



evropský  
sociální  
fond v ČR



EVROPSKÁ UNIE



MINISTERSTVO ŠKOLSTVÍ,  
MLÁDEŽE A TĚLOVÝCHOVY



OP Vzdělávání  
pro konkurenceschopnost



TECHNICKÁ  
UNIVERZITA  
V LIBERCI  
[www.tul.cz](http://www.tul.cz)

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Realizováno za finanční podpory ESF a státního rozpočtu ČR  
v rámci v projektu *Zkvalitnění a rozšíření možností studia  
na TUL pro studenty se SVP* reg. č. CZ.1.07/2.2.00/29.0011

# Jazyky pro popis dat

Pavel Satrapa

[pavel.satrapa@tul.cz](mailto:pavel.satrapa@tul.cz)

# Literatura

- Aaron Skonnard, Martin Gudgin:  
*XML pohotová referenční příručka*  
Grada, 2005
- stránky Jiřího Koska  
*[www.kosek.cz](http://www.kosek.cz)*
- stránky WWW konsorcia  
*[www.w3.org](http://www.w3.org)*



# Historie



# Datové soubory

- **binární**

- Pascal: **file of** *Zamestnanec*
- rychlé zpracování (přímý přenos do/z paměti)
- srozumitelné jen pro zasvěcené programy

- **textové**

- zápis/čtení vyžaduje konverzi (i dost složitou)
- (víceméně) srozumitelné, snadno editovatelné
- ideální pro přenositelnost mezi programy

# Tradiční textové soubory

- **pevná struktura**

- např. */etc/passwd* – pevný význam položek i oddělovač
- `satrapa:x:500:500:Pavel Satrapa:/home/satrapa:/bin/zsh`
- úsporné, ale nepřehledné

- **volná struktura**

- syntaktická pravidla umožňují jistou volnost v pořadí a/nebo formátu souboru
- např. CSS definice

# Příklad: CSS definice

```
p {  
  margin: 0,5em 0;  
}  
  
h1, h2, h3 {  
  font-family: sans-serif;  
  font-weight: bold;  
  color: blue;  
}
```

# Formalizace – značkovací jazyky

- historie: **typografické jazyky**
  - cíl: popsat pomocí příkazů přímo v textu požadovaný vzhled, resp. charakterizovat části textu
  - \*roff, T<sub>E</sub>X
  - zkušenost: výhodný je strukturální popis
- nověji: **jazyky popisující strukturu dat**
  - cíl: jednotný mechanismus, obecné nástroje pro zacházení s nimi (kontroly, konverze apod.)

# SGML (1)

- **1969: Generalized Markup Language (GML)**
  - vyvinulo IBM pro práci s dokumenty
  - editace, formátování, vyhledávání informací
- **1986: Standard Generalized Markup Language (SGML)**
  - přijato ISO jako mezinárodní standard
  - metajazyk: jazyk pro definici jazyků
  - popisuje struktury a významy, nikoli vzhled



# SGML (2)

- **aplikace SGML:**
  - HTML
  - DocBook
- **vzhled:**
  - Document Style Semantics and Specification Language (DSSSL)
  - nověji lze i CSS
- příliš složité – málo implementací



**XML**



# XML

- **eXtensible Markup Language**
- přijalo WWW konsorcium (1998)
- podmnožina SGML – jednodušší
  - míněno jako adaptace SGML pro web
- snazší implementace – rychlé rozšíření
- **metajazyk**, definuje
  - základní syntaktická pravidla dokumentů
  - nástroje pro popis prvků jazyka a jejich vztahů

# Příklad XML dokumentu

`<?xml version="1.0" encoding="ISO-8859-2"?>` ← *prolog (deklarace jazyka a kódování)*

`<diskoteka>` ← *kořenový prvek*

`<disk id="disk1">`

`<interpret>Divokej Bill</interpret>`

`<nazev>Propustka do pekel</nazev>` ← *běžné prvky*

`</disk>`

`<disk id="disk2">`

`<interpret>AC/DC</interpret>`

`<nazev>Highway to Hell</nazev>`

`</disk>`

`</diskoteka>`

# Možnosti a omezení XML

- XML definuje **strukturu dokumentu**
- obecné XML nástroje umožňují
  - zkontrolovat dokument
  - transformovat jej do jiného formátu (XML i ne-XML)
- XML **nic neříká o sémantice**, významy prvků musí znát aplikace
- „program X podporuje XML“ neznamená „program X rozumí jakémukoli XML dokumentu“

# XML prolog

- nepovinný
- `<?xml version="1.0" ... ?>`
- nepovinné součásti:
  - `encoding="název"` – použité kódování, implicitně UTF-8
  - `standalone="yes|no"` – zda je dokument soběstačný (neodkazuje se na externí definice)

# Prvky a značky

- dokument je strukturován rozdělením do prvků
- značka (tag) je zápisem prvku
- zápis: `<jméno>...</jméno>`
- ve jménech se rozlišují malá/velká písmena
- mezi značkami je obsah prvku (text, další prvky...)
- lze vnořovat: `<a>...<b>...</b>...</a>`
- nesmí se překřížit: `<a>...<b>...</a>...</b>` je chyba

# Prázdné prvky

- nemá-li prvek obsah, je nutno ho ihned ukončit  
`<prvek></prvek>`
- nebo explicitně vyjádřit, že je prázdný  
`<prvek/>`  
(v XHTML se doporučuje předřadit mezeru kvůli starým klientům `<prvek />`)
- obě varianty jsou ekvivalentní



# Kořenový prvek

- celý dokument je vždy zabalen do jednoho prvku
- označován jako kořenový (též dokumentový) prvek
- pro daný jazyk je pevně definován
- např. pro XHTML se jedná o prvek `html`, u DocBooku `article` apod.

# Atributy

- doplňkové informace o prvku
  - interní informace (identifikátor)
  - ovlivnění vzhledu (CSS styl, třída apod.)
- zapisují se do zahajující značky
  - `<div class="poznamka">...</div>`
- atribut musí mít hodnotu (checked="checked")
- hodnota musí být uzavřena do uvozovek  
"..." nebo '...'

# Atribut nebo vnořený prvek?

<cd id="disk1">

<nazev>Strach</nazev>

</cd>

<cd>

<id>disk1</id>

<nazev>Strach</nazev>

</cd>

- neexistuje striktní pravidlo, rozhoduje cit
- data, která jsou součástí popisovaných objektů, raději jako prvky
- data popisná/organizační lépe do atributů
- hierarchie je mezi prvky, nikoli atributy

# Entity

- jak vyjádřit speciální znaky (např. <)?
- pomocí entit ve tvaru  
*&jméno;*
- standardní entity pro speciální znaky:
  - *&lt;* <
  - *&gt;* >
  - *&amp;* &
  - *&quot;* “
  - *&apos;* ’

# Komentáře

- ignorovány při zpracování (oficiálně nejsou považovány za součást textu dokumentu)
- tvar:  
`<!-- text komentáře -->`
- text komentáře nesmí obsahovat --, nelze vnořovat
- dají se použít k vypuštění části textu:  
`<!--  
<p>Ignorovaný <em>text</em>.</p>  
-->`

# Instrukce pro zpracování

- doplňkové informace pro obslužný program

- tvar

*<?cíl data?>*

*cíl* identifikuje, komu/k čemu jsou informace určeny,  
*data* často sekvence *jméno*=“*hodnota*”; např.

*<?xml-stylesheet href=“styl.css” type=“text/css”?>*

- nejsou považovány za součást textu dokumentu, ale musí být předány zpracovávajícímu programu

# CDATA

- někdy se hodí, aby určitou část textu XML procesor neanalyzoval, nehledal v ní prvky apod., ale opsal ji beze změny
- toto chování má obsah sekce CDATA  
`<![CDATA[neinterpretovaný text]]>`
- vhodný např. pro ukázky XML kódu v textech o XML



INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Realizováno za finanční podpory ESF  
a státního rozpočtu ČR v rámci v projektu  
*Zkvalitnění a rozšíření možností studia na TUL  
pro studenty se SVP*  
reg. č. CZ.1.07/2.2.00/29.0011





evropský  
sociální  
fond v ČR



EVROPSKÁ UNIE



MINISTERSTVO ŠKOLSTVÍ,  
MLÁDEŽE A TĚLOVÝCHOVY



OP Vzdělávání  
pro konkurenceschopnost



TECHNICKÁ  
UNIVERZITA  
V LIBERCI  
[www.tul.cz](http://www.tul.cz)

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Realizováno za finanční podpory ESF a státního rozpočtu ČR  
v rámci v projektu *Zkvalitnění a rozšíření možností studia  
na TUL pro studenty se SVP* reg. č. CZ.1.07/2.2.00/29.0011

# Správnost XML dokumentu

# Správně strukturovaný dokument

- v originále **well-formed**
- dokument dodržuje syntaktická pravidla XML:
  - má jeden kořenový prvek, prvky jsou správně ukončeny (vnořeny)
  - správný zápis prvků a entit, hodnoty atributů jsou v uvozovkách atd.
  - správně vytvořené jsou i vložené prvky
- nutný předpoklad pro zpracování v XML nástrojích
- týká se formy, nikoli obsahu/struktury

Historie

# Správný (validní) dokument

- v originále **valid**
- musí platit:
  - dokument je správně strukturovaný
  - struktura jeho prvků odpovídá definici jazyka (obsahuje přípustné prvky v přípustných vztazích)
- zaručuje korektní zpracování v aplikacích podporujících daný jazyk
- jak definovat jazyk?



# Definicje języka



# Document Type Definition (DTD)

- definuje jazyk dokumentu
  - jaké existují prvky
  - co mohou obsahovat (a tedy jaká jsou pravidla pro jejich vzájemné vnořování)
  - jaké mají atributy
- zavádí obecnou strukturu
  - neumí datové typy – nelze např. omezit obsah prvku či hodnotu atributu na celá čísla od 1 do 100

# Definice prvku

- `<!ELEMENT jméno obsah>`
- *jméno* určuje jméno prvku, musí být jednoznačné
- *obsah* omezuje, co prvek smí a nesmí obsahovat
- dva typy textových obsahů:
  - PCDATA (Parsed Character Data) – text analyzovaný procesorem, rozpoznávají se prvky, expandují entity,...
  - CDATA (Character Data) – text není analyzován, bere se jako konstanta

# Jednoduché obsahy

- **EMPTY** – prvek je prázdný  
<!ELEMENT br EMPTY>
- **ANY** – prvek může mít libovolný analyzovatelný obsah; vzdáváme se přísnější kontroly
- **(#PCDATA)** – prvek obsahuje text  
<!ELEMENT den (#PCDATA)>

# Prvek jako obsah

- (prvek) – daný prvek
- (prvek1,prvek2,...) – prvky v daném pořadí
- (prvek\*) – libovolný počet těchto prvků
- (prvek+) – alespoň jeden tento prvek
- (prvek?) – nepovinný výskyt daného prvku
- (prvek1|prvek2) – jeden nebo druhý
- pomocí závorek lze operátory aplikovat na více prvků



# Příklad: Datum

<!ELEMENT datum (den,mesic,rok)>

<!ELEMENT den (#PCDATA)>

<!ELEMENT mesic (#PCDATA)>

<!ELEMENT rok (#PCDATA)>

Odpovídající XML:

```
<datum>
  <den>17</den>
  <mesic>10</mesic>
  <rok>2006</rok>
</datum>
```

Nevalidní (špatné pořadí):

```
<datum>
  <rok>2006</rok>
  <mesic>10</mesic>
  <den>17</den>
</datum>
```

# Příklad: Plné jméno

```
<!ELEMENT plnejmeno (titul*,krestni,dalsikrestni*,prijmeni+,titul*)>  
<!ELEMENT titul (#PCDATA)>  
<!ELEMENT krestni (#PCDATA)>  
<!ELEMENT dalsikrestni (#PCDATA)>  
<!ELEMENT prijmeni (#PCDATA)>
```

```
<plnejmeno>  
  <krestni>Jan</krestni>  
  <prijmeni>Nový</prijmeni>  
</plnejmeno>
```

```
<plnejmeno>  
  <titul>Ing.</titul>  
  <krestni>Emanuel</krestni>  
  <dalsikrestni>Ivo</dalsikrestni>  
  <dalsikrestni>Jan</dalsikrestni>  
  <prijmeni>Kyselý</prijmeni>  
</plnejmeno>
```

# Smíšený obsah

- text i vnořené prvky  
jako jednu z variant v „nebo“ uvést #PCDATA
- např. nadpis připouštějící text a zvýraznění  
(vnořený prvek em), to může být víceúrovňové  
<!ELEMENT nadpis (#PCDATA|em)\*>  
<!ELEMENT em (#PCDATA|em)\*>

# Definice atributů

- *<!ATTLIST prvek jméno typ implicit\_hodnota>*
- závěrečná trojice se opakuje pro každý atribut, nebo lze použít několik *<!ATTLIST ...>*
- *prvek* je jméno prvku, jehož atributy definujeme
- *jméno* určuje jméno atributu
- *typ* jeho typ (charakter, nikoli datový typ)
- *implicit\_hodnota* poskytuje informace o hodnotě

# Typy atributů (1)

- **CDATA** – libovolný (nezpracovávaný) text
- **(hod1|hod2|...)** – výčet přípustných hodnot
- **ID** – jednoznačný identifikátor (definice ident.)  
omezení: XML identifikátory nesmí začínat číslicí
- **IDREF** – identifikátor jiného prvku (odkaz na něj)
- **IDREFS** – seznam identifikátorů jiných prvků,  
oddělovány mezerami

# Typy atributů (2)

- **NMTOKEN** – platné XML jméno (písmena, číslice, -, \_, ., :)
- **NMTOKENS** – seznam jmen oddělených mezerami
- **ENTITY, ENTITIES** – jméno entity, seznam entit
- **NOTATION** – jméno notace definované pomocí `<!NOTATION...>`, nepoužívá se
- **xml:** – předdefinovaná XML hodnota

# Implicitní hodnota

- “hodnota” – konkrétní hodnota
- #REQUIRED – atribut je povinný
- #IMPLIED – atribut lze vynechat, implicitní hodnota není definována
- #FIXED “hodnota” – hodnota je neměnná

# Příklad: Telefonní seznam

- kořenovým prvkem je **seznam**
- obsahuje libovolné množství prvků **osoba**
- osoba obsahuje **jmeno** a alespoň jedno **cislo**, má také povinný atribut **id** obsahující jednoznačný identifikátor
- **cislo** má nepovinný atribut **typ** s hodnotami „mobil“, „stabil“ nebo „skype“



# DTD pro telefonní seznam

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

```
<!ELEMENT seznam (osoba*)>
```

```
<!ELEMENT osoba (jmeno,cislo+)>
```

```
<!ATTLIST osoba  
    id ID #REQUIRED
```

```
>
```

```
<!ELEMENT jmeno (#PCDATA)>
```

```
<!ELEMENT cislo (#PCDATA)>
```

```
<!ATTLIST cislo  
    typ (mobil|stabil|skype) #IMPLIED
```

```
>
```

# Telefonní seznam – příklad

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

```
<seznam>
```

```
<osoba id="elib">
```

```
  <jmeno>Eleonora Líbezná</jmeno>
```

```
  <cislo>123 456 789</cislo>
```

```
</osoba>
```

```
<osoba id="mojl">
```

```
  <jmeno>Mojmír Luzný</jmeno>
```

```
  <cislo typ="mobil">606 707 808</cislo>
```

```
</osoba>
```

```
</seznam>
```

# Definice entit

- **interní entity:**

`<!ENTITY jméno "hodnota">`

`<!ENTITY tul "Technická univerzita v Liberci">`

použití v XML: Naší školou je `&tul;`.

- **externí entity (textové):**

`<!ENTITY jméno SYSTEM "lokátor">`

`<!ENTITY kontakt SYSTEM "doc/kontakt.xml">`

`<!ENTITY jméno PUBLIC`

`"veřejný identifikátor" "lokátor">`

# Binární entity

- `<!ENTITY jméno SYSTEM “lokátor”  
                  NDATA notace>`
- notace určuje obslužný program  
`<!NOTATION notace SYSTEM “program”>`
- např.:  
`<!ENTITY logo SYSTEM “logo.gif” NDATA gif>`  
`<!NOTATION gif SYSTEM “c:/graphic/irfanview/i_view.exe”>`
- v podstatě se nepoužívá

# Parametrické entity

- zkratky používané přímo v DTD
- `<!ENTITY % jméno "hodnota">`  
použití v DTD: `%jméno;`
- příklad – standardní atributy:  
`<!ENTITY % stdattr "id ID #IMPLIED  
style CDATA #IMPLIED">`  
`<!ELEMENT nadpis (#PCDATA)>`  
`<!ATTLIST nadpis  
%stdattr;  
uroven (kapitola|cast|podcast) #IMPLIED  
>`

# Připojení DTD ke XML

- nejčastější – odkaz na externí soubor:  
`<?xml version="1.0"?>`  
`<!DOCTYPE seznam SYSTEM "telsez.dtd">`
- může být uvedeno i přímo v XML souboru:  
`<?xml version="1.0"?>`  
`<!DOCTYPE seznam [  
 <!ELEMENT seznam (osoba*)>`  
 ...  
`]>`

# Klady DTD

- **nejstarší definiční jazyk**
  - široce podporován
  - nástroje jsou běžně dostupné
- **jednoduché**
  - v podstatě tři konstrukce:  
ELEMENT, ATTLIST, ENTITY

# Nedostatky DTD

- **neumí datové typy**
  - velmi omezené možnosti pro definici obsahu prvků a hodnot atributů
- **nepodporuje jmenné prostory**
  - problém při kombinování několika DTD
- **nezvyklá syntaxe**
  - formálně je XML
  - obsahem je vůbec nepřipomíná





INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Realizováno za finanční podpory ESF a státního rozpočtu ČR  
v rámci v projektu *Zkvalitnění a rozšíření možností studia  
na TUL pro studenty se SVP* reg. č. CZ.1.07/2.2.00/29.0011

# Jmenné prostory

# Jmenné prostory

- počítá se s modularitou XML
  - jeden dokument může používat značky z několika specializovaných jazyků (např. pro matematické a chemické vzorce)
- problém: konfliktní jména prvků
- řešení: jmenné prostory
  - prostor zahrnuje příbuzné prvky (z jednoho modulu)
  - kompletní identifikace: jmenný prostor + prvek

# Definice (1)

## ■ **jmenný prostor**

- je identifikován lokátorem (URI) = jméno prostoru
- zahrnuje jména prvků a atributů
- jméno (URI) příliš dlouhé, bývá definována zkratka (prefix) pro zařazení identifikátoru do jmenného prostoru
- URI musí být absolutní

## ■ **expandované jméno**

- pár zahrnující jméno jmenného prostoru a lokální jméno

# Definice (2)

- **kvalifikované jméno**

- jméno, které je interpretováno z hlediska příslušnosti ke jmennému prostoru
- jména prvků v XML dokumentu
- bez prefixu (např. **h1**) – náleží do implicitního jmenného prostoru, závisí na kontextu
- s prefixem (např. **html:h1**) – jmenný prostor je určen prefixem

# Deklarace jmenného prostoru

- speciálním atributem prvku, platí pro daný prvek a jeho potomky (nejčastěji kořenový prvek)
- ***xmlns:prefix="URI"***  
deklaruje jmenný prostor označený  
v kvalifikovaných jménech daným *prefixem*
- ***xmlns="URI"***  
deklaruje implicitní jmenný prostor, platí pro jména  
prvků bez prefixu; ne pro atributy – atribut bez  
prefixu je ve stejném prostoru jako jeho prvek

# Příklad dokumentu

```
<h:html xmlns:xdc="http://www.xml.com/books"
        xmlns:h="http://www.w3.org/HTML/1998/html4">
  <h:head><h:title>Book Review</h:title></h:head>
  <h:body>
    <xdc:bookreview>
      <xdc:title h:style="color: red;">XML: A Primer</xdc:title>
      <h:p>Autor: <xdc:author>Simon St. Laurent</xdc:author></h:p>
      <h:p>Vydáno: <xdc:date>1998/01</xdc:date></h:p>
      <h:p>Dobrý úvod do XML.</h:p>
    </xdc:bookreview>
  </h:body>
</h:html>
```

# Implicitní prostor

```
<html xmlns:xdc="http://www.xml.com/books"
      xmlns="http://www.w3.org/HTML/1998/html4">
  <head><title>Book Review</title></head>
  <body>
    <xdc:bookreview>
      <xdc:title>XML: A Primer</xdc:title>
      <p>Autor: <xdc:author>Simon St. Laurent</xdc:author></p>
      <p>Vydáno: <xdc:date>1998/01</xdc:date></p>
      <p>Dobrý úvod do XML.</p>
    </xdc:bookreview>
  </body>
</html>
```

# Rezervované prefixy

- **xml**

- vázán na <http://www.w3.org/XML/1998/namespace>
- nesmí být přiřazen jinému URI a toto URI nesmí být přiřazeno jinému prefixu

- **xmlns**

- vázán na <http://www.w3.org/2000/xmlns/>
- nesmí být deklarován

- všechny prefixy začínající **xml...**





# **XML Schema**



# XML Schema

- označováno **XSD** (XML Schema Definition)
- specifikace vyvíjená WWW konsorciem
- jmenný prostor  
`xmlns:xsd="http://www.w3.org/2001/XMLSchema"`
- výrazově dost silné, ale kritizováno za složitost
- řada předdefinovaných typů
- hlavní síla ve schopnosti definovat si vlastní typy

# Jednoduché datové typy (1)

- celkově přes 50
- slouží **pro koncové hodnoty** (žádné vnořené prvky)
- **celá čísla:** integer, positiveinteger, unsignedint (32 bitů), short (16 bitů), byte (se znaménkem),...
- **necelá čísla:** decimal (s desetinnou částí), float (32 bitů), double (64 bitů),...
- **časy:** date (2006-10-23), time (14:20:00), duration (délka časového intervalu, P2H15M),...

