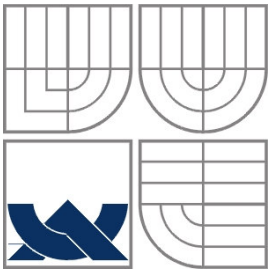
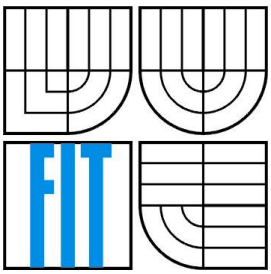


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INTERAKTIVNÍ VIZUALIZACE XML

INTERACTIVE VISUALIZATION OF XML

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — PAVEL ALBRECHT
AUTHOR

VEDOUCÍ PRÁCE — ING. PETR CHMELAŘ
SUPERVISOR

BRNO 2008

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Albrecht Pavel**
Obor: Informační technologie
Téma: **Interaktivní vizualizace XML**
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s problematikou XML a jeho definicemi.
2. Identifikujte a pokuste se vyřešit problémy interaktivního zobrazení XML dat včetně jejich definice.
3. Vytvořte grafický nástroj pro zobrazení a modelování libovolných datově orientovaných dokumentů XML.
4. Zhodnoťte vlastnosti a případné vylepšení nástroje.

Literatura:

- Quin, L. W3C. *Extensible Markup Language (XML)*. 2006. Dostupný z: http://www.w3.org/XML/.
- Oracle Corporation. *Oracle Technology Network*. 2006. Dostupný z: http://www.oracle.com/technology/.
- Herout, P. *Učebnice jazyka Java.* České Budějovice: Kopp, 2007. 381 s. ISBN 978-80-7232-323-4.
- Jelinek, J. - Slavik, P. XML visualization using tree rewriting. *Proceedings of the 20th spring conference on Computer graphics*. ACM, 2004. pp. 65-72. ISBN:1-58113-967-5.

Při obhajobě semestrální části projektu je požadováno:

- 1. a 2. bod zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Chmelař Petr, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan

Jméno a příjmení:	**Pavel Albrecht**
Id studenta:	79123
Bytem:	Pstruží 237, 739 11 Frýdlant nad Ostravicí
Narozen:	26. 12. 1985, Čeladná
(dále jen "autor")	

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií

se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....

(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   bakalářská práce

Název VŠKP:	Interaktivní vizualizace XML
Vedoucí/školitel VŠKP:	Chmelař Petr, Ing.
Ústav:	Ústav informačních systémů
Datum obhajoby VŠKP:	.....

VŠKP odevzdal autor nabyvateli v:

tištěné formě	počet exemplářů: 1
elektronické podobě	počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - □ ihned po uzavření této smlouvy
   - □ 1 rok po uzavření této smlouvy
   - □ 3 roky po uzavření této smlouvy
   - □ 5 let po uzavření této smlouvy
   - □ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Tato bakalářské práce se zabývá problematikou vizualizace formátu XML a jeho definicemi. Stručně shrnuje vývoj XML, jeho charakteristiku a syntaxi. Popisuje i různá řešení, jimiž jsou v současné době tyto dokumenty vizualizovány a vhodné implementační a vývojové nástroje, které je možno pro tento účel využít. Dále práce identifikuje problémy vizualizace a navrhuje možné řešení, jakými bude směřována.

Hlavním cílem je vytvořit jednoduchou a přehlednou aplikaci, která bude interaktivně vizualizovat zmiňované XML dokumenty.

Klíčová slova

XML, interaktivní vizualizace XML, Tabulka, XML Schema, JDOM, Trang, Java.

Abstract

The bachelor's thesis deals with problems of XML document visualization and its definitions. It briefly summarizes development of XML, its characteristic and syntax. The thesis reviews various solutions for XML visualization and proper development tools for this purpose. The work also describes future goals of visualization and a design of possible solution.

The main goal was to create a simple and well-arranged application, witch will visualize mentioned XML documents interactively.

Keywords

XML, interactive visualization of XML, Table, XML Schema, JDOM, Trang, Java.

Citace

Albrecht Pavel: Interaktivní vizualizace XML. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Interaktivní vizualizace XML

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Petra Chmelaře. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Tímto bych chtěl poděkovat Ing. Petru Chmelařovi za odbornou pomoc a cenné rady při psaní této bakalářské práce.

Obsah

1 Úvod

Informace a jejich výměna hrají v informatice významnou roli. Pro tento účel bylo třeba vytvořit formát, který by umožňoval jednoduché a rychlé zpracování. Počátky tohoto vývoje sahají do 80. let, kdy přišel na svět značkovací jazyk SGML (Standard Generalized Markup Language). Pro jeho složitost se později začal vytvářet jazyk nový. Jednalo se o XML (eXtensible Markup Language), které bylo představeno v roce 1998.

XML přineslo do světa informatiky jednoduchost, eleganci a výrazný skok dopředu ve zpracování informací. Je označováno za fenomén současné doby. Nachází vysoké uplatnění ve spoustě aplikací. A to nejen pro přenos dat. XML se využívá pro databázové systémy, tvorbu webových stránek, literatury nebo pro konfigurační soubory aplikací. Jedná se o otevřený formát, který byl standardizován v ISO normě.

Vyniká především svými dobrými vlastnostmi, jako je přenositelnost nebo jednoduché zpracování. Dokumenty v tomto formátu jsou po otevření čitelné. Nejedná se tedy o binární soubory, ale o textové. Často je struktura dat dosti složitá. Tudíž použití běžného textového editoru k jejich vizualizaci, tvorbě nebo editaci nemusí být vždy vhodné.

Cílem práce je vytvořit takový nástroj, který by v oblasti vizualizace, popř. editace XML dokumentů, byl vhodným pomocníkem. Ten by měl umět jednoduše a přehledně prezentovat jejich strukturu a data.

Text je rozdělen do několika navazujících kapitol. V následující, druhé, kapitole je uveden stručný popis technologie XML, její rozšíření a některé nástroje, které s ní pracují. Kapitola se konkrétně zabývá, jaký je původ, charakteristika a struktura těchto dokumentů. Rozšířeními se myslí definiční jazyky. Ty jsou rozebrány podrobněji, jelikož jsou pro XML a vytvářenou aplikaci důležité. Nezapomenu se ani zmínit, jaké jsou současné trendy vizualizace a zpracování těchto souborů.

Třetí kapitola pojednává o implementačním jazyce, ve kterém bude aplikace vyvíjena. Půjde o Javu, která se v poslední době s XML dosti spojuje a to nejen díky svým společným vlastnostem. Tato kapitola popisuje i některé nástroje pro práci s Javou a XML formátem.

Čtvrtá a pátá kapitola se věnuje už samotné aplikaci. Je zde uveden postupný přechod od jejího návrhu až po realizaci. Čtvrtá kapitola specifikuje nároky, které jsou na vytvářenou aplikaci kladeny a co se od ní očekává. Dále jak by měla aplikace ve výsledku vypadat a jakým stylem budou XML dokumenty vizualizovány. Už dopředu můžu prozradit, že půjde o tabulky. Konkrétně pátá kapitola se zaměřuje na nejdůležitější konstrukce v kódu a částečnou implementaci.

Poslední kapitola shrnuje dosažené výsledky. Dále popisuje, kterých cílů se podařilo dosáhnout a které by chtěli ještě zdokonalit. Jsou zde vyzdvihnuty i některé zlepšení a usnadnění, které byli do aplikace během vývoje zaneseny. Kapitola se zaobírá i konkrétními možnými rozšířeními.

2 XML

2.1 Charakteristika

2.1.1 Historie

Lze z jistotou říci, že předchůdcem XML byl jazyk SGML. Ten byl představen v roce 1986, kdy byl definován v ISO normě 8879. Ovšem jeho použití a zpracování bylo značně náročné. Tento fakt vznikal z veliké komplexnosti tohoto jazyka. Svoje uplatnění ale našel. Využívalo jej americké ministerstvo obrany pro správu dokumentace od dodavatelů. Později byl od SGML odvozen další jazyk. Jednalo se o poměrně známé HTML (HyperText Markup Language). HTML se využívá k tvorbě webových stránek. Výčet značek, které se používají, je dán pomocí DTD (Document Type Definition). Jazyk brzy získal velikou oblibu díky své jednoduchosti, avšak nebyl vhodný pro uchovávání a výměnu dat. Na základě zkušeností z HTML se začal vyvíjet jazyk jiný. Bylo to právě XML. První verze se objevila v roce 1998 a byla označena 1.0. Do dneška se stále používá. Existuje i druhá verze označena 1.1. Ta ale nemá oproti předchozí verzi významnější rozšíření.

2.1.2 Využití

XML je standardem konsorcia W3 a v současné době se jedná o jeden z nejdůležitějších formátů výměny dat strukturovaným způsobem. XML během posledních deseti let doznal v této oblasti značného rozšíření díky svým dobrým vlastnostem [2].

Mezi ně se řadí především přenositelnost formátu mezi jednotlivými typy operačních systémů. Často se v tomto směru uvádí symbióza XML dokumentů a programovacího jazyku Java. Ta tuto vlastnost má také. Důvodem přenositelnosti dokumentů je bezesporu jejich textový formát. Ten je možné zobrazit obyčejným textovým prohlížečem. S XML dokumenty běžně pracují i webové prohlížeče. Strukturu zobrazují přehledně a orientace v ní je tudíž velmi snadná.

Další vlastností je otevřený formát XML. Jedná se o zdarma přístupnou specifikaci, která je k dispozici na stránkách konsorcia W3 (viz. [4]). Značky v XML mají pevně stanovenou gramatiku, tak jak tomu je například u HTML. Při vytváření tohoto formátu se dodržuje několik základních pravidel. Pomocí nich lze gramatiku jednoduše kontrolovat. Poslední vlastností, která je pro tento jazyk charakteristická, je značkování uložených dat. Zde je zásadní rozdíl oproti HTML. HTML totiž převážně určuje, jak se data zobrazí, kdežto XML jednoznačně určuje, jaký mají data význam [2].

XML se dělí na dvě velké skupiny. Jednu skupinu tvoří *dokumentově orientované* XML instance. Jedná se o soubory, ve kterých je převážně uchováván text. Ty například nacházejí uplatnění pro tvorbu webových stránek nebo knih.

Druhou oblastí použití jsou *datově orientované* dokumenty. Tyto dokumenty ve většině případů obsahují pouze krátké textové řetězce nebo čísla a jsou strukturované, aby se nemíchal text s dalšími značkami. Podíl textu a značek je tedy téměř vyrovnaný. Takové dokumenty využívají třeba aplikace pro výměnu dat mezi sebou nebo databáze. Určit hranici mezi těmito dvěmi skupinami lze jen s obtížemi [2][4].

Práce a vytvářená aplikace bude dále směrována k této druhé skupině, datově orientovaných dokumentů.

2.1.3 Struktura

Obrázek 2.1 přehledně prezentuje syntaxi a strukturu XML dokumentu.

Obr. 2.1 Základní struktura XML dokumentů [16]

Z obrázku 2.1 lze postřehnout, že struktura zobrazovaného XML dokumentu je značně podobná jazyku HTML. Jedná se o hierarchickou nebo také stromovou strukturu. Oproti HTML přibila

možnost definovat si své vlastní značky. To značně přispívá k vlastnosti XML pro uchovávání a výměnu dat. XML deklarace nebo-li hlavička dokumentu popisuje verzi XML a kódování. Zbylou část tvoří textový obsah dokumentu. Ten je téměř vždy tvořen textovými značkami (elementy), atributy a komentáři. Každý element, pokud není koncový nebo prázdný, má své další potomky. Každý z těchto potomků může být dále tvořen dalšími potomky. Strukturu XML dokumentů přehledně prezentuje obrázek 2.1.

2.2 Definiční jazyky

Definiční jazyky se staly nedílnou a důležitou součástí XML dokumentů. Problémy nastaly při výměně dat pomocí XML formátu, kdy dokumenty mohly mít libovolnou strukturu a bylo možné používat libovolné značky. Mohly se tedy jednoduše stát nekompatibilní.

Definiční jazyky se používají především pro formální definici značkovacích jazyků, jako je XML. Jsou tvořeny sadou formálních pravidel. Výhoda formalizované definice spočívá především v tom, že je jednoznačná a znemožňuje různé interpretace. Pod touto interpretací si lze představit šablonu. Podle ní je možné XML soubor vytvářet, číst a modifikovat. Využívá se samozřejmě i pro validaci. Validace je proces, při kterém se ověřuje, zda nějaký konkrétní XML dokument vyhovuje všem omezením definovaným právě v této definici [3]. Některé typy definičních jazyků dokonce umožňují zavádět pro obsah jednotlivých elementů a atributů datový typ.

Definice dokumentu se také využívá jako zdroj pro vytvoření odpovídajícího objektového modelu včetně kódu, který jej implementuje. V neposlední řadě slouží jako dokumentace pro značkovací jazyk, který je popisován. Pro některé jazyky dokonce existují nástroje, které z původní definice vygenerují dokumentaci. Ta je většinou prezentována v HTML kódu [3].

V dnešní době patří mezi nejpoužívanější definiční jazyky DTD, XML Schema a do popředí se dostává i Relax NG. Jazyk Relax NG začíná být v poslední době dosti oblíbený, vzhledem ke své jednoduchosti a díky odstranění chyb prvních dvou zmiňovaných jazyků. Pro tuto práci je důležité XML Schema, které rozeberu trošku podrobněji.

2.2.1 DTD – Document Type Definition

Významnou vlastností, kterou jazyk XML zdědil od SGML, je koncepce DTD (Document Type Definition). Jde o nejstarší definiční jazyk. Jedná se o volitelnou, ale velmi silnou vlastnost XML dokumentů. DTD popisuje definici struktury. Definuje elementy, které mohou být použity, a určuje, kde mohou být použity ve vztahu k ostatním elementům. DTD tedy zavádí hierarchii i zrnitost dokumentu [1]. I když dnes už není tento jazyk moc využíván, je dobré se o něm zmínit. Hlavním důvodem je, že tento jazyk přišel společně s XML a lze se s ním ještě často setkat.

