GRADA

XML pro každého

PODROBNÝ PRŮVODCE

JIŘÍ KOSEK

Jiří Kosek

XML pro každého
podrobný průvodce

Kniha byla připravena ve formátu XML s využitím DTD DocBook. Závěrčná sazba byla provedena typografickým systémem TeX z písma Computer Modern ve variantě CS-font.

ISBN 80-7169-860-1

Obsah

Předmluva

Vážení čtenáři,

informace a jejich efektivní zpracování dnes mají pro mnoho jednotlivců i firem strategický význam. XML je jazyk, nebo chcete-li technologie, která do této oblasti nepřináší jen pokrok, ale přímo skok. Podobným skokem byl i vynález knihtisku, telegrafu, počítačů a Internetu.

Kniha, kterou právě držíte v ruce, by vás měla zasvětit do tajů jazyka XML. První část knihy vás po nezbytném úvodu detailně seznámí se syntaxí jazyka XML, s tvorbou odkazů mezi XML dokumenty a se stylovými jazyky, které se používají pro zobrazování a formátování informací uložených ve formátu XML.

Mnohem cennější je však druhá část knihy. V ní se dozvíte, k čemu se dá XML použít, jaké aplikace již XML úspěšně používají atd. Kromě toho je připojena i kapitola, jenž vás seznámí s programy, které vám umožní zpracování dokumentů XML. Samostatná kapitola je rovněž věnována využití XML na Webu.

Kniha obsahuje poměrně velké množství příkladů a odkazů na zajímavé programy a zdroje na Internetu. Ani příklady, ani internetové adresy nemusíte opisovat, vše naleznete na webové stránce `http://www.kosek.cz/xml/`. Pokud máte ke knize a jejímu obsahu nějaké připomínky, přivítám je na své e-mailové adrese `jirka@kosek.cz`.

Na tom, že kniha vůbec vznikla, má zasluhu mnoho lidí. V první řadě bych rád poděkoval své přítelkyně Lence, která trpělivě snášela večery, jež jsem trávil před obrazovkou počítače. Přitom věděla, že to není poprvé, ale ani naposledy. Pokud v knize nebude příliš chyb, je to díky redaktorovi Petru Somogyimu. Za všechny chyby, které zůstaly, však sypu popel na hlavu sobě.

Můj dík patří nespočetnému množství lidí, kteří se podíleli na vzniku jazyka XML a dalších technologií. Zpracování knihy mi nesmírně usnadnil textový editor Emacs a typografický systém TeX dík patří všem, kteří se na vývoji těchto výborných programů podíleli.

Přeji vám příjemné čtení knihy a příjemné chvíle strávené s XML.

Jirka Kosek

Praha Podolí, 12. března 2000

Typografické konvence

Aby byl text knihy pro čtenáře srozumitelnější, používám několik typografických konvencí, na které jste už zvyklí z mých předchozích knih a z knih nakladatelství Grada.

- `Neproporcionální písmo` používám pro zápis příkazů, funkcí, výpisů zdrojových kódů dokumentů a programů.
- *Kurzívu* používám pro zvýraznění nových pojmů v textu.
- V případě potřeby používám uzavření obecného pojmu do francouzských uvozovek. Tento pojem se pak v praxi vždy nahradí nějakou konkrétní hodnotou. (Např. *«soubor»* se nahradí konkrétním jménem souboru.)

Některé úseky textu jsou označeny piktogramy. Jejich význam je následující:

Takto označený text obsahuje důležitou informaci, jejíž neznalost vám může zkomplikovat život.

Např. *Pokud sáhnete na horká kamna, spálíte si ruku.*

Text obsahuje informaci, jejíž znalost vám může život usnadnit. Většinou zde naleznete různé tipy a triky, jak si ušetřit a zpříjemnit práci.

Např. *Pokud chcete sahat na horká kamna, pořiďte si azbestovou rukavici.*

Informace uvedené v takto označeném textu jsou zajímavé, ale jejich neznalost negativně neovlivní vaše základní životní funkce.

Např. *Oblíbená hudební skupina autora knihy jsou Jethro Tull.*

1. Úvod

Asi se shodneme na tom, že žijeme v informační době. Někdo tvrdí, že informační doba je tu od poloviny 20. století, někdo tak nazývá devadesátá léta a někdo až nové milénium. To však není podstatné. Podstatné je, že informace jsou pro naši společnost stále důležitější. Pokud se chceme jako firma uplatnit na trhu, musíme umět rychle a levně získat informace o konkurenci, o trhu a preferencích spotřebitelů. Jako spotřebitelé chceme bez námahy nalézt prodejce, který nám požadované zboží dodá nejrychleji a nejlevněji. Se svými zákazníky a dodavateli musíme rychle komunikovat, chceme, aby se objednávky a faktury vyřizovaly téměř okamžitě a ne s několikadenním prodlením. Pro chvíle volného času si chceme na Internetu rychle vybrat dovolenou podle našich snů nebo třeba lechtivé obrázky, které budou dle našeho gusta.

Aby to vše bylo možné, potřebujeme mít k dispozici efektivní způsoby sdílení a vyhledávání informací. Mohlo by se zdát, že dnes používané softwarové technologie ve spojení s moderními počítačovými sítěmi jako Internet našim požadavkům vyhoví. Opak je však pravdou. Dosud se pro výměnu dat používají proprietární formáty, se kterými dovedou pracovat jen úzké okruhy aplikací, výměna dat mezi informačními systémy jednotlivých firem je nákladná a zdaleka ne elegantní záležitost. Ani jazyk HTML se nestal dostatečně schopným lepidlem, které by dokázalo celý svět informačně propojit. HTML neuspělo ze dvou příčin. První spočívala v rozšiřování jazyka jednotlivými producenty prohlížečů, čímž došlo k nekompatibilitě jednotlivých prohlížečů. Druhým důvodem, který způsobil, že HTML již vyčerpalo svůj potenciál, jsou jeho poměrně omezené vyjadřovací schopnosti. Internet je dnes informacemi přehlcen a hledaná informace je často skryta ve velkém množství dalších pro nás nepotřebných informací.

Problém jazyka HTML je v tom, že se dnes používá spíše pro vyznačování vzhledu stránky, než pro označení logického významu jejích jednotlivých částí. Často se na stránkách používají složité tabulky, aby se dosáhlo požadovaného grafického layoutu. Podívejme se na to, jak může vypadat zdrojový kód části webové stránky, která obsahuje ceník.

```
<table border="0" cellspacing="0" cellpadding="4" bgcolor="#ffe0c0">
  <tr valign="top">
    <th width="120"><font size="-2" color="navy" face="Arial,
                          Arial CE, Helvetica,
                          sans-serif">Název</font></th>
    <th width="250"><font size="-2" color="navy" face="Arial,
                          Arial CE, Helvetica,
                          sans-serif">Popis</font></th>
    <th width="60"><font size="-2" color="navy" face="Arial,
```

```
                        Arial CE, Helvetica,
                        sans-serif">Cena</font></th>
  </tr>
  <tr valign="top" bgcolor="#fff0d0">
    <td><font size="-2" face="Arial, Arial CE, Helvetica,
              sans-serif"><strong>SuperInkJet 120TDi</strong></font></td>
    <td><font size="-2" face="Arial, Arial CE, Helvetica,
              sans-serif">Moderní inkoustová tiskárna</font></td>
    <td align="right"><font size="-2" face="Arial, Arial CE, Helvetica,
                            sans-serif">12.000,-</font></td>
  </tr>
  <tr valign="top" bgcolor="#fff0d0">
    <td><font size="-2" face="Arial, Arial CE, Helvetica,
              sans-serif"><strong>OfficeCom 56K</strong></font></td>
    <td><font size="-2" face="Arial, Arial CE, Helvetica,
              sans-serif">Faxmodem s hlasovými funkcemi</font></td>
    <td align="right"><font size="-2" face="Arial, Arial CE, Helvetica,
                            sans-serif">3.500,-</font></td>
  </tr>
  <tr valign="top" bgcolor="#fff0d0">
    <td><font size="-2" face="Arial, Arial CE, Helvetica,
              sans-serif"><strong>CD-RW 3246</strong></font></td>
    <td><font size="-2" face="Arial, Arial CE, Helvetica,
              sans-serif">Mechanika CD-ROM s možností vypalování
        a přepisu</font></td>
    <td align="right"><font size="-2" face="Arial, Arial CE, Helvetica,
                            sans-serif">9.500,-</font></td>
  </tr>
</table>
```

V prohlížeči pak na stránce uvidíme přehlednou tabulku s ceníkem.

Název	Popis	Cena
SuperInkJet 120TDi	Moderní inkoustová tiskárna	12.000,-
OfficeCom 56K	Faxmodem s hlasovými funkcemi	3.500,-
CD-RW 3246	Mechanika CD-ROM s možností vypalování a přepisu	9.500,-

Ne, mým cílem opravdu nebylo znechutit vám čtení knihy obludným výpisem HTML kódu hned na jejím začátku. Chtěl jsem, abychom si uvědomili, že dnešní, na pohled skvělé webové stránky, jsou jen soubory obsahující mnohdy nepřehledný balast. O tom, že v takovýchto datech se těžko něco hledá, není potřeba nikoho přesvědčovat.

Výše nastíněný problém se snaží odstranit nový jazyk *XML (eXtensible Markup Language)*. Největší přínos XML spočívá v tom, že v dokumentech můžeme používat vlastní značky (tagy). Pokud tedy vytváříme například ceník, můžeme v něm přehledně označit, co je název výrobku, jeho popis a cena. Vše bude přehlednější a kratší.

```
<ceník>
  <výrobek>
    <název>SuperInkJet 120TDi</název>
    <popis>Moderní inkoustová tiskárna</popis>
    <cena>12.000,-</cena>
  </výrobek>
  <výrobek>
    <název>OfficeCom 56K</název>
    <popis>Faxmodem s hlasovými funkcemi</popis>
    <cena>3.500,-</cena>
  </výrobek>
  <výrobek>
    <název>CD-RW 3246</název>
    <popis>Mechanika CD-ROM s možností vypalování a přepisu</popis>
    <cena>9.500,-</cena>
  </výrobek>
</ceník>
```

Na první pohled vidíme, že v takto strukturovaných datech se bude vyhledávat mnohem snáze. Inteligentní vyhledávací službě zadáme název výrobku a maximální cenu za pár sekund se nám na monitoru objeví seznam prodejen, kde mají k dostání zboží, které sháníme. XML však není určeno jen pro webové stránky, uplatnění nalezne i v elektronickém publikování nebo při výměně dat mezi různými systémy. Abychom lépe pochopili, proč vlastně XML vzniklo, podíváme se nyní stručně na historii vývoje tohoto jazyka. Zjistíme, že i když před třemi lety si pod zkratkou XML něco konkrétního představilo jen pár zasvěcených, jeho vývoj trvá již více než třicet let.

1.1 Stručná historie vývoje značkovacích jazyků

Pomineme-li vědecké výpočty, simulace jaderných zbraní a kryptografii, byly již od samotného počátku počítače využívány zejména pro přípravu a publikování textu. Situace v šedesátých letech však byla dosti odlišná od té dnešní. Laserové tiskárny byly hudbou budoucnosti. Pokud se na počítačích připravovaly dokumenty pro profesionální tisk knihy, časopisy apod., výsledek se pomocí osvitové jednotky přenesl na film, ze kterého pak tiskárny dokázaly vyrobit knihu nebo časopis. Osvitové jednotky tehdy vyrábělo několik firem a každá z nich používala vlastní jazyk pro její ovládání. Dokumenty pro sazbu se tedy připravovaly

tak, že se přímo do textu vepisovaly speciální řídící sekvence pro ovládání určité osvitové jednotky. Jednou vytvořený dokument byl tak úzce svázán s výstupním zařízením konkrétního výrobce. Jeho převod pro použití na konkurenční osvitové jednotce rozhodně nebyl jednoduchou záležitostí. V dnešní době, kdy všechny osvitové jednotky rozumí formátům PostScript a PDF, to zní neuvěřitelně, ale skutečně to tak tehdy bylo.

Tento stav rozhodně nebyl ideální a mnoho lidí si to uvědomovalo. Vzniklo proto několik systémů, které problém nekompatibility různých výstupních zařízení řešily. Princip byl většinou jednoduchý – v dokumentu se používaly nějaké obecné příkazy, které se pak pomocí speciálních konvertorů převedly do jazyka srozumitelného pro konkrétní zařízení. Dalo by se říci, že se jednalo o obdobu ovladačů různých výstupních zařízení, jak je známe dnes. Pokud jsme chtěli dokument vytisknout na nějakém novém zařízení, stačilo sehnat příslušný konvertor.[1] Samotný dokument se měnit nemusel.

Mezi nejrozšířenější z těchto systémů patřily bezesporu troff a TeX. Důležité je, že oba dva jazyky byly čistě prezentační – dalo se pomocí nich určit, jak se mají jednotlivé části textu formátovat. Troff používal poměrně kryptické dvouznakové příkazy. Oproti tomu byl TeX velice uživatelsky přívětivý – umožňoval definici maker ve vlastním programovacím jazyce a nekladl žádná omezení pro délku jednotlivých názvů. Bylo tak možné vytvářet přehledné a srozumitelné zdrojové zápisy dokumentů. Zdrojový kód v TeXu je poměrně lidsky čitelný, makropříkazy se míchají s textem.

```
Dokument v TeXu může obsahovat různé formátovací příkazy --
můžeme např. přepnout {\it na kurzívu} nebo {\bf na tučné} písmo.
```

Pro účely formátování textu pro tisk je v mnoha směrech TeX dodnes nepřekonán a stále se používá (například všechny mé knihy jsou vysázeny TeXem). Vývoj TeXu se nezastavil – dnes lze například pomocí upravené verze původního TeXu generovat dokumenty ve formátu PDF, v několika komerčních programech pro sazbu textu je integrováno jádro TeXu.

Programy jako TeX se však hodí pouze pro zpracování dokumentů, které se mají ve výsledku tisknout. Hlavně kvůli tomu, že nabízejí příkazy, které umožňují měnit druh použitého písma, způsob zarovnání a nepřeberné množství dalších parametrů. S rozmachem Internetu a dalších médií (např. CD-ROM) vznikla potřeba jedny a tytéž informace prezentovat mnoha způsoby – kvalitním tiskem na papíře, jako hypertextovou příručku na CD-ROMu či jako sadu provázaných webových stránek. Pro tyto účely je však potřeba znát logickou strukturu dokumentu. Musíme vědět, že tohle je nadpis a tohle zase popis obrázku. Konkrétní velikost písma a způsob formátování záleží až na tom, zda chceme produkovat tištěnou knihu nebo multimediální CD-ROM.

[1] Ostatně, pokud příslušný konvertor (ovladač) dosud neexistoval, nic nám nebránilo v tom napsat si vlastní.

Potřebujeme tedy jazyk, který umožní označit *význam* jednotlivých částí textu, a ne jejich *vzhled*. Takovýmto samopopisným jazykem je právě XML. Nejde však zdaleka o první jazyk svého druhu. Jazykům, které umožňují vyznačovat části textu, se říká *značkovací jazyky (markup languages)*.

Asi prvním známým značkovacím jazykem byl *GML (Generalized Markup Language)*, který vytvořili Charles Goldfarb, Edward Mosher a Raymond Lorie[2] při práci na systému pro uchovávání a následné využití právních textů pro IBM. Museli se tehdy vypořádat s nekompatibilitou jednotlivých systémů a programů a nejsnazší cesta vedla právě přes vytvoření nějakého obecného značkovacího jazyka.

Princip GML se osvědčil a v 80. letech začala na jeho základě vyvíjet standardizační organizace ANSI jazyk, který umožňoval definici vlastních značkovacích jazyků uživatel si podle potřeby mohl vytvořit vlastní sadu značek, vhodnou pro daný druh dokumentů. Sdružení GCA (Graphics Communications Association) již dříve vytvořilo standardní formátovací jazyk GenCode, použitelný na širokém spektru zařízení. Mnohé cíle obou projektů byly podobné, a proto se obě aktivity spojily. Výsledkem byl jazyk *SGML (Standard Generalized Markup Language)*, který je definován v ISO normě 8879 z roku 1986.

Jazyk SGML je skutečně hodně obecný samozřejmě umožňuje definici vlastních značkovacích jazyků (sad značek a jejich vzájemných vztahů) pomocí tzv. *definic typu dokumentu (DTD)*. Navíc má spoustu volitelných parametrů počínaje maximální délkou názvů značek a konče určením znaků použitelných jako oddělovače značek od textu. Komplexnost standardu SGML poněkud zbrzdila jeho praktické využití. Velkou podporu pro SGML znamenalo americké ministerstvo obrany, které od svých dodavatelů vyžadovalo dokumentaci právě ve formátu SGML. Důvod byl zřejmý bylo třeba, aby dokumentace byla použitelná v poměrně dlouhém období. Nebylo tedy možné použít nějaký proprietární formát textového procesoru, který se každých pár let mění.

Asi nejznámější aplikací SGML je jazyk HTML (Hypertext Markup Language), který se používá pro tvorbu webových stránek. Značky, které můžeme na stránkách používat, určuje příslušné DTD, které je pro každou verzi HTML trošku jiné.

V polovině 90. let došlo k paradoxní situaci. Jazyk HTML si získal velkou oblibu díky své jednoduchosti, která byla v ostrém kontrastu s komplexností SGML. Ukázalo se však, že pevně daná skupina značek, které HTML používá, už nestačí. Pro účely vyhledávání a vůbec efektivnější výměny dat by bylo lepší mít možnost používat vlastní značky, které by přesně vymezily význam textu. Požadavek by tedy mohl bez problémů splnit jazyk SGML.

---

[2] Když se podíváme na jména autorů, můžeme začít pochybovat o tom, jaký je skutečný význam zkratky GML.

Jak jsme se již zmínili, standard SGML je velmi komplexní a jeho úplná implementace velice náročná. Přitom se během deseti let používání SGML ukázalo, že se v praxi používá stejně jen část jeho možností. Tato nejdůležitější podmnožina SGML proto byla vybrána jako nový jazyk, který dovede Web do třetího tisíciletí. Správně již tušíte, že nový jazyk dostal jméno XML (eXtensible Markup Language). Jedná se o podmnožinu SGML, která si zachovává možnost definování vlastních DTD, a tedy i vlastních značek, pro jednotlivé skupiny dokumentů. Narozdíl od SGML je mnoho parametrů předem určeno a nelze je měnit maximální délka názvů značek, použité oddělovače a speciální znaky atd. XML už rovnou počítá s podporou všech možných jazyků, takže není tak úzce svázáno s angličtinou jako většina předchozích počítačových technologií. Navíc je syntaxe zápisu dokumentů v XML oproti SGML poměrně přísná, což umožní mnohem snazší a levnější vývoj aplikací, které umožňují s tímto jazykem pracovat.

Jak sami vidíte, XML pochází z oblasti, která se zaměřuje na uchovávání a zpracování textových dokumentů. Pro tyto účely se XML hodí výborně. Mnoho velkých i malých firem vyrábějících software, hardware nebo třeba letadla používá pro tvorbu dokumentace systémy založené na XML nebo SGML.

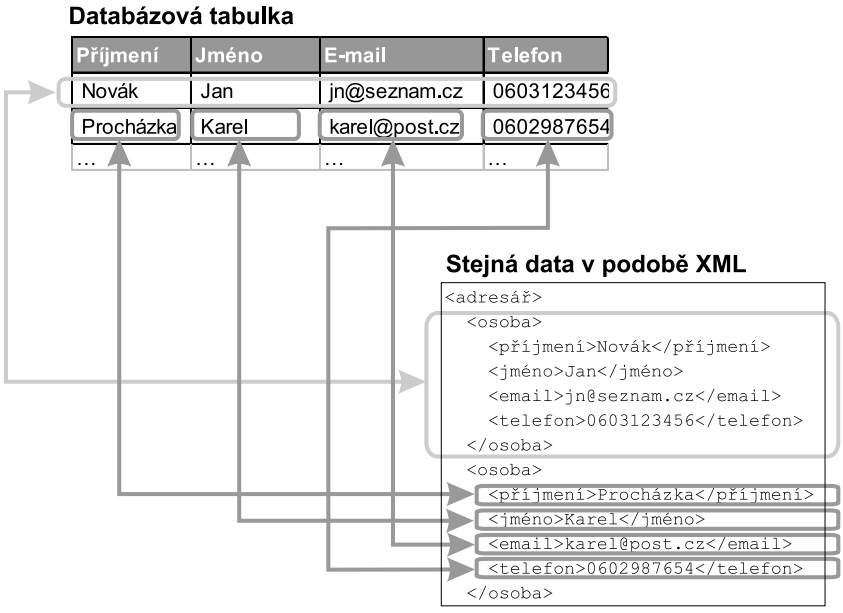
Elektronické publikování dokumentů však není jedinou doménou XML. Značky umožňují zachytit v dokumentu důležité informace o jeho struktuře a významu. Není proto problém do XML dokumentu uložit například obsah tabulky z relační databáze, jak je vidět na obrázku 1-1 na následující straně. O dokumentech bychom měli spíše uvažovat jako o nosičích informací není už tak důležité, jak moc jsou v nich data strukturovaná. Některé aplikace pracují s dokumentem, který je filosofickou esejí, jiné za dokument považují řadu čísel s burzovními indexy.

1.2 Co přináší XML nového

V médiích jsou možnosti XML líčeny jen v těch nejrůžovějších barvách. Soudný člověk pak často začne přemýšlet o tom, zda to vše není přeci jen přehnané. Pravdou je, že XML má mnoho rysů, které se diametrálně liší od dodnes používaných technologií. Jeho otevřenost a flexibilita skutečně může způsobit revoluci v práci s informacemi, která zasáhne každého uživatele, a nemusí to být jen uživatel klasického osobního počítače do hry přicházejí i mobilní telefony a různé elektronické organizéry (PDA). V následujících odstavcích se podíváme na jedinečné vlastnosti XML a jejich využití. Sami budete moci zvážit, zda XML představuje opravdu tak velký krok kupředu.

Standardní formát pro výměnu a sdílení informací

Dnešní doba přeje komunikaci. Komunikace není nic jiného, než výměna informací. V dnešním globálním světě není možné pro výměnu dat používat nějaké



Obr. 1-1: *XML není určeno jen pro texty, poradí si i s databázovými daty*

proprietární formáty, které jsou svázány s konkrétním softwarem nebo hardwarem. Nesluší se posílat informace ve wordovém formátu DOC, protože někdo s unixovým počítačem si je těžko přečte. Centrále nadnárodní společnosti asi nebudeme výroční zprávu české pobočky posílat ve formátu T602, protože ve své americké verzi kancelářského balíku si ji nikdo nepřečte. *Je potřeba používat nějaký jednoduchý otevřený formát, který není úzce svázán s nějakou platformou nebo proprietární technologií.*

Takovým formátem je například XML. Otevřený formát je to proto, že jeho specifikace je každému zdarma k dispozici na serveru konsorcia W3C, které se stará i o mnoho dalších technologií souvisejících s Webem. Každý tak může bez problémů do svých aplikací implementovat podporu XML. To představuje velký rozdíl oproti firemním formátům, k nimž není k dispozici žádná dokumentace a navíc se jedná v porovnání s XML o velice složité formáty, často binární.

Práci s XML usnadňuje i to, že celý formát je založen na obyčejném textu. I když pro většinu lidí zůstane kód XML skryt a budou ho používat pouze aplikace pro vzájemnou komunikaci, není problém kdykoliv otevřít XML dokument v libovolném textovém editoru a pár potřebných úprav provést ručně. Použití textového formátu může někomu připadat jako zbytečné plýtvání místem. Dnes se však mnohem větší důraz klade na srozumitelnost a snadnou práci s daty jestli ušetříme pár kilobajtů paměti, již nikoho příliš netrápí. Navíc většina protokolů pro síťovou komunikaci (včetně protokolu HTTP používaného na Webu)

umožňuje zcela transparentně pro potřeby přenosu data komprimovat a u příjemce zase dekomprimovat do původní podoby.

Mezinárodní podpora

XML je asi vůbec první formát, který hned od samého počátku dbá na potřeby jiných jazyků než je angličtina. Jako znaková sada se používá ISO 10646. Pod tím si asi nepředstavíte nic konkrétního. ISO 10646 je 32bitová znaková sada, která dokáže pojmout všechny znaky dnes používaných jazyků.[3]

V XML proto můžeme vytvářet dokumenty, které obsahují texty v mnoha jazycích najednou můžeme míchat např. češtinu, angličtinu, ruštinu, arabštinu a korejštinu zcela dle libosti. Pokud by dokumenty obsahovaly pouze český text, znamenalo by ukládání přímo v ISO 10646 zbytečné plýtvání místem. XML dokument proto může být v libovolném kódování (např. windows-1250, ISO 8859-2).[4] Kódování je však v každém dokumentu přesně určeno, takže odpadají problémy s konverzí z jednoho kódování do druhého. Každému je hned jasné, v jakém kódování je dokument.

Vysoký informační obsah

Pomocí XML značek vyznačujeme v dokumentu význam jednotlivých částí textu. Říkáme toto je název výrobku, tohle zase telefonní číslo a tohle je číslo našeho účtu. Dokumenty obsahují mnohem více informací, než kdyby se používalo prezentační značkovaní tohle je tučným písmem Arial o velikosti 12 bodů zarovnané vlevo.

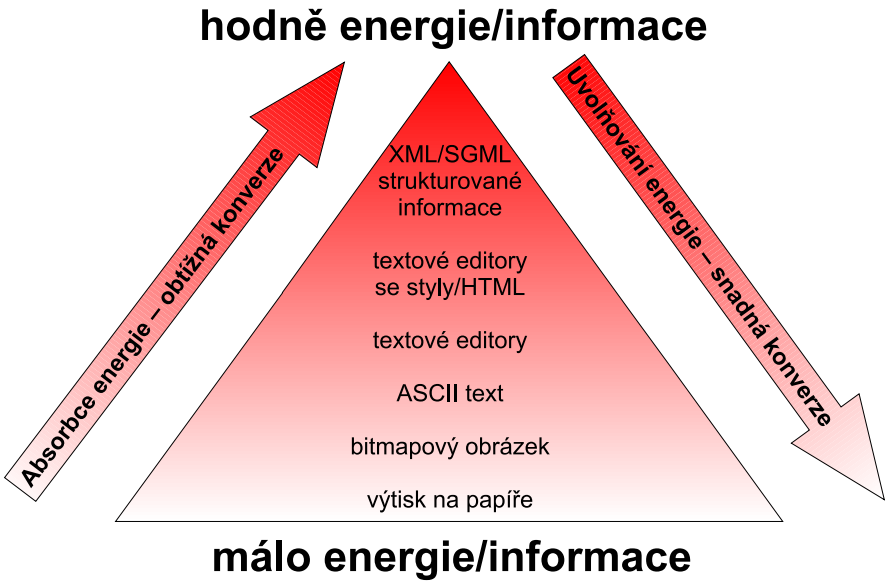
XML dokumenty jsou informačně bohatší. To lze samozřejmě s výhodou využít v mnoha oblastech. Největší přínos to bude znamenat samozřejmě pro prohledávání. Dnešní internetové vyhledávací služby jako AltaVista podporují pouze fulltextové vyhledávání. Zadáme hledaná slova a doufáme, že se nám vrátí dokumenty, které chceme. Pokud bychom mohli určit, že např. hledané slovo nás zajímá jako název firmy, bylo by při použití XML a vhodném označkování hledání mnohem přesnější.

Snadná konverze do dalších formátů

V mnoha případech potřebujeme XML dokument zobrazit na nějakém běžném médiu na obrazovce, na papíře. V tomto případě už samozřejmě chceme přesně

[3] Dokonce je v ní tolik místa, že existují návrhy na zařazení klingonského písma z televizního seriálu StarTrek.

[4] Kódování ISO 8859-2 je standardní kódování ISO, které by se mělo používat pro české znaky. V praxi je toto kódování používáno v unixových systémech včetně Linuxu. Microsoft toto kóování modifikoval a vytvořil tak nekompatibilní kódování windows-1250, které se používá pro české texty ve Windows.



Obr. 1-2: *XML dokumenty v sobě mají nejvíce informace, kterou mohou automaticky zpracovávat i počítače*

ovlivnit, jak se obsah jednotlivých značek zobrazí. XML samo o sobě žádné takové prostředky nenabízí. Existuje však naštěstí hned několik *stylových jazyků*, které umožňují definovat, jak se mají jednotlivé elementy zobrazit. Souboru pravidel nebo příkazů, které definují, jak se dokument převede do jiného formátu, se říká *styl*.

Výhodou je, že jeden styl můžeme aplikovat na mnoho dokumentů stejného typu. Dosáhneme tak jednotného formátování. Zároveň můžeme na jeden dokument aplikovat několik různých stylů. Jedním stylem vygenerujeme postscriptový soubor pro naše DTP studio, druhým HTML kód pro zařazení na naše webové stránky a třetím třeba jen obsah dokumentu, který pošleme mailem šéfovi.

Stylových jazyků existuje dnes několik. Mezi nejznámější patří asi kaskádové styly (CSS). Ty lze použít pouze pro jednoduché formátování, které dobře poslouží pro zobrazení dokumentu na obrazovce v XML editoru nebo v prohlížeči. Pro náročnější aplikace slouží jazyk XSL (eXtensible Stylesheet Language). Ten umožňuje před samotným formátováním dokument různě upravovat a transformovat (části dokumentu je možné třeba vypustit nebo naopak automaticky vygenerovat obsah dokumentu). Společně s XML lze použít i velice výkonný, i když pro některé aplikace příliš složitý jazyk DSSSL (Document Style Semantics and Specification Language), který byl původně vyvinut pro potřeby jazyka

SGML. Dříve se ještě hodně používal jazyk FOSI (Formatting Output Specification Instance).

Automatická kontrola struktury dokumentu

XML nám umožňuje definovat vlastní sadu značek, které chceme v dokumentu používat. Tuto možnost samozřejmě využít nemusíme můžeme používat libovolné značky. Pokud si však předem pomocí DTD definujeme, jaké značky může dokument obsahovat, bude náš další život mnohem lehčí. Zcela automaticky můžeme kontrolovat, zda dokument obsahuje pouze povolené značky. Programu, který kontroluje správnost XML dokumentů, se říká *parser*. Tento fakt má velký význam i při vývoji aplikací. Pro čtení dat můžeme použít parser, který za nás detekuje většinu chyb v datech obrovsky nám to ušetří práci. Kdo programuje, sám nejlépe ví, že většina kódu stejně ošetřuje různé chybové stavy a chyby ve vstupních datech.

DTD není jediný jazyk, který umožňuje definovat značky použitelné v dokumentech. DTD se hodí pro popis formátů, které se používají především pro textové dokumenty. Neobsahuje však nástroje pro kontrolu různých typů dat jako čísla, měnové údaje, údaje o datu a čase. To je přitom velice důležité pro aplikace, které si pomocí XML posílají data spíše databázového charakteru. Pro tyto potřeby existuje několik dalších jazyků, umožňujících určit správné schéma dokumentu. V současné době se pod názvem *XML schémata* pracuje na půdě konsorcia W3C na vytvoření jednotného standardu.

Příliš svobody může i škodit. Je sice hezké, že si každý může pojmenovat značky, jak chce, ale to zase přinese problémy při vyhledávání informací. Někdo název firmy označí pomocí značky `<název>`, někdo pomocí `<obchodníNázev>`, nebo třeba jako `<NázevFirmy>`. Jak se s tím pak má vyhledávací stroj vypořádat? Existují proto různé skupiny a sdružení, které vydávají DTD nebo schémata, jež by se měla používat v dané oblasti. Sem patří iniciativa Microsoftu známá pod názvem BizTalk nebo server XML.ORG provozovaný sdružením OASIS. Nejde přitom o nic jiného, než se shodnout na pár značkách, které se budou standardně používat pro označování určitých částí dokumentu. Dnes existují rozmanité sady značek počínaje těmi, které uspokojí potřeby e-businessu, a konče možností zachycovat informace o složitých chemických strukturách nebo astronomických údajích. Mezi tím si samozřejmě své místo našla např. i technická dokumentace nebo značky pro zápis matematických vzorců.

Velkou výhodou XML je to, že v jednom dokumentu můžeme používat najednou nezávisle na sobě několik druhů značkovaní díky *jmenným prostorům* (namespaces). Můžeme tak vytvářet dokumenty, které používají značky definované pro naše specifické účely, a pouze části dokumentu důležité pro vyhledávání označkujeme navíc pomocí nějakého standardizovaného DTD nebo schématu.

Hypertext a odkazy

XML samozřejmě umožňuje vytváření odkazů v rámci jednoho dokumentu i mezi dokumenty navzájem. Nabízí však mnoho možností nad rámec odkazů, které známe z HTML. Můžeme vytvářet i vícesměrné odkazy, které spojují několik dokumentů dohromady. Užitečná je i možnost uložení odkazů zcela mimo dokumenty, kterých se týkají. Tímto způsobem lze vytvářet různé anotace a komentáře k již existujícím stránkám.

Tvorba odkazů je dnes popsána ve třech standardech XLink, XPointer a XPath. XPath (XML Path Language) je jazyk, který umožňuje adresovat jednotlivé části dokumentu. Jeho možnosti dále rozšiřuje jazyk XPointer (XML Pointer Language). XPointer se používá k určování jednotlivých částí dokumentu ve stylu: zajímá mě první odstavec třetí kapitoly. Není proto potřeba všechny části dokumentu, na které chceme odkazovat, explicitně označovat pomocí návěstí jako v HTML.

XLink (XML Linking Language) je vlastní jazyk pro tvorbu odkazů. Jednotlivé dokumenty se samozřejmě určují pomocí jejich URL adresy, za kterou lze uvést ještě XPointer pro přesnější určení části dokumentu.

1.3 K čemu všemu můžeme XML použít

I když je technologie XML poměrně nová, je už dnes k dispozici mnoho softwarových balíků, které umožňují s XML pracovat. Mnoho z těchto programů je navíc k dispozici zdarma. My se teď podíváme na oblasti, ve kterých se XML nejčastěji nasazuje a přináší výhody oproti jiným technologiím.

B2B , business-to-business aplikace

Zkratka B2B je dnes v módě. V praxi se však nejedná o nic jiného, než o výměnu informací mezi obchodními partnery v elektronickém formátu. Jako vhodný formát pro přenos dat se jeví právě jazyk XML, který je velice jednoduchý a podporuje ho mnoho počítačových platforem. Pomocí XML si firmy mohou vyměňovat objednávky, faktury a mnoho dalších údajů.

To bylo možné již dříve díky EDI (Electronic Data Interchange). Datové formáty používané v EDI byly však dost složité a jejich implementace byla poměrně nákladná. Navíc jednotlivé systémy EDI nebyly mezi sebou kompatibilní, a tak bylo často nezbytné informační systém speciálně upravit pro každého dalšího obchodního partnera, se kterým jsme chtěli komunikovat elektronicky.

Použití XML však není omezeno jen na výměnu dat mezi obchodními partnery. V USA lze již pomocí XML posílat čtvrtletní finanční výkazy EDGAR pro úřad U. S. Securities. Dovedu si představit i formát, který umožní zasílání daňových přiznání v elektronické podobě. Použití XML pro tyto účely se jeví jako zcela ideální, protože přidání podpory tohoto formátu do stávajících účetních a ekonomických systémů je velice jednoduché.

Asi největší překážkou, která dnes brání masovému nasazení XML a dalších technologií pro důležitou obchodní a správní komunikaci, je neexistence zákona o digitálním podpisu, který by elektronicky provedené právní úkony postavil na roveň s klasicky podepsanými papírovými lejstry.[5]

£Inteligentní¥ webové stránky

S nasazením jazyka XML se počítá především na Webu. Možnost definice vlastních značek, které přesně vyznačí význam jednotlivých částí stránky, bude mít pozitivní efekt na přesnost a rychlost vyhledávání informací.

V zásadě lze XML pro tvorbu stránek využít dvěma způsoby. První, více revoluční přístup, počítá s tím, že stránky budou používat zcela vlastní sady značek. Pro mnoho aplikací je však mnohem jednodušší používat již zažité HTML značky a pouze je vhodně doplnit o pár dalších, kterými se označí části stránky důležité pro vyhledávání. Aby bylo rozšiřování HTML značek snadné, pracuje konsorcium W3C na převodu jazyka HTML do XML. Prvním výsledkem je jazyk XHTML 1.0, který odpovídá HTML 4.0. Drobné rozdíly mezi XHTML a HTML dokumenty jsou dány tím, že stránky v XHTML jsou zapisovány v souladu se syntaxí XML.

Nyní se pracuje na tom, aby se velké množství elementů, které XHTML (HTML 4.0) obsahuje, rozdělilo do několika nezávislých modulů. Budeme pak mít modul pro formátování textu, pro tvorbu odkazů, pro tabulky, pro výrobu formulářů nebo pro zařazování obrázků. Nebude problém vytvořit vlastní modul (tedy sadu značek), které budeme na našich stránkách používat např. pro označení důležitých informací o nabídce naší firmy. Tyto informace pak mohou využít inteligentní prohledávací služby.

W3C jde ve svých úvahách ještě dál. Každý XHTML dokument bude obsahovat i svůj profil seznam modulů, které používá, společně se seznamem grafických a dalších formátů, které se používají pro vložené objekty (jako např. obrázky). Každý prohlížeč pak bude společně s požadavkem na stránku posílat i svůj profil informaci o tom, co dané zařízení zvládne zobrazit. Pokud bude stránka dostupná ve více variantách, server vybere tu s odpovídajícím profilem. V případě potřeby může server automaticky stránku konvertovat pro profil klienta. Tento poměrně obecný model umožní vývoj stránek pro mnoho zařízení s rozdílnými schopnostmi PC, mobilní telefony, WebTV, organizéry nebo třeba herní konzole. Dnes nám to může připadat jako futuristická vize, ale podle mnoha výzkumů a studií již za pár let bude většina uživatelů přistupovat k Internetu právě pomocí různých speciálních zařízení. Klasická PC budou v menšině.

Moderní mobilní telefony podporují protokol WAP (Wireless Application Protocol), který je obdobou služby World Wide Web právě pro jednoduchá bez-

[5] V době psaní knihy bohužel vláda rozhodla o tom, že digitální podpis ještě nepotřebujeme. Doufejme, že brzy dostane rozum a zákon bude na světě.

drátová zařízení. Pro tvorbu stránek ve WAPu se používá jazyk WML (Wireless Markup Language), který je založen na XML a je definován pomocí DTD.

XML se na Webu nemusí používat jen pro tvorbu stránek. Funkčnost mnoha webových aplikací je dnes rozložena mezi server a prohlížeč. XML je optimální formát i pro výměnu dat mezi serverem a prohlížečem. Klient si vyžádá část dat od serveru a formátování podle různých požadavků uživatele je již plně v rukou klienta.

Pokud má spolu spolupracovat více různých webových aplikací, je rovněž potřeba, aby si vyměňovaly informace. Samozřejmě lze použít nějaké proprietární formáty, ale existují i de facto standardy XML-RPC a WDDX. První z nich umožňuje vyvolávání vzdálených procedur implementace XML-RPC je velice jednoduchá, protože vše je postaveno na dobře zavedených standardech. Data se mezi aplikacemi posílají pomocí protokolu HTTP a jsou uložena v XML. WDDX je zase datový formát založený na XML, který není závislý na žádném konkrétním programovacím jazyce. Pokud si mají nějaká data předávat webové aplikace napsané v ASP, PHP, Perlu nebo jiném jazyce, není nic snazšího, než k tomu využít WDDX.

Metadata

Metadata aneb data o datech. Pro vyhledávání, ale hlavně pro klasifikaci dokumentů je užitečné o nich znát co nejvíce metadat. Metadata pro dokument představují takové údaje jako autor dokumentu, datum vytvoření, vlastník copyrightu, druh dokumentu apod. Asi nejperspektivnější formát pro zápis a výměnu metadat je RDF (Resource Description Framework), který umožňuje k libovolnému dokumentu připojit libovolná metadata.

Do kategorie metadat patří i například formát Microsoftu CDF (Chanell Definition Format), který umožňuje jednoduchou syntaxí, založenou na XML, definovat zajímavé internetové zdroje. Prohlížeč pak pro nás může informace ze zdrojů určených pomocí CDF automaticky stahovat.

Elektronické publikování

Když vynalezl Guttenberg knihtisk, byla to skutečná revoluce v šíření informací. Do té doby se všechny knihy musely rozmnožovat ručním opisováním. Guttenbergův vynález umožnil jednou připravit předlohu stránky a z ní tiskem pořídit mnoho kopií. Tento princip využíváme i dnes z jedné předlohy vyrobíme mnoho kopií. Celý postup je sice odlišný, pro přípravu textu a jeho zlom se používají počítače, ale princip zůstal stejný.

Papír však dnes není jediné cílové médium. Jak jsme se již zmínili, často potřebujeme jeden dokument v několika různých formátech jako tištěnou knihu, sadu provázaných webových stránek nebo hypertextovou příručku na CD-ROMu. Stojíme před novým problémem už nestačí pohodlně vytvořit text

a ten rozmnožit v libovolném počtu výtisků. My navíc potřebujme tento text publikovat v několika naprosto odlišných formátech.

Dnešní textové editory nám pomohou s přípravou dokumentů, které se mají tisknout. Z vlastní zkušenosti však víte, že HTML stránky vytvořené pomocí běžných textových editorů z dlouhých dokumentů nejsou zrovna to pravé ořechové. Několika set stránkovou knihu prostě nemůžeme dát na Web jako jednu dlouhou stránku. O možnosti vytvoření dalších formátů, vhodných např. pro publikování na CD-ROMu nebo jako on-line nápověda v aplikacích, ani nemluvě.

Nové možnosti přinášejí do této oblasti (někdy poněkud vágně nazývané elektronické publikování) právě technologie SGML a XML. Pokud máme naše dokumenty uloženy v XML, můžeme je pomocí stylů velice snadno zcela automaticky konvertovat do mnoha dalších formátů. Stylové jazyky jako DSSSL a XSL jsou velice flexibilní. Jednou proto můžeme z dokumentu vygenerovat PDF soubor vhodný pro DTP, podruhé zase sadu HTML stránek, kde každá stránka představuje jednu kapitolu dokumentu.

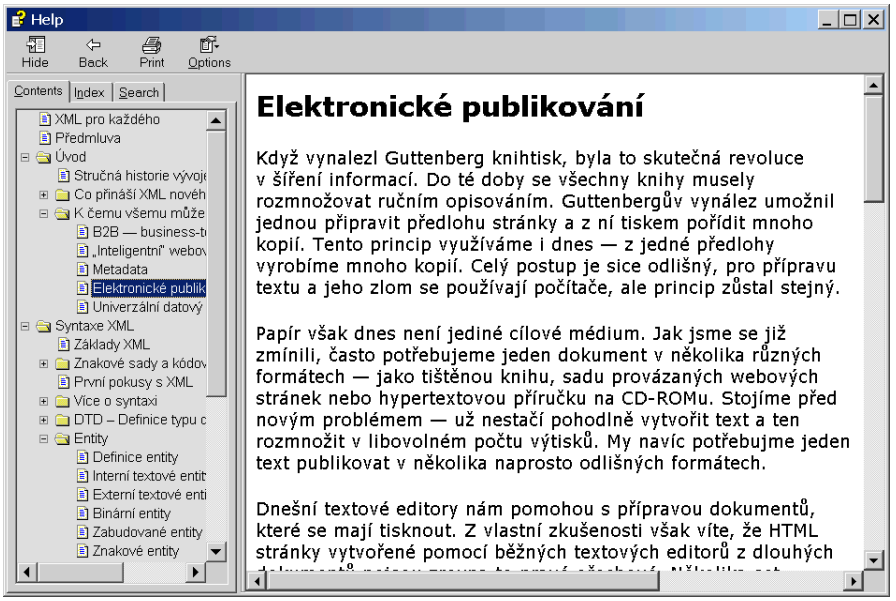
Flexibilita stylových jazyků umožňuje, aby se z jednoho zdroje generovalo několik druhů dokumentů s různým obsahem některé údaje v technické dokumentaci jsou například tajné a vytisknou se pouze pro potřeby firmy. Zakazníkům se ze stejného XML dokumentu vygeneruje okleštěná verze.