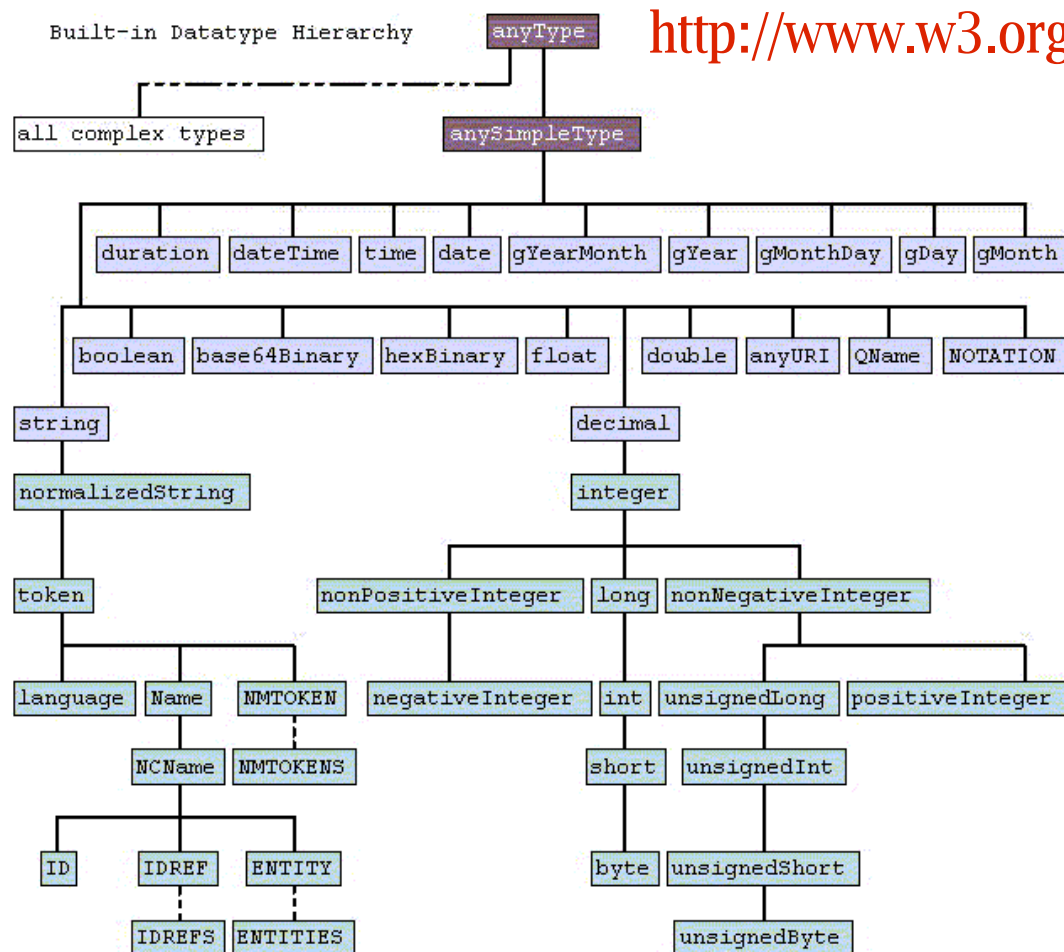
# Jednoduché datové typy (2)

- **řetězce:** string, normalizedString (tabulátory a konce řádků nahrazeny mezerami), token (navíc sloučeny skupiny mezer a odstraněny mezery na začátku/konci řetězce)
- **z DTD:** ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS, NOTATION

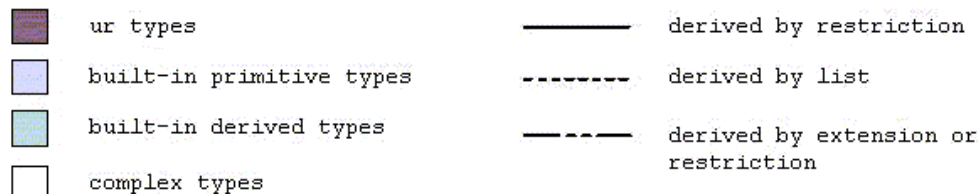
# Jednoduché datové typy (3)

- **z XML:** Name (à la jméno prvku/atributu), NCName (jméno bez dvojtečky), language (kód jazyka)
- **binární:** hexBinary (bajt=2 šestnáctkové číslice), base64Binary (data v MIME kódování BASE64)
- **z webu:** anyURI (lib. URI adresa, absolutní i relativní)

# Hierarchie typů



<http://www.w3.org/TR/xmlschema-2/#built-in-datatypes>



# Odvozování nových typů

- **restrikce (restriction):**
  - omezí se možné hodnoty
- **seznam (list):**
  - vytváření seznamů hodnot daného typu
- **spojení (union):**
  - spojuje několik typů
- **rozšíření (extension):**
  - přidává děti a/nebo atributy (pro složené typy)

# Restrikce řetězců (1)

- udává se výchozí typ a omezení jeho hodnot
- délka (**length**) udává počet znaků řetězce
- příklad: telefonní číslo ČR jako řetězec 9 znaků

```
<xsd:simpleType name="telefonType">  
  <xsd:restriction base="xsd:token">  
    <xsd:length value="9"/>  
  </xsd:restriction>  
</xsd:simpleType>
```



# Restrikce řetězců (2)

- rozmezí délek: **minLength** a **maxLength**

- příklad: 3 až 5písmenná zkratka

```
<xsd:simpleType name="zkratkaType">  
  <xsd:restriction base="xsd:string">  
    <xsd:minLength value="3"/>  
    <xsd:maxLength value="5"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

# Restrikce řetězců (3)

- výčet hodnot: **enumeration**

- příklad: fakulty naší univerzity

```
<xsd:simpleType name="fakultaTULType">  
  <xsd:restriction base="xsd:string">  
    <xsd:enumeration value="FS"/>  
    <xsd:enumeration value="FT"/>  
    <xsd:enumeration value="FP"/>  
    <xsd:enumeration value="EF"/>  
    <xsd:enumeration value="FM"/>  
    <xsd:enumeration value="FA"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

# Restrikce řetězců (4)

- regulární výrazy: **pattern**
- syntaxe podle jazyka Perl
- příklad: telefon – 3 skupiny 3 číslic, lze s mezerami

```
<xsd:simpleType name="telefonType">  
  <xsd:restriction base="xsd:token">  
    <xsd:pattern value="\d{3} ?\d{3} ?\d{3}"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

# Restrikce čísel

- rozmezí hodnot: **minInclusive**, **minExclusive**, **maxInclusive**, **maxExclusive**
- počet platných číslic: **totalDigits**
- přesnost: **fractionDigits**

# Příklad číselné restrikce

- cena, do 1 000 000, uvádí se na 2 desetinná místa
- ```
<xsd:simpleType name="cenaType">  
  <xsd:restriction base="xsd:decimal">  
    <xsd:minInclusive value="0"/>  
    <xsd:maxExclusive value="1000000"/>  
    <xsd:fractionDigits value="2"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

# Použití typu v definici prvku

- základní – pojmenovaný typ:

```
<xsd:element name="telefon" type="telefonType"/>
```

- lze i anonymně:

```
<xsd:element name="telefon">
```

```
  <xsd:simpleType>
```

```
    <xsd:restriction base="xsd:token">
```

```
      <xsd:pattern value="\d{3} ?\d{3} ?\d{3}"/>
```

```
    </xsd:restriction>
```

```
  </xsd:simpleType>
```

```
</xsd:element>
```

# Posuzování hodnot

- normalizace
  - nahrazení tabulátorů a konců řádků mezerami
  - sloučení skupin mezer do jedné
  - odstranění mezer na začátku a konci hodnoty
  - neprovádí se pro string a normalizedString a jejich odvozeniny
- po normalizaci se převede na interní hodnotu
  - další omezení se ověřují pro ni



# **Složené datové typy**





# Složené datové typy

- popisují strukturu dokumentu
- definují **vzájemné vztahy jednotlivých prvků**
- tři základní druhy:
  - **sequence** – prvky se musí vyskytovat v daném pořadí
  - **all** – prvky se mohou vyskytovat v libovolném pořadí
  - **choice** – vyskytuje se jeden z uvedených prvků
- lze vnořovat (kromě all)

# sequence

- prvky musí být obsaženy v daném pořadí
- počet výskytů jim lze předepsat atributy **minOccurs** a **maxOccurs**
  - hodnotou číslo od 0 nebo unbounded (neomezeně)

# Příklad sekvence

- prvek kontakt obsahující jméno, nepovinný mail a alespoň jeden telefon

```
<xsd:element name="kontakt">
```

```
  <xsd:complexType>
```

```
    <xsd:sequence>
```

```
      <xsd:element name="jmeno" type="xsd:string"/>
```

```
      <xsd:element name="email" type="xsd:string" minOccurs="0"/>
```

```
      <xsd:element name="telefon" type="telefonType"  
        minOccurs="1" maxOccurs="unbounded"/>
```

```
    </xsd:sequence>
```

```
  </xsd:complexType>
```

```
</xsd:element>
```

# choice

- vyskytuje se právě jeden z uvedených prvků
- příklad: osoba s rodným číslem nebo číslem pasu

```
<xsd:element name="osoba">
```

```
  <xsd:complexType>
```

```
    <xsd:sequence>
```

```
      <xsd:element name="jmeno" type="xsd:string"/>
```

```
      <xsd:choice>
```

```
        <xsd:element name="rodne" type="xsd:string"/>
```

```
        <xsd:element name="pas" type="xsd:string"/>
```

```
      </xsd:choice>
```

```
    </xsd:sequence>
```

```
  </xsd:complexType>
```

```
</xsd:element>
```

# all

- prvky se mohou vyskytovat v libovolném pořadí
- maxOccurs dětí musí být jedna
  - de facto má smysl uvažovat jen o minOccurs="0" pro nepovinné
- lze použít jen přímo uvnitř složeného typu
- **smí obsahovat jen definice prvků (element), nikoli složené typy**

# Volnější obsah

- diskoteka obsahuje libovolný počet cd a dvd v libovolném pořadí

```
<xsd:element name="diskoteka">  
  <xsd:complexType>  
    <xsd:sequence>  
      <xsd:choice minOccurs=0 maxOccurs=unbounded>  
        <xsd:element name="cd" type="cdType"/>  
        <xsd:element name="dvd" type="dvdType"/>  
      </xsd:choice>  
    </xsd:sequence>  
  </xsd:complexType>  
</xsd:element>
```

# Smíšený obsah

- atributem `mixed="true"` u prvku `complexType`
- povoluje text mezi definovanými prvky (přísnější než v DTD)
- chceme-li skutečně volný obsah
  - `<xsd:complexType mixed="true">`
  - `<xsd:choice minOccurs="0" maxOccurs="unbounded">`
  - povolené prvky*
  - `</xsd:choice>`
  - `</xsd:complexType>`

# Prvky

- prvek **element**; atributy:
  - **name** – jméno prvku, povinný
  - **type** – typ prvku (co je jeho obsahem); pokud chybí, může být typ obsahu definován přímo na místě jako obsah prvku element
  - **default** – implicitní obsah prvku
  - **fixed** – pevně definovaný obsah prvku



# Atributy

- prvek **attribute**; jeho atributy:
  - **name** – jméno atributu, povinný
  - **type** – typ atributu (omezuje hodnotu), povinný
  - **use** – hodnota “required” stanoví povinný atribut
  - **default** – implicitní hodnota
- **musí být na konci (za prvky)**
- do složených typů se přidává normálně
- **<xsd:attribute name=“id“ type=“xsd:ID“ use=“required“/>**

# Rozšíření jednoduchého typu

- přidání atributu je rozšířením jednoduchého typu
- příklad: odkaz

```
<xsd:element name="odkaz">  
  <xsd:complexType>  
    <xsd:simpleContent>  
      <xsd:extension base="xsd:string">  
        <xsd:attribute name="url" type="xsd:anyURI"/>  
      </xsd:extension>  
    </xsd:simpleContent>  
  </xsd:complexType>  
</xsd:element>
```



evropský  
sociální  
fond v ČR



EVROPSKÁ UNIE



MINISTERSTVO ŠKOLSTVÍ,  
MLÁDEŽE A TĚLOVÝCHOVY



OP Vzdělávání  
pro konkurenceschopnost



TECHNICKÁ  
UNIVERZITA  
V LIBERCI  
[www.tul.cz](http://www.tul.cz)

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Realizováno za finanční podpory ESF a státního rozpočtu ČR  
v rámci v projektu *Zkvalitnění a rozšíření možností studia  
na TUL pro studenty se SVP* reg. č. CZ.1.07/2.2.00/29.0011

# Vytvoření schématu

# Obal schématu

- celé schéma obaluje prvek **schema** z jmenného prostoru přiděleného XML Schema:

**<xsd:schema**

**xmlns:xsd="http://www.w3.org/2001/XMLSchema">**

... definice schématu ...

**</xsd:schema>**

- atributy umožňují určit různé vlastnosti

# Připojení schématu k dokumentu

- pomocí atributů z prostoru <http://www.w3.org/2001/XMLSchema-instance>
- obvyklým prefixem je xsi
- atribut **noNamespaceSchemaLocation** pokud nepoužíváme jmenné prostory, hodnotou lokátor

```
<mujdokument  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:noNamespaceSchemaLocation="schemadokumentu.xsd">  
...  
</mujdokument>
```

# Připojení se jmenným prostorem

- atribut **schemaLocation**
- hodnotou dvojice (i několik)  
*URI\_prostoru lokátor\_schématu*

```
<book isbn="0836217462"  
  xmlns="http://example.org/ns/books/"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation=  
    "http://example.org/ns/books/ file:library.xsd">
```

...

```
</book>
```

# Jak vytvářet schéma?

- obvyklé jsou tři základní metody
  - matrjoška
  - plátkování
  - slepý Benátčan
- volba závisí především na složitosti a charakteru popisovaného jazyka
- lze kombinovat (ale s mírou)

# Příklad: ceník zboží

- poslouží k ilustraci jednotlivých přístupů
- definice XML prvku **zbozi**, který má mít při použití následující tvar:

```
<zbozi kod="1267">  
  <nazev>Žízeň Killer – voda 1,5 l</nazev>  
  <cena>8,50</cena>  
</zbozi>
```



# Matrjoška

- definice prvků se do sebe ve schématu vkládají přímo, stejně jako prvky v dokumentu
- na nejvyšší úrovni vždy jen jeden prvek
- krátké a kompaktní schéma
- pro složitější jazyky nepřehledné
- nelze opakovaně využívat dílčí definice

# Matrjoška – příklad

```
<xsd:element name="zbozi" maxOccurs="unbounded">  
  <xsd:complexType>  
    <xsd:sequence>  
      <xsd:element name="nazev" type="xsd:string"/>  
      <xsd:element name="cena" type="xsd:decimal"/>  
    </xsd:sequence>  
    <xsd:attribute name="kod" type="xsd:integer"/>  
  </xsd:complexType>  
</xsd:element>
```

# Plátkování

- všechny prvky a atributy se definují na stejné (nejvyšší) úrovni
- odkazují se na sebe pomocí `ref="jméno"`
- definované prvky/atributy lze používat opakovaně
- koncepce blízká DTD
- obsah prvku se nemůže lišit podle kontextu (rodiče)

# Plátkování – příklad

```
<xsd:element name="nazev" type="xsd:string"/>
<xsd:element name="cena" type="xsd:decimal"/>
<xsd:attribute name="kod" type="xsd:integer"/>

<xsd:element name="zbozi">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="nazev"/>
      <xsd:element ref="cena"/>
    </xsd:sequence>
    <xsd:attribute ref="kod"/>
  </xsd:complexType>
</xsd:element>
```

# Slepý Benátčan

- předem se (na globální úrovni) definují typy
- prvky a atributy se definují lokálně (jako v matrjošce), ovšem triviálně s využitím připravených typů
- nejflexibilnější
- typy lze používat opakovaně
- ale schéma je delší
- vhodné pro složité jazyky

# Slepý Benátčan – příklad (1)

```
<xsd:simpleType name="nazevTyp">  
  <xsd:restriction base="xsd:string">  
    <xsd:maxLength value="50"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

```
<xsd:simpleType name="cenaTyp">  
  <xsd:restriction base="xsd:decimal">  
    <xsd:fractionDigits value="2"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

# Slepý Benátčan – příklad (2)

```
<xsd:simpleType name="kodTyp">  
  <xsd:restriction base="xsd:integer"/>  
</xsd:simpleType>  
  
<xsd:complexType name="zboziTyp">  
  <xsd:sequence>  
    <xsd:element name="nazev" type="nazevTyp"/>  
    <xsd:element name="cena" type="cenaTyp"/>  
  </xsd:sequence>  
  <xsd:attribute name="kod" type="kodTyp"/>  
</xsd:complexType>  
  
<xsd:element name="zbozi" type="zboziTyp"/>
```



# **Pokročilé konstrukce**





# Skupiny

- prvky **group** a **attributeGroup**
- umožňují definovat obvyklé kombinace (např. základní sadu atributů pro všechny prvky)
- nejsou typy, jen „kontejnery“
- použití: atribut `ref=“jméno skupiny“`

# Skupiny – příklad (1)

```
<xsd:group name="prvkyZbozi">
```

```
  <xsd:sequence>
```

```
    <xsd:element name="nazev" type="nazevTyp"/>
```

```
    <xsd:element name="cena" type="cenaTyp"/>
```

```
  </xsd:sequence>
```

```
</xsd:group>
```

```
<xsd:attributeGroup name="spolecneAtr">
```

```
  <xsd:attribute name="id" type="xsd:ID"/>
```

```
  <xsd:attribute name="style" type="xsd:string"/>
```

```
</xsd:attributeGroup>
```

# Skupiny – příklad (2)

```
<xsd:complexType name="zboziTyp">  
  <xsd:group ref="prvkyZbozi"/>  
  <xsd:attributeGroup ref="spolecneAtr"/>  
  <xsd:attribute name="kod" type="kodTyp"/>  
</xsd:complexType>
```

# Unikátní hodnoty

- lze požadovat, aby hodnoty určitého prvku či atributu (nebo jejich kombinace) byly v dané části dokumentu unikátní
- prvek **unique**
  - místo výskytu: kde má být jednoznačné
  - **selector**: v čem hledat prvek/atribut
  - **field**: který prvek/atribut má být jednoznačný
  - identifikace vychází z XPath

# Příklad – unikátní kódy zboží

```
<xsd:element name="cenik">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="zbozi" type="zboziTyp"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:unique name="unikatniKod">
    <xsd:selector xpath="zbozi"/>
    <xsd:field xpath="@kod"/>
  </xsd:unique>
</xsd:element>
```

# Klíče a jejich použití

- definice klíče: prvek **key**
  - odpovídá unique, povinný
- použití: prvek **keyref**
  - atribut **refer** – jméno definovaného klíče
  - vnořené prvky **selector** a **field** určují prvek/atribut, jehož hodnotou má být hodnota daného klíče

# Příklad klíče (1)

- příklad: zboží může být součástí jiného – nepovinný prvek soucast, jehož hodnotou je kód jiného zboží

```
<xsd:complexType name="zboziTyp">
```

```
...
```

```
<xsd:element name="soucast" type="kodTyp"  
             minOccurs="0" maxOccurs="unbounded"/>
```

```
</xsd:complexType>
```