Výhody a nevýhody DTD

V dnešní době je DTD stále rozšířeným a používaným jazykem. Ale převládají u něj spíše nevýhody než výhody. Hlavní a asi největší nevýhodou je, že sám o sobě netvoří XML dokument. To má neblahý důsledek na to, že vlastní DTD soubor nelze běžnými způsoby kontrolovat na správnost, což je u XML dokumentů považováno za samozřejmost. Další nevýhodou DTD je, že pomocí něj nelze popsat datové typy, jejich rozsahy popřípadě omezení. DTD rovněž vůbec nepracuje se jmennými prostory. Jmenné prostory jsou mechanizmus, který umožňuje v jednom XML dokumentu kombinovat více sad značek [3].

Tyto chyby vedly, již brzy po vzniku XML, k vytvoření zcela nových definičních jazyků. Mnoho z nich brzy zaniklo, používalo se jenom v omezené míře nebo jenom v některých firmách. Konkurencí se později stali definiční jazyky jako je XML Schema a Relax NG. Ty vyřešili většinu nevýhod DTD.

Struktura DTD

Použití DTD má smysl pro větší množství dokumentů, u kterých je potřeba zachovávat stejnou hierarchii. Dokumenty mohou definici obsahovat přímo ve svém těle nebo se mohou odkazovat na externí soubor. První případ je k vidění jen zřídka. Ztrácí totiž svoji efektivnost při větším počtu dokumentů. Pokud se změní definice v jednom z nich, musí se tento postup aplikovat i na ty ostatní. Soubory s externí definicí mají za jménem příponu `.dtd`. K dokumentu se připojují pomocí klíčového slova *DOCTYPE*. Připojení souboru `my_dtd.dtd`, s kořenovým elementem `root` popisuje příklad 2.1.

Příklad 2.1 Připojení formální definice DTD my_dtd.dtd k XML dokumentu

```
<!DOCTYPE root SYSTEM "my_dtd.dtd">.
```

DTD definice se skládají z řady deklarací. Jednotlivé deklarace musí být uzavřeny mezi značkami < ! a >. Ty mohou obsahovat klíčová slova, jako:

- *ELEMENT*
- *ATTLIST*
- *ENTITY*
- *NOTATION*

Jak už názvy napovídají, slovo *ELEMENT* bude sloužit pro definici značek nebo-li elementů. Tato deklarace musí obsahovat název značky, která je shodná s příslušnou značkou v XML dokumentu. Za

názvem následuje zápis, co má značka obsahovat. Obsah může být tvořen dalšími elementy, textem, pokud se jedná o koncovou značku nebo kombinací těchto dvou vlastností.

ATTLIST je určen k deklaraci seznamu atributů. Náleží rovněž příslušnému elementu v XML dokumentu. Deklarace musí obsahovat jméno elementu a k němu výpis atributů s patřičnými vlastnostmi. Mezi ně patří typ atributu a povinnost atributu nebo jeho výchozí hodnota. Typ udává, jaký bude obsah atributu. Variant obsahu se používá až pět. Povinností atributu je myšleno to, zda musí nebo nemusí být v dokumentu použit. Tato hodnota musí být vždy uvedena za typem atributu. Může být ovšem nahrazena již zmiňovanou implicitní hodnotou.

Méně používanými klíčovými slovy v DTD jsou *ENTITY* a *NOTATION*. *ENTITY* se dělí na interní, externí nebo parametrické. První dva typy se využívají u XML dokumentů pro nahrazení určitého řetězce textu nějakou konstantou se stejným významem. Toto využití má smysl pouze, pokud se text v dokumentu opakuje vícekrát než jednou. Parametrické entity se aplikují pouze v rozsahu DTD definic. Jejich účel je obdobný, jako u externích nebo interních entit. Využití opět najdou tam, kde se opakují stejné sekvence textu. Jedná se především o deklaraci elementů nebo atributů se stejnými vlastnostmi. Notace se skrývají pod klíčovým slovem *NOTATION*. Ty slouží pro přiřazení programu určitému typu souboru. Ten pak tyto data zpracovává.

Podrobná charakteristika nebo popis práce s těmito definicemi je uveden v referenční literatuře nebo literatuře, ze které bylo čerpáno (viz. [2][4]).

Příklad 2.2 XML dokument

```
<?xml version="1.0" encoding="UTF-8"?>
<zamestnanec id="vyvoj00">
  <jmeno>Leonardo da Vinci</jmeno>
  <narozeni>1452-04-15</narozeni>
  <rodne_cislo>5204159999</rodne_cislo>
  <telefon>5134</telefon>
  <telefon>2118</telefon>
</zamestnanec>
```

XML dokument v příkladě 2.2 obsahuje kořenový element `<zamestnanec>` s atributem *id*, který popisuje zaměstnance fiktivní firmy. V něm jsou dále obsaženy další elementy, které určují jméno, datum narození, rodné číslo a telefony zaměstnance.

Níže, v příkladě 2.3, je uvedena ukázka formální definice DTD pro část XML dokumentu z příkladu 2.2. Ta popisuje pouze uzel *zamestnanec*, jeho atribut *id* a potomky *jmeno* a *telefon*, které uzel *zamestnanec* obsahuje. Na příkladě 2.3 je vidět použití klíčových slov *ELEMENT* a *ATTLIST*.

Příklad 2.3 Formální definice DTD popisující část XML dokumentu pro příklad 2.2

```
<!ELEMENT zamestnanec (jmeno,telefon+)>
<!ATTLIST zamestnanec
  xmlns CDATA #FIXED ''
  id  #REQUIRED>

<!ELEMENT jmeno (#PCDATA)>
<!ATTLIST jmeno
  xmlns CDATA #FIXED ''>

<!ELEMENT telefon (#PCDATA)>
<!ATTLIST telefon
  xmlns CDATA #FIXED ''>
```

2.2.2 Relax NG

V poslední době se tento definiční jazyk stává velice oblíbeným a to díky své jednoduchosti. Začíná být pomalu přijímán i konsorciem W3. Jazyk slouží, jako DTD nebo XML Schema, k formální definici XML dokumentů. Avšak oproti nim odstraňuje většinu jejich chyb. Nedostatky, které obsahovalo DTD byli popsány v kapitole 2.2.1. Relax NG odstraňuje i dvě základní nevýhody XML Schema. Těmi jsou složitá specifikace a neelegantní zápis.

Zatímco XSD (XML Schema Definition) je založeno na datových typech, je Relax NG založen na vzorech. Celé schéma je vzorem dokumentu. Vzor se přitom skládá ze vzorů pro elementy, atributy a textové uzly. Ty pak mohou být dále kombinovány do uspořádaných i neuspořádaných skupin, mohou být volitelné a může u nich být určen i počet opakování. Tento jednoduchý princip umožňuje velmi jednoduše vyjádřit i složité struktury v dokumentu. Navíc je založen na solidním matematickém základu alejových gramatik (hedge grammars) [3].

Přípona pro tyto typy souborů je `.rng` a `.rnc`. Rozdíl mezi nimi je v jejich zápisu syntaxe. *RNG* soubory mají klasickou XML strukturu. V případě souborů *RNC* jde o textový nebo jinak řečeno kompaktní záznam definice. Ten plní stejnou úlohu, jako XML zápis, ale je mnohem úspornější.

Následující příklad 2.4 tvoří definici Relax NG pro část XML dokumentu z příkladu 2.2. Definován je pouze element *zamestnanec*, jeho atribut *id* a potomci *jmeno* a *telefon*.

Příklad 2.4 Formální definice Relax NG popisující část XML dokumentu pro příklad 2.2

```
<start>
  <element name="zamestnanec">
    <attribute name="id">
      <data type="string"/>
    </attribute>
    <element name="jmeno">
      <text/>
    </element>
    <oneOrMore>
      <element name="telefon">
        <data type="integer"/>
      </element>
    </oneOrMore>
  </element>
</start>
```

2.2.3 W3C XML Schema

W3C XML Schema je v dnešní době nejpoužívanější definiční jazyk.

Od května roku 2001 se stal všeobecně uznávaným standardem, který je v softwarové praxi podporován všemi významnými firmami, jako je například IBM, Sun, Microsoft nebo Oracle. W3C XML Schema se také někdy označuje zkratkou WXS (W3C XML Schema), ale častěji se používá zkratka XSD (XML Schema Definition). XSD odstraňuje většinu nevýhod DTD. Oproti DTD je XSD dokument tvořen XML strukturou. Další jeho vlastností je, že jednotlivým hodnotám elementů a atributů lze přiřadit datový typ. A poslední důležitou vlastností, která oproti DTD přibyla, je práce s jmennými prostory. Jazyk má ale dvě nevýhody. Těmi je složitá specifikace, která vzniká velikou komplexností toho jazyka a chybí mu jistá elegance při interpretaci. Zejména pro jednoduché XML soubory může být vytvořená definice několikanásobně větší, než je samotné XML [2]. To je například patrné na příkladě 2.5.

Struktura a datové typy XSD dokumentů

Jak už jsem výše uvedl, XSD dovoluje přiřazovat hodnotám elementů a atributů datové typy. To je jedna z vlastností, proč vůbec tento jazyk přišel na svět. Na výběr je z velké palety již definovaných

typů, s nimiž XSD pracuje. Mezi ty základní se řadí především typy pro práci s datem, časem, textovými řetězci, reálnými, desítkovými a celými čísly a logickými hodnotami. Pomocí určitých omezení, je možné z těchto daných typů vytvořit uživatelem požadovaný datový typ. Existuje možnost rozšířit již existující datové typy. Kompletní seznam datových typů, které jsou k dispozici, je uveden v referenčním manuálu o XSD (viz. [5]).

Nově definované datové typy se dělí na dvě kategorie. Jednoduché a složené. Složené datové typy jsou v dokumentech prezentovány značkou `<xs:complexType>` a jsou tvořeny jednoduchými datovými typy. Jednoduchý datový typ se značí `<xs:simpleType>` a je odvozen od základních, již existujících datových typů. Vždy ho tvoří koncový element nebo atribut.

Na jednoduché datové typy se váží výše zmíněné omezení zvané také restrikce. Princip je takový, že se vybere jednoduchý datový typ a podle očekávání charakteru dat je snahou nalézt co možná nejpřísnější restrikci [2]. V praxi to znamená, že na již existující datový typ se aplikuje integritní omezení tak, aby se zúžila škála přípustných hodnot.

Restrikce se dělí hned do několika skupin. Ty jsou dány podle charakteru datového typu. Jako příklad nejprve uvedu omezení pro číselné hodnoty nebo data. U nich se určuje například nejnižší možná hodnota nebo naopak, nejvyšší možná hodnota. Čísla se omezují i na délku. Další takovou jednoduchou restrikcí je omezení počtu znaků v textových řetězcích. Rovněž se využívá intervalu, v jakém délka řetězce může být. V neposlední případě se zmíním i o typu restrikce pro výčet hodnot. To znamená, že hodnota příslušného elementu nebo atributu v XML dokumentu, se musí shodovat s jednou z hodnot uvedených v seznamu u tohoto omezení.

Z vytvořených jednoduchých datových typů je možné tvořit složené nebo-li komplexní datové typy. Ty slouží k definici struktury dokumentu. Pomocí nich se u elementů určuje, v jakém pořadí se mají vyskytovat, kolikrát se mohou opakovat, zda jsou povinné, či volitelné [3]. Složené datové typy jsou tvořeny elementy *sequence*, *choice* a *all*. Element *sequence* je z těchto tří variant nejpoužívanější. Definuje přesné pořadí elementů v něm obsažených. Obsažené elementy musí po sobě následovat tak, jak jsou zapsány v XSD. Počet výskytů lze u elementů nastavit atributy *minOccurs* a *maxOccurs*, přičemž implicitně mají hodnotu 1. Pokud je atribut *minOccurs* nastaven na hodnotu 0, může být element vytvořen, anebo taky nemusí. Pořád ale platí, že jeho výskyt může být nanejvýš jednou, dokud se nenastaví atribut *maxOccurs*.

Nyní se dostávám k popisu využití elementu *choice*. Používá pro situaci, kdy je třeba v syntaxi XML dokumentu deklarovat pouze jeden element z určitého výčtu, definovaného v konstrukci definice. Z té se vybere a použije libovolný element, avšak jen jeden.

Poslední uvedenou konstrukcí je *all*. Ta představuje docela svobodný model definice, kdy uvedené elementy mohou být skládány v libovolném pořadí. Jistá omezení zde však jsou. Uvedené elementy se mohou vyskytovat maximálně jednou anebo vůbec.

U XSD jazyků, stejně jako tomu je u XML, existuje pojmem *smíšený obsah*. Smíšený obsah znamená kombinaci textu a elementů, které definuje složený datový typ. Tak jsou například tvořeny

XHTML (eXtensible HyperText Markup Language) dokumenty, které jsou odvozeny od HTML, avšak mají přísnější specifikaci. XHTML se řídí právě XSD. Takovým případem je například odstavec `<p>`, který obsahuje text. V něm můžou být některé úseky textu zvýrazněny elementem `<b>`.

Důležitou součástí komplexních datových typů jsou i atributy. Definice vlastností atributů je obdobná, jako u DTD. Každý atribut opět obsahuje deklaraci jména, uvedení, zda je jeho uvedení povinné nebo jakou má implicitní hodnotu. XSD tyto vlastnosti oproti DTD rozšiřuje ještě o určení datového typu atributu. Jejich škála je stejná jako pro elementy.

Pro lepší pochopení problematiky XSD, především jeho struktury a skládání jednotlivých deklarací, uvedu jednoduchý příklad. Základem je XML dokument v příkladě 2.2. Vytvořené XML Schema by mělo dodržet konzistenci této předlohy. To znamená, že elementy by měli být skládány v takovém pořadí, v jakém jsou uvedeny. Definice by měla dále určovat počet opakování pro uzly jako je `<narozeni>` nebo `<telefon>`.

