J I H O Č E S K Á U N I V E R Z I T A

PEDAGOGICKÁ FAKULTA

KATEDRA INFORMATIKY

TRANSFORMACE XML DOKUMENTŮ POMOCÍ JAZYKA XSLT

BAKALÁŘSKÁ PRÁCE

ONDŘEJ SVOBODA

vedoucí diplomové práce
PaedDr. Petr Pexa

ČESKÉ BUDĚJOVICE 2003

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, pouze s použitím literatury a zdrojů uvedených v části Použité zdroje.

Ondřej Svoboda

Obsah

1. Úvod do XML

1.1 Co je XML

XML (eXtended Markup Language) je značkovací jazyk pro dokumenty obsahující strukturované informace.

Strukturované dokumenty obsahují jak obsah (například text, obrázky...), tak i informaci, jakou roli v dokumentu hrají. Například text v záhlaví nebo v zápatí bude mít jiný význam, než ten samý text v těle dokumentu, nebo v tabulce, kterou obsahuje. Skoro všechny dokumenty nějakou strukturu mají.

Značkovací jazyk je mechanismus, kterým identifikujeme strukturu v dokumentu. Specifikace XML definuje standardní způsob, jak značkování do dokumentu přidat.

Tvůrcem XML je XML Working Group, jejíž činnost zaštiťuje konsorcium W3C, kterému vděčíme již za jazyk HTML. Na návrhu XML se začalo pracovat v druhé polovině devadesátých let s cílem vytvořit standard, který by lépe vyhovoval dnešním nárokům na zpracování informací než stávající obdobné formáty. Hlavním dosavadním výsledkem je především specifikace jazyka XML verze 1.0, jejíž první verze byla zveřejněna v únoru roku 1998, zatím poslední pak v říjnu roku 2000.

1.2 Proč XML?

K tomu, abychom XML mohli ocenit, musíme porozumět, proč bylo vytvořeno. XML bylo vytvořeno tak, aby mohli být na webu publikovány strukturované dokumenty, protože ostatní alternativy, jako je HTML nebo SGML, se pro tento problém příliš nehodí.

Nemůžeme ale říci, že XML může nahradit SGML úplně. SGML obsahuje dobré řešení pro tvorbu komplexních dokumentů, u nichž se předpokládá dlouhá životnost.

1.3 Cíle XML

Hlavní cíle jazyka XML shrnuli tvůrci do následujících deseti bodů:

- Musí být široce použitelný na internetu. Uživatelé musejí být schopni zobrazovat XML dokumenty stejně rychle jako dokumenty HTML. To je možné pouze pokud budou XML prohlížeče tak robustní a široce rozšířené jako prohlížeče pro HTML.

- Musí podporovat širokou škálu rozmanitých aplikací. XML by měl být prospěšný pro mnoho různých aplikací: publikování, prohlížení, analyzování a další práce s dokumenty.
- Má být kompatibilní se SGML. Protože mnoho firem má své dokumenty uloženy ve formátu SGML, byl jazyk XML navržen tak, aby přechod mezi těmito dvěma formáty byl co nejjednodušší.
- Má být jednoduché vytvářet aplikace zpracovávající XML dokumenty. Hovorové vyjádření tohoto požadavku je, aby odborníkovi vývoj takovéto aplikace trval zhruba dva týdny.
- Množství volitelných vlastností by mělo být v XML omezeno. Dodatečné vlastnosti přidané do jakékoliv specifikace nevyhnutelně přinášejí problémy při sdílení dokumentů mezi uživateli.
- XML dokumenty mají být člověku srozumitelné. Znamená to, že i když nemáme žádný XML prohlížeč, musíme být schopni přečíst obsah XML dokumentu.
- Návrh řešení XML by měl být rychlý.
- Návrh XML řešení by měl metodický a stručný.
- Tvorba dokumentů XML by měla být snadná. Ačkoli většinou budeme při tvorbě dokumentů XML používat editory k tomu určené, musí být možné takové dokumenty vytvořit v jakémkoli i nejjednodušším textovém editoru
- Stručnost v XML značkách má mít minimální důležitost. Mnoho rysů jazyka SGML bylo vytvářeno s ohledem na množství psaných znaků. Tyto vlastnosti již v XML podporovány nejsou.

1.4 Srovnání XML s jinými formáty

Nyní se pokusím srovnat jednotlivé formáty, které XML specifikace více či méně nahrazuje.

1.4.1 XML a SGML

SGML (Standard Generalized Markup Language) je velmi starý značkovací jazyk. Širší veřejnost se o něj začala zajímat hlavně s příchodem s XML, ale jako ISO standard byl přijatý již v roce 1986. SGML se vlastně stal základ pro mnoho značkovacích jazyků, které dnes používáme – stejně tak pro XML.

Překvapivé je, že SGML je dokonalejší a univerzálnější než samotný jazyk XML, který z něj vychází. Složitost a univerzálnost jazyka SGML je důvodem, proč není tolik rozšířený. Napsat aplikaci, která by byla schopna zpracovávat dokumenty v jazyku SGML je velice obtížné jak z hlediska programátorského, tak i z hlediska hardwarových nároků. I když takové aplikace samozřejmě existují, jejich cena je dosti značná.

XML je navržený jako podmnožina SGML, přičemž zachovává výhody tohoto standardu. Je vlastně jen o málo univerzálnější a flexibilnější, ale je podstatně méně náročnější na používané aplikaci i autory dokumentů. Protože XML je podmnožinou SGML, umožňuje jeho okamžité použití v systémech, kde se s formátem SGML pracovalo.

1.4.2 XML versus HTML

Oproti SGML, které je běžnému uživateli poměrně neznámý, je jazyk HTML (HyperText Markup Language) běžně používaný. Právě kvůli tomuto standardu se stalo elektronické publikování tak populární. HTML je opět podmnožinou jazyka SGML. Tato specifikace je na rozdíl od XML velice jednoduchá a obsahuje pouze málo tagů, takže tvorba těchto dokumentů je velice snadná. Tento jazyk samozřejmě podporuje celá řada aplikací, z nichž nejznámější jsou internetové prohlížeče.

Jednoduchost jazyka HTML je jeho největším nedostatkem. Neobsahuje podporu například pro vícesloupcovou sazbu, ani kontrolu prázdného prostoru. Nelze v něm též definovat vlastní tagy. Tagy které jazyk obsahuje jsou použitelné spíše pro fyzickou nežli logickou strukturu. Jeho další nevýhodou je nestandardní podpora prohlížečů, kteří přidávají do svých produktů mnoho nestandardních rozšíření.

XML vznik především na omezení a možnosti jazyka HTML. Zachovává stávající jednoduchost, přidává možnost definování vlastních tagů a zavádí používání přísnějších syntaktických pravidel. Do doby, než XML bude implementováno do internetových prohlížečů a dalších produktů je zajištěna existence obou těchto jazyků například pomocí XHTML

1.4.3 XML versus PDF

Posledním formátem, který se pokusím s XML srovnat je PDF (Portable Document Format). Tento formát byl vytvořený společností Adobe. Jejím cílem bylo vytvořit formát, umožňující jednoduchý přenos i velice vizuálně složitých dokumentů mezi platformami. K jeho tvorbě i zpracování dokumentů PDF již existuje celá řada aplikací, jak od firmy Adobe (Acrobat, Distiller), tak i od ostatních výrobců.

Hlavní nevýhodou tohoto formátu je orientace na zobrazení a ne na obsah dokumentu. To způsobuje problémy při indexování a prohledávání dokumentů PDF.

2. Model vrstev

Někdy je užitečné přemýšlet o celé architektuře XML jako o struktuře, která má 4 vrstvy: XML jádro, základní XML toolkit, schéma a dotazování a XML aplikace.

2.1 XML jádro

Nejdůležitější součástí jádra XML je specifikace samotné syntaxe XML, která definuje syntaxi “hranatých závorek" XML dokumentů, stejně tak jako trochu tajemnější specifikace pro Document Type Descriptions (DTD). Do této vrstvy náleží rovněž specifikace XML Namespaces, která popisuje, jak se mohou různé organizace využívající XML vyvarovat rozporů mezi různými slovníky, společně s XML Infoset, což není ani tak standard, jako spíše způsob modelování informací, které jsou v XML dokumentech obsaženy.

Brzy bude tato vrstva obohacena o XML 1.1 a XML Namespaces 1.1. Jak už číslování naznačuje, bude se jednat spíše o malé změny současné specifikace a nové funkce pravděpodobně mnohé uživatele vůbec nezasáhnou. XML 1.1 je navrženo tak, aby uspokojilo dvě zanedbávané menšiny: uživatele etiopského písemného systému a uživatele mainframů IBM.

Změny zasáhnou ty firmy, které produkují XML parsery. Je zde určité riziko, že během období přechodu firma zjistí, že přijímá XML 1.1 dokumenty od některého ze svých obchodních partnerů ještě před tím, než upgraduje na použití XML 1.1 parseru, a to možná dokonce ještě dříve, než bude XML 1.1 parser dostupný pro určitou výpočetní platformu.

Je samozřejmě možné argumentovat tím, že uživatelé, kterým změny přinesou nějaký užitek, jsou v menšině oproti těm, kteří se budou muset vyrovnat s narušením, které přechod na novou verzi způsobí. Ovšem vzhledem k tomu, že specifikace byla stabilní po dobu 5 let a World Wide Web Consortium (W3C) dělá vše pro to, aby si udrželo svoji pověst, není si proč stěžovat.

2.2 Základní XML toolkit

Vrstva číslo 2 je tím, co zde můžeme nazvat základním XML toolkitem. Nástroje, které obsahuje, pak lze klasifikovat jako DOM, SAX, Xpath a XSLT. Ty jsou zde také už několik let a většina XML programátorů je s nimi dobře obeznámena. Je dostupné množství implementací, které pokrývají všechny populární platformy při zachování dobré úrovně interoperability mezi nimi. Pravdou ale je, že má v rámci specifikace mnoho variant a modulů, jakož i hodně rozšíření od různých výrobců. Např. Uživatelé pracující na platformě Microsoftu často ani netuší, zda využívají standardní funkce DOM, nebo nadstavby Microsoftu.

Xpath a XSLT nyní procházejí obdobím výrazného upgradu směřujícího k vytvoření Xpath 2.0 a XSLT 2.0. Ačkoliv produkty založené na verzích 1.0 byly úspěšné, jejich možnosti jsou omezené a uživatelé dlouho čekali na možnosti jako seskupování dat podle společných hodnot či podporu regulárních výrazů. Jedním z důvodů, proč vývoj nových verzí trval tak dlouho, je vliv specifikací XML Schema a Xquery.

Pracovní skupiny W3C učinily kontroverzní rozhodnutí, úzce propojit příští verze XSLT a Xpath se specifikacemi XML Schema. To mění systém typů, který formuluje páteř jakéhokoliv programovacího jazyka, což vyžadovalo některé výrazné technické změny, které si vyžádaly dlouhou dobu pro své schválení. Nicméně v průběhu letošního roku by mělo k dokončení těchto specifikací dojít.

SAX je zajímavý jakožto vzácný příklad standardu, který byl vytvořen jednotlivci, spolupracujícími bez formální organizace, která by jejich aktivitu zaštítila a podporovala. I když to však fungovalo dobře při vytváření původní specifikace, objevují se známky toho, že tvorba dodatečných upgradů už může být při použití tohoto modelu těžší. Každopádně se možná ukáže, že tak lze předejít rostoucí komplikovanosti, jíž trpí prakticky všechny ostatní aplikace.

2.3 Schéma a dotazování

O 3. vrstvě můžeme někdy mluvit jako o “správě informací v XML", protože reprezentuje přítomnost XML, coby klíčové technologie ve způsobu, jakým firmy řídí svůj informační majetek. To je rozdíl oproti původní roli XML jakožto syntaxe pro označování běžných dokumentů.

Dvěma hlavními komponentami této vrstvy jsou XML Schema, které popisuje strukturu informací, a Xquery, dotazovací jazyk pro přístup k XML datům. XML Schema bylo jako standard přijato v květnu 2001. Jde o vysoce komplexní specifikaci a až dosud existuje relativně málo implementací, a ty jsou ještě nekompletní.

Srovnatelně komplexní a úzce související s XML Schema je Xquery, specifikace navržená pro dotazování v XML databázích, která je zkonstruována tak, aby mohla hrát stejnou roli ve světě XML, jako hraje SQL ve světě relačních dat. Zrod Xquery byl poměrně zdlouhavý – pravděpodobně proto, že v něm bylo zainteresováno množství firem.

XML Schema i Xquery rozpoutaly poměrně velkou polemiku. Jde o velmi komplexní standardy, kterým není jednoduché porozumět a už vůbec ne je implementovat, pročež nejsou příliš přístupné pro open source komunitu, ačkoliv právě ona odvedla velký kus práce pro použitelnost a dostupnost XML vrstev 1 a 2, a potažmo tedy pro jejich úspěch.

Nicméně u velkých firem mají W3C specifikace velmi širokou podporu a jak se zdá, budou tyto standardy v dohledné době adoptovány. To podtrhuje i oznámení Microsoftu týkající se nového balíku Office – vyplývá z něj, že pokud budete chtít využívat výhod XML, budete se zřejmě muset naučit žít s XML Schema, i když se jej možná nenaučíte milovat.

2.4 XML aplikace

Čtvrtou vrstvou je oblast, kterou pro jednoduchost názvu XML aplikacemi. Do této oblasti řadíme všechny specifikace vyšší úrovně, které představují využití XML, jako jsou např. standardy, jež podpírají architekturu webových služeb, a specifikace XML Formatting Objects, pro publikování založené na XML ve vysoké kvalitě. Můžeme sem zařadit rovněž nespočetné specializované XML slovníky, pokrývající snad každou myslitelnou oblast od finančních informací, obchodních reportů a přenosu zpráv, až po specializované slovníky, jako MusicXML pro publikování hudby či HumanXML, který popisuje “lidské charakteristiky".

2.5 Ostatní specifikace

Jsou zde samozřejmě i mnohé další specifikace, které do této struktury nezapadají zcela přesně – např. Standardy Xpointer a Xlink pro hyperlink.

Xpointer, který umožňuje, aby dokument obsahoval odkazy na obsah jiného dokumentu, byl několikrát přepsán a současná verze už snad může přilákat pozornost a zajistit širší podporu, než tomu bylo u jejích předchůdců.

Xlink, který staví na Xpointeru s cílem definovat model pro hyperlink mezi dokumenty, stále trpí díky faktu, že víceméně nikdo neví, jak vlastně zapadá mezi ostatní specifikace rodiny XML.

3. XSL

Extensible style Language (XSL) zahrnuje transformační i formátovací jazyk. Každý z těchto jazyků je vlastně XML aplikací. Transformační jazyk poskytuje prvky, které definují pravidla, jak má byt dokument XML transformován do jiného XML dokumentu, nebo do jiného formátu. Výsledný dokument může využít značkování a DTD originálního dokumentu, nebo může využívat úplně jinou množinu tagů.

3.1 Co je xsl?

XSL se v podstatě skládá ze dvou jazyků. První jazyk je transformační a druhý je formátovací. Formátovací část jazyka XSL, která překračuje rámec mé diplomové práce, je založena na speciálních formátovacích objektech. Zmíněná část XSL je proto často označována jako XSL-FO – XSL formating objects. Jazyk XSL-FO je velmi složitý, neboť návrh vzhledu dokumentů pomocí formátovacích objektů bývá obvykle velmi komplikovaný. Specifikace XSLT byla k jazyku XSL přidána původně hlavně proto, aby celý proces transformace dokumentů XML usnadnila.

Transformační a formátovací části jazyka XSL mohou fungovat nezávisle jedna na druhé. Transformační jazyk může například přetransformovat XML dokument na správně strukturovaný HTML soubor a vůbec nemusí využít jazyka XSL-FO.

3.2 Stromy

Při XSL transformaci přečte XSL procesor dokument XML a XSL šablonu, a podle instrukcí najde ve stylu XSL, jak má výsledný dokument vypadat.

Každý XML dokument je strom sestavený z uzlů. Jeho obsah je považován za množinu uzlů (prvků, komentářů, atributů, jmenných prostorů, transformačních instrukcí, atd) uspořádanou do určité hierarchie. XSLT procesor předpokládá XML strom obsahující sedm druhů uzlů:

1. Kořenový uzel – kořenový uzel je něco jiného než kořenový prvek v XML dokumentu
2. Uzel prvku
3. Textový uzel
4. Uzel atributu
5. Uzel jmenného prostoru
6. Uzel obsahující zpracovávající instrukce
7. Uzel komentáře

Uvažme například následující XML dokument. Ten obsahuje tabulku motocyklů, kterou využiji jako příklad. (Přesněji - obsahuje pouze první dva dva motocykly z celého seznamu).

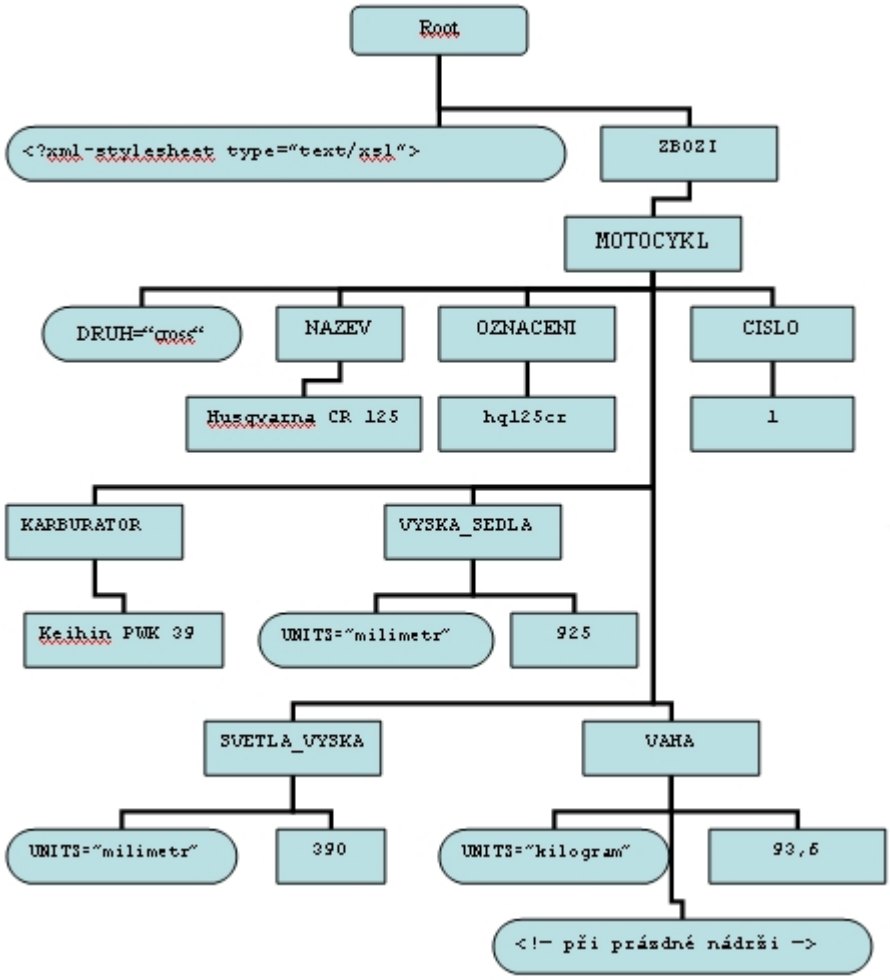
Kořenový prvek ZBOZI obsahuje potomka MOTOCYKL. Každý MOTOCYKL obsahuje mnoho potomků, jako je značka, hmotnost, typ karburátoru atd. Atribut UNIT specifikuje jednotku, která se vztahuje k danému prvku.

```
<?xml version="1.0" encoding="windows-1250"?>
<?xml-stylesheet type="text/xsl" href="zbozi.xsl"?>
<ZBOZI>
  <MOTOCYKL DRUH="cross">
    <NAZEV>Husqvarna CR 125</NAZEV>
    <OZNACENI>hq125cr</OZNACENI>
    <CISLO>1</CISLO>
    <KARBURATOR>Keihin PWK 39</KARBURATOR>
    <VYSKA_SEDLA UNITS="milimetr">925</VYSKA_SEDLA>
    <SVETLA_VYSKA UNITS="milimetr">390</SVETLA_VYSKA>
    <VAHA UNITS="kilogram">
      <!-- při prázdné nádrži -->93,6</VAHA>
  </MOTOCYKL>
  <MOTOCYKL DRUH="cross">
    <NAZEV>Husqvarna CR 250</NAZEV>
    <OZNACENI>hq250cr</OZNACENI>
    <CISLO>2</CISLO>
    <KARBURATOR>Keihin PWK 38 S</KARBURATOR>
    <VYSKA_SEDLA UNITS="milimetr">925</VYSKA_SEDLA>
    <SVETLA_VYSKA UNITS="milimetr">385</SVETLA_VYSKA>
    <VAHA UNITS="kilogram">
      <!-- při prázdné nádrži -->97,2</VAHA>
  </MOTOCYKL>
</ZBOZI>
```

Na následujícím obrázku je diagram, který popisuje tento XML dokument jako strom. Začíná v kořenovém uzlu(není to kořenový prvek!), který obsahuje dva potomky: zpracovávající in-strukce xml-stylesheet a kořenový prvek ZBOZI. (XML deklarace není pro XSL procesor viditelná a není ani zahrnutá ve stromu, který XSL procesor zpracovává). Prvek ZBOZI obsahuje opět dva potomky, oba to jsou prvky MOTOCYKL. Každý prvek MOTOCYKL má atribut DRUH a mnoho dalších potomků. Každý prvek obsahuje uzel obsahu, stejně tak i uzel pro všechny jeho atributy. Máme tu uzly pro text, atributy, komentáře a zpracovávající instrukce. Na rozdíl od CSS není XSL omezeno k práci pouze s celými prvky. Umožňuje mnohem jemnější pohled na dokument, který dává možnost aplikovat různé styly na komentáře, atributy atd.

