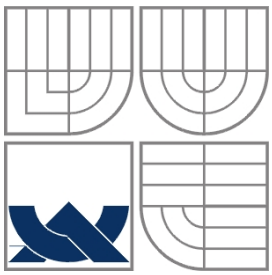
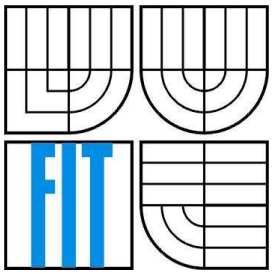


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VIZUALIZACE XML

XML VISUALIZATION

BAKALÁŘSKÁ PRÁCE

BACHELOR´S THESIS

AUTOR PRÁCE — RADIM HERNYCH
AUTHOR

VEDOUCÍ PRÁCE — ING. PETR CHMELAŘ
SUPERVISOR

BRNO 2007

Zadání bakalářské práce/4297/2006/xhemy00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Hernych Radim**
Obor: Informační technologie
Téma: **Vizualizace XML**
Kategorie: Počítačová grafika

Pokyny:

1. Seznamte se s problematikou XML a jeho transformacemi.
2. Identifikujte a pokuste se vyřešit problémy zobrazení obecného XML dokumentu včetně jejich definice.
3. Vytvořte grafický nástroj pro interaktivní zobrazení dokumentů XML.
4. Zhodnoťte vlastnosti a případné vylepšení nástroje.

Literatura:

- Bradley, N. *XML - kompletní průvodce.* Grada Publishing, Praha 2000, 536 stran. ISBN 80-7169-949-7.
- 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/.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2.

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 paměťovém médiu (disketa, CD-ROM), 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 2006
Datum odevzdání: 15. května 2007

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

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: **Radim Hernych**
Id studenta: 84377
Bytem: U Splavu 7/2874, 690 02 Břeclav
Narozen: 12. 02. 1985, Valtice
(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: 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 formě 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 pořizování 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

Cílem této bakalářské práce bylo nastudovat problematiku XML a jeho transformací, pokusit se o identifikaci problémů zobrazení obecného XML dokumentu a následně vytvořit grafický nástroj, který by umožnil interaktivní zobrazení a editaci XML dokumentu.

Tento dokument se zabývá problematikou vizualizace obecného XML. Prezentuje vizualizační techniku pro zobrazení obecných XML dokumentů bez použití XSL nebo CSS.

Výsledný program byl napsán v jazyce Java s využitím knihoven Jaxen 1.1 a dom4j.1.6.1.

Klíčová slova

Vizualizace XML, XML dokument, interaktivní zobrazení XML dokumentu, Java, DOM

Abstract

The goal of this bachelor's thesis was to study XML and its transformations, identify possible problems of displaying general XML document and to create graphical tool that provides interactive visualization and editing of XML document.

This text deals with problems of XML visualization. It presents a visualization technique for representing general XML documents without using XSL or CSS.

Final program was written in Java programming language using Jaxen1.1 and dom4j.1.6.1 libraries.

Keywords

XML Visualization, XML document, interactive XML document representation, Java, DOM

Citace

Radim Hernych: Vizualizace XML, bakalářská práce, Brno, FIT VUT v Brně, 2007

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.

……
Radim Hernych
1. května 2007

Poděkování

Chtěl bych poděkovat vedoucímu své bakalářské práce Ing. Petru Chmelařovi za jeho cenné rady a odbornou pomoc při řešení práce.

Obsah

Obsah ... 1
1 Úvod ... 3
1.1 Struktura práce ... 3
2 Teorie ... 4
2.1 Značkovací jazyk XML ... 4
2.1.1 Historie XML ... 4
2.1.2 Charakteristika XML ... 5
2.1.3 Struktura dokumentu ... 5
2.1.4 Zobrazení a transformace ... 6
2.1.5 Navigace v XML dokumentu ... 7
2.1.6 Metadata ... 8
2.1.7 Aplikace XML ... 9
2.2 Programovací jazyk Java ... 10
2.2.1 Historie Javy ... 10
2.2.2 Charakteristika Javy ... 11
2.2.3 Struktura programu ... 12
2.2.4 Tvorba grafického uživatelského rozhraní ... 12
2.2.5 Využití Javy ... 13
2.2.6 Zpracování XML v Javě ... 13
2.3 Vývojové prostředí NetBeans ... 15
2.3.1 Historie NetBeans ... 15
2.3.2 Charakteristika NetBeans ... 16
3 Koncepce a návrh programu ... 17
3.1 Koncepce programu ... 17
3.2 Návrh zobrazení XML dokumentu ... 17
3.2.1 Stromové zobrazení struktury dokumentu ... 18
3.2.2 Zobrazení obsahu elementu ... 18
3.2.3 Zobrazení seznamu ... 19
3.2.4 Nástroj pro usnadnění orientace v programu ... 20
4 Vlastní implementace ... 21
4.1 Volba rozhraní pro reprezentaci XML ... 21
4.2 Zobrazení obsahu elementu ... 21
4.3 Zobrazení spojnice oken ... 22
4.4 Strom struktury dokumentu ... 22

4.5 Podpora českého jazyka 22
4.6 Nástroj desktopView 22
5 Závěr 24
5.1 Zhodnocení výsledků 24
5.2 Možnosti dalšího vývoje 26
5.3 Vlastní zhodnocení 27
6 Literatura 28
7 Přílohy 29
7.1 Příloha 1. Uživatelská příručka programu VisualXML 29
7.2 Příloha 2: CD/DVD 36
7.3 Příloha 3: Diagram tříd 37

1 Úvod

Informace a jejich efektivní zpracování mají v dnešní době strategický význam. Pro firmy je velmi důležitá výměna informací s okolím (např. získání ceníku od dodavatele, odeslání objednávky nebo faktury apod.). Dříve se pro výměnu informací používaly proprietární formáty, které ale měly řadu nevýhod. Protože byly tyto formáty uzavřené, dovedla s nimi pracovat jen úzká skupina aplikací. Výměna dat mezi různými informačními systémy tak byla nákladná a ne zcela elegantní záležitost.

To se však s nástupem technologie XML změnilo. Výměna informací v tomto otevřeném textovém formátu má řadu výhod. XML umožňuje výměnu dokumentů mezi různými platformami, různými operačními systémy a aplikacemi. XML také nabízí jednoduché zpracování obsahu a umožňuje efektivní způsoby vyhledávání v dokumentech. Navíc je dokument v XML dobře počítačově zpracovatelný i čitelný pro lidi.

Protože jazyk XML definuje strukturu dokumentu, ale neříká nic o jeho vzhledu a zobrazení, je třeba vzhled externě definovat buď pomocí stylu, nebo dokument transformovat do formátu HTML. K tomu je však potřeba znát strukturu zobrazovaného dokumentu. Pokud je potřeba zobrazit obecný dokument, který nemá připojen žádný styl, obvyklým výsledkem je pouze zobrazení struktury dokumentu. Zde přichází na scénu vizualizace XML. Cílem této bakalářské práce bylo vyřešit možné problémy interaktivního zobrazení obecného XML dokumentu a navrhnout program, který by umožnil i „XML neznalému“ uživateli prohlížet a editovat obecný XML dokument.

Tato bakalářská práce přímo navazuje na semestrální projekt, jehož cílem bylo seznámení s problematikou XML a jeho transformacemi a také identifikace možných problémů při zobrazení obecného XML dokumentu.

1.1 Struktura práce

Úvod je věnován stručnému popisu jazyka XML, jeho historii, možnostem využití a také možnostem programového zpracování. Dále je popsán programovací jazyk Java a zpracování XML v Javě. Další část se věnuje integrovanému vývojovému prostředí NetBeans, ve kterém byl výsledný program implementován. Následující kapitola se věnuje koncepci programu a návrhu zobrazení obecného XML dokumentu. V další části je popsána vlastní implementace programu. Na závěr jsou zhodnoceny dosažené výsledky a nastíněny možnosti dalšího vývoje programu.

2 Teorie

2.1 Značkovací jazyk XML

Jazyk XML (*eXtensible Markup Language*, česky *Rozšiřitelný značkovací jazyk*) patří do rodiny značkovacích jazyků. Je to jednoduchý, velmi flexibilní a otevřený textový formát pro publikování a výměnu dokumentů. Přinesl významný pokrok do oblasti přenosu a efektivního zpracování informace.

2.1.1 Historie XML

Koncepce obecného značkování (*generalized markup*) vznikla již v šedesátých letech minulého století, avšak příliš se nerozšířila. Postupným rozšiřováním obecného značkovacího jazyka vznikl jazyk SGML, který byl roku 1986 schválen mezinárodní standardizační organizací ISO normou ISO 8879. SGML (*Standard Generalized Markup Language*) bylo velmi flexibilní a komplexní, aby vyhovělo různým požadavkům. To byla také jeho největší slabina, protože vývoj aplikací, které by podporovaly alespoň typickou část jazyka, byl velmi složitý úkol. Aplikace schopné práce s daty v SGML se ukázaly jako dlouho vyvíjené a přitom dražší, než řešení bez použití SGML (více v [2]).

První masově rozšířenou aplikací SGML se stal jazyk HTML (*HyperText Markup Language*), který se dnes používá k tvorbě webových stránek na internetu. Jazyk vznikl nejprve jako malá množina tagů pro vytváření hypertextových dokumentů. Postupně byly do HTML přidávány další a další tagy, aby se vyšlo vstříc narůstajícím požadavkům internetových uživatelů. To vedlo k problémům s kompatibilitou webových stránek a prohlížečů, protože co uměl jeden prohlížeč, neuměl druhý a naopak. Konsorcium W3C se snažilo tomuto trendu zabránit tím, že pomocí DTD přesně specifikovalo, které tagy a kde se mohou na webových stránkách vyskytovat. Vzniklo tak postupně více verzí HTML. Jazyk si díky své jednoduchosti získal velkou oblibu. Ukázalo se však, že pevně daná skupina značek jazyka HTML již nestačí. Pro účely efektivnější výměny dat a vyhledávání by bylo lepší mít možnost vytvářet vlastní značky (viz [1]).

Výsledkem práce několika předních odborníků byl jazyk XML, jehož finální verze byla publikována v roce 1998. Jazyk byl vytvořen a standardizován konsorciem W3C jako zjednodušená podmnožina jazyka SGML. Při vývoji jazyka se však vycházelo ze zkušeností s jazyky SGML i HTML. Oproti původnímu SGML je XML v mnohém zjednodušen. Flexibilita však zůstala. Původně byl jazyk XML navržen pro ukládání strukturovaného a semi-strukturovaného textu určeného pro šíření a publikaci na celé řadě médií, dnes však hraje stále důležitější úlohu při výměně dat .

2.1.2 Charakteristika XML

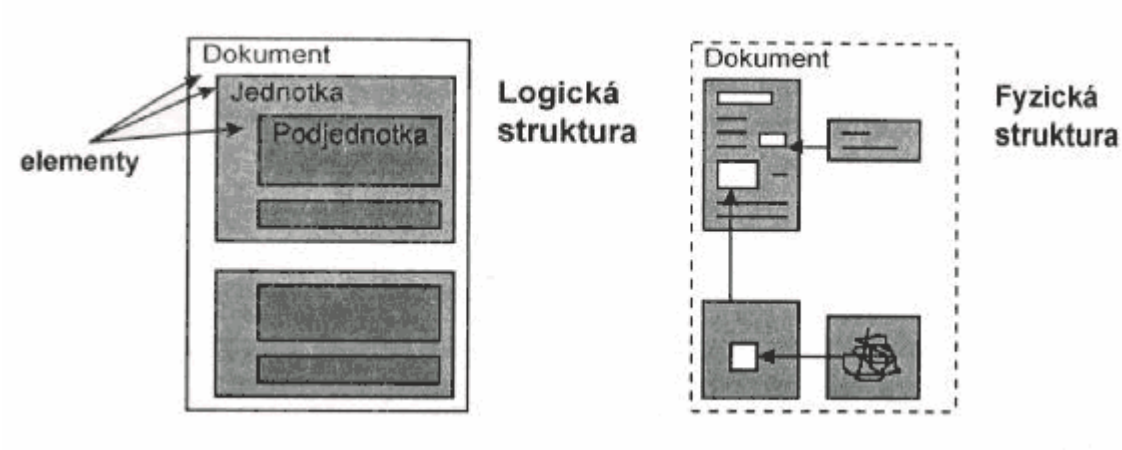
Jazyk XML je značkovací jazyk. Umožňuje popsat strukturu dokumentu z hlediska významu jednotlivých částí textu, nedefinuje však, narozdíl od jazyka HTML, konečné zobrazení dokumentu. Vzhled dokumentu se pak definuje připojeným stylem, nebo se pomocí jiných stylů provede jeho transformace do jiného formátu (např. HTML), popřípadě do jiné struktury XML.

XML používá jako výchozí znakovou sadu *Unicode*, což je tabulka znaků všech existujících abeced. Díky tomu je možné vytvářet dokumenty, které obsahují texty v mnoha jazycích najednou. Navíc je možné použít i jiné kódování než Unicode.

XML je ve skutečnosti metajazyk, což znamená, že se používá k popisu dalších jazyků. Poskytuje naprostou svobodu při pojmenovávání značek, protože v XML neexistuje žádný předdefinovaný seznam elementů. Je tedy možné použít taková jména elementů, která mají pro danou aplikaci smysl a zvyšují tak srozumitelnost dokumentu. Zmatku v pojmenování elementů můžeme zabránit definováním metadat. Přestože se historické kořeny XML nacházejí v oblasti publikování, jazyk je vhodný i pro popis a identifikaci komplexních datových struktur, které nebudou nikdy publikovány (více v [2]).

2.1.3 Struktura dokumentu

Dokument XML má fyzickou a logickou strukturu. Zatímco logická rozděluje dokument na pojmenované jednotky a podjednotky nazývané *elementy*, fyzická struktura umožňuje vkládat do dokumentů samostatné pojmenované části dokumentu zvané *entity*. Entity mohou být i v samostatných datových souborech, aby mohly být opakovaně použity v různých dokumentech a také aby bylo možné odkazovat data, která nejsou standardu XML (například obrázky či binární soubory).



Obr. 2.1: Fyzická a logická struktura XML dokumentu (převzato z [2])

Elementy se v textu vyznačují pomocí tzv. značek (*tagů*).Většina elementů má dvě značky – počáteční a ukončovací. Názvy značek se zapisují mezi lomené závorky (tedy znaky < a >), aby se tak odlišily od zbytku textu. Pokud je v textu třeba tyto znaky použít, je nutné použít zástupných textových entit *<* a *>*. Ukončovací značka má před svým názvem ještě znak /, aby se tak odlišila od počáteční. Ve jménech elementů

se rozlišují malá a velká písmena, takže například `<nazev>`, `<Nazev>` a `<NAZEV>` jsou tři různé elementy. Elementy mohou obsahovat další vnořené elementy a tím dle potřeby zachycovat strukturu informací uložených v dokumentu. Celý dokument je vložen do jediného kořenového elementu dokumentu (více v [2]).