Vytvořenou XSD definici pro XML dokument z příkladu 2.2 prezentuje následující příklad 2.5.

Příklad 2.5 Vytvořené XML Schema pro XML dokument v příkladě 2.2

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="zamestnanec">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="jmeno" type="xs:string"/>
        <xs:element name="narozeni" type="xs:date"
                    minOccurs="0" maxOccurs="1"/>
        <xs:element name="rodne_cislo" type="xs:integer"/>
        <xs:element name="telefon" type="xs:integer"
                    maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Důležitým a prvním znakem vytvořené XSD definice, je její struktura. Ta sama o sobě tvoří XML dokument. XSD pro prezentaci stromu využívá speciálních značek. Celý dokument je uzavřen značkou `<xs:schema>`. Pro ni se zavádí atribut `xmlns:xs`, který říká, že všechny obsažené elementy musí patřit do jmenného prostoru *http://www.w3.org/2001/XMLSchema*. Většinou se před tento jmenný prostor uvádí prefix *xs* nebo také *xsd*.

Elementy definice se dělí do dvou skupin. První skupinou jsou ty, které slouží pro popsání struktury dokumenty. Druhá skupina popisuje datové typy. Datové typy jsou tvořeny elementy `<xs:element>` pro definici elementů a `<xs:attribute>` pro definici atributů. U všech deklarací datových typů musí být v atributu name uvedeno jméno prvku, kterému náleží. Teprve poté následuje výpis vlastností.

Následujícím elementem za kořenovým uzlem, je výše popisovaný element `<xs:element>`. Ten má jméno *zamestnanec* (v atributu *name*). Při pohledu do dokumentu z příkladu 2.2, ze kterého vytvořená definice vychází, je patrné, že obsahuje potomky a vlastní atribut. V tomto případě se tedy bude jednat o komplexní nebo-li složený datový typ. U složených datový typů se v pořadí nejprve definují uzly a pak až atributy. Potomci elementu `<zamestnanec>` jsou uvedeni v konstrukci *sequence*. Ta pevně stanoví, v jaké posloupnosti musí být elementy skládány a obsahuje již jen jednoduché datové typy. Kromě uvedeného jména a datového typu, zavádí definice pro některé elementy počet opakování. Jedním z nich je element `<narozeni>`, pro který byl definován uzel `<xs:element>` se jménem *narozeni*. Ten nemusí být v posloupnosti uveden vůbec. Nebo může, ale nanejvýš jednou. Oproti tomu element `<telefon>` musí být, podle definice, uveden vždy a alespoň jednou. Ale může se opakovat až do nekonečna. Zbývající elementy se musí vyskytovat vždy, avšak bez opakování.

Předešlým odstavce byla rozebrána první část komplexního datového typu. Druhá část definuje atributy. Atribut je definicí popsán tak, jak je to uvedeno v teorii. Obsahuje svoje jméno, jakého je datového typu a jestli je povinný. V tomto případě musí být uveden vždy a je typu řetězec. Složitější případ může nastat tehdy, když je element konečný, obsahuje atribut(y) a neobsahuje již žádné další potomky.

2.3 Nástroj Trang

Trang je nástroj, který byl vytvořen pro konverzi mezi jednotlivými definičními jazyky. Pomocí něj je také možné z XML dokumentu tuto definici vygenerovat. Definici je schopen vytvořit i pro více, než jeden XML dokument. Byl implementován v jazyce Java. Nemá grafický interface a je ovládán přes příkazový řádek. Celá koncepce vychází z jazyku Relax NG. Jedná se o open source a freeware zároveň. K dispozici je volně stažitelný na internetu (viz. [17]). Podporovanými definičními jazyky jsou všechny, které byly uvedeny v předchozí kapitole 2.2. Jedná se o Relax NG, DTD a XML Schema. Pro Relaxu NG pracuje s oběma formáty. XML stromem (přípona `.rng`) a kompaktním zápisem (přípona `.rnc`). Snahou tohoto nástroje je generovat definice, tak aby jim uživatelé snadno porozuměli.

Jistá omezení při konverzi mezi jednotlivými typy jsou. Trang nedovede například převádět XSD dokumenty na ostatní definiční jazyky. Podporované formáty se tedy dělí do dvou skupin. Tu první tvoří vstupní formáty. Mezi ně patří:

- XML dokument (`.xml`)
- Relax NG (`.rng`)
- Relax NG (`.rnc`)
- DTD (`.dtd`).

Všechny tyto uvedené jazyky je možné konvertovat na výstupní formáty. Zde se řadí:

- Relax NG (`.rng`)
- Relax NG (`.rnc`)
- DTD (`.dtd`)
- XML Schema (`.xsd`).

Důležitým faktorem je, že Trang je schopen pro XML dokumenty vytvářet všechny čtyři typy formálních definic, pokud se berou v úvahu obě definice Relax NG.

2.3.1 Práce s Trangem

Trang je ke stáhnutí ve spustitelném formátu *jar*. To je typický formát pro Javu. Pro práci s ním je třeba mít Javu nainstalovanou. Ta by měla být ve verzi Standard Edition 1.4 a vyšší. Jestliže uživatel používá nižší verzi než je uvedená, mohl by se setkat s komplikacemi. Příkaz pro spuštění Trangu prezentuje příklad 2.6.

Příklad 2.6 Příkaz pro spuštění Trangu z příkazové řádky

```
java –jar trang.jar argumenty
```

Trang se spouští příkazem *java*. Za něj a za parametr `-jar` se zadává adresářová cesta k *jar* souboru *trang.jar*. Dále následují argumenty. Zde se uvádí dva dokumenty, které bude Trang zpracovávat. První je výchozí dokument, který se bude transformovat. Druhým je název dokumentu, do kterého se bude transformace ukládat. Argumenty se můžou blíže specifikovat pomocí parametru *–I* a *–O*. Parametr *–I* definuje vstupní dokumenty. *–O* naopak výstupní. Důvodem použití bližší specifikace může být větší počet souborů na vstupu nebo jejich zpřeházení.

2.4 Vizualizace XML

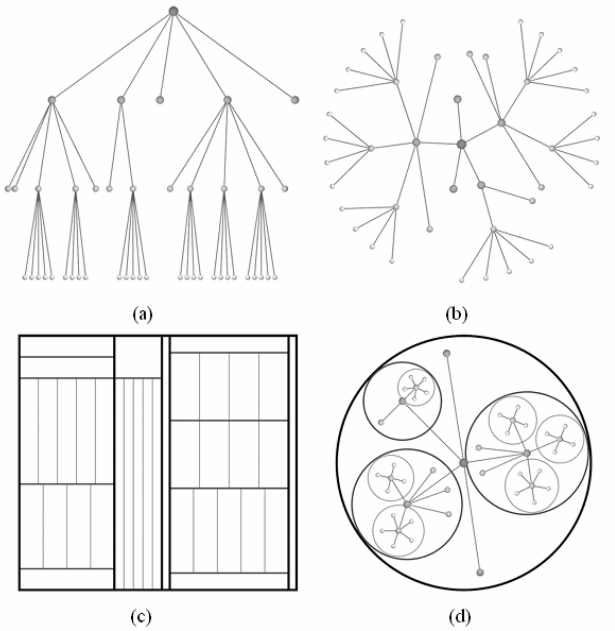
Tato kapitola se zabývá problematikou vizualizace XML dokumentů. Jejím účelem je popsat některé zavedené metody vizualizace hierarchických struktur, jako je XML. Dále uvádí vytvořené nástroje, které s těmito formáty dokumentů již pracují.

Hlavní otázkou v této kapitole je, jak data vizualizovat? Každý z těchto následně uvedených nástrojů formuluje nebo také zobrazuje dokumenty trochu jiným způsobem. XML často může být příliš strukturované až dosti komplikované. Tato vlastnost má ve většině případů negativní vliv na přehlednost. Otázku je tedy možné doplnit. Jak vizualizovat XML dokumenty, tak aby zobrazená data byli přehledně a jednoduše uspořádána?

Jednotlivé příklady budou prezentovány na XML dokumentu *weather.xml* [8].

2.4.1 Vizualizace stromovou strukturou (DOM)

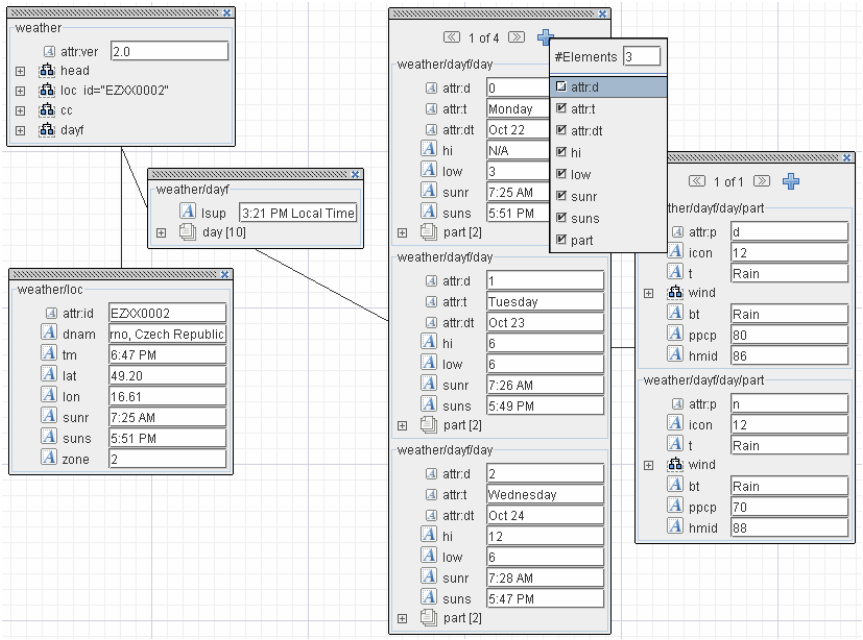
Tato metodika je pro XML dokumenty rozšířená i přehledná. Jedná se v podstatě o nejjednodušší možné zobrazení. Využívají ji například webové prohlížeče. Struktura XML dokumentu je vizualizovaná pomocí stromu nebo podobného modelu. Častou vlastností bývá, že jednotlivé uzly jsou schopny se rozbalovat nebo zpět zabalovat. Zobrazuje se tedy jen ta část, která je potřeba. Jednodušeji se dá pak v dokumentu zorientovat. Pro vizualizaci stromovou strukturou je definováno několik různých typů modelů. Ty popisuje obrázek 2.2 a jsou čtyři. První (a) je kořenová stromová struktura. Ta je asi nejběžnější. Další je stromová struktura ve tvaru hvězdice. Předposlední a poslední metodikou je rozvržení stromovou mapou a kruhová topologie.



Obr. 2.2 Techniky vizualizace hierarchických dat [18]: (a) kořenový strom, (b) hvězdicová topologie, (c) rozvržení stromovou mapou,(d) kruhová topologie.

2.4.2 Zobrazení založené na UML

Na obrázku 2.3 je znázorněn pohled do aplikace VisualXML [6], která modeluje XML dokument *weather.xml* a poskytuje tak rychlý úvod do problematiky.



Obr. 2.3 Zobrazení souboru weather.xml pomocí nástroje VisualXML [6]

Aplikace VisualXML je založena na UML (Unified Modeling Language) [19]. UML je charakterizován jako grafický jazyk pro vizualizaci, modelování a navrhování informačních systémů. Diagramy modelují elementy jako entity nebo třídy. Přístup VisualXML avšak do nich přidává data obsahující atributy a jednoduché sub-elementy s jejich hodnotami jednotnou cestou. Navíc tento způsob vizualizuje kolekci atributů a sub-elementů jako seznamy.

Je to jedna z dalších možností, jak zobrazovat obsahy XML dokumentů. Nástroj VisualXML je dobrý, jednoduchý a intuitivní na ovládání. Pracovní plocha této aplikace se skládá ze dvou částí. První část tvoří kořenový strom zachovávající strukturu XML dokumentu. Ta slouží především pro rychlou orientaci v obsáhlejších XML dokumentech. Druhou částí je pracovní plocha, na které jsou dokumenty modelovány. Tento nástroj umožňuje dále mazat, přidávat a modifikovat elementy a atributy.

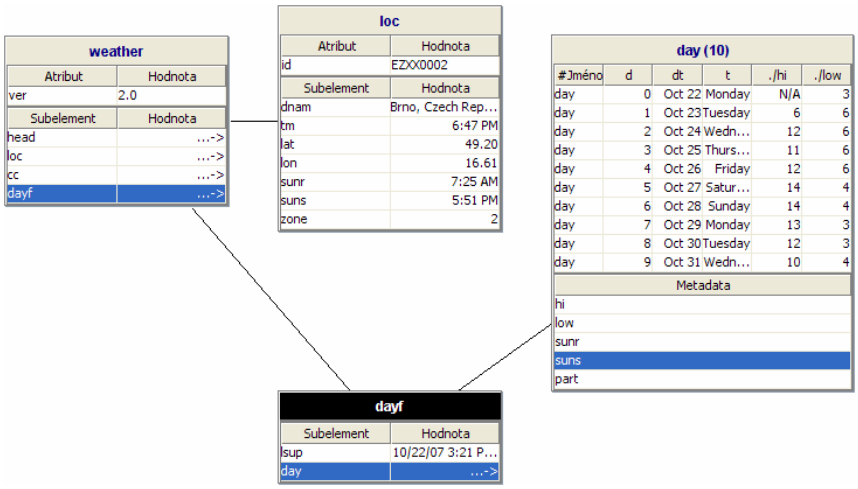
2.4.3 Tabulkově-orientovaný přístup

Myšlenka tohoto přístupu k zobrazování XML dokumentů, je použít klasické relační tabulky. V praxi se dá tabulka označit za strukturu záznamů s pevně stanovenými položkami (sloupci - atributy) [9]. Tabulky jsou ale často kritizovány kvůli své špatné flexibilitě pro zobrazování hierarchických a

objektových dat, přesto se od nich tento způsob nechává inspirovat. Jsou schopny přehledně a efektivně zobrazovat kolekce dat.

