# Transformační jazyk XSLT a jeho aplikace

Vilém Vychodil

4. března 2001

**Abstrakt**

*Tento referát se zabývá využitím transformačního jazyka XSLT — Extensible Stylesheet Language Transformations, který byl vedle formátovacích objektů původně vyvinutý jako jedna ze dvou částí XSL. I přesto, že tento jazyk nebyl původně navržen pro transformaci obecných XML stromů, lze jej snadno použít při řešení jednodušších obecných problémů nad XML stromy. Typické je jednoduché dotazování na hodnoty uzlů, transformace datových struktur, nebo transformace dokumentu do presentační podoby.*

## Motivace

*Well–formed* XML dokument, de facto representuje ohodnocený strom. Rootovský element je pouze jediný, ten representuje kořen stromu a jednotlivé elementy, ať už jsou to uzlové elementy, nebo hodnoty atributů, představují uzly stromu. Přirozeně, pro danou instanci stromu jsou i uzly příslušně ohodnoceny. Stejně tak, jako je v praxi potřebná transformace stromů, zdá se být potřebná i transformace XML dokumentů.

Jeden z největších problémů soudobé elektronické publikace, a nejen jí, je striktně neoddělená *logická informace* od její *presentace*. Typickým příkladem, jak by tomu být nemělo, je formát HTML. V něm se typicky slévají data se svou presentací. Oddělení těchto dvou částí publikace je nejen evidentní, ale je zejména praktické, zvlášť například v problematice hledání informací. Vyhledávací WWW servery by v budoucnu mohly vykazovat mnohem větší schopnosti, kdyby prohledávaly soubory vzhledem k jejich logickému obsahu. Toho lze dosáhnout právě oddělením obou složek.

Není těžké si představit, že presentovaná data by sestávala ze dvou částí. Jednak logické složky, jednak jazyka popisujícího její presentaci. Logickou složkou můžeme chápat jako XML dokument, jak jsme jej chápali doposud. Prezentační složka může být obecně transformace, po jejíž aplikaci se logická část dá presentovat, například na WWW, tisknout, a podobně. Pro každý dokument může existovat více transformací, vhodných pro jiné presentační účely. Stejně tak dokumenty stejné třídy, lze presentovat pomocí stejných transformací — stylů.

Ne méně významnou motivací, pro transformaci dokumentu je jednoduchá konverse mezi datovými formáty, v našem případě mezi dvěma jazyky založenými na XML. Uvažujme dva komunikující uzly, z nichž každý používá pro popis stejné problémové domény svůj vlastní XML jazyk. Tyto uzly spolu nemohou komunikovat přímo, ale transformací jednoho jazyka do druhého tak mohou učinit. Transformační jazyk lze tedy využít k vzájemné konversi mezi XSL jazyky.

V praxi se lze bohužel velmi často setkat s fenoménem „kanón na vrabce“. V situaci, kdy je potřeba z XML dokumentu vydolovat jistá data, je nasazení transformačního jazyka velmi přímočaré. Místo programování složitých kódů ve vyšším jazyce pomocí DOM nebo jiného rozhraní, můžeme transformovat dokument do podoby, v níž lze transformovaná dala snadno nalézt. Typickým příkladem budiž dotaz: *„V jakých ulicích bydlí naši zákazníci?“*. Pokud je hledána odpověď programově, znamená to napsat nemalý kód, který prochází strukturu XML, hledá v ní údaje o zákaznících a z nich vybírá ulice. Transformačním jazykem lze ale dokument převést do tvaru, v němž boudou pouze ulice, formulujeme-li transformaci jako dotaz: *„Vrať hodnoty ulic u všech zákazníků“*, potom jsme evidentně za vodou, protože výsledný dokument je právě to, co jsme hledali.