Každý element může ještě kromě svého jména obsahovat další informace, které blíže upřesňují jeho obsah. Tyto informace se nazývají atributy. Každý atribut musí mít jméno, protože element může mít více atributů. Obsah atributu musí být ohraničen v uvozovkách nebo apostrofech. Jednoduchý záznam o osobě v XML dokumentu může například vypadat následovně:

```
<osoba>
  <jmeno>Jan</jmeno>
  <prijmeni>Novák</prijmeni>
  <telefon typ="mobil">777111222</telefon>
  <adresa>
    <ulice cislopopisne="11">Uliční</ulice>
    <mesto>Břeclav</mesto>
    <psc>69002</psc>
  </adresa>
</osoba>
```

XML dokumenty mohou být rozděleny na textově orientované a datově orientované. U textově orientovaných dokumentů převažuje text. Převážná část značek zde slouží pouze k přidání určité informace k textu, ale značky nejsou nezbytně nutné pro pochopení podstaty dokumentu. I kdyby byly všechny značky vynechány, bylo by stále možné pochopit alespoň část textu. U datově orientovaných dokumentů je tomu jinak. Ty jsou určeny převážně pro počítačové zpracování a nehodí se k přímému čtení uživatelem. Značky zde vytvářejí strukturu dokumentu a přidávají datům určitý interpretační smysl. Kdyby byly z takového dokumentu značky vyjmuty, zbývající data by nedávala žádný smysl.

2.1.4 Zobrazení a transformace

Jazyk XML definuje pouze strukturu dokumentu. Umožňuje pojmenování elementů jmény, která popisují podstatu a obsah objektu. Tato informace je samopopisující a lze ji v dokumentu vyhledat, vyjmout a zpracovávat podle potřeby. Na rozdíl od HTML zde neexistuje žádný standard, jak se mají jednotlivé elementy zobrazovat. Chceme-li dokument přehledně zobrazit, je potřeba definovat, jak se jednotlivé elementy zobrazí. Pro definici vzhledu dokumentů se dnes používají stylové jazyky, kterých dnes existuje několik. Mezi nejznámější patří Kaskádové styly (*Cascading Style Sheets*, *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*).

XSL se v podstatě skládá ze dvou částí. První je XSLT (*XSL Transformations*), což je část jazyka určená pro transformaci XML dokumentů. Umožňuje před samotným zformátováním dokument různě upravovat a transformovat (např. části dokumentu vypustit nebo naopak automaticky vygenerovat obsah dokumentu). Druhou částí je XSL FO (*XSL Formatting Objects*) a definuje tzv. formátovací objekty, které slouží pro abstraktní popis vzhledu dokumentu a jeho jednotlivých komponent .

Základním prvkem XSL jsou takzvané šablony (*templates*). Každá šablona určuje, na jakou část vstupního dokumentu bude použita. To je definováno pomocí jazyka XPath (viz následující kapitola 2.1.5). Uvnitř šablony je pak popis toho, jak bude vstupní část dokumentu transformována do výstupního dokumentu (více v [1]).

Jako příklad lze uvést jednoduchou šablonu, která ze vstupního dokumentu převede elementy jménem odstavec na odstavce v jazyce HTML.

```
<xsl:template match=”odstavec”>
    <p><xsl:apply-templates /></p>
</xsl:template>
```

Pro zobrazení obecného XML dokumentu však XSL nelze využít, protože pro aplikaci XSL stylu je potřeba znát předem strukturu XML dokumentu. Navíc, vzhledem k absenci standardu pro zobrazení XML elementů, je zobrazení obecného XML dokumentu bez stylu v lidsky přijatelné podobě nemožné. Alternativou může být zobrazení struktury dokumentu v podobě stromu.

2.1.5 Navigace v XML dokumentu

Velkou výhodou elektronického publikování obecně je možnost provázání více dokumentů hypertextovými odkazy. Ty pak čtenáři usnadňují navigaci v dokumentech. XML rozšiřuje tyto možnosti třemi standardy - *XPath, XLink* a *XPointer* (více o těchto technologiích lze nalézt v [1]).

2.1.5.1 XPath

XPath (*XML Path Language*) je jazyk pro adresaci částí XML dokumentu. Pomocí tohoto jazyka je možné vybrat z XML dokumentu určitou část a pracovat s jejími elementy a jejich atributy. Výsledkem výrazu v jazyce XPath pak může být buď žádný, nebo jediný element, nebo dokonce celý seznam elementů, které splňují podmínku. Toho se s výhodou využívá například v XSL transformacích nebo při odkazování na jiný dokument (XPointer).

XPath také umožňuje pohodlně pracovat s hodnotami (textovými řetězci, čísly) obsaženými v elementech a jejich atributech v XML dokumentu. Výraz v jazyce XPath sestává ze tří částí:

- Identifikátoru osy
- Testu uzlu
- Podmínky

Identifikátor osy určuje uzly, které se budou v daném kroku zpracovávat. Podle výchozí osy to jsou podřízené (synovské) uzly aktuálního elementu, ale je možné adresovat i absolutně od kořenu dokumentu.

Uzly vybrané identifikátorem osy jsou pak dále testovány, čímž je omezena výsledná množina zpracovávaných elementů.

Nakonec postupují všechny elementy, které úspěšně splnily předchozí kritéria, do dalšího kroku, kde je pro každý element vyhodnocena podmínka.

Jednotlivé kroky se oddělují lomítkem stejně jako u adresářové struktury. Výraz `./` tak například znamená aktuální uzel, `../` pak rodičovský (nadřazený) element.

2.1.5.2 XLink

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

2.1.5.3 XPointer

XPointer (*XML Pointer language*) se používá k odkazování určitých částí XML dokumentů stylem „Chci druhý odstavec třetí kapitoly“.

2.1.6 Metadata

Metadata popisují povolenou strukturu dokumentu. Používanými metadaty pro XML jsou DTD a XML schémata. Velkou výhodou XML je to, že v jednom dokumentu můžeme používat najednou nezávisle na sobě několik druhů značkování díky tzv. *jmenným prostorům* (*namespaces*). Můžeme tak vytvářet dokumenty, které používají značky definované pro naše specifické účely.

Metadata se dají s výhodou použít k programové kontrole XML dokumentu pomocí tzv. validujícího parseru, zkráceně validátoru.

2.1.6.1 DTD (*Dokument Type Definition*)

DTD (česky *Deklarace Typu Dokumentu*) je jazyk pro definici legální stavby bloků XML dokumentu, popř. HTML nebo SGML dokumentu. Definuje strukturu dokumentu seznamem legálních elementů a atributů. Popisuje jak mohou být značky navzájem uspořádány a vnořeny. Vymezuje atributy pro každou značku a typ těchto atributů. Následující příklad definuje elementy osoba, jméno, adresa, ulice, číslo, město pro použití na příslušných místech ve všech relevantních dokumentech.

```
<!ELEMENT osoba (jméno, adresa*)>
<!ELEMENT jméno  (#PCDATA)>
<!ELEMENT adresa (ulice?, číslo?, město)>
<!ELEMENT ulice  (#PCDATA)>
<!ELEMENT číslo  (#PCDATA)>
<!ELEMENT město  (#PCDATA)>
```

DTD je již poměrně zastaralý jazyk, ač stále značně používaný. Jeho velkou nevýhodou pro použití v XML je, že sám o sobě není XML dokumentem (více v [1]).

2.1.6.2 XSD (*XML Schema Definition*)

XSD je XML Schéma používané k popisu struktury XML dokumentu. Jedná se o novější alternativu popisu struktury XML dokumentu ke staršímu DTD, roku 2001 se stalo oficiálním doporučením konsorcia W3C.

XML Schéma (převzato z [3]):

- definuje elementy, které se mohou v dokumentu vyskytovat
- definuje elementům přípustné atributy
- definuje, které elementy jsou potomky jiných elementů
- definuje pořadí elementů
- definuje počty elementů
- definuje, zda element může být prázdný, nebo zda může obsahovat text
- definuje datové typy elementů a jejich atributů
- definuje standardní hodnoty elementů a atributů

Velkou výhodou XML schémat je to, že jsou samy o sobě XML dokumentem, takže není potřeba mít speciální parser. Navíc oproti DTD umožňují kontrolovat správnost dat a jejich datových typů. Následující ukázka definuje element osoba, který může obsahovat textové elementy jmeno, prijmeni, adresa a telefon:

```
<xs:element name="osoba">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="jmeno" type="xs:string"/>
   <xs:element name="prijmeni" type="xs:string"/>
   <xs:element name="adresa" type="xs:string"/>
   <xs:element name="telefon" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

2.1.7 Aplikace XML

Díky své flexibilitě a otevřenosti má dnes XML celou řadu aplikací. Od platformně nezávislého přenášení dat přes popis multimédií až po prezentaci dat na webu.

Příklady aplikace XML:

- XHTML – moderní nástupce jazyka HTML, jde v podstatě o jazyk HTML přepsaný do XML. (V současnosti však W3C, které vyvíjí a standardizuje XHTML, přemýšlí o opětovném rozvíjení původního HTML)
- RSS –rodina XML formátů, sloužící pro čtení novinek na webových stránkách. Typicky se používá především na zpravodajských serverech, kde se informace a články mění velice rychle.

- SMIL (*Synchronized Multimedia Integration Language*) - popisuje multimédia pomocí XML. Strukturně se SMIL velmi podobá jazyku HTML.
- MathML (*Mathematical Markup Language*) - značkovací jazyk pro popis matematických vzorců a symbolů pro použití na webu.
- SVG (*Scalable Vector Graphics*) - jazyk pro popis dvourozměrné vektorové grafiky.
- DocBook – sada definic dokumentů a stylů pro publikační činnost. Určen především pro tvorbu dokumentace k softwaru. Lze jej však využít i pro psaní knih, článků nebo přípravu webů.
- Jabber – otevřený komunikační protokol založený na XML. Jeho základ je již standardizován. Jde o alternativu k firemním protokolům pro tzv. *instant messaging* jako jsou například ICQ nebo MSN.
- SOAP (*Simple Object Access Protocol*) – protokol pro komunikaci mezi webovými službami. (Jde například o výměnu ceníků mezi firmami, informací o zboží, objednávkách apod.)
- OpenDocument – souborový formát určený pro ukládání a výměnu dokumentů vytvořených kancelářskými aplikacemi.

2.2 Programovací jazyk Java

Java je programovací jazyk pocházející od firmy Sun Microsystems. Tento objektově orientovaný jazyk je v současné době jedním z nejpoužívanějších programovacích jazyků na světě. Je rozšířen na celé řadě elektronických zařízení: mobilními telefony počínaje, přes stolní počítače, notebooky a servery s rozsáhlými firemními aplikacemi konče. Navíc je pro Javu zdarma dostupné obrovské množství knihoven pro různorodé aplikační oblasti. K dispozici je také celá řada kvalitních vývojových prostředí (i zdarma) jako například *NetBeans*, *JBuilder*, *Eclipse* nebo *Visual Age for Java*.

2.2.1 Historie Javy

Java technologie vznikla jako tajný projekt „Green Project“ v Sun Microsystems v roce 1991. Skupina odborníků nazvaná „Green Team“, vedená Jamesem Goslingem, se tehdy na 18 měsíců odřízla od okolního světa, aby pracovala na něčem, co by mohlo vnést čerstvý vzduch do světa informačních technologií. Skupina si vytyčila za cíl vytvořit jednoduchý a stručný jazyk pro použití v domácích spotřebičích. Protože by měly programy v tomto jazyce běžet na různém hardware od různých výrobců, bylo rozhodnuto, aby jazyk byl platformně nezávislý.

Výsledkem byl programovací jazyk nazvaný *Oak*. Za předchůdce jazyka Oak můžeme považovat jazyk C++. Green Team se však snažil jazyk navrhnout tak, aby bylo jednoduché se ho naučit a používat. Proto z Oaku vypustil některé konstrukce jazyka C++, které se buď málo používají, nebo jejich použití vede ke vzniku velkého množství špatně odhalitelných chyb. Na druhou stranu byla přidána řada užitečných rozšíření. K největším odlišnostem patří silná typová kontrola, odstranění vícenásobné dědičnosti a nemožnost používání ukazatelů.

Green Team poté jako ukázku schopností nové technologie vyvinul dálkově ovládané zařízení, o které však nebyl zájem. V roce 1994 byla následkem toho skupina rozpuštěna. V té době se stával Internet stále populárnějším a Oak představoval možnost, jak oživit do té doby statické webové stránky. Proto byl v Oaku napsán prohlížeč *Web Runner*, do kterého byla navíc implementována možnost interpretovat kód v jazyce Oak. Tento malý kód, který byl interpretován prohlížečem na straně klienta, byl nazván *applet*.

V roce 1995 se zjistilo, že jeden programovací jazyk se jménem Oak již existuje, proto se jazyku Oak začalo říkat *Java* a prohlížeči Web Runner *HotJava*. Applety v prohlížeči HotJava a Java samotná byly předvedeny na konferenci SunWorld v roce 1995 (více na [4]).

Od té doby si Java díky svým vlastnostem získala velkou popularitu.

2.2.2 Charakteristika Javy

Společnost Sun definuje Javu jako jednoduchý, objektově orientovaný, na architektuře nezávislý, přenositelný, distribuovaný, vysoce výkonný, robustní, vícevláknový, dynamický a bezpečný jazyk (viz [5]).

Java je univerzální jazyk, což znamená, že není výhradně určen pro specifickou aplikační oblast. Je objektově orientovaná, takže všechno (s výjimkou primitivních datových typů) je objekt. Používá tzv. silnou typovou kontrolu – veškeré používané proměnné musí mít definovaný svůj datový typ.

Nejvíce charakteristickou vlastností Javy je její platformní nezávislost. To znamená, že program napsaný v jazyce Java je schopen běžet na jakékoli architektuře nebo jakémkoli operačním systému. Je tedy možné napsat program jednou, zkompilovat jednou a poté spustit na libovolném počítači nebo zařízení, které má k dispozici interpret. Toho je dosaženo tím, že zdrojové kódy v Javě (s příponou `.java`) jsou kompilovány do tzv. „bajtkódu“ (soubory s příponou `.class`). Bajtkód jsou specifické instrukce pro virtuální stroj Java platformy. Bajtkód je poté na cílové platformě interpretován a vykonán Virtuálním strojem (*Java Virtual Machine*), což je program napsaný v nativním kódu použitého hardware. Virtuální stroj navíc provádí automatickou správu paměti (*automatic garbage collection*) a sám odklízí již dále nepoužitelné objekty z paměti.

Obr. 2.1: Kompilace a spuštění programu (převzato z [5])

První implementace Java platformy používala Virtuální stroj, který přímo interpretoval bajtkód programu. Spolu s automatickou správou paměti to však vedlo k tomu, že programy v Javě byly mnohem pomalejší než programy překládané do nativního kódu (například v C nebo C++). Java tím získala reputaci nízké výkonnosti. Další implementace však díky mnohým technikám produkují programy, které běží o poznání rychleji než dříve. Příkladem může být přímé přeložení zdrojových textů do strojového kódu dané architektury.

Programy se tím výrazně urychlí, avšak za cenu ztráty přenositelnosti. Další technikou je kompilace za běhu (*Just-in-time*). Nejfrekventovanější části kódu (především cykly) jsou za běhu aplikace překládány do nativního kódu (víz [5]).

2.2.3 Struktura programu

Syntaxe Javy je zjednodušenou verzí syntaxe jazyků C/C++. Java je objektově orientovaný jazyk, takže všechno (kromě primitivních datových typů, které mají ale i své objektové ekvivalenty) je objekt. Základním stavebním kamenem je třída a veškerý kód a aplikační logika musí být obsažen v definicích tříd. Neexistují tedy žádné globální metody nebo proměnné, jako je tomu u jiných ne plně objektových jazyků. Každá veřejná třída musí být v samostatném zdrojovém souboru, jehož název je shodný s názvem třídy a má příponu `.java`. Java umožňuje pouze jednonásobnou dědičnost (každá třída smí mít maximálně jednoho přímého předka).

Každá třída patří do nějakého balíku (*package*). Balík je množina tříd, které logicky patří k sobě. Je to obdoba knihoven a zároveň prostoru jmen (*namespace*) z C++. Název balíku může být buď explicitně uveden (klauzulí *package*), nebo bude třída patřit do implicitně pojmenovaného balíku. Zdrojový soubor, aby mohl být spuštěn, musí obsahovat veřejnou třídu a musí být pod jménem této třídy uložen (viz [6]).

Java rozlišuje dva druhy datových typů: primitivní (reálná čísla, celá čísla, typ znak apod.) a referenční typy (typy, jejichž hodnotou není konkrétní objekt, ale reference na nějaký dynamicky vytvořený objekt).

Spuštění programu odpovídá spuštění metody `main` jedné ze tříd tvořících program.

Ukázka programu, který vypíše text “Hello, world!”.