# Příklad klíče (2)

```
<xsd:element name="cenik">
  <xsd:complexType>...</xsd:complexType>
  <!-- atribut kod je klíčem, musí být jednoznačný -->
  <xsd:key name="unikatniKod">
    <xsd:selector xpath="zbozi"/>
    <xsd:field xpath="@kod"/>
  </xsd:key>
  <!-- hodnotou soucast musí být existující kód zboží -->
  <xsd:keyref name="odkazNaKod" refer="unikatniKod">
    <xsd:selector xpath="zbozi"/>
    <xsd:field xpath="soucast"/>
  </xsd:keyref>
</xsd:element>
```

*<soucast> ve <zbozi>  
musí obsahovat platnou  
hodnotu klíče unikatniKod*



# Jmenný prostor jazyka

- atribut **targetNamespace** kořenového prvku schema definuje jazykový prostor pro definované prvky
- vhodné též definovat jako implicitní prostor
- problém: do jmenného prostoru patří jen prvky nejvyšší úrovně, nikoli prvky vnořené
- řešení: atribut **elementFormDefault="qualified"** určí, že všechny prvky schématu jsou kvalifikované (zařazené do jmenného prostoru)
- analogicky **attributeFormDefault**

# Příklad – jmenný prostor

```
<xsd:schema  
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
  targetNamespace="http://www.tul.cz/satrapa/pokus1"  
  xmlns="http://www.tul.cz/satrapa/pokus1"  
  elementFormDefault="qualified"  
  attributeFormDefault="qualified">  
... definice typů a skupin ...  
<xsd:element name="cenik">  
  ... definice prvků ...  
</xsd:element>  
</xsd:schema>
```

# Skládání schémat

- rozložení složitého schématu, knihovny prvků
- `<xsd:include schemaLocation="lokátor"/>`  
se nahradí textem z daného lokátoru („opíše“ text)
- `<xsd:redefine schemaLocation="lokátor">`  
předefinované součásti  
`</xsd:redefine>`  
vloží text z lokátoru, ale předefinuje jeho vybrané  
součásti podle svého těla

# Import schémat

- pro schémata, která mají mít svůj vlastní prostor
- prvek **import**, atributy
  - **namespace** – jmenný prostor vkládaného schématu
  - **schemaLocation** – lokátor souboru obsahujícího schéma
- např. chci vložit schéma pro MathML  
`<xsd:import  
  namespace="http://www.w3.org/1998/Math/MathML"  
  schemaLocation="schema/mathml.xsd"/>`

# Dokumentace

- lze používat XML komentáře
- většina konstrukcí může obsahovat (na začátku) prvek **annotation** obsahující popis jejich určení
- potomci:
  - **documentation** – pro lidského čtenáře
  - **appinfo** – pro zpracovávající aplikaci
- existují nástroje pro generování dokumentace (např. xnsdoc, XSDDoc)

# Příklad dokumentace

```
<xsd:element name="cenik">
```

```
  <xsd:annotation>
```

```
    <xsd:documentation xml:lang="cs">
```

Kořenový prvek obsahující ceník zboží.

```
  </xsd:documentation>
```

```
    <xsd:documentation xml:lang="en">
```

Root element containing the price list.

```
  </xsd:documentation>
```

```
  </xsd:annotation>
```

... definice obsahu prvku ...

```
</xsd:element>
```

# Zastupující typy

- ve schématu lze stanovit, že určité typy mohou zastoupit jiný typ
- oficiálně: vznikne **substituční skupina**
  - **hlava skupiny**: prvek zastupitelný ostatními členy skupiny; neobsahuje žádné speciální konstrukce
  - syntaxe: zastupující prvky ve své definici obsahují atribut **substitutionGroup="hlava\_skupiny"**
  - zastupující prvek musí být stejného nebo odvozeného typu jako hlava

# Příklad 1

- místo prvku `<zbozi>` se v datech může vyskytovat i `<ware>` nebo `<artikel>`
- pokud je typ stejný, nemusí se uvádět:

```
<xsd:element name="zbozi" type="zboziTyp"/>
```

```
<xsd:element name="ware" substitutionGroup="zbozi"/>
```

```
<xsd:element name="artikel" substitutionGroup="zbozi"/>
```



# Příklad 2

- zastupující typ může být odvozen od zastupovaného  

```
<xsd:element name="cd" type="cdTyp"  
              substitutionGroup="zbozi">  
<xsd:complexType name="cdTyp">  
  <xs:complexContent>  
    <xs:extension base="zboziTyp">  
      <xs:sequence>  
        <xs:element name="interpret" type="interpretTyp"/>  
      </xs:sequence>  
    </xs:extension>  
  </xs:complexContent>  
</xsd:complexType>
```

# Abstraktní typy

- atribut **abstract="true"** definuje prvek jako abstraktní – nepoužívá se přímo, musí být vždy zastoupen jiným

- příklad:

```
<xsd:element name="jm-vzor" type="xsd:string"  
  abstract="true"/>
```

```
<xsd:element name="jmeno" type="xsd:string"  
  substitutionGroup="jm-vzor"/>
```

```
<xsd:element name="prijmeni" type="xsd:string"  
  substitutionGroup="jm-vzor"/>
```



# **XML Schema 1.1**



# XML Schema 1.1

- přijato 2012
- přidána validační pravidla (inspirovaná Schematronem)
- odstraněna řada omezení pro **<all>**
- **<redefine>** zavrženo, nahrazeno obecnějším **<override>**
- nové typy
- řada dalších rozšíření

# Validační pravidla

- lze přidat k definici typu a určit dodatečná omezení pro jeho hodnoty – typicky jejich vzájemné vztahy
- **<assert>** pro složené typy, **<assertion>** pro jednoduché
- atribut **test** obsahuje pravdivostní XPath výraz (bude později); pokud je vyhodnocen jako nepravdivý, je dokument nevalidní – stejně jako při jakémkoli jiném porušení definice jazyka

# Příklad: Interval

```
<xsd:element name="interval">  
  <xsd:complexType>  
    <xsd:sequence>  
      <xsd:element name="from" type="xsd:decimal"/>  
      <xsd:element name="to" type="xsd:decimal"/>  
    </xsd:sequence>  
    <xsd:assert test="from le to"/>  
  </xsd:complexType>  
</xsd:element>
```

*hodnota ve <from> musí být  
menší nebo rovna hodnotě <to>*



evropský  
sociální  
fond v ČR



EVROPSKÁ UNIE



MINISTERSTVO ŠKOLSTVÍ,  
MLÁDEŽE A TĚLOVÝCHOVY



OP Vzdělávání  
pro konkurenceschopnost



TECHNICKÁ  
UNIVERZITA  
V LIBERCI  
[www.tul.cz](http://www.tul.cz)

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Realizováno za finanční podpory ESF a státního rozpočtu ČR  
v rámci v projektu *Zkvalitnění a rozšíření možností studia  
na TUL pro studenty se SVP* reg. č. CZ.1.07/2.2.00/29.0011

# RELAX NG

# RELAX NG

- **REgular LAnguage for XML Next Generation**
- vytvořila **OASIS** (Organization for the Advancement of Structured Information Standards)
  - původně sdružení výrobců SGML nástrojů
  - později přeorientováno na XML
  - vyvinulo desítky standardů (DocBook, OpenDocument, SAML,...)
- RELAX NG vznikl 2002



# Základní principy

- postaveno na vzorech, nikoli datových typech
  - schéma je vzorem dokumentu
  - obsahuje vzory prvků, atributů a textových uzlů
- dvě alternativní syntaxe
  - **XML** – jmenný prostor  
<http://relaxng.org/ns/structure/1.0>
  - **textová, kompaktní**
  - ekvivalentní a vzájemně převoditelné

# Základní vzory (1)

- **text** – neobsahuje vnořené prvky, i prázdný
  - `<text/>`
  - `text`
- **prvek** – obsahuje vzory pro svůj obsah
  - `<element name="jméno">`  
*obsah*  
`</element>`
  - `element jméno { obsah }`

# Příklad: osoba

```
<osoba>  
  <krestni>Josef</krestni>  
  <prijmeni>Novák</prijmeni>  
</osoba>
```

```
<element name="osoba">  
  <element name="krestni">  
    <text/>  
  </element>  
  <element name="prijmeni">  
    <text/>  
  </element>  
</element>
```

```
element osoba {  
  element krestni { text },  
  element prijmeni { text }  
}
```

# Základní vzory (2)

- atributy – uvnitř prvku, nezáleží na pořadí
  - `<attribute name="jméno"><text/></attribute>`  
lze zkrátit na `<attribute name="jméno"/>`
  - `attribute jméno { text }`

```
<element name="osoba">  
  <attribute name="id"/>  
  <element name="krestni">  
    <text/>  
  </element>
```

...

```
</element>
```

```
element osoba {  
  attribute id { text },  
  element krestni { text },  
  element prijmeni { text }  
}
```

# Prázdný prvek

- prvek bez obsahu
  - `<element name="jméno"><empty/></element>`
  - `element jméno { empty }`
- pokud obsahuje atributy, lze vzor empty vynechat

# Opakování (1)

- nepovinný výskyt
  - **<optional>** – prvky zabalené jsou nepovinné
  - **?** – připojuje se za konstrukci
- libovolný počet výskytů
  - **<zeroOrMore>**
  - **\***
- alespoň jeden výskyt
  - **<oneOrMore>**
  - **+**

# Příklad: nepovinné id, více jmen

```
<element name="osoba">  
  <optional>  
    <attribute name="id"/>  
  </optional>  
  <oneOrMore>  
    <element name="krestni">  
      <text/>  
    </element>  
  </oneOrMore>  
  <element name="prijmeni">  
    <text/>  
  </element>  
</element>
```

```
element osoba {  
  attribute id { text }?,  
  element krestni { text }+,  
  element prijmeni { text }  
}
```

# Opakování (2)

- nelze určit konkrétní rozsah pro počet opakování (něco jako minOccurs, maxOccurs)
- musí se vyjádřit explicitně – např. 2 až 4 jména:  
  - element jmeno { text },
  - element jmeno { text },
  - element jmeno { text }?,
  - element jmeno { text }?
- naštěstí je tato potřeba vzácná





# Kombinování vzorů



# Alternativy

- v dokumentu má být právě jeden z prvků
  - `<choice>`
  - |

```
<element name="stav">
```

```
  <choice>
```

```
    <element name="zenaty">
```

```
      <text/>
```

```
    </element>
```

```
    <element name="svobodny"/><empty/> </element>
```

```
  </choice>
```

```
</element>
```

```
element stav {
```

```
  element zenaty { text }
```

```
  | element svobodny { empty }
```

```
}
```

# Zachování pořadí

- implicitně zachovává
  - lze vyjádřit pomocí **<group>**
  - kompaktní schéma: čárky, případně závorky
- explicitní vyjádření může být potřebné, jedná-li se např. o pevnou podskupinu mezi alternativami

# Příklad

```
<element name="sluzba">
  <choice>
    <element name="jmeno"/>
      <text/>
    </element>
    <group>
      <element name="adresa">
        <text/>
      </element>
      <element name="port">
        <text/>
      </element>
    </group>
  </choice>
</element>
```

- služba je určena jménem nebo adresou a portem

```
element sluzba {
  element jmeno { text }
  | ( element adresa { text },
      element port { text } )
}
```

# Libovolné pořadí

- pokud připouštíme vzory v libovolném pořadí
  - obalit prvkem **<interleave>**
  - oddělovat znakem **&**

**<element name="kniha">**

**<element name="nazev"/><text/></element>**

**<element name="autor"/><text/></element>**

**<interleave>**

**<element name="isbn"><text/></element>**

**<element name="cena"><text/></element>**

**</interleave>**

**</element>**

**element kniha {**

**element nazev { text },**

**element autor { text },**

**( element isbn { text }**

**& element cena { text } )**

**}**

# Prolínání je silnější

- lze definovat několik prvků (či jejich skupin s definovaným pořadím), které se pak mohou navzájem prolínat
- např. definici  $a \{ (x,y) \& (p,q) \}$  vyhoví  
     $\langle a \rangle \langle x \rangle \langle y \rangle \langle p \rangle \langle q \rangle \langle /a \rangle$   
     $\langle a \rangle \langle x \rangle \langle p \rangle \langle q \rangle \langle y \rangle \langle /a \rangle$   
     $\langle a \rangle \langle p \rangle \langle x \rangle \langle q \rangle \langle y \rangle \langle /a \rangle \dots$   
    ( $\langle x \rangle$  musí být před  $\langle y \rangle$  a  $\langle p \rangle$  musí být před  $\langle q \rangle$ ,  
    ale vzájemná pozice těchto dvojic je libovolná)

# Příklad – komentáře

```
<element name="kniha">
  <interleave>
    <group>
      <element name="nazev"/><text/></element>
      <element name="autor"/><text/></element>
      <element name="isbn"><text/></element>
    </group>
    <zeroOrMore>
      <element name="koment"><text/></element>
    </zeroOrMore>
  </interleave>
</element>
```

```
element kniha {
  (
    element nazev { text },
    element autor { text },
    element isbn { text }
  )
  &
  element koment { text }*
}
```

# Smíšený obsah

- lze pomocí `<interleave>` kde jednou z variant je `<text/>` (nemusí se mu dávat opakování, vyhovuje libovolnému počtu uzlů)
- celkem časté – prvek `<mixed>` (resp. `mixed { ... }`) nemusí mít vnořený `<text/>`
  - problém: obsahuje-li více prvků, automaticky kolem nich vytváří `<group>`
  - řešení: zabalit je do `<interleave>` (nebo nepoužívat `<mixed>`)



# Příklad: <p> s vnořeným textem, <em> a <strong>

```
<element name="p">
```

```
<mixed>
```

```
<interleave>
```

```
<zeroOrMore>
```

```
<element name="em"/>
```

```
<text/>
```

```
</element>
```

```
</zeroOrMore>
```

```
<zeroOrMore>
```

```
<element name="strong"/>
```

```
<text/>
```

```
</element>
```

```
</zeroOrMore>
```

```
</interleave>
```

```
</mixed>
```

```
</element>
```

```
element p {
```

```
  mixed {
```

```
    element em { text }*
```

```
    & element strong { text }*
```

```
  }
```

```
}
```

```
element p {
```

```
  element em { text }*
```

```
  & element strong { text }*
```

```
  & text
```

```
}
```



# Datové typy



# Datové typy (1)

- sám má jen dva typy:
  - **string**
  - **token**
- oba reprezentují textový řetězec
- token před validací normalizuje prázdné místo (mezery, tabulátory, konce řádků)

# Datové typy (2)

- zpravidla se vkládají externí knihovny datových typů, nejčastěji z XML Schema
- **XML syntaxe**
  - prvek **<data>**, typ určují atributy:
    - **type** – datový typ
    - **datatypeLibrary** – knihovna, z níž pochází; XML Schema <http://www.w3.org/2001/XMLSchema-datatypes>; dědí se

# Příklad: kniha

```
<element name="kniha"  
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">  
  <element name="nazev"/>  
    <data type="string"/>  
  </element>  
  ...  
  <element name="cena">  
    <data type="decimal"/>  
  </element>  
</element>
```

# Datové typy (3)

- kompaktní syntaxe
  - pomocí **datatypes** se definuje prefix knihovny typů
  - ten se pak používá v definicích
  - pro XML schema předdefinován prefix **xsd**

```
datatypes xs = "http://www.w3.org/2001/XMLSchema-datatypes"
```

```
element kniha {  
  element nazev { xs:string },  
  ...  
  element cena { xs:decimal }  
}
```

# Omezení datových typů

- pomocí parametrů
- **XML syntaxe:**
  - prvek `<param name="jméno">hodnota</param>`
  - atribut `name` udává jméno parametru (typ omezení)
  - obsah prvku určuje hodnotu daného omezení
- **kompaktní syntaxe:**
  - ve složených závorkách za typem
  - ve tvaru `jméno="hodnota"`

# Příklad: XML syntaxe

```
<element name="kontakt"  
  xmlns="http://relaxng.org/ns/structure/1.0"  
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">  
  <element name="jmeno">  
    <data type="string">  
      <param name="maxLength">50</param>  
    </data>  
  </element>  
  <element name="telefon">  
    <data type="token">  
      <param name="pattern">\d{3}\s*\d{3}\s*\d{3}</param>  
    </data>  
  </element>  
</element>
```



# Příklad: kompaktní syntaxe

```
element kontakt {  
  element jmeno {  
    xsd:string { maxLength = "50" }  
  },  
  element telefon {  
    xsd:token { pattern = "\d{3}\s*\d{3}\s*\d{3}" }  
  }  
}
```

- parametry pro omezení vycházejí z XML Schema, kromě enumeration a whiteSpace

# Výčtový typ

- standardní výběr, dostupné hodnoty:

- `<value>hodnota</value>`
- `"hodnota"`

```
<element name="cena">
  <attribute name="mena">
    <choice>
      <value>CZK</value>
      <value>USD</value>
      <value>EUR</value>
    </choice>
  </attribute>
  <data type="decimal"/>
</element>
```

```
element cena {
  attribute mena {
    "CZK"
    | "USD"
    | "EUR" },
  xsd:decimal
}
```

# Zakázané hodnoty

- některé hodnoty lze vyloučit:

- `<except>hodnota</except>`
- `– ( hodnota )`

```
<element name="promenna">  
  <data type="string">  
    <except>  
      <choice>  
        <value>begin</value>  
        <value>end</value>  
      </choice>  
    </except>  
  </data>  
</element>
```

```
element promenna {  
  xsd:string –  
    (( "begin" | "end" ))  
}
```

# Seznamy

- hodnoty oddělované prázdným místem
  - `<list>vzor pro obsah</list>`
  - `list { vzor pro obsah }`
- raději méně – odporuje duchu XML

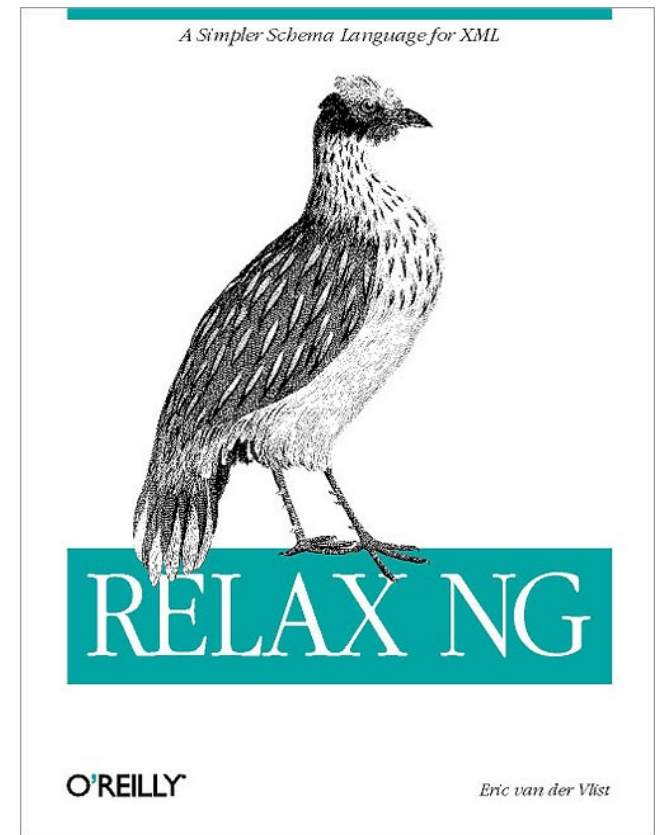
```
<element name="posloupnost">
  <list>
    <oneOrMore>
      <data type="decimal"/>
    </oneOrMore>
  </list>
</element>
```

```
element posloupnost {
  list { xsd:decimal+ }
}
```

# Doporučená četba

- Eric van der Vlist: *Relax NG*  
O'Reilly, 2003

dostupná volně na  
<http://books.xmlschemata.org/relaxng/>





evropský  
sociální  
fond v ČR



EVROPSKÁ UNIE



MINISTERSTVO ŠKOLSTVÍ,  
MLÁDEŽE A TĚLOVÝCHOVY



OP Vzdělávání  
pro konkurenceschopnost



TECHNICKÁ  
UNIVERZITA  
V LIBERCI  
[www.tul.cz](http://www.tul.cz)

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Realizováno za finanční podpory ESF a státního rozpočtu ČR  
v rámci v projektu *Zkvalitnění a rozšíření možností studia  
na TUL pro studenty se SVP* reg. č. CZ.1.07/2.2.00/29.0011

