ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA APLIKOVANÝCH VĚD
KATEDRA MATEMATIKY

Diplomová práce

Ukládání geodat do XML nativních databází

Plzeň, 2007 Zdeňka Albrechtová

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, vypracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem tuto diplomovou práci zpracovala samostatně, pouze s použitím pramenů, uvedených v seznamu, který je její součástí.

V Plzni, dne 25. května 2007 ……

Poděkování

Na tomto místě bych chtěla poděkovat všem, kteří mi pomáhali a podpořili při vypracování této diplomové práce.

Zvláště děkuji vedoucímu diplomové práce panu Ing. et Mgr. Otakaru Čerbovi.

Klíčová slova

XML, nativní XML databáze, geodata, dotazovací jazyky, XPath, XQuery, SŘBD

Abstrakt

Základní myšlenkou této práce je otestování možností využití nativních XML databází pro ukládání geografických dat. Po stručném seznámení se základy XML (XML, XPath, XQuery, XSLT, …), s nativními XML databázemi (druhy, základní charakteristiky, …) a s XML formáty geodat (GML, cGML, …) je na vybraných třech databázových systémech (4Suite, Berkeley DB XML, eXist) za pomoci šesti souborů s geodaty přímo ukázáno, zda by bylo v geoinformatice možné tento druh databází využít k řešení konkrétních úkolů.

Key words

XML, native XML database, geodata, query languages, XPath, XQuery, DBMS

Abstract

The basic idea of my work is possibility testing to use native XML databases for saving geographical data. After the brief acquaintance with the basics of XML (XML, Xpath, Xquery, XSLT, ...), native XML databases (kinds, basic charackteristics, ...) and XML formats (GML, cGML, ...) of geodata, it can be directly seen in six geodata files of three chosen database systems (4Suite, Berkeley DB XML, eXist) whether it is possible to use this kind of databases in geoinformatics for solutions of concrete projects.

Obsah

Seznam zkratek

ACID – *Atomic, Consistent, Isolated, Durale* – seznam požadavků na bezpečný transakční systém.

ANSI – *American National Standards Institute* – americká standardizační organizace.

API – *Application Programmers Interface* – rozhraní pro programování aplikací.

BLOB – *Binary Large Object* – datový typ užívaný v databázích nejčastěji pro ukládání audia, videa, obrázků.

BSD – *Berkeley Software Distribution* – operační systém vycházející z UNIXu.

CLOB – *Charakter Large Object* – datový typ užívaný v databázích nejčastěji pro ukládání textu.

CORBA – *Common Object Request Brooker Archive* – prostředí pro tvorbu objektově orientovaných aplikací; jazykově nezávislý objektový model.

DBMS – *DataBase Management System* – Systém řízení báze dat (SŘBD).

DOM – *Dokument Object Model* – objektový model dokumentu.

DTD – *Dokument Type Definition* – jazyk pro popis dokumentu.

FTP – *File Transfer Protocol* – protokol pro přenos souborů.

GCA – *Graphic Communications Association* – Asociace grafické komunikace.

GIS – *Geographic Information System* – geografický informační systém.

GNU GPL – *GNU General Public Licence* – licence pro volně dostupný software.

GUI – *Graphical User Interface* – grafické uživatelské rozhraní.

HTTP – *Hyper-Text Transfer Protocol* - hypertextový přenosový protokol.

IBM – *International Business Machines Corporation* – přední světová společnost v oboru informačních technologií.

InfoSet – XML Information Set – základní datový model XML dokumentu.

IPA – *Information-technology Promotion Agency* – japonská společnost zabývající se informačními technologiemi.

ITER – malá mezinárodní skupina vývojářů zabývajících se XML databázemi.

JDBC – *Java DataBase Connectivity* – SQL rozhraní pro Javu.

JDK – *Java Developer Kit* – soubor programů pro vývoj aplikací v Javě.

Mac OS – *Macintosh Operating System* – je označení operačního systému pro počítače Macintosh firmy Apple.

MODIS – *Management Of Data & Information Systems* – výzkumný a vývojový tým Institutu systémového programování ruské Akademie věd.

MS – *Microsoft* – softwarová firma.

ODBC – *Open DataBase Connectivity* – API pro přístup k databázovým systémům.

ODMG – *Object Data Management Group* – standard pro objektově orientované databázové systémy.

OGC – *OpenGis Consorcium* – mezinárodní neziskové konsorcium v oblasti GIS.

OQL – *Object Query Language* – dotazovací jazyk.

PHP – *PHP: Hypertext Preprocessor* – skriptovací programovací jazyk.

PLT – skupina lidí, která vytváří PLT Scheme, jenž obsahuje různé implementace programovacího jazyka Scheme.

POSIX – *Portable Operating System Interface* – přenositelné rozhraní pro operační systémy, standardizované jako IEEE 1003 a ISO/IEC 9945. Vychází ze systémů UNIX.

PSVI – *Post-Schema Validation Infoset* – datový model XML.

RAM – *Random Access Memory* – paměť s náhodným přístupem.

RDBMS – *Relational DataBase Management System* – SŘBD pro relační databáze.

RDF – *Resource Description Framework* – obecný mechanismus pro zápis metadat.

RELAX NG – *REgular LAnguage for XML Next Generation* – jazyk pro popis dokumentu.

RPC – *Remote Procedure Calling* – vzdálené volání procedur.

SAX – *Simple API for XML* – jednoduché API pro XML.

SDK – *Software Development Kit* – soubor vývojových nástrojů.

SGML – *Standard Generalized Markup Language* – značkovací jazyk.

SOAP – *Simple Object Access Protocol* – protokol používaný pro komunikací (používání XML zpráv).

SQL – *Structured Query Language* – dotazovací jazyk relačních databází.

TAX – *A Tree Algebra for XML* – algebra pracující se stromy XML.

UNIX – *Unary Information and Computing Service* – operační systém.

URI – *Uniform Resource Identifier* – jedinečná identifikace zdroje.

W3C – *World Wide Web Consorcium* – konsorcium pro vývoj webových standardů.

WebDAV – *Web-based Distributed Authoring and Versioning* – protokol pro vzdálený přístup.

WFS – *Web Feature Service* – definuje rozhraní mezi mapou a klientem.

X2QL – *eXtensible XML Query Language* – dotazovací jazyk pro XML.

XInclude – *XML Inclusions* – XML jazyk, který umožňuje vkládat XML dokumenty (nebo části) do XML dokumentů.

XLink – *XML Linking Language* – odkazovací jazyk pro XML.

XML – *eXtensible Markup Languague* – značkovací jazyk.

XPath – *XML Path Language* – jazyk pro identifikaci částí XML dokumentu.

XPointer – *XML Pointer Language* – odkazovací jazyk pro XML.

XQL – *XML Query Language* – dotazovací jazyk pro XML.

XQuery – *XML Query* – dotazovací jazyk pro XML.

XSLT – *eXtensible Stylesheet Language Transformations* – transformační stylový jazyk pro XML.

XUpdate – *XML Update Language* – jazyk pro aktualizaci XML dokumentu.

ZABAGED – *Základní báze geografických dat* - digitální geografický model území České republiky.

Seznam obrázků

Seznam tabulek

ÚVOD

V současné době se v oblasti geografických věd stále více uplatňují různé formáty značkovacího jazyka XML (např. GML, G-XML), a to především z důvodu jejich nenáročnosti na software. S rostoucím využitím těchto formátů se dá předpokládat, že data uložená tímto způsobem musí být někde uskladněna a spravována.

Jako i v jiných oblastech i zde se nabízí využití databázových systémů. Pro ukládání a spravování XML dat existuje speciální druh databázových systémů. Jedná se o nativní XML databáze, jejichž hlavní předností je přímé (nativní) ukládání XML dokumentů. Zda by bylo možné těchto databází využít také v oblasti geověd, se pokusím v této práci objasnit.

Vlastní obsah práce je rozdělen do dvou základních částí. V první části (jedná se o kapitoly 1, 2, 3 a 4) se může čtenář seznámit s problematikou značkovacího jazyka XML a s jeho technologiemi, se vztahem XML jazyka k databázím, s vlastními nativními XML databázemi a v neposlední řadě také s různými XML formáty geografických dat.

Druhá část, přibližně o stejném rozsahu, se již zabývá konkrétními nativními XML databázovými systémy (4Suite, Berkeley DB XML, eXist) a jejich možnostmi při správě náhodně vybraných XML souborů s geodaty. Mimo stručného popisu základních funkcí jednotlivých databází a ukázek správy dat, je vztah těchto databází ke geodatům ukázán především na testování atributových a prostorových dotazů, které jsou pro tato data typické.

1 ÚVOD DO XML

XML, neboli *rozšiřitelný značkovací*[1] *jazyk* (eXtensible Markup Language), byl vyvinut konsorciem W3C. Pro pochopení účelu tohoto jazyka se zaměřím na jeho název, protože právě ten vystihuje vše podstatné, co od XML můžeme očekávat.

Co je myšleno pod pojmem *rozšiřitelný* (eXtensible)? Jazyk XML je vysoce flexibilní a je tedy schopen přizpůsobit se jakémukoli účelu využití, např. v knihovnictví, matematice, hudbě, právě pomocí *značkovacího jazyka* (Markup Language). Základním principem značkovacího jazyka je označování dat pro člověka srozumitelnými značkami. Daty jsou myšleny textové řetězce a značkami textové značky, které tato data popisují, např. `<student>`, `<jmeno>`, `<prijmeni>`. XML se tak tedy stává ideálním formátem k ukládání strukturovaného a semi-strukturovaného[2] textu určeného pro šíření a publikaci na celé řadě médií. Popis struktury dokumentu zaručuje právě značkování, které ale již neříká nic o tom, jak má být ve výsledku dokument zobrazen. K tomuto určení se využívají stylové jazyky.

Z flexibility tohoto jazyka je zřejmé, že neexistuje žádná konečná množina předdefinovaných značek, což v některých případech může vést i k problémům či nepřehlednosti. Tomu se dá předejít využitím *jazyků pro popis dokumentů*, které určují, kdy a jak mohou být jednotlivé značky v dokumentu použity. Se schématem se pak konkrétní dokumenty porovnávají. Vyhovují-li, označujeme je jako platné (validní), v opačném případě jako neplatné. Platnost a neplatnost souvisí pouze s určitým schématem, a proto některé neplatné dokumenty mohou být i přes svoji neplatnost vůči konkrétnímu schématu formálně správné. Využití schématu je ryze dobrovolné. Více informací o schématech dokumentů je uvedeno v kapitole 1.5 (Jazyky pro popis dokumentů).

XML obsahuje mimo základní syntaktické specifikace další doplňkové standardy či pracovní verze týkající se dotazování (kapitola 1.6), odkazů (kapitola 1.7), výstupního formátování (kapitola 1.8), kódování matematických vzorců, chemických vzorců …

[1] Norma ČSN EN 28879 překládá slovo „markup“ jako „vyznačovací“.
[2] Data mají nepravidelnou strukturu a požadují uspořádání elementů.

Zdroje vztahující se k celé kapitole 1: [1], [3], [5], [23], [24], [33], [36], [40]

1.1 Historie XML

Jazyk XML rozhodně není prvním značkovacím jazykem. Již v 60. letech minulého století se objevila nutnost popisu textu podle jeho významu, nikoli podle výsledného vzhledu.

V té době existovaly dva možné způsoby formátování textu, a to tzv. specifické kódování, které mimo písmen, číslic a interpunkce obsahovalo speciální znaky určené pro formátování, a tzv. generické kódování, ve kterém se poprvé objevily pro popis vzhledu značky (tagy). U zrodu myšlenky generického kódování stáli nezávisle na sobě dva muži, William Tunnicliffe z GCA, který v září roku 1967 zavedl myšlenku rozdělení informačního obsahu od vlastního vzhledu dokumentu, a newyorský grafik Stanley Rice, který publikoval svojí ideu textové struktury tagů. Smysl těmto dvěma myšlenkám dal ředitel GCA Herman Scharpf prostřednictvím výboru zabývajícím se přímo generickým kódováním, jehož výsledkem bylo vytvoření konceptu GenCode. Dva základní poznatky z vývoje tohoto konceptu, a to: „pro různé druhy dokumentů je zapotřebí různých druhů kódů“, „menší dokumenty by mohly být jako elementy začleněny do větších“, byly využity v následujících letech (především v SGML).

V roce 1969 se americký vědec (původně právník) Charles Goldfarb účastnil výzkumného projektu na integrovaných informačních systémech v oblasti práv, v němž se mu spolu s Edwardem Mosherem a Raymondem Loriem podařilo vyvinout jazyk GML[1] (Generalized Markup Language). Také GML vycházel z myšlenek Tunnicliffa a Rice. Jednalo se o první jazyk, který formálně definoval typ dokumentu se stanovenou strukturou elementů (Dnes je často označován jako první známý značkovací jazyk.). Po dokončení GML pokračoval Goldfarb ve výzkumu, jeho další návrhy však byly začleněny až do jazyka SGML (Standard Generalized Markup Language).

Po úspěchu jazyka GML začala v roce 1978 americká standardizační společnost ANSI pracovat na vývoji nového obecného značkovacího jazyka (SGML), který by umožňoval uživatelům definovat si vlastní značkovací jazyk. Při vývoji se vycházelo především z již známého GML a také z GenCode. Ke konečnému výsledku se dospělo až v říjnu roku

[1] Často se uvádí, že zkratka GML vychází z počátečních písmen příjmení jednotlivých autorů tohoto jazyka (Goldfarb, Mosher, Lorie).

1985, kdy byla zveřejněna poslední pracovní verze SGML a v následujícím roce, po zapracování všech připomínek, byl vydán konečný standard SGML jako ISO 8879:1986.

Jazyk SGML byl však příliš obecný, a tak k průlomu ve značkovacích jazycích došlo až při vyvinutí jazyka HTML ze SGML pomocí DTD. Jazyk HTML má tedy pouze omezenou množinu značek určenou jen k popisu webových stránek, což pro jiné oblasti samozřejmě nebylo dostačující. Proto v roce 1996 začala práce na „jednodušší“ verzi SGML. Tato verze si ponechala většinu vlastností SGML, ale vynechala věci, které se v předchozích letech jevily jako příliš komplikované a nadbytečné (např. volba délky značky, volba oddělovače značek). Výsledkem byla 10. února 1998 první verze jazyka XML 1.0.

V současné době existuje již čtvrté vydání XML verze 1.0 vydané 29. září 2006 (předchozí vydání: 6. ledna 2000, 4. února 2004).

Souběžně s třetí edicí XML 1.0 vydalo konsorcium W3C doporučení XML 1.1, jehož druhé vydání vyšlo opět spolu se čtvrtým vydáním XML 1.0.

1.2 XML dokumenty

Do XML dokumentů lze ukládat data různého původu. V praxi se data ukládaná do XML rozdělují na dokumentově orientovaná (dokument-centric) a datově orientovaná (data-centric). V závislosti na uložených datech se pak vyskytují i dva druhy XML dokumentů.

1.2.1 Datově orientované XML dokumenty

Datově orientované XML dokumenty slouží především pro přenos dat. Většinou jsou určeny pro využití v jiných programech, zpracování lidmi se příliš nepředpokládá. Příklady takovýchto dokumentů: objednávky, faktury, jízdní řády, vědecká data (např. záznamy měření), …

Charakteristickým znakem těchto dokumentů je pravidelná struktura, co nejmenší logické jednotky dat v elementech a atributech a malý nebo žádný výskyt smíšeného obsahu.

1.2.2 Dokumentově orientované XML dokumenty

Dokumentově orientované XML dokumenty jsou dokumenty obvykle využívané pro lidskou potřebu. Velmi dobrým příkladem jsou knihy.

Vyznačují se méně pravidelnou, často až nepravidelnou, strukturou, hrubým členěním dat a velkým výskytem smíšeného obsahu. Nejčastěji jsou tyto dokumenty psány ručně přímo v XML nebo v jiných formátech, jako např. RTF, PDF, …, které jsou následně do XML převedeny.

1.3 Základy XML

Existují dva druhy značek, a to počáteční značka `<student>` a koncová značka `</student>`. Počáteční a k ní odpovídající koncová značka tvoří spolu s textem uvnitř těchto značek *element*. Jednotlivé elementy se mohou do sebe vnořovat. Pokud element neobsahuje mezi značkami žádný text ani další elementy, jedná se o tzv. *prázdný element* a je možné ho zapsat jako `<student/>`. Elementy v XML dokumentu musí splňovat několik pravidel:

- Každá počáteční značka musí mít koncovou značku.
- Elementy se nesmějí křížit.
- Musí existovat právě jeden *kořenový element* (např. `<studenti></studenti>` viz níže).

```
<studenti>
     <student>
          <jmeno>Pavel</jmeno>
          <prijmeni>Nový</prijmeni>
     </student>
     <student/>
</studenti>
```

V počáteční značce elementu se může vyskytovat libovolný počet *atributů*, tj. prvků, které upřesňují význam elementu. Atribut je tvořen názvem a hodnotou, která musí být uzavřena v uvozovkách či apostrofech. Element nesmí obsahovat dva atributy stejného názvu.

```
<student pohlavi="muž" typ_studia="prezenční">
```

Názvy prvků v XML dokumentu mohou obsahovat libovolné alfanumerické znaky, včetně třech interpunkčních, tj. podtržítko, pomlčka a tečka, přičemž nesmí začínat číslicí, tečkou či pomlčkou. Další použití interpunkčních znaků, jako uvozovka, apostrof, $, ^, % nebo středník, je zakázáno. Délka názvu může být libovolná, ale je nutné dbát na dodržování malých a velkých písmen.

Pokud XML dokument neodporuje výše uvedeným zásadám, lze jej vyjádřit stromovou strukturou, kde kořenový element je hlavním uzlem stromu a jeho podelementy tvoří dílčí uzly tohoto stromu.

Pro případ nutnosti jakýchkoli poznámek umožňuje standard XML vytváření *komentářů*. Komentář začíná znaky `<!--` a končí prvním výskytem znaků `-->`.

```
<!-- Toto je komentář. -->
```

Pokud by námi vytvářený dokument měl obsahovat například část kódu jiného značkovacího jazyka, jistě využijeme *sekci CDATA*. Vše, co se nachází mezi značkou `<![CDATA[` a značkou `]]>`, se považuje za holý znakový text.

```
<![CDATA[
     Připojení stylu k dokumentu:
     <?xml version="1.0" encoding=”windows-1250”?>
     <?xml-stylesheet href="mujstyl.xsl" type=”text/xsl”?>
     <dokument>
     .
     .
     </dokument>
]]>
```

V případě izolovaných znaků, které mají při značkování speciální význam (<, >, &, " a '), lze využít předdefinovaných entit, např. `<`, `>`, `&` (<, >, &).

Kdekoli v dokumentu (mimo značky) se mohou vyskytovat *instrukce pro zpracování* (procesní instrukce), které obsahují informace požadované konkrétní aplikací, která bude data XML zpracovávat. Často se využívají pro určení souboru stylů, který dokument formátuje.

```
<?xml-stylesheet href=“mujstyl.xsl“ type=“text/xsl“?>
```

Nepsaným pravidlem při vytváření XML dokumentů je uvedení *deklarace XML*. Deklarace XML vypadá jako procesní instrukce s názvem `xml`, která obsahuje atributy `version` (povinný – verze XML – např. „1.0“), `standalone` (volitelný – používání externích souborů – „yes“, „no“) a `encoding` (volitelný[1] – kódování – např. „ISO-8859-2“).

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

[1] „Volitelný“ je v tomto případě relativní pojem, pokud dokument nepíšeme v UTF-8 (popř. v UTF-16) musíme kódování v deklaraci uvést. Pokud totiž tento parametr neuvedeme, bude se předpokládat kódování „UTF-8“.

1.4 Jmenné prostory

Standard XML Namespaces (XML jmenné prostory) vznikl za účelem rozlišení elementů a atributů se stejným názvem pocházejících z různých aplikací XML, a také pro snazší rozpoznání souvisejících elementů a atributů.

Vlastní implementace spočívá pouze v doplnění příslušného prefixu před název elementu či atributu a v definování jmenného prostoru.

```
<... xmlns:prefix = „URI“>
.
.
.
<prefix:nazev_elementu>

...

<xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
version='1.0'>
<xsl:output ...>
```

1.5 Jazyky pro popis dokumentů

Jazyky pro popis dokumentů (schémové jazyky) definují, jaké elementy a atributy můžeme v XML dokumentu použít, co přesně mohou obsahovat a jak s nimi můžeme nakládat. Schématem si vlastně každý vytváří svůj vlastní XML jazyk, protože v něm nadefinuje své vlastní značky pro svůj vlastní účel.

Kontrolu platnosti XML dokumentů provádí tzv. validátory XML dat nebo také XML procesory.

V současné době patří mezi nejznámější schémové jazyky DTD (kapitola 1.5.1), XML Schema (kapitola 1.5.2), Relax NG či Schematron.