```
class helloWorld
{
   public static void main(String args[])
   {
      System.out.println(“Hello, world!”);
   }
}
```

2.2.4 Tvorba grafického uživatelského rozhraní

Aplikace využívající grafické uživatelské rozhraní se v Javě píší poměrně lehce. K dispozici je dnes několik variant *GUI* (*Graphical User Interface*, *Grafické uživatelské rozhraní*). Nejstarší z nich je *AWT* (*Abstract Windowing Toolkit*), které se objevilo hned na počátku. Mělo poměrně omezené možnosti a jeho nevýhodou bylo, že každá komponenta měla svůj nativní protějšek v systému. To poněkud ztěžovalo přenositelnost kvůli rozdílům v nativních GUI operačních systémů. Rozhraní sice bylo později přepracováno, ale jeho některé nevýhodné vlastnosti zůstaly. AWT je v dnešní době označeno jako zavržené (*deprecated*) a do budoucna se s ním již nepočítá.

Později se objevila nová grafická knihovna *JFC* (*Java Foundation Classes*), jejíž součástí je GUI nazvané *Swing*. Swing je v dnešní době standardem. Rozhraní Swing je kompletně napsáno v Javě a všechny

komponenty jsou zcela platformně nezávislé. GUI ve Swingu se skládá z grafických komponent a kontejnerů. Kontejnery obsahují komponenty nebo jiné kontejnery. Základním principem GUI je událostní řízení. Tok programu je určen zpracováním asynchronních událostí, které vznikají jako reakce na akce uživatele. Každá komponenta má svého posluchače (*listener*), kterému se při vyvolání události na komponentě zašle zpráva. Posluchač následně událost zpracuje (více na [5]).

Vzhled GUI je dán způsobem poskládání grafických elementů a kontejnerů. Komponenty se vkládají do kontejnerů. Pozice komponenty v kontejneru není dána absolutně, ale relativně ke kontejneru, do kterého je vložena. Způsob umístění, velikost a tvar záleží na správci umístění a pořadí vložení do kontejneru (viz [6]).

2.2.5 Využití Javy

Java se hodí pro celou řadu aplikačních oblastí. Podporuje tvorbu vícevláknových aplikací (*multithreaded applications*), přenositelných aplikací s GUI, hodí se také pro tvorbu aplikací na přenosná a vestavěná zařízení i škálovatelné výkonné aplikace běžící na serverech. Další oblastí využití jsou aplikace distribuované po síti (applety nebo Java Web Start), popřípadě webové aplikace jako JSP (*JavaServer Pages*).

2.2.6 Zpracování XML v Javě

Jazyk XML se stal v podstatě standardem pro popis strukturovaných dat. Spolu s platformně nezávislým programovacím jazykem Java vytváří skvělou kombinaci pro tvorbu (nejen) internetových aplikací. Následující podkapitoly popisují možnosti analýzy a zpracování XML dokumentů v Javě (více na [7]).

2.2.6.1 Parsování

Pro parsování XML dokumentů má Java k dispozici celou řadu programů - tzv. parserů. Prvním krokem ke zpracování dokumentu je jeho analýza, kontrola syntaxe a popřípadě ověření, že dokument odpovídá nějakému DTD nebo XSD, což má na starosti právě parser. Parser tak programátorovi zpřístupní obsah dokumentu v příjemné podobě. Mezi nejznámější parsery patří *Xerces*, *XP* a *Crimson*.

Aby byla práce s dokumentem co nejjednodušší, existují dnes standardizovaná aplikační rozhraní (*Application Programming Interface - API*). V zásadě existují dva přístupy pro parsování XML dokumentu. Jedním z nich je *SAX*, což je přístup založený na událostech, a druhým *DOM*, který je založen na stromové struktuře.

2.2.6.2 Rozhraní SAX

Rozhraní SAX (*Simple API for XML*) je založeno na řízení událostmi (*event driven*). Programátor vytvoří vazbu mezi parserem generovanými událostmi a svým kódem. V podstatě jde o definování funkcí, které budou volány v okamžiku, kdy parser narazí na začátek elementu, obsah elementu, na konec elementu, komentář či instrukce pro zpracování. Definovaným funkcím jsou pak předány potřebné parametry, například název elementu.

Výhodou událostmi řízeného zpracování je jeho rychlost a malá spotřeba paměti. Jednotlivé události jsou vyvolávány postupně, jak je čten dokument. Proto není třeba mít celý XML dokument načtený v paměti. Pokud

je tedy potřeba například vytvořit nějakou vlastní stromovou reprezentaci dokumentu nebo pouze jednoprůchodově zpracovat XML dokument, vyplatí se použít rozhraní SAX.

V případě následujícího XML dokumentu

