XML schémata

Jiří Kosek

XML schémata

Jiří Kosek

Datum vydání 24. května 2013

Dokument je zkrácenou verzí školicích materiálů používaných během školení XML schémata[1]. Školení se mj. věnuje i problematice databindingu a využití informací ze schématu uvnitř aplikací.

Právě čtete tutoriál, který popisuje základní znalosti nezbytné pro vytváření schémat XML dokumentů v jazycích XML Schema, Relax NG, DTD a Schematron. Dokument je v neustálém procesu vývoje a naleznete-li v textu nějakou chybu, dejte mi o ní prosím vědět[2].

Dokument byl připraven v systému DocBook[3] a kromě HTML verze[4] je dostupný ve formátech PDF pro tisk[5] a nápovědy HTML Help[6].

Abstrakt

Popularita XML je do velké míry dána jeho flexibilitou – každý uživatel si může podle potřeby vytvořit sadu značek umožňující přidání potřebné míry sémantiky do dokumentů XML. Z mnoha praktických důvodů je však rozumné takto vytvořené nové značkovací jazyky jednoznačně a formálně popsat. Standard XML proto přímo obsahuje DTD jako nástroj pro definici nových značkovacích jazyků. DTD pochází ještě z dob SGML a plně dostačuje při použití XML jako strukturovaného formátu pro přípravu dokumentů v procesu elektronického publikování.

Největší uplatnění však XML dnes paradoxně nalézá při výměně strukturovaných dat mezi informačními systémy, ve webových službách a obecně při realizaci B2B scénářů. Pro tyto aplikace jsou však DTD již nedostačující. Chybí zejména podpora datových typů. Postupně vzniklo několik jazyků pro popis schématu XML dokumentů. Mezi jejich společné rysy patří podpora datových typů, podpora jmenných prostorů a syntaxe založená na XML. Výsadní postavení přitom zaujímá standard XML schéma, který pochází z dílny konsorcia W3C a je podporován předními IT firmami.

Tutoriál posluchače seznámí s principy použití schémat pro XML dokumenty, s jejich historií a nejpoužívanějšími schémovými jazyky.

[1] http://xmlguru.cz/skoleni/xmlschema
[2] mailto:jirka@kosek.cz
[3] http://docbook.cz
[4] http://www.kosek.cz/xml/schema/
[5] http://www.kosek.cz/xml/schema/xmlschema.pdf
[6] http://www.kosek.cz/xml/schema/xmlschema.chm

Obsah

Seznam obrázků

Seznam tabulek

Seznam příkladů

Kapitola 1. Úvod

Jazyk XML se i přes mnoho skeptických hlasů výrazně prosadil a mezi IT technologiemi již zaujímá pevné místo. K hlavním oblastem jeho využití patří výměna dat mezi informačními systémy. XML se prosadilo především díky své jednoduchosti a flexibilitě, která spočívá v možnosti snadného zachycení běžných datových struktur do podoby XML zprávy.

Počáteční nadšení z toho, že XML dokumenty mohou mít libovolnou strukturu a že v nich lze používat libovolné značky, však brzy vystřídá vystřízlivění. Příliš volnosti škodí a v praxi je potřeba vyměňovat si XML zprávy s nějakou předem definovanou podobou, a ne jen nahodile pojmenované směsice elementů XML. A to je právě oblast, kde do hry vstupují jazyky pro popis schématu dokumentu XML. Tyto jazyky umožňují definovat jaké elementy a atributy můžeme v XML dokumentu používat, jak je lze vzájemně kombinovat, co mohou obsahovat. Schéma dokumentu XML vlastně definuje nový značkovací jazyk, který má syntaxi XML, ale používá námi vytvořené značky. Takto definovaných jazyků existují na světě tisíce, z těch nejznámějších jmenujme například WML (wapové stránky pro mobilní telefony), XHTML (nástupce jazyka HTML), SVG (vektorový grafický formát), MathML (zápis matematických výrazů) a DocBook (tvorba dokumentace). Schéma dokumentu XML tak plní podobnou funkci jako schéma databáze ve světě relačních databázových systémů.

1.1 Význam schémat

Primární význam schémat leží v jejich použití pro *formální definici* značkovacích jazyků, nebo chceteli datových formátů, založených na XML. Výhoda formalizované definice spočívá v tom, že je jednoznačná a znemožňuje různé interpretace, na rozdíl od definice popsané v přirozeném jazyce. Všichni, kdo chtějí dokumenty XML odpovídající danému schématu zpracovávat, proto vědí, jak mohou dokumenty vypadat.

Vzhledem k tomu, že schéma jednoznačně definuje, jak může dokument XML vypadat, můžeme jej samozřejmě použít i pro *validaci*. Ostatně se jedná asi o nejčastější použití schémat. Validace je proces, při kterém se ověřuje, zda nějaký konkrétní dokument XML vyhovuje všem omezením definovaným ve schématu. Můžeme si tak například ověřit, zda dokument, který nám někdo zaslal, opravdu vyhovuje formátu, na němž jsme se předtím domluvili. Další výhodou validace je ulehčení vývoje aplikací. Ty nemusí při čtení dat z XML provádět kontrolu správného vstupu, protože většinu případných chyb odhalí validace.

Některé jazyky pro popis schématu umožňují pro obsah jednotlivých elementů a atributů určit jejich *datový typ*, jako číslo, řetězec, datum apod. Během validace pak může být jednotlivým částem XML dokumentu přiřazen jejich datový typ. Aplikace pak při čtení dokumentu pracuje rovnou s otypovaným daty a ne jen s textovými řetězci, jak je ve světě XML běžné. Z abstraktního datového modelu XML (tzv. infoset) se po přiřazení typů stane PSVI (Post-Schema Validation Infoset).

Schéma dokumentu může být využito jako zdroj pro vytvoření odpovídajícího objektového modelu včetně kódu, který jej implementuje. Princip *databindingu* spočívá v tom, že pomocí automaticky vygenerovaného kódu ze schématu můžeme pak v aplikaci snadno manipulovat s dokumenty XML, které budou reprezentovány jako sada objektů v paměti.

V neposlední řadě slouží schéma jako *dokumentace* pro značkovací jazyk, který definuje. Navíc pro mnoho jazyků pro popis schématu existují nástroje, které z původního schématu vygenerují přehlednou dokumentaci například v podobě hypertextově provázaných webových stránek.

1.2 Historický vývoj jazyků pro popis schématu dokumentu

Přímo specifikace jazyka XML [2] popisuje DTD (Document Type Definition), což je jazyk pro popis schématu dokumentu pocházející ještě z jazyka SGML. DTD díky tomu mají jednu velkou výhodu – podporuje je velké množství aplikací. To je však současně jejich jedinou výhodou.

S rozšiřováním XML začínaly být nedostatky DTD stále patrnější. Za největší nedostatek můžeme označit nulovou podporu pro jmenné prostory. Jmenné prostory jsou mechanismus, který umožňuje v jednom dokumentu XML kombinovat více sad značek – např. do webové stránky v XHTML můžeme vložit obrázek v SVG a matematickou rovnici zapsanou v MathML.

Druhým nedostatkem je nemožnost určení datového typu pro obsah elementů a atributů. SGML, a později XML, bylo primárně navrženo pro značkování dokumentů textové povahy jako jsou knihy, technická dokumentace, články, webové stránky apod. V těchto dokumentech je v podstatě vše text. XML se však začalo masově používat i na výměnu strukturovaných dat mezi informačními systémy. V dokumentech jako faktura, stav zboží na skladě nebo kurzový lístek už zdaleka vše není jen text – kromě toho tu jsou číselné hodnoty, měnové údaje, data apod. Aby byla validace skutečně účinná, je potřeba omezit nejen strukturu vzájemného zanoření elementů, což DTD ještě umí, ale i obsah těchto elementů (a případně i atributů) jejich datovým typem.

Třetí slabina DTD již není tak jednoznačná. Pro zápis DTD se používá jednoduchá syntaxe připomínající regulární výrazy. Např. definovat element pro fakturu, která se skládá z dodavatele, odběratele a několika položek, je velmi jednoduché:

```
<!ELEMENT faktura (dodavatel, odberatel, polozka+)>
```

Problém je však v tom, že tento jednoduchý a kompaktní zápis používá syntaxi, která se nikde jinde v XML nepoužívá. Proto snad všechny jazyky pro popis schématu dokumentu, které přišly až po DTD, pro zápis schématu používají přímo XML. Zápis je pak sice delší a mnohdy méně přehlednější, ale se schématem lze pracovat stejně jako s jakýmkoliv jiným dokumentem XML – můžeme z něj například snadno pomocí XSLT generovat dokumentaci. Otevírá se tak i prostor pro grafické návrháře schémat, podobně jako lze použít CASE nástroje pro vytvoření datového modelu pro relační databázi.

Předchozí tři důvody byly natolik závažné, že se již v době vzniku standardu XML začalo pracovat na několika nových schémových jazycích, které měly překonat výše popsaná omezení. Většina z těchto jazyků nepřežila více než jen několik málo let nebo se jedná o proprietární jazyky používané pouze omezeným počtem firem.

Významnou roli však hrál jazyk XML-Data[1], který společenství několika firem zaslalo na začátku roku 1998 do W3C jako návrh. XML-Data nebyl ani příliš striktně vázán na XML, šlo o jazyk, který umožňoval definovat charakteristiky tříd objektů. Pro ověření použitelnosti byl však návrh příliš složitý, a proto vznikla ještě jeho zjednodušená verze XDR (XML-Data Reduced)[2]. Právě XDR ve svých produktech používal Microsoft, dokud nebyl přijat nějaký všeobecně uznávaný standard. Tím se v květnu 2001 stal jazyk XML Schema[3] přijatý jako doporučení konsorcia W3C.[4]

XML schéma se samozřejmě vyvinulo z XML-Data a XDR. Po DTD je to dnes nejpoužívanější schémový jazyk a podporují jej všechny velké firmy a mnoho projektů open-source. Jediným problémem XML

[1] http://www.w3.org/TR/1998/NOTE-XML-data/
[2] http://www.ltg.ed.ac.uk/~ht/XMLData-Reduced.htm
[3] http://www.w3.org/TR/xmlschema-0/
[4] Dále v textu mu budeme familiárně říkat jen XML schéma. V internetových diskusních skupinách se XML Schema často označuje pouhou zkratkou WXS vzniklou z W3C XML Schema, nebo XSD podle nejčastěji používané přípony souborů.

schémat je poněkud složitá specifikace a některá poněkud temná zákoutí jazyka. Nicméně po přečtení tohoto materiálu byste měli znát dost na to, abyste mohli XML schéma uspokojivě využívat.

XML schémata jsou sice pro mnoho věcí zcela dostačující a použitelná, ale přece jen jim chybí určitá elegance. Jako reakce na tento stav vznikl další schémový jazyk Relax NG[5]. Jeho popularita v poslední době hodně roste, protože je velmi jednoduchý na naučení a na použití a počet jeho implementací se rozšiřuje. Relax NG je od roku 2001 standardizován na půdě sdružení OASIS[6] a od roku 2003 je i normou ISO.

Výše popsané jazyky přistupují k tvorbě schématu systematicky – v případě DTD a Relax NG vytváříme gramatiku nového jazyka, XML schémata pak definují typy, které musí dokument obsahovat. Někdy však může být výhodné využití jiného a méně systematického přístupu, který však umí postihnout zcela jiné aspekty. Jedním z takových jazyků je Schematron[7], který umožňuje definovat sadu podmínek, které musí být pro dokument splněny. Schematronové schéma tedy nejde použít pro nic jiného než pro validaci, ale díky použití jazyka XPath jako základu pro zápis podmínek, máme v ruce velmi silný nástroj. Žádný z předchozích jazyků neumožňuje například popsat taková omezení, kdy obsah jednoho elementu musí obsahovat větší hodnotu než obsah jiného elementu, nebo vztahy mezi daty uloženými v několika dokumentech.

Vzhledem ke svým vlastnostem bývá Schematron často používán jen jako doplněk ke klasickým schématům. XML schéma i Relax NG nabízejí možnosti, jak přímo do schématu vložit schematronovské výroky.

Existence několika konkurenčních schémových jazyků naznačuje, že se každý hodí na něco trochu jiného a nejde tedy očekávat, že za pár let jeden z jazyků převáží nad ostatními. Ukazuje se spíše, že je vhodné jazyky navzájem kombinovat. Aby existoval nějaký jednotný rámec, jak popsat vícenásobnou validaci dokumentů, vzniká na půdě ISO nový projekt DSDL (Document Schema Definition Languages). Jeho cílem je jednak kodifikovat jazyky jako Relax NG a Schematron do podoby ISO norem a hlavně vytvořit standardní prostředí pro validaci dokumentu oproti několika schématům. Není to lehký úkol, protože dokumenty používající více jmenných prostorů musí být před validací obvykle rozebrány na několik samostatných fragmentů.

1.3 Srovnání nejpoužívanějších jazyků pro popis schématu dokumentu

Před podrobnějším výkladem jednotlivých schémových jazyků si na jednoduchém příkladě ukážeme základní vlastnosti jednotlivých jazyků. Předpokládejme, že potřebujeme vytvořit schéma pro následující XML dokument s informacemi o zaměstnancích.

Příklad 1.1. Ukázkový dokument XML – `uvod/zamestnanci.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<zamestnanci>
  <zamestnanec id="101">
    <jmeno>Jan</jmeno>
    <prijmeni>Novák</prijmeni>
    <email>jan@novak.cz</email>
    <email>jan.novak@firma.cz</email>
    <plat>25000</plat>
    <narozen>1965-12-24</narozen>
```

[5] http://www.relaxng.org/
[6] http://www.oasis-open.org
[7] http://www.schematron.com

```
  </zamestnanec>
  <zamestnanec id="102">
    <jmeno>Petra</jmeno>
    <prijmeni>Procházková</prijmeni>
    <email>prochazkovap@firma.cz</email>
    <plat>27500</plat>
    <narozen>1974-12-21</narozen>
  </zamestnanec>
</zamestnanci>
```

Schéma vyjádřené pomocí DTD neumožňuje definovat datový typ jednotlivých elementů, ale je poměrně jednoduché.

Příklad 1.2. Ukázka DTD – `uvod/zamestnanci.dtd`

DTD

```
<!ELEMENT zamestnanci (zamestnanec+)>
<!ELEMENT zamestnanec (jmeno, prijmeni, email+,
                       plat?, narozen)>
<!ELEMENT jmeno       (#PCDATA)>
<!ELEMENT prijmeni    (#PCDATA)>
<!ELEMENT email       (#PCDATA)>
<!ELEMENT plat        (#PCDATA)>
<!ELEMENT narozen     (#PCDATA)>
<!ATTLIST zamestnanec
          id     CDATA #REQUIRED>
```

XML schéma už umožňuje pro jednotlivé elementy definovat jejich datový typ a celé je zapsáno syntaxí XML. Výsledkem je však poměrně výřečný dokument, který není moc pohodlné psát přímo v ruce bez pomoci specializovaných editorů schémat.

Příklad 1.3. Ukázka W3C XML Schema – `uvod/zamestnanci.xsd`

WXS

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="zamestnanci">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="zamestnanec"
                    maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="jmeno" type="xs:string"/>
              <xs:element name="prijmeni" type="xs:string"/>
              <xs:element name="email" type="xs:string"
                          maxOccurs="unbounded"/>
              <xs:element name="plat" type="xs:decimal"
                          minOccurs="0"/>
              <xs:element name="narozen" type="xs:date"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:int"
                          use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

```
</xs:schema>
```

Relax NG samo o sobě nepodporuje datové typy, ale lze o tuto podporu rozšířit. Standardně se zapisuje rovněž v XML syntaxi, která je však poměrně stručná a navíc sjednocuje přístup k atributům a elementům.

Příklad 1.4. Ukázka Relax NG schématu – `uvod/zamestnanci.rng`

RNG

```
<?xml version="1.0" encoding="utf-8"?>
<element xmlns="http://relaxng.org/ns/structure/1.0"
         name="zamestnanci">
  <oneOrMore>
    <element name="zamestnanec">
      <attribute name="id">
        <text/>
      </attribute>
      <element name="jmeno">
        <text/>
      </element>
      <element name="prijmeni">
        <text/>
      </element>
      <oneOrMore>
        <element name="email">
          <text/>
        </element>
      </oneOrMore>
      <optional>
        <element name="plat">
          <text/>
        </element>
      </optional>
      <element name="narozen">
        <text/>
      </element>
    </element>
  </oneOrMore>
</element>
```

Relax NG lze zapisovat i v kompaktní textové syntaxi (RNC), která je velmi úsporná a zápis schématu je pak hračkou i v obyčejném textovém editoru.

Příklad 1.5. Ukázka Relax NG schématu v kompaktní syntaxi – `uvod/zamestnanci.rnc`

RNC

```
element zamestnanci {
  element zamestnanec {
    attribute id { text },
    element jmeno { text },
    element prijmeni { text },
    element email { text }+,
    element plat { text }?,
    element narozen { text }
  }+
}
```

Relax NG schéma může používat různé typové systémy, mj. i typový systém XML schémat. Můžeme tak vytvořit schéma, které pro náš seznam zaměstnanců definuje i datové typy.

Příklad 1.6. Ukázka Relax NG schématu s datovými typy – `uvod/zamestnanci2.rng`

RNG

```
<?xml version="1.0" encoding="utf-8"?>
<element xmlns="http://relaxng.org/ns/structure/1.0"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
         name="zamestnanci">
  <oneOrMore>
    <element name="zamestnanec">
      <attribute name="id">
        <data type="int"/>
      </attribute>
      <element name="jmeno">
        <data type="string"/>
      </element>
      <element name="prijmeni">
        <data type="string"/>
      </element>
      <oneOrMore>
        <element name="email">
          <data type="string"/>
        </element>
      </oneOrMore>
      <optional>
        <element name="plat">
          <data type="decimal"/>
        </element>
      </optional>
      <element name="narozen">
        <data type="date"/>
      </element>
    </element>
  </oneOrMore>
</element>
```

Příklad 1.7. Ukázka RNC schématu s datovými typy – `uvod/zamestnanci2.rnc`

RNC

```
element zamestnanci {
  element zamestnanec {
    attribute id { xsd:int },
    element jmeno { xsd:string },
    element prijmeni { xsd:string },
    element email { xsd:string }+,
    element plat { xsd:decimal }?,
    element narozen { xsd:date }
  }+
}
```

Schematron se od předchozích jazyků svým principem podstatně liší. Nedefinuje se v něm gramatika, ale sada podmínek, kterým musí dokument vyhovět.

Příklad 1.8. Ukázka schématu ve Schematronu – `uvod/zamestnanci.sch`

SCH

```
<?xml version="1.0" encoding="utf-8"?>
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
  <pattern>
    <title>Seznam zaměstnanců je neprázdný a máme na výplaty</title>
    <rule context="zamestnanci">
      <assert test="zamestnanec">V seznamu musí být alespoň
```

```
        jeden zaměstnanec</assert>
      <report test="sum(zamestnanec/plat) > 500000">Součet platů nemůže
        být větší než 500.000,-</report>
    </rule>
  </pattern>
  <pattern>
    <title>Podmínky pro zaměstnance</title>
    <rule context="zamestnanec">
      <assert test="@id">U zaměstnance musí být zadáno jeho osobní číslo.</assert>
      <report test="jmeno[2]|prijmeni[2]">Zaměstnanec nemůže mít více
        než jedno jméno.</report>
    </rule>
  </pattern>
  <pattern>
    <title>Duplicita osobních čísel</title>
    <rule context="zamestnanec">
      <report test="count(../zamestnanec[@id = current()/@id]) > 1">Duplicitní osobní číslo
          <value-of select="@id"/> u elementu <name/>.</report>
    </rule>
  </pattern>
</schema>
```

V současné době se pracuje na vývoji „metaschémat“, která popíší, jak se mají validovat dokumenty složené z několika částí v různých jmenných prostorech. Nejznámějším jazykem z této rodiny je NVDL (Namespace Validation and Dispatching Language).

Příklad 1.9. Ukázka NVDL schématu pro validaci XHTML+SVG dokumentů

NVDL

```
<rules xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0">
  <namespace ns="http://www.w3.org/1999/xhtml">
    <validate schema="xhtml.rng">
      <mode>
        <namespace ns="http://www.w3.org/2000/svg">
          <validate schema="svg.xsd">
            <mode>
              <anyNamespace>
                <attach/>
              </anyNamespace>
            </mode>
          </validate>
        </namespace>
      </mode>
    </validate>
  </namespace>
</rules>
```

1.4 Jaký jazyk pro popis schématu dokumentu použít

Současná situace, kdy paralelně existuje několik schémových jazyků, staví každého vývojáře před problém, který jazyk použít. Naštěstí se ukazuje, že v praktickém životě platí jednoduchá pravidla.

Pokud v dokumentech nepotřebujeme používat jmenné prostory a datové typy, vystačíme si s DTD. Jejich použití je bezproblémové, protože jsou podporovány největším počtem aplikací. Tato situace však dnes v praxi často nenastává. Zbývá nám tedy volba mezi jazyky XML schéma a Relax NG. Rozhodování

nám většinou usnadní okolní prostředí. Vývojové nástroje komerčních firem (Microsoft, IBM, Oracle, Sun) zatím podporují jen XML schémata a standardní XML knihovny rovněž. Volba je pak dána předem.

Nejsme-li však takto omezeni a spokojíme-li se s menším počtem dostupných nástrojů, je většinou jednodušší použít Relax NG. Budeme odměněni příjemným a jednoduchým jazykem. Relax NG navíc kromě „upovídané“ syntaxe XML umožňuje schéma zapsat v kompaktní textové podobě.

Současný vývoj směřuje k podpoře několika schémových jazyků, ze kterých si bude moci uživatel vybrat. Již dnes navíc existují nástroje, jako Trang[8], které umožňují převádět schéma z jednoho schémového jazyku do druhého. Mnoho projektů z výše popsaných pragmatických důvodů používá nebo bude používat XML schémata.

[8] http://thaiopensource.com/relaxng/trang.html

Kapitola 2. DTD

Ačkoliv jsou DTD poměrně starou a v mnoha ohledech nedostatečnou technologií, protože nepodporují dnes tolik používané jmenné prostory, i přesto se stále hojně používají. Je to díky výborné podpoře v různých aplikacích a parserech. Ostatní schémové technologie na DTD více či méně navazují, takže je dobré mít alespoň základní znalosti DTD.

Pro rychlé vpravení do problematiky si rovnou ukážeme jednoduchý příklad DTD a dokumentu XML, který mu vyhovuje. Dejme tomu, že chceme vytvořit schéma popisující dokumenty XML vhodné pro přenos informace o jednom zaměstnanci. U zaměstnance nás přitom bude zajímat jeho identifikační číslo, jméno, příjmení, plat a datum narození. Tyto informace můžeme v XML zachytit následujícím způsobem:

```
<zamestnanec id="101">
  <jmeno>Jan</jmeno>
  <prijmeni>Novák</prijmeni>
  <plat>25000</plat>
  <narozen>1965-12-24</narozen>
</zamestnanec>
```

Odpovídající schéma tedy definuje, že dokument musí obsahovat element `zamestnanec`, který obsahuje atribut `id` a čtyři podelementy `jmeno`, `prijmeni`, `plat` a `narozen`. Obsah těchto elementů můžeme pak v DTD definovat jako text, jiné datové typy nejsou k dispozici.

DTD
```
<!ELEMENT zamestnanec       (jmeno, prijmeni, plat, narozen)>
<!ATTLIST zamestnanec
          id            CDATA   #REQUIRED>
<!ELEMENT jmeno         (#PCDATA)>
<!ELEMENT prijmeni      (#PCDATA)>
<!ELEMENT plat          (#PCDATA)>
<!ELEMENT narozen       (#PCDATA)>
```

Jak vidíme, používá se pro zápis DTD zcela speciální syntaxe s klíčovými slovy jako je `<!ELEMENT…>` a `<!ATTLIST…>`, které slouží k deklaraci elementů a atributů. Mezi deklarace lze vkládat komentáře stejně jako v XML.

DTD
```
<!-- Informace o zaměstnanci -->
<!ELEMENT zamestnanec   (jmeno, prijmeni, plat, narozen)>
<!ATTLIST zamestnanec
          id            CDATA   #REQUIRED> <!-- Osobní číslo zaměstnance -->
<!ELEMENT jmeno         (#PCDATA)>
<!ELEMENT prijmeni      (#PCDATA)>
<!ELEMENT plat          (#PCDATA)>
<!ELEMENT narozen       (#PCDATA)>
```

U DTD se předpokládá, že je uloženo v kódování UTF-8 nebo UTF-16. Máte-li v dokumentu například znaky s diakritikou a z nějakého důvodu musíte použít jiné kódování, je možné jej určit pomocí deklarace kódování (má podobný tvar jako deklarace XML).

DTD
```
<?xml version="1.0" encoding="windows-1250"?>
<!-- Informace o zaměstnanci -->
<!ELEMENT zamestnanec   (jmeno, prijmeni, plat, narozen)>
…
```

Kromě deklarací elementů a atributů nabízí DTD prostředky pro deklaraci entit a notací. Tyto konstrukce však přímo neovlivňují definovanou strukturu dokumentu, a proto se jimi v našem tutoriálu nebudeme zabývat.

2.1 Deklarace elementů

Deklarace nového elementu je velice jednoduchá. Má následující tvar:

DTD
```
<!ELEMENT název elementu   obsah elementu>
```

Jako název elementu můžeme zvolit libovolné jméno, které má tvar dovolený specifikací XML. Nejzajímavější je však poslední část deklarace elementu, která definuje, co může element obsahovat. Nejjednodušší je prázdný element, který nemůže obsahovat žádné další elementy nebo text. Příkladem takového elementu mohou být například elementy `br` a `hr`, které známe z XHTML. Jejich deklarace by vypadala následovně:

DTD
```
<!ELEMENT br EMPTY>
<!ELEMENT hr EMPTY>
```

Právě klíčové slovo `EMPTY` určuje, že element nesmí nic obsahovat. V dokumentu pak musíme psát buď `<br></br>`, nebo zkráceně `<br/>`.

Protipólem k `EMPTY` je `ANY`. Toto klíčové slovo nám zajistí, že element může obsahovat libovolné další elementy (ty však musí být také deklarované) a text. `ANY` se v praxi moc často nevyužívá, protože pro potřeby většiny aplikací příliš uvolňuje strukturu dokumentu. Využít ho lze například při návrhu a ladění DTD, kdy nechceme najednou napsat celé DTD.

DTD
```
<!ELEMENT cokoliv ANY>
```

Většinou máme na vnořené elementy mnohem striktnější požadavky a s `EMPTY` a `ANY` nevystačíme. V tomto případě pak použijeme tzv. *model obsahu (content model)*. Model obsahu se používá pro definici elementů, které obsahují další elementy nebo mají smíšený obsah (obsahují již přímo text a elementy). Jeho zápis je přitom hodně podobný regulárním výrazům.

Model obsahu je vždy uzavřen do kulatých závorek a obsahuje alespoň jedno slovo. Tímto slovem nejčastěji bývá jméno elementu, který může být obsažen v právě definovaném elementu. Vnořené elementy můžeme navzájem kombinovat pomocí oddělovačů ‚`,`‘ a ‚`|`‘. Elementy oddělené čárkou musí následovat v pořadí, v jakém jsou uvedeny. Pokud má tedy element `html` obsahovat záhlaví (`head`) a tělo (`body`), použijeme deklaraci:

DTD
```
<!ELEMENT html (head, body)>
```

Pokud naopak elementy oddělíme znakem ‚`|`‘, může být v dokumentu uveden pouze jeden z nich. Příklad: potomkem může být dcera nebo syn. V DTD to vyjádříme takto:

DTD
```
<!ELEMENT potomek (dcera | syn)>
```

Pomocí závorek můžeme obě dvě varianty navzájem kombinovat. Pokud má nějaký element obsahovat elementy `a`, `b` a za nimi buď `c`, nebo `d`, použijeme model obsahu `(a, b, (c | d))`.

Kromě pořadí elementů musíme určit jejich počet, zda jsou povinné či zda se mohou opakovat. Pokud v modelu obsahu uvedeme pouze jméno elementu, musí být element přítomen právě jednou. Je-li však výskyt elementu nepovinný, uvedeme za jeho jméno znak ‚`?`‘. Pokud například článek obsahuje vždy název, ale autora obsahovat nemusí, můžeme použít následující deklaraci:

```
<!ELEMENT clanek (nazev, autor?)>
```

DTD

Další obvyklou situací je, že nějaký element se může opakovat, ale musí být přítomen alespoň jednou. Například kniha se skládá z několika kapitol a musí obsahovat alespoň jednu kapitolu:

DTD

```
<!ELEMENT kniha (kapitola+)>
```

Vraťme se k předchozímu příkladu a předpokládejme, že článek může mít více autorů a nemusí mít autora žádného. V tomto případě s výhodou využijeme znak ‚*‘, který indikuje libovolný počet opakování.

DTD

```
<!ELEMENT clanek (nazev, autor*)>
```

Vše můžeme podle potřeby kombinovat. Při troše fantazie lze obejít i to, že nemůžeme specifikovat přesný počet opakování určitých elementů. Chceme-li například vyjádřit, že seznam obsahuje alespoň dvě položky, můžeme použít model obsahu `(polozka, polozka+)`.

Indikátor počtu výskytů můžeme připojit i za celý model obsahu. Například zápis `(a, b)?` říká, že se elementy `a` a `b` buď musí vyskytovat v daném pořadí, nebo nesmí být použity vůbec.

Speciální pozornost si zaslouží případ, kdy je obsahem elementu již samotný text. To vyjádříme pomocí slova `#PCDATA`. Pokud by například element `em` obsahoval již pouze text, a ne další elementy, použili bychom pro jeho deklaraci zápis:

DTD

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

Pokud může element obsahovat jak text, tak elementy, říkáme, že má *smíšený obsah*. V tomto případě musí mít deklarace jeho obsahu speciální tvar. `#PCDATA` musí být uvedeno ve skupině jako první, skupina musí být spojena pomocí operátoru ‚|‘ a musí být volitelně opakovatelná (`*`). Například:

DTD

```
<!ELEMENT em  (#PCDATA | sub | sup)*>
<!ELEMENT sub (#PCDATA)>
<!ELEMENT sup (#PCDATA)>
```

Dokument pak může obsahovat následující text, vyhovující DTD:

```
<em>Pozor na líh - C<sub>2</sub>H<sub>5</sub>OH.</em>
```

2.2 Deklarace atributů

V XML může mít každý element libovolné množství atributů. Atributy se většinou používají pro připojení různých metainformací k elementům. Deklarace atributů pro element má poměrně jednoduchý tvar:

DTD

```
<!ATTLIST jméno elementu   deklarace atributů>
```

Deklarace jednotlivých atributů se skládá ze tří částí. První částí je jméno atributu. Za jménem následuje typ atributu. Poslední část určuje standardní hodnotu atributu, popřípadě zda je používání atributu u daného elementu povinné. Deklarace se opakuje pro každý atribut, který má být u elementu k dispozici.

Nejobecnějším typem atributu je `CDATA`, který umožňuje jako jeho hodnotu zadat libovolný textový řetězec. Po deklaraci:

DTD

```
<!ATTLIST kniha autor CDATA …>
```

můžeme v dokumentu používat atribut `autor` následovně

```
<kniha autor="Karel Čapek">
```

Mnohem restriktivnějším typem než CDATA je NMTOKEN. Atribut tohoto typu může obsahovat jedno slovo, které se skládá z písmen, číslic a několika dalších speciálních znaků (platí zde stejné omezení jako pro jména elementů a atributů).

Můžeme použít i typ NMTOKENS. Atribut pak může obsahovat několik slov vyhovujících typu NMTOKEN oddělených mezerou.

DTD
```
<!ATTLIST dokument format NMTOKEN …
                  okraje NMTOKENS …>
```

V dokumentu pak můžeme použít element dokument například takto:

```
<dokument format="A4" okraje="2cm 3cm 2.5cm 3cm">
        …
</dokument>
```

Mezi další typy atributů patří ID, IDREF a IDREFS, které se používají pro vytváření odkazů v rámci dokumentu. Pokud atribut definujeme jako ID, musí mít v rámci dokumentu přiřazenu jedinečnou hodnotu. Přitom všechny atributy typu ID sdílí stejný prostor – omezení na jedinečnost platí i pro atributy s různým názvem u různých elementů. Pokud pravidlo jedinečnosti porušíme, ohlásí parser chybu. U jednoho elementu můžeme přitom deklarovat maximálně jeden atribut typu ID.

Atributy s typem IDREF pak mohou obsahovat pouze hodnotu použitou v nějakém atributu typu ID. Typ IDREFS umožňuje použít v jednom atributu více hodnot najednou – podobně jako u NMTOKENS se jednotlivé hodnoty oddělují mezerami.

Poslední možností, jak vymezit typ atributu, je uvedení výčtu přípustných hodnot atributu. Například:

DTD
```
<!ATTLIST obrazek zarovnani (vlevo | vpravo | doprostred) …>
```

U elementu obrazek nyní můžeme jako hodnotu atributu zarovnani uvést pouze jedno ze slov vlevo, vpravo a doprostred.

Atribut může mít ještě typ ENTITY, ENTITIES nebo NOTATION. Tyto typy se v praxi dnes již téměř nepoužívají, protože souvisejí s možnostmi, který byly do XML převzaty z SGML, ale neukázaly se moc životaschopné.

V deklaraci všech atributů jsme zatím na posledním místě používali tři tečky. Na tomto místě se uvádí standardní hodnota atributu, nebo to, zda je atribut povinný.

Pokud je atribut povinný, uvede se za deklarací jeho typu slovo #REQUIRED. Parser při kontrole dokumentu ohlásí chyby vždy, když nebude tento atribut u elementu použit.

Opačným případem je situace, kdy můžeme atribut vynechat, a je pak věcí aplikace, která dokument zpracovává, jak se v tomto případě zachová. K určení nepovinného atributu slouží klíčové slovo #IMPLIED.

Další možností je specifikování standardní hodnoty, která se použije v případě, kdy atribut není v dokumentu uveden. Chceme-li, aby byl obrázek zarovnán vlevo, pokud není určeno jinak, použijeme deklaraci:

DTD
```
<!ATTLIST obrazek zarovnani (vlevo | vpravo | doprostred) "vlevo">
```

Před standardní hodnotou atributu můžeme ještě uvést slovo #FIXED. Tím stanovíme, že v dokumentu nemůže mít atribut jinou hodnotu než standardní. Tento postup se dá použít pro uložení metainformací do DTD, které se automaticky doplní do každého dokumentu. Nicméně tento postup se nedoporučuje, jak je vysvětleno v sekci 7.4 – „Defaultní a fixní hodnoty“.

Pokud má jeden element více atributů, je úplně jedno, zda je deklarujeme v jedné deklaraci `<!ATTLIST …>` nebo v několika postupně. Následující dva zápisy jsou tedy totožné.

DTD

```
<!ATTLIST para align (left|right|center) #IMPLIED
              id    ID                   #IMPLIED>

<!ATTLIST para align (left|right|center) #IMPLIED>
<!ATTLIST para id    ID                   #IMPLIED>
```

2.3 Připojení DTD k dokumentu

DTD se k dokumentu připojuje pomocí deklarace typu dokumentu, která se uvádí bezprostředně za deklarací XML. DTD přitom můžeme napsat přímo do interní podmnožiny deklarace typu dokumentu a je pak součástí dokumentu.

Příklad 2.1. DTD v interní podmnožině – `dtd/doctype-internal.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE zamestnanec [
  <!ELEMENT zamestnanec (jmeno, prijmeni, plat, narozen)>
  <!ATTLIST zamestnanec
            id          CDATA   #REQUIRED>
  <!ELEMENT jmeno       (#PCDATA)>
  <!ELEMENT prijmeni    (#PCDATA)>
  <!ELEMENT plat        (#PCDATA)>
  <!ELEMENT narozen     (#PCDATA)>
]>
<zamestnanec id="101">
  <jmeno>Jan</jmeno>
  <prijmeni>Novák</prijmeni>
  <plat>25000</plat>
  <narozen>1965-12-24</narozen>
</zamestnanec>
```

Tato metoda je však poměrně nepraktická, protože DTD se musí kopírovat do každého dokumentu. Nejčastěji se proto DTD uloží do samostatného souboru a v deklaraci typu dokumentu se pak načte jako externí podmnožina.

Příklad 2.2. Externí DTD – `dtd/doctype-external.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE zamestnanec SYSTEM "zamestnanec.dtd">
<zamestnanec id="101">
  <jmeno>Jan</jmeno>
  <prijmeni>Novák</prijmeni>
  <plat>25000</plat>
  <narozen>1965-12-24</narozen>
</zamestnanec>
```

Oba přístupy je možné kombinovat. To lze využít v případech, kdy chceme pro jeden dokument upravit chování DTD. Lokální deklarace v interní podmnožině mají přednost před deklaracemi z externího DTD. Následující příklad ukazuje, jak atribut `id` předefinovat jako nepovinný.

Příklad 2.3. Kombinování externích a interních deklarací – `dtd/doctype-both.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE zamestnanec SYSTEM "zamestnanec.dtd" [
  <!ATTLIST zamestnanec
            id  CDATA   #IMPLIED>
]>
<zamestnanec>
  <jmeno>Jan</jmeno>
  <prijmeni>Novák</prijmeni>
  <plat>25000</plat>
  <narozen>1965-12-24</narozen>
</zamestnanec>
```

Kapitola 3. XML schémata

Pro rychlé vpravení do problematiky si rovnou ukážeme jednoduchý příklad XML schématu a dokumentu XML, který mu vyhovuje. Dejme tomu, že chceme vytvořit schéma popisující dokumenty XML vhodné pro přenos informace o jednom zaměstnanci. U zaměstnance nás přitom bude zajímat jeho identifikační číslo, jméno, příjmení, plat a datum narození. Tyto informace můžeme v XML zachytit následujícím způsobem:

```
<zamestnanec id="101">
  <jmeno>Jan</jmeno>
  <prijmeni>Novák</prijmeni>
  <plat>25000</plat>
  <narozen>1965-12-24</narozen>
</zamestnanec>
```

Odpovídající schéma tedy musí definovat, že dokument musí obsahovat element `zamestnanec`, který obsahuje atribut `id` a čtyři podelementy `jmeno`, `prijmeni`, `plat` a `narozen`. Obsah těchto elementů přitom musí odpovídat určitému datovému typu. Všechny tyto požadavky může zachytit následující jednoduché XML schéma.

WXS
```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="zamestnanec">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="jmeno" type="xs:string"/>
        <xs:element name="prijmeni" type="xs:string"/>
        <xs:element name="plat" type="xs:decimal"/>
        <xs:element name="narozen" type="xs:date"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Vidíme, že celé schéma je dokument XML, který používá speciální elementy. Všechny tyto elementy musí patřit do jmenného prostoru `http://www.w3.org/2001/XMLSchema`. Obvykle se pro tento jmenný prostor používá prefix `xs` nebo `xsd`.

Celé schéma musí být vždy uzavřeno v elementu `schema` obsahujícím definice elementů, které lze použít jako kořenové elementy, tj. dokument XML jimi začíná. Definice elementu se zapisuje pomocí elementu `element`.

Pro každý element musí schéma určit jeho typ. Rozlišovány jsou přitom dva druhy typů – jednoduché a komplexní. Jednoduché typy se používají pro skalární hodnoty jako řetězec, číslo, datum apod. Obsahuje-li element však další elementy nebo atributy, musíme použít komplexní typ (`complexType`). Pomocí elementu `sequence` říkáme, že zaměstnanec se skládá z po sobě následujících elementů pro jméno, příjmení, plat a datum narození. Každý z těchto elementů je povinný, má určené své jméno pomocí atributu `name` a datový typ pomocí atributu `type`. Datové typy se uvádějí jako kvalifikované názvy patřící rovněž do jmenného prostoru XML schémat.

Atributy se deklarují až za vnořenými elementy pomocí elementu `attribute`. U atributu rovněž musíme určit jeho název a datový typ.

3.1 Datové typy

Datové typy jsou samotným základem XML schémat, protože v nich je v podstatě vše datový typ. Jak jsme již řekli, datové typy mohou být jednoduché či komplexní.

Největší síla XML schémat spočívá v možnosti definovat si vlastní datové typy. Ty přitom mohou vznikat zejména restrikcí nebo rozšířením některého z již existujících typů. Jako základ pro odvozování našich uživatelských typů můžeme použít některý ze zabudovaných typů. Na obrázku 3.1 – „Hierarchie zabudovaných datových typů“ si můžete prohlédnout hierarchii zabudovaných datových typů. Obrázek byl převzat přímo ze specifikace [8].

3.2 Jednoduché datové typy

XML schémata obsahují běžné základní datové typy jako textový řetězec, celá a desetinná čísla, binární data, logická hodnota, datum, čas, časový interval a několik typů převzatých z DTD pro jejich snazší konverzi do XML schémat. V mnoha případech si s těmito typy vystačíme. Naše první ukázka si například naprosto vystačila se zabudovanými typy pro textové řetězce (`string`), datum (`date`) a desetinné číslo (`decimal`).

Tabulka 3.1. Zabudované datové typy

Typ	**Popis**	**Ukázka/Poznámka**
string	řetězec znaků	`ahoj`
boolean	logická hodnota	`true`, `false`, `1`, `0`
decimal	desetinné číslo	`-1372.42` desetinné číslo s přesností nejméně 18 platných číslic
float	desetinné číslo	`1e-3`, `0.001`, `3.1415` 32bitové číslo v plovoucí desetinné čárce
double	desetinné číslo	`1e-3`, `0.001`, `3.1415` 64bitové číslo v plovoucí desetinné čárce
duration	délka časového intervalu	`P1Y2M3DT10H30M` ukázka reprezentuje interval o délce jeden rok, dva měsíce, tři dny, deset hodin a třicet minut
dateTime	datum a čas	`2005-07-05T10:58:53+02:00`, `2005-07-05T08:58:53Z` údaj je ve formátu ISO8601 a může obsahovat i časovou zónu
time	čas	`12:15:00+02:00`, `10:15:00Z`, `10:15:00` časový údaj ve formátu ISO8601; může obsahovat časovou zónu
date	datum	`2005-07-05` datum ve formátu ISO8601, tj. rok-měsíc-den

Typ	Popis	Ukázka/Poznámka
gYearMonth	měsíc v daném roce	`1975-09` ukázka reprezentuje září 1975
gYear	rok	`2005`
gMonthDay	den v daném měsíci	`--12-24` hodí se pro zachycení datumu, který se každý rok opakuje; ukázka např. reprezentuje Štědrý večer
gDay	určitý den v měsíci	`---12` hodí se pro zachycení dne, ve kterém se každý měsíc něco opakuje; ukázka může například reprezentovat, že výplata je vždy dvanáctého
gMonth	určitý měsíc v roce	`--05` ukázka reprezentuje měsíc květen
hexBinary	binární data	`0FB7` binární data jsou zakódována tak, že každému bajtu odpovídají dva znaky reprezentující hodnotu bajtu v šestnáctkové soustavě
base64Binary	binární data	`D7c=` binární data jsou zakódována metodou Base64
anyURI	adresa URI	`http://www.kosek.cz/xml/db/index.html` jakákoliv URI adresa; může být absolutní i relativní
NOTATION	notace externích dat	v praxi téměř nepoužívaný typ, který byl do WXS převzat z DTD
normalizedString	normalizovaný řetězec znaků	`ahoj` před kontrolou dalších integritních omezení jsou v řetězci konce řádků a tabulátory nahrazeny mezerou
token	ještě více normalizovaný řetězec znaků	`ahoj` před kontrolou dalších integritních omezení jsou v řetězci konce řádků a tabulátory nahrazeny mezerou, opakující se mezery nahrazeny mezerou jedinou a jsou odstraněny mezery na začátku a konci řetězce
language	jazykový kód	`cs`, `en-US` kód identifikující jazyk
NMTOKEN		typ byl převzat z DTD kvůli zpětné kompatibilitě; lze jej používat pouze pro atributy
NMTOKENs		typ byl převzat z DTD kvůli zpětné kompatibilitě; lze jej používat pouze pro atributy
Name	jméno	`xsl:template`, `pre`

Typ	Popis	Ukázka/Poznámka
		jméno musí splňovat pravidla XML pro názvy elementů/atributů
NCName	jméno bez dvojtečky	`pre`, `template` podobné jako typ `Name`, ale nesmí obsahovat dvojtečku
ID	unikátní identifikátor	`uvod` převzatý z DTD kvůli zpětné kompatibilitě
IDREF	odkaz na identifikátor	`uvod` převzatý z DTD kvůli zpětné kompatibilitě
IDREFS	odkazy na několik identifikátorů	`uvod stat zaver` převzatý z DTD kvůli zpětné kompatibilitě
ENTITY	odkaz na externí binární entitu	převzatý z DTD kvůli zpětné kompatibilitě; může se použít pouze pro definici atributu
ENTITIES	odkazy na externí binární entity	převzatý z DTD kvůli zpětné kompatibilitě; může se použít pouze pro definici atributu
integer	celé číslo	`-17`, `0`, `65542`
nonPositiveInteger	nekladné celé číslo	`-17`, `0` celé číslo menší nebo rovné 0
negativeInteger	záporné celé číslo	`-17` celé číslo menší než 0
long	celé číslo	`-17`, `0`, `65542` číslo je v rozsahu od -9223372036854775808 do 9223372036854775807, což odpovídá 64bitovému celému číslu
unsignedLong	nezáporné celé číslo	`0`, `65542` číslo je v rozsahu od 0 do 18446744073709551615, což odpovídá 64bitovému celému číslu
int	celé číslo	`-17`, `0`, `65542` číslo je v rozsahu od -2147483648 do 2147483647, což odpovídá 32bitovému celému číslu
unsignedInt	nezáporné celé číslo	`0`, `65542` číslo je v rozsahu od 0 do 4294967295, což odpovídá 32bitovému celému číslu
short	celé číslo	`-17`, `0`, `29700` číslo je v rozsahu od -32768 do 32767, což odpovídá 16bitovému celému číslu

Typ	Popis	Ukázka/Poznámka
unsignedShort	nezáporné celé číslo	`0`, `42` číslo je v rozsahu od 0 do 65535, což odpovídá 16bitovému celému číslu
byte	celé číslo	`-17`, `0`, `42` číslo je v rozsahu od -128 do 127, což odpovídá 8bitovému celému číslu
unsignedByte	nezáporné celé číslo	`0`, `42` číslo je v rozsahu od 0 do 255, což odpovídá 8bitovému celému číslu
nonNegativeInteger	nezáporné celé číslo	`42`, `0` celé číslo větší nebo rovné 0
positiveInteger	kladné celé číslo	`42` celé číslo větší než 0

Vyžadujeme-li však v aplikacích striktnější kontrolu dat, musíme si od zabudovaných typů odvodit vlastní typy. Nejběžnějším způsobem odvození je restrikce, kdy pomocí integritních omezení zúžíme obor přípustných hodnot. Kromě toho lze odvozovat nové typy vytvořením seznamu a sjednocením typů.

3.2.1 Odvození typů restrikcí

Nejpoužívanějším typem odvození nového typu je bezesporu restrikce. Na existující datový typ můžeme najednou aplikovat několik integritních omezení a tak zúžit přípustné hodnoty.

Omezení délky

Podívejme se nejprve na integritní omezení použitelná pro textové řetězce. Nejjednodušším z nich je `length`, které umožňuje definovat přesnou délku řetězce. Nový datový typ pro PSČ tak můžeme snadno odvodit ze zabudovaného typu pro řetězce:

WXS
```
<xs:simpleType name="PSČType">
  <xs:restriction base="xs:string">
    <xs:length value="6"/>
  </xs:restriction>
</xs:simpleType>
```

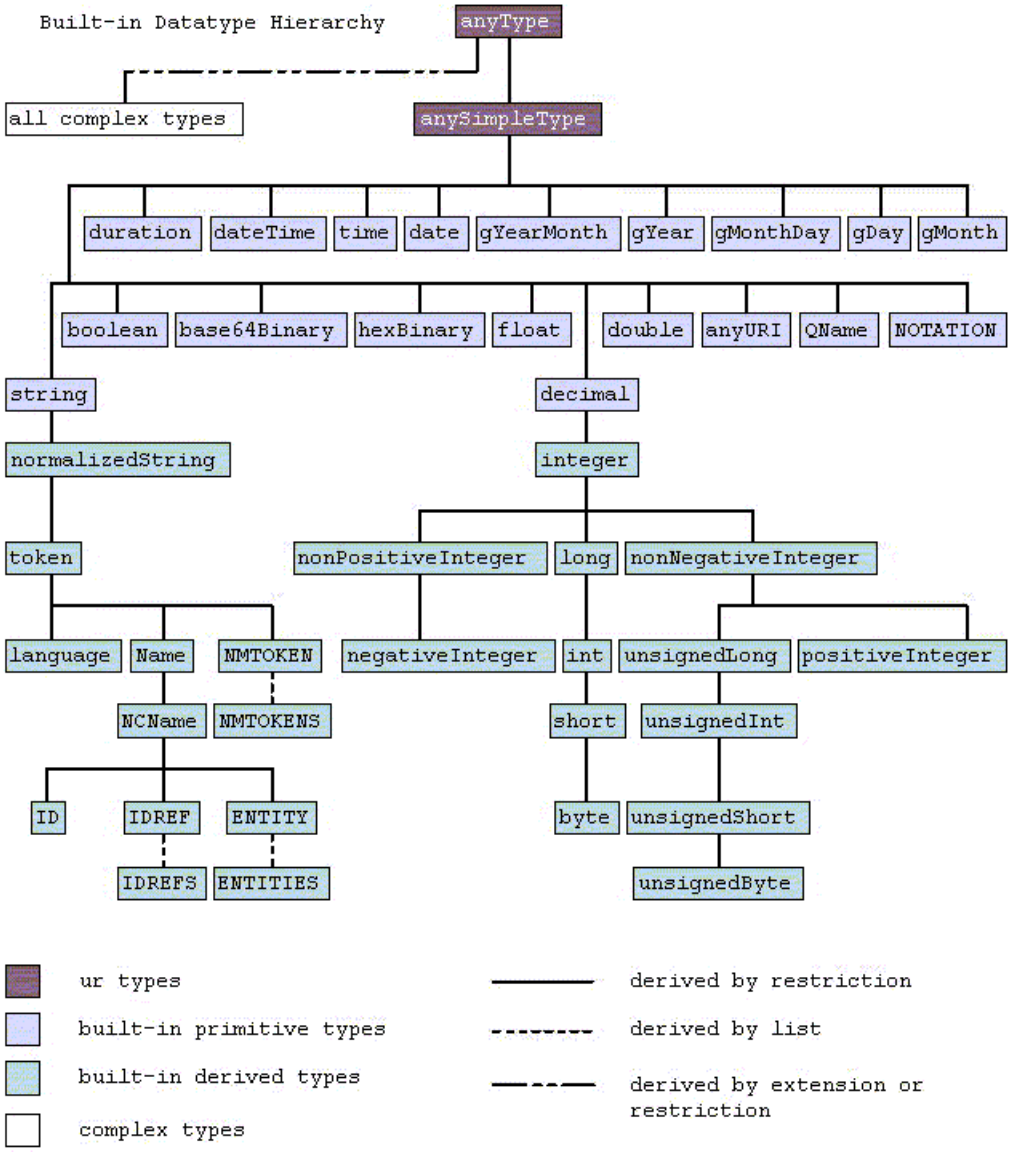
Element `simpleType` slouží k definici nového jednoduchého datového typu. Atribut `name` pak nese jméno nového typu. Element `restriction` říká, že odvození probíhá právě restrikcí od typu uvedeného v atributu `base` (v našem případě odvozujeme od běžného textového řetězce). Uvnitř elementu `restriction` pak uvádíme všechna omezení, v našem případě jen omezení na přesnou délku řetězce.

Nově definovaný typ můžeme použít při definici elementů a atributů nebo při definici dalších z něj odvozených typů. Nově definované typy se při vytváření elementů používají stejně jako zabudované typy:

WXS
```
<xs:element name="psc" type="PSČType"/>
```

Mezi další omezení aplikovatelná na řetězcové typy patří minimální (`minLength`) a maximální délka (`maxLength`).

Obrázek 3.1. Hierarchie zabudovaných datových typů



WXS

```
<xs:simpleType name="jménoType">
  <xs:restriction base="xs:string">
    <xs:minLength value="1"/>
    <xs:maxLength value="15"/>
  </xs:restriction>
</xs:simpleType>
```

Výčet hodnot

Můžeme rovněž určit výčet přípustných hodnot pro datový typ:

WXS

```
<xs:simpleType name="kódMěnyType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="CZK"/>
    <xs:enumeration value="EUR"/>
    <xs:enumeration value="USD"/>
  </xs:restriction>
</xs:simpleType>
```

Výčet hodnot lze použít pro všechny datové typy, ne jen pro řetězce.

Maska hodnoty

Velmi silným nástrojem jsou regulární výrazy, které umožňují zapsání masky, jíž musí hodnota vyhovět. Pro zápis omezení na základě regulárního výrazu se používá element `pattern`. XML schémata používají podobnou syntaxi regulárních výrazů jako jazyk Perl. K této syntaxi se za chvíli ještě vrátíme.

WXS
```
<xs:simpleType name="PSČType">
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{3} \d{2}"/>
  </xs:restriction>
</xs:simpleType>
```

Omezení číselných typů

U číselných typů můžeme omezit přípustné hodnoty zdola (`minInclusive`, `minExclusive`) i shora (`maxInclusive`, `maxExclusive`). Navíc lze definovat celkový počet platných číslic (`totalDigits`) a počet míst za desetinnou čárkou (`fractionDigits`).

WXS
```
<xs:simpleType name="částkaType">
  <xs:restriction base="xs:decimal">
    <xs:minInclusive value="0"/>
    <xs:maxExclusive value="1000000"/>
    <xs:fractionDigits value="2"/>
  </xs:restriction>
</xs:simpleType>
```

Anonymní definice typu

Chceme-li pro nějaký element nebo atribut použít vlastní datový typ, nemusíme jej ve schématu přímo deklarovat. Lze používat i anonymní datové typy, které definují uživatelský typ pouze pro jeden element či atribut a nejde se na ně odkazovat z jiných definic. Příklad s definicí elementu pro PSČ tak můžeme alternativně zapsat jako:

WXS
```
<xs:element name="psc">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="6"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

3.2.2 Regulární výrazy

Regulární výrazy umožňují efektivním způsobem omezit hodnoty, které lze použít jako hodnoty pro uživatelsky vytvořený datový typ. Při definici typů můžeme uvést několik omezení `pattern`, alespoň jednomu z nich pak musí vyhovět hodnota uvedená v dokumentu.

WXS
```
<xs:simpleType name="pohlaví">
  <xs:restriction base="xs:string">
    <xs:pattern value="muž"/>
    <xs:pattern value="žena"/>
  </xs:restriction>
</xs:simpleType>
```

Na výše uvedenou definici samozřejmě nemusíme používat regulární výrazy, na to by nám stačil i prostý výčet. Silou regulárních výrazů je možnost používat speciální obecné vzory, kterým vyhoví velké množství hodnot v určitém tvaru.

Poznámka

Regulárnímu výrazu ve WXS musí vždy vyhovět celá hodnota, ne jen její část, jak je to běžné například pro perlové regulární výrazy. Ekvivalentní perlový výraz by se v našem případě musel napsat jako `^muž$`.

Určení počtu opakování

Za normálních okolností se každý znak v regulárním výrazu musí právě jednou vyskytovat i v testované hodnotě. Toto chování však můžeme jednoduše ovlivnit tím, že znakem uvedeme indikátor počtu opakování.

Tabulka 3.2. Opakování znaku v regulárním výrazu

Znak	Význam
`?`	Předchozí znak se nemusí v testované hodnotě vyskytovat (odpovídá `{0,1}`).
`+`	Předchozí znak se musí v testované hodnotě vyskytovat alespoň jednou (odpovídá `{1,}`).
`*`	Předchozí znak se může v testované hodnotě vyskytovat kolikrát chce (odpovídá `{0,}`).
`{`*m*`,`*n*`}`	Předchozí znak se musí v testované hodnotě vyskytovat nejméně *m*-krát a nejvíce *n*-krát.
`{`*n*`,}`	Předchozí znak se musí v testované hodnotě vyskytovat nejméně *n*-krát.
`{`*n*`}`	Předchozí znak se musí v testované hodnotě vyskytovat právě *n*-krát.

Řetězec, který začíná dvěma písmeny ‚a‘, za kterými se opakuje libovolný počet písmen ‚b‘ (minimálně však jedno) a na konci může končit znakem ‚c‘, tak můžeme popsat pomocí výrazu `a{2}b+c?`.

Zápis speciálních znaků

Jak jsme viděli, některé znaky jako ‚`?`‘ nebo ‚`{`‘ mají ve výrazu speciální význam. Pokud chceme tyto znaky ve výrazu zapsat v jejich původním významu, musíme použít jejich přepis, který začíná zpětným lomítkem.

Tabulka 3.3. Zápis speciálních znaků v regulárním výrazu

Znak	Přepis
konec řádky (LF)	`\n`
konec řádky (CR)	`\r`
tabulátor	`\t`
`\`	`\\`
`|`	`|`
`.`	`\.`
`-`	`\-`
`^`	`\^`
`?`	`\?`

Znak	Přepis
*	*
+	\+
{	\{
}	\}
(	\(
)	\)
[	\[
]	\]

Divoký znak

Znak „.“ má speciální význam – zastupuje libovolný znak (s výjimkou znaků konce řádky). Například výrazu `.*` vyhoví jakýkoliv řetězec i prázdný, `.+` zase vyhoví jakýkoliv řetězec, který má alespoň jeden znak.

Můžeme tak například jednoduše definovat typ, který zahrne jen sudá čísla (končí na 0, 2, 4, 6 nebo 8).

WXS
```
<xs:simpleType name="sudéČíslo">
  <xs:restriction base="xs:int">
    <xs:pattern value=".*0"/>
    <xs:pattern value=".*2"/>
    <xs:pattern value=".*4"/>
    <xs:pattern value=".*6"/>
    <xs:pattern value=".*8"/>
  </xs:restriction>
</xs:simpleType>
```

Třídy znaků

Třídy znaků jsou asi vůbec nejsilnější zbraní regulárních výrazů. Třída znaků vždy zastupuje celou skupinu znaků, které mají nějakou společnou vlastnost. Například třída `\d` zastupuje jakoukoliv číslici 0–9. Výraz pro poštovní směrovací číslo tak můžeme zapsat jako `\d\d\d \d\d`, resp. `\d{3} \d{2}`. Podobných tříd znaků máme k dispozici několik. Základní třídy shrnuje tabulka 3.4 – „Základní třídy znaků“.

Tabulka 3.4. Základní třídy znaků

Třída	Popis
\s	Mezery (kromě klasické mezery zahrnuje i tabulátory a znaky konce řádku).
\S	Všechny znaky kromě mezer.
\d	Číslice 0–9 včetně číslic v dalších abecedách, např. ٠١٢٣٤٥٦٧٨٩ (arabsky), ०१२३४५६७८९ (devanagari).
\D	Všechny znaky kromě číslic.
\w	Znaky, které tvoří slovo (tj. všechny znaky kromě těch, které jsou v Unicode definované jako oddělovače, interpunkce nebo ostatní).
\W	Znaky, které netvoří slovo.

Třída	Popis
`\i`	Znaky, kterými může v XML začínat jméno (tj. všechna písmena a znaky ‚-' a ‚:'[a]).
`\I`	Znaky, kterými nemůže v XML začínat jméno.
`\c`	Znaky, které může obsahovat jméno elementu/atribut v XML.
`\C`	Znaky, které nemůže obsahovat jméno elementu/atribut v XML.

[a]Jen připomeňme, že samotné XML dovoluje použít dvojtečku ve jménu elementu/atributu, nicméně v praxi se to nedělá, protože tento znak odděluje prefix jmenného prostoru od lokálního jména.

Kromě základních tříd znaků můžeme v regulárních výrazech používat unicodové třídy znaků a bloky unicodových znaků. Unicodové třídy zastupují vždy celou skupinu znaků se stejným použitím – např. číslice, interpunkce – bez ohledu na to, do jaké patří abecedy. Oproti tomu bloky unicodových znaků vždy obsahují znaky patřící do nějaké abecedy – např. latinky, azbuky, devanagari.[1]

Na unicodovou třídu znaků se odvoláme zápisem `\p{`*jméno*`}`. Na blok znaků se odvoláme zápisem `\p{Is`*jméno*`}`. Můžeme se odkázat i na všechny znaky, které do dané třídy nebo bloku znaků nepatří, stačí použít velké písmeno `\P{`*jméno*`}`.