1.5.1 DTD (Document Type Definition)

Schémový jazyk DTD je mimo vlastní specifikace obsažen i ve specifikaci XML. Pochází přímo z jazyka SGML a stal se výchozím jazykem pro novější schémové jazyky. Je velmi jednoduchý, proto se stále těší velké oblibě. Pro složitější aplikace či pro přesnější definici struktury XML dokumentu je však již nepostačující – nepodporuje jmenné prostory ani datové typy.

```
<!ELEMENT studenti (student)+>
<!ELEMENT student ((jmeno,prijmeni)|(prijmeni,jmeno))>
     <!ATTLIST student pohlavi (muž|žena)#REQUIRED>
     <!ATTLIST student typ_studia CDATA #IMPLIED>
<!ELEMENT krestni (#PCDATA)>
<!ELEMENT prijmeni (#PCDATA)>
```

1.5.2 XML Schema

XML Schema je další specifikací konsorcia W3C, vyvinul se z XML-Data a XDR (XML Data-Reduced). Je to druhý nejpoužívanější schémový jazyk (hned po DTD).

Protože XML Schema vychází přímo z jazyka XML může využívat všech nástrojů vytvořených pro práci s XML dokumenty. Oproti jazyku DTD podporuje jmenné prostory (samotné schéma využívá vlastních elementů, proto musí být jmenných prostorů využito přímo ve schématu), a také datové typy (např. string, boolean, date), přičemž umožňuje i vlastní uživatelsky definované datové typy. K dalším výhodám tohoto jazyka patří využívání objektově-orientovaných rysů, definice elementů mnoha různými způsoby, apod.

```
<xs:schema xmlns:xs=“http://www.w3.org/2001/XMLSchema“>
  <xs:element name=“studenti“>
   <xs:complexType>
     <xs:sequence>
       <xs:element name=“student“ maxOccurs=“unbounded“>
        <xs:complexType>
          <xs:all>
           <xs:element name="jmeno" type="xs:string"/>
           <xs:element name="prijmeni" type="xs:string"/>
          </xs:all>
          <xs:attribute name=“pohlavi“ use=“required“>
           <xs:simpleType>
            <xs:restriction base=“xs:string“>
             <xs:enumeration value=“muž“/>
             <xs:enumeration value=“žena/>
            </xs:restriction>
           </xs:simpleType>
          </xs:attribute>
          <xs:attribute name=“typ_studia“ type=“xs:string“/>
        </xs:complexType>
       </xs:element>
     </xs:sequence>
   </xs:complexType>
  </xs:element>
</xs:schema>
```

1.6 Dotazovací jazyky

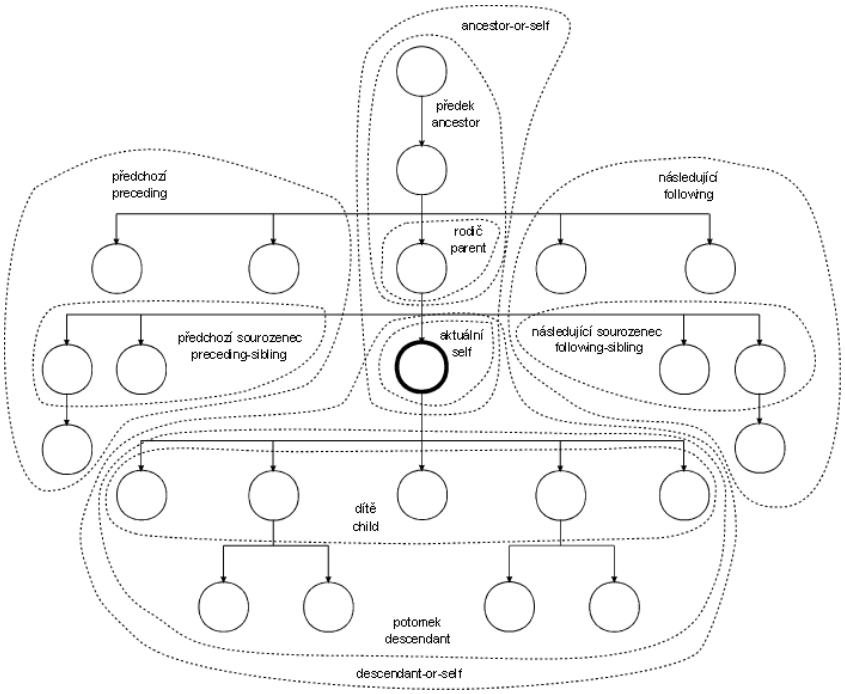
Základním principem těchto jazyků je získávání částí XML dokumentů za určitých podmínek, které pak mohou být využívány i dále, například se mohou tyto části podílet na vzniku nových dokumentů.

Od vzniku XML 1.0 se objevilo velké množství různě propracovaných dotazovacích jazyků, např. XML-QL (XML Query Language), XQL, X2QL, XPath (kapitola 1.6.1), XQuery (kapitola 1.6.2).

1.6.1 XPath

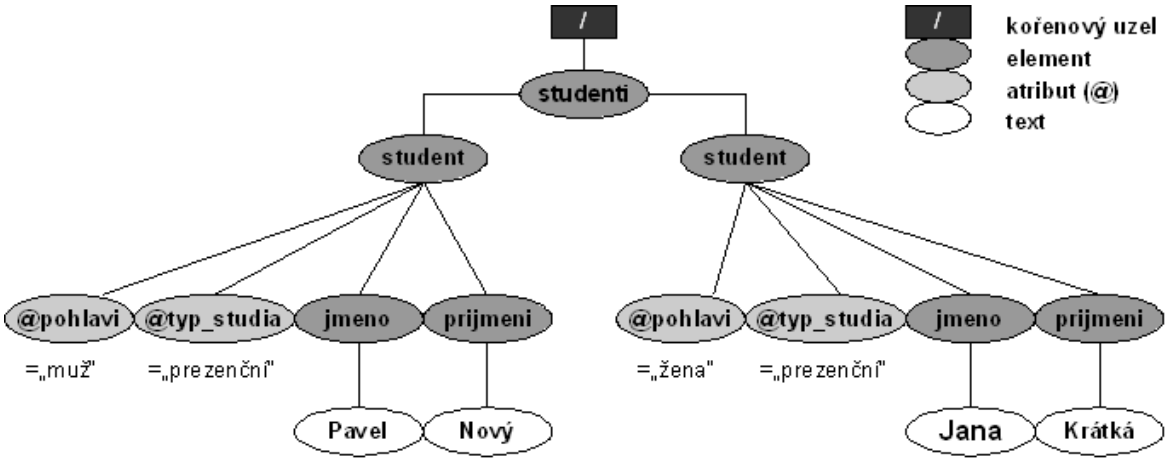
Jazyk XML Path Language (XPath) umožňuje adresování k jednotlivým částem dokumentu, a kromě toho poskytuje několik základních funkcí pro práci s řetězci, čísly, apod. Jedná se také o standard W3C konsorcia, v současné době ve verzi 1.0 a nově i 2.0. S jazykem XML toho příliš společného nemá, jeho notace je úplně odlišná, ale i přesto je tento jazyk hojně využíván v jiných XML jazycích (XPointer, XQuery, XSLT).

Datovým modelem XPath je strom, který obsahuje 7 typů uzlů: kořenový uzel, element, atribut, text, komentář, procesní instrukce, jmenný prostor. Kořenový uzel není totožný s uzlem kořenového elementu. Mezi uzly můžeme procházet po tzv. osách (obrázek 1.1), což jsou relace, které umožňují velikou variabilitu výrazů jazyka XPath.



Obrázek 1.1: Vztahy mezi uzly ve stromové reprezentaci XML dokumentu (převzato z [24])

Výsledkem vyhodnocení XPath výrazu může být množina uzlů, číslo, řetězec nebo booleovská hodnota.



Obrázek 1.2: Ukázka stromové struktury XML dokumentu

Na obrázku 1.2 je znázorněna stromová struktura XML dokumentu ukázkového příkladu, pro snazší pochopení některých příkazů uvedených níže.

```
/studenti/student - výběr všech elementů student, které jsou dětmi
                elementu studenti, který je kořenovým elementem
                dokumentu
/studenti/student[1] - výběr prvního elementu student, který je
                dítětem elementu studenti, který je kořenovým
                elementem dokumentu
student/@pohlavi - výběr atributu pohlavi elementu student, který
                je dítětem aktuálního uzlu
//student[@pohlavi='žena']/jmeno - výběr všech elementů jmeno,
                které jsou dětmi elementu student s atributem
                pohlavi="žena"
. - výběr aktuálního uzlu
* - výběr všech elementů, které jsou dětmi aktuálního uzlu
.. - výběr rodiče aktuálního uzlu
@* - výběr všech atributů aktuálního uzlu
//jmeno - výběr všech elementů jmeno
./prijmeni - výběr všech elementů prijmeni, které jsou dětmi
                aktuálního uzlu
*/prijmeni - výběr všech elementů prijmeni, které jsou vnoučaty
                aktuálního uzlu
self::prijmeni - výběr aktuálního uzlu, pokud je to element se
                jmenem prijmeni
preceding::*[1] - výběr posledního elementu, který se nachází před
                aktuálním uzlem
preceding-sibling::*[1] - výběr posledního elementu, který se
                nachází před aktuálním uzlem a je jeho sourozencem
following::*[1] - výběr prvního elementu, který se nachází za
                aktuálním uzlem
```

Funkce jazyka XPath jsou k dispozici na http://www.w3.org/TR/xpath-functions/.

1.6.2 XQuery

XQuery vznikl z experimentálního dotazovacího jazyka Quilt, jehož základní rysy vycházely z několika dalších jazyků, včetně XPath 1.0, XQL, XML-QL, SQL a OQL. Verze 1.0 jazyka XQuery velice úzce souvisí s verzí 2.0 jazyka XPath. Jedná se o velmi komplexní dotazovací jazyk, který mimo základních výrazů, které jsou známé již z jazyka XPath 1.0 (tj. cesty, množinová porovnávání, aritmetické výrazy, filtrující predikáty), zahrnuje pokročilejší konstrukce jako FLWOR[1] výrazy, třídění, podmíněné výrazy nebo konstruktory.

Mimo standardu datového modelu PSVI, který je striktním rozšířením datového modelu InfoSet, využívá XQuery i vlastního datového modelu, který definuje povolené hodnoty všech výrazů i podvýrazů XQuery 1.0, XPath 2.0 a XSLT 2.0. Mezi základní rozšíření vlastního datového modelu patří zejména možnost využití kolekcí dokumentů a složených hodnot, atomické hodnoty a podpora uspořádaných heterogenních posloupností.

Posloupnosti jsou základní jednotkou datového modelu. Mohou obsahovat dva základní druhy položek, a to: atomické (string, celé číslo atd.) a uzly definované svým druhem (kořen dokumentu, element, atribut, text, komentář, jmenný prostor, procesní instrukce), jménem a typem.

K obecným operátorům porovnání (`<`, `>`, `<=`, `>=`, `=`, `!=`), které definoval jazyk XPath 1.0, přidává XQuery spolu s jazykem XPath 2.0 ještě tzv. hodnotová porovnání v podobě operátorů `lt`, `gt`, `le`, `ge`, `eq`, `ne` („menší, „větší“, „menší nebo rovno“, „větší nebo rovno“, „rovno“, „nerovno“) a tzv. uzlové porovnání `is`. Pro porovnání pořadí uzlů v dokumentu se využívá operátorů `<<` („před“) a `>>` („následující po“). Logickými operátory potom jsou `and`, `or` a `not()`.

Pro snazší práci s daty má XQuery předdefinováno několik vstupních funkcí, a to: `doc()` – vrací dokument identifikovaný URI, `collection()` – vrací kolekci dokumentů, která je spojena s URI, `root()` – vrací kořen aktuálního dokumentu, `id()` – vrací posloupnosti uzlů náležící danému jménu, `idref()` – vrací posloupnost uzlů, které odkazují před ID – IDREF na uzly daného typu. Tyto funkce jsou často spojeny s cestou určující bližší výskyt uzlů, jednotlivé cesty samozřejmě vycházejí z XPath výrazů. Mimo vstupních funkcí lze využít při dotazování i velkého množství vestavěných funkcí (např. `min()`, `max()`,

[1] Vyslovováno „flower“.

concat(), string-length(), starts-with(), empty(), exists()) nebo funkcí, které si sám uživatel může definovat.

```
--- Deklarace funkce ---
define function název_funkce(vstupní_hodnoty)
  as výstupní_hodnoty
{
  .
  tělo funkce
  .
}

--- Volání funkce ---
nazev_funkce(vstupni_hodnoty)
```

Při dotazovaní si může uživatel vytvořit vlastní XML data, tento „nadstandard“ umožňují tzv. konstruktory, které jsou vymezeny složenými závorkami {}. Pro konstrukci dat lze použít také konstrukci typ_uzlu název {obsah}.

```
element osoba{
     attribute pohlavi {"muž"},
     element krestni_jmeno{"Jaroslav"},
     element prijmeni{"Bureš"}
  }
---
<osoba pohlavi="muž">
     <krestni_jmeno>Jaroslav</krestni_jmeno>
     <prijmeni>Bureš</prijmeni>
</osoba>
```

Velmi podstatnou konstrukcí jazyk XQuery jsou tzv. FLWOR výrazy, které jsou obdobou výrazu SELECT-FROM-WHERE z SQL. Název je složen z prvních písmen klauzulí:

- for – přiřazuje jednu nebo více proměnných jednotlivým výrazům,
- let – přiřazuje proměnným celé výsledky výrazů, ke kterým se proměnná vztahuje,
- where – funguje jako filtr na již získané výsledky,
- order by – třídí výsledky podle zadaného kritéria,
- return – umožňuje dát výsledkům určitou podobu.

```
<seznam_osob>
     {
     for $i in doc('pracovni.xml')//student
     where ($i/@typ_studia = "prezenční")
     order by $i/prijmeni
     return
          <osoba pohlavi="{ $i/@pohlavi }">
               <krestni_jmeno>{ $i/jmeno/text() }</krestni_jmeno>
               { $i/prijmeni }
```

```
            </osoba>
      }
</seznam_osob>
---
<seznam_osob>
      <osoba pohlavi="žena">
            <krestni_jmeno>Jana</krestni_jmeno>
            <prijmeni>Krátká</prijmeni>
      </osoba>
      <osoba pohlavi="muž">
            <krestni_jmeno>Pavel</krestni_jmeno>
            <prijmeni>Nový</prijmeni>
      </osoba>
</seznam_osob>
```

V klauzuli `where` je možné využít kvantifikátorů, a to jak existenčního (`some`), tak i obecného (`every`), což umožňuje ovlivnění výsledné filtrace výsledků. Oba kvantifikátory lze samozřejmě také nahradit funkcemi `exists()` resp. `empty()`.

```
some proměnná satisfies podmínka
<=>
exists (for proměnná where podmínka return 1)
---
every proměnná satisfies podmínka
<=>
empty (for proměnná where not(podmínka) return 1)
```

1.6.3 SQL/XML

Tento dotazovací jazyk není XML jazykem. Do této kapitoly jsem ho zařadila z důvodu jeho využívání při práci s XML dokumenty, a to při zpracování v relačních databázových systémech. Hlavním úkolem SQL/XML je poskytnout možnost vytvořit XML data z relačních dat, bohužel je využitelný pouze v již existujícím SQL prostředí.

Jak je patrné z názvu, vychází z jazyka SQL, vlastně se jedná pouze o rozšíření SQL 2003, umožňující práci s XML daty. Těmito rozšířeními jsou:

- funkce pro práci s XML,
- datový typ XML,
- mapovací pravidla – určují jak prezentovat SQL hodnoty jako XML data.

Funkce pro práci s XML jsou přímo aplikovatelné v SQL dotazu, jedná se o `xmlelement()` – vytvoří element se zadaným jménem, `xmlattributes()` – vytvoří atributy elementu ze sloupců tabulky, `xmlroot()` – vytvoří kořenový uzel dokumentu, `xmlcomment()` – vytvoří XML komentář, `xmlpi()` – vytvoří procesní instrukci, `xmlparse()` - zanalyzuje řetězec a vytvoří z něj XML strukturu, `xmlforest()`– vytvoří elementy ze sloupců

tabulky, `xmlconcat()` – sloučí více XML hodnot do jedné, `xmlagg()` – vytvoří kolekci elementů.

Podle [36] je implementace SQL/XML dostupná v aplikacích Oracle a IBM, Microsoft tento jazyk nepodporuje.

Studenti

jmeno	prijmeni	pohlavi	typ_studia
Pavel	Nový	muž	prezenční
Jana	Krátká	žena	prezenční

Tabulka 1.1: Příklad SQL/XML – relace Studenti(jmeno,prijmeni,pohlavi,typ_studia)

```
select xmlelement(„student“,
                  xmlforest(s.jmeno as krestni_jmeno,s.prijmeni))
from Studenti s
where s.pohlavi = „muž“;

---

<student>
     <krestni_jmeno>Pavel</krestni_jmeno>
     <prijmeni>Nový</prijmeni>
</student>
```

1.7 Jazyky pro odkazování

V současné době existují dvě specifikace jazyků přímo zaměřených na odkazování, a to XPointer (kapitola 1.7.1) a XLink (kapitola 1.7.2). Obě tyto specifikace se oddělily od původního odkazovacího jazyka XLL (eXtensible Linking Language), který definoval konstrukce pro tvorbu odkazů umožňujících propojení dokumentů XML do hypertextové sítě.

1.7.1 XPointer

Standard XPointer úzce souvisí se standardem XPath. Jeho hlavním úkolem je adresace na jednotlivé části XML dokumentu, na tyto části odkazuje právě pomocí jazyka XPath. Přičemž mimo základních datových typů XPath využívá ještě svých dvou, a to bodů (point) a rozsahů (range). Body umožňují přímou adresaci k elementům nebo jednotlivým znakům v dokumentu a rozsahy vybírají část dokumentu mezi dvěma body.

```
xpointer(/studenti/student/point[position()=2])
xpointer(range-to(//student/jmeno))
```

Pokud odkazujeme na objekty v jiných dokumentech, pak je výraz XPointeru přidán na konec URL za identifikátorem fragmentu (#).

```
http://.../x.xml#xpointer(/studenti/student[2])
```

1.7.2 XLink

Úkolem jazyka XLink je popis spojení mezi dokumenty.

Spojení, která odkazy XLink umožňují, mohou být od klasických jednosměrných, která známe z HTML (element `a`), přes dvousměrné až k vícesměrným. Typ spojení je definován atributem `type`, který může nabývat hodnot `simple`, `extended`, `locator`, `arc`, `title` a `resource`, z nichž asi nejvíce využívaným je typ pro vazby jednoduché (`simple`) a rozšířené (`extended`).

```
<jednoduchy_odkaz xlink:type="simple"
     xlink:href="http://www.w3.org/"> W3C </jednoduchy_odkaz>

<kniha xlink:type="extended">
     <nazev>Čaroděj ze země Oz</nazev>
     <autor> L. Frank Baum</autor>
     <edice xlink:type="locator" xlink:href="..."/>
     <edice xlink:type="locator" xlink:href="..."/>
     <edice xlink:type="locator" xlink:href="..."/>
</kniha>
```

1.8 Stylové jazyky (XSLT)

Protože nám samotný XML dokument nic neříká o tom, v jaké podobě se data v něm uložená mají zobrazit, bylo nutné získat nějaký nástroj, který by tento problém řešil. HTML na rozdíl od XML stylový jazyk přímo nepotřebuje, má již předdefinované značky, které formátování obstarají, ale i přesto pro tento jazyk byl vyvinut standard pro styly. Jedná se o jazyk CSS, který byl vytvořen roku 1996. Protože tento jazyk obsahuje vlastnosti pro aplikaci stylů na elementy mimo sadu HTML, může být využíván i v rámci XML. Pro XML však tento standard nebyl dostačující, a proto byl navržen formát XSL.

Během příprav standardu XSL byly vytvořeny jakési dvě jeho podmnožiny. Jednou je transformační jazyk XSLT (XSL Transformations), jehož pomocí lze vytvářet styly, které definují, jak se XML dokumenty mají převádět do formátu HTML, XHTML, do XML dokumentů s jinou strukturou nebo do obyčejných textových dokumentů. Druhá část, tzv.

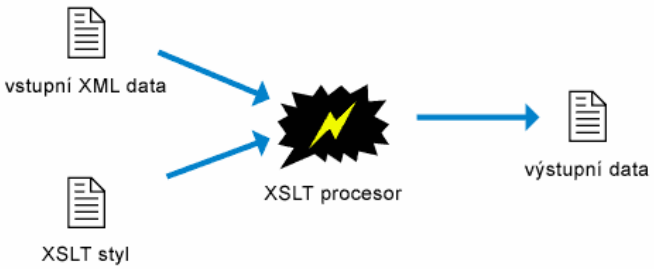
formátovací část, pak slouží k přesnému popisu vzhledu dokumentu a je označována jako XSL-FO.

Jazyk XSLT využívá také dalších XML jazyků jako jazyk XPath či XML Namespaces, ten je obzvláště důležitý pro možnost kombinace různých sad elementů zajišťujících smysl transformace XML dokumentu. Většinou je pro jmenný prostor používaný prefix `xsl`.