```
<?xml version="1.0"?>
<dokument>
  <element>Textik</element>
</dokument>
```

vygeneruje SAX parser tuto sekvenci událostí:

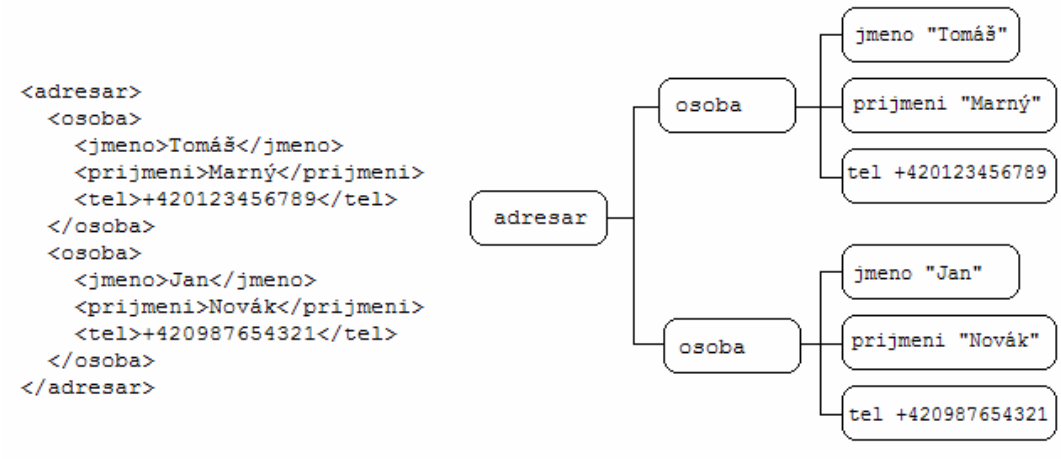
```
start document
start element: dokument
start element: element
characters: Textik
end element: element
end element: dokument
end document
```

2.2.6.3 Rozhraní DOM

Rozhraní DOM (*Document Object Model*) je objektový model pro reprezentaci dokumentů XML a HTML. Dokument je reprezentován jako stromová hierarchická struktura objektů. Tato struktura pak slouží k dalšímu zpracování. Technologie se nazývá GROVE (*Graph Representation Of property ValuEs*) a vyžaduje umístění celého parsovaného dokumentu v paměti, z čehož plynou větší nároky na operační paměť stroje. Její optimální využití je tam, kde je k jednotlivým uzlům přistupováno v náhodném pořadí nebo opakovaně.

Rozhraní DOM umožňuje celý strom dokumentu libovolně procházet, přidávat do něj nové uzly, stávající modifikovat nebo je mazat. Narozdíl od SAXu není potřeba procházet dokument od začátku do konce. Stromem se lze pohybovat libovolným směrem, například od synovského uzlu k rodičovskému, mezi sourozeneckými uzly apod. Proto se rozhraní DOM používá především v aplikacích, které provádějí složitější operace s dokumentem – např. editory nebo prohlížeče.

Následuje ukázka XML dokumentu a zjednodušená stromová struktura dokumentu:



Obr. 2.2: Ukázka stromu XML dokumentu

DOM bylo vyvinuto a standardizováno konsorciem W3C jako reakce na vznikající firemní objektové modely pro HTML, používané především ve webových prohlížečích. W3C se tím snažilo dosáhnout toho, aby nové prohlížeče používaly stejný objektový model pro přístup k dokumentu ze skriptovacích jazyků jako JavaScript.

DOM je obecná specifikace nezávislá na konkrétním programovacím jazyku. V dnešní době existuje celá řada implementací pro Javu - například *JDOM* nebo *dom4j*.

2.3 Vývojové prostředí NetBeans

Vývojové prostředí NetBeans (*NetBeans IDE*) je volně dostupné open source prostředí pro vývoj software v jazyce Java (může ale podporovat jakýkoli programovací jazyk). Vývojové prostředí běží na celé řadě platforem včetně Windows, Linuxu, Solaris OS a Mac OS. Poskytuje vývojářům mnoho užitečných nástrojů potřebných k vytvoření profesionálních multiplatformních aplikací pro desktop, web nebo mobilní telefony.

2.3.1 Historie NetBeans

NetBeans původně vznikl jako studentský projekt na Matematicko-fyzikální fakultě Univerzity Karlovy v České republice v roce 1996 . Projekt nesl původní název *Xelfi*. Cílem bylo v Javě napsat vývojové prostředí pro jazyk Java, což se podařilo. Xelfi se tak stalo prvním vývojovým prostředím napsaným v Jave. Autory tento projekt natolik zaujal, že se po dokončení studií rozhodli udělat z Xelfi komerční produkt. Většina z nich je v NetBeans dodnes nějak angažována.

Kolem Xelfi se postupně utvořila pracovní skupina. Ta byla kontaktována Romanem Staňkem, který hledal nějakou dobrou myšlenku, do které by mohl investovat. Dohodli se na plánu vytvořit síťové JavaBeans komponenty. Jarda Tulach, který navrhl architekturu a základy IDE, přišel s návrhem změnit jméno z Xelfi na NetBeans, aby tak jméno lépe vystihovalo cíle projektu.

Na jaře roku 1999 byla vydána verze NetBeans DeveloperX2, která podporovala rozhraní swing. S příchodem JDK 1.3, které obsahovalo výkonové vylepšení, se stal z NetBeans životaschopný nástroj pro vývoj aplikací v Javě.

V roce 1999 se Sun Microsystem poohlížel po lepším vývojovém prostředí pro Javové aplikace a začal se o NetBeans zajímat. Spolu s novou generací NetBeans Developeru by se NetBeans staly vlajkovou lodí nástrojů pro vývoj Java aplikací. Na podzim 1999 tedy společnost Sun Microsystems koupila NetBeans a rozhodla se přejmenovat toto IDE na Forté for Java. Jméno NetBeans se na nějaký čas ztratilo.

O šest měsíců později, v roce 2000, bylo rozhodnuto udělat z tohoto vývojového prostředí open source, které by bylo sponzorováno samotnou společností Sun. Forté bylo přejmenováno zpět na NetBeans a byla zřízena domovská stránka projektu netbeans.org (převzato z [8]).

2.3.2 Charakteristika NetBeans

NetBeans je úspěšný open source projekt, který dnes tvoří dva produkty: vývojové prostředí NetBeans (*NetBeans IDE*) a platforma NetBeans (*NetBeans Platform*).

Netbeans IDE je nástroj, pomocí kterého mohou programátoři psát, překládat a ladit aplikace. Dnes existuje mnoho modulů, které rozšiřují toto prostředí a přidávají mu nové vlastnosti.

Platforma NetBeans je modulární a dále rozšiřitelný základ pro použití při vytváření rozsáhlých aplikací. Nezávislí dodavatelé navíc nabízejí další moduly pro integraci do této platformy (viz [9]).

NetBeans IDE se snaží poskytnout prostředky a nástroje pro:

- co nejjednodušší a nejrychlejší návrh
- co nejefektivnější ladění
- co největší možnost znovupoužitelnosti kódu

Mezi hlavní rysy vývojového prostředí NetBeans patří:

- swing GUI builder (pro tvorbu grafického uživatelského rozhraní)
- integrovaný debugger
- integrovaná podpora kontroly verzí (*CVS* a volitelně i *SVN*)
- pokročilý editor kódu
- XML editor
- možnost vytvářet aplikace pro mobily a jiná zařízení
- možnost modelování v UML
- možnost programování v jazycích C/C++
- podpora pro tvorbu webových aplikací

3 Koncepce a návrh programu

Vizualizační program by měl splňovat následující požadavky:

- měl by umět zobrazit obecný dokument XML včetně jeho definic
- měl by umožnit interaktivní editaci obsahu

Program je zaměřen spíše na datově orientované XML dokumenty (viz kapitola 2.1.3 Struktura dokumentu). Oproti textově zaměřeným dokumentům jsou v textové podobě hůře čitelné, a proto více vhodné pro vizualizaci. To však nevylučuje, že by textově orientované dokumenty nebylo možné v programu zobrazit, pouze není zaručeno, že budou zobrazeny správně.

Program podporuje zobrazení definic i pro dokumenty, které neobsahují DTD nebo referenci na něj. Nezaručuje však a nehlídá omezení, která by z explicitně definovaného DTD vyplývala.

Interaktivní editace umožňuje kamkoli do dokumentu vložit nový element, editovat textový obsah již existujícího elementu nebo smazat element. Dále je možné vkládat a mazat atributy elementů a také editovat jejich názvy a obsah.

3.1 Koncepce programu

Protože je XML platformně nezávislý textový formát, bylo by vhodné, aby byl i program nezávislý na platformě. Volba proto padla na Javu. Java také navíc poskytuje velké množství nástrojů a knihoven pro práci s XML.

Program pomocí standardního grafického uživatelského rozhraní nabídne uživateli přehledné a intuitivní ovládání. Aplikace bude rozdělena na dvě části. V levé bude zobrazen strom aktuálně otevřeného dokumentu, na pravé straně bude pracovní plocha, do které se budou otevírat okna jednotlivých elementů.

3.2 Návrh zobrazení XML dokumentu

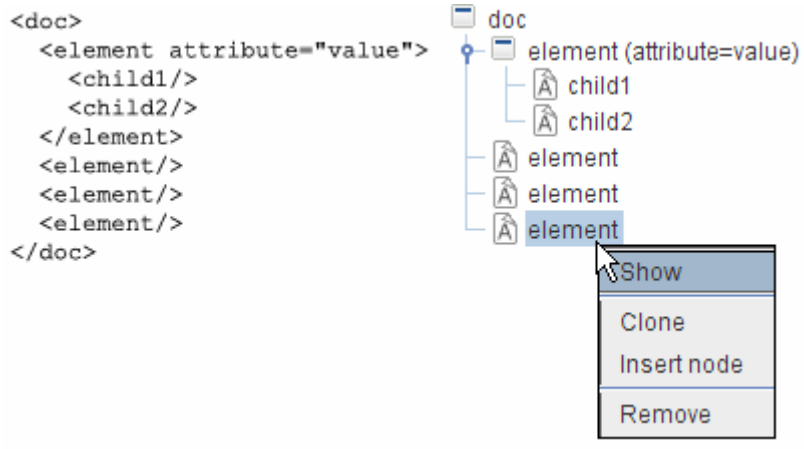
Datově orientovaný dokument může v zásadě obsahovat tři typy prvků:

- Seznam elementů
- Záznam elementů
- Kombinace předchozích

Zatímco u seznamu a záznamu máme možnost unifikovaného zobrazení, při kombinaci obou již mohou vznikat problémy se zobrazením.

3.2.1 Stromové zobrazení struktury dokumentu

Pro usnadnění orientace ve struktuře XML dokumentu, aby se v ní dobře zorientoval i „XML neznalý“ uživatel, bude v programu k dispozici panel se stromovým zobrazením kompletní struktury dokumentu. Stromem bude možné libovolně procházet. Pomocí voleb v kontextovém menu každého uzlu bude možné přidávat nové elementy a mazat existující. Dále bude možné „naklonovat“ existující element i s jeho veškerým obsahem (tzv. *deep clone*) a atributy. Naklonovaný element se stane sourozencem původního elementu. Přes volbu v kontextové nabídce bude také možné zobrazit obsah elementu. Jak je patrné z obrázku č. 3.1, každý uzel má vlastní ikonu, která znázorňuje, zda je to textový uzel (nemá již další vnořené uzly), nebo uzel s dalšími potomky.



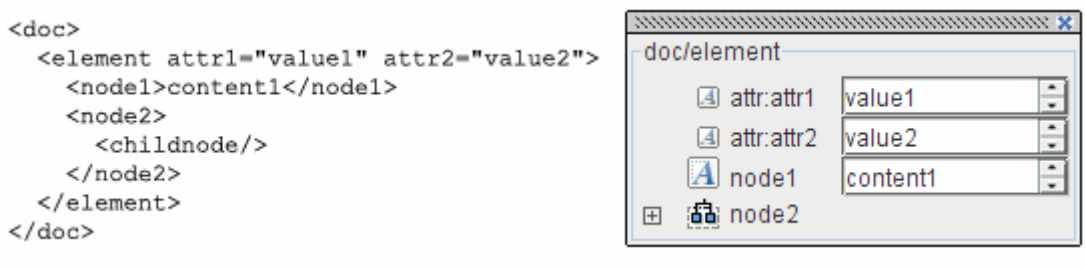
Obr. 3.1: Zobrazení stromové struktury XML dokumentu

3.2.2 Zobrazení obsahu elementu

Element je možné reprezentovat buď jeho jménem, nebo obsahem. Obsah každého elementu bude pro přehlednost zobrazen v novém „vnitřním okně“ pracovní plochy aplikace. V titulku okna bude XPath cesta elementu (viz. kapitola 2.1.5 Navigace v dokumentu), aby tak bylo možné zobrazený element jednoznačně a rychle identifikovat. Obsah elementu bude zobrazen jako seznam jeho atributů a jejich hodnot, dále seznam všech synovských elementů a pokud element obsahuje i text, pak i tento textový obsah elementu.

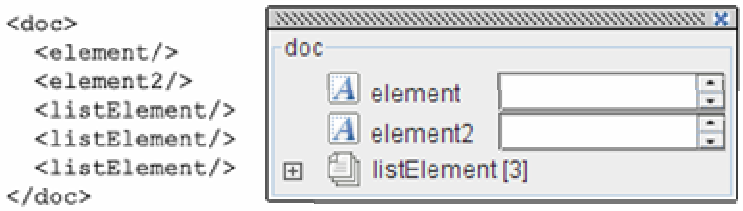
Pro synovské elementy, jejichž obsah je opět strukturovaný (mají atributy nebo obsahují další vnořené elementy), bude nutné zobrazit ke jménu ještě tlačítko odkazující na obsah těchto synovských elementů.