Klíčovým aspektem aplikací, založených na tomto principu, je najít kolekce záznamů v XML. Nalezení takových struktur je jednoduché, protože XML dokument je tvořen elementy. Úlohou takovéto aplikace je tedy najít takový element, který obsahuje kolekci sub-elementů (potomků). Tu pak zobrazit.

Existuje aplikace, která zpracovává XML dokumenty právě tímto způsobem. Její název je XMLAD [7]. Jak aplikace funguje a jak jsou data vizualizovaná, prezentuje obrázek 2.4. Jedná se opět o zobrazení souboru *weather.xml*.



Obr. 2.4 Vizualizace XML dokumentu weather.xml tabulkově-orientovaným přístupem pomocí aplikace XMLAD

Na obrázku 2.4 je vidět omezení tabulkově-orientovaného přístupu. Zobrazená kolekce musí být modelována relačně. Relační model je definovaný tak, že sdružuje data do tzv. relací (tabulek), které obsahují n-tice (řádky) [9]. XMLAD je nástroj, který zobrazuje XML data efektivně. Jedná se v podstatě zatím o jedno z nejlepších možných řešení pro zobrazení XML dat. Na obrázku 2.4 tabulka „day“ obsahuje samotná data a dále metadata. Metadata lze přidávat do tabulky jako nové sloupce. Jsou to další potomci elementů `<day>`. Samotná data jsou tvořena atributy a jejich hodnotami.

3 Implementační jazyk Java

3.1 Stručná historie

Historie programovacího jazyku Java spadá až do roku 1991. Java byla již od svého počátku vyvíjena společností Sun Microsystems pod vedením Jamese Goslinga. Její využití se mělo především uplatnit pro tzv. „vestavěné systémy“, což jsou běžná zařízení, ovládány mikročipem. Java byla vyvíjena na principech programovacích jazyků C a C++. První jméno pro tento jazyk bylo *Oak* (v překladu dub). Název vznikl podle dubu rostoucího před Goslingovou kanceláří. Později se zjistilo, že již programovací jazyk s tímto názvem existuje. Jméno bylo tedy změněno na Java. Existuje teorie, že slovo bylo vybráno ze seznamu náhodných slov.

Prvního představení se Java dočkala v květnu roku 1995 na konferenci SunWorld. Jednalo se o verzi 1.0. Při svém představení odhalila spoustu svých kladných vlastností. Stala se tak velice brzy populární a byla začleněna do většiny webových prohlížečů jako *applety*.

Applety jsou softwarové komponenty, které běží v kontextu jiného programu, typicky právě webového prohlížeče. Bývají většinou orientovány na plnění konkrétních funkcí a nepředpokládá se, že bude používán jako samostatná aplikace. Z bezpečnostních důvodů se applety spouštějí v „sandboxu“. To znamená, že mezi nimi a systémem vzniká bariéra, díky níž nemůžou applety přistupovat k souborovému systému a provádět tak potenciálně nebezpečné činnosti [12].

S příchodem nové verze Java 2, byl jazyk rozdělen na jednotlivé typy podle platformy, na které pracoval. Byla to J2EE (Java 2 Enterprice Edition) pro vývoj a provoz webových aplikací a služeb, J2ME (Java 2 Micro Edition) pro mobilní zařízení a J2SE (Java 2 Standard Edition), jako standardní verze pro tvorbu konzolových a grafických aplikací.

3.2 Charakteristika

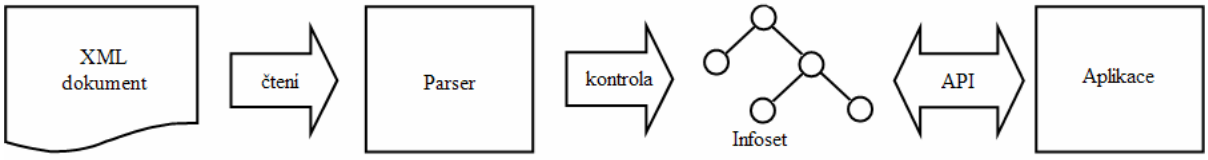
Java je jedním z nejrozšířenějších a nejpoužívanějších programovacích jazyků na světě. Je objektově orientovaná a svojí syntaxí se inspiruje od jazyků C a C++. Oproti nim však odpadla spousta konstrukcí, které nebyly vhodné. Ty byly z jazyka buď odstraněny úplně nebo nahrazeny jinými, jednoduššími, principy. Velkou výhodou, která dělá k Javy tak mocný nástroj, je její přenositelnost mezi různými platformami operačních systémů. Je tedy jedno, na jakém počítači je vytvářená aplikace spouštěna. Soubory s kódem mají příponu `.java`. Přeložený program, *bajtkód*, se ukládá do souborů typu `.class`. Tento soubor je pak z disku zaváděn do paměti počítače a současně probíhá ověření bajtkódu, což je možné provést jednotně díky jeho nezávislosti na platformě [13]. Program je po zpracování a zkontrolování spouštěn virtuálním strojem (*Java Virtual Machine*). Jde tedy o interpretovaný jazyk.

Virtuální stroj v této souvislosti považuje za počítačový software, který izoluje aplikace od počítače. Protože verze virtuálního stroje jsou psány pro různé počítačové platformy, jakákoliv aplikace psaná pro virtuální stroj může být provozována na kterékoliv počítači [15].

3.3 Java a XML

Programovací jazyk Java tvoří společně s XML velmi dobrou symbiózu. Java je přenositelný programovací jazyk a XML je přenositelný formát pro popis dat [2]. Proto také Java nabízí spoustu možností pro práci s tímto formátem dokumentů. Při zpracovávání XML není možné tento typ dokumentů považovat za textové soubory. Ke zpracování se tedy využívá speciálních nástrojů, zvaných *parsery*. Ty jsou dostupné v různých podobách ve většině implementačních jazycích. Mezi základní vlastnosti parserů bychom mohli zařadit čtení, modifikaci a vytváření XML souborů. Čtení je bezesporu nejpoužívanější funkcí. Tyto nástroje by dále měli umožňovat kontrolu správné strukturovanosti dokumentu. Parserům by ani neměla činit problémy práce s komentáři, CDATA sekcemi nebo entitami.

Zpracovaný XML dokument může být nabízen několika způsoby, proto existují různá rozhraní (API – Application Programming Interface). Ty se dělí na dvě velké skupiny. První skupinou je *událostmi řízené zpracování*. Tu druhou tvoří *stromová prezentace* dokumentu. Postup čtení dokumentu přes rozhraní znázorňuje obrázek 3.1.



Obr. 3.1 Čtení XML dokumentu aplikací, přes rozhraní nabízené parserem

3.3.1 Událostmi řízené zpracování

Tato metoda se rovněž označuje jako proudové čtení. Typickým představitelem je rozhraní SAX (Simple API for XML). SAX původně vznikal pro jazyk Java. V dnešní době se používá, jako novější verze, v dalších programovacích jazycích. Jedná se o rozhraní SAX2. To podporuje na rozdíl od první verze i jmenné prostory a některé další užitečné vlastnosti [10]. SAX2 se stal Součástí Javy od verze JDK 1.4 (Java Development Kid 1.4).

Velkou výhodou událostmi řízeného zpracování je rychlost a nízké paměťové nároky. XML dokument je zpracováván pouze v jednom sekvenčním průchodu. Na uživateli je, aby si v tomto průchodu odchytil pro něj důležité informace a ty si pamatoval ve stavových proměnných.

3.3.2 Stromová prezentace dokumentu

Stromová prezentace je jednodušší technika zpracování XML dokumentů, oproti událostmi řízenému zpracování. Principem je načtení celého XML dokumentu do paměti. Dokument je ukládán jako stromová struktura, kde je zpřístupněn pomocí objektů. Ty jsou tvořeny elementy, atributy, komentáři nebo textovými prvky. Nejznámějším zástupcem této kategorie je DOM (Document Object Model). Jeho první využití plnilo úlohu pro tvorbu interaktivních webových stránek. Pomocí JavaScriptu a zmiňovaného DOMu bylo možné přistupovat k jednotlivým značkám HTML stránek. Později toto rozhraní standardizovalo konsorcium W3 a objevila se verze DOM1. Ta obsahovala dvě rozdílné verze rozhraní. Jedna pracovala právě s HTML a druhá s XML dokumenty. Druhé rozhraní začalo být podporováno tzv. *DOM XML Parsery*. Po verzi DOM1 se ještě objevila verze DOM2. Ta oproti té první pracovala i se jmennými prostory. Přinesla však další vylepšení v podobě snadnějšího průchodu stromem, a podobně.

Výhodou těchto parserů je libovolný a opakovaný průchod strukturou XML. To znamená, že je možné se vracet zpět, přeskakovat libovolné části, nebo číst jen určité oblasti. Oproti událostmi řízenému zpracování lze do načteného dokumentu přidávat nové objekty, nebo modifikovat ty stávající. Další možností je, vytvořit od základu nový XML dokument.

Nesmím opomenout úzkou spolupráci DOMu a jazyku XPath. XPath se využívá pro dotazování nad zpracovaným dokumentem. Dotazy mají podobu cesty v adresářové struktuře a je jím vybrána jen patřičná část XML dokumentu.

Nevýhodou stromové prezentace dokumentu je paměťová náročnost. Uložený XML dokument se může v paměti svojí velikostí roztáhnout dvakrát až desetkrát. To je zapříčiněno objektovým modelem dokumentu. Další nevýhodou, která plyne z té první, je dlouhé načítání do paměti. Tuto nepříjemnou vlastnost způsobují především větší XML dokumenty.

3.4 JDOM

JDOM (Java Document Object Model) byl vyvíjen za účelem zjednodušení obecně složitého rozhraní DOM. Plní převážně tutéž úlohu, akorát s menším úsilím vynaloženým ze strany vývojáře. Neobsahuje svůj vlastní parser. K tomuto účelu využívá libovolný existující parser. Implementačním jazykem se mu stala Java.

Pokud chce vývojář pracovat s XML dokumentem pomocí JDOMu, musí nejprve dokument načíst do paměti klasickým DOMem. Teprve poté předat ukazatel na tuto strukturu. JDOM, oproti DOMu, umožňuje snazší manipulaci s celým XML dokumentem. Jde především o průchod stromem, přidávání, editaci a mazání elementů a atributů a v neposledním případě i práci s ostatními informacemi, jako jsou komentáře, jmenné prostory, atd.

I když se jedná o dobrý a používaný nástroj, není součástí ani Java Core API v JDK 1.6, ani JWSDP 2.0 (Java Web Services Developer Pack 2.0).

3.5 Vývojové prostředí NetBeans

NetBeans je Open Source projekt založen v roce 2000. Zakladatelem se stala firma Sun Microsystems. Počátky NetBeans spadají do roku 1997, kdy byl vyvíjen, jako projekt pod názvem Xelfi, studenty na Matematicko-fyzikální fakulty Univerzity Karlovy v Praze. Studenti i po dokončení studií na projektu pracovali a z Xelfi se stal komerční produkt. Jméno bylo později změněno na NetBeans, aby přesněji vystihovalo podstatu projektu. NetBeans byl nástroj sloužící pro vytváření aplikací v Javě. Sám byl i v tomto jazyce napsán. V roce 1999 se o tento projekt začala zajímat firma Sun Microsystems a koncem tohoto roku ho koupila. V polovině roku 2000, konkrétně v červnu, se z komerčního produktu stal Open Source projekt. Sun Microsystems je do nynějška i jeho hlavním sponzorem. NetBeans je dnes vytvářen stále se rozšiřující komunitou vývojářů. Má přes 100 partnerů po celém světě. Nabízenými produkty tohoto vývoje je vývojové prostředí NetBeans (NetBeans IDE) a vývojová platforma NetBeans (The NetBeans Platform).

NetBeans IDE (Integrated Development Environment) je vývojový nástroj využívající platformu NetBeans. Slouží především k tvorbě aplikací. V posledních letech kromě podporované Javy, pracuje i s programovacími jazyky, jako je C, C++, Ruby a další. Tento nástroj je spustitelný na mnoha různých operačních systémech. Podporovanými jsou Windows, Linux, Mac OS a Solaris. Poslední verzí, která se objevila ke stažení, je NetBeans IDE 6.0. Ta byla vydána v prosinci loňského roku. Oproti starším verzím nabízí snazší instalaci, jednoduchou aktualizaci softwaru, lepší Swing GUI (Graphical User Interface), nástroj pro práci s XML Schema, UML modelování a spoustu další užitečných vlastností. Za zmínku stojí ještě stojí rozšíření existující Javy EE.

NetBeans pro vývoj aplikací v jazyce Java mají velice schopný a interaktivní editor kódu. Automatické doplňování určitých úseků kódu je jen malou částí pozitivní stránky tohoto nástroje. Editor zároveň v průběhu psaní kódu kontroluje správnost syntaxe a chyby hlásí. V některých případech je snahou NetBeans navrhnout možné řešení problému a to po volbě uživatele posléze automaticky doplnit. Z toho plyne, že vytvářená aplikace je většinou kompilována pouze s malou pravděpodobností výskytu chyby. Další pozitivní vlastností je tvorba grafického vzhledu aplikace.

4 Koncepce aplikace

Tato rozsáhlejší kapitola a kapitoly v ní obsažené, charakterizují požadavky na vytvářenou aplikaci. Hlavním cílem vytvářené aplikace, na který bude kladen největší nárok, je jednoduché zpracování a přehledné zobrazení XML dokumentů. Druhým požadavkem je, aby aplikace měla schopnost vytvořit a následně zobrazit formální definici pro zobrazovaný XML dokument. Postup návrhu programu je možný rozdělit do několika základních podproblémů.

- Zpracování XML
- Vizualizace
- Manipulace s obsahem

Těmito podproblémy se následně zabývají jednotlivé kapitoly.