XML se proto používá zejména při tvorbě různé technické dokumentace, kdy lze z jednoho zdroje generovat mnoho formátů podle potřeb jednotlivých uživatelů. Řešení založená na SGML používaly zejména velké firmy, protože byla poměrně nákladná. Použití XML však vše zjednodušuje, a vyplatí se v podstatě komukoliv. Například tato kniha je rovněž napsána v XML. Pro redakci nakladatelství z tohoto zdrojového tvaru dokumentu vytvořím PDF soubor, který se použije pro tisk knihy. Stejně tak mohu vygenerovat knihu ve formátu HTML Help, který se dnes používá v mnoha nových aplikacích pro Windows. Kdybych toho všeho chtěl dosáhnout pomocí nějakého běžného editoru z kancelářského balíku, asi bych se dost zapotil.

Jen pro ilustraci, jak různorodí mohou být jednotliví uživatelé značkovacích jazyků určených pro tvorbu dokumentace: dodavatelé amerického ministerstva obrany, výrobce letadel Boeing, vývojové týmy Linuxu, FreeBSD, PHP a mnoha dalších open-source projektů. Pokud budete chtít vydat encyklopedii na CD-ROMu a zároveň i jako sérii tištěných svazků pro bibliofily, těžko pro to naleznete lepší technologii než XML.

Univerzální datový formát

Pokud jste někdy psali nějaký větší program, brzy jste asi zjistili, že je potřeba, aby si tento program někde pamatoval nastavení svých parametrů. Klasicky se parametry ukládají do různých textových nebo INI souborů, do registrů Windows apod. Pro ukládaní složitějších parametrů se však tyto způsoby nehodí a navíc je poměrně komplikované psát program, který bude načítat jednotlivé



Obr. 1-3: *XML dokumenty lze snadno konvertovat do mnoha dalších formátů*

parametry z konfiguračního souboru. Pokud by si program ukládal data v XML, mohl by je programátor velice snadno číst pomocí některé z knihoven pro práci s XML. Do souboru by bylo možné uložit i složité struktury a navíc by byl pořád jen textovým souborem, kterému by zkušený uživatel rozuměl a na jehož úpravy by si vystačil s jednoduchým editorem (už teď slyším, jak všichni administrátoři volají hurá ;).

Pokud by se navíc výrobci jednotlivých druhů aplikací shodli na společných formátech, těžili by z toho především uživatelé. Kdyby například pro ukládání záložek používal Internet Explorer i Netscape Navigator stejný formát vycházející z XML, byl by konec nepříjemnostem s konverzí záložek při přechodu z jednoho typu prohlížeče na druhý. Navíc by šlo adresy našich oblíbených stránek velice snadno poslat komukoliv e-mailem bylo by jedno, jaký kdo používá prohlížeč.

Pokud mají informační a komunikační systémy fungovat efektivně, je potřeba, aby se používaly jednotné formáty dat. Bez nich se spousta prostředků zbytečně vyplýtvá na implementaci různých pomocných utilit a modulů, které umožní spolupráci softwaru od různých výrobců. Dnes si je toho vědoma většina komerčních firem i akademických institucí. Profesní sdružení proto vytvářejí formáty založené na XML, které vyhovují jejich potřebám.

2. Syntaxe XML

Samotný princip XML je velice jednoduchý. V této kapitole se podrobně seznámíme se syntaxí. Narozdíl od jazyka HTML, kde můžeme udělat spoustu chyb, se kterými si prohlížeče většinou poradí, je správný zápis XML dokumentů předpokladem pro jejich další zpracování.

Pokud vás syntaxe jazyka XML nezajímá, ale chcete znát aplikace, které XML podporují, můžete pokračovat kapitolou *Aplikace podporující XML* na straně 121.

2.1 Základy XML

Každý XML dokument se skládá z *elementů*, které jsou do sebe navzájem vnořené. Elementy se v textu vyznačují pomocí tzv. *tagů*. Většině elementů odpovídají dva tagy počáteční a ukončovací.[1]

```
<para>Toto je obsah elementu para.</para>
```

Ukázka obsahuje jeden element `para`. Jeho obsah je vyznačen pomocí tagů `<para>` (počáteční tag) a `</para>` (ukončovací tag). Jen na okraj poznamenejme, že výše uvedená ukázka je asi nejjednodušším XML dokumentem, který můžeme vytvořit.

Názvy tagů se zapisují mezi znaky '<' a '>'. Ukončovací tag má před svým názvem ještě znak '/', aby se snadno odlišil od počátečního.

Některé elementy nemusejí mít žádný obsah. Můžeme je samozřejmě zapisovat tak, že za počátečním tagem uvedeme hned ten ukončovací.

```
<para>Toto je obsah elementu para.<br></br> A tohle taky.</para>
```

Není to však příliš pohodlné, a proto můžeme v XML použít ještě jednu variantu tagu, která říká, že element nemá žádný obsah. Za jméno elementu v počátečním tagu se uvede znak '/'. Ukončovací tag se pak už nepoužije.

```
<para>Toto je obsah elementu para.<br/> A tohle taky.</para>
```

Každý XML dokument musí obsahovat pro všechny počáteční tagy odpovídající ukončovací tag, nebo musí být počáteční tag zapsán jako element s prázdným obsahem. Následující ukázky jsou ukázkami špatných dokumentů, které nevyhovují specifikaci XML.

[1] V některých českých publikacích se pojmy element a tag nerozlišují a používá se pro ně společný pojem značka.

Obr. 2-1: *Základní komponenty XML dokumentů*

```
<para>Toto je obsah elementu para.<br> A tohle taky.</para>
```

Tag `<br>` není ukončen.

```
<para>Toto je obsah elementu para.<br/> A tohle taky.</oara>
```

Počáteční tag `<para>` není ukončen a k ukončovacímu tagu `</oara>` v dokumentu neexistuje odpovídající počáteční tag. Chybou rovněž je, když se elementy v dokumentu kříží.

```
<b>Ukázka <i>překřížení</b> elementů</i>
```

Elementy jsou základním stavebním kamenem každého dokumentu. U každého počátečního tagu můžeme použít ještě *atributy*. Atributy se obvykle používají k upřesnění významu elementu.

```
<para zabezpečení="důvěrné">Nějaká tajná informace.</para>
```

V naší ukázce jsme atributu `zabezpečení` přiřadili hodnotu `důvěrné`. Hodnotu atributu musíme vždy uzavřít do uvozovek nebo do apostrofů. U jednoho tagu lze použít více atributů najednou, stačí je oddělit mezerou.

```
<para zabezpečení="důvěrné" autor="Jan Novák">Nějaká tajná informace.</para>
```

Vzhledem k tomu, že se znaky '<' a '>' používají pro oddělení tagů od okolního textu, není možné tyto znaky zapsat do dokumentu jen tak. Pro jejich zápis musíme použít tzv. *znakové entity*. Pro zápis znaku '<' je určena entita `<` a pro '>' to je `>`.

```
Vyřešte nerovnost 3x &lt; 5
```

Pro samotný zápis ampersandu (`&`) se používá znaková entita `&`.

```
Křupavé rohlíčky vám dodá pekařství Žemlička &amp; syn
```

Pokud potřebujeme uvnitř hodnoty atributu použít zároveň uvozovky i apostrofy, s výhodou využijeme odpovídající entity `&quot;` a `&apos;`.

Ve skutečnosti stačí entitami v textu nahrazovat pouze znaky '<' a '&'. Pokud však nahradíme i '>', nic tím nezkazíme.

Každý XML dokument musí být celý obsažen v jednom elementu. Následující ukázka tedy nepředstavuje správný XML dokument.

```
<nadpis>Pokusný nadpis</nadpis>
<odstavec>První odstavec</odstavec>
<odstavec>Druhý odstavec</odstavec>
<odstavec>Třetí odstavec</odstavec>
```

Stačí však přidat jeden element, který vše obalí, a vše je v pořádku.

```
<článek>
  <nadpis>Pokusný nadpis</nadpis>
  <odstavec>První odstavec</odstavec>
  <odstavec>Druhý odstavec</odstavec>
  <odstavec>Třetí odstavec</odstavec>
</článek>
```

Splňuje-li dokument všechna výše uvedená pravidla, je syntakticky v pořádku a říkáme o něm, že je *správně strukturovaný (well-formed)*. Takový dokument můžeme směle vypustit do světa, protože si s ním poradí všechny aplikace podporující formát XML.

Většinou však existují pro dokument různá omezení chceme mít pod kontrolou elementy, které se mohou v dokumentu používat apod. O tom, jak to zařídit, se více dočtete v dalších částech této kapitoly a v kapitole *XML schémata* na straně 93.

2.2 Znakové sady a kódování

Současné počítače pracují vnitřně pouze s čísly. Na obrazovce sice vidíme texty, obrázky nebo třírozměrný model bludiště, ale někde za tím vším jsou již jen čísla. S texty se pracuje také jako s čísly. Každému znaku je přiřazeno číslo. Sada znaků a jím odpovídajících čísel je *znaková sada*.

Mezi nejstarší a nejznámější znakové sady patří ASCII. Tato znaková sada byla 7bitová obsahovala znaky s kódy 0 až 127. Kromě písmen anglické abecedy, číslic a dalších znaků obsahovalo ASCII i některé řídící znaky. Potřebám angličtiny ASCII zcela vyhovovalo. Pro ostatní jazyky zde však chyběla některá písmena pro češtinu například znaky s diakritikou. Vzniklo proto několik 8bitových kódování, která obsahovala 256 znaků. Prvních 128 znaků bylo kvůli zpětné kompatibilitě shodných s ASCII. Horní část znakové sady pak obsahovala národní znaky.

Pro češtinu a další středoevropské jazyky existuje znaková sada ISO 8859-2, pro rušinu ISO 8859-5 apod. Microsoft ve Windows používá vlastní znakové sady, které se od ISO mírně liší. Pro češtinu je vhodná znaková sada windows-1250.

Pokud bychom však chtěli v jednom dokumentu používat více různých jazyků, dostaneme se do problémů, protože například znaková sada pro češtinu už neobsahuje azbuku. Postupně proto vznikly ještě další znakové sady, které obsahovaly více znaků. Mezi nejznámější patří 16bitová sada Unicode a 32bitová ISO 10646.

Výhodou těchto znakových sad je, že obsahují znaky všech běžně používaných jazyků kromě češtiny či ruštiny zde nalezneme i pro nás exotické jazyky jako arabštinu, korejštinu, japonštinu a mnoho dalších.

XML používá jako znakovou sadu ISO 10646, protože je to dnes nejkomplexnější znaková sada a dá se očekávat, že v blízké budoucnosti ji bude přímo podporovat většina operačních systémů. V následujícím textu se podíváme na to, co pro nás použití znakové sady ISO 10646 znamená v praxi.

Kódování znakových sad

Zatímco znaková sada definuje, jaké znaky a pod jakým číslem máme k dispozici, kódování znakové sady určuje, jak jsou jednotlivé kódy znaků převedeny na sekvenci bajtů, které znak reprezentují v paměti počítače, v souboru, při přenosu počítačovou sítí apod.

Mezi nejjednodušší kódování patří UCS (Universal Multiple-Octet Coded Character Set). Existují dvě varianty tohoto kódování UCS-4 a UCS-2. V první

variantě se jeden znak ukládá jako čtyři bajty a jeho kód tedy odpovídá i číslu přiřazenému ve znakové sadě. Kódování UCS-2 kóduje jeden znak do dvou bajtů. Pomocí tohoto kódování nejsou dostupné všechny znaky, ale pouze prvních 65536. To však příliš nevadí, protože ostatní znaky s vyšší číslem zatím nejsou definovány. Výhodou kódování UCS-4 a UCS-2 je, že znaky z ASCII mají stejný kód. Pro angličtinu jsou tedy tato kódování zpětně kompatibilní.

Další zajímavostí je, že kódování UCS-2 přímo odpovídá dvoubajtové znakové sadě Unicode. Pro těchto prvních 65536 znaků se vžilo označení BMP (Basic Multilingual Plane). Všechny identifikátory v XML (jména elementů, atributů, návěstí apod.) by měly obsahovat pouze znaky z BMP.

Nevýhodou kódování jako UCS-4 a UCS-2 je velké plýtvání prostorem. Pro přenos jednoho znaku jsou potřeba 4 resp. 2 bajty. Pokud bychom pomocí těchto kódování přenášeli například jen anglické texty, bylo by to velice neefektivní. Z tohoto důvodu vznilo kódování UTF (UCS Transformation Format), které znaky z ASCII kóduje do jednoho bajtu. Méně obvyklé znaky jsou pak kódovány v několika bajtech.

Kódování UTF-8 je identické s ASCII. Další znaky nad rámec ASCII jsou kódovány do sekvencí 2 až 6 bajtů. Pokud by se dokument skládal ze samých exotických znaků, byl by tedy výsledek delší než při použití UCS-2 nebo UCS-4. České znaky s diakritikou se v UTF-8 kódují do dvou bajtů, takže si oproti UCS pomůžeme.

Existuje ještě kódování UTF-16. To je velice podobné kódování UCS-2, ale navíc některým kódům přiřazuje speciální význam pro přepnutí do dalších 15 znakových rovin, jinak dostupných pouze v UCS-4. Kódování UTF-16 je de facto totožné s Unicode. Zatímco ISO 10646 dosud není příliš podporováno aplikacemi, Unicode se používá např. ve Windows NT nebo v Javě.

Všechny aplikace, které podporují XML, by měly zvládat práci s kódováními UTF-8 a UTF-16. Pro nás tato kódování nejsou nejvhodnější, jsme zvyklí spíše na windows-1250 a ISO 8859-2.[2] Naštěstí lze u každého XML dokumentu určit používané kódování. Pokud však budeme používat jiné než UTF-8 nebo UTF-16, musíme si dát pozor, aby toto kódování podporovaly všechny aplikace, s nimiž budeme XML dokument zpracovávat.

Pro dokumenty, které posíláme do širého světa, bychom měli používat kódování UTF-8, protože jej podporují všechny aplikace.

[2] I když jsme dosud o windows-1250 a ISO 8859-2 mluvili jako o znakových sadách, můžeme na ně pohlížet i jako na kódování, která pokrývají pouze část ISO 10646.

Určení kódování použitého v dokumentu

Pokud v dokumentu používáme jiné kódování než UTF-8 nebo UTF-16, musíme ho specifikovat pomocí *XML deklarace*, která musí představovat první řádku dokumentu. Nejjednodušší XML deklarace má následující tvar.

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

Deklaraci můžeme používat ve všech dokumentech. Standardně obsahuje pouze určení verze XML, pro zachování zpětné kompatibility v případě dalších verzích XML.

Použité kódování můžeme určit pomocí parametru `encoding`, který je součástí XML deklarace.

```
<?xml version="1.0" encoding="windows-1250"?>
<dokument>
  Dokument si vesele píši ve starém špatném Notepadu.
</dokument>
```

Pokud dokument nepíšeme v UTF-8 nebo UTF-16, musíme kódování určit v XML deklaraci. Pokud to neuděláme, totálně tím zmateme aplikaci, která má s XML dokumentem pracovat.

V další části kapitoly uvidíme, že jeden XML dokument může být fyzicky uložen v několika souborech. Dílčí soubory se načítají jako tzv. externí entity. Pro každou externí entitu můžeme rovněž určit její kódování a to tak, že na prvním řádku použijeme deklaraci kódování.

```
<?xml encoding="«kódování»"?>
```

Zápis znaku s libovolným kódem

Může se stát, že náš editor nepodporuje všechny znaky, které potřebujeme použít. Například píšeme dokument v češtině, ale potřebujeme v něm použít řecké písmeno ?. To se ve většině 8bitových kódování (jako např. windows-1250 nebo ISO 8859-2) nevyskytuje. Proto můžeme do XML dokumentu vložit libovolný znak pomocí *znakové entity*. Znaková entita má tvar `&#x`*«kód znaku»*`;`, kde *«kód znaku»* je kód znaku ze znakové sady ISO 10646 zapsaný v šestnáctkové soustavě. Můžeme použít i tvar `&#`*«kód znaku»*`;`, kde je *«kód znaku»* zapsán v desítkové soustavě.

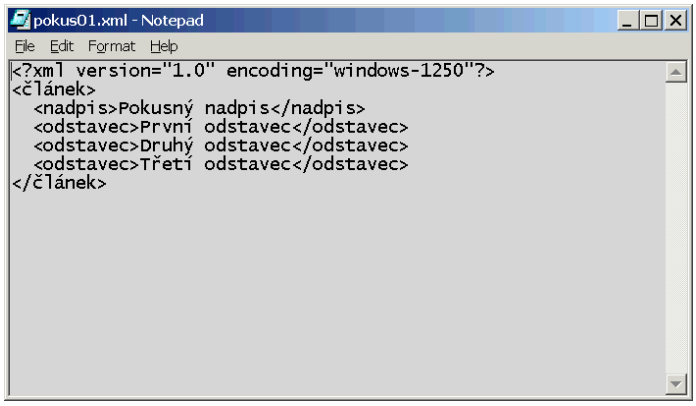
```
Obsah kruhu se vypočte podle vzorce &#x3c0;r<sup>2</sup>.
Originální&#32;způsob, jak vložit do textu mezeru.
```

Přehled znaků a jejich kódů naleznete například na serveru sdružení Unicode.[3]

[3] `http://charts.unicode.org`

2.3 První pokusy s XML

Už toho dovedeme dost na to, abychom začali vytvářet první pokusné dokumenty a na nich vše testovali. Syntaktickou správnost XML dokumentu lze kontrolovat pomocí *parseru*. Parser může mít mnoho podob samostatný program, integrální součást prohlížeče nebo editoru, knihovna pro vyšší programovací jazyk. Asi nejjednodušší je pro první pokusy použít parser, který je přímo obsažen v prohlížeči. Internet Explorer obsahuje podporu XML od verze 5.0. Netscape také slíbil podporu XML ve verzi 5.0 svého prohlížeče, ale v době psaní knihy nebyla tato verze Navigatoru ještě k dispozici. Nicméně podpora XML je obsažena v prohlížeči Mozilla, který je vývojovou verzí Netscape Navigatoru. Pro první pokusy s XML bychom si měli alespoň jeden z těchto prohlížečů opatřit.



Obr. 2-2: *Příprava XML dokumentu v editoru*

Pokusné dokumenty pak stačí ukládat do souborů s příponou `xml`. Pozor si musíme dát na kódování dokumentu. Patrně nebudeme mít k dispozici editor, který podporuje kódování UTF-8.[4] Při psaní proto budeme používat kódování, které je obvyklé v našem operačním systému. Ve Windows by proto všechny naše XML dokumenty měly začínat řádkou

```
<?xml version="1.0" encoding="windows-1250"?>
```

V unixovém operačním systému pak

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

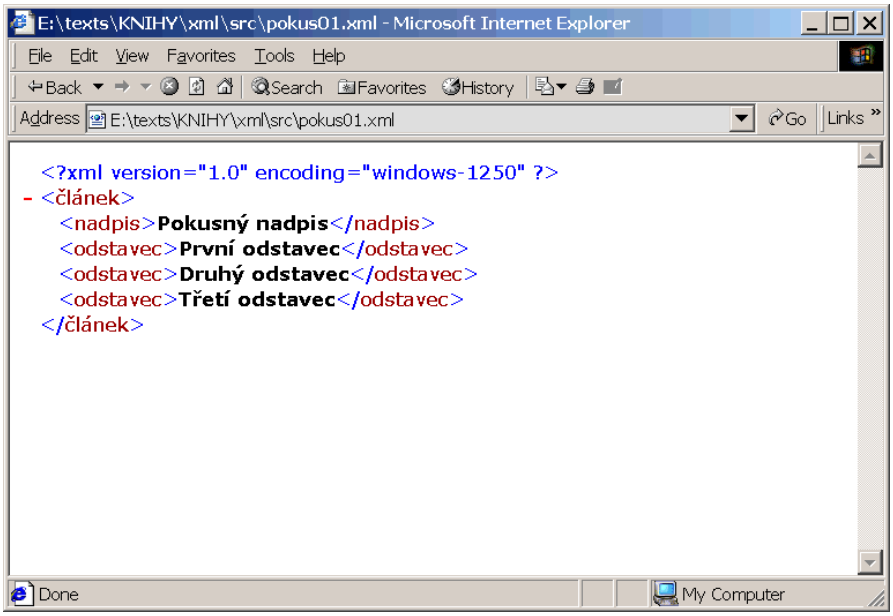
I když takto napevno určíme kódování dokumentů, nijak se tím nesníží jejich přenositelnost. Většina parserů totiž obě dvě kódování podporuje.

[4] I když třeba Windows 2000 obsahují novou verzi Notepadu (Poznámkového bloku), která již umí pracovat s dokumenty v kódování UTF-8.

Zkusíme si vytvořit jednoduchý XML dokument a zkontrolujeme ho pomocí parseru v prohlížeči.

```
<?xml version="1.0" encoding="windows-1250"?>
<článek>
  <nadpis>Pokusný nadpis</nadpis>
  <odstavec>První odstavec</odstavec>
  <odstavec>Druhý odstavec</odstavec>
  <odstavec>Třetí odstavec</odstavec>
</článek>
```

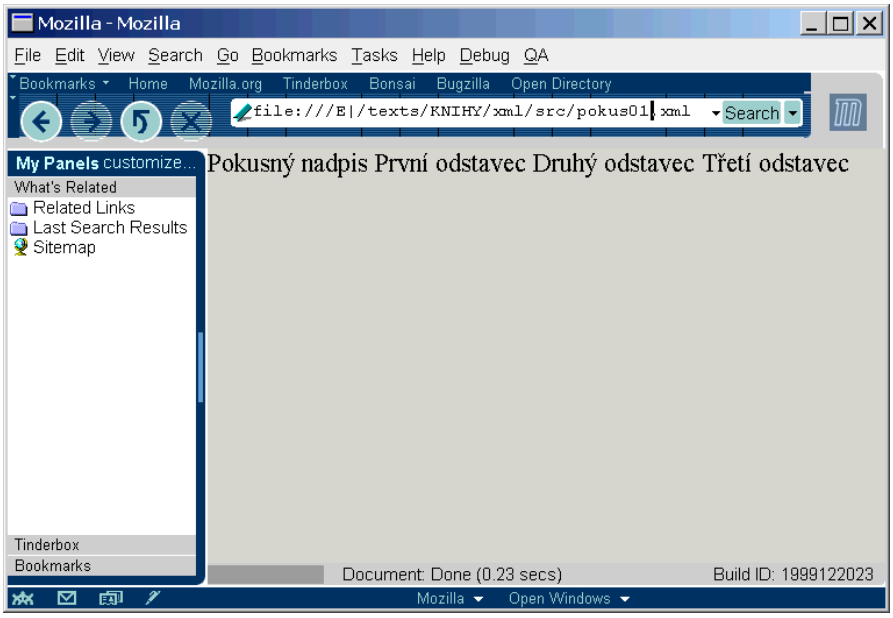
Pokud dokument neobsahuje žádné chyby, měl by se bez problémů zobrazit v prohlížeči (viz obrázky 2-3 a 2-4 na následující straně). Zobrazení v Internet Exploreru je o něco názornější celý XML dokument je znázorněn hierarchicky. To, že dokumenty nejsou zobrazeny zrovna hezky, je způsobeno tím, že k nim není připojen styl, který by definoval jejich vzhled. Této problematice se ještě budeme podrobně věnovat.



Obr. 2-3: *Zobrazení správně strukturovaného XML dokumentu v Internet Exploreru*

Podívejme se teď na to, jak prohlížeč zareaguje v případě chyby v XML dokumentu. Upravíme náš pokusný dokument tak, aby obsahoval chybu místo ukončovacího tagu `</nadpis>`, použijeme `</nadpos>`.

```
<?xml version="1.0" encoding="windows-1250"?>
<článek>
```



Obr. 2-4: Zobrazení správně strukturovaného XML dokumentu v Mozille

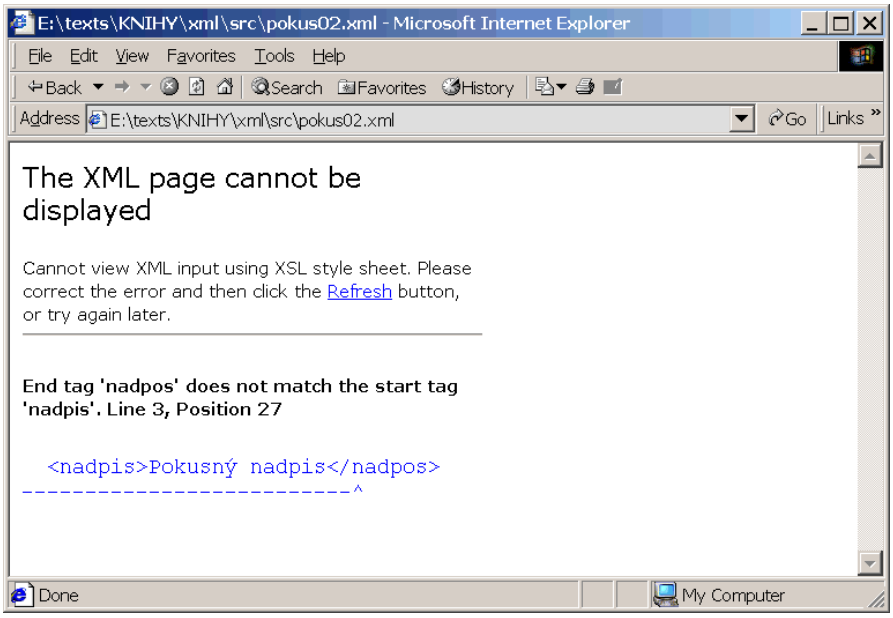
```
  <nadpis>Pokusný nadpis</nadpos>
  <odstavec>První odstavec</odstavec>
  <odstavec>Druhý odstavec</odstavec>
  <odstavec>Třetí odstavec</odstavec>
</článek>
```

V tomto případě prohlížeč dokument nezobrazí, ale vypíše chybové hlášení, ze kterého lze většinou poměrně dobře zjistit, kde máme v dokumentu chybu. Prohlížeč označí místo výskytu chyby a chybu stručně popíše (viz obrázky 2-5 na následující straně a 2-6 na následující straně).

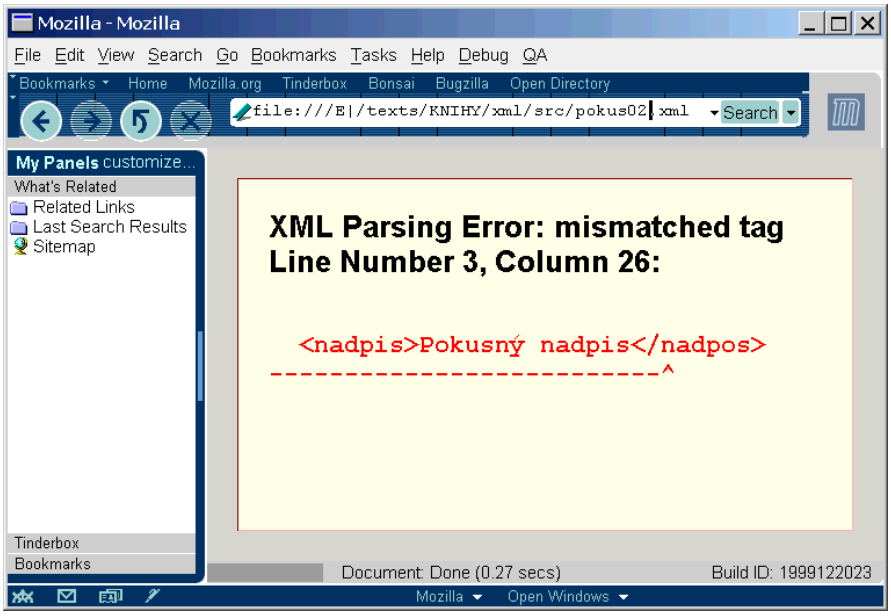
Pokud v dokumentu máme chybu, stačí ji v textovém editoru opravit a v prohlížeči pomocí tlačítka Obnovit (Reload) načíst stránku znovu. To je stejné jako při vytváření a ladění běžných HTML stránek.

Pro zajímavost si ještě ukážeme, jak to dopadne, když zapomene v dokumentu nastavit kódování pomocí XML deklarace. Pokud dokument obsahuje nějaké české znaky, parser z toho bude zmaten, protože bude dokument chybně interpretovat. Na obrázku 2-7 na straně 34 vidíme, jak dopadne zobrazní následujícího dokumentu bez XML deklarace v prohlížeči.

```
<článek>
  <nadpis>Pokusný nadpis</nadpis>
  <odstavec>První odstavec</odstavec>
  <odstavec>Druhý odstavec</odstavec>
```



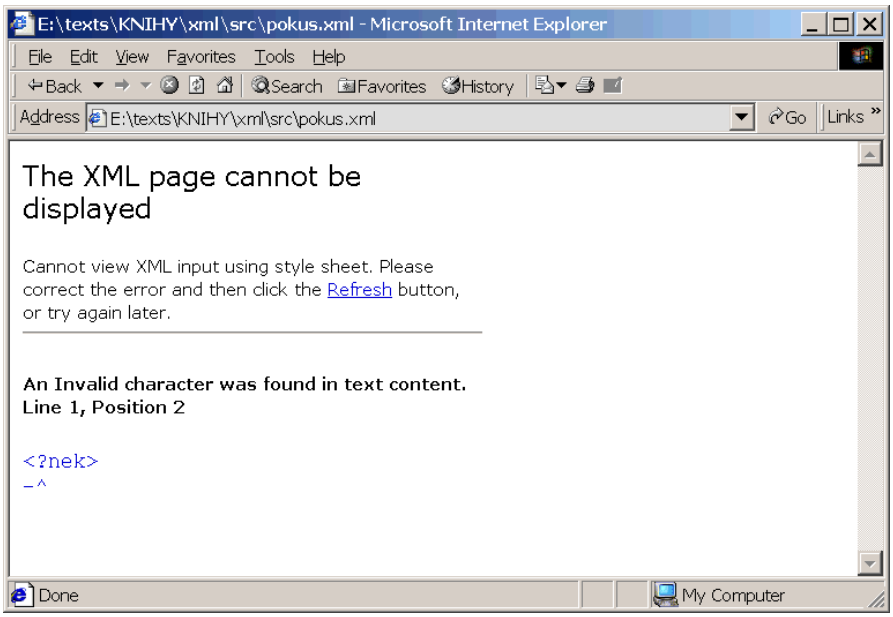
Obr. 2-5: Zobrazení chyby v XML dokumentu v Internet Exploreru



Obr. 2-6: Zobrazení chyby v XML dokumentu v Mozille

```
    <odstavec>Třetí odstavec</odstavec>
</článek>
```

Pomocí prohlížeče s podporou XML si můžete vyzkoušet všechny vlastnosti jazyka, které probereme dále v této kapitole.



Obr. 2-7: *Pokud v XML deklaraci neurčíme použité kódování a použijeme v dokumentu české znaky, nedopadne to zrovna dobře*

2.4 Více o syntaxi

Komentáře

Pokud potřebujeme v dokumentu něco vysvětlit nebo část textu dočasně skrýt, s výhodou k tomu použijeme komentář. Komentář je součástí dokumentu, ale parsery jej ignorují a není dále zpracováván. Komentář se zapisuje mezi znaky `<!--` a `-->`.

```
<!-- Vysvětlující text -->
```

Komentář může obsahovat cokoliv, kromě posloupnosti znaků `--`. V komentáři dokonce můžeme používat tagy atd. Jsou však zcela ignorovány. To se hodí pro dočasné vyřazení části dokumentu ze zpracování.

```
<para>První odstavec.</para>
<!-- <para>Šéf mi leze krkem.</para> -->
<para>Třetí odstavec.</para>
```

Komentáře se mohou vyskytovat v podstatě kdekoliv, ale nesmějí být součástí ostatního značkování. Nemůžeme tedy například okomentovat atribut:

```
<para <!--align="left"-->>První odstavec.</para>
```

Sekce CDATA

Pokud potřebujeme do dokumentu vložit větší kus textu, kde se hojně používají znaky se speciálním významem jako '<', '>' a '&', je nepohodlné je zapisovat pomocí znakových entit. Sekce CDATA oceníme zejména v případech, kdy je součástí XML dokumentu kód nějakého programu nebo HTML či XML kód. Použití sekcí CDATA si ukážeme na následujícím dokumentu.

```
<script language="JavaScript">
<![CDATA[
  for (i=0; i < 10; i++)
  {
    document.writeln("<p>Ahoj</p>");
  }
]]>
</script>
```

Bez použití sekce CDATA by byl zápis přece jen poněkud krkolomný.

```
<script language="JavaScript">
  for (i=0; i &lt; 10; i++)
  {
    document.writeln("&lt;p&gt;Ahoj&lt;/p&gt;");
  }
</script>
```

Obecně se tedy sekce CDATA zapisují jako `<![CDATA[`*«text»*`]]>`. Text přitom může obsahovat cokoliv, kromě sekvence znaků `]]>`. Konstrukce CDATA existuje v XML pro větší pohodlí autorů, kteří zapisují XML kód ručně.

Instrukce pro zpracování

XML dokumenty mohou být zpracovávány různými programy. Někdy může být užitečné do dokumentu uložit řídící informace, které jsou určeny pouze pro některý program. Můžeme tak do dokumentu zařadit odkaz na styl definující zobrazení v prohlížeči, formátovacímu programu můžeme naznačit, kde má zalomit

stránku. Moderní skriptové jazyky pro generování dynamických webových stránek se také zapisují přímo do těla dokumentů. Pro všechny tyto účely má XML k dispozici standardní způsob pro zařazení nestandardních informací. Na libovolné místo dokumentu (kromě značkovaní podobně jako u komentářů) můžeme vložit *instrukce pro zpracování (processing instructions)*. Tyto instrukce XML parser ignoruje, předá je nadřazené aplikaci záleží na ní, zda je nějak využije. Syntaxe instrukcí je velice jednoduchá.

`<?`*«identifikátor» «data»*`?>`

Pomocí *«identifikátoru»* můžeme rozlišovat jednotlivé druhy instrukcí do jednoho dokumentu můžeme umístit instrukce pro několik různých programů. Samotná *«data»* instrukce mohou mít libovolný tvar, ale nesmějí obsahovat sekvenci znaků `?>`.

Pomocí instrukcí pro zpracování lze do dokumentů zařadit například příkazy mého oblíbeného skriptovacího jazyka PHP.

```
<dokument>
  <datum>Dnešní datum je <?php echo Date("d.m.Y")?></datum>
  <para>Nějaké důležité informace.</para>
</dokument>
```

Pomocí instrukcí pro zpracování se k XML dokumentu připojují i styly, definující zobrazení v prohlížeči.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet href="styl.css" type="text/css"?>
<clanek>
  <zahlavi>
    <rubrika>Téma týdne</rubrika>
    <nazev>XML a stylové jazyky</nazev>
    <autor>Jiří Kosek</autor>
  </zahlavi>
  ...
</clanek>
```

Instrukce, jejichž identifikátory začínají na `xml` nebo na `XML`, jsou vyhrazeny pro další standardizaci jazyka XML a souvisejících standardů.

Využití instrukcí může být opravdu různorodé. Některé editory si pomocí nich zaznamenávají pozici kurzoru v době uložení souboru nebo třeba požadované místo zlomu stránky.

```
<dokument>
  ...
  <para>Odstavec, ve kterém skončila editace <?Pub Caret?></para>
  ...
```

```
  <?DTPLayout Force-Page-Break?>
  <para>Tohle chceme mít až na další stránce.</para>
  ...
</dokument>
```

2.5 DTD , Definice typu dokumentu

V úvodu jsme si řekli, že XML dokument může vyhovovat určitému typu dokumentu. Definice typu dokumentu (DTD) přitom říká, které elementy a atributy můžeme v dokumentu použít. Navíc je zde definováno, v jakých vzájemných vztazích mohou být jednotlivé elementy použity. DTD je tedy užitečný nástroj, který nám umožní hlídat, zda mají naše dokumenty správnou strukturu. Ve světě se používá mnoho DTD, které vyhovují různým požadavkům. Mezi jedno z nejznámějších patří například DTD DocBook, které definuje elementy a atributy vhodné pro značkování technické dokumentace.

Tím, že naše dokumenty založíme na určitém DTD, získáme hned dvě výhody. Jednak můžeme pomocí parseru kontrolovat, zda má náš dokument strukturu odpovídající danému DTD. Druhá výhoda je patrná při použití standardních DTD jako HTML nebo DocBook k dispozici budeme mít mnoho užitečných a jednoúčelových nástrojů navržených pro konkrétní DTD. Například není problém sehnat pro DocBook definici stylů vhodných pro formátování dokumentace či programy, které umí dokumenty DocBooku konvertovat do HTML a dalších formátů.

Pokud chceme využít výhody DTD, musíme použité DTD určit pomocí *deklarace typu dokumentu* (DOCTYPE), která se umísťuje na samotný začátek dokumentu, ihned za XML deklaraci. Obvykle je DTD uloženo v samostatném souboru, aby mohlo být opakovaně využíváno mnoha dokumenty. V tomto případě má deklarace tvar

```
<!DOCTYPE «kořenový element» SYSTEM "«URL»">
```

«URL» přitom udává adresu nebo jméno souboru, ve kterém je uloženo DTD. *«Kořenový element»* je jméno elementu, ve kterém bude obsažen celý dokument (instance DTD).

Pro některá běžně používaná a standardizovaná DTD je zbytečné, aby si parser a další aplikace četly DTD vždy ze sítě. Mnohem logičtější by bylo, aby v systému byla přítomna lokální kopie souborů s DTD. To je v XML možné pomocí takzvaných veřejných identifikátorů. Pro označení DTD použijeme nějaký textový řetězec. XML aplikace pak ve svém konfiguračním souboru zjistí, ve kterém souboru je uloženo příslušné DTD. Místo slova `SYSTEM` nyní použijeme výraz `PUBLIC`, za kterým uvedeme identifikátor DTD. Nakonec však stejně musíme připojit URL, které ukazuje na soubor s DTD, aby mohla aplikace

získat DTD i v případě, že nerozpozná veřejný identifikátor. Deklarace typu dokumentu pak může dopadnout třeba takto:

```
<!DOCTYPE book PUBLIC "-//Norman Walsh//DTD DocBk XML V3.1.4//EN"
                      "docbookx.dtd">
```

Pokud chceme dokumenty publikovat na Internetu, je lepší umístění DTD určit pomocí absolutního URL:

```
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
                     "http://www.wapforum.org/DTD/wml_1.1.xml">
```

DTD můžeme umístit i přímo do dokumentu pomocí následujícího zápisu

```
<!DOCTYPE kniha [
   «... DTD ...»
]>
```

Umístění DTD přímo do dokumentu není příliš časté, protože pak ztrácíme možnost sdílení jednoho DTD mezi několika dokumenty. Užitečnější je však kombinace obou předešlých metod a externí DTD upravit pomocí lokálně uvedených definic. Lokální část DTD se zpracovává ještě před tou externí a může tedy změnit některé definice uložené v externím DTD.

```
<!DOCTYPE kniha SYSTEM "kniha.dtd" [
   «... definice, které změní nebo rozšíří kniha.dtd ...»
]>
```

Když víme, jak připojit DTD k dokumentu, můžeme se podívat na to, jak samotné DTD vypadá. DTD obsahuje deklarace čtyř typů:

- deklarace elementů;
- deklarace atributů;
- deklarace entit;
- deklarace notací.

Deklarace elementů

Deklarace nového elementu je velice jednoduchá. Má následující tvar:

```
<!ELEMENT «název elementu» «obsah elementu»>
```

Název elementu musí začínat písmenem. Další znaky názvu mohou obsahovat písmena, číslice a některé speciální znaky jako ‘.’, ‘-’, ‘_’ a ‘:’. Nejsme omezeni jen na písmena anglické abecedy, jak bývá v počítačových systémech zvykem. V názvech můžeme klidně používat diakritiku, azbuku nebo třeba hebrejštinu.[5]

[5] Parser Microsoftu v rozporu se specifikací XML tyto znaky v názvech považuje za chybu.

Délka jména není nikterak omezená. Narozdíl od HTML a SGML je důležitá velikost písmen. Následující dva řádky deklarují dva různé elementy `kapitola` a `Kapitola`.

```
<!ELEMENT kapitola ...>
<!ELEMENT Kapitola ...>
```

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 HTML. Jejich deklarace by vypadala následovně:

```
<!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/>`. Není však možný zápis samotného `<br>`, protože by parser zcela marně hledal ukončovací tag `</br>`.

Kromě výše zmíněných případů se prázdné elementy používají například pro vkládání obrázků. Pomocí atributů elementu pak určíme soubor, ve kterém je obrázek uložen.

Protipólem k `EMPTY` je `ANY`. Toto klíčové slovo nám zajistí, že element může obsahovat libovolné další elementy 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.

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

Většinou máme na vnořené elementy mnohem striktnější požadavky a s `EMPTY` a `ANY` nevystačíme. V tomto případě pak použijeme tzv. *modelovou skupinu (model group)*. Modelová skupina se používá pro definici elementů, které obsahují další elementy nebo mají smíšený obsah (obsahují již přímo text a elementy).

Modelová skupina je vždy uzavřena 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:

```
<!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:

```
<!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 modelovou skupinu `(a, b, (c | d))`.

Kromě pořadí elementů musíme určit jejich počet, zda jsou povinné či zda se mohou opakovat. Pokud v modelové skupině uvedeme pouze jméno elementu, musí být element přítomen právě jednou. Pokud je 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?)>
```

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:

```
<!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.

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

Vše můžeme podle potřeby kombinovat. Při troše fantazie lze obejít i to, že nemůžeme specifikovat přesný počet opakování určitých elementů. Pokud chceme například vyjádřit, že seznam obsahuje alespoň dvě položky, můžeme použít modelovou skupinu `(polozka, polozka+)`.

Indikátor počtu výskytů můžeme připojit i za modelovou skupinu. 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:

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

Pokud může element obsahovat jak text, tak elementy, říkáme, že má *smíšený obsah (mixed content)*. 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:

```
<!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>
```

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:

<!ATTLIST *«jméno elementu» «deklarace atributů»*>

Deklarace jednotlivých atributů se skládá ze tří částí. První částí je jméno atributu. Pro jména atributů platí stejná omezení jako pro jména elementů. Atributy jsou rovněž citlivé na velikost písmen, a tak `MujAtribut` a `MUJATRIBUT` jsou dva zcela rozdílné atributy.

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:

```
<!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.

```
<!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.

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. Podrobněji si o využití těchto typů povíme v částí věnované odkazům mezi XML dokumenty.

Další dva typy jsou `ENTITY` a `ENTITIES`, které použijeme v případech, kdy atribut obsahuje jméno (resp. jména) entity. Entitami se v této kapitole budeme zabývat samostatně a jejich použití v atributech si ještě ujasníme.

Dalším typem atributu je `NOTATION`. V praxi se však tento typ moc často nevyužije. Atributy typu `NOTATION` slouží k určení typu dat elementu, pokud tato data nejsou ve formátu XML.