Tabulka 3.5. Unicodové třídy znaků

Název třídy	Popis
Písmena	
L	Všechna písmena
Lu	Velká písmena
Ll	Malá písmena
Lt	Písmena titulku (velmi speciální případ, má smysl pro některé unicodové znaky, které reprezentují složení dvou znaků, např. Dž)
Lm	Modifikátor písmene (používá se například pro upřesnění výslovnosti předchozího písmene)
Lo	Ostatní písmena (zahrnuje především písmena abeced, které nerozlišují malá a velká písmena – např. arabština, hebrejština, různé indické a asijské abecedy)
Znaménka	
M	Všechna znaménka
Mn	Znaménka nezabírající místo (zejména diakritická znaménka, která jde kombinovat s dalšími znaky)
Mc	Znaménka zabírající místo (využívající se zejména v indických skriptech)
Me	Ohraničující znaménka (např. ◯, které se umí zkombinovat s předchozím znakem a obalit ho)
Číslice	
N	Všechny číslice
Nd	Desítkové číslice
Nl	Číslice zapisované písmeny (například římské číslice – MMVI)
No	Ostatní číslice (například horní indexy, zlomky – ¼, ², ○)

[1]Příslušnost znaku do třídy jde zjistit v unicodové databázi – http://www.unicode.org/Public/UNIDATA/UnicodeData.txt. K dispozici je i přehled bloků – http://www.unicode.org/Public/UNIDATA/Blocks.txt.

Název třídy	Popis
Interpunkce	
P	Veškerá interpunkce
Pc	Spojky (například podtržítko)
Pd	Pomlčky (například -, –, —)
Ps	Otevírací interpunkce (například všechny levé závorky)
Pe	Uzavírací interpunkce (například všechny pravé závorky)
Pi	Otevírací uvozovky (ale i apostrof, francouzská uvozovka apod. – «„‚)
Pf	Uzavírací uvozovky (ale i apostrof, francouzská uvozovka apod. – »“‘)
Po	Ostatní interpunkce (například otazník a vykřičník)
Oddělovače	
Z	Všechny oddělovače
Zs	Mezery
Zl	Oddělovač řádky
Zp	Oddělovač odstavce
Symboly	
S	Všechny symboly
Sm	Matematické symboly (například ±×↑∀∋√∞)
Sc	Měnové symboly (například $¢£¤¥€)
Sk	Modifikátory (například samostatná diakritická znaménka)
So	Ostatní symboly (například §©®°¶≠)
Ostatní znaky	
C	Všechny ostatní znaky
Cc	Řídící znaky (například tabulátor, znaky CR a LF)
Cf	Formátovací znaky (například indikátor možného rozdělení slova)
Co	Znaky v privátní oblasti Unicode
Cn	Dosud nezařazené znaky

Tabulka 3.6. Bloky unicodových znaků

BasicLatin
Latin-1Supplement
LatinExtended-A
LatinExtended-B
IPAExtensions
SpacingModifierLetters
CombiningDiacriticalMarks
Greek
Cyrillic
Armenian
Hebrew
Arabic
Syriac
Thaana
Devanagari
Bengali
Gurmukhi
HangulJamo
Ethiopic
Cherokee
UnifiedCanadianAboriginalSyllabics
Ogham
Runic
Khmer
Mongolian
LatinExtendedAdditional
GreekExtended
GeneralPunctuation
SuperscriptsandSubscripts
CurrencySymbols
CombiningMarksforSymbols
LetterlikeSymbols
NumberForms
Arrows
KangxiRadicals
IdeographicDescriptionCharacters
CJKSymbolsandPunctuation
Hiragana
Katakana
Bopomofo
HangulCompatibilityJamo
Kanbun
BopomofoExtended
EnclosedCJKLettersandMonths
CJKCompatibility
CJKUnifiedIdeographsExtensionA
CJKUnifiedIdeographs
YiSyllables
YiRadicals
HangulSyllables
PrivateUse

Gujarati	MathematicalOperators	CJKCompatibilityIdeographs
Oriya	MiscellaneousTechnical	AlphabeticPresentationForms
Tamil	ControlPictures	ArabicPresentationForms-A
Telugu	OpticalCharacterRecognition	CombiningHalfMarks
Kannada	EnclosedAlphanumerics	CJKCompatibilityForms
Malayalam	BoxDrawing	SmallFormVariants
Sinhala	BlockElements	ArabicPresentationForms-B
Thai	GeometricShapes	Specials
Lao	MiscellaneousSymbols	HalfwidthandFullwidthForms
Tibetan	Dingbats	Specials
Myanmar	BraillePatterns	
Georgian	CJKRadicalsSupplement	

Se znalostí těchto tříd si klidně můžeme definovat typ, který akceptuje částku včetně měny (např. `123.5 $`):

WXS
```
<xs:simpleType name="částkaVčetněMěny">
  <xs:restriction base="xs:token">
    <xs:pattern value="\d+\.?\d*\s*\p{Sc}"/>
  </xs:restriction>
</xs:simpleType>
```

Třídy znaků si můžeme definovat i vlastní. Třída se definuje jako množina znaků zapsaná do hranatých závorek. Například zápis `[abc]` definuje třídu, která zastupuje libovolný ze znaků ‚a', ‚b' a ‚c'.

Pomocí znaku ‚-' můžeme jednoduše definovat interval znaků. Například všechna písmena od ‚a' do ‚z' můžeme zapsat jako `[a-z]`, číslice od 0 do 9 zase jako `[0-9]`.

K uživatelsky definované třídě lze vytvořit doplněk, pokud jako první znak ve hranatých závorkách použijeme ‚^'. Všechny znaky kromě ‚a', ‚b' a ‚c' můžeme definovat jako `[^abc]`, všechny znaky kromě písmen ‚a' až ‚z' pak jako `[^a-z]`.

Uvnitř hranatých závorek se můžeme odvolávat i na ostatní třídy znaků. Můžeme například definovat výraz, kterému vyhoví jen řecká písmena, číslice a matematické symboly – `[\p{IsGreek}0-9\p{Sm}]`. Uživatelské třídy znaků jde samozřejmě kombinovat s dalšími konstrukcemi regulárních výrazů. Pro poštovní směrovací číslo tak můžeme vytvořit přesnější regulární výraz, který nám nedovolí zadat číslo třeba v devanagari, ale jen skutečně pomocí číslic 0–9: `[0-9]{3} [0-9]{2}`.

Hranaté závorky nedefinují nic jiného než množinu složenou ze znaků, případně její doplněk (jeli první znak ‚^'). Uživatelskou třídu znaků lze definovat i pomocí množinového rozdílu, slouží k tomu zápis `[`*`A`*`-[`*`B`*`]]`. Můžeme tak snadno definovat regulární výraz, kterému vyhoví jen písmena řecké abecedy s výjimkou znaků ‚α', ‚β' a ‚γ'.

WXS
```
<xs:simpleType name="řeckéSlovoBezAlfaBetaGama">
  <xs:restriction base="xs:string">
    <xs:pattern value="[\p{IsGreek}-[αβγ]]+"></xs:pattern>
  </xs:restriction>
</xs:simpleType>
```

Kombinování dílčích výrazů

V regulárních výrazech ještě můžeme používat znak ‚|', který umožňuje zadat několik variant, kterým mají testované hodnoty vyhovět. Naše dřívější příklady tak můžeme zjednodušit na:

WXS
```
<xs:simpleType name="pohlaví">
  <xs:restriction base="xs:string">
    <xs:pattern value="muž|žena"/>
```

```
  </xs:restriction>
</xs:simpleType>
```

wxs
```
<xs:simpleType name="sudéČíslo">
  <xs:restriction base="xs:int">
    <xs:pattern value=".*0|.*2|.*4|.*6|.*8"/>
  </xs:restriction>
</xs:simpleType>
```

Pomocí závorek pak můžeme dílčí regulární výrazy kombinovat a vytvářet složitější. Poslední příklad můžeme zapsat také jako `.*(0|2|4|6|8)` nebo `.*[02468]`.

Jde pak vytvářet v podstatě libovolně složité výrazy. Následující příklad definuje typ, kterému vyhoví jakýkoliv seznam slov oddělených čárkami nebo středníky.

wxs
```
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="\w+\s*([,;]\s*\w+)*"/>
  </xs:restriction>
</xs:simpleType>
```

Příklad 3.1. Využití regulárních výrazů – `wxs/re.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="doc">
    <xs:complexType>
      <xs:sequence>

        <xs:element name="řeckyBezAlfaBetaGama" maxOccurs="unbounded">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="[\p{IsGreek}-[αβγ]]+"></xs:pattern>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>

        <xs:element name="seznamSlovOddělenýČárkamiNeboStředníky" maxOccurs="unbounded">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="\w+\s*([,;]\s+\w+)*"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>

        <xs:element name="pščJakoToken" maxOccurs="unbounded">
          <xs:simpleType>
            <xs:restriction base="xs:token">
              <xs:pattern value="[0-9]{3} [0-9]{2}"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>

        <xs:element name="pščJakoString" maxOccurs="unbounded">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="[0-9]{3} [0-9]{2}"/>
```

```
          </xs:restriction>
        </xs:simpleType>
      </xs:element>

      <xs:element name="rč" maxOccurs="unbounded">
        <xs:simpleType>
          <xs:restriction base="xs:token">
            <xs:pattern value="[0-9]{6}/[0-9]{3,4}"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>

      <xs:element name="rčLépeKontrolované" maxOccurs="unbounded">
        <xs:simpleType>
          <xs:restriction base="xs:token">
          <xs:pattern value="[0-9]{2}([05][1-9]|[16][0-2])([012][0-9]|3[01])/[0-9]{3,4}"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>

      <xs:element name="email" maxOccurs="unbounded">
        <xs:simpleType>
          <xs:restriction base="xs:token">
            <xs:pattern ▶
value="[&#33;-&#126;-[()&lt;>@;:\\&quot;\[\]]]+@[&#33;-&#126;-[()&lt;>@;:\\&quot;\[\]]]+"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>

      <xs:element name="částkaVčetněMěny" maxOccurs="unbounded">
        <xs:simpleType>
          <xs:restriction base="xs:token">
            <xs:pattern value="\d+\.?\d*\s*\p{Sc}"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>

      <xs:element name="sudéČíslo" maxOccurs="unbounded">
        <xs:simpleType>
          <xs:restriction base="xs:int">
            <xs:pattern value=".*[02468]"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>

    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

3.2.3 Normalizace hodnot a integritní omezení

Pro správné pochopení integritních omezení je potřeba si uvědomit, že hodnota, kterou kontrolují, nemusí být stejná jako hodnota zapsaná přímo do dokumentu XML.

Jak ukazuje obrázek 3.2 – „Jak se text z dokumentu XML přemění na hodnotu“ údaj uložený v dokumentu XML se nejprve normalizuje. Při této normalizaci se bílé znaky jako konce řádky a tabulátory nahradí mezerou. Poté se oříznou veškeré mezery na začátku a konci hodnoty a vícenásobný výskyt znaku mezera se nahradí mezerou jedinou. Tato normalizace se provádí vždy s výjimkou typů `string`, `normalizedString` a od nich odvozených uživatelských typů. Pro typ `string` se neprovádí vůbec žádná normalizace a pro `normalizedString` se pouze znaky konce řádků a tabulátory převedou na mezery.

Po normalizaci údaje z dokumentu XML získáme jeho lexikální reprezentaci. Nad tímto lexikálním prostorem pracuje integritní omezení pomocí regulárního výrazu (`pattern`). Všechny ostatní kontroly, jako počet desetinných míst, přípustný rozsah hodnot apod., se provádějí až nad prostorem hodnot. Ten obsahuje již abstraktní reprezentaci původního údaje. Na obrázku 3.2 – „Jak se text z dokumentu XML přemění na hodnotu“ je tak vidět, jak se původně tři různé zápisy nakonec převedly na stejnou hodnotu 3½.

3.2.4 Odvození typů seznamem

Ačkoliv je ve většině případů lepší veškeré struktury značkovat pomocí elementů nebo atributů, může se někdy hodit strukturu nevyznačovat explicitně přímo prostředky jazyka XML. WXS podporují jako datový typ seznamy hodnot oddělené mezerami (obecně bílými znaky). Tento seznam se může skládat z několika hodnot stejného datového typu.

Pro seznamy si musíme vytvořit vlastní datový typ, který je definován pomocí `list` a tento typ pak použít při definici elementu nebo atributu.

WXS
```
<xs:simpleType name="seznamČísel">
  <xs:list itemType="xs:int"/>
</xs:simpleType>

<xs:element name="čísla" type="seznamČísel"/>
```

V dokumentu pak můžeme používat zápisy jako

```
<čísla>1 2 3 4 5 6 42 999 17</čísla>
```

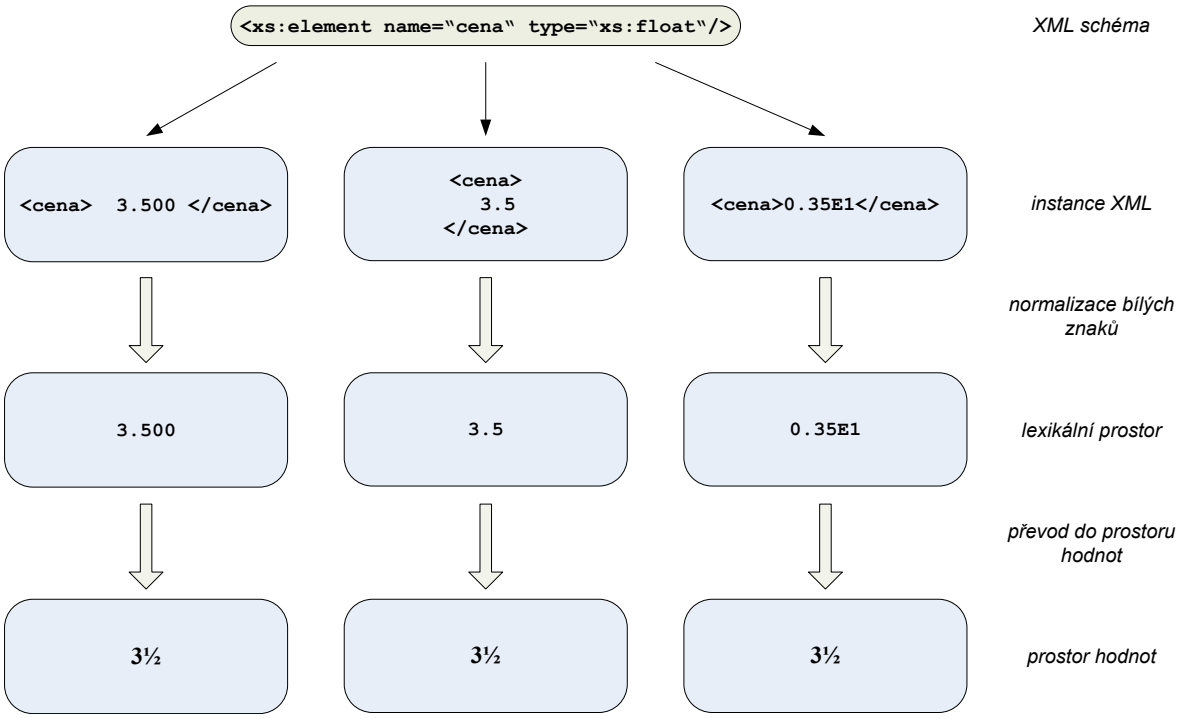
Typ položek seznamu nemusíme určovat jen pomocí atributu `itemType`, ale můžeme použít vnoření definice typu:

WXS
```
<xs:simpleType name="seznamČísel">
  <xs:list>
    <xs:simpleType>
      <xs:restriction base="xs:int"/>
    </xs:simpleType>
  </xs:list>
</xs:simpleType>
```

Pro celý seznam lze nastavit integritní omezení omezující délku seznamu. Musíme si však vytvořit nový seznam, odvozený restrikcí od existujícího, a integritní omezení nastavit na tom nově vytvářeném. Následující příklad ukazuje definici typu pro seznam s přesně třemi hodnotami:

WXS
```
<xs:simpleType name="míry">
  <xs:restriction base="seznamČísel">
    <xs:length value="3"/>
  </xs:restriction>
</xs:simpleType>
```

Obrázek 3.2. Jak se text z dokumentu XML přemění na hodnotu



3.2.5 Odvození typů sjednocením

WXS umožňují vytvořit datový typ, který akceptuje hodnoty několika jiných různých datových typů. Předpokládejme, že chceme definovat element `věk` tak, aby akceptoval číselné hodnoty a zároveň hodnotu „neuvedeno“:

```
<věk>30</věk>

<věk>neuvedeno</věk>
```

Stačí využít konstrukci `union` a sjednotit datový typ pro kladná čísla s typem pro řetězec „neuvedeno“:

WXS
```
<xs:element name="věk">
  <xs:simpleType>
    <xs:union>
      <xs:simpleType>
        <xs:restriction base="xs:positiveInteger"/>
      </xs:simpleType>
      <xs:simpleType>
        <xs:restriction base="xs:token">
          <xs:enumeration value="neuvedeno"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:union>
  </xs:simpleType>
</xs:element>
```

3.3 Komplexní datové typy

Komplexní typy slouží k modelování struktury dokumentu, protože se mohou skládat z elementů a atributů. U elementů můžeme určit v jakém pořadí se mají vyskytovat, kolikrát se mohou opakovat, zda jsou povinné či volitelné.

Komplexní typ se definuje pomocí elementu `complexType`. Podobně jako u jednoduchých typů můžeme definovat komplexní typ samostatně, aby pak šel používat opakovaně pro různé elementy. Následující příklad ukazuje, jak definovat elementy `odberatel` a `dodavatel` tak, že mají stejný obsah, právě pomocí společně použitého uživatelsky definovaného komplexního typu `subjektType`.

WXS
```
<xs:element name="faktura">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="odberatel" type="subjektType"/>
      <xs:element name="dodavatel" type="subjektType"/>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="subjektType">
  <xs:sequence>
    <xs:element name="nazev" type="xs:string" />
    <xs:element name="adresa" type="xs:string" />
    <xs:element name="ico" type="xs:string" />
    <xs:element name="dic" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

Druhou možností je použít `complexType` přímo v definici elementu. Takto je v předchozím příkladě definován obsah elementu `faktura`.

Uvnitř elementu `complexType` pak můžeme použít další elementy `sequence`, `choice` a `all` (tzv. kompozitory). Elementy definované uvnitř `sequence` se musí v dokumentu vyskytovat v definovaném pořadí. Oproti tomu `all` říká, že elementy se mohou vyskytovat v libovolném pořadí. Konečně `choice` říká, že se v dokumentu může vyskytnout jen jeden z obsažených elementů. Tyto elementy lze do sebe podle potřeby zanořovat (s výjimkou `all`) a tak modelovat složitější situace.

Následující příklad definuje schéma pro článek, který má název, libovolný počet autorů a pak se v něm podle potřeby vyskytuje libovolný počet obrázků anebo odstavců.

WXS
```
<xs:element name="clanek">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nazev" type="xs:string"/>
      <xs:element name="autor" type="xs:string"
                  minOccurs="0" maxOccurs="unbounded"/>
      <xs:choice minOccurs="1" maxOccurs="unbounded">
        <xs:element name="odstavec" type="xs:string"/>
        <xs:element name="obrazek" type="xs:base64Binary"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Příklad také demonstruje, jakým způsobem lze určovat počet výskytů elementu nebo celé jejich skupiny. Slouží k tomu atributy `minOccurs` a `maxOccurs`. Jejich standardní hodnota je jedna. Pro nekonečno se používá identifikátor `unbounded`.

3.3.1 Sekvence elementů – `sequence`

Nejpoužívanější konstrukcí uvnitř komplexního typu je sekvence elementů (`sequence`). Tento kompozitor říká, že v něm obsažené definice elementů (případně dalších vnořených struktur) se musí v dokumentu vyskytovat přesně v tom pořadí, jak jsou uvedeny. Počet výskytů jednotlivých elementů je možné ovlivnit právě pomocí atributů `minOccurs` a `maxOccurs`. Implicitně mají oba hodnotu jedna, což odpovídá povinnému výskytu elementu.

Následující příklad definuje element pro uložení článku, který má povinný název, nepovinného autora a dále následuje libovolný počet odstavců, přičemž vždy musí být uveden alespoň jeden odstavec.

Příklad 3.2. Sekvence elementů

```
<článek>
  <název>Ukázka</název>
  <autor>Pepa</autor>
  <odstavec>...</odstavec>
  <odstavec>...</odstavec>
</článek>
```

WXS

```
<xs:element name="článek">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="název" type="xs:string"/>
      <xs:element name="autor" type="xs:string" minOccurs="0"/>
      <xs:element name="odstavec" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

3.3.2 Výběr jednoho z elementů – `choice`

Chceme-li říci, že na určitém místě dokumentu se může vyskytovat jeden z několika elementů, použijeme k tomu kompozitor `choice`. Na jeho místě se pak může vyskytnout jakýkoliv element definovaný uvnitř této konstrukce.

Následující příklad ukazuje schéma, které modeluje seznam osob. U každé osoby přitom musí být uveden jeden ze tří identifikátorů – rodné číslo, číslo pasu nebo číslo sociálního pojištění.

Příklad 3.3. Výběr jednoho z elementů – `wxs/choice.xsd`

```
<osoby>
  <osoba>
    <jméno>Pepa Tuzemec</jméno>
    <RČ>681203/0123</RČ>
  </osoba>
  <osoba>
    <jméno>Pepa Cizinec</jméno>
    <pas>1234567</pas>
  </osoba>
  <osoba>
    <jméno>Pepa Rozvědčík</jméno>
```

```
    <SSN>987654321</SSN>
  </osoba>
</osoby>
```

WXS
```
<xs:element name="osoby">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="osoba" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="jméno" type="xs:string"/>
            <xs:choice>
               <xs:element name="RČ" type="xs:string"/>
               <xs:element name="pas" type="xs:string"/>
               <xs:element name="SSN" type="xs:string"/>
            </xs:choice>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

3.3.3 Elementy v libovolném pořadí – `all`

V mnoha případech nám nezáleží na pořadí, v jakém jsou nějaké elementy uvedeny a nechceme uživatele zbytečně nutit k tomu, aby dodržoval nějaké konkrétní pořadí. V tomto případě se obsah komplexního typu definuje pomocí kompozitoru `all`. Nepříjemným omezením je, že u jednotlivých elementů ve skupině `all` můžeme počet jejich výskytů definovat pouze v rozmezí od nuly do jedné. Větší počet výskytů není dovolen, protože by se příliš zesložitil model obsahu.

Následující příklad ukazuje definici elementu `osoba`, který může obsahovat jméno a příjmení v libovolném pořadí, navíc se v elementu může vyskytovat i titul.

Příklad 3.4. Elementy v libovolném pořadí – `wxs/all.xsd`

```
<osoba>
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
</osoba>
<osoba>
  <příjmení>Novák</příjmení>
  <jméno>Jan</jméno>
</osoba>
<osoba>
  <titul>Ing.</titul>
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
</osoba>
<osoba>
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
  <titul>CSc.</titul>
</osoba>
```

WXS
```
<xs:element name="osoba">
  <xs:complexType>
```

```
    <xs:all>
      <xs:element name="jméno" type="xs:string"/>
      <xs:element name="příjmení" type="xs:string"/>
      <xs:element name="titul" type="xs:string" minOccurs="0"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Uvedené schéma však není schopno zachytit případy, kdy má jeden člověk tituly dva, jeden před jménem a druhý za jménem, protože uvnitř `all` nejde elementům nastavit `maxOccurs` na větší hodnotu než jedna. Vzhledem k tomu, že možnost kombinování `all` s ostatními konstrukty je velmi omezená,[2] je potřeba se při řešení našeho problému obejít bez `all` a ručně vypsat všechny kombinace elementů v různém pořadí.

Příklad 3.5. Schéma pro osobu s tituly a se jménem a příjmením v libovolném pořadí

WXS

```
<xs:element name="osoba">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="titul" type="xs:string" minOccurs="0"/>
      <xs:choice>
        <xs:sequence>
          <xs:element name="jméno" type="xs:string"/>
          <xs:element name="příjmení" type="xs:string"/>
        </xs:sequence>
        <xs:sequence>
          <xs:element name="příjmení" type="xs:string"/>
          <xs:element name="jméno" type="xs:string"/>
        </xs:sequence>
      </xs:choice>
      <xs:element name="titul" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

3.3.4 Smíšený obsah

V textově zaměřených dokumentech jako jsou různé články, dokumentace apod. často potřebujeme používat tzv. smíšený obsah (mixed content). Je to situace, kdy může být text na stejné úrovni kombinován s dalšími elementy. Typickým příkladem je element pro odstavec. Ten může obsahovat jak přímo text, tak i další elementy pro zvýraznění textu, vytváření odkazů apod. Definice smíšeného obsahu je velmi jednoduchá. U definice komplexního typu elementu stačí přidat atribut `mixed` s hodnotou `true`.

Smíšený obsah v XML schématech funguje poněkud odlišně než v DTD. V DTD smíšený obsah vždy automaticky implikuje, že elementy na stejné úrovni jako text se mohou vyskytovat v libovolném pořadí a opakovaně. V XML schématech smíšený obsah říká, že text se může objevit kdekoliv mezi elementy, které komplexní typ definuje. Smíšený obsah v klasickém slova smyslu se proto musí definovat pomocí skupiny `choice`, která má libovolný počet výskytů a obalí všechny elementy, které se mohou mísit s textem.

[2] Skutečnost je taková, že `all` může být použito jen přímo uvnitř komplexního typu a nemůže být kombinováno s dalšími konstrukcemi jako `sequence` a `choice`.

Příklad 3.6. Smíšený obsah – `wxs/odstavce.xsd`

```
<odstavec>Odstavce typicky obsahují <pojem>smíšený
obsah</pojem>. Text se může střídat
s <odkaz url="http://www.kosek.cz">odkazy</odkaz>
a dalšími <pojem>elementy</pojem>.</odstavec>

<odstavec>Odstavec může obsahovat i jen text.</odstavec>

<odstavec><pojem>Nebo jen element.</pojem></odstavec>
```

WXS

```
<xs:element name="odstavec">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="pojem" type="xs:string"/>
      <xs:element name="odkaz">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute name="url" type="xs:anyURI"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

3.3.5 Prázdné elementy

Prázdné elementy jsou takové elementy, které nemají žádný obsah. V instanci dokumentu se zapisují jedním z následujících dvou způsobů:

```
<prazdny/>
<prazdny></prazdny>
```

Prázdné elementy mají obvykle atributy. Atributy se musí definovat jako součást komplexního typu. Např. pro element `img` s atributem `src`

```
<img src="obrazek.png"/>
```

by definice elementu vypadala následovně

WXS

```
<xs:element name="img">
  <xs:complexType>
    <xs:attribute name="src" type="xs:anyURI"/>
  </xs:complexType>
</xs:element>
```

Jedná se přitom o zkrácený zápis. Plná syntaxe (kterou samozřejmě nemusíte používat) vychází z idey, že elementu odebereme jeho obsah a přidáme k němu atribut.

WXS

```
<xs:element name="img">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:attribute name="src" type="xs:anyURI"/>
      </xs:restriction>
```

```
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

3.3.6 Atributy

Atribut je součástí komplexního typu a definuje se pomocí elementu `attribute`:

WXS
```
<xs:attribute name="vek" type="xs:positiveInteger"/>
```

U atributů můžeme určit nejen jejich jméno a typ, ale zda mají být povinné, jaká je jejich defaultní hodnota apod. Např.:

WXS
```
<xs:attribute name="ident" type="xs:ID" use="required"/>

<xs:attribute name="měna" type="xs:string" default="USD"/>
```

Atributy se uvádějí vždy až na konci komplexního typu. Jediným složitějším případem je situace, kdy chceme atributy přidat k elementu, který neobsahuje již další podelementy, ale má jednoduchý datový typ. Výsledný zápis je pak poněkud krkolomný. Představme si, že chceme definovat element `cena`, který bude mít atribut `měna` pro uložení kódu měny.

```
<cena měna="USD">23.69</cena>
```

Chceme přitom využít dříve definované datové typy. V řeči XML schémat vypadá definice následovně:

WXS
```
<xs:element name="cena">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="částkaType">
        <xs:attribute name="měna" type="kódMěnyType"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Přeloženo do lidské řeči: Vytvoř komplexní typ, který vznikne rozšířením jednoduchého obsahu s typem `částkaType` o atribut `měna`. Obecně tedy definice elementu, který má nějaký jednoduchý typ jako svůj obsah a kromě toho má další atributy, vypadá:

WXS
```
<xs:element name="cena">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="typ_obsahu_elementu">
 …definice atributů…
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

3.3.7 anyType

Pokud u nějakého elementu typ neurčíme (tj. vynecháme atribut `type` i anonymní definici typu), bere se to stejně, jako kdyby jeho typ byl `anyType`. Elementu tohoto typu vyhoví cokoliv – může obsahovat text nebo další libovolné vnořené elementy.

3.4 Globální a lokální deklarace

Pro dokonalé pochopení XML schémat, se musíme seznámit s lokálními a globálními deklaracemi a rozdílem mezi nimi.

Za globální se považují deklarace provedené na nejvyšší úrovni schématu, přímo v elementu `schema`. Globálně deklarované elementy a datové typy mají v některých ohledech speciální chování. Dokument vyhovující schématu může začínat libovolným globálním elementem. Ve většině schémat má proto smysl deklarovat jako globální pouze ten element, který chceme použít jako obálku okolo celého dokumentu XML. O různých možnostech, jak toho dosáhnout pojednává sekce 3.7 – „Přístupy k návrhu schématu“.

Jako ukázka nevhodného přístupu může sloužit následující schéma faktury, ve kterém jsou jako globální deklarovány elementy `faktura` a `polozka`. Validací tedy projde nejen kompletní faktura, ale i jedna jediná položka.