## XSL, XSLT a formátovací objekty

Jak již bylo předesláno v úvodu, XSL — eXtensible Stylesheet Language, byl vyvinut především jako náhrada stávajících kaskádových stylů, v současné době to ale vypadá tak, že oba standardy se budou doplňovat. XSL je jazyk, kterým vytváříme pravidla pro transformaci jedné třídy XML dokumentů na jiný XML dokument. Výsledný dokument, který vznikne aplikací transformačních pravidel, je použit pro zobrazení. Dnes se nejčastěji XSL styl vytváří tak, aby výsledkem jeho aplikace byl HTML nebo XHTML dokument. Samotná transformace se obvykle děje na straně serveru, protože tím odpadají problémy se vzájemnou nekompatibilitou. V současné době dokáže s XSL pracovat jen málo prohlížečů, které navíc neimplementují standard úplně.

XSL je prozatím návrh standardu, který připravuje konsorcium W3C. XSL kromě transformačního jazyka, který bývá někdy označován jako XSLT obsahuje i vlastní formátovací model, který můžeme použít například místo výstupu do HTML.

## Základy XSLT

XSL styl je sám o sobě XML dokumentem. Měl by proto začínat odpovídající XML deklarací.

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

Informace o kódování může být vynechána. XML a tím pádem i XSL bere jako standardní kódování `UTF-8`, což je specifický zápis UNICODE, ve kterém jsou všechny ASCII znaky zachovány a znaky z vyšších kódování jsou vyjádřeny několika jednobytovými znaky. I když je tento zápis například proti `ISO-8859-2` o něco více plýtvá prostorem, není zdaleka tak neúsporný jako dvojbytový UNICODE. A jeho výhody jsou rovněž evidentní — velká podpora.

Jelikož je výsledkem aplikace XSL stylu XML dokument, musíme ve stylu používat elementy, které se vyskytnou ve výsledném XML dokumentu. Aby se navzájem nepomíchaly elementy z cílového dokumentu a ze stylu, používají se *jmenné prostory*. Před názvy všech elementů stylu, které souvisejí s XSL se předřadí `xsl:`, aby se odlišily od ostatních. Použití jmenného prostoru je potřeba deklarovat u kořenového elementu stylu.

```
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  ... jednotlivá transformační pravidla ...
</xsl:stylesheet>
```

Samotný styl se skládá převážně ze *šablon*. Pomocí šablony vybereme vždy určitý element a určíme, co se má s jeho obsahem udělat. Definice šablony se provádí pomocí elementu `xsl:template` a elementy, na které se použije, se určují pomocí vzoru zapsaného do atributu `match`. Pokud bychom chtěli, aby se element `par` pomocí stylu přeměnil na HTML element `p` pro odstavec, použijeme následující šablonu.

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

Element `xsl:apply-templates` XSL procesoru říká, aby obsah elementu `par` postoupil dalším šablonám, které mohou provádět další transformace.

Pokud nechceme, aby byl zpracován celý obsah elementu, můžeme u `xsl:apply-templates` použít atribut `select` a pomocí vzoru určit, které části elementu se mají zpracovat. Použití atributu `select` je velmi silný nástroj. Jeho vhodným použitím můžeme části dokumentu vyřadit ze zobrazení, nebo měnit pořadí elementů na výstupu. Například použitím následující šablony

```
<xsl:template match="document">
  <xsl:apply-templates select="chapter"/>
  <xsl:apply-templates select="appendix"/>
</xsl:template>
```

dojde nejprve ke zpracování kapitol a až potom dodatků nehledě na to, v jakém pořadí jsou ve zdrojovém XML dokumentu uvedeny. Atributy `match` i `select` berou jako argumenty hodnoty typu XPath — *variable reference* a *expr*, pro detaily viz specifikaci [XPath].

Nejčastěji používané vzory jsou následující.