Naopak u atributů a těch synovských elementů, které obsahují přímo textovou hodnotu (tj. obsahují pouze text a žádné elementy), se za název zobrazí editovatelné pole s jejich obsahem. To podstatně ulehčí editaci dokumentu. Obrázek číslo 3.2 ukazuje zobrazení obsahu elementu, který má dva atributy a dva synovské elementy, přičemž druhý synovský element je ještě dále strukturovaný.



Obr. 3.2: Zobrazení obsahu elementu

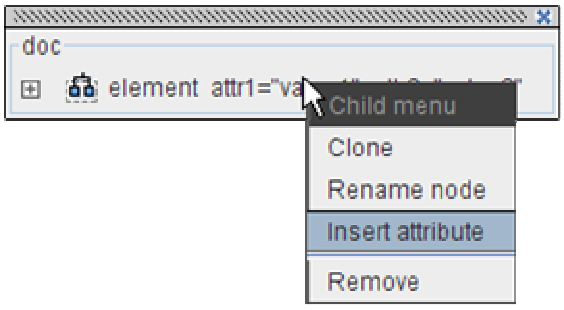
Problém se zobrazením kombinace seznamu a záznamu v elementu jsem vyřešil tak, že se při výpisu synovských elementů seskupují elementy stejného jména do seznamu, který je pak standardně zobrazitelný. Ostatní elementy se dále zobrazují podle pořadí v dokumentu. Příklad zobrazení elementu, který obsahuje kombinaci seznamu s ostatními elementy, je na obrázku číslo 3.3.



Obr. 3.3: Zobrazení smíšeného obsahu elementu

Pokud má element nějaké atributy, zobrazí se jako seznam vedle jeho jména ve tvaru `jmenoAtributu= “obsahAtributu“.`

Každý ze synovských elementů má pak kontextové menu, které opět nabízí vložení elementu, smazání tohoto elementu, přejmenování a naklonování (viz obrázek č. 3.4).



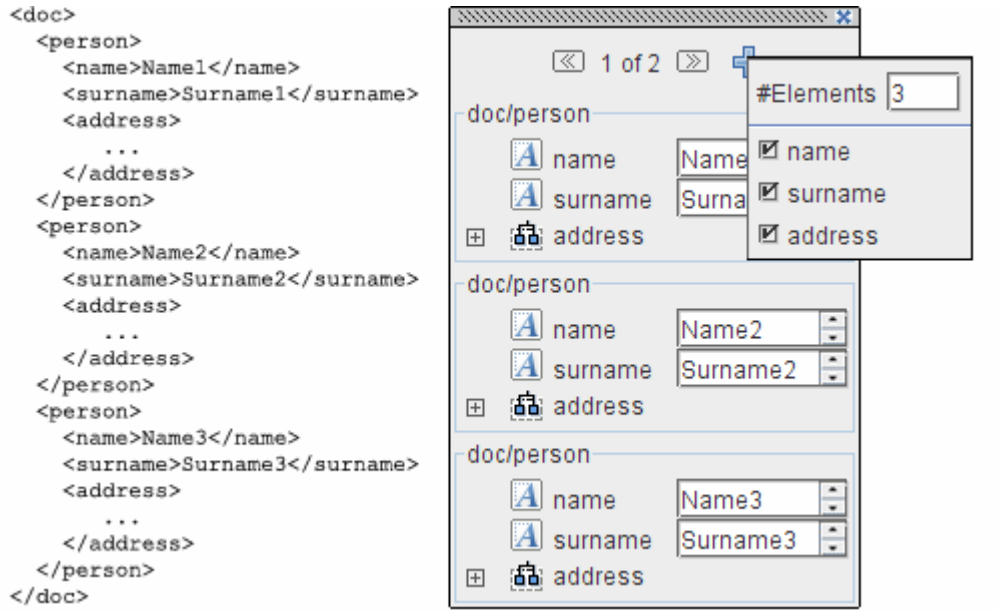
Obr. 3.4: Zobrazení obsahu elementu – kontextové menu synovského elementu

3.2.3 Zobrazení seznamu

Speciálním případem pro zobrazování je seznam elementů. Ten lze zobrazit vhodně uspořádaným seznamem vnitřních prvků těchto elementů. Nabízí se zde možnost stránkování seznamu a také filtrování zobrazeného obsahu elementů. Pokud by například elementy obsahovaly velké množství prvků, mohl by si

uživatel zvolit, které prvky chce zobrazit a které ne. Dále také přichází v úvahu nastavení počtu zobrazených elementů na stránku.

Příklad zobrazení seznamu elementů je na následujícím obrázku.

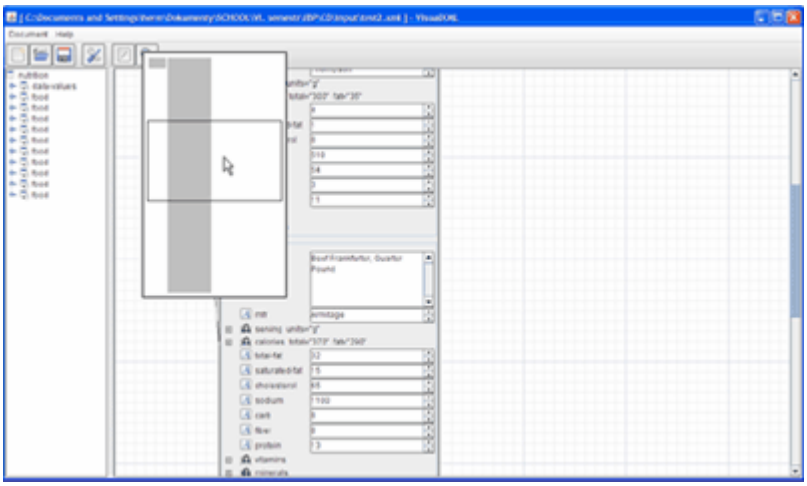


Obr. 3.5: Zobrazení seznamu elementů

3.2.4 Nástroj pro usnadnění orientace v programu

Pro usnadnění orientace na pracovní ploše programu, kde může být otevřeno i velké množství oken elementů, jsem navrhl nástroj *desktopView*, pomocí kterého se lze velice rychle přesouvat na různá místa plochy. Nástroj zobrazuje zmenšenou plochu se všemi aktuálně otevřenými okny a umožňuje interaktivně posouvat viditelnou část pracovní plochy.

DesktopView obsahuje průhledný panel, který reprezentuje skutečnou viditelnou část pracovní plochy. Tažením myší se tento panel přesouvá a zároveň s ním se posouvá i viditelný výřez plochy.



Obr.3.6 : Ukázka nástroje desktopView

4 Vlastní implementace

Program byl implementován v jazyce Java (viz kapitola 2.2 Java). Při vývoji programu bylo využito vývojové prostředí NetBeans 5.5. Programu jsem dal vývojové jméno *VisualXML*.

Následující podkapitoly obsahují popis základních částí implementace.

4.1 Volba rozhraní pro reprezentaci XML

Před vlastní implementací jsem se rozhodoval, jaké rozhraní pro parsování a reprezentaci XML dokumentu použít. Rozhraní SAX je rychlejší a má mnohem menší paměťové nároky než rozhraní DOM. Pro tuto aplikaci se však nehodí, protože program má umožnit náhodný přístup k elementům. Proto jsem s výhodou využil rozhraní DOM, které načte celý obsah dokumentu do stromové struktury v paměti. Rozhraní DOM poskytuje přesně to, co program potřebuje. Pro jazyk Java jsem použil implementaci *dom4j*, se kterou jsem již měl zkušenosti z dřívější doby.

4.2 Zobrazení obsahu elementu

Protože jsem chtěl zobrazit XML dokument a jeho elementy podobně jako např. UML (*Unified Modelling Language*), rozhodl jsem se elementy zobrazovat v komponentě *JInternalFrame*, která poskytuje sama o sobě velké množství požadované funkčnosti (např. zavření okna, přesouvání okna po pracovní ploše apod.). Zároveň jsem však zakázal tlačítka pro minimalizaci a maximalizaci okna, protože jsem je považoval pro aplikaci za nežádoucí. Obsah elementu je zobrazen v samostatném kontejneru *JPanel*. Kontejnery se řadí pod sebe, což umožňuje zobrazit libovolně dlouhý seznam. K určení rozmístění prvků k zobrazení obsahu elementu jsem použil vizuálního nástroje prostředí NetBeans. Každému prvku (elementu, atributu nebo seznamu elementů) jsem přiřadil kontextové menu, které dále rozšiřuje možnosti editace. Kontextových nabídek je však více, protože např. operace pro atributy se liší od operací s elementy. Také samotný *JPanel*, reprezentující rodičovský element, má kontextovou nabídku.

Všechny tyto *JInternalFramy* (otevřená okna elementů) jsou umístěny v kontejneru *JDesktopPane*, který tvoří vlastní pracovní plochu aplikace. Na pozadí kontejneru je zobrazena mřížka, která by měla usnadnit orientaci uživatele na ploše. Protože je pozadí dlaždicováno jediným obrázkem, implementoval jsem předlaždicování kontejneru při změně velikosti nebo při posunu viditelného výřezu. Původní jednoduché dlaždicování se ukázalo jako neefektivní, protože se i při velké pracovní ploše překreslovaly všechny dlaždice. Proto jsem algoritmus upravil tak, že se překreslují dlaždice pouze ve viditelném výřezu. *JDesktopPane* při umístění okna mimo jeho viditelnou plochu automaticky nezobrazuje scrollovací lišty. Musel jsem proto dále tento kontejner umístit do *JScrollPane*, což je komponenta umožňující scrollování svého obsahu.

4.3 Zobrazení spojnice oken

Abych docílil vizuálního spojení zobrazovaného elementu s jeho rodičovským elementem (které jsou reprezentovány vnitřními okny pracovní plochy), bylo potřeba navrhnout komponentu, která by mezi svými dvěma krajními body vykreslovala jednoduchou čáru. K vykreslení čáry jsem využil standardních funkcí. Řešení však zkomplikovalo rozhodnutí úsečku dynamicky přichytávat k různým bodům spojených oken v závislosti na poloze oken. Při pohybu okna elementu je určen kvadrant, ve kterém se nachází pohybované okno vůči oknu rodičovskému (je-li nějaké). V každé pozici oken existuje vždy několik možností, kam úsečku přichytnout. Konečné přichycení se pak vybírá podle úhlu, který by svírala vykreslená úsečka s osou kvadrantu. Použita je pak úsečka, která s osou kvadrantu svírá nejméně ostrý úhel.

4.4 Strom struktury dokumentu

Pro další zpříjemnění a zefektivnění práce s dokumentem jsem do levé části aplikace umístil komponentu *JTree*, která zobrazuje skutečnou strukturu dokumentu. Různým typům uzlů jsou přiřazeny různé obrázkové ikony, aby bylo zřetelné, zda se jedná o uzel s dalším obsahem, nebo o čistě textový uzel.

Každý uzel má své kontextové menu, které umožňuje provádět s uzlem další editační akce.

4.5 Podpora českého jazyka

Při implementaci jsem kladl důraz na to, aby také bylo možné editovat a ukládat dokumenty v českém jazyce. Na různých operačních systémech se pro češtinu používají různá kódování (např. na operačním systému Windows je to kódování *Windows-1250*, jinak také *CP1250*, na Linuxu je to *ISO-8859-2*). Defaultním kódováním jazyků Java i XML je kódování Unicode. Aby byla umožněna editace XML dokumentu v jakémkoli kódování, je při načtení dokumentu provedeno překódování do Unicode a při pozdějším uložení je dokument překódován zpět do původního kódování. Pokud se však skutečné kódování XML dokumentu neshoduje s kódováním uvedeným v XML hlavičce, dokument nemusí být kvůli neznámým znakům správně parsován a načtení dokumentu se nezdaří.

4.6 Nástroj desktopView

Základem nástroje je komponenta *JPanel*, jejíž velikost odpovídá jedné pětině plné velikosti pracovní plochy. Do tohoto kontejneru jsou dále umístěny další JPanely, které reprezentují otevřená okna elementů. Tyto panely jsou pozicovány absolutně, přičemž mají poměrově stejné rozmístění a velikost jako původní okna. Vše je v poměru jedna ku pěti. Panely jsou v šedé barvě a celá komponenta má bílé pozadí, takže je celá kompozice dostatečně kontrastní.

Poslední část nástroje tvoří pohyblivý *JPanel*, který nemá definováno pozadí a je tudíž průhledný. Je však ohraničen černým okrajem, takže je dobře viditelný. Velikost tohoto průhledného panelu je jedna pětina aktuální velikosti výřezu pracovní plochy. Při pohybu tohoto panelu (tažením myší) se pohybuje také viditelná část pracovní plochy.

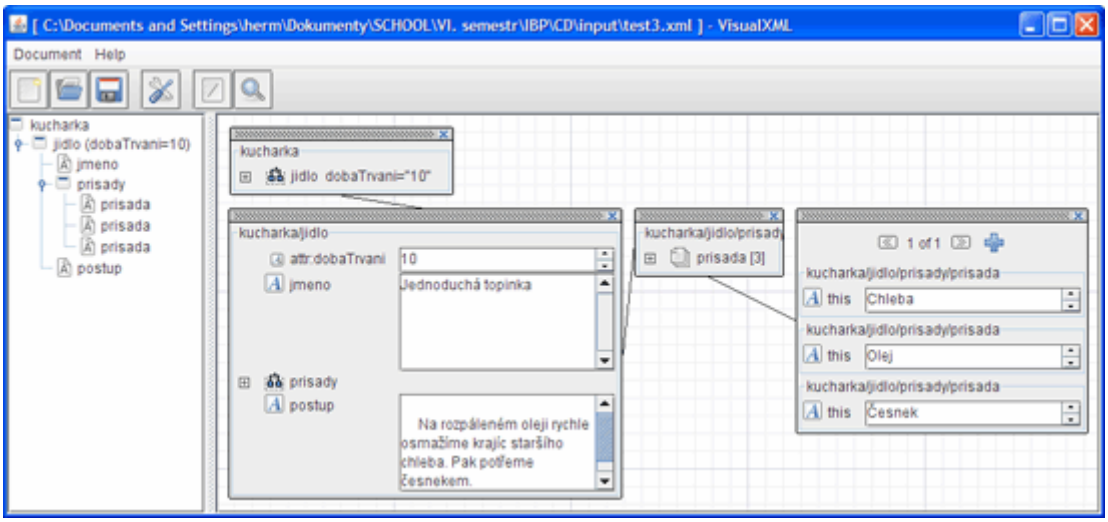
Celá komponenta je umístěna v kontextovém menu tlačítka v nástrojové liště, aby byla vždy pohodlně přístupná.

5 Závěr

5.1 Zhodnocení výsledků

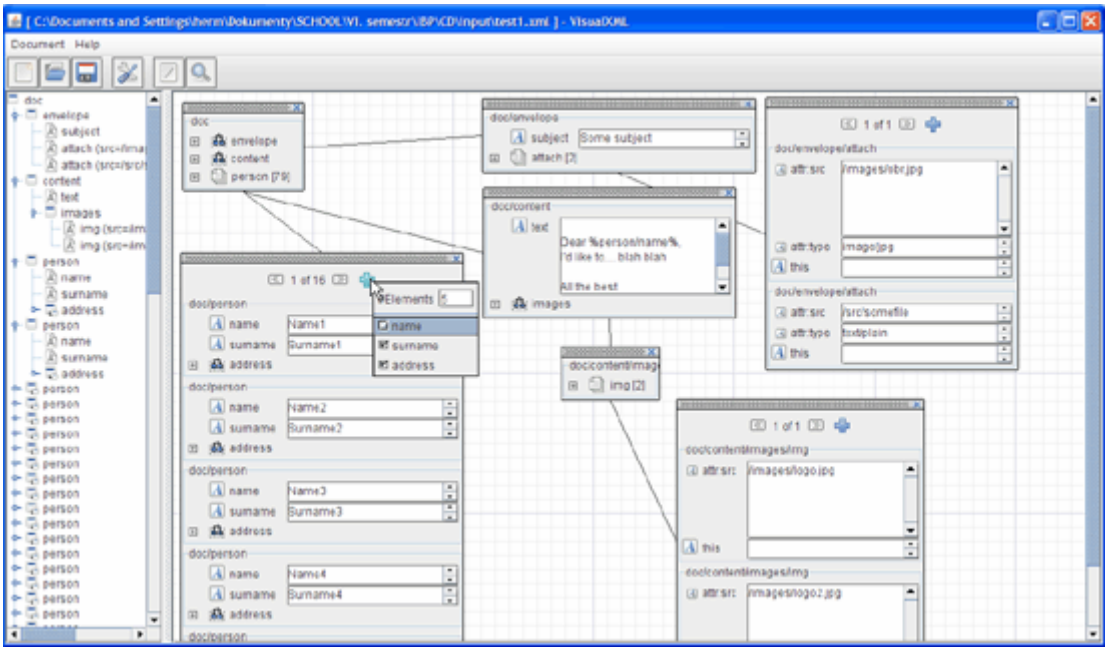
V této kapitole bych se rád věnoval zhodnocení dosažených výsledků. Pokud v programu otevřeme malý datově orientovaný XML dokument, je vše zobrazeno jak má a s dokumentem lze poměrně intuitivně pracovat. Například následující XML dokument bude v programu zobrazen jak je vidět na obrázku číslo 5.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<kucharka>
<jidlo dobaTrvani="10">
  <jmeno>Jednoduchá topinka</jmeno>
  <prisady>
    <prisada>Chleba</prisada>
    <prisada>Olej</prisada>
    <prisada>Česnek</prisada>
  </prisady>
  <postup>
Na rozpáleném oleji rychle osmažíme krajíc staršího chleba. Pak potřeme
česnekem.
  </postup>
</jidlo>
</kucharka>
```