Základním stavebním kamenem stylového dokumentu jsou tzv. šablony, v nichž se pomocí XPath výrazu definuje místo (uzly) v původním XML dokumentu, které má být zpracováno. Tyto uzly jsou následně ukládány do výsledného stromu i s úpravami definovanými obsahem šablony.

```
<!-- deklarace šablony -->
<xsl:template match='XPath_výraz'>
   <!-- obsah šablony -->
</xsl:template>
```

Aby mohlo dojít k vlastní transformaci, musíme aplikovat na vstupní XML data a XSLT styl procesor, který vygeneruje výstupní data. K dispozici je dnes celá řada softwarových procesorů, které umožňují zpracování XML dokumentů na základě XSLT stylu, např. Saxon, Xalan, XT, MS XML. XSLT procesor na začátku své práce načte do paměti vstupní XML dokument a vytvoří si jeho stromovou strukturu. Tento strom je pak postupně procházen od kořene v pořadí, v jakém jsou elementy seřazeny v dokumentu, a pokud nalezne uzel odpovídající šabloně, začne ho zpracovávat. Potomci tohoto uzlu ale automaticky zpracováváni nejsou, pokud tedy chceme zpracovat i tyto potomky, musíme v šabloně použít instrukci `<xsl:apply-templates>`. Ta říká, že se má daná větev prohledávat dále a mají se pro její uzly hledat odpovídající šablony. Nepovinný atribut `select` nám umožňuje opět XPath výrazem specifikovat větev dokumentu ke zpracování.



Obrázek 1.3: Formátování XML dokumentu (převzato z [11])

Variabilita jednotlivých šablon je, dalo by se říci, téměř „bezmezná“. Uživatel si může nastavením v počátečním elementu šablony pomocí atributů určovat, kdy má být tato

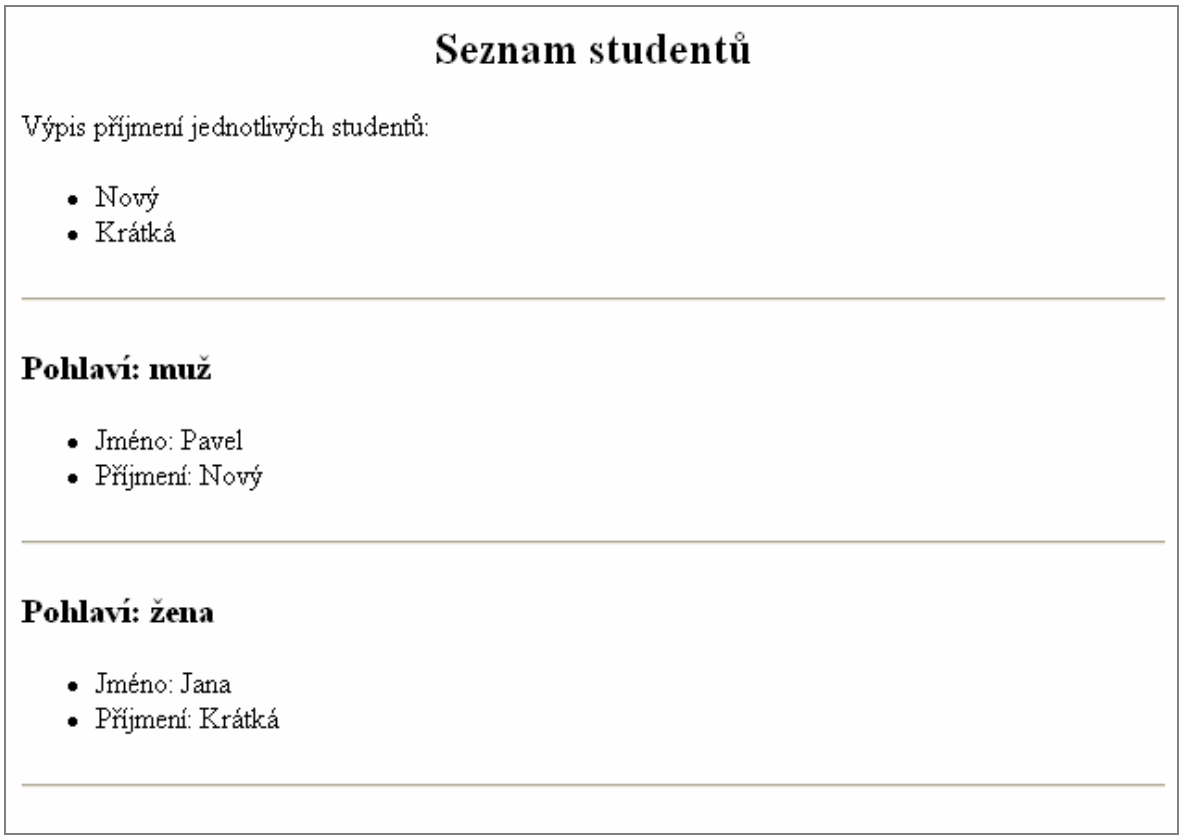
šablona volána, dokonce je možné využívat šablon jako funkcí (nahrazení atributu `match` v deklaraci šablony atributem `name` – volání funkce pomocí `xsl:call-template`). Což bylo velkou předností XSLT verze 1.0, poněvadž tato verze deklaraci funkcí neobsahovala, ale při tvorbě stylu bylo takových částí kódu často zapotřebí. Vzhledem k existenci funkcí pak již nikoho nepřekvapí možnost využívání cyklů (`xsl:for-each`), podmíněných zpracování(`xsl:if`, `xsl:choose`), řazení (`xsl:sort`), proměnných (`xsl:variable`) či parametrů (`xsl:param`).

I přes velký komfort, který verze 1.0 při tvorbě stylu umožňovala, měla řadu nedostatků, které měly být odstraněny v nové verzi a to verzi 2.0. Specifikace této verze vyšla 21. října 2006. Mezi základní zdokonalení patří jasná deklarace funkcí (není třeba využívat šablon), možnost využití datových typů, možnost výstupu do více souborů (to dříve umožňovaly pouze některé procesory, možnost seskupování apod.

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

  <xsl:output method="html" encoding="windows-1250" />

  <xsl:template match='/'>
    <html>
    <head>
      <title>Priklad</title>
    </head>
    <body>
      <h2 align="center">Seznam studentů</h2>
      <p>Výpis příjmení jednotlivých studentů:</p>
       <ul><xsl:apply-templates select="studenti/student"
mode="prijmeni" /></ul>
       <hr/>
      <xsl:apply-templates select="studenti/student"/>
    </body>
    </html>
  </xsl:template>

  <xsl:template match="student" mode="prijmeni">
    <li><xsl:value-of select="prijmeni"/></li>
  </xsl:template>

  <xsl:template match='student'>
    <h3>Pohlaví: <xsl:value-of select="@pohlavi"/></h3>
    <ul><xsl:apply-templates/></ul>
    <hr/>
  </xsl:template>

  <xsl:template match='jmeno'>
    <li>Jméno: <xsl:apply-templates/></li>
```

```
  </xsl:template>

  <xsl:template match='prijmeni'>
    <li>Příjmení: <xsl:apply-templates/></li>
  </xsl:template>

</xsl:stylesheet>
```

Výsledek po transformaci:



Obrázek 1.4: Výsledek XSLT transformace

1.9 Jazyk pro aktualizaci XML dokumentu (XUpdate)

Jazyk XUpdate je doposud jediným značkovacím jazykem, který umožňuje měnit obsah a strukturu již existujícího XML dokumentu. Na jeho vývoji od úplného počátku pracuje XML:DB[1].

[1] Iniciativa utvořená organizacemi: SMB Gesellschaft für Softwareentwicklung mbH (Lipsko), dbXML Group L.L.C, OpenHealth Care Group.

Jako všechny XML jazyky, je i XUpdate velice dobře strukturovaný. K výběru částí dokumentů, kterých se má týkat aktualizace, využívá XUpdate jazyka XPath. Principielně je tento jazyk velice podobný transformačnímu jazyku XSLT.

Celý XUpdate dokument je uzavřen do kořenového elementu `<xupdate:modification>`, z čehož vyplývá jasná podpora jmenných prostorů. Mimo jmenného prostoru specifikovaného v tomto elementu je zde ještě povinný atribut pro stanovení verze (v současné době verze 1.0). Kořenový element může obsahovat následující podelementy: `xupdate:insert-before` (vloží uzel před vymezený uzel), `xupdate:insert-after` (vloží uzel za vymezený uzel), `xupdate:append` (vloží uzel jako dětský podelement do vymezeného uzlu), `xupdate:update` (aktualizuje obsah uzlu), `xupdate:remove` (odstraní uzel), `xupdate:rename` (přejmenuje uzel), `xupdate:variable` (definuje proměnnou), `xupdate:value-of` (vrátí hodnotu vymezeného uzlu), `xupdate:if` (konstrukce pro podmíněné zpracování).

Vkládanými uzly mohou být elementy (`xupdate:element`), atributy (`xupdate:attribute`), prostý text (`xupdate:text`), procesní instrukce (`xupdate:processing-instruction`) či komentáře (`xupdate:comment`).

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
       xmlns:xupdate="http://www.xmldb.org/xupdate">

  <xupdate:insert-after select="/studenti/student[1]" >

     <xupdate:element name="student">
          <xupdate:attribute name="pohlavi">
               muž
          </xupdate:attribute>
          <xupdate:attribute name="typ_studia">
               prezenční
          </xupdate:attribute>
          <jmeno>Martin</jmeno>
          <prijmeni>Lhoták</prijmeni>
     </xupdate:element>

  </xupdate:insert-after>

</xupdate:modifications>
```

2 XML A DATABÁZE

Celá předchozí kapitola byla věnována značkovacímu jazyku XML, a proto nebudu v úvodu této kapitoly jazyk XML nadále popisovat, ale zaměřím se pouze na popis databází a nástrojů s nimi souvisejích.

Databáze je určitá uspořádaná množina informací (dat) uložená na paměťovém médiu počítače, která je spravována systémem řízení báze dat (SŘBD – z anglického DataBase Management System – DBMS). SŘBD zahrnuje softwarové prostředky umožňující definování struktury dat, ukládání dat, výběr dat, ochranu dat, komunikaci mezi uživatelem a systémem apod. Databáze spolu se SŘBD tvoří tzv. databázový systém, který bývá také často označován slovem „databáze“, a to hlavně tam, kde je z kontextu zřejmé, že se jedná o databázový systém nikoli o samotnou databázi. Ve své práci jsem pro označení databázového systému využívala obou těchto výrazů.

Na základě způsobu ukládání dat a vazeb mezi nimi se můžeme setkat s několika druhy databázových modelů, z nichž nejznámější jsou určitě relační databáze[1], které měly být nahrazeny v 90. letech minulého století databázemi objektovými[2], což se ale nezdařilo a došlo ke vzniku objektově relačních databází[3], tedy kompromisu mezi oběma těmito druhy databázových systémů. Za zmínku také stojí dva starší modely, a to síťové databáze[4] (první byla vytvořena roku 1965) a o něco mladší hierarchické databáze[5] (z počátku 70. let).

Z výše uvedeného popisu, co je databáze, by se dalo usoudit, že vlastní XML dokument je také databází, ale pokud by byl takovýto dokument označen za databázi, mohl by být databází jakýkoli jiný soubor zpracovávaný počítačem. XML dokument má ale pro toto označení daleko větší předpoklady, je totiž velmi dobře strukturovaný na základě značek, je platformě nezávislý, atd.

[1] Databázový systém založený na relačním modelu a relační algebře. Data jsou uspořádána do relací (tabulek), nad kterými lze provádět přípustné operace.
[2] Databázový systém založený na objektově orientovaném přístupu (OOP). Do databáze ukládá přímo objekty, poskytuje nástroje pro objektově orientované dotazy.
[3] Databázový systém, který rozšiřuje relační databázi o možnost práce s objekty.
[4] Databázový systém založený na síťovém modelu. Data jsou uspořádána jako uzly rovinného grafu.
[5] Databázový systém založený na hierarchickém modelu. Data jsou uspořádána do stromové struktury.

K databázi ale samozřejmě také náleží SŘBD. Může být XML se svými doplňkovými standardy a rozšířeními takovýmto systémem? Dalo by se říci, že ano: ukládání dat je zajištěno přímo XML dokumentem, strukturu dat definují jazyky pro popis dokumentu (DTD, XML Schema, …), výběr dat umožňují dotazovací jazyky (XPath, XQuery, …), komunikaci mezi uživatelem a systémem poskytují programová rozhraní pro XML (SAX, DOM, …). Bohužel má ale XML celou řadu nedostatků, pro něž nemůže být tento jazyk řazen k plnohodnotným SŘBD. Postrádá například efektivní ukládání dat, indexování, bezpečnost, transakce, integritu dat, víceuživatelský přístup, a další.

Pro malé množství dat, uživatelů a nízké požadavky na výkon může XML naprosto postačovat i jako databázový systém, při zvýšení nároků pouze s XML vystačit nelze. Tím ale XML nemusíme úplně odepisovat, pro uložení XML dokumentů můžeme využít jiné strategie: uložení s využitím systému souborů, uložení v relačním databázovém systému, uložení v objektově orientovaném či objektově relačním databázovém systému a přímé (nativní) uložení XML dat (viz kapitola 3 – Nativní XML databáze).

S databázemi souvisí i druhy XML dokumentů. Datově orientovaný XML dokument je svojí pravidelnou strukturou přímo předurčen k využití v databázovém systému (často jsou takové dokumenty přímo databázovým výstupem), a to především v databázových systémech nenavržených přímo pro zpracování XML (např. relační, objektově orientované databázové systémy). Dokumentově orientované XML dokumenty sice nejsou vhodné pro přímé ukládání do „ne-XML“ databází kvůli své malé strukturovanosti, ale s využitím nativních XML databázových systémů lze i těchto dokumentů v databázích využívat a zpracovávat.

Zdroje vztahující se k celé kapitole 2: [5], [6], [9]

2.1 Obecné způsoby uložení XML dokumentů

Jak již bylo uvedeno výše, existuje několik možností, jakým způsobem XML dokumenty spravovat. Všechny způsoby mají řadu výhod i nevýhod, ale jejich užití už záleží pouze na posouzení konkrétního uživatele v závislosti na spravovaných datech. Jasným favoritem pro ukládání XML dokumentů jsou samozřejmě nativní XML databáze, pro ně je ale vyhrazena celá hlavní kapitola (kapitola 3), proto se jim zde nebudu věnovat.

2.1.1 Uložení v systému souborů

Tento způsob je pro svoji nenáročnost a jednoduchost velice oblíbený, je založen pouze na správě souborů v systému.

Při dotazování obvykle dochází k vyhledání, načtení a analýze požadovaných XML souborů. Veškerá data jsou pak uložena do speciálních struktur (např. stromů W3C DOM).

Hlavními nevýhodami je nutná analýza celých dokumentů a uchovávání vytvořených stromů po celou dobu dotazování. Těmto nevýhodám lze zamezit vhodným indexováním, ale tím vznikne další (dost podstatná) nevýhoda, a to nemožnost aktualizace dokumentu. S využitím jiného druhu indexování, tzv. blokových stromů nebe klasických B-stromů, je i tento problém z části řešitelný, ale pak už tento způsob ztrácí svoji hlavní výhodu, tedy jednoduchost implementace.

2.1.2 Uložení v relační databázi

Protože se relační databáze často využívají, existuje možnost ukládat XML data i tímto způsobem.

Vhodnými kandidáty pro tento způsob uložení jsou datově orientované XML dokumenty. Po jejich převedení do relačních tabulek (s využitím mapovacích metod – viz kapitola 2.4) je s nimi možné nakládat obvyklými způsoby pro relační databáze.

Dotazování je v tomto systému samozřejmě možné, ale při složitých dotazech dochází již k obtížnější implementaci, v některých případech je dokonce nutné využít, mimo SQL, nějakého vyššího programovacího jazyka.

2.1.3 Uložení v objektové databázi

Při využití objektových databází se pohlíží na jednotlivé XML elementy jako na samostatné objekty, které jsou automaticky uloženy do databázového systému.

2.1.4 Uložení v objektově relační databázi

Objektově relační databáze spravují data v tabulkách shodně s relačními databázemi, mohou ale navíc aplikovat abstraktní datové typy[1], které umožňují „rozvětvit“ některé položky do bohatších struktur.

Abstraktních datových typů využívají objektově relační systémy i pro ukládání XML dat, nutná je však existence DTD – pokud není k dispozici, odvodí si ho systém sám. Definice typu dokumentu napoví systému, jak bude vypadat výsledná struktura tabulek pro uložení dat. Způsob převodu dat je obdobný uložení dokumentu do relační databáze.

2.2 Ukládání dokumentově orientovaných XML dokumentů

Základním požadavkem na ukládání těchto dokumentů je zachování XML dokumentu jako celku, tedy zachování pořadí elementů, komentářů, sekcí CDATA atd. Úroveň *round trippingu*[2] musí být značně vysoká, tj. výsledkem rekonstrukce uloženého dokumentu musí být dokument co nejpodobnější dokumentu původnímu (nejlépe dokument totožný).

V zásadě existují tři možné způsoby ukládání dokumentově orientovaných XML dokumentů:

- ukládání v systému souborů,
- ukládání jako BLOB (nebo CLOB) v relačních databázích,
- využití nativních XML databázových systémů (kapitola 3).

2.2.1 Ukládání v systému souborů

Ukládání pomocí tohoto systému je nejsnazším způsobem ukládání dokumentově orientovaných XML dokumentů, ale ne vždy je úplně výhodné. Hodí se především pro správu většího množství malých dokumentů. Dotazování je v tomto systému velice omezené, lze využít pouze standardního fulltextového vyhledávání (např. prohledávání obsahu souborů ve Windows).

[1] Vedle struktury obsahují také zapouzdřené funkce a operace.
[2] Výraz round tripping (v překladu „okružní jízda“) značí operaci uložení XML dokumentu do systému a jeho zpětnou rekonstrukci.

2.2.2 Ukládání v BLOB

Poněkud propracovanější je ukládání obsahu dokumentů do sloupce tabulky typu BLOB (či CLOB). Tento způsob už poskytuje některé výhody databázových systémů (kontrola transakcí, víceuživatelský přístup, …). Navíc, mnoho relačních databází má nástroje pro fulltextové vyhledávání, vyhledávání sousedů, duplicit, … a v případě databázových systémů podporujících XML se často stává situace ještě lepší - umožňují práci s obsahem dokumentu na vyšší úrovni než jen jako s prostým textem.

Také lze využít jednoduchého indexování, například prostřednictvím dvou tabulek, kde jedna tabulka obsahuje sloupec primárních klíčů[1] a sloupec typu BLOB a druhá tabulka sloupec s hodnotou indexů a sloupec cizích klíčů[2] odkazující na sloupec primárních klíčů první tabulky.

2.3 Ukládání datově orientovaných XML dokumentů

Z velmi dobré strukturovanosti datově orientovaných XML dokumentů vyplývá vhodnost pro využití v klasických databázových systémech. Základní myšlenkou je tedy ukládání a zpracování XML dat v relační (nebo objektově relační) databázi. Pro převod XML dat do tabulek se využívá tzv. *mapovacích metod.*

Na tento druh dokumentů se nejčastěji aplikují *databáze s XML rozšířením* (XML enabled database), tedy klasický SŘBD s rozšířeními, nebo se využívá *XML middleware*, což je samostatný software pro přenos XML dat mezi XML dokumenty a tabulkami vybrané databáze. Pro ukládání a zpracování dat v objektově relačních systémech jsou také velmi důležité *XML data binding* nástroje, které se zabývají mapováním XML dat na objekty vybraného objektově orientovaného jazyka.

Velkým usnadněním při zpracování je to, že není nutné zachovat dokument jako celek, tj. není důležité pořadí elementů, nezachovávají se komentáře, sekce CDATA atd. Úroveň *round trippingu* stačí nízká.

[1] Primární klíč je atribut nebo množina atributů, která jednoznačně určuje prvek v relaci (řádek v tabulce).
[2] Cizí klíč je atribut nebo množina atributů odkazující na jiný prvek v relaci (většinou jiné), tento prvek musí mít s tímto klíčem totožný primární klíč.

2.4 Mapovací metody

Při využití relačních nebo objektově relačních databázových systémů je prvním krokem převedení XML dokumentu do tabulek těchto systémů pomocí mapovacích metod.

Mapování je prováděno pouze nad elementy, atributy a prostým textem, nedochází tedy k uchovávání komentářů, procesních instrukcí apod., z čehož je zřejmé, že tabulky databází neobsahují všechny informace z XML dokumentu, a tak využití těchto metod není vhodné pro dokumentově orientované XML dokumenty, u nichž se vyžaduje jejich úplné zachování.

V základu existují tři typy mapovacích metod – *generické, schématem řízené* a *uživatelsky definované* mapování. Základním rozdílem mezi prvními dvěma typy mapování je využití XML schématu. *Generické mapování* mapuje XML dokument nezávisle na schématu, čímž umožňuje ukládat libovolné XML dokumenty, na rozdíl od *schématem řízených metod*, které využívají existující XML schéma pro definici databázového schématu a umožňují ukládat pouze validní dokumenty. Poslední typ – *uživatelsky definované mapování* – ponechává celý proces na uživateli.