# Pojmenované vzory

# Pojmenované vzory (1)

- nezbytné pro složitější jazyky – opakované použití
- **XML syntaxe:**
  - `<define name="jméno">obsah</define>`
  - použití `<ref name="jméno"/>`
  - XML vyžaduje jeden kořenový prvek – schéma zabaleno do `<grammar>...</grammar>`
  - `<grammar>` obsahuje právě jeden prvek `<start>` určující výchozí bod schématu

# Příklad – XML

<grammar>

<start>

  <element name="osoba">

    <ref name="jmeno"/>

    <element name="adresa"><text/></element>

  </element>

</start>

<define name="jmeno">

  <element name="krestni"><text/></element>

  <element name="prijmeni"><text/></element>

</define>

</grammar>



# Pojmenované vzory (2)

- **kompaktní syntaxe:**
  - definice: *jméno* = *obsah*
  - při použití se jednoduše uvede *jméno*
  - není XML, přesto je i zde formálně definován obalový **grammar** { ... }, je ale implicitní – nemusí se uvádět
  - uvnitř **grammar** je právě jedna definice *start* = *obsah* určující výchozí bod schématu

# Příklad – kompaktní

```
grammar {
```

```
start =
```

```
    element osoba {
```

```
        jmeno,
```

```
        element adresa { text }
```

```
    }
```

```
jmeno =
```

```
    element krestni { text },
```

```
    element prijmeni { text }
```

```
}
```



# Definicje języka



# Obvyklé návrhy schémat

- **matrjoška**
  - přímé vkládání vzorů, jen pro jednoduché jazyky
- **à la DTD**
  - samostatná definice pro každý prvek, určující jeho obsah; odkazuje se na definice ostatních prvků
  - definice prvku snadno k nalezení
- **orientovaný na obsah**
  - definuje obsah každého prvku jako samostatný vzor
  - nejsnadněji rozšiřitelné

# Ceník jako matrjoška

```
element cenik {  
  element zbozi {  
    element nazev { text },  
    element cena { xsd:decimal }  
  }+  
}
```

# Ceník à la DTD

```
grammar {  
  start = prvek-cenik  
  prvek-cenik = element cenik { prvek-zbozi+ }  
  prvek-zbozi = element zbozi {  
    prvek-nazev,  
    prvek-cena  
  }  
  prvek-nazev = element nazev { text }  
  prvek-cena = element cena { xsd:decimal }  
}
```

# Ceník podle obsahů

```
grammar {  
  start = element cenik { cenik-obsah }  
  cenik-obsah = element zbozi { zbozi-obsah }+  
  zbozi-obsah =  
    element nazev { nazev-obsah },  
    element cena { cena-obsah }  
  nazev-obsah = text  
  cena-obsah = xsd:decimal  
}
```



# **Pokročilé konstrukce**





# Vkládání definic

- do grammar lze vložit další z externího souboru – vloží se všechny jeho definice
  - XML: `<include href="lokátor-souboru"/>`
  - kompaktní: `include "lokátor-souboru"`
- umožňuje vytvářet knihovny „předvařených“ definic

```
<grammar>
  <include href="typedef.rng"/>
  ...
</grammar>
```

```
grammar {
  include "typedef.rnc",
  ...
}
```

# Úprava vložených definic

- lze změnit vybrané definice z vloženého souboru

- XML:

```
<include href="lokátor-souboru">  
  <define name="jméno">  
    nová-definice  
  </define>  
</include>
```

- kompaktní:

```
include "lokátor-souboru" {  
  jméno = nová-definice  
}
```

# Odkazy na externí definice

- lze využívat vzory uložené v jiných souborech (obsahem souboru musí být jeden vzor)
  - XML: `<externalRef href="lokátor-souboru"/>`
  - kompaktní: `external "lokátor-souboru"`

```
<element name="prodejna">  
  <externalRef href="cenik.rng"/>  
  <element name="vedouci">  
    <text/>  
  </element>  
</element>
```

```
element prodejna {  
  external "cenik.rnc",  
  element vedouci { text }  
}
```

# Kombinování definic

- schéma může obsahovat několik stejnojmenných definic – kombinují se
- nutno definovat způsob kombinace
  - XML: atribut **combine**, hodnoty **choice** a **interleave**
  - kompaktní: znaky **|=** a **&=** místo **=**
  - všechny definice pro stejné jméno musí mít totožný způsob kombinování
  - u jedné lze vynechat – výhodné pro úpravy knihoven

# Příklad: alternativy pro jméno

```
<define name="jmeno">  
  <element name="jmeno">  
    <text/>  
  </element>  
</define>
```

```
<define name="jmeno" combine="choice">  
  <element name="krestni">  
    <text/>  
  </element>  
  <element name="prijmeni">  
    <text/>  
  </element>  
</define>
```

```
jmeno =  
  element jmeno { text }  
  
jmeno |=  
  element krestni { text },  
  element prijmeni { text }
```

# Jmenné prostory (1)

- Relax NG podporuje jmenné prostory
- XML syntaxe:
  - atribut **ns**=“*lokátor-jmenného-prostoru*“ u prvku **element**
  - dědí se, stačí uvést u kořenového prvku
  - atribut **ns** lze uvést i u prvků **attribute**; implicitní hodnotou je prázdný řetězec
  - jmenné prostory lze deklarovat i standardně a používat prefixy v hodnotě atributu **name**

# Jmenné prostory (2)

- kompaktní syntaxe:
  - na začátku je třeba deklarovat prefixy jmenných prostorů  
`namespace prefix = "lokátor-jmenného-prostoru"`
  - deklarace se nemohou vnořovat, nepřenášejí se z vkládaných souborů
  - lze deklarovat implicitní jmenný prostor:  
`default namespace = "lokátor-jmenného-prostoru"`  
(opět se nevztahuje na atributy)
  - nemá-li vložený soubor (`include`, `external`) vlastní implicitní jmenný prostor, zdědí jej od vkládajícího

# Příklad s implicitním prostorem

```
<element name="cenik"  
  ns="http://www.kdesi.cz/relax/cenik">  
  <oneOrMore>  
    <ref name="zbozi"/>  
  </oneOrMore>  
</element>  
  
<define name="zbozi">  
  <element name="zbozi">  
    <element name="nazev">  
      <text/></element>  
    <element name="cena">  
      <text/></element>  
    </element>  
  </define>
```

```
default namespace =  
  "http://www.kdesi.cz/  
  relax/cenik"  
  
element cenik {  
  zbozi+  
}  
  
zbozi = element zbozi {  
  element nazev { text },  
  element cena { text }  
}
```



# Obsah podle kontextu

- Relax NG umožňuje definovat strukturu dokumentu v závislosti na jeho obsahu
- hodnota prvku či atributu může ovlivnit, jaké další prvky/atributy dokument bude obsahovat
- technická realizace: výběr ze skupin, do nichž patří prvek/atribut s danou hodnotou a věci na něm závislé
- příklad: u hudby nás zajímá médium, u SW licence

# Příklad závislosti na kontextu

```
<element name="cd">
  <element name="nazev">
    <text/></element>
  <choice>
    <group>
      <attribute name="druh">
        <value>hudba</value>
      </attribute>
      <element name="interpret">
        <text/></element>
      </group>
      ...
    </choice>
  </element>
```

```
element cd {
  element nazev { text },
  ( attribute druh { "hudba" },
    element interpret { text } )
  |
  ( attribute druh { "sw" },
    element licence { text } )
}
```

# Dokumentace (1)

- nemá speciální dokumentační prvky, lze vložit cokoli
- **XML syntaxe:**
  - prvky z jiných jmenných prostorů než <http://relaxng.org/ns/structure/1.0> jsou ve schématu ignorovány
  - lze využít pro vkládání dokumentace a komentářů

<element name="zbozi">

<html:p><html:strong>zbozi:</html:strong>

jedna položka ceníku.</html:p>

...

</element>

# Dokumentace (2)

- **kompaktní syntaxe:**

- dokumentace bývá v XML
- pro ohraničení se používají [ ... ]
- přesná syntaxe závisí na umístění (před/za/mimo prvek)

```
[ html:p [ html:strong [ zboží ]  
jedna položka ceníku. ] ]  
element zboží {  
    ...  
}
```

# Dokumentace (3)

- **kompaktní syntaxe:**

- připouští komentáře, zahájeny znakem *#*
- specifikace kompatibility s DTD zavádí prvek `documentation`; zkrácená syntaxe *## text*

*## jedna položka ceníku*  
*element zbozi {*

*...  
}*

*namespace a =*

*"http://relaxng.org/ns/compatibility/annotations/1.0"*

*[ a:documentation [  
    jedna položka ceníku ] ]*

*element zbozi {*

*...  
}*



evropský  
sociální  
fond v ČR



EVROPSKÁ UNIE



MINISTERSTVO ŠKOLSTVÍ,  
MLÁDEŽE A TĚLOVÝCHOVY



OP Vzdělávání  
pro konkurenceschopnost



TECHNICKÁ  
UNIVERZITA  
V LIBERCI  
[www.tul.cz](http://www.tul.cz)

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Realizováno za finanční podpory ESF a státního rozpočtu ČR  
v rámci v projektu *Zkvalitnění a rozšíření možností studia  
na TUL pro studenty se SVP* reg. č. CZ.1.07/2.2.00/29.0011

# Strom dokumentu

# XML strom

- textový soubor je *zápisem* datového souboru
- zpracovávající software si zpravidla staví strom odpovídající XML datům – interní reprezentace odrážející vzájemné vztahy prvků
- výhodné pro posuzování struktury a manipulaci s ní

# Příklad: XML dokument...

```
<?xml version="1.0" encoding="ISO-8859-2"?>
```

```
<diskoteka>
```

```
<disk id="d1">
```

```
  <interpret>Divokej Bill</interpret>
```

```
  <nazev>Propustka do pekel</nazev>
```

```
</disk>
```

```
<disk id="d2">
```

```
  <interpret>AC/DC</interpret>
```

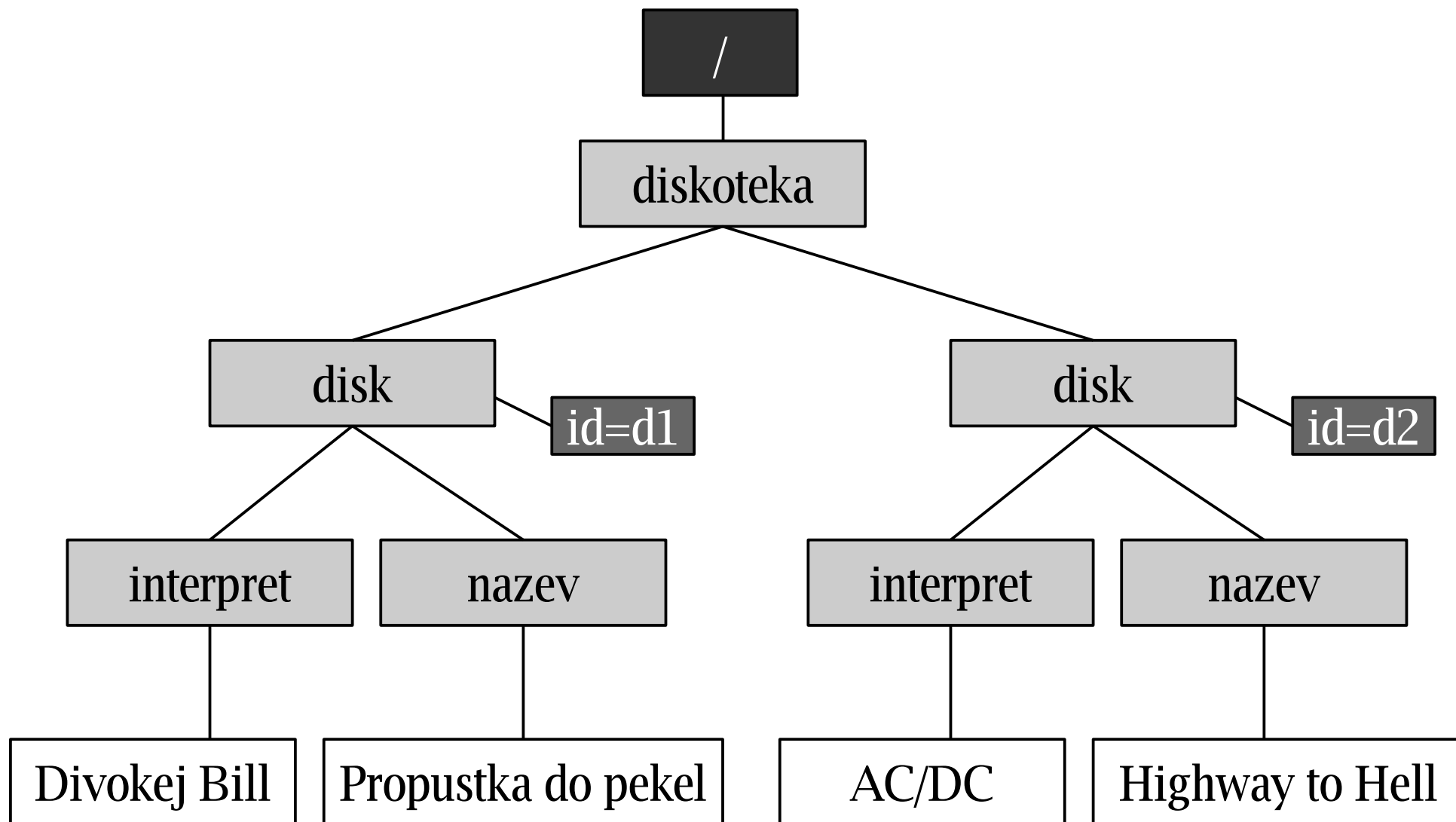
```
  <nazev>Highway to Hell</nazev>
```

```
</disk>
```

```
</diskoteka>
```



# ...a jeho strom



# Typy uzlů

- **kořen** – uměle přidaný
- **prvek** – odpovídají prvkům dokumentu
- **text** – textový obsah, bezejmenné, vždy listové
- **atribut** – připojen k prvku, který jej nese (ten je jeho rodičem, ale atribut není považován za dítě)
- **jmenný prostor** – název=prefix, dědí se
- **instrukce pro zpracování** – názvem je její cíl
- **komentář** – bezejmenný

# Vztahy prvků

- **A je předkem B, B je potomkem A:**
  - B je obsažen v A
- **A je rodičem B, B je dítětem A:**
  - B je přímo obsažen v A (je ve struktuře právě o jednu úroveň níže)
  - rodič je vždy právě jeden
- **A je sourozencem B:**
  - mají stejného rodiče

# **XPath**

# XPath

- nástroj pro vyhledávání informací v XML dokumentech – identifikaci uzlů a jejich skupin ve stromě (prvků, atributů,...)
- řada (více než sto) vestavěných funkcí
- není samostatný jazyk
  - definuje syntax výrazů
  - používány jako součást dalších mechanismů (zejména XSLT, XQuery)

# XPath

- nemá XML syntaxi
  - nepotřebuje – typicky tvoří hodnotu atributu
  - připomíná cestu v systému souborů
- **výsledkem** XPath výrazu je hodnota nebo skupina uzlů XML stromu vyhovujících podmínkám
- interpretace začíná ve **výchozím uzlu**
  - může/nemusí být kořen
  - typicky identifikuje uzly relativně vůči výchozímu

# XPath cesta

- XPath cesta je sekvencí kroků, oddělovány lomítky
  - *krok1/krok2/.....*
- každý **krok** může mít tři části
  - **identifikátor osy** – směr procházení, výchozí množina
  - **test uzlu** – kterých uzlů se týká, povinná část
  - **podmínka (predikát)** – zužuje výsledky předchozího výběru
  - plný tvar: *osa::uzel[podmínka]*

# Vyhodnocení cesty

- každý krok se vyhodnocuje v určitém **kontextu** – vychází se z konkrétního uzlu
  - první krok: z aktuálního nebo kořenového uzlu
  - další kroky: z uzlů vybraných předchozím krokem (postupně pro každý z nich)
- části kroku postupně omezují množinu vyhovujících uzlů: osa vybere základní množinu, test uzlu z ní vybere jen některé a případné podmínky ponechají jen ty, jež jim vyhovují



# Oddělování kroků

- **znak /**

- prostý oddělovač
- díky implicitní ose **child::** se další krok typicky týká přímých potomků uzlů vybraných předchozím krokem
- na začátku cesty: absolutní cesta, začíná v kořeni

- **dvojice //**


- mezi uzly se může nacházet libovolný počet mezilehlých
- na začátku cesty: začíná se v kořeni, první prvek může být libovolně hluboko, trvá tak báječně dlouho...

# Vybírání uzlů podle názvu

- nejjednodušší je uvést jméno prvku
  - **nazev** – nazev jako potomek aktuálního uzlu
- často požadujeme prvek v určitém kontextu, vyjádřen sekvenčí kroků s příslušnými oddělovači
  - **/diskoteka/disk/nazev**
  - **/diskoteka//nazev** – nazev kdekoli uvnitř diskoteka
  - **//nazev** – nazev kdekoli v dokumentu
- **\*** – vyhoví libovolný prvek

# Vybírání uzlů podle typu

- lze požadovat libovolný uzel zadaného typu
- nezáleží na jménu, jen na charakteru
- dostupné typy:
  - `text()` – textový uzel, např. `nazev/text()`
  - `comment()` – uzel s komentářem
  - `processing-instruction()` – instrukce pro zpracování, v závorkách lze uvést i její požadované jméno
  - `node()` – libovolný uzel



# Osy a podmínky



# Identifikátor osy (1)

- **child::** děti aktuálního uzlu, implicitní osa
- **descendant::** všichni potomci aktuálního uzlu
- **descendant-or-self::** uzel sám a jeho potomci
- **parent::** rodič aktuálního uzlu
- **ancestor::** všichni předci aktuálního uzlu
- **ancestor-or-self::** uzel sám a jeho předci
- **self::** sám aktuální uzel

# Identifikátor osy (2)

- **following-sibling::** následující sourozenci uzlu
- **preceding-sibling::** předchozí sourozenci uzlu
- **following::** uzly následující v dokumentu za aktuálním uzlem, kromě jeho potomků
- **preceding::** uzly předcházející v dokumentu před aktuálním uzlem, kromě jeho předků
- **attribute::** atributy aktuálního uzlu
- **namespace::** jmenné prostory aktuálního uzlu

# Rozdělení dokumentu

- osy

self

ancestor

preceding

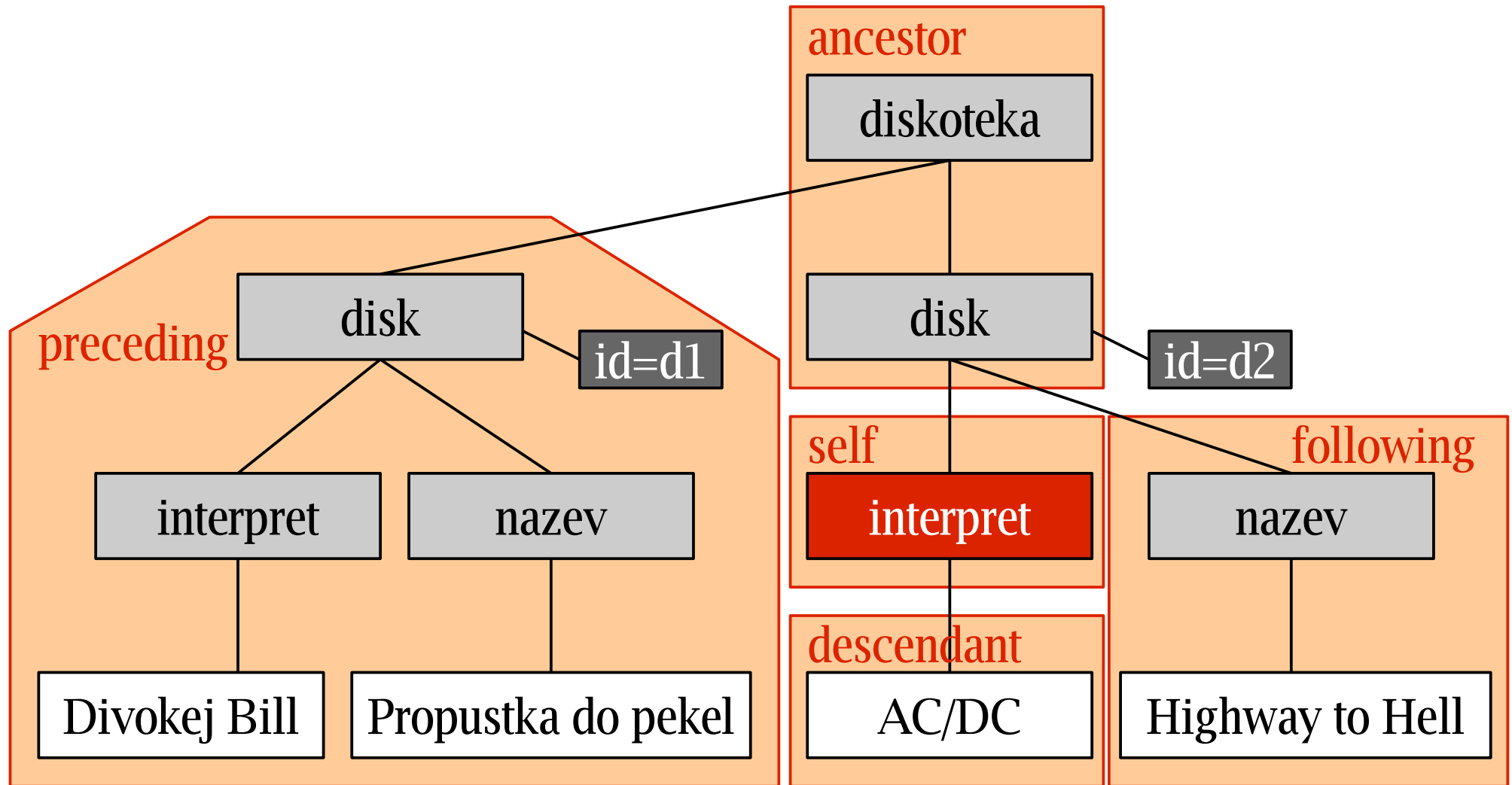
descendant

following

rozdělují dokument na pět navzájem disjunktních částí a kompletně jej pokrývají

- nezahrnují atributy a jmenné prostory

# Příklad rozdělení





# Zkratky

- **.** aktuální uzel **self::node()**  
**./navez** názvy, které jsou potomky akt. uzlu
- **..** rodič **parent::node()**  
**../interpret** interpret jako sourozenec
- **@** atribut **attribute::**  
**../@id** rodičův atribut id  
**@\*** všechny atributy aktuálního prvku
- **//** lib. část cesty **/descendant-or-self::node()/**  
**//@id/..** všechny prvky obsahující atribut id