Zatím jsem se nezmínil, jakým způsobem budou data vizualizovaná. Přístupem k jednoduchému zobrazení obsahu XML dokumentů by měli být tabulky. Tato metoda se nechává částečně inspirovat tabulkově orientovaným přístupem. Ty ovšem nebudou zobrazovány v relačním modelu. Aplikace by měla obsahovat jedinou tabulku, která zobrazí uživatelem vybranou část XML. Tím je myšlen element nebo kolekce elementů, většinou potomků s příslušnými hodnotami a atributy.

Jelikož tabulkami ve většině případů nejde zobrazit celý XML dokument, je potřeba se v něm nějak orientovat. Pro zobrazení celé struktury dokumentu je v aplikace použita stromová prezentace.

4.1 Zpracování XML

Před samotným zpracováním, je XML dokument třeba načíst. Uživatel by měl mít možnost vyhledávat soubor v celé adresářové struktuře počítače. Po dokončení výběru dokumentu je aplikaci předává adresářová cesta i název souboru. Struktura obsažená v dokumentu je následně uložena do paměti pomocí nástroje DOM.

Dokument je možné v aplikaci procházet pomocí nástroje JDOM a rekurze. Jedná se o jednoduchý postup, tudíž není problém zobrazit celou strukturu dokumentu pomocí libovolné stromové prezentace. Java pro tento účel má vlastní komponentu. Aplikace by dále měla umožnit se stromem pracovat. Nabízí se, jako nejjednodušší možné řešení, využít myš. Komponenta v Javě, pro prezentaci stromu, umožňuje jistým způsobem pouze strom rozbalovat nebo zabalovat. Obsluhu nad jednotlivými uzly stromu je možné vyřešit pomocí kontextového menu.

Na uživateli je, který element nebo kolekci elementů si nechá zobrazit. Bude se zřejmě jednat o složitější proces. Jde o to, že stromová struktura prezentována komponentou je složena pouze z řetězců. Jestliže si uživatel vybere libovolný uzel, musí být podle této šablony nalezen v XML

dokumentu. Strom je procházen paralelně s XML dokumentem, dokud se nenajde požadovaný uzel. Při nalezení je možné uzel i jeho obsah zobrazit.

Další důležitou vlastností aplikace, kromě načtení dokumentu a jeho procházení, je i ukládání modifikovaných dat. Pokud je dokument libovolně upraven, uživatel by měl mít možnost ho uložit. Editovaná nebo vytvářená data dokumentu se během vizualizace ukládají do struktury v paměti. Tam zůstávají po celou dobu práce uchovávána. Pro uložení dokumentu na disk je použit opět, jako pro načtení, DOM parser.

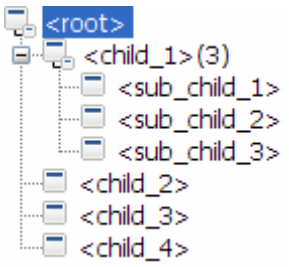
4.2 Vizualizace

V této části bakalářské práce je popsána idea, jak vizualizovat obsah XML dokumentů ve vytvářené aplikaci. Kapitola se konkrétně zabývá vizualizací hierarchie XML pomocí stromové struktury a obsahu XML pomocí tabulek.

4.2.1 Stromová struktura

Stromová struktura dokáže zobrazit celou hierarchii XML dokumentu i s patřičnými hodnotami a atributy. Ve vytvářené aplikaci půjde pouze o holé zobrazení. Metoda bude modelovat elementy obsažené v dokumentu pouze tak, jak jsou umístěné v hierarchii a jak jdou po sobě. Neponesou tedy s sebou žádné další informace o atributech a hodnotách elementů a atributů. Účelem stromového zobrazení je přinést uživateli co nejjednodušší orientaci a přehled ve struktuře dokumentu. Další důležitou vlastností, kterou stromová struktura poskytuje, je volba uzlů, které mají být vizualizovány tabulkou. Ta je zadávána uživatelem. Pro zlepšení orientace v dokumentu nebude na škodu u každého elementu uvést i jeho počet potomků, pokud nějaké obsahuje. Opět je tím snaha přinést nějaké zjednodušení.

Stromová struktura je první náznak vizualizace dokumentu. Pouze s tím omezením, že nezobrazujeme příslušná data dokumentu. Jak výsledný strom vypadá ilustruje následující obrázek 4.1.



Obr. 4.1 Příklad stromové struktury prezentované vytvářenou aplikací

4.2.2 Tabulkový přístup

Tabulky poskytují širokou škálu možností, jakým stylem data XML dokumentů zobrazit.

Ve vytvářené aplikaci jsou považovány za pracovní plochu. Na ní se odehrává veškerá práce s obsahem XML dokumentů. Jde především o vizualizaci a editaci. Tabulky prezentují vždy jen určitý úsek dokumentu. Podle zobrazeného obsahu, může být tato část dokumentu v aplikaci rozdělena na dva módy. V obou případech se jedná vždy o jeden konkrétní element. První mód dovoluje zobrazit pouze samotný element. Druhý mód je použit pro vizualizaci kolekce potomků tohoto elementu. Princip zobrazení je v u obou případů stejný. Tabulky umožňují rozšířit zobrazení ještě o jednu úroveň zanoření níž. Tím je myšleno zobrazení další potomků již zobrazeného elementu nebo elementů.

Tabulky jsou tvořeny řádky a sloupci. Klíčovým parametrem tabulek pro tuto práci jsou sloupce. V jednotlivých sloupcích se budou zobrazovat informace o každém elementu. Informace jsou skládány ze jména elementu, jeho hodnoty a atributů, které mu náleží. Strukturu a formát tabulky nastiňuje obrázek 4.2.

<root>

#ID	Elementy	Hodnota	Atribut mistnost		../ sub_
1	child_1	Výroba	A304	(3)...>	Klempíři
2	child_2	Vedení	A305	NO	
3	child_3	Vývoj	A306	NO	
4	child_4	Vývoj	A308	NO	

Obr. 4.2 Vizualizace struktury dokumentu z obrázku 4.1 pomocí tabulky (1. část)

Obsahu řádků je tvořen elementy, tak jak jdou v hierarchii XML dokumentu po sobě. Každý řádek představuje jeden element, kterému přiřazujeme jednotlivé hodnoty ve sloupcích.

Důležitými sloupci z obrázku 4.2 jsou první dva. První sloupec, označen *#ID*, obsahuje pořadové čísla elementů, tak jak jsou uloženy ve struktuře dokumentu. Jména elementů jsou uvedena ve druhém sloupci. Ten je označen v tomto případě *Elementy*. Pokud si uživatel vyžádá zobrazení pouze jediného uzlu, bude název sloupce *Element*. Dále následuje obsah uzlů. Obsah je tvořen hodnotou elementu a atributy. Hlavička sloupce u atributů vždy uvádí jméno. Ve sloupci jsou pak příslušné hodnoty.

Poslední sloupec není označen. Ten nese informaci o počtu dalších pod-elementů již zobrazovaných elementů. Případ, kdy element obsahuje další potomky, je uveden číslem, které udává jejich počet. V opačném případě, kdy je element koncový a neobsahuje žádné další elementy, je mu přiřazen řetězec „NO“. Tento sloupec rovněž slouží jako separátor nebo taky oddělovač. Za ním tabulka za určitých okolností může pokračovat dalšími daty.

Tuto část tabulky mohou tvořit již zmiňované sub-elementy. Tedy potomci, již zobrazovaného elementu nebo kolekce elementů. Ty se zobrazují pouze v omezeném množství. Je to především z důvodu orientace v tabulce. Například zobrazovaný element může obsahovat až stovky pod-elementů s atributy. Tabulka by se natáhla a ztratila by tak svoji eleganci, co se týče přehlednosti vizualizace dat. Druhou část tabulky s pod-elementy znázorňuje obrázek 4.3.

	../ Sub element **sub_child_1**	../ Atribut **mistnost**	../ Sub element **sub_child_2**	../ Sub element **sub_child_3**
(3)...>	Klempíři	A304.1	Montéři	Technici
NO				
O				

Obr. 4.3 Vizualizace struktury dokumentu z obrázku 4.1 pomocí tabulky (2. část)

Obrázek je pokračování obrázku 4.2. Jsou zde čtyři celé sloupce. Tři z nich tvoří pod-elementy. Čtvrtý atribut. Princip zobrazení je takový, že se vždy uvede název sub-elementu a za ním, v dalších sloupcích, jeho atributy. Obsah těchto sloupců se přiřazuje jednotlivým elementům pouhým křížením.

Usnadnění by měla přinést možnost, jednotlivé sloupce skrývat a zase zpět zobrazovat.

4.2.3 Záložky a orientace

Jelikož pomocí tabulek nelze zobrazit více části dokumentu najednou, pokusím se tento problém kompenzovat záložkami. Pomocí každé záložky je možné zobrazit jednu tabulku. Pomocí více záložek je možné zobrazit celý dokument. Toto je hlavní důvod jejich využití. Tabulka se při otevření automaticky zobrazuje v záložce. Uživateli by aplikace měla poskytnout i jednoduché zavírání záložek a pohyb mezi nimi.

Vlastností aplikace by mělo být i zobrazení aktuální cesty elementu, jenž je aktuálně prezentován v tabulce. Jedná se v podstatě o vypsání všech elementů, které jsou na cestě mezi kořenovým uzlem a zobrazeným elementem. Je to obdoba adresářové cesty, jenž se využívá k přístupu k jednotlivým souborů nebo složkám na disku. Tato vlastnost by měla opět ulehčit orientaci v dokumentu. Cesta by se měla vypisovat na pohledem dostupném místě, avšak neměla by nějak výrazně zasahovat do zobrazované části dokumentu. Nejvhodnější místo by mělo být ve stavovém řádku ve spodní části okna.

4.3 Manipulace s XML

4.3.1 Klávesové zkratky

Klávesové zkratky jsou užitečným nástrojem především pro usnadnění práce. Jde hlavně o to, že uživatel vůbec nemusí sáhnout na myš. Což krátí čas práce.

Tabulka sama o sobě již klávesnici využívá. Je toho důkazem pohyb pomocí šipek mezi jednotlivými buňkami nebo stisknutím klávesy *Enter* se začne buňka editovat. Nově přidané klávesové zkratky by měli umožnit pohyb v celé struktuře dokumentu, přepínání záložek anebo zjednodušit editaci.

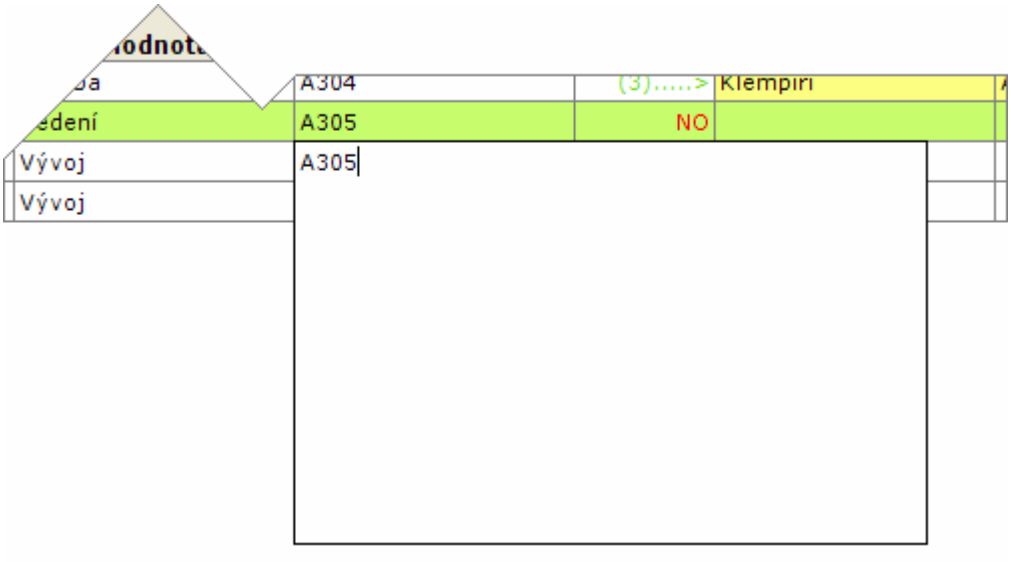
Zanoření o jednu úroveň níž: *Ctrl* + šipka dolů. Tato klávesová zkratka umožňuje posunout se ve struktuře dokumentu o jednu úroveň níž. To znamená, že pokud zvolený element na patřičném řádku obsahuje další potomky a uživatel na něj aplikuje tyto klávesy, měla by se celá tabulka překreslit. Novým obsahem by měli být již potomci zvoleného elementu.

Návrat zpět o jednu úroveň výš: *Ctrl* + šipka nahoru. Pomocí této klávesové zkratky se uživatel může v hierarchii XML dokumentu vracet o jednu úroveň výš. V tomto případě nezáleží na pozici kurzoru. Při aktivaci této kombinace kláves, na libovolném místě v tabulce, je obsah opět překreslen. Následným obsahem se stávají všichni potomci uzlu nadřazeného, pokud se nejedná o samotný kořenový element.

Posun o jednu záložku doleva: *Ctrl* + šipka vlevo. Zde klávesová zkratka umožňuje jednoduché přepínání záložek. Pokud záložka, ve které se zobrazuje určitá část XML dokumentu, není první, může se uživatel pomocí této zkratky přepnout o jednu záložku směrem doleva. Při přepínání je zachovávaná pozice kurzoru v tabulce. Když se tedy uživatel vrátí k obsahu, ze kterého se posunul na jiný, měl by se kurzor v tabulce nacházet na tom samém místě, jako když tabulku opustil.

Posun o jednu záložku doprava: *Ctrl* + šipka vpravo. Jedná se v podstatě o stejný případ jako v předchozí definici. Akorát přepínání záložek je v opačném směru, tedy doprava. Aby se uživatel mohl přepnout na další záložku, tak ta, na které se nachází, nesmí být poslední v pořadí. Opět se zachovávají pozice kurzorů na správných místech.