2.4.1 Generické mapování

Jak jsem již uvedla výše, při generickém mapování lze do databáze uložit jakýkoli XML dokument, což ale nemusí být vždy pravda. Jediným omezením původního dokumentu může být výsledné databázové schéma. Toto omezení je skutečně využíváno, týká se pouze jedné, ale hojně využívané, skupiny metod generického mapování, a to *tabulkového mapování*. Další metody *Gengeric-tree mapping, Stucture-centred mapping* a *Simple-path mapping* už mohou opravdu uložit libovolný XML dokument.

Předpokladem pro využití *tabulkového mapování* je existence XML dokumentu, který přímo definuje databázové schéma. Tedy při náhledu na dokument je ihned zřejmé rozdělení příslušných dat do jedné či více tabulek. Například pro tuto tabulku:

Studenti

jmeno	prijmeni	pohlavi	typ_studia
Pavel	Nový	muž	prezenční
Jana	Krátká	žena	prezenční

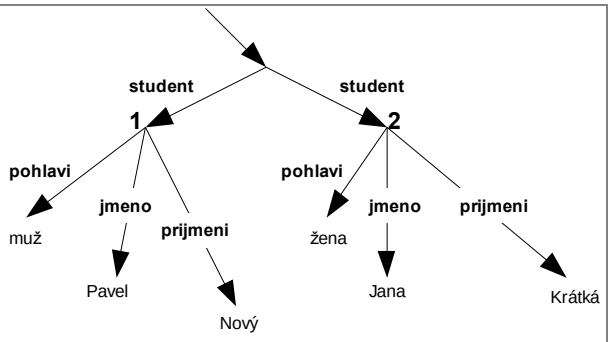
Tabulka 2.1: Výsledná tabulka po generickém tabulkovém mapování - příklad

měl XML dokument tuto strukturu:

```
<studenti>
     <student>
          <jmeno>Pavel</jmeno>
          <prijmeni>Nový</prijmeni>
           <pohlavi>muž</pohlavi>
           <typ_studia>prezenční</typ_studia>
     </student>
     <student>
          <jmeno>Jana</jmeno>
          <prijmeni>Krátká</prijmeni>
           <pohlavi>žena</pohlavi>
           <typ_studia>prezenční</typ_studia>
     </student>
</studenti>
```

Metoda *Generic-tree mapping* vidí vstupní dokument jako orientovaný strom, u něhož mají vnitřní uzly jednoznačné identifikátory a listy obsahují hodnoty atributů nebo textové obsahy jednotlivých elementů. Pomocí názvů elementů jsou označeny hrany stromu.

```
...
<student id="1" pohlavi="muž">
     <jmeno>Pavel</jmeno>
     <prijmeni>Nový</prijmeni>
</student>
<student id="2" pohlavi="žena">
     <jmeno>Jana</jmeno>
     <prijmeni>Krátká</prijmeni>
</student>
...
```



Na základě tohoto stromu je pak XML dokument ukládán do databáze různými způsoby, např.

- mapováním hran - ukládá všechny hrany stromu do jediné tabulky (jeden řádek tabulky obsahuje popis jedné hrany stromu),
- atributovým mapováním – snaží se udržovat pohromadě pouze data spolu související (každý název hrany má vlastní tabulku),
- univerzálním mapováním – také ukládá všechny hrany stromu do jediné tabulky (výsledná tabulka je shodná se sjednocením všech tabulek atributového mapování),
- normalizovaným univerzálním mapováním – zpracovává data obdobným způsobem jako předchozí mapování, ale pro každý název hrany obsahuje pouze jeden záznam, ostatní záznamy jsou uloženy ve speciálních tabulkách se stejnou strukturou jako tabulky atributového mapování.

Také další metoda, *Structure-centred mapping*, vychází ze zobrazení XML dokumentu jako orientovaného stromu, ale s jinou strukturou. Každý uzel stromu odpovídá čtveřici

`N = (t, l, c, n)`, kde `t` je typ uzlu, `l` název uzlu, `c` textový obsah a `n` seznam dětí. Získání poslední hodnoty je poněkud obtížné, a proto existuje několik variant tohoto algoritmu:

- využití primárních a cizích klíčů tabulek – každý uzel má přiřazen primární klíč a také cizí klíč odkazující na rodičovský uzel,
- DF (depth first) algoritmus – každý uzel je identifikován dvojicí čísel značící pořadí navštívení a opuštění daného uzlu při průchodu stromem do hloubky,
- SICF (simple continued fraction) algoritmus – každému uzlu je přiřazena jednoznačná SICF hodnota.

I poslední metoda, *Simple-path mapping*, vychází z orientovaného stromu, navíc ale předpokládá, že všechny dotazy nad uloženými daty budou v jazyce XPath. Strom obsahuje tři typy uzlů – elementové uzly (obsahují název a seznam dětí), atributové uzly (obsahují název a textovou hodnotu) a textové uzly (obsahují textovou hodnotu). Algoritmus pak vychází z myšlenky uložit do databáze přímo plné cesty (jednoduché cesty jazyka XPath – např. `/student/jmeno`) do všech uzlů grafu. Zároveň je ale nutné zohlednit pozici uzlu a jeho pořadí vzhledem k sousedním uzlům. Databázové schéma má tedy následující strukturu:

```
Element(ID_dokumentu, ID_cesty, pořadí, pozice)
Atribut(ID_dokumentu, ID_cesty, hodnota, pozice)
Text(ID_dokumentu, ID_cesty, hodnota, pozice)
Cesta(ID_cesty, text_cesty)
```

2.4.2 Schématem řízené mapování

Metody schématem řízeného mapování vychází z existence XML schématu, ze kterého se snaží načerpat maximální množství informací o ukládaných XML dokumentech. Před vlastním uložení validních dokumentů do databáze je nutné provést mapování XML schématu na databázové schéma.

Pro zdrojová schémata je nejčastěji využíván jazyk DTD, méně pak jazyk XML Schema, a pro cílová schémata relační schéma, popřípadě objektově relační schéma (ve spojení s XML Schema).

Obecné charakteristiky a společné principy:

- Podelementy s maximálním výskytem jedna jsou mapovány do tabulky nadelementu (tzv. inlining).
- Inlinované podelementy s nepovinným výskytem jsou mapovány na sloupce s prázdnými hodnotami.
- Podelementy s vícenásobným výskytem jsou mapovány do samostatných tabulek, kde vztahy element-podelement jsou mapovány pomocí primárních, cizích klíčů a referenční integrity[1].
- Alternativní podelementy jsou mapovány do samostatných tabulek, nebo do jediné univerzální tabulky s mnoha prázdnými hodnotami.
- Pokud je třeba zachovat pořadí sousedních elementů, je tato informace obvykle uložena do speciálního sloupce.
- Elementy se smíšeným obsahem obvykle vůbec podporovány nejsou. Důvodem je následek zkomplikování a zpomalení algoritmu.

Podle způsobu optimalizace existují dva základní typy metod – *fixní* a *flexibilní. Fixní* metody pracují pouze s daným XML schématem. Úkolem *flexibilních* metod je další optimalizace výsledného databázového schématu.

Hlavními představiteli fixního mapování využívající jazyk DTD jsou algoritmy *Basic*, *Shared* a *Hybrid*. Zástupcem fixní metody využívající XML Schema je algoritmus *Constrainst preserving*, jehož cílovým schématem je relační schéma. Obě tyto skupiny algoritmů využívají při mapování konstrukcí pomocných modelů – algoritmy *Basic, Shared* a *Hybrid* konstruují *DTD graf* a algoritmus *Constraints preserving* tzv. *EER model* (Extended Entity-Relationship Model), což je rozšířený ER model[2].

Ze zástupců flexibilních metod bych uvedla *LegoDB mapping* a *Hybrid object-relational mapping*.

2.4.3 Mapování definované uživatelem

Jak již vyplývá z označení poslední metody mapování, tj. *uživatelsky definované mapování*, je celý mapovací proces pouze v rukou uživatele, který si sám nejprve definuje

[1] Kontroluje správné použití cizích klíčů – zda k cizímu klíči existuje totožný primární klíč.
[2] Soubor pojmů, s jejichž pomocí se popisuje realita.

databázové schéma a pak pomocí nějakého rozhraní (např. dotazovací jazyk, anotace v XML dokumentech) specifikuje požadované mapování.

3 NATIVNÍ XML DATABÁZE

Nativní XML databáze jsou speciální databázové systémy přímo určené k ukládání XML dat. Také jako klasické databáze podporují transakce, zabezpečení, víceuživatelský přístup, dotazovací jazyky apod. Jediným rozdílem těchto databázových systémů je jejich vnitřní model, který je celý založen jen na XML.

Hlavní rysy nativních XML databází se dají shrnout do třech bodů:

- Objekty k uskladnění v nativních XML databázích jsou XML data.
- XML dokument je považován za základní jednotku uložení dat.
- Nativní XML databáze ukládá XML dokumenty v „nativní“ (přirozené) formě, ale nemusí se jednat o samostatný databázový systém.

Termín „nativní XML databáze“ (v originále „native XML database“) se poprvé objevil při marketingové kampani projektu Tamino, nativní XML databáze od Software AG, a hned po ní se celosvětově uchytil. Bohužel doposud neexistuje žádná formální definice „nativní XML databáze“.

XML dokumenty ukládané do nativních XML databází mohou být libovolné struktury, v praxi se však do těchto systémů ukládají častěji dokumentově orientované XML dokumenty, protože s využitím dotazovacích jazyků dostupných právě v těchto databázových systémech je možné rychle a jednoduše získat odpověď na téměř jakoukoli otázku. Dalším důvodem pro použití těchto systémů je zachovávání struktury, procesních instrukcí, sekcí CDATA, komentářů, ... , což databáze založené na XML (XML-enabled Database) neumožňují.

V současné době existuje velké množství nativních XML databází, a to jak komerčních, tak i nekomerčních, přičemž většina z nich má běžnou funkcionalitu:

- Všechny ukládají XML data na úrovni dokumentů a mnoho z nich tyto dokumenty seskupuje do tzv. kolekcí, nad kterými potom pracuje.
- Všechny podporují aspoň jeden dotazovací jazyk, většina podporuje XPath. Některé podporují i aktualizace.
- Téměř všechny využívají indexování nad kolekcemi dokumentů, nebo dokonce přímo nad elementy a atributy.

Příklady komerčních databází podle [9]:

Název databáze	Typ databáze	Vývojář/Tým vývojářů
eXtc	post-relační	M/Gateway Developments Ltd.
Infonyte	vlastní model	Infonyte GmbH
SQL/XML-IMDB	vlastní model, relační	QuiLogic Inc.
Tamino	vlastní model, relační (ODBC)	Software AG
X-Hive/DB	vlastní model	X-Hive Corporation

Tabulka 3.1: Komerční nativní XML databáze

Zástupci nekomerčních nativních XML databází jsou uvedeny v kapitole 3.3.

Zdroje vztahující se k úvodu kapitoly 3 a podkapitolám 3.1, 3.2: [9], [41]

3.1 Architektura

Architektura nativních XML databází se dělí do dvou kategorií: textově a modelově založené databáze.

3.1.1 Textové nativní XML databáze

Textově orientované nativní XML databáze ukládají XML dokumenty jako prostý text, a to např. do souborového systému, jako BLOB (nebo CLOB) v relační databázi nebo v jiném textovém formátu.

Společným znakem všech textově založených XML nativních databází je využívání indexů, které umožňují rychlejší vyhledání požadovaného místa dokumentu. Ke zrychlení dojde především při získávání celého XML dokumentu nebo jeho částí, protože při takových operacích stačí využít pouze jednoho indexu.

Obecně lze říci, že při získávání XML dokumentu nebo jeho jednotlivých částí ve stejné struktuře jako má původní dokument se dospěje k výsledku velice rychle, v opačném případě, pokud chceme dát výslednému dokumentu (či nějakému fragmentu) jinou strukturu, bude tato operace časově náročná.

3.1.2 Modelové nativní XML databáze

Druhá kategorie, modelově orientované nativní XML databáze, přistupuje k problému jiným způsobem. XML dokumenty neukládá jako text, ale vytváří jejich objektové modely, které následně ukládá do relačních či objektově orientovaných databází nebo do databází s vlastním modelem.

Modelově založené XML databáze vystavěné nad relačními, objektovými nebo objektově orientovanými databázemi mají s těmito databázemi srovnatelnou výkonnost. Zatímco databáze vystavěné nad vlastním databázovým modelem jsou srovnatelné s textově orientovanými nativními databázemi, tedy pokud načítají data ve stejné struktuře jako výchozí XML dokument.

Také modelově orientované databáze se setkávají s výkonnostními problémy při získávání a vracení dat v jiné formě, než ve kterém byly uloženy.

3.2 Základní charakteristiky XML nativních databází

3.2.1 Kolekce dokumentů

Většina nativních XML databází využívá při ukládání XML dokumentů tzv. kolekcí. Kolekce plní podobnou funkci jako tabulka v relačních databázích. V jedné kolekci mohou být uloženy XML dokumenty různého (nebo také žádného) XML schématu, odlišnost dokumentů může mít však za následek problémy s prováděním operací (např. vyhledávání, aktualizace, ...).

3.2.2 Dotazování a aktualizace

Dotazovací jazyky ke XML nativním databázím neodmyslitelně patří, každý takovýto systém podporuje alespoň jeden dotazovací jazyk. K nejčastěji podporovaným patří samozřejmě XPath, dále pak XQuery.

Aktualizaci ukládaných dokumentů řeší jednotlivé nativní XML databáze různými způsoby. Některé podporují jazyk XUpdate nebo rozšíření nad jazykem XQuery.

3.2.3 Transakce, zamykání a víceuživatelský přístup

Všechny nativní XML databáze podporují transakce. Zamykání je často pouze na úrovni celých XML dokumentů, než na jednotlivých částech dokumentu. Proto také víceuživatelský přístup může v některých případech způsobovat značné obtíže, záleží ovšem na stylu ukládání dokumentů – pro velké množství malých dokumentů může být zamykání transakcí na úrovni celých dokumentů optimální.

3.2.4 Aplikační programové rozhraní (API)

Téměř všechny nativní XML databáze dávají uživatelům možnost využít nějakého API (v některých případech i vlastního), které jsou obvykle ve formě ODBC rozhraní umožňující správu dat a metadat. Výsledky ze zpracování bývají často vráceny jako prostý text, v DOM stromu nebo v SAX.

V roce 2004 byly vyvinuty dvě XML API:

- XML:DB API od XML:DB – je programově nezávislý, jako dotazovací jazyk užívá XPath, podporuje XQuery
- JSR 225: XQuery API for Java (XQJ) – je založený na JDBC, jako dotazovací jazyk užívá XQuery

3.2.5 Round tripping

Všechny nativní XML databáze mohou zrekonstruovat dokument z původního dokumentu na úrovni elementů, atributů, sekcí CDATA a struktury. V jaké míře lze tento round tripping ještě zdokonalit, závisí pouze na jednotlivých databázích.

Obecně lze říci, že textově orientované nativní XML databáze jsou schopné dosáhnout nejvyšší možné úrovně round trippingu, tedy dokáží vrátit XML dokument totožný s původním. To už ale neplatí u modelově založených nativních XML databází, které jsou vhodnější pro ukládání datově orientovaných XML dokumentů, takže zde nemusí být úroveň round trippingu tak vysoká.

3.2.6 Indexování

Pro nativní XML databáze je typické využívání indexů ke zrychlení dotazování. Základní typy:

- hodnotové indexy – označují texty elementů nebo atributy (využití - např. „Najdi všechny elementy nebo atributy, jejichž hodnota je ‘muž’.”),
- strukturální indexy – označují umístění elementů a atributů (využití – např. „Najdi všechny elementy s touto adresou: ...“),
- full-textové indexy – označují jednotlivé znaky v textu a hodnotách atributů (využití – např. „Najdi všechny dokumenty, které obsahují slovo ‘muž’“).

Většina databází podporuje hodnotové a strukturální indexování, full-textovými indexy disponují jen některé nativní XML databáze.

3.2.7 Normalizace a referenční integrita

Základním principem *normalizace* je navržení takového databázového schématu, který zajistí, aby se data uložená v databázi zbytečně neopakovala – snižuje redundanci (nadbytečnost) dat. I v XML lze normalizaci aplikovat, například navržením správného XML schématu. Ne vždy je ale normalizace na místě. Při ukládání dokumentově orientovaných XML dokumentů by normalizace způsobila ztrátu smyslu využití takovýchto dokumentů, které mají ulehčovat práci svojí přesnou dokumentovou strukturou, i přes drobné redundance, jež často obsahují.

V relačních databázích *referenční integrita* zajišťuje, že cizí klíče vždy odkazují na existující primární klíče, v nativních XML databázích pak, že ukazatel v XML dokumentu ukazuje na validní dokumenty nebo jejich části. Ukazateli v XML dokumentech mohou být atributy ID nebo IDREF, pole typu key a keyref (definované v XML Schema), odkazy XLink nebo vlastní mechanizmy jednotlivých nativních XML databází. V nativních XML databázích existují dva druhy referenční integrity:

- integrita vnitřních ukazatelů – uvnitř jednoho XML dokumentu: Užívá mechanizmů ID, IDREF, key nebo keyref, lze ji tedy zajistit validací. Zajištění této integrity je možné ve většině nativních XML databází jen z části, protože často nedochází k validaci po aktualizaci dat, ale pouze při vkládání XML dokumentu.
- integrita vnějších ukazatelů – mezi XML dokumenty: Tato referenční integrita nebývá v nativních XML databázích podporována. V zajištění integrity mezi XML

dokumenty v rámci jedné databáze nejsou žádné překážky, problémem je integritu zabezpečit při odkazování mimo databázi, např. na nějakou URL adresu.

3.3 Nekomerční XML nativní databáze

Obsahem této kapitoly je stručné seznámení se zástupci nekomerčních XML nativních databází podle [9]:

Název databáze	Typ databáze	Vývojář/Tým vývojářů
4Suite	objektově orientovaná	FourThought
Berkeley DB XML	klíč - hodnota	Sleepycat Software
DBDOM	relační	K. Ari Krupnikov
dbXML	vlastní model	dbXML Group
eXist	vlastní model	Wolfgang Meier
myXMLDB	MySQL	Mladen Adamovic
Ozone	objektově orientovaná	ozone-db.org
Sedna XML DBMS	vlastní model	ISP RAS MODIS[1]
Timber	Shore, Berkeley DB	University of Michigan
XDBM	vlastní model	Matthew Perry, Paul Sokolovsky
Xindice	vlastní model	Apache Software Foundation
XpSQL	relační	Makato Yui

Tabulka 3.2: Nekomerční nativní XML databáze

Protože každá databáze je vytvořena jinými subjekty, nebylo možné přes veškerou moji snahu sehnat o všech výše uvedených databázích totožné informace (např. historie vývoje, maximální velikost zpracovávané databáze a ukládáných dokumentů,…), proto se obsah jednotlivých podkapitol v některých případech velmi odlišuje.

[1] Management Of Data & Information Systems, Institute for System Programming of the Russian Academy of Science.

3.3.1 DBDOM

Základním principem XML databáze DBDOM je aplikace DOM v SQL na relační databázi, a vzhledem k tomu, že se jedná o aplikaci pracující s XML, tak užívá RDBMS právě na XML.

DBDOM se od ostatních aplikací, které rovněž nějakým způsobem využívají DOM, liší především tím, že pomocí RDBMS dokáže ukládat data a zároveň uchovávat jejich vzájemné vztahy. Ostatní aplikace totiž pro stromovou strukturu DOM používají RAM a pro vlastní uložení dat pak textové soubory (flat soubory), které jsou vhodné pouze pro nevelkou manipulaci s daty a malé množství uživatelů.

Pro efektivnější práci s databází je možné i zde využít indexování a i další optimalizační techniky.

DBDOM dokáže využívat knihovny programovacího jazyka Java implementující DOM. Tato rozšíření zahrnují i JDBC a umožňují tak veškerou práci s daty pomocí SQL.

DBDOM může pracovat nad jakýmikoli RDBMS, zatím jsou k dispozici verze pro PostgreSQL, Oracle, DB2 a MS SQL-Server.

Tuto aplikaci vyvinul K. Ari Krupnikov z mezinárodní skupiny vývojářů ITER.

Zdroje: [19], [41]

3.3.2 dbXML

Databáze dbXML je nativní XML databáze, která je napsána v programovacím jazyce Java. Pro její užívání je tedy nutné mít nainstalované Java 2 SDK 1.4.2 a vyšší.

Vývoj dbXML začal koncem roku 1999, u zrodu stál Tom Bradford. V úplném počátku byl dbXML psán v jazyce C++ a předpokládalo se, že bude sloužit pouze k ukládání XML dokumentů. Teprve po roce se přešlo k programovacímu jazyku Java a k modelu klient/server. Na počátku roku 2001 byla verze 1.0 dbXML Core rozdělena na dvě části. První část byla přejmenována na Xindice a darovaná Apache Software Foundation. Druhá část pak byla ponechána dbXML Group, která z ní měla vyvinout komerční produkt. Což se také podařilo. Tato „komerční verze“ byla uveřejněna i jako Open Source za podmínek GNU General Public License (GPL).

V současné době tedy existují dvě verze tohoto programu, a to verze 1.0 a 2.0. Mezi oběma těmito verzemi jsou značné rozdíly, protože verze 2.0 je, dalo by se říct, úplným přepracováním verze předešlé. Celková architektura zůstala v podstatě stejná, a také některé základní API jsou si velmi podobné, ale z větší části je verze 2.0 úplně novým produktem.