Příklad 3.7. Schéma definující dva globální elementy – `wxs/faktura.xsd`

WXS

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="urn:x-kosek:schemas:faktura:1.0"
           xmlns="urn:x-kosek:schemas:faktura:1.0"
           elementFormDefault="qualified">
  <xs:element name="faktura">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="odberatel" type="subjektInfoTyp" />
        <xs:element name="dodavatel" type="subjektInfoTyp" />
        <xs:element ref="polozka" minOccurs="1" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="cislo" type="cisloFakturyTyp" use="required" />
      <xs:attribute name="vystaveni" type="xs:date" use="required" />
      <xs:attribute name="splatnost" type="xs:date" use="required" />
      <xs:attribute name="vystavil" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:complexType name="subjektInfoTyp">
    <xs:sequence>
      <xs:element name="nazev" type="xs:string" />
      <xs:element name="adresa" type="xs:string" />
      <xs:element name="ico" type="icoTyp" />
      <xs:element name="dic" type="dicTyp" />
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="icoTyp">
    <xs:restriction base="xs:string">
      <xs:pattern value="\d{10}" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="dicTyp">
    <xs:restriction base="xs:string">
      <xs:pattern value="\d{3}-\d{10}" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="cisloFakturyTyp">
    <xs:restriction base="xs:string">
```

```
      <xs:pattern value="\d{4}/\d{3}" />
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="polozka">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="popis" type="xs:string" minOccurs="0" maxOccurs="1" />
        <xs:element name="cena" type="částkaTyp" />
        <xs:element name="dph" type="dphTyp" />
        <xs:element name="ks" type="xs:positiveInteger" minOccurs="0" maxOccurs="1" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="částkaTyp">
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="0" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="dphTyp">
    <xs:restriction base="xs:decimal">
      <xs:enumeration value="22" />
      <xs:enumeration value="5" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Na globálně deklarované typy a elementy se lze odvolávat z jiných schémat (viz 3.12.5 – „Schéma definující elementy v několika jmenných prostorech“). V další sekci uvidíme, že globálně a lokálně deklarované elementy mají i odlišné vlastnosti vzhledem ke jmenným prostorům.

3.5 Jmenné prostory

Bývá dnes dobrým zvykem přiřadit každému nově vytvořenému značkovacímu jazyku vlastní jmenný prostor, aby jej šlo snadno identifikovat. Jmenný prostor, do něhož budou elementy patřit, se určuje pomocí atributu `targetNamespace` u kořenového elementu schématu. Z praktických důvodů se tento jmenný prostor obvykle deklaruje i jako implicitní, abychom před naše elementy nemuseli všude psát nějaký prefix.

XML schéma má jednu nepříjemnou vlastnost – do jmenného prostoru patří jen globální elementy. Jsou to pouze ty elementy, které jsou ve schématu definované na nejvyšší úrovni přímo pod elementem `schema`. Předpokládejme například následující schéma:

WXS

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="urn:x-kosek:schemas:pokus"
           xmlns="urn:x-kosek:schemas:pokus">
  <xs:element name="a">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="b" type="xs:string"/>
        <xs:element name="c" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Instance, která schématu vyhovuje, pak odporuje ustáleným praktikám pro tvorbu dokumentů XML. Můžeme ji zapsat buď jako:

```
<a xmlns="urn:x-kosek:schemas:pokus">
  <b xmlns="">foo</b>
  <c xmlns="">bar</c>
</a>
```

nebo

```
<p:a xmlns:p="urn:x-kosek:schemas:pokus">
  <b>foo</b>
  <c>bar</c>
</p:a>
```

Nepřehledný a nekonzistentní zápis je právě důsledkem toho, že elementy `b` a `c` nepatří do žádného jmenného prostoru. Každý element by přitom měl patřit do nějakého jmenného prostoru. Toho můžeme dosáhnout tím, že pomocí atributu `elementFormDefault` řekneme, že celé schéma musí používat kvalifikované (rozuměj zařazené do nějakého jmenného prostoru) elementy.

WXS
```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="urn:x-kosek:schemas:pokus"
           xmlns="urn:x-kosek:schemas:pokus"
           elementFormDefault="qualified">
  <xs:element name="a">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="b" type="xs:string"/>
        <xs:element name="c" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

S tímto schématem už můžeme vytvářet dokumenty, tak jak je zvykem.

```
<a xmlns="urn:x-kosek:schemas:pokus">
  <b>foo</b>
  <c>bar</c>
</a>

<p:a xmlns:p="urn:x-kosek:schemas:pokus">
  <p:b>foo</p:b>
  <p:c>bar</p:c>
</p:a>
```

Ve skutečnosti je možné u každého elementu nebo atributu určit, zda má nebo nemá patřit do cílového jmenného prostoru. Slouží k tomu atribut `form`, jehož použití ukazuje následující příklad, který rovněž definuje všechny tři elementy v jednom jmenném prostoru.

WXS
```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="urn:x-kosek:schemas:pokus"
  xmlns="urn:x-kosek:schemas:pokus">
  <xs:element name="a">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="b" type="xs:string" form="qualified"/>
        <xs:element name="c" type="xs:string" form="qualified"/>
```

```
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

V praxi se však příslušenství elementů (případně atributů) do jmenného prostoru obvykle definuje globálně a využívají se proto atributy `elementFormDefault` a `attributeFormDefault`.

Použití jmenných prostorů ve schématech přináší ještě jednu záludnost. Do cílového jmenného prostoru patří nejen globálně deklarované elementy, ale i typy. Odvoláváme-li se pak při definici elementu nebo atributu na uživatelsky definovaný typ, musíme jej určit pomocí jeho kvalifikovaného jména. Dosáhnout tohoto cíle lze dvě způsoby. První spočívá v deklaraci prefixu pro cílový jmenný prostor.

Příklad 3.8. Odkaz na typ s využitím prefixu – `src/ns-typy1.xsd`

WXS

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://example.org/ns"
           xmlns:tns="http://example.org/ns"
           elementFormDefault="qualified">

<xs:element name="root" type="tns:mujTyp"/>

<xs:simpleType name="mujTyp">
  <xs:restriction base="xs:string"/>
</xs:simpleType>

</xs:schema>
```

Druhá možnost využívá toho, že XML schémata rozšiřují dopad výchozího jmenného prostoru ze jmen elementů i na jména datových typů (samozřejmě pouze uvnitř schématu, změna se netýká instancí). Předchozí schéma tak můžeme napsat i následujícím způsobem, který je většinou i běžnější.

Příklad 3.9. Odkaz na typ s využitím výchozího jmenného prostoru – `src/ns-typy2.xsd`

WXS

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://example.org/ns"
           xmlns="http://example.org/ns"
           elementFormDefault="qualified">

<xs:element name="root" type="mujTyp"/>

<xs:simpleType name="mujTyp">
  <xs:restriction base="xs:string"/>
</xs:simpleType>

</xs:schema>
```

3.6 Připojení schématu k dokumentu

Chceme-li přímo v dokumentu XML určit, kde může parser najít schéma například pro účely validace, musíme k tomu použít speciální globální atributy patřící do jmenného prostoru `http://www.w3.org/2001/XMLSchema-instance`.

Nepoužíváme-li jmenné prostory, stačí do atributu `noNamespaceSchemaLocation` uvést URL adresu, na které se nachází schéma. Můžeme použít samozřejmě i relativní URL a jednoduše se tak odkázat na soubor se schématem, který je ve stejném adresáři jako dokument.

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

Používáme-li jmenné prostory, musíme použít atribut `schemaLocation`. Ten může obsahovat několik dvojic hodnot URI jmenného prostoru a umístění schématu. Hodnoty jsou přitom oddělené mezerami nebo jinými bílými znaky.

```
<dokument
  xmlns="urn:x-kosek:schemas:dokument:1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:x-kosek:schemas:dokument:1.0
                      dokument.xsd">
  ...
</dokument>
```

Některé novější aplikace již podporují novou instrukci `<?xml-model?>`. Ta umožňuje připojit jakékoliv schéma bez nutnosti používat speciální atributy.

```
<?xml-model href="dokument.xsd"?>
<dokument>
  ...
</dokument>
```

Obecně však tento způsob připojování schématu nelze doporučit. Hodí se pro účely testování a různých hrátek se schématy. V praxi bychom však atributy `schemaLocation` a `noNamespaceSchemaLocation` neměli používat. Schéma vhodné pro validaci by si měla vybrat sama aplikace, která validaci provádí jako součást zpracování dokumentu. Můžeme tak vždy určit jedno pevné schéma, nebo schéma vybrat na základě nějaké mapovací tabulky[3], kde je pro každý jmenný prostor určeno odpovídající schéma. Tím budeme mít zaručeno, že zvalidovaný dokument opravdu odpovídá schématu, se kterým naše aplikace pro další zpracování dokumentu počítá.

Když se spolehneme na atributy pro připojení schématu, může nám kdokoliv podvrhnout jakýkoliv dokument, pro který někde na webu vystavil svoje vlastní schéma. Takový dokument projde validací, ale naše aplikace bude mít v lepším případě problémy s jeho zpracování, v horším ji to zcela vyvede z míry.

3.7 Přístupy k návrhu schématu

Postupem času se vyvinulo několik přístupů k tomu, jak správně vystavět schéma. Problém, který je potřeba rozhodnout, spočívá ve výběru, kdy používat lokální, a kdy globální elementy, jak moc bude schéma rozšiřitelné a použitelné v dalších schématech.

Rozdíly mezi jednotlivýmu přístupy si ukážeme na následujícím jednoduchém dokumentu.

[3]Pro tyto účely lze využít i NVDL. Některé aplikace nabízejí vlastní formát konfiguračních souborů pro navázání schématu k dokumentu. Například v editoru oXygen je tato volba dostupná pomocí Option▻Preferences…▻Editor▻Default Schema Associations.

Příklad 3.10. Ukázkový dokument – `wxs/zamestnanec.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<zamestnanec>
  <jmeno>Jan</jmeno>
  <prijmeni>Novák</prijmeni>
  <adresa>
    <ulice>Dlouhá 2</ulice>
    <město>Praha 1</město>
    <psč>110 00</psč>
  </adresa>
  <plat>34500</plat>
</zamestnanec>
```

První přístup je označován jako matrjóška, tedy ruská panenka, kdy do sebe zapadají jednotlivé menší a menší panenky. V tomto přístupu je globální jen jeden element a všechny ostatní jsou uvnitř něj definovány jako lokální.

Příklad 3.11. Schéma ve stylu matrjóška – `wxs/zamestnanec-matrjoska.xsd`

WXS

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="zamestnanec">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="jmeno" type="xs:string"/>
        <xs:element name="prijmeni" type="xs:string"/>
        <xs:element name="adresa">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ulice" type="xs:string"/>
              <xs:element name="město" type="xs:string"/>
              <xs:element name="psč" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="plat" type="xs:decimal"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Instance tohoto schématu může začínat jen elementem `zamestnanec`, a jiný element ani nejde znovu použít v jiných schématech, které by toto schéma importovaly. Výhodou je naopak krátké a kompaktní schéma, které je rychle napsané.

Druhý častý přístup se nazývá salámová kolečka. Všechny elementy jsou definovány jako globální a pak jsou složeny dohromady pomocí odkazů.

Příklad 3.12. Schéma ve stylu salámových koleček – `wxs/zamestnanec-salam.xsd`

WXS

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

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

```
  <xs:element name="prijmeni" type="xs:string"/>
  <xs:element name="ulice" type="xs:string"/>
  <xs:element name="město" type="xs:string"/>
  <xs:element name="psč" type="xs:string"/>
  <xs:element name="plat" type="xs:decimal"/>

  <xs:element name="adresa">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ulice"/>
        <xs:element ref="město"/>
        <xs:element ref="psč"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="zamestnanec">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="jmeno"/>
        <xs:element ref="prijmeni"/>
        <xs:element ref="adresa"/>
        <xs:element ref="plat"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Dokument může začínat libovolným elementem, libovolný element můžeme znovu použít. Nevýhoda je, že v tomto přístupu nejde pro element stejného názvu definovat dva různé modely obsahu v závislosti na kontextu jeho výskytu. Tuto možnost první způsob nabízí.

Třetí metoda, označovaná jako metoda slepého Benátčana, pro všechny elementy nejprve definuje typy, které lze znovu používat. Elementy jsou pak definovány lokálně pomocí těchto typů, takže mohou mít při shodě jmen různé modely obsahu. Tento přístup nabízí nejvíce možností a komfortu, a ve většině případů je nejvhodnější. Jeho nevýhodou je větší pracnost a složitost v porovnání s předchozími metodami.

Příklad 3.13. Schéma ve stylu slepého Benátčana – `wxs/zamestnanec-benatcan.xsd`

WXS

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:simpleType name="jmenoType">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="15"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="prijmeniType">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="20"/>
    </xs:restriction>
  </xs:simpleType>
```

```
  <xs:simpleType name="uliceType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>

  <xs:simpleType name="městoType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>

  <xs:simpleType name="pscType">
    <xs:restriction base="xs:token">
      <xs:pattern value="[0-9]{3} [0-9]{2}"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="platType">
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="0"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="adresaType">
    <xs:sequence>
      <xs:element name="ulice" type="uliceType"/>
      <xs:element name="město" type="městoType"/>
      <xs:element name="psč" type="pscType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="zamestnanecType">
    <xs:sequence>
      <xs:element name="jmeno" type="jmenoType"/>
      <xs:element name="prijmeni" type="prijmeniType"/>
      <xs:element name="adresa" type="adresaType"/>
      <xs:element name="plat" type="platType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="zamestnanec" type="zamestnanecType"/>

</xs:schema>
```

3.8 Práce s prázdnými hodnotami

Velkou nevýhodou XML z pohledu databázistů byla neexistence standardního mechanismu pro zaznamenání toho, že nějaký element obsahuje nedefinovanou hodnotu (NULL). Oba XML zápisy

```
<autor></autor>
<autor/>
```

můžeme chápat jako element `autor`, který obsahuje hodnotu prázdný řetězec. Pro zachycení hodnoty NULL můžeme použít další globální atribut (používají se i pro připojení schématu):

```
<autor xsi:nil="true"></autor>
<autor xsi:nil="true"/>
```

To, že element může nabývat prázdnou hodnotu, musíme předtím definovat ve schématu:

```
<xs:element name="autor" nillable="true" type="xs:string"/>
```

WXS

Konstrukci `xsi:nil` využívají zejména různé nástroje pro databinding. Obvyklý přístup ve světě XML je ten, že pokud může být nějaká hodnota nedefinovaná, tak se odpovídající element/atribut do výstupu vůbec nevloží – ve schématu pak musí být definován jako volitelný.

3.9 Zajištění jedinečnosti hodnot

Chceme-li zabránit výskytu duplicitních hodnot v nějaké množině elementů nebo atributů, můžeme si definovat unikátní klíč. Klíčů může být v jednom schématu definováno několik a pro jejich definici se používá dotazovací jazyk XPath.

Použití klíče si ukážeme na příkladě. Předpokládejme, že v následujícím seznamu zaměstnanců chceme zajistit unikátnost jejich osobních čísel (atribut `oc`).

Příklad 3.14. Seznam zaměstnanců – `wxs/zamestnanci.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<zamestnanci>
  <zamestnanec oc="1164">
    <jmeno>Procházka Karel</jmeno>
    <sef>2021</sef>
  </zamestnanec>
  <zamestnanec oc="1168">
    <jmeno>Novotná Alena</jmeno>
    <sef>2021</sef>
  </zamestnanec>
  <zamestnanec oc="1230">
    <jmeno>Klíma Josef</jmeno>
    <sef>1168</sef>
  </zamestnanec>
  <zamestnanec oc="1564">
    <jmeno>Pinkas Josef</jmeno>
    <sef>2021</sef>
  </zamestnanec>
  <zamestnanec oc="2021">
    <jmeno>Kládová Adéla</jmeno>
  </zamestnanec>
</zamestnanci>
```

Klíč proto definujeme u elementu `zamestnanci`, který obsahuje jednotlivé zaměstnance, pro které se má dodržovat unikátnost osobního čísla.

Příklad 3.15. Definice unikátního klíče – `wxs/zamestnanci-unique.xsd`

```
<xs:element name="zamestnanci">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="zamestnanec" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:unique name="oc_je_unikatni">
    <xs:selector xpath="zamestnanec"/>
    <xs:field xpath="@oc" />
  </xs:unique>
</xs:element>
```

WXS

Důležité je, aby definice klíče byla umístěna v definici elementu, který pod sebou jako potomky obsahuje elementy, pro které se referenční integrita hlídá. V opačném případě nebude klíč fungovat správně.

Klíč definovaný pomocí `unique` kontroluje pouze unikátnost hodnot, ale neohlásí chybu, pokud u některého elementu klíč chybí. Chceme-li mít zaručeno, že hodnoty jsou unikátní a jsou uvedeny všude, můžeme použít element `key` (viz příklad `wxs/zamestnanci-key.xsd`).

3.10 Referenční integrita

Podobně jako v relační databázi, můžeme i na půdě jednoho dokumentu XML definovat referenční integritu. Postup je obdobný jako u databází – definujeme si klíče a pak cizí klíče, které na ně ukazují. Následující ukázka ilustruje omezení, které zajistí, že pro každého zaměstnance bude existovat jeho šéf.

Příklad 3.16. Definice referenční integrity – `wxs/zamestnanci-keyref.xsd`

WXS

```
<xs:element name="zamestnanci">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="zamestnanec" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:key name="osobni_cislo">
    <xs:selector xpath="zamestnanec"/>
    <xs:field xpath="@oc"/>
  </xs:key>

  <xs:keyref name="sef_je_existujici_oc"
             refer="osobni_cislo">
    <xs:selector xpath="zamestnanec"/>
    <xs:field xpath="sef"/>
  </xs:keyref>

</xs:element>
```

3.11 Objektově orientované rysy

Při návrhu schémat máme k dispozici několik vlastností, které jsou hodně podobné vlastnostem známým z objektově orientovaných jazyků. Možnost odvozování nových typů na základě již existujících není nic jiného než dědičnost. Odvozování jde přitom použít i pro komplexní typy, ne jen pro jednoduché, jak jsme si ukázali.

Komplexní typy lze odvozovat rozšířením a restrikcí. Při rozšíření jde však nové elementy přidávat pouze na konec modelu obsahu, což je velmi omezující. Při odvození nového komplexního typu restrikcí je potřeba celý nový model obsahu vypsat. Díky těmto omezením jsou tyto rysy WXS většinou nepoužívané, protože je lze nahradit jednodušeji pomocí ostatních konstrukcí. Jediné kdy se jejich použití hodí, je v případech, kdy čteme dokument pomocí PSVI a chceme mít k dispozici informaci o tom, jak jsou od sebe jednotlivé typy navzájem odvozené.

Zajímavým rysem jsou substituční skupiny. Ty umožňují definovat element, který může být v instanci nahrazen jakýmkoliv jiným elementem patřícím do stejné substituční skupiny. Výhodou tohoto přístupu je pak zejména možnost přidávat nové varianty pro nahrazení elementu v modularizovaných schématech.

Následující dokument ukazuje adresář, kde osoba může být identifikována buď přezdívkou, nebo plným jménem.

```
<adresář>
  <osoba>
    <plnéJméno>
      <křestní>Jan</křestní>
      <příjmení>Novák</příjmení>
    </plnéJméno>
    <email>jan.novak@example.org</email>
  </osoba>
  <osoba>
    <přezdívka>Drsňák</přezdívka>
    <email>jiri.prochazka@example.org</email>
  </osoba>
</adresář>
```

Tohoto efektu můžeme samozřejmě dosáhnout pomocí `choice`, ale substituční skupina umožní v budoucnu snadno přidávat další varianty identifikace osoba.

Příklad 3.17. Schéma se substituční skupinou – `wxs/substituce.xsd`

WXS

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="jméno" abstract="true"/>

  <xs:element name="přezdívka" substitutionGroup="jméno" type="xs:string"/>

  <xs:element name="plnéJméno" substitutionGroup="jméno">
    <xs:complexType>
      <xs:all>
        <xs:element name="křestní" type="xs:string"/>
        <xs:element name="příjmení" type="xs:string"/>
      </xs:all>
    </xs:complexType>
  </xs:element>

  <xs:element name="osoba">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="jméno"/>
        <xs:element name="email" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="adresář">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element ref="osoba"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Vidíme, že element `jméno`, který slouží jako začátek substituční skupiny je definován jako abstraktní, aby jej nešlo použít přímo v instanci dokumentu.

Substituční skupiny lze využít pro psaní schémat, které umožňují následné snadné přizpůsobení. Pomocí elementu `include` je možné schéma skládat dohromady z několika souborů. V novém schématu tak můžeme načíst již existující a přidávat nové elementy do substituční skupiny.

Příklad 3.18. Přidání elementu do substituční skupiny – `wxs/substituce-include.xsd`

WXS
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:include schemaLocation="substituce.xsd"/>

  <xs:element name="krycíJméno" substitutionGroup="jméno" type="xs:string"/>

</xs:schema>
```

Mezi další objektově inspirované rysy patří možnost zablokovat další dědění nebo substituce od nějakého datového typu. Slouží k tomu atributy `final` a `block`. První blokuje další úpravy ve schématu, druhý v jeho instancích.

3.12 Modularizace schémat

W3C XML schémata nabízejí několik prostředků, které usnadňují modularizaci schémat.

3.12.1 Skupiny elementů

Pokud se nějaká část modelu obsahu opakuje na více místech schématu, můžeme ji definovat pouze jednou pomocí elementu `group`, a pak se na ni opakovaně odkazovat pomocí stejného elementu. Tento postup je v němčem podobný definici komplexního datového typu. Rozdíl je v tom, že skupina elementů může být použita kdekoliv, třeba i uvnitř sekvence elemenů.

Příklad 3.19. Opakované využití skupiny elementů – `wxs/group.xsd`

WXS
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:group name="adresa">
    <xs:sequence>
      <xs:element name="ulice" type="xs:string"/>
      <xs:element name="město" type="xs:string"/>
      <xs:element name="psč" type="xs:string"/>
    </xs:sequence>
  </xs:group>

  <xs:element name="osoba">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="jméno" type="xs:string"/>
        <xs:group ref="adresa"/>
        <xs:element name="zaměstnání">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="firma" type="xs:string"/>
```

```
            <xs:group ref="adresa"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

3.12.2 Skupiny atributů

Podobně jako skupiny elementů fungují i skupiny atributů. Pro jejich definici se však používá element `attributeGroup`.

Příklad 3.20. Opakované využití skupiny atributů – `wxs/attributegroup.xsd`

WXS

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:attributeGroup name="společnéAtributy">
    <xs:attribute name="id" type="xs:ID"/>
    <xs:attribute name="lang" type="xs:language"/>
  </xs:attributeGroup>

  <xs:element name="doc">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attributeGroup ref="společnéAtributy"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="p">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attributeGroup ref="společnéAtributy"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attributeGroup ref="společnéAtributy"/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

3.12.3 Složení schématu z několika souborů

Máme-li jednou velké schéma a chceme jej rozdělit do více menších souborů pro lepší přehlednost, můžeme výsledné schéma složit z několika částí pomocí elementu `include`.

```
<xs:include schemaLocation="cast-schematu.xsd"/>
```

Element `include` má ještě jedno speciální využití. Pokud načítané schéma nemá určen cílový jmenný prostor, převezmou načítané deklalarace cílový jmenný prostor schématu, které obsahuje `include`. Tomuto přístupu se přezdívá „chameleon design“ a nelze jej obecně doporučit.

3.12.4 Načtení a redefinování schématu

Někdy se nám může hodit načtení existujícího schématu a jeho následné drobné úpravy. K tomu slouží element `redefine`. Chová se podobně jako `include`, ale umožňuje změnit libovolné načítané definice jednoduchého a komplexního datového typu a skupiny elementů a atributů. Jsou přitom dovoleny jen změny, které vyhovují podmínkám pro odvozování pomocí restrikce nebo extenze. Nově přidávané elementy se tedy mohou objevit pouze na konci modelu obsahu a zúžené modely obsahy musí být podmnožinou těch původních.

Následující příklad využití `redefine` ukazuje přidání nového elementu do již existujícího komplexního typu. Pro tento způsob rozšiřování schémat je vhodné používat při návrhu „slepého Benátčana“, kdy je pro vše k dispozici datový typ, který lze dále upravovat.

Příklad 3.21. Ukázka přidání podelementu redefinicí – `wxs/redefine-pridani.xsd`

WXS

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:redefine schemaLocation="zamestnanec-benatcan.xsd">

    <xs:complexType name="adresaType">
      <xs:complexContent>
        <xs:extension base="adresaType">
          <xs:sequence>
            <xs:element name="stát" type="xs:string"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

  </xs:redefine>

</xs:schema>
```

Pokud by schéma bylo navrženo s využitím skupin elementů, bylo by možné nový element přidat i na jiné místo, než na samotný konec elementu `adresa`.

Příklad 3.22. Ukázka přidání podelementu redefinicí skupiny elementů – `wxs/redefine-pridani-group.xsd`

WXS

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:redefine schemaLocation="zamestnanec-benatcan-group.xsd">

    <xs:group name="adresaGroup">
      <xs:sequence>
        <xs:element name="stát" type="xs:string"/>
        <xs:group ref="adresaGroup"/>
      </xs:sequence>
```

```
    </xs:group>

  </xs:redefine>

</xs:schema>
```

Na závěr tu máme ještě ukázku využití `redefine` ve spojení se složitějším schématem. Ve schématu pro XHTML upravíme definici elementu `tr` tak, aby dovoloval jen podelement `td`.

Příklad 3.23. Ukázka redefinice – `wxs/redefine.xsd`

WXS

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.w3.org/1999/xhtml"
           xmlns="http://www.w3.org/1999/xhtml">

  <xs:redefine schemaLocation="xhtml11/xhtml11.xsd">

    <xs:group name="tr.content">
      <xs:sequence>
        <xs:choice maxOccurs="unbounded">
          <!-- Odstraníme element th: <xs:element ref="th"/> -->
          <xs:element ref="td"/>
        </xs:choice>
      </xs:sequence>
    </xs:group>

  </xs:redefine>

</xs:schema>
```

Varování

Výše uvedený příklad nepůjde v některých validátorech zpracovat. Je to způsobeno tím, že předefinovávaná skupina není přímo v načítaném schématu (`xhtml11/xhtml11.xsd`), ale je do něj načtena pomocí `include`. Ze specifikace není zcela zřejmé, jak se má v tomto případě postupovat. Konstrukci `redefine` je proto bezpečnější používat pouze v případě, že redefinovaná skupina je ve schématu obsažena přímo.

Nejasnosti v používání `redefine` došly tak daleko, že nová verze schémat 1.1 nedoporučuje tuto konstrukci používat.

3.12.5 Schéma definující elementy v několika jmenných prostorech

Chceme-li definovat schéma dokumentu, který se skládá z elementů v různých jmenných prostorech, využijeme naopak elementu `import`. Ten umožňuje do schématu načíst definice elementů a typů patřících do jiného jmenného prostoru. Následující příklad ukazuje import schématu jazyka XHTML do našeho schématu.

Příklad 3.24. Schéma importují schéma pro XHTML – `wxs/message.xsd`

WXS

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'
           xmlns:html="http://www.w3.org/1999/xhtml">
```

```
  <xs:import namespace="http://www.w3.org/1999/xhtml"
             schemaLocation="xhtml11/xhtml11.xsd"/>

  <xs:element name="message">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="subject" type="xs:string"/>
        <xs:choice>
          <xs:element name="body" type="xs:string"/>
          <xs:element ref="html:body"/>
        </xs:choice>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Příklad 3.25. Zpráva s textovým tělem – `wxs/zprava1.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="message.xsd">
  <from>adam@example.org</from>
  <to>eva@example.org</to>
  <subject>Pokusný email v textu</subject>
  <body>Tělo e-mailu.</body>
</message>
```

Příklad 3.26. Zpráva s XHTML tělem – `wxs/zprava2.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="message.xsd">
  <from>adam@example.org</from>
  <to>eva@example.org</to>
  <subject>Pokusný email v textu</subject>
  <body xmlns="http://www.w3.org/1999/xhtml">
    <h1>E-mail zapsaný v HTML</h1>
    <p>První odstavec.</p>
    <p>Druhý odstavec</p>
    <table>
      <tr><th>A</th><th>B</th></tr>
      <tr><td>2</td><td>7</td></tr>
    </table>
  </body>
</message>
```

Tuto metodu musíme využít vždy, kdy chceme definovat schéma s elementy v několika jmenných prostorech, protože v jednom schématu lze definovat pouze elementy patřící do cílového jmenného prostoru určeného atributem `targetNamespace`.

Při kombinování elementů z několika jmenných prostorů si však můžeme vybrat, jak těsně jednotlivá schémata svážeme dohromady. Předchozí příklad zcela explicitně vyjadřoval, kde se může nějaký element

vyskytovat. Je možný však i volnější přístup. Předpokládejme například, že chceme validovat zprávy SOAP, které přenášejí námi definovaná data.

Příklad 3.27. SOAP zpráva – `wxs/soap-zprava.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://www.w3.org/2003/05/soap-envelope"
          xmlns:p="urn:x-pokus:ws-payload">
  <Header>
    <p:UserId>Pepa</p:UserId>
  </Header>
  <Body>
    <p:GetFoo>
      <p:foo>abc</p:foo>
      <p:bar>def</p:bar>
    </p:GetFoo>
  </Body>
</Envelope>
```

Elementy `Envelope`, `Header` a `Body` jsou přitom definovány ve standardně dostupném schématu pro SOAP (`wxs/soap.xsd`). Protože však uvnitř zprávy SOAP mohou být libovolná uživatelská data, nemůže schéma SOAP předdefinovat konkrétní obsah pro elementy `Header` a `Body`. Místo toho se zde používá konstrukce `any`, která umí validovat jakýkoliv element.

```
<xs:any namespace="##any" processContents="lax"
        minOccurs="0" maxOccurs="unbounded"/>
```

V tomto případě parametry `any` určují, že zastupuje element z libovolného jmenného prostoru (`namespace="##any"`) a že tento element se může validovat vůči dalšímu schématu, je-li takové schéma k dispozici (`processContents="lax"`).

Aby byla validace důkladná, musíme si vytvořit schéma pro naše elementy uvnitř zprávy SOAP.

Příklad 3.28. Schéma pro obsah zprávy SOAP – `wxs/payload.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="urn:x-pokus:ws-payload"
           elementFormDefault="qualified">

  <xs:element name="GetFoo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="foo" type="xs:string"/>
        <xs:element name="bar" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="UserId" type="xs:string"/>

</xs:schema>
```

Nyní je potřeba validátoru sdělit, kde má najít schémata pro validaci. Jednou z možností je využití atributu `schemaLocation`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://www.w3.org/2003/05/soap-envelope"
          xmlns:p="urn:x-pokus:ws-payload"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instamce"
   xsi:schemaLocation="http://www.w3.org/2003/05/soap-envelope soap.xsd
                       urn:x-pokus:ws-payload payload.xsd">
  …
</Envelope>
```

Při větším počtu schémat je však zadávání všech dvojic jmenný prostor a schéma do atributu `schemaLocation` poměrně nepraktické. Můžeme si proto pomoci tím, že vytvoříme jedno schéma, které naimportuje všechna potřebná schémata.

Příklad 3.29. Složení schémat – `wxs/soap-zprava.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:import namespace="http://www.w3.org/2003/05/soap-envelope" schemaLocation="soap.xsd"/>
  <xs:import namespace="urn:x-pokus:ws-payload" schemaLocation="payload.xsd"/>

</xs:schema>
```

Nyní stačí pro validaci použít toto jedno schéma.