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

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

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

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. XML editor si může zadání hodnoty atributu vynutit přímo při vložení elementu, který povinný atribut obsahuje.

Opačným případem je situace, kdy můžeme atribut vynechat, a v tomto případě si jeho hodnotu podle svých potřeb domyslí sama konkrétní aplikace, která právě s dokumentem pracuje. V těchto případech se použije klíčové slovo `#IMPLIED`.

Další možností je specifikování standardní hodnoty. Stačí ji uvést. Pokud chceme, aby byl obrázek zarovnán vlevo, pokud není určeno jinak, použijeme deklaraci:

```
<!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. Na první pohled to vypadá nelogicky a neužitečně, ale opravdu existují situace, kdy se to hodí.

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.

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

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

Deklarace elementů a atributů je základem každého DTD. DTD může obsahovat ještě další prvky, i když to už není tak časté. Pokud navrhujeme nějaká složitější DTD, mohou se nám hodit tzv. *parametrické entity*, které umožňují zkrátit a zefektivnit zápis.

Parametrické entity

V DTD často opakujeme stejné sekvence textu. V praxi např. mívají všechny elementy několik společných atributů. Je pak zbytečné znovu a znovu opisovat deklaraci stejných atributů. Práci nám mohou usnadnit parametrické entity. Pomocí zápisu:

```
<!ENTITY % «entita» "«nějaký text»">
```

definujeme parametrickou entitu, na kterou se v DTD budeme odvolávat pomocí zápisu `%`*«entita»*`;`. Použití parametrické entity z funkčního hlediska odpovídá napsání celého textu, který entita zastupuje, šetří se však práce. Následuje ukázka deklarace parametrické entity `%spolecne-atributy;` a jejího využití.

```
<!ENTITY % spolecne-atributy
           "jazyk   CDATA   #IMPLIED
            utajeni (verejne|tajne|prisne-tajne) 'verejne'">

<!ELEMENT clanek     (nazev, autor?, p*)>
<!ATTLIST clanek     %spolecne-atributy;>
<!ELEMENT nazev      (#PCDATA)>
<!ATTLIST nazev      %spolecne-atributy;>
<!ELEMENT autor      (#PCDATA)>
<!ATTLIST autor      %spolecne-atributy;>
<!ELEMENT p          (#PCDATA)>
<!ATTLIST p          %spolecne-atributy;
          id         ID      #REQUIRED>
```

V ukázkovém DTD deklarujeme elementy `clanek`, `nazev`, `autor` a `p`. Všechny elementy přitom mají atributy `jazyk` a `utajeni`. Element `p` má navíc ještě atribut `id`. Kdybychom nevyužili parametrické entity, byl by zápis mnohem delší a jeho pozdější modifikace by byla náročnější.

```
<!ELEMENT clanek (nazev, autor?, p*)>
<!ATTLIST clanek
          jazyk   CDATA   #IMPLIED
          utajeni (verejne|tajne|prisne-tajne) 'verejne'>
```

```
<!ELEMENT nazev   (#PCDATA)>
<!ATTLIST nazev
          jazyk   CDATA   #IMPLIED
          utajeni (verejne|tajne|prisne-tajne) 'verejne'>
<!ELEMENT autor   (#PCDATA)>
<!ATTLIST autor
          jazyk   CDATA   #IMPLIED
          utajeni (verejne|tajne|prisne-tajne) 'verejne'>
<!ELEMENT p  (#PCDATA)>
<!ATTLIST p
          jazyk   CDATA   #IMPLIED
          utajeni (verejne|tajne|prisne-tajne) 'verejne'
          id      ID      #REQUIRED>
```

2.6 Entity

V mnoha případech není vhodné uložit celý dokument do jednoho souboru. Důvodů může být mnoho. Pokud píšeme nějakou delší knihu, je mnohem pohodlnější ukládat jednotlivé kapitoly do samostatných souborů. V textu jedné kapitoly se orientujeme snáze než v jednom dlouhém dokumentu. Dokument můžeme rozdělit klidně na ještě menší části. Výhoda pak může být v tom, že jeden velký dokument může editovat více uživatelů najednou každý edituje pouze jednu malou část. Rozdělení dokumentu na více částí se hodí i v případech, kdy se určitá část dat využívá v několika dokumentech zároveň. Nemusíme pak zbytečně informace kopírovat a přepisovat.

XML umožňuje informace obsažené v dokumentu rozdělit na části, kterým se říká *entity*. Každá entita má své jméno, pomocí kterého může být jednoznačně identifikována. Výjimku tvoří dokumentová entita (document entity), která odpovídá celému dokumentu. V praxi je dokumentová entita nejčastěji soubor, který se předává parseru ke zpracování. Buď jde o jedinou entitu, nebo obsahuje deklaraci dalších entit, ze kterých se celý dokument skládá.

XML podporuje několik druhů entit, které se liší vlastnostmi. Entity mohou reprezentovat buď data ve formátu XML, anebo v nějakém jiném formátu. Hovoříme pak o *textových entitách* a *binárních entitách*. Entity můžeme rozlišovat i podle toho, zda jsou uloženy přímo v hlavním dokumentu nebo v externím souboru. Dostáváme tak *interní entity* a *externí entity*. Tomuto rozdělení pak odpovídají i druhy entit, které můžeme používat interní textové entity, externí textové entity a externí binární entity. Interní binární entity nelze používat, protože bychom velice těžko uložili nějaká binární data přímo do XML dokumentu.

Definice entity

Entita musí být definována ještě předtím, než ji v dokumentu použijeme. Deklarace všech entit musí být uvedeny v DTD. Je jedno, zda budou v lokální části v deklaraci typu dokumentu, nebo v externím DTD.

```
<!DOCTYPE «kořenový element» «identifikátor DTD» [
  «deklarace entit»
]>
```

«Identifikátor DTD» je přitom buď veřejný, nebo systémový identifikátor DTD. Pokud DTD v dokumentu nevyužíváme, nebo je přímo součástí deklarace typu dokumentu, můžeme tuto část vynechat.

Deklarace entity má tvar:

```
<!ENTITY «název entity» ...>
```

Takto definovanou entitu pak můžeme použít v dokumentu pomocí *odkazu na entitu*, který má tvar:

```
&«název entity»;
```

Nyní se podíváme na to, jak lze jednotlivé druhy entit používat.

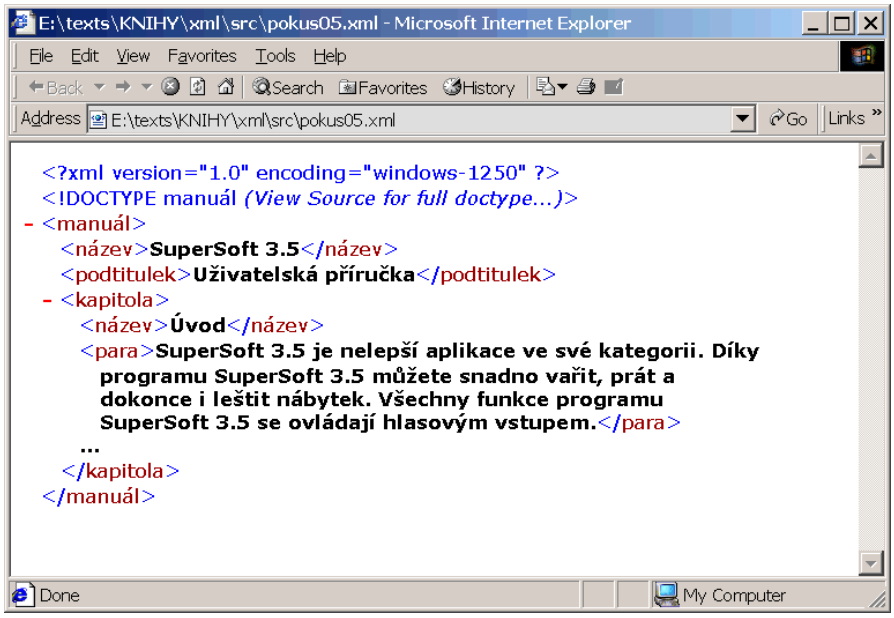
Interní textové entity

Tento druh entit nalezne uplatnění v případě, kdy v dokumentu často opakujeme nějaký text. Pokud například píšeme dokumentaci k nějakému programu a chceme, aby šlo snadno měnit název a verzi programu v celém dokumentu, je to optimální úkol pro interní textové entity. Deklarace interní textové entity je velice jednoduchá:

```
<!ENTITY «název entity» "«text»">
```

Praktické využití textových entit přináší následující ukázka.

```
<?xml version="1.0" encoding="windows-1250"?>
<!DOCTYPE manuál [
<!ENTITY program "SuperSoft 3.5">
]>
<manuál>
  <název>&program;</název>
  <podtitulek>Uživatelská příručka</podtitulek>
  <kapitola>
    <název>Úvod</název>
    <para>&program; je nelepší aplikace ve své kategorii. Díky
      programu &program; můžete snadno vařit, prát a dokonce i leštit
      nábytek. Všechny funkce programu &program; se ovládají hlasovým
```



Obr. 2-8: Odkaz na textovou entitu je automaticky nahrazen jejím obsahem

```
      vstupem.</para>
    ...
  </kapitola>
</manuál>
```

Textová entita nemusí obsahovat jen text. Může klidně obsahovat značkování i odkazy na další entity. Předpokládejme, že máme k dispozici entitu `trade`, která obsahuje text obchodní značky (). Dále chceme, aby se název našeho programu zobrazoval a tisknul zvýrazněný. Následující ukázka naznačuje, jak lze používat odkazy na entity a tagy uvnitř deklarace entity.

```
<!ENTITY trade "<sup>TM</sup>">
<!ENTITY program "<tučně>SuperSoft 3.5&trade;</tučně>">
```

Pokud omylem nebo úmyslně deklarujeme několik entit se stejným jménem, bude platná pouze ta první, ostatní deklarace budou ignorovány.

Externí textové entity

Princip použití externích textových entit je stejný jako u interních. Rozdíl je v tom, že při deklaraci nespecifikujeme přímo text, který entita zastupuje, ale

pouze odkážeme na soubor, který požadovaná data obsahuje. To je výhodné v případech, kdy jsou data, vkládaná pomocí entity, poměrně dlouhá nebo když chceme jednu externí entitu použít ve více dokumentech.

Při deklaraci entity typicky určujeme URL adresu souboru, který text entity obsahuje.

```
<!ENTITY «název entity» SYSTEM "«URL»">
```

Jelikož lze používat i relativní URL a entity jsou obvykle ve stejném adresáři jako hlavní dokument, stačí jako *«URL»* použít jméno souboru.

```
<!ENTITY kapitola1 SYSTEM "kap1.xml">
```

Pokud pak v dokumentu někde použijeme odkaz na entitu `&kapitola1;` je to totéž, jako bychom na toto místo vložili soubor `kap1.xml`.

```
<?xml version="1.0" encoding="windows-1250"?>
<!DOCTYPE manuál [
<!ENTITY program "SuperSoft 3.5">
<!ENTITY kapitola1 SYSTEM "kap1.xml">
]>
<manuál>
  <název>&program;</název>
  <podtitulek>Uživatelská příručka</podtitulek>
  &kapitola1;
</manuál>
```

V souboru `kap1.xml` nesmíme zapomenout na prvním řádku uvést deklaraci použitého kódování, pokud je jiné než UTF-8 nebo UTF-16.

```
<?xml encoding="windows-1250"?>
<kapitola>
  <název>Úvod</název>
  <para>&program; je nelepší aplikace ve své kategorii. Díky
    programu &program; můžete snadno vařit, prát a dokonce i leštit
    nábytek. Všechny funkce programu &program; se ovládají hlasovým
    vstupem.</para>
  ...
</kapitola>
```

Při deklaraci entity můžeme kromě systémového identifikátoru uvedeného za `SYSTEM` použít i veřejný identifikátor (`PUBLIC`), podobně jako při deklaraci typu dokumentu.

Binární entity

Binární entity mohou být deklarovány pouze jako externí. Jejich deklarace je obdobná jako u externích textových entit. Za veřejným nebo systémovým iden-

tifikátorem je však potřeba uvést typ dat binární entity. Pomocí tohoto typu (tzv. notace) pak aplikace pozná, jak má s uvedenými daty naložit. Deklarace může vypadat např. takto:

```
<!ENTITY logo SYSTEM "/obr/logo.eps" NDATA "EPS">
```

Deklarovali jsme entitu `logo`, která odpovídá souboru `/obr/logo.eps`. Formát dat je `EPS`. Aplikace sice může odhadnout typ dat podle přípony, ale to není vždy spolehlivé.

Takto definovanou entitu nemůžeme přímo vložit do textu dokumentu ani by to nemělo smysl. Můžeme na ní však odkázat pomocí atributu, který má typ `ENTITY` nebo `ENTITIES`.

```
Od roku 1999 má naše firma logo, které je na obrázku <picture name="logo"/>.
```

Zabudované entity

Jak už víme, máme v každém XML dokumentu k dispozici pět předem definovaných entit `<` (<), `>` (>), `&` (&), `&quot;` (") a `&apos;` (').

Pokud dokument používá DTD, je lepší z důvodu kompatibility se staršími aplikacemi tyto entity deklarovat. Poslouží k tomu následující fragment kódu:

```
<!ENTITY lt     "&#38;#60;">
<!ENTITY gt     "&#62;">
<!ENTITY amp    "&#38;#38;">
<!ENTITY apos   "&#39;">
<!ENTITY quot   "&#34;">
```

Znakové entity

Na straně 29 jsme se zmínili o možnosti zapsat libovolný znak pomocí jeho kódu. Jedná se také o speciální druh entity.

Parametrické entity

Již dříve jsme si řekli, že v DTD můžeme používat parametrické entity. Deklarace se odlišuje použitím znaku `%` před názvem entity.

```
<!ENTITY % «název entity» ...>
```

V dokumentu můžeme používat běžnou i parametrickou entitu se stejným názvem, protože se odlišují odkazy na ně: `&entita;` a `%entita;`.

Parametrické entity mohou být i externí. Často se používají pro načtení definic znakových entit, sdílených v mnoha DTD. Složitější DTD bývají uložena ve více souborech a dohromady se skládají také pomocí externích parametrických entit.

```
<!ENTITY % ISOpub PUBLIC "ISO 8879:1986//ENTITIES Publishing//EN//XML"
                         "ent/iso-pub.ent">
%ISOpub;
```

V dokumentu pak můžeme používat entity pro speciální znaky. Naše ukázka pracovala se souborem definujícím mnoho užitečných znaků pro sazbu textu. Například `&mdash;` pro pomlčku (), `&ndash;` pro krátkou pomlčku () a `&dash;` pro spojovník (-).

2.7 Kontrola dokumentu podle DTD

Výhoda ukládání dokumentů v XML spočívá především v možnosti zachytit pomocí elementů strukturu dokumentu. Už víme, že základním nástrojem, který se při práci s XML dokumenty používá, je parser. Parser je program, který kontroluje, zda je dokument správně strukturovaný (well-formed). Lepší parsery zároveň kontrolují, zda dokument odpovídá danému DTD (samozřejmě jen pokud DTD pro dokument existuje). Vznikají i další způsoby, jak popsat pravidla pro určitou skupinu dokumentů (např. XML schémata). Pro ně samozřejmě také existují parsery.

Specifikace XML definuje několik úrovní správnosti dokumentu. Pokud dokument splňuje základní syntaktická pravidla, jako párové tagy apod., je *správně strukturovaný*. Pokud dokument obsahuje DTD nebo je na něj odkaz v deklaraci typu dokumentu, parser kontroluje, zda dokument odpovídá DTD. Pokud ano, říkáme, že dokument je *validní*. Pokud dokument porušuje pravidla daná v DTD, je *invalidní*. To, že je dokument invalidní, se nevylučuje s tím, že je správně strukturovaný. Zároveň také platí, že validní dokument musí být správně strukturovaný.

Parsery dnes existují ve dvou podobách. Tou první jsou knihovny pro různé programovací jazyky, které můžeme využívat v našich programech. Parser nám pak kromě detekce chyb v dokumentu umožní velmi snadné čtení dat z XML dokumentů. K tomu, jak parser využít z pohledu programátora, se ještě vrátíme v sekci *Parsery* na straně 140. Pro nás je nyní důležité, že většina parserů existuje i v podobě jednoduchých programů, na jejichž vstup předáme XML dokument a na výstupu obdržíme případný přehled chyb v dokumentu. Můžeme také využít parser zabudovaný přímo v prohlížeči s podporou XML.

Parser využijeme zejména v situaci, kdy chceme ověřit, zda náš dokument neobsahuje nějaké syntaktické a strukturní chyby. Tato situace může nastat v případě, kdy dokument vytváříme v editoru, který nemá podporu XML a umožní nám vytvořit chybný dokument. Dnes je k dispozici mnoho parserů napsaných v různých programovacích jazycích. Velice populární platformou pro psaní parserů je jazyk Java. Z hlediska výkonu jsou však dnes lepší parsery napsané v jazyce C++. Použitý jazyk nás však jako uživatele nemusí až zase tolik zajímat, důležitá je funkčnost.

Nyní si ukážeme, jak se používají některé nejrozšířenější parsery. Pro naše pokusy si můžeme DTD ze strany 43 uložit do souboru `clanek.dtd`. Do souboru `test.xml` si pak uložíme dokument vyhovující našemu DTD.

```
<?xml version="1.0" encoding="windows-1250"?>
<!DOCTYPE clanek SYSTEM "clanek.dtd">
<clanek>
<nazev>O myších bez lidí</nazev>
<p>Myši možná nejsou tak...</p>
<p>Myšlenka, že myši ovládají...</p>
</clanek>
```

Parser SP

Za jeden z nejlepších parserů se dnes považuje parser SP od Jamese Clarka. SP je k dispozici zdarma včetně zdrojových textů, pracuje ve Windows i na Unixu (dokonce existuje i verze pro MS-DOS). Jeho kvality dokládá i to, že se stal základem mnoha komerčních parserů, které jsou prodávané za velké peníze. SP byl původně parser vyvinutý pro jazyk SGML, ale od verze 1.3 podporuje i XML. Pokud ještě nemáte parser nainstalován, postupujte podle návodu v příloze *Parser SP* na straně 152. Pomocí příkazu

```
nsgmls -wxml -s test.xml
```

parser spustíme. Pokud je náš dokument správně strukturovaný a vyhovuje zadanému DTD, neohlásí parser žádnou chybu. Zkuste nyní v dokumentu záměrně udělat nějakou chybu (zapomeňte ukončovací tag, překřižte tagy, použijte neexistující atribut apod.) a uvidíte, jak vám o tom podá `nsgmls` zprávu.

Pokud máme XML dokument, ke kterému neexistuje DTD, můžeme jej pomocí `nsgmls` také zkontrolovat. Stačí použít parametr `-wno-valid` a u dokumentu se pouze zkontroluje, zda je správně strukturovaný.

Parser od Microsoftu

Narozdíl od jiných oblastí, Microsoft v oblasti XML akceptuje ve svých produktech standardy XML. Parser od MS je distribuován jako COM komponenta. V příloze *Parser od Microsoftu* na straně 152 je popsáno, jak si vyrobit dávkový soubor pro spouštění parseru z příkazové řádky. Dokument pak můžeme zkontrolvat pomocí příkazu:

```
msxml test.xml
```

2.8 XML pro fajnšmekry

V předchozí části kapitoly jsme popsali ty nejpodstatnější a nejpoužívanější části standardu XML. Nyní se podíváme na několik dalších vlastností XML, o kterých jsme se dosud nezmínili.

XML deklarace

Zatím jsme se zmínili o tom, že v XML deklaraci lze používat dva pseudoatributy `version` a `encoding`. K dispozici je však ještě třetí pseudoatribut `standalone`. Pomocí něj můžeme specifikovat, zda může parser, který nekontroluje DTD, dokument zpracovat i bez čtení DTD uloženého v externím souboru. Někdy může DTD obsahovat deklarace entit, bez nichž by nebylo možné dokument korektně interpretovat.

Pokud lze dokument číst i bez externího DTD, použije se v deklaraci `standalone=`yes. Pokud jsou některé deklarace entit v externím DTD, měli bychom v dokumentu použít `standalone=`no. Pokud `standalone` nepoužijeme, je to totéž, jako kdybychom použili hodnotu `no`.

```
<?xml version="1.0" encoding="iso-8859-2" standalone="yes"?>
```

I když jednotlivé parametry XML deklarace vypadají jako atributy elementů, nemají s nimi nic společného. Narozdíl od atributů elementů záleží u parametrů XML deklarace na jejich pořaí. Jediné správné pořadí je `version`, `encoding` a `standalone`.

Prolog

V souvislosti s XML dokumentem se velice často hovoří o jeho *prologu*. Prolog je souhrnné označení pro XML deklaraci a deklaraci typu dokumentu.

Podmíněné sekce

Některé části DTD občas potřebujeme mít pro jeden dokument v několika variantách, podle toho, co chceme s dokumentem dělat. XML proto nabízí snadný způsob, jak některé deklarace vyřadit, nebo naopak zařadit do zpracování. Důležité je, že tento způsob lze použít pouze v DTD a že DTD musí být v externím souboru (nemůže být součástí deklarace typu dokumentu).

Pokud chceme, aby se nějaké deklarace zpracovaly, můžeme použít zápis:

```
<![INCLUDE[
  «deklarace»
```

```
]]>
```

Význam je tedy stejný, jako kdybychom použili samostatně *«deklarace»*. Pokud chceme, aby se část deklarací ignorovala, použijeme:

```
<![IGNORE[
  «deklarace»
]]>
```

Podmíněné sekce se často kombinují s parametrickými entitami, čímž získáme poměrně efektivní a mocný nástroj, kdy přepsáním jednoho slova z `IGNORE` na `INCLUDE` (nebo naopak) můžeme měnit DTD na mnoha místech najednou.

```
<!ENTITY % VolitelnéElementy "IGNORE">

<![%VolitelnéElementy;[
  ...
]]>
```

Notace

O notacích jsme již hovořili na dvou místech. Jednak jsme si říkali, že v deklaraci externí binární entity se určuje typ dat obsažených v entitě. Určení se provede připojením textu `NDATA` *«typ»* za identifikátor entity. *«Typ»* přitom předtím musí být definován jako notace. Druhý případ použití notací jsou atributy typu `NOTATION`. Jejich obsah musí být také předem definován jako notace. Deklarace notace je velice jednoduchá:

```
<!NOTATION «název» «identifikátor»>
```

«Název» je libovolný název, který si pro daný typ dat zvolíme. Pomocí systémového nebo veřejného identifikátoru pak odkazujeme na aplikaci, která umí nestandardní typ dat zpracovat.

```
<!NOTATION TeX SYSTEM "c:\tex\view.exe">
<!NOTATION EPS SYSTEM "c:\gstools\gsview.exe">
```

Použití notací však neznamená, že by například parser musel uvedené programy spustit. Je to spíše pomůcka, kterou mohou aplikace pracující s XML využít. Zvláště při sdílení dokumentů mezi různými systémy je více než pravděpodobné, že ne všude budou potřebné programy nainstalovány na stejném místě. Je potřeba brát tato omezení v úvahu. To je také jeden z hlavních důvodů, proč se notace v praxi příliš nepoužívají.

Určení jazyka

Ve vícejazyčných dokumentech může být pro další zpracování velice užitečné, když jednoznačně identifikujeme, jakým jazykem jsou jednotlivé části dokumentu zapsány. V XML pro tyto účely existuje vyhrazený atribut `xml:lang`.

Pokud ho použijeme u nějakého elementu, říkáme tím, že obsah elementu včetně vnořených elementů a atributů je v daném jazyce. Účinek atributu lze změnit opětovným použitím u vnořeného elementu.

```
<kapitola xml:lang="cs">
  <para>Celá kapitola je česky.</para>
  <para xml:lang="en">This is the only exception
                      because it is in English.</para>
  <para>Obsah předchozího elmentu byl anglicky.</para>
</kapitola>
```

Obsahem atributu `xml:lang` je kód jazyka podle RFC 1766. V praxi to znamená, že se používají jazykové kódy uvedené v příloze *Jazykové kódy podle ISO 639* na straně 156. Pro češtinu je to např. kód `cs`, pro slovenštinu `sk` a pro angličtinu `en`. Za kódem jazyka se může uvádět ještě subkód, který nejčastěji odpovídá kódu země (viz příloha *Kódy států podle ISO 3166* na straně 157). Pro angličtinu používanou ve Velké Británii je pak kód `en-GB` a pro americkou `en-US`.

Pokud dokument používá DTD, musíme v něm atribut `xml:lang` deklarovat. Jako jeho typ musíme použit `NMTOKEN`.

Mezery a konce řádků

Standard XML definuje, jak má parser předávat konce řádek dalším aplikacím. Na různých systémech se používají různé znaky pro určení konce řádky někde se používá kombinace znaků CR a LF, někde jen LF nebo CR. Parser by měl všechny konce řádek normalizovat na jeden znak LF (to je znak s kódem 10).

Pro přehledné formátování zdrojového textu v XML je dobré používat mezery. Kromě mezer lze samozřejmě používat i tabelátor a konce řádek. Souhrnně se těmto znakům říká bílé znaky (whitespaces). Ve značkování těchto znaků můžeme za sebou použít tolik, kolik chceme. Jejich vícenásobný výskyt bude nahrazen jednou mezerou. Následující dva zápisy jsou tudíž totožné.

```
<odstavec zarovnání="vlevo">
Text odstavce.
</ostavec>

<odstavec
                zarovnání        =          "vlevo"
>
Text odstavce.
</ostavec           >
```

Jak s bílými znaky naloží aplikace uvnitř textu záleží na ní. Pokud jsou však mezery a další znaky významné například ve zdrojovém textu programu, ve verších básně můžeme určit, že se mezery v žádném případě nemají vypouštět. Stačí

u elementu použít atribut `xml:space` a jako jeho hodnotu použít `preserve`. Pokud chceme, aby se použil standardní způsob pro zpracování bílých znaků, použijeme hodnotu `default`. Pokud dokument používá DTD, musíme v něm atribut `xml:space` deklarovat.

```
<!ATTLIST výpis xml:space (default|preserve) "preserve">
```

Struktura veřejných identifikátorů

Externí entity mohou být určeny pomocí veřejného identifikátoru. Na první pohled se může veřejný identifikátor jevit jako podivné uskupení slov, ale jeho struktura je zcela logická. Veřejné identifikátory se již nějaký čas používají ve světě SGML. My si význam jednotlivých částí veřejného identifikátoru popíšeme na následující ukázce:

```
-//OASIS//DTD DocBook V3.1//EN
```

První znak '`-`' znamená, že identifikátor není zaregistrován. Znamená to, že není zaručena jeho celosvětová jednoznačnost. Pro registrované identifikátory se na tomto místě používá znak '`+`'. Pokud je vlastníkem organizace ISO, uvádí se na začátku řetězec `ISO 8879:1986`.

Další část identifikátoru označuje organizaci nebo osobu, která je vlastníkem souboru označeného veřejným identifikátorem. Například pro HTML má identifikátor tvar:

```
-//W3C//DTD HTML 4.0 Transitional//EN
```

Na první pohled vidíme, že vlastníkem je konsorcium W3C. U DocBooku je to zase sdružení Oasis.

Za dalšími dvěma lomítky veřejného identifikátoru následuje určení typu souboru. V našem případě tam bude vždy DTD, protože odkazujeme na soubor s DTD. Poté následuje textové označení dokumentu například `DocBook V3.1` pro verzi 3.1 DocBooku. Za posledními dvěma lomítky je kód jazyka, ve kterém je DTD zapsáno. Nejčastěji se setkáme s kódem `EN`, který odpovídá angličtině.

Katalogy

Jelikož s XML dokumenty pracuje na jednom systému obvykle několik různých programů, existují standardní formáty pro zápis mapování z veřejných identifikátorů na soubory uložené na našem disku. Nazývají se katalogy a původem jsou ze světa SGML. V souborech, obvykle pojmenovaných `catalog`, byly ve velice jednoduché textové notaci uloženy všechny potřebné údaje. Aplikace obvykle tento soubor hledaly v aktuálním adresáři nebo na místech uložených v proměnné prostředí `SGML_CATALOG_FILES`.

Některé XML aplikace podporují SGML katalogy, některé nepodporují žádný obdobný mechanismus. Syntaxe SGML katalogů je založena na obyčejných textových souborech. Existuje návrh nového formátu katalogových souborů XCatalog, který je založen na XML. Otázkou je, jak jednotlivé aplikace budou katalogové soubory používat. Možná, že katalogy časem úplně zaniknou, protože připojení k Internetu je stále dostupnější a rychlejší a odpadne potřeba uchovávat lokální kopie DTD a dalších důležitých souborů.

2.9 Jmenné prostory

V některých případech může být velice užitečné v jednom dokumentu používat několik sad značek definovaných v různých DTD. Můžeme v XML vytvářet katalog knih podle nějakého DTD určeného pro tvorbu katalogů. Aby v tomto katalogu mohly snadno vyhledávat služby zaměřené na prohledávání bibliografických záznamů, můžeme názvy knih, autorů, ISBN a další důležité údaje navíc označit pomocí nějakého bibliografického DTD. Využívání sad již hotových značek má i další výhody. Nemusíme znovu vynalézat kolo, stačí poskládat dohromady již existující věci.

Jmenné prostory (XML namespaces) nabízejí možnost, jak v jednom dokumentu kombinovat více sad značek. Pod pojmem sada značek přitom máme na mysli elementy a atributy. Každá sada značek je jednoznačně definována svojí URI adresou.[6] Pokud chceme v nějakém elementu a jeho podelementech používat elementy patřící do určité sady značek, musíme si o to říct pomocí speciálního atributu `xmlns`.

`xmlns:`*«prefix»*`="`*«URI sady značek»*`"`

Pokud nyní chceme použít nějaký element nebo atribut, který patří do dané sady značek, použijeme navíc k jeho označení námi definovaný prefix *«prefix»*`:`*«element»* a *«prefix»*`:`*«attribut»*. Následující ukázka dohoromady kombinuje dvě sady značek.

```
<ceník:nabídka xmlns:ceník="http://www.ecena.cz/e-cenik"
               xmlns:bib="http://www.book.org/bibliography">
  <ceník:položka ceník:dph="22%">
```

[6] Použití URI není překlep. Ve skutečnosti totiž existují tři druhy adres URI, URL a URN. URI (Uniform Resource Identifier) je z nich nejobecnější, označuje identifikátor nějakého informačního zdroje. URL (Uniform Resource Locator) jednoznačně určuje umístění zdroje v počítačové síti. URN (Uniform resource Name) pak jednoznačně identifikuje zdroj informací. K jednomu URN může existovat několik URL, pokud je dokument k dispozici v několika kopiích na různých místech. URN je tedy vlastně adresa dokumentu nezávislá na jeho fyzickém umístění.

Používání URN není dnes ještě příliš časté, protože je technicky mnohem náročnější než použití URL. URN totiž vyžaduje nějakou překladovou službu, která bude převádět adresy mezi URN a URL.

```
      <ceník:název>
        <bib:book>
          <bib:author>Jiří Kosek</bib:author>
          <bib:title>HTML - tvorba dokonalých WWW stránek</bib:title>
        </bib:book>
      </ceník:název>
      <ceník:cena ceník:měna="CZK">259</ceník:cena>
    </ceník:položka>
  </ceník:nabídka>
```

Pokud jedna sada značek výrazně převažuje nad druhou, můžeme jeden ze jmenných prostorů deklarovat jako implicitní. Stačí vynechat jeho prefix.

```
<nabídka xmlns="http://www.ecena.cz/e-cenik"
         xmlns:bib="http://www.book.org/bibliography">
  <položka>
    <název>
      <bib:book>
        <bib:author>Jiří Kosek</bib:author>
        <bib:title>HTML - tvorba dokonalých WWW stránek</bib:title>
      </bib:book>
    <název>
    <cena>259</cena>
  </položka>
</nabídka>
```

Vidíme, že zápis je mnohem úspornější. Záměrně jsme v této ukázce nepoužili atributy `ceník:dph` a `ceník:měna`, protože implicitní jmenný prostor se vztahuje pouze na elementy, ne na atributy. Pokud chceme používat atributy z nějakého jménného prostoru, musíme mu přiřadit prefix.

Jmenné prostory mají platnost pro všechny podřazené elementy a mohou se navzájem přebíjet. Naši ukázku proto můžeme zapsat i úspornějí jako:

```
<nabídka xmlns="http://www.ecena.cz/e-cenik">
  <položka>
    <název>
      <book xmlns="http://www.book.org/bibliography">
        <author>Jiří Kosek<author>
        <title>HTML - tvorba dokonalých WWW stránek</title>
      </book>
    <název>
    <cena>259</cena>
  </položka>
</nabídka>
```

Pokud chceme, aby byl element chápán jako nepatřící do žádné sady značek, můžeme u něj použít atribut `xmlns` a nastavit ho na prázdný řetězec.

Jmenné prostory představují velice výkonný mechanimus a bohužel nemáme dostatek prostoru popsat všechny jejich finesy. Stávající parsery neumějí kontrolovat validitu dokumentu, který používá jmenné prostory, protože obvykle nemáme k dispozici odpovídající DTD, vzniklé sloučením několika sad značek. Usilovně se však pracuje na tom, aby tuto možnost nabízela XML schémata.

3. XML a odkazy

Důležitou vlastností webových stránek, která výraznou měrou přispěla k jejich velké oblibě, je možnost tvorby hypertextových odkazů. V jazyce HTML můžeme vytvořit z jednoho místa dokumentu odkaz na jiný související dokument nebo jeho část. Možnosti tvorby odkazů jsou v HTML však velice omezené. Pro potřeby jazyka XML byl proto vytvořen jazyk XLink, který umožňuje vytváření několika různých druhů odkazů. V této kapitole se podíváme na možnost tvorby odkazů v XML, a to nejen pomocí jazyka XLink. Ve stručnosti se seznámíme i s jazykem XPointer, který možnosti XLinku rozšiřuje.

3.1 Odkazy v rámci dokumentu

Pokud potřebujeme vytvářet odkazy mezi jednotlivými částmi dokumentu, vystačíme se standardními vlastnostmi XML. Víme, že v XML může být atribut typu ID.

```
<!ATTLIST kapitola id ID #IMPLIED>
```

Tomuto atributu můžeme v rámci dokumentu přiřadit jedinečnou hodnotu.

```
<kapitola id="kap1">
  <název>Ze života hmyzu</název>
  ...
</kapitola>
```

Pokud pak chceme na tento element vytvořit odkaz z jiného místa dokumentu, musíme použít atribut, který je typu `IDREF` nebo `IDREFS`.

```
Více o broucích naleznete v <odkaz cíl="kap1">této kapitole</odkaz>.
```

Prohlížeče mohou automaticky uživateli nabízet odkazy mezi elementy, které mezi sebou mají vztah na základě atributů typu `ID` a `IDREF`. Pokud parser kontroluje dokument podle DTD nebo schématu, automaticky také kontroluje, zda jsou všechny odkazy v pořádku. To znamená, zda v dokumentu není více elementů se stejným ID a zda v dokumentu není odkaz na neexistující ID.

Odkazy založené na ID lze bohužel použít pouze v rámci jednoho dokumentu. Pokud potřebujeme vytvářet odkazy mezi dokumenty, musíme sáhnout po jazyku XLink.

3.2 XLink

Podobně jako další standardy pochází i XLink (XML Linking Language) z dílny konsorcia W3C. Jeho cílem je poskytnout standardní nástroj pro tvorbu odkazů mezi XML dokumenty, ale nejen mezi nimi. Oproti odkazům, které známe z HTML, nabízí zejména následující nové vlastnosti:

- možnost tvorby odkazů mezi více než dvěma zdroji;
- možnost doplnění každého odkazu o metadata (přídavné informace);
- možnost vytvoření odkazů, které jsou uloženy mimo odkazované dokumenty.

Aby šlo odkazy kombinovat s libovolným typem dokumentu, používá se pro elementy a atributy, které tvoří odkazy, samostatný jmenný prostor. V době psaní knihy bohužel ještě nebyla schválena finální podoba standardu XLink. V návrhu standardu se pro jmenný prostor XLinku používá URI:

```
http://www.w3.org/1999/xlink/namespace
```

Ve finální podobě standardu se toto URI změní, takže si nezapomeňte zjistit jeho nový tvar. Ve všech následujících příkladech budeme předpokládat, že jsme pro celý dokument definovali prefix `xlink`. Samozřejmě, že prefix můžeme definovat i pro menší části dokumentu, ale obvykle používáme odkazy v celém dokumentu, takže je logické i jmenný prostor deklarovat pro celý dokument.

```
<dokument xmlns:xlink="http://www.w3.org/1999/xlink/namespace">
  ...
</dokument>
```

Jednoduché odkazy

Jednoduché odkazy nabízejí podobnou funkčnost jako odkazy, které známe z HTML. Tento druh odkazů bude ještě dlouhou dobu nejpoužívanější, a proto pro něj existuje možnost poměrně jednoduchého zápisu. Vezměme si příklad jednoduchého odkazu z HTML:

```
Informace <a href="http://www.kosek.cz">o tvorbě stránek</a>.
```

Tento odkaz spojuje dva informační zdroje. Cílem odkazu je stránka (informační zdroj) s adresou `http://www.kosek.cz`. Druhým informačním zdrojem, který je v naší ukázce odkazem propojen, je text o tvorbě stránek.

Podívejme se teď, jak se jednoduchý odkaz vytvoří pomocí XLinku. Slouží k tomu element `xlink:simple`. Povinně musíme použít atribut `xlink:href`, který slouží pro zápis cíle odkazu. Cíl se samozřejmě udává jako URL adresa.

```
Informace <xlink:simple xlink:href="http://www.kosek.cz">
o tvorbě stránek</xlink:simple>.
```

Kromě atributu `href` můžeme společně s elementem `simple` používat další atributy, které ovlivňují význam a chování odkazu.

`role` Tento atribut slouží k určení role odkazu. Jako hodnotu atributu můžeme použít identifikátor, který definuje druh odkazu. Hodnoty atributu nejsou předem nijak určeny, závisí na aplikacích.

Můžeme například použít identifikátor `copyright` pro označení odkazu, který vede na dokument s vymezením autorských práv.

`title` Pomocí tohoto atributu můžeme popsat cíl odkazu v podobě srozumitelné pro člověka.

`show` Tento atribut určuje, kde se objeví dokument, na který odkaz míří, pokud je aktivován (např. na něj klikne uživatel). K dispozici jsou tři hodnoty: `embed` dokument se zobrazí jako součást dokumentu, který obsahuje odkaz; `new` pro dokument se otevře nové okno; `replace` nový dokument nahradí v okně stávající.

`actuate` Atribut určuje, zda se odkaz aktivuje automaticky při načtení dokumentu (hodnota `onLoad`), nebo až na výslovný požadavek uživatele (hodnota `onRequest`).

Odkaz, který využívá všechny popsané atributy, by mohl vypadat například takto

```
<xlink:simple
        xlink:href="zamestnanci.xml"
        xlink:role="seznamzamestnancu"
        xlink:title="Aktuální seznam zaměstnanců"
        xlink:show="replace"
        xlink:actuate="onRequest">Naši zaměstnanci</xlink:simple>
```

Protože je neustále opisování prefixu jmenného prostoru dost nepohodlné, podle specifikace XLink můžeme u atributů tento prefix vynechat. To samozřejmě vyžaduje, aby toto vůči jmenným prostorům nestandardní chování aplikace podporovaly.

```
<xlink:simple
        href="zamestnanci.xml"
        role="seznamzamestnancu"
        title="Aktuální seznam zaměstnanců"
        show="replace"
        actuate="onRequest">Naši zaměstnanci</xlink:simple>
```

Odkaz můžeme vytvořit z libovolného elementu. V tomto případě však musíme u atributů používaných XLinkem uvádět prefix jmenného prostoru. Navíc musíme použít atribut `type`, kterým určíme typ odkazu.

```
<link xlink:type="simple"
      xlink:href="zamestnanci.xml"
      xlink:title="Aktuální seznam zaměstnanců">Naši zaměstnanci</link>
```

Chceme-li si ušetřit psaní, můžeme hodnotu atributů používaných XLinkem nastavit v DTD na nějakou pevnou hodnotu. Pokud by se například element `link` používal vždy jen pro jednoduché odkazy, v DTD použijeme deklaraci:

```
<!ATTLIST link xml:type (simple) #FIXED "simple">
```

Rozšířené odkazy

Narozdíl od jednoduchého odkazu slouží rozšířené odkazy ke spojení více dokumentů dohromady. Rozlišují se dva druhy rozšířených odkazů out-of-line a inline.

Odkazy out-of-line odkazují jen na externí dokumenty. Hodí se proto pro vytváření databází odkazů, které nejsou přímo součástí samotných dokumentů. To má mnohé výhody. Můžeme vytvářet odkazy, které spojují dokumenty, k nimž nemusíme mít právo zápisu (například protože jsou na cizím serveru) nebo které odkazy nepodporují (například formáty pro ukládání videozáznamů). Je pak samozřejmě věcí aplikace, která XLink podporuje, zda bude umět pracovat i s požadovanými databázemi odkazů. Až se nějaký podobný systém začne masově užívat, umožní například, aby každý uživatel připojil ke stránce, která ho zaujala nebo naštvala, nějaký komentář přístupný ostatním.

Inline odkazy musí zahrnovat alespoň jeden zdroj, který je přímo součástí dokumentu s rozšířeným odkazem. Jedná se tedy o obdobu jednoduchých odkazů s tím rozdílem, že může vést k více externím dokumentům najednou.

Zatímco jednoduché odkazy jsou jednosměrné a je u nich jasné, odkud a kam odkaz vede, rozšířené odkazy ve své základní podobě pouze zachycují vztah mezi informačními zdroji. Nijak nezachycují, který zdroj je podřazený nebo nadřazený.

Rozšířené odkazy se vytvářejí pomocí elementu `extended`, který patří do jmenného prostoru XLinku. Podobně jako u jednoduchých odkazů, můžeme i zde použít libovolný element, musíme u něj však nastavit atribut `type` na hodnotu `extended`.

Element `extended` typicky obsahuje další elementy, které přesně vymezují odkaz.

`locator`	Určuje jednotlivé externí zdroje, které rozšířený odkaz spojuje dohromady.
`resource`	Element obsahuje lokální zdroj, který je součástí odkazu (používá se pouze v inline odkazech).
`title`	Popis odkazu, který má být prezentován uživateli.
`arc`	Pomocí tohoto elementu můžeme určit směry, ve kterých jsou jednotlivé vztahy mezi zdroji platné. Pomocí `arc` se vytvářejí jednosměrné odkazy.

Kromě toho lze u elementu `extended` použít atributy `role` a `title`, které mají stejný význam jako u jednoduchých odkazů.

Každý rozšířený odkaz musí obsahovat alespoň jeden externí zdroj určený pomocí `locator`.

```
<xlink:extended>
  <xlink:title>Servery Seznamu</xlink:title>
  <xlink:locator href="http://www.seznam.cz" title="Seznam stránek"/>
  <xlink:locator href="http://www.novinky.cz" title="Internetový magazín"/>
  <xlink:locator href="http://kompas.seznam.cz" title="Fulltextové hledání"/>
</xlink:extended>
```

Pro určení role odkazu můžeme u elementu `locator` použít atribut `role` podobně jako u jednoduchých odkazů. Atributy `role` a `title` lze používat i u lokálních zdrojů.