Mezi hlavní novinky verze 2.0 patří:

- protokolování transakcí,
- tři typy zabezpečení správců,
    - NoSecurityManager – nezahrnuje žádné zabezpečení a dovoluje kterémukoli uživateli přístup k databázi,
    - SimpleSecurityManager – pro přístup k databázi vyžaduje uživatelské jméno a heslo,
    - DefaultSecurityManager – jediný uživatel (správce), který má přístup k databázi,
- webové služby,
- zlepšení funkcí příkazové řádky,
- jednoduché užívání GUI Administratora,
- nová databáze API,
- zlepšení dotazování.

Verze 1.0 používala pro komunikaci klient/server systém CORBA, novější verze už pro tuto komunikaci volí webové služby (především Labrador).

Pro práci s dbXML je možné využít třech typů rozhraní:

- GUI Administrator,
- rozhraní příkazové řádky,
- třídy API.

Dokumenty, které jsou spravovány dbXML, jsou řazeny do kolekcí. Systém dbXML se zaměřuje spíše na větší množství malých XML dokumentů, které jsou potom ukládány do jednotlivých kolekcí. Shodná struktura jednotlivých dokumentů v kolekcích není příliš důležitá. Samozřejmě, pokud bychom chtěli vytvořit spíše relační databázi, musely by této podmínce odpovídat i dané soubory, ale předností dbXML je, že právě struktura dokumentů může být v rámci jedné kolekce jakákoli. Úplně odlišná struktura dokumentů

se ale samozřejmě odrazí na pozdější kvalitě zpracování a dotazování. Jedna kolekce může souviset s různými indexy, rozšířeními, triggery a jinými kolekcemi.

Databáze dbXML podporuje rozšíření různými knihovnami Javy, ale i scripty psanými v JavaScriptu či Pythonu.

XML jazyky, které dbXML užívá, jsou: XPath, XSLT, XUpdate. K vyhledávání slov v dokumentu využívá fulltextové vyhledávání.

dbXML může pracovat v operačních systémech Mac OS (Mac OS X), Solaris, Linux, Windows.

Zdroje: [32], [39], [42], readme.txt

3.3.3 myXMLDB

myXMLDB je XML databázový server realizovaný pomocí MySQL. Ukládá dokumenty jako BLOB a může pracovat s dokumenty až o velikosti 256 MB. Pro podporu XPath a XQuery využívá procesor Saxon a poskytuje implementaci XML:DB API v Javě. Má grafické uživatelské rozhraní.

Práce s velkými dokumenty (většími než 8 MB) je pro tento systém jedinečná. Většina nativních XML databázových systémů, totiž používá DOM, což neumožňuje zpracování takto velkých dokumentů, protože je DOM velice náročný na paměť.

Zdroje: [26]

3.3.4 Ozone

Ozone je objektově orientovaný databázový systém vytvořený v Javě a distribuovaný GNU Library General Public License. Původní myšlenka vytvořit takovýto systém vznikla v druhé polovině devadesátých let minulého století jako multimediální studijní projekt od Falka Braeutigama. Oficiálně byl databázový systém Ozone poprvé předveden v březnu roku 1999 a ještě v červnu téhož roku začaly práce na vytvoření rozšíření pro tento systém nazvaném ozone/XML, které by umožňovalo spravovat XML data.

Základem ozone/XML je implementace DOM nad základní systém Ozone. Tato DOM implementace, nazývána jako MonsterDOM, je specifikována W3C DOM Level 1, a je

možné ji aplikovat prostřednictvím DOM API nebo jazyka XPath. Výhodou využití tohoto modelu je to, že využívá databázi k ukládání uzlů.

Poněvadž se jedná o objektově orientovaný databázový systém, pracuje Ozone s dokumenty jako s objekty a každý uložený soubor zpracovává v objektově orientovaném modelu.

Ozone není závislý na žádné jiné databázi, ani na mapovacích technologiích. Má pouze vlastní nástroje na ovládání klasických objektů Javy. V základní verzi, mimo implementace DOM kompatibilní s W3C specifikací, zajišťuje podporu ODMG 3.0 standardu. Pro práci s XML obsahuje podpůrné třídy pro softwarový procesor Xalan a XML parser Xerces.

Vzhledem k tomu, že Ozone využívá procesor Xalan, je pro dotazování využíván jazyk XPath, který je ale bohužel jediným dotazovacím jazykem využívaným v tomto systému.

Zdroj: [45], [30], [27], [10]

3.3.5 Sedna XML DBMS

Sedna je nativní XML databázový systém, který byl vyvinut úplně od počátku v programovacích jazycích C/C++ a Scheme[1]. Jedná se o plnohodnotný databázový systém, který podporuje dotazování, aktualizace, ACID transakce, zabezpečení, apod. Byl vyvinut týmem MODIS Institutu pro systémové programování ruské Akademie věd a je dostupný pod Apache License 2.0.

Při návrhu a zpracování měla Sedna dva základní cíle:

- systém musí být plně funkční – to znamená, že musí obsahovat všechny nástroje databázových systémů,
- systém musí být uzpůsoben práci s XML.

Z toho vyplývá těsné spojení databázového systému s jazykem XML.

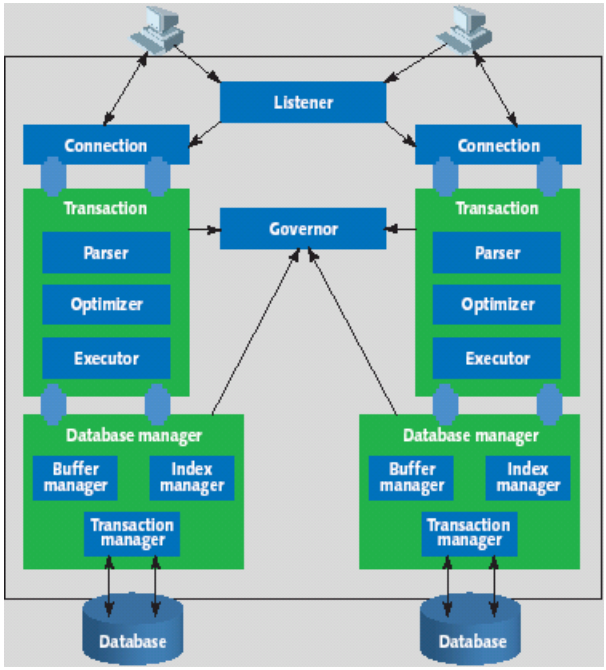
Základem pro realizaci se stal jazyk XQuery 1.0 a jeho datový model. Pro podporu aktualizace byl ještě použit jazyk XUpdate.

[1] Dialekt funkcionálního programovacího jazyka Lisp.

Architektura systému je znázorněna na obrázku 3.1, ze kterého je patrná úroveň jednotlivých součástí:

- regulátor (*governer*) slouží jako řídící středisko systému – registruje všechny zbývající součásti; ví s jakými databázemi se pracuje a dohlíží na provádění transakcí,
- posluchač (*listener*) buduje pro každého nového klienta složku připojení (*connection*) a nastavuje jejich vzájemné propojení. Při každém požadavku klienta začít novou transakci se vytvoří složka transakce (*transaction*). V této složce se uvádějí dílčí složky prováděných požadavků:
    - *parser* – přetvoří dotaz na logickou reprezentaci, což je strom podobný XQuery,
    - *optimizer* – přebírá logickou reprezentaci od parseru a formuluje plán vyřízení dotazu – strom nízkoúrovňových operací nad fyzickou strukturou dat,
    - *executor* – interpretuje plán vyřízení.

  Každý případ je v manažeru databáze převeden na jednotlivou databázi. Manažer databáze zahrnuje: *index manager* – sleduje indexy nad databází, *buffer manager* – odpovídá za vazby mezi diskovou a operační pamětí, *transaction manager* – zabezpečuje korektnost souběžně prováděných transakcí.



Obrázek 3.1: Architektura – Sedna XML DBMS (převzato z [28])

Organizace dat je řešena tak, aby docházelo k efektivnímu vyhledávání i aktualizaci. Pro popsání vztahů mezi blízkými uzly XML dokumentu (tj. rodič, dítě, sourozenci) jsou využívány přímé ukazatele. Pro celkový popis dokumentu slouží strom, tedy popisné schéma, které vzniká při ukládání a spočívá ve shlukování uzlů z XML dokumentu podle jejich pozic v popisném schématu dokumentu, které vychází z dat dokumentu a představuje stručný a přesný strukturovaný přehled (nejedná se o DTD, XML Schema, apod.).

Pro administraci je možné využít příkazové řádky.

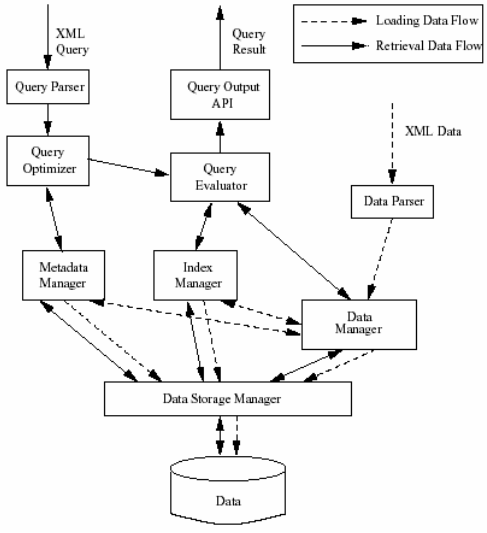
Programátoři mohou využít Java API, C API, Scheme API (Chicken, Gambit, PLT), PHP API, Python API, .Net API, OmniMark API a další (využití client/server).

Sedna umožňuje spojení s dtSearch a s Apache HTTP serverem. Byla navržena jako univerzální software neomezující se na určitou platformu. V současné době je dostupná v operačních systémech Windows (Windows XP, Windows 2000) a Linux (Linux x86 kernel verze 2.4 a vyšší).

Zdroje: [18], [28]

3.3.6 Timber

Tento nativní XML databázový systém vyvinula „Database Group“ z Katedry elektrotechniky a informatiky Michiganské univerzity. Projekt byl dotován IBM a National Science Foundatin (grant: IIS-9986030 a IIS-0208852).



Obrázek 3.2: Architektura – Timber (převzato z [46])

Základní myšlenkou projektu Timber[1] bylo vytvoření XML databáze co nejvíce podobné relační databázi. Jedná se tedy o systém, který využívá stejné základní charakteristiky jako relační databázový systém, včetně algebry, optimalizátoru dotazů a přístupových metod určených pro správné zhodnocení dat.

Relační algebru zde samozřejmě nebylo možné využít, a proto se přistoupilo k jakémusi ekvivalentu algebry pro XML, k algebře TAX, která manipuluje se soubory stromů. Základním problémem je ale komplikovaná a proměnná struktura stromů a potíže s organizací.

XML dokumenty jsou reprezentovány stromy a jsou řazeny do kolekcí, se kterými se následně pracuje.

Timber podporuje dotazování ve standardu XQuery 1.0. Každý dotaz je rozebrán do algebraické reprezentace a ta je následně zpracována optimalizátorem, jenž vybere vhodný plán dotazu a dotaz vyhodnotí.

Protože XQuery nedovoluje aktualizace, poskytuje Timber vlastní rozšíření k XQuery, např. mazání, modifikace, vkládání elementů, uzlů. Mimo těchto funkcí pro rozšíření XQuery Timber ještě umožňuje přidávat XML dokumenty do souboru, který je již uložený. Nutná znalost přesné struktury dokumentu pro zadání korektního dotazu (což je typické pro XQuery) také není podmiňující, byla totiž vytvořena funkce pro dotazování, která povoluje i omezené znalosti struktury XML dokumentu a přitom podává smysluplné výsledky.

Uživatel tohoto systému má možnost využití příkazové řádky, GUI a rozhraní pro přístup k Timberu přes web.

Jako základní DBMS užívá Timber Shore, nebo je také možné použít Berkeley DB.

Pro správné fungování systému je nutné mít:

a) Windows 2000 nebo Windows XP
b) Visual Studio 2002 nebo 2003

Zdroje: [46]

[1] Timber je ve skutečnosti akronym pro: **T**ree-structured native XML database **I**mplemented at the University of **M**ichigan by **B**right :-) **E**nergetic **R**esearchers.

3.3.7 XDBM

XDBM je databáze navržená speciálně pro práci s XML daty. Jako verze 1.0 byla poprvé předvedena v lednu roku 2000 společností Bowerbird Computing. Tato společnost spravuje XDBM a zároveň spolupracuje s externími vývojáři, kteří se podílejí na zdokonalování tohoto systému.

Předností tohoto systému je možnost využití ve vlastních softwarech. Je navržen tak, aby umožňoval snazší práci s XML soubory v aplikacích, které tento formát podporují. Lehčí práci zaručuje především:

- uložení předem zanalyzovaného XML souboru – formátovací program nemusí při práci načítat celý soubor,
- vytvoření spojového seznamu, který zajišťuje rychlejší a snadnější prohledávání,
- používání vyhledávacích funkcí, které umožňují uživatelům práci s libovolnými částmi XML dokumentu.

XDBM API je založeno na DOM standardu.

Bowerbird Computing vyvinul další aplikaci, která je využitelná pro XDBM. Jedná se o *freeDOM*, což je jakési rozhraní určené pro tento systém, které může být využíváno různými objektově orientovanými programovacími jazyky.

XDBM je určen pro operační systémy Linux a Windows (32bitové).

Zdroje: [22]

3.3.8 Xindice

Databázový server Xindice[1] byl od základu navržen k ukládání XML dat. Je velice flexibilní svým přístupem k datům – semi-strukturovaný přístup a model dat nezávislý na schématu.

Xindice navazuje na projekt dbXML Core. Zdrojový kód tohoto projektu byl roku 2001 darován Apache Software Foundation. V současné době je k dispozici ve verzi 1.0.

Programovatelná API podporují DOM a SAX.

[1] Správná výslovnost je [zýn-dý-čej].

Dokumenty se ukládají do kolekcí, které mohou být dotazovány jako celek. Je možné vytvořit kolekce, jenž obsahují jen dokumenty stejného typu, nebo kolekce ukládající všechny dokumenty společně. Kolekce se ukládají v hierarchii podobné souborovému systému UNIX či Windows. Pro dotazování nad kolekcemi dokumentů a dokumenty se používá XPath.

Za účelem zlepšení dotazování nad velkým množstvím dokumentů je k dispozici indexování elementů a atributů, což může velice podstatným způsobem snížit čas potřebný k vykonání dotazu.

Také změna dat v kolekcích dokumentů, ani v dokumentech jednotlivých, není pro Xindice neznámou, pro tento účel využívá XUpdate.

Xindice server může být zpřístupněný buď pomocí API, nebo z příkazové řádky pomocí příkazů. V současné době Xindice nabízí tři možnosti API, které je možné využít k vývoji aplikací:

- XML: DB API – pro aplikace psané v jazyce Java; jedná se o koncept z 7. května roku 2001,
- Xindice XML-RPC API – pro zpřístpunění Xindice jiným jazykům než je Java,
- Core Server API – vnitřní Java API pro jádro databázového systému. Toto API je užívané pro vybudování XML-RPC API a pro vložení XML:DB API rozšiřující funkčnost Xindice.

Příkazová řádka má všechny nástroje, které poskytuje API.

Zdrojový kód Xindice je napsán v Javě. Systém by měl tedy pracovat ve všech operačních systémech podporující JDK. Oficiální testování úspěšně proběhlo na těchto operačních systémech: Linux, Windows (Windows NT), Solaris, Mac OS (Mac OS X).

Zdroje: [38], readme.txt

3.3.9 XpSQL

XpSQL je multifunkční XML databázové prostředí užívající PostgreSQL. Tento systém rozkládá XML dokumenty na části a PostgreSQL užívá k vytvoření relací mezi těmito částmi.

Pro zpřístupnění dokumentů využívá:

- funkce DOM ke konstrukci a prozkoumání XML dokumentů,
- XPath k získávání informací z dokumentů,
- aktualizační funkce pro modifikaci dokumentů.

Základní projekt byl sponzorovaný IPA v Japonsku roku 2002. Později byl vydaný pod licencí GNU GPL.

Zdroje: [17]

Pozn. Databáze 4Suite, Berkeley DB XML a eXist budou popsány až v příslušných podkapitolách kapitoly 5 (Ukládání geodat do XML nativních databází), protože právě tyto tři databázové systémy byly vybrány jako vzorové pro ukládání geodat do XML nativních databází a tedy i podrobněji prozkoumány (instalace, funkce, atd.).

4 GEODATA A XML

V oblasti geografických věd se pod pojmem geodata (nebo též geografická data, geoprostorová data) rozumí data zahrnující zároveň popisnou (atributovou) a prostorovou složku popisující objekty reálného světa. Přesná definice podle ČSN EN ISO 19109 zní: *„Geografická data jsou data s implicitní nebo explicitní referencí k místu vztaženému k Zemi.“*

Geodata jsou zpracovávána v geografických informačních systémech (GIS), jejichž hlavním úkolem je správa a následné zpracování těchto dat (např. analýzy dat, tvorba map).

Již v polovině 80. let minulého století došlo k velkému rozmachu GIS v oblasti komerční sféry. A jak tomu bylo i v jiných oblastech, začala se i oblast geografických věd potýkat s nekompatibilitou jednotlivých formátů GIS aplikací. Po vzniku jazyka XML bylo zřejmé, že právě tento jazyk může problémy s nekompatibilitou vyřešit. Proto hned po vydání prvního standardu začaly některé instituce (např. OGC) pracovat na vývoji formátů umožňující spravovat geografická data.

Asi nejznámějším XML formátem pro správu geodat je GML (kapitola 4.1), z dalších bych uvedla cGML (kapitola 4.2), G-XML (kapitola 4.3), LandXML (kapitola 4.4), SOTF (Spatial Object Transfer Format), …

Zdroje: [7], [8]

4.1 Geography Markup Language (GML)

V roce 1994 bylo za účelem definování jednotného výměnného formátu pro práci s geodaty založeno OpenGIS Consorcium (OGC). Velký rozmach využívání jazyka XML, jenž je pro jednotný výměnný formát velmi vhodný, vedl konsorcium k vytvoření značkovacího jazyka, který by umožňoval na bázi značkování správu geodat. Tímto jazykem se stal Geography Markup Language (GML). Obecně lze říci, že GML je XML aplikace určená pro modelování, transport a ukládání geografických informací, které zahrnují mimo geometrie také vlastnosti geografických prvků.

Od verze 2.0 je GML napsán v XML Schema, první verze GML 1.0 z 20. dubna 2000 byla definována pomocí DTD.

Do verze 3.0 byla geometrie omezena jen na tzv. jednoduché prvky (Simple Features), jež byly definovány pouze dvourozměrnými souřadnicemi a křivky byly vytvářeny lineární interpolací. Pro znázornění skutečného světa jednoduché prvky dostačovaly. Od verze 3.0 se však setkáváme mimo jednoduchých 2D lineárních prvků s prvky komplexními, nelineárními, s 3D geometrií, s prvky s 2D topologií, s časovými vlastnostmi, s dynamickými prvky, měřením, ... Rozvoj GML je patrný i ze schémat GML verze 3, která jsou osmkrát větší než základní schémata GML verze 2. Původními schématy GML byly *geometry.xsd* (zahrnuje geometrické složky), *xlink.xsd* (poskytuje XLink vlastnosti pro odkazování mezi elementy nebo dokumenty), *feature.xsd* (definuje jednotlivé prvky). Tato tři základní schémata byla nahrazena schématy jako *gml.xsd*, *observation.xsd*, *dynamicFeature.xsd*, *topology.xsd*, *coordinateReferenceSystems.xsd*, *feature.xsd*, *valueObjects.xsd*, *grids.xsd*, *geometryComplexes.xsd*, *datums.xsd*, a dalšími.

Protože je GML značkovacím jazykem, nepopisuje vlastnosti pro zobrazení. Pro vytvoření grafického výstupu je nutné GML soubor transformovat stylovým jazykem XSLT, ve kterém si sám uživatel definuje v jaké formě dojde k zobrazení – zda bude zobrazení grafického rázu ve formátu SVG, X3D (eXtensible 3D Graphics) či VML (Vector Markup Language) nebo bude zobrazení v textové podobě ve formátu HTML, RTF či PDF.

Jazyk GML je hojně využíván po celém světě např. ve Velké Británii, v Nizozemí, v Kanadě - ani Česká republika není výjimkou (Český úřad zeměměřický a katastrální poskytuje v tomto formátu data ze ZABAGED – [15]). Mnohdy se stal i základním stavebním kamenem dalšího jazyka určeného pro práci s geografickými daty.

Zdroje: [14], [16], [44]

4.2 compact Geographical Markup Language (cGML)

cGML je XML specifikace vytvořená CRS4[1] a zaměřená na poskytování digitálních map na malých bezdrátových zařízeních. Tento jazyk byl testován na J2ME zařízeních (Nokia

[1] Center for Advanced Studies, Research and Development in Sardinia.