Kombinování takto volně svázaných schémat lze ovlivnit pomocí parametrů u `any`. Atribut `processContents` určuje, jak se mají elementy popsané pomocí `any`, validovat. Hodnota `skip` říká, že se takové elementy nebudou vůbec validovat, hodnota `strict` říká, že se musí validovat a konečně hodnota `lax` říká, že se validace provede pouze v případě, že se pro elementy najde odpovídající schéma.

Atribut `namespace` pak určuje v jakém jmenném prostoru elementy mohou být. Jmenný prostor jde zadat buď jeden konkrétní, kromě toho lze použít několik speciálních hodnot.

`##any`	zastupuje libovolný jmenný prostor
`##targetNamespace`	zastupuje cílový jmenný prostor schématu
`##other`	zastupuje jakýkoliv jmenný prostor s výjimkou cílového jmenného prostoru
`##local`	elementy nebudou v žádném jmenném prostoru

3.13 Omezení schémat

Při psaní schémat můžeme poměrně snadno narazit na několik omezení. Některé konstrukce jsou ve schématech zakázané. Někdy to má dobrý důvod, někdy lze jen o umělý relikt, protože v době práce na specifikaci nebyl široce znám efektivní algoritmus pro validaci určitých konstrukcí.

3.13.1 Pravidlo „Unique Particle Attribution“

Unique Particle Attribution (UPA) je postrach každého návrháře schémat. Toto pravidlo říká, že schéma musí být vždy zcela jednoznačné, musí být vždy zcela zřejmé a jednoznačné vůči jaké části schématu se daná část dokumentu XML validuje a musí to být jasné, aniž by se ve validovaném dokumentu XML četly elementy „dopředu“.

Například následující typ pravidlo UPA poručuje, protože obě větve schématu obsažené v kompozitoru `xs:choice` začínají elementem `a` – pokud se však ve validovaném dokumentu takový element vyskytne, není jasné jakou větví validace se vydat aniž bychom znali elementy, které za `a` pokračují.

WXS
```
<xs:complexType name="typ1">
  <xs:choice>
    <xs:sequence>
      <xs:element name="a"/>
      <xs:element name="b"/>
    </xs:sequence>
    <xs:sequence>
      <xs:element name="a"/>
      <xs:element name="c"/>
    </xs:sequence>
  </xs:choice>
</xs:complexType>
```

V tomto případě je řešení jednoduché – element `a` můžeme vytknout před výběr mezi elementy `b` a `c`. Tento postup však není možné použít vždy.

WXS
```
<xs:complexType name="typ2">
  <xs:sequence>
    <xs:element name="a"/>
    <xs:choice>
      <xs:element name="b"/>
      <xs:element name="c"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

3.13.2 Pravidlo konzistentní deklarace typu elementu

Pokud je v modelu obsahu na jedné úrovni několikrát deklaraván element stejného jména, musí mít vždy stejný typ. Následující definice typu je tak špatná.

WXS
```
<xs:complexType name="typ3">
  <xs:sequence>
    <xs:element name="a" type="xs:string"/>
    <xs:element name="a" type="xs:integer"/>
  </xs:sequence>
</xs:complexType>
```

3.13.3 Pravidla při použití kompozitoru `xs:all`

Kompozitor `xs:all` nelze kombinovat s jinými kompozitory. Navíc v něm obsažené deklarace elementů nemohou mít větší počet opakování než jedna. Praktická použitelnost konstrukce je tak velmi omezená.

3.14 Dokumentování schématu

XML schéma umožňuje k libovolné jeho části připojit dokumentaci. Ta může obsahovat buď prostý text, nebo můžeme použít v odpovídajícím jmenném prostoru nějaký speciální značkovací jazyk jako XHTML nebo DocBook.

WXS
```
<xs:element name="zamestnanec">
  <xs:annotation>
    <xs:documentation>Element slouží pro uchování důležitých
```

```
      údajů o zaměstnanci.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="jmeno" type="xs:string"/>
      <xs:element name="prijmeni" type="xs:string"/>
      <xs:element name="plat" type="xs:decimal"/>
      <xs:element name="narozen" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

WXS

```
<xs:simpleType name="DruhDokumentuType">
  <xs:annotation>
    <xs:documentation xmlns="http://www.w3.org/1999/xhtml">
      <p>Druh účetního dokumentu.</p>
      <p>Podporovány jsou následující hodnoty:</p>
      <table>
        <tr>
          <td>1</td>
          <td>faktura</td>
        </tr>
        <tr>
          <td>2</td>
          <td>dobropis</td>
        </tr>
        <tr>
          <td>3</td>
          <td>vrubopis</td>
        </tr>
      </table>
      <p>Více podrobností <a href="http://example.com/ciselnik">v číselníku MF</a>.</p>
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:integer">
    <xs:enumeration value="1"/>
    <xs:enumeration value="2"/>
    <xs:enumeration value="3"/>
  </xs:restriction>
</xs:simpleType>
```

Existují nástroje, které jsou pak schopné na základě schématu a v něm obsažených komentářů vygenerovat přehlednou dokumentaci. Jedním z těchto nástrojů je xs3p dostupné na adrese http://sourceforge.net/projects/xs3p/. Jedná se o XSLT styl, který umí schéma převést do webové stránky.

Vygenerování dokumentace tak spočívá v prostém aplikování XSLT stylu na schéma:

```
saxon -o faktura.html faktura.xsd ../tools/xs3p.xsl
```

Analogickou a většinou ještě komfortnější funcionalitu mívají přímo do sebe zabudované i editory XML a schémat.

Podobným způsobem lze do elementu `appInfo` přidat i různé aplikačně závislé informace. Třeba doplňkovou validaci pomocí Schematronu.

3.15 Novinky v XML Schema 1.1

Na jaře 2012 byla vydána nová verze XML Schema 1.1, která se snaží odstranit některá omezení a nedostatky předchozích verzí. Bohužel praktické nasazení bude pomalé. V současné době novou verzi podporují pouze dva validátory Saxon-EE a Xerces-J. Kromě uzavřených scénářů tak není vhodné schémata ve verzi 1.1 vytvářet, protože většina nástrojů je nebude schopna použít. I přesto se však alespoň stručně podíváme na některé nejdůležitější novinky.

3.15.1 Méně omezení pro model obsahu

Některá víceméně umělá omezení byla odstraněna, např.:

- UPA omezení se nyní nevztahují na nejednoznačnost mezi elementem a `xs:any`;
- psát rozšiřitelná schémata je nyní jednoduší, protože lze vytvářet otevřené modely obsahu, kde není nutné explicitně uvádět `xs:any`;
- uvnitř `xs:all` lze používat větší počet opakování než jedna.

3.15.2 Podmíněné přiřazení typu

V praxi se poměrně často setkáme se situacemi, kdy model obsahu se liší na základě hodnoty uvedené v nějakém atributu. XML schémata to uměla řešit jen velmi nešikovně pomocí `xsi:type`. V nové verzi to lze elegantněji vyřešit podmíněným přiřazením typu na základě výsledku výrazu XPath.

WXS
```
<xs:element name="adresa" type="adresaType">
  <xs:alternative test="@stat='usa'" type="americkaAdresaType"/>
  <xs:alternative test="@stat='eu'" type="evropskaAdresaType"/>
  <xs:alternative type="xs:error"/>
</xs:element>
```

V našem příkladě by přitom oba dva typy `americkaAdresaType` a `evropskaAdresaType` musely být odvozeny od typu `adresaType`.

3.15.3 Nové datové typy

XML Schema 1.1 přidává podporu pro nové datové typy `xs:dayTimeDuration` a `xs:yearMonthDuration`, které jsou používány v XQuery a v XSLT 2.0.

Nově je také dovoleno, aby jednotlivé implementace nabízely své vlastní primitivní datové typy nad rámec standardu.

3.15.4 Validační podmínky

Jedná se o asi nejužitečnější rozšíření, které je inspirováno Schematronem. Každý typ může mít k sobě připojené podmínky zapsané v jazyce XPath 2.0. Ty se nevyhodnocují proti celému dokumentu, ale jen vůči podstromu odpovídajícímu datovému typu. Pokud některá z podmínek vrátí chybu nebo false, dokument není považován za validní.

WXS
```
<xs:element name="faktura">
  <xs:complexType>
    <xs:sequence>
      …
    </xs:sequence>
```

```
    <xs:attribute name="vystaveni" type="xs:date" use="required"/>
    <xs:attribute name="splatnost" type="xs:date" use="required"/>
    <xs:assert test="@splatnost ge @vystaveni"/>
  </xs:complexType>
</xs:element>
```

Podstatnou nevýhodou oproti Schematronu je nemožnost definice vlastního chybového hlášení.

Kapitola 4. Relax NG

Pro rychlé vpravení do problematiky si rovnou ukážeme jednoduchý příklad Relax NG schématu a dokumentu XML, který mu vyhovuje. Dejme tomu, že chceme vytvořit schéma popisující dokumenty XML vhodné pro přenos informace o jednom zaměstnanci. U zaměstnance nás přitom bude zajímat jeho identifikační číslo, jméno, příjmení, plat a datum narození. Tyto informace můžeme v XML zachytit následujícím způsobem:

```
<zamestnanec id="101">
  <jmeno>Jan</jmeno>
  <prijmeni>Novák</prijmeni>
  <plat>25000</plat>
  <narozen>1965-12-24</narozen>
</zamestnanec>
```

Odpovídající schéma tedy musí definovat, že dokument musí obsahovat element `zamestnanec`, který obsahuje atribut `id` a čtyři podelementy `jmeno`, `prijmeni`, `plat` a `narozen`. Tyto požadavky může zachytit následující jednoduché schéma.

RNG
```
<element xmlns="http://relaxng.org/ns/structure/1.0"
         name="zamestnanec">
  <attribute name="id">
    <text/>
  </attribute>
  <element name="jmeno">
    <text/>
  </element>
  <element name="prijmeni">
    <text/>
  </element>
  <element name="plat">
    <text/>
  </element>
  <element name="narozen">
    <text/>
  </element>
</element>
```

Vidíme, že celé schéma je dokument XML, který používá speciální elementy. Všechny tyto elementy musí patřit do jmenného prostoru `http://relaxng.org/ns/structure/1.0`. Schéma tak můžeme atlernativně zapsat i s využitím prefixů.

RNG
```
<rng:element xmlns:rng="http://relaxng.org/ns/structure/1.0"
         name="zamestnanec">
  <rng:attribute name="id">
    <rng:text/>
  </rng:attribute>
  <rng:element name="jmeno">
    <rng:text/>
  </rng:element>
  <rng:element name="prijmeni">
    <rng:text/>
  </rng:element>
  <rng:element name="plat">
    <rng:text/>
  </rng:element>
```

```
  <rng:element name="narozen">
    <rng:text/>
  </rng:element>
</rng:element>
```

Zatímco WXS jsou založena na datových typech, je Relax NG založeno na vzorech. Celé schéma je vzorem dokumentu. Vzor se přitom skládá ze vzorů pro elementy, atributy a textové uzly. Tyto vzory pak mohou být dále kombinovány do uspořádaných i neuspořádaných skupin, mohou být volitelné a může u nich být určen počet opakování. Tento jednoduchý princip umožňuje velmi jednoduše vyjádřit i složité struktury v dokumentu a navíc je založen na solidním matematickém základu alejových gramatik (hedge grammars).

Kouzelnou vlastností Relax NG je kompaktní syntaxe. Ta umožňuje zapsat schéma v textové syntaxi, která je mnohem úspornější než ta založená na XML.

RNC
```
element zamestnanec {
  attribute id { text },
  element jmeno { text },
  element prijmeni { text },
  element plat { text },
  element narozen { text }
}
```

V dalším textu budeme většinu ukázek zapisovat paralelně v obou syntaxích. Záleží na vás, kterou z nich si pak vyberete pro vaše použití. Někomu více vyhovuje upovídanější syntaxe XML, někomu zase spíše ta kompaktní textová. Je však úplně jedno jakou syntaxi zvolíte, protože jsou mezi sebou navzájem převoditelné.

4.1 Základní vzory

Jak jsme si již řekli, celé schéma v Relax NG se skládá ze vzorů. Podívejme se proto na základní vzory a jejich použití.

4.1.1 Textový obsah – `text`

Tomuto vzoru vyhoví jakýkoliv text (i prázdný), používá se pro definici obsahu elementů, které už neobsahují další podelementy, nebo pro definici obsahu atributů.

RNG
```
<text/>
```

RNC
```
text
```

4.1.2 Atribut – `attribute`

Vzoru pro atribut vyhovují atributy uvedené v dokumentu XML. Jméno atributu se určuje pomocí atributu `name` a obsah atributu pak obsahem vzoru `attribute`. Atributy nejde dále strukturovat, proto se pro ně nejčastěji používá vzor `text`, ale dále si ukážeme, jak lze použít i jiné datové typy.

RNG
```
<attribute name="měna">
  <text/>
</attribute>
```

Protože atributy vždy obsahují nějaký text, může se při jejich definici vynechat vzor `text`. Předchozí příklad jde tedy zapsat také jako

```
<attribute name="měna"/>
```
RNG

V kompaktní syntaxi takové vynechání definice typu atributu není možné.

```
attribute měna { text }
```
RNC

4.1.3 Element – `element`

Vzoru pro element samozřejmě vyhovují elementy. Podobně jako u atributu musíme pomocí atributu `name` určit jméno elementu. Obsah elementu je pak určen dalšími vzory, které se píší dovnitř definice elementu. Například element, který umožňuje uložení jména, můžeme deklarovat jako:

```
<element name="jméno">
  <text/>
</element>
```
RNG

```
element jméno { text }
```
RNC

Elementy mohou obsahovat atributy. Do vzoru pro elementy proto můžeme přidávat více vzorů, včetně vzorů pro atributy. Např. zápis ceny doplněné o kód měny:

```
<cena měna="USD">29.95</cena>
```

Můžeme ve schématu definovat jako:

```
<element name="cena">
  <attribute name="měna"/>
  <text/>
</element>
```
RNG

```
element cena {
  attribute měna { text },
  text
}
```
RNC

V definici elementu se může vzor pro atributy vyskytovat kdekoliv, nemusí být jen na začátku. Následující schéma je tak totožné s předchozím příkladem.

```
<element name="cena">
  <text/>
  <attribute name="měna"/>
</element>
```
RNG

```
element cena {
  text,
  attribute měna { text }
}
```
RNC

Element může samozřejmě obsahovat podelementy. Ve schématu to zapíšeme tak, že do definice elementu vložíme vzory pro podelementy. Pro dokument:

```
<osoba narozen="1.1.1970">
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
  <email>jan.novak@example.com</email>
</osoba>
```

tak můžeme použít následující schéma

RNG
```
<element name="osoba">
  <element name="jméno">
    <text/>
  </element>
  <element name="příjmení">
    <text/>
  </element>
  <element name="email">
    <text/>
  </element>
  <attribute name="narozen"/>
</element>
```

RNC
```
element osoba {
  element jméno { text },
  element příjmení { text },
  element email { text },
  attribute narozen { text }
}
```

4.1.4 Volitelný vzor – `optional`

V praxi se často setkáme s požadavkem, aby nějaký atribut, element či celá skupina elementů byla nepovinná. V dokumentu se tedy objevit může, ale nemusí. RELAX NG tento požadavek řeší velmi elegantně. Jakýkoliv jiný vzor lze obalit vzorem `optional`, který říká, že vzor v něm obsažený se nemusí v dokument objevit.

Chceme-li, aby v předchozím příkladě bylo možné vynechat e-mailovou adresu nebo datum narození, stačí vzory pro odpovídající elementy a atributy zabalit do `optional`.

RNG
```
<element name="osoba">
  <element name="jméno">
    <text/>
  </element>
  <element name="příjmení">
    <text/>
  </element>
  <optional>
    <element name="email">
      <text/>
    </element>
  </optional>
  <optional>
    <attribute name="narozen"/>
  </optional>
</element>
```

V kompaktní syntaxi se volitelnost vyjadřuje znakem „?“ zapsaným za konec vzoru. Syntaxe tak připomíná DTD nebo regulární výrazy.

RNC
```
element osoba {
  element jméno { text },
  element příjmení { text },
  element email { text }?,
  attribute narozen { text }?
}
```

Tomuto schématu pak kromě původního schématu vyhoví i následující dokumenty:

```
<osoba>
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
</osoba>

<osoba narozen="1.1.1970">
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
</osoba>

<osoba>
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
  <email>jan.novak@example.com</email>
</osoba>
```

Volitelný vzor se vždy chápe jako jeden celek. Následující schéma má proto úplně jiný význam, i když to na první pohled nemusí být patrné.

RNG
```
<element name="osoba">
  <element name="jméno">
    <text/>
  </element>
  <element name="příjmení">
    <text/>
  </element>
  <optional>
    <element name="email">
      <text/>
    </element>
    <attribute name="narozen"/>
  </optional>
</element>
```

RNC
```
element osoba {
  element jméno { text },
  element příjmení { text },
  (element email { text },
   attribute narozen { text })?
}
```

Volitelný je nyní vzor, který obsahuje atribut `narozen` a element `email`. Oba dva proto musí zároveň buď chybět, nebo být přítomné. Bude-li uveden jen atribut `narozen`, nebo element `email`, nebude dokument validní. Následující dva dokumenty jsou příklady dokumentů, které schématu nevyhovují.

```
<osoba narozen="1.1.1970">
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
</osoba>

<osoba>
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
  <email>jan.novak@example.com</email>
</osoba>
```

4.1.5 Opakovaný vzor – `oneOrMore`

Vzor `oneOrMore` říká, že jeho obsah se může v dokumentu opakovat vícekrát. Můžeme tak například snadno definovat, že jedna osoba může mít více emailových adres.

```
<osoba narozen="1.1.1970">
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
  <email>jan.novak@example.com</email>
  <email>jenda@example.org</email>
</osoba>
```

V odpovídajícím schématu je definice elementu `email` obalena vzorem `oneOrMore`.

RNG
```
<element name="osoba">
  <element name="jméno">
    <text/>
  </element>
  <element name="příjmení">
    <text/>
  </element>
  <oneOrMore>
    <element name="email">
      <text/>
    </element>
  </oneOrMore>
  <attribute name="narozen"/>
</element>
```

V kompaktní syntaxi se opakovaní vzoru zapisuje pomocí znaku „+“:

RNC
```
element osoba {
  element jméno { text },
  element příjmení { text },
  element email { text }+,
  attribute narozen { text }
}
```

4.1.6 Libovolný počet opakování vzoru – `zeroOrMore`

Chceme-li, aby se určitá část dokumentu mohla libovolněkrát opakovat nebo úplně chybět, můžeme s výhodou použít vzor `zeroOrMore`. Chceme-li umožnit zadat libovolný počet emailových adres, případně adresu vynechat, můžeme použít následující schéma.

RNG
```
<element name="osoba">
  <element name="jméno">
    <text/>
  </element>
  <element name="příjmení">
    <text/>
  </element>
  <zeroOrMore>
    <element name="email">
      <text/>
    </element>
  </zeroOrMore>
```

```
  <attribute name="narozen"/>
</element>
```

RNC
```
element osoba {
  element jméno { text },
  element příjmení { text },
  element email { text }*,
  attribute narozen { text }
}
```

Tomuto schématu pak vyhoví všechny následující varianty dokumentu.

```
<osoba narozen="1.1.1970">
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
</osoba>

<osoba narozen="1.1.1970">
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
  <email>jenda@example.org</email>
</osoba>

<osoba narozen="1.1.1970">
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
  <email>jan.novak@example.com</email>
  <email>jenda@example.org</email>
</osoba>
```

4.1.7 Přesné určení počtu opakování vzoru

Vzory `optional`, `oneOrMore` a `zeroOrMore` slouží k určení počtu opakování vzoru, který obalují. Pokud žádný z těchto vzorů nepoužijeme, je počet opakování jedna.

Tabulka 4.1. Určení počtu opakování vzoru

Počet opakování	RNG	RNC
1	–	–
0 nebo 1	`optional`	`?`
1 a více	`oneOrMore`	`+`
0, 1 nebo více	`zeroOrMore`	`*`

RELAX NG bohužel nenabízí prostředky pro snadný zápis situací, kdy se nějaký vzor má opakovat a my chceme přesně určit jeho počet opakování. Například chceme říci, že u jedné osoby chceme mít uvedena nejméně dvě a nejvíce pět telefonních čísel. V WXS toho můžeme snadno dosáhnout pomocí atributů `minOccurs` a `maxOccurs`. RELAX NG podobnou možnost nenabízí, a proto tento požadavek musíme vyřešit tak, že dvě telefonní čísla necháme povinná a další tři zadáme jako volitelná.

RNG
```
<element name="osoba">
  <element name="jméno">
    <text/>
  </element>
  <element name="telefon">
    <text/>
  </element>
```

```
  <element name="telefon">
    <text/>
  </element>
  <optional>
    <element name="telefon">
      <text/>
    </element>
  </optional>
  <optional>
    <element name="telefon">
      <text/>
    </element>
  </optional>
  <optional>
    <element name="telefon">
      <text/>
    </element>
  </optional>
</element>
```

RNC

```
element osoba {
  element jméno { text },
  element telefon { text },
  element telefon { text },
  element telefon { text }?,
  element telefon { text }?,
  element telefon { text }?
}
```

Zvláště pro větší počty opakování složitějších vzorů je tento postup poměrně nepřehledný. Situaci si lze zjednodušit použitím pojmenovaných vzorů. V praxi si naštěstí velmi často vystačíme se vzory pro opakování, které nabízí RELAX NG. Další možností je pak použít vzor RELAX NG, který je méně restriktivní než náš požadavek a dodatečné požadavky na počet výskytů elementu vyjádřit pomocí doplňkového schématu ve Schematronu.

4.2 Pokročilé vzory

4.2.1 Seskupování vzorů – `group`

V následujícím schématu očekáváme, že uvnitř elementu `osoba` se budou vyskytovat podelementy `jméno`, `příjmení` a `email` v pořadí, v jakém jsou uvedeny ve schématu.

RNG

```
<element name="osoba">
  <element name="jméno">
    <text/>
  </element>
  <element name="příjmení">
    <text/>
  </element>
  <element name="email">
    <text/>
  </element>
  <attribute name="narozen"/>
</element>
```

Toto chování je zajištěno tím, že vzory uvnitř definice elementu se obalí vzorem `group`. Takže naše schéma je ekvivalentní následujícímu schématu:

RNG
```
<element name="osoba">
  <group>
    <element name="jméno">
      <text/>
    </element>
    <element name="příjmení">
      <text/>
    </element>
    <element name="email">
      <text/>
    </element>
    <attribute name="narozen"/>
  </group>
</element>
```

Vzory uvedené uvnitř `group` se musí v dokumentu vyskytovat ve stejném pořadí, v jakém jsou definovány ve schématu. Toto omezení se nevztahuje na atributy, které mohou být ve vzoru kdekoliv, ale testují se samozřejmě u počátečního tagu daného elementu.

V kompaktní syntaxi místo `group` oddělíme jednotlivé vzory čárkou.

Vzor `group` se explicitně používá zejména tehdy, když potřebujeme několik vzorů seskupit do jednoho pro další použití ve vzorech jako je `choice`.

4.2.2 Výběr vzoru – `choice`

Vzor `choice` říká, že v dokumentu se může vyskytovat právě jeden z vzorů uvedených uvnitř `choice`.

Dejme tomu, že chceme, aby se u osoby musel kromě jména zadat jeden z identifikátorů rodné číslo, číslo pasu nebo číslo sociálního pojištění. Tj. aby všechny tři následující dokumenty byly validní:

```
<osoba>
  <jméno>Pepa Tuzemec</jméno>
  <RČ>681203/0123</RČ>
</osoba>

<osoba>
  <jméno>Pepa Cizinec</jméno>
  <pas>1234567</pas>
</osoba>

<osoba>
  <jméno>Pepa Rozvědčík</jméno>
  <SSN>987654321</SSN>
</osoba>
```

V tomto případě můžeme vyjádřit požadavek na výběr jednoho z elementů právě pomocí `choice`. V kompaktní textové syntaxi se pak pro oddělení variant používá znak „|“.

RNG
```
<element name="osoba">
  <element name="jméno">
    <text/>
  </element>
  <choice>
```

```
    <element name="RČ">
      <text/>
    </element>
    <element name="pas">
      <text/>
    </element>
    <element name="SSN">
      <text/>
    </element>
  </choice>
</element>
```

RNC

```
element osoba {
  element jméno { text },
  (element RČ { text }
   | element pas { text }
   | element SSN { text })
}
```

Má-li být součástí jedné z variant více elementů nebo atributů, musíme je obalit pomocí `group`. Následující schéma umožňuje u osoby zadat buď její jméno, nebo kombinaci křestního jména a příjmení. Zároveň je potřeba zadat e-mailovou adresu, která může být zadána jako element nebo atribut. Zde se s výhodou využívá toho, že vzor pro atribut může být uveden kdekoliv v definici elementu.

RNG

```
<element name="osoba">
  <choice>
    <element name="jméno">
      <text/>
    </element>
    <group>
      <element name="křestní">
        <text/>
      </element>
      <element name="příjmení">
        <text/>
      </element>
    </group>
  </choice>
  <choice>
    <element name="email">
      <text/>
    </element>
    <attribute name="email"/>
  </choice>
</element>
```

RNC

```
element osoba {
  (element jméno { text }
   | (element křestní { text },
      element příjmení { text })),
  (element email { text }
   | attribute email { text })
}
```

Takovému schématu pak vyhoví například následující dokumenty:

```
<osoba>
  <jméno>Pepa</jméno>
```

```
  <email>pepan@example.com</email>
</osoba>

<osoba email="pepan@example.com">
  <jméno>Pepa</jméno>
</osoba>

<osoba email="pepan@example.com">
  <křestní>Pepa</křestní>
  <příjmení>Novák</příjmení>
</osoba>

<osoba>
  <křestní>Pepa</křestní>
  <příjmení>Novák</příjmení>
  <email>pepan@example.com</email>
</osoba>
```

Naopak nevyhoví následující dokumenty, v nichž se vyskytuje nedovolená duplicita.

```
<osoba email="pepan@example.com">
  <jméno>Pepa</jméno>
  <email>pepan@example.com</email>
</osoba>

<osoba email="pepan@example.com">
  <jméno>Pepan</jméno>
  <křestní>Pepa</křestní>
  <příjmení>Novák</příjmení>
</osoba>
```

4.2.3 Prolínání vzorů – `interleave`

V praxi se často setkáme s případy, kdy je nám jedno, v jakém pořadí se nějaké elementy vyskytnou. Dejme tomu, že chceme, aby se uvnitř elementu `osoba` mohly v libovolném pořadí vyskytovat elementy `jméno`, `příjmení` a případně `přezdívka`, za kterými bude následovat e-mailová adresa.

RNG

```
<element name="osoba">
  <interleave>
    <element name="jméno">
      <text/>
    </element>
    <element name="příjmení">
      <text/>
    </element>
    <optional>
      <element name="přezdívka">
        <text/>
      </element>
    </optional>
  </interleave>
  <element name="email">
    <text/>
  </element>
</element>
```

V kompaktní syntaxi se místo vzoru `interleave` používá konektor „`&`“.

RNC
```
element osoba {
  (element jméno { text }
   & element příjmení { text }
   & element přezdívka { text }?),
  element email { text }
}
```

Toto schéma pak umožňuje například zápis následujících dokumentů:

```
<osoba>
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
  <přezdívka>Mařena</přezdívka>
  <email>jan@example.com</email>
</osoba>

<osoba>
  <jméno>Jan</jméno>
  <přezdívka>Mařena</přezdívka>
  <příjmení>Novák</příjmení>
  <email>jan@example.com</email>
</osoba>

<osoba>
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
  <email>jan@example.com</email>
</osoba>

<osoba>
  <příjmení>Novák</příjmení>
  <jméno>Jan</jméno>
  <email>jan@example.com</email>
</osoba>

<osoba>
  <přezdívka>Mařena</přezdívka>
  <příjmení>Novák</příjmení>
  <jméno>Jan</jméno>
  <email>jan@example.com</email>
</osoba>
```

Vzor `interleave` je velmi flexibilní a vysoce převyšuje podobné možnosti v WXS nebo DTD. Není proto problém namodelovat dokument, který umožní zadat titul před i za jméno a přitom nebude záležet na pořadí jména a příjmení.

RNG
```
<element name="osoba">
  <optional>
    <element name="titul">
      <text/>
    </element>
  </optional>
  <interleave>
    <element name="jméno">
      <text/>
    </element>
    <element name="příjmení">
      <text/>
```

```
    </element>
  </interleave>
  <optional>
    <element name="titul">
      <text/>
    </element>
  </optional>
</element>
```

RNC

```
element osoba {
  element titul { text }?,
  (element jméno { text }
   & element příjmení { text }),
  element titul { text }?
}
```

Vzor `interleave` toho však umí mnohem více. Jak ostatně napovídá jeho název, umožňuje prolínání elementů ve skupinách. Jednotlivé vzory uvnitř `interleave` se tak mohou prolínat, ale pořadí elementů v těchto vzorech zůstane zachováno. Pojďme si to ukázat na příkladě. Dejme tomu, že si chceme v dokumentu XML uchovávat informace o zákaznících:

```
<zákazník>
  <jméno>Pepa</jméno>
  <ulice>Nádražní 7</ulice>
  <město>Liberec</město>
  <psč>460 00</psč>
  <telefon>800121314</telefon>
  <email>pepa@example.com</email>
</zákazník>
```

Mezi tyto elementy pak budeme chtít libovolným způsobem vkládat poznámky jako element `poznámka`. Například takto:

```
<zákazník>
  <poznámka>Je bohatý, určitě ho přesvědčíme, ať si něco koupí</poznámka>
  <jméno>Pepa</jméno>
  <ulice>Nádražní 7</ulice>
  <město>Liberec</město>
  <psč>460 00</psč>
  <poznámka>To PSČ musíme ještě upřesnit</poznámka>
  <telefon>800121314</telefon>
  <email>pepa@example.com</email>
  <poznámka>Ta e-mailová adresa vypadá nějak divně</poznámka>
</zákazník>
```

S využitím vzoru `interleave` bude výsledné schéma velmi jednoduché.

