tvorbe webových aplikácií. Vysvetľujem základné črty XML dokumentov a ich rozdelenie podľa charakteru dát. Zaoberám sa problematikov zobrazovania XML dokumentov na strane internetového prehliadača a to s využitím CSS a XSLT.

Ďalej sa v mojej práci sa zaoberám návrhom jadra jednoduchého frameworku s využitím princípov MVC. Zároveň som sa rozhodol pri jeho návrhu čo najviac využiť súčasné technológie XML. Nezávislosť XML a techník jeho spracovania na platforme dáva možnosť vytvoriť framework, ktorý sa po menších úpravach dá použiť aj v iných prostrediach.

Kľúčovou vrstvou MVC modelu je Model. V praxi je jeho rozhranie definované ako údajový typ odvodený od Object. V mojej práci som pristúpil k značnému zjednodušeniu, kedy je rozhraním Modelu XML súbor.

Celý framework je vytvorený a funguje v prostredí open-source technológii. Vrstva Controller bola naprogramovaná v jazyku PHP. Tento jazyk má od verzie 5 silnú podporu pre jazyk XML, založenú (aj) na objektovom princípe. Jednoduchosť práce s XML mi dovoľovala sústrediť sa na logiku a návrh namiesto pohybovania sa v mútnych vodách tzv. enterprise riešení založených na Jave alebo .NET.

1. Úvod do XML

1.1 Jazyk XML

XML - *Rozšíriteľný značkovací jazyk*, je framework pre definovanie značkovacích jazykov. Na rozdiel od najznámejšieho značkovacieho jazyka súčasnoti HTML, tu nie je pevne daná množina značiek – *tagov*. Namiesto toho nám XML umožňuje definovať vlastné značky, ušité na mieru pre ten druh informácií, ktoré chceme prezentovať. Každý XML jazyk je zameraný na určitú aplikačnú doménu, avšak jednotlivé jazyky môžu zdieľať množstvo nástrojov pre prácu s XML dokumentmi.

Dokument XML

Dokumentom XML obvykle nazývame súbor alebo skupinu súborov, ktoré obsahujú nejaké *značkovanie*. Toto značkovanie musí byť v súlade s pravidlami určenými v štandardoch XML. Štandardizáciu XML zabezpečuje organizácia s názvom *World Wide Web Consorcium*. Základné pravidlá, ktorým musí vyhovovať každý XML dokument sú tieto:

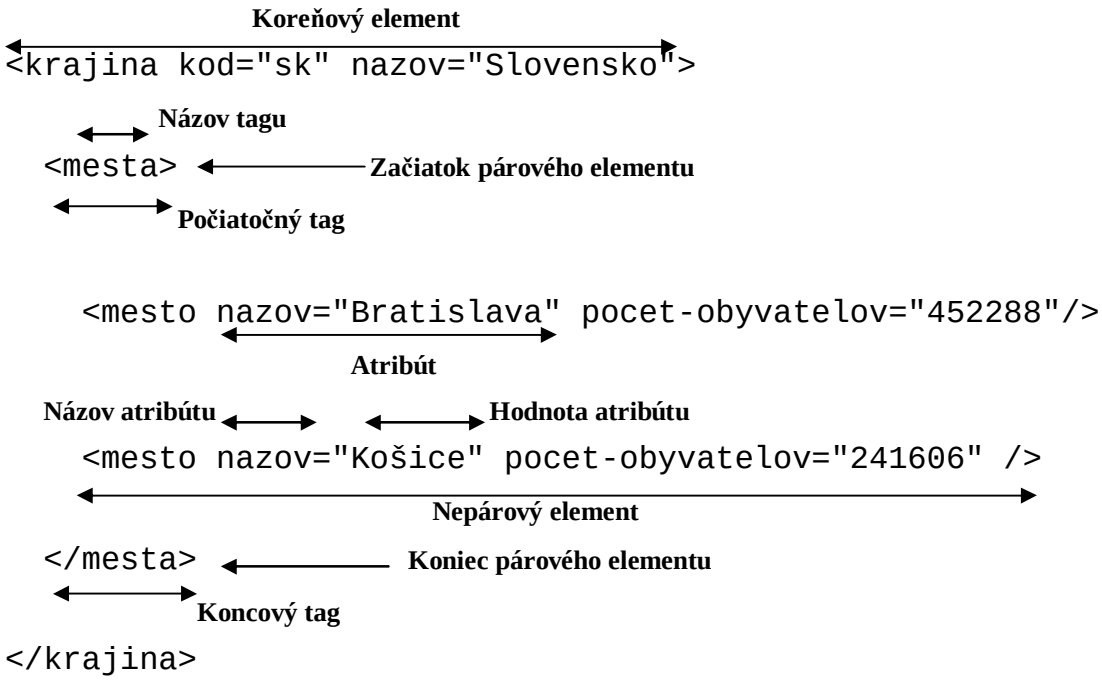
1. Dokument obsahuje jeden alebo viacero *elementov*.
2. Elementy sú do seba správne vnorené. Ak je *počiatočný tag* vo vnútri nejakého elementu, je taktiež *koncový tag* vo vnútri toho istého elementu.
3. Každý element, okrem *koreňového* má práve jedného *rodiča* a všetky elementy dokumentu tvoria *stromovú štruktúru*.

Jednoduchý XML dokument spĺňajúci tieto kritéria môže vyzerať napríklad takto:

```
<?xml version="1.0" encoding="windows-1250" ?>
<krajina kod="sk" nazov="Slovensko">
  <mesto nazov="Bratislava" pocet-obyvatelov="452288"/>
  <город имя="Košice" счет-жителей="241606" />
</krajina>
```

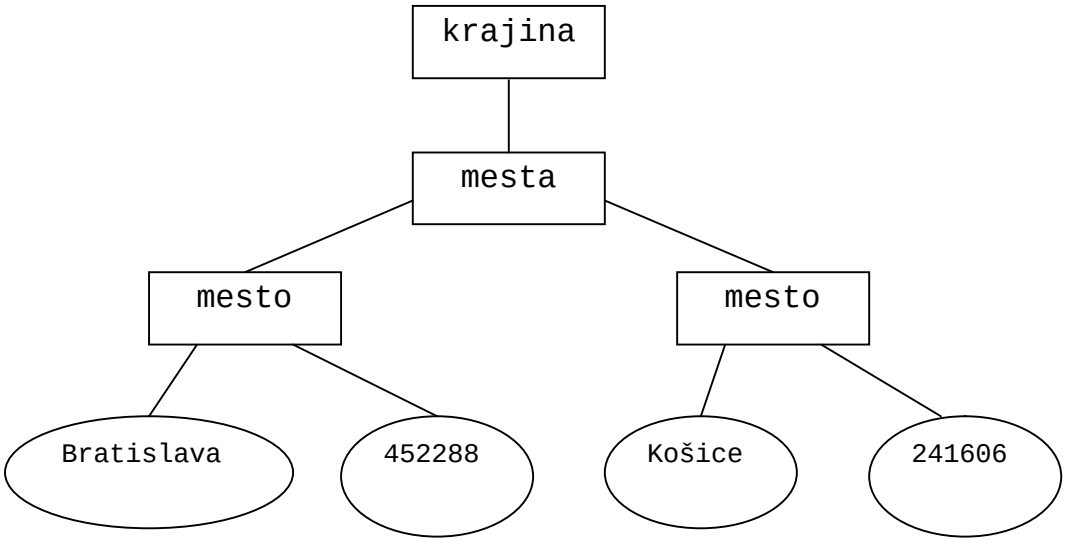
Na nasledujúcej schéme si vysvetlíme základné pojmy XML ako je element, atribút, tag (počiatočný, koncový, párový, nepárový), koreňový element.

Obrázok 1. Základné pojmy XML.



Každý XML dokument je hierarchickou štruktúrou nazývanou *XML strom.* Pozostáva z uzlov rôzneho typu zoskupených do stromu. Vyššie uvedená ukážka XML je prezentovaná ako strom v tejto podobe:

Obrázok 2. Stromová prezentácia XML dokumentu.



Obrázok ukazuje XML strom. Uzly sú znázornené ako obdĺžniky a elipsy. *Uzol typu element* – obdĺžnik, obsahuje názov tagu. *Uzol typu atribút* – elipsa obsahuje hodnotu atribútu. Uzol typu element a atribút môžme formálne vyjadriť:

<p a_1="A_1" … a_n="A_n">c_1 … c_m</p>	pre n >= 0, m >= 0
<p a_1="A_1" … a_n="A_n"/>	pre n >= 0

Element *typu p* obsahuje zoznam detí c_i a množinu atribútov, ktoré sú tvorené dvojicou *meno atribútu* a_i a *hodnota atribútu* A_i. Špecifikácia XML hovorí, že typ elementu je určený jeho názvom. Na rozdiel od detí pri atribútoch nezávisí na poradí, v akom sú uvedené. Text medzi počiatočným a koncovým tagom sa nazýva *obsahom elementu.*

Druhý uvedený výraz je špeciálnym prípadom prvého výrazu, pre m = 0, je teda vyjadrením že element nemá žiadny obsah.

Úlohy:

---

1. Jazyk HTML je najpoužívanejší značkovací jazyk na webe, podobne ako XHTML. Obe sú značkovacie jazyky, avšak iba jeden z nich vyhovuje syntaxi XML. Povedzte ktorý z nich to je a vysvetlite prečo.

2. Poznáte iné typy uzlov dokumentu XML ako je uzol typu element a atribút. Ktoré sú to? Je možné definovať vlastné typy atribútov ?

3. Je deklarácia dokumentu <?xml version="1.0"?> povinná ?

1.2 Kódovanie a znaková sada

XML súbory sú prezentované ako textové súbory to znamená že sú postupnosťou znakov. Historicky je textový súbor sekvenciou *Bytov.* Mapovanie z bytov na znaky často závisí od platformy, operačného systému a od jednotlivých aplikácií. Najznámejšie mapovanie je *ASCII*, ktoré pokrýva západoeurópske znaky.
Webové aplikácie by mali pokrývať potreby všetkých kultúr a musia byť univerzálne dostupné. Vyžaduje si to teda iné riešenie, ktoré je známe ako *Unicode*. Unicode je *znaková sada*, ktorej cieľom je pokryť všetky znaky v súčasných ako aj známych minulých písaných jazykoch.

Znak je symbol, ktorý sa nám javí ako časť textu. Znaky obsahujú zvyčajné alfabetické symboly ale aj abstraktnejšie piktogramy (napríklad znak copyright (c)) a modifikátory (dĺžne, mäkčene, ...). Znaky Unicode sú abstraktné *entity,* ktoré sú vyjadrené názvami ako napríklad:

1. LATIN CAPITAL LETTER A – VEĽKÉ PÍSMENO A
2. LATIN CAPITAL LETTER A WITH RING ABOVE – VEĽKÉ PÍSMENO A S KRÚŽKOM HORE
3. HIRAGANA LETTER SA – PÍSMENO JAPONSKEJ HIRAGANY

Tieto písmena samé osebe nemajú grafickú reprezentáciu ani reprezentáciu v bytoch.

Glyph je grafická reprezentácia určitej časti textu. Písmeno Å je grafickou reprezentáciou druhej možnosti napísanej vyššie. Jeden znak môže mať viacero grafických vyjadrení a naopak, to znamená, že tu neexistuje mapovanie 1:1 medzi znakmi a glyphmi.

Štandard Unicode definuje niekoľko spôsobov normalizácie sekvencií znakov, ktoré môžu byť použité ak je požadovaná unikátna reprezentácia znakov. Číslo znaku je unikátne číslo priradené ku každému znaku. Je niečo ako primárny kľúč. Unicode dovoľuje použiť čísla v rozmedzí 0 až 1 114 112 znakov. V súčasnosti je však

využitých iba niečo okolo stotisíc priradení znakov, z ktorých väčšina je čínskych. Napríklad písmeno HIRAGANA SA má číslo 12 373. Čísla od nula do 127 sú priradenia k rovnakým znakom ako v sade ASCII. Z technických dôvodov niektoré čísla neboli priradené žiadnym znakom. Používajú sa zvyčajne ako kontrolné body.

Kódovanie znakov interpretuje sekvenciu bytov ako sekvenciu čísiel znakov. V štandardnej sade ASCII sme si na vyjadrenie čísla vystačili s jedným bytom, v sade Unicode existuje niekoľko spôsobov ako vyjadriť hodnotu znaku na viacerých bytoch, hovoríme o *kódovaní*. XML dokument môže existovať v štandardnom kódovaní jeden byte = jeden znak alebo v iných. Východzím kódovaním pre XML súbory je *UTF-8*.

V UTF-8 je základnou *kódovacou jednotkou* jeden Byte. Každé číslo znaku je reprezentované jedným až štyrmi bytmi. Znaky, ktorých číselná hodnota je v rozmedzí 0 až 127 (pôvodné ASCII znaky), sú zapísané na jednom byte. Ak kódovacia jednotka začína 110XXXXX, znamená to, že znak bude zapísaný na dvoch bytoch. Ak začína 1110XXXX, znamená to, že znak bude zapísaný na troch bajtoch. Analogicky 11110XXX, znamená štvorbajtové vyjadrenie znaku. Výhodou tohoto prístupu je veľkosť výsledného súboru. Pri návrhu UTF-8 sa vychádzalo z toho, že väčšinu písmen tvoria aj tak štandardné znaky Latinky. Nevýhodou UTF-8 je premenlivá dĺžka znakov, ktorá kladie vyššie nároky na textové editory a parsery XML súborov.

Úlohy

---

1. UTF-8 nie je jediným kódovaním pre zápis širokej škály znakov. Okrem nich existujú kódovania UTF-16 a UTF-32 pre vyjadrenie znakov. Viete povedať rozdiely medzi nimi, ich výhody a nevýhody.

2. Pri prenose čísiel transportnou vrstvou internetu sa používa kódovanie Little Endian a Big Endian. Líšia sa spôsobom pozície významových bitov. Ako môžeme do súboru v kódovaní UTF-8 zapísať, v akom z týchto dvoch kódovaní sú znaky zapísané ?

1.3 Aplikácie XML

XML je hlavne a prdovšetkým formát pre ukladanie dokumentov. Vždy bol určený hlavne pre webové stránky, knihy, vedecké články a podobné dokumenty, ktoré čítajú hlavne ľudia. Jeho použitie ako syntaxe pre počítačové dáta v aplikáciách, ako je spracovanie napr. objednávok, serializácia objektov, komunikácia medzi databázami alebo elektronická výmena dát prišlo až neskôr.