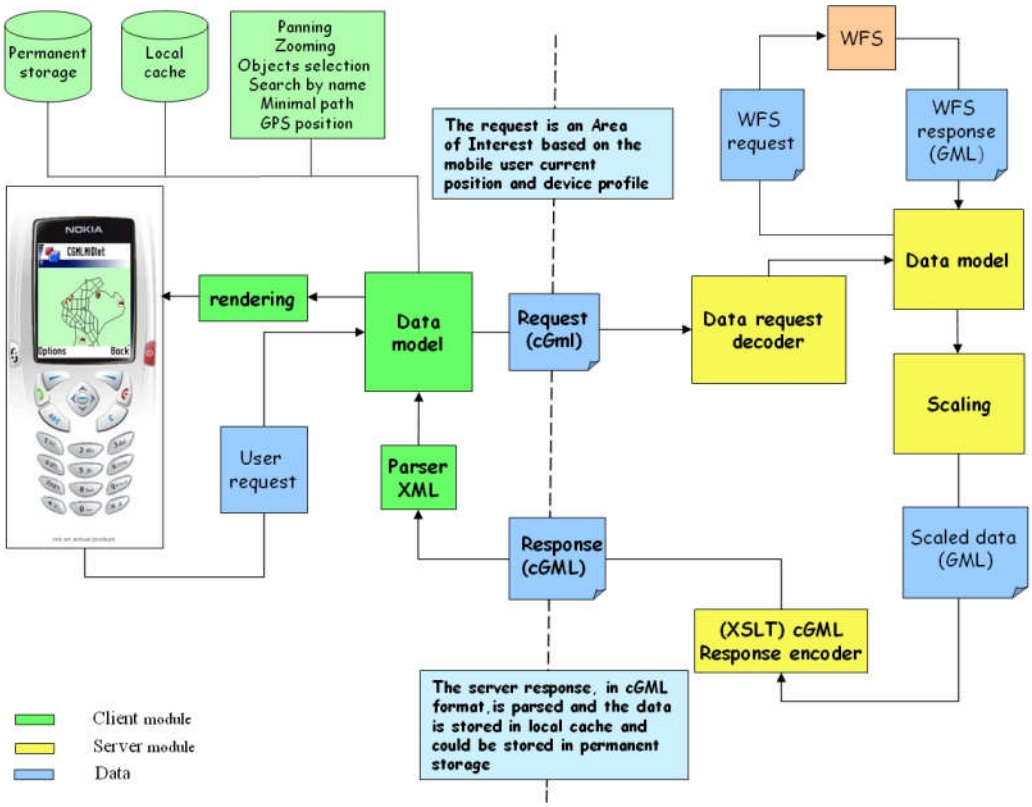
7650, Nokia 3650, Nokia 6600 a Palm) a na PersonalJava PDA (Compaq iPAQ). První model cGML byl vytvořen v říjnu roku 2002, první veřejný koncept cGML schématu byl uvolněn v dubnu 2003 a od dubna roku 2004 se již můžeme setkávat s verzí 1.0.

cGML vychází z jazyka GML (Geogpraphy Markup Lanquage) především pro jeho platformní nezávislost a nenáročnost na hardware. Samotné využití GML v mobilních zařízeních však není možné, a to pro jeho velikost, která by vyžadovala velkou paměť a vlnový rozsah. Proto byly dlouhé GML tagy (značky) nahrazeny tagy krátkými (tabulka 4.1), čímž se velikost souboru samozřejmě zmenšila a náklady na paměť a přenosovou rychlost klesly až o 60 %.

GML	cGML
FeatureCollection	FtCl
_FeatureMember	FtMb
LineStringMember	LnStMb
coordinates	cds

Tabulka 4.1: Srovnání GML a cGML tagů

Při aplikaci v mobilních zařízeních posílá klient cGML žádost k serveru, ve které stanoví absolutní souřadnice zájmové plochy, referenční systém, požadované rozlišení a název vrstvy pro maximální definování. Tato žádost je pak převedena na datový model, který pomocí WFS definuje GML dokument zájmového území. GML dokument je upraven pro využití v mobilním zařízení a pomocí XSL transformace převeden na výsledný cGML soubor. Vlastní zobrazení zajišťuje opět datový model, k němuž se data dostala přes XML parser. Uživatel pak s takto zobrazenými daty může nakládat různým způsobem, např. ukládat, zvětšovat a zmenšovat, vybírat nebo vyhledávat body podle názvu, zjišťovat vzdálenost mezi dvěma body.



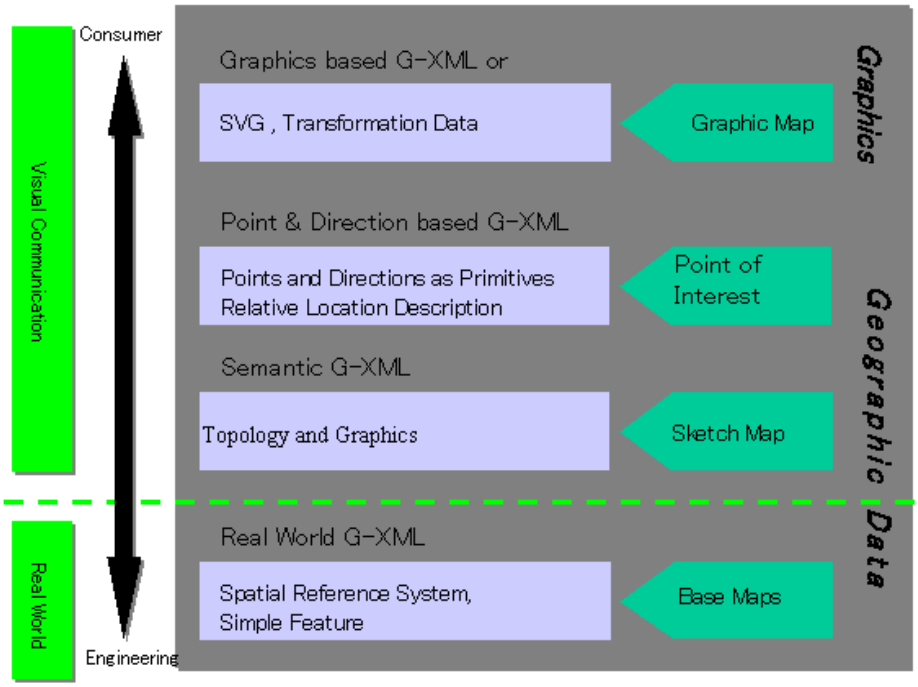
Obrázek 4.1: Architektura cGML – Klient/Server (převzato z [13])

Zdroje: [13]

4.3 Geospatial-eXtensible Markup Language (G-XML)

Jazyk G-XML vznikal pod záštitou Database Promotion Center, Japan a neziskové organizace, která je pod patronací japonského Ministry of Economy, Trade and Industry (METI) od roku 1999. Hlavním cílem projektu G-XML bylo vytvořit protokol pro kódování prostorových dat pomocí XML. Protože G-XML vznikal souběžně s jazykem GML, snažili se japonští vývojáři o těsnou spolupráci s OGC.

První verze tohoto jazyka, tedy verze 1.0, byla oficiálně uvedena na konci března roku 2000 a měla čtyři DTD schémata (Real World, Point and Direction, Semantic a Graphic … obrázek 4.2). Druhá verze, verze 2.0, vyšla jako nová JIS specifikace 25. srpna 2001. Úzké spojení s GML měla vyřešit verze 2.5, na které byly práce dokončeny v březnu 2001. Největšího rozšíření se dočkala verze 3.0 z konce března 2003, která mimo podpory GML, umožňovala například využití G-XML v mobilních telefonech nebo uživatelská rozšíření. Poslední verze 3.1 byla jako specifikace vydána v dubnu roku 2004.



Obrázek 4.2: Struktura G-XML verze 1.0 (převzato z [20])

V Japonsku se tento jazyk nečastěji využívá k navigaci uživatelů mobilních telefonů, automobilů a v administrativě.

Zdroje: [20], [21]

4.4 LandXML

Tvorbu jazyka LandXML má na starosti celosvětová organizace LandXML.org, která vznikla v prosinci roku 1999 především na popud firmy Autodesk a US DOT EAS-E(U.S. Department of Transportation Engeneering and Survey – Exchange). V březnu roku 2000 měla 26 členů. Postupem času se tato organizace rozšířila po celém světě. V říjnu roku 2006 měla již 600 představitelů ze 511 členských společností a státních orgánů ve 37 zemích světa.

LandXML je specializovaný formát XML dat zaměřující se na stavební inženýrství a správu měřených dat obvykle užívaných v územním plánování, při terénních úpravách, v dálkovém průzkumu Země a v dopravním inženýrství. Obě dosud vydané verze, tedy verze 1.0 ze 17. července 2002 a verze 1.1 z 21. července 2006, splňují základní cíle projektu, a to:

- umožnit využití v jakémukoli softwaru,

- vytvořit takový datový formát, který by umožnil dlouholetou archivaci,
- vytvořit snadno zobrazitelný datový formát – data uložená ve formátu LandXML lze pomocí XSL transformace zobrazit jakýmkoli způsobem.

Obě verze jsou vytvořeny takovým způsobem, aby uživatel nebyl omezen verzí, ve které svá data spravuje, tj. data poprvé uložená ve verzi 1.1 lze zpracovávat ve verzi 1.0. Schémovým jazykem je XML Schema.

V současné době podporuje LandXML více jak 55 registrovaných softwarových aplikací, např. Autodesk, CAiCE, Carlson, Eagle Point, Leica GeoSystems, MicroSurvey, Trimble Navigation, Tripod Data Systems, Topcon.

V praxi je tento jazyk využitý například pro katastrální systém Landonline na Novém Zélandu a EPlan v Queenslandu v Austrálii, ve Slovinsku pro zobrazení 3D modelu silnic, v USA pro Interactive Highway Safety Design Model (IHSDM).

Zdroje: [12], [14]

5 UKLÁDÁNÍ GEODAT DO XML NATIVNÍCH DATABÁZÍ

Výběr databází

Při řešení problematiky ukládání geodat do XML nativních databází bylo nutné v úplném počátku vybrat zástupce nekomerčních XML nativních databází, poněvadž nekomerčních databázových systémů je poměrně velký počet a zaobírat se každou databází by pro názornost nemělo velký smysl.

Protože se jednotlivé databázové systémy liší především typem, byla tato charakteristika zvolena jako zásadní, a tak na základě typu došlo k výběru kandidátů. V dalším kroku byl brán zřetel na rozšířenost a známost každého jednotlivého databázového systému, přičemž se samozřejmě upřednostnily známější a rozšířenější aplikace. V konečném výsledku byly vybrány čtyři databázové systémy, ze kterých ale pouze tři mohly být použity:

- 4Suite (kapitola 5.2) jako zástupce objektově-orientovaných databází,
- Berkeley DB XML (kapitola 5.3) jako zástupce databáze typu klíč-hodnota,
- eXist (kapitola 5.4) jako zástupce databází s vlastním modelem, jež v rámci nekomerčních XML nativních databází tvoří nejobsáhlejší skupinu.

Čtvrtým vybraným databázovým systémem, a tedy nakonec nepoužitým, byl DBDOM jako zástupce relačních databází. Začátky práce s touto aplikací probíhaly bez jakéhokoli problému. Po instalaci v systému PostgreSQL, který byl jedním z možných systémů využitelných pro chod databázového systému DBDOM, byla vytvořena databáze, v níž se automaticky vytvořilo 19 tabulek a 57 funkcí určených pro vkládání dat do připravených tabulek. V další fázi „vytvoření XML dokumentu v databázi“ se již vyskytly podstatné problémy s předdefinovanými funkcemi, které se nepodařilo odstranit. Podle mého mínění by ani při správné funkci všech funkcí nebylo možné vložit již existující XML dokument, aniž by byla vytvořena další, tj. 58, funkce, která by umožňovala, samozřejmě s využitím předdefinovaných funkcí, i tento způsob vložení dat do databáze. Databáze DBDOM se tak stala zcela nevyhovující a tudíž nepoužitelná.

DBDOM není jediným relačním databázovým systémem, dalším, a zároveň také posledním možným kandidátem této skupiny byl systém XpSQL. Nad operačním systémem Windows se však tato aplikace nepodařila nainstalovat, a tak nakonec není skupina relačních nekomerčních XML nativních databází bohužel zastoupena.

Geodata

Volba vhodného vzorku geodat pro uložení do již vybraných databázových systémů také nebyla triviální záležitostí. Pro objektivní posouzení možnosti využití XML nativních databází v oblasti geografických věd bylo vhodné vybrat reálná data.

Západočeská univerzita spolupracovala při tvorbě Atlasu mezinárodních vztahů, který vznikal především s využitím XML technologií. Zdrojová data pro tento atlas, která pochází z databáze ESRI Data and Maps, byla tedy také v XML formátu, přesněji ve formátu JML[1] (JUMP GML).

Protože zdrojová data Atlasu mezinárodních vztahů byla přímo v XML formátu a jednalo se o reálná data, bylo několik z těchto zdrojových souborů vybráno pro využití v nativních databázích. Šest získaných souborů se nemusí jevit jako dostatečný počet, ale jejich odlišné, a ve třech případech velmi velké, velikosti poskytnou podle mého názoru dobrý nástin využití nativních XML databází v oblasti geografických věd.

Název souboru	Velikost souboru [B]
body_miny1.jml	5 974
body_miny2.jml	27 182
magda.jml	14 720 878
miny.jml	14 612 497
zbrojeni.jml	14 625 758
zbrojeni_popisky.jml	6 673

Tabulka 5.1: Velikosti použitých souborů s geodaty

Dotazování

Na všechny druhy dat lze v rámci běžných databázových systémů vytvářet různé druhy dotazů, jejichž výsledkem je vždy výběr všech dat splňujících určitou podmínku. Ani geodata nejsou výjimkou.

Protože geodata obsahují atributovou a zároveň také prostorovou složku (kapitola 4), lze nad těmito druhy dat vykonávat v zásadě tři druhy dotazů:

a) atributové dotazy – výsledkem těchto dotazů jsou všechny geografické objekty splňující určitou vlastnost,

[1] Tento formát má základ ve formátu GML (Geography Markup Language) a je využíván především v rámci Open Source programů JUMP a openJUMP. Soubory formátu JML mají v sobě již zabudovanou strukturu dokumentu, tedy jakési schéma v úvodu vlastního dokumentu, narozdíl od klasických souborů ve formátu GML.

b) prostorové dotazy – výsledkem těchto dotazů jsou všechny geografické objekty s určitou prostorovou závislostí,
c) kombinované dotazy – u těchto dotazů se zohledňuje prostorová příslušnost s příslušností atributovou.

Při řešení dotazů nad geodaty ve formátu XML by se mohl potřít rozdíl mezi jednotlivými druhy dotazů, poněvadž všechna data uložená v XML souboru se jeví naprosto stejně (tj. souřadnice jsou pouhým textem, stejně jako název státu), a tak v případě prostorové složky není na první pohled patrné žádné prostorové určení. V této práci však byly vytvořeny jak atributové, tak prostorové dotazy, přičemž za prostorové dotazy byly označeny všechny ty, které se jakýmkoli způsobem dotýkaly geometrie či souřadnic.

5.1 XQuery a XPath dotazy pro vybrané databáze

Dotazování nad geodaty v prostředí XML nativních databází je zřejmě jediná oblast, ve které se geodata od ostatních dat podstatně odlišují. Vzhledem k existenci prostorové složky roste i náročnost dotazů, obzvláště prostorových, které se samozřejmě nad běžnými daty nevytvářejí. Proto bylo nutné vytvořit několik jednoduchých i poněkud složitějších dotazů, jak atributových, tak prostorových, aby bylo možné zhodnotit vhodnost využití dané databáze pro ukládání geodat.

Protože většina XML nativních databází podporuje dva základní dotazovací jazyky, XPath a XQuery, vznikaly dotazy právě v těchto formátech, a to pomocí softwaru Altova XML Spy® 2007 Enterprise Edition.

Prvotně byly zformulovány dotazy v jazyce XQuery. Konečný počet těchto dotazů je dvanáct – osm atributových dotazů a čtyři dotazy prostorové[1]. Z některých jednodušších dotazů pak vycházely obdobné dotazy v jazyce XPath, konečný počet XPath dotazů je pět – tři atributové a dva prostorové.

5.1.1 Atributové XQuery dotazy

Prvním atributovým dotazem, dále (XQ_1), je řádkový dotaz nad kolekcí dokumentů `geodata`, jehož výstupem jsou všechny elementy `property` prvku (elementu `feature`), který v elementu `property` obsahuje hodnoty „`Czech Republic`“ nebo „`CZE`“.

[1] Prostorové dotazy byly vzhledem k vnitřní struktuře *.jml souborů velmi časově náročné, proto byl jejich počet omezen.

- atrib_dotaz_CR_radka.xquery ... (XQ_1)

```
collection("geodata")//feature[contains(property,("Czech
Republic","CZE"))] /property
```

Výsledek dotazu:

```
<property name="GMI_CNTRY">CZE</property>
<property name="CNTRY_NAME">Ceska republika</property>
<property name="miny">B</property>
<property name="CNTRY_NAME">Czech Republic
</property>
<property name="kod"/>
<property name="REGION">Eastern Europe       </property>
<property name="CONTINENT">Europe       </property>
<property name="FIPS_CNTRY">EZ</property>
<property name="GMI_CNTRY">CZE</property>
<property name="hustota">129.5</property>
<property name="CNTRY_NAME">Czech Republic
</property>
<property name="SOVEREIGN">Czech Republic                </property>
<property name="TAB1_COLOR">B</property>
<property name="TAB2_COLOR">C</property>
<property name="TAB3_COLOR">A</property>
<property name="TAB4_COLOR">B</property>
<property name="TAB5_COLOR">A</property>
<property name="TAB6_COLOR">C</property>
```

Dalšími dvěma dotazy jsou atributové dotazy se stejnou myšlenkou, tedy i výsledkem, ale pouze jinou strukturou. Jeden dotaz je opět řádkový a ve druhém je využit cyklus FOR. Výstupem těchto dotazů jsou názvy všech států, jež patří pod suverenitu Spojeného království.

- atrib_dotaz_UK_radka.xquery ... (XQ_2), atrib_dotaz_UK.xquery ... (XQ_3)

```
collection("geodata")//feature[property[@name="SOVEREIGN"]="United
Kingdom                "]/property[@name="CNTRY_NAME"]/text()

...

for $i in collection('geodata')//feature
where $i/property[@name="SOVEREIGN"] = "United Kingdom                "
return
$i//property[@name="CNTRY_NAME"]/text()
```

Výsledek dotazů:

```
Anguilla
Bermuda
Cayman Islands
Falkland Islands (Islas Malvinas)
Gibraltar
Guernsey
Man, Isle of
British Indian Ocean Territory
Jersey
Montserrat
Pitcairn Islands
```

```
St. Helena
South Georgia and the South Sandwich Is
Turks and Caicos Islands
United Kingdom
British Virgin Islands
```

Výstupem následujícího dotazu, dále (XQ_4), je výpis hodnot hustoty v intervalu (200;300), přičemž výsledné hodnoty jsou seřazené podle velikosti (`order by`).

- atrib_dotaz_hustota_200_300.xquery ... (XQ_4)

```
for $i in doc('magda.jml')//feature
where some $j in $i/property[@name ="hustota"] satisfies ($j > 200 and $j
< 300)
order by $i/property[@name ="hustota"]
return
$i/property[@name ="hustota"]/text()
```

Výsledek dotazu:

```
215.6
224.0
231.2
241.0
245.2
246.1
250.8
261.4
265.8
290.5
291.5
292.9
299.4
```

Dotaz k získání názvů všech kontinentů (XQ_5) využívá pro získání výsledku mimo jiné odstranění duplicit (`distinct-values()`). Stejně jako dotaz předchozí řadí výsledné hodnoty (`order by`).

- atrib_dotaz_kontinenty.xquery ... (XQ_5)

```
let $kontinenty :=
collection("geodata")//feature/property[@name="CONTINENT "]
for $kontinent in distinct-values($kontinenty)
order by $kontinent
return
$kontinent
```

Výsledek dotazu:

```
Africa
Antarctica
Asia
Australia
Europe
North America
Oceania
South America
```

Další dotaz (XQ_6) využívá předdefinovaných funkcí pro získání průměru, maximální a minimální hodnoty. Těchto funkcí využívá k získání průměrné, minimální a maximální hustoty zalidnění světa. Zároveň také vytváří nové elementy, do nichž jsou pomocí konstruktorů ({}) příslušné výsledky umístěny.

- atrib_dotaz_hustoty.xquery ... (XQ_6)

```
<prumer_hustota>{avg(doc("magda.jml")//feature/property[@name="hustota"])
}</prumer_hustota>,
<min_hustota>{min(doc("magda.jml")//feature/property[@name="hustota"])}</
min_hustota>,
<max_hustota>{max(doc("magda.jml")//feature/property[@name="hustota"])}</
max_hustota>
```

Výsledek dotazu:

```
<prumer_hustota>257.38804780876484</prumer_hustota>
<min_hustota>0</min_hustota>
<max_hustota>16717.9</max_hustota>
```

Následující atributový dotaz (XQ_7) vytváří nové elementy `hustota` s hustotou zalidnění evropských států a v nich atributy `stat` (název evropského státu) a `zkr` (zkratka evropského státu).