RNG

```
<element name="zákazník">
  <interleave>
    <group>
      <element name="jméno">
        <text/>
      </element>
      <element name="ulice">
        <text/>
      </element>
      <element name="město">
        <text/>
```

```
      </element>
      <element name="psč">
        <text/>
      </element>
      <element name="telefon">
        <text/>
      </element>
      <element name="email">
        <text/>
      </element>
    </group>
    <zeroOrMore>
      <element name="poznámka">
        <text/>
      </element>
    </zeroOrMore>
  </interleave>
</element>
```

RNC

```
element zákazník {
  ((element jméno { text },
    element ulice { text },
    element město { text },
    element psč { text },
    element telefon { text },
    element email { text })
   & element poznámka { text }*)
}
```

4.2.4 Smíšený obsah – `mixed`

Smíšený obsah je název pro obsah elementu, který může obsahovat jak text, tak další vnořené elementy. Tato struktura je obvyklá například pro odstavce. Ty většinou obsahují text, ale občas je tento text obalen v elementech, které textu přiřazují význam, mění formátování nebo třeba vkládají odkaz.

Například na následující ukázce má element `odstavec` smíšený obsah, protože kromě textu se v něm podle potřeby mohou opakovat elementy `pojem` a `odkaz`.

```
<odstavec>Odstavce typicky obsahují <pojem>smíšený
obsah</pojem>. Text se může střídat
s <odkaz url="http://www.kosek.cz">odkazy</odkaz>
a dalšími <pojem>elementy</pojem>.</odstavec>

<odstavec>Odstavec může obsahovat i jen text.</odstavec>

<odstavec><pojem>Nebo jen element.</pojem></odstavec>
```

V takovém případě stačí použít `interleave` a pomocí něj dovolit prolínání textu s libovolným počtem elementů `pojem` a `odkaz`.

RNG

```
<element name="odstavec">
  <interleave>
    <zeroOrMore>
      <element name="pojem">
        <text/>
      </element>
    </zeroOrMore>
    <zeroOrMore>
```

```
      <element name="odkaz">
        <attribute name="url"/>
        <text/>
      </element>
    </zeroOrMore>
    <text/>
  </interleave>
</element>
```

RNC

```
element odstavec {
  element pojem { text }*
  & element odkaz { attribute url { text }, text }*
  & text
}
```

U vzoru `text` není potřeba nastavovat opakování, protože tento vzor automaticky vyhovuje libovolném počtu textových uzlů včetně nulového.

Protože je smíšený obsah v dokumentově orientovaných schématech používán poměrně často, existuje pro něj v RELAX NG zkrácený zápis. Místo `interleave` můžeme použít vzor `mixed`, ve kterém se už neuvádí vzor pro text. Dostaneme tak o něco kratší a přehlednější schéma.

RNG

```
<element name="odstavec">
  <mixed>
    <zeroOrMore>
      <element name="pojem">
        <text/>
      </element>
    </zeroOrMore>
    <zeroOrMore>
      <element name="odkaz">
        <attribute name="url"/>
        <text/>
      </element>
    </zeroOrMore>
  </mixed>
</element>
```

Nicméně, když toto schéma vyzkoušíme, zjistíme, že vyžaduje, aby se elementy `pojem`, vyskytovaly před elementy `odkaz`. Toto zvláštní chování je dáno tím, že specifikace RELAX NG říká, že pokud vzor `mixed` obsahuje více jak jedno dítě jako svůj obsah, jsou všechny děti obaleny vzorem pro skupinu `group`. Naše schéma tak ve skutečnosti odpovídá následujícímu zápisu.

RNG

```
<element name="odstavec">
  <mixed>
    <group>
      <zeroOrMore>
        <element name="pojem">
          <text/>
        </element>
      </zeroOrMore>
      <zeroOrMore>
        <element name="odkaz">
          <attribute name="url"/>
          <text/>
        </element>
      </zeroOrMore>
    </group>
```

```
  </mixed>
</element>
```

A jak vidíme, práve seskupení vzorů má v případě klasického smíšeného obsahu dost nepříjemné vedlejší efekty. Můžeme však ručně `group` nahradit za `interleave` a dostaneme schéma, které se chová tak, jak potřebujeme.

RNG
```
<element name="odstavec">
  <mixed>
    <interleave>
      <zeroOrMore>
        <element name="pojem">
          <text/>
        </element>
      </zeroOrMore>
      <zeroOrMore>
        <element name="odkaz">
          <attribute name="url"/>
          <text/>
        </element>
      </zeroOrMore>
    </interleave>
  </mixed>
</element>
```

Tomuto zápisu odpovídá následující kompaktní syntaxe.

RNC
```
element odstavec {
  mixed {
    element pojem { text }*
    & element odkaz { attribute url { text }, text }*
  }
}
```

4.2.5 Prázdný element – `empty`

Pokud chceme definovat element, který je prázdný, musíme použít vzor `empty`.

RNG
```
<element name="br">
  <empty/>
</element>
```

RNC
```
element br { empty }
```

Obsahuje-li element i nějaké atributy, nemusíme vzor `empty` používat.

4.3 Datové typy

RELAX NG sám o sobě nijak oslnivou podporu datových typů nenabízí. Nicméně nabízí mechanismus, jak jej používat s libovolnou externí knihovnou datových typů. Nejčastěji se přitom používají datové typy z WXS. Díky tomu můžeme i v RELAX NG kontrolovat, zda element nebo atribut obsahuje hodnotu vyhovující podmínkám nějakého datového typu.

4.3.1 Určení datového typu – `data`

Pro nastavení datového typu pro obsah elementu nebo atributu se používá vzor `data`. Jeho atribut `type` pak určuje datový typ. Identifikátor použité knihovny typů se zadává v atributu `datatypeLibrary`. Knihovna datových typů WXS je identifikována pomocí URI `http://www.w3.org/2001/XMLSchema-datatypes`.

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

Zapisování použité knihovny typů u každého elementu `data` by bylo poměrně pracné. RELAX NG proto dědí definici aktuální knihovny typů na všechny podelementy. V praxi se tak knihovna nadefinuje jen jednou na kořenovém elementu schématu a platí pro celé schéma.

RNG
```
<element name="souřadnice"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <element name="x">
    <data type="double"/>
  </element>
  <element name="y">
    <data type="double"/>
  </element>
</element>
```

V kompaktní syntaxi si můžeme pro knihovnu datových typů definovat prefix a ten pak používat při určení typu:

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

element cena { xs:decimal }
```

Nicméně pro knihovnu datových typů WXS je implicitně předdefinován prefix `xsd`, takže jej deklarovat nemusíme a můžeme jej rovnou použít:

RNC
```
element cena { xsd:decimal }
```

4.3.2 Parametry datových typů – `param`

U každého datového typu je možné určit parametry, které jej dále upřesňují. Při použití datových typů WXS tak můžeme přímo v RELAX NG nastavit většinu integritních omezení, která známe z WXS. Pro nastavení parametrů se používá element `param` s atributem `name`.

Následující schéma demonstruje využití datových typů z WXS s nastavenými integritními omezeními. Jméno zaměstnance musí mít délku mezi 2 a 15 znaků, plat musí být menší než sto tisíc korun, ale vyšší než minimální mzda a nakonec rodné číslo se musí skládat z devíti nebo desíti číslic oddělených lomítkem.

RNG
```
<element name="zaměstnanec" xmlns="http://relaxng.org/ns/structure/1.0"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <element name="jméno">
    <data type="string">
      <param name="minLength">2</param>
      <param name="maxLength">15</param>
    </data>
  </element>
```

```
  <element name="plat">
    <data type="decimal">
      <param name="minInclusive">6700</param>
      <param name="maxExclusive">100000</param>
    </data>
  </element>
  <element name="rč">
    <data type="token">
      <param name="pattern">\d{6}/\d{3,4}</param>
    </data>
  </element>
</element>
```

RNC

```
element zaměstnanec {
  element jméno {
    xsd:string { minLength = "2" maxLength = "15" }
  },
  element plat {
    xsd:decimal { minInclusive = "6700" maxExclusive = "100000" }
  },
  element rč {
    xsd:token { pattern = "\d{6}/\d{3,4}" }
  }
}
```

Všimněte si, že pro rodné číslo používáme datový typ `token`. Jeho obsah se před validací normalizuje tak, že se odříznou bílé znaky na jeho začátku a konci a opakovaný výskyt bílých znaků za sebou uvnitř hodnoty se nahradí jednou mezerou. Validací tak projdou i zápisy jako:

```
<rč>  123456/7890 </rč>
```

Kdybychom místo `token` použili typ `string`, normalizace bílých znaků se nebude provádět, a validací by prošly jen zápisy bez mezer:

```
<rč>123456/7890</rč>
```

Jako parametr můžeme použít všechna integritní omezení s výjimkou `enumeration` a `whiteSpace`.

4.3.3 Výčtové typy

Použití integritního omezení `enumeration` je zakázáno, protože RELAX NG nabízí vlastní prostředky pro definování seznamu hodnot přípustných pro nějaký atribut nebo element. Pro určení přípustné hodnoty se používá vzor `value`. Pomocí vzoru `choice` můžeme více přípustných hodnot zkombinovat do jednoho výčtu.

Následující schéma definuje element `cena`, který obsahuje desetinné číslo a atribut `měna`. Přípustné hodnoty pro tento atribut jsou definovány výčtem.

RNG

```
<element name="cena" xmlns="http://relaxng.org/ns/structure/1.0"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <attribute name="měna">
    <choice>
      <value>CZK</value>
      <value>USD</value>
      <value>EUR</value>
    </choice>
  </attribute>
```

```
  <data type="decimal"/>
</element>
```

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

Zcela obdobně lze výčtový typ použít i pro určení obsahu elementu.

Před porovnáním hodnoty z dokumentu s hodnotami ve výčtu se provede normalizace této hodnoty (stejně jako v případě datového typu `token`). Pomocí atributu `type` můžeme určit i jiný typ. Pak se použijí jeho pravidla pro normalizaci a kromě shody s hodnotou se bude testovat i shoda s datovým typem. Nejčastěji se používá typ `string`, který nenormalizuje bílé znaky a zaručí, že ve validním dokumentu se hodnota z výčtu vyskytla přesně, včetně případných bílých znaků.

Každá hodnota ve výčtu přitom může mít určen svůj vlastní datový typ. U každé hodnoty se tak může provádět jiný způsob normalizace bílých znaků. Jde například vytvořit schéma, kterým projde dokument:

```
<cena měna=" CZK ">67.50</cena>
```

Ale neprojde dokument:

```
<cena měna=" USD ">67.50</cena>
```

Protože pro hodnotu `USD` nastavíme typ `string` a pro hodnotu `CZK` typ `token`.

RNG
```
<element name="cena" xmlns="http://relaxng.org/ns/structure/1.0"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <attribute name="měna">
    <choice>
      <value type="token">CZK</value>
      <value type="string">USD</value>
      <value>EUR</value>
    </choice>
  </attribute>
  <data type="decimal"/>
</element>
```

RNC
```
element cena {
  attribute měna { token "CZK" | string "USD" | "EUR" },
  xsd:decimal
}
```

4.3.4 Vyloučené hodnoty – `except`

Někdy se nám může více hodit určit povolené hodnoty negativně, namísto pozitivně. Tj. místo výčtu povolených hodnot potřebujeme použít výčet zakázaných hodnot. Toho můžeme dosáhnout pomocí vzoru `except`.

Můžeme pak snadno definovat, že v dokumentu nechceme mít osoby, které se prohlašují za fašisty.

RNG
```
<element name="osoba">
  <element name="jméno">
    <text/>
  </element>
  <element name="politickéVyznání">
    <data type="string">
```

```
      <except>
        <value>fašista</value>
      </except>
    </data>
  </element>
</element>
```

RNC
```
element osoba {
  element jméno { text },
  element politickéVyznání { string - ("fašista") }
}
```

Zakázaných hodnot můžeme určit i více, pokud vzor `except` zkombinujeme s výčtovým vzorem `choice`.

RNG
```
<element name="osoba">
  <element name="jméno">
    <text/>
  </element>
  <element name="politickéVyznání">
    <data type="string">
      <except>
        <choice>
          <value>fašista</value>
          <value>komunista</value>
          <value>anarchista</value>
        </choice>
      </except>
    </data>
  </element>
</element>
```

RNC
```
element osoba {
  element jméno { text },
  element politickéVyznání {
    string - (("fašista" | "komunista" | "anarchista"))
  }
}
```

4.3.5 Seznamy – `list`

RELAX NG umožňuje definovat, že obsahem nějakého elementu nebo atributu je seznam složený z několika hodnot oddělených bílými znaky. Je tak možné kontrolovat strukturu, která není explicitně vyjádřená pomocí značkování. Z hlediska XML to sice není úplně nejlepší přístup, ale mnoho jazyků používá strukturovaný obsah elementů pro zkrácení zápisu.

To, že se někde může vyskytovat seznam hodnot, se zapisuje pomocí vzoru `list`. Jeho obsahem je pak vzor, který definuje datové typy nebo hodnoty, které se mohou v seznamu objevit.

Následující schéma tak ukazuje použití seznamu.

RNG
```
<element name="osoba">
  <element name="jméno">
    <text/>
  </element>
  <element name="oblíbenéHudebníStyly">
    <list>
      <oneOrMore>
        <data type="token"/>
```

```
      </oneOrMore>
    </list>
  </element>
</element>
```

RNC
```
element osoba {
  element jméno { text },
  element oblíbenéHudebníStyly { list { token+ } }
}
```

Můžeme pak vytvářet dokumenty jako:

```
<osoba>
  <jméno>Pepa</jméno>
  <oblíbenéHudebníStyly>jazz folk rock</oblíbenéHudebníStyly>
</osoba>
```

Vzor pro seznam může samozřejmě používat další možnosti RELAX NG pro tvorbu složitějších vzorů, takže není problém určit výčtem přípustné hodnoty pro položky seznamu.

RNG
```
<element name="osoba">
  <element name="jméno">
    <text/>
  </element>
  <element name="oblíbenéHudebníStyly">
    <list>
      <oneOrMore>
        <choice>
          <value>jazz</value>
          <value>rock</value>
          <value>folk</value>
          <value>country</value>
          <value>blues</value>
          <value>ska</value>
          <value>klasika</value>
          <value>hiphop</value>
          <value>jungle</value>
          <value>drum'n'bass</value>
        </choice>
      </oneOrMore>
    </list>
  </element>
</element>
```

RNC
```
element osoba {
  element jméno { text },
  element oblíbenéHudebníStyly {
    list { ("jazz" | "rock" | "folk" | "country" | "blues" | "ska"
            | "klasika" | "hiphop" | "jungle" | "drum'n'bass" )+ }
  }
}
```

Položky seznamu mohou mít dokonce různé datové typy. Například můžeme definovat element pro zaznamenání rozměru nábytku:

```
<skříňka rozměry="40 38.5 90 cm"/>
```

Atribut `rozměry` je v tomto případě rozumné definovat jako seznam, jehož první tři hodnoty jsou číslo a čtvrtá je řetězec s jednou z předdefinovaných délkových jednotek.

RNG
```
<element name="skříňka">
  <attribute name="rozměry">
    <list>
      <data type="decimal"/>
      <data type="decimal"/>
      <data type="decimal"/>
      <choice>
        <value>cm</value>
        <value>mm</value>
      </choice>
    </list>
  </attribute>
</element>
```

RNC
```
element skříňka {
  attribute rozměry {
    list { xsd:decimal, xsd:decimal, xsd:decimal, ("cm" | "mm" ) }
  }
}
```

4.3.6 Zabudované typy

RELAX NG nabízí dva zabudované datové typy `string` a `token`, které můžeme používat bez určení knihovny datových typů. Oba typy reprezentují textový řetězec. Typ `token` navíc před validací provádí normalizaci bílých znaků.

4.3.7 Kontextové modelování obsahu

Jsou situace, kdy chceme, aby obsah elementu závisel na nějaké jiné hodnotě v dokumentu – například na hodnotě nějakého atributu. Většina schémových jazyků se s tímto požadavkem neumí vyrovnat, ale pro RELAX NG to není žádný problém. Tím, že vzory lze téměř libovolně kombinovat a mohou obsahovat i výčet (i jednoprvkový) povolených hodnot pro element a atribut, můžeme podobný požadavek velice jednoduše a přirozeně vyjádřit ve schématu.

Dejme tomu, že budeme chtít zaznamenávat informace z nějaké seznamky. Jak to tak bývá, budou nás zajímat jiné údaje u dam a jiné u pánů. Schéma by proto mělo na základě pohlaví vyžadovat jiné elementy u muže a jiné u ženy:

```
<seznamka>
  <osoba pohlaví="muž">
    <jméno>Pepa</jméno>
    <věk>29</věk>
    <okres>Bruntál</okres>
    <auto>false</auto>
    <chata>true</chata>
    <konto>14500</konto>
  </osoba>
  <osoba pohlaví="žena">
    <jméno>Martina</jméno>
    <věk>27</věk>
    <okres>Kladno</okres>
    <míry>90 60 90</míry>
    <vlasy>blondýna</vlasy>
    <oči>modré</oči>
  </osoba>
</seznamka>
```

Chceme přitom samozřejmě zaručit, že u muže nepůjde zadat jeho míry a naopak. Následující schéma ukazuje, jak můžeme naše požadavky zachytit formalizovaně v podobě schématu:

RNG

```
<element name="seznamka" xmlns="http://relaxng.org/ns/structure/1.0"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <oneOrMore>
    <element name="osoba">
      <element name="jméno">
        <text/>
      </element>
      <element name="věk">
        <data type="positiveInteger"/>
      </element>
      <element name="okres">
        <text/>
      </element>
      <choice>
        <group>
          <attribute name="pohlaví">
            <value>muž</value>
          </attribute>
          <element name="auto">
            <data type="boolean"/>
          </element>
          <element name="chata">
            <data type="boolean"/>
          </element>
          <element name="konto">
            <data type="decimal"/>
          </element>
        </group>
        <group>
          <attribute name="pohlaví">
            <value>žena</value>
          </attribute>
          <element name="míry">
            <list>
              <data type="decimal"/>
              <data type="decimal"/>
              <data type="decimal"/>
            </list>
          </element>
          <element name="vlasy">
            <choice>
              <value>blondýna</value>
              <value>bruneta</value>
              <value>zrzka</value>
            </choice>
          </element>
          <element name="oči">
            <data type="string"/>
          </element>
        </group>
      </choice>
    </element>
  </oneOrMore>
</element>
```

RNC
```
element seznamka {
  element osoba {
    element jméno { text },
    element věk { xsd:positiveInteger },
    element okres { text },
    ( (attribute pohlaví { "muž" },
       element auto { xsd:boolean },
       element chata { xsd:boolean },
       element konto { xsd:decimal })
      |
      (attribute pohlaví { "žena" },
       element míry { list { xsd:decimal, xsd:decimal, xsd:decimal } },
       element vlasy { ("blondýna" | "bruneta" | "zrzka") },
       element oči { xsd:string })
    )
  }+
}
```

4.4 Modularizace schématu

RELAX NG nabízí velmi silné možnosti pro ukládání schémat do více souborů a jejich vzájemné kombinování a předefinovávání. Podrobný popis těchto možností naleznete například v [14]. My se podíváme alespoň na základní možnost modularizace v rámci jednoho schématu.

4.4.1 Pojmenované vzory

V praxi se často setkáme s tím, že se na několika místech schématu opakuje stejný vzor. Abychom jej nemuseli opisovat, je možné si pro něj definovat jméno a to pak opakovaně používat místo vzoru.

Předpokládejme, že si chceme u osoby ukládat následující údaje:

```
<osoba>
  <jméno>Jan Novák</jméno>
  <narozen>29.5.1957</narozen>
  <trvaléBydliště>
    <ulice>Korunní 2</ulice>
    <město>Praha 2</město>
    <psč>120 00</psč>
  </trvaléBydliště>
  <korespondenčníAdresa>
    <ulice>W. Piecka 17</ulice>
    <město>Praha 2</město>
    <psč>120 00</psč>
  </korespondenčníAdresa>
</osoba>
```

Vidíme, že obsah elementů `trvaléBydliště` a `korespondenčníAdresa` je zcela shodný a že by bylo zbytečné jejich definici duplikovat. Můžeme si proto pomocí pomocí elementu `define` definovat pojmenovaný vzor a pak jej použít. Při použití definic musíme použít ještě element `start`, kterým určíme, na jakém elementu validace začíná. Aby celé schéma i nadále vyhovovalo syntaxi XML a mělo jeden kořenový element, obalíme jej elementem `grammar`.

RNG
```
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <start>
    <element name="osoba">
      <element name="jméno">
```

```
        <text/>
      </element>
      <element name="narozen">
        <text/>
      </element>
      <element name="trvaléBydliště">
        <ref name="adresa"/>
      </element>
      <element name="korespondenčníAdresa">
        <ref name="adresa"/>
      </element>
    </element>
  </start>

  <define name="adresa">
    <element name="ulice">
      <text/>
    </element>
    <element name="město">
      <text/>
    </element>
    <element name="psč">
      <text/>
    </element>
  </define>
</grammar>
```

RNC

```
start =
  element osoba {
    element jméno { text },
    element narozen { text },
    element trvaléBydliště { adresa },
    element korespondenčníAdresa { adresa }
  }

adresa =
  element ulice { text },
  element město { text },
  element psč { text }
```

Pojmenované vzory nám umožňují zkrátit a zpřehlednit zápis i v dalších případech. Například můžeme výrazně zkrátit naše schéma, které u osoby definovalo výskyt dvou až pěti telefonů.

RNG

```
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <start>
    <element name="osoba">
      <element name="jméno">
        <text/>
      </element>
      <ref name="telefon"/>
      <ref name="telefon"/>
      <optional>
        <ref name="telefon"/>
      </optional>
      <optional>
        <ref name="telefon"/>
      </optional>
      <optional>
```

```
      <ref name="telefon"/>
    </optional>
  </element>
</start>

<define name="telefon">
  <element name="telefon">
    <text/>
  </element>
</define>
</grammar>
```

RNC

```
start =
  element osoba {
    element jméno { text },
    telefon,
    telefon,
    telefon?,
    telefon?,
    telefon?
  }

telefon = element telefon { text }
```

4.4.2 Skládání schémat

Schémata uložená v několika souborech lze skládat pomocí elementu `include`. RELAX NG navíc umožňuje velice jednoduše rozšiřovat existující vzory tím, že se k nim přidají nové vzory pomocí vzoru `choice` nebo `interleave`. Kombinováním těchto přístupů lze vytvářet velice flexibilní, modulární a snadno modifikovatelná schémata. Ukažme si vše na příkladě. Dejme tomu, že máme následující schéma definující jednoduchý adresář.

Příklad 4.1. Schéma jednoduchého adresáře – `rng/adresar.rng`

RNG

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <start>
    <ref name="adresář"/>
  </start>
  <define name="přezdívka">
    <element name="přezdívka">
      <text/>
    </element>
  </define>
  <define name="plnéJméno">
    <element name="plnéJméno">
      <element name="křestní">
        <text/>
      </element>
      <element name="příjmení">
        <text/>
      </element>
    </element>
  </define>
  <define name="jméno">
    <choice>
      <ref name="přezdívka"/>
```

```
      <ref name="plnéJméno"/>
    </choice>
  </define>
  <define name="email">
    <element name="email">
      <text/>
    </element>
  </define>
  <define name="osoba">
    <element name="osoba">
      <ref name="jméno"/>
      <ref name="email"/>
    </element>
  </define>
  <define name="adresář">
    <element name="adresář">
      <oneOrMore>
        <ref name="osoba"/>
      </oneOrMore>
    </element>
  </define>
</grammar>
```

Příklad 4.2. Schéma jednoduchého adresáře – `rng/adresar.rnc`

RNC

```
start = adresář
přezdívka = element přezdívka { text }
plnéJméno =
  element plnéJméno {
    element křestní { text },
    element příjmení { text }
  }
jméno = přezdívka | plnéJméno
email = element email { text }
osoba = element osoba { jméno, email }
adresář = element adresář { osoba+ }
```

Nyní budeme chtít vytvořit nové schéma, které v místě, kde jsou nyní dovoleny elementy `plnéJméno` a `přezdívka`, dovolí zadat i element `krycíJméno`. Všimněte si, že schéma je již dopředu navrženo tak, aby jej šlo snadno rozšiřovat a proto existuje pro každý element odpovídající pojmenovaný vzor – v případě jména to je pojmenovaný vzor `jméno`.

RELAX NG umožňuje při definici pojmenovaného vzoru říci, že se má jeho obsah zkombinovat s již existujícím vzorem stejného jména. Způsob kombinace se přitom určuje pomocí atribut `combine`. Našeho cíle tak dosáhneme velice snadno, stačí načíst již existující schéma a do vzoru pro jméno jako další alternativu přidat `krycíJméno`.

Příklad 4.3. Skládání vzorů – `rng/adresar-rozsireni.rng`

RNG

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <include href="adresar.rng"/>
  <define name="krycíJméno">
    <element name="krycíJméno">
      <text/>
    </element>
  </define>
```

```
  <define name="jméno" combine="choice">
    <ref name="krycíJméno"/>
  </define>
</grammar>
```

V kompaktní syntaxi se `combine` nahrazuje pomocí `|=` a `&=`.

Příklad 4.4. Skládání vzorů – `rng/adresar-rozsireni.rnc`

RNC

```
include "adresar.rnc"
krycíJméno = element krycíJméno { text }
jméno |= krycíJméno
```

Výše popsaný přístup se hodí v případech, kdy chceme existují schéma rozšířit. Existují však situace, kdy jej chceme zúžit nebo změnit. V tomto případě můžeme během načítání externího schématu předefinovat libovolný pojmenovaný vzor. Stačí nové definice uvést uvnitř elementu `include`. Následující příklad upraví stávající schéma adresáře tak, že jako kořenový element se musí uvést element `osoba` a bude se provádět mnohem striktnější kontrola tvaru emailové adresy.

Příklad 4.5. Redefinice vzorů během načítání schématu – `rng/adresar-redefinice.rng`

RNG

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0" datatypeLibrary="http://www.w3.org/2001/▶
XMLSchema-datatypes">
  <include href="adresar.rng">
    <start>
      <ref name="osoba"/>
    </start>
    <define name="email">
      <element name="email">
        <data type="string">
          <param ▶
name="pattern">[&#33;-&#126;-[()&lt;>@;:\\&quot;\[\]]]+@[&#33;-&#126;-[()&lt;>@;:\\&quot;\[\]]]+</▶
param>
        </data>
      </element>
    </define>
  </include>
</grammar>
```

Příklad 4.6. Redefinice vzorů během načítání schématu – `rng/adresar-redefinice.rnc`

RNC

```
include "adresar.rnc" {
  start = osoba
  email =
    element email {
      xsd:string {
        pattern = '[!~-[()<>@;:\\"\[\]]]+@[!~-[()<>@;:\\"\[\]]]+'
      }
    }
}
```

4.5 Jmenné prostory

RELAX NG samozřejmě umožňuje deklarovat elementy a atributy, které patří do nějakého jmenného prostoru. Všechny předchozí ukázky zatím jmenné prostory nepoužívaly a elementy a atributy tak nepatřily do žádného jmenného prostoru.

V deklaraci každého elementu a atributu můžeme použít atribut `ns` a do něj zadat jmenný prostor, do kterého má element nebo atribut patřit. Takže následujícímu schématu:

RNG
```
<element xmlns="http://relaxng.org/ns/structure/1.0"
         name="a" ns="urn:x-kosek:schemas:pokus">
  <element name="b" ns="urn:x-kosek:schemas:pokus">
    <text/>
  </element>
  <element name="c" ns="urn:x-kosek:schemas:pokus">
    <text/>
  </element>
</element>
```

Vyhoví například dokumenty:

```
<a xmlns="urn:x-kosek:schemas:pokus">
  <b>foo</b>
  <c>bar</c>
</a>

<p:a xmlns:p="urn:x-kosek:schemas:pokus">
  <p:b>foo</p:b>
  <p:c>bar</p:c>
</p:a>
```

Protože opakování atributu `ns` u každé definice elementu by bylo velmi pracné a vedlo by k chybám v případě, že na něj zapomeneme, dědí se jmenný prostor z elementů, které jsou ve schématu předky elementu. Toto však platí jen pro elementy, atributy musíme vždy do jmenného prostoru přidat explicitně. Předchozí schéma proto můžeme zkrátit jako:

```
<element xmlns="http://relaxng.org/ns/structure/1.0"
         name="a" ns="urn:x-kosek:schemas:pokus">
  <element name="b">
    <text/>
  </element>
  <element name="c">
    <text/>
  </element>
</element>
```

Definujeme-li dokument, který používá více jmenných prostorů, můžeme si pro jmenné prostory nadeklarovat prefixy a ty pak používat před jménem elementu a atributu. Zápis je tak opět jednodušší než v případě použití atributu `ns`.

Například následujícímu schématu:

RNG
```
<element xmlns="http://relaxng.org/ns/structure/1.0"
         name="art:článek"
         xmlns:art="urn:x-kosek:schemas:clanek"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
  <element name="dc:title">
```

```
    <text/>
  </element>
</element>
```

které je ekvivalentní schématu:

RNG
```
<element xmlns="http://relaxng.org/ns/structure/1.0"
         name="článek"
         ns="urn:x-kosek:schemas:clanek">
  <element name="title" ns="http://purl.org/dc/elements/1.1/">
    <text/>
  </element>
</element>
```

vyhoví například následující dokumenty:

```
<článek xmlns="urn:x-kosek:schemas:clanek"
        xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:title>Ze života hmyzu</dc:title>
</článek>

<art:článek xmlns:art="urn:x-kosek:schemas:clanek"
            xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:title>Ze života hmyzu</dc:title>
</art:článek>

<článek xmlns="urn:x-kosek:schemas:clanek">
  <title xmlns="http://purl.org/dc/elements/1.1/">Ze života hmyzu</title>
</článek>
```

V kompaktní syntaxi se prefix jmenného prostoru definuje pomocí klíčového slova `namespace`. Poslední schéma tak můžeme v kompaktní syntaxi zapsat jako:

RNC
```
namespace art = "urn:x-kosek:schemas:clanek"
namespace dc = "http://purl.org/dc/elements/1.1/"

element art:článek {
  element dc:title { text }
}
```

Jeden jmenný prostor můžeme definovat jako implicitní. Elementy, které pak v názvu nemají prefix, do tohoto jmenného prostoru budou patřit. Předchozí schéma proto můžeme zapsat i jako:

RNC
```
default namespace = "urn:x-kosek:schemas:clanek"
namespace dc = "http://purl.org/dc/elements/1.1/"

element článek {
  element dc:title { text }
}
```

Případně jako:

RNC
```
namespace art = "urn:x-kosek:schemas:clanek"
default namespace = "http://purl.org/dc/elements/1.1/"

element art:článek {
  element title { text }
}
```

Implicitní jmenný prostor je však vhodný zejména pro schémata, kde je většina elementů v jednom jmenném prostoru.