Dátovo orientované XML dokumenty

Tieto dokumenty uchovávajú dáta, ktoré by za normálnych okolností bolo lepšie ukladať napríklad v relačných databázových systémoch. Typickým príkladom môže byť napríklad zoznam objednávok, aktuálne kurzy, stav bankových kônt a podobne. Dátovo orientované dokumenty sa vyznačujú týmito vlastnosťami:

1. Dokumenty zvyknú obsahovať množstvo dát
2. Štruktúra dokumentu je jednoduchá. Typicky pozostáva z množstva záznamov avšak s nie príliš mnohými vnoreniami elementov.
3. Dokumenty nie sú perzistentné, používajú sa napríklad ako formát pre prenos údajov, napríklad medzi rôznymi firemnými systémami.
4. Nemusia ani existovať ako samostatné dokumenty. Môže ísť o výstupný formát pri dopyte nad relačnou databázou. Riadky sú vracané ako XML fragmenty, s ktorými sa vykonávajú ďalšie operácie.

Dátovo orientovaný dokument môže vyzerať takto:

```
<?xml version="1.0" encoding="utf-8" ?>
<študenti>
  <študent meno=”Jozef Vrtinôžka” odbor=”G-I” />
  <študent meno=”ЮРИЙ ГАГАРИН” odbor=”F-I” />
  . . .
  . . .
  <študent meno=”پـپـيـز تـفـق” odbor=”F-I” />
</študenti>
```

Dokumentovo orientované XML dokumenty

Troška neforemný názov vyjadruje tie typy dokumentov, pri ktorých je okrem obsahu dôležitá aj ich logická štruktúra. Najznámejším predstaviteľom sú XHTML dokumenty, ktoré sú najdôležitejšou časťou webových stránok na strane webového prehliadača (tými ďalšími sú CSS štýly, klientské skripty a binárne objekty). Tento typ dokumentov sa vyznačuje týmito vlastnosťami:

1. Značkovací jazyk je použitý na opis logickej štruktúry obsahu dokumentu.
2. Veľkosť súboru nie je príliš veľká, keďže cieľom je poskytnúť užívateľovi čo najjednoduchší pohľad na obsah dokumentu
3. V mnohých prípadoch sú dokumenty uložené staticky a ich obsah sa nemení tak často.
4. Obsah XML dokumentu sa generuje dynamicky s využitím technológii založených na XML alebo pomocou programovacích jazykov.

Ukážka XML dokumentu s bohatšou logickou štruktúrou:

```
<?xml version="1.0" encoding="windows-1250" ?>
<krajina>
<politika kod="sk" nazov="Slovensko>
  <zriadenie typ="parlamentná republika" />
  <prezident meno="Ivan Gašparovič" />
  <premier meno="Róbert Fico" />
  <hlavme-mesto nazov="Bratislava" />
</politika>
<fyzicka-geografia rozloha="49035">
  <podnebie januar="-0.7" jul="19.1" zrazky="649" />
  <vyuzitie-zeme lesy="40.6"pastviny="16.5"orna-poda="33.4" />
</fyzicka-geografia>
<humanna-geografia>
  <obyvatelstvo pocet="5530000" urbanizacia="59">
    <dlzka-zivota muzi="69" zeny="77" />
    <etnicke-zlozenie>
      <etnikum nazov="slováci" zastupenie="85.6" />
      <etnikum nazov="maďari" zastupenie="10.6" />
      <etnikum nazov="česi" zastupenie="1.1" />
      <etnikum nazov="" zastupenie="2.7" />
    </etnicke-zlozenie>
  </obyvatelstvo>
</humanna-geografia>
</krajina>
```

Už pri prvom pohľade na takýto typ dokumentu je jasné, že elementy sú hierarchizované podľa obsahu, ktorý poskytujú. Je zrejmé, že element s názvom `etnikum` bude potomok elementu `etnicke-zlozenie`, ktorý je zasa potomkom elementu s názvom `humanna-geografia`. Je zrejmé, že ak by bolo etnické zloženie obyvateľstva v časti fyzická geografia šlo by síce o syntakticky správny XML dokument ale logická štruktúra by bola narušená (teda pre geografa znalého veci).

XML dokumenty ako zápis algoritmu a chovania aplikácie

Táto skupina dokumentov obsahuje XML technológie ako sú XML schéma, XSLT transformácie alebo WSDL jazyk na popis webových služieb. Taktiež tu môžeme zaradiť rôzne špecializované XML deriváty, ako napríklad konfiguračné súbory. Napríklad JSP stránky v Jave umožňujú od verzie 2.0 používať na zápis algoritmu namiesto syntaxe jazyka Java výrazy XML:

```
<html>
<head>
  <title>UkážkaJSP stránky so syntaxou XML</title>
</head>
<body>
  <x:forEach varStatus=”i” select=”$studenti/student”>
    <x:if select=”*[local-name() = ‘jano’]”>
      <p>
        študenti :
        <x:out select=”*” />
      </p>
    </x:if>
  </x:forEach>
</body>
</html>
```

Úlohy

1. V sekcii dátovo orientované dokumenty ste si určite všimli znaky, ktoré nepatria do nášho „geografického priestoru“. Sú to znaky azbuky a arabštiny. XML dokument má deklarované kódovanie utf-8 hneď v prvom riadku:

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

Táto deklarácia by mala zabezpečiť, že všetky znaky budú správne rozpoznané a interpretované. Nie je to však postačujúca podmienka. Čo je ešte potrebné urobiť s XML dokumentom ?

2. XHTML je najznámejším predstaviteľom „dokumentovo“ orientovaných dokumentov. Poznáte aj iné XML aplikácie, ktoré patria do tejto kategórie a sú prakticky využívané ?

3. Rozhodnutie medzi tým, ktoré dáta v XML súbore zapísať ako textový obsah elementov a ktoré vložiť do atribútov sa nazýva aj tvorba *granularity* XML dokumentu. Kedy je lepšie použiť atribúty namiesto nových elementov ?

4. Je možné použiť v názvoch elementov nasledujúce znaky ? :

- Písmená slovenskej abecedy vrátane dĺžňov a mäkčeňov.
- Pomlčku. „-„
- Podčiarkovník „_“
- Medzeru
- Dvojbodku „:“
- Znak percento, dolár, alebo ampersand „%”, „$“, „&”.
- Znaky čínštiny

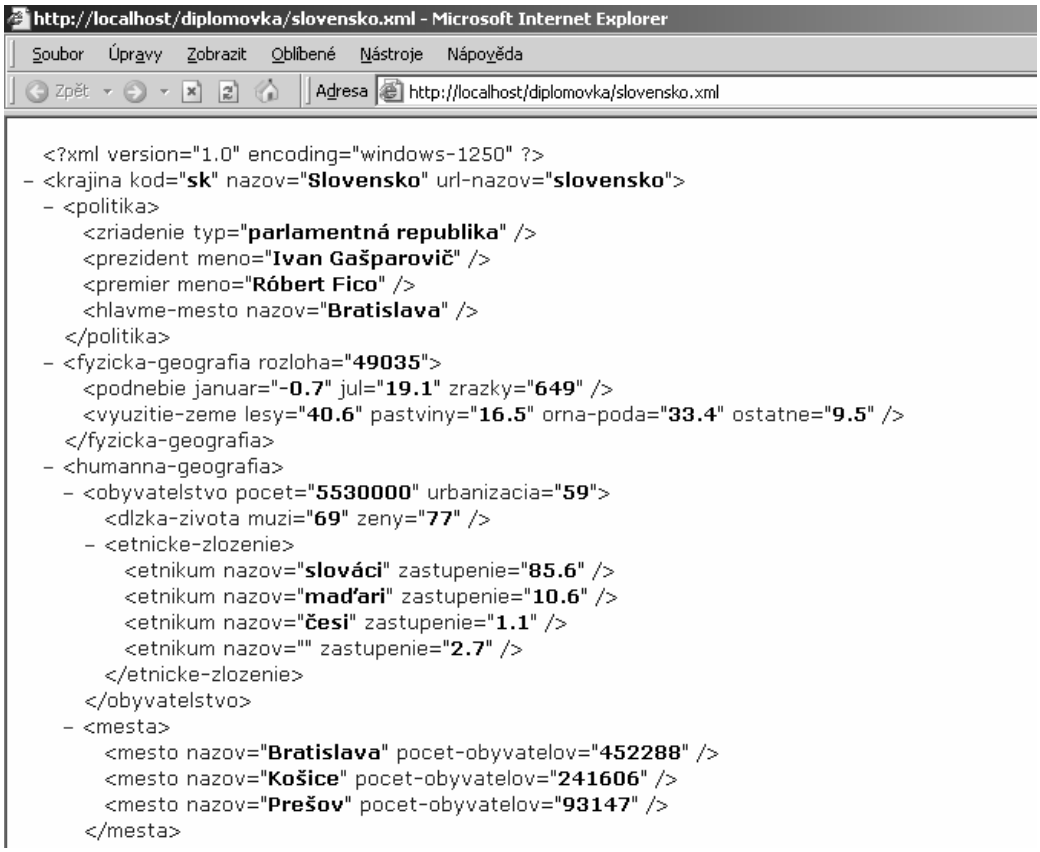
2. Prezentácia XML dokumentov na webe

2.1 Priame zobrazenie dokumentu

V predchádzajúcich kapitolách sme videli, že už čistý XML dokument je veľmi dobre čitateľný človekom. V časti 1.3 boli ukážky dokumentov, z ktorých boli dobre prečítateľné informácie o Slovensku, vrátane ich logickej štruktúry a súvislosti medzi nimi. Ak sa pohybujeme vo webovom prostredí, XML dokumenty máme uložené na webovom serveri. Pokiaľ sú priamo prístupné (nie je obmedzenie na ich stiahnutie, napríklad v pravidlách .htaccess), tak webový prehliadač môže priamo zobraziť „surový“ XML dokument = súbor.

Čo sa stane ak si v internetovom prehliadači vyžiadame čistý XML súbor? Súčasné prehliadače sa zachovajú veľmi podobne. Pokiaľ prehliadač obdrží XML dokument (bez pripojenej šablóny, či štýlu) použije implicitnú zabudovanú šablónu XSLT, ktorá zobrazí stromovú štruktúru dokumentu XML doplnenú o určité prvky DHTML, ktoré umožňujú užívateľovi zabaliť alebo rozbaliť jednotlivé uzly v strome.

Obrázok 3. Zobrazenie surového XML dokumentu.



Tento spôsob prezentácie XML dokumentu však nemá praktické využitie. Je vhodný snáď len na ladiace účely. Môže byť síce dobre čitateľný, to znamená, že užívateľ pochopí význam obsahu bez problémov, pravdepodobne bude mať však pocit, že nastala nejaká chyba pri zobrazovaní.

2.2 Prezentácia pomocou kaskádových štýlov

Keďže si v XML vymýšľame vlastné elementy, prehliadač nemá žiadne preddefinované znalosti ako tieto elementy zobraziť. Vytvorenie a pridanie vlastného *kaskádoveho štýlu* do dokumentu XML je jednou z možností ako povedať prehliadaču, ako má zobraziť dané elementy dokumentu. Vytvorenie a pripojenie kaskádového štýlu je najjednoduchšou metódou pre zobrazenie dokumentu XML. Jazyk *CSS* je hojne podporovaný všetkými súčasnými prehliadačmi. Jedinou prekážkou sú drobné rozdiely v interpretácii jednotlivých vlastností CSS, ktoré sa však pri znalosti a citu pre vec dajú obísť.

2.2.1 Základné kroky pri použití kaskádového štýlu

Ak chceme prezentovať XML dokument pomocou kaskádových štýlov, musíme vykonať dve kroky:

1. Vytvoriť súbor štýlu
2. Zviazať tento štýl s dokumentom XML

Predpokladajme, že máme jednoduchý XMl súbor, obsahujúci tieto dáta:

```
<?xml version="1.0" encoding="windows-1250" ?>
<krajina>
     <nazov>Slovensko</nazov>
     <prezident>Ivan Gašparovič</prezident>
     <fyzicka-geografia>
          <rozloha>49035</rozloha>
```

```
          <vyuzitie-zeme>
               <lesy>40.6</lesy>
               <pastviny>16.5</pastviny>
               <orna-poda>33.4</orna-poda>
               <ostatne>9.5</ostatne>
          </vyuzitie-zeme>
     </fyzicka-geografia>
</krajina>
```

Tento súbor uložíme pod názvom css-ukazka-1.xml. Ako druhý krok vytvoríme súbor kaskádových štýlov, ktorá zobrazí obsah tohoto súboru v nami želanej podobe. Súbor bude mať názov styl.css a bude umiestnený v tom istom adresári ako XML dokument. Obsah súboru bude vyzerať takto:

```
* {
     font-size: 16px;
     font-weight: bold;
     text-decoration: underline;
     display: block;
}

krajina {
     margin: 20px;
     padding: 20px;
     border: 4px double black;
}
```

Súbor obsahuje pravidlá pre zobrazenie obsahu elementov. Ich významom sa budem zaoberať neskôr. Vzhľadom k tomu, že vzľad sme určili v samostatnom súbore, je potrebné náš XML dokument prepojiť so štýlom. Priamo v XML slúži k tomu inštrukcia *xml-stylesheet*:

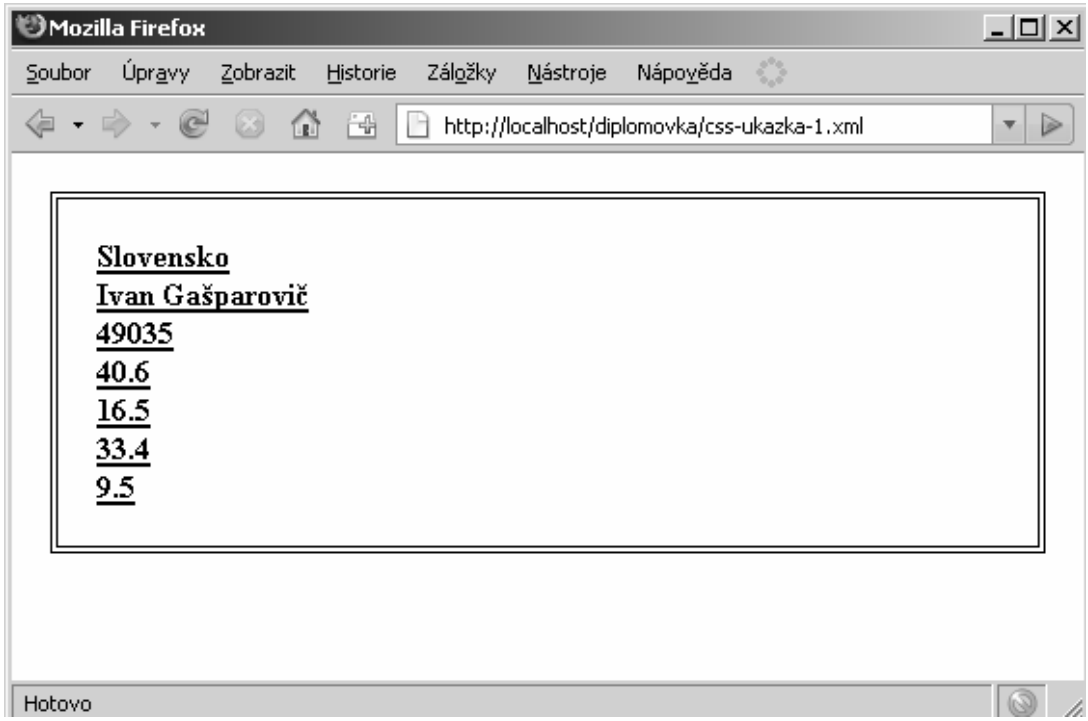
```
<?xml-stylesheet href=“meno súboru“ type=“mime typ“ ?>
```

Teda v našom prípade doplníme za úvodný riadok súboru css-ukazka-1.xml:

```
<?xml-stylesheet href="styl.css" type="text/css" ?>
```

Výsledok v internetovom prehliadači bude vyzerať tentoraz celkom odlišne

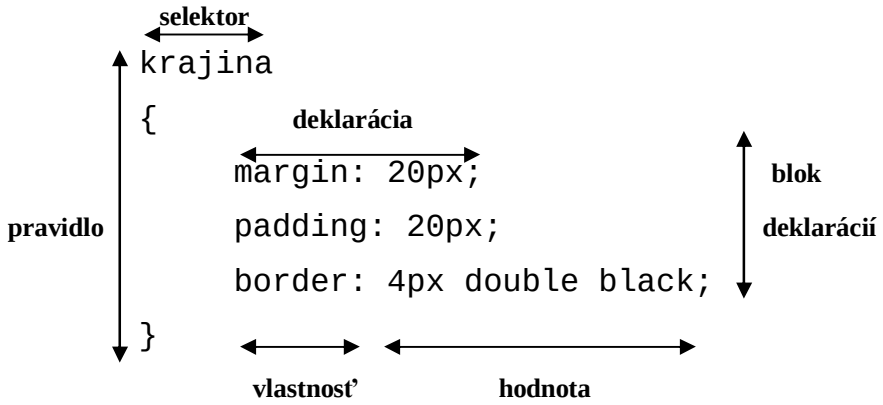
Obrázok 4. XML dokument s pripojeným kaskádovým štýlom:



Obsah XML dokumentu sa nezmenil, pribudla iba deklarácia, zaručujúca pripojenie štýlu. Súbor štýlu zobrazuje zoznam pravidiel, ktoré vravia zhruba toto:

Pre všetky elementy, nastav veľkosť písma na 16 pixelov, podčiarkni ho a zobraz každý element v novom riadku. Naviac, element krajina nech je orámovaný dvojitou čiarou. Formálne znázorňuje základné pojmy CSS nasledujúci obrázok:

Obrázok 5. Základné pojmy CSS.



Úlohy

1. Kaskádové štýly sa ku XML dokumentu pripájajú pomocou deklarácie xml-stylesheet. V XHTML, ako praktickej aplikácii HTML pre XML sa štýly môžu vkladať štyrmi rôznymi spôsobmi. Viete povedať, ktoré sú to ?

2. Je možné použiť jeden selektor v CSS súbore dvakrát, prípadne viackrát po sebe ? Čo by sa stalo ak by sme do nášho CSS súboru doplnili ešte nasledujúce riadky:

```
krajina {
     background-color: rgb(222, 200, 200);
}
```

2.2.2 Zobrazenie tabuliek pomocou CSS

Prehľad vlastností a hodnôt CSS nie je cieľom tejto práce. CSS má množstvo vlastností, ktorými môžeme nastaviť veľmi detailne zobrazenie akéhokoľvek elementu. Vybral som si jednu oblasť využitia CSS a tou je tabuľkové zobrazenie dát. Predpokladajme XML dokument s nasledovnou štruktúrou:

```
<obyvatelstvo>
  <zahlavie>
    <rok> - </rok>
    <rok>1900</rok><rok>1950</rok>><rok>2000</rok>
  </zahlavie>
  <krajina>
    <nazov>slovensko</nazov>
    <pocet>3 160 000</pocet>
    <pocet>3 910 000</pocet>
    <pocet>5 330 000</pocet>
  </krajina>
  ... <!—- zoznam ďalších krajín -->
</obyvatelstvo>
```

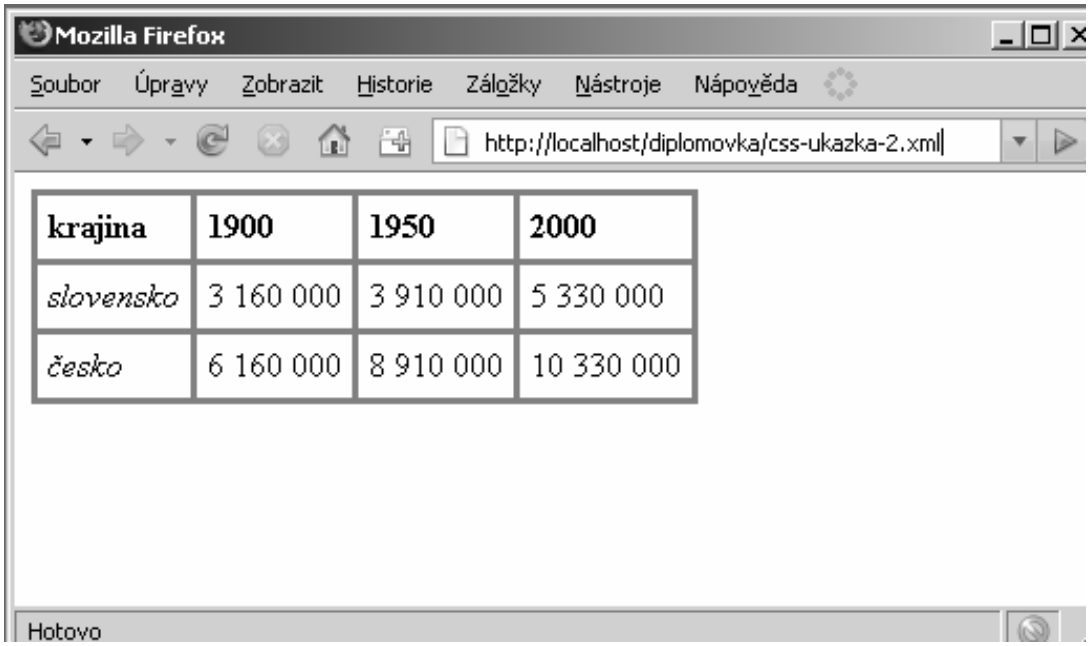
Dokument zobrazuje zoznam krajín a vývoj počtu obyvateľstva v jednotlivých krajinách v rokoch 1900, 1950 a 2000. Potrebovali by sme dosiahnuť v prehliadači tabuľkový vzhľad. Bez ohľadu na názvy jednotlivých elementov je vidno, že uvedená štruktúra má formu tabuľky. V našom prípade element *obyvatelstvo* zastrešuje celú tabuľku, element *krajina* predstavuje jednotlivé riadky a jeho potomkovia *nazov* a *pocet* predstavujú bunky tabuľky. Element *zahlavie* predstavuje záhlavie tabuľky.

CSS umožňuje definovať zobrazenie elementu ako súčasť tabuľkovej štruktúry. Jednoducho povedané, umožňuje zapísať pravidlo pre element ako „zobraz sa ako bunka tabuľky“, „zobraz sa ako riadok tabuľky“. Slúžia k tomu vlastnosť *display* s hodnotami table, table-row a table-cell (okrem iných hodnôt). Najjednoduchší štýl pre tabuľkový výpis by mohol vyzerať nasledovne:

```
obyvatelstvo {
     display: table;
     border-collapse: collapse;
}
zahlavie, krajina {
     display: table-row;
}
rok, pocet, nazov {
     display: table-cell;
     border: 2px solid red;
     padding: 5px;
}
rok {
     font-weight: bold;
}
nazov {
     font-style: italic;
}
```

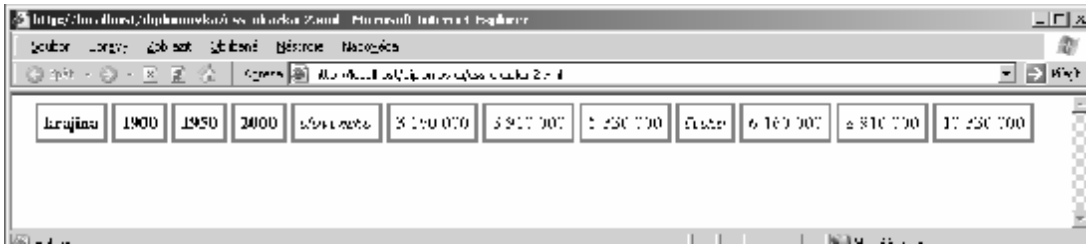
Výsledok v prehliadači Mozilla Firefox bude vyzerať nasledovne:

Obrázok 6. Tabuľkové zobrazenie dát v prehliadači Mozilla Firefox.



Výsledok v prehliadači Internet Explorer nás však nepoteší:

Obrázok 7. Tabuľkové zobrazenie dát v prehliadači Interner Explorer.



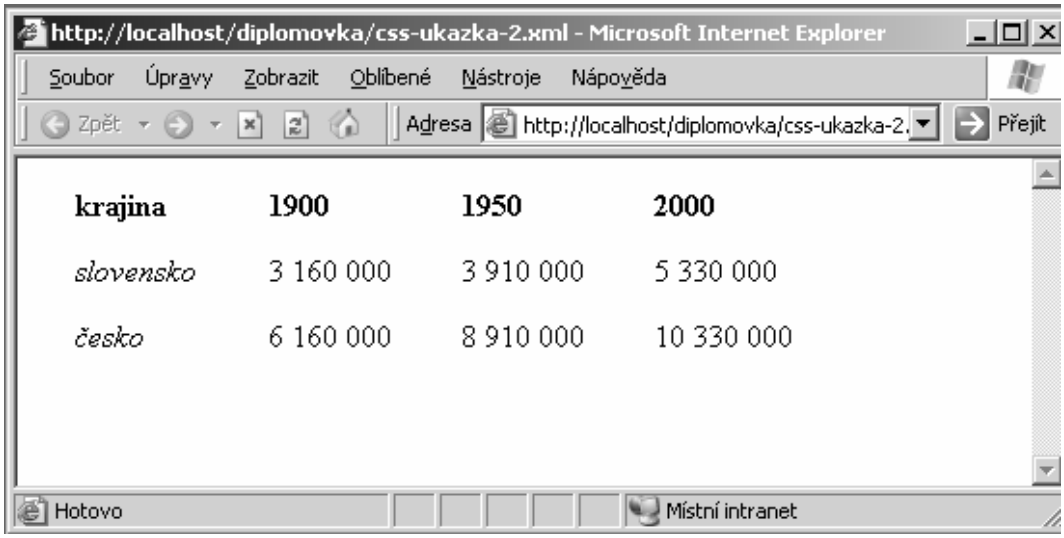
Ukážka v prehliadači Firefox je v poriadku a výsledok sa zobrazil podľa nášho očakávania. Internet Explorer však evidentne nepozná tabuľkové vlastnosti display. Na dosiahnutie tabuľkového vzhľadu musíme definovať kaskádový štýl iným spôsobom. Pre dosiahnutie želaného vzhľadu musíme náš štýl zmeniť nasledovne:

```
obyvatelstvo {
     display: block;
     margin: 10px;
}
zahlavie, krajina {
     display: block;
```

```
     clear: both;
     margin: 10px;
}
rok, pocet, nazov {
     display: block;
     float: left;
     width: 100px;
}
. . . /* ďalšie pravidlá ako v predošlom štýle */
```

Toto riešenie zabezpečí, že výsledok sa zobrazí približne rovnako vo všetkých súčasných prehliadačoch. Je však pomerne nevýhodné, pretože narába s plávajúcimi blokmi a treba sa s ním „pohrať“. Zmenu priniesla až najnovšia verzia Internet Exploreru, ktorá by už mala podporovať tabuľkové vlastnosti, avšak potrvá pár rokov, kým vytlačí svojich predchodcov.

Obrázok 8. Tabuľkové zobrazenie dát za pomoci plávajúcich boxov.



Zhrnutie

Pomocou kaskádových štýlov môžeme veľmi jednoducho a s využitím známych pravidiel priradiť vzhľad elementom z dokumentov XML. Ide však skôr provizórnu možnosť, niektoré veci sa pomocu CSS realizujú len veľmi ťažko, prípadne vôbec – napríklad zmena poradia elementov, vkladanie prefixov, ďalších XML elementov.

Úlohy

1. Čo by sa stalo pri zobrazení ak by sme doplnili do súboru s počtom obyvateľstva do každej *krajiny* jeden element *pocet* a *zahlavie* s elementmi rok by sme nechali bezo zmeny?

2. Pri použití plávajúcich boxov ako náhrady za tabuľkové vlastnosti sme použili vlastnosť *width* s pevne stanovenou šírkou boxu. Toto riešenie má nevýhodu v tom, že ak je obsah elementu širší (dlhé slovo bez medzery), tabuľkový vzhľad sa nám naruší. Ako je možné aspoň čiastočne odstrániť toto obmedzenie ?