Stejně jako XML deklarace není DTD podmnožinou deklarace DOCTYPE, a proto také není součástí XSL stromu.

Obrázek č.1 – Strom XML dokumentu



Transformační jazyk XSL transformuje jeden XML strom do druhého. Jazyk XSL obsahuje operátory pro vybírání příslušných uzlů ze stromu, přeskupování uzlů a pro jejich výstup. Všechny tyto operátory včetně vstupu i výstupu jsou navrženy pro ope-race se stromem, proto také nemohou být použity při transformování libovolných nestrukturovaných dat.

XSL transformace přijímá jako vstup strom, který reprezentuje XML dokument a vytváří výstup ve formě jiného stromu, který také reprezentuje XML dokument. Kvůli tomu se transformační část jazyka XSL někdy také nazývá částí vytvářející strom. Vstupem i výstupem musejí být dokumenty XML. Nemůžeme proto využívat XSL k transformaci z, nebo do formátů, které XML dokumenty nejsou, jako je například PDF, Tex, Microsoft Word, PostScript, MIDI nebo jiné. XSL můžeme využít pro transformaci XML, do takzvaných mezi formátů, jako je TeXML a potom využít software pro transformování dokumentu do formátu, který chceme.

HTML a SGML jsou zvláštní případy, protože jsou to formáty blízké XML. Můžeme proto využít XSL k transformaci do, nebo z HTML a SGML.

3.3 XSL Šablony

XSL dokument obsahuje seznam vzorových nebo jiných pravidel. Vzorové pravidlo má vzor, specifikující na jakou část stromu má být použito a šablonu, která má být použita jako výstup pokud se vzor shoduje. Když XSL procesor formátuje XML dokument pomocí xsl:stylesheet, zkoumá každý podstrom XML dokumentu. Když přečte každý podstrom XML dokumentu, porovná jej s každým vzorem v šabloně. Jestliže nalezne strom, který nějaké šabloně odpovídá, je jeho výstup daný šablonou. Tento výstup se většinou skládá z nových dat, dat zkopírovaných, z originálního dokumentu a značek.

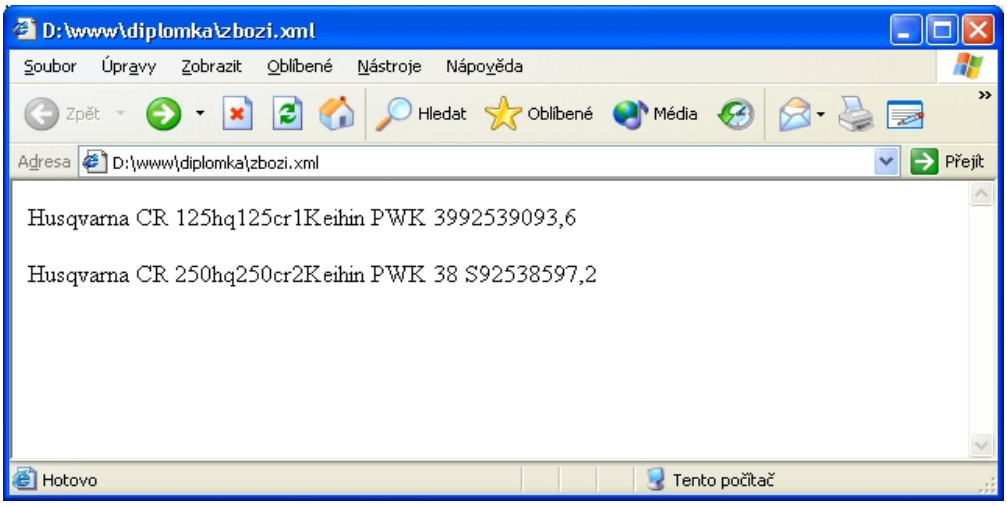
K popsání těchto pravidel, šablon a vzorů využívá XSL dokument XML. Sám o sobě je XSL dokument xsl:stylesheet prvek. Každé vzorové pravidlo xsl:template je prvek. Vzor tohoto pravidla odpovídá hodnotě atributu xsl:template. Výstupem je obsah prvku xsl:template. Všechny instrukce, které používáme, jsou vykonány pomocí nějakého XSL prvku. Identifikovány jsou prefixem xsl:.

Můj příklad ukazuje velmi jednoduchou XSL šablonu se dvěma vzorovými pravidly. První pravidlo odpovídá kořenovému prvku ZBOZI. Tento prvek je nahrazen prvkem HTML. Obsah prvku HTML je výsledkem aplikování dalších vzorů na obsah prvku ZBOZI.

Druhý vzor odpovídá prvku MOTOCYKL. Nahrazuje prvek MOTOCYKL na vstupu dokumentu prvkem P na výstupu. Pravidlo xsl:apply-templates přidává do výstupu text, který odpovídá vstupní hodnotě tak, že obsah prvku P je text(neznačkovaný) obsahující odpovídající prvek MOTOCYKL.

Příklad 1

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="ZBOZI">
    <html>
      <xsl:apply-templates/>
    </html>
  </xsl:template>
  <xsl:template match="MOTOCYKL">
  <P>
    <xsl:apply-templates/>
  </P>
  </xsl:template>
</xsl:stylesheet>
```


D:\www\diplomka\zbozi.xml
Soubor Úpravy Zobrazit Oblíbené Nástroje Nápověda
Zpět
Hledat
Oblíbené
Média
Adresa
D:\www\diplomka\zbozi.xml
Přejít
Husqvarna CR 125hq125cr1Keihin PWK 3992539093,6
Husqvarna CR 250hq250cr2Keihin PWK 38 S92538597,2
Hotovo
Tento počítač

4. Práce se šablonami

4.1 Prvek xsl:template

Vzorová pravidla definovaná pomocí xsl:template jsou nejdůležitější částí každého XSL dokumentu. Každé vzorové pravidlo je prvek xsl:template. Každý prvek xsl:template má atribut match, který specifikuje, na který uzel vstupního dokumentu má být aplikován.

Vzor může obsahovat text, který je předdefinován v šabloně, nebo můžeme XSL instrukcí kopírovat data přímo ze souboru XML. Protože všechny XSL instrukce patří do jmenného prostoru xsl (proto začínají prefixem xsl:), je jednoduché rozlišit prvky, které jsou skutečnými daty kopírovanými na výstup pomocí XSL instrukce. Jako příklad uvedu šablonu, která je aplikována na kořenový prvek vstupního stromu:

```
<xsl:template match="/">
  <html>
    <head>
    </head>
    <body>
    </body>
  </html>
</xsl:template>
```

Když XSL procesor čte vstupní dokument, první uzel na který narazí je kořen(root). Toto pravidlo říká procesoru, aby jako výstup použil následující text:

```
<html>
<head>
</head>
<body>
</body>
</html>
```

Tento text je správě strukturovaný dokument HTML. Protože XSL dokument je sám o sobě XML dokument, musí být jeho obsah – obsah jeho šablon správně strukturovaný.

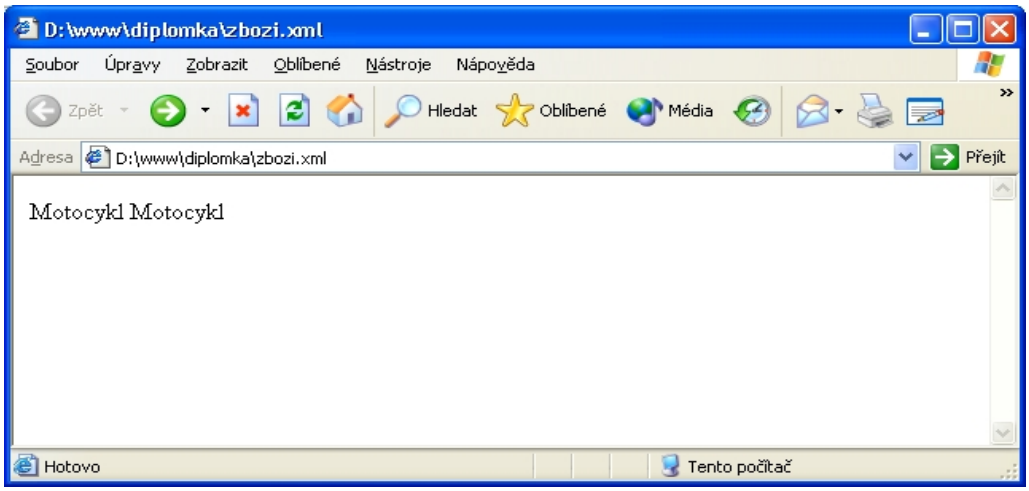
Pokud na XML dokument použijeme pouze tuto šablonu, tak výstup bude obsahovat šest prvků, které budou nakonec zhuštěny do následujících čtyř: <html><head/><body/></html>. Je tomu tak proto, že žádná instrukce neříká XSL procesoru, aby se posunul ve stromu o další pozici a zpracoval tak další potomky.

4.2 Prvek xsl:apply-templates

Abychom se ve zpracování našeho dokumentu dostali dále, je nutné nějak sdělit XSL procesoru, aby ve zpracovávání dokumentu pokračoval. V zásadě se pro zahrnutí potomků do výstupu používá rekurzivní procházení jednotlivých prvků. Toto volání zajišťují prvky xsl:apply-templates. Zahrnutím xsl:apply-templates do výstupu říkáme překladači, aby porovnal každý prvek, který je potomkem daného uzlu se šablonami v XSL souboru. Pokud žádnou šablonu nenajde, bude na výstupu šablona uzlu, který vyhovuje. Šablona vyhovujícího prvku může opět obsahovat prvek xsl:apply-templates, což zapříčiní vyhledávání v potomcích daného prvku. Když formátovací stroj uzel zpracuje, je uzel zpracovaný jako kompletní strom, což je výhoda stromové struktury. Každá část může být zpracována stejným způsobem jako celý dokument. Na dalším příkladu je vidět použití xsl:apply-templates pro zpracování potomků daného uzlu.

Příklad 2

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <xsl:apply-templates/>
    </html>
  </xsl:template>
  <xsl:template match="ZBOZI">
    <body>
      <xsl:apply-templates/>
    </body>
  </xsl:template>
  <xsl:template match="MOTOCYKL">
    Motocykl
  </xsl:template>
</xsl:stylesheet>
```



Nyní si ukážeme, co se stane, pokud použijeme tento styl na náš příklad s motocykly:

1. Procesor porovná kořenový uzel se všemi šablonami v XSL souboru. Vyhovuje hned ta první.
2. Na výstup je zapsán tag <html>
3. Prvek xsl:apply-templates způsobí zpracování potomků
   3.1. První potomek kořene – zpracovávající instrukce xml-stylesheet, je porovnána se šablonou. Nevyhovuje, a proto se negeneruje žádný výstup.
   3.2. Prvek ZBOZI, což je druhý potomek, je porovnán se šablonou a porovnání vyhovuje.
   3.3. Na výstup je zapsán tag <body>
   3.4. Prvek xsl:apply-templates v prvku body způsobí zpracování potomků uzlu ZBOZI
      a) První potomek uzlu ZBOZI, což je MOTOCYKL Husqvarna CR 125, porovná procesor se šablonou a vyhovuje třetí pravidlo
      b) Na výstup se vypíše text „Motocykl“
      c) Druhý potomek uzlu ZBOZI, což je MOTOCYKL Husqvarna CR 250, porovná procesor se šablonou a vyhovuje třetí pravidlo
      d) Na výstup se opět vypíše text „Motocykl“
   3.5. Tag </body> se zapíše na výstup
4. Vypíše se tag </html>
5. Proces je u konce

Zde vidíme výsledek:

```
<html>
   <body>
     Motocykl
     Motocykl
   </body>
</html>
```

4.2.1 Atribut select

Abychom mohli nahradit text „Motocykl“ jménem prvku MOTOCYKL tak, jak je uvedené v jeho obsahu, je nutné specifikovat, že šablona by měla být aplikována na tag NAZEV prvku MOTOCYKL. K výběru určité množiny potomků namísto všech potomků použijeme opět xsl:apply-templates nyní s atributem select určující potomka, který má být zpracován. Zde je příklad:

```
<xsl:template match="MOTOCYKL">
  <xsl:apply-templates select="NAZEV"/>
</xsl:template>
```

Atribut select využívá stejný druh vzoru, jako atribut match prvku xsl:template. Nyní nám pro jednoduchý výpis jména prvku postačí tato jednoduchá šablona, ale dále si probereme mnoho možností jak pro prvek select, tak pro match. Jestliže atribut select není zadán, jsou vybráni všichni potomci daného uzlu.
Výsledek po přidání tohoto pravidla je zde:

```
<html><head/><body>
Husqvarna CR 125
Husqvarna CR 250
</body></html>
```

4.2.2 Atribut mode

Někdy potřebujeme vložit stejný obsah zdrojového dokumentu do dokumentu výstupního vícekrát. To můžeme jednoduše udělat tak, že aplikujeme jednu a tu samou šablonu vícekrát všude tam, kde chceme data vložit. Představme si ale, že bychom chtěli, aby data byla na různých místech v dokumentu různě formátována.

Například bychom chtěli vypsat již dobře známý seznam motocyklů tak, aby na každý motocykl byl odkaz na jeho místo dále v dokumentu, kde by byl jeho podrobnější popis. V tomto případě bychom chtěli, aby dokument začínal takto:

```
<UL>
<LI><A HREF="#hq125cr">Husqvarna 125 CR</A></LI>
<LI><A HREF="#hq250cr">Husqvarna 250 CR</A></LI>
<LI><A HREF="#hq410te">Husqvarna 410 TE</A></LI>
<LI><A HREF="#hq610tc">Husqvarna 610 TC</A></LI>
…
```

Dále v dokumentu aby jednotlivé motocykly vypadali takto:

```
<H3><A NAZEV="hq125cr">Husqvarna 125 CR</A></H3><P>
Husqvarna 125 CR
hq125cr
1
Keihin PWK 39
925
390
93,6
</P>
```

Toto řazení je běžné vždy, pokud automaticky generujeme hypertextovou tabulku s obsahem. NAZEV motocyklu musí být formátován jinak v obsahu a jinak v těle dokumentu. Potřebujeme tedy dvě pravidla, obě aplikovaná na prvek MOTOCYKL na různých místech dokumentu. Řešením, jak toto provést, je nastavit atribut mode. Můžeme potom vybrat, kterou šablonu chceme použít pouze nastavením atributu mode v prvku xsl:apply-templates, jako je vidět na následujícím výpisu:

Příklad 3

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/ZBOZI">
    <HTML>
      <HEAD><TITLE>Motocykly</TITLE></HEAD>
      <BODY>
        <H2>Obsah</H2>
        <UL>
       <xsl:apply-templates select="MOTOCYKL" mode="nadpis"/>
        </UL>
         <H2>Motocykly</H2>
         <xsl:apply-templates select="MOTOCYKL" mode="vypis"/>
      </BODY>
    </HTML>
  </xsl:template>
  <xsl:template match="MOTOCYKL" mode="nadpis">
    <LI><A>
    <xsl:attribute name="HREF">#<xsl:value-of
    select="OZNACENI"/></xsl:attribute>
    <xsl:value-of select="NAZEV"/>
  </A></LI>
  </xsl:template>
  <xsl:template match="MOTOCYKL" mode="vypis">
  <H3><A>
    <xsl:attribute name="NAZEV">
    <xsl:value-of select="OZNACENI"/>
    </xsl:attribute>
    <xsl:value-of select="NAZEV"/>
  </A></H3>
  <P>
    <xsl:value-of select="."/>
  </P>
  </xsl:template>
</xsl:stylesheet>
```

4.3 Určování hodnoty uzlu pomocí xsl:value-of

Tento prvek kopíruje hodnotu uzlu ze vstupního dokumentu na výstupní. Atribut select prvku xsl:value-of určuje, které uzly mají být vybrány.

Představme si například, že bychom chtěli nahradit text „Motocykl“ jménem prvku MOTOCYKL tak, jak je uvedeno u každého prvku NAZEV. Můžeme to udělat takto:

```
<xsl:template match="MOTOCYKL">
  <xsl:value-of select="NAZEV"/>
</xsl:template>
```

Po aplikování tohoto stylu dostaneme následující výstup:

```
<html><head/><body>
Husqvarna CR 125Husqvarna CR 250
</body></html>
```

Prvek, který byl vybrán, v našem případě prvek NAZEV, se vztahuje ke zdrojovému uzlu. Zdrojový uzel je ten prvek, který šabloně vyhovuje, což je v našem případě prvek MOTOCYKL. Když pomocí výběru <xsl:template-match=“MOTOCYKL“> vyhovuje MOTOCYKL Husqvarna CR 125, vybereme jeho prvek NAZEV pomocí xsl:value-of. Když vyhovuje MOTOCYKL Husqvarna CR 250, je jeho prvek MOTOCYKL vybrán také pomocí xsl:value-of.

Hodnota prvku je vždy typu řetězec nebo prázdný řetězec. Přesný obsah řetězce závisí na typu uzlu, přičemž nejběžnější typ uzlu je prvek. Je to zřetězení všech analyzovaných znaků (kromě značek) mezi začátkem a koncem prvku. Na příklad první prvek MOTOCYKL v našem příkladu vypadá takto:

```
<MOTOCYKL DRUH="cross">
  <NAZEV>Husqvarna CR 125</NAZEV>
  <OZNACENI>hq125cr</OZNACENI>
  <CISLO>1</CISLO>
  <KARBURATOR>Keihin PWK 39</KARBURATOR>
  <VYSKA_SEDLA UNITS="milimetr">925</VYSKA_SEDLA>
  <SVETLA_VYSKA UNITS="milimetr">390</SVETLA_VYSKA>
  <VAHA UNITS="kilogram"><!— při prázdné nádrži—>93,6</VAHA>
</MOTOCYKL>
```

Hodnota tohoto prvku vypadá takto:

```
Husqvarna CR 125
hq125cr
1
Keihin PWK 39
925
390
93,6
```

Tento výstup dostaneme pouhým vyjmutím všech tagů z obsahu. Všechno ostatní včetně prázdných mezer zůstane nedotčeno. Hodnota všech šesti prvků je vypočítána stejně, většinou snadno pochopitelnou cestou. Hodnoty uzlů uvedu v následující tabulce:

Tabulka č.1 – Tabulka uzlů

Tabulka uzlů	
Typ uzlu	***Hodnota***
Kořen	hodnota kořenového prvku
Text	zřetězení všech analyzovaných znaků obsažených v prvku, včetně všech znaků obsažených v potomcích prvku
Atribut	
Jmenný prostor	URI jmenného prostoru
Zpracovávající instrukce	hodnota zpracovávající instrukce, neza-hrnuje jméno instrukce, a znaky <? a ?>
Komentáře	obsahuje text, který obsahuje, kromě značek <? a ?>

4.4 Určování hodnoty uzlu pomocí current()

Tato funkce nám vrací množinu uzlů, které obsahuje uzel, který funkci předáme v parametru. Obvykle uzel, který právě zpracováváme a uzel kontextu jsou stejné uzly.

Proto

```
<xsl:value-of select=".">
```

má stejný výsledek jako

```
<xsl:value-of select="current()">
```

Existuje ovšem i jedna výjimka, kdy existuje rozdíl. Následující výraz jazyka Xpath "ZBOZI/MOTOCYKL" vybere uzel <ZBOZI>, což je potomek uzlu, který právě zpracováváme a potom vybere jeho potomka <motocykl>. To znamená, že v každém kroku označuje tečka "." jinou hodnotu.
Následující kód:

```
<xsl:apply-templates select="//VYSKA_SEDLA[@UNITS=current()/@VALUE]"/>
```

zpracuje všechny prvky VYSKA_SEDLA, které mají atribut UNITS se stejnou hodnotou, jako je hodnota atributu VALUE, právě zpracovávaného prvku. To je ovšem rozdílné od:

```
<xsl:apply-templates select="//VYSKA_SEDLA[@UNITS=./@VALUE]"/>
```

což zpracuje všechny prvky VYSKA_SEDLA, které mají atribut UNITS i atribut VALUE se stejnou hodnotou.

4.5 Zpracování více prvků pomocí xsl:for-each

Prvek xsl:value-of by měl být používán pouze v takovém případě, kdy je jednoznačné, který uzel potřebujeme získat. Jestliže existuje více možností, které mohou být vybrány, bude vybrána pouze první vyhovující možnost. Například toto pravidlo, které vybere pouze první prvek z naší tabulky:

```
<xsl:template match="ZBOZI">
  <xsl:value-of select="MOTOCYKL"/>
</xsl:template>
```

a zde je výstup:

```
Husqvarna CR 125 hq125cr 1 Keihin PWK 39 925 390 93,6
```

Existují dva způsoby, jak můžeme zpracovat více prvků najednou. První způsob jsme si již ukázali: jednoduše použijeme xsl:apply-templates s atributem select, který vybere příslušné prvky.:

```
<xsl:template match="ZBOZI">
  <xsl:apply-templates select="MOTOCYKL"/>
</xsl:template>
<xsl:template match="MOTOCYKL">
  <xsl:value-of select="."/>
</xsl:template>
```

Hodnota atributu select="." v druhé šabloně říká XSL procesoru, aby použil hodnotu shodujícího se prvku, což je v našem příkladě prvek MOTOCYKL.

Druhou možností je xsl:for-each. Tento prvek zpracuje každý prvek vybraný pomocí atributu select. Nepotřebujeme k tomu ani žádné další šablony.