Obr. 5.1: Ukázka otevření jednoduchého datově orientovaného XML dokumentu

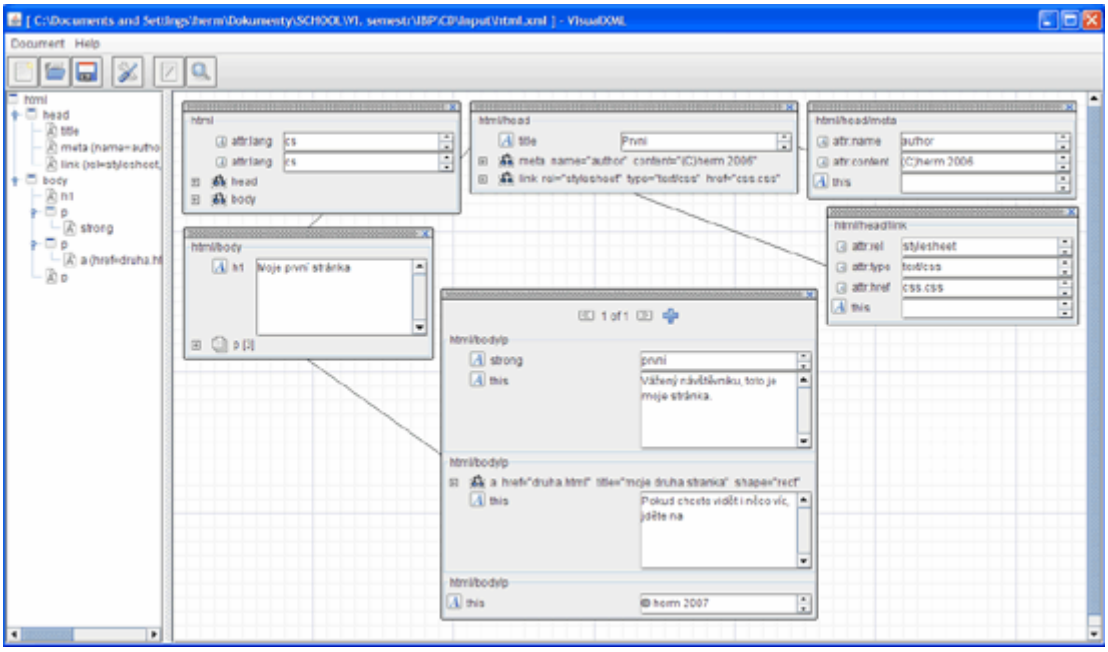
Na dalším příkladu bude demonstrováno zobrazení dlouhého seznamu. Otevřený dokument obsahuje v kořenovém elementu mimo jiné i seznam 80 elementů. Jak je vidět na obrázku 5.2, stromová struktura vlevo ztrácí svou přehlednost. Je zobrazen dlouhý seznam všech elementů seznamu. Naproti tomu v elementech na pracovní ploše se dá stále bez problémů orientovat. U zobrazeného seznamu je navíc viditelný filtr, který je užitečný při zobrazení seznamu obsáhlejších elementů. Zobrazeno je 5 elementů na stránku.