- `/` — kořenový element.
- `.` a `..` — aktuální element a element rodiče.
- ◁*jméno*▷ — vzoru vyhovují všechny elementy daného jména.
- ◁*jméno1*▷ `|` ◁*jméno2*▷ — logická disjunkce.
- ◁*jméno1*▷ `/` ◁*jméno2*▷ — vzoru vyhovují elementy, které mají název ◁*jméno2*▷ a jsou přímo obsaženy v elementu ◁*jméno1*▷.
- ◁*jméno1*▷ `//` ◁*jméno2*▷ — vzoru vyhovují elementy, které mají název ◁*jméno2*▷ a v cestě od kořenového elementu k ◁*jméno2*▷ se vyskytuje element ◁*jméno1*▷. V podstatě jako předchozí případ, jen se nemusí jednat o přímého potomka.
- `text()` a `node()` — odpovídají textovému elementu a jakémukoliv elementu kromě kořenu a atributu.
- ◁*jméno*▷`[`◁*posice*▷`]` — odpovídá elementu se daného jména, který je na posici ◁*posice*▷ od svého rodiče. Například `par[1]` vrátí ten element `par`, který je první `par` element svého rodiče.
- `*` — libovolný element, například ◁*blah*▷`/*` znamená libovolný potomek elementu ◁*blah*▷.
- `@`◁*atribut*▷ — elementu odpovídá obsah atributu. Využívá se hodně v následujícím.
- `[`◁*podmínka*▷`]` — vzor je platný pouze tehdy, když platí podmínka. Jako podmínku můžeme použít test na výskyt elementu, atributu, hodnot atributu a další. Viz dále.

Na každý element dokumentu se aplikuje šablona, která má vyhovující vzor. Pokud je element možno zpracovat pomocí více šablon, má přednost ta poslední. Podle specifikace XSL by měl mít každý XSL procesor zabudovány šablony, které způsobí vypsání celého dokumentu — *implicitní pravidla*.

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

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

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

První šablona zajistí zpracování kořenového elementu dokumentu. Druhá šablona se postará o to, že budou zpracovány všechny elementy. Konečně poslední šablona do výsledného dokumentu vypíše textový obsah všech elementů na nejnižší úrovni. Díky tomu, že později uvedené šablony mají vyšší prioritu, neovlivníme další zpracování dokumentu, pokud je umístíme za začátek dokumentu. V třetí šabloně je použit výkonný element `xsl:value-of`, který do výstupního dokumentu vloží hodnotu argumentu určeného atributem `select`.

## Nevýkonné elementy

Pojmem nevýkonný element mám v tuto chvíli na mysli ty elementy, které přímo neřídí nebo neovlivňují proces transformace z hlediska postupování jednotlivých šablon.

Pomocí elementu `xsl:output` je možné definovat typ výstupních dat a jejich kódování. Mezi nejdůležitější atributy tohoto elementu patří method a encoding. Výstupní metoda může být „`xml`“, „`html`“, nebo „`text`“. Využití první je evidentní, druhou metodu lze použít například při presentaci — konversi do HTML. Třetí metoda je zcela obecná, pomocí něj lze vytvořit cokoliv, třeba zdrojový kód v TEXu.

Atribut encoding. Určuje kódování *výstupního dokumentu*, kódování vstupu je automaticky detekováno pomocí XML deklarace vstupního dokumentu. V úvahu připadají hodnoty „`utf-8`“, „`utf-16`“, standardy „`iso-8859-2`“ a tak dále.

Pokud je zdrojový kód šablony dlouhý, lze jej rozdělit do více zdrojových souborů a vložit jej pomocí elementu `<xsl:import href="`◁*url*▷`"/>`, kde atribut href ukazuje na vkládaný dokument.

## Další významné elementy

Tady se pokusím jmenovat ještě další zajímavé elementy, v žádném případě to ale nebude úplný popis jazyka, jeho specifikaci je možné nalézt na stránkách W3C, viz [XSLT99], [XSL99], přehlednou referenční příručku s příklady naleznete na [ZVON].

### Vyvolání šablony

Vyvolání šablony působí jako *podprogram*, pokud narazíte na problém duplicity kódu v různých šablonách, můžete definovat *pojmenovanou šablonu* pomocí elementu `<xsl:template name="`◁*name*▷`">`, kde hodnota ◁*name*▷ je jméno definované šablony. Pojmenovanou šablonu lze zavolat pomocí elementu `<xsl:call-template name="`◁*name*▷`"/>`, hodnota ◁*name*▷ je jméno již definované šablony.