RNC
```
default namespace = "urn:x-kosek:schemas:pokus"

element a {
  element b { text },
  element c { text }
}
```

4.6 Dokumentování schématu

RELAX NG umožňuje k libovolnému elementu schématu přidat atributy patřící do libovolného jmenného prostoru a skoro na každé místo ve schématu můžeme vložit element v jakémkoliv jiném jmenném prostoru než používá RELAX NG. To můžeme využít pro zápis dokumentace. Pomocí XSLT pak můžeme poměrně snadno generovat dokumentaci třeba v podobě hypertextově provázané sady HTML stránek.

Příklad 4.7. Dokumentace RNG schématu zapisovaná v XHTML – `rng/dokumentace-xhtml.rng`

RNG
```
<?xml version="1.0" encoding="UTF-8"?>
<element name="osoba" xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:html="http://www.w3.org/1999/xhtml">
  <optional>
    <html:p>Přehled titulů úvaděných před jménem. Jejich popis najdete
            <html:a href="http://www.stuba.sk/svk1/vztahy_s_ver/spektrum/2003/april2003/▶
tituly.html">na
              stránce SAV</html:a>.</html:p>
    <element name="titul">
      <text/>
    </element>
  </optional>
  <interleave>
    <element name="jméno">
      <text/>
    </element>
    <element name="příjmení">
      <text/>
    </element>
  </interleave>
  <optional>
    <html:p>Titul za jménem</html:p>
    <element name="titul">
      <text/>
    </element>
  </optional>
</element>
```

Protože kompaktní syntaxe není založená na XML, musí se dokumentace vkládaná v XML převést do speciální notace, kde se hranaté závorky používají pro simulaci elementů.

Příklad 4.8. Dokumentace schématu zapsaná v kompaktní syntaxi – `rng/dokumentace-xhtml.rnc`

RNC

```
namespace html = "http://www.w3.org/1999/xhtml"

element osoba {
  [
    html:p [
      "Přehled titulů úvaděných před jménem. Jejich popis najdete \x{a}"
      html:a [
        href =
          "http://www.stuba.sk/svk1/vztahy_s_ver/spektrum/2003/april2003/tituly.html"
        "na stránce SAV"
      ]
      "."
    ]
  ]
  (element titul { text }?),
  (element jméno { text }
   & element příjmení { text }),
  [ html:p [ "Titul za jménem" ] ] (element titul { text }?)
}
```

Vystačíme-li si pro dokumentaci pouze s textovými komentáři, můžeme používat RELAX NG anotace. Zapisují se ve vlastním jmenném prostoru.

Příklad 4.9. Dokumentace RNG schématu

RNG

```
<?xml version="1.0" encoding="UTF-8"?>
<element name="osoba" xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0">
  <optional>
    <a:documentation>Titul před jménem</a:documentation>
    <element name="titul">
      <text/>
    </element>
  </optional>
  <interleave>
    <element name="jméno">
      <text/>
    </element>
    <element name="příjmení">
      <text/>
    </element>
  </interleave>
  <optional>
    <a:documentation>Titul za jménem</a:documentation>
    <element name="titul">
      <text/>
    </element>
  </optional>
</element>
```

Výhodou tohoto přístupu je, že kompaktní syntaxe nabízí jednoduchou syntaxi pro zápis takové dokumentace. Začíná-li text na řádce dvojicí znaků „##“, považuje se za komentář, který je součástí dokumentace.

Příklad 4.10. Dokumentace v RNC schématu

RNC

```
element osoba {

  ## Titul před jménem
  (element titul { text }?),
  (element jméno { text }
   & element příjmení { text }),

  ## Titul za jménem
  (element titul { text }?)
}
```

Kapitola 5. Schematron

Dříve zmíněné schémové jazyky jako DTD, W3C XML Schema a RELAX NG víceméně definují gramatiku pro dokument XML. Schematron je založen na zcela odlišném principu. Pomocí tohoto jazyka jde zapsat tvrzení o přítomnosti nebo absenci určitých vzorů v dokumentu. Validace oproti Schematronu pak vrací seznam tvrzení, který vznikne kontrolou vzorů oproti dokumentu. Pro zápis vzorů se přitom používá dobře známý jazyk XPath. To má dvě velké výhody – máme k dispozici poměrně silné vyjadřovací prostředky XPathu a pro validaci dokumentu nám stačí XSLT procesor, protože schematronové schéma lze převést na XSLT transformaci.

Celé schéma má velmi jednoduchou strukturu. Kořenový element `schema`, který patří do jmenného prostoru `http://purl.oclc.org/dsdl/schematron`[1] stejně tak jako ostatní elementy, obsahuje několik vzorů `pattern`. Před vzory může být ve schématu ještě uveden název schématu (element `title`) a deklarovány prefixy jmenných prostorů pomocí elementu `ns`.

Každý vzor se skládá z jednoho nebo více pravidel `rule`, které mají pomocí atributu `context` určen kontext pro jejich vyhodnocení. Jedná se v podstatě o vzor v jazyce XPath, který ze vstupního dokumentu vybere uzly, a ty se pak chápou jako aktuální uzly pro vyhodnocení XPath výrazů uvnitř pravidla.

Uvnitř pravidla se pak používají elementy `assert` a `report`, který k sobě mají připojen atribut `test` s XPath výrazem. V případě, že tento výraz není splněn (`assert`) nebo je splněn (`report`) je výsledkem validace text uvnitř příslušného elementu `assert`, resp. `report`.

Uvnitř textu validačního hlášení můžeme používat další elementy. Obsah elementu `name` bude nahrazen jménem aktuálního elementu, obsah elementu `value-of` bude nahrazen textovou hodnotou XPath výrazu uvedeného v atributu `select`.

Příklad 5.1. Ukázka schématu ve Schematronu – `sch/zamestnanci.sch`

SCH

```
<?xml version="1.0" encoding="utf-8"?>
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
  <pattern>
    <title>Seznam zaměstnanců je neprázdný a máme na výplaty</title>
    <rule context="zamestnanci">
      <assert test="zamestnanec">V seznamu musí být alespoň
        jeden zaměstnanec</assert>
      <report test="sum(zamestnanec/plat) &gt; 500000">Součet platů nemůže
        být větší než 500.000,-</report>
    </rule>
  </pattern>
  <pattern>
    <title>Podmínky pro zaměstnance</title>
    <rule context="zamestnanec">
      <assert test="jmeno">U zaměstnance musí být zadáno jméno.</assert>
      <assert test="prijmeni">U zaměstnance musí být zadáno příjmení.</assert>
      <assert test="email">U zaměstnance musí být zadán e-mail.</assert>
      <assert test="narozen">U zaměstnance musí být zadáno datum narození.</assert>
      <assert test="@id">U zaměstnance musí být zadáno jeho osobní číslo.</assert>
      <report test="jmeno[2]|prijmeni[2]">Zaměstnanec nemůže mít více
        než jedno jméno.</report>
    </rule>
  </pattern>
```

[1] Předchozí verze Schematronu používaly jmenný prostor `http://www.ascc.net/xml/schematron`. Při použití Schematronu si zjistěte, pro jakou verzi je validátor určen. Mnoho dnes používaných validátorů ještě nebylo rozšířeno o podporu ISO Schematronu.

```
  <pattern>
    <title>Duplicita osobních čísel</title>
    <rule context="zamestnanec">
     <report test="count(../zamestnanec[@id = current()/@id]) &gt; 1">Duplicitní osobní číslo
          <value-of select="@id"/> u elementu <name/>.</report>
    </rule>
  </pattern>
</schema>
```

5.1 Validace pomocí XSLT

Kromě samostatných knihoven a programů, které umějí provádět validaci oproti Schematronu, můžeme k použít libovolný XSLT procesor. Existuje totiž sada XSLT transformací[2], které ze Schematronového schématu vygenerují další transformaci XSLT. Takto vygenerovanou trnasformací pak zpracujeme dokument, který chceme validovat. Výsledkem validace je pak seznam chyb.

Validaci pro naše ukázkové schéma tak můžeme provést v následujících krocích. Nejprve v několika postupných krocích vygenerujeme ze schématu transformaci XSLT (`validuj.xsl`):

```
saxon -o temp1.sch zamestnanci.sch ..\tools\iso_dsdl_include.xsl
saxon -o temp2.sch temp1.sch ..\tools\iso_abstract_expand.xsl
saxon -o validuj.xsl temp2.sch ..\tools\iso_schematron_message.xsl
```

Získanou transformací pak zpracujeme dokument, který chceme zvalidovat:

```
saxon zamestnanci-chyby.xml validuj.xsl
Součet platů nemůže být větší než 500.000,-
Duplicitní osobní číslo 102 u elementu zamestnanec.
Duplicitní osobní číslo 102 u elementu zamestnanec.
```

Pro další zpracování chyb je vhodnější než prostý textový výstup formát SVRL (Schematron Validation Report Language). Pokud jej chceme získat, stačí při generování validační transformace použít v poslední fázi transformaci `iso_svrl_for_xslt1.xsl` místo `iso_schematron_message.xsl`.

Příklad 5.2. Ukázka chybových hlášení ve formátu SVRL

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<svrl:schematron-output xmlns:svrl="http://purl.oclc.org/dsdl/svrl">
   <svrl:active-pattern name="Seznam zaměstnanců je neprázdný a máme na výplaty"/>
   <svrl:fired-rule context="zamestnanci"/>
   <svrl:successful-report test="sum(zamestnanec/plat) &gt; 500000" location="/zamestnanci">
      <svrl:text>Součet platů nemůže být větší než 500.000,-</svrl:text>
   </svrl:successful-report>
   <svrl:active-pattern name="Podmínky pro zaměstnance"/>
   <svrl:fired-rule context="zamestnanec"/>
   <svrl:fired-rule context="zamestnanec"/>
   <svrl:fired-rule context="zamestnanec"/>
   <svrl:active-pattern name="Duplicita osobních čísel"/>
   <svrl:fired-rule context="zamestnanec"/>
   <svrl:fired-rule context="zamestnanec"/>
   <svrl:successful-report test="count(../zamestnanec[@id = current()/@id]) &gt; 1"
      location="/zamestnanci/zamestnanec[2]">
      <svrl:text>Duplicitní osobní číslo 102 u elementu zamestnanec.</svrl:text>
   </svrl:successful-report>
   <svrl:fired-rule context="zamestnanec"/>
```

[2] http://code.google.com/p/schematron/

```
   <svrl:successful-report test="count(../zamestnanec[@id = current()/@id]) &gt; 1"
      location="/zamestnanci/zamestnanec[3]">
      <svrl:text>Duplicitní osobní číslo 102 u elementu zamestnanec.</svrl:text>
   </svrl:successful-report>
</svrl:schematron-output>
```

5.2 Vložení Schematronu do WXS

Protože Schematron umožňuje postihnout i jiné aspekty dokumentu, než klasické schémové jazyky, bývá s nimi často kombinován. Do WXS můžeme schematronová pravidla vkládat pomocí elementu `appInfo`, který je přímo určen pro vkládání uživatelských dat.

Příklad 5.3. Schematron vložený WXS schématu – `sch/zamestnanci-limit-platu.xsd`

WXS

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:sch="http://purl.oclc.org/dsdl/schematron">

  <xs:element name="zamestnanci">
    <xs:annotation>
      <xs:appinfo>
        <sch:pattern>
          <sch:title>Máme dost na vyplacení platů</sch:title>
          <sch:rule context="zamestnanci">
           <sch:report test="sum(zamestnanec/plat) &gt; 50000">Součet platů nemůže být větší ▶
než 50.000,-.
              Současný součet je <sch:value-of select="sum(zamestnanec/plat)"/></sch:report>
          </sch:rule>
        </sch:pattern>
      </xs:appinfo>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="zamestnanec"
                    maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="jmeno" type="xs:string"/>
              <xs:element name="prijmeni" type="xs:string"/>
              <xs:element name="email" type="xs:string"
                          maxOccurs="unbounded"/>
              <xs:element name="plat" type="xs:decimal"
                          minOccurs="0"/>
              <xs:element name="narozen" type="xs:date"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:int"
                          use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Takové schéma je možné validovat buď speciálním validátorem, nebo si z něj pomocí XSLT vyextrahovat samostatné schéma ve Schematronu.

```
saxon -o schema.sch zamestnanci-limit-platu.xsd ../tools/wxs2sch.xsl
```

5.3 Vložení Schematronu do RELAX NG

Schematron lze samozřejmě vkládat i do RELAX NG. Schematronová pravidla můžeme vložit v podstatě na libovolné místo, protože gramatika RELAX NG je v tomto ohledu velmi volná.

Příklad 5.4. Schematron vložený do RELAX NG – `sch/zamestnanci-limit-platu.rng`

RNG

```
<?xml version="1.0" encoding="utf-8"?>
<element xmlns="http://relaxng.org/ns/structure/1.0"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
         xmlns:sch="http://purl.oclc.org/dsdl/schematron"
         name="zamestnanci">
  <sch:pattern>
    <sch:title>Máme dost na vyplacení platů</sch:title>
    <sch:rule context="zamestnanci">
     <sch:report test="sum(zamestnanec/plat) &gt; 50000">Součet platů nemůže být větší než ▶
50.000,-.
        Současný součet je <sch:value-of select="sum(zamestnanec/plat)"/></sch:report>
    </sch:rule>
  </sch:pattern>
  <oneOrMore>
    <element name="zamestnanec">
      <attribute name="id">
        <data type="int"/>
      </attribute>
      <element name="jmeno">
        <data type="string"/>
      </element>
      <element name="prijmeni">
        <data type="string"/>
      </element>
      <oneOrMore>
        <element name="email">
          <data type="string"/>
        </element>
      </oneOrMore>
      <optional>
        <element name="plat">
          <data type="decimal"/>
        </element>
      </optional>
      <element name="narozen">
        <data type="date"/>
      </element>
      <sch:pattern>
        <sch:title>Duplicita osobních čísel</sch:title>
        <sch:rule context="zamestnanec">
         <sch:report test="count(../zamestnanec[@id = current()/@id]) &gt; 1">Duplicitiní ▶
osobní číslo
           <sch:value-of select="@id"/> u elementu <sch:name/>.</sch:report>
        </sch:rule>
      </sch:pattern>
    </element>
  </oneOrMore>
</element>
```

Schematron jde vkládat i do schémat zapsaných v kompaktní syntaxi, i když to už pak nevypadá tak pěkně.

Příklad 5.5. Schematron vložený do RNC – `sch/zamestnanci-limit-platu.rnc`

```
namespace sch = "http://purl.oclc.org/dsdl/schematron"

[
  sch:pattern [
    sch:title [ "Máme dost na vyplacení platů" ]
    sch:rule [
      context = "zamestnanci"
      sch:report [
        test = "sum(zamestnanec/plat) > 50000"
        "Součet platů nemůže být větší než 50.000,-.\x{a}" ~
        "        Současný součet je "
        sch:value-of [ select = "sum(zamestnanec/plat)" ]
      ]
    ]
  ]
]
element zamestnanci {
  element zamestnanec {
    attribute id { xsd:int },
    element jmeno { xsd:string },
    element prijmeni { xsd:string },
    element email { xsd:string }+,
    element plat { xsd:decimal }?,
    element narozen { xsd:date }
    >> sch:pattern [
         sch:title [ "Duplicita osobních čísel" ]
         sch:rule [
           context = "zamestnanec"
           sch:report [
             test = "count(../zamestnanec[@id = current()/@id]) > 1"
             "Duplicitiní osobní číslo\x{a}"
             sch:value-of [ select = "@id" ]
             " u elementu "
             sch:name [ ]
             "."
           ]
         ]
       ]
  }+
}
```

RELAX NG schéma s obsaženými schematronovými pravidly je možné validovat buď speciálním validátorem, nebo si z něj opět můžeme pomocí XSLT vyextrahovat samostatné schéma ve Schematronu.

```
saxon -o schema.sch zamestnanci-limit-platu.rng ../tools/RNG2Schtrn.xsl
```

5.4 Pokročilá validace

Schematron se používá pro validaci těch struktur, které klasické jazyky nezvládnou. Následující schéma například ukazuje, jak jde porovnávat hodnoty v jednom dokumentu nebo kontrolovat data oproti číselníku v externím souboru.

Příklad 5.6. Ukázka síly Schematronu – `sch/objednavka.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<objednávka xmlns="urn:x-eshop:document-schemas:purchase-order">
  <adresa typ="doručovací">
    <jméno>Jan Novák</jméno>
    <ulice>Bělehradská 147</ulice>
    <město>Praha 2</město>
    <psč>120 00</psč>
  </adresa>
  <adresa typ="účtovací">
    <jméno>Petra Nováková</jméno>
    <ulice>Anglická 15</ulice>
    <město>Praha 2</město>
    <psč>120 00</psč>
  </adresa>
  <doprava>DPD</doprava>
  <platba>dobírka</platba>
  <přijata>2004-12-14</přijata>
  <doručení>2004-11-19</doručení>
  <položky>
    <položka kód="2N7-516">
      <název>Sekačka na trávu</název>
      <počet mj="ks">1</počet>
      <cena>2999</cena>
      <popis>http://example.org/sekacka.html</popis>
    </položka>
    <položka kód="Q3Y-116">
      <název>Travní semeno</název>
      <počet mj="kg">2.5</počet>
      <cena>127.50</cena>
    </položka>
  </položky>
  <komentář>O dodávku mám zájem pouze v případě, že se jedná o trávu
            v odrůdě konopí.</komentář>
</objednávka>
```

Příklad 5.7. Ukázka síly Schematronu – `sch/sklad.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<sklad>
  <produkt>
    <kod>2N7-516</kod>
    <nazev>Sekačka</nazev>
  </produkt>
  <produkt>
    <kod>2N7-517</kod>
    <nazev>Kosa</nazev>
  </produkt>
</sklad>
```

Příklad 5.8. Ukázka síly Schematronu – `sch/objednavka.sch`

SCH

```
<?xml version="1.0" encoding="utf-8"?>
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
  <title>Ukázka pokročilé validace objednávky</title>
  <ns uri="urn:x-eshop:document-schemas:purchase-order" prefix="o"/>
  <pattern>
```

```
    <title>Datum doručení je správně</title>
    <rule context="o:objednávka">
      <assert test="number(translate(o:doručení,'-','')) > ▶
number(translate(o:přijata,'-',''))">Datum doručení musí být větší
        než datum přijetí objednávky.</assert>
    </rule>
  </pattern>
  <pattern>
    <title>Produkt je v číselníku</title>
    <rule context="o:položka">
     <assert test="@kód = document('sklad.xml')/sklad/produkt/kod">Objednaný produkt (kód: ▶
<value-of select="@kód"/>) není na skladě.</assert>
    </rule>
  </pattern>
</schema>
```

Standardně Schematron pro zápis podmínek používá XPath 1.0. Nicméně pomocí atributu `queryBinding` můžeme zvolit jiný dotazovací jazyk – nejpoužívanější je asi hodnota `xslt2`, která zastupuje XPath 2.0 obohacený o přídavné funkce definované v rámci XSLT 2.0. Nejzajímavější je pak možnost využít zejména nové funkce podporující přímo typy XML schémat.

Příklad 5.9. Schematron s podmínkami zapsanými v XPath 2.0 – `sch/objednavka-xpath2.sch`

SCH

```
<?xml version="1.0" encoding="utf-8"?>
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
        queryBinding="xslt2">
  <title>Ukázka pokročilé validace objednávky</title>
  <ns uri="urn:x-eshop:document-schemas:purchase-order" prefix="o"/>
  <ns uri="http://www.w3.org/2001/XMLSchema" prefix="xs"/>
  <pattern>
    <title>Datum doručení je správně</title>
    <rule context="o:objednávka">
      <assert test="xs:date(o:doručení) gt xs:date(o:přijata)">Datum doručení musí být větší
        než datum přijetí objednávky.</assert>
     <assert test="xs:date(o:přijata) - xs:date(o:doručení) gt xs:dayTimeDuration('P7D')">Na ▶
doručení
        musí být lhůta alespoň sedmi dnů.</assert>
    </rule>
  </pattern>
</schema>
```

Kapitola 6. NVDL

NVDL (Namespace-based Validation Dispatching Language)[10] je nový schémový jazyk, který zastřešuje všechny stávající schémové jazyky a dovoluje je navzájem kombinovat. Zároveň tím řeší některé problémy a nedostatky klasických schémových jazyků.

6.1 Problémy současných schémových jazyků

Ač je dnes situace na poli schémových jazyků mnohem jednodušší, než na přelomu tisíciletí, kdy jich existovalo a používalo se několik desítek, přesto se nedá říci, že by současné schémové jazyky uspokojivě pokrývaly všechny potřeby uživatelů.

6.1.1 Vícenásobná validace

Jednotlivé schémové jazyky byly navrhovány s různými cíli, a liší se proto i jejich schopnost popsat různá omezení, která musí dokument XML splňovat. Pro důkladnější kontrolu je většinou potřeba použít dvě schémata – jedno založené na gramatice a druhé na pravidlech. V praxi se tak nejčastěji setkáme s kombinací RELAX NG + Schematron nebo W3C XML schéma a Schematron.

Potřebovali bychom proto mechanismus, který by snadno dovolil říci, že dokument se má validovat vůči dvěma či více schématům najednou. Mechanismus by přitom měl být dostatečně univerzální a měl by podporovat libovolnou kombinaci schémových jazyků.[1]

6.1.2 Připojení schématu k dokumentu

Chceme-li při zpracování dokumentu využívat jeho schéma, potřebujeme mechanismus, který určí, jaké schéma se má použít. Dříve bylo poměrně časté schéma určovat přímo uvnitř dokumentu XML. Pro DTD bylo možné využít deklaraci typu dokumentu:

```
<!DOCTYPE book SYSTEM "http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd">
```

XML schémata pak přišla s globálními atributy `schemaLocation` a `noNamespaceSchemaLocation`:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.w3.org/1999/xhtml
                          http://www.w3.org/2002/08/xhtml/xhtml1-transitional.xsd">
```

Existuje i speciální instrukce pro zpracování, kterou lze použít s jakýmkoliv schémovým jazykem:

```
<?xml-model href="xhtml.rnc" type="application/relax-ng-compact-syntax"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  …
</html>
```

Historie však ukazuje, že takovýto přístup není vhodný. Schémové jazyky se neustále vyvíjejí, vylepšují a vznikají nové s ještě silnějšími vyjadřovacími prostředky. Oproti tomu samotné dokumenty XML mají poměrně dlouhou životnost. Je proto velice pravděpodobné, že v průběhu životního cyklu dokumentu budeme chtít změnit schéma, vůči kterému dokument validujeme. Pokud je však odkaz na schéma vložen přímo do dokumentu, musíme modifikovat původní dokument, což je nepříjemné a mnohdy dokonce zcela nežádoucí.

[1] Nespokojíme se tedy s tím, že konkrétně Schematron lze vkládat přímo do schématu v RELAX NG nebo W3C XML schéma.

Dalším problém je bezpečnost při zpracování dokumentů odkazujících na schéma. XML je dnes často používáno pro výměnu dat, kde koncový bod přijímá data, provede jejich validaci a pak je předá aplikaci k vlastnímu zpracování. Pokud se však validace řídí schématem určeným v instanci dokumentu, není problém poslat zprávu, která je nesmyslná, ale odkazuje na vlastní schéma, vůči kterému je validní. Aplikace pak není validací chráněna a ke zpracování dostane data, se kterými nepočítá, což může vést ke vzniku chyb a nekonzistentních stavů.

Ve většině případů je proto vhodné, aby schéma bylo určeno externě nezávisle na dokumentu XML. Jako ideální se ukazuje mapovací tabulka, která na základě jmenného prostoru a případně dalších charakteristik přiřadí dokumentu schéma.

6.1.3 Komponované dokumenty

Komponované dokumenty jsou ty dokumenty, které používají elementy z několika jmenných prostorů najednou. V posledních letech vzrůstá jejich obliba, což je zcela pochopitelné. Do XML se ukládají stále komplexnější informace, a bylo by mrhání časem vymýšlet stejné věci pořád dokola, ale trochu jinak. Pokud tedy v našem typu dokumentu potřebujeme ukládat data, pro která již existuje formát založený XML, použijeme jej.

Nevznikají tak velká a složitá monolitická schémata, ale dokument je složen z menších specializovaných částí. Do jisté míry je možné tento vývoj srovnat s oblibou komponentově orientovaného vývoje software. Jednotlivé specializované jazyky XML lze chápat, jako komponenty, ze kterých se složí výsledný dokument. Programové komponenty podporující zpracování jednotlivých jazyků se pak často použití při zpracování komponovaného dokumentu.

Asi největší propagace se myšlenka komponovaných dokumentů dočkala na webu, byť podpora v prohlížečích zatím není zdaleka ideální. Do jazyka XHTML lze vkládat podle potřeby kusy kódu v jiných specializovaných jazycích a dosáhnout tak potřebné funkcionality. Jaké jazyky má smysl míchat v jednom dokumentu s XHTML? Nejčastěji bývají jako příklad uváděny jazyky SVG a MathML, které slouží pro reprezentaci vektorových obrázků a matematických vzorců. Existuje však mnoho dalších smysluplných kombinací. Do XHTML stránky můžeme pomocí elementů a atributů SMIL nastavit časový průběh zobrazování jednotlivých částí stránky. Pomocí VoiceXML můžeme do stránky vložit formulář pro hlasovou interakci s uživatelem. Běžný formulář můžeme vložit pomocí jazyka XForms, přídavná metadata můžeme vkládat přímo v RDF. Smysluplných možností je opravdu mnoho a záleží jen na naší představivosti (a schopnosti aplikací takové dokumenty zpracovat).

Následující příklad ukazuje komponovaný dokument, kde je do XHTML stránky vložen obrázek v SVG. Všimněme si, že elementy XHTML a SVG jsou odlišeny právě pomocí svých jmenných prostorů.

Příklad 6.1. Ukázka komponovaného dokument XHTML + SVG

```
<?xml version="1.0" encoding="utf-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
  <head>
    <title>Ukázka SVG obrázku</title>
  </head>
  <body>
    <h1>Vektorový obrázek v SVG</h1>
    <svg:svg width="4in" height="3in">
      <svg:g>
        <svg:circle style="fill: yellow; stroke: red" cx="200" cy="200" r="150"/>
        <svg:text x="80" y="200"
          style="font-size: 36px; font-family: Verdana; color: red; fill: red"
          >Dobrou chuť</svg:text>
```

```
      </svg:g>
    </svg:svg>
  </body>
</html>
```

Komponované dokumenty se zcela běžně používají pro tiskové výstupy XML, které jsou definovány pomocí jazyka XSL-FO, do kterého jsou grafika a grafy vložena jako fragmenty SVG. V poslední době je mnoho pozornosti věnováno formátům kancelářských balíků. Oba dva formáty Open Document Format i Office Open XML jsou ve skutečnosti komponované dokumenty. XML se velice často používá pro komunikaci mezi aplikacemi. Webové služby zasílané zprávy vkládají do obálky SOAP – výsledná SOAP zpráva je tak vždy rovněž komponovaný dokument.

Je jasné, že pokud ne již dnes, tak v brzké budoucnosti budou většinu obsahu v XML tvořit právě komponované dokumenty. Je tedy potřeba, aby nástroje umožnily snadné a plnohodnotné zpracování takových dokumentů.

Oba dva jazyky W3C XML Schema a RELAX NG umožňují vytvářet schémata pro dokumenty obsahující elementy z více jmenných prostorů. Čistě teoreticky tak nic nebrání tomu, aby se v těchto jazycích vytvářela schémata pro komponované dokumenty. Výhoda komponovaných dokumentů však spočívá v možnosti opakovaného využití již existujícího. Budeme-li chtít vytvořit schéma pro XHTML+SVG, nebudeme chtít psát celé schéma od začátku, ale vzít již existující samostatná témata pro XHTML a SVG a ta zkombinovat dohromady.

Zkombinování dvou schémat dohromady však není vždy jednoduché. Aby šlo schémata navzájem kombinovat, musí být zapsána ve stejném jazyce. Těžko bychom mohli navzájem kombinovat schéma v RELAX NG se schématem W3C XML Schema. Schémata lze sice mezi jednotlivými jazyky konvertovat, ale konverze není vždy bezztrátová, protože jednotlivé schémové jazyky mají odlišnou vyjadřovací sílu.

I když máme schémata v jednom jazyce, ještě není vyhráno. Aby šlo schémata kombinovat musí být navržena modulárně. Jedině tak, lze jednoduše rozšiřovat již existující definice elementů o elementy z jiných jmenných prostorů. V praxi to vyžaduje, aby schémata v maximálně možné míře využívala pojmenované vzory (RELAX NG) nebo substituční skupiny (W3C XML Schema).

Jsou-li výše uvedené předpoklady splněny a my se dokážeme v cizích schématech zorientovat, je možné sestavit schéma pro komponovaný dokument. Nicméně pro větší počet jazyků než dva či tři se úloha většinou stává velice komplikovanou.

6.2 Základy NVDL

Mnoho z výše uvedených problémů řeší nový schémový (či spíše meta-schémový) jazyk NVDL (Namespace-based Validation Dispatching Language)[10]. Jazyk NVDL je mezinárodní normou ISO/IEC 19757-4 přijatou v roce 2006. Nicméně nevznikl na zelené louce, ale evolucí z předchozích projektů NRL (Namespace Routing Language), MNS (Modular Namespaces), RELAX Namespace a Namespace Switchboard.

NVDL umožňuje na jednom místě určit, vůči jakým schématům se mají validovat elementy z jednotlivých jmenných prostorů a jak mohou být jmenné prostory navzájem kombinovány.

Následující schéma ukazuje, jak je v NVDL jednoduché říci, že XHTML dokumenty se mají validovat najednou vůči dvěma schématům v různých jazycích (W3C XML Schema a Schematron).