```
<xsl:template match="ZBOZI">
  <xsl:for-each select="MOTOCYKL">
    <xsl:value-of select="."/>
  </xsl:for-each>
</xsl:template>
```

4.5 Implicitní pravidla pro šablony

Je celkem nepohodlné pečlivě opisovat hierarchii XML dokumentu do našeho XSL stylu. To se projeví obzvláště, když dokumenty nejsou hierarchicky stejné, jako je většina webových stránek. V těchto případech bychom měli mít nějaké implicitní pravidlo, které bychom aplikovali na všechny prvky bez ohledu na to, kde se v dokumentu vyskytují.

Aby byl tento proces jednodušší, XSL definuje dvě implicitní šablony. První rekurzivně prochází strom prvků a aplikuje šablonu na všechny potomky všech prvků. To nám dává záruku, že se pravidlo skutečně aplikuje na všechny prvky. Druhé implicitní pravidlo se aplikuje na textové uzly a kopíruje jejich hodnotu do výstupního toku. Dohromady tyto dvě pravidla zajistí, že dokonce i prázdná XSL šablona, nemající žádné prvky, bude stále vypisovat nezpracovaná znaková data ze vstupního dokumentu na výstup.

4.5.1 Implicitní pravidlo pro prvky

První implicitní pravidlo se aplikuje na uzly jakéhokoliv typu, nebo na kořenový uzel:

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

* | / je zkratka pro „všechny prvky, nebo pro kořenový prvek“. Smysl tohoto pravidla je zajištění rekurzivního zpracování všech prvků, dokonce i když nejsou zpracovány pomocí explicitního pravidla. To znamená, že pokud toto pravidlo přetížíme, budou zpracovány všechny prvky.

Ovšem když explicitně uvedeme pravidlo pro rodičovský prvek, toto pravidlo nebude aktivováno pro potomky, pokud toto pravidlo neobsahuje xsl:apply-templates. Například můžeme zastavit celé zpracování tak, že aplikujeme šablonu na kořenový prvek a šablona bude vypadat takto:

```
<xsl:template match="/">
</xsl:template>
```

4.5.2 Implicitní pravidlo pro textové uzly

Implicitní pravidlo pro textové uzly nám jednoduše kopíruje zdrojový uzel na výstup tak, že jeho hodnota je úplně stejná. Toto pravidlo vypadá takto:

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

Toto pravidlo odpovídá všem textovým uzlům (match="text()") a výstupní hodnota je hodnota tohoto uzlu (<xsl:value-of select="."/>). Jinými slovy kopíruje text ze vstupu na výstup.

Toto pravidlo zajišťuje, že i ten nejmenší textový prvek je vytištěn na výstup, dokonce i když není specifikované žádné jiné pravidlo.

4.5.3 Důsledek dvou implicitních pravidel

Dohromady tyto dvě pravidla způsobují, že aplikování prázdné šablony pouze s prvkem xsl:stylesheet, ale s žádnými potomky na XML dokument, zkopíruje všechna #PCDATA ze vstupu na výstup. Proto tato metoda nevytváří ani žádné značky. Toto je hodně jednoduché pravidlo a proto je přetíženo jakýmkoli jiným, které specifikujeme mi jako programátoři.

jednoduchá šablona:

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

5. Porovnávání

Atribut match uzlu xsl:template podporuje kompletní syntaxi, která umožňuje přesné vyjádření uzlu, který potřebujeme. Atribut select prvků xsl:apply-templates, xsl:value-of, xsl:for-each, xsl:copy-of a xsl:sort podporují dokonce ještě výkonnější syntaxi, která nám umožňuje přesně vyjádřit, který uzel zahrnout chceme a který ne. Nyní ukáži jednotlivé šablony.

5.1 Porovnávání uzlu Root

Aby výsledný dokument byl správně formátovaný, je třeba, aby prvním prvkem na výstupu byl kořenový prvek dokumentu. Z toho vyplývá, že XSL styl vždy začíná pravidlem, které vyhovuje kořenovému prvku. Abychom uzel Root specifikovali, použijeme jako hodnotu atributu match „/“.

```
<xsl:template match="/">
  <html>
    <xsl:apply-templates/>
  </html>
</xsl:template>
```

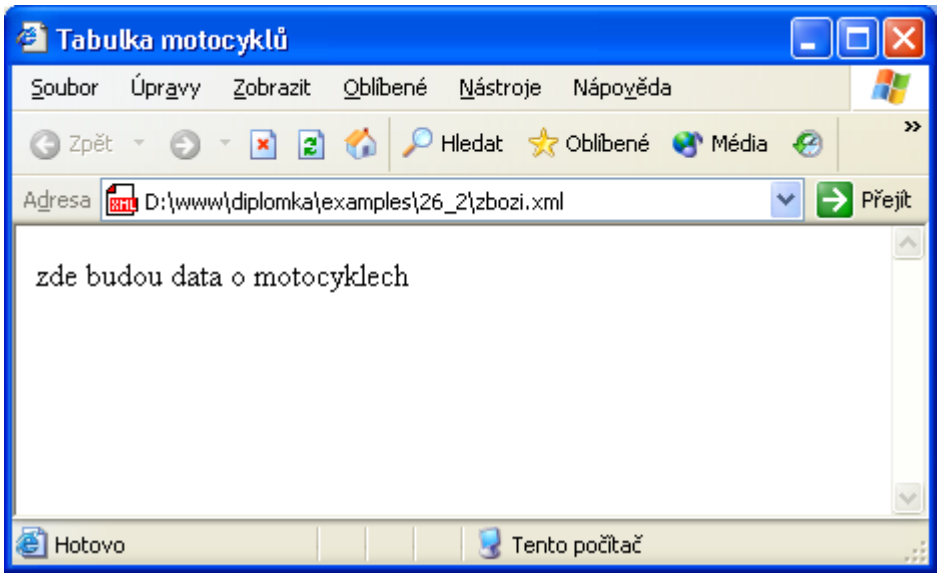
Toto pravidlo se aplikuje pouze na kořenový prvek vstupního stromu. Tato šablona nám říká, že když na vstupu přečteme kořenový prvek, na výstup zapíšeme tag <html>, zpravujeme jeho potomky, a uzavřeme tagem </html>. Toto pravidlo přetíží implicitní pravidlo, které se standardně používá. Na následujícím příkladě je jednoduché pravidlo, které na kořenový prvek aplikujeme:

Příklad 4

```
<?xml version="1.0" encoding="windows-1250"?>
  <xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <title>Tabulka motocyklů</title>
      </head>
      <body>
        <table>
          zde budou data o motocyklech
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Dokud naše šablona bude obsahovat pouze takovéto pravidlo pro kořenový prvek a žádné další pro jeho potomky, bude na výstupu pouze výpis, který je v šabloně uveden, jak je vidět na následujícím výstupu:

```
<html><head><title>Tabulka motocyklů</title></head><body><table>
zde budou data o motocyklech
</table></body></html>
```



5.2 Porovnávání potomků pomocí /

Je důležité si uvědomit, že nejsme omezeni pouze na potomky uzlu, který právě zpracováváme pomocí atributu match. Můžeme použít symbol / k označení specifické hierarchie prvků. Použití samotného / označuje kořenový uzel. My ale můžeme použít tento znak mezi prvky a označit tak, že druhý prvek je potomkem prvního. Například MOTOCYKL/NAZEV označuje prvek NÁZEV, který je potomkem MOTOCYKL.

V prvku xsl:template nám toto umožňuje, abychom zpracovaly pouze daný druh prvků. Například následující pravidlo označí silně prvky OZNACENI, které jsou potomky prvku MOTOCYKL.

```
<xsl:template match="MOTOCYKL/OZNACENI">
  <strong><xsl:value-of select="."/></strong>
</xsl:template>
```

Pozor na to, že toto pravidlo označuje prvky OZNACENI, které jsou potomky prvku MOTOCYKL a ne prvky MOTOCYKL, které mají potomka OZNACENI. Takže potom „.“ v prvku <xsl:value-of select=“.“/> označuje OZNACENI a ne prvek MOTOCYKL.

Můžeme samozřejmě označit prvky ještě hlouběji jednoduše tak, že vzory zřetězíme. Například ZBOZI/MOTOCYKL/NAZEV označuje prvky MOTOCYKL, jejichž rodičem je MOTOCYKL, jehož rodičem je prvek ZBOZI.

Můžeme také použít * pro nahrazení libovolného prvku v hierarchii. Následující příklad vybere všechny prvky OZNACENI, které jsou vnuky prvku ZBOZI.

```
<xsl:template match="ZBOZI/*/OZNACENI">
  <strong><xsl:value-of select="."/></strong>
</xsl:template>
```

Nakonec, jak je vidět na dalším příkladě, znak / sám o sobě vybere kořenový uzel dokumentu. Další pravidlo vybere všechny prvky ZBOZI, které jsou v kořeni dokumentu.

```
<xsl:template match="/ZBOZI">
  <html><xsl:apply-templates/></html>
</xsl:template>
```

Zatímco / odkazuje na kořenový uzel, /* odkazuje na kořenový prvek, jak je vidět na dalším příkladě:

```
<xsl:template match="/*">
  <html>
  <head>
    <title>Tabulka motocyklů</title>
  </head>
  <body>
    <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>
```

5.3 Porovnávání pomocí //

Někdy, hlavně když máme nerovnoměrnou hierarchii uzlů, může být jednodušší překlenout mezilehlé uzly a jednoduše označit všechny prvky daného typu, ať už jsou bezprostředními následovníky, vnuky, pravnuky nebo jiné. Dvojité lomítko // označuje následující uzel v libovolné úrovni. Následující pravidlo aplikujeme na všechny prvky NAZEV následující po ZBOZI, přičemž nám nezáleží, jak hluboko pod ním leží:

```
<xsl:template match="ZBOZI//NAZEV">
  <i><xsl:value-of select="."/></i>
</xsl:template>
```

Náš příklad s tabulkou motocyklů je mělký – má malou strukturu rodičů a potomků. Tento trik nám ale přijde vhod v hlubokých hierarchiích, hlavně když uzly obsahují jiné uzly stejného typu (v našem případě, kdyby MOTOCYKL obsahoval další prvek MOTOCYKL).

Operátor // na začátku porovnávací šablony vybere jakéhokoliv potomka kořenového uzlu. Na následujícím příkladu nám šablona zpracuje úplně všechny uzly CISLO a nebere ohled na jejich umístění:

```
<xsl:template match="//CISLO">
  <i><xsl:value-of select="."/></i>
</xsl:template>
```

5.4 Porovnávání jmen prvků

Jak jsem již zmínil, nejzákladnější porovnávací vzory obsahují jméno prvku, které vyhovuje všem prvkům s tímto jménem. Na následujícím příkladu zobrazíme prvek CISLO tučně:

```
<xsl:template match="MOTOCYKL">
  <b><xsl:value-of select="CISLO"/></b>
</xsl:template>
```

Na následujícím příkladě máme šablonu, která rozšiřuje šablonu z předchozího příkladu – xsl:apply-templates vložíme do šablony odpovídající kořenovému uzlu. Toto pravidlo využívá atribut select k zajištění, aby byly zpracovány pouze prvky ZBOZI.

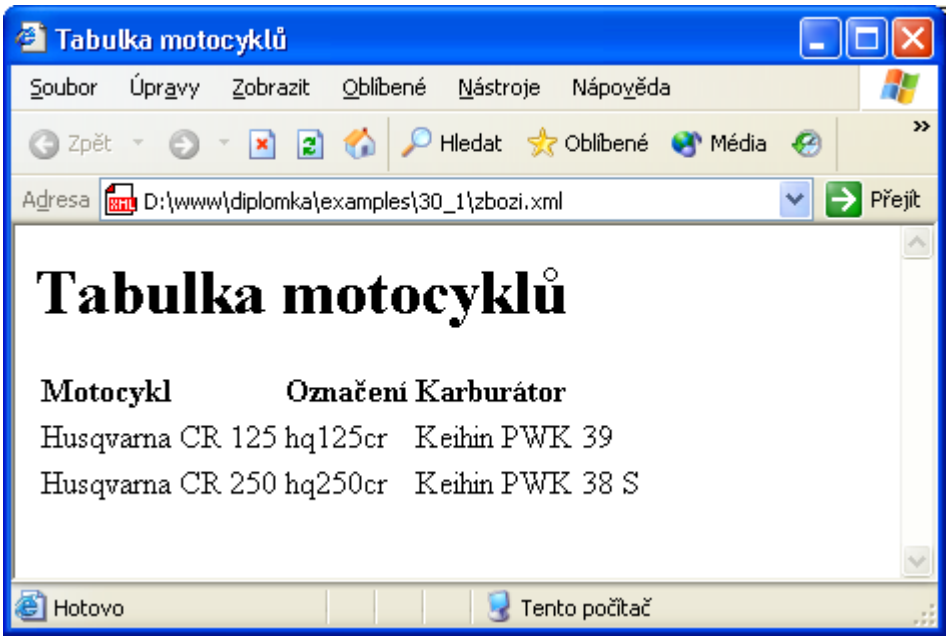
Dále máme v příkladu pravidlo, které je aplikováno pouze na prvek ZBOZI. Toto pravidlo je tvořené výrazem match=“ZBOZI“. Toto pravidlo nám nastaví hlavičku tabulky, a potom volá šablony prvku MOTOCYKL.

Nakonec nám další pravidlo vybere z prvku MOTOCYKL jeho potomky NAZEV, OZNACENI a KARBURATOR pomocí <xsl:apply-templates select=“NAZEV“/>, <xsl:apply-templates select=“OZNACENI“/>, a <xsl:apply-templates select=“ KARBURATOR“/>. Tyto výpisy jsou „obaleny“ HTML tagy tr a td tak, aby poslední řádek tabulky končil karburátorem motocyklu. Za výpisem z XSL souboru je vidět výstup v internet exploreru.

Je ještě jedna věc, kterou bych měl zmínit o této šabloně: pořadí prvků NAZEV, OZNACENI a KARBURATOR ve zdrojovém XML dokumentu není vůbec podstatné. Výstup závisí na pořadí, v jakém byly prvky vybrány, v našem případě: nejdříve název, potom označení. Naproti tomu jsou jednotlivé motocykly uspořádány tak, jak jsou ve vstupním dokumentu. Později si ukážeme, jak můžeme výstupní pořadí prvků ovlivnit pomocí prvku xsl:sort třeba tak, aby byly řazeny podle jejich čísla.

Příklad 5

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
    <html>
      <head>
        <title>Tabulka motocyklů</title>
      </head>
      <body>
        <xsl:apply-templates select="ZBOZI"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="ZBOZI">
    <h1>Tabulka motocyklů</h1>
    <table>
      <tr>
        <td><b>Motocykl</b></td>
        <td><b>Označení</b></td>
        <td><b>Karburátor</b></td>
      </tr>
    <xsl:apply-templates select="MOTOCYKL"/>
  </table>
  </xsl:template>
  <xsl:template match="MOTOCYKL">
    <tr>
   <td><xsl:value-of select="NAZEV"/></td>
   <td><xsl:value-of select="OZNACENI"/></td>
   <td><xsl:value-of select="KARBURATOR"/></td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```



5.5 Porovnávání podle ID

Někdy můžeme chtít aplikovat zvláštní styl na jednotlivý prvek bez toho, abychom měnily všechny další prvky stejného typu. Nejednoduší cestou, jak toho pomocí XSL dosáhnout, je použít styl na prvek určitého ID atributu. Provedeme to pomocí selektoru id(), který obsahuje ID hodnotu v jednoduchých uvozovkách. Na příkladu vypíšeme tučně prvek s ID e47:

```
<xsl:template match="id('e47')">
  <b><xsl:value-of select="."/></b>
</xsl:template>
```

To samozřejmě předpokládá, že prvek, který chceme vybrat pro náš styl, má deklarovaný atribut ve zdrojovém DTD dokumentu, což ale většinou není náš případ.

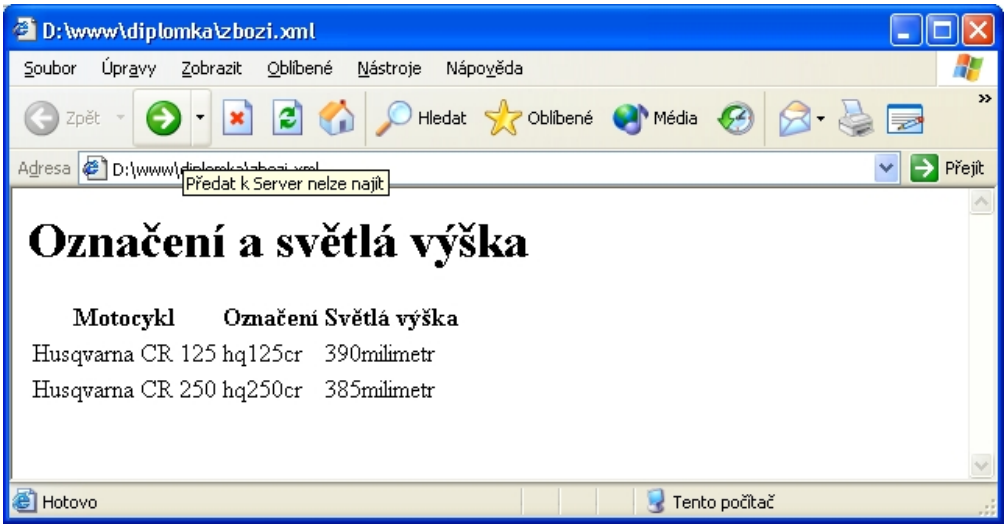
5.6 Porovnávání pomocí @

Znak @ se používá pro vybrání uzlu s daným jménem atributu. Jednoduše uvedeme @ jako prefix atributu, který chceme vybrat. Další výpis ukazuje styl, který vypíše tabulku označení motocyklů a jejich světlou výšku. Nevypisuje pouze hodnotu prvku SVETLA_VYSKA, ale také hodnotu atributu UNIT, čehož docílíme zápisem <xsl:value-of select="@UNITS"/>.