```
<xlink:extended>
  <xlink:locator href="http://email.seznam.cz" title="Seznam"/>
  <xlink:locator href="http://mail.atlas.cz" title="Atlas"/>
  <xlink:locator href="http://www.post.cz" title="Post"/>
  <xlink:locator href="http://email.centrum.cz" title="Centrum"/>
  <xlink:resource>Přehled poštovních serverů</xlink:resource>
</xlink:extended>
```

Pomocí triku s deklarací atributu `type` můžeme i pro rozšířené odkazy používat libovolné elementy.

```
<!ATTLIST sluzba       xlink:type (extended) #FIXED "extended">
<!ATTLIST poskytovatel xlink:type (locator)  #FIXED "locator">
<!ATTLIST zdroj        xlink:type (resource) #FIXED "resource">
...
<sluzba>
  <poskytovatel xlink:href="http://email.seznam.cz" title="Seznam"/>
  <poskytovatel xlink:href="http://mail.atlas.cz" title="Atlas"/>
  <poskytovatel xlink:href="http://www.post.cz" title="Post"/>
  <poskytovatel xlink:href="http://email.centrum.cz" title="Centrum"/>
  <zdroj>Přehled poštovních serverů</zdroj>
</sluzba>
```

Pro specifikování směrů, ve kterých odkazy platí, slouží element `arc`. U něj můžeme použít atributy `from` a `to`, které určují, odkud a kam odkaz vede. Jako hodnota atributů se uvádí některá z hodnot použitá v atributu `role` ostatních elementů.

```
<xlink:extended>
  <xlink:locator href="..." role="otec" title="Jan Novák">
```

```
  <xlink:locator href="..." role="matka" title="Jana Nováková">
  <xlink:locator href="..." role="potomek" title="Štěpánka Nováková">
  <xlink:locator href="..." role="potomek" title="Karel Novák">
  <xlink:locator href="..." role="potomek" title="Barbora Nováková">
  <xlink:arc from="potomek" to="matka"/>
  <xlink:arc from="potomek" to="otec"/>
</xlink:extended>
```

Rozšířený odkaz říká, že má být možné přejít ze stránek Štěpánky, Karla a Barbory na stránky Jana a Jany.

U elementu `arc` můžeme navíc používat atributy `show` a `actuate`, které mají stejný význam jako u jednoduchých odkazů.

3.3 XPointer

Jak jsme viděli, XLink je velice mocný jazyk a nabízí toho mnohem více než odkazy z HTML. Všechny ukázky však odkazovaly na celé dokumenty určené pomocí URL zapsaného do atributu `xlink:href`. V mnoha případech však potřebujeme jemnější nástroj. Potřebujeme odkázat na určité místo dokumentu nebo část dokumentu. V HTML odkazech můžeme na konci URL použít fragment za znak '`#`' stačí napsat jméno návěstí, které jsme předtím definovali pomocí `<a name=`*«jméno»*`>`.

Jazyk XPointer (XML Pointer Language) je doplňkem jazyka XLink. Umožňuje na konec URL připojit výraz, který určuje pouze část celého dokumentu s daným URL.

Pro zachování zpětné kompatibility lze v XPointeru používat i fragmenty, jak jsme zvyklí z HTML. Dejme tomu, že dokument s názvem `dokument.xml` obsahuje uvnitř kapitolu:

```
...
<kapitola id="kap1">
  ...
</kapitola>
...
```

Pokud chceme vytvořit odkaz na tuto kapitolu, použijeme k tomu odkaz ve tvaru:

```
<xlink:simple href="dokument.xml#kap1">...</xlink:simple>
```

Pokud chceme v HTML vytvořit odkaz na určitou část dokumentu, která nemá své navěstí, máme smůlu. XPointer však nabízí způsob, jak identifikovat libovolnou část dokumentu na základě struktury dokumentu, textu elementů, obsahu atributů apod. V tomto případě se jako fragment použije zápis:

```
#xpointer(«výraz»)
```

«Výraz» přitom specifikuje určité místo dokumentu, na které vytváříme odkaz. Chceme-li například odkázat na třetí kapitolu v dokumentu, použijeme zápis:

```
#xpointer(kapitola[3])
```

Jazyk, který umožňuje vybírat jednotlivé části dokumentu, se jmenuje XPath (XML Path Language). Je oddělen od specifikace XPointer, protože se používá i v jazyce XSL. XPointer pro své potřeby pouze XPath drobně rozšiřuje.

XPath

Jazyk XPath slouží k výběru částí XML dokumentu. Pro potřeby jazyka je dokument chápán jako stromová hierarchie, kde jsou jednotlivé uzly tvořeny elementy, atributy a obsahem elementů. Výsledkem výrazu v XPath je pak skupina uzlů. Pro potřeby jazyka XPointer výrazy XPath konstruujeme obvykle tak, aby jim vyhovoval právě jeden uzel a odkaz byl jednoznačný. V následujícím přehledu si ukážeme některé výrazy a jim odpovídající výběry. Všechny operátory jazyka XPath se vždy vztahují k nějakému aktuálnímu uzlu. V XPointeru je aktuálním uzlem kořenový uzel. Kořenový uzel má jen jediné dítě, a tím je kořenový element dokumentu. Obrázek 3-1 na následující straně znázorňuje vztahy jako předek a potomek, které se používají v operátorech. Snadno si tak představíte, které uzly pomocí daného operátoru vybereme.

`para` Vybere všechny elementy `para`, které jsou dětmi aktuálního uzlu.

`*` Vybere všechny elementy, které jsou dětmi aktuálního uzlu.

`text()` Vybere všechny textové uzly, které jsou dětmi aktuálního uzlu.

`id(kap1)` Vybere uzel, který má atribut typu ID nastaven na hodnotu `kap1`.

`@name` Vybere atribut `name` aktuálního uzlu.

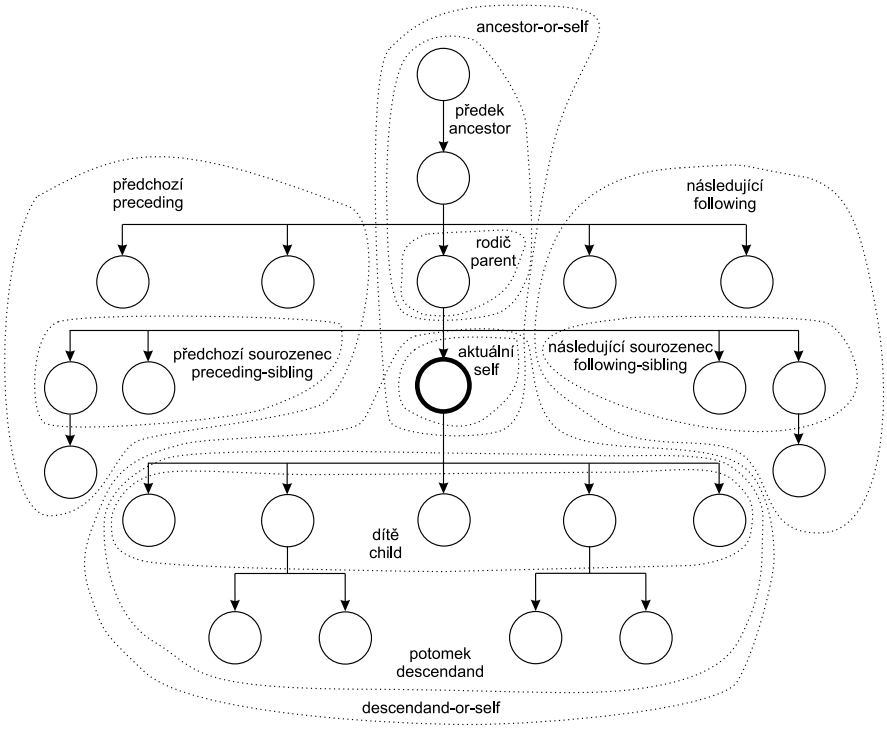
`@*` Vybere všechny atributy aktuálního uzlu.

`para[1]` Vybere první element `para`, který je dítětem aktuálního uzlu.

`para[last()]` Vybere poslední element `para`, který je dítětem aktuálního uzlu.

`*/para` Vybere všechny elementy `para`, které jsou vnoučaty (dětmi dětí) aktuálního uzlu.

`/dokument/kapitola[7]/sekce[3]` Vybere třetí sekci, v sedmé kapitole dokumentu. Výraz není vztažen k aktuálnímu uzlu, ale ke kořenovému uzlu, protože jsme na začátku výrazu použili znak '`/`'. Kořenový uzel má jen jediné dítě, a tím je kořenový element dokumentu.



Obr. 3-1: *Vztahy mezi uzly v stromové reprezentaci XML dokumentu*

`kapitola//para`
: Vybere všechny elementy `para`, které jsou potomky elementu `kapitola`, který je dítětem aktuálního uzlu.

`//para`
: Vybere všechny elementy `para`, které jsou potomky kořenového uzlu. To v praxi znamená, že jsou vybrány všechny elementy `para`, které se nacházejí ve stejném dokumentu jako aktuální uzel.

`//seznam/položka`
: Vybere všechny položky, které jsou ve stejném dokumentu jako aktuální uzel a mají jako rodiče element `seznam`.

`.`
: Vybere aktuální uzel.

`.//para`
: Vybere všechny elementy `para`, které jsou potomky aktuálního uzlu.

`..`
: Vybere rodiče aktuálního uzlu.

`../@lang`
: Vybere atribut `lang` u rodiče aktuálního uzlu.

`para[@type=warning]`
: Vybere všechny elementy `para`, které jsou dětmi aktuálního uzlu a které mají atribut `type` nastaven na hodnotu `warning`.

`para[@type=warning][5]`
: Vybere pátý element `para`, který je dítětem aktuálního uzlu a který má atribut `type` nastaven na hodnotu `warning`.

`para[5][@type=warning]`
: Vybere pátý element `para`, který je dítětem aktuálního uzlu, pouze pokud má zároveň atribut `type` nastaven na hodnotu `warning`. Omezující podmínky v hranatých závorkách jsou tedy vyhodnocovány postupně zleva doprava.

`kapitola[nadpis=Úvod]`
: Vybere element `kapitola`, který je dítětem aktuálního uzlu, pokud obsahuje jako dítě element `nadpis` s textem `Úvod`.

Uvedené výrazy používaly zkrácenou syntaxi. XPath umožňuje použití i nezkrácené syntaxe. Například výraz `kapitola/nadpis` je ekvivalentní se zápisem:

```
child::kapitola/child::para
```

O tom, že nezkrácená syntaxe rozhodně není nijak příjemná z hlediska zápisu, nás přesvědčí i následující příklad, který je ekvivalentem k `//para`:

```
/descendant-or-self::node()/child::para
```

V hranaté závorce obsahující omezující podmínku pro výběr uzlů můžeme používat několik operátorů a funkcí. K dispozici máme logické spojky `or` a `and`, relační operátory pro porovnávání a základní matematické operátory.

Nemáme zde prostor, abychom podrobně probrali všechny funkce. Omezíme se jen na malou ukázku jejich využití. Funkce `count(`*«výraz»*`)` vrací počet uzlů, které vyhovují *«výrazu»*. Pokud chceme vybrat seznamy, které obsahují právě dvě položky, můžeme k tomu použít zápis:

```
//seznam[count(položka)=2]
```

Rozšíření XPath pro potřeby XPointeru

Výše popsané vlastnosti jazyka XPath jsou pro potřeby XPointeru ještě rozšířeny a upřesněny.

Výrazy v XPath se vztahovaly k nějakému aktuálnímu uzlu ve stromové reprezentaci dokumentu. Pro XPointer zapsaný za URL je aktuálním uzlem kořenový uzel, který obsahuje celý dokument.

Největším rozšířením oproti XPath je přidání dalších dvou druhů objektů, které mohou vyhovovat nějakému výrazu. XPath umožňoval v dokumentu vybrat jen skupinu uzlů. Oproti tomu nabízí XPointer ještě body a rozsahy.

Bod (point) je konkrétní místo v dokumentu, které lze určit s přesností na jeden znak. Nejmenší jednotkou, kterou můžeme adresovat, tedy není element nebo atribut, ale konkrétní znak v dokumentu.

Bod je jen pomocnou strukturou, kterou využívá rozsah (range). Rozsah je definován dvěma body a reprezentuje tu část dokumentu, která je mezi těmito dvěma body. Pomocí rozsahu tak můžeme adresovat libovolný úsek dokumentu, podobně jako ho můžeme v editoru nebo prohlížeči označit myší. Rozsahy používají následující syntaxi:

«výraz-start» `to` *«výraz-konec»*

kde jednotlivé výrazy určují začátek a konec rozsahu. V rozsazích se nemusíme omezovat jen na elementy a atributy. K dispozici je i funkce `string-range`, která vrací seznam rozsahů na základě vyhledávání řetězce v textu. Následující ukázka vrací sedmý výskyt textu hledaný text v elementu `kapitola`.

```
string-range(//kapitola,"hledaný text")[7]
```

K dispozici jsou i další funkce. Jen namátkou si ukážeme použití funkcí `start-point` a `end-point`, které vybírají první, resp. poslední bod v daném výrazu. Pokud bychom chtěli vytvořit rozsah, který pokryje druhou a třetí kapitolu, můžeme použít zápis:

```
start-point(//kapitola[2]) to end-point(//kapitola[3])
```

XPointer v praxi

XPointer je ve spojení s jazykem XLink velice silný nástroj. Z celé skupiny XML technologií se jedná poměrně o novinku, takže v době psaní knihy ještě nebyly XLink a XPointer standardizovány. Nepodporovaly je ani nejrozšířenější prohlížeče Internet Explorer a Netscape Navigator. Jejich podporu nabízí například prohlížeč InDelv.

Už jsme si řekli, že pro zachování kompatibility s HTML odkazy umožňuje XPointer používat staré fragmenty. Odkaz ve tvaru:

```
http://nějaké.url/cesta/dokument.xml#an908
```

je ekvivalentní XPointeru:

```
http://nějaké.url/cesta/dokument.xml#xpointer(id("an908"))
```

XPointer umožňuje použití několika fragmentů za sebou. Ty se vyhodnocují zleva doprava tak dlouho, dokud některému z nich nevyhovuje část dokumentu. Tento mechanismus můžeme využít například v případech, kdy k dokumentu není vždy dostupné DTD a prohlížeč tedy neví, který z atributů je typu ID. Můžeme pak použít dva XPointery, přičemž ten druhý hledá v atributu s názvem `ID`.

```
#xpointer(id("an908"))xpointer(//*[@ID="an908"])
```

XPointery bývají často přímo součástí XML dokumentů. Pokud XPointer obsahuje nějaký znak, který v XML nelze použít, musíme ho nahradit odpovídající znakovou entitou. Týká se to zejména znaků '&' a '<'. Například místo zápisu:

```
<xlink:simple href='dok.xml#xpointer(kapitola/sekce[position() <= 5])'/>
```

musíme použít:

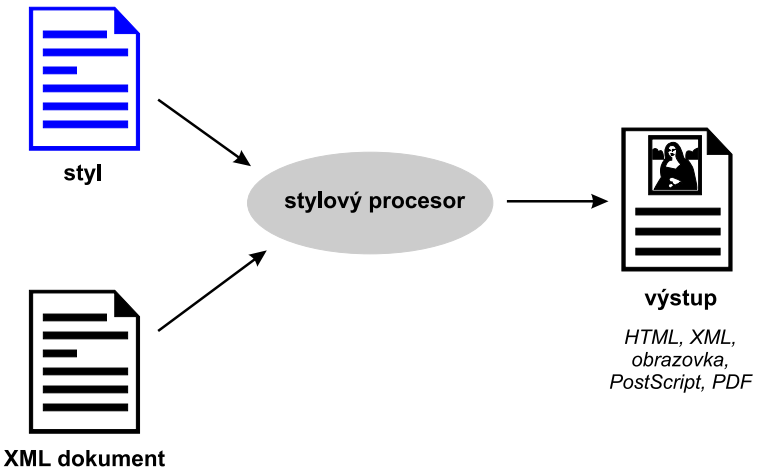
```
<xlink:simple href='dok.xml#xpointer(kapitola/sekce[position() &lt;= 5])'/>
```

4. Stylové jazyky

Většina dokumentů a informací v nich obsažených je určených pro člověka. Od uživatele však nemůžeme očekávat, že se bude chtít probírat XML kódem. Očekává, že se mu informace v přehledné a atraktivní podobě zobrazí na obrazovce, nebo že je dostane vytištěné na papíře. Jednou ze základních myšlenek XML je oddělení informačního obsahu dokumentu od definice grafického vzhledu. Jak bude konkrétní XML dokument vypadat, se definuje pomocí stylových jazyků. V těchto jazycích můžeme vytvořit *styl*, což je definice vzhledu jednotlivých elementů. Jednotlivé stylové jazyky nabízejí různě účinné prostředky pro definici prezentace jednotlivých elementů a atributů obsažených v dokumentu.

Výhodou tohoto přístupu je, že jeden styl může být použit pro mnoho dokumentů stejného druhu. Všechny dokumenty pak zachovávají jednotný vzhled, což jen pomáhá budovat image firmy či jednotlivce. Pokud se rozhodneme image změnit, chceme používat jiné barvy, jiná písma, stačí změnit definici stylu a všechny naše dokumenty rázem dostanou požadovaný vzhled.

Můžeme využít i obrácený přístup. Pro jeden dokument můžeme mít k dispozici více stylů. Jsme pak schopni uživatelům našich dokumentů informace nabídnout ve formě, která jim nejvíce vyhovuje. Někdo má rád stránky přeplácané barvami, někdo dává přednost střízlivé úpravě.



Obr. 4-1: *Vhodnou volbou stylu a stylového procesoru můžeme získat požadovaný výstup*

O zpracování dokumentu na základě stylu se stará speciální program, kterému se často říká *stylový procesor*. Většina uživatelů s tímto programem však

nepřijde do styku, protože bývá integrální součástí dalších programů editorů, prohlížečů, formátovačů apod. Prohlížeče a editory obsahují stylový procesor, který dokument podle stylu zobrazí na obrazovce, případně jej vytiskne na tiskárně. K dispozici jsou i procesory, které na základě definic ve stylu umějí XML dokument zformátovat a výsledek uložit v požadovaném formátu (např. HTML, PDF, RTF nebo PostScript). Tyto podoby dokumentu pak mohu být použity dále můžeme je umístit na webový server nebo odnést do tiskárny jako podklad pro tisk.

V této kapitole se podíváme na možnosti nejrozšířenějších stylových jazyků a ukážeme si, jak připojit styl k dokumentu.

4.1 Připojení stylu k dokumentu

Aby byla přenositelnost XML dokumentů mezi různými aplikacemi co největší, existuje standardizovaný způsob pro připojení stylu k dokumentu, který je definován v dokumentu *Associating Style Sheets with XML documents Version 1.0* [10]. K připojení stylu se využívá instrukce zpracování ve tvaru:

```
<?xml-stylesheet href="«URI»" type="«typ stylu»">
```

«URI» přitom určuje adresu (nejčastěji URL) souboru s definicí stylu. Jako *«typ stylu»* je potřeba uvést MIME typ použitého stylového jazyka. Například `text/css` pro kaskádové styly a `text/xsl` pro XSL. Kromě těchto atributů můžeme používat ještě jiné, které umožňují specifikovat další parametry chování a použití stylu.

Instrukce pro připojení stylu by měla být součástí prologu dokumentu. Obvykle ji tedy klademe hned za XML deklaraci.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet href="mujstyl.css" type="text/css"?>
<dokument>
  ...
</dokument>
```

Alternativní styly

Pokud chceme, aby si uživatel mohl vybrat pro zobrazení dokumentu z několika námi dodaných stylů, můžeme jich připojit k dokumetu více. Použijeme však navíc atribut `alternate`, kterému přiřadíme hodnotu `yes`. Do atributu `title` pak vložíme pro uživatele srozumitelný popis stylu. Na základě těchto popisů si pak uživatel může zvolit styl, který mu vyhovuje.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet alternate="yes" title="Normální"
```

```
                href="mujstyl.css" type="text/css"?>
<?xml-stylesheet alternate="yes" title="Kompaktní"
                href="mujmaly.css" type="text/css"?>
<?xml-stylesheet alternate="yes" title="Velký text"
                href="mujvelky.css" type="text/css"?>
<dokument>
  ...
</dokument>
```

Můžeme použít i více stylů se stejným popisem, jsou pak navzájem zkombinovány. Jak se styly navzájem kombinují, však záleží na použitém stylovém jazyku.

Ač to vypadá úplně skvěle, dnešní prohlížeče bohužel obvykle neumožní uživateli vybrat si styl, který chce. Takže se nedivte, až budete s touto vlastností experimentovat a nebude fungovat dle vašich představ.

Styly pro různá média

Při připojování stylu můžete určit, pro jaké médium je určen. Můžete například nabízet jeden styl pro zobrazení na obrazovce a druhý pro tisk. K čemu je to dobré? Samozřejmě, že pro větší pohodlí uživatele. Například je zjištěno, že na obrazovce se lépe čtou bezpatková písma (Arial, Helvetica, Verdana) a na papíře zase patková (Times New Roman, Palatino, Bookman). Tímto způsobem můžeme vzhled dokumentu přizpůsobit vlastnostem výstupního zařízení.

Médium, pro které je styl určen, se nastavuje pomocí atributu `media`. Jako jeho hodnotu můžeme použít jednu z následujících:

`screen`	počítačová obrazovka;
`tty`	terminál obrazovka pracující pouze v textovém režimu bez možnosti zobrazování grafiky;
`tv`	televizní přijímač obvykle s malým rozlišením (oproti počítačové obrazovce), s omezenou možností pohybu po zobrazené stránce;
`projection`	projektor;
`handheld`	kapesní zařízení malá obrazovka, obvykle bez možnosti zobrazování barev;
`print`	tiskárna;
`braille`	hmatová čtečka Braillova písma;
`aural`	hlasový syntetizátor;
`all`	styl je vhodný pro všechna výstupní zařízení.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet media="all" href="mujzaklad.css" type="text/css"?>
<?xml-stylesheet media="screen" href="mujobrazovka.css" type="text/css"?>
<?xml-stylesheet media="print" href="mujtypo.css" type="text/css"?>
<?xml-stylesheet media="aural" href="mujjednoduchy.css" type="text/css"?>
<?xml-stylesheet media="braille" href="mujjednoduchy.css" type="text/css"?>
<?xml-stylesheet media="projector" href="mujvelky.css" type="text/css"?>
<dokument>
  ...
</dokument>
```

4.2 Kaskádové styly

Kaskádové styly (CSS Cascading Style Sheets) se daly používat i s jazykem HTML. Na webových stránkách se s nimi bohužel nesetkáváme tak často, jak by se slušelo, protože výrobci jejich podporu do svých prohlížečů implementovali pomalu a nekvalitně. CSS existují ve verzích 1 a 2, v současné době se pracuje na třetí verzi. Verze 2 mimo jiné přinesla rozšíření, která byla nutná pro použití CSS společně s XML dokumenty.

Kaskádové styly mají poměrně jednoduchou a názornou syntaxi. Celý styl se skládá z pravidel, která mohou vypadat například takto:

```
kapitola nadpis { font-size: 24pt }
```

První části pravidla (`kapitola nadpis`) se říká selektor. Ten určuje, na které části dokumentu bude pravidlo aplikováno. V našem případě na všechny elementy `nadpis`, které jsou obsaženy v elementu `kapitola`. Druhou částí pravidla je deklarace, která určuje vzhled části dokumentu, vyhovující selektoru. Každé pravidlo má dvě části vlastnost a jí přiřazenou hodnotu. V našem příkladě nastavíme vlastnost `font-size` na hodnotu `24pt`. Tato deklarace říká, že se má použít písmo o velikosti 24 bodů.

Pro ovládnutí kaskádových stylů stačí znát pravidla pro tvorbu selektorů a jednotlivé vlastnosti včetně hodnot, které pro ně můžeme použít. My se podíváme na tvorbu selektorů, podrobný popis všech vlastností by se do naší útlé příručky nevešel. Podrobný přehled vlastností můžete prostudovat ve specifikaci *Cascading Style Sheets, level 2 CSS2 Specification* [5] v knize *HTML tvorba dokonalých WWW stránek* [21] nebo na webové stránce *Přehled vlastností stylů* [20].

V případech, kdy se stejná deklarace hodí pro více selektorů, si můžeme ušetřit práci a pro několik selektorů použít společnou deklaraci. Stačí jednotlivé selektory oddělit čárkami.

```
kapitola nadpis, příloha nadpis { font-size: 24pt }
```

Lze slučovat i deklarace, ty se však oddělují středníkem.

```
kapitola nadpis, příloha nadpis { font-size: 24pt;
                                  color: blue;
                                  text-align: center }
```

Více o selektorech

Základem efektivního vytvoření stylu je vhodné použití selektorů. CSS2 jich nabízejí poměrně širokou paletu. V následujícím přehledu si ukážeme příklady nejdůležitějších selektorů.

`*` Vybere všechny elementy.

`para` Vybere všechny elementy `para`.

`list item` Vybere všechny elementy `item`, které jsou potomkem elementu `list`.

`kapitola > nadpis`
Vybere všechny elementy `nadpis`, které jsou dětmi elementu `kapitola`.

`para:first-child`
Vybere element `para`, pokud je to první dítě svého rodiče.

`seznam + tabulka`
Vybere všechny elementy `tabulka`, které bezprostředně následují za elementem `seznam`.

`para[zarovnání]`
Vybere všechny elementy `para`, které mají nastaven atribut `zarovnání` na libovolnou hodnotu.

`para[zarovnání=vlevo]`
Vybere všechny elementy `para`, které mají atribut `zarovnání` nastaven na hodnotu `vlevo`.

`para[platforma~=Linux]`
Vybere všechny elementy `para`, které mají v atributu `platforma` seznam hodnot oddělených mezerami a jednou z těchto hodnot je `Linux`.

`kapitola#kap1`
Vybere element `kapitola`, který má ID nastaveno na `kap1`.

Výše uvedené typy selektorů lze navíc navzájem mezi sebou kombinovat a tím dosáhnout požadovaného efektu.

Zásady tvorby stylu

Kaskádové styly jste možná používali již při tvorbě webových stránek. Při vytváření stylu pro XML dokument však musíme postupovat trochu odlišně. Styl připojený k HTML stránce většinou pouze upravuje vzhled některých elementů. V případě stylu pro XML dokument začínáme na zelené louce a musíme definovat vzhled všech elementů.

Obvykle začneme s definicí stylu pro kořenový element, který v sobě obsahuje celý dokument. U něj deklarujeme vlastnosti, které mají být společné pro celý dokument např. barvu pozadí nebo použitý druh písma.

Dalším důležitým krokem při tvorbě stylu je rozlišení *blokových* a *inline* elementů. Blokové elementy jsou ty, které způsobují zalomení řádky např. odstavce textu, nadpisy, tabulka apod. Inline elementy jsou pak všechny ostatní, které se obvykle projeví pouze změnou písma v odstavci. Implicitně jsou prohlížečem všechny elementy považovány za inline, takže celý dokument je zobrazen jako jeden dlouhý nepřehledný odstavec (viz obrázek 2-4 na straně 32).

Poté, co rozdělíme elementy na blokové a inline, pak postupně upravujeme další parametry zobrazení velikosti písma a okrajů, barvy, způsoby zarovnání apod.

Ukázka použití CSS v praxi

Na závěr naší stručné exkurze si na jednoduchém příkladě ukážeme využití kaskádových stylů. Dejme tomu, že v XML ukládáme krátké články a chceme je pěkně prezentovat na našich stránkách. Značkování použité v článcích je velice jednoduché.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet href="clanek.css" type="text/css"?>
<clanek>
  <zahlavi>
    <rubrika>Aktuality</rubrika>
    <nazev>EU chce použít XML při rozvoji e-businessu</nazev>
    <autor>Jiří Kosek</autor>
  </zahlavi>
  <perex>Evropská komise si nechala vypracovat zprávu, která se zabývá
možnostmi rozvoje elektronického obchodování zejména pro malé a
střední podnikatele. Výsledky studie možná někoho překvapí --
e-business se bude rozvíjet díky použití XML a lepších vyhledávacích
technologií.</perex>
  <para>Studii pro Evropskou komisi vypracovala společnost
InfoConsult, která sdružuje několik předních evropských konzultantů,
kteří se nezaměřují jen na podnikání, ale i na informační a
komunikační technologie. Podle nich je dnes největší překážkou pro
```

```
uplatnění e-businessu <em>obtížné vyhledávání informací</em>. Pokud
hledáme na Internetu určitý výrobek, službu či firmu, musíme použít
klasické vyhledávací služby, které nás zahltí stovkami mnohdy
neužitečných odkazů.</para>
  <para>Studie navrhuje vytvoření sady XML tagů, kterými by se na
webových stránkách označovaly jednotlivé údaje o firmě, jejích
výrobcích a službách. Současně by se měla vytvořit nová vyhledávací
služba, která by "rozuměla" nově definovaným tagům. Pomocí této služby
by pak každý mohl velice snadno nalézt informace nebo výrobek, který
potřebuje.</para>
  <para>Jistě krásná idea však vyvolává mnoho praktických otázek a
problémů. První překážkou je zatím slabá podpora XML v běžně
používaných prohlížečích. Doplnění speciálních tagů do současných
webových stránek by nemuselo dělat dobře některým starším prohlížečům.
Aby byl celý systém opravdu užitečný, musela by speciální tagy na
svých stránkách používat většina firem. Jazyk XML zatím však autoři
webových stránek neovládají a dosud není na trhu cenově dostupný
WYSIWYG editor pro XML.</para>
</clanek>
```

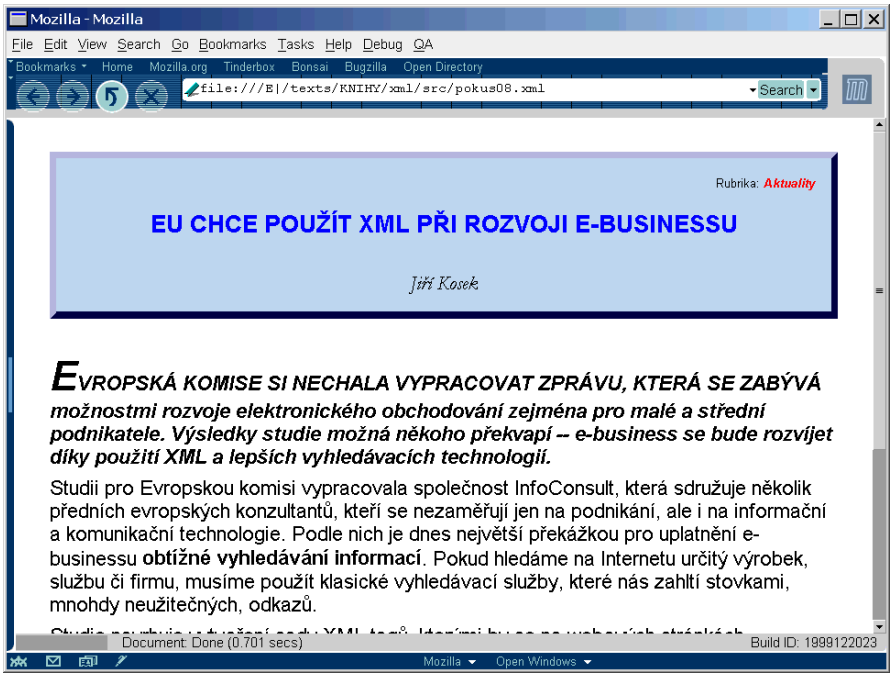
Druhá řádka dokumentu obsahuje instrukci pro připojení stylu ze souboru `clanek.css`. Tím, že všechny články budou používat jeden styl, snadno zajistíme jejich jednotný vzhled. Podívejme se na to, jak může vypadat styl v souboru `clanek.css`.

```
/* Celý dokument bude na bílém pozadí, fontem Arial */
clanek { background-color: white;
         font-family: Arial, Verdana, Helvetica, sans-serif }

/* Rozdělení elementů na blokové a inline */
zahlavi, nazev, autor, perex, para, rubrika { display: block }
em { display: inline }

/* Záhlaví článku */
zahlavi { text-align: center;
          padding: 10pt;
          margin: 20pt;
          background-color: #C0D0F0;
          border: 4pt outset navy }

/* Rubrika je plovoucí objekt a je před ní doplněn text */
rubrika { float: right;
          color: red;
```

```
            font-size: xx-small;
            font-weight: bold;
            font-style: italic;
            white-space: nowrap }
rubrika:before { content: "Rubrika: ";
                 color: black;
                 font-weight: normal;
                 font-style: normal }

/* Název článku */
nazev { font-size: 120%;
        text-transform: uppercase;
        font-weight: bold;
        color: blue;
        margin: 20pt }

/* Autor článku */
autor { font-family: Palatino, Garamond, serif;
        font-style: italic }

/* Mezera mezi odstavci */
para, perex { margin: 5pt 20pt }

/* Perex je zvýrazněným písmem a s odlišnou úpravou první řádky */
perex { font-style: italic;
        font-weight: bold; }
perex:first-letter { font-size: 200% }
perex:first-line { text-transform: uppercase }

/* Zvýraznění textu */
em { font-weight: bold }
```

Ve stylu je použito několik selektorů, o kterých jsme se dosud nezmínili. Kaskádové styly od verze 2 umožňují před a za element doplnit text. Pokud chceme, aby se před název rubriky vypsal text Rubrika:, použijeme selektor `rubrika:before`. Můžeme pak použit vlastnost `content` a do ní vložit text, který se má při formátování dokumentu pomocí stylu objevit. Jak se můžeme přesvědčit na obrázcích 4-2 na následující straně a 4-3 na straně 78, Mozilla tuto vlastnost podporuje, ale Internet Explorer má ještě co dohánět.

Podobně jsou na tom i prohlížeče s podporou pseudoelementů `first-letter` a `first-line` v selektorech. Slouží k nastavení stylu pro první písmeno a první řádku elementu. Snadno tak dosáhneme velice zajímavých efektů.



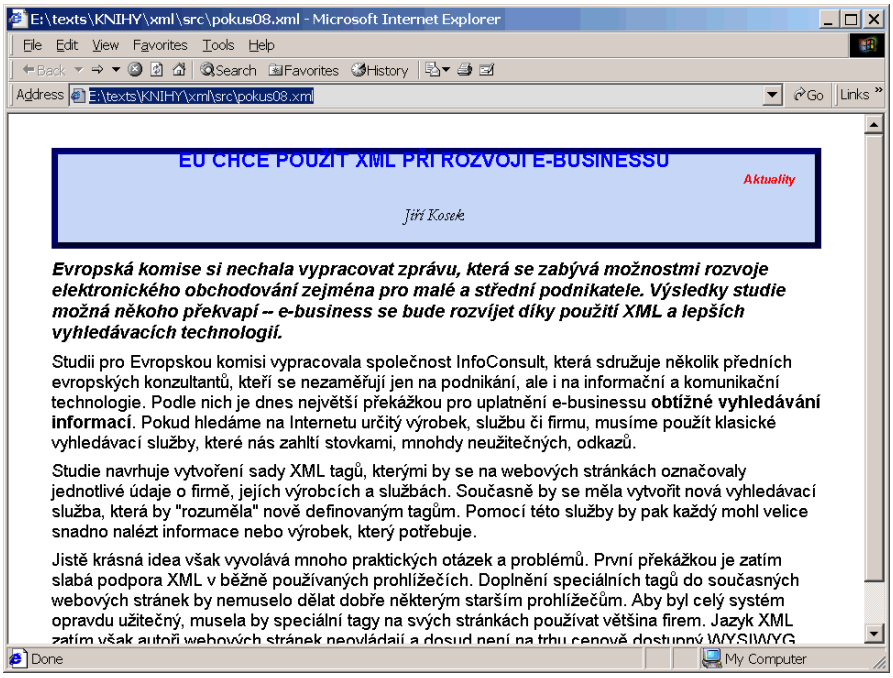
Obr. 4-2: Mozilla má velice dobrou podporu CSS

4.3 XSL

XSL (eXtensible Stylesheet Language) vznikl jako univerzální stylový jazyk, který by měl nabízet funkčnost všech ostatních existujících stylových jazyků a dále ji rozšířit. Samotná syntaxe jazyka je samozřejmě založena na XML, takže pro zpracování stylu lze použít všechny nástroje, které umějí s XML pracovat.

Samotný standard XSL *Extensible Stylesheet Language (XSL) Version 1.0 W3C Working Draft* [4] rozděluje jazyk XSL na dvě části. První část jazyka obsahuje nástroje pro *transformaci* XML dokumentů. Této části se říká XSLT (XSL Transformations). Druhá část standardu pak definuje *formátovací objekty*. Ty slouží jako abstraktní popis vzhledu stránky a jejích jednotlivých komponent. Formátovací objekty mají (jak jinak) syntaxi XML.

XSLT se dá použít pro transformaci XML dokumentu do XML dokumentu s jinou strukturou, případně do HTML nebo textového formátu. Samozřejmě, že lze XSLT použít i pro transformaci, jejímž výstupem je dokument skládající se z formátovacích objektů. Takový dokument pak může být pomocí speciálních programů zformátován a zobrazen či vytištěn.



Obr. 4-3: Internet Explorer bohužel nepodporuje všechny možnosti CSS

Transformace pomocí XSLT

Základním stavebním kamenem stylů v XSLT jsou šablony. Každá šablona určuje dvě věci na jaké části vstupního dokumentu je použita a jak bude tato část dokumentu transformována do výstupního dokumentu. Definice šablony má následující tvar:

```
<xsl:template match="«výraz»">
  «výstup»
</xsl:template>
```

Vidíme, že styl používá jmenný prostor `xsl`. Ten je obvyklý právě pro styly v XSL. Celý styl by proto měl být obalen v jednom elementu, který definuje jmenný prostor.

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  «deklarace šablon»
</xsl:stylesheet>
```

«Výraz» v šabloně určuje elementy, na které bude šablona aplikována. Pro zápis *«výrazu»* se používá jazyk XPath, který jsme si popsali v předchozí kapitole.

Pokud bychom chtěli pomocí stylu převést všechny nadpisy na elementy `H1`, které známe z HTML, můžeme použít šablonu:

```
<xsl:template match="nadpis">
  <H1><xsl:apply-templates/></H1>
</xsl:template>
```

Element `apply-templates` přitom XSL procesoru říká, aby na obsah vybraného elementu (v našem případě `nadpis`) aplikoval další šablony. Jejich výstup je vložen na místo `apply-templates`. Díky tomu, že každý procesor má v sobě automaticky zabudované šablony pro zpracování samotného textového obsahu elementu, bude vždy po použití `apply-templates` minimálně vypsán obsah elementu.

XSLT nabízí při tvorbě šablon další nepřeberné možnosti, pro jejich popis zde bohužel nemáme dostatek prostoru. V jednotlivých šablonách můžeme používat podmínky a cykly, elementy lze řadit na základě různých kritérií atd. XSLT samozřejmě nabízejí prostředky pro automatické číslování můžeme číslovat kapitoly, obrázky, položky seznamu apod. Styl lze rozdělit na více částí a navzájem je kombinovat. Pro složitější styly se hodí možnost používání proměnných a parametrů.

Na začátku stylu můžeme použít element `output` a určit požadovaný typ výstupu XML, HTML nebo text. Lze určit kódování použité ve výstupním souboru. Podrobný popis XSLT naleznete v jeho specifikaci *XSL Transformations (XSLT) Version 1.0* [12]. Nakladatelství Grada pro vás rovněž připravuje překlad knihy *The XML companion* [8], kde je tématu XSLT a XSL věnován široký prostor.

Praktické využití XSLT

Některé z možností XSLT si nyní ukážeme na příkladě. Dejme tomu, že chceme články, které jsou podobné tomu z naší ukázky použití kaskádových stylů, automaticky převádět do HTML pro starší prohlížeče a do jazyka WML pro moderní mobilní telefony. Použití XSLT představuje v tomto případě optimální řešení. Stačí vytvořit jednoduché styly pro konverzi do HTML a do WML, a všechny články mohou být konvertovány zcela automaticky.

Začneme se stylem pro HTML. Naším cílem bude vygenerovat velice jednoduchou HTML stránku, která neobsahuje žádné speciální efekty. K tomu můžeme použít následující styl:

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

  <!-- Výstup bude do HTML v kódování ISO Latin 2 -->
```

```
<xsl:output indent="yes" method="html" encoding="iso-8859-2"/>

<!-- Zpracování celého dokumentu, kořenový element vygeneruje
     kostru HTML dokumentu -->
<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="/clanek/zahlavi/autor"/>:
             <xsl:value-of select="/clanek/zahlavi/nazev"/></title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>

<xsl:template match="zahlavi">
  <table border="1">
    <tr><td><xsl:apply-templates/></td></tr>
  </table>
</xsl:template>

<xsl:template match="rubrika">
  <p align="right"><font size="-2">
    Rubrika: <b><xsl:apply-templates/></b>
  </font></p>
</xsl:template>

<xsl:template match="nazev">
  <h1 align="center"><xsl:apply-templates/></h1>
</xsl:template>

<xsl:template match="autor">
  <p align="center"><i><xsl:apply-templates/></i></p>
</xsl:template>

<xsl:template match="perex">
  <p><i><xsl:apply-templates/></i></p>
</xsl:template>

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

```
  </xsl:template>

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

</xsl:stylesheet>
```

Tento styl můžeme na článek aplikovat dvěma způsoby. Jednak lze umístit na začátek každého článku instrukci pro připojení stylu:

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

Dokument pak můžeme rovnou zobrazit v prohlížeči, který podporuje XSLT. Většina dnešních prohlížečů však XSLT ani XML nepodporuje, a proto je asi lepší dokument rovnou převést do HTML. Můžeme k tomu použít nějaký XSLT procesor např. XT,[1]

```
xt clanek.xml clanek.xsl clanek.html
```

nebo MSXSL:[2]

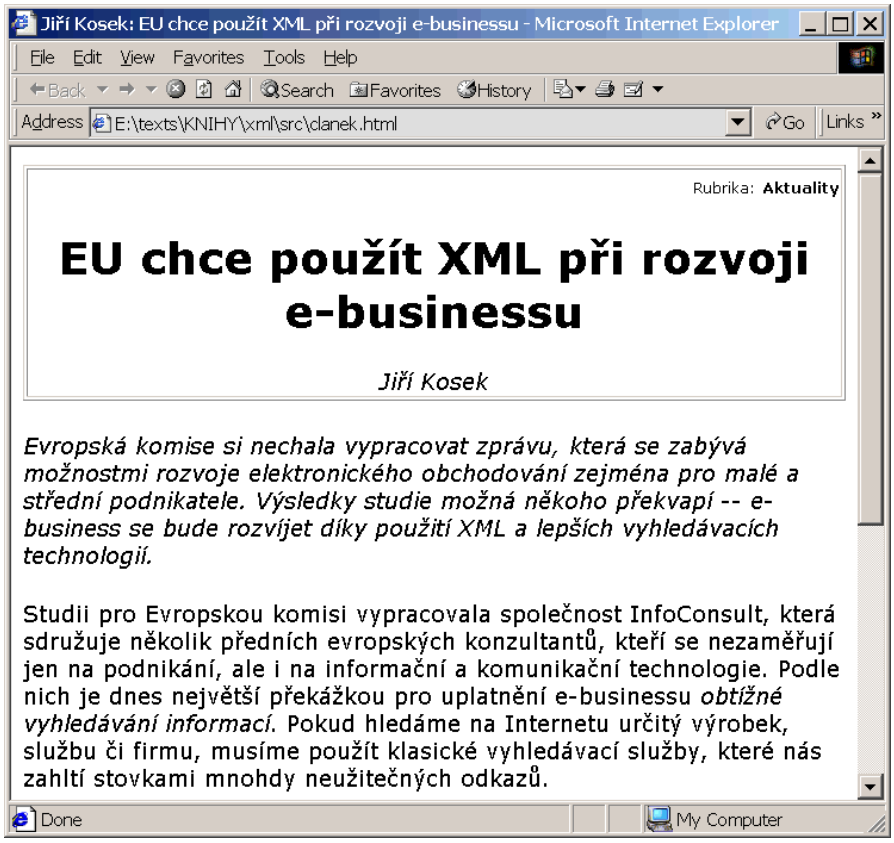
```
msxsl clanek.xml clanek.xsl clanek.html
```

V souboru `clanek.html` získáme verzi článku v HTML. Na obrázku 4-4 na následující straně se můžeme podívat, jak se stránka zobrazí v prohlížeči.