### Podmíněné výrazy

Podmíněné výrazy fungují v XSL analogicky k ostatním programovacím jazykům. Podle testované hodnoty se volí mezi vykonanými příkazy, případně je možné dodat alternativní větev, která je provedena v případě, že žádná podmínka nebyla splněna.

Podmíněné zpracování je v šabloně uvozeno elementem `xsl:choose`. Tento element v sobě zahrnuje několik *podmínek* a *následujících akcí*. Podmínky jsou zpracovány postupně, pokud je nějaká z podmínek pravdivá, provede se příslušná akce. Podmíněné výrazy lze definovat pomocí elementu

`<xsl:when test="`◁*expr*▷`">`◁*tělo*▷`</xsl:when>`,

kde ◁*expr*▷ je testovaný výraz a ◁*tělo*▷ se provede, pokud je ◁*expr*▷ pravdivý.

Konversi hodnoty na *boolean* definuje [XPath], zjednodušeně lze říct, že prázdný řetězec je *false* a všechno ostatní je *true*.

Pře testování lze ve výrazech použít relační predikát rovnosti `=`, negaci representovanou funkcí `not`, dále konjunkci a disjunkci `and`, `or`. Samostatnou kapitolou je porovnávání elementů. Typickým problémem může být dotaz ve stylu: *„Jsem na prvním odstavci v elementu* `section`*?“*. Každý element v rámci XML stromu lze jednoznačně identifikovat. V XSL existuje funkce `generate-id(`◁*element*▷`)`, která pro daný ◁*element*▷ vrací jeho jednoznačný identifikátor. Potom lze shodu elementů testovat jako shodu jejich identifikátorů, například

```
generate-id (.) = generate-id (../par[1])
```

vyjadřuje podmínku, že aktuální element, je prvním `par` elementem ve svém rodičovském elementu.

Alternativní větev nemusí být uvnitř `xsl:choose` elementu vůbec uvedena a musí být nanejvýš jedna. Element nemá argumenty, příkazy uvedené v jeho těle se provedou, pokud nebyla splněna žádná podmínka z `xsl:when` elementů.

### Cykly

Iterovat přes jednotlivé uzly lze pomocí elementy `xsl:for-each`. Tvar elementu je `<xsl:for-each select="`◁*elements*▷`">`, kde ◁*elements*▷ specifikuje ty elementy, přes které se iteruje. Pro každý takový element je provedeno tělo cyklu.

Velmi často je potřeba iterovat přes *setříděné elementy*. Pomocí elementu `xsl:sort`, který lze použít nejen uvnitř `xsl:for-each`, ale i uvnitř `xsl:apply-templates` lze setřídit jednotlivé elementy. Přitom pokud uvádíme klausule `xsl:sort` za sebou, vyjadřujeme tím vlastně sekundární, terciární a další klíče.

V klausuli `xsl:sort` je možné určit, podle čeho se třídí (atribut select), implicitní hodnota je „`.`“. Déle je možné pomocí atributu order deklarovat buďto vzestupné — *ascending*, nebo sestupné — *descending*, třídění. Atribut `data-type` určuje, jakého charakteru jsou klíče, podle kterých se třídí, mohou to být buďto řetězce, nebo čísla. Při třídění řetězců jsou použit jazyk, který je buďto definován atributem lang nebo je jazyk rozeznán z nastavení prostředí — v UNIXu jsou to `locale`(1). Příklad iterace.

```
<xsl:for-each select="//node">
  <xsl:sort order="ascending" select="."/>
  <xsl:value-of select="."/>
</xsl:for-each>
```

### Módy šablon

Během zpracování šablony se lze dostat do situace, kdy by bylo potřeba zpracovat jeden element v různých kontextech. Typickým příkladem budiž element `footnote`, který má v dokumentu význam poznámky pod čarou. Jedním módem jeho zpracování je tisk odkazu na poznámku, druhým módem je tisk samotné poznámky.