Příklad 5

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/ZBOZI">
  <html>
    <body>
    <h1>Označení a světlá výška</h1>
    <table>
      <th>Motocykl</th>
      <th>Označení</th>
      <th>Světlá výška</th>
      <xsl:apply-templates/>
    </table>
    </body>
  </html>
  </xsl:template>
  <xsl:template match="MOTOCYKL">
    <tr>
    <td><xsl:value-of select="NAZEV"/></td>
    <td><xsl:value-of select="OZNACENI"/></td>
    <td><xsl:apply-templates select="SVETLA_VYSKA"/></td>
  </tr>
  </xsl:template>
```

```
  <xsl:template match="SVETLA_VYSKA">
  <xsl:value-of select="."/>
  <xsl:value-of select="@UNITS"/>
  </xsl:template>
</xsl:stylesheet>
```



Hodnota atributu je jednoduše řetězcová hodnota atributu. Když aplikujeme styl jako v předchozím případě, prvek MOTOCYKL může být formátován například takto:

```
<tr><td>Husqvarna CR 125</td>  <td>hq125cr</td><td>390milimetr</td>
</tr>
<tr><td>Husqvarna CR250</td>   <td>hq250cr</td><td>385milimetr</td>
</tr>
```

Můžeme kombinovat atributy s prvky s využitím různých operátorů hierarchie. Například VYSKA_SEDLA/@UNITS odkazuje na atribut UNITS prvku VYSKA_SEDLA. MOTOCYKL/*/@UNITS odpovídá atributu UNITS potomků prvku MOTOCYKL. To je obzvláště užitečné, když v šabloně porovnáváme atributy. Musíte ale pamatovat že to, co je porovnáváno je atribut uzlu, a ne jeho obsah. Je to velmi častá chyba splést si atribut uzlu s jeho obsahem. Zkuste například vyřešit následující pravidlo, které se pokusí aplikovat pravidlo na všechny potomky, které mají atribut UNITS:

```
<xsl:template match="MOTOCYKL">
  <xsl:apply-templates select="@UNITS"/>
</xsl:template>
```

To co se ve skutečnosti stane je to, že šablona se aplikuje na neexistující atribut UNITS všech prvků MOTOCYKL.

Můžeme také použít * k vybrání všech atributů daného prvku, například VYSKA_SEDLA/@* vybere všechny atributy prvku VYSKA_SEDLA.

5.7 Porovnávání komentářů pomocí comment()

Většinou bychom měli komentáře v XML dokumentech ignorovat. Není dobré začleňovat komentáře do našeho dokumentu. Pokud byste opravdu chtěli nebo potřebovali komentáře do výsledného dokumentu vložit, můžete to pomocí XSL skutečně udělat.

K vybrání komentáře slouží vzor comment(). Ačkoli tento vzor obsahuje závorky, jako by to byla funkce, nikdy mu nepředáváme žádné parametry. Nemůžeme jednoduše rozhodovat mezi různými komentáři. V našem příkladě měl komentář prvek VAHA:

```
<VAHA UNITS="kilogram"><!— při prázdné nádrži —>
6.51
</VAHA>
```

Následující šablona vypíše nejen hodnotu hmotnosti a jednotku, ale také podmínku, při které tato hmotnost platí:

```
<xsl:template match="VAHA">
  <xsl:value-of select="."/>
  <xsl:value-of select="@UNITS"/>
  <xsl:apply-templates select="comment()"/>
</xsl:template>

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

Jediný důvod pro používání komentáře místo atributu nebo prvku, je znázorněn na dalším příkladu. V praxi bychom nikdy neměli vkládat důležité informace jako komentáře. Jediný pravý důvod, proč nám XSL dovoluje označovat komentáře je ten, abychom mohli existující komentáře ve vstupním souboru jednoduše zkopírovat do dokumentu výstupního. Jakékoliv jiné použití ukazuje na špatně navržený vstupní dokument. Následující příklad vybere všechny komentáře ze vstupního souboru a zkopíruje je do výstupu:

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

Pamatujme, že implicitní pravidlo používané pomocí apply-templates nezahrnuje komentáře. Takže když chceme toto pravidlo aktivovat, musíme zahrnout všude, kde se komentáře vyskytují, prvek xsl:apply-templates.

Můžeme využít operátor hierarchie k označení příslušných komentářů. Na následujícím příkladu použijeme pravidlo pro označení komentářů na všech prvcích VAHA:

```
<xsl:template match="VAHA/comment()">
  <xsl:comment><xsl:value-of select="."/></xsl:comment>
</xsl:template>
```

5.8 Porovnávání textových uzlů pomocí text()

Textové uzly jsou implicitně jako uzly ignorovány, a jejich hodnota je zahrnuta jako část hodnoty daného prvku. Proto nám operátor text() neumožňuje přesně označit textového potomka prvku. Navzdory závorkám tento operátor neakceptuje žádné argumenty.

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

Hlavní důvod existence tohoto operátoru je použití pro implicitní pravidla. XSL procesor musí obstarat následující implicitní pravidlo, ať už ho autor specifikuje, nebo ne:

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

Znamená to, že kdykoliv je šablona aplikována na textový uzel, pošle se jeho text na výstup. Pokud nám implicitní pravidlo nevyhovuje, můžeme ho přetížit. Například zahrnutím následující prázdné šablony může zabránit tomu, aby byly textové uzly, které nevyhovují jiné specifikované šabloně zapsány na výstup,

```
<xsl:template match="text()">
</xsl:template>
```

což bychom mohli také zkráceně zapsat jako

```
<xsl:template match="text()"/>
```

6. Operátory

6.1 Používání Or operátoru |

Svislá čára (|) dovoluje šabloně, aby vyhovovala více vzorům. Pokud uzel vyhovuje jedné, nebo druhé šabloně, je šablona aktivována. Například toto pravidlo platí jak na prvky CISLO, tak na prvky KARBURATOR:

```
<xsl:template match="CISLO|KARBURATOR">
  <B><xsl:apply-templates/></B>
</xsl:template>
```

Mezi názvy prvky a | můžeme samozřejmě udělat mezery, což umožňuje lepší čtení:

```
<xsl:template match="CISLO | KARBURATOR">
  <B><xsl:apply-templates/></B>
</xsl:template>
```

Můžeme také použít více než dva porovnávací vzory. Například tato šablona se aplikuje jak na prvky CISLO a KARBURATOR, tak na prvek OZNACENI:

```
<xsl:template match="CISLO | KARBURATOR | OZNACENI">
  <B><xsl:apply-templates/></B>
</xsl:template>
```

Musíme si dát pozor na to, že operátor / je zpracován dříve, než operátor |. To znamená, že následující příklad vybere prvek CISLO, který je potomkem prvku MOTOCYKL, nebo KARBURATOR bez specifikovaného předka, a ne prvky CISLO a KARBURATOR, které jsou potomky uzlu MOTOCYKL:

```
<xsl:template match="MOTOCYKL/CISLO | KARBURATOR">
  <B><xsl:apply-templates/></B>
</xsl:template>
```

6.2 Testování pomocí [ ]

Nyní již umíme otestovat všechny možné uzly. Můžeme ale uzly testovat ještě detailněji a k tomu použijeme operátor []. Využít ho můžeme k různým testům zahrnujícím:

- Jestli uzel obsahuje daného potomka, atribut, nebo jiný uzel
- Jestli je hodnota atributu určitý řetězec

- Jestli hodnota prvku odpovídá určitému řetězci
- Jakou polohu má uzel v hierarchii

Při získávání informací o motocyklech od různých výrobců dostáváme různé množství informací. Tak třeba o motocyklech KTM můžeme získat mnoho informací jak z internetu, tak z různých informačních a propagačních letáků. Proto je u těchto motocyklů jednoduché všechny parametry vyplnit. Pak ale existují výrobci, od kterých se jen těžko jednotlivé technické parametry dozvíme, a některé nám zůstanou utajeny úplně. Proto náš výsledný dokument jistě bude obsahovat motocykly, u kterých nebudou vyplněny, oproti jiným značkám, vyplněny všechny požadované údaje. Pokud například u některých motocyklů nebudeme znát světlou výšku, a budeme vytvářet tabulku, ve které motocykly podle této hodnoty seřadíme, měli bychom tyto motocykly z výsledku vyloučit. Musíme proto vybrat pouze ty prvky MOTOCYKL, které mají potomka SVETLA_VYSKA:

```
<xsl:template match="MOTOCYKL[SVETLA_VYSKA]">
  <tr>
    <td><xsl:value-of select="NAZEV"/></td>
    <td><xsl:value-of select="SVETLA_VYSKA"/></td>
  </tr>
</xsl:template>
```

Poznamenejme, že tato šablona se aplikuje na prvky MOTOCYKL, a ne na prvky SVETLA_VYSKA, jako v případě MOTOCYKL/SVETLA_VYSKA.

Test hranatých závorek může obsahovat více, než jednoduchý následující prvek. Ve skutečnosti mohou obsahovat jakýkoli výraz jazyka XPath. Jestliže daný prvek má potomka, který výběru vyhovuje, je zahrnut do celkového výsledku. Například tato šablona vybere prvky, které mají potomka NAZEV, nebo OZNACENI.

```
<xsl:template match="MOTOCYKL[NAZEV | OZNACENI]">
</xsl:template>
```

Tato šablona vybere prvky MOTOCYKL s potomkem VAHA, který má atribut UNITS:

```
<xsl:template match="MOTOCYKL[VAHA/@UNITS]">
</xsl:template>
```

Nyní, pokud budeme chtít vybrat všechny potomky prvku MOTOCYKL mající atribut UNITS, užijeme * pro vyhledání všech prvků a [@UNITS] k výběru těch prvků, které daný atribut mají:

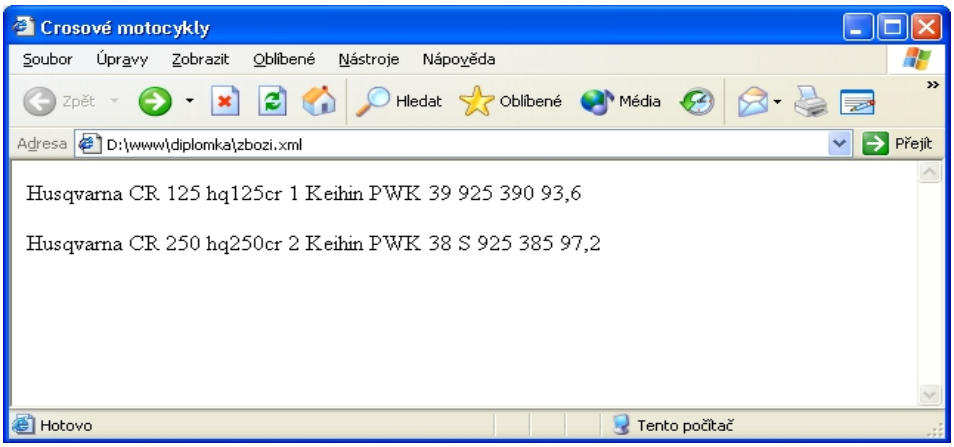
```
<xsl:template match="MOTOCYKL">
  <b><xsl:apply-templates select="*[@UNITS]"/></b>
</xsl:template>
```

Testování prvků pomocí jejich obsahu může být velmi složité, kvůli potřebě získat prvky přesně, včetně bílých mezer. Může být pro nás jednodušší testovat atributy, protože ty většinou žádné bílé mezery neobsahují. Například následující výpis ukazuje výběr pouze těch prvků MOTOCYKL, jejichž hodnota atributu DRUH je cross.

Příklad 6

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="ZBOZI">
    <html>
    <head><title>Krosové motocykly</title></head>
      <body>
      <xsl:apply-templates select="MOTOCYKL[@DRUH='cross']"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="MOTOCYKL">
    <P><xsl:value-of select="."/></P>
  </xsl:template>
</xsl:stylesheet>
```

Zde vidíme výsledek:



7. Prvky

Často je nezbytné nechat rozhodnutí o tom, jaký tag na výstupu použit až na to, jaký vstupní dokument načteme. Například můžeme chtít měnit obsah prvku JMENOSOUBORU v atribut HREF prvku A, nebo nahradit jeden typ prvku na vstupu mnoha dalšími typy prvku na výstupu v závislosti na hodnotě jejich atributu. Toho můžeme docílit pomocí prvků xsl:element, xsl:attribute, xsl:pi, xsl:comment a xsl:text. XSL instrukce jsou použity v obsahu těchto prvků a hodnoty atributů jsou použity pro vyhovění těm prvkům a hodnotám atributům, které jsou použity v hodnotě atributů prvků na výstupu.

7.1 Používání hodnot atributů

Šablony hodnoty atributů kopírují data z obsahu prvku na vstupu, do hodnoty atributu prvku v šabloně. Odtud může být zapsán na výstup. Představme si například, že chceme zkonvertovat naši tabulku motocyklů do prázdného prvku MOTOCYKL, pouze s parametry v následujícím tvaru:

```
<MOTOCYKL NAZEV="Ktm 125 sx"
KARBURATOR="Keihin PWK 39"
CISLO="15"
VYSKA_SEDLA="925mm"
SVETLA_VYSKA="386mm"
OZNACENI="ktm125sx"
VAHA="93,6 kilogram"
/>
```

Abychom dostali takovýto výstup, je nutné vytáhnout obsah prvků ze vstupního dokumentu a vložit jejich hodnotu do hodnoty atributu ve výstupním dokumentu:

```
<xsl:template match="MOTOCYKL">
  <MOTOCYKL NAZEV="<xsl:value-of select='NAZEV'/>"
  KARBURATOR="<xsl:value-of select='KARBURATOR'/>"
  CISLO="<xsl:value-of select='CISLO'/>"
/>
</xsl:template>
```

Dostaneme ale špatný XML dokument. Nemůžeme totiž použít znak < uvnitř hodnoty atributu. Mimoto je velmi těžké napsat program, který by takovýto zápis uměl zpracovat za každé situace.

Místo toho můžeme data uzavřít do složených závorek {}. Správný zápis tedy vypadá takto:

```
<xsl:template match="MOTOCYKL">
  <MOTOCYKL NAZEV="{NAZEV}/>"
  KARBURATOR="{KARBURATOR}/>"
  CISLO="{CISLO}/>"
  />
</xsl:template>
```

Výraz {NAZEV} je ve výstupu nahrazen hodnotou prvku NAZEV právě zpracovávaného uzlu. {KARBURATOR} je nahrazeno hodnotou prvku KARBURATOR a {CISLO} je nahrazeno hodnotou prvku CISLO atd.

Vzorová hodnota atributu může být komplikovanější, než pouze jméno prvku. Ve skutečnosti můžeme použít jakýkoliv řetězec, který byl zmíněn dříve. Například na následujícím výpisu vybereme prvky VAHA:

```
<xsl:template match="VAHA">
  <PRESNA_JEDNOTKA
    NAZEV="VAHA"
    MOTOCYKL="{../NAZEV}"
    VALUE="{.}"
    UNITS="{@UNITS}"
  />
</xsl:template>
```

Tato šablona nám převede prvek VAHA do prvku PRESNA_JEDNOTKA tak, jak je vidět na následujícím výpisu:

```
<PRESNA_JEDNOTKA NAZEV="VAHA" MOTOCYKL="Husqvarna CR 250"
VALUE="97,2" UNITS="kilogram"/>
```

Hodnoty atributů nejsou omezeny pouze na jednoduchou vzorovou šablonu. Můžeme kombinovat hodnotu atributu s přímo zadaným řetězcem, nebo s hodnotou jiného atributu. Například tato šablona vybere prvek MOTOCYKL a nahradí jej jeho jménem zformátovaným jako odkaz na soubor hq125cr.html, hq250cr.html, atd. Hodnota jména souboru je odvozena od hodnoty {OZNACENI}.

```
<xsl:template match="MOTOCYKL">
  <A HREF="{OZNACENI}.html">
    <xsl:value-of select="NAZEV"/>
  </A>
</xsl:template>
```

Do výstupu můžeme také zahrnout více než jeden atribut. Například toto pravidlo zahrne jednotku hmotnosti jako část atributu VALUE, místo toho, aby to byl oddělený atribut:

```
<xsl:template match="VAHA">
  <PRESNA_JEDNOTKA
    NAZEV="VAHA"
    MOTOCYKL="{../NAZEV}"
    VALUE="{.}{@UNITS}"
  /></xsl:template>
```

7.2 Vkládání prvků do výstupu pomocí xsl:element

Prvky obvykle do výstupního dokumentů vkládáme jednoduše tak, že zapíšeme do šablony jeho hodnotu. Například pro vložení prvku P jednoduše napíšeme <P> a </P> do šablony. Někdy ale můžeme potřebovat použít nějaké detaily ze vstupního souboru k rozhodnutí, který prvek do výstupu zahrnou a který ne. To se může stát například když děláme transformaci ze vstupního dokumentu, který používá atributy pro výstupní dokument, který používá různé prvky pro stejné informace.

Prvek xsl:element vloží prvek do výstupního dokumentu. Jméno prvku je předáno jako atribut name. Obsah prvku je odvozený od obsahu xsl:element, který může obsahovat instrukce xsl:attribute, nebo xsl:comment.

Předpokládejme například ,že chceme nahradit prvek MOTOCYKL prvky cross, enduro a street, v závislosti na hodnotě atributu DRUH. Použitím xsl:element nám jednoduché pravidlo konvertuje hodnotu atributu DRUH na jméno prvku:

```
<xsl:template match="MOTOCYKL">
  <xsl:element name="{@DRUH}">
    <NAZEV><xsl:value-of select="NAZEV"/></NAZEV>
    <!-- pravidla pro další potomky -->
  </xsl:element>
</xsl:template>
```

7.3 Vkládání atributu do výstupu pomocí xsl:attribute

Atributy do výstupního dokumentu můžeme jednoduše zahrnout tak, že použijeme v šabloně jejich řetězcové vyjádření. Například pro vložení prvku DIV s atributem ALIGN s hodnotou CENTER, můžeme jednoduše napsat <DIV ALIGN=“CENTER“></DIV>. Většinou ale hodnota atributu závisí na hodnotě uvedené ve vstupním souboru.

Předpokládejme například, že chceme mít šablonu, která nám vybere jména prvků a zformátuje je jako odkazy na soubory nazvané hq125cr.html, hq250cr.html, atd...:

```
<LI><A HREF="hq125cr.html">Husqvarna CR 125</A></LI>
<LI><A HREF="hq250cr.html">Husqvarna CR 250</A></LI>
<LI><A HREF="hq410te.html">Husqvarna TE 410</A></LI>
```

Každý prvek má jinou hodnotu atributu HREF. Prvek xsl:attribute nastaví jméno atributu a jeho hodnotu a vloží jej do výstupního dokumentu. Jméno atributu je dané atributem name pomocí xsl:attribute. Hodnota atributu je nastavená pomocí obsahu prvku xsl:attribute name:

```
<xsl:template match="MOTOCYKL">
  <LI><A>
    <xsl:attribute name="HREF">
      <xsl:value-of select="OZNACENI"/>.html
    </xsl:attribute>
    <xsl:value-of select="NAZEV"/>
  </A></LI>
</xsl:template>
```

Všechny prvky xsl:attribute musí být definovány dříve, než jakýkoliv jiný obsah rodičovského prvku. Nemůžeme přidat atribut do prvku, který jsme již začali vypisovat na výstup. Následující šablona je špatně:

```
<xsl:template match="MOTOCYKL">
  <LI><A>
    <xsl:value-of select="NAZEV"/>
    <xsl:attribute name="HREF">
      <xsl:value-of select="OZNACENI"/>.html
    </xsl:attribute>
  </A></LI>
</xsl:template>
```

7.3.1 Definování množin atributů

Někdy se nám může hodit aplikovat stejnou skupinu atributů na mnoho různých prvků, buď ve stejné, nebo v jiné třídě. Například můžeme chtít aplikovat atribut style na každou buňku v tabulce. Abychom si práci zjednodušili, můžeme definovat jeden nebo více atributů jako členy skupiny atributů pomocí xsl:attribute-set, a potom zahrnout tuto množinu pomocí xsl:use.

Například tento prvek xsl:attribute-set definuje prvek nazvaný stylbunky pomocí atributu font-family, nastavený na New York, Times New Roman, Time, serif a velikost písma na 12 bodů.