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<title>Jiří Kosek: EU chce použít XML při rozvoji e-businessu</title>
</head>
<body>
  <table border="1">
    <tr><td>
      <p align="right"><font size="-2">Rubrika: <b>Aktuality</b></font></p>
      <h1 align="center">EU chce použít XML při rozvoji e-businessu</h1>
      <p align="center"><i>Jiří Kosek</i></p>
    </td></tr>
  </table>
  <p><i>Evropská komise si nechala vypracovat zprávu, která se zabývá
možnostmi rozvoje elektronického obchodování zejména pro malé a
střední podnikatele. Výsledky studie možná někoho překvapí --
```

[1] Návod na instalaci XT naleznete v příloze *XSLT procesor XT* na straně 153.

[2] Popis instalace XSL procesoru od Microsoftu naleznete v příloze *XSLT procesor od Microsoftu* na straně 154.



Obr. 4-4: Zobrazení HTML stránky automaticky vygenerované pomocí XSLT

```
e-business se bude rozvíjet díky použití XML a lepších vyhledávacích
technologií.</i></p>
  <p>Studii pro Evropskou komisi vypracovala společnost InfoConsult,
která sdružuje několik předních evropských konzultantů, kteří se
nezaměřují jen na podnikání, ale i na informační a komunikační
technologie. Podle nich je dnes největší překážkou pro uplatnění
e-businessu <i>obtížné vyhledávání informací</i>. Pokud hledáme na
Internetu určitý výrobek, službu či firmu, musíme použít klasické
vyhledávací služby, které nás zahltí stovkami mnohdy neužitečných
odkazů.</p>
  <p>Studie navrhuje vytvoření sady XML tagů, kterými by se na
webových stránkách označovaly jednotlivé údaje o firmě, jejích
výrobcích a službách. Současně by se měla vytvořit nová vyhledávací
služba, která by "rozuměla" nově definovaným tagům. Pomocí této služby
by pak každý mohl velice snadno nalézt informace nebo výrobek, který
```

```
potřebuje.</p>
  <p>Jistě krásná idea však vyvolává mnoho praktických otázek a
problémů. První překážkou je zatím slabá podpora XML v běžně
používaných prohlížečích. Doplnění speciálních tagů do současných
webových stránek by nemuselo dělat dobře některým starším prohlížečům.
Aby byl celý systém opravdu užitečný, musela by speciální tagy na
svých stránkách používat většina firem. Jazyk XML zatím však autoři
webových stránek neovládají a dosud není na trhu cenově dostupný
WYSIWYG editor pro XML.</p>
</body>
</html>
```

Podobně můžeme vytvořit i styl pro generování WML kódu. Jelikož jsou displeje mobilních zařízení malé, je každá WML stránka rozdělena na několik kart. Zobrazena je pak vždy jen jedna z nich. Do WML stránky se proto musí přidat prvky, které umožní přechod mezi jednotlivými kartami. Každé kartě proto musíme přiřadit jednoznačný identifikátor. XSLT naštěstí nabízí prostředky pro generování čítačů, a proto není problém jednotlivé odstavce očíslovat. Na stylu pro WML si předvedeme i trochu odlišný přístup k celkové konstrukci stylu.

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

  <!-- Výstup bude do WML v kódování UTF-8 -->
  <xsl:output method="xml" encoding="utf-8" indent="yes"
              doctype-public="-//WAPFORUM//DTD WML 1.1//EN"
              doctype-system="http://www.wapforum.org/DTD/wml_1.1.xml"/>

  <!-- Zpracování článku -->
  <xsl:template match="/clanek">
    <wml>
      <!-- Tlačítko Zpět bude na každé kartě -->
      <template>
        <do type="prev">
          <prev/>
        </do>
      </template>

      <!-- Úvodní karta s názvem článku -->
      <card>
        <!-- Tlačítko pro přechod na první odstavec článku -->
        <do type="accept" label="Další">
```

```
        <go href="#perex"/>
      </do>
      <p align="center">
        <big><xsl:value-of select="zahlavi/nazev"/></big></p>
      <p align="center"><i><xsl:value-of select="zahlavi/autor"/></i></p>
      <p align="right">
        Rubrika: <b><xsl:value-of select="zahlavi/rubrika"/></b>
      </p>
    </card>

    <!-- První odstavec článku -->
    <card id="perex">
      <!-- Tlačítko pro přechod na další odstavec článku -->
      <do type="accept" label="Další">
        <go href="#no1"/>
      </do>
      <p><xsl:value-of select="perex"/></p>
    </card>

    <!-- Zpracování všech odstavců -->
    <xsl:for-each select="para">
      <!-- Vygenerování jedinečného id každé karty -->
      <xsl:element name="card">
        <xsl:attribute name="id">no<xsl:number
          value="position()" format="1"/></xsl:attribute>
        <!-- U všech odstavců kromě posledního vygenerujeme tlačítko
             pro přechod na další kartu -->
        <xsl:if test="position()!=last()">
          <do type="accept" label="Další">
            <xsl:element name="go">
              <xsl:attribute name="href">#no<xsl:number
                value="position()+1" format="1"/></xsl:attribute>
            </xsl:element>
          </do>
        </xsl:if>
        <!-- Na obsah odstavce aplikujeme další šablony,
             např. pro zvýraznění písma <em>...</em> -->
        <p><xsl:apply-templates select="."/></p>
      </xsl:element>
    </xsl:for-each>
  </wml>
</xsl:template>
```

```
    <!-- Zvýraznění písma -->
    <xsl:template match="em">
      <i><xsl:apply-templates/></i>
    </xsl:template>

  </xsl:stylesheet>
```

Zobrazení článku ve WML prohlížeči ilustruje obrázek 4-5. Zobrazení je výsledkem WML kódu, který byl vygenerován pomocí stylu.

Obr. 4-5: *Z XML dokumentu můžeme velice snadno generovat formát vhodný pro mobilní zařízení*

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
                     "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <template>
    <do type="prev">
      <prev/>
    </do>
  </template>
  <card>
    <do type="accept" label="Další">
      <go href="#perex"/>
    </do>
```

```
    <p align="center">
      <big>EU chce použít XML při rozvoji e-businessu</big>
    </p>
    <p align="center">
      <i>Jiří Kosek</i>
    </p>
    <p align="right">
      Rubrika: <b>Aktuality</b>
    </p>
  </card>
  <card id="perex">
    <do type="accept" label="Další">
      <go href="#no1"/>
    </do>
    <p>Evropská komise si nechala vypracovat zprávu, která se zabývá
možnostmi rozvoje elektronického obchodování zejména pro malé a
střední podnikatele. Výsledky studie možná někoho překvapí --
e-business se bude rozvíjet díky použití XML a lepších vyhledávacích
technologií.</p>
  </card>
  <card id="no1">
    <do type="accept" label="Další">
      <go href="#no2"/>
    </do>
    <p>Studii pro Evropskou komisi vypracovala společnost
InfoConsult, která sdružuje několik předních evropských konzultantů,
kteří se nezaměřují jen na podnikání, ale i na informační a
komunikační technologie. Podle nich je dnes největší překážkou pro
uplatnění e-businessu <i>obtížné vyhledávání informací</i>. Pokud
hledáme na Internetu určitý výrobek, službu či firmu, musíme použít
klasické vyhledávací služby, které nás zahltí stovkami mnohdy
neužitečných odkazů.</p>
  </card>
  <card id="no2">
    <do type="accept" label="Další">
      <go href="#no3"/>
    </do>
    <p>Studie navrhuje vytvoření sady XML tagů, kterými by se na
webových stránkách označovaly jednotlivé údaje o firmě, jejích
výrobcích a službách. Současně by se měla vytvořit nová vyhledávací
služba, která by "rozuměla" nově definovaným tagům. Pomocí této služby
by pak každý mohl velice snadno nalézt informace nebo výrobek, který
```

```
potřebuje.</p>
  </card>
  <card id="no3">
    <p>Jistě krásná idea však vyvolává mnoho praktických otázek a
problémů. První překážkou je zatím slabá podpora XML v běžně
používaných prohlížečích. Doplnění speciálních tagů do současných
webových stránek by nemuselo dělat dobře některým starším prohlížečům.
Aby byl celý systém opravdu užitečný, musela by speciální tagy na
svých stránkách používat většina firem. Jazyk XML zatím však autoři
webových stránek neovládají a dosud není na trhu cenově dostupný
WYSIWYG editor pro XML.</p>
  </card>
</wml>
```

Z ukázek je patrné, že XSLT je velice mocný jazyk. Jeho použití se vyplatí zejména v případech, kdy potřebujeme převádět do několika dalších formátů velké množství dokumentů.

Formátovací objekty

Možnost snadné konverze XML dokumentů do HTML a do dalších formátů založených na XML je užitečná. V mnoha případech však potřebujeme dokument velice kvalitně formátovat pro tisk nebo pro prezentaci na obrazovce. Pro tyto potřeby nabízí XSL mechanismus nazvaný formátovací objekty (FO). Pomocí těchto objektů můžeme přesně specifikovat rozměry stránky, způsob zarovnání, použitá písma, barvy atd. V závislosti na použitém procesoru pak dostaneme třeba soubor PDF nebo se nám dokument zobrazí na obrazovce.

Při tvorbě stylu kombinujeme XSLT a FO. Formátovaný dokument se tak pomocí stylu transformuje do dokumentu, který obsahuje FO, a ty jsou pak procesorem interpretovány.

Formátovací model použitý v XSL nabízí mnoho možností a je proto poměrně složitý. Jeho implementace teprve postupně vznikají, ani standard ještě není ve své finální podobě.

Výsledkem XSL musí být dokument, který používá jmenný prostor:

```
http://www.w3.org/1999/XSL/Format
```

a má kořenový element `root`. Pro formátovací objekty se obvykle používá prefix `fo`, a proto kostra XSL stylu, který používá FO, obvykle vypadá zhruba takto:

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

```
    <xsl:template match="/">
      <fo:root>
        ...
      </fo:root>
    </xsl:template>

    ...

  </xsl:stylesheet>
```

Element `fo:root` pak obsahuje dva základní druhy elementů. Jednak je to `fo:layout-master-set`. Tento element umožňuje definovat layout jednotlivých stránek jejich rozměry, velikost okrajů, počet sloupců apod. Kořenový element dále obsahuje element `fo:page-sequence`, který už zastupuje obsah jednotlivých stránek.

Stránky vytváří XSL procesor tak, že do prostoru definovaného layoutem stránky umisťuje obsah elementu `fo:flow`. Kromě elementu `fo:flow` může `fo:page-sequence` obsahovat i element `fo:static-content`, který se používá pro specifikování částí stránky, které se opakují (záhlaví, zápatí apod.).

Uvnitř `fo:flow` se pak mohou objevit objekty, které zastupují jednotlivé části výsledného dokumentu. Mezi ty nejpoužívanější patří:

`block` Objekt odpovídá blokovým elementům, které známe z kaskádových stylů. Typicky se používá se pro odstavce, nadpisy apod.

`external-graphic` Objekt zastupuje obrázek, který je uložen mimo výsledný dokument formátovacích objektů. Obvykle je obrázek uložen v externím souboru (GIF, JPEG, PNG, EPS apod.).

`float` Plovoucí objekt umístí se na vhodné místo stránky. Obvykle se používá pro obrázky a tabulky.

`footnote` Objekt se používá pro poznámky pod čarou.

`footnote-citation` Objekt se používá pro značku poznámky pod čarou použitou v textu.

`inline` Formátovací objekt nezpůsobující vznik nového odstavce. Používá se například pro změny druhu písma uvnitř odstavce.

`leader` Objekt se používá pro čáry nebo opakované znaky (tečky), které mají vyplnit daný prostor. Používá se například v obsahu pro oddělení názvu kapitoly od čísla strany.

`list-block`, `list-item`, `list-item-body`, `list-item-label` Objekty se používají pro seznamy.

`simple-link` Umožňuje do výsledného dokumentu zařadit odkazy.

`table`, `table-*`
Několik objektů, které umožňují vytváření tabulek.

`wrapper` Objekt se používá v případech, kdy je potřeba pro několik objektů nastavit společné vlastnosti.

Formátovací objekty slouží k popisu základních komponent, které se budou skládat na stránky. U každého formátovacího objektu můžeme formou atributů nastavit nepřeberné množství vlastností barvy, způsob zarovnání, použité písmo, okraje atd. K dispozici jsou vlastně všechny vlastnosti, které nabízí poslední verze kaskádových stylů, DSSSL a několik dalších. XSL je tak v současné době stylový jazyk s největšími možnostmi.

Pro ilustraci XSL a FO si ukážeme styl, který umožní formátovat náš článek. Výsledek si můžeme prohlédnout na obrázku 4-6.


Rubrika: **Aktuality**

EU chce použít XML při rozvoji e-businessu

Evropská komise si nechala vypracovat zprávu, která se zabývá možnostmi rozvoje elektronického obchodování zejména pro malé a střední podnikatele. Výsledky studie možná někoho překvapí – e-business se bude rozvíjet díky použití XML a lepších vyhledávacích technologií.

Studii pro Evropskou komisi vypracovala společnost InfoConsult, která sdružuje několik předních evropských konzultantů, kteří se nezaměřují jen na podnikání, ale i na informační a komunikační technologie. Podle nich je dnes největší překážkou pro uplatnění e-businessu *obtížné vyhledávání informací*. Pokud hledáme na Internetu určitý výrobek, službu či firmu, musíme použít klasické vyhledávací služby, které nás zahltí stovkami mnohdy neužitečných odkazů.

Studie navrhuje vytvoření sady XML tagů, kterými by se na webových stránkách označovaly jednotlivé údaje o firmě, jejích výrobcích a službách. Současně by se měla vytvořit nová vyhledávací služba, která by "rozuměla" nově definovaným tagům. Pomocí této služby by pak každý mohl velice snadno nalézt informace nebo výrobek, který potřebuje.

Jistě krásná idea však vyvolává mnoho praktických otázek a problémů. První překážkou je zatím slabá podpora XML v běžně používaných prohlížečích. Doplnění speciálních tagů do současných webových stránek by nemuselo dělat dobře některým starším prohlížečům. Aby byl celý systém opravdu užitečný, musela by speciální tagy na svých stránkách používat většina firem. Jazyk XML zatím však autoři webových stránek neovládají a dosud není na trhu cenově dostupný WYSIWYG editor pro XML.

— Jiří Kosek —

Obr. 4-6: *Dokument formátovaný pomocí XSL a formátovacích objektů*

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

  <xsl:template match="/">
    <fo:root>
      <!-- Definice layoutu stránky -->
      <fo:layout-master-set>
        <!-- Rozměry stránky a její okraje -->
        <fo:simple-page-master master-name="page"
                               page-height="297mm"
                               page-width="210mm"
                               margin="1in">
          <!-- Tiskové zrcadlo - oblast pro samotný obsah stránky -->
          <fo:region-body margin-bottom="15mm"/>
          <!-- Oblast pro patu stránky -->
          <fo:region-after extent="10mm"/>
        </fo:simple-page-master>
      </fo:layout-master-set>

      <!-- Definice obsahu stránky -->
      <fo:page-sequence master-name="page">
        <!-- Společný obsah všech stránek v patě stránky -->
        <fo:static-content flow-name="xsl-region-after">
          <fo:block>
            <!-- Číslo strany na každé stránce -->
            <xsl:text>Strana </xsl:text>
            <fo:page-number/>
          </fo:block>
        </fo:static-content>
        <!-- Samotný text dokumentu -->
        <fo:flow flow-name="xls-region-body">
          <!-- Zpracování všech elementů zdrojového dokumentu -->
          <xsl:apply-templates/>
      <xsl:apply-templates select="clanek/zahlavi/autor"/>
        </fo:flow>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>

  <!-- Šablona pro záhlaví článku -->
```

```
<xsl:template match="zahlavi">
  <fo:block text-align="center" space-before="20pt"
            space-after="14pt" font-family="Helvetica">
    <xsl:apply-templates select="rubrika"/>
    <xsl:apply-templates select="nazev"/>
  </fo:block>
</xsl:template>

<!--Šablona pro rubriku -->
<xsl:template match="rubrika">
  <fo:block font-size="6pt" text-align="end">
    <xsl:text>Rubrika: </xsl:text>
    <fo:inline font-weight="bold">
      <xsl:apply-templates/>
    </fo:inline>
  </fo:block>
</xsl:template>

<!--Šablona pro název článku -->
<xsl:template match="nazev">
  <fo:block font-size="24pt" font-weight="bold" space-before="6pt"
            space-after="4pt">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<!--Šablona pro autora článku -->
<xsl:template match="autor">
  <fo:block font-style="italic" text-align="end" space-before="6pt">
    <xsl:text>&#x2014; </xsl:text>
    <xsl:apply-templates/>
    <xsl:text> &#x2014; </xsl:text>
  </fo:block>
</xsl:template>

<!--Šablona pro perex -->
<xsl:template match="perex">
  <fo:block text-align="justify" font-style="italic" font-weight="bold">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

```
  <!--Šablona pro odstavec -->
  <xsl:template match="para">
    <fo:block text-indent="20pt" text-align="justify">
      <xsl:apply-templates/>
    </fo:block>
  </xsl:template>

  <!--Šablona pro zvýraznění -->
  <xsl:template match="em">
    <fo:inline font-style="italic">
      <xsl:apply-templates/>
    </fo:inline>
  </xsl:template>

</xsl:stylesheet>
```

5. XML schémata

Jazyk XML vychází z SGML, které se používalo především pro značkování textových dokumentů. XML dnes kromě této oblasti směřuje i do oblasti databází. Svět databází má však již dlouhou dobu mnohem větší požadavky na možnosti definice typů dat, než nám mohou nabídnout DTD. Aby mohly být využity výhody validace dokumentů vůči DTD i pro více databázově orientované dokumenty, vzniklo postupně několik jazyků pro definici schématu dokumentu.

Většina jazyků pro určení schématu dokumentu si za cíl kladla nahradit DTD a dále rozšířit jeho možnosti definování přípustné struktury dokumentů. Dva hlavní požadavky na schémata přitom téměř vždy představovala lepší možnost definice datových typů pro obsah atributů a elementů a syntaxe schémat založená na XML. Jazyků pro zápis schémat vzniklo nezávisle na sobě několik. Mezi asi nejznámější patří XML-Data, který pochází z dílny Microsoftu. Konsorcium W3C nyní pracuje na vytvoření jednotného jazyka pod názvem XML schémata. My se v této krátké kapitole podíváme na to, co XML schémata nabízejí. Je možné, že finální verze standardu se v některých syntaktických detailech bude od prezentovaných informací lišit.

5.1 Datové typy

Schémata řeší podporu různých datových typů velice komplexně. Jednak přímo obsahují několik nejběžnějších datových typů. Z nich jsou pak odvozeny ještě další, jemnější datové typy. Navíc si můžeme definovat i typy vlastní.

Základní datové typy

Kromě mnohem jemnější rozlišovací schopnosti, kterou datové typy schémat přinášejí oproti DTD, je zde důležitá další novinka. Na rozdíl od DTD můžeme datový typ použít i pro obsah elementu, nejen pro hodnoty atributů. Tabulka 5-1 na následující straně stručně popisuje datové typy definované v návrhu standardu XML schémat.

Odvozené datové typy

Na základě předešlých datových typů jsou definovány odvozené datové typy. Pro každý odvozený typ jsou jeho přípustné hodnoty podmnožinou některého ze základních datových typů.

První skupinu odvozených datových typů tvoří typy, které známe z DTD a ze specifikace XML. Součástí schémat jsou především kvůli snadné konverzi stávajících DTD. Nalezneme zde proto typy jako `NMTOKEN`, `ID`, `IDREF` apod.

Typ	Popis
`string`	textový řetězec
`boolean`	logická hodnota (true/false)
`float`, `double`, `decimal`	desetinná čísla s různou přesností
`timeInstant`	časový okamžik
`timeDuration`	časový interval
`recurringInstant`	opakující se časový okamžik
`binary`	binární data
`uri`	adresa internetového zdroje

Tab. 5-1: *Základní datové typy*

Typ	Popis
`integer`	celé číslo
`non-negative-integer`	nezáporné celé číslo
`positive-integer`	kladné celé číslo
`non-positive-integer`	nekladné celé číslo
`negative-integer`	záporné celé číslo
`date`	datum
`time`	čas

Tab. 5-2: *Některé odvozené datové typy*

K dispozici jsou i typy, které zastupují znaky přípustné v názvech elementů a atributů.

Druhá skupina odvozených typů pouze definuje další bežně používané typy, které lze odvodit z těch základních (viz tabulka 5-2).

Definice vlastních typů a integritní omezení

Největší síla schémat však spočívá v možnosti definice vlastních datových typů. Typy pak můžeme používat ve schématu stejně jako standardní typy. Kontrolu typů tak máme plně pod kontrolou.

Nový typ se definuje odvozením z nějakého již existujícího. V definici můžeme zároveň použít různá integritní omezení, která dále zúží okruh přípustných hodnot pro tento typ. Mezi integritní omezení patří například minimální a maximální délka, výčet přípustných hodnot, přesnost u čísel apod.

Jednoduchý datový typ se definuje pomocí elementu `datatype`. My si ukážeme, jak si definovat typ, který se bude hodit pro údaje o peněžních částkách, které nepřesáhnou milión a budou uváděna s přesností na haléře.

```
<datatype name="částka" source="decimal">
  <precision value='8'/>
  <scale value='2'/>
```

```
</datatype>
```

Náš nový typ se jmenuje `částka` a je odvozen od typu `decimal`. Má osm platných cifer, z nichž dvě jsou vyhrazeny pro desetinnou část. Pomocí elementů `precision` a `scale` jsme definovali omezení na typ `decimal`, ze kterého byl odvozen náš typ.

Integritní omezení mají poměrně bohaté možnosti. Mohou například obsahovat výčet povolených hodnot. Jako ukázku si definujeme typ `kódMěny`, který může obsahovat kód české, německé nebo americké měny.

```
<datatype name="kódMěny" source="string">
  <enumeration value="CZK"/>
  <enumeration value="DEM"/>
  <enumeration value="USD"/>
</datatype>
```

Pro zkušené je k dispozici možnost vytvoření omezení na základě regulárního výrazu. Regulární výraz je maska, které musí řetězec vyhovět. Následující ukázka definuje typ vhodný pro uložení PSČ.

```
<datatype name="psč" source="string">
  <pattern value="\d{3} \d{2}"/>
</datatype>
```

V regulárním výrazu můžeme použít mnoho znaků se speciálním významem. Například `\d` zastupuje libovolnou číslici. Zápis `{3}` zase říká, že předchozí znak se musí vyskytnout právě třikrát. Našemu výrazu tedy vyhoví řetězce, které obsahují tří číslice, mezeru a dvě číslice.

V mnoha případech vystačíme i s primitivnějšími integritními omezeními, která jsou však velmi užitečná. Následující ukázka definuje typ pro uložení uživatelského jména a pro výši odměny.

```
<datatype name="uživatelskéJméno" source="string">
  <minlength value="3"/>
  <maxlength value="8"/>
</datatype>

<datatype name="odměna" source="integer">
  <minInclusive value="0"/>
  <maxInclusive value="20000"/>
</datatype>
```

5.2 Definice elementů

Syntaxe schémat je založena na XML, a tak se dostáváme do zajímavé situace, kdy elementy definujeme pomocí elementu `element`. Jméno elementu je přitom určeno pomocí atributu `name` a typ pomocí atributu `type`.

```
<element name="jméno" type="string"/>
```

Takto definovaný element může obsahovat pouze data typu `string` tedy text. Nemůže obsahovat již další elementy. Dokument vyhovující našemu schématu proto může obsahovat:

```
<jméno>Jan Novák</jméno>
```

Nemůže však již obsahovat:

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

protože schéma omezilo obsah elementu pouze na text, ne na další elementy.

Pokud chceme, aby element mohl obsahovat ještě další vnořené elementy, musíme je deklarovat uvnitř elementu. Pokud bychom chtěli používat druhý způsob zápisu jména, vypadalo by schéma zhruba takto:

```
<element name="jméno">
  <type>
    <element name="křestní" type="string"/>
    <element name="příjmení" type="string"/>
  </type>
</element>
```

Definice říká, že element `jméno` musí obsahovat elementy `křestní` a `příjmení`.

Opakování elementů

U každého elementu můžeme specifikovat, kolikrát se může v daném místě opakovat. Určit lze minimální a maximální počet. Tím do rukou dostáváme mnohem účinnější nástroj, než v DTD nabízené znaky '`?`', '`+`' a '`*`'. Definici elementu pro uchování knihy, která musí mít jeden název, nemusí mít autora a obsahuje alespoň dvě kapitoly, lze zapsat ve schématech velice jednoduše.

```
<element name="kniha">
  <type>
    <element name="název" type="string"/>
    <element name="autor" type="string" minOccurs="0" maxOccurs="1"/>
    <element name="kapitola" minOccurs="2" maxOccurs="*">
```

```
      ...
    </element>
  </type>
</element>
```

Modely obsahu na druhou

Schémata rozšiřují možnosti DTD snad ve všech směrech. Výjimkou není ani specifikování modelu obsahu elementu. K dispozici máme samozřejmě možnosti, které známe z DTD.

Smíšený obsah lze vytvořit velice jednoduše. Stačí použít atribut `content` a uvést hodnotu `mixed`. Obsahem elementu pak kromě dalších elementů může být i přímo text.

```
<element name="jméno">
  <type content="mixed">
    <element name="křestní" type="string"/>
    <element name="příjmení" type="string"/>
  </type>
</element>
```

Obvykle se mají elementy v dokumentu vyskytovat v pořadí, které specifikujeme ve schématu. Někdy se může hodit, aby se vyskytly v libovolném pořadí. Třeba u jména je nám jedno, zda je nejprve uvedeno křestní jméno nebo příjmení. Tuto možnost nám již DTD v XML nenabízejí.

```
<element name="jméno">
  <type>
    <group order="all">
      <element name="křestní" type="string"/>
      <element name="příjmení" type="string"/>
    </group>
  </type>
</element>
```

Element `group` může v atributu `order` obsahovat ještě hodnoty `seq` a `choice`. První z nich říká, že elementy se musí v dokumentu vyskytovat ve stejném pořadí jako ve schématu. Druhá varianta pak říká, že v dokumentu se smí vyskytnout jen jeden z elementů definovaných ve skupině. Samozřejmě, že skupiny lze podle potřeby do sebe vzájemně zanořovat.

5.3 Definice atributů

Atributy se definují pomocí elementu `attribute`, který se uvádí jako součást typu elementu.

```
<element name="kniha">
  <type>
    <element name="název" type="string"/>
    <element name="autor" type="string" minOccurs="0" maxOccurs="1"/>
    <element name="kapitola" minOccurs="2" maxOccurs="*">
      ...
    </element>
    <attribute name="jazyk" type="langauge"/>
    <attribute name="verze" type="string"/>
  </type>
</element>
```

Pokud je použití atributu povinné, stačí do definice přidat atribut `minOccurs` a nastavit ho na hodnotu 1.

```
<attribute name="jazyk" type="langauge" minOccurs="1"/>
```

Pokud má mít atribut pevně stanovenou hodnotu, můžeme k tomu využít atribut `fixed`.

```
<attribute name="jazyk" type="langauge" fixed="cs"/>
```

5.4 Modularizace nám ušetří práci

Velkou výhodou schémat je, že kromě vlastních jednoduchých typů můžeme typ vytvořit i ze skupiny elementů či atributů. Jednou vytvořený typ pak můžeme využít na mnoha místech schématu. To se hodí zejména v případech, kdy mají elementy některé atributy společné nebo se u několika elementů opakovaně používá stejný model obsahu. Následující ukázka obsahuje definici skupiny atributů a její použití ve dvou elementech.

```
<attributeGroup name="společnéAtributy">
  <attribute name="jazyk" type="langauge"/>
  <attribute name="autor" type="string"/>
</attributeGroup>

<element name="para">
  <type>
    ...
    <attributeGroup ref="společnéAtributy"/>
  </type>
</element>

<element name="nadpis">
```

```
    <type>
      ...
      <attributeGroup ref="společnéAtributy"/>
    </type>
  </element>
```

Podobně můžeme vytvářet i typy pro obsah elementů, na které se budeme dále odvolávat.

```
<type name="obsahOdstavce" content="mixed">
  <element name="em" type="string"/>
  <element name="strong" type="string"/>
  <element name="link" type="string">
    <attribute name="href" type="uri"/>
  </element>
</type>

<element name="para" type="obsahOdstavce"/>
<element name="nadpis" type="obsahOdstavce"/>
```

Podobných vymožeností nabízejí schémata celou řádu, ale na jejich detailní popis zde bohužel nemáme prostor.

5.5 Bez dokumentace by to nešlo

Schémata budou sloužit k formálnímu popisu datových formátů používaných po celém světě. Pro jejich správné použití je proto důležitá dobrá dokumentace. Tu je přitom možné zařadit přímo do schématu, aby se někde po cestě neztratila.

```
<datatype name="kódMěny" source="string">
  <annotation>
    <info>Nejpoužívanější kódy měn</info>
  </annotation>
  <enumeration value="CZK">
    <annotation>
      <info>Česká koruna</info>
    </annotation>
  </enumeration>
  <enumeration value="DEM">
    <annotation>
      <info>Německá marka</info>
    </annotation>
  </enumeration>
</datatype>
```

6. Použití XML v praxi

Doufám, že po přečtení předchozích kapitol jste již konečně pochopili, že XML je úplně skvělá technologie. Počáteční nadšení však může odpadnout a je na čase zjistit, k čemu a jak můžeme v praxi XML použít. V této kapitole se proto podíváme na typické oblasti, kde se XML používá.

Pokaždé, když řešíme nějaký problém, musíme ho samozřejmě nejprve podrobit důkladné analýze. Použití XML může být dobrým řešením, ale ne vždy tím nejlepším. XML také není jediná technologie, kterou použijeme. Musíme si vybrat odpovídající software, upravit ho pro naše požadavky, vyškolit uživatele a mnoho a mnoho dalšího. Existují celé teorie a metodologie, které popisují, jak projektovat a zavádět informační systémy. My zde nemáme prostor se jimi podrobně zabývat,[1] nicméně musíte mít vždy na paměti, že si nestačí říci: použijeme XML. XML bude jen jedna z technologií, kterou můžeme použít. Může to být sice ústřední prvek celého řešení, ale nikdy není jediný.

My se pro účely této kapitoly dopustíme určitého zjednodušení, a budeme se zajímat zejména o to, co nám XML přinese v jednotlivých oblastech, kde se dnes počítače používají.

6.1 Potřebujeme standardy?

Pokud se rozhodneme, že pro ukládání nebo výměnu dat budeme v naší aplikaci používat XML, stojíme před dalším problémem. Musíme vytvořit DTD nebo schéma, které bude vyhovovat našim potřebám. Máme přitom několik možností. Samozřejmě si můžeme schéma vytvořit zcela sami na zelené louce, tak, aby přesně vyhovovalo našim požadavkům. V mnoha případech to může být poměrně dlouhý a nákladný postup. Často pro naši aplikaci již vhodné schéma existuje a můžeme je bez úprav využít. Ušetříme tím mnoho práce a času. Výhodou použití již existujícího DTD jsou i existující jednoúčelové programy, které dané schéma podporují.

V mnoha případech se nám již existující schéma hodí, až na pár drobností. Nejrychlejší pak může být ho použít a podle potřeby upravit. I pokud budeme vytvářet vlastní schéma, je dobré prostudovat již existující schémata z příbuzných oblastí a nechat se inspirovat.

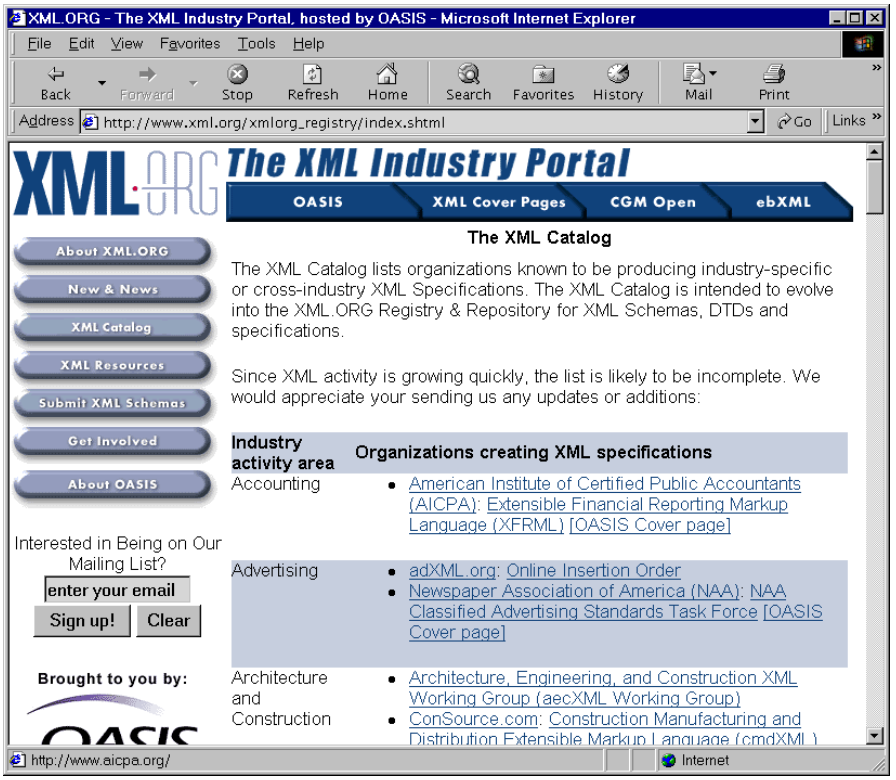
Vždy se tedy vyplatí podívat se na již existující schémata. Kde však hledat informace o již existujících schématech?

Na Internetu začínají postupně vznikat speciální servery, které mají za cíl shromažďovat a třídit DTD a schémata. Často se jím říká repozitáře, protože schraňují informace, důležité pro mnoho uživatelů XML.

[1] Což je nakonec dobře, protože se alespoň neukáže, že této oblasti rozumím podstatně méně než XML.

XML.ORG

Mezi nejznámější repozitáře patří server XML.ORG,[2] který má na starosti sdružení OASIS. Cílem sdružení je vytvářet široce akceptované standardy pro aplikace založené na XML. OASIS proto s firmami a vývojáři spolupracuje na vytváření schémat, která jsou vhodná pro jednotlivé oblasti.



Obr. 6-1: *Za serverem XML.ORG stojí společnost OASIS, která má v oblasti značkovacích jazyků velký kredit*

BizTalk

Za vznikem repozitáře BizTalk[3] stojí firma Microsoft. Na serveru nalezneme mnoho schémat, která se hodí pro výměnu informací v různých oblastech lidské

[2] `http://www.xml.org`

[3] `http://www.biztalk.org`

Obr. 6-2: *Na serveru BizTalk jsou všechna schémata přehledně rozčleněna do několika kategorií*

činnosti. Kromě samotného schématu je k dispozici i dokumentace a ukázkové dokumenty.

Schémata jsou na serveru BizTalk uložena v jazyce XML-Data, který je dílem Microsoftu. Bohužel není kompatibilní s posledními návrhy XML schémat. Microsoft se však zavázal, že po zveřejnění konečné specifikace schémat opustí svůj schémový jazyk ve prospěch standardu konsorcia W3C.

Zájmová sdružení

Možná, že v budoucnu bude možné pomocí repozitářů nalézt libovolné DTD nebo schéma. Dnes jsou však repozitáře teprve v plenkách. Mnoho DTD je přístupných pouze na serverech různých zájmových sdružení, která si pro svoji potřebu vytvořila formát založená na XML. Vyplatí se proto navštívit i servery,

které se věnují oblasti, pro kterou aplikaci vyvíjíte. Mnohdy mohou pomoci i vyhledávací služby.

Výborným zdrojem informací jsou i stránky, které shromažďují odkazy na zajímavé aplikace XML. Mezi asi nejlepší patří stránky Robina Covera,[4] které naleznete opět na serveru OASIS.

6.2 Elektronické publikování

Papírové knihy jsou krásné, ale nehodí se pro všechny oblasti použití. Pro mnoho aplikací je potřeba kromě vytištěné verze nějakého dokumentu získat i jeho elektronickou podobu. Ať už pro publikování na Webu, na CD-ROMu nebo v intranetu. Na samotné přípravě tiskovin se počítače dnes podílejí ve velkém měřítku. Velká většina dnes produkovaných knih a časopisů je připravována na počítačích v DTP systémech. Ty nabízejí bohaté možnosti pro práci s grafikou a s formátováním dokumentu, ale obvykle nenabízejí vhodné nástroje pro vytvoření elektronických verzí dokumentů.

Použít pro elektronické publikování řešení založené na XML se vyplatí zejména v následujících případech:

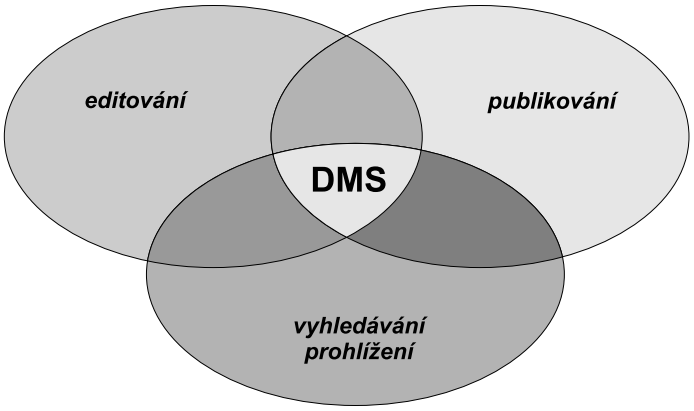
- Potřebujeme mít výsledný dokument k dispozici v několika formátech (tištěný, Web, CD-ROM, nápověda v programu apod.).
- Produkujeme rozsáhlé dokumenty technická dokumentace, vědecké sborníky, dokumentace k programům, slovníky, encyklopedie apod.
- Produkujeme velké množství dokumentů, i když ty nemusí být nutně velké redakce časopisů, zpravodajské agentury, legislativa, technická podpora.

Pokud budeme dokumenty ukládat do XML, získáme tím především možnost je snadno publikovat v několika odlišných formátech, což se dnes stává téměř nezbytností.

Systémy pro elektronické publikování mohou vypadat různě, ale většinou mají společné alespoň základní rysy. Celý proces přípravy dokumentů a jejich zpracování probíhá ve třech poměrně oddělených fázích. První fáze spočívá v *editování a vytváření* samotných dokumentů ve formátu XML. Při této fázi autoři píši knihy, články, dokumentaci apod. Vytvářené dokumenty jsou upravovány editory a korektory, schvalovány pro publikování nadřízenými pracovníky apod. Hotové dokumenty pak mohou vstoupit do *publikační fáze*. V této fázi se z dokumentů generují potřebné výstupní formáty. U některých druhů dokumentů tento proces může probíhat zcela automaticky například konverze do HTML, formátování beletrie a technické dokumentace. Pokud se však z XML dokumentů skládají dohromady například tištěné noviny, musí se výsledný layout

[4] `http://www.oasis-open.org/cover/`

stránky upravit ručně. Některé věci prostě nelze automatizovat. V některých aplikacích pak může v úvahu připadat ještě *prohledávání databáze dokumentů*. To se uplatní v případech, kdy potřebujeme vyvolat již existující dokumenty, používat jejich části apod.



Obr. 6-3: *Naše dokumenty a data jsou při elektronickém publikování to nejcennější, proto se vše točí okolo nich*

Jádrem většiny systémů podporujících elektronické publikování je systém pro správu dokumentů. V nejjednodušších verzích může jít o nějaký sdílený disk, kam mají přístup všichni oprávnění uživatelé. Mnohem lepší je však použití specializovaných programů, které dokumenty ukládají do speciální databáze. Přístup k dokumentům pak může být velice dobře kontrolován, systém eviduje změny provedené jednotlivými uživateli a může zajišťovat i funkce pro work-flow dokumentů před tím, než je dokument považován za finální, jej musí schválit tiskové oddělení firmy a ředitel.

Se systémem pro správu dokumentů (DMS) pak spolupracují editory, ve kterých jednotliví autoři vytvářejí a upravují dokumenty. Při publikování jsou z DMS získány potřebné dokumenty a převedeny do výsledných formátů. Většina DMS navíc obsahuje modul, který umožňuje dokumenty kontextově i fulltextově prohledávat.

Podrobnější popis DMS, editorů a dalších programů použitelných pro elektronické publikování naleznete v následující kapitole. My se teď podíváme na nejpoužívanější DTD, které se hodí pro elektronické publikování. Možná zjistíte, že některé z nich plně splňuje vaše požadavky.

DocBook

DocBook je dnes asi druhá nejpoužívanější aplikace SGML/XML, hned za jazykem HTML. DocBook vznikl v roce 1991 jako formát založený na SGML,

určený především pro výměnu unixové dokumentace. U jeho zrodu stála firma HaL Computers a nakladatelství O'Reilly. O vývoj a údržbu formátu se staralo sdružení Davenport. V 90. letech DocBook používalo mnoho velkých firem, které se podílely i na jeho vývoji (např. Novell, Digital, Hewlett-Packard, SCO, Fujitsu).

V roce 1999 se péče o DocBook přesunula do sdružení OASIS. Aktuální verze DocBooku nese označení 3.1. Ta je založena na SGML, existuje i neoficiální verze, která používá XML. Během roku 2000 by měla být uvolněna verze 4.0, která již bude existovat ve dvou verzích pro SGML i pro XML.

DocBook se vyvinul do podoby systému, který se hodí zejména pro tvorbu počítačové dokumentace. Bez problému ho však lze použít pro zápis libovolných knih a článků. V DocBooku je například vytvořena tato kniha, dokumentace k mnoha programům je vytvářena rovněž v DocBooku např. k operačnímu systému FreeBSD, ke skriptovacímu jazyku PHP, Linux také přechází na DocBook. DocBook používají i velká počítačová nakladatelství jako O'Reilly.

DocBook obsahuje elementy, které umožňují členit dokumenty do kapitol, podkapitol atd. Kromě toho máme k dispozici mnoho elementů, jimiž můžeme označit názvy programů, souborů, parametry příkazů, výpisy programů, obrázky, snímky obrazovky, klávesové zkratky, položky nabídek apod. Získáme tak velice dobře označkovaný dokument, který se dobře převádí do dalších formátů.

Výhodou DocBooku je, že mnoho editorů a nástrojů pro práci s XML v sobě přímo zahrnuje jeho podporu. Stačí program spustit a můžeme v DocBooku začít psát. Na stránkách Norma Walshe[5] jsou volně k dispozici XSL a DSSSL styly, které lze použít pro formátování dokumentů v DocBooku. Ve spojení s vhodným procesorem (XT, Jade) tak můžeme z našich dokumentů snadno vytvořit např. HTML stránky, dokument v RTF, v PDF nebo v PostScriptu.

Samotná DTD pro DocBook nalezneme opět na serveru OASIS, tentokrát na adrese `http://www.oasis-open.org/docbook/`. K DocBooku existuje výborná dokumentace *DocBook: The Definitive Guide* [28], která je k dispozici i v elektronické podobě.[6]

TEI , The Text Encoding Initiative

Cílem projektu TEI[7] je vytvořit postupy pro uchovávání textových materiálů pro potřeby výzkumu. Kromě elementů pro popis struktury textů nalezneme v TEI speciální elementy, který umožňují v dokumentech vyznačit údaje důležité pro literární, historický nebo lingvistický výzkum. Původně byl TEI určen pro SGML, dnes existuje i verze pro XML. Počátky projektu TEI sahají do roku

[5] `http://www.nwalsh.com`

[6] `http://www.docbook.org/`

[7] `http://www.tei-c.org/`

1988, a proto je dnes k dispozici poměrně velké množství nástrojů, které tento formát přímo podporují.

Open eBook

Sdružení Open eBook[8] se snaží vytvořit standardy pro vydávání elektronických knih. Prvním z jeho počinů je vytvoření jazyka Open eBook Publication Structure (OEB), který vychází z XML. Standardně v sobě zahrnuje většinu elementů HTML 4.0. Tím je umožněno prohlížení elektronických knih v současných prohlížečích. Při návrhu OEB mysleli jeho tvůrci na budoucnost OEB je kompatibilní i s jazykem XHTML. OEB podporuje grafické formáty PNG a JPEG. Může obsahovat i data v jiných formátech, v tomto případě však musí nabídnout i alternativní reprezentaci pro prohlížeče OEB, které nestandardní formáty nepodporují.

Aby bylo vyhledávání a katalogizování elektronických knih co nejvíce usnadněno, musí každá kniha obsahovat svoji klasifikaci. Využívá se přitom dnes již etablovaný systém metadat Dublin Core. Při definování vzhledu knihy se používají kaskádové styly (CSS).

ISO 12083

Mezinárodní standard ISO 12083[9] obsahuje několik DTD, která se hodí pro značkování knih, článků, seriálových publikací a matematických vzorců. Původní verze v SGML se nyní přepracovává do XML.

CALS

CALS (Continuos Acquisition and Lifecycle Support)[10] je skupina několika DTD a dalších formátů pro výměnu dokumentace mezi Ministerstvem obrany USA a jeho dodavateli. Jedno z DTD je vhodné pro zápis tabulek a v praxi se používá velice často, mnoho editorů umožňuje tabulky CALS editovat ve WYSIWYG režimu. DTD pro tvorbu dokumentů jsou oproti ostatním poměrně restriktivní, slouží k popisu přesných technologických postupů, popisu údržby apod.

6.3 Elektronická komerce

Na úspěchu a propagaci XML se největší měrou podílí oblast elektronické komerce. XML se zde uplatňuje jako formát pro výměnu dat mezi obchodními partnery, mezi zákazníkem a firmou apod. Zápis těchto typů dokumentů v XML

[8] http://www.openebook.org/

[9] http://www.xmlxperts.com/12083.htm

[10] http://www.acq.osd.mil/cals/

je velmi přirozený, protože je potřeba skloubit výrazně strukturovaná data s daty méně strukturovanými, jak je vidět z následující ukázky objednávky zapsané v XML.