# Podmínky

- v hranatých závorkách za testem uzlu
- **[*n*]** *n*-tý uzel v množině
  - **\*[3]** ... třetí dítě aktuálního prvku
  - **(//nazev)[2]** ... druhý název v dokumentu
  - **following::\*[1]** ... první prvek následující za aktuálním
  - osy mířící zpět (ancestor, preceding,...) číslují odzadu
    - **preceding::\*[1]** ... nejbližší předchozí prvek
- častou podmínkou je hodnota atributu
  - **//disk[@medium="dvd"]**



# Funkce



# Funkce (1)

- **position()** ... číslo
  - pozice (pořadí, číslováno od 1) aktuálního uzlu v množině
  - [3] je zkratka za [position()=3]
- **last()** ... číslo
  - pozice posledního uzlu v množině
- **count(*množina\_uzlů*)** ... číslo
  - počet uzlů v dané množině

# Funkce (2)

- *id(objekt)* ... množina uzlů
  - vydá uzel nesoucí daný identifikátor
  - *id("d1")* ... prvek s identifikátorem „d1“
  - *id("d1")/nazev* ... jeho název
- *name(množina\_uzlů?)* ... řetězec
- *local-name(množina\_uzlů?)* ... řetězec
- *namespace-uri(množina\_uzlů?)* ... řetězec
- vydají kvalifikované jméno, lokální jméno a URI jmenného prostoru daného (aktuálního) uzlu

# Řetězcové funkce (1)

- **string(*objekt?*)** ... řetězec
  - převede svůj argument na řetězec znaků
  - prvky typicky převedeny na svůj textový obsah
  - neexistující hodnota: NaN; nekonečno: Infinity (–Infinity)
- **concat(*řetězec1, řetězec2,...*)** ... řetězec
  - spojí (zřetězí) své argumenty
- **string-length(*řetězec*)** ... číslo
  - počet znaků v řetězci

# Hledání řetězců

- **contains(*kde*, *co*)** ... boolean
  - vydá true, pokud je řetězec *co* obsažen v řetězci *kde*
- **starts-with(*kde*, *co*)** ... boolean
  - vydá true, pokud řetězec *kde* začíná podřetězcem *co*
- **substring-before(*kde*, *co*)** ... řetězec
- **substring-after(*kde*, *co*)** ... řetězec
  - vydá část řetězce *kde* před/za prvním výskytem řetězce *co* v něm; prázdný řetězec, pokud *kde* neobsahuje *co*

# Řetězcové funkce (2)

- **substring(*řetězec*, *začátek*, *počet\_znaků*?)** ... řetězec
  - vydá část řetězce začínající daným znakem a obsahující zadaný počet znaků (až do konce, pokud chybí)
- **normalize-space(*řetězec*?)** ... řetězec
  - vrátí řetězec s normalizovaným volným místem
- **translate(*kde*, *vzor*, *náhrada*)** ... řetězec
  - v řetězci kde změní znaky ze vzoru za nahrazující znaky
  - nahrazují se znaky, ne řetězce (viz tr z Unixu)



# Logické funkce

- **boolean(*objekt*)** ... boolean
  - konverze (false=NaN, 0, prázdný řet. a prázdná množina)
- **true(), false()**
- **lang(*řetězec*)** .. boolean
  - má aktuální uzel atribut xml:lang s hodnotou odpovídající zadanému *řetězci*?
  - např. **lang("en")** pro **<p xml:lang="en-us">..**</p>**** je true
- **not(*boolean*)** ... boolean
- operátory (ne funkce) **and** a **or**

# Číselné funkce

- **number(*objekt?*)** ... číslo
  - řetězec: přeskočí počáteční mezery, pokud následuje zápis čísla, výsledkem je toto číslo; jinak NaN
  - boolean: true=1, false=0
  - množina uzlů: převede se na řetězec voláním string()
- **sum(*množina\_uzlů*)** ... číslo
  - uzly se převedou na čísla a následně sečtou

# Aritmetické a relační operátory

- standardní sada aritmetických operací
  - $+$ ,  $-$ ,  $*$  v obvyklých významech
  - **div** je dělení *reálných* čísel (v plovoucí desetinné čárce)
  - **mod** je zbytek po celočíselném dělení
- obvyklé relační operátory
  - $=$ ,  $\neq$
  - $>$ ,  $\geq$ ,  $<$ ,  $\leq$

# Zaokrouhlovací funkce

- **round(*číslo*)** ... číslo
  - nejbližší celé číslo
- **floor(*číslo*)** ... číslo
  - nejbližší vyšší celé číslo
- **ceiling(*číslo*)** ... číslo
  - nejbližší nižší celé číslo

# Složitější podmínky

- podmínek může být několik
  - `//disk[@medium="dvd"][5]` ... páté DVD v dokumentu
  - vyhodnocují se zleva doprava
- podmínka se nemusí nutně vztahovat přímo k aktuálnímu uzlu
  - `//disk[.//nazev]` ... všechny prvky disk obsahující (libovolně hluboko zanořený) nazev
  - `//nazev[../interpret]` ... všechny prvky nazev, které mají sourozence interpret

# Kombinování XPath výrazů

- | v obvyklé roli „nebo“
  - výsledek je sjednocením množin uzlů vybraných jednotlivými výrazy
  - `//interpret | //nazev` ... všechny prvky interpret a nazev
  - pro hodnoty atributů je zpravidla vhodnější použít `or`:  
`//cena[@jednotka="EUR" or @jednotka="USD"]` versus  
`//cena[@jednotka="EUR"] | //cena[@jednotka="USD"]`

# Další vývoj

# XPath 2.0

- vyvíjeno společně s XQuery 1.0
- **podpora typů** – základní jednoduché typy podle XML Schema (19 typů) plus uzly
- každá hodnota má 2 složky:
  - vlastní hodnotu
  - typ
- **vše je sekvence**



# Sekvence

- libovolně dlouhá posloupnost obsahující atomické hodnoty a uzly
  - záleží na pořadí
  - může obsahovat duplicity
- **uzel nebo atomická hodnota je ekvivalentní jednoprvkové sekvenci obsahující tuto položku**
- **sekvence jsou jednoúrovňové**
  - při vnoření se zploští –  $(1, (2, 3))$  se převede na  $(1, 2, 3)$

# Složité výrazy

- **for cyklus:**

for \$cd in //disk return \$cd/nazev

- **podmíněný výraz:**

if (count(//disk)>100) then "moc" else "málo"

- **kvantifikátory:**

some \$cd in //disk satisfies \$cd/nazev="Cosi"

every \$cd in //disk satisfies \$cd/nazev="Cosi"

# XPath 3.0

- standardizováno 2014
- plný funkcionální jazyk
  - funkce je hodnota jako každá jiná
  - může být argumentem a/nebo výsledkem jiné funkce
  - více o funkcionálních jazycích viz *Alternativní metody programování*



INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Realizováno za finanční podpory ESF a státního rozpočtu ČR  
v rámci v projektu *Zkvalitnění a rozšíření možností studia  
na TUL pro studenty se SVP* reg. č. CZ.1.07/2.2.00/29.0011

# Definice vzhledu

# Prezentace XML

- XML popisuje strukturu dat, neřeší vzhled
- definice vzhledu:
  - CSS – jednoduchá varianta
  - XSL – vyvinuto pro XML, možnosti výrazně přesahují oblast prezentace

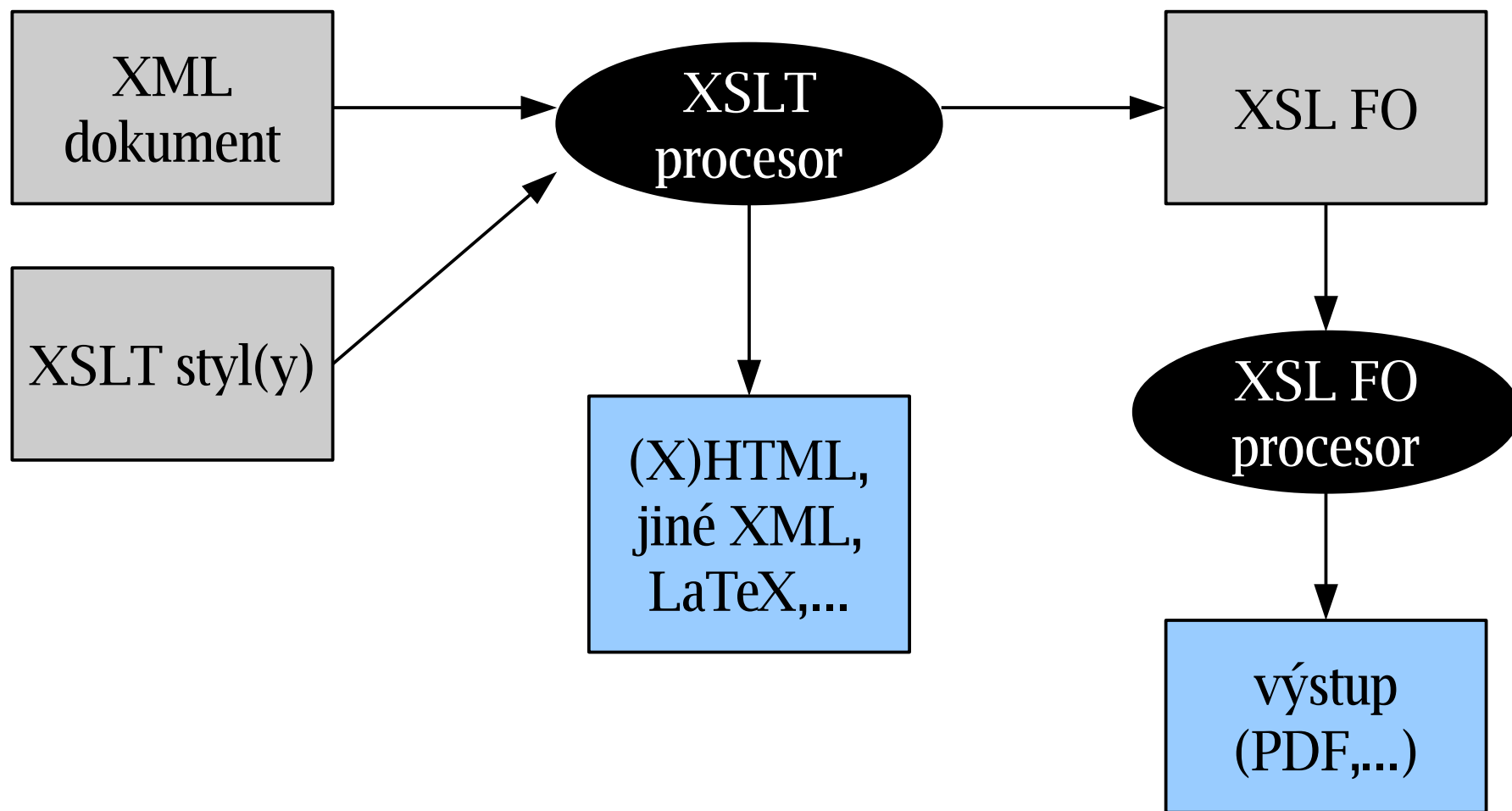
# CSS

- Cascading Style Sheets
- vyvinuto pro definici vzhledu HTML stránek
- použitelné i pro XML
- vložení do XML dokumentu – instrukce pro zpracování xml-stylesheet:  
`<?xml-stylesheet type="text/css" href="styl.css"?>`  
na začátku (za identifikací verze XML)

# XSL

- eXtensible Stylesheet Language
- původně určeno pro prezentaci XML dokumentů
- později rozděleno na dvě části:
  - **XSL FO (XSL Formatting Objects)** – jazyk definující formátovanou podobu dokumentu; po zpracování XSL FO procesorem vede k finální podobě
  - **XSLT (XSL Transformations)** – jazyk pro transformaci XML dokumentů, do XML FO, ale i jiných formátů

# Formátování XML





# XSL FO (1)

- popisuje formátování XML dokumentu, zejména pro stránkovaná média
- jmenný prostor  
<http://www.w3.org/1999/XSL/Format>
- dva klíčové prvky:
  - **<fo:layout-master-set>** definuje předlohy stránek
  - **<fo:page-sequence>** určuje obsahovou náplň stránek (s odkazy na předlohy)

# XSL FO (2)

- stále poměrně abstraktní:  

```
<fo:page-sequence>  
  <fo:flow flow-name="xsl-region-body">  
    <fo:block font-size="14pt" color="red">  
      Tady bude vlastní text bloku.  
    </fo:block>  
  </fo:flow>  
</fo:page-sequence>
```
- malý zájem – WWW Consortium dále nerozvíjí,  
nahrazeno CSS3-paged



# XSLT



# Charakteristika XSLT

- XML jazyk
- slouží k transformaci dokumentu
  - změna struktury a/nebo prvků
  - přeuspořádání, přidávání, výběr informací
- výstupní formáty
  - XML (např. XHTML, XSL FO, ale i XML data pro jinou aplikaci)
  - HTML
  - text

# XSLT procesor

- program implementující XSLT
- na základě stylového předpisu (XSLT stylu) transformuje vstupní dokument do výstupního
- interně transformuje stromy – uplatňováním šablon na vstupní strom vytváří výstupní strom

# XSLT styl

- sada **šablon** definujících transformací
- společný styl umožňuje jednotnou prezentaci řady dokumentů
- stejný dokument zpracovaný různými styly vede k diametrálně odlišným výsledkům
- příklad: DocBook poskytuje jednotný XML jazyk pro tvorbu dokumentace, XSLT styly (a doprovodné nástroje) z něj generují HTML, PDF,...

# Vazba stylu a dokumentu

- teoreticky stejně jako CSS (xml-stylesheet), jen typ je jiný:

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

- podporují jen některé nástroje
- procesory spouštěné z příkazového řádku někdy vyžadují uvést styl jako parametr

# Jmenný prostor a obal stylu

- jmenný prostor XSLT je  
<http://www.w3.org/1999/XSL/Transform>
- celý styl je obalen prvkem **stylesheet** (nebo **transform** – synonymum)
- **<xsl:stylesheet**  
    **xmlns:xsl="http://www.w3.org/1999/XSL/Transform"**  
    **version="1.0">**  
    *vlastní styl*  
    **</xsl:stylesheet>**



# Šablony

# XSLT šablona

- `<xsl:template match="vzor">`  
*obsah\_šablony*  
`</xsl:template>`
- **vzor** určuje, na které prvky výchozího dokumentu bude použit
  - vychází z XPath, ale připouští jen omezený rozsah konstrukcí – lze `child::`, `attribute::` a `//`
- **obsah\_šablony** definuje, co se vloží do výstupního stromu

# Příklad vstupních dat

```
<?xml version="1.0"?>
<cenik>
  <zbozi id="zb001" druh="potravina">
    <nazev>Houska</nazev>
    <cena>1.70</cena>
  </zbozi>
  <zbozi id="zb004" druh="potravina">
    <nazev>Voda</nazev>
    <cena>7.50</cena>
  </zbozi>
</cenik>
```

# Jednoduché šablony

- **implicitní šablona**

- provedena pro prvky, jež nevyhovují žádné šabloně
- opíše do výstupu textový obsah prvku
- rekurzivně prochází a transformuje jejich obsah

- **konverze prvku na jiný**

- `<xsl:template match="nazev">`  
    `<td><xsl:apply-templates/></td>`  
    `</xsl:template>`
- `<xsl:apply-templates/>` způsobí rekurzivní procházení

# Jednoduchý styl (1)

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:template match="/">
    <html>
      <head><title>Ceník</title></head>
      <body>
        <h1>Ceník</h1>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
```

# Jednoduchý styl (2)

```
<xsl:template match="cenik">
  <table><xsl:apply-templates/></table>
</xsl:template>
<xsl:template match="zbozi">
  <tr><xsl:apply-templates/></tr>
</xsl:template>
<xsl:template match="nazev">
  <td><xsl:apply-templates/></td>
</xsl:template>
<xsl:template match="cena">
  <td align="right"><xsl:apply-templates/></td>
</xsl:template>
</xsl:stylesheet>
```



# Prvky a atributy



# Generování prvků

- **opisováním**

- obsahuje-li šablona prvky z jiného jmenného prostoru (než prostor XSLT), opíše se do výstupního stromu

- **pomocí xsl:element**

- `<xsl:element name="jméno">obsah</xsl:element>`
- zdlouhavější, ale mocnější – jméno prvku je hodnotou atributu a tu lze vytvořit podle obsahu dokumentu



# Příklad obou přístupů

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

dělá totéž co

```
<xsl:template match="zbozi">  
  <xsl:element name="tr">  
    <xsl:apply-templates/>  
  </xsl:element>  
</xsl:template>
```

# Využívání hodnot z dokumentu

- **v attributech**
  - *{ výraz }*
  - vyhodnotí výraz a přiřadí výsledek jako hodnotu atributu
- **v těle prvků**
  - *<xsl:value-of select="výraz"/>*
  - vyhodnotí výraz a výsledek vloží na místo svého použití

# Příklad xsl:value-of

```
<zbozi id="zb001" druh="potravina">  
  <nazev>Houska</nazev>  
  <cena>1.70</cena>  
</zbozi>
```

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

```
  <tr>
```

```
    <td><xsl:value-of select="@id"/></td>
```

```
    <xsl:apply-templates/>
```

```
  </tr>
```

```
</xsl:template>
```



```
<tr>
```

```
  <td>zb001</td>
```

```
  <td>Houska</td>
```

```
  <td align="right">1.70</td>
```

```
</tr>
```

# Odvození hodnoty atributu

```
<zbozi id="zb001" druh="potravina">  
  <nazev>Houska</nazev>  
  <cena>1.70</cena>  
</zbozi>
```

```
<xsl:template match="zbozi">  
  <xsl:element name="{@druh}">  
    <xsl:apply-templates/>  
  </xsl:element>  
</xsl:template>
```

↓

```
<potravina>
```

...