3. Vykonajte zmeny v kaskádovom štýle používajucom tabuľkové vlastnosti zobrazenia tak aby výsledný dokument vyzeral nasledovne:

- Pozadie záhlavia tabuľky bolo tmavomodré a text v záhlaví nech má bielu farbu. Písmo ostane tučné s väčším rozostupom medzi písmenami.
- Tabuľka bude mať šírku celého prehliadača.
- Názvy krajín budu zobrazené kapitálkami

2.3 Transformácia dokumentov s využitím XSLT

2.3.1 Úvod do problematiky

V predchádzajúcej časti sme si vysvetlili vizuálnu prezentáciu XML dokumentu s využitím kaskádových štýlov. Už pri použití trocha zložitejších zobrazení naráža CSS na svoje mantinely. Príkladom je problematické zobrazenie tabuliek. Okrem toho s využitím CSS nemôžeme nijako docieliť napríklad:

- Zmenu poradia elementov. V príklade s vývojom počtu obyvateľstva nemôžeme prehodiť poradie krajín ani stĺpcov.
- Nemôžeme si vynútiť zmenu elementov. To má za následok, že výsledná stránka má len statický charakter, nemôže obsahovať formulárové prvky ako textové pole, rozbaľovací zoznam a podobne.

Potrebovali by sme silnejší nástroj na prezentáciu XML dokumentov. Vieme, že jazyk (X)HTML umožňuje používanie celej škály aktívnych formulárových prvkov. Potrebovali by sme nástroj ktorý by umožňoval napríklad previesť časť XML dokumentu z tvaru:

```
<krajina>
  <nazov>Slovensko</nazov>
  <nazov>Česká republika</nazov>
  <nazov>Poľsko</nazov>
</krajina>
```

do zoznamu krajín, ktoré sa v prehliadači zobrazia ako rozbaľovací zoznam, teda do XHTML kódu:

```
<select name=”krajiny”>
  <option>Slovensko</option>
  <option>Česká republika</option>
  <option>Poľsko</option>
</select>
```

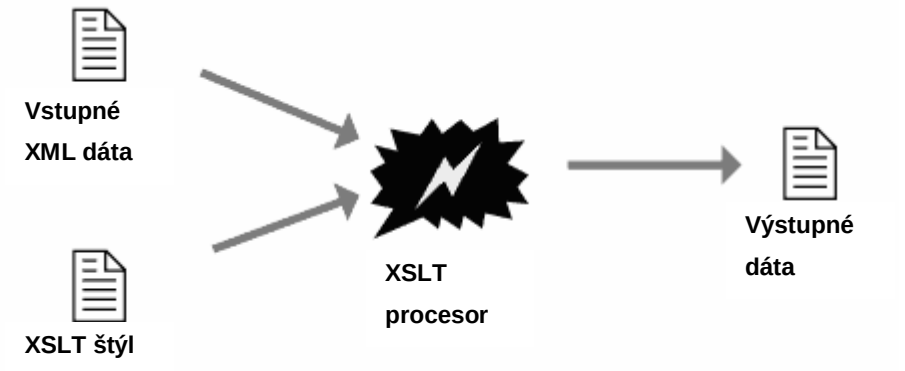
Takýto nástroj na transformáciu XML dokumentov existuje a samotné transformácie budú tiež písané v jazyku XML. Nazýva sa *XSLT – Extensible Stylesheet Language Transformations.*

Prečo je potrebné XSLT ?

Popisný značkovací jazyk XML má celú radu vlastností, vďaka ktorým sa mohol uchytiť. Je ľahko zapisovateľný i čitateľný ako pre človeka tak aj pre stroj a to hlavne kôli existencii nástrojov a knižníc pre všetky možné platformy. S dátami zapísanými v dokumente XML však obvykle treba ďalej pracovať. Samotný XML dokument nerobí totiž nič – iba proste existuje. XML parsery umožňujú dáta dolovať z dokumentu ale pre programátora to v konečnom dôsledku znamená dosť nezáživné iterovanie cez vrátené elementy a pozorne si strážiť priebeh algoritmu spracovania dokumentu.

Jadro problému spočíva vlastne v transformácii dát z XML do podoby, ktorú vyžadujeme. Bežné programovacie jazyky sú síce pomerne silné, avšak jednoduchšie by bolo využitie jazyka určeného priamo na takéto XML transformácie. Navyše ak by bol takýto jazyk tiež odvodený od XML. Takýmto jazykom je XSLT a jeho princíp sa dá vyjadriť nasledujúcim obrázkom:

Obrázok 9. Princíp XSLT



Programátor vo svojom kóde zavolá XSLT procesor a predá mu vstupné dáta a XSLT štýl, čo je vlastne predpis, podľa ktorého sa XML dáta buú spracovávať. XSLT procesor potom vygeneruje výstupné dáta.

Tento postup je pre programátora omnoho jednoduchší ako „ručné spracovanie“. Pokiaľ by to riešil algoritmicky, bolo by to niečo na spôsob „*nájdi element A, vlož ho do elementu B, pridaj element C a toto opakuj dookola*“. XSLT ale umožňuje napísať niečo ako „*elementy A nahraď elementmi B s doplnením C*“. Na akom základe postavíme jazyk pre spracovávanie XML ? Samozrejme, opäť na XML.

Prakticky všetky súčasné programovacie jazyky majú podporu pre nejaký XSLT procesor. TO nám dovoľuje použiť pre tvorbu internetových aplikácií ich širokú paletu. V najjednoduchšom prípade môžeme využiť možnosti súčasných prehliadačov, ktoré majú v sebe XSLT procesor. Vstupný XML dokument je potom možné parsovať pripojením deklarácie o použití XSLT štýlu podobne ako sme pripájali CSS štýl. Deklarácia vyzerá takto:

```
<?xml-stylesheet type=”text/xsl” href=”štýl.xsl” ?>
```

2.3.2 Jednoduchý príklad

Dostali sme sa k pripojeniu XSLT štýlu k dokumentu. Ako však bude vyzerať XSLT štýl, XML dokument a výsledný dokument? Uvažujme nasledujúci XML dokument:

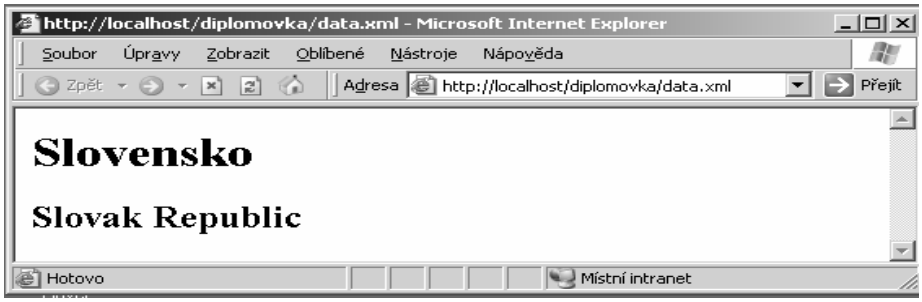
```
<krajina>
  <nazov-sk>Slovensko</nazov-sk>
  <nazov-en>Slovak Republic</nazov-en>
</krajina>
```

A nasledujúci XSLT štýl:

```
<xsl:stylesheet version = '1.0'
     xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="/">
  <h1>
    <xsl:value-of select="//nazov-sk" />
  </h1>
  <h2>
    <xsl:value-of select="//nazov-en" />
  </h2>
</xsl:template>
</xsl:stylesheet>
```

Pokiaľ vstupný dokument uložime pod názvom *data.xml* a štýl pod názvom *styl.xsl* a do súboru data.xml pridáme deklaráciu pripojenia štýlu, výsledkom bude v internetovom prehliadači:

Obrázok 10. Výsledok aplikácie jednoduchého XSLT štýlu v prehliadači



Pri náhľade do zdrojového kódu vygenerovanej stránky síce uvidíme pôvodný XML súbor, XSLT šablóna však vygenerovala nasledovný HTML kód:

```
<h1>Slovensko</h1>
<h2>Slovak Republic</h2>
```

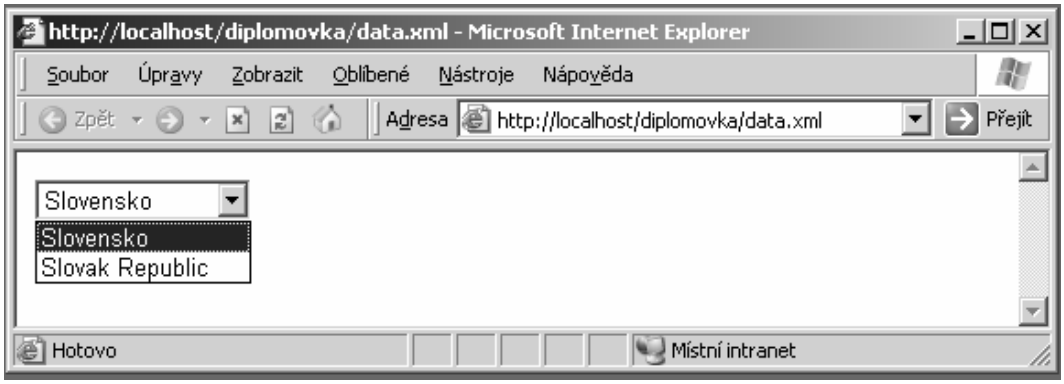
Šablóna XSLT nahradila teda elementy `nazov-sk` a `nazov-en` elementmi `h1` a `h2`.

Pozrime sa na fungovanie XSLT štýlu a jeho príkazov. Element `xsl:template` vymedzuje šablónu, ktorá bude nahradzovať koreňový element vstupného XML súboru, čo je zabezpečené hodnotou atribútu `match`. Element xsl:value-of vyťahuje zo vstupu požadované dáta, v prvom prípade nám vráti obsah elementu `nazov-sk` a v druhom prípade obsah elementu `nazov-en`. V šablóne sa nachádzajú aj elementy jazyka XHTML, ktoré sa skopírujú do výsledného dokumentu.

Úlohy

1. Upravte XSLT šablónu tak, aby sa vo výpise zobrazil najprv anglický názov krajiny uzavretý v elemente H1 a potom slovenský názov krajiny uzavretý elemente H2.

2. Upravte XSLT šablónu tak, aby sa krajiny zobrazili v prehliadači ako rozbaľovací zoznam (Dropdownlist). V prehliadači bude teda výsledok vyzerať nasledovne:



2.3.3 Šablóny

Najdôležitejším prvkom jazyka XSLT sú šablóny (templates). Ako sa s nimi pracuje? Veľmi jednoducho. Vyberieme Xpath výrazom nejaký element zo zdrojového dokumentu a nahradíme ho šablónou. Zápis vyzerá takto:

```
<xsl:template match='výraz'>
  <!-- obsah šablóny -->
</xsl:template>
```

Vkladanie dát

Pokiaľ vstupné dáta iba nahradíme inými, ktoré sme si pôvodne zadali, asi nám to nebude stačiť. Preto môžeme na vybrané miesta do šablóny opäť vložiť nejaké dáta z pôvodného dokumentu. Budeme ich vyberať opäť výrazom Xpath, pričom východzím bodom bude element, ktorý bol nahradený šablónou. K tomu slúži príkaz *value-of* a do jeho atribútu *select* napíšeme výraz k vyhľadávaniu dát pre vloženie.

Vnorené spracovanie šablón

Vo vnútri šablóny môžeme explicitne prikázať procesoru, že pre ďalšie spracovanie vstupných dát chceme nechať na ňom, nech v danej vetve vstupných dát ďalej pokračuje hľadaním šablón. K tomu slúži inštrukcia *xsl:apply-templates*. Nepovinný atribút *select* nám umožňuje opäť Xpath výrazom špecifikovať vetev stromu dokumentu určenú na spracovanie. Môžeme dokonca uniknúť niekde o úroveň vyššie a vykonať rekurziu (trebárs aj nekonečnú). Ukážeme si jednoduchý príklad vnorených šablón. Majme vstupný XML dokument:

```
<krajina pomenovanie="Slovensko">
  <mesto nazov="Košice">
    <inak nazov="Kaschau " /><inak nazov="Cassovia" />
  </mesto>
  <mesto nazov="Bratrislava">
    <inak nazov="Pressburg" /><inak nazov="Braslau" />
  </mesto>
</krajina>
```

A nasledujúcu XSL šablónu:

```
<xsl:stylesheet
  xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
  version='1.0'>
  <xsl:template match='/krajina'>
    <html>
    <head>
      <title>
        <xsl:value-of select="@pomenovanie" />
      </title>
    </head>
    <body>
      <xsl:apply-templates select="mesto" />
    </body>
    </html>
  </xsl:template>
  <xsl:template match='mesto'>
    <h3><xsl:value-of select="@nazov" /></h3>
    <ul>
      <xsl:apply-templates select="inak/@nazov" />
    </ul>
  </xsl:template>
  <xsl:template match='@nazov'>
    <li><xsl:value-of select="." /></li>
  </xsl:template>
</xsl:stylesheet>
```

V tomto dokumente sú definované celkom tri šablóny. Každá slúži ku spracovaniu určitej úrovne XML dokumentu. Ako vidíme, šablónu ktorá spracuje element *mesto* stačí napísať iba raz. Vykoná sa toľkokrát koľkokrát sa element mesto opakuje v XML dokumente. Výsledný vzhľad dokumentu bude vyzerať nasledovne:

Košice

- Kaschau
- Cassovia

Bratrislava

- Pressburg
- Braslau

Zrejme tu máme dve nečíslované XHTML zoznamy, každá s dvoma položkami.

Úlohy

---

1. Zmeňte XSLT šablónu tak aby vo výslednom dokumente boli položky číslovaného zoznamu. Teda výsledný dokument bude mať nasledovný vzhľad:

Košice

1. Kaschau
2. Cassovia

. . . ďalšie mesto . . .

2. Zmeňte XSLT šablónu tak, aby boli položky namiesto v číslovanom zozname zobrazené v tabuľke. (Použite XHTML tagy *table, tr, td*).

2.3.4 Vkladanie dát

Do výstupu XSL transformácie obvykle potrebujeme vložiť na rôzne miesta vybrané dáta zo vstupného dokumentu. Ukážeme si použitie niekoľkých inštrukcií, ktoré slúžia práve na vkladanie dát.

Použitie východzích šablón

Najjednoduchšou možnosťou, ako preniesť dáta zo vstupu na výstup, je zavolanie **<apply-templates/>,** pričom môžeme pridať nepovinný parameter *select*, ktorým vyberieme vo vstupnom dokumente uzol, odkiaľ sa bude čerpať. Pokiaľ pre XML

elementy na vstupe nie sú definované vhodné šablóny, použijú sa východzie šablóny XSLT procesoru, ktoré sa postarajú o prekopírovanie textového obsahu vybraného uzlu.