```
<objednávka datum="12.01.2000">
  <dodavatel>
    <název>Švadlenka a spol.</název>
    <ulice>Krejčovská 7</ulice>
    <město>Dolní Jehla</město>
    <psč>555 43</psč>
    <ičo>12345678</ičo>
    <dič>007-12345678</dič>
  </dodavatel>
  <odběratel>
    <název>Krejčík Jan</název>
    <ulice>Horní 55</ulice>
    <město>Zadov</město>
    <psč>550 12</psč>
    <ičo>76543210</ičo>
    <dič>007-76543210</dič>
  </odběratel>
  <položka>
    <název>Nit černá</název>
    <množství>12</množství>
    <cena měna="Kč">200</cena>
  </položka>
  <položka>
    <název>Knoflík čtyřdírkový, průměr 1 cm</název>
    <množství>4</množství>
    <cena měna="Kč">7.50</cena>
  </položka>
</objednávka>
```

Nemůžeme samozřejmě očekávat, že si každý vymyslí vlastní formáty pro jednotlivé obchodní dokumenty jako my. V této oblasti vznikly standardy pochopitelně velice rychle. Repozitáře obsahují mnoho takových schémat. My se stručně seznámíme alespoň s těmi nejznámějšími.

ebXML

Cílem ebXML[11] je vytvoření jednotného standardu pro bezpečnou výměnu obchodních informací založeného na XML. Tato iniciativa má velké naděje na

[11] http://www.ebxml.org/

úspěch, protože ji zaštiťuje sdružení OASIS a UN/CEFACT. OASIS je etablovaná na poli XML. UN/CEFACT je zase orgán Spojených národů, který již dříve vytvořil standrad UN/EDIFACT.

cXML

Commerce XML[12] je jednoduchý formát určený pro elektronickou výměnu objednávek, katalogů, zpráv apod. Finální specifikace je k dispozici od poloviny roku 1999, takže technologii můžeme používat již dnes.

tpaXML

Výše popsané formáty se většinou snažily popsat formát dat používaný pro komunikaci mezi obchodními partnery. Formát tpaXML (Trading Partner Agreement Markup Language)[13] jde ještě o krok dál. Umožňuje, aby se partneři dohodli, které údaje si budou vyměňovat, jaké se použijí protokoly, bezpečnostní mechanismy, formáty pro výměnu dat, co se stane v případě chyby apod.

Implementace tohoto protokolu umožní, aby se bez nutnosti ručního programování mohly snadno propojit dva systémy různých firem. Všechen kód potřebný pro obsluhu transakcí by se měl vygenerovat ze specifikace uložené v tpaXML.

6.4 Věda a výzkum

Když před deseti lety Web vznikal, sloužil potřebám vědců, kteří si chtěli vyměňovat výsledky svého výzkumu. Tehdy tím byly myšleny články a zprávy, které shrnovaly výsledky jejich práce. XML však umožňuje sdílení výsledků a získaných údajů ve formě, které může být snadno dále využívána. V XML dokumentu můžeme zachytit vývoj teploty na určitém území nebo třeba strukturu složité molekuly. Vědec na druhém konci světa může využít výsledky práce svých kolegů. Nemusí přitom nic pracně ručně opisovat, pouze myší přetáhne soubor z prohlížeče do svého analytického programu.

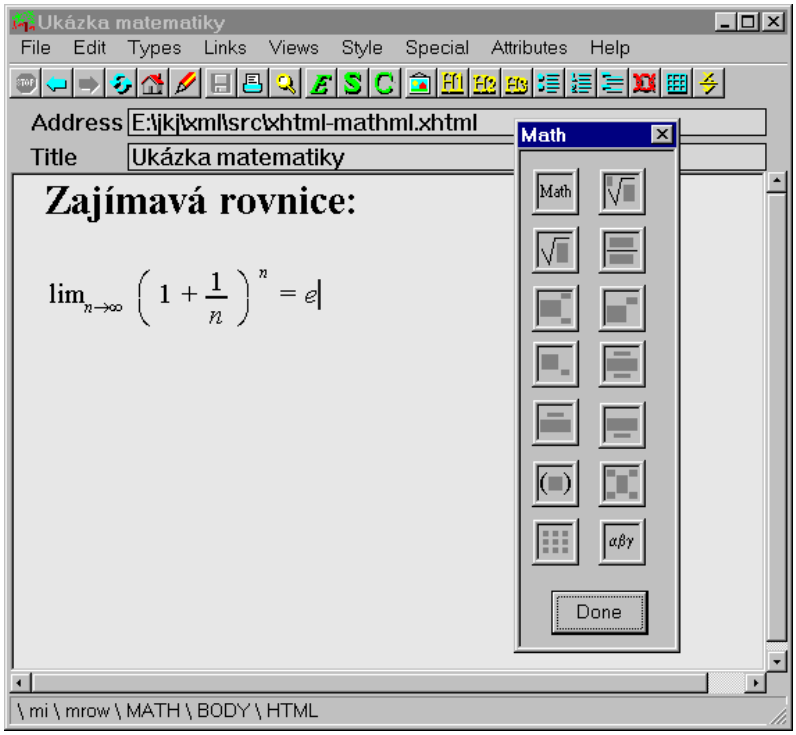
Během posledního roku vznikly datové formáty, které postihují snad každou oblast lidského zájmu. Pokusil jsem pro vás vybrat ty nejrozšířenější a podle mne nejzajímavější.

MathML

Matematika je matka všech věd. Jazyk pro zápis matematických vzorců založený na XML proto vznikl již v dubnu 1998. MathML (Mathematical Markup

[12] `http://www.cxml.org/home/`

[13] `http://www-4.ibm.com/software/developer/library/tpaml.html`



Obr. 6-4: *Amaya obsahuje i vizuální editor vzorců pro MathML*

Language)[14] umožňuje zachytit u vzorců nejen jejich vzhled (výsledné formátování), ale i sémantiku (co skutečně znamenají). Díky tomu se dá MathML použít pro výměnu dat mezi matematickými programy od různých výrobců.

MathML však primárně vzniklo jako jazyk pro zápis matematických vzorců na webových stránkách. Bohužel, výrobci komerčních prohlížečů nespatřují svůj cíl v podpoře matematiky na Webu. Jazyku MathML dnes rozumí například prohlížeč Amaya[15] a podpora je přidávána i do novějších verzí Mozilly.[16] Na obrázku 6-4 je vidět, jak se stránka obsahující MathML zobrazí v prohlížeči.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Ukázka matematiky</title>
  </head>
```

[14] http://www.w3.org/TR/REC-MathML/

[15] http://www.w3.org/Amaya/

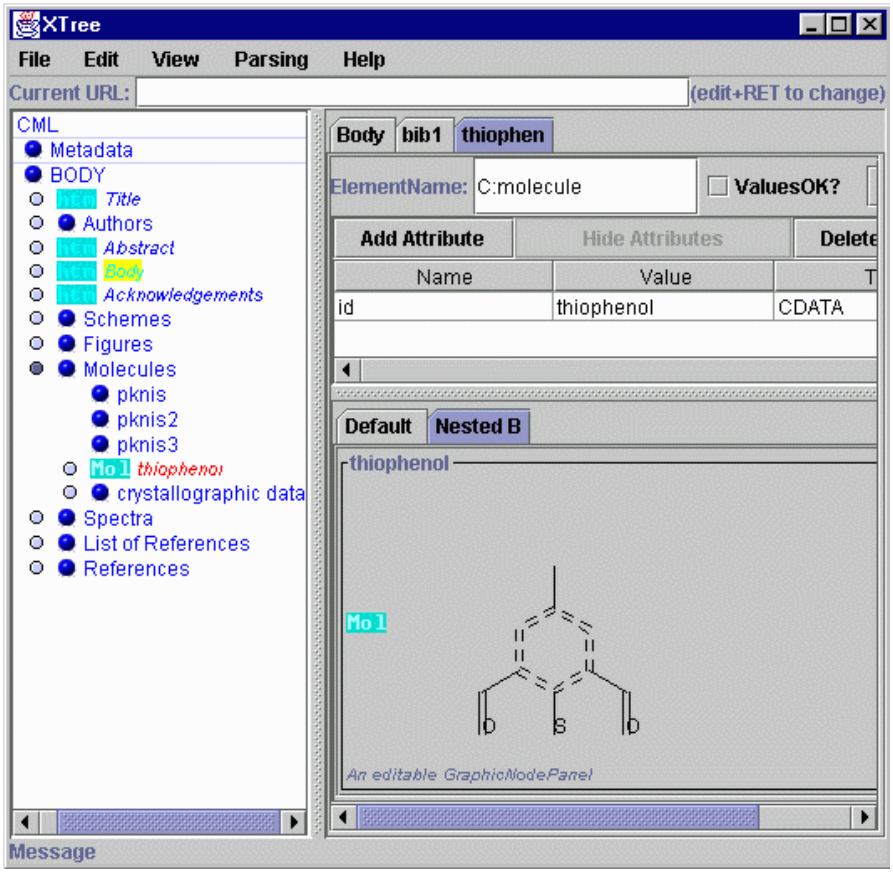
[16] http://www.mozilla.org/

```
<body>
  <h1>Zajímavá rovnice:</h1>
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <mrow>
      <mrow>
        <msub>
          <mo>lim</mo>
          <mrow>
            <mi>n</mi>
            <mo>&RightArrow;</mo>
            <mi>&infin;</mi>
          </mrow>
        </msub>
        <mo>&ApplyFunction;</mo>
        <msup>
          <mrow>
            <mf>(</mf>
            <mrow>
              <mn>1</mn>
              <mo>+</mo>
              <mfrac>
                <mn>1</mn>
                <mi>n</mi>
              </mfrac>
            </mrow>
            <mf>)</mf>
          </mrow>
          <mi>n</mi>
        </msup>
      </mrow>
      <mo>=</mo>
      <mi>e</mi>
    </mrow>
  </math>
  </body>
</html>
```

CML

Jazyk CML (Chemical Markup Language),[17] jak už název napovídá, je určen pro chemiky. Je vybaven nástroji pro zachycení struktury chemických molekul,

[17] http://www.xml-cml.org/



Obr. 6-5: *Jumbo je XML prohlížeč a editor se speciální podporou CML*

lze v něm přímo psát i odborné články. K dispozici je i specializovaný program Jumbo[18] pro práci s dokumenty v CML.

```
<?jumbo:namespace ns="http://www.xml-cml.org"
       prefix="C" java="jumbo.cmlxml.*Node" ?>
<C:molecule id="thiophenol">
  <C:atomArray builtin="elsym">
       C C C C C C S C C O O
  </C:atomArray>
  <C:atomArray builtin="x2" type="float">
       0 0.866 0.866 0 -0.866 -0.866 0.0 0.0 1.732 -1.732 1.732 -1.732
  </C:atomArray>
  <C:atomArray builtin="y2" type="float">
```

[18] http://www.xml-cml.org/jumbo.html

```
        1 0.5 -0.5 -1.0 -0.5 0.5 -2.0 2.0 1.0 1.0 2.0 2.0
    </C:atomArray>
    <C:bondArray builtin="atid1">
        1 2 3 4 5 6 1 4 2 9 6 10
    </C:bondArray>
    <C:bondArray builtin="atid2">
        2 3 4 5 6 1 8 7 9 11 10 12
    </C:bondArray>
    <C:bondArray builtin="order" type="integer">
        4 4 4 4 4 4 1 1 1 2 1 2
    </C:bondArray>
  </C:molecule>
```

XSIL

Zatímco MathML a CML jsou poměrně úzce zaměřené jazyky, XSIL (Extensible Scientific Interchange Language)[19] je obecný jazyk, jenž umožňuje výměnu libovolných dat. XSIL umožňuje popsat data, která se nacházejí přímo v dokumentu nebo jsou dostupná externě. XSIL tak lze použít jako formát pro výměnu dat mezi různými vědeckými pracovišti.

6.5 Vývoj a distribuce softwaru

Softwarové systémy se postupně stávají složitější a složitější. Hledají se proto způsoby, jak co nejlépe podpořit jejich vývoj i údržbu. Do této oblasti pronikají i řešení založená na XML.

XMI

Při vývoji velkých softwarových aplikací se již poměrně dlouho používají nástroje CASE, které usnadňují vývoj. Pomocí různých grafických nadstaveb umožňují nástroje CASE modelovat data a funkce potřebné ve vyvíjeném systému. Umožňují tak udržet pořádek v modelu celé aplikace, navíc často obsahují i nástroje, které dokáží z vytvořeného datového modelu vygenerovat kostry zdrojových kódů nebo databázová schémata. Problém je však v tom, že každý z CASE nástrojů používá vlastní formát a možnost výměny dat mezi těmito aplikacemi je poměrně omezená.

Většina z CASE nástrojů je navíc obvykle úzce svázaná s nějakou metodologií, která říká, jak by se mělo při návrhu informačních systémů postupovat. Před pár lety vznikl společným úsilím výrobců a odborníků jazyk UML (Unified Modeling Language), který se snaží sjednotit do té doby roztříštěné přístupy.

[19] `http://www.cacr.caltech.edu/XSIL`

Jazyk XMI (XML Metadata Interchange)[20] je výsledkem spolupráce skupiny OMG a předních softwarových firem. XMI umožňuje přenášet data UML, popisující návrhy informačních systémů, softwarové komponenty, schémata databází apod. mezi různými systémy.

OSD

Software se dnes skládá z mnoha komponent, většina programů je dostupná pro několik platforem, a jak se v tom má chudák uživatel vyznat.[21] Firmy Microsoft a Marimba proto vytvořily formát OSD (Open Software Description Format),[22] do kterého lze uložit informace potřebné pro instalaci programů na různých platformách. Inteligentní instalační agent pak automaticky stáhne správné komponenty vhodné pro uživatelovu platformu. Ten si jen tiše hoví a nemusí nic složitě instalovat.

XUL

Většina programů se dnes neobejde bez slušivého uživatelského rozhraní. I když nám dnes při generování uživatelského rozhraní pomáhají různí pomocníci, obsažení v moderních vývojových nástrojích, je nakonec vždy uživatelské rozhraní v programu reprezentováno jako poměrně dlouhý a nepřehledný kód. Vývojáři prohlížeče Mozilla si řekli, že mnohem snazší, než psát kód pro uživatelské rozhraní prohlížeče, bude vytvořit speciální jazyk pro popis rozhraní. Jazyk je samozřejmě založen na XML a jmenuje se XUL (XML-based User Interface Language).[23]

XUL má tu výhodu, že je snadno přenositelný mezi různými platformami. Uživatelské rozhraní je definováno pomocí pojmů jako dialogové okno, nabídka, položka nabídky, seznam, vstupní pole apod. Na jednotlivé prvky pak můžeme navázat volání kódu v JavaScriptu nebo v C++. Tím, že kompletní rozhraní Mozilly je v XULu, můžeme jednoduchou modifikací několika souborů změnit vzhled našeho prohlížeče. XUL můžeme samozřejmě použít i v dalších aplikacích ne jen v Mozille. Následující kód v XUL definuje jednoduché menu (výsledné zobrazení si pak můžeme prohlédnout na obrázku 6-6 na následující straně).

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window id="test-window" title="Sample menu" align="vertical"
   xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
```

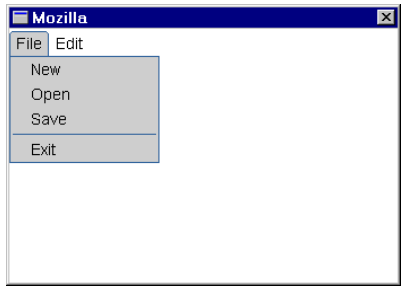
[20] `http://www.omg.org/news/pr99.html#xmi`

[21] O chudákovi programátorovi nemluvě.

[22] `http://www.w3.org/TR/NOTE-OSD.html`

[23] `http://www.mozilla.org/xpfe/xulref/XUL_Reference.html`

```
    <menubar>
      <menu id="file-menu" value="File">
        <menupopup id="file-popup">
          <menuitem value="New"/>
          <menuitem value="Open"/>
          <menuitem value="Save"/>
          <menuseparator/>
          <menuitem value="Exit"/>
        </menupopup>
      </menu>
      <menu id="edit-menu" value="Edit">
        <menupopup id="edit-popup">
          <menuitem value="Undo"/>
          <menuitem value="Redo"/>
        </menupopup>
      </menu>
    </menubar>
  </window>
```



Obr. 6-6: *Zobrazení menu definovaného v XUL*

6.6 Web a Internet

Mnoho aplikací XML se samotným Webem a Internetem souvisí opravdu velice úzce. Na pár příkladech si ukážeme, kde všude se v Internetu používá XML.

SOAP

V distribuovaném prostředí si nevystačíme jen s výměnou informací. Potřebujeme, aby bylo možné distribuovaně po síti využívat i služby. Již poměrně dlouho existuje protokol RPC (Remote Procedure Call), který umožňuje spouštění programů na vzdálených počítačích. Pro běžného smrtelníka to je však protokol příliš složitý.

S tím, jak se svět stává globálním, potřeba komunikace mezi systémy roste. Jednotlivé servery v Internetu si potřebují vyměňovat informace firemní servery si budou předávat objednávky, ze serveru burzy budou číst data zpravodajské servery, aby je zařadily na své stránky apod. Protokol SOAP (Simple Object Access Protocol)[24] umožňuje použít obdobu vzdáleného volání procedur založenou na protokolu HTTP a jazyce XML. Volání procedur je zajišťováno standardním HTTP požadavkem, parametry volání jsou předávány pomocí jazyka XML. Implementace SOAPu je poměrně snadná, protože staví na dobře známých technologiích jako HTTP a XML.

SOAP vzniká pod záštitou IETF (Internet Engeneering Task Force).[25] Navazuje na jednodušší protokol XML-RPC,[26] pro který existuje implementace v mnoha jazycích.

WebDAV

WebDAV (World-Wide Web Distributed Authoring and Versioning)[27] je rozšíření protokolu HTTP, které umožňuje na dálku pracovat se soubory uloženými na webových serverech. Pomocí tohoto protokolu například můžeme s vhodným HTML editorem modifikovat obsah stránek přímo na serveru. Protokol WebDAV používá pro předávání důležitých parametrů jazyk XML, protože XML je snadno rozšiřitelné a navíc má vyřešenou i podporu jazyků jiných než je angličtina.

XBEL

Na mnoha webových stránkách najdeme seznamy oblíbených odkazů jejich autorů. Pokud se nám něčí odkazy líbí a chceme si je přidat do svých záložek, musíme tak učinit většinou ručně. Každý prohlížeč používá svůj vlastní formát pro uchování odkazů a výměna tedy není jednoduchá věc. Možná se však blýská na lepší časy. Jazyk XBEL (XML Bookmark Exchange Language)[28] slouží k uchovávání oblíbených internetových adres. Odkazy můžeme pojmenovávat a třídit do kategorií. Až budou tento (nebo nějaký podobný) formát používat všechny prohlížeče, bude výměna odkazů na oblíbené stránky s kamarády opravdu radostí.

Podíváme-li se, jak je formát XBEL jednoduchý, asi se podivíme, proč něco takového prohlížeče už dávno nepoužívají.

---

[24] `http://www.ietf.org/internet-drafts/draft-box-http-soap-01.txt`

[25] `http://www.ietf.org`

[26] `http://www.xmlrpc.com`

[27] `http://www.ics.uci.edu/pub/ietf/webdav/`

[28] `http://www.python.org/topics/xml/xbel/`

```
<?xml version="1.0"?>
<!DOCTYPE xbel SYSTEM "http://www.python.org/topics/xml/dtds/xbel-1.0.dtd">
<xbel>
  <info>
    <metadata owner="Jirka Kosek"/>
  </info>
  <folder>
    <title>Jirka doporučuje o XML</title>
    <bookmark href="http://www.oasis-open.org/cover/">
      <title>Stránky Robina Covera</title>
    </bookmark>
    <bookmark href="http://www.xml.com">
      <title>Aktuální zpravodajství</title>
    </bookmark>
    <bookmark href="http://www.w3.org/xml/">
      <title>Specifikace XML</title>
    </bookmark>
  </folder>
</xbel>
```

ColdFusion

Pro dynamické generování webových stránek a pro tvorbu internetových aplikací se dnes ponejvíce používají skriptová prostředí jako ASP nebo PHP. Obdobnou funkčnost nabízí i systém ColdFusion[29] od firmy Allaire.[30] Nejedná se však o programovací jazyk s poměrně volnou syntaxí. Všechny příkazy, práce s proměnnými nebo dotazování do databáze se provádí zápisem speciálních elementů přímo do dokumentu.

Následující ukázka obsahuje dokument, který po zpracování systémem ColdFusion bude obsahovat ceník, jehož jednotlivé položky budou získány z databáze.

```
<cenik>
<CFQUERY NAME="NabidkaSeznam" DATASOURCE="katalog">
SELECT * FROM nabidka
</CFQUERY>
<CFOUTPUT QUERY="NabidkaSeznam">
<polozka>
  <nazev>#Nazev#</nazev>
  <cena>#Cena#</cena>
  <dph>#Dazba#</dph>
```

[29] http://www1.allaire.com/documents/cf4/dochome.htm

[30] http://www.allaire.com

```
</polozka>
</CFOUTPUT>
</cenik>
```

6.7 Grafika a multimédia

Ač by se na první pohled mohlo zdát, že textový formát jako XML se pro použití v oblasti počítačové grafiky a multimédií příliš nehodí, opak je pravdou. Existuje několik aplikací, které XML úspěšně používají.

SVG

SVG (Scalable Vector Graphics)[31] je vektorový formát pro dvourozměrnou grafiku, určený zejména pro webové stránky. Tento formát má dnes nejlepší předpoklady stát se standardem pro vektorovou grafiku na Webu. Na jeho vývoji pracuje konsorcium W3C a existuje již i několik implementací. Například od firmy Adobe si můžete stáhnout podporu SVG pro Internet Explorer a Netscape Navigator.[32]

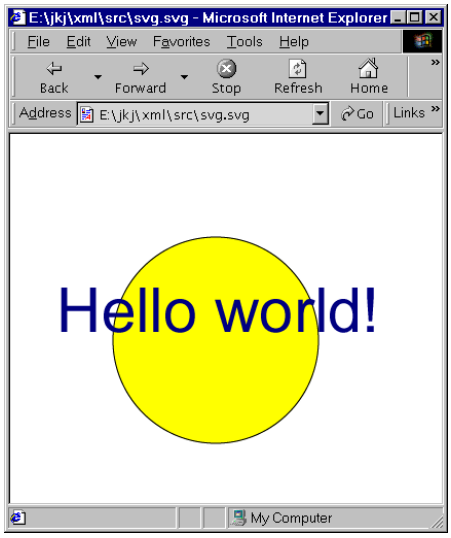
Pro různé vektorové obrázky a mapky je SVG mnohem úspornější než bitmapové formáty jako GIF nebo JPEG. Samotný jazyk není příliš složitý, nicméně se nepočítá s tím, že by někdo grafiku vytvářel ve svém textovém editoru. Pomalu se začínají objevovat exportní filtry pro různé grafické editory, které umožňují ukládání dokumentů do formátu SVG.

Jen pro ilustraci si ukážeme, jak vypadá jednoduchý obrázek v SVG. Na obrázku 6-7 na následující straně pak vidíme zobrazení v prohlížeči.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG July 1999//EN"
                    "http://www.w3.org/Graphics/SVG/svg-19990706.dtd">
<svg width="4in" height="3in">
  <desc>This is a yellow circle with a black outline</desc>
  <g>
    <circle style="fill: yellow; stroke: black" cx="200" cy="200" r="100"/>
    <text x=".3in" y="2in" style="font-size: 60px; color: navy">
      Hello world!
    </text>
  </g>
</svg>
```

[31] http://www.w3.org/TR/WD-SVG/

[32] http://beta1.adobe.com/svgpreview_alpha/SVG/main.html



Obr. 6-7: Zobrazení vektorového obrázku SVG v prohlížeči

SMIL

SMIL (Synchronized Multimedia Integration Language)[33] slouží pro tvorbu multimediálních prezentací. Pomocí jazyka můžeme definovat jednotlivé zdroje audio a video dat, jejich umístění na obrazovce a čas, ve kterém budou jednotlivé části prezentace přehrány. Jazyk je podporován již několika multimediálními přehrávači, včetně tak známých jako je RealNetworks Player.

6.8 XML lze použít opravdu na všechno

Následující sekce vám ukáže, že XML lze použít opravdu na všechno. Není to pouhá další technologie pro příznivce počítačů.

HL7

Organizace Health Level Seven (HL7)[34] se stará o tvorbu standardů pro výměnu dat ve zdravotnictví. Jejím cílem je usnadnit komunikaci mezi jednotlivými zdravotnickými subjekty, které si potřebují vyměňovat informace o pacientech, výsledcích testů, lécích apod. Poslední specifikace protokolů počítají s použitím XML.

[33] `http://www.w3.org/TR/1998/REC-smil-19980615/`

[34] `http://www.hl7.org`

SWAP

Procesy, které probíhají při zpracování obchodních dokumentů, zvláště ve větších firmách, jsou poměrně složité. Objednávky, směrnice, ceníky všechny tyto dokumenty někdo vytváří, někdo jiný připomínkuje a někdo je musí schválit. Během posledních deseti let začalo mnoho firem poměrně úspěšně používat systémy, které automatizují tok dokumentů. V angličtině se používá pojem workflow.

Dnes však potřeba výměny dokumentů a jejich oběhu podle předem daných pravidel přesahuje rámec jedné organizace. Pod záštitou IETF proto vzniká protokol SWAP (Simple Workflow Access Protocol),[35] která umožní spolupráci jednotlivých firemních workflow systémů mezi sebou. Protokol SWAP je rozšířením protokolu HTTP, samotné informace se předávají v XML.

HR-XML

Český překlad anglického human resource (HR) lidský zdroj nezní uchu zrovna lahodně. Pod tímto pojmem moderní doby se však neskrývá nic převratného. Pokud jste zaměstnavatel, znamená pro vás HR přijímaní nových zaměstnanců (nebo také propouštění). A pokud zrovna sháníte práci, nabízíte své lidské zdroje. Aby bylo sdílení informací o volných pracovních místech a poptávce po zaměstnání snadné, vytvořilo sdružení HR-XML[36] několik standardních formátů založenách na XML. Firmy, personální agentury a jednotlivci si tak mohou bez problémů vyměňovat informace o nových pracovních místech a o zájemcích o práci. Formát myslí i na předávání životopisů ve standardním formátu.

Celá aktivita je jistě chvályhodná. Pokud se nezaměstnanost nesníží díky efektivnějšímu fungování pracovního trhu, určitě se sníží díky tomu, že bude potřeba mnoho lidí, aby v jednotlivých firmách a personálních agenturách podporu nového formátu přidali do stávajících systémů.

6.9 Metadata

Pomocí metadat můžeme k dokumentům přidat popisné informace, jako jméno autora, klíčová slova, použitý jazyk apod. Metadata se uplatní především v různých prohledávacích službách, které díky nim mohou zlepšit výsledky vyhledávání a prezentaci výsledků uživateli.

RDF

RDF (Resource Description Framework)[37] je standardem z dílny konsorcia W3C. Definuje standardní způsob, kterým je možné k dokumentům připojit

[35] http://www.ics.uci.edu/~ietfswap/

[36] http://www.hr-xml.org/

[37] http://www.w3.org/TR/REC-rdf-syntax/

metainformace. Praktické využití tohoto standardu znamená zlepšení práce vyhledávacích strojů a agentů, navigačních aplikací a aplikací pro hodnocení obsahu stránek.