Editace buňky: *Enter*. Klávesa *Enter* již v tabulce své uplatnění má. Slouží pro editaci buňky, na kterém se nachází kurzor. Ve vytvářené aplikaci zůstane vlastnost editace nad klávesou *Enter* zachována. Změní se však způsob editace. Buňka bude modifikována pomocí kontextového menu. Situaci, jak editace buňky vypadá, znázorňuje následující obrázek 4.4.

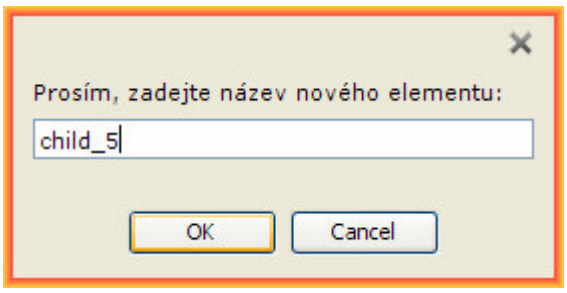


Obr. 4.4 Editace buňky pomocí klávesy ENTER a textového pole

Důvodem použití je fakt, že hodnoty v XML dokumentech můžou být dosti obsáhlé. K jejich zobrazení a editaci obyčejná buňka nemusí ani po roztažení stačit. Pro tento účel se využije libovolná komponenta, která dokáže zobrazit delší řetězce. Tou je například textové pole.

4.3.2 Modifikace

Editací dokumentu se myslí modifikace a mazání již obsažených dat nebo přidávání nových dat. Těmi jsou ve většině případů právě elementy a atributy. Pokud uživatel vytváří nový uzel nebo atribut, mělo by být jeho povinností zadat název nového objektu. U atributu i hodnotu. Tyto situace se řeší pomocí dialogových oken se vstupními položkami. Těmi můžou být opět textové pole. Jedná se v podstatě o klasické vyskakovaní okno. Jak takové okno vypadá, ilustruje obrázek 4.5.



Obr. 4.5 Dialogové okno pro vytvoření nového elementu

5 Realizace aplikace

Předchozí kapitola definovala koncepci a požadavky na vytvářenou aplikaci. Tato kapitola se zabývá realizací aplikace pomocí kódu v Javě.

Realizace práce může být usnadněna použitím vývojového prostředí NetBeans. Při vytváření aplikace je dobré se zhruba držet následující schéma postupu.

Nejprve je třeba rozvrhnout jednotlivé komponenty v okně, se kterými bude uživatel pracovat a pomocí kterých bude XML dokument vizualizován. Druhým krokem je otevření dokumentu a jeho načtení do paměti. Ve fázi, kdy je dokument načtený, je možné jej zpracovat. Zpracováním je myšleno především zobrazení dokumentu stromovou strukturou. Nejrozsáhlejší částí implementace budou jistě uživatelské funkce. Ty jsou ovládány a spouštěny samotným uživatelem. Zde na pořadí implementace funkcí nesejde. Výjimkou je ovšem zobrazení dokumentu tabulkou. Pro většinu funkcí je toto klíčový krok, aby mohli být vůbec aplikovány. Nejdříve tedy musí být implementována tato vlastnost. Posléze je možné přistoupit k realizaci ostatních funkcí, pracujících s dokumentem. Posledním, větším, krokem je zobrazení XML Schema pro daný dokument. Tento postup je na většině předchozí implementaci nezávislý.

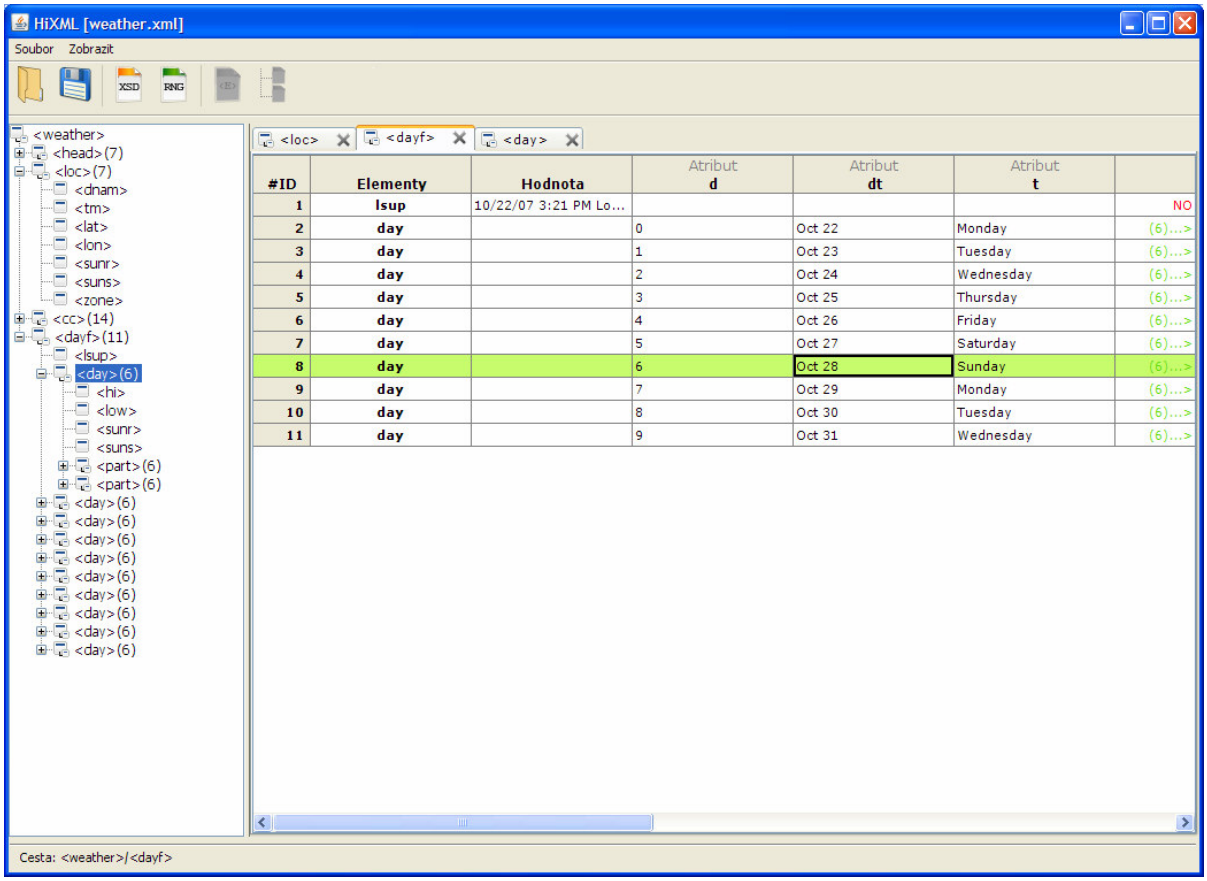
Nejdůležitějšími a nejčastějšími problémy při implementaci aplikace se zabývají následující podkapitoly. Vždy je uveden samotný problém a za ním řešení nebo prostředky, které jsou pro tento účel využity.

5.1 Rozvržení komponent

Cílem je vytvořit a co nejvhodněji rozmístit uživatelské komponenty. Pro základní komponenty, které nejsou dynamicky generovány a jsou viditelné hned při spuštění aplikace moc řádku kódu nenapíšu. Snad jenom pro nastavení určitých parametrů. V tomto směru opět vychází vstříc vývojové prostředí NetBeans, ve kterém je aplikace implementována. Umožňuje snadné rozmístění komponent v okně. Zároveň s umístěním každé komponenty NetBeans generují příslušný kód. Ten komponentu blíže specifikuje a udává její pozici v rozvržení.

Hlavní okno musí obsahovat pět prvků, které jsou nezbytné pro vizualizaci. První je menu v horní liště okna. Jedná se o klasické menu, tvořené komponentou *JMenuBar*. Jednotlivé položky menu jsou pak tvořeny pomocí objektů *JMenu*, *JMenuItem* a nebo *JCheckBoxMenuItem*. Menu by mělo obsahovat položky pro nastavení aplikace a spouštění příslušných funkcí, jako je například otevření souboru, uložení souboru nebo vytvoření XML Schema. Pod menu by se měl nacházet panel nástrojů pro takzvanou rychlou volbu funkce. K tomuto účelu se využívá prvek z knihoven Swingu zvaný *JToolBox*. Ten by měl obsahovat pouze tlačítka pro nejpoužívanější volby. Těmi jsou opět otevření a uložení souboru nebo tlačítka pro zobrazení elementů. Dostávám se k vytvoření dvou

nejdůležitějších komponent, pomocí kterých jsou XML dokumenty vizualizovány. První z nich je umístěna na levou stranu okna. Jedná se o *JTree*. To je komponenta, která umožňuje zobrazit stromovou strukturu dokumentu. Druhou komponentou je *JTabbedPane*. Ta zabírá největší část plochy a používá se pro vytváření záložek. Posledním prvkem, který bude stále zobrazen, je stavový řádek ve spodní části okna. V něm budou generovány především informační texty a aktuální poloha ve stromové struktuře dokumentu. Výsledné rozmístění komponent je prezentováno obrázkem 5.1.



Obr. 5.1 Rozložení komponent pro vytvářenou aplikaci. Zároveň je vizualizován soubor weather.xml

5.2 Načtení dat

Jelikož JDOM neobsahuje vlastní parser, využívá se k načtení dat jiný nástroj, který je k dispozici a který tuto funkci obsahuje. Naskytuje se možnost použít klasický DOM z knihoven Javy. Název knihovny a jak ji připojit demonstruje následující algoritmus 5.1.

Algoritmus 5.1 Připojení knihovny s DOM parserem v Javě

```
import javax.xml.parsers.*;
```

Před přístupem k obsahu XML je nejdříve třeba vytvořit v paměti dokument pomocí klasického DOMu. Jak takovýto postup v jazyce Java vypadá, demonstruje algoritmus 5.2 níže.

Algoritmus 5.2 Uložení XML dokumentu do paměti počítače DOM parserem

```
DocumentBuilderFactory doc_bf = DocumentBuilderFactory.newInstance();
doc_bf.setValidating(false);
doc_bf.setIgnoringElementContentWhitespace(true);

org.w3c.dom.DocumentBuilder dom_builder = doc_bf.newDocumentBuilder();

Document dom_document = dom_builder.parse(xml_file);
```

V prvním řádku předchozího kódu se nejprve vytváří nová třída *DocumentBuilderFactory*. Ta umožňuje aplikaci získat odkaz na novou instanci rozhraní, které převádí XML dokumenty na stromovou prezentaci [11]. V dalších dvou řádcích je provedeno nastavení parametrů vytvořené třídy. *DocumentBuilder* definuje API, které umožňuje aplikaci převádět XML dokumenty na stromovou strukturu. Instanci této třídy je získána voláním metody `newDocumentBuilder()` již vytvořené třídy *DocumentBuilderFactory*. V posledním kroku je vygenerován už samotný dokument definovaný balíkem `org.w3c.dom`, vytvořením stromové struktury v paměti metodou `parse()`.

V této fázi je možné k dokumentu přistupovat, ale pouze přes rozhraní DOM. Aby bylo možné pracovat s dokumentem v rozhraní JDOM, musí být dokument transformován. To jednoduše provádí následující kód v algoritmu 5.3.

Algoritmus 5.3 Transformace DOM dokumentu na JDOM dokument

```
jdom_document = new DOMBuilder().build(dom_document);
```

Algoritmus 5.4 prezentuje kód v Javě, pomocí kterého je XML dokument procházen. Tato funkce se provádí v aplikaci před zobrazením hned několikrát, aby byli získány požadované data. Funkce se například používá pro naplnění stromu nebo naplnění tabulky.

Algoritmus 5.4 Funkce printSubChildren() pro zpracování potomků elementu v parametru

```
function printSubChildren(Element element) {
  java.util.Iterator iterator = element.getChildren().iterator();

  while (iterator.hasNext()) {
    Element child = (Element) iterator.next();
    //Získání atributů
    getChildAttributes(child);
    //Další zanoření – získání potomků aktuálního elementu child
    printSubChildren(child);
  }
}
```

Pro průchod dokumentu je nejvhodnějším řešením rekurze. Interpretovaný algoritmus 5.4 představuje funkci `printSubChildren()`. Předává se jí jediný parametr a to element, pro který je potřeba zpracovat jeho potomky (pod-elementy). V těle funkce je obsažen cyklus, který postupně tyto uzly projde. Pro každý takovýto pod-element se funkce znovu v rekurzi volá. Předávaným parametrem funkce `printSubChildren()` je nyní aktuálně zpracovávaný pod-element. Pro něj se budou opět procházet jeho potomci. V cyklu algoritmu 5.4 je rovněž volána funkce `getChildAttributes()`. Jejím účelem je získat atributy u zpracovávaného potomku.

5.3 Tabulka

Pro vizualizaci obsahu XML dokumentů, je třeba vytvořit tabulku, naplnit ji daty a pak zobrazit v záložce. Tabulku v Javě tvoří komponenta *JTable*. Naplnit ji daty není obtížné. Jedná se pouze o získání příslušné hodnoty z XML dokumentu a její vložení na určitý řádek a do příslušného sloupce. Hlavička tabulky se vytváří obdobně, akorát s tím rozdílem, že se adresuje jako jednorozměrné pole. Tento postup v aplikaci implementuje třída *NewTabPlacement*, kde se pro tento účel využívají funkce:

- `fillTableBody();`
- `fillTableHeader();`
- `fillTableEdit();`

O poslední funkci jsem se nezmínil. Ta určuje, které sloupce je možno editovat a které ne.

Obtížněji se realizuje formátování tabulky. Pod tímto pojmem se skrývá nastavení vzhledu a dalších vlastností. Jelikož tabulka obsahuje rozdílná data, je dobré je nějakým způsobem od sebe odlišit. Formátováním jde nastavit typ, velikost, tloušťka a barva písma. Dále barva pozadí, orámování nebo do buňky vložit jinou komponentu či obrázek. Hlavička i tělo tabulky se rendruje stejným způsobem, ale pro každou tuto část je v aplikaci vytvořená zvláštní třída. Sloupce a buňky se formátují podle jejich obsahu a pozice. Tato vlastnost umožňuje nastavit i vzhled kurzoru, pomocí kterého se uživatel po tabulce pohybuje.