```
</potravina>
```

# Generování atributů

- **opisováním**

- atributy opisovaných prvků lze zapsat přímo (viz align)
- hodnotu lze vytvořit pomocí {}

- **pomocí xsl:attribute**

- `<xsl:attribute name="jméno">hodnota</xsl:attribute>`
- vhodné uvnitř `xsl:element`
- umožňuje vytvořit i jméno atributu

# Příklad

```
<xsl:template match="zbozi">  
  <xsl:element name="{ @druh }">  
    <xsl:attribute name="ident">  
      <xsl:value-of select="@id"/>  
    </xsl:attribute>  
    <xsl:apply-templates/>  
  </xsl:element>  
</xsl:template>
```

```
<zbozi id="zb001" druh="potravina">  
  <nazev>Houska</nazev>  
  <cena>1.70</cena>  
</zbozi>
```



```
<potravina ident="zb001">  
  ...  
</potravina>
```

# Sady atributů

- lze si připravit sady atributů – skupiny přiřazované více prvkům
  - `<xsl:attribute-set name="jméno">`  
*definice atributů (prvky xsl:attribute)*  
`</xsl:attribute-set>`
- lze se pak na ně opakovaně odvolávat atributem `use-attribute-sets`
  - hodnotou je seznam jmen sad oddělovaných mezerami

# Příklad sady atributů

```
<xsl:attribute-set name="zboziAttr">  
  <xsl:attribute name="ident">  
    <xsl:value-of select="@id"/>  
  </xsl:attribute>  
  <xsl:attribute name="prodejna">Husova</xsl:attribute>  
</xsl:attribute-set>  
  
<xsl:template match="zbozi">  
  <xsl:element name="{ @druh}" use-attribute-sets="zboziAttr">  
    <xsl:apply-templates/>  
  </xsl:element>  
</xsl:template>
```



# Kopírování částí (1)

- vhodné, pokud chcete do výsledku převzít část původního dokumentu
- **xsl:copy**
  - zkopíruje do výstupního stromu aktuální uzel (který vyhověl atributu match šablony)
  - příklad: zkopírovat zboží a zpracovat jeho obsah:

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

# Kopírování částí (2)

- **xsl:copy-of**

- prázdný prvek
- zkopíruje vše, co vyhoví jeho atributu select
- příklad: kompletní kopie prvku zboží

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

```
  <xsl:copy>
```

```
    <xsl:copy-of select="@*" />
```

```
    <xsl:copy-of select="*" />
```

```
  </xsl:copy>
```

```
</xsl:template>
```

# Komentáře

- **generování**

- `<xsl:comment>` text komentáře `</xsl:comment>`
- text lze i generovat – `xsl:apply-templates` uvnitř

- **čtení**

- `match="comment()"` v podmínce šablony
- např. zkopírování do výstupu  
`<xsl:template match="comment()">`  
    `<xsl:comment>`  
        `<xsl:value-of select="."/>`  
    `</xsl:comment>`  
`</xsl:template>`

# Instrukce pro zpracování

- analogicky komentářům
- **generování**
  - `<xsl:processing-instruction name="cíl">`  
obsah instrukce  
`</xsl:processing-instruction>`
- **čtení**
  - podmínka `match="processing-instruction()`
  - případně `match="processing-instruction(cíl)"`



evropský  
sociální  
fond v ČR



EVROPSKÁ UNIE



MINISTERSTVO ŠKOLSTVÍ,  
MLÁDEŽE A TĚLOVÝCHOVY



OP Vzdělávání  
pro konkurenceschopnost



TECHNICKÁ  
UNIVERZITA  
V LIBERCI  
[www.tul.cz](http://www.tul.cz)

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Realizováno za finanční podpory ESF a státního rozpočtu ČR  
v rámci v projektu *Zkvalitnění a rozšíření možností studia  
na TUL pro studenty se SVP* reg. č. CZ.1.07/2.2.00/29.0011

# Cykly a koncepce transformace

# Cykly

- `<xsl:for-each select="XPath výraz">`  
    *šablona*  
`</xsl:for-each>`
- pro každý uzel vyhovující *XPath výrazu* bude do výstupního stromu vložena *šablona* (**vyhodnocuje se v kontextu uzlu vyhovujícího select cyklu**)
- vhodné zejména pro sumarizace informací

# Příklad: Seznam obrázků

- na začátku stránky chceme mít seznam obrázků

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

```
...
```

```
<h2>Seznam obrázků:</h2>
```

```
<ul>
```

```
<xsl:for-each select="//figure">
```

```
  <li><xsl:value-of select="@title"/></li>
```

```
</xsl:for-each>
```

```
</ul>
```

```
...
```

```
</xsl:template>
```

# Základní kostra transformace (1)

- **varianta 1: Přirozená rekurze**
  - šablony odpovídající zhruba 1:1 prvkům původního dokumentu
  - mnoho šablon, časté uplatnění **xsl:apply-templates**
  - výstupní strom je sestavován na základě rekurzivního procházení stromu vstupního
  - trochu připomíná podprogramy – neděláme složité kroky, řešení rozložíme do série kroků jednoduchých
  - nejčastější a typický přístup



# Příklad vstupních dat

```
<?xml version="1.0"?>
<cenik>
  <zbozi id="zb001" druh="potravina">
    <nazev>Houska</nazev>
    <cena>1.70</cena>
  </zbozi>
  <zbozi id="zb004" druh="potravina">
    <nazev>Voda</nazev>
    <cena>7.50</cena>
  </zbozi>
</cenik>
```

# Rekurzivní styl (1)

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:template match="/">
    <html>
      <head><title>Ceník</title></head>
      <body>
        <h1>Ceník</h1>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
```

# Rekurzivní styl (2)

```
<xsl:template match="cenik">
  <table><xsl:apply-templates/></table>
</xsl:template>
<xsl:template match="zbozi">
  <tr><xsl:apply-templates/></tr>
</xsl:template>
<xsl:template match="nazev">
  <td><xsl:apply-templates/></td>
</xsl:template>
<xsl:template match="cena">
  <td align="right"><xsl:apply-templates/></td>
</xsl:template>
</xsl:stylesheet>
```

# Základní kostra transformace (2)

## ■ varianta 2: Řízení shora

- složité šablony využívající cykly k procházení obsahu vstupních prvků
- málo šablon, **xsl:apply-templates** se používá zřídka či vůbec ne
- sestavení výstupního stromu je explicitně řízeno stylovým předpisem
- kompaktnější, ale obtížněji modifikovatelné
- nepoužitelné pro rekurzivní data (různá hloubka vnoření)
- příliš se nepoužívá

# Řízený styl (1)

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
<xsl:template match="/">
  <html>
  <head><title>Ceník</title></head>
  <body>
  <h1>Ceník</h1>
  <table>
```

# Řízený styl (2)

```
<xsl:for-each select="descendant::zbozi">
```

```
  <tr>
```

```
    <td><xsl:value-of select="nazev"/></td>
```

```
    <td align="right"><xsl:value-of select="cena"/></td>
```

```
  </tr>
```

```
</xsl:for-each>
```

```
</table>
```

```
</body>
```

```
</html>
```

```
</xsl:template>
```

```
</xsl:stylesheet>
```

# Řazení

# Řazení

- pořadí uplatnění šablon standardně odpovídá pořadí prvků ve vstupním stromě, lze změnit
- `<xsl:sort select=“hodnota pro uspořádání“ />`
- lze jen uvnitř `xsl:for-each` nebo `xsl:apply-templates`
- příklad:  
`<xsl:for-each select=“descendant::zbozi“>`  
    `<xsl:sort select=“nazev“ />`  
    `<tr>...</tr>`  
`</xsl:for-each>`



# Atributy xsl:sort

- ovlivňují způsob řazení
  - `select=“výraz”` určuje hodnoty, podle nichž se třídí (implicitně uzel převedený na řetězec)
  - `data-type=“text” | “number”` abecední/číselné
  - `order=“ascending” | “descending”` vzestupné/sestupné
  - `lang=“kód jazyka”` jazyková pravidla třídění (naše „ch“)
  - `case-order=“upper-first” | “lower-first”` pořadí malých/velkých písmen

# Řazení podle více kritérií

- několik **xsl:sort** za sebou
- priorita klesá – rozhodne první, kde se uzly liší

```
<xsl:template match="cenik">  
  <table>  
    <xsl:apply-templates>  
      <xsl:sort select="nazev"/>  
      <xsl:sort select="cena" data-type="number"/>  
    </xsl:apply-templates>  
  </table>  
</xsl:template>
```

# Větvení

# Podmíněné zpracování

- `<xsl:if test="XPath výraz">`  
    *šablona*  
    `</xsl:if>`
- šablona bude použita, pokud boolovský XPath výraz vydá hodnotu true
- neexistuje else

# Větvení (1)

- podmíněný příkaz s více větvemi:

`<xsl:choose>`

`<xsl:when test="XPath výraz">`

*šablona*

`</xsl:when>`

`<xsl:when test="XPath výraz">`

*šablona*

`</xsl:when>`

...

`<xsl:otherwise>`

*šablona*

`</xsl:otherwise>`

`</xsl:choose>`

# Větvení (2)

- analogie  
if  
elseif  
...  
else
- uvnitř **xsl:choose** mohou být jen **xsl:when** a **xsl:otherwise**
- použije první *šablonu*, jejíž **test** vydal hodnotu true

# Příklad větvení

- řekněme, že <zbozi> obsahuje prvek <skladem> obsahující počet kusů na skladě

<xsl:template match="zbozi">

<tr>...

<xsl:choose>

<xsl:when test="skladem &lt; 10">

<td>nakupujte rychle</td>

</xsl:when>

<xsl:when test="skladem &lt; 100">

<td>mame toho dost</td>

</xsl:when>

znak „<“ entitou

<xsl:otherwise>

<td>akce!</td>

</xsl:otherwise>

</xsl:choose>

</tr>

</xsl:template>

# Proměnné



# Proměnné

- spíše konstanty – hodnotu nelze změnit
- definice – dvě možnosti:  
`<xsl:variable name="jméno">hodnota</xsl:variable>`  
`<xsl:variable name="jméno" select="XPath výraz"/>`
- použití: `$jméno` (ve výrazech: `{...}` a `xsl:value-of` vyhodnotí; `xsl:copy-of` zkopíruje)
- mohou být globální i lokální v šabloně (má přednost)
- vhodné pro opakující se konstrukce (snadná změna)

# Příklad – textová proměnná

**<xsl:variable name="tuladr">**

Technická univerzita v Liberci,<br />

Studentská 2,<br />

461 17 Liberec 1

**</xsl:variable>**

...

**<xsl:template match="zahlavi">**

  <div class="zahlavi">

**<xsl:apply-templates/>**

**<xsl:value-of select="\$tuladr"/>**

  </div>

**</xsl:template>**

# Příklad – mezery zleva

```
<xsl:variable name="sirkaPole">30</xsl:variable>
```

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

```
  <xsl:variable name="delka" select="string-length(.)"/>
```

```
  <xsl:variable name="vypln"  
    select="$sirkaPole - $delka"/>
```

```
    <!-- vloží mezery -->
```

```
    <xsl:value-of select="substring('
```

```
      <!-- za ně vlastní název -->
```

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

```
</xsl:template>
```

alespoň 30 mezer



```
      ,1,$vypln)"/>
```

# Výstupy

# XSLT a mezery (1)

- prázdné znaky: mezera, tabulátor, CR, LF
- XML zachází s volným místem dost liberálně
- `<xsl:strip-space elements="cenik zbozi"/>`  
vypustí z prvků cenik a zbozi prázdné textové uzly  
(volné místo mezi prvky)
- existuje i `xsl:preserve-space`, implicitní chování
- Pozor: `<nazev> Cukr </nazev>`  
neobsahuje prázdné textové uzly, ale textový uzel  
„ Cukr “

# XSLT a mezery (2)

- často přecházejí do výstupu uplatněním implicitní šablony (opis textu) na prázdné uzly
- explicitní vkládání do výstupu zajistí **xsl:text**  
**<xsl:text>**   **</xsl:text>**   **<!-- mezery -->**  
**<xsl:text>**  
**</xsl:text>**   **<!-- konec řádku -->**
- normalizace mezer pomocí **xsl:value-of** a XPath funkce **normalize-space()**  
**<xsl:value-of select="normalize-space(nazev)"/>**

# Odřádkování za výstupem (1)

- ze <zbozi> byly odstraněny prázdné textové uzly pomocí **xsl:strip-space**, chceme vypsát název a odřádkovat

- **varianta 1: doslovná**

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

```
  <em><xsl:value-of select="nazev"/></em>
```

```
  <xsl:text>
```

```
</xsl:text>
```

```
</xsl:template>
```

nevýhoda: **</xsl:text>** musí být na začátku řádku

# Odřádkování za výstupem (2)

- **varianta 2: použití proměnné**

```
<xsl:variable name="NL">
```

```
  <xsl:text>
```

```
</xsl:text>
```

```
</xsl:variable>
```

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

```
  <em><xsl:value-of select="nazev"/></em>
```

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

```
</xsl:template>
```

vyžaduje přípravu, použití je elegantní



# Odřádkování za výstupem (3)

- **varianta 3: proměnná + concat**

```
<xsl:variable name="NL">
```

```
  <xsl:text>
```

```
</xsl:text>
```

```
</xsl:variable>
```

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

```
  <xsl:value-of select="concat('<em>', nazev,  
                                '</em>', $NL)">
```

```
</xsl:template>
```

velmi kompaktní, ale méně přehledné

# Formát výstupu

- `<xsl:output method="metoda"/>`  
dostupné metody: xml (implicitní), html, text
- další atributy xsl:output:
  - `doctype-system="URI"`, `doctype-public="URI"` – určují obsah `<!DOCTYPE...>` výstupního dokumentu
  - `encoding="kódování"` – kódování výstupu
  - `omit-xml-declaration="yes" | "no"` – vypustit úvodní XML deklaraci
  - `indent="yes" | "no"` – odsazovat prvky



Realizováno za finanční podpory ESF a státního rozpočtu ČR  
v rámci v projektu *Zkvalitnění a rozšíření možností studia  
na TUL pro studenty se SVP* reg. č. CZ.1.07/2.2.00/29.0011

# Pojmenované šablony

# Pojmenované šablony

- pro opakované konstrukce, často parametrizovány

- definice:

`<xsl:template name="jméno">`

*šablona*

`</xsl:template>`

- použití:

`<xsl:call-template name="jméno"/>`

# Parametry

- definovány konstrukcí **xsl:param**, definovaná hodnota se chová jako implicitní
- lze používat stejně jako proměnné, ale při volání stylu či šablony jim lze nastavit hodnotu
- nastavení globálních parametrů závisí na procesoru
- nastavení lokálních parametrů šablony:  
**<xsl:with-param name="jméno">**  
*hodnota*  
**</xsl:with-param>**

# Příklad – buňka tabulky

```
<xsl:template name="bunkaTab">
  <xsl:param name="align">left</xsl:param>
  <td align="{ $align }"><xsl:apply-templates/></td>
</xsl:template>
<xsl:template match="nazev">
  <xsl:call-template name="bunkaTab"/>
</xsl:template>
<xsl:template match="cena">
  <xsl:call-template name="bunkaTab">
    <xsl:with-param name="align">right</xsl:with-param>
  </xsl:call-template>
</xsl:template>
```

# Krok stranou: current()

- často z dokumentu vybíráme prvky, v nichž se určitý údaj shoduje s jiným údajem aktuálního uzlu
- s výhodou lze využít XPath funkci **current()**, která vždy vrátí uzel, pro nějž byla volána šablona
- např: <utvar> obsahuje @zkratka a <nazev>, <osoba> obsahuje <pracoviste> se zkratkou útvaru, v šabloně pro osobu chceme jméno útvaru:  
**<xsl:value-of select=**  
**"//utvar[@zkratka = current()/pracoviste]/nazev"/>**

# Číslování



# Automatické číslování (1)

- čísla generuje šablona

`<xsl:number value="hodnota" format="formát"/>`

- bez atributů: generuje číslo podle pozice kontextového uzlu nebo pořadí v xsl:for-each

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

`<tr>`

`<td><xsl:number/></td>`

`<xsl:apply-templates/>`

`</tr>`

`</xsl:template>`

# Automatické číslování (2)

- hodnotou **value** je XPath výraz  
`<xsl:number/><xsl:number value="count(..zbozi)"/>`
- **format** určuje intuitivně podobu hodnoty
  - 1, 001 – arabské číslice
  - I, i – římské číslice
  - a, A – písmena
  - **format="I. "** – přidá k římskému číslu tečku a mezeru
- **grouping-separator**, **grouping-size** umožňují oddělovat řády (jakým znakem, po kolika)

# Víceúrovňové číslování (1)

- `level="multiple"` čísluje hierarchicky
- `count="XPath výraz"` které uzly počítat (implicitně jen uzly stejného jména a typu jako kontextový)
- ```
<xsl:template match="section">  
  <xsl:number format="1. " level="multiple"  
    count="chapter|section"/>  
  <xsl:apply-templates/>  
</xsl:template>
```

# Víceúrovňové číslování (2)

- **level="any"** čísluje všechny prvky průběžně
- atribut **from** umožňuje určit, jaké prvky restartují číslování
- příklad: číslování obrázků průběžně v jednotlivých kapitolách

```
<xsl:template match="figure">  
  <xsl:number format="1. " level="any" from="chapter"/>  
  <xsl:apply-templates/>  
</xsl:template>
```

# Další vývoj

# XSLT 2.0

- používá XPath 2.0
- více výstupů (více výstupních stromů):  
**<xsl:result-document href="{jmeno}.html" format="xhtml">**  
    **<xsl:value-of select="popis"/>**  
**</xsl:result-document>**
  - **href** typicky určuje jméno souboru
  - **format** odpovídá nějaké <output> definici

# Seskupování (1)

- **<xsl:for-each-group select=" *XPath výraz*"  
group-by=" *XPath výraz*">**  
*tělo*  
**</xsl:for-each-group>**
- vybere vrcholy vyhovující **select** a seskupí je do skupin se stejnou hodnotou **group-by**
- pro každou skupinu jednou uplatní *tělo*

# Seskupování (2)

- uvnitř `<xsl:for-each-group>` je k dispozici:
  - `current-group()` – členové aktuální skupiny
  - `current-grouping-key()` – (společná) hodnota `group-by` této skupiny
- příklad: chceme zaměstnance uspořádat po útvarech  
`<osoba id="zam1234">`  
    `<jmeno>Satrapa Pavel</jmeno>`  
    `<utvar>NTI</utvar>`  
`</osoba>`



# Seskupování (3)