Hodnota výrazu (xsl:value-of)

Ak nechceme spracovávať určitý fragment žiadnymi šablónami, dá sa použiť inštrukcia *xsl:value-of* s parametrom *select*, ktorá prevedie všetok textový obsah uzlu (vyhodnoteného v atribúte *select*) na výstup.

Kopírovanie uzlov (xsl:copy, xsl:copy-of)

Nie vždy nás zaujíma iba textový obsah, môžeme niekedy chcieť kopírovať XML uzly zo vstupu. Sú dve možnosti ako to spraviť, líšia sa v tom či chceme kopírovať iba jeden uzol alebo hneď celú štruktúru.

Kopírovanie jedného uzlu (xsl:copy)

Inštrukcia *xsl:copy* skopíruje na výstup iba aktuálny uzol. Ak ide o XML dokument, nekopírujú sa automaticky jeho atribúty! Pokiaľ má element vo výsledku obsahovať nejaké ďalšie uzly, musia byť príslušné elementy (alebo inštrukcie, ktoré ich vytvoria) vložené dovnútra tejto inštrukcie, teda medzi značky *<xsl:copy> a </xsl:copy>*.

Kopírovanie všetkých uzlov (xsl:copy-of)

Inštrukcia *xsl:copy-of* má povinný parameter *select*, ktorý je vyhodnotený a ktorého výsledok je skopírovaný do výstupu ako celý XML fragment.

Šablóny hodnôt atribútov

Atribúty k elementom v šablónach môžeme zapisovať priamo do týchto elementov. Máme však možnosť zápisu Xpath výrazu do hodnôt týchto atribútov, a to pomocou zložených zátvoriek:

```
<a href="{@nazov}/.html">odkaz</a>
```

Text v zložených zátvorkách sa vyhodnotí, teda nájde sa príslušný uzol zo vstupu, vytiahne sa jeho reťazcová hodnota a doplní sa zbytkom hodnoty atribútu. Takto sa veľmi ľahko dajú používať hodnoty zo vstupného dokumentu ako atribúty ľubovoľných elementov.

Vytváranie elementov (xsl:element)

Do výstupu sa dajú vkladať elementy ich jednoduchým vložením na správne miesto do šablóny v transformačnom štýle. Čo však, ak názov elementu dopredu nepoznáme? V tejto situácii použijeme značku *xsl:element* s atribútom *name*, ktorý je reťazec, určujúci výsledný názov elementu (možno použiť Xpath výraz):

```
<xsl:element name="{@nazov}">
  <xsl:apply-templates />
</xsl:element">
```

Vytváranie atribútov (xsl:attribute)

Táto inštrukcia vytvorí k nadriadenému elementu atribút. Názov je daný opäť v parametri *name* a opäť je možné využiť XPath výrazy:

```
<img src="img/obrazok.gif">
  <xsl:attribute name="alt">popis obrázku</xsl:attribute>
</img>
```

2.3.5 Zložitejší príklad – tranzitívny uzáver a rekurzia

Jazyk XSLT má množstvo ďalších inštrukcií ako tie, ktoré boli popísané v predchádzajúcich kapitolách. Umožňuje používať konštrukcie známe z klasických programovacích jazykov ako napríklad vetvenie, cykly, triedenie, parametre, premenné, používanie rôznych funkcií a podobne. Vybral som si z tohoto obrovského množstva jednu oblasť XSLT, nazývanú kľúče. Pomocou kľúčov, môžeme vo výslednom dokumente vizuálne znázorniť dáta, ktoré majú vo vstupnom XML dokumente charakter relačnej tabuľky a ktoré sú prepojené navzájom pomocou cudzích kľúčov s inými záznanami v tej istej „tabuľke“.

Študent (každej lepšej) vysokej školy prichádza do kontaktu s informačným systémom, v ktorom si môže vytvoriť rozvrh na nasledujúci ročník. Pri výbere predmetu však musí zohľadniť prerekvizity daného predmetu. To znamená, že musí mať absolvované (alebo aspoň prísľub skorého absolvovania **J**) predmety, ktoré sú podmieňujúce pre zapísanie nového predmetu. V databáze by sme tento vzťah mohli vyjadriť napríklad nasledovnou štruktúrou tabuľky:

```
CREATE TABLE predmet (
  id INTEGER PRIMARY KEY,
  nazov VARCHAR,
  prerekvizita INTEGER
);
```

Stĺpec *id* obsahuje unikátne číslo predmetu, stĺpec *názov* obsahuje (na prekvapenie) názov predmetu a stĺpec *prerekvizita* obsahuje id predmetu, ktorý je potrebný absolvovať predtým. Vzhľadom na nasledujúci výklad sme počet priamych prerekvizít zredukovali (a zjednodušili) na jednu. Predpokladajme, že máme v tabuľke nasledovné hodnoty:

Id	**Nazov**	**prerekvizita**
1	Matematika 1	-
2	Matematika 2	1
3	Teória strún	2
4	Kozmografia	3

Na základe tabuľky je vidno, že ak si chce študent zapísať predmet *Matematika 2*, musí mať predtým úspešne absolvovaný predmet *Matematika 1*. Predmet *Matematika 2* má totiž v stĺpci *prerekvizita* číslo *1* a to je práve *id* predmetu *Matematika 1*. Najhoršie dopadne študent, ktorý si chce zapísať predmet *Kozmografia* pretože musí mať absolvované všetky predmety uvedené v tabuľke.

To, že pre predmet *Kozmografia* musíme absolvovať všetky predmety, ktoré sú jeho prerekvizitou, znamená, že medzi jednotlivými predmetmi je relácia. Ak je predmet *Kozmografia* v relácii s predmetom *Teória strún* a ten je zasa v relácii s predmetom *Matematika 2*, znamená, že medzi *Kozmografiou* a *Matematikou 2* je tranzitívna relácia. Tranzitívny uzáver znamená, zjednodušene povedané odpoveď na otázku „kde táto tranzitivita končí“.

Pomocou jazyka SQL môžeme v pokročilejšom databázovom vypísať zoznam všetkých prerekvizít pre daný predmet. Keďže sa však pohybujeme v prostredí XSLT chceme využiť možnosti tohoto jazyka.

Inštrukcia xsl:key

V XSLT môžeme definovať tzv. kľúč – *xsl:key*. Je to element najvyššej úrovne a definuje jeden alebo viacero kľúčov, na ktoré sa potom dá z ostatných miest v šablónach odkazovať pomoocu funkcie *key()*. Každý kľúč má svoj názov, reťazcovú hodnotu a uzol. Atribúty (všetky povinné) inštrukcie *xsl:key* sú nasledovné:

1. *name*: Názov kľúča
2. *match*: Porovnávací vzor, podobný vzoru v *xsl:template*, ktorý špecifikuje, k akým uzlom je tento kľúč priradený. Ak tomuto vzoru odpovedá v zdrojovom dokumente viacero uzlov, potom môže takto jeden element *xsl:key* definovať viac kľúčov, všetky s tým istým názvom, avšak pre rôzne uzly.
3. *use*: Výraz v Xpath, ktorý sa prevedie na reťazcovú hodnotu, ktorá určuje hodnotu kľúčov definovaných týmto elementom. Tento výraz sa vyhodnocuje s ohľadom na jednotlivé uzly kľúča – ak teda atribút *match* identifikuje viac uzlov než jeden, môže výraz v *use* vygenerovať pre každý uzol inú hodnotu.

Predpokladajme, že tabuľkovú štruktúru, navrhnutú vyššie reprezentujeme pomoocu XML v nasledovnej podobe, doplnenú o nejaké ďalšie predmety:

```
<predmety>
  <predmet id=”1”>
    <nazov>Matematika 1</nazov>
  </predmet>
  <predmet id=”2”>
    <nazov>Matematika 2</nazov>
    <prerekvizita>1</prerekvizita>
  </predmet>
  <predmet id=”3”>
    <nazov>Teória strún</nazov>
    <prerekvizita>2</prerekvizita>
  </predmet>
  <predmet id=”4”>
    <nazov>Kozmografia</nazov>
    <prerekvizita>3</prerekvizita>
  </predmet>
 . . . <!-- ďalšie predmety -->
</predmety>
```

Štruktúra dokumentu je zrejmá na prvý pohľad. Pripomínam, že prvý predmet neobsahuje element *prerekvizita*, čím je vyjadrené, že žiadnu prerekvizitu nemá. Potrebovali by sme vypísať prerekvizity pre jednotlivé predmety, navyše obojsmerne. Teda najprv vyjadriť vzťah:

- Podmieňovaný – podmieňujúci

  Zoznam predmetov .....

a potom:

- Podmieňujúci – podmieňovaný

  Zoznam predmetov ....