- atrib_dotaz_hustota_Evropa.xquery ... (XQ_7)

```
for $i in collection('geodata')//feature
where $i/property[@name ="CONTINENT"] = "Europe       "
return
for $j in doc('magda.jml')//feature
where
contains($j/property[@name="CNTRY_NAME"]/text(),substring($i/property[@na
me="CNTRY_NAME"]/text(),1,20))
return
element hustota {
      attribute stat{$i/property[@name="CNTRY_NAME"]/text()},
      attribute zkr {$j/property[@name="GMI_CNTRY"]/text()},
      $j/property[@name="hustota"]/text()}
```

Výsledek dotazu (49 států):

```
<hustota stat="Albania                                    "
zkr="ALB">109.3</hustota>
<hustota stat="Andorra                                    "
zkr="AND">145.7</hustota>
<hustota stat="Austria                                    "
zkr="AUT">96.6</hustota>
.
.
.
<hustota stat="Czech Republic                             "
zkr="CZE">129.5</hustota>
.
.
.
<hustota stat="United Kingdom                             "
zkr="GBR">245.2</hustota>
```

```
<hustota stat="Ukraine                                          "
zkr="UKR">78.6</hustota>
<hustota stat="Russia                                           "
zkr="RUS">8.5</hustota>
```

Poslední atributový XQuery dotaz (XQ_8) je zároveň nejsložitějším atributovým dotazem. Využívá uživatelem definovanou funkci, pomocí níž je schopen určit průměrnou hustotu zalidnění jednotlivých kontinentů.

- atrib_dotaz_prum_hustota_svet.xquery ... (XQ_8)

```
declare function local:prum_hustota ($kontinent)
{avg(
for $i in doc('zbrojeni.jml')//feature
where $i/property[@name ="CONTINENT"] = $kontinent
return
for $j in doc('magda.jml')//feature
where
contains($j/property[@name="CNTRY_NAME"]/text(),substring($i/property[@na
me="CNTRY_NAME"]/text(),1,20))
return
      $j/property[@name="hustota"])
};

<prumer_hustota kontinent="Afrika">{
local:prum_hustota("Africa       ")}
</prumer_hustota>,

<prumer_hustota kontinent="Antarktida">{
local:prum_hustota("Antarctica   ")}
</prumer_hustota>,

<prumer_hustota kontinent="Asie">{
local:prum_hustota("Asia         ")}
</prumer_hustota>,

<prumer_hustota kontinent="Australie a Oceanie">{
local:prum_hustota(("Australia    ","Oceania      "))}
</prumer_hustota>,

<prumer_hustota kontinent="Evropa">{
local:prum_hustota("Europe       ")}
</prumer_hustota>,

<prumer_hustota kontinent="Jizni a Severni Amerika">{
local:prum_hustota(("South America","North America"))}
</prumer_hustota>
```

Výsledek dotazu:

```
<prumer_hustota kontinent="Afrika">63.458064516129035</prumer_hustota>
<prumer_hustota kontinent="Antarktida">0</prumer_hustota>
<prumer_hustota kontinent="Asie">582.6472727272726</prumer_hustota>
<prumer_hustota kontinent="Australie a
Oceanie">56.806451612903224</prumer_hustota>
<prumer_hustota kontinent="Evropa">467.18163265306123</prumer_hustota>
<prumer_hustota kontinent="Jizni a Severni
Amerika">78.46666666666667</prumer_hustota>
```

5.1.2 Prostorové XQuery dotazy

Prvním prostorovým dotazem (XQ_9) je běžný dotaz porovnávající, zda příslušný element obsahuje danou hodnotu. V našem případě, zda element `gml:coordinates` obsahuje souřadnice `"152.258246352413,4992.122441603189"`. Výsledkem tedy bude zjištění, jaký stát má ve svých hranicích dané souřadnice.

- prostor_dotaz_souradnice.xquery ... (XQ_9)

```
declare namespace gml="http://www.opengis.net/gml";
let $souradnice := "152.258246352413,4992.122441603189"
for $i in collection('geodata')//feature
where   some   $j   in   $i[property/@name="CONTINENT"]//gml:coordinates
satisfies contains($j/text(),$souradnice)
return
      element stat{$i/property[@name!="kod"]}
```

Výsledek dotazu:

```
<staty>
    <stat>
        <property name="CNTRY_NAME">Czech Republic
</property>
        <property name="REGION">Eastern Europe       </property>
        <property name="CONTINENT">Europe       </property>
    </stat>
    <stat>
        <property name="CNTRY_NAME">Germany
</property>
        <property name="REGION">Western Europe       </property>
        <property name="CONTINENT">Europe       </property>
    </stat>
</staty>
```

U dalšího dotazu (XQ_10) by se dalo pochybovat o jeho zařazení, v základu řeší pouze počet výskytů určitého elementu (`gml:MultiPolygon`, `gml:Polygon`), ale poněvadž by i tento dotaz mohl pomoci například při vyhledávání států s komplikovanými hranicemi (což je jasná prostorová tématika), je zařazen do skupiny prostorových dotazů.

- prostor_dotaz_multipolygony.xquery ... (XQ_10)

```
declare namespace gml="http://www.opengis.net/gml";
element multipolygony{
      attribute celkovy_pocet {

      count(doc("magda.jml")//feature[boolean(geometry/gml:MultiPolygon)]
)},
      for                                $i                                in
doc("magda.jml")//feature[boolean(geometry/gml:MultiPolygon )]
      return
      element stat {
            attribute pocet_polygonu {count($i//gml:Polygon)},
            $i/property[@name="CNTRY_NAME"]/text()
      }
}
```

Výsledek dotazu byl poněkud rozsáhlejší, pro představu je uvedena pouze zkrácená verze:

```
<multipolygony celkovy_pocet="114">
    <stat pocet_polygonu="2">Antigua and Barbuda
</stat>
    <stat pocet_polygonu="4">Azerbaijan
</stat>
    <stat pocet_polygonu="2">Armenia
</stat>
.
.
.
<stat pocet_polygonu="5">Yemen                                    </stat>
    <stat pocet_polygonu="194">Russia
</stat>
    <stat pocet_polygonu="6">Fiji
</stat>
</multipolygony>
```

Poslední dva dotazy jsou již klasickými prostorovými dotazy, řešícími přímo nějaký problém. Jejich zdrojový kód je poněkud komplikovanější, a čas potřebný k vyhodnocení i v běžném softwaru (nikoliv ještě v databázi) byl také velmi značný – první z dotazů potřeboval na svůj průběh cca 5 min, druhý, ještě složitější pak čas vysoce překračující 30minutovou hranici.

První ze složitějších dotazů (XQ_11) zjišťuje, jaké sousední státy má Česká republika. Za dlouhý časový průběh dotazu mohou především četná porovnávání hraničních souřadnic jednotlivých států.

- prostor_dotaz_sousedi.xquery ... (XQ_11)

```
declare namespace gml="http://www.opengis.net/gml";
<sousedni_staty nazev="česká republika">
{
let                              $retezec                              :=
doc('magda.jml')//feature[property="CZE"]//gml:coordinates/text()
let $souradnice1_0 := substring($retezec,0,30)
for $k in (0 to string-length($retezec) idiv 40)
      let          $souradnice1          :=          substring(substring-
after(substring($retezec,$k*40,150),"                      "),0,30)
      let          $souradnice2          :=          substring(substring-
after(substring($retezec,($k+1)*40,150),"                      "),0,30)
      for $i in doc('magda.jml')//feature
      where some $j in $i//gml:coordinates satisfies (
            if($k  =  0)  then  (contains($j/text(),$souradnice1_0)  and
not(contains($j/text(),$souradnice2 )))
            else(contains($j/text(),$souradnice1) and
not(contains($j/text(),$souradnice2)) ))
      return
            element nazev_statu {$i/property[@name ="CNTRY_NAME"]/text()}
}
</sousedni_staty>
```

Výsledek dotazu:

```
<sousedni_staty nazev="Česká republika">
    <nazev_statu>Germany                                    </nazev_statu>
    <nazev_statu>Poland                                     </nazev_statu>
    <nazev_statu>Slovakia                                   </nazev_statu>
    <nazev_statu>Austria                                    </nazev_statu>
</sousedni_staty>
```

Druhý složitý dotaz (XQ_12), který činil velké problémy i klasickému ne-databázovému softwaru, řeší problematiku výskytu států v určitém okolí, přesněji výskyt států od určité souřadnice v okruhu 1000 km. V tomto případě byly vytvořeny dvě uživatelsky definované funkce, v nichž jedna je volána druhou. Toto řešení kód značně zpřehlednilo ale ke zrychlení rozhodně nedošlo. Velkou měrou se na dlouhém čase podílí prohledávání a porovnávání všech souřadnic (pro každou dvojici souřadnic X,Y muselo dojít k jejich rozdělení a převedení na numerickou hodnotu) bez ohledu na to, že daný stát už například svými souřadnice podmínku splnil – částečným řešením by mohlo být násilné ukončení cyklu v případě, že tato situace nastane (tato myšlenka se však nepodařila zrealizovat).

- prostor_dotaz_okruh_1000km.xquery ... (XQ_12)

```
declare namespace gml="http://www.opengis.net/gml";

declare function local:porovnani($retezec, $X_0 ,$Y_0 )
{
for $k in (0 to string-length($retezec) idiv 40)
let $souradnice := substring(substring-
after(substring($retezec,$k*40,150), "                        "),0,35)
let $X := number(substring-before($souradnice,","))
let $Y := number(substring-after($souradnice,","))
where (($X - $X_0)*($X - $X_0) + ($Y - $Y_0)*($Y - $Y_0))<(1000*1000)
return
      element souradnice{
      attribute x {$X},
      attribute y {$Y}}
};

declare function local:oblast($i, $X_0 ,$Y_0 )
{
for $LinearRing in $i/geometry//gml:LinearRing
let $retezec := $LinearRing//gml:coordinates/text()
return
      local:porovnani($retezec,$X_0 ,$Y_0 )
};


let $X_0 := 500
let $Y_0 := 5000
return
      element okruh_1000km{
      attribute X_0 { $X_0 },
      attribute Y_0 { $Y_0 },
      for $i in doc('magda.jml')//feature
      return
            if (boolean(local:oblast($i,$X_0 ,$Y_0 )))
```

```
            then(
            <nazev_statu>{$i/property[@name       ="CNTRY_NAME"]/text()}<
/nazev_statu>
            )
            else ()
      }
```

Výsledek dotazu:

```
<okruh_1000km X_0="500" Y_0="5000">
    <nazev_statu>Albania                                </nazev_statu>
    <nazev_statu>Austria                                </nazev_statu>
    <nazev_statu>Belgium                                </nazev_statu>
    <nazev_statu>Bosnia and Herzegovina                 </nazev_statu>
    <nazev_statu>Byelarus                               </nazev_statu>
    <nazev_statu>Bulgaria                               </nazev_statu>
    <nazev_statu>Denmark                                </nazev_statu>
    <nazev_statu>Czech Republic                         </nazev_statu>
    <nazev_statu>France                                 </nazev_statu>
    <nazev_statu>Germany                                </nazev_statu>
    <nazev_statu>Greece                                 </nazev_statu>
    <nazev_statu>Croatia                                </nazev_statu>
    <nazev_statu>Hungary                                </nazev_statu>
    <nazev_statu>Italy                                  </nazev_statu>
    <nazev_statu>Latvia                                 </nazev_statu>
    <nazev_statu>Lithuania                              </nazev_statu>
    <nazev_statu>Slovakia                               </nazev_statu>
    <nazev_statu>Liechtenstein                          </nazev_statu>
    <nazev_statu>Luxembourg                             </nazev_statu>
    <nazev_statu>Moldova                                </nazev_statu>
    <nazev_statu>Macedonia                              </nazev_statu>
    <nazev_statu>Montenegro                             </nazev_statu>
    <nazev_statu>Netherlands                            </nazev_statu>
    <nazev_statu>Poland                                 </nazev_statu>
    <nazev_statu>Romania                                </nazev_statu>
    <nazev_statu>Slovenia                               </nazev_statu>
    <nazev_statu>San Marino                             </nazev_statu>
    <nazev_statu>Serbia                                 </nazev_statu>
    <nazev_statu>Sweden                                 </nazev_statu>
    <nazev_statu>Switzerland                            </nazev_statu>
    <nazev_statu>Ukraine                                </nazev_statu>
    <nazev_statu>Russia                                 </nazev_statu>
</okruh_1000km>
```

5.1.3 Atributové XPath dotazy

První atributový XPath dotaz, dále (XP_1), vychází z prvního atributového XQuery dotazu vyhledávajícího všechny elementy property týkající se České republiky. Tento dotaz je na rozdíl od XQuery vztažen nikoli k celé kolekci dokumentů, ale pouze k jednomu dokumentu (magda.jml), a proto je upravena i podmínka pro určení správného elementu `feature`, který se České republiky týká. Výsledek není výpis elementů `property`, ale výpis přímo jejich hodnot.

- XPath_atrib_dotaz_CR_magda.txt ... (XP_1)

```
//feature[property='CZE']/property/text()
```

Výsledek:

```
EZ
CZE
129.5
Czech Republic
Czech Republic
B
C
A
B
A
C
```

Dalším vhodným XQuery dotazem pro odvození dotazu XPath (XP_2) byl dotaz vypisující názvy všech států pod suverenitou Spojeného království. XPath dotaz podává naprosto stejný výsledek, opět pracuje s dokumentem magda.jml. Vzhledem k problémům při dotazování v databázi 4Suite (více v podkapitole 5.2.3) je i v tomto dotazu podmínka pro správné vyhodnocení odlišná od dotazu v XQuery.

- XPath_atrib_dotaz_UK_magda.txt … (XP_2)

```
//feature[contains(property[@name='SOVEREIGN'],'Kingdom')]/property[@name
="CNTRY_NAME"]/text()
```

Výsledek:

```
Anguilla
Bermuda
Cayman Islands
Falkland Islands (Islas Malvinas)
Gibraltar
Guernsey
Man, Isle of
British Indian Ocean Territory
Jersey
Montserrat
Pitcairn Islands
St. Helena
South Georgia and the South Sandwich Is
Turks and Caicos Islands
United Kingdom
British Virgin Islands
```

Posledním atributovým XPath dotazem (XP_3) je dotaz pro zjištění průměrné hustoty světa. V jazyce XPath neexistuje žádná funkce pro přímé zjištění průměru, proto je zde využit klasický výpočet – součet všech hodnot hustoty je vydělen počtem uzlů s hustotou zalidnění.

- XPath_atrib_dotaz_prum_hustota_sveta_magda.txt ... (XP_3)

```
sum(//feature/property[@name='hustota'])div(count(//feature/property[@nam
e='hustota']))
```

Výsledek:

```
257.38804780876494023904382470119 52
```

5.1.4 Prostorové XPath dotazy

Pro odvození prostorových XPath dotazů byly vhodné pouze dva jednodušší prostorové dotazy XQuery, tj. dotaz zjišťující státy, které obsahují ve svých hraničních souřadnicích určitou souřadnici, a dotaz zabývající se element `gml:MultiPolygon`.

První z XPath dotazů (XP_4) je schopen vrátit naprosto totožný výsledek jako XQuery dotaz, tedy výpis všech států, které obsahují ve svých hranicích danou souřadnici. Před vlastní aplikací XPath dotazu na dokument musí ale v rámci příslušného databázového systému předcházet definování jmenného prostoru (`gml="http://www.opengis.net/gml"`), tato podmínka by však neměla činit problémy.

- XPath_prostor_dotaz_souradnice_magda.txt ... (XP_4)

```
//feature[.//gml:coordinates[contains(.,'152.258246352413,4992.122441603189')]]/property[@name='CNTRY_NAME']/text()
```

Výsledek dotazu:

```
Czech Republic
Germany
```

Druhý prostorový XPath dotaz (XP_5) již s dotazem XQuery příliš společného nemá, zjišťuje pouze celkový počet elementů `gml:MultiPolygon` v dokumentu. I zde je nutná deklarace jmenného prostoru gml.

- XPath_prostor_dotaz_multipolygony_magda.txt ... (XP_5)

```
count(//feature[boolean(geometry/gml:MultiPolygon)])
```

Výsledek dotazu:

```
114
```

5.2 4Suite … objektově-orientovaná databáze

Obecné charakteristiky

4Suite, navržený pro XML zpracování a správu objektově orientovaných databází, je vyvinut především firmou Fourthought, Inc.

4Suite umožňuje uživatelům využívat standardů XML a také například zdokonalit své webové aplikace, protože podporuje mnoho metod pro přístup k datům, dotazování a indexování.

Celý programový systém je vytvořen v programovacím jazyce Python s rozšířeními v programovacím jazyce C. Podporuje vzdálený přístup pomocí HTTP, RPC, FTP a CORBA. Je realizovatelný na platformách:

- POSIX - Linux, FreeBSD, OpenBSD, NetBSD, Solaris, Cygwin, Mac OS, atd.
- Windows - Windows 2000, XP nebo Server 2003; pro Windows 98, Me nebo NT není účinnost programu garantována.

Pro správný chod programu musí být nainstalován Python 2.2.1 nebo novější, a pro případné zasahování do zdroje je nutný mít kompilátor jazyka C.

Základním prvkem 4Suite je knihovna nástrojů (včetně těch, které mohou být implementovány i pomocí příkazové řádky) pro XML zpracování, tj. DOM, SAX, XSLT, XInclude, XPointer, XLink, XPath, XUpdate, RELAX NG a XML/SGML.

Pomocí 4Suite je tedy například možné:

- popsat dokument pomocí struktury DOM – tj. znázornit strukturu objektů, které tvoří XML dokument,
- analyzovat dokument na základě systému SAX,
- vytvořit XPath dotazy nad již zanalyzovaným dokumentem,
- použít XSLT a tedy využít možnost vytvořit jinou strukturu vstupního XML dokumentu,
- aktualizovat dokument pomocí XUpdate,
- zvalidovat dokument prostřednictvím RELAX NG.

Zdroje: [25], [37], [41], README.txt

5.2.1 Instalace

Jak již bylo uvedeno výše, pro správný chod programu je nutné mít nainstalován programovací jazyk Python, přičemž každá verze 4Suite odpovídá určité verzi Python.

Pro svoji práci jsem zvolila 4Suite verze 1.0b1 pro Python verze 2.3, protože tato verze 4Suite umožňuje také správu dokumentů a práci s RDF. Vyšší a tedy i novější verze tato

„rozšíření“, která byla ve starších verzích označována jako 4Suite Server (4ss), již neobsahují.

Postup instalace:

a) Instalace Python 2.3

Z internetové adresy http://www.python.org/download/releases/2.3/ jsem stáhla instalační soubor Python-2.3.exe pro Windows, z čehož je zřejmé, že jeho instalace nebyla vůbec složitá.

b) Instalace 4Suite

Z adresy ftp://ftp.4suite.org/pub/4Suite/4Suite-1.0b1.win32-py2.3.exe jsem si stáhla instalační soubor a prostřednictvím instalátoru pro Windows jednoduše nainstalovala.

Pro využívání 4Suite Serveru je nutné nastavení konfiguračního souboru (4ss.conf), který je umístěn v …/Python23/Share/Settings/4Suite. Server může pracovat na dvou různých bázích. První způsob práce může být vykonáván nad databázovým serverem PostgreSQL a druhý pak využívá systému „FlatFile“. Tento způsob je sice pomalejší, ale stačí mu pouze prázdný adresář a nějaké místo na disku.

c) Nastavení konfiguračního souboru

Konfigurační soubor 4ss.conf obsahuje nastavení pro oba způsoby práce. Uživatel pouze zakomentováním určí, zda bude využívat PostgreSQL či „FlatFile“. Já zvolila druhou možnost, tedy „souborový“ způsob práce s 4Suite Serverem, protože ani zbývající testované nativní XML databázové systémy (Berkeley DB XML, eXist) nepracují nad jiným databázovým systémem. Část pro PostgreSQL jsem zakomentovala a následně jsem odstranila komentář z části určené pro „FlatFile“. Posledním krokem před uložením nového konfiguračního souboru bylo nastavení cest pro PidFile a LogFile. Ukázka mého konfiguračního souboru je v příloze A (Konfigurační soubor „4ss.conf“).

5.2.2 Pracovní prostředí a základní funkce

Pracovat s databázovým systémem 4Suite lze dvěma způsoby, buď v prostředí programovacího jazyka Python pomocí vestavěných knihoven a pro Python obvyklých

operací, nebo v klasické příkazové řádce DOS s využitím nástrojů přímo určených pro příkazovou řádku.