```
<xsl:attribute-set name="stylbunky">
  <xsl:attribute name="font-family">
    New York,Times New Roman,Times,serif
  </xsl:attribute>
  <xsl:attribute name="align">center</xsl:attribute>
</xsl:attribute-set>
```

Tato šablona použije předchozí skupinu atributů na prvky td. Stejně jako pomocí xsl:attribute můžeme vložit skupinu atributů pomocí xsl:use:

```
<xsl:template match="MOTOCYKL">
  <tr>
    <td>
      <p xsl:use-attribute-sets="stylbunky">
      <xsl:value-of select="NAZEV"/>
      </p>
    </td>
    <td>
      <p xsl:use-attribute-sets="stylbunky">
      <xsl:value-of select="CISLO"/>
      </p>
    </td>
  </tr>
</xsl:template>
```

Pokud prvek použije více než jeden atribut z množiny, jsou na prvek použity všechny atributy z dané množiny. Pokud více než jedna množina atributů definuje stejný atribut s rozdílnou hodnotou, je vybrána ta s větší důležitostí. Šablona, ve které jsou množiny se stejnou důležitostí, nám zahlásí chybu.

7.4 Vytváření komentářů pomocí xsl:comment

Tento prvek nám do výstupu vloží komentář. Nemá žádné atributy. Jeho obsahem je text komentáře:

```
<xsl:template match="MOTOCYKL">
  <xsl:comment>Toto je pouze komentář místo informací o
  motocyklu</xsl:comment>
</xsl:template>
```

Následující šablona tedy nahradí prvky MOTOCYKL následujícím výstupem:

```
<!—Toto je komentář místo informací o motocyklu.—>
```

Obsah prvku xsl:comment může obsahovat prvky xsl:value-of, nebo xsl:apply-templates. Neměl by obsahovat prvky xsl:element a ostatní instrukce, které prvek na výstupu generují. Také by neměl obsahovat žádné instrukce ani řetězce, které obsahují dvojité uvozovky. Způsobilo by to špatně strukturovaný dokument na výstupu, což je zakázáno.

7.5 Vytváření textu pomocí xsl:text

Tento prvek vloží svůj obsah do výstupního dokumentu jako obyčejný text. Následující šablona nahradí každý prvek MOTOCYKL řetězcem „Toto je obyčejný text.“

```
<xsl:template match="MOTOCYKL">
  <xsl:text>Toto je obyčejný text.</xsl:text>
</xsl:template>
```

Tento prvek se moc nepoužívá, protože je jednoduší text zahrnout do šablony přímo. Má ale jednu svoji výhodu, kvůli které bychom ho mohli využít. Je užitečný pokud potřebujeme na výstup vytisknout třeba báseň, zdrojový kód programu nebo jiný text, ve kterém mají význam bílé mezery.

7.6 Definování proměnných pomocí xsl:variable

Pojmenované konstanty nám mohou vyčistit a usnadnit čtení kódu. Mohou nahradit standardní text jednoduchými jmény. Použití konstant je také výhodné, jestliže chceme změnit její hodnotu. Pokud máme konstantu pojmenovanou a odkazujeme se na ni pouze jejím jménem, stačí její hodnotu změnit v deklaraci a změna se promítne do všech míst, kde ji používáme.

Prvek xsl:variable definuje jméno pro konstantu. Je to prázdný prvek, který musí být přímým potomkem xsl:stylesheet. Má jednoduchý atribut name, který nám udává jméno konstanty, kterým se na ní budeme odkazovat. Obsahem prvku je text, kterým je vždy jméno konstanty nahrazeno. Tak například konstanta copy03 s hodnotou Copyright 2003:

```
<xsl:variable name="copy03">
  Copyright 2003
</xsl:variable>
```

Pro přístup k hodnotě proměnné použijeme jako prefix znak dolaru $. Pro vložení konstanty například do hodnoty atributu provedeme takto:

```
<BLOCK COPYRIGHT="{$copy03}">
</BLOCK >
```

Hodnotu atributu můžeme použít také v prvku xsl:value-of :

```
<xsl:value-of select="$copy03"/>
```

Obsah xsl:variable může obsahovat další XSL instrukce. To znamená, že hodnota konstanty může záviset na hodnotě nějaké informace obsažené v dokumentu. Konstanta ale nemůže obsahovat odkaz sama na sebe, protože by se volání rekurzivně zacyklilo. Následující příklad by zahlásil chybu:

```
<xsl:variable name="copy">
  <xsl:value-of select="$copy"/> Ondra Svoboda
</xsl:variable>
```

Podobně na sebe nemůžou odkazovat dvě proměnné navzájem:

```
<xsl:variable name="Clovek1">
  Člověk_1 má rád <xsl:value-of select="$Clovek2"/>
</xsl:variable>
<xsl:variable name="Clovek2">
  Člověk_2 má rád <xsl:value-of select="$Clovek1"/>
</xsl:variable>
```

7.7 Kopírování uzlu pomocí xsl:copy

Prvek xsl:copy kopíruje zdrojový uzel na výstup. Jeho potomci ani atributy nejsou automaticky kopírovány. Může obsahovat i prvek xsl:template, který nám tyto prvky do výstupu také zkopíruje. To je často užitečné, když transformujeme dokument z jednoho označkovaného dokumentu do stejného, nebo jen málo odlišného. Následující pravidlo vypne kopírování potomků a atributů a nahradí je hodnotou jejich obsahu:

```
<xsl:template match="MOTOCYKL">
  <xsl:copy>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

Následující šablona nám vytvoří identický dokument tak, že zkopíruje úplně celý dokument na vstupu do výstupního dokumentu:

```
<xsl:template match="*|@*|comment()|text()">
  <xsl:copy>
    <xsl:apply-templates select="*|@*|comment()|text()"/>
  </xsl:copy>
</xsl:template>
```

Můžeme trochu upravit šablonu, abychom nedostali úplně stejný dokument. Můžeme například chtít, aby se nezkopírovaly žádné komentáře, ale dokument jinak zůstal nedotčený:

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="*|@*|text()">
    <xsl:copy>
    <xsl:apply-templates select="*|@*|text()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

xsl:copy pouze kopíruje zdrojový uzel. Můžeme kopírovat i ostatní uzly pomocí xsl:copy-of. Označením atributu xsl:copy-of vybereme uzel, který má být zkopírován. Pomocí xsl:copy-of se zkopírují i všichni potomci, prvky jmenného prostoru i atributy. Například následující výpis používá xsl:copy-of pro vynechání prvků bez zadané světlé výšky a zkopíruje pouze prvky MOTOCYKL, u kterých je světlá výška zadána.

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/ZBOZI">
  <ZBOZI>
    <xsl:apply-templates select="MOTOCYKL"/>
  </ZBOZI>
  </xsl:template>
  <xsl:template match="MOTOCYKL">
  <xsl:apply-templates select="SVETLA_VYSKA"/>
  </xsl:template>
  <xsl:template match="SVETLA_VYSKA">
  <xsl:copy-of select="..">
  </xsl:copy-of>
  </xsl:template>
  <xsl:template match="*|@*|text()">
  <xsl:copy>
    <xsl:apply-templates select="*|@*|text()"/>
  </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

7.8 Funkce element-available

Funkce element-available se používá pro zjištění, zda je zadaný prvek podporovaný XSLT procesorem. Vrací nám hodnotu typu boolean. Tuto funkci můžeme použít pouze k testování prvků, které se mohou vyskytnout v těle šablony. Jsou to prvky:

- xsl:apply-imports
- xsl:apply-templates

- xsl:attribute
- xsl:call-template
- xsl:choose
- xsl:comment
- xsl:copy
- xsl:copy-of
- xsl:element
- xsl:fallback
- xsl:for-each
- xsl:if
- xsl:message
- xsl:number
- xsl:processing instruction
- xsl:text
- xsl:value-of
- xsl:variable

Funkce akceptuje jeden parametr, kterým je řetězcová hodnota prvku. Zde je příklad:

```
<xsl:template match="/">
  <xsl:choose>
    <xsl:when test="element-available('xsl:comment')">
      <p>xsl:comment is supported.</p>
    </xsl:when>
    <xsl:otherwise>
      <p>xsl:comment is not supported.</p>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:choose>
    <xsl:when test="element-available('xsl:delete')">
      <p>xsl:delete is supported.</p>
    </xsl:when>
    <xsl:otherwise>
      <p>xsl:delete is not supported.</p>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

A zde máme výsledek:

```
xsl:comment is supported.
xsl:delete is not supported.
```

7.9. Řazení prvků

7.9.1 Prvek xsl:sort

Prvek xsl:sort seřadí výstupní prvky v jiném pořadí, než v jakém jsou na vstupu. Prvek xsl:sort se používá jako potomek prvku xsl:apply-templates nebo xsl:for-each. Atribut select prvku xsl:sort definuje klíč, podle kterého budou prvky řazeny.

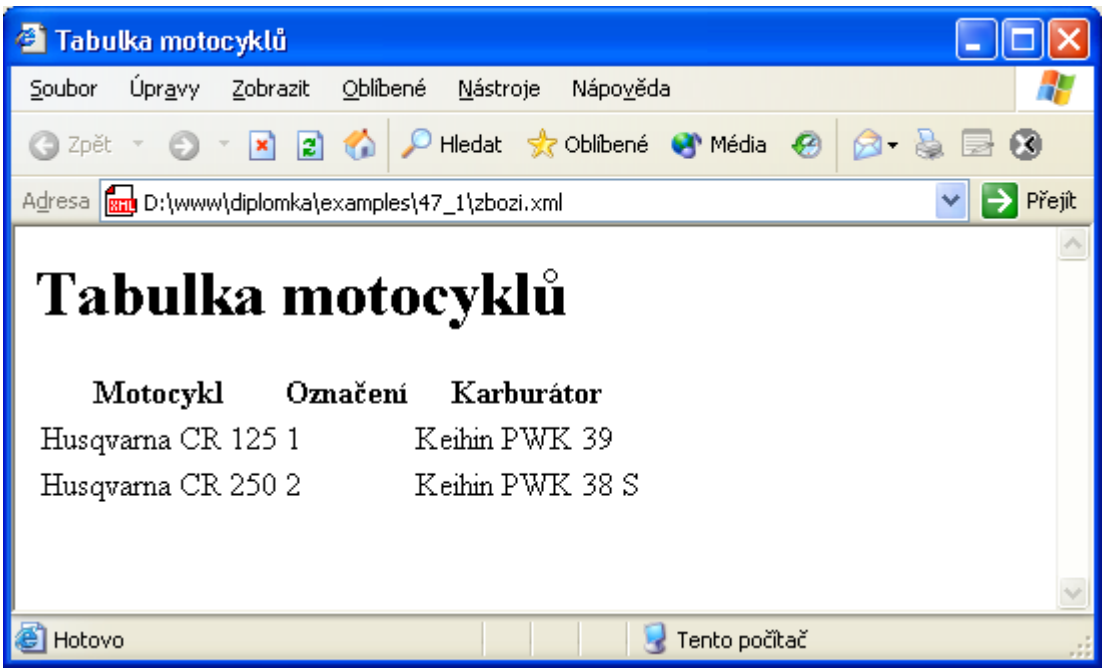
Obvykle je řazení prováděno v abecedním pořadí klíčů. Jestliže zadáme více než jeden prvek xsl:sort v prvcích xsl:apply-templates nebo xsl:for-each, jsou výstupní prvky řazeny nejdříve podle prvního a potom podle druhého prvku. Jestliže jsou na vstupu dva prvky totožné, zůstane jejich pořadí stejné, jako ve vstupním dokumentu.

Představme si například, že máme náš soubor s MOTOCYKLy. Pro seřazení podle jejich zadaných čísel můžeme použít následující šablonu:

Příklad 8

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="ZBOZI">
  <html>
    <head>
    <title>Tabulka motocyklů</title>
  </head>
  <body>
    <h1>Tabulka motocyklů</h1>
    <table>
    <th>Motocykl</th>
    <th>Označení</th>
    <th>Karburátor</th>
    <xsl:apply-templates>
      <xsl:sort select="CISLO"/>
    </xsl:apply-templates>
      </table>
    </body>
  </html>
  </xsl:template>
 <xsl:template match="MOTOCYKL">
    <tr>
    <td><xsl:apply-templates select="NAZEV"/></td>
    <td><xsl:apply-templates select="CISLO"/></td>
    <td><xsl:apply-templates select="KARBURATOR"/></td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

Na následujícím obrázku máme výsledek. Husqvarna CR 125 s číslem 1 je uveden jako první. Druhým motocyklem by ale nemusel být motocykl Husqvarna CR 250 číslem 2, ale například ktm 125 sx, který by měl číslo 10. Ačkoliv víme, že 10 leží až za 9, abecedně je 10 ještě před 2.



Můžeme proto upravit výstup tak, aby byly prvky řazeny skutečně podle velikosti čísla. Uděláme to tak, že nastavíme hodnotu atributu data-type na number:

```
<xsl:sort data-type="number" select="CISLO"/>
```

Můžeme samozřejmě také změnit pořadí řazení z implicitního rostoucího na klesající nastavením hodnoty atributu order na descending:

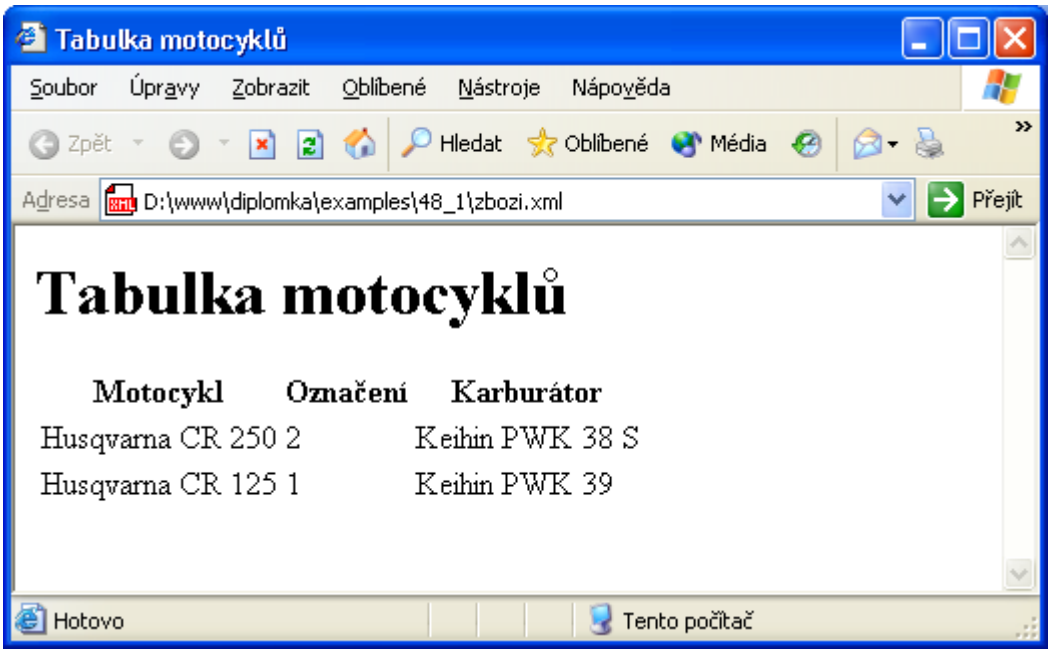
Příklad 9

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="ZBOZI">
  <html>
    <head>
      <title>Tabulka motocyklů</title>
    </head>
    <body>
```

```
      <h1>Tabulka motocyklů</h1>
      <table>
        <th>Motocykl</th>
        <th>Označení</th>
        <th>Karburátor</th>
        <xsl:apply-templates>
          <xsl:sort order="descending" data-type="number"
          select="CISLO"/>
        </xsl:apply-templates>
      </table>
    </body>
  </html>
  </xsl:template>
    <xsl:template match="MOTOCYKL">
    <tr>
      <td><xsl:apply-templates select="NAZEV"/></td>
      <td><xsl:apply-templates select="CISLO"/></td>
      <td><xsl:apply-templates select="KARBURATOR"/></td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

Tato šablona seřadí prvky od největšího k nejmenšímu, takže nyní bude Husqvarna CR 125 s číslem 1 na konci seznamu.

Abecední řazení závisí na abecedě kterou použijeme. Atribut lang může nastavit jazyk pro daný klíč. Můžeme také nastavit atribut case-order na jednu ze dvou hodnot upper-fisrt nebo lower-first. Určíme tím, jestli budeme nejdříve řadit velká, nebo malá písmena.



8. Práce s čísly

8.1 Prvek xsl:number

Prvek xsl:number vloží formátované číslo typu integer do výsledného dokumentu. Hodnota čísla je dána zaokrouhlením čísla, vypočítaného pomocí atributu value na nejbližší číslo integer, přičemž úprava formátu odpovídá hodnotě atributu format. Pro oba atributy jsou nabízeny přiměřené implicitní hodnoty. Uvažujme následující příklad:

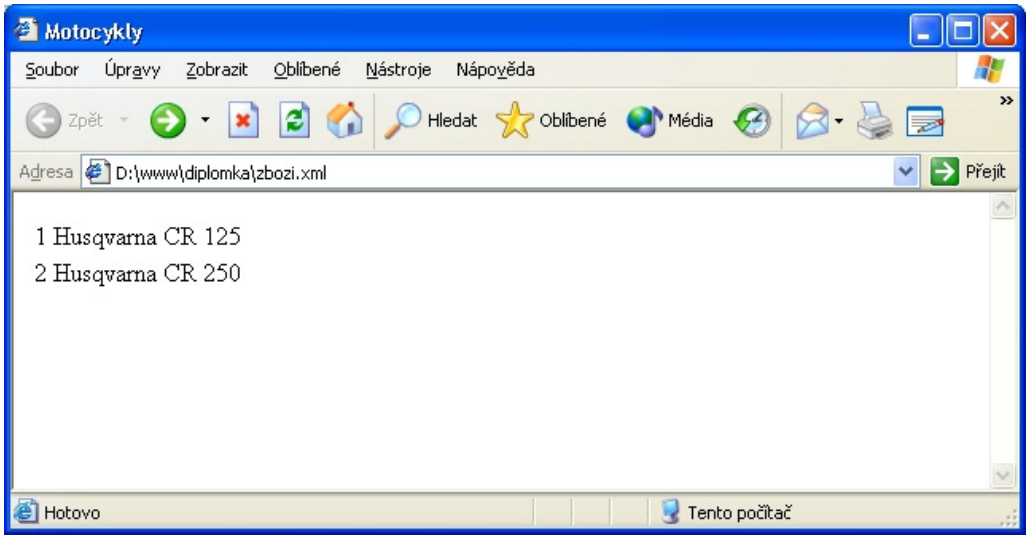
Příklad 7

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="ZBOZI">
  <html>
    <head><title>Motocykly</title></head>
    <body>
    <table>
      <xsl:apply-templates select="MOTOCYKL"/>
    </table>
    </body>
  </html>
  </xsl:template>
  <xsl:template match="MOTOCYKL">
  <tr>
    <td><xsl:number value="position()"/></td>
    <td><xsl:value-of select="NAZEV"/></td>
  </tr>
  </xsl:template>
</xsl:stylesheet>
```

Výstup bude následující:

```
<html><head><title>Motocykly</title></head><body><table><tr><td>
1</td><td>Husqvarna CR 125<
/td></tr>
<tr><td>2</td><td>Husqvarna CR 250</td></tr>
</table></body></html>
```

Husqvarna CR 125 bude mít číslo jedna, protože to je první prvek v seznamu. Husqvarna CR 250 dostane číslo dvě, protože je uvedené jako druhý prvek MOTOCYKL. Budou tak mít stejná čísla jako jsou jejich skutečná zadaná čísla, protože jsou tak uspořádány ve vstupním dokumentu.



Jestliže atribut value vynecháme, je jako číslo použita pozice právě zpracovávaného uzlu ve zdrojovém stromě. Výsledná hodnota může být upravena následujícími atributy:

- level
- count
- from
- format
- letter-value
- grouping-sep
- grouping-size
- lang

8.1.1 Atribut level

Implicitně bez atributu value spočítá prvek xsl:number sourozence zdrojového uzlu. To znamená, že jsou číslovány prvky, které jsou potomky stejného uzlu – jsou ve stejné úrovni.

Pokud nastavíme atribut level prvku xsl:number na na hodnotu any, budou se počítat všechny prvky stejného druhu, jako stejný uzel v dokumentu. Bude nyní zahrnovat nejen jeden prvek, který náleží právě zpracovávanému uzlu, ale všechny prvky stejného typu.

```
<xsl:template match="MOTOCYKL">
  <xsl:apply-templates select="NAZEV"/>
</xsl:template>
```

```
<xsl:template match="NAZEV">
  <td><xsl:number level="any"/></td>
  <td><xsl:value-of select="."/></td>
</xsl:template>
```

Pokud by byla hodnota atributu nastavená na single, dostaneme následující výsledek:

```
<td>1</td><td>Husqvarna CR 125</td>
<td>1</td><td>Husqvarna CR 250</td>
```

Když ale nastavíme hodnotu na any, budou se počítat všechny prvky NAZEV a dostaneme takovýto výsledek:

```
<td>1</td><td>Husqvarna CR 125</td>
<td>2</td><td>Husqvarna CR 250</td>
```

Při nastavení atributu na hodnotu multiple můžeme tvořit víceúrovňové číslování seznamů. Budeme-li například požadovat očíslování prvku NAZEV ve formátu 1.1.1..., nastavíme hodnotu level na multiple a count na * pro počítání všech prvků:

```
<xsl:template match="MOTOCYKL">
  <xsl:apply-templates select="NAZEV"/>
</xsl:template>
<xsl:template match="NAZEV">
  <xsl:number level="multiple" format="1.1 " count="*"/>
  <xsl:value-of select="."/>
</xsl:template>
```

Dostaneme následující výstup:

```
1.1.1 Husqvarna CR 1251.2.1 Husqvarna CR 250
```

8.1.2 Atribut count

Implicitně bez uvedení atributu value jsou počítány pouze prvky stejného typu, jako je právě zpracovávaný prvek. Můžeme ale nastavit atribut count prvku xsl:number a specifikovat tak, co se bude počítat. Například toto pravidlo přiřadí čísla všem potomkům prvku MOTOCYKL:

```
<xsl:template match="MOTOCYKL/*">
  <td><xsl:number count="*"/></td>
  <td><xsl:value-of select="."/></td>
</xsl:template>
```

Jako výstup z této šablony dostaneme toto:

```
<td>1</td><td>Husqvarna CR 250</td>
<td>2</td><td>hq250cr</td>
<td>4</td><td>Keihin PWK 39</td>
<td>5</td><td>925</td>
<td>6</td><td>390</td>
<td>7</td><td>93,6</td>
```

Stejný výpis bude i ostatních motocyklů.

8.1.3 Atribut from

Atribut from obsahuje výraz select, který specifikuje prvek, od kterého má být zpracováván vstupní strom, přičemž počítání začne od jedničky a ne třeba od čísla dvě, nebo deset nebo nějakého jiného čísla. Atribut from jen mění to, který prvek bude považován za první.

Až doteposud jsme implicitně předpokládali čísla ve tvaru 1, 2, 3 atd, což jsou evropské číslice začínající od 1 a každá následující je o jedničku vyšší. To ale není jediná možnost. Můžeme třeba požadovat, aby stránky v úvodu knihy byly číslovány Římskými číslicemi třeba i, ii, iii, iv atd. Jiné země mohou používat úplně jiné skupiny číslic i oddělovače desetinných míst.

Existuje proto pět atributů prvku xsl:number:

8.1.4 Atribut format

Můžeme přizpůsobit formát čísla s využitím atributu format. Tento atribut má zpravidla jednu z následujících hodnot:

- i: vytváří sekvence malé znaky Římské abecedy
- I: vytváří sekvence velké znaky Římské abecedy
- a: vytváří sekvence malých znaků a,b,c,d,e,f,...
- A: vytváří sekvence velkých znaků A,B,C,D,E,F,...