Příklad 6.2. Validace jednoho dokumentu vůči více schématům

NVDL

```
<rules xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0">
  <namespace ns="http://www.w3.org/1999/xhtml">
    <validate schema="xhtml1-transitional.xsd"/>
    <validate schema="xhtml.sch"/>
  </namespace>
</rules>
```

NVDL však bylo navrženo zejména pro validaci komponovaných dokumentů. Následující příklad ukazuje, jak jednoduše můžeme říci, vůči jakým schématům se mají validovat elementy z jednotlivých jmenných prostorů (v našem případě se jedná o XHTML a SVG).

Příklad 6.3. Jednoduché schéma pro XHTML+SVG komponované dokumenty

NVDL

```
<rules xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0">
  <namespace ns="http://www.w3.org/1999/xhtml">
    <validate schema="xhtml.xsd"/>
  </namespace>
  <namespace ns="http://www.w3.org/2000/svg">
    <validate schema="svg.rng"/>
  </namespace>
</rules>
```

Navíc je vidět, že schéma pro jednotlivé jazyky může být zapsáno v rozdílných schémových jazycích. Jmenné prostory, pro které schéma neurčíme, jsou při výchozím nastavení zakázané a vyvolají chybu validace.

Proces validace vůči NVDL schématu je proces skládající se z několika postupně navazujících kroků:

1. rozdělení dokumentu do sekcí (nezávislé na NVDL schématu)
2. přiřazení akcí jednotlivým sekcím (vytvoření interpretací)
3. kombinování a filtrování sekcí
4. validace fragmentů vůči dílčím schématům

6.2.1 Rozdělení do sekcí

Před validací se dokument rozdělí do sekcí. Toto dělení je nezávislé na použitém NVDL schématu. Samostatnou sekci vytvoří každý element a jeho potomci, kteří používají jiný jmenný prostor než rodičovský element. Vše ilustruje následující příklad.

Příklad 6.4. Jednoduchý XHTML + SVG dokument

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
  <head>
    <title>Ukázka SVG obrázku</title>
  </head>
  <body>
    <h1>Vektorový obrázek v SVG</h1>
    <svg:svg width="4in" height="3in">
      <svg:g>
        <p>ahoj</p>
        <svg:circle style="fill: yellow; stroke: red" cx="200" cy="200" r="150"/>
```

```
        <svg:text x="80" y="200"
          style="font-size: 36px; font-family: Verdana; color: red; fill: red"
          >Dobrou chuť</svg:text>
      </svg:g>
    </svg:svg>
  </body>
</html>
```

Příklad 6.5. Rozdělení dokumentu do sekcí

```
Sekce 1:
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
  <head>
    <title>Ukázka SVG obrázku</title>
  </head>
  <body>
    <h1>Vektorový obrázek v SVG</h1>
    (…odkaz na sekci 2)
  </body>
</html>

Sekce 2:
<svg:svg width="4in" height="3in">
  <svg:g>
    (… odkaz na sekci 3)
    <svg:circle style="fill: yellow; stroke: red" cx="200" cy="200" r="150"/>
    <svg:text x="80" y="200"
      style="font-size: 36px; font-family: Verdana; color: red; fill: red"
      >Dobrou chuť</svg:text>
  </svg:g>
</svg:svg>

Sekce 3:
<p>ahoj</p>
```

6.2.2 Akce

Po vytvoření sekcí se na základě NVDL schématu ke každé sekci přiřadí akce. Akce může určit schéma, vůči kterému se má sekce validovat. Kromě toho některé akce dovolují před validací spojit několik sekcí zase dohromady.

Akce validate

Nejpoužívanější je akce `validate`. Ta způsobí to, že sekce se validuje vůči určenému schématu. Vše ukazují následující příklady. Předpokládejme následující jednoduché schéma.

Příklad 6.6. Akce validate – `nvdl/xhtml-svg1.nvdl`

```
<?xml version="1.0" encoding="UTF-8"?>
<rules xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0">
  <namespace ns="http://www.w3.org/1999/xhtml">
    <validate schema="../wxs/xhtml11/xhtml11.xsd"/>
  </namespace>
  <namespace ns="http://www.w3.org/2000/svg">
    <validate schema="svg/svg11.rng"/>
```

```
  </namespace>
</rules>
```

Jednotlivé sekce se před validací nijak nekombinují, pouze se samostatně validují vůči odpovídajícím schématům.

```
Sekce 1 --> xhtml.xsd:
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
  <head>
    <title>Ukázka SVG obrázku</title>
  </head>
  <body>
    <h1>Vektorový obrázek v SVG</h1>
    (…odkaz na sekci 2)
  </body>
</html>

Sekce 2 --> svg.rng:
<svg:svg width="4in" height="3in">
  <svg:g>
    (… odkaz na sekci 3)
    <svg:circle style="fill: yellow; stroke: red" cx="200" cy="200" r="150"/>
    <svg:text x="80" y="200"
      style="font-size: 36px; font-family: Verdana; color: red; fill: red"
      >Dobrou chuť</svg:text>
  </svg:g>
</svg:svg>

Sekce 3 --> xhtml.xsd:
<p>ahoj</p>
```

Dalo by se říci, že schéma je nejspíše špatné, protože se element `p` validuje bez ohledu na místo, kde se vyskytuje. Pravděpodobně nebudeme chtít dovolit XHTML element uvnitř SVG obrázku, protože by se obtížně definovalo jeho chování. Následující akce umožňují tyto aspekty detailněji kontrolovat.

Akce attach

Akce `attach` způsobí, že před validací je sekce připojena zpět ke své rodičovské sekci. Pro náš příklad XHTML+SVG dokumentu by proto mnohem lepší bylo následující schéma.

Příklad 6.7. Akce attach – `nvdl/xhtml-svg2.nvdl`

NVDL

```
<?xml version="1.0" encoding="UTF-8"?>
<rules xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0">
  <namespace ns="http://www.w3.org/1999/xhtml">
    <validate schema="../wxs/xhtml11/xhtml11.xsd">
      <mode>
        <namespace ns="http://www.w3.org/2000/svg">
          <validate schema="svg/svg11.rng">
            <mode>
              <anyNamespace>
                <attach/>
              </anyNamespace>
            </mode>
          </validate>
        </namespace>
```

```
      </mode>
    </validate>
  </namespace>
</rules>
```

Toto schéma říká, že dokument musí začínat XHTML sekcí, do které mohou být vnořené sekce v SVG. Všechny cizí elementy uvnitř SVG se před validací připojí zpět na SVG sekci (díky akci `attach`). Vycházíme přitom z předpokladu, že míchat cizí elementy do SVG nemá smysl, a proto je věcí SVG schématu, zda tyto elementy povolí.

Při zpracování dokumentu se jednotlivé sekce tedy nejdříve pomocí `attach` spojí do větších fragmentů a ty se pak validují.

```
Fragment 1 --> xhtml.xsd:
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
  <head>
    <title>Ukázka SVG obrázku</title>
  </head>
  <body>
    <h1>Vektorový obrázek v SVG</h1>
  </body>
</html>

Fragment 2 --> svg.rng:
<svg:svg width="4in" height="3in">
  <svg:g>
    <p>ahoj</p>  <!-- Připojená XHTML sekce -->
    <svg:circle style="fill: yellow; stroke: red" cx="200" cy="200" r="150"/>
    <svg:text x="80" y="200"
      style="font-size: 36px; font-family: Verdana; color: red; fill: red"
      >Dobrou chuť</svg:text>
  </svg:g>
</svg:svg>
```

Akce unwrap

V některých případech chceme, aby se při validaci některé sekce chovaly tak, jako by v dokumentu ani nebyly. Předchozí příklad bychom také mohli chtít validovat z pohledu aplikace, která nerozumí SVG. Pro ní je pak důležité, jak by dokument vypadal, kdyby se vypustily všechny elementy SVG. K tomu lze využít právě akci `unwrap`, jak ukazuje následující NVDL schéma.

Příklad 6.8. Akce unwrap – `nvdl/xhtml-svg3.nvdl`

```
<?xml version="1.0" encoding="UTF-8"?>
<rules xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0" startMode="xhtml">
  <mode name="xhtml">
    <namespace ns="http://www.w3.org/1999/xhtml">
      <validate schema="../wxs/xhtml11/xhtml11.xsd"
                useMode="svg"/>
    </namespace>
  </mode>
  <mode name="svg">
    <namespace ns="http://www.w3.org/1999/xhtml">
      <attach/>
    </namespace>
    <namespace ns="http://www.w3.org/2000/svg">
```

```
      <unwrap/>
    </namespace>
  </mode>
</rules>
```

Dané schéma říká, že dokument musí vždy začínat XHTML sekcí. Jakékoliv na ní napojené sekce se pak zpracovávají v režimu `svg`. Tento režim říká, že SVG elementy se zcela ignorují (`unwrap`) a XHTML elementy se napojí na rodičovskou sekci (`attach`). V praxi se tak validuje původní dokument s odfiltrovanými SVG sekcemi.

```
Fragment 1 --> xhtml.xsd:
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
  <head>
    <title>Ukázka SVG obrázku</title>
  </head>
  <body>
    <h1>Vektorový obrázek v SVG</h1>
    <p>Ahoj</p> <!-- Připojená XHTML sekce -->
  </body>
</html>
```

Kombinování akcí

NVDL umožňuje pro každou sekci provést libovolný počet akcí najednou. Předchozí příklad tak můžeme rozšířit o to, aby se navíc SVG sekce validovaly oproti SVG schématu.

Příklad 6.9. Kombinování akcí – `nvdl/xhtml-svg4.nvdl`

NVDL

```
<?xml version="1.0" encoding="UTF-8"?>
<rules xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0" startMode="xhtml">
  <mode name="xhtml">
    <namespace ns="http://www.w3.org/1999/xhtml">
      <validate schema="../wxs/xhtml11/xhtml11.xsd"
                useMode="svg"/>
    </namespace>
  </mode>
  <mode name="svg">
    <namespace ns="http://www.w3.org/1999/xhtml">
      <attach/>
    </namespace>
    <namespace ns="http://www.w3.org/2000/svg">
      <unwrap/>
      <validate schema="svg/svg11.rng" useMode="xhtml"/>
    </namespace>
  </mode>
</rules>
```

NVDL schéma pak způsobí provedení následujících dvou dílčí validací.

```
Fragment 1 --> xhtml.xsd:
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
  <head>
    <title>Ukázka SVG obrázku</title>
  </head>
  <body>
    <h1>Vektorový obrázek v SVG</h1>
```

```
    <p>ahoj</p>  <!-- Připojená XHTML sekce -->
  </body>
</html>

Fragment 2 --> svg.rng:
<svg:svg width="4in" height="3in">
  <svg:g>
    <svg:circle style="fill: yellow; stroke: red" cx="200" cy="200" r="150"/>
    <svg:text x="80" y="200"
      style="font-size: 36px; font-family: Verdana; color: red; fill: red"
      >Dobrou chuť</svg:text>
  </svg:g>
</svg:svg>
```

Kontext

Zatím jsme viděli, že NVDL umožňuje kombinování elementů řídit pomocí jmenných prostorů. Chceme-li mít nad vzájemných kombinováním dokumentů lepší kontrolu, můžeme použít element `context`. Ten dovoluje specifikovat konkrétní element, uvnitř něhož jsou dovoleny elementy z jiných jmenných prostorů.

Následující příklad ukazuje schéma pro XHTML+RDF, kdy RDF metadata se mohou vyskytovat pouze v záhlaví dokument (element `head`).

NVDL

```
<rules xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0" startMode="root">
  <mode name="root">
    <namespace ns="http://www.w3.org/1999/xhtml">
      <validate schema="xhtml.rng">
        <context path="head"
                 useMode="rdf"/>
      </validate>
    </namespace>
  </mode>
  <mode name="rdf">
    <namespace ns="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
      <validate schema="rdf.rng" useMode="attach"/>
    </namespace>
  </mode>
  <mode name="attach">
    <anyNamespace>
      <attach/>
    </anyNamespace>
  </mode>
</rules>
```

6.3 Využití NVDL

I když jsme si popsali jen část možností celého jazyka NVDL, je zřejmé, že se jedná o velice silný nástroj. I přes jeho poměrně mladý věk již existuje řada standardů nebo jejich návrhů, které NVDL používají. Jedná se například o jazyky SVG 1.2, Office Open XML, DocBook, ITS nebo UBL.

6.4 Implementace

V současné době existují tři implementace jazyka NVDL, všechny z nich jsou open-source. Jedná se o implementace JNVDL[2] a oNVDL[3] v jazyce Java a enovdl pro platformu Mono/.NET.

[2] http://jnvdl.sf.net
[3] http://www.oxygenxml.com/onvdl.html

Kapitola 7. Best-practices pro návrh schémat

V následující části se podíváme na některá pravidla získaná praktickým používáním schémat. Tato pravidla nemusí platit ve všech případech, ale ve většině případů se vám rozhodně vyplatí se jimi řídit.

7.1 Elementy nebo atributy

Ve všech případech je možné atribut nahradit vnořeným podelementem. Mohlo by se proto zdát, že atributy jsou v XML zcela zbytečné. Nicméně existují dva případy, kdy se jejich použití může hodit:

- Potřeba kompaktního zápisu – hodnota atributu je uzavřená v uvozovkách nebo apostrofech, není potřeba ji uzavírat koncovým tagem. Ve většině případů je tak zapsání atributu podstatně kratší. To může mít význam při návrhu schémat pro dokumenty, které se budou editovat především ručně bez použití nějakých speciálních XML editorů. Na druhou stranu je pravda, že bychom se při návrhu schématu neměli nechat příliš omezovat požadavky kladenými nějakou nedostatečnou aplikací. Aplikace se mění a data v XML zůstávají – návrh dobrého formátu by proto měl mít přednost.

- Zjednodušení modelu obsahu – atributy se ve schémových jazycích jako WXS nebo DTD definují odděleně od podelementů. V některých případech by proto přesunutí atributu do podelementu mohlo vést k nutnosti rozvolnit stávající model obsahu a výsledkem by bylo schéma, které by bylo volnější než jsme původně zamýšleli.

Jaká existují další pravidla? U jednoho elementu není možné mít dva atributy stejného jména. Z tohoto důvodu nejde do atributu ukládat hodnoty s opakovaným výskytem.

Hodnotu atributu také nejde dále strukturovat pomocí podelementů. Atributy se proto nehodí pro uchovávání hodnot, které je potřeba dále strukturovat nebo lze očekávat, že v budoucnosti potřeba dalšího strukturování vznikne. Potřeba dalšího strukturování však hrozí u všech informací, které mohou obsahovat volný text. Nikdy nevíme, kdy náš formát dat začnou používat třeba v Japonsku nebo v Izraeli. Do japonského textu se však poměrně často vkládají tzv. Ruby anotace, které slouží k zápisu alternativy nějakého textu v jednodušší slabikové abecedě nebo v latince. Podobně, při kombinaci anglického textu s hebrejským je potřeba pomocí elementu označit, který text je anglicky a který hebrejský, aby se při zobrazování mohl správně změnit směr sazby textu.

Například v japonštině se „Tokyo“ v písmu kanji zapíše jako „東京“. U méně obvyklých slov se však nad kanji zápis často zapisuje i výslovnost pomocí slabikové abecedy hiragana:

text anotace Ruby → とう　きょう
základní text → 東　京

Je jasné, že text s anotací nejde reprezentovat prostým řetězcem, ale je k tomu potřeba struktura elementů. Naše ukázka by se pomocí značkování Ruby zapsala jako:

```
<ruby>
  <rb>東</rb>
  <rt>とう</rt>
  <rb>京</rb>
  <rt>きょう</rt>
</ruby>
```

Docházíme tedy k závěru, že do atributů je vhodné ukládat pouze hodnoty, u kterých dopředu známe obor hodnot a tento obor je poměrně omezený. Do této kategorie spadají například čísla, data nebo textové hodnoty, pro které máme číselník (např. kódy měn apod.).

7.2 Jmenné prostory

Každý nově navržený XML formát by měl definovat elementy ve vlastním jmenném prostoru. Elementy, které nepatří do jmenného prostoru nejde snadno a jednoznačně kombinovat v jednom dokumentu s jinými elementy. Uzavíráme si tak do budoucna cestu pro využití našeho formátu způsoby, které si dnes ještě ani nedovedeme představit.

Do jmenného prostoru by měly patřit všechny elementy, jejich atributy bychom přitom do jmenného prostoru zařazovat neměli. To znamená, že v WXS použijeme atribut `targetNamespace` a atribut `elementFormDefault` nastavíme na hodnotu `qualified` (viz 3.5 – „Jmenné prostory“). V RELAX NG se přiřazení jmenného prostoru provádí atributem `ns` (viz 4.5 – „Jmenné prostory“).

Otázkou zůstává, jak správně zvolit název jmenného prostoru pro nově vytvářené schéma. Vhodné je, aby byl název intuitivní, snadno zapamatovatelný a navržený tak, aby vás neomezil do budoucna. Název jmenného prostoru musí mít tvar URI adresy a dnes se běžně používají jak URN tak URL. Nejde jednoznačně říci, která z těchto variant je lepší.

Při používání URN musíme použít buď experimentální prostor začínající písmenem `x` nebo jej odvodit z doménového jména. Například já, jako vlastník domény kosek.cz, bych mohl používat URN začínající na:

```
urn:x-traktor:…
urn:x-test:…
urn:cz-kosek:…
```

Další část URN by měla naznačit, že URI se používá jako identifikátor jmenného prostoru (použitím slova jako je `ns` nebo `namespace`) a identifikovat jméno schématu. Následuje ukázka několika existujících i smyšlených jmenných prostorů, která využívají URN:

```
urn:oasis:names:tc:entity:xmlns:xml:catalog
urn:oasis:names:specification:ubl:schema:xsd:DocumentStatusCode-1.0
urn:x-test:ns:faktura
urn:x-test:namespace:faktura
urn:cz-kosek:ns:kalendar
```

Stále častější bývá používání URL jako názvů jmenných prostorů. Jejich výhoda spočívá v tom, že na jejich adresu lze umístit dokumentaci schématu a odkazy na související dokumenty. Nevýhodu lze zase spatřovat v tom, že začínající uživatelé XML jsou poněkud zmateni tím, že název jmenného prostoru je jen identifikátor a že se při zpracování dokumentu nečtou žádná data z URL adresy odpovídající názvu jmenného prostoru.

Rozhodneme-li se používat pro názvy jmenných prostorů URL, musíme je obzvláště pečlivě navrhnout, aby nekolidovaly s adresami používanými pro dokumenty umístěnými na webovém serveru. Obvykle se to řeší tak, že hned první část cesty v URL obsahuje slovo `ns` nebo `namespace` a tyto URL adresy jsou vyhrazeny pro tvorbu jmenných prostorů:

```
http://docbook.org/ns/docbook
http://relaxng.org/ns/structure/1.0
http://obix.com/ns/module/version
http://kosek.cz/namespace/kalendar
```

Někdy může být užitečné do URL zabudovat i číslo roku, aby se v budoucnu dal prostor URL rozšiřovat. Tento přístup využívají například jmenné prostory W3C:

```
http://www.w3.org/1999/xhtml
http://www.w3.org/1999/XSL/Transform
http://www.w3.org/2001/XMLSchema
```

Na adresu jmenného prostoru pak bývá vhodné umístit HTML stránku (nebo ještě lépe dokument RDDL[1]), která stručně popíše jmenný prostor, odkáže na dokumentaci jeho elementů a případně i na odpovídající schéma, existuje-li. Dokument bychom měli vystavit už z toho důvodu, že většina méně znalých se bude snažit psát adresu jmenného prostoru do prohlížeče, protože bude vypadat jako obyčejné URL.

7.3 Názvy elementů a atributů

Volbě názvu elementů a atributů je dobré věnovat zvýšenou pozornost, protože přispějí ke srozumitelnosti a použitelnosti schématu. Rozhodně není vhodné používat nesrozumitelné a kryptické názvy elementů jako A01, A02, … A72, byť by se třeba takto jednotlivé pole jmenovaly v nějakém starším formátu, ze kterého se přechází na XML. Ve starších formátech je toto pojmenování většinou dáno technickými omezení doby a zvolené technologie pro ukládání dat. Potřebujeme-li z důvodů zpětné kompatibility zachovat původní jména, můžeme je uložit do atributů s fixní hodnotou ve schématu nebo si mapování uložit v nějaké externí mapovací tabulce (která samozřejmě může mít podobu dokumentu XML, aby šla snadno použít např. při zpracování dokumentů pomocí XSLT).

Na druhou stranu bychom se samozřejmě měli vyvarovat i opačných extrémů, jako je element s názvem `SPZneboRegistračníZnačkaMotorovéhoVozidla`.

Název by měl být samopopisný v jeho kontextu, ale neměl by být na druhou stranu příliš dlouhý. U víceslovných názvů je dobré jednotlivá slova oddělit např. změnou velikosti písma, nebo vložením pomlčky či podtržítka. Např.: `JednotkováCena`, `jednotkováCena` nebo `jednotková_cena`. Důležité je rozhodnout se pro jednu konvenci a tu pak konzistentně používat. V opačném případě budou autoři vytvářející dokumenty podle vašeho schématu zmatení.

7.4 Defaultní a fixní hodnoty

Většina schémových jazyků umožňuje pro atributy (a někdy i pro elementy) určit implicitní hodnotu, která se atributu/elementu přiřadí, není-li atribut/element v dokumentu uveden. Používání této vlastnosti se v poslední době nedoporučuje, protože dokument pak obsahuje jiné informace, když je zpracován se schématem, a bez schématu.

7.5 Verzování schémat

Ve většině případů není schéma statický dokument, který jednou vznikne a už se dále nemění. Budoucnost obvykle přináší nové požadavky, pro které je nutné schéma upravit. Dnes se nejčastěji používá následující přístup.

Verze schématu se uvádí v atributu `version` u kořenového elementu dokumentu. S každým novou verzí schématu se toto číslo zvyšuje. Dojde-li pak ve schématu k radikální změně – změna názvu elementů, změna struktury dokumentu – obvykle nová verze schématu používá jiný jmenný prostor, než ta předchozí. Je-li však změna zpětně kompatibilní – přidají se například jen nové nepovinné elementy – jmenný prostor se nemění. To umožňuje beze změny používat i starší aplikace na nové dokumenty.

[1] http://www.rddl.org/

7.6 Rozšiřitelnost schémat

Počítáme-li s tím, že se schéma bude dále vyvíjet, nebo že má smysl, aby se dokumentu vkládaly i jiné elementy než definované ve schématu, je dobré je navrhnout jako otevřené. To znamená, že k libovolnému elementu dovoluje schéma přidat další podelementy.

Ve WXS se pro tuto definici používá element `any`. Jde u něj určit z jakého jmenného prostoru jsou elementy povoleny a kolik jich může být. Nejvolnější varianta je

```
<xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
```

která říká, že v daném místě se může vyskytovat libovolný počet elementů z libovolného jmenného prostoru. Kromě `##any` můžeme použít ještě identifikátory `##targetNamespace` pro aktuální cílový jmenný prostor, `##other` pro jakýkoliv jmenný prostor kromě cílového a pak jde samozřejmě zadat URI adresu nějakého konkrétního jmenného prostoru.

Pomocí atributu `processContents` ještě můžeme určit, jak se mají elementy popsané pomocí `any` validovat. Hodnota `skip` říká, že se elementy nebudou validovat, hodnota `strict` říká, že se musí validovat a konečně hodnota `lax` říká, že se validace provede v případě, že se pro elementy najde odpovídající schéma.

Vložení `any` alespoň na některá místa schématu podporuje rozšiřitelnost a je velmi žádoucí. Je však potřeba dávat pozor na to, aby před elementy definovanými pomocí `any` nebyly volitelné elementy (`minOccurs="0"`), protože pak schéma není jednoznačné. Například následující schéma tutu podmínku porušuje a není proto korektní:

WXS
```
<xs:element name="osoba">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="jmeno" type="xs:string"/>
      <xs:element name="email" type="xs:string" minOccurs="0"/>
      <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Problém je v tom, že při validaci dokumentu:

```
<osoba>
  <jmeno>Pepa</jmeno>
  <email>pepa@example.com</email>
</osoba>
```

nejde určit, zda se element `email` má validovat podle `element` nebo `any`. WXS je však navržené tak, aby vždy šlo instanci dokumentu jednoznačně mapovat na datové typy ve schématu – to je nesmírně důležité například při data-bindingu.

Podobně rozšiřitelně a navíc s mnohem méně omezeními lze schémata navrhovat v RELAX NG. Musíme si nejprve definovat vzor pro jakýkoliv i opakující se element a ten pak použít na odpovídajícím místě schématu.

RNG
```
<define name="anything">
  <zeroOrMore>
    <choice>
      <element>
        <anyName/>
```

```
        <ref name="anything"/>
      </element>
      <attribute>
        <anyName/>
        <text/>
      </attribute>
    </choice>
  </zeroOrMore>
</define>
```

Je zde použita speciální notace pro zápis jména elementu nebo atributu. Vzor `anyName` vyhoví jakémukoliv názvu. Jde jej kombinovat i se vzorem `except` a přesně určit jaké jmenné prostory jsou povolené a zakázané.

Kapitola 8. Nástroje

Následující část popisuje některé nástroje pro práci se schématy a pro validaci.

8.1 Validátory spouštěné z příkazové řádky

8.1.1 xmllint[1]

Pro zkontrolování správné strukturovanosti (well-form):

```
xmllint --noout dokument.xml
```

Pro validaci dokumentu oproti DTD:

```
xmllint --noout --valid dokument.xml
```

Pro validaci dokumentu oproti RELAX NG schématu:

```
xmllint --noout --relaxng schéma dokument.xml
```

Pro validaci dokumentu oproti WXS schématu:

```
xmllint --noout --schema schéma dokument.xml
```

xmllint bohužel nepodporuje WXS úplně, chybí např. podpora datových typů. Nebuďte proto překvapeni, že validací projdou i v tomto ohledu nevalidní dokumenty.

8.1.2 Xerces[2]

Pro zkontrolování správné strukturovanosti (well-form):

```
xerces dokument.xml
```

Pro validaci dokumentu oproti DTD:

```
xerces -v dokument.xml
```

Pro validaci dokumentu oproti WXS schématu:

```
xerces -v -s dokument.xml
```

xerces je přitom dávka, která spouští třídu `sax.Counter`.

8.1.3 Jing[3]

Jing je parser provádějící validaci oproti Relax NG schématu. Spuštění:

```
jing schéma.rng dokument.xml
```

Prvním parametrem může být i DTD, WXS nebo Schematron schéma.

[1] http://xmlsoft.org/
[2] http://xerces.apache.org/xerces2-j/
[3] http://www.thaiopensource.com/relaxng/jing.html

8.1.4 MSV[4]

MSV zvládá mnoho schémových jazyků, včetně DTD, RELAX NG a WXS. Pro validaci stačí zadat příkaz:

```
msv schéma dokument.xml
```

Existuje i upravená verze MSV, která dovoluje provádět validaci oproti RELAX NG schématu se schematronovými pravidly:

```
relames schéma.rng dokument.xml
```

8.1.5 XSV[5]

```
xsv dokument.xml schéma.xsd
```

8.1.6 xjparse[6]

Pro zkontrolování správné strukturovanosti (well-form):

```
xjparse -w dokument.xml
```

Pro validaci dokumentu oproti DTD:

```
xjparse -v dokument.xml
```

Pro validaci dokumentu oproti WXS schématu:

```
xjparse -s dokument.xml
```

```
xjparse -S schéma dokument.xml
```

xjparse je jen obálka okolo Xercesu, která umožňuje jeho snazší spuštění.

xjparse je dávka, která spouští třídu `com.nwalsh.parsers.xjparse`.

8.1.7 jnvdl[7]

Validace oproti NVDL schématu:

```
jnvdl -s schéma dokument.xml
```

[4] https://msv.dev.java.net/
[5] http://www.ltg.ed.ac.uk/~ht/xsv-status.html
[6] http://nwalsh.com/java/xjparse/index.html
[7] http://sourceforge.net/projects/jnvdl

Literatura

[1] P.V. Biron a A. Malhotra. *XML Schema Part 2: Datatypes*. W3C. 2001.
URL: http://www.w3.org/TR/xmlschema-2/.

[2] T. Bray, J. Paoli, C.M. Sperberg-McQueen a E. Maler. *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C. 2000. URL: http://www.w3.org/TR/REC-xml.

[3] T. Bray, D. Hollander a A. Layman. *Namespaces in XML*. W3C. 1999.
URL: http://www.w3.org/TR/REC-xml-names.

[4] J. Clark a MURATA Makoto. *RELAX NG Specification*. OASIS. 2001.
URL: http://relaxng.org/spec-20011203.html.

[5] J. Clark a MURATA Makoto. *RELAX NG Tutorial*. OASIS. 2003.
URL: http://relaxng.org/tutorial-20030326.html.

[6] J. Clark. *RELAX NG Compact Syntax*. OASIS. 2002. URL: http://relaxng.org/compact-20021121.html.

[7] J. Clark, J. Cowan a MURATA Makoto. *RELAX NG Compact Syntax Tutorial*. OASIS. 2003.
URL: http://relaxng.org/compact-tutorial-20030326.html.

[8] D. C. Fallside. *XML Schema Part 0: Primer*. W3C. 2001. URL: http://www.w3.org/TR/xmlschema-0/.

[9] P. Grosso a J. Kosek. *Associating Schemas with XML documents 1.0 (Third Edition)*. W3C. 2012.
URL: http://www.w3.org/TR/xml-model/.

[10] *Document Schema Definition Languages (DSDL) — Part 4: Namespace-based Validation Dispatching Language — NVDL ISO/IEC 19757-4*. 2006.
URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/c038615_ISO_IEC_19757-4_2006(E).zip.

[11] R. Jelliffe. *Schematron*. FDIS. ISO. URL: http://www.schematron.com/iso/Schematron.pdf.

[12] Petr Nálevka. *NVDL Tutorial*. URL: http://jnvdl.sourceforge.net/tutorial.html.

[13] H.S. Thompson, D. Beech, M. Maloney a N. Mendelsohn. *XML Schema Part 1: Structures*. W3C. 2001. URL: http://www.w3.org/TR/xmlschema-1/.

[14] E. van der Vlist. *Relax NG*. O'Reilly. 2003. ISBN ISBN 0-596-00421-4.
URL: http://books.xmlschemata.org/relaxng/page2.html.

[15] E. van der Vlist. *XML Schema*. O'Reilly. 2002. ISBN ISBN 0-596-00252-1.

Rejstřík

U

V

X

Z