```
<xsl:for-each-group select="osoba" group-by="utvar">
  <h2>
    <xsl:value-of select="current-grouping-key()"/>
  </h2>
  <ul>
    <xsl:for-each select="current-group()">
      <li><xsl:value-of select="jmeno"/></li>
    </xsl:for-each>
  </ul>
</xsl:for-each-group>
```

# XSLT 3.0

- standardizace probíhá
- podpora balíčků: `<xsl:package>` a `<xsl:use-package>` pro oddělenou kompilaci
- lepší podpora pro streaming (průběžné zpracování bez vytváření celého stromu v paměti)
- mapy – množiny dvojic klíč–hodnota (např. klíčem číslo, hodnotou jméno měsíce), analogie asociativních polí

# Iterace (1)

- **<xsl:iterate select="XPath výraz">**  
*tělo*  
**</xsl:iterate>**
- *tělo* se vyhodnotí pro každou položku vstupní sekvence (výsledek **select**)
- podobá se **for-each**, ale ve **for-each** se vyhodnocuje nezávisle, lze i paralelně
- v **iterate** se postupuje sekvenčně, položka může připravit hodnotu pro následující (parametry)

# Iterace (2)

- uvnitř lze **<xsl:next-iteration>**, která nemá žádný výstup, ale pomocí **<xsl:with-param>** připraví parametry pro další iteraci
  - nemá-li parametr přiřazenu hodnotu, zachová si pro příští iteraci svou stávající hodnotu
- **<xsl:break>** ukončí zpracování – žádná další položka vstupní sekvence už nebude zpracována

# Příklad iterací (1)

- máme informace o změnách na účtu:

<zmeny>

<zmena>10000</zmena>

<zmena>-2100</zmena> ...

</zmeny>

- chceme průběžné stavy účtu po změnách:

<ucet>

<stav>10000</stav>

<stav>7900</stav> ...

</ucet>

# Příklad iterací (2)

<ucet>

**<xsl:iterate select="zmeny/zmena">**

**<xsl:param name="stav" select="0.00" as="xs:decimal"/>**

**<xsl:variable name="novystav"**

**select="\$stav + xs:decimal(.)"/>**

**<stav><xsl:value-of select="\$novystav"/></stav>**

**<xsl:next-iteration>**

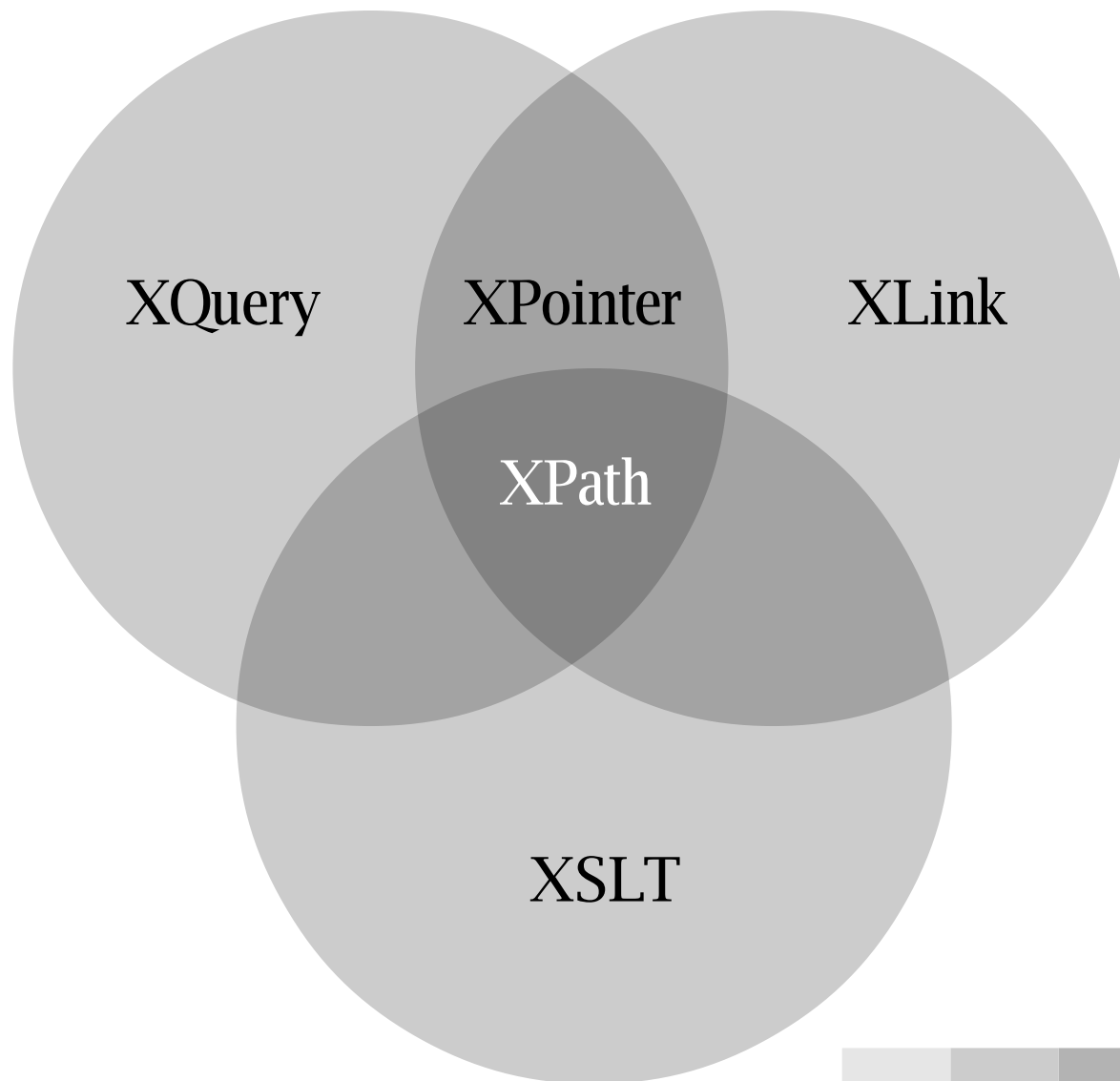
**<xsl:with-param name="stav" select="\$novystav"/>**

**</xsl:next-iteration>**

**</xsl:iterate>**

</ucet>

# Rodina XML konceptů





# ***XLink***





# XLink

- XML Linking Language
- obecný mechanismus pro definici odkazů v XML dokumentech
- nedefinuje jména prvků, ale mechanismy (atributy), jak prohlásit libovolný prvek za odkaz a popsat jeho strukturu a chování
- jmenný prostor XLink:  
<http://www.w3.org/1999/xlink>

# Jednoduchý odkaz

- základním atributem je **xlink:type**
  - určuje XLink typ prvku
  - hodnota **simple** pro jednoduché odkazy
  - atribut **xlink:href** obsahuje cíl
- příklad: doplníme ke zboží odkazy na stránky

```
<cenik xmlns:xlink="http://www.w3.org/1999/xlink">  
<zbozi id="zb78546578"> ...  
    <web  xlink:type="simple"  
        xlink:href="http://www.kdesi.cz/produkty/houska.html"/>  
</zbozi>  
</cenik>
```

# Atributy XLink

type	typ (z hlediska odkazů) dotyčného prvku
href	lokátor
title	význam odkazu (běžný text)
role, arcrole	úloha (obsahuje URI dokumentu)
show	jak prezentovat odkazovaný materiál
actuate	kdy jej prezentovat
label	definuje návěští
from, to	odkud kam vede

# Atributy chování

- **show** – požadovaný způsob prezentace
  - **new** – do nového okna
  - **replace** – nahradit obsah stávajícího okna
  - **embed** – vložit do stávajícího
  - ...
- **actuate** – kdy má dojít k následování odkazu
  - **onLoad** – při načtení dokumentu
  - **onRequest** – na žádost uživatele (kliknutí,...)
  - ...

# Rozšířené odkazy

- umožňují kombinovat několik zdrojů
- rodičovský prvek má **xlink:type="extended"**
- obsahuje potomky typů
  - **locator** – externí zdroj
  - **resource** – interní zdroj (přímo obsažen)
  - **arc** – pravidla pro přechod mezi zdroji
  - **title** – slovní popis

# Příklad

```
<vyrobek xlink:type="extended">  
  <info xlink:type="locator" xlink:label="produkt"  
    xlink:href="/prod/stan.xml" />  
  <soucast xlink:type="locator" xlink:label="komponenta"  
    xlink:href="/prod/tycka.xml"/>  
  <soucast xlink:type="locator" xlink:label="komponenta"  
    xlink:href="/prod/plachta.xml"/>  
  <slozeni xlink:type="arc"  
    xlink:from="produkt" xlink:to="komponenta"/>  
</vyrobek>
```

# Problém: implementace

- málo a nedokonalé
- přehled:  
<http://www.w3.org/XML/2000/09/LinkingImplementations.html>
- WWW klienti v podstatě nepodporují (velmi omezeně jen Mozilla & spol.)
- ostatní aplikace nevyužívají specifika odkazů



# **XPointer**





# XPointer

- XML Pointer Language
- umožňuje **odkazy na konkrétní části XML dokumentů**
- cíl: umožnit adresaci míst v dokumentu bez nutnosti jeho úpravy (nepotřebuje id)
- rozdělen do čtyř dokumentů:
  - rámeček – definuje základní pravidla pro schémata
  - schémata element(), xmlns() a xpointer()

# Rámec pro XPointer

- zavádí schéma jako formát odkazujících dat
- **zkrácené ukazatele**
  - obsahují jen jméno, vycházejí z XML identifikátorů (ID)  
`<h2 id="instalace">Postup instalace</h2>`  
XPointer: `instalace`
- **ukazatele založené na schématu**
  - *`schéma(odkazující_data)`*  
syntaxe a význam odkazujících dat závisí na schématu;  
může mít více částí (oddělovány prázdným místem),  
použije první úspěšnou část

# Schéma element()

- jednoduchá základní identifikace prvků
- základem jsou čísla oddělovaná „/“ – odkazují na n-tého potomka předchozího uzlu  
`element(/1/3)` – třetí potomek kořenového prvku
- lze používat identifikátory  
`element(zb002)` je totéž co zkrácený `zb002`  
`element(zb002/2)` – druhý potomek prvku s identifikátorem `zb002`

# Schéma xmlns()

- pro správnou reprezentaci jmenných prostorů v ukazatelích
- definuje prefixy, které lze používat v následujících XPathinterech

*xmlns(prefix=URI)*

- např.

*xmlns(zb=http://www.kdesi.cz/zbozi)*

a poté lze *xpointer(//zb:zbozi)*

# Schéma xpointer()

- nejsložitější, vypracován návrh, později opuštěn
- vychází z XPath a přidává možnost adresovat řetězce a další prvky à la DOM 2
- XPath identifikuje uzly, XPointer přidává
  - bod – místo bez obsahu a potomků (např. uvnitř řetězce či mezi dvěma sousedními uzly)
  - rozsah – část mezi dvěma body

# Vztah XLink a XPointer

- XLink definuje konstrukce obsahující odkazy (na jiné dokumenty či jejich části)
- XPointer lze použít jako obsah v attributech `xlink:href` při vytváření konkrétních odkazů
- XPointer je od části URI identifikující XML dokument oddělena znakem „#“ (jako v HTML)
- `/doc/manual.xml#xpointer(id('hw')/para[3])  
#instalace`



evropský  
sociální  
fond v ČR



EVROPSKÁ UNIE



MINISTERSTVO ŠKOLSTVÍ,  
MLÁDEŽE A TĚLOVÝCHOVY



OP Vzdělávání  
pro konkurenceschopnost



TECHNICKÁ  
UNIVERZITA  
V LIBERCI  
[www.tul.cz](http://www.tul.cz)

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Realizováno za finanční podpory ESF a státního rozpočtu ČR  
v rámci v projektu *Zkvalitnění a rozšíření možností studia  
na TUL pro studenty se SVP* reg. č. CZ.1.07/2.2.00/29.0011

# XQuery

# XQuery

- dotazovací jazyk pro XML informace (něco jako SQL pro databáze)
- umožňuje jen dotazování, data nelze měnit
- zdrojem nemusí být jen XML, ale i „podobná“ data
- cíl: sjednotit XML a databáze a ke kolekcím XML souborů přistupovat databázovým způsobem
- standardizace dokončena v lednu 2007



# Principy XQuery

- funkcionální jazyk – vše je výraz, jehož vyhodnocením vznikne určitá hodnota
- základní typy – stejné jako v XML Schema:
  - čísla
  - řetězce znaků (nelze do nich zasahovat)
  - pravdivostní hodnoty true/false
  - časové hodnoty
  - několik typů souvisejících s XML, podstatný je typ uzel


# Příklad na úvod – ceník

```
<html>
<head><title>Ceník</title></head>
<body>
<h1>Ceník</h1>
<table>
{ for $zbozi in /cenik/zbozi
  return
    <tr>
      <td>{$zbozi/nazev/text()}</td>
      <td align="right">{$zbozi/cena/text()}</td>
    </tr> }
</table>
</body></html>
```

# Syntax XQuery

- není XML (resp. existuje XML syntax jazyka označovaná jako XQueryX)
- připomíná PHP – míchají se opisované části výstupu s konstrukcemi XQuery
  - uzavřeny do složených závorek { ... }
- nejjednodušším XQuery výrazem je XPath výraz, výsledkem jsou vybrané uzly:  
`doc("cenik.xml")//zbozi[nazev="Rohlík"]`

# Vytvoření prvku (1)

- v XPath nelze vytvořit nový prvek, jen vybírat existující
- pro XQuery není problém, nejjednodušší je opisem:  
`<polozka>Rohlík</položka>`
- lze samozřejmě využívat informace ze vstupu:  
`<polozka>`  
`{$zbozi/nazev/text()}`  
`</polozka>`  
*proč je zde text()?*

# Vytvoření prvku (2)

- je-li třeba vytvořit název vytvářeného prvku, lze použít **element {jméno} {hodnota}**
- např. do výstupního XML vložíme prvek, jehož jméno odpovídá atributu **druh** prvku **zbozi** ve vstupním XML, obsah zůstane zachován:  
**for \$zbozi in /cenik/zbozi**  
**return**  
**element {\$zbozi/@druh} {\$zbozi/\*}**

# Vytvoření atributu (1)

- opisem nebo konstrukcí z údajů ze vstupu (automaticky se převádí na text)  
**<polozka cena="{ \$zbozi/cena }">  
    { \$zbozi/nazev/text() }  
</polozka>**
- prvky převzaté ze vstupu se přebírají se všemi potomky a atributy

# Vytvoření atributu (2)

- vložením do těla prvku

`<polozka>`  
    `{ $zbozi/@id }` ← *přidá prvku <polozka> atribut id*  
    `{ $zbozi/nazev/text() }`  
`</polozka>`

je-li vložen atribut ze vstupních dat, stane se atributem prvku, do nějž byl vložen – výsledek:

`<polozka id="zb001">Rohlík</polozka>`

- chcete-li jen hodnotu, použijte `data($zbozi/@id)` nebo `string($zbozi/@id)`

# Vytvoření atributu (3)

- pro dynamické určení jména atributu:  
**attribute {jméno} {hodnota}**
- existují i další konstruktory, používány méně často:
  - **text {hodnota}** pro vytvoření textového uzlu
  - **document {hodnota}** pro vytvoření dokumentového uzlu



# Výraz vložený v prvku

- jeho výsledek se stává obsahem obalujícího prvku, konkrétní efekt závisí na typu výsledku:
  - **prvek** (nebo sekvence prvků) – potomci
  - **atribut** – stane se atributem obalujícího prvku
  - **atomická hodnota** – převedena na řetězec a vložena jako textový uzel

# Seznamy

- XQuery vedle jednoduchých hodnot podporuje i seznamy (sekvence)
- uzavřené do (...), hodnoty oddělené čárkami
- nelze vnořovat
- k dispozici speciální operace a funkce
  - množinové: union, intersect, except
  - remove, index-of, count, sum, avg, max, min,...

# Vstupní data

- **implicitní**

- XPath výrazy začínající / předpokládají, že zpracovávaná XML data existují
- jak zadat zpracovávaný soubor závisí na implementaci
- `for $zbozi in /cenik/zbozi`

- **explicitní**

- XPath výrazy začínající `doc()` berou data z konkrétního souboru
- `for $zbozi in doc("cenik.xml")/cenik/zbozi`

# for

- **for *proměnná* in *hodnoty* return *výsledek***
  - *proměnná* nabude každou *hodnotu* a určí se pro ni *výsledek*
  - vyhodnocení v samostatných vláknech – lze paralelně
- příklad: seznam potravin

<ol>

```
{ for $zbozi in /cenik/zbozi[@druh="potravina"]  
  return  
  <li>{ data($zbozi/nazev) } </li> }
```

</ol>

# FLWOR výrazy

- základní stavební kámen XQuery
- For-Let-Where-Order-Return

```
for $odd in doc("oddeleni.xml")//deptno
let $zam := doc("zamest.xml")//employee[deptno = $odd]
where count($zam) >= 10
order by avg($zam/plat) descending
return
```

```
<megaoddeleni id="{ $odd }">
  <pocetlidi>{ count($zam) }</pocetlidi>
  <prumplat>{ avg($zam/plat) }</prumplat>
</megaoddeleni>
```

# FLWOR pravidla

- (for ... | let ... )+ (where ... )? (order by ... )? return ...
- for a let lze použít v libovolném počtu a pořadí, alespoň jedno být musí
- je povoleno jen jedno where
  - ale může obsahovat složitou podmínku s and a or
- jen jedno order by
  - ale může mít více kritérií pro řazení, oddělovat čárkami
- return je povinné

# let

- **let *\$proměnná* := výraz**
- vyhodnotí *výraz* a výsledek uloží do *proměnné*, ta se dál nemění – nabývá hodnotu jen při vytvoření
- *proměnná* může obsahovat uzel či posloupnost uzlů
- použití: často opakované výrazy, parametry
- *proměnné* mohou zjednodušit složité transformace
  - může být čitelnější než XSLT

# Příklad: Druhy zboží

```
<h1>Druhy zboží</h1>
```

```
<ol>
```

```
{
```

```
  let $druhy := /cenik/zbozi/string(@druh)
```

```
  for $d in distinct-values($druhy)
```

```
    return
```

```
      <li>{$d}</li>
```

```
}
```

```
</ol>
```



# let versus for

- **for \$zb in /cenik/zbozi**
  - vytvoří seznam vazeb \$zb na jednotlivá zboží z ceníku
  - return se vyhodnotí pro každou zvlášť (tolikrát, kolik zboží je v ceníku)
- **let \$zb := /cenik/zbozi**
  - vytvoří jednu vazbu \$zb na sekvenci prvků zboží z ceníku
  - return se vyhodnotí je jednou

# Řazení

- pokud má XQuery procesor k dispozici typy hodnot, bere na ně ohled
- netypované hodnoty řadí lexikograficky
  - lze změnit: **order by number(\$zbozi/cena)**
- nepovinné modifikátory za výrazem:
  - **ascending/descending**
  - **empty greatest/empty least**
  - **collation "URI"** – řadit podle národních specifik
  - **stable** (před order by) – u shodných zachovat pořadí

# Příklad: Druhy zboží

```
<h1>Druhy zboží</h1>
```

```
<ol>
```

```
{
```

```
  let $druhy := /cenik/zbozi/string(@druh)
```

```
  for $d in distinct-values($druhy)
```

```
    order by $d
```

```
    return
```

```
    <li>{$d}</li>
```

```
}
```

```
</ol>
```

# Podmíněný výraz

- *if (podmínka) then výsledek1 else výsledek2*  
v obvyklém významu

```
for $zbozi in doc("cenik.xml")/cenik/zbozi  
  return
```

```
...
```

```
  if ($zbozi/@druh = "potravina")  
    then <kategorie>potravina</kategorie>  
    else <kategorie>ostatni</kategorie>
```

# Struktura dokumentu

- XQuery dokument obsahuje nepovinný prolog a povinné tělo
- **prolog**
  - pouze deklarace
  - proměnných, funkcí, jmenných prostorů apod.
  - oddělovány středníky
- **tělo**
  - akční část dokumentu

# Příklad prologu

```
xquery version="1.0" encoding="UTF-8";  
declare variable $kapacita := 100;  
declare variable $zbozi := /cenik/zbozi;  
declare namespace zb = "http://www.tul.cz/nti/zbozi";
```

# Jmenné prostory (1)

- deklarace v prologu zavádí prefixy pro XQuery
- pokud se vytváří XML dokument, hrají roli i jeho deklarace, pozor na implicitní jmenný prostor – bude uplatněn i na prvky v XPath výrazech
- `<html xmlns="http://www.w3.org/1999/xhtml" ...>`  
...  
`{for $zbozi in /cenik/zbozi ... }`  
způsobí, že **cenik** a **zbozi** budou zařazeny do  
jmenného prostoru XHTML

# Jmenné prostory (2)

- řešení 1: používat pro své jazyky vlastní jmenné prostory

- `<html xmlns="http://www.w3.org/1999/xhtml" xmlns:zb="http://www.tul.cz/nti/zbozi" ...>`

...

`{ for $zbozi in /zb:cenik/zb:zbozi ... }`

- řešení 2: vyhnout se implicitním jm. prostorům

- `<h:html xmlns:h="http://www.w3.org/1999/xhtml" ...>`

...

`{ for $zbozi in /zb:cenik/zb:zbozi ... }`



# Uživatelské funkce (1)

- lze definovat vlastní funkce  
**declare function *jméno (parametry)***  
**{ *výraz* };**
- jméno musí být kvalifikované (s prefixem)
- výsledek není třeba explicitně definovat – tělem je výraz, ten se vyhodnotí a určí výsledek funkce
- použití stejné jako u vestavěných funkcí:  
***jméno(parametry)***

# Příklad: Připočteme DPH

```
declare namespace zb = "http://www.tul.cz/nti/zbozi";
```

```
declare variable $zb:dphkoef := 1.21;
```

```
declare function zb:dph ( $cena )  deklarace  
{
```

```
    $cena * $zb:dphkoef  
};
```

 *u proměnné není prefix povinný, zařazením do společného prostoru zdůrazňují jejich vztah*

```
...
```

```
for $zbozi in /cenik/zbozi
```

```
    ...  
    <td align="right"> {zb:dph(data($zbozi/cena))}</td>
```

*použití*

# Uživatelské funkce (2)

- tělo může být samozřejmě výrazně složitější, včetně rekurze
- příklad: vydá prvek sám + všechny jeho potomky

```
declare function muj:descendant-or-self ($x)
```

```
{
```

```
    $x,
```

```
    for $y in $x/*
```

```
        return muj:descendant-or-self($y)
```

```
}
```

# Moduly (1)

- složitější XQuery lze rozložit do modulů
- modul obsahuje jen preambuli
- syntaxe lehce pozměněna:
  - začíná  
**module namespace prefix = "URI";**
  - definuje cílový jmenný prostor
  - všechny proměnné a funkce definované modulem musí být v tomto prostoru

# Příklad modulu