XSLT šablóna bude vyzerať nasledovne:

```
<xsl:stylesheet
  xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
  version='1.0'>
<xsl:output method="html" encoding="windows-1250"/>
<xsl:key name="podmienovany" match="//predmet"
use="@id"/>
<xsl:key name="podmienujuci" match="//predmet"
use="prerekvizita"/>

<xsl:template match="/">
  <html>
    <body>
      <h1>Prerekvizita. Vzťah podmieňovaný -
podmieňujúci</h1>
      <xsl:for-each select="//predmet">
        <xsl:value-of select="nazov"/> -
        <xsl:value-of select="key('podmienovany',
prerekvizita)/nazov"/>
        <br/>
      </xsl:for-each>
      <h1>Prerekvizita. Vzťah podmieňujúci-
podmieňovaný</h1>
      <xsl:for-each select="//predmet">
        <xsl:value-of select="nazov"/> -
          <xsl:for-each select="key('podmienujuci',
@id)">
            <xsl:value-of select="nazov"/>,
          </xsl:for-each>
        <br/>
      </xsl:for-each>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Výsledok v internetovom prehliadači bude vyzerať nasledovne:

Prerekvizita. Vzťah podmieňovaný – podmieňujúci

Matematika 1 -
Matematika 2 - Matematika 1
Teória strún - Matematika 2
Kozmografia - Teória strún
Solárna sústava - Teória strún
Galaxie a čierne diery - Teória strún

Prerekvizita. Vzťah podmieňujúci – podmieňovaný

Matematika 1 - Matematika 2,
Matematika 2 - Teória strún,
Teória strún - Kozmografia, Solárna sústava, Galaxie a čierne diery,
Kozmografia -
Solárna sústava -
Galaxie a čierne diery –

Úlohy

1.Vylepšite XSLT šablónu tak, aby boli jednotlivé vzťahy zobrazené v tabuľke.

2. Myslíte si, že uvedený spôsob detekovania tranzitívnej relácie je menej alebo viac výpočtovo náročný ako riešenie pomocou rekurzívnych selektov pomocou SQL. ?

---

3. Webová aplikácia a jej vrstvy

3.1 Vymedzenie pojmu webová aplikácia

Webová aplikácia je klient/server software, ktorý komunikuje s užívateľom alebo iným systémom prostredníctvom protokolu HTTP. Ako klient je používaný najčastejšie webový prehliadač, ako napríklad Internet Explorer, Mozilla alebo Opera. V súčasnosti pod pojmom klient webovej aplikácie rozumieme akékoľvek zariadenie schopné implementovať jazyk HTML podľa špecifických požiadaviek. Takýmto zariadením sú aj prehliadače v mobilných zariadeniach alebo napríklad hlasové čítačky pre nevidiacich. Klient na základe interakcie s užívateľom zasiela serveru jednotlivé požiadavky a následne zobrazuje obdržené webové stránky napísané spravidla v jazyku (X)HTML.

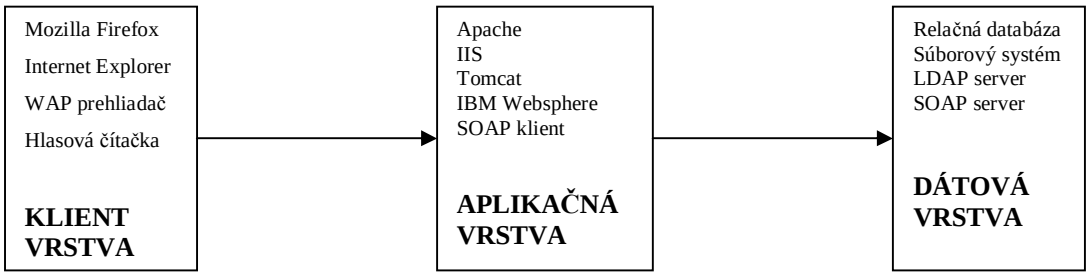
Vrstvy webovej aplikácie

Z definície vyplýva, že každá webová aplikácia pozostáva najmenej z dvoch vrstiev, spravidla však z troch. Užívateľ aplikácie prichádza do kontaktu len s tzv. tenkým klientom. V tomto prípade ide o webové stránky (prípadne WAP stránky a iné). Obslužná logika aplikácie je uložená na aplikačnom serveri, s ktorým klientská vrstva , teda prehliadač, komunikuje pomoocu protokolu HTTP. Dátovú vrstvu reprezentujú dáta uložené v ľubovoľnej forme, najčastejšie však organizované v databázovom systéme. Jedna webová aplikácia môže využívať dáta z rôznych zdrojov a na základe toho zvoliť vhodný komunikačný kanál. Dáta sa môžu získavať nasledovnými spôsobmi:
môžu získavať nasledovnými spôsobmi:

- Pomocou protokolu TCP / IP komunikuje aplikačný server s relačným databázovým systémom.
- Dáta sú získavané zo súborov fyzicky umiestnených na počítači kde beží aj aplikačný server. Tieto súbory môžu byť vo formáte určenom programátorom alebo v nejakom všeobecne známom a podporovanom formáte. V ďalších častiach práce budem uvažovať XML súbory.

- Potrebná dáta sú poskytované pomocou volania webovej služby. Webová služba je podmnožina vzdialeného volania procedúr. Dáta sa prenášajú s využitím protokolu HTTP a žiadosť aj odpoveď je poskytovaná vo formáte XML.

Obrázok 11. Trojvrstvová architektúra webovej aplikácie



Potreba rozdelenia aplikačnej vrstvy

Aplikačná vrstva predstavuje ťažisko celého systému. Táto vrstva zabezpečuje prístup k dátam, ich spracovanie a prezentáciu vo vhodnom formáte pre klientskú vrstvu. Komplexnosť aplikačnej vrstvy vedie k potrebe jej rozdelenia na podvrstvy. Toto rozčlenenie transparentne vymedzuje zodpovednosť danej podvrstvy v rámci aplikačnej vrstvy. Podvrstvy sú definované ako *prezentačné* (presentation), *aplikačné* (business, enterprise) a *dátové* (persistence). Existuje veľa prístupov k návrhom aplikačnej vrstvy webovej aplikácie, napríklad MVC.

3.2 MVC architektúra webovej aplikácie

Pojem architektúra aplikácie

Architektúra aplikácie je všeobecné označenie určujúce celkovú štruktúru a základnú konštrukciu časti alebo kompletného počítačového systému. V našom prípade architektúry aplikácie ide o spôsob rozdelenia aplikácie, aplikačných dát, procesov a dátových tokov do logických celkov, stanovenie štruktúry týchto komponentov, vzájomných vzťahov medzi nimi a to na dostatočne všeobecnej úrovni. Je zrejmé, že sa k návrhu a voľbe architektúry musí pristupovať na samom počiatku vývoja akejkoľvek aplikácie a to s veľkou rozvahou a opatrnosťou.

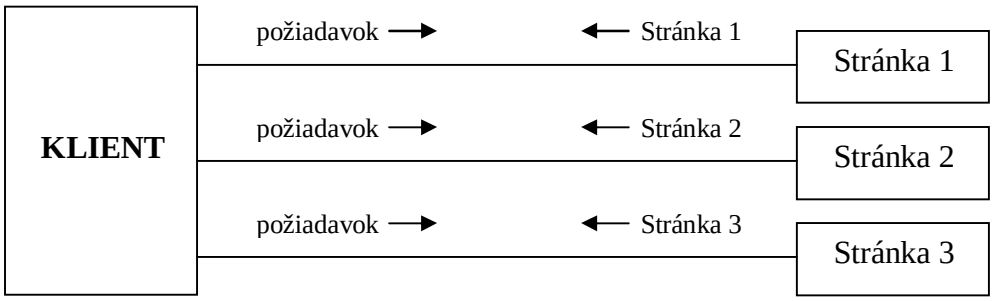
Modely architektúry webovej aplikácie

Vývojar webovej aplikácie spravidla rieši návrh aplikačnej vrstvy. O zobrazenie sa stará webový prehliadač a fyzické usporiadanie dát je záležitosť napríklad relačnej databázy. Vývojar systému tak musí vyriešiť otázku v akej forme budú predávané požiadavky (*request*) klienta na jednotlivé webové stránky. Z tohoto hľadiska môžeme rozlíšiť dve modely.

Model 1

Model 1 je taká architektúra webovej aplikácie, kde prehliadač pristupuje k jednotlivým stránkam priamo, otvára ich z URL, na ktorom sa skutočne nachádzajú. Nasledujúca stránka je potom určená pevnými odkazmi umiestnenými priamo v dokumente. Riadenie aplikácie založené na tomto modele je *decentralizované,* pretože aktuálna stránka už definítivne určuje nasledujúcu otváranú stránku. Každá stránka samostatne spracúvava svoje vstupy od klienta v parametroch *GET* alebo *POST*.

Obrázok 12. Model 1

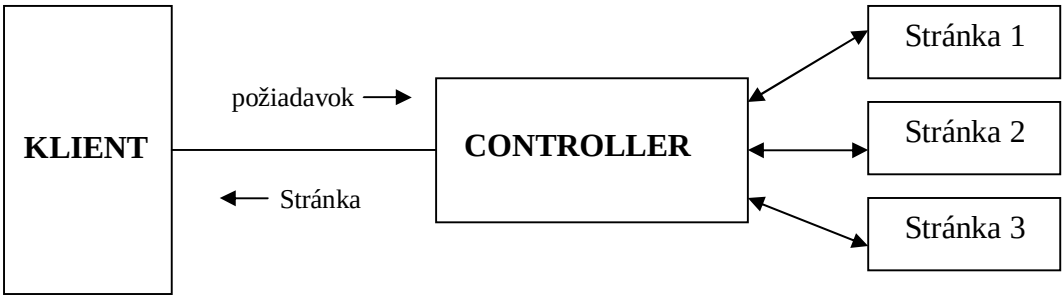


Model 2

Model 2 zavádza tzv. Controller, ktorý sa v procese spracovávania požiadavky nachádza medzi prehliadačom a otváranými stránkami alebo skriptami. *Controller* centralizuje logiku vyhodnocovania zaslanej požiadavky. Na základe požiadavky klienta, požadovaného URL, vstupných paramaetrov a celkového stavu aplikácie vykonáva príslušné akcie a vyberá ďalšiu stránku, ktorá sa v prehliadači zobrazí.

Jednotný Controller taktiež predstavuje ideálne miesto pre implementáciu takých funkcií ako je centralizované logovanie, autorizácia a podobne.

Obrázok 13. Model 2



Model 1 je vhodný pre statické webové prezentácie, nanajvýš pre veľmi jednoduché aplikácie. Štruktúra aplikácie spravidla zodpovedá adresárovej štruktúre jednotlivých stránok. Pre väčšinu interaktívnych aplikácií sa preto doporučuje používať Model 2. jeho implementáciu výrazne uľahčuje prístup zvaný MVC.

Model – View – Controller

Model – View – Controller je obľúbený spôsob návrhu webovej aplikácie. Svoje korene má v Smalltalku. Obecná definícia však umožňuje jeho používanie aj mimo trojvrstvových webových aplikácií a v ľubovoľnom jazyku. Príkladom využitia MVC mimo webu je Java Swing.

MVC rozdeľuje aplikáciu na tri časti. Model, View a Controller. Tieto časti úzko súvisia s troma časťami aplikačnej vrstvy. avšak v tomto prípade nejde o fyzické rozdelenie aplikácie ale o skôr o softvérové riešenie spolupráce už spomínaných vrstiev. MVC je návrhový vzor, ktorý pre každú jeho časť presne definuje, za čo je v *rámci aplikácie* zodpovedná. Pod pojmom *„presne definuje*“ rozumieme rámec aplikácie postavenej nad MVC. Samotné MVC je skôr filozofia prístupu a preto aj nasledovné definície jeho časti treba chápať ako orientačné.

Model

Model je funkčným a dátovým základom celej aplikácie. Poskytuje prostriedky pre prístup k dátovej základni a stavom aplikácie, ako aj pre ich ukladanie a aktualizáciu.

Mal by byť ako celok zapúzdrený a pre View a Controller má poskytovať presne definované rozhranie.
Na Model sa môžeme dívať ako na dve celky. Vyšší celok je implementovaný najčastejšie ako objekt, prípadne funkcia. Nižší celok je samozrejme tvorený nejakým úložiskom dát. Táto pomerne *všeobecná definícia nám však nebráni* v inej prezentácii, prípadne k čiastočnému spojeniu oboch celkov, napríklad XML fragment ako vyjadrenie dát a zároveň ich štruktúry.

View

View zobrazuje obsah Modelu , zaisťuje grafický či iný výstup aplikácie. View pristupuje k dátam aplikácie cez Model a špecifikuje ako majú byť prezentované. Pri zmene stavu Modelu sa aktualizuje aj zobrazenie. V prípade webovej aplikácie to znamená, že nasledujúca požiadavka od klienta zobrazí užívateľovi aktualizovanú stránku (takže View vygeneruje odpovedajúci HTML kód). V standalone aplikáciach slúži zobrazený výstup aj ako generátor udalostí. Napríklad prekreslené okno zachytí udalosť kliknutie myšou. V tomto prípade predáva View túto udalosť Controlleru.

Controller

Controller definuje chovanie aplikácie. Spracúvava všetky vstupy a udalosti pochádzajúce od užívateľa. Na základe nich vyvolá príslušné procesy Modelu, mení jeho stav (resp. dáta). Podľa udalostí prijatých od užívateľa vyberie Controller vhodné View pre ďalšie zobrazenie. V prípade webovej aplikácie je udalosťou HTTP GET alebo POST požiadavka.

Výhody MVC

- Ľahké sprístupnenie pre rôzne druhy klientov. Pre zavedenie podpory ďalšieho klienta stačí zadefinovať iba nové View a vykonať menšie úpravy v Controlleri. Dobre navrhnutý Controller môže tieto zmeny akceptovať napríklad z konfiguračného súboru, takže sa nemusí prepisovať jeho zdrojový kód.
- Minimalizácia duplicitného kódu a znovupoužiteľnosť komponentov.

- Rozdelenie vývojarských rolí. Jednotlivé časti tímu pracujú samostatne, medzi tímami sú stanovené určité rozhrania. Máloktorý vývojar je rovnako excelentný pri písaní databázových Selectov a zároveň pri grafickom návrhu webovej stránky.
- Centralizovaný Controller umožňuje jednotnú kontrolu autentifikácie a autorizácie užívateľa systému.
- Čistota designu, spôsob premýšľania programátora nad návrhom.

4. Návrh MVC Frameworku založenom na XML a XSLT

V predchádzajúcej časti boli vysvetlené základné princípy MVC modelu. S využitím XML technológii XML aplikujem tieto princípy na návrh vlastného MVC frameworku. Vytvorím jadro jednoduchého frameworku na poskytovanie informácii uložených v XML a pomocou XSL transformácií prezentovaných klientovi, teda prehliadaču webových stránok.

4.1 Architektúra navrhovaného MVC modelu frameworku

1. S dátovou vrstvou bude pracovať ako s reťazcom XML. To znamená, že XML formát je v MVC Modelom.

2. Pre jednoduchosť a názornosť budeme predpokladať, že tieto XML dáta budú uložené ako súbory na lokálnom disku. V prípade potreby však bude možné framework doplniť tak, aby akceptoval XML dáta ako výsledok napríklad SQL dotazu, prípadne ako výsledok volania webovej služby.

3. Prezentáciu XML dát z dátovej vrstvy do požadovaného formátu pre klienta zabezpečia šablóny XSLT. V databázovom vyjadrení, medzi XML súbormi a XSLT šablónami je vzťah M : N. Vrstvou View v modeli MVC budú teda šablóny XSLT.

4. Controller bude vyberať, na aký XML súbor bude aplikovaná nejaká XSLT šablóna. O výbere rozhodnú pravidlá mapovania v konfiguračnom súbore, ktorý bude (nečakane) vo formáte XML.

5. Klient webovej aplikácie rozhoduje o výbere pravidla odoslaním adresy URL. Zároveň, tieto adresy budú „SEO Friendly“, to znamená, že namiesto komplikovaných adries typu `example.sk/index.php?id=5&order=3` budú adresy v tvare `example.sk/studenti/zotried/abecedne`.

6. Implementácia Controllera bude v jazyku PHP 5. Jazyk PHP od verzie 5 umožňuje mimoriadne jednoduchú prácu s XML formátmi.

7. XSLT šablóny budú môcť prijímať parametre aj zvonku, čo zabezpečia pravidlá v konfiguračnom súbore. Odovzdávanie parametrov šablóne je podporované jazykom PHP 5.

Systémové požiadavky pre framework

Na strane webového servera je potrebné:

- Ak je webovým serverom Apache tak zapnutá podpora pre `mod_rewrite`.
- Skriptovací engine PHP >= 5.0, ideálne aspoň 5.1. PHP môže bežať ako modul webového servera alebo ako samostatná CGI aplikácia
- PHP skompilované s knižnicou libxml. (štandardne áno).

Na strane klienta je potrebné mať:

- Webový prehliadač schopný zobraziť XHTML kód. Prakticky všetky súčasné prehliadače to spĺňajú.
- Podpora CSS a prípadne JavaScriptu

Poznámka k mapovaniu virtuálnych URL na fyzické URL

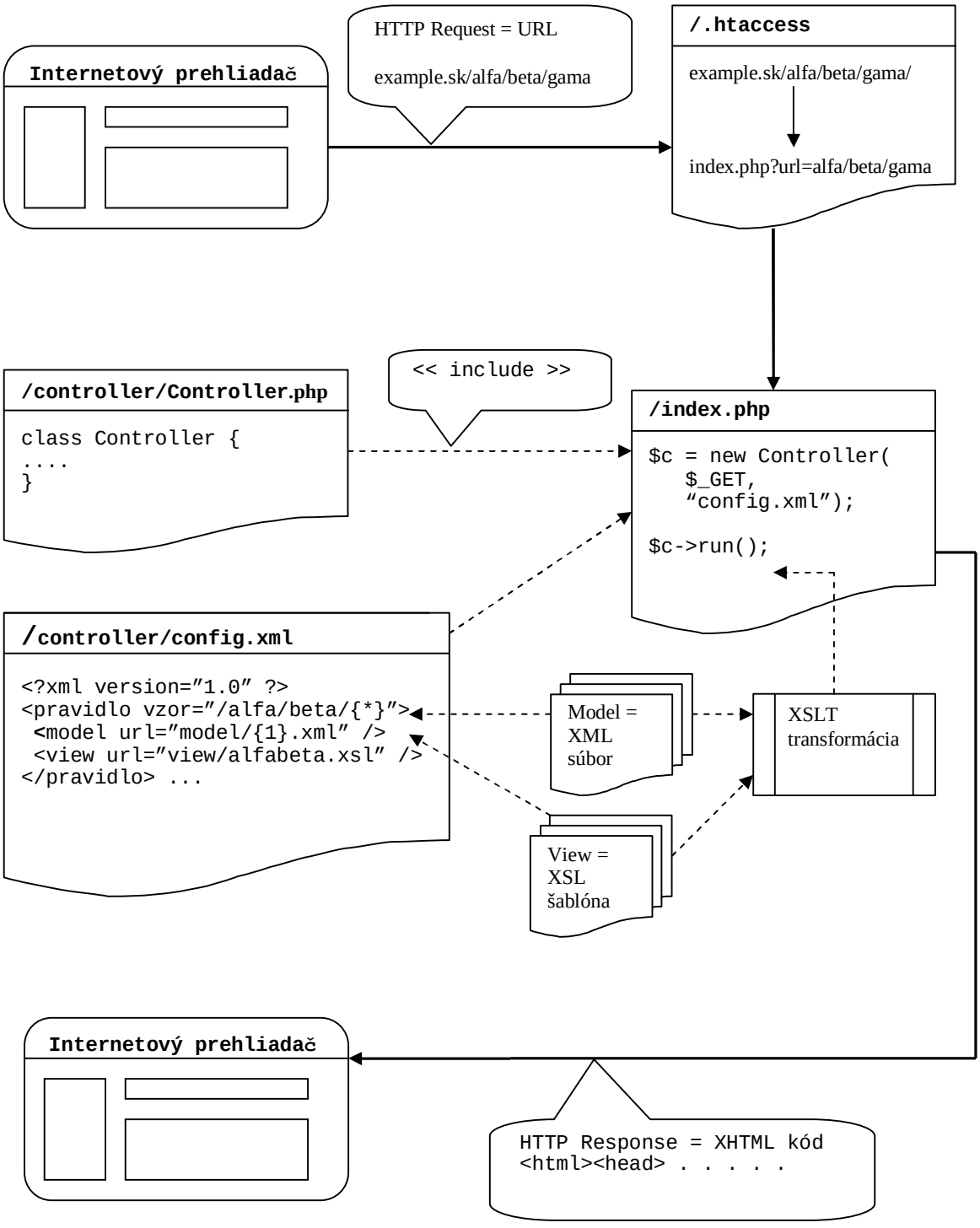
V piatom bode časti „architektúra frameworku“ bola uvedená požiadavka na tvar URL, ktoré v konečnom dôsledku rozhodujú o správaní aplikácie. Z uvedenej

požiadavky vyplýva, že klient a teda užívateľ vidí iba virtuálnu adresárovú štruktúru, ktorá vôbec nezodpovedá adresárovej štruktúre na webovom serveri. Prakticky všetky platformy webových serverov, počnúc *Apache*, cez *Tomcat* až po *IIS* umožňujú nejakú formu mapovania virtuálnych URL adries na fyzické. V Apache sa toto dá docieliť písaním pravidiel pre tzv. modul rewrite v súbore s názvom `.htaccess`.

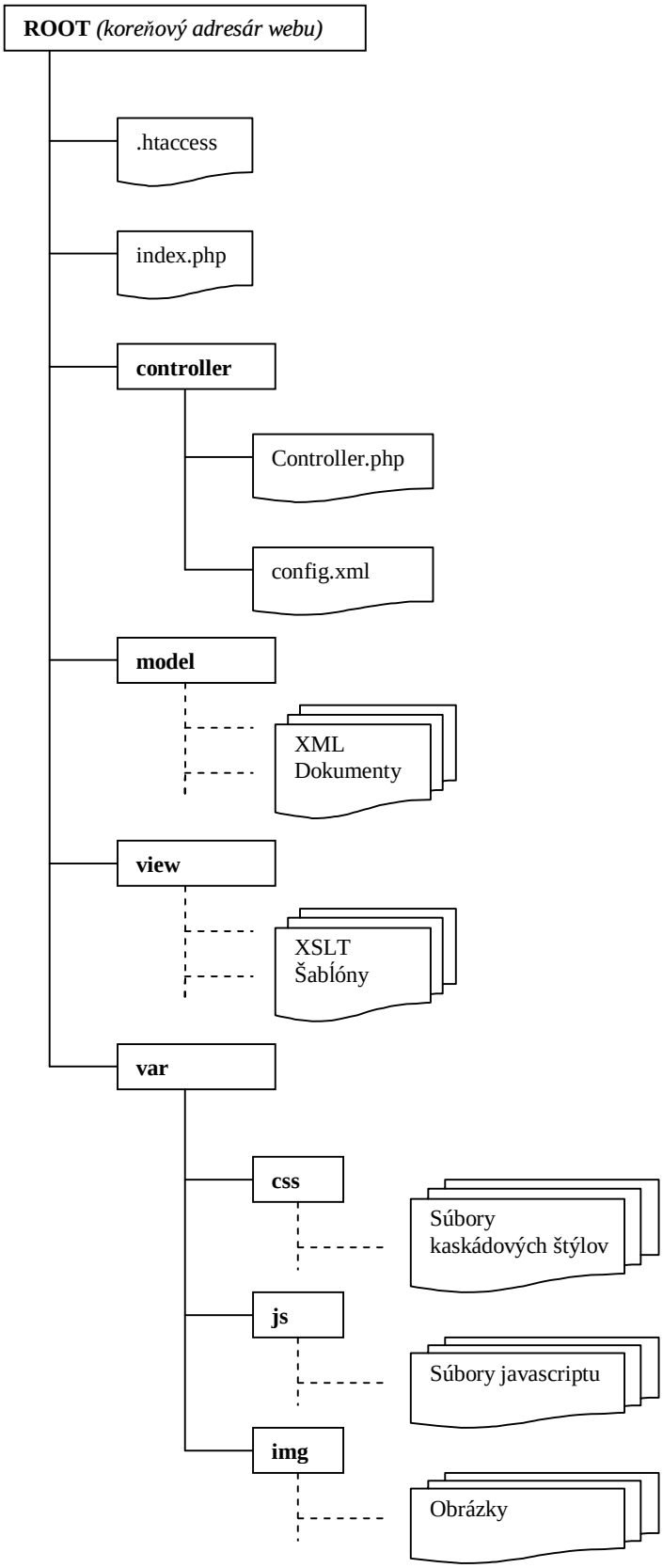
Spracovanie požiadavky a odoslanie odpovede

Klientskú časť aplikácie tvoria webové stránky s `XHTML` kódom. Komunikácia so serverovou časťou prebieha pomocou URL adries. Užívateľ systému môže zadať URL buď priamo do adresového políčka v prehliadači alebo aktivovaním nejakého hypertextového odkazu vo vygenerovanej webovej stránke. Tomuto budeme vravieť „požiadavka“, alebo teda `HTTP Request.` Výsledkom odovzdania požiadavky webovému serveru je aktivovanie príslušných komponentov frameworku a vygenerovanie príslušnej odpovede, teda `HTTP Response.` Celý postup ilustruje nasledovný obrázok.

Obrázok 14. Činnosť frameworku od prijatia požiadavky po odoslanie odpovede.



Obrázok 15. Adresárová štruktúra frameworku na strane webového servera



4.2 Činnosť frameworku a popis komponentov

Odoslanie URL požiadavky od užívateľa

Predpokladajme, že náš framework beží na doméne druhého rádu `www.geo.sk`. Nášmu frameworku dajme názov `XFrame (XML Framework).` To znamená, že projekt *www.geo.sk* je postavený nad *XFrame*. Na východzej URL adrese sa objaví titulná stránka webu. Nech náš web zobrazuje jednoduché informácie o krajinách európy – približne v rozsahu wikipedie. Predpokladajme, že užívateľ klikne na odkaz s názvom „Slovensko-hospodárstvo“. Užívateľ očakáva, že sa mu zobrazia informácie o hospodárskych ukazovateľoch Slovenska. Keďže na XFrame je kladená požiadavka „SEO Friendly“ URL adries, po aktivovaní odkazu bude mať URL požiadavka tvar *www.geo.sk/slovensko/hospodarstvo/*. Odoslaná požiadavka je vlastne *HTTP GET Request*, kde GET je už spomínaná URL.

Spracovanie požiadavky webovým serverom – implementácia v Apache.

Ak webový server nájde v koreňovom adresári súbor s názvom *.htaccess*, tak príkazy obsiahnuté v tomto súbore budú vykonávané postupne ako sa nachádzajú v súbore. Pripomínam, že súbor je textový a teda čitateľný človekom. Tento súbor má pre náš XFrame zásadný význam. Postará sa totiž o to aby naša URL adresa v tvare:

www.geo.sk/slovensko/hospodarstvo/

bola spracovaná do tvaru:

www.geo.sk/index.php?action=slovensko/hospodarstvo

Súbor .htaccess môžeme chápať ako prvú z komponentov XFrame. Jej obsah bude vyzerať nasledovne.