Aby byli jednotlivé funkce nad tabulkou použitelné, je třeba implementovat příslušné události. Těmi jsou myšleny klávesové zkratky, kontextové menu, editace a pohyb po tabulce. Zmíním se pouze o řešení editace pomocí kontextového menu a textového pole. Jedná se o aktivaci klávesové zkratky *Enter* nad tabulkou. V tomto případě je do kontextového menu umístěna komponenta *JTextArea*. Pokud buňka má již libovolný obsah, musí být přenesen i do textového pole. Je třeba ještě nastavit kurzor na konec řetězce. Menu je vždy zobrazováno pod buňkou která je editována, pokud ho systém neusadí jinak vzhledem k hranici obrazovky. Dále musí mít textové pole implementovanou vlastnost, která dovede vysázet znak *Enter*, jelikož pomocí této klávesy se kontextové okno zavírá. Je třeba zvolit jinou klávesu nebo jejich kombinaci. Nabízí se možnost použít zkratku *Ctrl* a *ENTER*. S ní pracují již i jiné aplikace, tudíž uživatelům nemusí být tak neznámá.

5.4 Vytvoření XML Schema

V kapitole 2.3 bylo řečeno, jak Trang použít. Tato kapitola se zaměřuje na práci s ním ve vytvářené aplikaci. Vytvoření XML Schema pro soubor *weather.xml* ilustruje následující příklad 5.1. Výsledné schéma se ukládá do souboru *weather.xsd*. Kód je specifikován pro platformu Microsoft Windows.

Příklad 5.1 Příkaz pro vytvoření XSD dokumentu pro XML dokument pomocí nástroje Trang

```
java –jar "cesta\trang.jar" "cesta\weather.xml" "cesta\weather.xsd"
```

Takovýto řetězec, z příkladu 5.1, musí být vytvořen v aplikaci, aby se transformace mohla aplikovat. Přičemž za parametry *cesta* se dosazují plnohodnotné adresářové cesty. Následně se vytvoří nový proces, kterému je tento příkaz předán a který jej bude zpracovávat. Řešení prezentuje algoritmus 5.5.

Algoritmus 5.5 Spuštění příkazu z příkladu 5.1 pomocí kódu v Javě

```
Process process = Runtime.getRuntime().exec(command);
process.waitFor();
```

V kódu algoritmu 5.5 je vytvořen proces *process*, kterému je předáván příkaz z příkladu 5.1 pro zpracování XML dokumentu, který je reprezentován proměnnou *command*. Druhý řádek kódu způsobí čekání na dokončení operace.

Nyní je XSD soubor, definovaný posledním parametrem příkazu, vytvořený a uložený na disku. Ten je možné otevřít nástrojem DOM. Ke zpracování načteného dokumentu je opět možné přistoupit pomocí JDOMu. Obsah dokumentu se následně zobrazuje do příslušného textového pole.

5.5 Dialogová okna

Pro zobrazení nebo získání některých informací využívá aplikace dialogová okna. Pokud jsou zachovány klasické vlastnosti okna, zobrazí se kolem něj systémový rámeček. V aplikaci jsem se rozhodl tyto rámečky změnit. Jednalo se především o vzhled. Abych vytvořil svůj vlastní rámeček, bylo třeba zrušit stávající. Ale po jeho odebrání jsem se setkal s problémy. Dialogové okna ztratily svoje běžné funkce, které umožňovaly právě klasické rámečky. Jednalo se hlavně o přesouvání, změnu velikosti, fokus a zavírání okna. Chybějící funkce musely být doimplementovány. Obsah dialogového okna je postaven na komponentě *JPanel*. Ta umožňuje vytvořit nový rámeček, který je tvořen nastavením okrajů.

Složitějšími implementovanými funkcemi jsou pohyb a změna velikosti okna. Pohyb je umožněn u všech dialogových oken. Změna pouze u těch, které to vyžadují. V tomto případě se jedná především o zobrazení XML Schema. Ztráta nebo zachycení fokusu je řešeno pouze změnou barvy orámování. Jak výsledné dialogové okno s novým rámečkem vypadá, je vidět na následujícím obrázku 5.2.

Obr. 5.2 Rámeček dialogových oken: (a) Okno je aktivní a kurzor myši naznačuje možnou změnu velikosti, (b) okno je neaktivní. V pravém horním rohu se nachází tlačítko pro zavření okna

6 Závěr

Následující odstavce shrnují dosažené výsledky, vlastnosti aplikace, její funkčnost a možná rozšíření.

Vytvářená aplikace byla od samého počátku směřována k co nejjednoduššímu a nejpřehlednějšímu zobrazení XML dat. Existuje již docela početná skupina aplikací, které tuto možnost podporují. Oproti nim jsem se snažil přinést něco nového do tohoto oboru.

Výsledkem je aplikace, která zobrazuje XML data v tabulkách. Tím je splněn první požadovaný cíl zadání. Druhým cílem je dovednost vytvářet pro XML dokumenty formální definice. Tento bod je do programu zanesen také.

Tabulky jsou nejvhodnější technikou zobrazení strukturovaných dat. Důkazem jsou třeba databázové tabulky, které můžou čerpat data právě z XML dokumentů. Běžnou technikou, která se často používá, je stromové zobrazení. To přináší přehlednou vizualizaci struktury dokumentu. Nehodí se ovšem pro přímé zobrazení obsahu dokumentu. Tím jsou myšleny data. Tyto dva prvky jsem se v aplikaci pokusil skloubit, abych dosáhl co nejlepšího výsledku. Lze tedy říci, že stromová struktura lépe prezentuje hierarchii a tabulky naopak obsah XML dokumentu.

Aby se jednalo o interaktivní vizualizaci, byla nad XML a XSD dokumentem implementována spousta užitečných funkcí. Z této množiny je možné vyzdvihnout klávesové zkratky, modifikaci a ukládání změněného nebo vytvořeného dokumentu. První zmiňovaná vlastnost není zcela intuitivní a často se stává, že uživatel nemá v používaných aplikacích o klávesových zkratkách potuchy. Vyjma zkratek, jako jsou například *Ctrl+C* nebo *Ctrl+V*. Zde jsou považovány za příjemné zlepšení, které by mohlo ušetřit čas, tedy i peníze. Pomocí zkratek se uživatel může ve struktuře pohybovat daleko rychleji než pomocí běžných tlačítek nebo myši. To samé platí i pro zmíněnou editaci.

Další příjemnou funkcí je číslování jednotlivých elementů a kurzor v tabulce. Obě položky slouží především pro orientaci. Často se může stát, že XML dokument obsahuje spousty elementů stejného názvu. Pomocí číslování jdou rozlišit. Tato vlastnost se hodí například i při rozsáhlejších dokumentech, které obsahují stovky uzlů ač různých názvů.

Aplikace dovede rovněž zobrazovat smíšený obsah elementů. Dalšími vlastnostmi, kterými se aplikace snaží uživateli přinést co nejlepší zjednodušení je možnost skrývání jednotlivých sloupců, podbarvování patřičných buněk při zobrazení sub-elementů nebo také poskytování informací o počtu sub-elementů u příslušných elementů.

Jedná se o použitelnou a funkční aplikaci, která by mohla být nasazena do praxe pro práci s XML dokumenty. Aplikace je určena především pro pokročilejší a profesionální uživatele v oblasti XML, kteří chtějí s těmito dokumenty pracovat co nejrychleji a nejefektněji. Dalo by se uvažovat i o možném rozšíření a implementaci spousty dalších užitečných funkcí popsaných níže.

6.1 Možná rozšíření

Rozšíření, o které by mohla být vytvořená aplikace obohacena, je mnoho. Záleželo by na požadavcích uživatelů, kteří s aplikací budou pracovat. Může se jednat o libovolnou změnu v přístupu k zobrazení XML dokumentu nebo také pro práci s ním.

Nyní uvedu návrhy na rozšíření, které aplikace postrádá a které by mohli rozšířit možnosti vizualizace a zpracování XML dokumentů.

Rozhodně by aplikaci mělo být umožněno vytvářet nové XML dokumenty. Při psaní této práce jsem hledal libovolný nástroj, pomocí kterého bych si mohl vytvořit vzorový XML dokument. Tato vlastnost může být dosti užitečná i používaná. Implementace by nemusela být ani náročná, jelikož spoustu funkcí, které slouží pro libovolnou modifikaci dokumentu, jsou již v programu integrovány

Dalším vhodným rozšířením, pro některé uživatele, by se mohla stát možnost vytvářet kromě XML Schema také definici v jazyce DTD. Tento jazyk je v dnešní době stále používaný a někteří uživatelé mohou pracovat právě s ním.

Funkci, kterou aplikace rovněž postrádá je vyhledávání v celé struktuře dokumentu. Tato myšlenka by mohla být dosti užitečná a leckdy by mohla usnadnit i práci s dokumentem. Vyhledávání by mohlo fungovat jak pro jména elementů a atributů, tak pro hodnoty těchto dvou prvků. Výsledky hledání by se zobrazovaly opět pomocí tabulky. Jednotlivé řádky by obsahovaly všechny elementy, u kterých byl hledaný řetězec nalezen.

Posledním zavedeným zlepšením by se mohlo stát nastavení aplikace. Jelikož každý uživatel má trošku rozdílný názor na vzhled a funkci aplikace, mělo by být možno některé vlastnosti zobrazení měnit nebo dokonce vypínat. Tím je například myšlena barva, velikost a celkový formát písma, barva kurzoru v tabulce nebo také počet zobrazených sub-elementů pro elementy. Rovněž by nebylo špatné poslední zmiňovanou možnost zakázat, stejně jako podbarvování určitých buněk tabulky.

Literatura

[1] BRADLEY, N.: *XML - kompletní průvodce*, kapitola Logická struktura (DTD). Harlow: Addison Wesley, 2000, ISBN 80-7169-949-7, s. 54–75.

[2] HEROUT, P.: *Java a XML*. České Budějovice: KOOP, první vydání, 2007, ISBN 978-80-7232-307-4, 313 s.

[3] KOSEK, J.: *XML schémata*. [online], [rev. 2005-08-18], [cit. 2008-04-15]. URL <http://www.kosek.cz/xml/schema/index.html>

[4] W3C: *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. [online], Verze REC-xml-20060816 (2006), poslední aktualizace 29.09.2006, [cit. 2008-04-15]. URL <http://www.w3.org/TR/2006/REC-xml-20060816/>

[5] W3C: *XML Schema Part 0: Primer Second Edition*. [online], Verze REC-xmlschema-0-20041028 (2004), poslední aktualizace 28.10.2004, [cit. 2008-04-16]. URL <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/>

[6] HERNYCH, R.: *Vizualizace XML*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2007, vedoucí bakalářské práce Chmelař Petr, Ing.

[7] KUBÍČEK, D.: *Interaktivní vizualizace XML*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2007, vedoucí bakalářské práce Chmelař Petr, Ing.

[8] Meteorologists at The Weather Channel®. 2007. URL <weather.com>

[9] Wikipedia: *Relační model*. [online], poslední aktualizace 09.01.2008, [cit. 2008-04-20]. URL <http://cs.wikipedia.org/wiki/Relační_model>

[10] KOSEK, J.: *XML API*. [online], [cit. 2008-04-20]. URL < http://www.kosek.cz/xml/api/>

[11] KADLEC, V.: *JAXP – Java API for XML Processing*. [online], [cit. 2008-04-27]. URL <http://nb.vse.cz/~zelenyj/it380/eseje/xkadv02/JAXP.htm>

[12] Wikipedia: *Applet*. [online], poslední aktualizace 21.04.2008, [cit. 2008-04-29]. URL <http://cs.wikipedia.org/wiki/Aplet>

[13] HEROUT, P.: *Učebnice jazyka Java*. České Budějovice: KOOP, druhé vydání, 2006, ISBN 80-7232-115-3, 349 s.

[14] Wikipedia: *Java*. [online], poslední aktualizace 31.03.2008, [cit. 2008-04-29]. URL <http://cs.wikipedia.org/wiki/Java>

[15] Wikipedia: *Virtuální stroj*. [online], poslední aktualizace 04.04.2008, [cit. 2008-04-29]. URL < http://cs.wikipedia.org/wiki/Virtuální_stroj>

[16] KOSEK, J.: *XML pro každého – podrobný průvodce*. Praha: Grada Publishing, první vydání, 2000, ISBN 80-7169-860-1, 164 s.

[17] CLARK, J.: *Trang*. [online], © 2002 – 2003, [cit. 2008-05-01]. URL <http://www.thaiopensource.com/relaxng/trang.html>

[18] HARROLD, E. R.; MEANS W. S.: *XML in a Nutshell*. O'Reilly & Associates, druhé vydání, 2002, ISBN 0596002920, 640 s.

[19] CHMELAŘ, P.; HERNYCH, R.; KUBÍČEK, D.: *Interactive Visualization of Data-Oriented XML Documents*. Brno: Vysoké učení technické v Brně, Fakulta informačních technologií, 2007, 4 s.

[20] JELÍNEK, J.; SLAVÍK, P.: *XML visualization using tree rewriting*. In Proceedings of the 20th spring conference on Computer graphics, ročník 20, editace A. PASKO, Budmerice, Slovensko: ACM, Duben 2004, ISBN 1-58113-967-5, s. 65–72.