```
module namespace zb = "http://www.tul.cz/nti/zbozi";  
declare variable $zb:dphkoef := 1.21;  
declare function zb:dph ( $scena )  
{  
    $scena * $zb:dphkoef  
};
```

# Moduly (2)

- použití modulu:  
**import module namespace *prefix* = "URI"**  
**at "*soubor*"**
- *URI* musí odpovídat deklaraci v modulu, *prefix* se může změnit
- následně lze používat proměnné a funkce definované v *souboru* (s příslušným *prefixem*)
- lze použít libovolný počet modulů

# Typy (1)

- nejsou nezbytné
- lze používat **typy z XML Schema**
  - zpřístupní deklarace jmenného prostoru  
`declare namespace xsd =`  
`"http://www.w3.org/2001/XMLSchema";`
  - přetypování hodnoty: `typ(hodnota)`  
`xsd:decimal($cena) * $zb:dphkoef`

# Typy (2)

- **vlastní typy**

- importovat XML Schema definici:

- `import schema namespace zb =`

- `"http://www.tul.cz/nti/zbozi" at "cenik.xsd";`

- nebo

- `import schema default element namespace "" at "...";`

- následně lze používat prvky, atributy a typy ze schématu

- **testování typů:**

- `instance of`, `castable as`

- `typeswitch`



# Textový výstup (1)

- v principu možný – důsledně do výstupu posílat textové uzly nebo převádět na řetězce znaků
- např. ceník v textové podobě:

```
for $zbozi in /cenik/zbozi  
  return (  
    $zbozi/nazev/text(),  
    " ... ",  
    $zbozi/cena/text(),  
    "&#xa;"  
  )
```

# Textový výstup (2)

- formátování pomocí konstant
- může vyžadovat potlačení XML prologu a/nebo dalších složek
  - specifické pro konkrétní implementaci
  - např. Saxon:

```
declare namespace saxon = "http://saxon.sf.net/";  
declare option saxon:output "indent=no";  
declare option saxon:output "method=text";
```

# Implementace

- poměrně hojné, i open source, například:
  - **Saxon** (též XSLT procesor) – [saxon.sourceforge.net](http://saxon.sourceforge.net), 100% úspěch v testu compatibility
  - **Galax** – [www.galaxquery.org](http://www.galaxquery.org)
  - **Oracle Berkeley DB XML** – XML databáze
  - **Qizx** – <http://www.axyana.com/qizxopen/>
- test compatibility:  
<http://www.w3.org/XML/Query/test-suite/XQTSReport.html>

# XQuery versus XSLT

- podobné schopnosti, stejný datový model vycházející z XPath (strom dokumentu, uzly,...)
- XSLT implementace bývají optimalizovány na transformaci celých dokumentů, XQuery spíše na vybírání fragmentů
- XQuery více připomíná programovací jazyk
- syntax XQuery bývá kompaktnější



evropský  
sociální  
fond v ČR



EVROPSKÁ UNIE



MINISTERSTVO ŠKOLSTVÍ,  
MLÁDEŽE A TĚLOVÝCHOVY



OP Vzdělávání  
pro konkurenceschopnost



TECHNICKÁ  
UNIVERZITA  
V LIBERCI  
[www.tul.cz](http://www.tul.cz)

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Realizováno za finanční podpory ESF a státního rozpočtu ČR  
v rámci v projektu *Zkvalitnění a rozšíření možností studia  
na TUL pro studenty se SVP* reg. č. CZ.1.07/2.2.00/29.0011

# API pro XML

# XML a programování

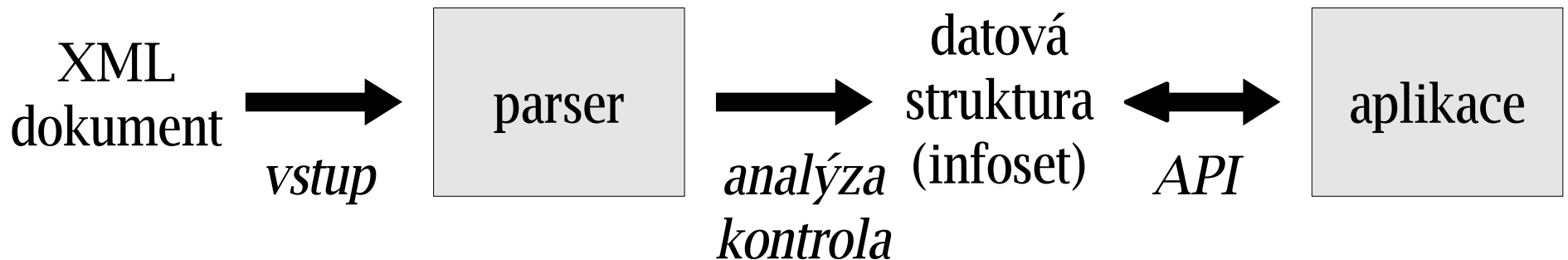
- **rukodělně**

- XML dokument je textový soubor – lze zpracovávat běžnými textovými funkcemi, regulárními výrazy apod.
- pracné, ale může být velmi rychlé
- použitelné, pokud je struktura souboru jednoduchá

- **využití existující knihovny (API)**

- méně práce (ale seznámení s knihovnou stojí čas)
- sdílení zkušeností s ostatními programátory
- doporučená cesta

# Vstup XML dat – parser



- parser zajistí načtení, syntaktickou analýzu a kontrolu správnosti XML dokumentu
- vytvoří datovou strukturu odpovídající obsahu (typicky strom)

# Základní typy API

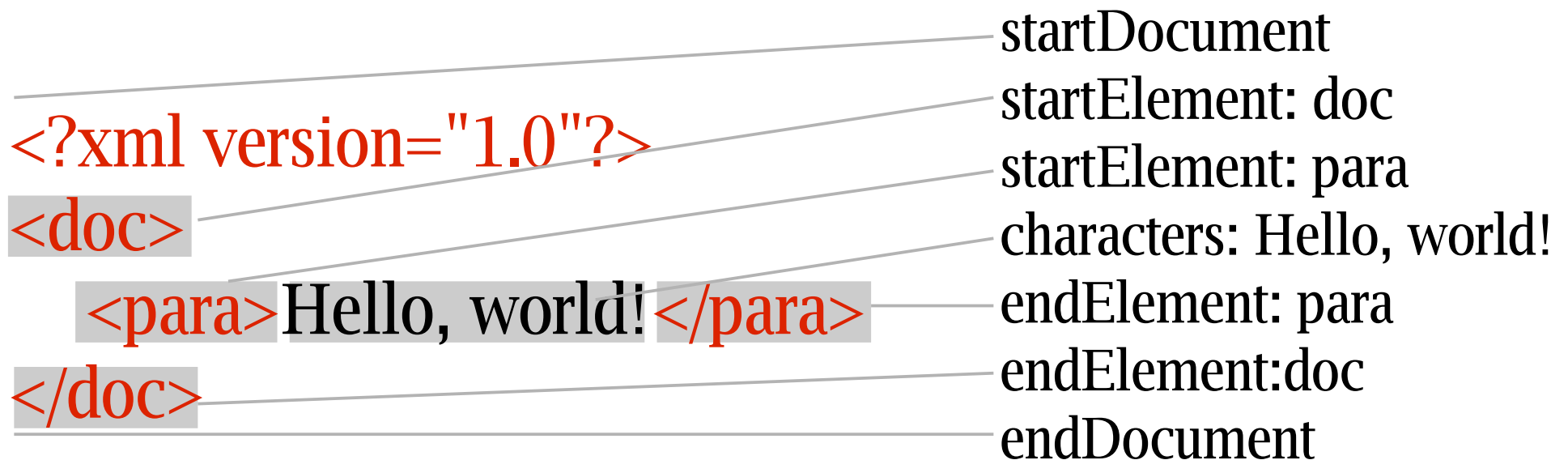
- **řízené událostmi (SAX)**
  - kdykoli při čtení dokumentu dojde k důležité události (např. začátek prvku), parser volá obslužnou funkci
  - minimální režie (nevytváří strom v paměti)
  - veškeré souvislosti si musí udržovat aplikace
- **stromové (DOM)**
  - převede dokument na strom v paměti
  - volný přístup ke všem datům
  - režie úměrná velikosti dokumentu



# SAX

- **Simple API for XML**
- jeden průchod dokumentem, významné jevy při čtení jsou přenášeny jako **události**, typicky:
  - startDocument, endDocument
  - startElement, endElement
  - characters
- návod, odkazy na konkrétní parsery apod.:  
[www.saxproject.org](http://www.saxproject.org)

# Příklad SAX událostí



# Nevýhody SAX

- SAX je „tlačné“ (push) rozhraní – aplikace je de facto řízena parserem
  - „tažná“ (pull) rozhraní používají podobný způsob práce s dokumentem, ale o události si říká aplikace, např. opakovaným voláním funkce next()
- je jednosměrný – pouze pro čtení dokumentů
  - žádná podpora pro XML výstup
  - výstup je realizován jako běžný text

# StAX

- **Streaming API for XML**
- **kurzor** označuje aktuální pozici v XML dokumentu
  - ukazuje na logické prvky (textový uzel, značku prvku,...)
  - pohybuje se jen vpřed, nesmí couvat
  - pohyb kurzoru řídí aplikace – metodou next()
  - získávání informací metodami getName(), getText(),...
- základní rozhraní: třídy XMLStreamReader a XMLStreamWriter
- [stax.codehaus.org](http://stax.codehaus.org)

# Příklad – jména prvků

```
while (true) {  
    int event = parser.next();  
  
    if (event == XMLStreamConstants.END_DOCUMENT) {  
        parser.close();  
        break;  
    }  
  
    if (event == XMLStreamConstants.START_ELEMENT) {  
        System.out.println(parser.getLocalName());  
    }  
}
```

viz <http://www.xml.com/pub/a/2003/09/17/stax.html>

# DOM

- **Document Object Model**
- stromová reprezentace dokumentu (hierarchie objektů odpovídajících částem XML – prvkům, atributům atd.)
- všechny informace přístupné naráz – lze využívat libovolně a opakovaně (ale vyžaduje paměť a čas)
- vytvořilo W3C jako univerzální nástroj
  - používá i JavaScript pro manipulaci WWW stránkou

# XOM

- **XML Object Model**
- demonstrace převoditelnosti přístupů – postaví strom dokumentu, sám ale využívá SAX
- snaha o jednoduché, snadno zvládnutelné a efektivní rozhraní
- umožňuje zpracovávat dokument po částech
- [www.xom.nu](http://www.xom.nu)

# API vázané na data (data binding)

- podobný DOMu – vytváří v paměti hierarchii odpovídající dokumentu
- objekty odpovídají „na míru“ použitému XML jazyku, nikoli obecným součástí XML
  - objekty Cenik, Zbozi atd. nikoli Element, jehož atribut Name má hodnotu Cenik či Zbozi
- typicky staví na **kompilaci schématu** – připraví třídy
- např. **XMLBeans** ([xmlbeans.apache.org](http://xmlbeans.apache.org))



# Transformační/dotazovací API

- aplikační interface pro XSLT, XPath a podobně
- hlavní kód většinou leží mimo API, knihovna jen zajišťuje tlumočení mezi aplikací a kódem provádějícím XML operace
- např.  
**TrAX** (<http://xml.apache.org/xalan-j/trax.html>)  
**Jaxen** ([jaxen.codehaus.org](http://jaxen.codehaus.org))



# **Příklady XML jazyků**



# Konkrétní XML jazyky (formáty)

- XML je nástroj pro definici jazyků pro konkrétní aplikace a služby
- zveřejněných jazyků s ambicí na neutralitu a pozici de facto či de iure standardu jsou stovky – viz [http://en.wikipedia.org/wiki/Category:XML-based\\_standards](http://en.wikipedia.org/wiki/Category:XML-based_standards)
- následují vybrané příklady

# XHTML

- **eXtensible HyperText Markup Language**
- následník HTML
- jazyk webových stránek
- nejpoužívanější jazyk založený na XML
- notorické spory vizuálního a sémantického tábora vyřešila kombinace XHTML+CSS
- [validator.w3.org](http://validator.w3.org)

# RSS

- **Rich Site Summary (RDF Site Summary, Really Simple Syndication)**
- pro oznamování novinek na WWW serverech
- houfně používají zpravodajské servery a blogy
- sledovat lze specializovaným programem, WWW klientem i na integrujících stránkách (Google Reader)
- velmi jednoduchý jazyk, přesto existuje několik nekompatibilních verzí

# Příklad RSS

- tématický kanál = soubor

```
<rss version="0.91">
```

```
<channel>
```

```
  <title>Satrapovy novinky</title>
```

```
  <link>http://www.kai.tul.cz/~satrapa/rss.xml</link>
```

```
  <description>Mé fiktivní zpravodajství pro studenty.</description>
```

```
  <language>cs</language>
```

```
<item>
```

```
  <title>Vypsány termíny z Počítačových sítí</title>
```

```
  <link>http://stag.tul.cz/</link>
```

```
  <description>Termíny ze sítí jsou ve STAGu, zapisujte se.</description>
```

```
</item> ...
```

```
</channel>
```

```
</rss>
```

# ATOM

- Atom Syndication Format
- nástupce RSS
- nese více informací (různé povinné položky)
- větší možnosti pro obsah – různé formáty, včetně binárních

# DocBook

- pro technickou dokumentaci (postupně se protlačuje do pozice de facto standardu)
- původně v SGML, více se používá XML verze
- spousta prvků – definována podmnožina Simplified DocBook (o něco menší spousta prvků)
- k dispozici konverze (na bázi XSLT) do řady formátů – (X)HTML, PDF, RTF,...
- [www.docbook.org](http://www.docbook.org)



# Příklad DocBook

```
<book id="kniha-pokus">
  <title>Minimalistická kniha</title>
  <chapter id="kap-uvod">
    <title>Úvod</title>
    <para>No nazdar!</para>
    <para>Začal jsem psát knihu, snad to dobře dopadne...</para>
  </chapter>
  <chapter id="kap-xml">
    <title>XML</title>
    <para>V první kapitole se seznámíme s XML.</para>
  </chapter>
</book>
```

# OpenDocument format (ODF)

- připravila OASIS (stejně jako DocBook), nyní ISO standard
- původně jej vytvořil a používá **OpenOffice.org**, používá i KOffice
- formát pro kancelářské aplikace
- může být jednoduchý XML soubor, častěji ale ZIP archiv obsahující řadu adresářů a souborů obsahujících jednotlivé prvky dokumentu
- ISO standard

# Office Open XML

- vytvořil Microsoft pro nové verze svého balíku Office (od 2007)
- přímý konkurent ODF
- po řadě kontroverzí přijat jako ISO standard

# MathML

- Mathematical Markup Language, W3C doporučení
- původní cíl: matematické vzorce do webu
- typický představitel masivního značkování

- $y=|x|$

`<math xmlns="http://www.w3.org/1998/Math/MathML">`

`<ci>y</ci><mo>=</mo>`

`<apply>`

`<abs/>`

`<ci>x</ci>`

`</apply>`

`</math>`

# SVG

- **Scalable Vector Graphics**
- W3C doporučení
- dvojrozměrná vektorová grafika (ale může obsahovat i rastrová data)
- podporuje řada prohlížečů (nativně či pomocí plug-inů) – vektorová grafika pro web
- implementace: [www.svgi.org](http://www.svgi.org)  
doporučuji **Inkscape** ([www.inkscape.org](http://www.inkscape.org))

# Příklad

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"  
    "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">  
  
<svg width="100%" height="100%"  
    version="1.1" xmlns="http://www.w3.org/2000/svg">  
  
    <rect width="300"  
        height="100"  
        style="fill:rgb(0,0,255);  
            stroke-width:1;stroke:rgb(0,0,0)"/>  
  
</svg>
```

# XML-RPC

- **Remote Procedure Call**
- jednoduchý mechanismus pro síťovou spolupráci aplikací – klient volá podprogram na serveru
- přenosovým protokolem HTTP
- data formátována v minimalistickém XML jazyce (méně než 10 datových typů)

# Příklad XML-RPC

```
<?xml version="1.0"?>
<methodCall>
  <methodName>
    priklad.getPSCMesto
  </methodName>
  <params>
    <param>
      <value>
        <i4>46000</i4>
      </value>
    </param>
  </params>
</methodCall>
```



```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <string>Liberec</string>
      </value>
    </param>
  </params>
</methodResponse>
```



# SOAP

- **Simple Object Access Protocol**
- následník XML-RPC, univerzální vrstva pro výměnu zpráv ve webových aplikacích
- různé přenosové modely a mechanismy, nejběžnější je RPC – požadavek a odpověď kódovány do SOAP zpráv

# Příklad – SOAP dotaz

```
<soap:Envelope  
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">  
  <soap:Body>  
    <getPSCMesto xmlns="http://kdesi.cz/psc">  
      <psc>46000</psc>  
    </getPSCMesto>  
  </soap:Body>  
</soap:Envelope>
```

# Příklad – SOAP odpověď

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getPSCMestoResponse xmlns="http://kdesi.cz/psc">
      <getPSCMestoResult>
        <jmeno>Liberec</jmeno>
        <psc>46000</psc>
        <kraj>Liberecky</kraj>
      </getPSCMestoResult>
    </getPSCMestoResponse>
  </soap:Body>
</soap:Envelope>
```

# XUL

- **XML User Interface Language**
- definuje uživatelské rozhraní Mozilly
- obsahuje prvky pro popis běžných součástí GUI (okna, tlačítka, menu)
- využívá existující technologie (CSS, JavaScript,...)
- MS Windows mají **XAML** (Extensible Application Markup Language)



# **Alternativ XML**



# JSON

- **JavaScript Object Notation**
- využíván především pro on-line aplikace
- syntax vychází z JavaScriptu
- existují parsery pro řadu jazyků – viz [json.org](http://json.org)
- **podobné:** textový zápis, hierarchické uspořádání
- **odlišné:** kratší (nemá koncové značky), obsahuje pole, nemá atributy, chybí nadstavbový ekosystém
- ECMA-404, RFC 7159

# Příklad

```
{ "cenik" : [  
  { "nazev" : "Houska",  
    "cena" : 1.70 },  
  { "nazev" : "Voda",  
    "cena" : 7.50 }  
}]
```

# JSON typy

- řetězec znaků – uzavřen do " ... "
- číslo – neřeší varianty
- pravdivostní hodnota – true, false
- objekt – uzavřen do { ... }, nezáleží na pořadí
- pole – uzavřeno do [ ... ], indexováno od 0



# JSON Schema

- cíl: umožnit strojovou kontrolu korektnosti dat
- standardizace teprve probíhá
- vychází z JSON syntaxe, definuje názvy položek a typy jejich hodnot
- obvyklý vývoj – JSON je populární pro svou jednoduchost, ale uživatelé chtějí další schopnosti

# Příklad

```
{  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "Ceník",
  "descripton" : "Informace o nabídce a cenách zboží",
  "type" : "object",
  "properties" : {
    "cenik" : {
      "type" : "array",
      "items" : {
        "type" : "object",
        "properties" : {
          "nazev" : { "type" : "string" },
          "cena" : { "type" : "number", "minimum" : 0 } }
        "required" : [ "nazev", "cena" ]
      }
    }
  }
}
```

# YAML

- **YAML Ain't Markup Language** (původně Yet Another Markup Language)
- inspirováno programovacími jazyky, XML a formátem elektronické pošty
- založeno na odsazování
  - sourozenci odsazeni stejně od levého okraje
  - potomci odsazeni více

# Příklad

cenik:

- **nazev:** Houska  
**cena:** 1.70
- **nazev:** Voda  
**cena:** 7.50

# YAML typy (1)

- **řetězec znaků** – nevyžaduje uvozovky
- **číslo** – rozlišuje celá a s plovoucí desetinnou čárkou
- **pravdivostní hodnota** – true, false
- **čas** – podle ISO 8601, příp. s mezerami  
2015-01-14t16:23:37.15-02:00
- **datum** – v pořadí rok-měsíc-den podle ISO 8601  
2015-01-14

# YAML typy (2)

- **pole** – položky zahájeny pomlčkou a mezerou:

- první
  - druhá
  - třetí

lze i kompaktně [ první, druhá, třetí ]

- **asociativní pole (mapy)** – zápis *klíč: hodnota*

nazev: Houska

cena: 1.70

lze i kompaktně { nazev: Houska, cena: 1.70 }

# YAML typy (3)

- lze i definovat vlastní
- obvykle se určují automaticky, ale lze i explicitně převést na daný typ **!!float 10**
- nemá prostředky pro definici schématu, ale vznikají doprovodné prostředky
  - Rx
  - Kwalify

# Srovnání

- vlastní XML lze nahradit snadno
  - moc toho neobsahuje, totéž lze i efektivněji
- kontrolu dat už obtížněji
  - nabídka nástrojů pro popis a validaci datových struktur je poměrně omezená
- transformace/dotazování velmi obtížně
  - univerzální nástroje XSLT či XQuery nemají alternativu
  - pouze vlastní aplikace na míru pro konkrétní použití