Obr. 5.2: Zobrazení dokumentu, který obsahuje velké množství elementů

Problémy se mohou objevit při otevření textově orientovaného dokumentu jako například dokumentu XHTML (u dokumentů formátu HTML není zaručeno, že je program otevře). Následuje zdrojový kód XHTML dokumentu. Na obrázku číslo 5.3 je pak vidět, jak program VisualXML dokument zobrazil. Protože je dokument spíše textově orientovaný, lze si všimnout, že element p s původním zdrojovým kódem `<p>Vážený návštěvníku, toto je moje <strong>první</strong> stránka.</p>` má textový obsah „Vážený návštěvníku, toto je moje stránka.“. Byl z něj tedy vynechán obsah elementu strong, který je uveden zvlášť. To je typická chyba, která se bude vyskytovat u všech textově orientovaných dokumentů. Jinak je ale dokument zobrazen správně.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="cs" lang="cs">
  <head>
   <title>Prvni</title>
   <meta name="author" content="(C)herm 2006"/>
   <link rel="stylesheet" type="text/css" href="css.css"/>
  </head>
  <body>
    <h1>Moje první stránka</h1>
    <p>Vážený návštěvníku, toto je moje <strong>první</strong> stránka.</p>
    <p>Pokud chcete vidět i něco víc, jděte na
      <a href='druha.html' title='moje druha stranka'>mou druhou
stránku</a></p>
    <p>&copy; herm 2007</p>
  </body>
</html>
```



Obr. 5.3: XHTML dokument otevřený v programu VisualXML

5.2 Možnosti dalšího vývoje

Aplikace VisualXML poskytuje základní nástroje a možnosti pro interaktivní zobrazení a editaci obecného XML dokumentu. Další vývoj by se týkal rozšíření stávajících možností programu a také vylepšení a rozšíření uživatelského rozhraní.

Pro rozšíření možností práce s dokumentem by bylo vhodné implementovat funkci *drag and drop* na elementy a editační textová pole. Popřípadě by bylo možné implementovat funkce pro kopírování a vkládání elementů.

Dalším vylepšením by jistě bylo přidání funkcí *Undo* a *Redo* (česky *Zpět* a *Opakovat*) a to jak na editovatelný text, tak i na otevírání, přesun a zavírání oken elementů. Výhodnou by také mohla být funkce pro zapamatování nedávno otevřených dokumentů. Zapamatováno by také mohlo být rozložení oken dokumentu na pracovní ploše.

Další zvýšení použitelnosti by přinesla implementace pokročilého vyhledávání v dokumentu. Vyhledávací funkce by mohla poskytovat rozhraní pro určení prohledávané skupiny elementů pomocí jazyka XPath a hledané textové hodnoty. Výsledek by se pak zobrazil jako seznam elementů, splňujících daná kritéria.

Přidáním nástroje pro interaktivní zobrazení zdrojového kódu dokumentu by uživatel získal možnost editovat přímo zdrojový kód XML dokumentu. Všechny změny by ovšem musely být dynamicky přenášeny do ostatních částí programu, aby tak nevznikaly nekonzistence dat.

Další užitečnou vlastností aplikace by jistě byla lepší podpora XML Schémat (viz kapitola 2.1.6 Metadata). A to například umožněním editace dokumentu s možností validace oproti XSD. Do aplikace by také

mohl být přidán nástroj pro zobrazení XML Schéma dokumentu. Schéma je sice možné programem přímo otevřít, ale zbytečně to uživateli komplikuje práci.

Jako poslední možnost dalšího vývoje lze uvést implementaci funkce pro automatické uspořádání již otevřených oken, která by uživateli pomohla v orientaci ve velkém množství otevřených a po ploše roztroušených oken.

5.3 Vlastní zhodnocení

Práce se zabývá vizualizací XML. Její velká část je věnována problematice XML a možnostem jeho vizualizace.

Cílem práce bylo také vytvoření programu, který by umožnil interaktivně zobrazit obecný XML dokument. Výsledkem je fungující aplikace, která splňuje všechna základní kritéria. Navíc jsem nad rámec zadání implementoval nástroj desktopView, který usnadňuje orientaci na pracovní ploše programu. Také jsem rozšířil možnosti editace dokumentu o manipulaci s elementy a atributy.

V budoucnu bych se chtěl problematikou dále zabývat a pokusit se vylepšit aplikaci o nástroje a funkce uvedené v podkapitole 5.2.

Při práci na projektu jsem se zdokonalil v programování v jazyce Java.

6 Literatura

[1] Kosek, J.: *XML pro každého: podrobný průvodce.* 1. vyd. Praha : Grada Publishing s.r.o., 2000. 164 s. ISBN 80-7169-860-1.
Dostupný z WWW: <http://www.kosek.cz/xml/xmlprokazdeho.pdf>.

[2] Bradley, N.: *XML: kompletní průvodce.* Praha: Grada, 2000. 537 s. Obsahuje rejstřík. ISBN 80-7169-949-7.

[3] *Introduction to XML Schema* [online]. [Refsnes Data], c1999-2007 [cit. 2007-05-09].
Dostupný z WWW: <http://www.w3schools.com/schema/schema_intro.asp>.

[4] Byous, J.: *JAVA TECHNOLOGY: THE EARLY YEARS* [online]. c1994-2007 [cit. 2007-05-02]. Dostupný z WWW: <http://java.sun.com/features/1998/05/birthday.html>.

[5] Sun Microsystems. *The Java Tutorials* [online]. c1995-2007 , Last update: April 30th, 2007 [cit. 2007-05-01].
Dostupný z WWW: <http://java.sun.com/docs/books/tutorial/>.

[6] Kočí, R.: Přednášky do předmětu Seminář Java

[7] Armstrong, E., et al. *The Java API for Xml Processing (JAXP) Tutorial* [online]. Version 1.1. Sun Microsystems, c1994-2007 , Update 35 -- 10 Dec 2001 [cit. 2007-05-02].
Dostupný z WWW: <http://java.sun.com/xml/tutorial_intro.html>.

[8] *A Brief History of NetBeans* [online]. [2006] [cit. 2007-04-30].
Dostupný z WWW: <http://www.netbeans.org/about/history.html>.

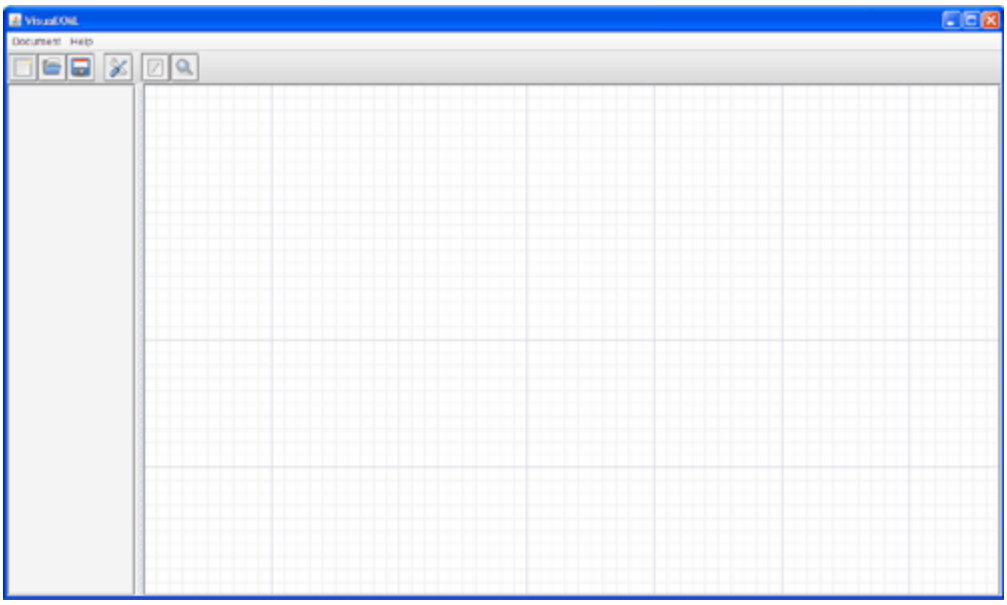
[9] *Introduction to the NetBeans IDE 5.5* [online]. [2007] [cit. 2007-05-02].
Dostupný z WWW: <http://www.netbeans.org/products/ide/>.

7 Přílohy

7.1 Příloha 1. Uživatelská příručka programu VisualXML

7.1.1 Spuštění programu

Ke spuštění aplikace je potřeba mít nainstalováno běhové prostředí Java 6 nebo novější. Pak lze program spustit z aktuálního adresáře buď poklepáním na soubor VisualXML.jar, nebo z příkazové řádky příkazem `java –jar VisualXML.jar`.



Obr. 7.1: Okno programu po spuštění

Po spuštění se otevře hlavní okno programu. Okno programu obsahuje hlavní menu, nástrojovou lištu, dále vlevo je zobrazována struktura otevřeného dokumentu v tzv. *XMLStromu* a vpravo se nachází pracovní plocha programu.

7.1.2 Popis programu VisualXML

7.1.2.1 Nástrojová lišta

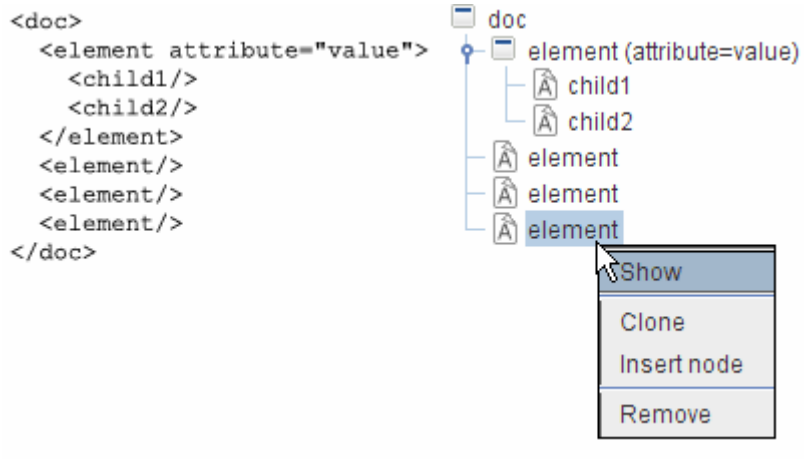
Obr. 7.2: Nástrojová lišta programu

Význam tlačítek zleva:

- nový XML dokument
- otevření dokumentu
- uložení dokumentu
- nastavení [neimplementováno]
- zobrazení kořenového elementu
- nástroj desktopView

7.1.2.2 Strom struktury dokumentu

V levé části programu (v tzv. *XMLStromu*) je stromově zobrazena struktura XML dokumentu.



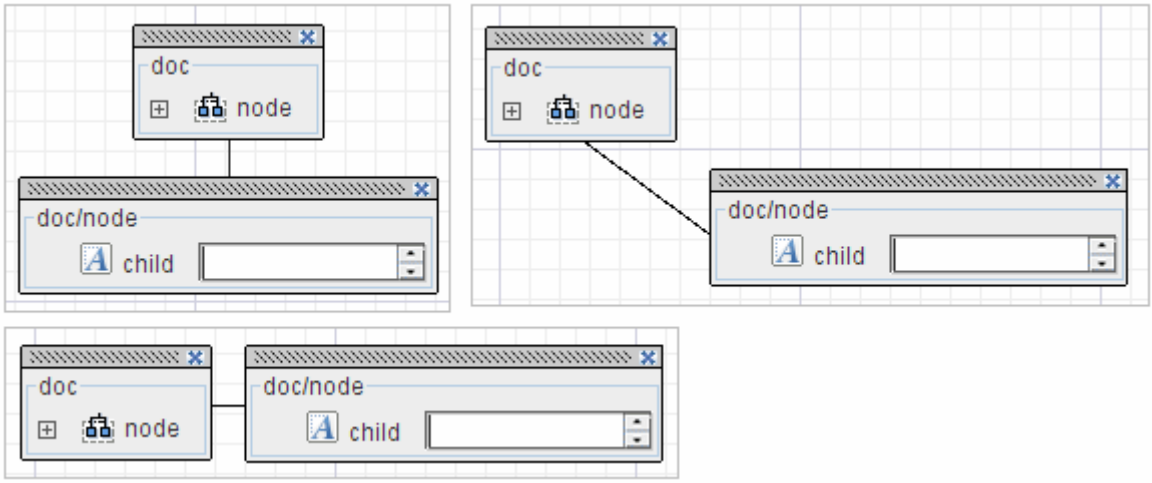
Obr. 7.3: Strom struktury dokumentu

Kliknutím pravým tlačítkem myši na uzel zobrazíte kontextové menu uzlu. Menu umožňuje přidat synovský uzel k aktuálnímu nebo aktuální uzel odstranit. Také lze zobrazit obsah elementu na pracovní ploše.

7.1.2.3 Pracovní plocha

Nejdůležitější částí aplikace je pracovní plocha, která umožňuje samotnou práci s dokumentem. Pracovní plocha je pro usnadnění orientace pokryta mřížkou. Právě na ní se zobrazují elementy.

Aby bylo zřejmé, který element je potomkem kterého, jsou tyto elementy (resp. okna těchto elementů) spojeny úsečkou. Pro usnadnění orientace úsečka mění záchytné body oken, a to v závislosti na vzájemné pozici obou oken na pracovní ploše. Okna elementů jsou uzavíratelná a při zavření okna rodičovského elementu jsou rekurzivně zavřena všechna okna potomků.



Obr. 7.4: Změny záchytných bodů mezi okny

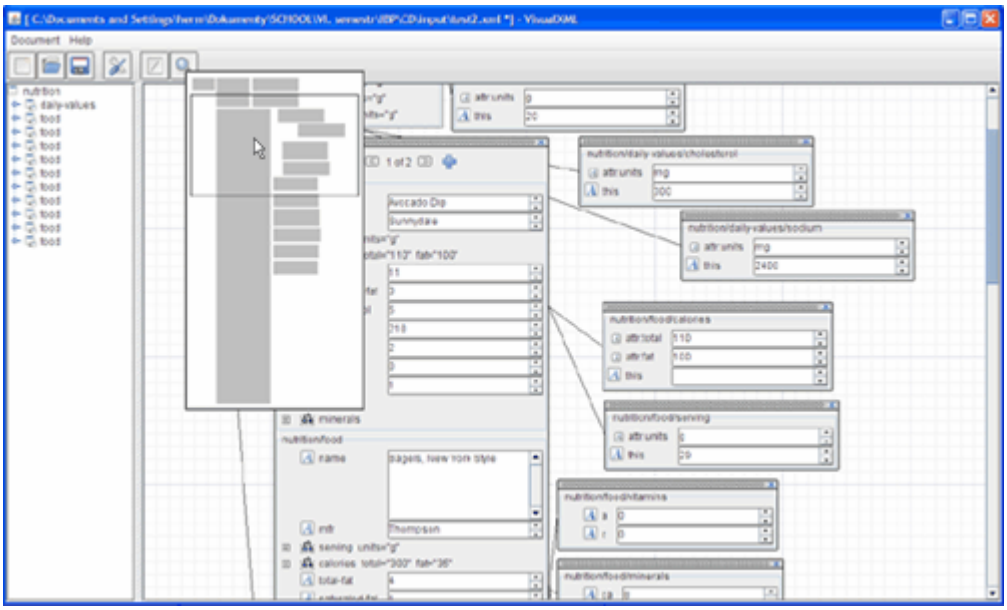
7.1.2.4 Zobrazení kořenového elementu

Pokud byl kořenový element na pracovní ploše uzavřen, tímto tlačítkem je možné jej opět zviditelnit.

7.1.2.5 desktopView

Nástroj pro snadnější orientaci na rozsáhlé pracovní ploše. Pomocí desktopView se lze velice rychle přesouvat na různá místa plochy. Nástroj zobrazuje zmenšenou plochu se všemi aktuálně otevřenými okny a umožňuje interaktivně posouvat viditelnou část pracovní plochy.

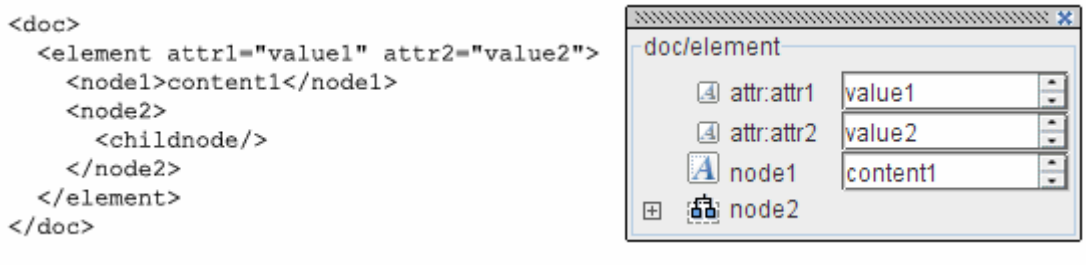
DesktopView obsahuje průhledný panel, který reprezentuje skutečně viditelnou část pracovní plochy. Tažením myší se tento panel přesouvá a zároveň s ním se posouvá i viditelný výřez plochy.



Obr. 7.5: Nástroj desktopView

7.1.2.6 Zobrazení elementu

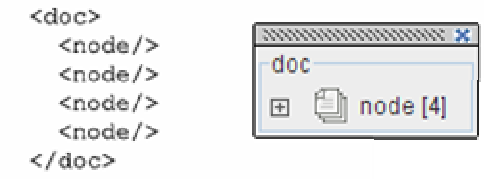
Obsah elementu je zobrazen jako seznam jeho atributů, synovských elementů, reprezentovaných jejich jmény, a jeho případného textového obsahu (pod názvem *this*). Atributy a čistě textové synovské elementy jsou zobrazeny s editačním polem vedle svého jména, takže mohou být okamžitě editovány. Pokud je však synovský element dále strukturován, editační pole se nezobrazí, místo toho se vedle jména elementu objeví tlačítko, pomocí kterého lze daný element otevřít (resp. zobrazit jeho obsah).



Obr. 7.6: Ukázka zobrazení obsahu elementu

Jak je patrné z obrázku číslo 7.6, u názvu elementu je vždy ikona, která znázorňuje, zda se jedná o atribut, textový či dále strukturovaný element. Navíc pokud se jedná o atribut, je před jeho jméno přidán text „*attr:*“, aby tak byl ještě více odlišen od elementů.

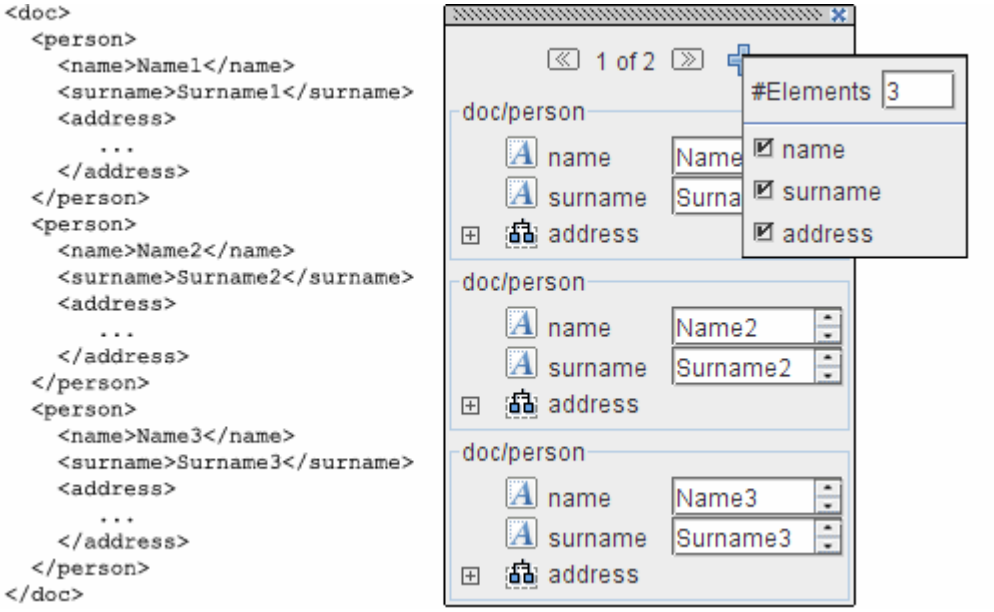
Pokud element obsahuje více elementů stejného jména (seznam elementů), jsou tyto elementy sdruženy pod jediný název, za kterým následuje číselná velikost seznamu. Viz následující obrázek:



Obr. 7.7: Sdružení více elementů stejného jména do seznamu

7.1.2.7 Zobrazení seznamu