Dublin Core

Zatímco RDF definuje obecný mechanismus pro připojení metadat, Dublin Core[38] představuje sadu metainformací, které je užitečné u jednotlivých informačních zdrojů uvádět. K dokumentům se metadata Dublin Core připojují pomocí obecného mechanismu RDF. Metainformace podle Dublin Core dnes umí využívat několik internetových i intranetových vyhledávacích služeb. Následující ukázka obsahuje metainformace o mojí webové stránce zapsané v RDF a vyjádřené pomocí Dublin Core.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns="http://purl.org/dc/elements/1.0/">
 <rdf:Description about="http://www.kosek.cz">
   <title>Téměř vše o WWW</title>
   <description>Domovská stránka Jirky Koska se spoustou informací o
        tvorbě webových stránek a aplikací.</description>
   <author>Jirka Kosek</author>
   <creator>Jirka Kosek</creator>
   <date>2000-02-10</date>
   <format>text/html</format>
   <language>cs</language>
 </rdf:Description>
</rdf:RDF>
```

[38] http://purl.oclc.org/dc/

7. Aplikace podporující XML

XML je jistě skvělý formát pro výměnu a ukládání dat, ale sám o sobě by byl téměř k ničemu. Abychom mohli potenciál XML plně využít, potřebujeme aplikace, které umí s daty ve formátu XML pracovat. Těchto aplikací dnes naštěstí už existuje opravdu hodně, takže při řešení konkrétních požadavků máme z čeho vybírat. V této kapitole se podíváme na typické druhy aplikací, které jsou pro práci s XML potřeba. Asi nebudete potřebovat všechny, ale je dobré vědět, co je k dispozici. U každé skupiny aplikací se stručně seznámíme i s konkrétními produkty, které se dnes používají. Pokusíme se nastínit i obecné požadavky na schopnosti určitého druhu aplikace.

7.1 Prohlížeče

Na Webu se dnes používá jazyk HTML, ale v blízké budoucnosti bude nahrazen jazykem XHTML a dalšími jazyky založenými na XML. Než se tak stane, musí existovat odpovídající množství dostatečně kvalitních prohlížečů, které zvládnou práci s XML dokumenty. Prohlížeč XML bude základem, bez něj by si uživatelé mohli prohlížet jen zdrojové kódy XML dokumentů, což by jistě nebylo příliš pohodlné.

Požadavky na prohlížeč

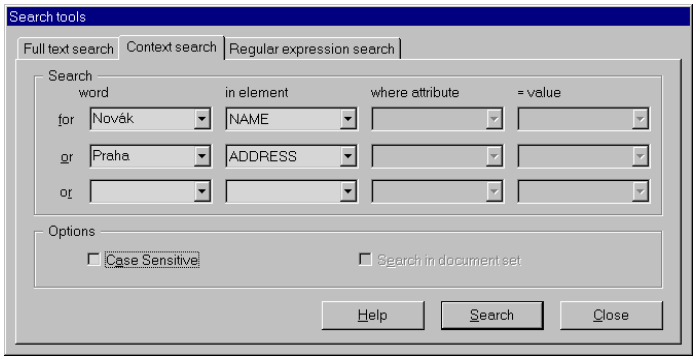
Co by měl takový prohlížeč zvládnout? Předně musí umět zobrazit XML dokument podle pravidel definovaných v připojeném stylu. Prohlížeč by měl podporovat alespoň kaskádové styly (CSS). Pro složitější dokumenty však bude užitečná i podpora stylového jazyka XSL.

Samozřejmostí by měla být podpora všech běžných grafických a multimediálních formátů. Kromě dnes běžně používaných formátů jako GIF, JPEG a PNG přicházejí na scénu i zcela nové technologie. Web dlouho postrádal standardizovaný formát pro přenos vektorové grafiky. Konsorcium W3C nyní pracuje na formátu SVG (Scalable Vector Graphics), který je vektorový. To znamená, že mnoho druhů obrázků (zejména různá schémata a loga) se bude přenášet mnohem rychleji. Zajímavostí je, že formát SVG není binární, ale je to obyčejný XML dokument.

Formátů založených na XML dnes vzniká velké množství. Některé z nich lze zobrazovat pomocí klasických stylů. Jiné jsou však natolik složité nebo staví na odlišném formátovacím modelu, takže pro ně běžné stylové jazyky nestačí. Jako příklad mohou posloužit jazyky MathML (Mathematical Markup Language) a CML (Chemical Markup Language), které slouží k zápisu matematických a chemických vzorců. Je jasné, že tyto formáty asi nebudou podporovat všechny

prohlížeče. Ale je více než pravděpodobné, že pomocí mechanismu, který bude připomínat dnešní plug-iny, bude možné do prohlížeče přidat podporu pro různé další formáty založené na XML.

Důležitá bude i podpora odkazů tedy jazyka XLink. Možnosti tohoto jazyka v mnoha směrech převyšují možnosti odkazů v té podobě, jak je známe dnes. S touto situací se budou muset vypořádat i prohlížeče.



Obr. 7-1: *Možnosti vyhledávání v prohlížeči MultiDoc Pro*

Dobrý prohlížeč by měl nabízet i další funkce, mezi něž patří navigace v dokumentu a kontextové vyhledávání. Pod navigací si můžeme představit například schopnost vygenerovat a zobrazit strom struktury dokumentu, který může posloužit zároveň jako obsah a jako výborná navigační pomůcka. Kontextové prohledávání není nic jiného, než že při hledání textu na stránce můžeme zadat element, ve kterém se má text hledat. Prohledávání tak můžeme omezit například jen na popisy tabulek.

Podpora XML v dnešních prohlížečích

Oba dva přední výrobci prohlížečů ohlásili podporu XML v pětkových verzích svých produktů. V době psaní této knihy byl v této verzi k dispozici Internet Explorer.[1] Netscape zatím ostrou verzi svého prohlížeče neuvolnil, k dispozici je pouze vývojová verze Mozilla.[2]

Oba dva prohlížeče v sobě obsahují XML parser a umějí dokumenty formátovat pomocí CSS stylu. S dokumentem lze rovněž manipulovat pomocí klientských skriptů (JavaScript). Využívá se přitom standardní rozhraní DOM. Internet Explorer navíc podporuje i transformační část stylového jazyka XSL. V době uvedení IE 5.0 na trh bohužel nebylo ještě XSL standardizováno, a tak se

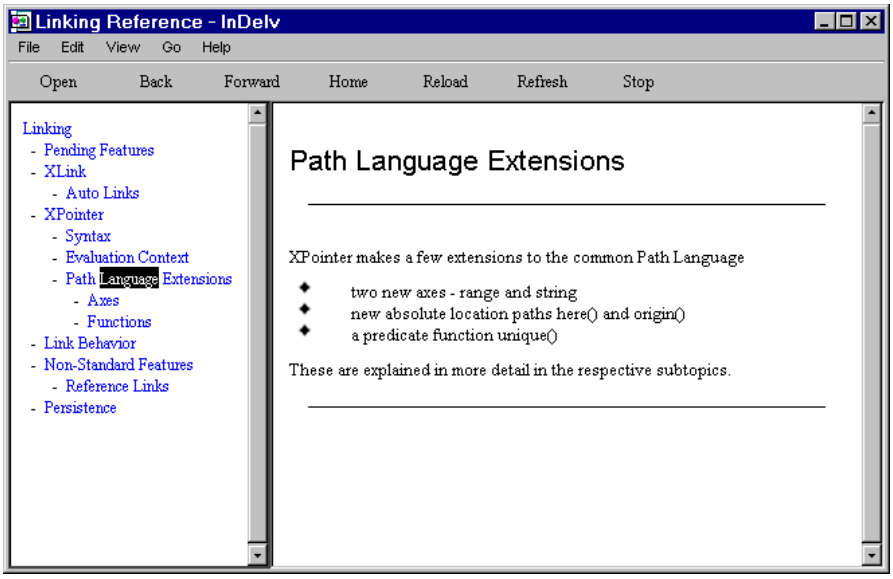
[1] `http://www.microsoft.com/windows/ie/`

[2] `http://www.mozilla.org/`

XSL v Internet Exploreru v některých ohledech liší od standardu. Microsoft se však zavázal, že další verze prohlížečů budou v souladu s posledními standardy.

Ani jeden z prohlížečů zatím nepodporuje všechny možnosti rozšířené tvorby odkazů definované v jazyce XLink. Nabízené možnosti jsou v podstatě na úrovni jednoduchých odkazů, které známe z jazyka HTML.

Prohlížeče s dokonalejší podporou všech XML technologií a lepší možností navigace a vyhledávání budeme muset hledat u jiných výrobců, než jsou Netscape a Microsoft. Uspějeme mezi výrobci tradičních SGML prohlížečů, kteří své programy rozšiřují o podporu XML, a mezi mladými firmami, které se snaží zaplnit nově vzniklou mezeru na trhu. Zajímavým prohlížečem, který staví na základech Mozilly, je DocZilla.[3] Kromě XML zvládá i SGML a mnoho dalších hi-end technologiích.



Obr. 7-2: *InDelv je jeden z prvních prohlížečů, který podporuje XLink a XPointer*

K dispozici jsou i další zajímavé projekty, které jsou však zatím většinou ve stádiu beta verzí. Například prohlížeč s integrovaným editorem InDelv[4] umí zobrazovat dokumenty pomocí XSL stylu, kde přímo interpretuje formátovací instrukce. Nepoužívá tedy XSL pouze pro transformaci do HTML. Navíc podporuje moderní jazyky pro tvorbu odkazů XLink a XPointer. Podporu XLinku obsahuje i prohlížeč HyBrick,[5] který dokumenty formátuje pomocí DSSSL stylu.

[3] `http://www.doczilla.com/`

[4] `http://www.indelv.com/`

[5] `http://www.fujitsu.co.jp/hypertext/free/HyBrick/en/index.html`

Pro samotné nasazení XML na Webu nemusí být špatná nebo neexistující podpora XML v prohlížeči překážkou. Není problém ještě na serveru převést XML dokument pomocí XSL stylu do HTML. Již dnes existují řešení, která umožní automatické zaslání XML dokumentu v podobě vhodné pro klienta prohlížeči se zašle buď HTML, nebo XML kód, pro potřeby mobilních zařízení se dokument převede do WML nebo do jiného úsporného a jednoduchého jazyka.

7.2 Editory

Pokud budeme chtít psát dokumenty v XML, budeme k tomu dozajista potřebovat nějaký vhodný editor. Jelikož je XML dokument vlastně obyčejný textový soubor doplněný o tagy, můžeme k jeho vytváření a následné editaci použít libovolný textový editor. Přijdeme tak však o spoustu užitečných funkcí lepší editor může automaticky kontrolovat strukturu dokumentu. Nedovolí nám pak vytvořit dokument, který není správně strukturovaný či neodpovídá požadovanému DTD nebo schématu. Chytrý editor nám také může napovídat, které elementy a atributy můžeme v daném místě dokumentu použít.

Luxusní editor může být přímo integrován s nějakým formátovacím modulem, který dokáže dokument formátovat pro tisk nebo převést do dalších formátů. Pro někoho může být důležitá i integrace editoru se systémem pro management dokumentů. Při větším objemu dokumentů je ukládání do souborů poměrně nepohodlné. Lepší je dokumenty ukládat do nějakého systému pro jejich správu, který bývá postaven nad databází a často v sobě zahrnuje i funkce pro řízení oběhu dokumentů apod.

XML editory můžeme rozdělit do několika skupin, podle toho, nakolik je uživatel vystaven styku se samotným jazykem XML. Nejjednodušší editory zobrazují přímo zdrojový kód XML s tím, že mohou tagy zvýraznit jinou barvou pro odlišení od textu. Některé editory pouze zvýrazní syntaxi, jiné v tomto režimu automaticky kontrolují strukturu dokumentu a nabízejí uživateli elementy a atributy, které může na určitém místě dokumentu použít. Pokročilejší editory již pracují v částečném WYSIWYG režimu, kdy je text vidět zformátovaný pomocí stylu, ale jsou v něm vyznačeny jednotlivé tagy. Poslední stupeň již před uživatelem kód jazyka XML úplně schová. Vkládání tagů je pak plně v režii editoru, který může podle nějakých pravidel mapovat jednotlivé vizuální styly na XML elementy.

Výše popsané editory se hodí spíše na editování textových dokumentů. Pokud však do XML ukládáme především databázová nebo hodně strukturovaná data, může nám více vyhovovat editor, který dokument zobrazuje jako hierarchickou stromovou strukturu. Pro speciální aplikace se může vyplatit vytvoření specializovaného programu, který umožní velice efektivně zpracovávat data vyhovující určitému konkrétnímu DTD. Příkladem budiž například editory pro zápis matematických vzorců v MathML.

Pro masové použití XML se nejvíce hodí editory, které pracují ve WYSIWYG režimu a hodně tak připomínají klasický textový editor, jako je třeba Microsoft Word. Důkazem správnosti této cesty budiž například HTML editor FrontPage, který si mnoho uživatelů oblíbilo právě proto, že v něm nemusejí vůbec přijít do styku s HTML kódem. FrontPage základní formátovací příkazy automaticky převádí do odpovídajících struktur jazyka HTML. Podobně je tomu i u vyspělých XML editorů.

V následujících odstavcích se podíváme na nejrozšířenější XML editory, které uživateli nabízejí více než jen zvýrazňovaní syntaxe.

Adept a Epic

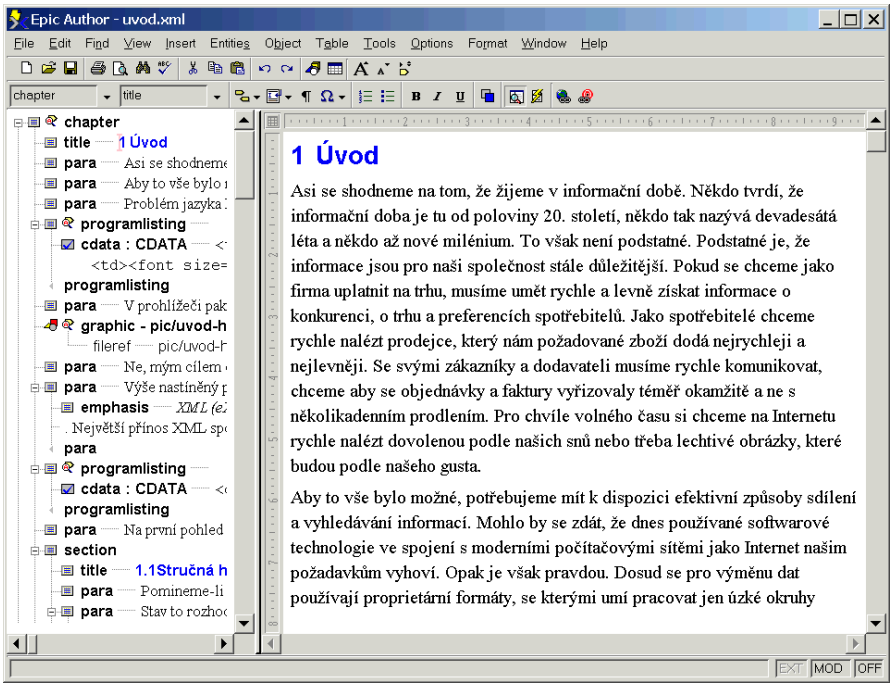
Firma ArborText[6] patří v oblasti elektronického publikování založeného na značkovacích jazycích ke špičce. Původně nabízela editory a další nástroje pro práci se SGML. Dnes její produkty samozřejmě podporují i XML. Firma nabízí celou řadu editorů, které používají stejné jádro, ale nabízejí různou funkčnost. Ty nejjednodušší umožňují pouze vytváření dokumentů pomocí předem definovaných DTD, ty nejpokročilejší naopak umožňují plně přizpůsobit editor danému DTD, včetně definice různých stylů pro tisk a konverze do dalších formátů. Různých verzí editorů je na můj vkus až příliš, ale každý si alespoň může vybrat. Hlavní řady editorů nesou názvy Adept a Epic a jsou dostupné pro Windows a Solaris.

Velkou výhodou všech editorů je, že podporují libovolné DTD. Výrobce navíc dodává k editoru již několik přednastavených DTD, jejíž používání je asi tak stejně složité jako používání šablon v kancelářských textových editorech. Epic je přizpůsoben zejména pro psaní dokumentace a tak obsahuje několik DTD založených na DocBooku. Samozřejmě obsahuje i standardní verzi DocBooku a některá další DTD například HTML 4.0 a XHTML. Zajímavá je však verze DocBooku od ArborTextu, která obsahuje některá zajímavá rozšíření. Samozřejmostí je bezproblémové zařazování grafiky v několika formátech a vizuální editace tabulek. Příjemně překvapí možnost zařazování matematických vzorců přímo do dokumentu. K jejich vytvoření můžeme použít zabudovaný editor rovnic.

Samotné prostředí editoru je velice příjemné. Kromě WYSIWYG režimu nabízí editor i hierarchický pohled na strukturu dokumentu. Dokument můžeme editovat v libovolném z těchto pohledů. Ve WYSIWYG zobrazení lze navíc zapnout zobrazování tagů, čímž získáme přesný přehled o struktuře dokumentu.

Editor samozřejmě hlídá strukturu dokumentu a ke vložení nám nabízí jen platné elementy. Editace atributů je také velice pohodlná a probíhá pomocí dialogových oken. Pomocí zabudovaného jazyka a vývojového prostředí můžeme vytvořit i vlastní dialogová okna pro pohodlné zadávání atypických dat.

[6] `http://www.arbortext.com`

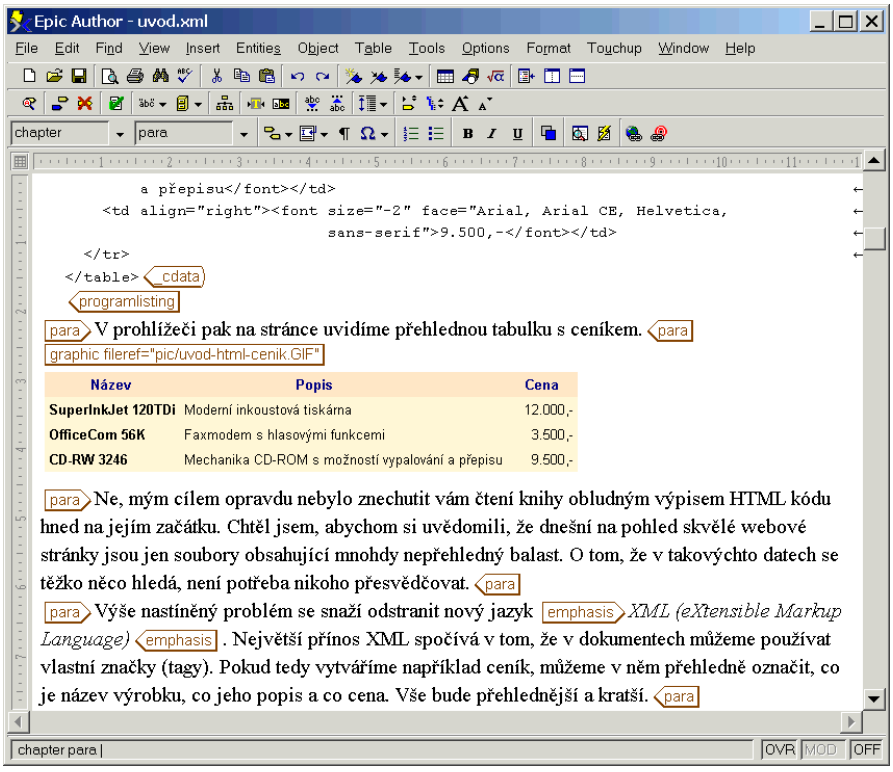


Obr. 7-3: *Epic Author 2.0 patří k editorům, které toho nabízejí opravdu hodně*

Epic má v sobě zabudovaný i výkonný formátovací modul. Pro definici vzhledu dokumentu se používá stylový jazyk FOSI, který má svá zlatá léta již za sebou, ale většinu požadavků s přehledem zvládne. Vytvořené dokumenty lze tedy přímo tisknout, nebo z nich můžeme vygenerovat PostScript. Pokud máme nainstalován Adobe Acrobat, můžeme z dokumentu generovat i PDF.

Editor lze pomocí konfiguračních souborů a zabudovaného jazyka ACL přizpůsobit k obrazu svému. Pro každé DTD tak můžeme definovat položky, které se objeví v nabídce. Pro rozšířenou verzi DocBooku, který je s Epicem dodáván, najdeme v nabídce rovnou i možnost generování webového sídla z dokumentu. Z editoru lze volat i další programy, například různé procesory pro převod do dalších formátů jako je Jade a XT (oba dva jsou standardní součástí instalace). Zabudovaný jazyk ACL můžeme použít i pro manipulaci s dokumentem pomocí standardního rozhraní DOM.

Zajímavou funkcí Epicu je možnost generování podkladů pro CD-ROM z dokumentu. Epic vytvoří vše, včetně instalačního programu. Dokumenty takto publikované na CD-ROMu samozřejmě zahrnují i možnost fulltextového prohledávání. Neméně zajímavá je i možnost importu a exportu do Wordu. Dodáván je nástroj, který umožňuje definovat mapování wordových stylů na jednotlivé elementy.



Obr. 7-4: Většina editorů umí nabídnout i hybridní WYSIWYG režim, ve kterém jsou viditelné i jednotlivé tagy

K dispozici je mnoho rozšiřujících modulů, které umožňují pracovat s dokumenty uloženými v databázi namísto v souborech, provádět v textu revize několika uživateli (redlining) atd.

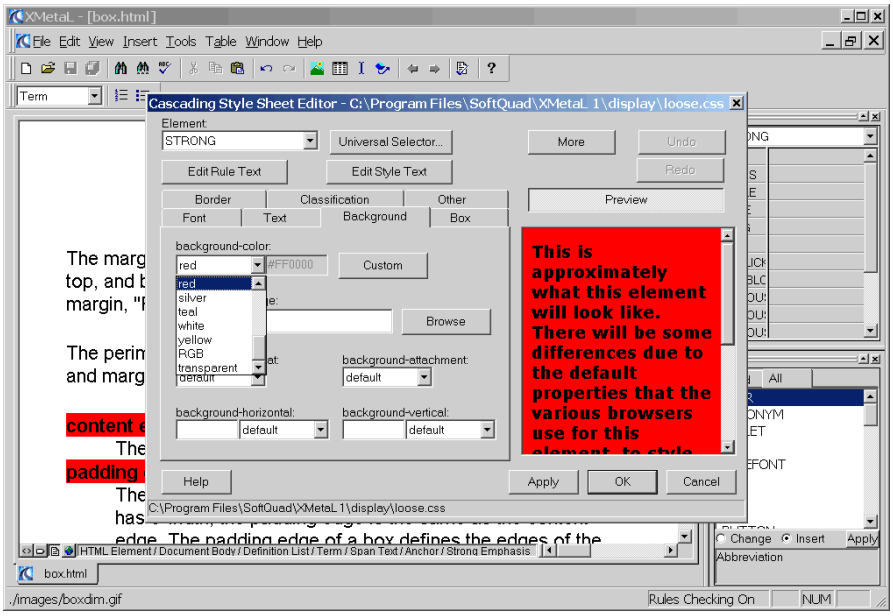
Editor obsahuje kontrolu pravopisu a tezaurus pro několik jazyků, čeština mezi nimi však chybí. Naštěstí však editor obsahuje alespoň české vzory pro dělení slov, takže se nemusíme bát, že by nám špatně fungovalo. Co se týče podporovaných kódování, pro češtinu můžeme použít UTF-8, ISO 8859-2 a windows-1250. Verze Epicu, kterou jsem měl k dispozici, bohužel neuměla rozpoznat kódování uvedené v XML deklaraci, budoucí verze by tuto chybu neměly obsahovat.

XMetaL

Editor XMetaL[7] pochází z dílny firmy SoftQuad, která se už poměrně dlouhou dobu věnuje vývoji nástrojů pro práci s SGML a HTML. XMetaL je WYSIWYG

[7] `http://www.xmetal.com`

editor, který zvládá editaci XML a SGML dokumentů. Cílem XMetaLu je co nejvíce zjednodušit editaci dokumentů, ale přitom zachovat plnou flexibilitu XML.



Obr. 7-5: *XMetaL k zobrazování používá styly CSS a obsahuje i editor pro úpravu stylu*

XMetaL umožňuje vytváření dokumentů, které vyhovují libovolnému DTD. Standardní nabídka DTD je však velice chudá kromě HTML je nabízeno pouze DTD Journalist, které je okleštěnou verzí DocBooku určenou pro psaní článků. Naštěstí lze velice jednoduše editor přizpůsobit pro libovolné DTD. Pro urychlení práce můžeme často používaná DTD uložit do binárního předkompilovaného tvaru.

Vzhled dokumentu v editoru můžeme definovat pomocí kaskádových stylů CSS. To je velice příjemné, protože stejný styl lze použít i v prohlížečích. CSS však nemají dostatečné prostředky pro definici vzhledu stránky pro potřeby DTP a tak je XMetaL pouze šikovný editor, pro formátování dokumentu musíme použít samostatný program.

Kaskádové styly nemusíme vytvářet ručně, XMetaL má pro ně zabudovaný poměrně komfortní editor. Kromě plného WYSIWYG režimu jsou k dispozici ještě další dva editační režimy. Jednak si můžeme nechat ve WYSIWYG režimu zobrazit hranice jednotlivých tagů, dále pak můžeme editovat přímo zdrojový text dokumentu. Čtvrtým pohledem na dokument je zabudovaný prohlížeč, který využívá komponentu Internet Exploreru, takže dokument je zobrazen

úplně stejně jako v prohlížeči. Verze XMetaLu 1.2 má v sobě již zabudovaný i XSL procesor XT, takže je možné provádět náhled dokumentu zformátovaného pomocí XSL.

Editor samozřejmě podporuje vizuální editaci tabulek a vkládání obrázků. Podporovány jsou nejrozšířenější bitmapové formáty. Editor průběžně kontroluje správnost dokumentu a nabízí vkládání pouze elementů platných v daném kontextu a editaci atributů pomocí dialogových oken. Pohodlně lze vkládat i entity a různé speciální symboly.

Editor je velice dobře konfigurovatelný. S dokumentem můžeme manipulovat pomocí rozhraní DOM, k dispozici jsou skriptové jazyky VBScript a JScript. XMetaL podporuje rozhraní ActiveX Scripting, takže lze doplnit libovolný další skriptový jazyk např. PerlScript.

XMetaL rovněž podporuje různé repozitáře pro ukládání entit, takže nejsme odkázáni pouze na ukládání do souborů. Součástí editoru je i korektor pravopisu a tezaurus, čeština mezi podporovanými jazyky však chybí. XMetaL bohužel podporuje pouze Windows, verze pro další platformy neexistují.

Největší problém XMetaLu však spočívá v jeho podpoře kódování. XMetaL podporuje pouze kódování ISO 8859-1, které je určené pro západoevropské jazyky. České znaky v tomto kódování nenajdeme, a proto je stávající verze XMetaLu pro editování českých textů nepoužitelná. Podle zástupců firmy by měla další verze XMetaLu interně používat Unicode a podporovat tak všechna možná kódování.

FrameMaker+SGML

FrameMaker+SGML[8] je verze DTP systému FrameMaker rozšířená o podporu SGML a XML. Hlavní zaměření tohoto produktu je produkce kvalitního tištěného výstupu z XML dokumentů. FrameMaker proto nabízí bohaté formátovací vlastnosti, možnost tvorby rejstříků apod. Samozřejmě lze dokumenty konvertovat i do HTML a PDF.

WordPerfect

Pro většinu těch, kdo pamatují zlatý věk MS-DOSu, je WordPerfect synonymem pro textový editor, který v mnoha směrech překonal své konkurenty. Po příchodu Windows už tak populární nebyl, ale v některých oblastech si stále udržoval technologický náskok. WordPerfect ve verzi 8.0 přinesl uživatelům kancelářského textového editoru podporu SGML. V poslední verzi 9.0 (je součástí WordPerfect Office 2000[9]) je kromě SGML podporováno i XML.

Editor tak vlastně obsahuje dva režimy, jeden klasický a druhý pro práci s XML/SGML dokumenty. S editorem je dodáváno několik rozšířenějších DTD

[8] `http://www.adobe.com/products/framemaker/prodinfosgml.html`

[9] `http://www.corel.com/Office2000/index.htm`

včetně DocBooku. Škoda jen, že k DTD se nedodávají žádné styly definující zobrazení – ty si musíme vytvořit sami. WordPerfect nepoužívá žádný standardní stylový jazyk. Ve speciálním programu se musí každé DTD, které chceme používat, zkompilovat do šablony. Do ní je možné přidat ještě definice vzhledu jednotlivých elementů. K dispozici máme bohaté formátovací možnosti WordPerfectu, takže výsledkem může být typografický kvalitní výstup.

Editor nabízí tři druhy pohledů na dokument, jak bývá zvykem – WYSIWIG, WYSIWYG kombinovaný s tagy a stromové zobrazení hierarchie elementů. WordPerfect samozřejmě hlídá validitu dokumentu a nabízí doplňování elementů. WordPerfect si bez problémů poradí i s tabulkami a obrázky.

Přestože není WordPerfect 9 k dispozici v lokalizované české verzi, umožňuje i jeho anglická mutace zápis dokumentů s českými znaky. Pro ukládání si můžeme vybrat kódování UTF-8 nebo UTF-16. Bohužel nejsou podporována jednobajtová kódování jako např. ISO 8859-2.

Doufejme, že i ostatní výrobci kancelářských balíků si brzy uvědomí potenciál XML a zařadí jeho podporu do svých editorů.

Microsoft Office 2000 a XML

Mnoho uživatelů se ptá, zda a jak Office 2000 podporuje XML. Microsoft tvrdí, že jeho Office 2000 XML podporuje. Je však podpora a podpora. Office 2000 (Word 2000) nepodporuje XML v tom smyslu, že by uměl editovat dokumenty vyhovující nějakému DTD. Word 2000 tedy v žádném případě není XML editor.

Neznamená to však, že Microsoft lže. Office 2000 skutečně formát XML používá, ale pouze pro své potřeby. Office 2000 umožňuje ukládání dokumentů ve formátu HTML místo klasických formátů DOC a XLS, narozdíl od předchozích verzí se při ukládání do HTML neztrácejí žádné informace. Formát HTML je tak funkčně ekvivalentní s nativními formáty. Aby šlo do formátu HTML uložit všechny přídavné informace, používá se místo HTML ve skutečnosti XML a jmenné prostory. Jako implicitní jmenný prostor je zvoleno HTML 4.0. Ostatní jmenné prostory obsahují speciální elementy – pro vektorovou grafiku, pro uložení informací o dokumentu, pro uložení speciálních formátovacích vlastností apod. Starší prohlížeče přídavné jmenné prostory ignorují a vidí pouze HTML kód, který zobrazí. Novější verze Internet Exploreru a programy z Office 2000 ale rozumí všem elementům.

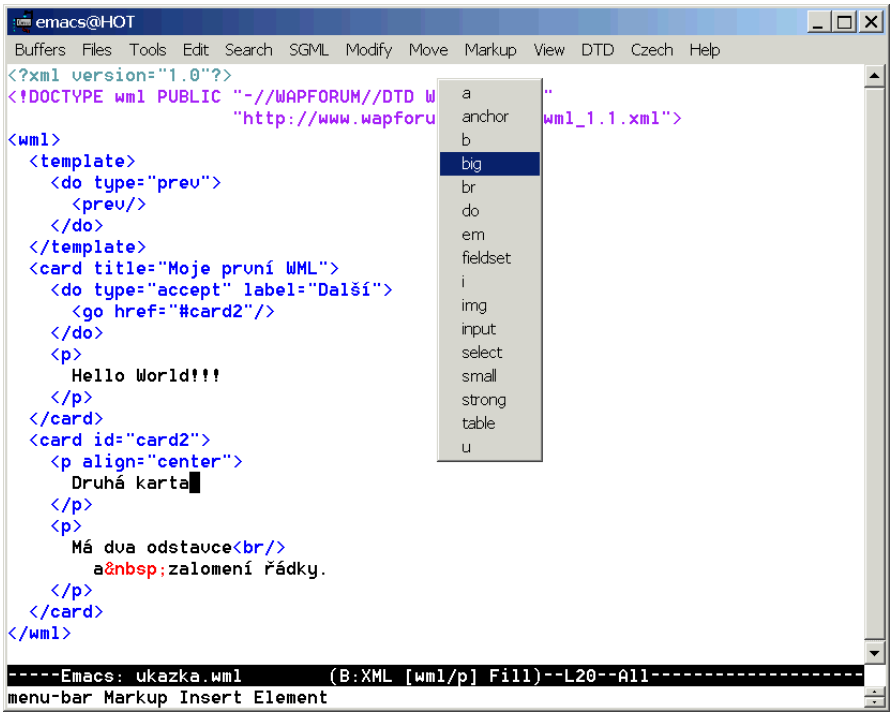
Emacs a PSGML

Emacs[10] je bezpochyby kultovním textovým editorem. Je to jeden z prvních programů, který byl šířen pod GNU licencí – tedy zadarmo včetně zdrojových

[10] `http://www.emacs.org/`

textů. Díky tomu je dnes Emacs dostupný na většině používaných platforem. Původně byl vyvinut pro Unix, kde může běžet v textové konzoli i jako X-Window aplikace. Dnes jsou k dispozici i verze pro Windows[11] a MS-DOS.

Asi největší zbraní Emacsu je zabudovaný interpret jazyka Lisp, pomocí kterého lze upravovat činnost editoru a psát makra. V tomto jazyce je pro něj napsáno nepřeberné množství modulů emailový klient, FTP klient, čtečka diskusních skupin, webový prohlížeč, vývojová prostředí pro mnoho programovacích jazyků a mnoho dalšího. Pro nás je však důležité, že existuje balík PSGML,[12] který z Emacsu udělá plnohodnotný SGML/XML editor.



Obr. 7-6: *Emacs je velice flexibilní pomocí PSGML a jednoho DTD jsme z něj rázem udělali editor WML stránek*

Vzhledem k tomu, že dnešní verze GNU Emacsu zatím není WYSIWYG editor, probíhá editace XML dokumentů v jejich zdrojovém tvaru. PSGML režim však nabízí mnoho nástrojů, jež nám editaci usnadní. Především jsou dokumenty zobrazovány se zvýrazněnou syntaxí značkování má jinou barvu než samotný text dokumentu, což usnadňuje orientaci.

[11] `http://www.gnu.org/software/emacs/windows/ntemacs.html`

[12] `http://www.lysator.liu.se/projects/about_psgml.html`

PSGML samozřejmě umí pracovat s DTD a díky tomu nám během psaní umí nabídnout elementy, které může v daném místě použít. U každého elementu lze snadno editovat i atributy. PSGML má zkrátka pořád přehled o tom, kde v dokumentu právě jsme. Lze se proto snadno pohybovat mezi elementy a dokonce si můžeme obsah zvolených elementů schovat při editování dlouhých dokumentů se pak v textu snáze orientujeme. PSGML umí automaticky zdrojový kód přehledně formátovat. Z DTD si pamatuje i všechny definované entity, takže pokud si na nějakou nemůžeme vzpomenout, stačí si ji vybrat v nabídce. PSGML si poradí i s editací dokumentu, který je uložen ve více souborech.

PSGML má vestavěn jednoduchý parser, který obvykle pro kontrolu správnosti dokumentu stačí. Velice pohodlně lze volat další aplikace jako externí parsery a různé formátovací programy (např. Jade).

Emacs je ve spojení s PSGML opravdu výborný nástroj. Asi po něm sáhnete v případě, kdy nebudete mít prostředky na nákup dříve zmíněných poměrně drahých WYSIWYG editorů. Z počátku vám možná nebude příliš vyhovovat, protože jeho ovládání je poněkud odlišné než to, na které jste zvyklí z dnešních aplikací. Emacs přece jen nějaký ten pátek pamatuje, a i když se neustále vyvíjí, ovládání se kvůli zpětné kompatibilitě nemění. Popis práce v Emacsu včetně popisu PSGML režimu naleznete v knize *GNU nástroje pro tvorbu WWW stránek* [27].

XML Notepad

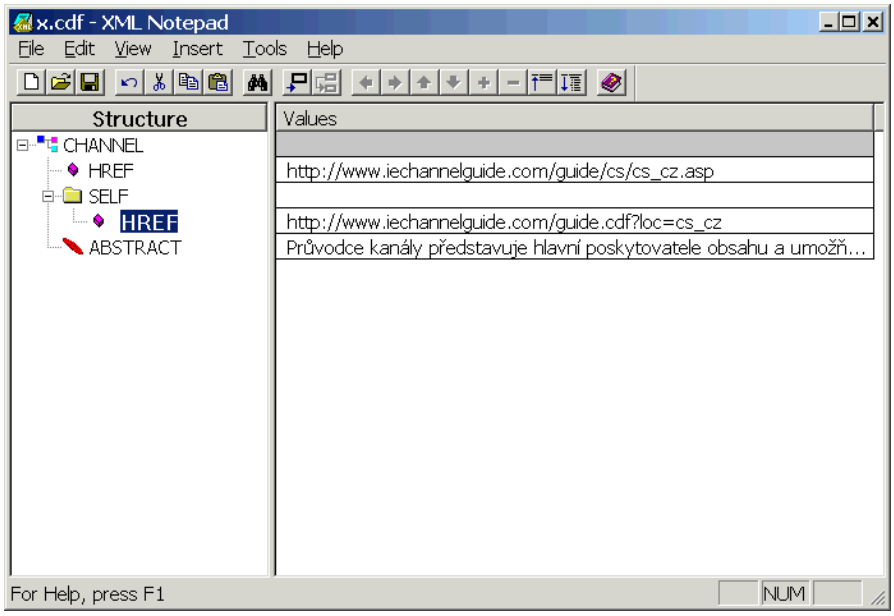
Všechny dosud popisované editory byly komfortní a hodily se pro editaci dokumentů reprezentujících i rozsáhlé texty. Pokud však někdy potřebujeme rychle vytvořit jednoduchý XML dokument, který slouží spíše pro ukládání strukturovaných dat, než dlouhých textů, můžeme použít Microsoft XML Notepad.[13] Editor zobrazuje dokument jako hierarchickou strukturu elementů a atributů, jejichž hodnoty můžeme editovat.

XML Notepad je na stránkách Microsoftu k dispozici zdarma. Editor umí provádět validaci dokumentu oproti DTD a schématům, ale pouze při načítání dokumentu. Během editace můžeme vkládat libovolné elementy a atributy, editor pouze hlídá, aby byl vytvořený dokument správně strukturovaný. O jeho validitu se již nestará.

7.3 Systémy pro správu dokumentů

Pokud produkujeme velké množství dokumentů, které připravuje více lidí najednou, je nemyslitelné, aby se všechny dokumenty ukládaly na nějakém sdíleném síťovém disku v souborech. Je potřeba zajistit přístupová prává k jednotlivým dokumentům a jejich částem, velké dokumenty je nutné rozdělit na menší části,

[13] `http://msdn.microsoft.com/xml/notepad/`



Obr. 7-7: *XML Notepad se hodí jen na editaci velice jednoduchých dokumentů*

které budou moci jednotliví uživatelé editovat nezávisle na sobě. Programům, které popsané funkce nabízejí, se obvykle říká systémy pro správu dokumentů (DMS – Document Management Systems).

DMS obvykle ukládají dokumenty do relačních nebo objektových databází jako Oracle, MS SQL Server, Object Store apod. Obvykle lze nastavit, jak velké kusy dokumentu (kapitoly, podkapitoly, odstavce) budou ukládány jako samostatný objekt, který lze editovat. Pokud vše rozumně navrhneme a rozdělíme, můžeme některé části dokumentů opakovaně používat a tím si ušetřit práci s upravováním dokumentů na více místech.

Aby bylo možno s dokumenty uloženými v DMS pracovat, nabízejí jejich výrobci rozšíření pro jednotlivé XML editory. Editor pak umí načítat a ukládat dokumenty přímo z DMS. Pokud to potřebujeme, může pro nás DMS snadno evidovat jednotlivé verze dokumentů, pamatovat si, kdo je kdy změnil a mnoho dalšího.

Mnohé DMS obsahují i nástroje pro work-flow dokumentů. Můžeme tak specifikovat, která firemní oddělení nebo kteří pracovníci musí požadovaný dokument schválit nebo upravit. DMS požadovaný tok dokumentů pomáhá automatizovat a hlídá, aby nebyl nějaký článek při cestě dokumentu vynechán.

Jednotlivé DMS se mezi sebou navzájem liší, ale často obsahují ještě další moduly. Většinou je součástí DMS prohlížeč a navigátor, který umožňuje snadné prohlížení uložených dokumentů. Obvykle je zařazen i nějaký vyhledávací sys-

tém, některé DMS jsou přímo integrovány s formátovacími programy, které umějí dokumenty formátovat pro tisk nebo převést do dalších formátů vhodných pro publikování na Webu.

Mezi nejznámější DMS patří například SigmaLink,[14] Life*CDM,[15] BladeRunner,[16] Astoria,[17] Poet Content Management Suite,[18] MultiDoc Pro[19] a SIM.[20]

7.4 Vyhledávací nástroje

Dnes jsou na Internetu využívány zejména techniky fulltextového vyhledávání. Schopnosti fulltextových technologií však narážejí na velký objem informací, které se na Webu nacházejí. Pokud tedy zadáme nějakému fulltextovému vyhledávači slovo, které nás zajímá, dostaneme jako odpověď tisíce stránek.

Pokud by byly dnešní fulltextové vyhledávače schopné využít přídavné informace uložené v XML dokumentech, byla by možnost efektivního vyhledávání na Internetu opět o něco pokročilejší. Vyhledávací servery by mohly lépe rozpoznat důležitost dokumentů podle částí textu, kde se hledané slovo vyskytuje. Dokument, který dané slovo obsahuje v nadpisu, je jistě více relevantní, než když ho obsahuje pouze v nějaké poznámce pod čarou. Zlepšení přinese i kontextové vyhledávání, které umožní u hledaného slova zadat, zda se jedná o jméno člověka, jméno ulice nebo popis fotografie.

Bez toho, aby počítače porozuměly přirozenému jazyku, nebude vyhledávání informací nikdy dokonalé. Nicméně již dnes se experimentuje s technikami, které mohou lecčemu pomoci. Problém kontextového vyhledávání spočívá v tom, že v různých DTD a schématech se pro označování téhož využívají elementy s různými názvy. Pokročilé vyhledávače proto budou obsahovat slovníky, které umožní překlad významu značek mezi několika DTD.

Internetové vyhledávací služby podporující XML

Na Internetu je dnes k dispozici velké množství vyhledávacích služeb. Jejich společným znakem je, že jsou převážně založeny jen fulltextově indexují textový obsah stránky. Nestarají se však o strukturu dokumentu a o význam jednotlivých částí textu. Na Internetu se však již začínají objevovat stránky, které jsou

[14] http://www.step.de/sigmalink.htm

[15] http://www.corena.dk

[16] http://www.xmlecontent.com/products/brintro.htm

[17] http://www.chrystal.com/products/astoria/astoria.htm

[18] http://www.poet.com/products/cms/cms.html

[19] http://www.citec.fi/company/it/mdp/index.html

[20] http://www.simdb.com/

dostupné pouze v XML a logicky tak začaly vznikat vyhledávací služby, které podporují kontextové vyhledávání.

Služba GoXML[21] se snaží kontextové prohledávání učinit co nejvíce uživatelsky přátelské. Po zadání hledaných slov se vám zobrazí stránka s nalezenými dokumenty. Kromě toho se zobrazí seznam názvů elementů, ve kterých se hledaná slova vyskytují. Pokud si vyberete nějaký z nabízených elementů, ve výsledku se objeví jen dokumenty, kde jsou hledaná slova v daném elementu. Nejedná se tedy o nic převratného, ale pro rychlé upřesnění výsledků to stačí. GoXML nabízí i zajímavou službu, kdy se dotaz serveru zašle v podobě XML dokumentu a seznam výsledků dostaneme zpět opět v podobě XML dokumentu.

Experimentální služba IBM xCentral[22] slouží spíše než pro vyhledávání v XML dokumentech k vyhledávání informací o XML. Můžeme si totiž vybrat, zda chceme prohledávat dokumenty, DTD, archivy konferencí o XML nebo jiné materiály vztahující se k XML.

Vyhledávače pro £osobní použití¥

Nejméně stejně důležité jako internetové vyhledávače jsou systémy, které umožňují prohledávání XML dokumentů, které máme na svém počítači nebo třeba na firemním intranetu. Již dnes existuje poměrně široká nabídka nástrojů, které pro tento účel můžeme použít.

Pokud dokumenty ukládáme do databáze, máme většinou vyhráno. Většina dnešních databázových serverů již nějakou dobu obsahuje možnosti fulltextového vyhledávání. S příchodem XML postupně výrobci databází rozšiřují funkčnost hledání tak, aby šlo zadávat i kontextové dotazy. Nové možnosti lze obvykle využít pomocí nových příkazů, které rozšiřují jazyk SQL (Structured Query Language) používaný v relačních databázích na dotazování.

Průkopníkem v podpoře XML v databázích je Oracle, ale ani IBM se svojí DB2 a Microsoft s SQL Serverem nespí a podporu XML ve svých produktech neustále zlepšují.

Ne vždy máme to štěstí (smůlu;), že všechny dokumenty najdeme v databázi nebo na svém Intranetu. Zajímavé možnosti nabízí program Xdex,[23] který umožňuje indexování a následovné vyhledávání v XML dokumentech. Indexované dokumenty přitom mohou být na mnoha místech. Xdex umí indexovat dokumenty uložené na WWW a FTP serverech, na síťových i lokálních discích. Lze jej požádat i o to, aby indexoval dokumenty uložené v databázi.

Mezi unikátní vlastnost Xdexu patří možnost vytvoření tezauru pro názvy elementů. Pokud indexované dokumenty odpovídají různým DTD nebo schématům, ale některé elementy mají stejný význam, můžeme říci, které elementy

[21] `http://www.goxml.com`

[22] `http://xcentral.alphaworks.ibm.com/cgi-bin/newbasic.pl`

[23] `http://www.xmlindex.com/`

jsou vzájemná synonyma. Kontextové hledání pak hledá ve všech elementech s daným významem.

Rozhraní Xdexu je webové, vyhledávač obsahuje i XSL procesor, takže lze dokumenty pro zobrazení převést pro starší prohlížeče do HTML.

Všechny předchozí vyhledávače fungovaly na principu, kdy nejsou prohledávány přímo originální dokumenty, ale pomocná datová struktura, tzv. index, který hledání výrazně urychluje. Index se však musí neustále aktualizovat, aby odrážel skutečný obsah dokumentů. Navíc je index obvykle stejně velký (ne-li větší) než indexované dokumenty, takže dochází ke zbytečnému plýtvání místem. Pokud potřebujeme kontextové prohledávání použít jen občas pro nějakou menší skupinu dokumentů, může se nám hodit program sgrep.[24] Tato jednoduchá utilitka pracuje podobně jako známý program grep, s tím rozdílem, že můžeme vytvářet kontextové dotazy, které zadané fráze hledají jen v určitých částech dokumentu.

Dotazovací jazyky

Všechny výše popsané druhy prohledávačů používaly vlastní jazyk pro zadávání dotazů nad XML dokumentem. Často je tento jazyk od uživatele ještě odstíněn nějakým příjemným uživatelským rozhraním. Pro všechny by však jistě bylo výhodné, kdyby existoval nějaký standardizovaný jazyk pro dotazování v XML dokumentech. Převážná většina relačních databází podporuje dotazovací jazyk SQL (Structured Query Language). Kdyby se pro dotazování v XML používal nějaký stejně rozšířený a standardizovaný jazyk, bylo by to jen dobře.

Mezi jednoduché dotazovací jazyky pro XML můžeme zařadit XPath a XSLT. Ty umožňují vybírat určité části dokumentu. Jsou však orientovány na XML dokumenty, které se používají pro data textového charakteru. Pro vyhledávání v silně strukturovaných datech nejsou příliš vhodné. Konsorcium W3C proto pracuje na dotazovacím jazyku pro XML, tzv. XML-QL (XML Query Language). Ten má kromě snadného výběru informací v XML dokumentu umožnit i jejich uspořádání do formy, ve které potřebujeme výsledek.

Existují i inicativy, které se snaží podporu pro XML přidat do jazyka SQL. Nové verze komerčních databázových serverů často již nějaký dotazovací jazyk pro XML mají v sobě zahrnutý. Zatím však nelze hovořit o nějakém standardu. Bude ještě potřeba nějaký čas, než se na základě zkušeností a požadavků vytvoří jednotný a všemi používaný standard.

7.5 Konvertory a formátovače

Velkou předností XML je možnost snadné transformace do dalších formátů. Z jednoho dokumentu můžeme vygenerovat PDF nebo PostScript pro tisk,

[24] `http://www.cs.helsinki.fi/~jjaakkol/sgrep.html`

stránku HTML pro Web, WML stránku pro mobilní telefon nebo RTF dokument pro odeslání jako přílohu mailu.

Jelikož je dokumenty potřeba převádět a formátovat rychle a efektivně, používají se pro tyto účely stylové jazyky, které nás nenutí vázat se na nějaký konkrétní produkt, protože jsou standardizovány. Pro potřeby konverze mezi různými formáty založenými na XML, mezi XML a HTML a mezi XML a textovými formáty se výborně hodí transformační část jazyka XSL. Pro výsledné formátování dokumentu se pak používá rovněž XSL, konkrétně jeho formátovací objekty. V některých případech může být vhodné i použití jiných jazyků, např. DSSSL.

XT

XT[25] je XSL procesor napsaný v Javě. Jeho velkou výhodou je poměrně úplná implementace transformační části XSL. XT bohužel nepodporuje formátovací objekty, takže se hodí pouze pro konverzi mezi různými XML schématy nebo mezi XML a HTML.

FOP

FOP[26] je součástí projektu Apache Cocoon a umí XML dokumenty konvertovat přímo do formátu PDF. FOP je napsán v Javě a kromě výstupu do PDF umí dokumenty i přímo zobrazovat. Jeho rozhraní umožňuje zařazení do dalších aplikací, které ho mohou využívat jako jednoduchý prohlížeč. FOP umí soubory PDF generovat ze souboru obsahujícího formátovací objekty nebo přímo z XML dokumentu a odpovídajícího XSL stylu. FOP je velice nadějný projekt a doufejme, že jeho autoři budou program neustále vylepšovat.

RenderX

Firma RenderX[27] na svých stránkách inzeruje program FO2PDF, který umí převádět dokument s formátovacími objekty do PDF. Ukázky jsou opravdu impozantní, FO2PDF zvládá i sazbu do více sloupců a další náročnější prvky.

Passive TeX

Passive TeX[28] je balík maker pro typografický systém TeX, který umožňuje sazbu formátovacích objektů. Podobně jako FOP je k dispozici zdarma.

---

[25] `http://www.jclark.com/xml/xt.html`

[26] `http://xml.apache.org/fop/`

[27] `http://www.renderx.com`

[28] `http://users.ox.ac.uk/~rahtz/passivetex/`

Jade

Jade[29] je DSSSL procesor. Kromě konverzí z XML do XML a HTML umí dokumenty převést do formátů RTF, TeX a MIF (FrameMaker). Z TeXu pak můžeme snadno vygenerovat PostScript nebo PDF. Výhoda Jade a DSSSL spočívá v tom, že jsou na světě déle než XSL, a tak dnes existují DSSSL styly pro nejrozšířenější DTD.

7.6 Editory stylů

Pokud chceme XML dokument přehledně zobrazit na obrazovce nebo pěkně vytisknout, potřebujeme styl, který určí, jak se bude dokument formátovat. Styl můžeme samozřejmě ručně zapisovat přímo v odpovídajícím stylovém jazyku. Pro složité dokumenty to však může být poměrně nevděčná a zdlouhavá práce. Existují editory, které nám umožní vytvoření stylu v pohodlném vizuálním prostředí bez nutnosti znát používaný stylový jazyk. Často jsou tyto editory součástí XML editorů a nejsou k dispozici jako samostatné programy.

7.7 Editory DTD a schémat

Vytváření DTD nebo schémat si lze rovněž usnadnit pomocí různých podpůrných programů. Složitější DTD nebo schémata jsou poměrně velké a složitě strukturované textové dokumenty. Vizuální editory dokáží schéma dokumentu znázornit v grafické podobě a takto s ním i pracovat. Můžeme se soustředit na strukturu dokumentů a nemusíme myslet na syntaktické detaily použitého jazyka pro zápis schémat.

Příkladem editoru schémat je XML Authority,[30] který umí vizuálně editovat schémata. XML Authority rozumí několika nejpoužívanějším schémovým jazykům a DTD, takže je mezi sebou můžeme i převádět.

7.8 Konverze do XML

Pokud vytváříme nové dokumenty, není problém je pomocí vhodného editoru vytvářet rovnou v XML. Často však potřebujeme do nového systému založeného na XML importovat staré dokumenty, vytvořené v nějakém jiném formátu. Do hry pak vstupují programy, které se snaží usnadnit konverzi dokumentů z proprietárních formátů do XML.

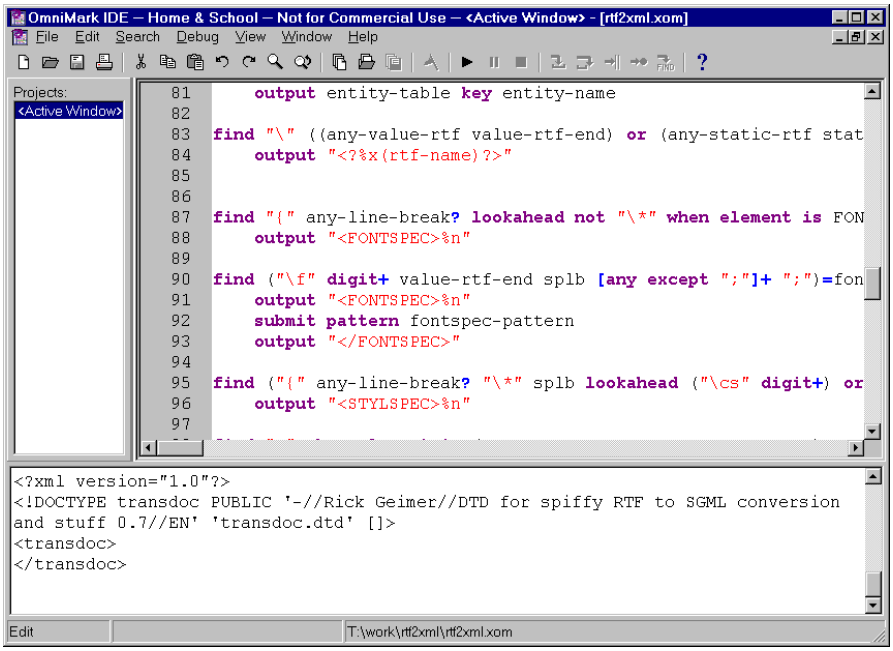
---

[29] `http://www.jclark.com/jade/`

[30] `http://www.extensibility.com/products/xa/`

OmniMark

OmniMark[31] je programovací jazyk, který je speciálně uzpůsoben pro konverzi mezi XML a ostatními formáty. Kromě klasických programových konstrukcí, které známe z běžných programovacích jazyků, nabízí efektivní syntaxi, umožňující na základě vzorů vybírat jednotlivé části souboru a dále je zpracovávat. OmniMark se již dlouhou dobu používá v oblasti konverze textů a existuje proto pro něj mnoho programů pro konverzi mezi různými formáty (např. z RTF do XML).



Obr. 7-8: Integrované vývojové prostředí OmniMarku

DynaTag

DynaTag[32] je komerční produkt, který umožňuje konvertovat dokumenty z formátů textových editorů do XML. Konverze probíhá v grafickém prostředí, a poté, co jsou na několika prvních dokumentech daného druhu definována konverzní pravidla, může převod probíhat automaticky.

---

[31] http://www.omnimark.com/develop/om5/index.html

[32] http://www.enigma.com/products/dynatag.htm

7.9 Parsery

Práci s parserem jsme si už vyzkoušeli. Řekli jsme si, že existují dva druhy parserů. Kromě těch, jejichž služeb můžeme využívat přímo jejich spuštěním, existují parsery i v podobě knihoven, které můžeme použít v našich programech. Má to pro nás obrovskou výhodu, protože nemusíme kontrolovat syntaxi dokumentu. Tu automaticky zkontroluje parser, navíc může ověřit i validitu oproti danému DTD nebo schématu. Naše aplikace pak nemusí obsahovat zdaleka tolik kódu pro ošetření chyb ve zpracovávaných datech.

Další výhodou parseru je to, že nám obsah dokumentu zpřístupní v příjemné podobě. Aby byl život vývojářů co nejjednodušší, existují standardizovaná rozhraní (API) pro práci s dokumentem. Většina producentů parserů tato rozhraní používá. Dnes se používají dvě rozhraní DOM a SAX. Poznamenejme ještě, že parsery dnes existují pro většinu běžně používaných jazyků Java, C, C++, Perl a další. Ve Windows jsou parsery často dostupné jako COM objekty, takže je lze využít v libovolném jazyce, který podporuje rozhraní COM (těch je dnes převážná většina).

SAX

Rozhraní SAX (Simple API for XML)[33] je založeno na řízení pomocí událostí (event-driven). Pomocí rozhraní vytvoříme vazbu mezi událostmi, které generuje parser, a naším kódem. V praxi to znamená, že si definujeme funkce, které se zavolají v okamžiku, kdy parser narazí na začátek elementu, na obsah elementu, na konec elementu, na komentář, na instrukce pro zpracování apod. Naší funkci jsou pak předány všechny potřebné parametry jako např. název elementu.

Výhoda událostmi řízeného přístupu je v jeho rychlosti a malé spotřebě paměti. Jednotlivé události jsou vyvolávány postupně, jak je čten dokument. Jak za chvíli uvidíme, rozhraní DOM vyžaduje načtení celého dokumentu předtím, než s ním začneme pracovat. Pokud tedy nepotřebujeme funkčnost DOMu, vyplatí se použít SAX, protože naše aplikace bude rychlejší a bude mít menší paměťové nároky.

Rozhraní SAX dnes podporuje velké množství parserů, i když samotné rozhraní není definováno pomocí žádného standardu konsorcia W3C nebo jiné standardizační organizace. Rozhranní vzniklo společným úsilím vývojářů z diskusní skupiny xml-dev a představuje de facto standard. Parsery se rozhraní SAX drží, takže můžeme v naší aplikaci klidně zaměnit jeden parser za druhý.

[33] `http://www.microstar.com/sax.html`

DOM

Rozhraní DOM (Document Object Model)[34] je postaveno na zcela odlišném principu než SAX. Dokument je reprezentován jako stromová hierarchická struktura, kdy každému elementu odpovídá jeden uzel stromu. Odpovídající uzly mají samozřejmě i komentáře, instrukce pro zpracování atd. Tomuto způsobu reprezentace se říká grove (Graph Representation Of property ValuEs).[35] Rozhraní DOM obsahuje funkce, které nám umožňují celý strom dokumentu procházet, modifikovat jeho jednotlivé uzly, mazat je a přidávat. Narozdíl od SAXu nemusíme dokument procházet od začátku do konce, ale můžeme se v něm pohybovat dle naší potřeby. Proto se rozhraní DOM uplatní v aplikacích, které provádějí náročnější operace s dokumentem editory, prohlížeče a formátovače.

Rozhraní DOM je standardem z dílny konsorcia W3C. Původně byl DOM vytvořen zejména proto, aby nové verze prohlížečů podporující XML používaly stejný objektový model pro přístup k dokumentu ze skriptových jazyků jako JavaScript. Bez tohoto standardu bychom si o kompatibilitě prohlížečů mohli nechat jen zdát. Rozhraní DOM obsahuje i Internet Explorer a Mozilla.

[34] `http://www.w3.org/TR/REC-DOM-Level-1/`

[35] V angličtině je zkratka grove skutečně poetická. Slovo grove totiž znamená lesík nebo hájek, a odpovídá tak představě, že dokument může být reprezentován i několika stromy jeden pro DTD, jeden pro samotnou instanci dokumentu, další pak pro hodnoty atributů.

8. XML na Webu

Nazvat kapitolu XML na Webu může dnes působit poněkud schizofrenně. XML vzniklo především jako prostředek pro výměnu informací v prostředí Internetu, který už zasahuje téměř do všech oblastí informačních a komunikačních technologií. Web je přitom mnoha lidmi dnes chápán jako synonymum k Internetu. Z jistého úhlu pohledu je proto možné považovat celou tuto knihu za pojednání o použití XML na Webu.

My se tomuto zjednodušení vyhneme a v této kapitole se podíváme na to, jak XML efektivně využít při tvorbě webových stránek. Nejprve si něco povíme o XHTML, což je nová verze jazyka HTML. Dále si vysvětlíme způsoby, jak zařídit, aby si XML dokumenty mohla prohlédnout co nejširší obec uživatelů. Nakonec se seznámíme s nejnovějšími přístupy ke tvorbě stránek, kdy potřebujeme tytéž informace prezentovat ve formátech vhodných pro různá koncová zařízení – pro osobní počítače, mobilní telefony, osobní organizéry apod.

8.1 XHTML 1.0

Možnost vytvářet v jazyce XML dokumenty, které budou používat vlastní sadu tagů vyhovujících našim požadavkům, je velice lákavá a užitečná. Na druhou stranu, pokud budou dokumenty zveřejňovány především na Webu, není nutné vše dělat znovu od začátku. V mnoha případech bude mnohem jednodušší a rychlejší použít zažité a časem prověřené značky, které známe z HTML, a pouze je doplnit o pár nových značek, které našim dokumentům přidají inteligenci.

Konsorcium W3C proto vytvořilo novou verzi jazyka HTML, která je založena na XML a lze ji tedy snadno rozšiřovat. Nový jazyk se jmenuje *XHTML (eXtensible HyperText Markup Language)*.

Stránky v XHTML musí splňovat mnohem přísnější syntaktická pravidla než dnešní HTML stránky. Současné prohlížeče se dokáží vypořádat s mnoha chybami ve stránkách, jako jsou překřížené elementy, špatně spárované tagy apod. XHTML však vyžaduje, aby všechny stránky byly platné XML dokumenty, a musí proto dodržovat správnou strukturu.

Stránky v XHTML musí odpovídat jednomu ze tří DTD, stejně jako v HTML 4.0. První DTD (Strict) je určeno pro stránky, které neobsahují žádné prezentační elementy a atributy definující vzhled dokumentu (jako `font`, `align` apod.). Vzhled stránky je pak upraven pouze pomocí stylů. Druhé DTD (Transitional) je určeno pro všechny stránky – zahrnuje v sobě všechny elementy a atributy z HTML 4.0, včetně těch prezentačních. Poslední DTD (Frameset) se použije na stránkách, které obsahují definici rozdělení okna prohlížeče na několik rámů.

Každý dokument v XHTML proto musí začínat jednou ze tří následujících deklarací.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
               "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
               "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
               "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

Systémový identifikátor ukazující na DTD přitom nemusí nutně ukazovat na veřejnou kopii DTD, vystavenou na serveru konsorcia W3C. Můžeme klidně uvést nějakou lokální cestu k našemu souboru, protože jej budou chtít číst programy, které budou se stránkami pracovat jako s XML dokumenty.

Aby dokument vyhovoval specifikaci XHTML, musíme jako kořenový element použít `html` a přiřadit mu jmenný prostor `http://www.w3.org/1999/xhtml`. Na následující ukázce si můžete prohlédnout zdrojový kód jednoduché stránky v XHTML.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Nadpis</title>
  </head>
  <body>
    <p>Odstavec</p>
  </body>
</html>
```