Například toto pravidlo očísluje prvky MOTOCYKL pomocí Římských kapitálek:

```
<xsl:template match="MOTOCYKL">
  <P>
    <xsl:number value="position()" format="I"/>
    <xsl:value-of select="."/>
  </P>
</xsl:template>
```

Můžeme také specifikovat desítkový formát tak, že budeme požadovat dvě číslice. Tak například format="01" bude generovat sekvence 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, …. Toto se nám může hodit při zarovnávání čísel do sloupců. Dejme pozor na to, že číslování pomocí dvojic číslic, ani pomocí malých nebo velkých římských číslic, není podporováno v Netscape Navigatoru ani ve verzi 6.

8.1.5 Atribut letter-value

Tento atribut používáme pro rozlišení číselných sekvencí používající písmena. V mnoha jazycích jsou dvě běžně používané číselné sekvence, které používají písmena. Jedna z nich přiřazuje číselné hodnoty písmenům v abecedním pořadí, a druhá přiřazuje číselnou hodnotu každému písmenu, jak je zvykem v daném jazyce. Jedná se například o sekvence začínající písmenem a a i. Hodnota alphabetic specifikuje abecední pořadí a hodnota traditional určuje jinou sekvenci.

Je možné, že dva různé procesory xslt nepřevedou čísla do stejné sekvence písmen, z důvodu různých jazykových sad.

8.1.6 Atribut grouping separator a grouping-size

Je celkem obvyklé oddělovat řády tisíců mezi každými třemi číslicemi. V Evropě se jako oddělovač používá tečka např: 4.567.123,45. V Americe se používá čárka, protože jako desetinný oddělovač používají tečku např: 4,567,123.45. V některých jiných zemích je ale zvykem oddělovat každé čtyři číslice, namísto u nás obvyklých tří: 4.5671.2345,00. Těmito problémy se pravděpodobně budete zabývat pouze pokud budete zpracovávat dlouhá čísla.

Atribut grouping-size specifikuje skupinový oddělovač mezi skupinami číslic. Atribut grouping-size naproti tomu určuje počet čísel v každé skupině. Obecně můžeme tyto atributy používat v závislosti na druhu jazyka:

```
<xsl:number grouping separator=" " grouping-size="3"/>
```

8.1.7 Atribut lang

Tento atribut nám specifikuje jazyk, který bude použit pro číslování pomocí písmen. Pozor na to, že tento atribut zatím není podporován v Netscape Navigator.

8.2 Funkce format-number()

Funkce format-number() se používá pro konvertování řetězců na čísla. Tato funkce akceptuje několik atributů. První atribut number je povinný a udává hodnotu čísla, které má být konvertováno. Druhý atribut format je taktéž povinný. Pomocí tohoto parametru udáváme vzor šablony:

Tabulka č.2 – Atribut format funkce format-number()

Atribut format	
Hodnota	***Význam***
#	Udává číslice
0	Udává vyžadované nuly
.	Udává pozici desetinné čárky
,	Oddělovač řádů
%	Zobrazí číslo jako procento
;	Oddělovač šablon. První šablona je použita pro kladná, druhá pro záporná čísla.

Vše si ukážeme na následujícím příkladu:

```
<xsl:template match="/">
  <xsl:value-of select='format-number(500100, "#.00")' />
  <br />
  <xsl:value-of select='format-number(500100, "#.0")' />
  <br />
  <xsl:value-of select='format-number(500100, "###,###.00")' />
  <br />
  <xsl:value-of select='format-number(0.23456, "##%")' />
  <br />
  <xsl:value-of select='format-number(500100, "#######")' />
</xsl:template>
```

A zde je výsledek:

```
500100.00
500100.0
500,100.00
23%
500100
```

8.3 Prvek xsl:decimal-format

Pomocí tohoto prvku deklarujeme desítkový formát pro číslo, které je výsledkem funkce format-number(). Pokud uvedeme nepovinný atribut name, vytvoříme pojmenovaný desítkový formát, jinak vytvoříme nepojmenovaný. Chyba nastane, pokud definujeme více těchto formátů se stejným jménem, dokonce i s jinou importovanou důležitostí.

Tento prvek může akceptovat mnoho atributů:

- name - Je to volitelný atribut a specifikuje jméno formátu.
- decimal-separator – Volitelný parametr, kterým předáváme znak, který odděluje skupiny řádů. Implicitní hodnota je čárka „.
- infinity – Tento atribut definuje znak, který se používá pro nekonečno. Je to volitelný atribut.
- minus-sign - Volitelný atribut. Určuje znak, který používáme jako implicitní znak pro záporná čísla. Jeho implicitní hodnota je znak "-" což je "#x2D".
- Nan – Také volitelný atribut. Jeho hodnotou je řetězec používaný pro situace kdy výsledkem není číslo. Jeho implicitní hodnota je NaN.
- percent – Udává znak používaný jako znak pro označení procent. Nepovinný atribut s implicitní hodnotou %.
- per-mille – Definuje znak „každých tisíc" jeho implicitní hodnota je "‰". Také nepovinný atribut.
- zero-digit – Tento atribut specifikuje znak pro hodnotu nula. Je také nepovinný a jeho implicitní hodnota je ("0").
- digit – Definuje znak používaný jako zástupný znak pro nevýznamné nuly. Je nepovinný a jeho implicitní hodnota je #.
- pattern-separator – Tento atribut definuje znak, který v šabloně odděluje kladná a záporná čísla. Jeho implicitní hodnota je ; a je také nepovinný.

8.4 Funkce odvozené od jazyka XPath

8.4.1 Funkce ceiling()

Tato funkce vrací nejmenší hodnotu typu integer – to znamená celé číslo, které není menší než číslo, které jí předáváme v argumentu.

Příklad:

```
ceiling(1,25)
```

Výsledek:

4

8.4.2 Funkce floor()

Vrací největší hodnotu typu integer, která není větší než číslo předávané v parametru.

Příklad:

```
floor(1,25)
```

Výsledek:

1

8.4.3 Funkce number()

Funkci number() použijeme pokud budeme chtít zkonvertovat hodnotu argumentu na číslo.

Příklad:

```
number('100')
```

Výsledek:

100

8.4.4 Funkce round()

Tohle je vlastně funkce zaokrouhlovací. Vrací nám hodnotu typu integer, která je nejblíže číslu předávaném v argumentu.

Příklad:

```
round(1,25)
```

Výsledek:

1

8.4.5 Funkce sum()

Tato funkce vrací hodnotu součtu číselných hodnot prvků předaných v parametru.

Příklad:

```
sum(/zbozi/cena)
```

8.4.6 Operátory

Jazyk XPath vždy zkonvertuje každý operand na číselnou hodnotu před tím, než s ním provede matematickou operaci.

Tabulka č. 3 – Popis operátorů

Popis operátorů			
Operátor	***Popis***	***Příklad***	***Výsledek***
+	sčítání	6+5	11
-	odčítání	6-5	1
*	násobení	6*5	30
div	dělení	6 div 2	3
mod	modulo	6 mod 5	1
=	rovno	6 = 5	false
!=	nerovno	6 != 5	true
<	menší	6 < 5	false
<=	menší rovno	6 <= 5	false

9. Práce s řetězci

9.1 Bílé mezery

Když XSLT procesor zpracovává dokument, aby z něj vytvořil strom uzlů, nejdříve v některých uzlech vymaže bíle mezery – whitespaces. Pokud je uzel textový, nikdy z něj mezery nejsou odstraněny – pouze pokud obsahuje pouze bílé mezery. Při odstraňování textových uzlů odebíráme textový uzel ze stromu. Proces odstraňování přijímá jako vstup množinu jmen prvků, ze kterých musí bílé mezery odstranit. Tento proces je aplikován jak na zdrojový dokument, tak na šablonu, ale množina uzlů, kde se bílé mezery zachovávají, je určována rozdílně.

Z textového uzlu nejsou mezery odstraněny, pokud nastane alespoň jeden z následujících případů:

- Pokud jméno prvku prvku rodiče textového uzlu je v množině uzlů, ze kterých se bílé mezery neodstraňují
- Textový uzel obsahuje alespoň jeden znak, který není znakem mezery. V XML je znak mezery vyjádřen pomocí: #x20, #x9, #xD, #xA
- Předek textového uzlu má atribut xml:space s hodnotou nastavenou na preserve, přičemž žádný bližší předek nemá tento atribut s hodnotou default.

Ve všech ostatních případech je textový uzel upravený. Atributy xml:space z výsledného uzlu nejsou odstraňovány. U šablon jsou mezery zachovávány pouze u prvků xsl:text.

9.1.1 Prvek xsl:strip-space

Pokud z nějakého prvku chceme odstranit mezery, použijeme prvek xsl:strip-space pro identifikování určitých prvků ve vstupním dokumentu. U těchto prvků budou mezery považovány za nevýznamné a budou z dokumentu při posílání na výstup odstraněny. Tento prvek je prvek nejvyšší úrovně, to znamená, že musí být uvedený před všemi šablonami.

Například toto pravidlo můžeme přidat do stylu našeho seznamu motocyklů pro vynechání mezer:

```
<xsl:strip-space element=""/>
<xsl:strip-space element="VYSKA_SEDLA"/>
<xsl:strip-space element="SVETLA_VYSKA"/>
```

tento zápis můžeme přepsat také takto:

```
<xsl:strip-space element="SVETLA_VYSKA VYSKA_SEDLA VAHA"/>
```

9.1.2 Prvek xsl:preserve-space

Prvek xsl:preserve-space je protikladem prvku xsl:strip-space. Tento prvek má také atribut element, který udává jméno prvku, na který máme prvek aplikovat. Tento prvek je stejně jako xsl:strip-space top-element a musí být uvedený před všemi šablonami. Pomocí tohoto prvku určíme, že se bílé mezery budou zachovávat. Tohle pravidlo je aplikováno implicitně, takže ho uvádět nemusíme.

```
<xsl:preserve-space element="MOTOCYKL"/>
```

Bíle mezery v šabloně nejsou sami osobě považovány za podstatné a implicitně jsou převedeny na jednu jedinou mezeru. Tomuto můžeme předejít tak, že použijeme prvek xsl:text:

```
<xsl:template match="MOTOCYKL">
  <xsl:text>   Tento text obsahuje prefix tří mezer.</xsl:text>
</xsl:template>
```

Posledním „trikem“ pomocí něhož si můžeme upravovat v dokumentu prázdné mezery, je přidat do prvku xsl.stylesheet atribut indent-result. Pokud mu nastavíme hodnotu na yes, dovolíme tím procesoru přidávat do výstupu (ale ne odebírat) bíle mezery, například pro pěkný tisk. Umožní to odsazení textu a zalamování řádek:

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" indent-result="yes">
  <!-- sem vložte šablony-->
</xsl:stylesheet>
```

Pokud generujeme HTML, umožníme tím pouze lepší čitelnost výsledného kódu stránky. Implicitní hodnota je nastavena na *no*, protože jiné výstupní formáty, než je HTML, mohou považovat mezery jako důležité.

9.2 Znak <

Standard XSL neobsahuje žádnou značku, která by umožňovala přímé vložení znaku < tak, aby nebyla braná jako část tagu. Místo toho můžeme vložit znakovou reprezentaci tohoto znaku &# 60, nebo entitní referenci &lt.

Tento způsob zápisu nám ale nebude fungovat, pokud budeme potřebovat vložit do našeho dokumentu JavaScript, protože JavaScript používá znak < jako operátor porovnávání čísel a neakceptuje proto jeho znakovou reprezentaci v HTML formě &lt.

Můžeme ale do výstupu vložit přímo znaky > a >=. Takže pokud budeme v obsahu JavaScriptu potřebovat operátor porovnání menší než <, můžeme operandy mezi sebou zaměnit, a znak nahradit operátorem větší než >:

```
function zaporne(parametr){
  if (parametr < 0){
    Response.write('zaporne cislo')
  }
}
```

Tento kód by byl procesorem XSL nepřeložitelný, proto tyto řádky přepíšeme do následujícího tvaru:

```
function zaporne(parametr){
  if (0 > parametr){
    Response.write('zaporne cislo');
  }
}
```

Mohli bychom také skript uložit do samostatného dokumentu a potom jej vkládat pomocí prvku SCRIPT s atributem SRC. Tato možnost ale není moc spolehlivá v dřívějších verzí prohlížečů Internet Explorer 4 a Netscape Navigator 3.

9.2.1 prvek msxsl:script

Pokud pracujeme s procesorem od společnosti Microsoft, můžeme využívat při vkládání skriptů do dokumentu prvek msxsl:script. Tento prvek obsahuje kód skriptu a jeho funkce mohou být použity při XSLT transformacích. Tento prvek je prvkem nejvyšší úrovně.

Akceptuje dva atributy. První atribut language označuje jméno skriptovacího jazyka. Pokud explicitně žádné jméno neuvedeme, vezme se jeho implicitní hodnota, což je Jscript.

Druhým atributem, který je ale povinný, je atribut implement-prefix. Tím deklarujeme jmenný prostor a asociujeme jej se skriptem. Hodnota atributu je prefix, který reprezentuje jmenný prostor.

Tento prvek patří do jmenného prostoru urn:schemas-microsoft-com:xslt. Můžeme v něm deklarovat proměnné a definovat funkce. Vyskytovat se může v prvku xsl:stylesheet.

9.3 Funkce zděděné z jazyka XPath

9.3.1 Funkce concat()

Tato funkce je vlastně funkcí zřetězení. Vrací nám hodnotu, která vznikne zřetězením argumentů, které funkci předáváme.

Příklad:

```
concat('nazdar',' ','světe')
```

Výsledek:

```
nazdar světe
```

9.3.2 Funkce contains()

Funkce vracející hodnotu typu boolean. Funkce vrací hodnotu true, jestli druhý řetězec je obsažen v řetězci prvním.

Příklad:

```
contains('XML','ML')
```

Výsledek:

```
true
```

9.3.3 Funkce normalize-space()

Tato funkce nám odstraní počáteční a koncové mezery z hodnoty typu string, kterou ji předáme jako parametr.

Příklad:

```
normalize-space( 'nazdar   světe ')
```

Výsledek:

```
'nazdar světe'
```

9.3.4 Funkce starts-with()

Vrací nám hodnotu true, jestliže první řetězec začíná řetězcem druhým.

Příklad:

```
starts-with('nazdar','n')
```

Výsledek:

```
true
```

9.3.5 Funkce string()

Tato funkce nám pouze zkonvertuje hodnotu atributu na hodnotu typu string.

Příklad:

```
starts-with(1,25)
```

Výsledek:

```
'1.25'
```

9.3.6 Funkce string-lenght()

Tato funkce nám vrátí jako výsledek počet znaků v řetězci.

Příklad:

```
string-lenght('nazdar světe');
```

Výsledek:

```
12
```

9.3.7 Funkce substring()

Vrací část řetězce. Tato funkce akceptuje tři parametry: Prvním je počáteční řetězec, druhým parametrem je začátek a třetím konec požadovaného výběru.

Příklad:

```
substring('Beatles',1,4)
```

Výsledek:

```
Beat
```

9.3.8 Funkce substring-after() a funkce substring-before()

Funkce substring-after(), respektive substring-before(), nám vracejí část řetězce předaného jako parametr, která se vyskytuje před, respektive za zadaným znakem.

Příklad:

```
substring-after('Beatles',t)
```

Výsledek:

```
les
```

9.3.9 Funkce translate()

Tato funkce vezme hodnotu argumentu a nahradí všechny výskyty znaku, který předáme jako parametr jiným znakem.

Příklad:

```
translate('noc','n','m')
```

Výsledek:

```
moc
```

10. Ošetřování chybových stavů

10.1 Prvek xsl:fallback

Tento prvek je navržen proto, aby ošetřil situaci, když překladač nemůže obsloužit XSL prvek, který může být třeba částí nové verze, nebo může obsahovat nerozpoznatelný znak. Pracuje pomocí volání obsahu šablony, která může obsahovat text, který nás na vzniklou situaci upozorní. Neakceptuje žádné parametry.

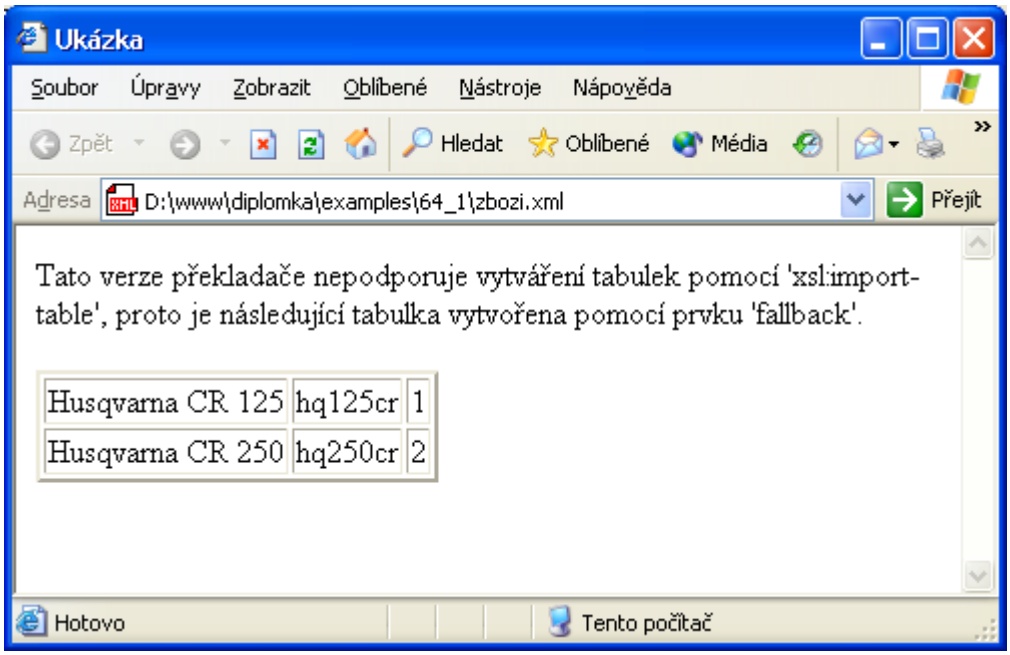
Když je soubor XSL poprvé nahrán do paměti, překladač provede validaci tohoto souboru. Jestliže verze obsažená v prvku xsl:version je větší, než verze podporovaná překladačem a překladač narazí na neznámý prvek, provede překladač volání prvku xsl:fallback, nebo pokud tento prvek není definovaný neprovede se nic. Jestliže je prvek verzí překladače podporován, prvek xsl:fallback se nikdy nezavolá. Ale pokud se obě verze shodují a překladač narazí na chybu, vypíše svou chybovou hlášku.

Prvky fallback jsou částí dopředných mechanismů (forward-processing mechanism), které procesor XSLT používá k ošetřování zlepšení. Při vytváření alternativních cest pro ošetřování příkazů, kdy prvek není podporován , dopředné mechanismy zaručují, že použitý kód je relativně robusní a dostatečně citlivý ke změnám v parseru.

Jak toto funguje si ukážeme na následujícím příkladu:

```
<xsl:stylesheet version="1.1"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
     <HTML>
     <HEAD><TITLE>Ukázka fallback</TITLE></HEAD>
     <BODY>
     <xsl:import-table HREF="tabulka.asp" name="sample">
      <xsl:fallback>
      <p>Tato verze překladače nepodporuje vytváření tabulek pomocí
'xsl:import-table', proto je následující tabulka vytvořená pomocí
prvku 'fallback'.</p>
        <table border='2'>
         <xsl:for-each select='ZBOZI/MOTOCYKL'>
           <tr>
            <td><xsl:value-of select='NAZEV'/></td>
            <td><xsl:value-of select='OZNACENI'/></td>
            <td><xsl:value-of select='CISLO'/></td>
           </tr>
         </xsl:for-each>
        </table>
      </xsl:fallback>
     </xsl:import-table>
      </BODY>
   </HTML>
  </xsl:template>
</xsl:stylesheet>
```

A zde máme výstup:



10.2 Prvek xsl:message

Tento prvek posílá textovou zprávu buď zásobníku zpráv, nebo do dialogového okna, v závislosti na prostředí, ve kterém je prvek vytvořen. Může být také součástí chybových hlášek generovaných procesorem.

Akceptuje jeden nepovinný atribut terminate, který určí, zda při vytvoření tohoto prvku má být zbývající překlad ukončen.

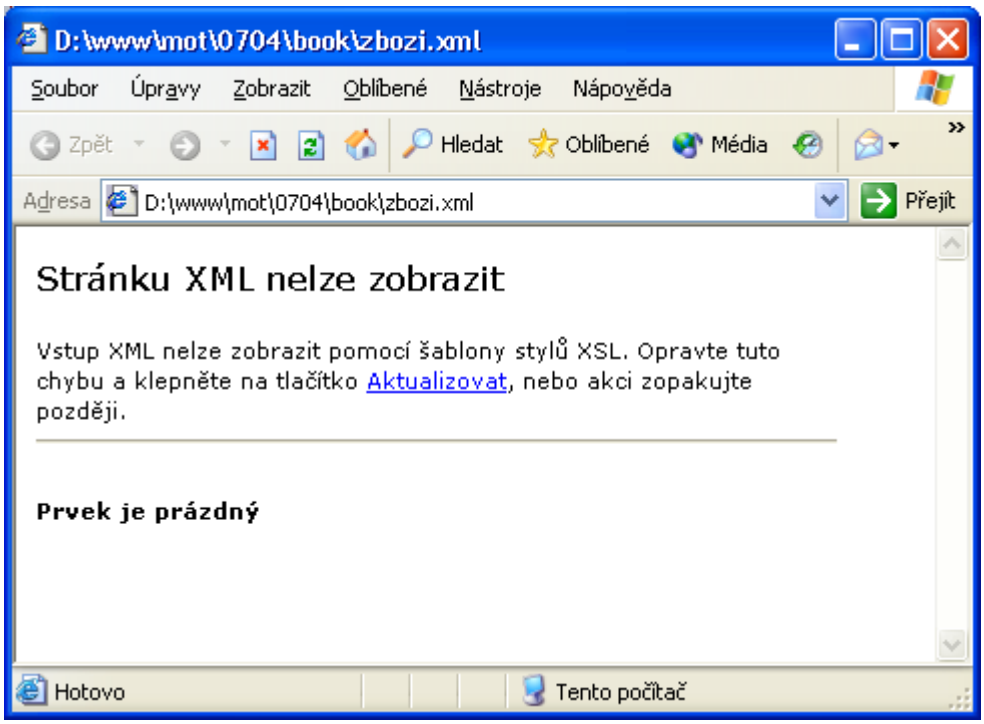
Podívejme se na následující příklad a předpokládejme, že náš XML dokument s motocykly obsahuje alespoň jeden prvek NAZEV prázdný:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="/">
     <xsl:apply-templates select="*"/>
     <xsl:copy-of select="."/>
  </xsl:template>

  <xsl:template match="*">
     <xsl:apply-templates select="//NAZEV"/>
  </xsl:template>
```

```
  <xsl:template match="NAZEV">
     <xsl:if test=".=''">
         <xsl:message terminate="yes">Prvek je prázdný
         </xsl:message>
     </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

Zde je výsledek:



11. Ještě jednou šablony

11.1 Pojmenované šablony

Proměnné jsou omezeny textem a značkami. XSL nabízí mocnější a snadnější možnost jak oddělit měnící se věci od neměnících se. Například pokud bychom chtěli vypsat například hmotnosti motocyklů, výšky sedadel a ostatní hodnoty do jednotlivých buněk tučným modrým písmem Times tak, abychom dostaly následující výstup:

```
<td>
<font face="Times,serif" color=="blue" size=="2">
<b>52</b>
</font>
</td>
```

mohli bychom šablonu samozřejmě vytvořit takto:

```
<xsl:template match="CISLO">
  <td>
    <font face="Times,serif" color="blue" size="2">
      <b>
        <xsl:value-of select="."/>
      </b>
    </font>
  </td>
</xsl:template>
```

Toto značkování bychom museli opakovat pro každý atribut motocyklu. Museli bychom toto opakovat všude, kde bychom chtěli mít výstup v těchto buňkách. Jiné – lepší řešení je přes pojmenované šablony. Pojmenované šablony se podobají proměnným. Umožňují nám ale vkládat data z jednoho místa deklarace na jakékoliv místo v dokumentu.

Prvek xsl:template může obsahovat atribut name, který může explicitně zavolat šablonu daného jména. Na následujícím výstupu je příklad pojmenované šablony:

```
<xsl:template name="MOTOCYKL_POJMSAB">
  <td>
    <font face="Times,serif" color="blue" size="2">
      <b>
        <xsl:value-of select="."/>
      </b>
    </font>
  </td>
</xsl:template>
```

Prvek <xsl:value-of select“.“/> uprostřed kódu bude nahrazen obsahem uzlu, ze kterého je šablona volána. Při zpracování je prvek xsl:call-template nahrazen obsahem prvku xsl:template. Nyní můžeme přepsat pravidlo pro CISLO pomocí xsl:call-template takto:

```
<xsl:template match="CISLO">
  <xsl:call-template name="MOTOCYKL_POJMSAB"/>
</xsl:template>
```

Toto jednoduché pravidlo nám nyní ušetří jen několik řádek kódu, ale při komplikovaných šablonách nám může ušetřit spoustu práce. Pojmenované šablony mají jako proměnné stejnou výhodu v tom, že pokud bychom se v budoucnu rozhodli změnit barvu buněk třeba na červenou, stáčí nám udělat tuto změnou pouze na jednom místě v dokumentu, a ne v každém výskytu šablony. To umožní jednodušší spravování kódu, hlavně pokud ho dlouho používáme a často měníme jeho hodnoty.

11.1.1 Prvek xsl:param a xsl:with-param

Při každém volání pojmenované šablony jí můžeme předat parametry pro úpravu jejího výstupu. Pomocí prvku xsl:param deklarujeme pojmenovaný parametr pro použití v prvcích xsl:stylesheet a xsl:template. Prvek nám také dovoluje specifikovat implicitní hodnotu. Pokud máme parametr definovaný, můžeme se na něj odkazovat pomocí prefixu $.

Prvek xsl:param akceptuje dva atributy. První atribut name, který je povinný, specifikuje jméno parametru. Druhým atributem je select.

Hodnota tohoto atributu může být objekt, nebo jakýkoliv jiný typ, který vrací výraz. Prvek xsl:param může specifikovat hodnotu proměnné třemi různými způsoby:

- Pokud má prvek definovaný atribut select, hodnota atributu musí být výraz a hodnota parametru je objekt, který je výsledkem výrazu. V tomto případě hodnota prvku musí být prázdná.
- Jestliže prvek nemá atribut select definovaný a má neprázdný obsah, jako je třeba jeden nebo několik prvků, jeho obsah specifikuje jeho hodnotu. Obsah je šablona, která vrací hodnotu parametru.
- Jestliže je obsah prvku prázdný a atribut select není definovaný, je hodnota prvku prázdný řetězec.

Následující dva prvky jsou totožné:

```
<xsl:param name="x"/>
```

```
<xsl:param name="x" select="''"/>
```

Prvek xsl:with-param naopak použijeme v pojmenované šabloně, kterou jsme s prvkem xsl:param zavolali. Obsahuje dva atributy. První name nám udává jméno parametru a je povinný. Druhý atribut select je nepovinný a obsahuje výraz, který je porovnáván s kontextovým uzlem. Neexistuje zde žádná implicitní hodnota. Pokud neuvedeme žádný obsah, je vygenerován prázdný řetězec.

Pokud například budeme chtít do výstupu zahrnout odkaz na každý prvek, výstup by měl tedy vypadat takto:

```
<td>
  <font face="Times,serif" color=="blue" size=="2">
    <b>
      <a href="cislo_motocyklu.html">1</a>
    </b>
  </font>
</td>
```

Hodnota atributu je závislá na místě volání šablony a může být pokaždé jiná. Například hmotnost může mít následující formát:

```
<td>
  <font face="Times,serif" color=="blue" size=="2">
    <b>
      <a href="hmotnost_motocyklu.html">93,6</a>
    </b>
  </font>
</td>
```

Šablona, která nám takovéto výstupy zajistí vypadá takto:

```
<xsl:template name="MOTOCYKL_POJMSAB">
  <xsl:param name="file">
    index.html
  </xsl:param>
  <td>
    <font face="Times,serif" color="blue" size="2">
      <b>
```

```
        <a href="{$file}"><xsl:value-of select="."/></a>
      </b>
    </font>
  </td>
</xsl:template>
<xsl:template match="NAZEV">
  <xsl:call-template name="MOTOCYKL_POJMSAB">
    <xsl:with-param name="file">nazev_motocyklu.html
    </xsl:with-param>
  </xsl:call-template>
  <xsl:value-of select="."/>
</xsl:template>
```

Toto je pouze jednoduchý příklad pro názornost problému. Užitečnost této šablony pocítíme hlavně ve složitých šablonách. Například, když bychom chtěli definovat záhlaví a zápatí pro stránky na webu a importovat do nich několik stylů, každý by mohl mít několik parametrů jako je autor, titulek stránky, nebo datum vytvoření.

11.2 Spojování šablon

Někdy můžeme potřebovat použít v jedné šabloně obsah i dalších šablon. K tomu slouží prvky xsl:import a xsl:include.

11.2.1 Importování pomocí xsl:import

Prvek xsl:import je prvek nejvyšší úrovně a musí ležet před všemi ostatními prvky v prvku xsl:stylesheet. Například tento styl importuje šablony ze souborů grafika.xsl a menu.xsl:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="grafika.xsl"/>
  <xsl:import href="menu.xsl"/>
  <!-- sem vložíme ostatní potomky -->
</xsl:stylesheet>
```

Pravidla v importované šabloně mohou být v konfliktu s pravidly definovanými v jiném dokumentu, do kterého šablonu importujeme. Pokud toto nastane, bude platit pravidlo z dokumentu, do kterého šablonu importujeme. Jestliže jsou v konfliktu dvě pravidla z více importovaných šablon, potom má přednost to později importované (v našem případě menu.xsl).

Prvek xsl:apply-imports je slabší variantou xsl:apply-templates, která použije pouze importovaná pravidla. Můžeme tak docílit volání importované šablony, i když ji ve svém dokumentu máme přetíženou.

11.2.2 Zahrnování pomocí xsl:include

Prvek xsl:include je prvek nejvyšší úrovně, který kopíruje jinou šablonu do šablony, ve které je použit. (Přesněji, kopíruje obsah prvku xsl-stylesheet ze vzdáleného dokumentu.) Jeho atribut href obsahuje URI adresu zdrojového dokumentu. Prvek xsl:include můžeme použít všude, nejen na nejvyšší pozici, jako tomu je u prvku xsl:import.

Na rozdíl od pravidel vloženými pomocí prvku xsl:import, pravidla vložená pomocí xsl:include, mají stejnou prioritu, jako kdyby byly ze zdrojového dokumentu zkopírovány a vloženy přímo do zpracovávaného dokumentu. Když je formátovací nástroj spojí, není rozdíl mezi prvky přímo zapsanými v dokumentu a prvky vloženými pomocí xsl:include.

1.2.3 Vkládání vzorů do dokumentů pomocí xsl:stylesheet

Do dokumentu můžeme přímo vkládat XSL šablonu. Nedoporučuji toto provádět v praxi, protože mnoho prohlížečů a XSL procesorů tento způsob nepodporuje. Přesto se o této možnosti krátce zmíním.

Abychom šablonu vložily, musí být prvek xsl:stylesheet obsažen jako potomek kořenového prvku dříve, než kořenový prvek sám. Může mít definován atribut id, který mu přiřadí jednoznačné jméno, na které se můžeme odkazovat pomocí atribut href v prvku xsl:stylesheet, následovaný znakem #. Následující výpis nám toto objasní:

```
<?xml version="1.0" encoding="windows-1250"?>
<?xml-stylesheet type="text/xsl" href=="#id(novystyl)"?>
<ZBOZI>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" id="novystyl">
  <xsl:template match="/">
    <html>
      <xsl:apply-templates/>
    </html>
  </xsl:template>
  <xsl:template match="ZBOZI">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="MOTOCYKL">
    <P>
      <xsl:value-of select="."/>
    </P>
  </xsl:template>
</xsl:stylesheet>
```

```
<MOTOCYKL>
    <NAZEV>Husqvarna CR 125</NAZEV>
    <OZNACENI>hq125cr</OZNACENI>
    <KARBURATOR>Keihin PWK 39</KARBURATOR>
    <CISLO>1</CISLO>
    <VYSKA_SEDLA UNITS="milimetr">925</VYSKA_SEDLA>
    <SVETLA_VYSKA UNITS="milimetr">390</SVETLA_VYSKA>
    <VAHA UNITS="kilogram"><!-- při prázdné nádrži -->93,6</VAHA>
  </MOTOCYKL>
</ZBOZI>
```

11.3 Spojování pomocí funkce document()

Funkci document() používáme pro zpřístupnění prvků v jiném – externím XML dokumentu. Externě připojovaný XML dokument musí samozřejmě být správně strukturovaný a přeložitelný.

Tato funkce se nám výborně hodí, pokud potřebujeme vytvořit výsledný dokument, který obsahuje takové informace, které by správně logicky uspořádaný XML dokument nemohl obsahovat.

12. Rozhodování

XSL nabízí dva prvky, které nám umožňují vytvářet výstupní dokument v závislosti na vstupním. Prvek xsl:if buď zahrne, nebo nezahrne určitou část kódu do výsledného dokumentu v závislosti na vstupních datech. Prvek xsl:choose vybere jednu z více možností také v závislosti na vstupu. To, co můžeme vytvořit pomocí těchto prvků, můžeme udělat také pomocí odpovídající šablony. Někdy ale může být řešení s xsl:if a xsl:choose jednodušší a zřetelnější.

12.1 Prvek xsl:if

Prvek xsl:if nabízí jednoduchou volbu pro změnu výstupu v závislosti na vstupních datech. Atribut test tohoto prvku obsahuje výraz select, který je vyhodnocen jako booleovský. Jestliže je výraz pravdivý, obsah prvku xsl:if je zapsán do výstupu. Například toto pravidlo vypíše jména všech prvků MOTOCYKL. Čárka a mezera je vložena za každý motocykl, kromě posledního:

```
<xsl:template match="MOTOCYKL">
  <xsl:value-of select="NAZEV"/>
  <xsl:if test="not(position()=last())">,</xsl:if>
</xsl:template>
```

To zajistí následující výpis: „Husqvarna CR 125, Husqvarna CR 250“ a nikoliv „Husqvarna CR 125, Husqvarna CR 250, „

XSL nenabízí žádné prvky jako třeba xsl:else nebo xsl:else-if jako je tomu například v programovacím jazyku Java. Tyto vlastnosti jsou obsaženy v prvku xsl:choose.

12.2 Prvek xsl:choose

Tento prvek vybírá jednu z více možností podle různých podmínek. Každá podmínka a jí přiřazená výstupní šablona je zajištěna potomkem xsl:when. Atribut test prvku xsl:when je výraz výběru s pravdivostní hodnotou. Jestliže vyhovuje více podmínek, je provedena podmínka pouze první. Jestliže nevyhovuje žádná, použije se hodnota prvku xsl:otherwise, který je potomkem prvku xsl:choose. Například následující šablona nám změní barvu výstupu v závislosti na atributu DRUH prvku MOTOCYKL:

```
<xsl:template match="MOTOCYKL">
  <xsl:choose>
    <xsl:when test="@DRUH='cross'">
      <P style="color:green">
        <xsl:value-of select="."/>
      </P>
    </xsl:when>
    <xsl:when test="@DRUH='enduro '">
      <P style="color:brown">
        <xsl:value-of select="."/>
      </P>
    </xsl:when>
    <xsl:when test="@DRUH='street'">
      <P style="color:blue">
        <xsl:value-of select="."/>
      </P>
    </xsl:when>
    <xsl:otherwise>
      <P style="color:yellow">
        <xsl:value-of select="."/>
      </P>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

12.3 Booleovské funkce

Booleovské funkce jsou obsažené snad v každém programovacím jazyce. Obsahuje je i jazyk XSLT. Všechny tyto funkce jsou odvozené od jazyka XPath.

12.3.1 Funkce boolean()

Tato funkce pouze zkonvertuje hodnotu argumentu na hodnotu typu boolean. Jak funkce pracuje si nejlépe ukážeme na příkladu:

```
<li><b>boolean(0)</b> =
  <xsl:value-of select="boolean(0)"/>
</li>
<li><b>boolean(1)</b> =
  <xsl:value-of select="boolean(1)"/>
</li>
<li><b>boolean(-100)</b> =
  <xsl:value-of select="boolean(-100)"/>
</li>
<li><b>boolean(100)</b> =
  <xsl:value-of select="boolean(100)"/>
</li>
<li><b>boolean(NaN)</b> =
  <xsl:value-of select="boolean(NaN)"/>
</li>
<li><b>boolean('text')</b> =
  <xsl:value-of select="boolean('text')"/>
</li>
```

```
<li><b>boolean('')</b> =
  <xsl:value-of select="boolean('')"/>
</li>
<li><b>boolean(//motocykl)</b> =
  <xsl:value-of select="boolean(//MOTOCYKL)"/>
</li>
<li><b>boolean(//notfound)</b> =
  <xsl:value-of select="boolean(//notfound)"/>
</li>
```

Zde jsou výsledky:

- *boolean(0) = false*
- *boolean(1) = true*
- *boolean(-100) = true*
- *boolean(100) = true*
- *boolean(NaN) = false*
- *boolean('text') = true*
- *boolean('') = false*
- *boolean(//MOTOCYKL) = true*
- *boolean(//notfound) = false*

12.3.2 Funkce true() a false()

Tyto dvě funkce vrací v prvním případě hodnotu true a v druhém případě false. Jazyk XSLT nemá žádné definované konstanty, takže pokud chceme porovnat nějakou hodnotu proti pravdivostní hodnotě, musíme použít jednu z těchto funkcí.

12.3.3 Funkce lang()

Výsledkem této funkce je hodnota true, jestliže jazyk kontextového uzlu šablony je stejný jako jazyk, který jsme jí předali jako parametr.

Jazyk kontextového uzlu je určený hodnotou atributu xml:lang, nebo pokud kontextový uzel nemá parametr lang() definovaný, použije se hodnota atributu xml:lang nejbližšího uzlu, který je předkem kontextového uzlu. Pokud atribut xml:lang není definovaný, vrací funkce hodnotu false. Hodnotu true dostaneme, pokud se atribut xml:lang shoduje s parametrem funkce.

12.3.4 Funkce not()

Tato funkce je vlastně negace. Vrací hodnotu true, pokud výsledek výrazu předaný jako parametr je false, a vrací hodnotu false, pokud je výsledek výrazu true.

13. Ostatní funkce

13.1 Funkce function-available()

Tato funkce zjišťuje, zda existuje funkce obsažená v jejím argumentu. Argument funkce musí být vyhodnocen jako řetězec, který je hodnotou QName. QName je rozšířeno v rozšířené jméno s použitím deklarace jmenného prostoru v oblasti pro daný výraz. Funkce vrací hodnotu true, pouze pokud rozšířené jméno je jménem funkce v knihovně funkcí. Pokud rozšířené jméno má nenulový jmenný prostor URI, odkazuje na rozšířenou funkci, jinak odkazuje na funkci definovanou pomocí jazyka XPath, nebo XSLT.

Příklad:

```
function-available("format-number")
```

Výsledek:

```
true
```

13.2 Funkce system-property()

Tato funkce vrací objekt, reprezentující hodnotu systémové proměnné, které funkci předáme jako parametr.

Parametry mohou být:

- xsl:version – verze xsl
- xsl:vendor – jméno dodavatele xsl procesoru
- xsl:vendor-url – url adresa dodavatele xsl procesoru

Pokud budete používat procesor od firmy Microsoft, dostaneme následující výsledky:

```
1
Microsoft
http://www.microsoft.com
```

13.3 Funkce unparsed-entity-uri()

Pomocí této funkce získáme hodnotu neparsovaného objektu. Jméno objektu se musí shodovat s argumentem funkce. Pokud žádný objekt který, se jmenuje jako argument funkce

neexistuje, je vrácen prázdný řetězec.

Příklad:

```
<!ENTITY pic SYSTEM "http://www.mototip.cz/obr.jpg" NDATA JPEG>
```

***Výsledek** výrazu unparsed-entity-uri('pic'):*

file:///D:/www/obr.jpg

13.4 Funkce pro práci s uzly odvozené od jazyka XPaTH

Pro ukázku následujících dvou funkcí budeme používat následující XML soubor:

```
<test>
    <x a="1">
      <x a="2">
        <x>
          <y>y31</y>
          <y>y32</y>
        </x>
      </x>
    </x>
    <x a="1">
      <x a="2">
        <y>y21</y>
        <y>y22</y>
      </x>
    </x>
    <x a="1">
      <y>y11</y>
      <y>y12</y>
    </x>
    <x>
      <y>y03</y>
      <y>y04</y>
    </x>
</test>
```

13.4.1 Funkce count()

Výsledkem této funkce je počet uzlů obsažených v argumentu node-set.

Příklad:

```
<xsl:template match="/">
  <xsl:value-of select="count(//x)"/><br/>
  <xsl:value-of select="count(//x[1])"/><br/>
  <xsl:value-of select="count(//x/y)"/><br/>
  <xsl:value-of select="count(//x/y[1])"/><br/>
  <xsl:value-of select="count(//x[1]/y[1])"/>
</xsl:template>
```

Zde je výsledek:

7
4
8

```
4
2
```

13.4.2 Funkce last()

Tato funkce vrátí poslední uzel v daném kontextu uzlů.

Příklad:

```
<xsl:template match="/test">
    <xsl:apply-templates select="//x/y[last()]"/>
</xsl:template>
```

Výsledek:

```
y32
y22
y12
y04
```

13.4.3 Funkce local-name()

Tato funkce vrací lokální část rozšířeného jména uzlu obsaženého v seznamu uzlů předaných jako parametr funkce.

Příklad:

```
<xsl:template match="*">
    <xsl:value-of select="local-name()"/> =
    <xsl:value-of select="text()"/><br/>
    <xsl:apply-templates select="*"/>
</xsl:template>
```

Výsledek:

```
ZBOZI =
MOTOCYKL =
NAZEV = Husqvarna CR 125
OZNACENI = hq125cr
CISLO = 1
KARBURATOR = Keihin PWK 39
VYSKA_SEDLA = 925
SVETLA_VYSKA = 390
VAHA = 93,6
MOTOCYKL =
NAZEV = Husqvarna CR 250
OZNACENI = hq250cr
CISLO = 2
KARBURATOR = Keihin PWK 38 S
VYSKA_SEDLA = 925
```

```
SVETLA_VYSKA = 385
VAHA = 97,2
```

13.4.4 Funkce name()

Tato funkce vrací celé kvalifikované jméno QName uzlu obsaženého v seznamu uzlů. Příklad je uveden v další kapitole společně s další funkcí.

13.4.5 Funkce namespace-uri()

Funkce vrací jmenný prostor URI uzlu specifikovaného v parametru.

Příklad:

Předpokládejme, že máme v našem XML dokumentu definovaný jmenný prostor b:.