Speciálním případem pro zobrazování je seznam elementů. Ten je zobrazován uspořádaným seznamem obsahu těchto elementů. Pomocí navigace, zobrazené v horní části okna, je možné seznam stránkovat a také filtrovat zobrazený obsah elementů (je možné si zvolit, které synovské elementy chceme zobrazit a které ne). Dále lze také nastavit počet zobrazených elementů na stránku.

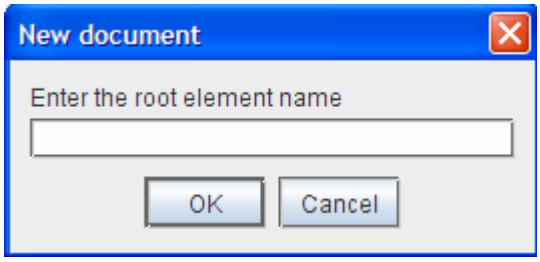


Obr. 7.8: Zobrazení seznamu

7.1.3 Práce s XML dokumentem

7.1.3.1 Nový XML dokument

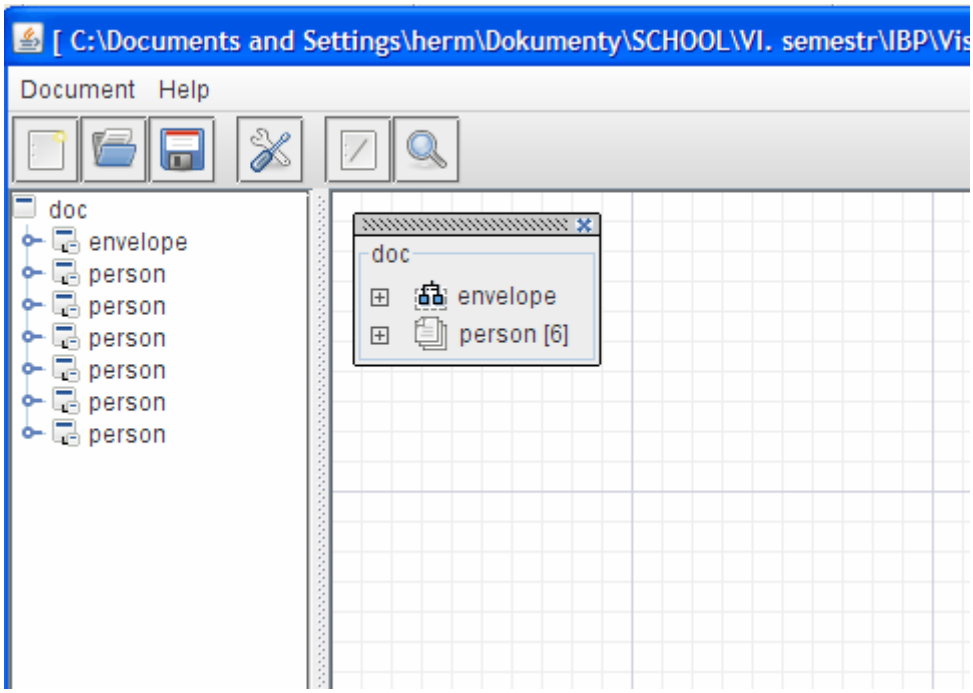
Po stisku tlačítka "Nový dokument" se zobrazí dialog pro zadání jména kořenového elementu. Po jeho zadání se v XMLStromu zobrazí kořenový element zadaného jména a na pracovní ploše se objeví okno s obsahem tohoto kořenového elementu.



Obr. 7.9: Dialog pro zadání názvu kořenového elementu

7.1.3.2 Otevření XML dokumentu

Po otevření existujícího XML dokumentu standardním dialogem se vlevo v XMLStromu se zobrazí struktura dokumentu a na pracovní ploše bude otevřen kořenový element dokumentu.



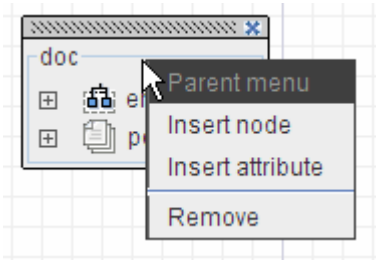
Obr. 7.10: Ukázka otevřeného dokumentu

7.1.3.3 Uložení dokumentu

Pokud byl XML dokument editován (zobrazí se hvězdička u jména souboru), je možné jej uložit. V případě, že jde o nový dokument, bude použito kódování Unicode. Jinak bude dokument uložen v původním kódování.

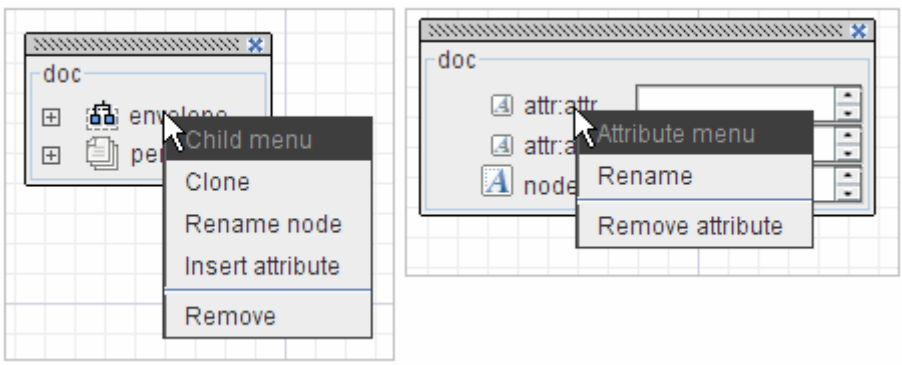
7.1.3.4 Editace obsahu

Editace obsahu elementů je velmi intuitivní. Pro práci se samotnými elementy jsou pak k dispozici různá kontextová menu. Při kliknutí pravým tlačítkem myši nad rodičovským elementem se zobrazí menu pro tento nadřazený element (viz obrázek 7.11). Menu umožňuje vložení nového synovského elementu nebo nového atributu a také odstranění celého elementu i s obsahem.



Obr. 7.11: Menu rodičovského elementu

Vlastní nabídku mají také jednotlivé synovské elementy a atributy. Příklad těchto menu je na obrázku č. 7.12. Pokud se jedná o element, menu umožňuje přejmenování elementu, odstranění a klonování elementu a vložení atributu. Pokud jde o atribut, lze jej přejmenovat nebo odstranit.

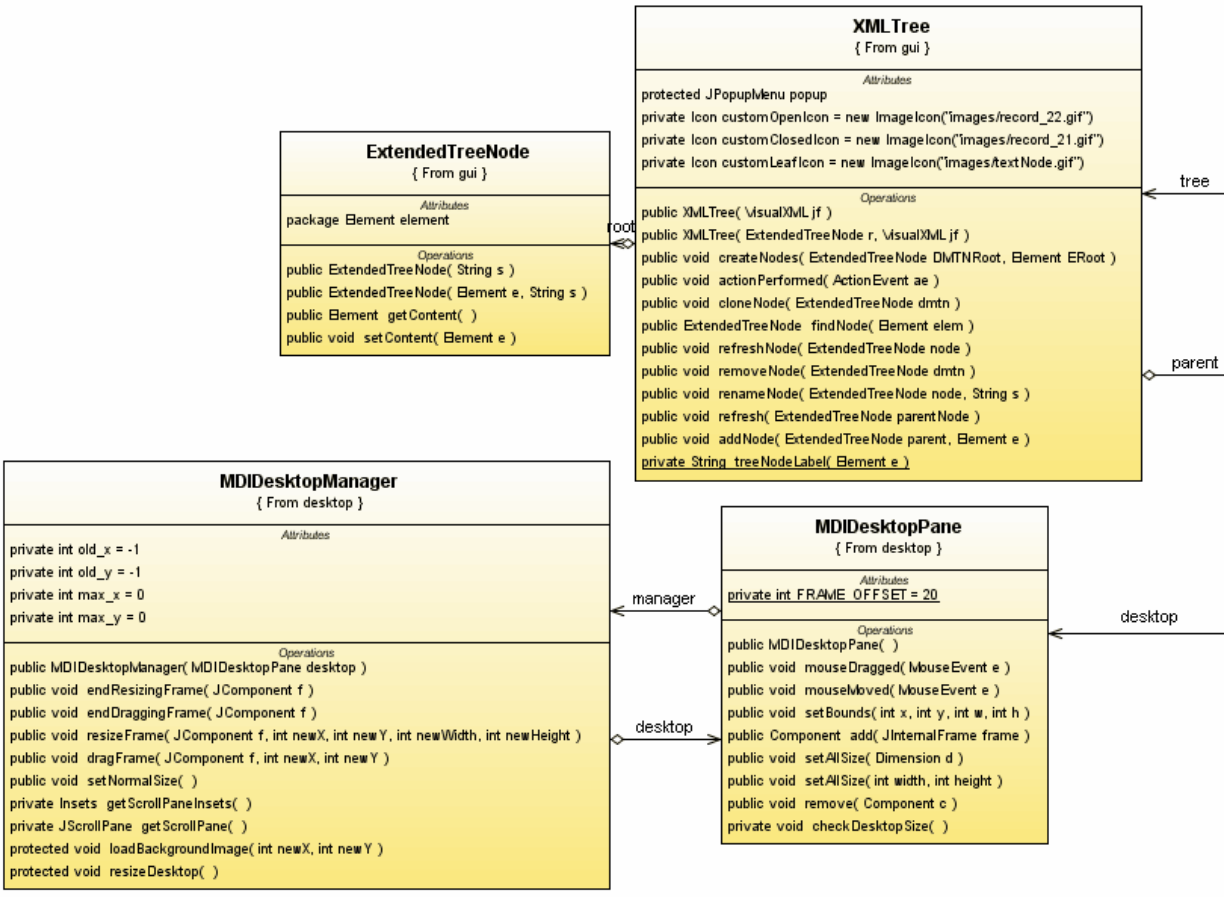


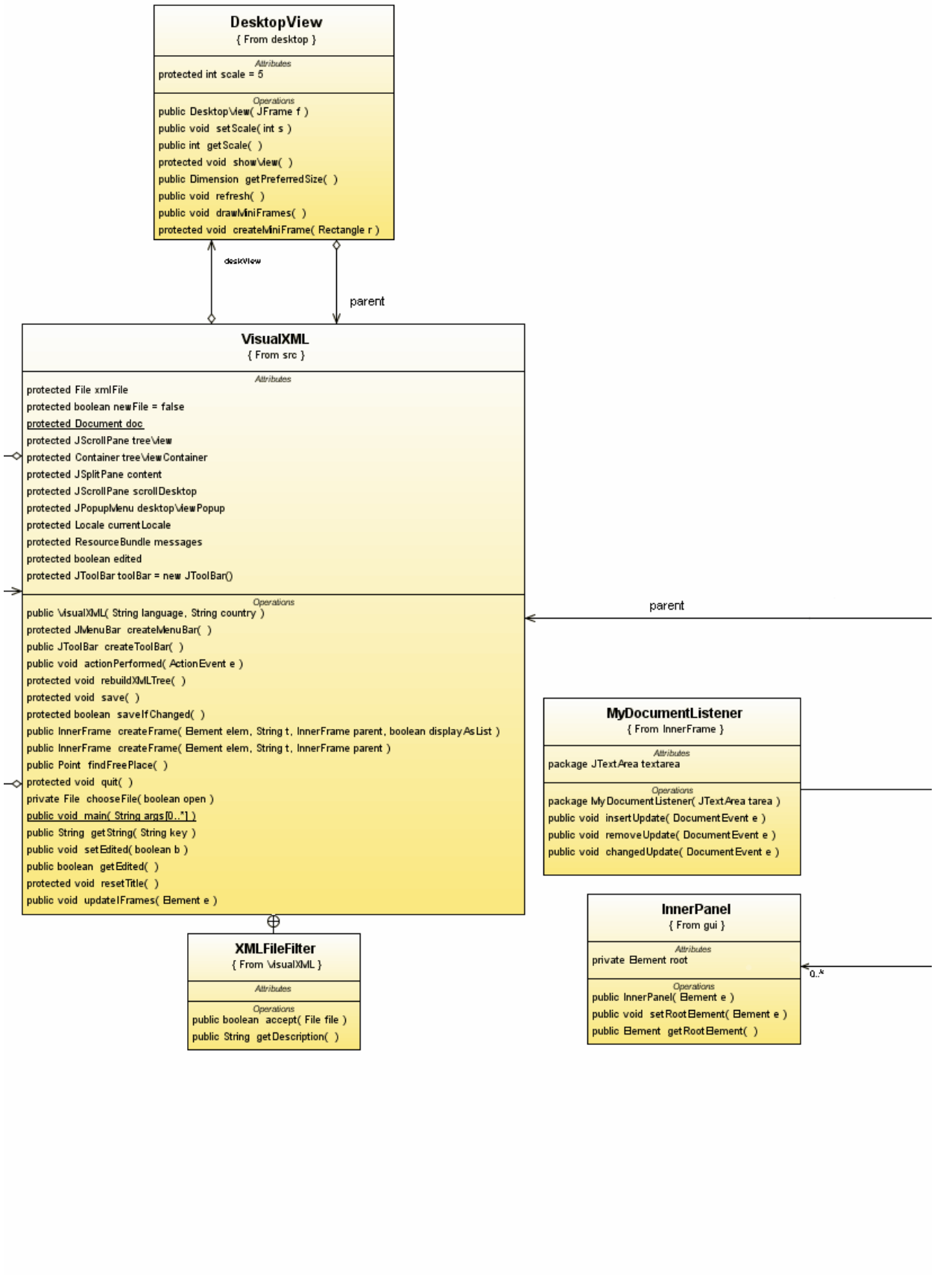
Obr. 7.12: Kontextová menu pro synovský element a atribut

7.2 Příloha 2: CD/DVD

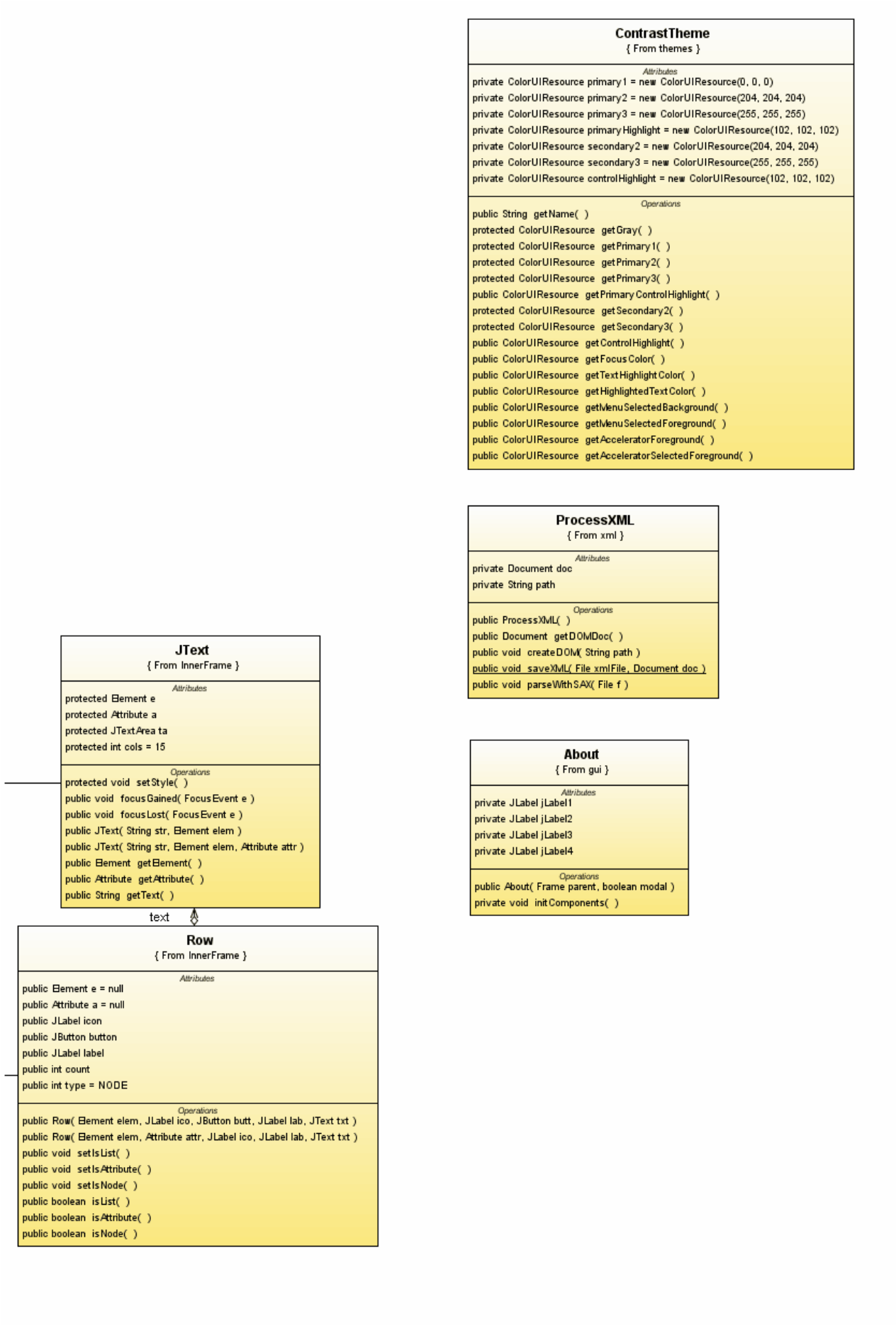
Na přiloženém CD se v kořenovém adresáři v souboru `bcprace.pdf` nachází text bakalářské práce. Ve složce `/src` jsou uloženy zdrojové kódy programu. V adresáři `/doc` je uložena programová dokumentace k programu. Zkompilovaný a spustitelný program se nachází ve složce `/dist`. V adresáři `/input` je pro demonstraci funkčnosti uloženo několik XML dokumentů.

7.3 Příloha 3: Diagram tříd




DesktopView
{ From desktop }
Attributes
protected int scale = 5
Operations
public DesktopView(JFrame f)
public void setScale(int s)
public int getScale( )
protected void showView( )
public Dimension getPreferredSize( )
public void refresh( )
public void drawMiniFrames( )
protected void createMiniFrame(Rectangle r)
deskView
parent
VisualXML
{ From src }
Attributes
protected File xmlFile
protected boolean newFile = false
protected Document doc
protected JScrollPane treeView
protected Container treeViewContainer
protected JSplitPane content
protected JScrollPane scrollDesktop
protected JPopupMenu desktopViewPopup
protected Locale currentLocale
protected ResourceBundle messages
protected boolean edited
protected JToolBar toolBar = new JToolBar()
Operations
public VisualXML(String language, String country)
protected JMenuBar createMenuBar( )
public JToolBar createToolBar( )
public void actionPerformed(ActionEvent e)
protected void rebuildXMLTree( )
protected void save( )
protected boolean saveIfChanged( )
public InnerFrame createFrame(Element elem, String t, InnerFrame parent, boolean displayAsList)
public InnerFrame createFrame(Element elem, String t, InnerFrame parent)
public Point findFreePlace( )
protected void quit( )
private File chooseFile(boolean open)
public void main(String args[0..*])
public String getString(String key)
public void setEdited(boolean b)
public boolean getEdited( )
protected void resetTitle( )
public void updateIFrames(Element e)
parent
MyDocumentListener
{ From InnerFrame }
Attributes
package JTextArea textarea
Operations
package MyDocumentListener(JTextArea tarea)
public void insertUpdate(DocumentEvent e)
public void removeUpdate(DocumentEvent e)
public void changedUpdate(DocumentEvent e)
XMLFileFilter
{ From VisualXML }
Attributes
Operations
public boolean accept(File file)
public String getDescription( )
InnerPanel
{ From gui }
Attributes
private Element root
Operations
public InnerPanel(Element e)
public void setRootElement(Element e)
public Element getRootElement( )
0..*


Vector2D
{ From InnerFrame }
Attributes
package Point p1
package Point p2
package double length
Operations
package \Vector2D(Point P1, Point P2)
public double getLength( )
public Point getStartPoint( )
public Point getEndPoint( )
public double getAngleInDegrees( )
public double getAngle( )
plFrame
InnerFrame
{ From desktop }
Attributes
private int NODE = 0
private int LIST = 1
private int ATTR = 2
private Element root
private SortedMap rowsGlobal[0..*]
public boolean isList = false
protected boolean displayAsList = true
protected int page = 1
protected int pageLimit = 5
protected int listLength = 1
protected JPopupMenu labelPopupMenu = null
protected JPopupMenu listLabelPopupMenu = null
protected JPopupMenu attrLabelPopupMenu = null
protected JPopupMenu panelPopupMenu = null
protected JPopupMenu filterPopupMenu = null
protected Element elementClicked = null
protected Attribute attributeClicked = null
Operations
public InnerFrame(\visualXML pFrame, Element r, String title, InnerFrame parentIFrame, boolean asList)
public void setDisplayList(boolean list)
public void setParentInnerFrame(InnerFrame p)
public InnerFrame getParentInnerFrame( )
public Element getRootElement( )
protected String getString(String key)
protected void generateGUI( )
protected JPopupMenu createLabelPopupMenu(boolean list)
protected JPopupMenu createPanelPopupMenu( )
protected JPopupMenu createAttrPopupMenu( )
protected JButton createButton(String action, String icons[0..*], String description)
public void itemStateChanged(ItemEvent e)
protected InnerPanel createPanel(Element node, int c)
public InnerFrame getParentFrame( )
public void drawLine( )
public void actionPerformed(ActionEvent e)
public void updateContent( )
public void killChildIFrames(InnerFrame iframe)
public Rectangle getPanelLocation(Element e)
line
Line
{ From gui }
Attributes
private Point start
private Point end
Operations
public Line(int init_x, int init_y, int last_x, int last_y)
public void redraw(int init_x, int init_y, int last_x, int last_y)
protected void paintComponent(Graphics g)
public Dimension getPreferredSize( )
public Dimension getMinimumSize( )
LabelMouseAdapter
{ From InnerFrame }
Attributes
Operations
public void mouseClicked(MouseEvent e)


ContrastTheme
{ From themes }
Attributes
private ColorUIResource primary1 = new ColorUIResource(0, 0, 0)
private ColorUIResource primary2 = new ColorUIResource(204, 204, 204)
private ColorUIResource primary3 = new ColorUIResource(255, 255, 255)
private ColorUIResource primaryHighlight = new ColorUIResource(102, 102, 102)
private ColorUIResource secondary2 = new ColorUIResource(204, 204, 204)
private ColorUIResource secondary3 = new ColorUIResource(255, 255, 255)
private ColorUIResource controlHighlight = new ColorUIResource(102, 102, 102)
Operations
public String getName( )
protected ColorUIResource getGray( )
protected ColorUIResource getPrimary1( )
protected ColorUIResource getPrimary2( )
protected ColorUIResource getPrimary3( )
public ColorUIResource getPrimaryControlHighlight( )
protected ColorUIResource getSecondary2( )
protected ColorUIResource getSecondary3( )
public ColorUIResource getControlHighlight( )
public ColorUIResource getFocusColor( )
public ColorUIResource getTextHighlightColor( )
public ColorUIResource getHighlightedTextColor( )
public ColorUIResource getMenuSelectedBackground( )
public ColorUIResource getMenuSelectedForeground( )
public ColorUIResource getAcceleratorForeground( )
public ColorUIResource getAcceleratorSelectedForeground( )
ProcessXML
{ From xml }
Attributes
private Document doc
private String path
Operations
public ProcessXML( )
public Document getDOMDoc( )
public void createDOM(String path)
public void saveXML(File xmlFile, Document doc)
public void parseWithSAX(File f)
JText
{ From InnerFrame }
Attributes
protected Element e
protected Attribute a
protected JTextArea ta
protected int cols = 15
Operations
protected void setStyle( )
public void focusGained(FocusEvent e)
public void focusLost(FocusEvent e)
public JText(String str, Element elem)
public JText(String str, Element elem, Attribute attr)
public Element getElement( )
public Attribute getAttribute( )
public String getText( )
text
Row
{ From InnerFrame }
Attributes
public Element e = null
public Attribute a = null
public JLabel icon
public JButton button
public JLabel label
public int count
public int type = NODE
Operations
public Row(Element elem, JLabel ico, JButton butt, JLabel lab, JText txt)
public Row(Element elem, Attribute attr, JLabel ico, JLabel lab, JText txt)
public void setIsList( )
public void setIsAttribute( )
public void setIsNode( )
public boolean isList( )
public boolean isAttribute( )
public boolean isNode( )
About
{ From gui }
Attributes
private JLabel jLabel1
private JLabel jLabel2
private JLabel jLabel3
private JLabel jLabel4
Operations
public About(Frame parent, boolean modal)
private void initComponents( )