Z tohoto důvodu XSL definuje *módy šablon*. Atribut mode lze uvést u elementů `xsl:call-template`, `xsl:apply-templates` a `xsl:template`. Při specifikaci módu šablony je daná šablona postoupena, právě když element, který ji vyžádá, ji vyžádá v daném módu. Mějme pojmenované šablony.

```
<xsl:template name="blah" mode="první">
  ... tělo první šablony ...
</xsl:template>

<xsl:template name="blah" mode="druhý">
  ... tělo druhé šablony ...
</xsl:template>
```

Při volání `<xsl:call-template name="blah" mode="druhý"/>` je postoupena druhá šablona z předcházejících dvou.

### Číslování

Další užitečným elementem je automatické číslování. Lépe bychom měli spíš hovořit o *čítačích*. Automatické číslování najde využití při číslování odkazů, kapitol a podobně. XSL definuje dostatečně silnou metodu číslování, která umožňuje zahrnout i hierarchické čítače vhodné pro označování podkapitol a podobně.

Element `xsl:number` slouží pro vkládání čísel do vstupního dokumentu. Mezi nejdůležitější atributy elementu `xsl:number` patří následující.

- level — definuje, na kterých hladinách se má číslování uvažovat, může nabývat hodnot „`single`“, „`multiple`“ a „`any`“.
- count — představuje vzor, který specifikuje elementy, které jsou spočítány na úrovni dané atributem level. Například, pokud je úroveň „`any`“ a count má hodnoty elementu `blah`, pak jsou sečteny všechny elementy `blah` předcházející aktuálnímu elementu.
- from — počáteční hodnota počítadla.
- value — přímo uvedené číslo, které se má vytisknout.
- format — specifikuje formát, v jakém se má výsledek presentovat. Formát je řetězec, v němž se vyskytují speciální symboly
  - `0*1`, to jest jednička prefixovaná nulami — výstupem je číslo, které má v případě uvedených nul, pevný počet cifer,
  - `A` — sekvence `A`, `B`,`CD`, ...,
  - `a` — sekvence `a`, `b`, `c`, `d`, ...,
  - `I` — sekvence `I`, `II`, `III`, `IV`, ...,
  - `i` — sekvence `i`, `ii`, `iii`, `iv`, ....

Element `xsl:number` je velmi mocný, na druhou stranu je jeho použití dost komplikované.

### Proměnné

XSL rozlišuje definici *proměnné* a *parametru*, který lze předat šabloně. Proměnnou šabloně předávat *nelze*. Proměnnou lze definovat pomocí klausule `xsl:variable`, kde atribut name definuje jméno proměnné a musí být vždy uveden. Na proměnnou je možné odkazovat se ve výrazech přes symbol $ podobně, jako například na skalár v PERLu nebo v PHP. Hodnotu proměnné lze definovat buďto atributem select. Nebo v těle elementu `xsl:variable`.

Pomocí elementu `xsl:param` lze definovat parametr, který je možné předat šabloně. Syntaxe je shodná jako s definicí proměnné, parametr má však význam definovat jen uvnitř šablony, nikoliv vně — globálně. Pokud je při definici parametru uveden atribut select, nebo hodnota v těle, je tato hodnota brána jako implicitní.

Skutečnou hodnotu parametru lze předat pomocí elementu `xsl:with-param`, který uvedeme uvnitř volajícího elementu, tedy například `xsl:apply-templates`, nebo `xsl:call-template`. Syntaxe `xsl:with-param` je opět shodná se syntaxí pro `xsl:variable`, musíme definovat název předávaného argumentu a předávanou hodnotu.

## Reference


[XSLT99] W3C: *eXtensible Stylesheet Language Transformation*,
state: 16 November 1999, recommendation,
URL: `http://www.w3.org/TR/1999/REC-xslt-19991116`

[XSL99] W3C: *eXtensible Stylesheet Language*,
state: 21 April 1999, working draft,
URL: `http://www.w3.org/TR/WD-xsl`

[XPath] W3C: *XPath*,
state: 14 February 2001, working draft,
URL: `http://www.w3.org/TR/xpath20`

[ZVON] URL: `http://www.zvon.org`