Vidíme, že na začátku dokumentu je uvedena i XML deklarace. Její použití není povinné, pokud je dokument uložen v kódování UTF-8 nebo UTF-16. Pro ostatní kódování musíme deklaraci použít.

Kombinování více typů dokumentů v jednom

XHTML je založeno na XML, a proto lze velice snadno na stránky zařazovat i data v jiných formátech. Využívá se přitom standardní mechanismus jmenných prostorů. Kdybychom chtěli na stránku např. zařadit matematický vzoreček v MathML, není nic snazšího.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
```

```
    <title>Ukázka matematiky</title>
  </head>
  <body>
    <p>Zajímavá rovnice:</p>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <expr>
        <limit>
          <lowlimit> n <tendsto/> &inf; </lowlimit>
          <expr>
            <expr>
              1 <plus/> <expr> 1 <over/> n </expr>
            </expr>
            <power/>
            n
          </expr>
        </limit>
        <eq/>
        e
      </expr>
    </math>
  </body>
</html>
```

Konverze stávajících HTML stránek do XHTML

Díky své flexibilitě a rozšiřitelnosti se během několika let bude XHTML používat na většině stránek místo HTML. Podívejme se proto podrobněji na to, co pro nás v praxi znamená převedení stránek z HTML do XHTML.

Jak jsme si již řekli, XHTML stránky musí začínat platnou deklarací typu dokumentu a definováním jmenného prostoru u elementu `html`. Jelikož stránky v XHTML jsou XML dokumenty, musí splňovat stejné požadavky jako každý XML dokument. Základním požadavkem je dodržení správné strukturovanosti (well-formedness). To znamená, že všechny tagy musí být párové nebo musí být zapsány speciálním způsobem (viz níže). Zároveň nesmí stránka obsahovat žádné překřížené tagy, jak je na dnešních stránkách zvykem.

```
<!-- špatný zápis -->
<p>Odstavec se <em>zvýrazněním.</p></em>

<!-- správný zápis -->
<p>Odstavec se <em>zvýrazněním</em>.</p>
```

V XML záleží v názvech elementů a atributů na velikosti písmen. Všechny elementy a atributy proto musí být zapisovány malými písmeny. V XHTML slouží

tag `<li>` jako začátek položky seznamu, oproti tomu tag `<LI>` nemá přiřazen žádný význam.

Dalším omezením, které má v XHTML existuje, je nutnost uvádění ukončovacích tagů u všech elementů, které nejsou prázdné. Většina z nás je například zvyklá, že stačí zahájit odstavec pomocí tagu `<p>`. V XHTML však musíme odstavec ukončit odpovídajícím tagem `</p>`.

Na rozdíl od HTML musíme nyní vždy obsah atributů uzavírat do uvozovek nebo apostrofů. Kromě toho nemůžeme používat minimalizaci atributů. To znamená, že například u přepínacích tlačítek formulářů musíme místo:

```
<input type="radio" name="platba" value="hotovost" checked>
```

použít zápis:

```
<input type="radio" name="platba" value="hotovost" checked="checked">
```

Vidíme, že vše je podřízeno co nejsnazšímu automatickému zpracování dokumentů.

Aby XML procesor ihned poznal, zda tag je párový či nepárový, musíme všechny nepárové elementy jako `br` a `hr` ukončovat pomocí `/>` `<br/>`, `<hr/>` apod.

XHTML dokumenty budou zpracovávány běžnými parsery. Ve skriptech a stylech obsažených ve stránce nemůžeme proto přímo používat znaky '`<`' a '`&`', protože mají speciální význam. Můžeme je sice přepsat pomocí odpovídajících znakových entit, ale to je dost nepohodlné. Lepší je proto styly a skripty uložit do externích souborů nebo použít sekci CDATA.

Pro tvorbu návěstí se v HTML používal atribut `name` u elementu `a`. V XHTML bychom měli používat atribut `id`, který lze použít u libovolného elementu.

Při vytváření nových stránek je poměrně jednoduché držet se výše popsaných změn. Mnohdy však budeme stát před úkolem převést již existující stránky z HTML do XHTML. Ruční provádění těchto změn by bylo jistě zdlouhavé a pracné. Konsorcium W3C proto zdarma nabízí program HTML Tidy,[1] který umí odstranit ze stávajících HTML stránek mnoho chyb a převést je do XML. Program se hodí i pro mnoho dalších účelů. Například umí krásně začistit chybný HTML kód, vygenerovaný různými textovými editory.

Kompatibilita XHTML se stávajícími prohlížeči

Stránky vytvářené pomocí XHTML přinášejí mnoho nových možností jak autorům, tak uživatelům stránek. Nemůžeme však čekat, že jako mávnutím kouzelného proutku budou všechny používané prohlížeče podporovat XHMTL. V následujících odstavcích naleznete rady, jak psát stránky v XHTML tak, aby

[1] `http://www.w3.org/Status.html#TIDY`

byly čitelné i pro dnes běžně používané prohlížeče, které podporují pouze jazyk HTML.

V XML jsou nepárové tagy s prázdným obsahem ukončeny pomocí '/>'. S touto dvojicí znaků však většina prohlížečů nepočítá. Před ukončovací znak bychom proto měli psát mezeru starší prohlížeče pak lomítko ignorují jako neznámý atribut. Příklad:

```
<br />
<hr />
<img src="logo.gif" width="320" height="160" alt="Logo" />
```

Pokud potřebujeme použít element, který je obvykle párový, pouze jako nepárový, neměli bychom používat minimalizovanou formu. Příklad:

```
<p></p>              <!-- správné oddělení textu -->
<p />                <!-- špatné oddělení textu -->
```

V XML mají znaky '<', '&' a ']]>' speciální význam. Pokud skripty nebo kaskádové styly, vložené ve stránce, obsahují tyto znaky, je lepší skripty a styly uložit do samostatného souboru.[2]

V XML jsou v obsahu atributů všechny zbytečné mezery a konce řádků nahrazovány mezerou jedinou. Toto chování je odlišné od praxe dnešních prohlížečů. V obsahu atributů bychom se proto měli vyhnout opakování více mezer za sebou a koncům řádek.

Pokud u nějakého elementu určujeme jazyk jeho obsahu, měli bychom kromě atributu `xml:lang` použít i `lang`, který je podporován prohlížeči a je součástí HTML 4.0.

Pokud v XML odkazujeme na části dokumentu pomocí fragmentu (za URL je přidáno '`#návěstí`'), vztahuje se fragment k elementu označenému pomocí atributu `id`. Většina prohlížečů však atribut `id` nepodporuje a musíme použít i starší atribut `name` u elementu `a`.

```
<h2><a name="nadpisx" id="nadpisx">Ukázkově označený nadpis</a></h2>
```

Pokud máme v dokumentu určeno použité kódování pomocí XML deklarace:

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

měli bychom kódování určit i v HTML hlavičce, odkud jej čtou dnešní prohlížeče.

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
```

Při tomto určení způsobu kódování bychom se měli ujistit, že na serveru neběží nějaký modul pro automatickou změnu kódu, který při překódování dokumentu nezmění příslušné názvy kódování v dokumentu. Prohlížeči by pak jiné kódování dorazilo v hlavičkách HTTP odpovědi, a jiné v samotném XHTML kódu.

[2] HTML prohlížeče neznají sekce CDATA, a proto je nemůžeme použít.

Poslední věc, kterou si musíme uvědomit, je neschopnost některých prohlížečů korektně ignorovat instrukce pro zpracování. Tyto instrukce se v XML používají například pro připojení stylu definujícího vzhled dokumentu. Lepší je se jim tedy vyhnout.

8.2 Web v topinkovači

Po roce 2000 bude čím dál tím více lidí přistupovat k Webu z méně výkonných zařízení, než jsou dnešní počítače PC. Přístup k Internetu se stane běžnou záležitostí a budou se pro něj využívat různé WebTV, mobilní telefony, osobní komunikátory, telefony s jednoduchým terminálem a další poměrně primitivní zařízení. Tato zařízení nemohou obsahovat složitý kód dnešních prohlížečů, který z HTML stránek odstraňuje syntaktické chyby autorů. Tím, že XHTML dokumenty vyhovují XML, je lze velice snadno číst pomocí poměrně malých a jednoduchých XML parserů. XHTML ve verzi 1.0 udělalo nezbytný krok k tomu, aby byl Web přístupný opravdu komukoliv s libovolným koncovým zařízením.

V dalších verzích XHTML se počítá s tím, že celý dnešní jazyk HTML bude rozdělen do několika modulů. Každý modul bude obsahovat pouze určitou skupinu elementů např. základní elementy, elementy pro tvorbu seznamů, tabulek či formulářů.

Jelikož má každé výstupní zařízení (PC, WebTV, mobilní komunikátor) jiné schopnosti, bude zapotřebí vytvořit jednotný způsob, jak definovat schopnosti určité třídy výstupních zařízení. Pro skupinu zařízení se stejnými schopnostmi proto bude existovat jeden tzv. profil, který bude nést informace o možnostech zařízení. Profil bude obsahovat výčet podporovaných modulů XHTML a jiných DTD, grafických formátů apod. Autoři stránek pak budou jasně vědět, na kterých zařízeních bude možné jejich stránky prohlížet. Není ani vyloučeno, že webové servery budoucnosti budou umět na požádání klienta automaticky převádět dokumenty mezi různými profily (pokud to samozřejmě bude technicky možné) nebo nabídnout dokument v jedné z několika různých verzí.

W3C už pracuje na formátu *CC/PP (Composite Capability/Preference Profiles)* pro výměnu informací o profilech. Nikoho asi nepřekvapí, že tento formát opět staví na XML, konkrétně na jazyce RDF pro přidávání popisných dat k informačním zdrojům.

Vývoj nových technologií je v plenkách, ale zdá se, že rozdělení HTML na více modulů je jistě krok správným směrem. V současné době je totiž jazyk HTML 4.0 příliš komplexní a většina stránek stejně využívá pouze část jeho možností.

V době psaní knihy již vznikala nová verze XHTML 1.1. Ta už byla rozdělena do několika modulů a autoři stránek si mohli vybrat ty, které potřebují. Zároveň konsorcium W3C definovalo podmnožinu XHTML 1.1 pod názvem XHTML Basic. Tato verze XHTML je velice jednoduchá a obsahuje pouze ty nejdůležitější

elementy. Bez větších problémů se proto hodí například pro zobrazování stránek na mobilních zařízeních.

XHTML Basic je tak jednoduché, že by jej měli dokázat zobrazit všichni klienti. Bude to společný prezentační jazyk, jakým je dnes HTML. Přídavnou funkčnost (další elementy a atributy) bude možné do XHTML Basic přidat standardním způsobem. Nemělo by tedy dojít k situaci, kterou známe z poloviny 90. let, kdy si téměř každý výrobce prohlížečů přidával do HTML vlastní nestandardní rozšíření.

8.3 A co když nám XHTML nestačí?

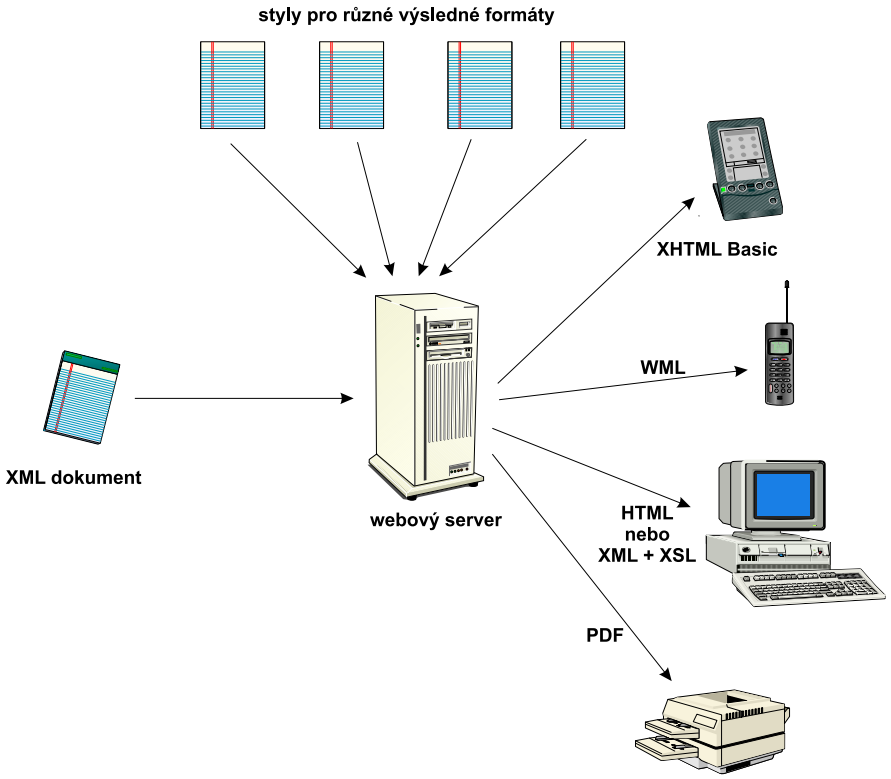
XHTML přece jen posouvá Web zase o kousek dál. Na druhou stranu XHTML dnes není přímo podporováno prohlížeči a mnohé problémy nevyřeší ani ve svých dalších verzích. Web během deseti let své existence prošel dvěma výraznými stádii a dnes vstupuje do třetího. Zpočátku se Web používal především pro prezentaci informací v podobě statických stránek. Později se z něj stala aplikační platforma, kdy se pomocí dynamicky generovaných stránek zpřístupnily služby jako elektronické obchodování, přístup do podnikových informačních systémů apod. Během této doby se jako klienti používaly klasické osobní počítače s klasickými prohlížeči, které více či méně podporovaly jeden jazyk HTML.

Dnes chceme mít pomocí Webu stále přístupné dokumenty a služby, ale spektrum koncových zařízení a formátů, které požadujeme, se rozšiřuje. Již dnes si můžeme koupit mobilní telefon s podporou protokolu WAP (Wireless Application Protocol), což je obdoba Webu pro kapesní zařízení. Postupně jednotlivé servery začínají nabízet svůj obsah i v jazyce WML, který se používá ve WAPu místo HTML. Schopnosti HTML a WML jsou tak rozdílné, že nejde jedny a tytéž stránky vytvářet v nějakém formátu vhodném pro obě dvě platformy. Rozsáhlejší informace jako jsou ceníky, dokumentace, delší články apod. je vhodné kromě on-line verze v HTML nabízet i ve formátu PDF, abychom si je mohli v pěkném formátu vytisknout a v klidu prostudovat večer nebo o víkendu v posteli.

Pokud dnes někdo nabízí své informace ve více formátech, obvykle je musí poměrně pracně konvertovat mezi jednotlivými formáty, které podporuje. V případě potřeby dalšího formátu pro nějaké nové zařízení nebo službu se musí všechny dokumenty konvertovat do dalších formátů.

Řešením výše popsaného problému je ukládání všech dokumentů ve formátu XML. Musíme si přitom zvolit schéma, které dokáže co nejlépe zachytit sémantiku ukládaných informací. Pomocí stylů pak můžeme tento informačně bohatý dokument konvertovat do dalších formátů do HTML, WML, PDF. Pokud v budoucnu vznikne potřeba podpory dalšího formátu, stačí vytvořit nový styl pro konverzi.

Aby vše fungovalo zcela hladce, bude potřeba použít speciálně upravené webové servery, které automaticky rozpoznají klienta a požadovaný dokument



Obr. 8-1: *Již dnes vzniká potřeba nabízet jeden dokument v několika formátech*

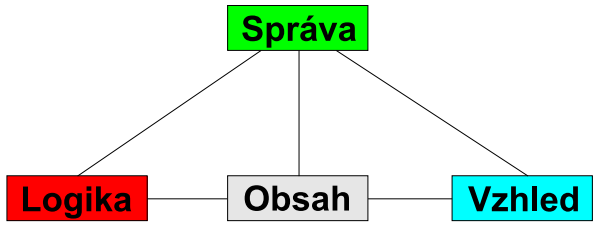
pro něj automaticky převedou do vhodného formátu. Takováto řešení existují již dnes. Jako příklad si můžeme uvést XSL ISAPI Extension 1.1[3] od Microsoftu a projekt Apache Cocoon.[4] Oba dva produkty používají jako stylový jazyk XSL. V konfiguračních souborech lze určit, pro které klienty se použijí jednotlivé styly.

Výše popsaným způsobem můžeme poměrně snadno vyřešit problém, jak nabídnout statické dokumenty v několika formátech. Nemůžeme však vytvořit interaktivní aplikaci, která by byla přístupná pomocí několika uživatelských rozhraní. Dokázali jsme oddělit vzhled od obsahu, ale v praxi od sebe mnohdy potřebujeme oddělit vzhled, obsah a aplikační logiku. O to se snaží i projekt Cocoon. Na něm si ukážeme, jaká architektura se možná za pár let prosadí na většině webových serverů.

Tím, že se oddělí jednotlivé části aplikace, mohou na nich pracovat různí lidé, specialisté ve svém oboru. O logiku aplikace se starají programátoři, o obsah editoři a o vzhled grafici. Oproti jiným řešením je zajímavé, že aplikační logika

[3] `http://msdn.microsoft.com/downloads/webtechnology/xml/xslisapi.asp`

[4] `http://xml.apache.org/cocoon/`



Obr. 8-2: Cocoon schéma architektury

vůbec nezávisí na definici vzhledu. Pro toho, kdo někdy psal nějakou webovou aplikaci jako CGI skripty nebo v ASP či v PHP, to musí znít jako rajská píseň.

Samotné dokumenty v Cocoonu mohou být buď statické stránky, stránky obsahující speciální tagy nebo dynamicky generované dokumenty. Speciální tagy umožňují bezpečným a jednoduchým způsobem zařadit do dokumentu volání programového kódu.

Po obdržení požadavku Cocoon předá odpovídajícímu generátoru obsahu parametry. Generování výsledného XML dokumentu může probíhat mnoha různými způsoby. Získaný XML dokument se pak pomocí XSL stylu transformuje do požadovaného výstupního formátu. Cocoon navíc obsahuje inteligentní vyrovnávací paměť, která v mnoha případech dokáže efektivně snížit zátěž serveru.

9. Pár slov závěrem

To, že jste knihu dočetli až sem, mi dává jistou naději, že jste ji neodložili do sběru a že vás problematika XML zaujala. Snad je vám mnoho věcí, které souvisejí s XML, mnohem jasnějších, než když jste knihu knihu začali číst. Je však celkem pravděpodobné, že vám v hlavě leží spousta dalších problémů, na které nám nezbyl čas. Zejména při popisu stylových jazyků byl výklad velmi stručný a pouze se snažil celou oblast přiblížit.

Nemusíte však propadat panice. Význam XML jsem patřičně zdůraznil šéfredaktorovi počítačové redakce Grady Rudovi Pecinovskému, a ten zase přesvědčil ještě vyšší šéfy. Několik týdnu po tom, co vyjde tato kniha, bude na pultech knihkupectví k dostání překlad knihy *The XML companion* [8] od Neila Bradleyho. Podle mne je to vůbec nejlepší kniha o XML, pokud už víte, o co jde, a chcete se dozvědět o všech technologiích XML co největší podrobnosti. Součástí knihy je velice podrobný výklad XLinku, XSL a XSLT.[1] Na své si přijdou i programátoři, protože v knize je popsána praktická práce s rozhraními DOM a SAX. Kniha obsahuje mnoho dalších zajímavostí, ale přestanu s rolí reklamního agenta. Zkrátka si knihu přečtěte a uvidíte.

Ani sebelepší kniha vám nezodpoví všechny otázky. V těchto případech je vhodné dotaz poslat například do diskusní skupiny `cz.comp.lang.xml`, která se věnuje jazyku XML.

Práce s diskusními skupinami je velice snadná. Do prohlížeče stačí napsat URL adresu `news:cz.comp.lang.xml`. Pokud by tato adresa nefungovala, nemáte v konfiguraci vašeho prohlížeče správně nastavenu adresu serveru pro diskusní skupiny (news server). Tuto adresu by vám měl sdělit vás poskytovatel připojení. Pokud byste adresu vašeho news serveru nezjistili, můžete zkusit některý z volně dostupných serverů např. `news://news.cesnet.cz/cz.comp.lang.xml`. Pokud ani to nepomůže, je vaše sít k Internetu připojena přes firewall, který vás k news serveru nepustí. Pak máte prostě smůlu. ;(

Nejlepší způsob, jak se naučit XML, je však jen jeden. *Používat ho dnes a denně.* Přeji vám při tom mnoho úspěchů.

[1] Překládáno je druhé vydání knihy z konce roku 1999, které popisuje poslední návrhy standardů.

A. Instalace užitečných programů

V této příloze naleznete stručné návody na instalaci programů a nástrojů, které používáme v ukázkách.

A.1 Parser SP

SP si můžete stáhnout ze adresy `http://www.jclark.com/sp/`. K dispozici jsou zdrojové texty i binární distribuce připravené ke spuštění. Pro spuštění SP ve Windows budete konkrétně potřebovat soubor `ftp://ftp.jclark.com/pub/sp/win32/sp1_3_4.zip`. My si ukážeme postup instalace ve Windows, pro ostatní systémy je velice podobný.

Distribuční soubor SP rozbalíme například do adresáře `c:\sp`. Pokud si chceme usnadnit spouštění parseru, přidáme si do proměnné `PATH` cestu k adresáři `c:\sp\bin`.

Samotný parser je realizován programem `nsgmls`. Před jeho spuštěním musíme nastavit ještě několik proměnných prostředí, aby správně pracovala podpora XML. Proměnná `SP_CHARSET_FIXED` musí být nastavena na hodnotu `YES`. Proměnná `SP_ENCODING` by měla mít hodnotu `XML`. V tomto případě je kódování dokumentů odvozeno z XML deklarace. Výstup programu je však v kódování UTF-8, takže nemusí být vždy čitelný.

Proměnnou `SP_ENCODING` můžeme nastavit i na hodnotu `windows` nebo `iso-8859-2`. Parser pak předpokládá, že dokument je v kódování windows-1250 nebo ISO 8859-2.

Poslední proměnnou, kterou musíme nastavit, je `SGML_CATALOG_FILES`. Do proměnné uložíme cestu k souboru `xml.soc` např. `c:\sp\pubtext\xml.soc`. Tento soubor obsahuje deklarace, které umožní parseru SP původně určenému pro SGML korektně pracovat i s XML dokumenty. Kontrola dokumentu se pak spustí pomocí příkazu:

```
nsgmls -wxml -s «dokument»
```

A.2 Parser od Microsoftu

Parser od Microsoftu je součástí Internet Exploreru 5.0. Pokud tedy máme nainstalován Internet Explorer, máme k dispozici i parser. Ve standardní verzi IE 5.0 je k dispozici pouze parser, který kontroluje správnou strukturovanost dokumentu, ale ne jeho validitu. Nová verze parseru, která umí kontrolovat dokument podle DTD a schémat je k dispozici na adrese `http://msdn.microsoft.com/downloads/webtechnology/xml/msxml.asp`.

Parser Microsoftu je k dispozici ve formě COM objektu. Můžeme ho tedy využívat přímo v našich programech. K parseru však není standardně dodáván žádný nástroj, který by umožnil jeho ruční spuštění z příkazové řádky. Nic nám však nebrání v tom, abychom si takový program napsali.

Pro vytvoření instance komponenty a její spuštění je optimální prostředí Windows Scripting Host (WSH), které umožňuje tvorbu dávkových souborů. WSH je standradní součástí Windows 98 a Windows 2000. Do ostatních verzí Windows se dá instalovat dodatečně.[1] My si vytvoříme jednoduchou dávku `msxml.js` pro spouštění parseru.

```
var xml = WScript.CreateObject("MSXML2.DOMDocument");
xml.async = false;
if (WScript.Arguments.Count() == 0)
{
  WScript.Echo("Musíte zadat jméno souboru ke kontrole!!!");
  WScript.Quit(1);
}
xml.load(WScript.Arguments(0));
var err = xml.parseError;
if (err.errorCode != 0)
{
  WScript.Echo(err.srcText);
  var s = "";
  for (i=1; i<err.linepos; i++) s += "-";
  s += "^";
  WScript.Echo(s);
  WScript.Echo(err.url + ":" + err.line + ":" + err.linepos +
           ":" + err.reason);
}
```

Soubor bychom měli uložit někam, kam ukazuje proměnná `PATH`. Mají-li se chybová hlášení parseru vypisovat na příkazovou řádku, a ne do samostatného okna, je dobré zadat na příkazové řádce příkaz:

```
cscript //H:cscript
```

Parser pak spustíme pomocí příkazu:

`msxml` *«dokument»*

A.3 XSLT procesor XT

XT je XSLT procesor napsaný v Javě. Stáhnout si jej můžete z adresy `http://www.jclark.com/xml/xt.html`. Pro jeho spuštění budete potřebovat na vašem

[1] `http://msdn.microsoft.com/scripting/`

počítači Javu. XT potřebuje pro svou správnou činnost i XML parser. Stránky obsahují informace o tom, kde si pro XT vhodný parser sehnat. Existuje i speciální verze XT, která je určena pro Windows[2] a v jednom spustitelném souboru obsahuje vše potřebné, včetně parseru.

Parser, který se v XT normálně používá, bohužel nepodporuje kódování windows-1250 a iso-8859-2. Na mých stránkách je na adrese `http://www.kosek.cz/xml/xt-czech/xt-czech.html` k dispozici upravená verze XT, která obě v Česku používaná kódování podporuje. XT je zde k dispozici jako jeden javový archiv, který v sobě kromě XT rovnou obsahuje i parser XP. Pro úspěšné spuštění XT si tedy stačí stáhnout soubor `xt.jar`. Soubor uložíme do libovolného adresáře. XT pak spustíme příkazem:

`java -cp` *«cesta»*`xt.jar com.jclark.xsl.sax.Driver` *«dok.» «styl» «výstup»*

Pro časté spouštění se nám samozřejmě vyplatí vytvoření dávkového souboru. Ve Windows si můžeme do souboru `xt.bat` uložit následující příkaz pro volání XT:

```
@java -cp c:\xmllib\xt.jar com.jclark.xsl.sax.Driver %1 %2 %3 %4 %5 %6 %7 %8
```

Předpokládáme přitom, že jsme si archiv `xt.jar` uložili do adresáře `c:\xmllib`. Na Unixu může skript pro spouštění vypadat zhruba takto:

```
#!/bin/sh
java -cp /usr/share/java/xml/xt.jar com.jclark.xsl.sax.Driver $*
```

V tomto případě předpokládáme, že soubor `xt.jar` jsme uložili do adresáře `/usr/share/java/xml`. Nesmíme zapomenout nastavit skriptu práva pro spuštění například pomocí `chmod +x xt`.

Dávky pro spouštění XT je vhodné umístit do cesty, kde systém hledá programy ke spuštění.

A.4 XSLT procesor od Microsoftu

XSLT procesor je součástí stejné komponenty jako XML parser. Před samotným použitím XSLT si proto musíme sehnat příslušné komponenty postup je stejný jako v případě instalace parseru od Microsoftu.

Pro snadné aplikování stylu na dokument si opět vytvoříme dávku pro WSH. Pojmenujmeme ji třeba `msxsl.js`.

```
if (WScript.Arguments.Count() != 3)
{
  WScript.Echo("Musíte zadat tři parametry: dokument styl vystup");
  WScript.Quit();
}
```

[2] `ftp://ftp.jclark.com/pub/xml/xt-win32.zip`

```
var xml = WScript.CreateObject("msxml2.DOMDocument");
xml.async = false;
xml.validateOnParse = false;
xml.load(WScript.Arguments(0));

var xsl = WScript.CreateObject("msxml2.DOMDocument");
xsl.async = false;
xsl.validateOnParse = false;
xsl.load(WScript.Arguments(1));

var out = WScript.CreateObject("msxml2.DOMDocument");
out.async = false;
out.validateOnParse = false;
xml.transformNodeToObject(xsl, out);
out.save(WScript.Arguments(2));
```

B. Kódy jazyků a států

V této příloze naleznete ISO kódy nejpoužívanějších jazyků a nejznámějších států. Důvodem zařazení této přílohy je časté používaní chybných kódů. Například v mnoha HTML stránkách nalezneme jako kód českého jazyka `cz`. Není to však kód jazyka, ale země České republiky. Správný jazykový kód pro češtinu je `cs`.

B.1 Jazykové kódy podle ISO 639

U libovolného elementu můžeme určit jazyk jeho obsahu pomocí atributu `xml:lang`. Podobnou funkci má atribut `lang` v jazyce HTML. Jazyk se přitom zadává pomocí svého ISO kódu, jejichž přehled naleznete v tabulce B-1.

Kód	Jazyk	Kód	Jazyk
`sq`	albánština	`ar`	arabština
`hy`	arménština	`bg`	bulharština
`be`	běloruština	`ca`	katalánština
`zh`	čínština	`hr`	chorvatština
`cs`	čeština	`da`	dánština
`en`	angličtina	`eo`	esperanto
`et`	estonština	`fi`	finština
`fr`	francouzština	`de`	němčina
`el`	řečtina	`iw`	hebrejština (také `he`)
`hu`	maďarština	`it`	italština
`ja`	japonština	`ko`	korejština
`la`	latina	`no`	norština
`pl`	polština	`pt`	portugalština
`ro`	rumunština	`ru`	ruština
`sr`	srbština	`sh`	srbochorvatština
`sk`	slovenština	`sl`	slovinština
`es`	španělština	`sv`	švédština
`th`	thajština	`tr`	turečtina
`uk`	ukrajinština	`vi`	vietnamština

Tab. B-1: *Vybrané jazykové kódy podle ISO 639*

Kód	Stát	Kód	Stát
AF	Afghánistán	AL	Albánie
DZ	Alžírsko	AD	Andorra
AO	Angola	AR	Argentina
AM	Arménie	AU	Austrálie
AT	Rakousko	AZ	Ázerbajdžán
BY	Bělorusko	BE	Belgie
BR	Brazílie	BG	Bulharsko
CA	Kanada	CL	Chile
CN	Čína	CU	Kuba
CZ	Česko	DK	Dánsko
EG	Egypt	EE	Estonsko
FI	Finsko	FR	France
DE	Německo	GR	Řecko
HU	Maďarsko	IS	Island
IN	Indie	ID	Indonésie
IR	Írán	IQ	Irák
IE	Irsko	IL	Izrael
IT	Itálie	JM	Jamajka
JP	Japonsko	KZ	Kazachstán
KP	KLDR	KR	Jižní Korea
LI	Lichtenštejnsko	LU	Lucembursko
MX	Mexiko	MN	Mongolsko
NP	Nepál	NL	Nizozemí
NZ	Nový Zéland	NO	Norsko
PK	Pákistán	PH	Filipíny
PL	Polsko	PT	Portugalsko
RO	Rumunsko	RU	Rusko
SK	Slovensko	SI	Slovinsko
ES	Španělsko	SE	Švédsko
CH	Švýcarsko	UA	Ukrajina
AE	Arabské emiráty	GB	Velká Británie
US	Spojené státy	VA	Vatikán
VE	Venezuela	VN	Vietnam

Tab. B-2: *Vybrané kódy států podle ISO 3166*

B.2 Kódy států podle ISO 3166

Kódy států se používají například jako jméno domén nejvyšší úrovně. Rovněž se používají při přesnějším určování jazyka. Například `en-GB` označuje britskou angličtinu a `en-US` americkou.

Literatura

[1] Altheim, M. – McCarron, S.: *XHTML 1.1 Module-based XHTML W3C Working Draft.* W3C 2000.
http://www.w3.org/TR/xhtml11

[2] Altheim, M. – McCarron, S.: *Building XHTML Modules W3C Working Draft.* W3C 2000.
http://www.w3.org/TR/xhtml-building

[3] Altheim, M. – Boumphrey, F. – Dooley, S. – McCarron, S. – Wugowski, T.: *Modularization of XHTML W3C Working Draft.* W3C 2000.
http://www.w3.org/TR/xhtml-modularization

[4] Adler, S. – Berglund, A. – Caruso, J. – Deach, S. – Milowski, A. – Pernell, S. – Richman, J. – Zilles, S.: *Extensible Stylesheet Language (XSL) Version 1.0 W3C Working Draft.* W3C 2000.
http://www.w3.org/TR/xsl

[5] Bos, B. – Wium Lie, H. – Lilley, Ch. – Jacobs, I.: *Cascading Style Sheets, level 2 CSS2 Specification.* W3C 1998.
http://www.w3.org/TR/REC-CSS2/

[6] Biron, P. – Malhotre, A.: *XML Schema Part 2: Datatypes W3C Working Draft.* W3C 1999.
http://www.w3.org/TR/xmlschema-2/

[7] Bradley, N.: *The concise SGML companion.* Addison Wesley Longman Limited, Harlow 1997. ISBN 0-201-41999-8.

[8] Bradley, N.: *The XML companion.* Addison Wesley Longman Limited, Harlow 1998. ISBN 0-201-34285-5.

[9] Bray, T.: *Annotated XML Specification.* 1998.
http://www.xml.com/axml/testaxml.htm

[10] Clark, J.: *Associating Style Sheets with XML documents Version 1.0.* W3C 1999.
http://www.w3.org/TR/xml-stylesheet

[11] Clark, J. – DeRose, S.: *XML Path Language (XPath) Version 1.0.* W3C 1999.
http://www.w3.org/TR/xpath

[12] Clark, J.: *XSL Transformations (XSLT) Version 1.0.* W3C 1999.
http://www.w3.org/TR/xslt

[13] *Practical Transformation Using XSLT and XPath.* Crane Softwrights, 1999. ISBN 1-894049-02-0.
http://www.cranesoftwrights.com

[14] DeRose, S.: *The SGML FAQ Book.* Kluwer Academic Publishers, Boston 1997. ISBN 0-7923-9943-9.

[15] DeRose, S. – Maler, E. – Orchard, D. – Trafford, B.: *XML Linking Language (XLink) W3C Working Draft.* W3C 1999.
`http://www.w3.org/TR/xlink`

[16] DeRose, S. – Daniel, R. – Maler, E.: *XML Pointer Language (XPointer) W3C Working Draft.* W3C 1999.
`http://www.w3.org/TR/xptr`

[17] Goldfarb, Ch. – Prescod, P.: *The XML handbook.* Prentice Hall PTR, Upper Saddle River 1998. ISBN 0-13-081152-1.

[18] Homer, A.: *XML IE5 Programmer's Reference.* Wrox Press, Birmingham 1999. ISBN 1-861001-57-6.

[19] Ion, P. – Miner, R.: *Mathematical Markup Language (MathML) 1.0 Specification.* W3C 1998.
`http://www.w3.org/TR/REC-MathML`

[20] Kosek, J.: *Přehled vlastností stylů.*
`http://www.kosek.cz/clanky/dhtml/css-properties.html`

[21] Kosek, J.: *HTML tvorba dokonalých WWW stránek.* Grada Publishing, Praha 1998. ISBN 80-7169-608-0.
`http://www.kosek.cz/html/`

[22] Megginson, D.: *Structuring XML Documents.* Prentice Hall PTR, Upper Saddle River 1998. ISBN 0-13-642299-3.

[23] *XML Syntax Quick Reference.* Mulberry Technologies 1999.
`http://www.mulberrytech.com/quickref/XMLquickref.pdf`

[24] *XSLT and XPath Quick Reference.* Mulberry Technologies 2000.
`http://www.mulberrytech.com/quickref/XSLTquickref.pdf`

[25] Nič, M.: *Učební materiály Zvonu.*
`http://zvon.vscht.cz/ZvonHTML/Zvon/zvonTutorials_cs.html`

[26] Thompson, S. – Beech, D. – Maloney, M. – Mendelsohn, N.: *XML Schema Part 1: Structures W3C Working Draft.* W3C 1999.
`http://www.w3.org/TR/xmlschema-1/`

[27] Třísková, L.: *GNU nástroje pro tvorbu WWW stránek.* Grada Publishing, Praha 2000. ISBN 80-7169-861-X.

[28] Walsh, N. – Muellner, L.: *DocBook: The Definitive Guide.* O'Reilly, Cambridge 1999. ISBN 1-56592-580-7.
`http://www.docbook.org/`

Rejstřík

Jiří Kosek
XML pro každého
podrobný průvodce

Vydala Grada Publishing, spol. s r. o.
U Průhonu 22, Praha 7
jako svou 1063. publikaci

Odpovědný redaktor Petr Somogyi
Návrh obálky Libor Samec
Realizace obálky Matouš Přikryl
Počet stran 164
První vydání, Praha 2000
Vytiskly Tiskárny Havlíčkův Brod, a.s.
Husova ulice 1881, Havlíčkův Brod