7 Seznam příloh

7.1 Příloha 1: CD

Přiložené CD k bakalářské práci obsahuje:

- Zdrojové kódy
- Kompletní text
- Testovací data
- Komprimovaný program (*jar* soubor)

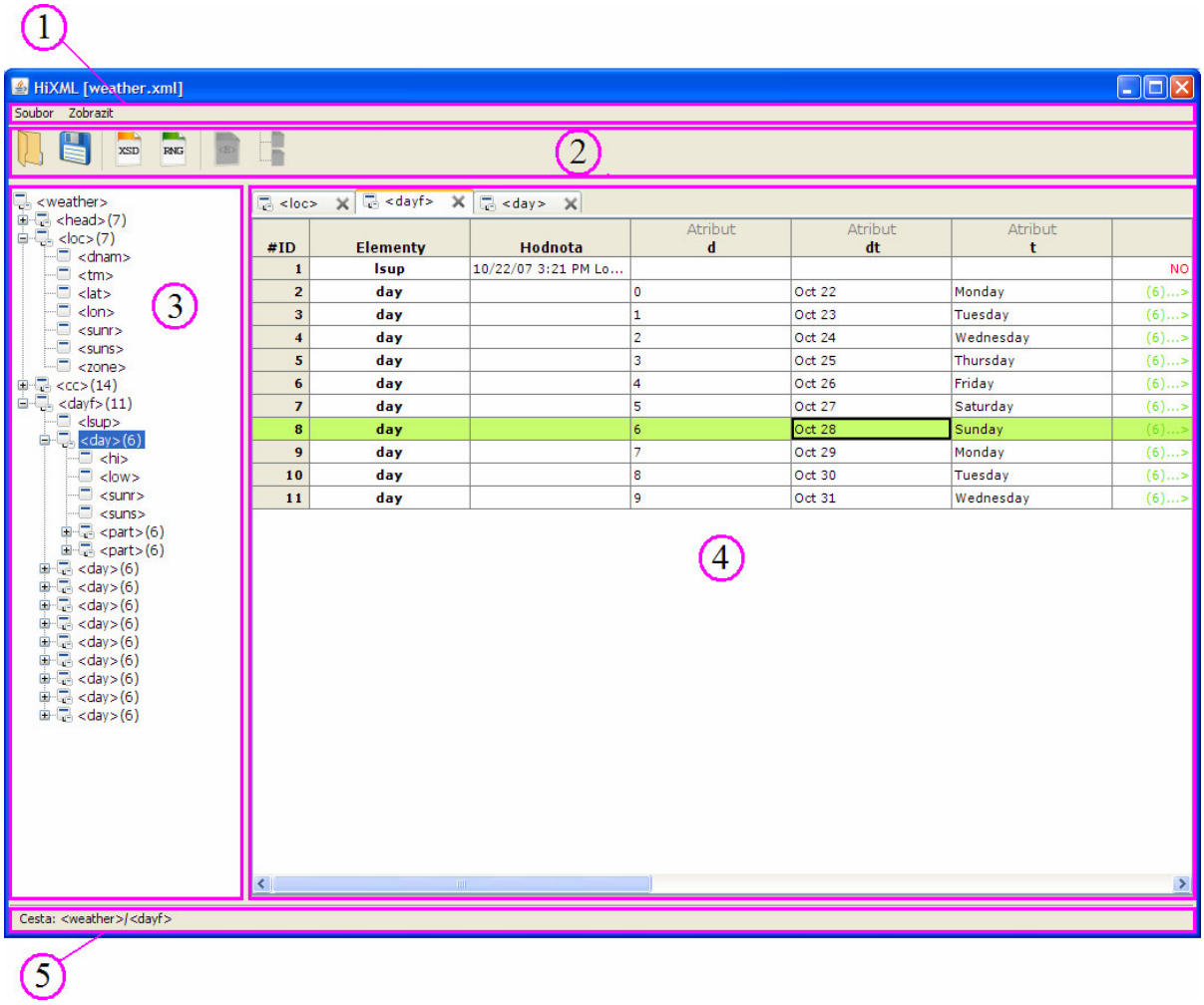
7.2 Příloha 2: Manuál programu HiXML

Aplikace HiXML je určena pro vizualizaci XML dokumentů. Jejím účelem je jednoduše a přehledně prezentovat obsah těchto souborů a vytvářet pro ně formální definice v jazyce XSD nebo Relax NG. Aplikace je zaměřena na datově orientované XML dokumenty. Včetně strukturovaného obsahu umožňuje zobrazit i smíšený obsah. Dokumenty jsou vizualizovány pomocí stromové struktury a tabulek.

Návodem na použití aplikace HiXML se zabývají následující kapitoly. Ty popisují nejběžnější funkce, které jsou systematicky za sebou řazeny.

7.2.1 Popis spuštěné aplikace

Aby mohla být aplikace spuštěna, je potřeba mít nainstalované běhové prostředí Java. Aplikace se spouští přes soubor *HiXML.jar*. Otevřenou aplikaci prezentuje obrázek 7.1.



Obr. 7.1 Spuštěná aplikace HiXML

Strukturu aplikace popisují následující body 1 až 5 synchronizované z body v obrázku 7.1.

1 - Menu aplikace. Obsahuje nejdůležitější funkce a je tvořeno dvěma položkami, *Soubor* a *Zobrazit*. *Soubor* obsahuje tlačítka pro otevření nebo uložení XML dokumentu a pro ukončení aplikace. Položka *Zobrazit* je tvořena tlačítky pro zobrazení XML definic XML dokumentu a tlačítky pro nastavení některých vlastností aplikace.

2 – Panel Nástrojů. Je tvořen tlačítky pro takzvanou rychlou volbu. Jednotlivými tlačítky zleva jsou:

- Otevření souboru (přístupné vždy)
- Uložení souboru (přístupné pouze pro otevřený a modifikovaný XML dokument)
- Zobrazení XSD (přístupné pouze pro otevřený XML dokument)

- Zobrazení Relax NG (přístupné pouze pro otevřený XML dokument)
- Zobrazení elementu (přístupné pouze pro práci se stromovou prezentací)
- Zobrazení kolekce elementů (přístupné pouze pro práci se stromovou prezentací)

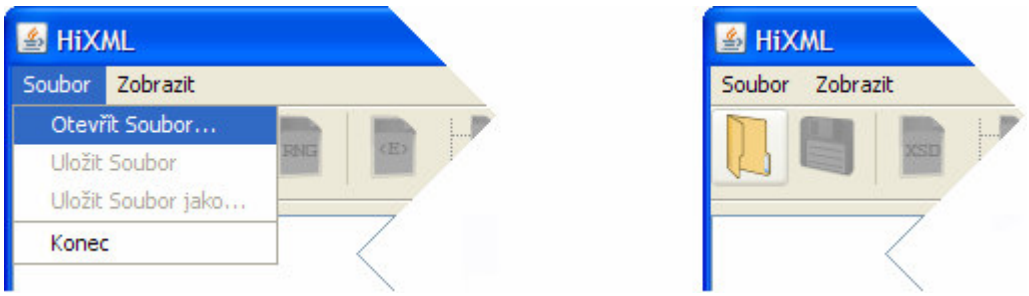
3 – Stromová prezentace. Ta modeluje strukturu XML dokumentu a umožňuje její vizualizaci. Podle tohoto modelu si uživatel vybírá, které úseky XML dokumentu chce zobrazit.

4 – Tabulky. Vizualizují jednotlivé části XML dokumentu a jsou prezentovány záložkami. Jedná se o pracovní plochu aplikace. Pomocí nich je možný přístup k textovému obsahu a atributům XML dokumentu. Dovolují rovněž vizualizovaný XML dokument editovat.

5 – Stavový řádek. Zobrazuje především informaci o cestě k zobrazovanému elementu. Využívá se také pro poskytnutí informačních nebo varovných sdělení uživateli.

7.2.2 Otevření XML dokumentu

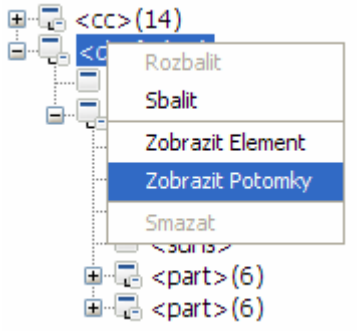
Otevřít XML dokument pro vizualizaci lze dvěma způsoby. Přes klasické menu nebo panel nástrojů v horní části okna aplikace. Obrázek 7.2 znázorňuje tyto dva způsoby.



Obr. 7.2 Otevření XML dokumentu v aplikaci HiXML: (a) přes menu a (b) panel nástrojů

7.2.3 Stromová struktura

Je automaticky vygenerována pro vizualizovaný XML dokument po jeho načtení. Pomocí ní je možné vizualizovat obsah získaného XML dokumentu. Část stromové struktury z aplikace HiXML představuje obrázek 7.3.



Obr. 7.3 Část stromové struktury z aplikace HiXML, nad níž je otevřené kontextové menu

Na uživateli je, aby si pomocí pravého tlačítka myši a kontextového menu z této struktury vybral uzel, který chce vizualizovat. Kontextové menu nad uzlem stromové struktury obsahuje tlačítka pro rozbalení, zabalení, zobrazení nebo smazání uzlu nebo pro zobrazení jeho potomků.

7.2.4 Vizualizace tabulkou

Obsah XML dokumentů je vizualizován tabulkou. Každá tabulka je prezentována záložnou. Řádky v ní jsou tvořeny elementy a sloupce nesou informace o těchto elementech. Těmi jsou atributy nebo textové hodnoty. Vizualizaci obsahu XML dokumentů tabulkou demonstruje obrázek 7.4.

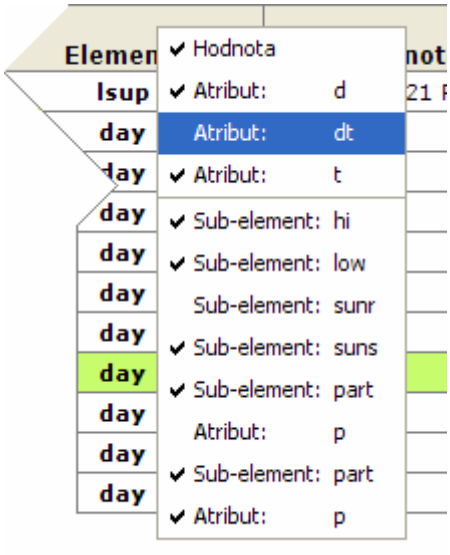
<loc> <dayf> <day>

#ID	Elementy	Hodnota	Atribut d	Atribut dt	Atribut t	
1	lsup	10/22/07 3:21 PM Lo...				NO
2	day		0	Oct 22	Monday	(6)...>
3	day		1	Oct 23	Tuesday	(6)...>
4	day		2	Oct 24	Wednesday	(6)...>
5	day		3	Oct 25	Thursday	(6)...>
6	day		4	Oct 26	Friday	(6)...>
7	day		5	Oct 27	Saturday	(6)...>
8	day		6	Oct 28	Sunday	(6)...>
9	day		7	Oct 29	Monday	(6)...>
10	day		8	Oct 30	Tuesday	(6)...>
11	day		9	Oct 31	Wednesday	(6)...>

Obr. 7.4 Vizualizace kolekce elementů uzlu <dayf> tabulkou

7.2.5 Skrývání sloupců tabulky

Jednotlivé sloupce tabulky je možné skrývat a to pomocí kontextového menu, které může uživatel aktivovat stisknutím pravé myši nad hlavičkou tabulky. Toto menu demonstruje obrázek 7.5.

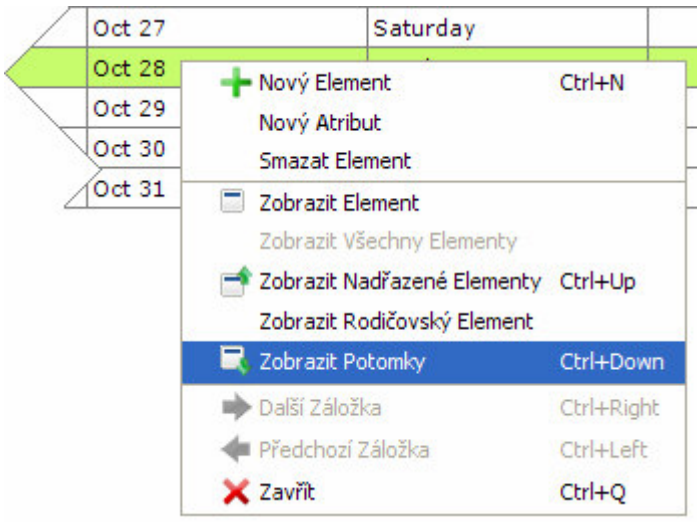


Obr. 7.5 Kontextové menu v aplikaci HiXML pro skrývání sloupců nad hlavičkou tabulky

Jednotlivé položky menu jsou tvořeny tlačítky, které je možné zatrhávat nebo odznačovat. Pokud je tlačítko zatrhnuto, je sloupec zobrazen. V opačném případě je sloupec skryt.

7.2.6 Kontextové menu tabulky

Kontextové menu tabulky je možné zobrazit pomocí pravého tlačítka myši. To musí být stisknuto nad tělem tabulky. Menu slouží pro editaci nebo pohyb ve struktuře dokumentu a mezi záložkami. Představuje ho obrázek 7.6.



Obr. 7.6 Kontextové menu v aplikaci HiXML nad tělem tabulky

Menu umožňuje přidávání nového elementu nebo atributu do tabulky a smazání stávajícího. Dále je pomocí něj možné zobrazit vybraný element z kolekce elementů nebo naopak zobrazit ostatní elementy, pokud je zobrazován jen jeden uzel. Dalšími funkcemi jsou zobrazení nadřazených elementů, zobrazení rodičovského elementu a zobrazení potomků vybraného elementu. Poslední tři tlačítka umožňují práci se záložkami. Konkrétně přepínání mezi záložkami a zavírání záložky.

7.2.7 Uložení modifikovaného XML dokumentu

Postup je obdobný jako pro otevření XML dokumentu z kapitoly 7.2.2. Dokument lze uložit dvěma způsoby. Přes hlavní menu aplikace nebo před panel nástrojů. V menu jsou pro tento účel určena dvě tlačítka pojmenovaná *Uložit Soubor* a *Uložit Soubor Jako*. První z nich se využívá pro uložení změněného XML dokumentu do původního dokumentu. Druhé umožňuje vytvořit zcela nový soubor a vybrat jeho umístění na disku počítače. Tlačítko na panelu nástrojů plní tutéž úlohu, jako tlačítko *Uložit soubor* v menu.