```
<xsl:template match="*">
    <xsl:value-of select="namespace-uri()"/> - <xsl:value-of
     select="name()"/> = <xsl:value-of select="text()"/><br/>
    <xsl:apply-templates select="*"/>
</xsl:template>
```

Výsledek

```
zbozi.xml - b:ZBOZI =
zbozi.xml - b:MOTOCYKL =
zbozi.xml - b:NAZEV = Husqvarna CR 125
zbozi.xml - b:OZNACENI = hq125cr
zbozi.xml - b:CISLO = 1
zbozi.xml - b:KARBURATOR = Keihin PWK 39
zbozi.xml - b:VYSKA_SEDLA = 925
zbozi.xml - b:SVETLA_VYSKA = 390
zbozi.xml - b:VAHA = 93,6
zbozi.xml - b:MOTOCYKL =
zbozi.xml - b:NAZEV = Husqvarna CR 250
zbozi.xml - b:OZNACENI = hq250cr
zbozi.xml - b:CISLO = 2
zbozi.xml - b:KARBURATOR = Keihin PWK 38 S
zbozi.xml - b:VYSKA_SEDLA = 925
zbozi.xml - b:SVETLA_VYSKA = 385
zbozi.xml - b:VAHA = 97,2
```

13.4.6 Funkce position()

Funkce vrací pozici uzlu v seznamu uzlů, který právě zpracováváme.

Příklad:

```
<xsl:template match="/">
  <xsl:apply-templates select="//NAZEV"/>
</xsl:template>

<xsl:template match="NAZEV">
    <xsl:value-of select="position()"/><br/>
</xsl:template>
```

Výsledek:

```
1
2
```

14. Vytváření seznamů

Na závěr jsem nechal kapitolu o vytváření seznamů z XML souborů pomocí XSLT. Není to proto, že by tato technika byla málo využívanou, ba naopak. Tato technika není zase tak složitá, ale myslím, že je třeba porozumět obsahu skoro všech předcházejících kapitol, abyste mohli seznamy vytvářet.

Na co se vlastně vytváření seznamů hodí? Za příklad opět vezmeme náš známý soubor s motocykly, jen jej trochu poupravíme:

```
<?xml version="1.0" encoding="windows-1250"?>
<?xml-stylesheet type="text/xsl" href="zbozi.xsl"?>
<ZBOZI>
  <MOTOCYKL DRUH="cross">
    <ZNACKA>husqvarna</ZNACKA>
    <NAZEV>CR 125</NAZEV>
    <OZNACENI>hq125cr</OZNACENI>
    <CISLO>1</CISLO>
    <KARBURATOR>Keihin PWK 39</KARBURATOR>
    <VYSKA_SEDLA UNITS="milimetr">925</VYSKA_SEDLA>
    <SVETLA_VYSKA UNITS="milimetr">390</SVETLA_VYSKA>
    <VAHA UNITS="kilogram">
      <!-- při prázdné nádrži -->93,6</VAHA>
  </MOTOCYKL>
  <MOTOCYKL DRUH="cross">
    <ZNACKA>husqvarna</ZNACKA>
    <NAZEV>CR 250</NAZEV>
    <OZNACENI>hq250cr</OZNACENI>
    <CISLO>2</CISLO>
    <KARBURATOR>Keihin PWK 38 S</KARBURATOR>
    <VYSKA_SEDLA UNITS="milimetr">925</VYSKA_SEDLA>
    <SVETLA_VYSKA UNITS="milimetr">385</SVETLA_VYSKA>
    <VAHA UNITS="kilogram">
      <!-- při prázdné nádrži -->97,2</VAHA>
  </MOTOCYKL>
</ZBOZI>
```

Nyní budeme chtít vytvořit seznam všech značek motocyklů. Mohli bychom to samozřejmě jednoduše udělat takto:

```
<xsl:template match="/">
  <xsl:apply-templates select="//ZNACKA"/>
</xsl:template>
<xsl:template match="ZNACKA">
  <xsl:value-of select="."/>
</xsl:template>
```

Dostaneme následující výsledek:

```
Husqvarna Husqvarna
```

Tohle ale není moc dobrý výsledek. O mnoho lepší by bylo, aby to každou značku vypsalo pouze jednou.

Podívejme se ale nejdříve ještě na několik důležitých pojmů:

14.1 Prvek xsl:key a funkce key()

Prvek xsl:key je prvkem nejvyšší úrovně. Musí tedy být definován dříve, než jakákoli jiná šablona. Pomocí tohoto prvku deklarujeme pojmenovaný klíč, který můžeme použít v šabloně pomocí funkce key().

Důležité je, že klíč nemusí být unikátní.

- Tento prvek akceptuje tři parametry, přičemž všechny tři jsou povinné.
- Prvním z nich je parametr name. Tím specifikujeme jméno našeho klíče.
- Druhý parametr – match – definuje uzly, na které bude klíč aplikován.
- Posledním parametrem, který se jmenuje use, definujeme hodnotu klíče pro každý uzel.

Mějme následující XML dokument:

```
<osoby>
  <osoba name="Tarzan" id="050676"/>
  <osoba name="Donald" id="070754"/>
  <osoba name="Dolly" id="231256"/>
</osoby>
```

klíč může být definován takto:

```
<xsl:key name="catId" match="osoba" use="@id"/>
```

Pokud máme nadefinovaný prvek xsl:key, můžeme se na něj odkazovat pomocí funkce key(). Ta nám jako výsledek vrátí množinu uzlů, které vyhovují zadanému klíči. Tato funkce akceptuje dva parametry: první z nich je řetězcová hodnota jména klíče a druhý parametr je řetězec, který hledáme.

Abychom našli osobu například s číslem 070754, použijeme následující šablonu:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:key name="catId" match="osoba" use="@id"/><xsl:template
    match="/">
```

```
    <html>
      <body>
        <xsl:for-each select="key('catId','050676')">
          <p>
            Id: <xsl:value-of select="@id"/><br />
            Name: <xsl:value-of select="@name"/>
          </p>
        </xsl:for-each>
    </body>
  </html>
</xsl:template></xsl:stylesheet>
```

14.2 Funkce generate-id()

Tato funkce vrací řetězcovou hodnotu, která unikátně identifikuje určitý uzel. Pokud je množina uzlů prázdná, výsledkem je prázdný řetězec. Akceptuje jeden nepovinný parametr node-set, kterým specifikujeme pro které uzly máme unikátní klíč generovat.

Při každém zavolání funkce generate-id() nám pro stejnou hodnotu vygeneruje jiné číslo, ovšem vždy při jednom volání pro stejné uzly generuje stejné výsledky.

Nyní můžeme již přejít k vysvětlení postupu pro vygenerování našeho seznamu. Nejdříve si na začátku před šablonami nadefinujeme klíč, pomocí něhož se budeme odkazovat na prvek znacka. Použijeme proto dva vnořené cykly. Ty vytvoříme pomocí xsl:for-each. V prvním cyklu nastavíme unikátní id pro každou značku, a porovnáme jí s unikátním id klíče. V druhém cyklu vyberu rodiče prvku znacka, což je prvek motocykl. Dále nadefinujeme proměnnou n_znacka, která se bude rovna prvku znacka a na výstup pošleme její hodnotu.

Zde je kód:

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:key name="NAZEVMap" match="NAZEV" use="."/>
<xsl:template match="ZBOZI">
  <xsl:for-each select="(//NAZEV[generate-id()=generate-
    id(key('NAZEVMap',.)[1])])">
    <xsl:for-each select="ancestor::*">
      <xsl:variable name="n_NAZEV" select="NAZEV"/>
      <xsl:value-of select="$n_NAZEV"/>
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

15. Kde se může transformace odehrávat?

Existují tři hlavní místa, kde může být XML dokument transformován pomocí XSL style sheet do jiných formátů, jako je třeba HTML.

1. XML dokument a přidružený style sheet se posílají ze serveru klientovi (Webovému prohlížeči), který provede transformaci dokumentu podle dané šablony a zobrazí výsledek.
2. Druhou možností, jak transformovat XML dokument, je transformovat ho pomocí programu ručně a na server umístit již jen výstup z transformace. Potom klient i server pracují již jen z výslednými dokumenty.
3. Další možností je, že server nejdříve aplikuje na XML dokument transformaci, převede ho na nějaký jiný formát (většinou HTML) a pošle ho klientovi (Webovému prohlížeči)

Každý z těchto tří možností využívá jiný software, kterým jsou XML dokumenty transformovány. Každá z možností má své výhody a nevýhody.

Pokud budeme XML dokument spolu se XSL šablonou posílat klientovi, musíme se spoléhat na to, že klient používá webový prohlížeč, který transformaci umožňuje. Jestliže dokumenty budeme generovat předem a na server umístíme již výsledné HTML stránky, tak přijdeme o interakci s klientem, jako je třeba řazení seznamu výrobků jednou podle ceny a jindy podle data výroby.

V praktické části své diplomové práce využívám třetí možnosti – tedy generování HTML kódu na serveru a posílání jej klientovi. Odpadá tak problém s podporou prohlížečů jazyka XML. Není nutné, aby měli uživatelé na svých počítačích nainstalovaný nový webový prohlížeč, když navíc podle posledních průzkumů je uživatelé již tak často neinovují.

15.1 Transformace na serveru pomocí ASP

Stránky ASP lze spouštět v systémech Microsoft Windows NT, Windows 2000 nebo Windows XP. Stránky ASP šli vždy spouštět pouze na webových serverech dodávanými společností Microsoft. Nyní je již lze spouštět i na rozšířeném webovém serveru Apache, Sun ONE Web Server a Solaris Operating Enviroment. Lze toho docílit instalací produktu, který se dříve jmenoval ChilliSoft ASP. Nyní tento projekt převzala firma Sun a jmenuje se Sun ONE Active Server Pages, nejnověji ve verzi 3.6.2.

Stránky ASP můžeme pro transformaci na serveru psát v jazyce JavaScript, nebo v jazyce VBScript. Ukážeme programové kódy, které použijeme v každém z jazyků.

15.1.1 Stránka ASP pomocí JavaScriptu

Stránka ASP začíná deklarací bloku programového kódu. Uvnitř zmíněného bloku nejdříve definujeme typ obsahu generovaného souboru. Použijeme hodnotu „text/html“, takže výsledný dokument bude považován za běžný dokument HTML.

```
<%@LANGUAGE="JavaScript"%>
```

Dále nadefinujeme to, aby se stránka nenahrávala z cache prohlížeče, ale aby se pokaždé stahovala ze serveru. Zamezíme tím, aby klientovi zůstaly skryté změny, které by mohli být provedeny od jeho poslední návštěvy serveru.

```
Response.expires=-1;
```

Aby byl programový kód dobře čitelný a lehce editovatelný, použijeme pro transformaci následující 4 metody:

```
loadSource()
getProcessor()
transformData()
main()
```

Metoda loadSource zajistí zpracování dokumentu XML, přičemž getProcessor zpracuje XSL dokument. Metoda transformData provede danou transformaci. Celý průběh transformace je zajištěn metodou main(), ve které je zajištěno volání jednotlivých metod. V závislosti na vstupních parametrech voláme zpracovávající metody loadSource a getProcessor s parametry příslušných dokumentů. Před samotnou transformací do dokumentu XSL pošleme inicializované proměnné.

Zde je kód:

```
<%@LANGUAGE="JavaScript"%>
<%Response.expires=-1;
function loadSource(sourceName){
  var xmlDoc=
      new ActiveXObject("MSXML2.FreeThreadedDOMDocument.3.0");
  xmlDoc.async=false;
  xmlDoc.load(Server.mapPath(sourceName));
  return xmlDoc;
}
function getProcessor(transformName){
  var xslDoc=
    new ActiveXObject("MSXML2.FreeThreadedDOMDocument.3.0");
  var xslTemplate=
    new ActiveXObject("MSXML2.XSLTemplate.3.0");
```

```
  xslDoc.async=false;
  xslDoc.load(Server.mapPath(transformName));
  xslTemplate.stylesheet=xslDoc;
  xslProcessor=xslTemplate.createProcessor();
  return xslProcessor;
}

function transformData(srcDoc,processor){
  processor.input=srcDoc;
  processor.output=Response;
  processor.transform();
  return true;
}

function main(){
  var srcDoc;
  var processor;
  srcDoc=loadSource("zbozi.xml");
  processor=getProcessor("zbozi.xsl");
}

main();
%>
```

15.1.2 Stránka ASP pomocí jazyka VBScript

V programovém kódu jazyka VBScript nejprve také určíme typ obsahu generovaného souboru. V tomto případě deklaraci souboru XML i XSL provedeme v jediné metodě loadXMLFile(strXMLFile, strXSLFile), která přijímá jako parametry jména souborů XML a XSL.

```
<%@LANGUAGE="VBScript"%>
<%
Response.ContentType="text/html"

Function loadXMLFile(strXMLFile, strXSLFile)
  Dim objXML
  Dim objXSL
  set objXML = Server.CreateObject("Microsoft.XMLDOM")
  objXML.async = false
  objXML.load(strXMLFile)
  set objXSL = Server.CreateObject("Microsoft.XMLDOM")
  objXSL.async = false
  objXSL.load(strXSLFile)
  Response.Write(objXML.transformNode(objXSL))
End Function

loadXMLFile server.MapPath("bazar.xml"),  server.MapPath("bazar.xsl")
End Select
%>
```

15.2 Transformace na serveru PHP

Další možností jak provádět transformaci XML dokumentů na serveru je pomocí skriptů PHP. Skripty PHP mohou běžet na serveru Apache na různých platformách. Rozšíření skriptů PHP o transformaci XML dokumentů je ale zatím pouze experimentální a ještě nebyla vydána žádná oficiální verze. Toto rozšíření využívá Sablotron a expat, což jsou XSL procesory.

```
<?php
  // Vytvoří XSLT procesor
  $xsltHandle = xslt_create();
  // Provede příslušnou transformaci
  $path='file://d:/www/pes/mototip/source/';
  if(isset($_GET['item']))
    $item=($_GET['item']);
  else $item='uvod';
  $xml=$path.$item.'.xml';
  $xslt=$path.$item.'.xslt';
  $html = xslt_process($xsltHandle, $xml, $xslt);

  //Ohlásí případné chyby
  if (!$html) die('XSLT processing error: '.xslt_error($xsltHandle));
  // Uvolní XSLT procesor
  xslt_free($xsltHandle);
  // Vypíše výsledné HTML
  echo $html;
?>
```

15.3 Transformace v prohlížeči

XML soubory můžeme také transformovat v prohlížeči na straně klienta. Již existuje celá řada prohlížečů, které tuto transformaci podporují. Jejich kompletní popis je k dispozici na adrese www.xmlsoftware.com. Nejznámější z nich je určitě Explorer od společnosti Microsoft nyní ve verzi 6.

Internet Explorer obsahuje XSL procesor od verze 5. Bohužel však verze 5 a 5.5 byly vydány dříve, než byl standard XSL kompletní. Z tohoto důvodu není v těchto verzích podpora XSL kompletní. Podle doporučení konsorcia W3C využívají dokumenty XSL jmenného prostoru: xmlns:xsl="http://www.w3.org/1999/XSL/Transform". V případě Internet Exploreru ve verzích 5 a 5.5 se jedná o jmenný prostor z pracovního návrhu (XSL Working Draft):
xmlns:xsl="http://www.w3.org/TR/WD-xsl"
Tento rozdíl se může zdát zanedbatelný, ale musíme si uvědomit, že v každém ze jmenných prostorů mohou být určité prvky zcela odlišné.

Dále musíme ještě trochu pozměnit XML soubor. Místo typu „text/xml“ musíme použít typ „text/xsl“.

```
<?xml-stylesheet type="text/xsl" href="zbozi.xsl"?>
```

Použité zdroje:

Literatura

Jiří Kosek. *XML pro každého, podrobný průvodce*,
Grada 2000

Neil Bradley *XML kompletní průvodce*,
Grada 2000

Benoit Marchal *XML v příkladech*,
Computer Press

Steven Holzner *XSLT - příručka internetového vývojáře*,
Computer Press 2002

Elliotte Rusty Harold *XML bible*,
IDG WorldWide Inc. 1999

John W.Shipman *XSLT reference*,
New Mexico Tech

Jiří Bulíček *Tvorba internetových aplikací v XML*. Diplomová práce.
Jihočeská univerzita, Pedagogická fakulta, Katedra informatiky 2000

Vít Profant *Vývoj aplikací XML s využitím programovacího jazyka Java*
Diplomová práce
Jihočeská univerzita, Pedagogická fakulta, Katedra informatiky 2003

Internet:

http://www.w3c.org

http://www.xml.com

http://www.15seconds.com

http://msdn.microsoft.com

http://www.xmlfiles.com

http://www.grafika.cz

http://zvon.org

http://xml.coverpages.org

http://www.kosek.cz

http://www.interval.cz

http://nb.vse.cz/~zelenyj/IT380/Eseje/xdeva01/xsl.htm

http://www.devarticles.com/art/1/105/2

http://ww.cw.cz

http://wes.wyda.cz/Pocitace/Texty/slovnik.asp

http://www.datis.cdrail.cz/Black/Scripts/ZKRATKY/slovnik.asp

http://www.jenitennison.com/xslt/grouping/muenchian.html

http://phoenix.inf.upol.cz/~kopka/courses/cs409/xml/XML_FAQ.htm

http://www.w3schools.com

news://news.cesnet.cz/cz.comp.lang.xml

news://news.microsoft.com/microsoft.public.xml

news://news.microsoft.com/microsoft.public.xml.msxml-webrelease

http://web.quick.cz/ichladil/commerce/xml_intro21.cs.html

http://edu.webpark.cz

Přílohy

Příloha č. 1 - Slovníček zkratek

Příloha č. 1

Slovníček zkratek

- **ASP -** Active Server Pages. Technologie pro generování dynamických HTML stránek na straně serveru vyvinutá firmou Microsoft
- **DTD** – Document Type Definitions – Konkrétní specifický markup jazyk napsaný s použitím HTML
- **HTML** – Hyper Text Markup Language – Kódovací jazyk používaný pro vytváření formátovaných dokumentů na web.
- **HTTP** – Hyper Text Transport Protocol – Protokol určený k transportu hypertextových souborů po internetu.
- **ISO** – International Organization for Standardization – Mezinárodní autoritativní organizace pro zavádění celosvětových standardů
- **MIDI** – Musical Instrument Digital Interface – Standardizovaný popis hardwarových a softwarových parametrů pro spojení počítače s vnějšími hudebními zaří-zeními
- **PDF** – Portable Document Format – Formát pro přenositelnost dokumentů s potencionálně složitým layoutem vyvinutý firmou Adobe
- **PHP -** Technologie pro generování dynamických HTML stránek na straně serveru
- **PostScript** – Jazyk pro kompletní popis tiskové strany, nezávislý na výstupním zařízení.
- **QName** – Jsou to jména, která se skládají z prefixu jmenného prostoru, dvojtečky a lokální části. Prefix jmenného prostoru je zkratkou pro URI
- **SGML** – Standard Generalized Markup language – Obecný metajazyk, výchozí pro všechny další značkovací jazyky.
- **SQL** – Structured Query Language – Databázový dotazovací jazyk
- **URI** – Universal Resource Identifier – Univerzální jmenné schéma pro inden-tifikaci v prostoru WWW. Popsáno RFC 1630
- **UTF** – USC Transformation Format – Kódování znaků, které ukládá ASCII znaky do jednoho byte, méně obvyklé znaky do více bytes. Varianty UTF-8 a UTF-16
- **W3C** – World Wide Web Consortium – Organizace, která definuje standardy pro www
- **XML** – eXtended Markup Language – Platformově nezávislý formát pro publikování a výměnu dat a dokumentů
- **Xquery** – Dotazovací jazyk pro XML dokumenty
- **XSL** – eXtensible Stylesheet Language – Deklarativní jazyk pro popis tříd a procesů