```
RewriteEngine On

RewriteCond %{HTTP_HOST} ^geo\.sk
RewriteRule ^(.*)$ http://www.geo.sk/$1 [R=301,
QSA]

RewriteCond %{HTTP_HOST} ^www\.geo\.sk
RewriteRule ^(.*)/$ index.php?action=$1 [L, QSA]
```

Poznámka:

Nastavenie niektorých serverov Apache môžu spôsobiť, že nám uvedené príkazy zlyhajú. V tom prípade môže pomôcť vložiť ako prvý príkaz `Options FollowSymlinks`.

Význam jednotlivých príkazov je nasledujúci:

Prvý riadok iba zapína podporu pre modul rewrite webového servera Apache. Tento modul má názov mod_rewrite. Druhý príkaz testuje či URL adresa neobsahuje reťazec „www“. To znamená, že ak je URL zadaná v tvare bez „www“ tak tretí príkaz spôsobí, že tento reťazec sa do URL vloží. Nemôžeme sa totiž spoľahnúť na užívateľovu dôslednosť a preto budeme dôsledne podstrkávať spomínaný reťazec. (i keď z praktického hľadiska je „www“ v URL adrese historickým prežitkom).
Štvrtý príkaz je testovaním podmienky, či URL už obsahuje reťazec „www“ a ak áno spustí sa piaty príkaz, ktorý spôsobí, že nech je HTTP GET odoslaná od klienta v akomkoľvek tvare, tak sa požiadavka presmeruje na súbor index.php, ktorý sa nachádza v koreňovom adresári webu. K tomuto súboru sa zároveň pripojí parameter s názvom `action`, ktorý obsahuje „skutočnú adresu“, teda URL, ako sa javí klientovi.
Podrobnejšie vysvetlenie príkazov nie je v tejto chvíli podstatné, skúsenejší čitateľ postrehne, že základom pre pravidlá modulu rewrite sú regulárne výrazy. Za zmienku stojí v poslednom príkaze ešte reťazec [L, QSA]. Je to tzv. príznak, písmeno L značí, že ide o konečné pravidlo uplatňované pri prepisovaní a QSA *(Query string append)* znamená, že ak klientská URL obsahovala aj ozajstné GET parametre, tak tieto sa prenesú súboru index.php. Význam modulu Rewrite pre náš projekt je zrejmý, klientovi sa URL javia ako skutočné, keďže sa po odoslaní požiadavky nemenia a nám to dovoľuje manipulovať s URL ako uznáme za vhodné.

Súbor index.php

Pravidlami pre modul Rewrite sme zaistili, že vždy bude volaný súbor index.php. Tento súbor bude obsahovať PHP kód, ktorý spustí Controller našej aplikácie implementovaný ako class, vytvorí sa teda jeho inštancia. Kód súboru index.php je krátky:

```
<?php
require_once 'controller/Controller.php'
$controller = new Controller($_GET,
'controller/config.xml');
$controller->run();
?>
```

Prvý príkaz vloží do súboru triedu Controller, ktorá zapúzdruje všetkú logiku. Vytvorí sa jeho inštancia, i keď vzhľadom na to, že Controller je len jeden mohlo by ísť aj o statické volanie.

Prvým paramerom konštruktora je premenná s názvom `$_GET`. Ide vlastne o asociatívne pole reťazcov. Približným ekvivalentom v Jave je HashMapa. Ak klientská URL bola v tvare www.geo.sk/slovensko/hospodarstvo/ tak po aplikovaní pravidiel v súbore .htaccess pole $_GET obsahuje:

```
Array (
   “action” : “slovensko/hospodarstvo”
)
```

Druhým parametrom konšruktoru Controllera je umiestnenie konfiguračného súboru s pravidlami pre mapovanie adries. Metóda run() spustí celý proces od výberu správnych šablón až po odoslanie výstupu späť ku klientovi.

Mapovanie URL adries – Súbor config.xml

Súbor config.xml obsahuje zoznam pravidiel. Controller prechádza všetkými pravidlami a porovnáva obsah premennej `$_GET['action']` s atribútom `vzor` elementu `pravidlo`. Ak nájde zhodu uplatní sa nájdené pravidlo.
V najjednoduchšom tvare (avšak funkčnom) vyzerá config.xml takto:

```
<?xml version=”1.0” encoding=”windows-1250” ?>
<konfiguracia>
  <pravidlo vzor=”slovensko/hospodarstvo”>
    <model url=”model/slovensko.xml” />
    <view url=”view/hospodarstvo.xsl” />
  </pravidlo>
</konfiguracia>
```

Náš súbor obsahuje iba jedno pravidlo, ktoré vraví, že:

1. Ak bola odoslaná URL adresa v tvare (www.geo.sk)/slovensko/hospodarstvo ...
2. ... tak vykonaj XSL Transformáciu, pričom ...
3. ... aplikuj šablónu hospodarstvo.xsl ...
4. ... na XML súbor slovensko.xml ...
5. a výsledok odošli klientovi.

Pravidiel pre mapovanie môže byť mnoho, pokryjeme nimi celú štruktúru webu. Určite sa nám nechce písať samostatné pravidlo pre každú krajinu, ktorých môže byť v tom najhoršom prípade okolo dvesto. V atribúte `vzor` je preto možné používať zástupné znaky, ktorými celý proces automatizujeme:

```
<?xml version=”1.0” encoding=”windows-1250” ?>
<konfiguracia>
  <pravidlo vzor=”{*}/hospodarstvo”>
    <model url=”model/{1}.xml” />
    <view url=”view/hospodarstvo.xsl” />
  </pravidlo>
</konfiguracia>
```

V atribúte vzor nám elementu pravidlo nám pribudol zástupný znak hviezdička uzatvorený v kučeravých zátvorkach. Význam znaku je zrejmý. Vzor vyhovie všetkým URL v tvare www.geo.sk/slovensko/hospodarstvo, www.geo.sk/polsko/hospodarstvo atď. Nevyhovie však URL www.geo.sk/europa/slovensko/hospodarstvo. Zástupný znak hviezdička má totiž platnosť medzi dvoma lomítkami, to znamená, že ním môžeme vyjadriť názov iba jedného virtuálneho adresára a nie viacerých.

V atribúte `url` elementu `model` je uvedený odkaz na to, čo vlastne hviezdička nahradzuje. V príklade je vidno, že je na danom mieste použité číslo v zátvorkách. Toto číslo určuje poradie nahradzovaného reťazca. Vzorka môže obsahovať viacero zástupných znakov, v tomto prípade je ich číslovanie odvodené od toho ako sa zástupné znaky nachádzajú v atribúte `vzor` elementu `pravidlo` zľava doprava. Nasledujúci príklad ukazuje použitie viacerých zástupných znakov:

```
<?xml version=”1.0” encoding=”windows-1250” ?>
<konfiguracia>
  <pravidlo vzor=”{*}/hospodarstvo/{*}”>
    <model url=”model/{1}.xml” />
    <view url=”view/hospodarstvo-{2}.xsl” />
  </pravidlo>
</konfiguracia>
```

Vzor vyhovie všetkým URL v tvare www.geo.sk/slovensko/hospodarstvo/dovoz, www.geo.sk/madarsko/hospodarstvo/odberatelia-europa atď. Keďže hviezdička

nahrádza iba jeden virtuálny adresár, vzoru nevyhovie napríklad www.geo.sk/slovensko/dovoz/ropa.

Predávanie parametrov šablónam v súbore config.xml

Používaním zástupných znakov môžeme výrazne zredukovať počet pravidiel pre mapovanie URL adries. Niekedy však nestačí ani tento mechanizmus. Uvážme nasledujúci príklad, ktorý zobrazuje rôzne spôsoby výpisu dovozu komodít pre vybranú krajinu:

```
<?xml version="1.0" encoding="windows-1250" ?>
<konfiguracia>
  <pravidlo vzor="{*}/hospodarstvo/dovoz/zotried/{*}">
   <model url="model/{1}.xml" />
    <view  url="view/hospodarstvo-dovoz-zotried-{2}.xsl"
/>
  </pravidlo>
</konfiguracia>
```

Zrejme je cieľom nášho pravidla zachytiť všetky URL, ktoré zobrazujú informácie o dovoze ľubovoľnej krajiny. Takisto je zrejmé, že tieto informácie chceme zobraziť zotriedené, či už abecedne podľa krajín, podľa percentuálneho zastúpenia jednotlivých dovozcov alebo podľa iného triediaceho kritéria.

Z pravidla vyplýva, že pre každý typ triedenia bude použitá iná šablóna. Pre url www.geo.sk/slovensko/hospodarstvo/dovoz/zotried/abecedne sa na xml súbor slovensko.xml aplikuje šablóna hospodarstvo-dovoz-zotried-abecedne.xsl. Z definície pravidla vyplýva, že pre iný typ triedenia sa použije na transformáciu iná XSL šablóna. Triedenie je však v jazyku XSLT aplikované jednoduchým spôsobom, stačí zmeniť hodnotu atribútu `select` elementu `xsl:sort`. Nevýhoda tohoto prístupu je zrejmá, máme množstvo XSL šablón líšiacich sa iba jediným parametrom.

Riešenie tohoto problému spočíva v predávaní parametrov XSL šablóny zvonku. Na porovnanie, ide o obdobný princíp ako predávanie parametrov spustiteľnému programu v Jave alebo v C/C++ z konzoly. Tieto parametre sú v týchto jazykoch prístupné v metóde / funkcii s názvom main(). Pre náš framework potrebujeme teda:

1. Zadefinovať v súbore config.xml názvy a hodnoty vstupných parametrov, ktoré sa predajú XSL transformátoru.
2. V Controlleri musíme tieto parametre odchytiť a predať ich transformátoru.
3. Jazyk PHP musí podporovať predávanie parametrov.

Nasledujúci príklad ukazuje, ako zadefinujeme vstupné parametre na strane konfiguračného súboru. Upravíme predchádzajúci pravidlo tak, aby pre každý typ triedenia bola použitá rovnaká XSL šablóna.

```
<?xml version="1.0" encoding="windows-1250" ?>
<konfiguracia>
  <pravidlo vzor="{*}/hospodarstvo/dovoz/zotried/{*}">
    <model url="model/{1}.xml" />
    <view url="view/hospodarstvo-dovoz.xsl" />
    <parameter nazov="triedenie" hodnota="{2}" />
  </pravidlo>
</konfiguracia>
```

Pre naše pravidlo sme si zadefinovali nový element s názvom `parameter`. Vhodným návrhom triedy Controller neskôr zabezpečíme predanie tohoto parametru. Parametrov môžeme predať aj viacero. Výhoda je zrejmá, na základe pravidla vidíme, že pracujeme stále s tou istou XSL šablónou.

Pre spracovanie konfiguračného súboru ako aj spustenie transformácie musíme opustiť syntax XML a vytvoriť kód v programovacom jazyku PHP.

4.3 Controller a jeho implementácia v PHP

Controller v našom frameworku vyberá na základe pravidiel v konfiguračnom súbore príslušnú transformáciu. Voliteľne transformácii predá vstupné parametre a výsledok transformácie odošle späť k užívateľovi.

Spracovanie konfiguračného súboru pomocou SimpleXML

Konfiguračný súbor je vo formáte XML. Parser Controllera sa na súbor môže pozerať ako na objektový model dokumentu (DOM) a pracovať s ním podľa štandardov W3C. Tento spôsob je v PHP 5 podporovaný. Pre prácu s XML súbormi však ponúka PHP aj vlastné rozšírenie nazvané SimpleXML. Toto rozhranie patrí medzi takzvané natívne implementácie XML parsera. To znamená, že práca s XML pripomína prácu s údajovými štruktúrami jazyka, pre ktorý bol parser vytvorený. Pomocou SimpleXML môžeme s XML súborom pracovať rovnako ako s objektami jazyka PHP. Celý XML súbor je objekt typu SimpleXML. Dcérske elementy sú prístupné ako členské premenné, cez ktoré môžeme iterovať a atribúty sú prístupné ako položky asociatívneho poľa. Natívne parsery sú známe aj z iných jazykov. V Jave je to napríklad rozšírenie JDOM.

Predpokladajme nasledujúci jednoduchý XML súbor:

```
<?xml version=”1.0” encoding=”windows-1250”?>
<krajina>
  <nazov>Slovensko</nazov>
  <obyvatelstvo>
    <mesto pocet=“450000“>Bratislava</mesto>
    <mesto pocet=“243000“>Košice</mesto>
    <mesto pocet=“96000“>Prešov</mesto>
  </obyvatelstvo>
</krajina>
```

Na začiatok musíme vytvoriť objekt SimpleXML:

```
$s = simplexml_load_file('nazov-xml-suboru.xml');
```

K elementu `nazov` môžeme pristúpiť ako k členskej premennej:

```
echo $s->nazov;   // vrati Slovensko
```

Textový obsah prvého elementu `mesto` získame nasledovne:

```
echo $s->obyvatelstvo->mesto[0];   // vrati Bratislava
```

Pre vypísanie všetkých názvov miest a počtu ich obyvateľov použijeme cyklus:

```
foreach($s->obyvatelstvo->mesto as $m) {
  echo $m; echo $m[‘pocet’]; }
```

S vyššie uvedenými konštrukciami si vystačíme aj pri parsovaní konfiguračného súboru. O spracovanie sa postará metóda parse() triedy Controller. V tejto metóde prechádzame postupne pravidlá a ak nájdeme to, ktorého vzor sa zhoduje s klientsku URL, tak spracúvame jeho obsah. To znamená, že vyberieme správny názov XML súboru s dátami a XSL šablóny pre transformáciu a uložíme si aj prípadné vstupné parametre. Návratová hodnota metódy je true, ak sa pravidlo našlo.

```
  private function parse() {
     // nacitanie konfiguracneho suboru
     $pravidlo = null;
     $xml = simplexml_load_file($this->config);
     // Prechadza kazdy element pravidlo a porovnava vstupne URL so
vzorom
     foreach($xml->pravidlo as $p) {
        $vzor = str_replace(array('{*}', '/'),
                            array('([a-z0-9_-]+)', '\/'),
                            $p['vzor']);
        // Ak sa našlo vyhovujúce pravidlo vyberie ho a skončí
cyklus
        if (preg_match('/^'. $vzor .'$/i', $this->action, $matches))
{
           $pravidlo = $p; break;
        }
     }
     // Ak sa nenaslo ziadne pravidlo vyhovujuce URL tak skonci
     if (!$pravidlo) {
        return false;
     }
     // Nahradenie odkazov na zastupne znaky hodnotou zastupnych
znakov
     foreach($matches as $i => $match) {
        $pravidlo->model['url'] = str_replace("{$i}",
                                              $match,
                                              $pravidlo->model['url']);
        $pravidlo->view['url'] = str_replace("{$i}",
                                             $match,
                                             $pravidlo->view['url']);
     }
     // Spracovanie pripadnych parametrov
     foreach($pravidlo->parameter as $parameter) {
        foreach($matches as $i => $match) {
           $parameter['hodnota'] = str_replace("{$i}",
              $match,
$parameter['hodnota']);
        }
        $this->params[(string)$parameter['nazov']] = (string)

$parameter['hodnota'];
     }
     $this->model = $pravidlo->model['url'];
     $this->view = $pravidlo->view['url'];

     return true;
  }
```

O vlastnú transformáciu vstupného XML dokumentu šablónou sa postára metóda run(). URI oboch týchto súborov máme uložené v členských premenných `model` a `view`. Prípadné parametre, predávané šablóne sú prístupné v členskej premennej `params`, ktorá je asociatívnym poľom. Indexy poľa sú textové reťazce obsahujúce názov parametre a hodnotou príslušného indexu je hodnota parametra.

```
public function run() {
  if (!$this->parse()) {
    return;
  }
  // Načítanie modelu
  $xml = new DOMDocument();
  $xml->load($this->model);

  // Načítanie view
  $xsl = new DOMDocument();
  $xsl->load($this->view);

  // vytvorenie XSLT procesoru
  $proc = new xsltprocessor();
  $proc->importStylesheet($xsl);

  // Predanie parametrov procesoru
  foreach($this->params as $nazov => $hodnota) {
    $proc->setParameter("", $nazov, $hodnota);
  }
  // Odoslanie vystupu klientovi
  echo $proc->transformToXML($xml);
}
```

Zdrojový kód je pre názornosť zjednodušený. Pri načítaní akéhokoľvek zdroja treba vykonať detekciu existenciu súboru. Takisto, ak sa nenašlo žiadne vyhovujúce pravidlo pre klientskú URL, tak najvhodnejšie je presmerovať na nejakú stránku s oznámením.

Záver

V prvej časti práce som rozdelil XML dokumenty podľa ich charakteru a vysvetlil základné princípy XML. Pozornosť som venoval znakovej sade a kódovaniu dokumentov XML. Druhá časť práce je venovaná prezentácii XML dokumentov. Prezentácia pomocou CSS má slabiny v tom, že umožňuje iba pasívne zobrazenie obsahu XML dokumentu, prakticky bez akejkoľvek možnosti jeho zmeny. Tento nedostatok odstraňuje XSLT, ktorý má podľa mňa najväčšiu perspektívu, pretože umožňuje efektnú a efektívnu manipuláciu s obsahom XML dokumentu. Ako ukážku možností XSLT som použil trocha akademický príklad, a to tranzitívny uzáver.

V ďalšej časti práce boli podané všeobecné informácie o štruktúre webových aplikácií. Táto štruktúra je im daná z princípu fungovania internetu a protokolu HTTP. V súčasnosti narážajú webová aplikácie na hranice možností tohoto protokolu.
Protokol HTTP bol od začiatku budovaný ako bezstavový s primárnou funkciou výmeny dokumentov. S využitím funkcií jazyka JavaScript sa pre užívateľa systému snažia vytvoriť ilúziu desktopového programu, tzv. Rich Internet Application. Na internete existuje nepreberné množstvo tzv. Ajax riešení. Ich autori svorne tvrdia, že tvorba webu sa použitím ich frameworku značne zjednoduší a zrýchli. Podľa môjho názoru, riešenie, ktoré sa snaží obísť obmedzenia súčasných technológií a balancuje na hrane možností prehliadača nemôže byť pre vývojara ani jednoduchšie na pochopenie ani rýchlejšie na vytvorenie projektu a už vôbec nie na udržovateľnosť transparentnosti kódu v budúcnosti.
V tejto časti som vysvetlil všeobecné princípy MVC. Pridržiavaním sa jeho princípov môže vývojar vytvárať transparentné webové aplikácie, ktoré sa udržujú ľahšie ako webové aplikácie založené na tzv. Modeli 1.

V štvrtej časti som sa venoval návrhom implementácie MVC prístupu pre jednoduchý framework založený na XML. Pre názornosť som zjednodušil najmä vrstvu Model, ktorou sú XML dáta. Generovanie správneho View zabezpečuje takisto technológia založená na XML a to XSLT. O výber správneho Modelu a View sa stará Controller, ktorého pravidlá sú opäť napísané v čistom XML.

Rozoberám do podrobna fungovanie všetkých komponentov frameworku. Vrstvu Controller som implementoval v PHP. Okrem dôvodou pre výber tohoto jazyka, ktoré som spomenul už v úvode práce sú to aj subjektívne dôvody. S jazykom PHP mám niekoľkoročnú praktickú skúsenosť. Oceňujem jeho výhody pričom si však uvdomujem aj jeho nedostatky.

Jadro frameworku, ktoré som vypracoval má využitie hlavne pre jednoduchšie webové aplikácie, zamerané na poskytovanie informácií užívateľovi. Celá funkcionalita je na strane servera. Hlavnú výhodu vidím v nezávislosti fyzickej adresárovej štruktúry na URL. Pre vytvorenie webovej aplikácie založenej na mojom frameworku stačia len znalosti XML, XSLT a spôsob zápisu pravidiel pre mapovanie URL. Prehliadač aj klient vidí jednoduché URL adresy, ktoré sú základom pre SEO – Optimalizáciu pe vyhľadávače.

Na záver som vysvetlil fungovanie triedy Controller. Zobrazené sú dve kľúčové metódy. Okrem nich tvorí kód triedy konštruktor a definícia piatich privátnych premenných.

Literatúra

[1] Bráza, J. : XML praktické príklady,
Grada publishing, Praha 2003

[2] Bollingerr, G., Natarajan, B. : JSP – Java Server Pages,
Grada, Praha 2003

[3] Harold, E. R., Means W. S. : XML v Kostce,
O' Reilly, Computer Press, Praha 2002

[4] Moller, A., Schwartzbach, M. : An Introduction to XML and Web technologies,
Pearson Education Limited, 2006

[5] Young, M. J. : XML krok za krokem,
Computer Press, Brno 2006

[6] http://www.kosek.cz/xml/xslt/

[7] http://www.interval.cz/

[8] http://www.php.net/manual/en/

[9] http://www.zvon.org/