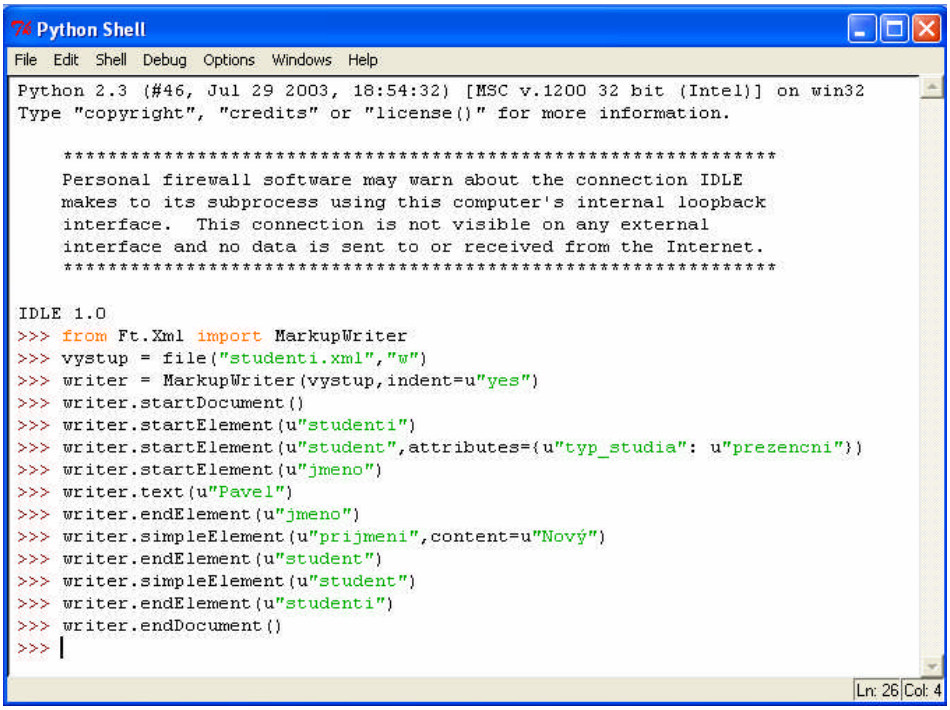
Vestavěné knihovny se samozřejmě pro mnou zvolenou verzi 4Suite od jiných verzí ve svém obsahu liší, a to především v poskytovaných funkcích jednotlivých modulů. Poněvadž klasifikace jednotlivých modulů a jejich porovnání s jinými verzemi 4Suite je již nad rámec mé práce, uvedu zde pouze pro představu 4 základní knihovny, jež mohou obsahovat další moduly:

- Ft.Lib – různé knihovny, které je možné využívat nezávisle na sobě,
- Ft.Rdf – nástroje pro RDF zpracování, včetně dotazovacích jazyků,
- Ft.Server – umožňuje webový přístup,
- Ft.Xml – obsahuje další moduly pro práci s XML (např. modul XUpdate, Xslt, XPath, XPointer, XLink).

Python je běžně spustitelný pomocí:

- Nabídka Start => Programy => Python 2.3 => IDLE (Python GUI),
- Nabídka Start => Programy => Python 2.3 => Python (command line),
- C:\Python23\Python.exe.

Protože jsem pro svoji práci využívala především klasické příkazové řádky DOS, ukáži pracovní prostředí Pythonu na jednoduchém příkladu vytvoření XML dokumentu:



Obrázek 5.1: 4Suite – pracovní prostředí Python GUI

Výsledkem těchto programových příkazů je vytvoření XML dokumentu „studenti.xml“ s následující strukturou:

```
<?xml version="1.0" encoding="UTF-8"?>
<studenti>
     <student typ_studia="prezencni">
          <jmeno>Pavel</jmeno>
          <prijmeni>Nový</prijmeni>
     </student>
     <student/>
</studenti>
```

Pro ukládání XML dokumentů do databáze klasická příkazová řádka plně vyhovuje, a to díky nástrojům určeným přímo pro příkazovou řádku:

- 4xml – analýzy XML dokumentů, ...,
- 4xpath – vyhodnocení výrazů XPath,
- 4xslt – nástroj pro zpracování pomocí XSLT,
- 4xupdate – XUpdate zpracování,
- 4rdf – RDF/XML analýzy, dotazování, ...,
- 4ss_manager – správa dokumentů a RDF zdrojů,
- 4ss – příkazy uživatele na dokumenty a RDF,
- 4versa – dotazovaní pomocí jazyka Versa[1].

Vytvoření kolekce dokumentů a operace s již existující kolekcí

Kolekce dokumentů je v případě 4Suite označována stejně jako v Berkeley DB XML jako kontejner. Vytvoření první kolekce dokumentů musí předcházet inicializace 4Suite Serveru. Protože novější verze 4Suite mají pouze čtyři nástroje pro příkazovou řádku (4xml, 4xpath, 4xupdate, 4xslt) a neobsahují ani vestavěné knihovny určené pro práci se serverem a RDF (Ft.Server, Ft.RDF), nelze server jakýmkoli způsobem inicializovat a tedy ani vytvořit kolekci dokumentů. Právě z tohoto důvodu jsem zvolila starší verzi, která vytvoření kolekcí dokumentů a zároveň jejich správu umožňuje.

4Suite Server lze zinicializovat následujícím způsobem:

```
...> 4ss_manager init
You are about to initialize 4Suite Server. This will erase ALL
DATA in the databases.
Are you sure (yes/no)? y
Add super user name (or just "enter" to pass): uzivatelske_jmeno
```

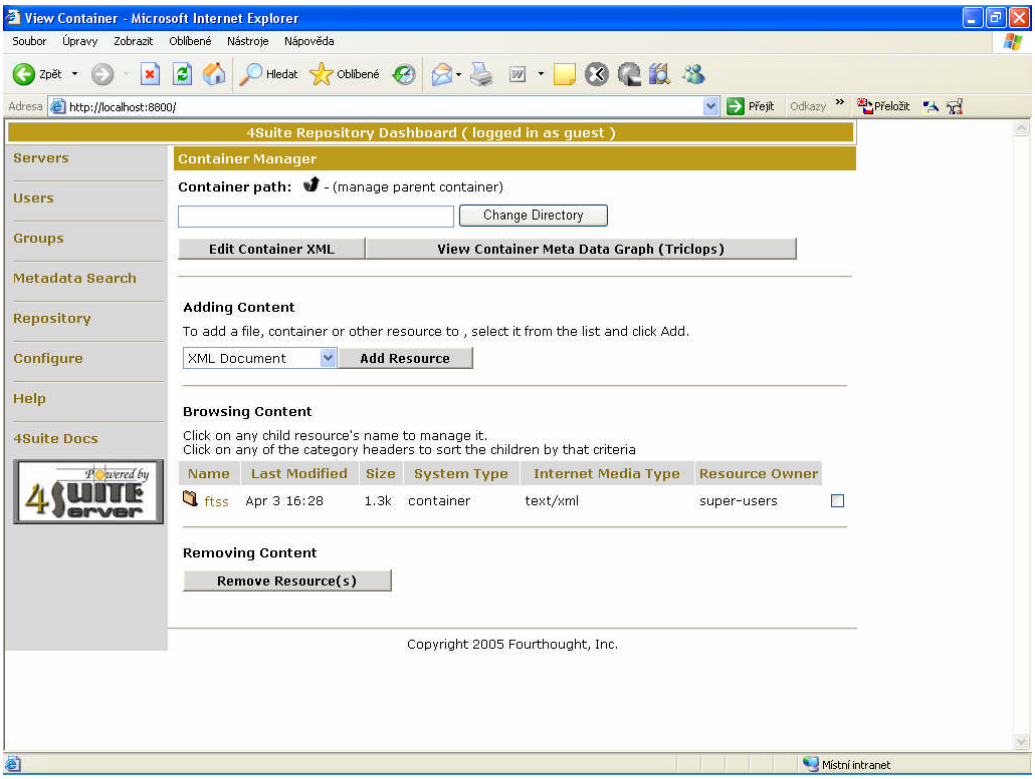
[1] Speciální jazyk vytvořený pro dotazování nad RDF daty. Přestože byl pro vývoj tohoto jazyka velkou inspirací jazyk XPath (Versa podporuje všechny XPath funkce), podobá se po své funkční stránce spíše jazyku Lisp.

```
Password: heslo
Reenter password: heslo
Add super user name (or just "enter" to pass):
Installing Repository
Create Container: /ftss
Create Container: /ftss/users
Create Container: /ftss/groups
Create User: /ftss/users/h      docDef=None
Create Group: /ftss/groups/super-users
.
.
.

...>
```

Po úspěšně dokončené inicializaci, která trvá několik minut, a spuštění pomocí dalšího příkazu:

```
...> 4ss_manager start
4SS Manager Name: uzivatelske_jmeno
Password for uzivatelske_jmeno:heslo
4ssd started (pid 1620)
```

lze využít pro práci s databázemi, správu serveru a uživatelů další pracovní prostředí, a to GUI na URL: http://localhost:8800/.



Obrázek 5.2: 4Suite – GUI na URL: http://localhost:8800/

Pro vytvoření kolekce dokumentů může uživatel využít:

- příkazové řádky – `4ss create container /nazev` (pro úspěšné vytvoření musí uživatel uvést své uživatelské jméno a heslo),
- GUI – v rozbalovacím menu části „Adding Content“ vybrat `Container` a potvrdit tlačítkem „Add Resource“, následně stačí jen uvést název kontejneru a dojde k vytvoření.

Pro práci s kolekcí je možné opět použít příkazovou řádku i GUI. Operace nad kolekcí však v případě 4Suite nejsou příliš rozsáhlé, mimo vytvoření kolekce poskytuje v obou pracovních prostředích pouze mazání stávající kolekce (`4ss delete /nazev`).

Vkládání dokumentů do kolekce dokumentů a operace s nimi

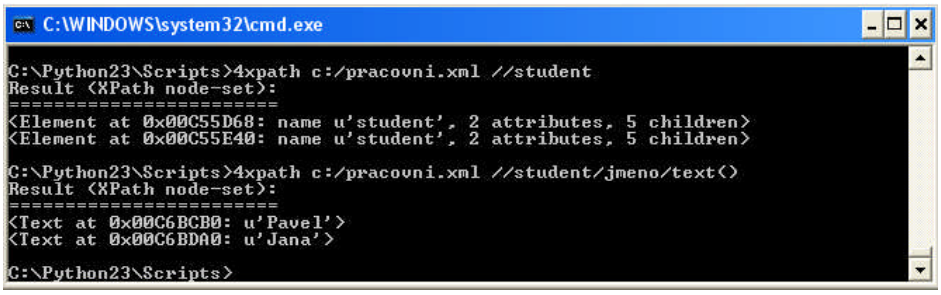
Vložení dokumentů do kolekce funguje na podobném principu jako při vytvoření kolekce dokumentů:

- pomocí příkazové řádky – `4ss create document --type=... /nazev_contejneru/nazev_dokumentu URI_zdrojoveho_dokumentu` – typem dokumentu (`--type=...`) může být například XML dokument (`xml`), XSLT dokument (`xslt`), dokument schémového jazyka Schematron (`schematron`), RDF dokument (`rdf`),
- pomocí GUI – v rozbalovacím menu části „Adding Content“ vybrat příslušný typ dokumentu (`XML Document`, `XSLT Document`, `DocumentDefinition`, `RDF Document`, `File`) a potvrdit tlačítkem „Add Resource“, následně stačí jen určit zdroj dokumentu nebo dokument přímo vytvořit.

Operace s dokumenty jsou také velice omezené, obdobně jako u kolekcí, lze dokumenty pouze mazat. V GUI je možné navíc využít editaci dokumentů.

Dotazování, aktualizace a transformace dat

Základním dotazovacím jazykem 4Suite je jazyk XPath, pro nějž je vytvořen vlastní nástroj příkazové řádky.



Obrázek 5.3: 4Suite – XPath dotazy

V rámci serveru lze ještě využít dotazovací jazyk Versa. Pro jeho využití však musí předcházet vytvoření RDF dokumentu pro XML dokumenty. Dotaz je pak vykonán přímo nad RDF dokumentem. Tuto informaci, kterou jsem získala z [29] se mi bohužel nepodařilo ověřit.

Pro aktualizaci dat lze ve 4Suite využít jazyka XUpdate, který je plně podporován. Stejně jako XPath má i XUpdate vlastní nástroj příkazové řádky. Pro aktualizaci dat pak stačí aplikovat příkaz `4xupdate --out-file=nazev_vysledneho_souboru URI_zdroje_XML URI_souboru_XUpdate`.

Naprosto stejným způsobem jako aktualizace probíhá transformace dat podporovaným jazykem XSLT – `4xslt --outfile=nazev_transformovaneho_souboru URI_zdroje_XML URI_souboru_XSLT`.

5.2.3 Geodata a 4Suite

Po úspěšném zinicializování a přihlášení se k 4Suite Serveru jsem již mohla přistoupit k vytvoření kolekce dokumentů `geodata`:

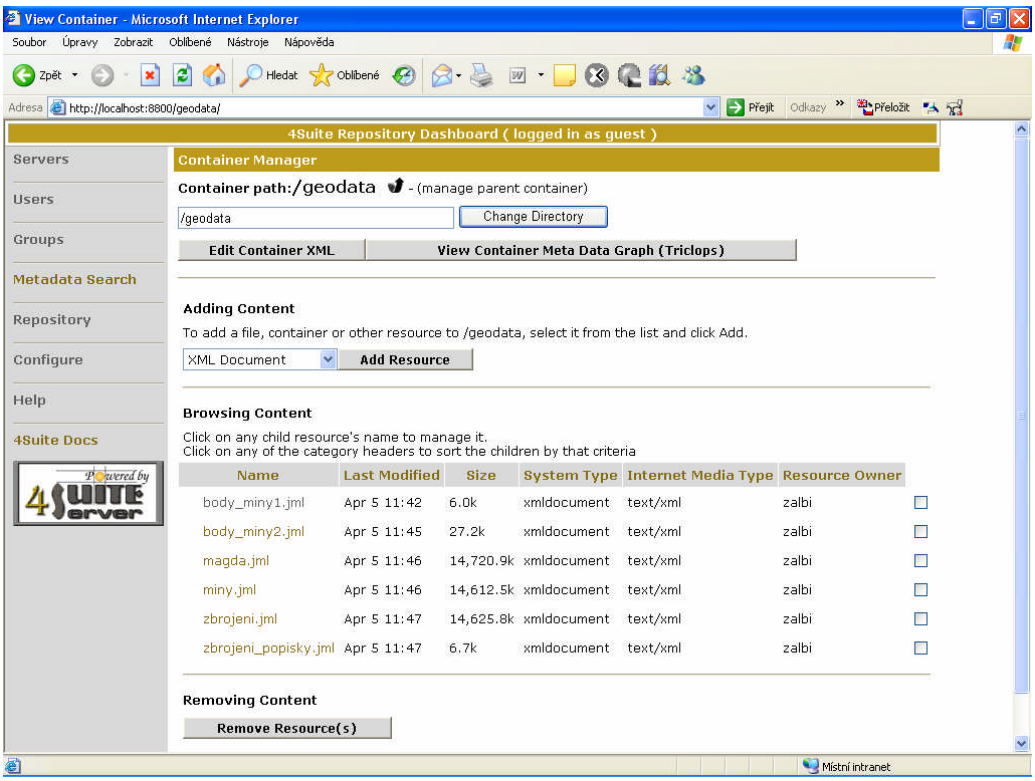
```
...> 4ss create container /geodata
4SS user name: zalbi
Password for zalbi:
Host (localhost):
Port (8803):

...>
```

Do vytvořené kolekce jsem pomocí příkazové řádky vložila všechny JML dokumenty jako XML dokumenty s příponou *.jml, což 4Suite umožňuje:

```
...> 4ss create document -type=xml /geodata/body_miny1.jml
e:/Data/body_miny1.jml
4SS user name: zalbi
Password for zalbi:
Host (localhost):
Port (8803):

...> 4ss create document -type=xml /geodata/body_miny2.jml
e:/Data/body_miny2.jml
4SS user name: zalbi
Password for zalbi:
Host (localhost):
Port (8803):

...> 4ss create document -type=xml /geodata/magda.jml e:/Data/magda.jml
4SS user name: zalbi
Password for zalbi:
Host (localhost):
Port (8803):
```

```
...> 4ss create document -type=xml /geodata/miny.jml e:/Data/miny.jml
4SS user name: zalbi
Password for zalbi:
Host (localhost):
Port (8803):

...> 4ss create document -type=xml /geodata/zbrojeni.jml
e:/Data/zbrojeni.jml
4SS user name: zalbi
Password for zalbi:
Host (localhost):
Port (8803):

...> 4ss create document -type=xml /geodata/zbrojeni_popisky.jml
e:/Data/zbrojeni_popisky.jml
4SS user name: zalbi
Password for zalbi:
Host (localhost):
Port (8803):

...>
```

Při vkládání jednotlivých dokumentů nebylo možné rozpoznat rozdíl ve velikosti vkládaných dokumentů, všechny dokumenty byly vloženy přibližně ve stejném čase (cca 2 s). Pro ověření správnosti vložení všech dokumentů (včetně jejich velikosti) stačilo využít GUI (obrázek 5.4).



Obrázek 5.4: 4Suite – kolekce dokumentů

Při dotazování jsem bohužel narazila na problém v přístupu ke kolekci dokumentů. Vlastní vyhledání dokumentu v kolekci proběhlo v pořádku, ale 4Suite Server nepracoval zřejmě tak, jak by měl, a proto k vyhodnocení dotazu kvůli chybě nemohlo vůbec dojít:

```
...> 4xpath http://localhost:8800/geodata/magda.jml //feature
XML parse error in u'http://localhost:8800/geodata/magda.jml' at line 7,
column 22: mismatched tag

...>
```

Protože 4Suite z dotazovacích jazyků podporuje XPath, neměla by vytvořená kolekce dokumentů na průběh dotazování zřejmě žádný vliv, také by se muselo pracovat pouze s jednotlivými soubory, a proto jsem přistoupila ke klasickému souborovému uložení dokumentů a XPath dotazy aplikovala na dokumenty uložené v běžném souborovém systému.

Při dotazování nepodává 4Suite žádné informace o čase potřebném k vyhodnocení dotazu, abych však získala představu o tomto údaji, měřila jsem průběh jednotlivých dotazů na stopkách. Protože mohou být reflexy člověka zpožděné, zopakovala jsem dotaz vždy pětkrát a každý čas jsem zaznamenala. Za výsledný časový údaj pak považuji průměrnou hodnotu z naměřených hodnot.

I přesto, že má databázový systém 4Suite normálně podporovat XPath, vyskytly se při testování jednotlivých XPath dotazů právě v tomto systému problémy se znakovou sadou. Při dotazování v této aplikaci bylo nutné nahradit uvozovky vymezující řetězce v dotazu jednoduchými apostrofy, a pokud se v podmínce vyskytla mezera, bylo nutné přistoupit k jiné formulaci dotazu. Druhý případ se týkal dotazu (XP_2), v němž byl původně porovnáván určitý řetězec s „United Kingdom“ a nikoli jen s „Kingdom“. Kvůli porovnání tohoto databázového systému s jinými však byly tyto změny natrvalo zohledněny u všech XPath dotazů určených k testování.

Po odstranění výše uvedených nedostatků proběhlo testování všech dotazů v pořádku:

- (XP_1) – čas potřebný k vyhodnocení dotazu = 2,078 s (měřené časy: 2,14 s; 2,06 s; 2,04 s; 2,04 s; 2,11 s)

```
...> 4xpath e:/Data/magda.jml //feature[property='CZE']/property/text()
Result (XPath node-set):
========================
<Text at 0x01AFA800: u'EZ'>
<Text at 0x01AFA800: u'CZE'>
<Text at 0x01AFA878: u'129.5'>
<Text at 0x01AFA8F0: u'Czech Republic...\xc68\x00'>
<Text at 0x01AFA968: u'Czech Republic...\xc68\x00'>
```

```
<Text at 0x01AFA9E0: u'B'>
<Text at 0x01AFAA58: u'C'>
<Text at 0x01AFAAD0: u'A'>
<Text at 0x01AFAB48: u'B'>
<Text at 0x01AFABC0: u'A'>
<Text at 0x01AFAC38: u'C'>

...>
```

- (XP_2) – čas potřebný k vyhodnocení dotazu = 2,484 s (měřené časy: 2,81 s; 3,25 s; 2,14 s; 2,11 s; 2,11 s)

```
...> 4xpath e:/Data/magda.jml //feature[contains(property[@name=
'SOVEREIGN'],'Kingdom')]/property[@name='CNTRY_NAME']/text()
Result (XPath node-set):
========================
<Text at 0x01247BC0: u'Anguilla      ...\u02d6H\x00'>
<Text at 0x01260AD0: u'Bermuda       ...\u02d6H\x00'>
<Text at 0x0194BD28: u'Cayman Islands...\u02d6H\x00'>
<Text at 0x01B05440: u'Falkland Islan...\u02d6H\x00'>
<Text at 0x01B1B468: u'Gibraltar     ...\u02d6H\x00'>
<Text at 0x01B1E288: u'Guernsey      ...\u02d6H\x00'>
<Text at 0x01F6F968: u'Man, Isle of  ...\u02d6H\x00'>
<Text at 0x01F77350: u'British Indian...\u02d6H\x00'>
<Text at 0x01F8C260: u'Jersey        ...\u02d6H\x00'>
<Text at 0x021E5080: u'Montserrat    ...\u02d6H\x00'>
<Text at 0x0238CEB8: u'Pitcairn Islan...\u02d6H\x00'>
<Text at 0x025A2BC0: u'St. Helena    ...\u02d6H\x00'>
<Text at 0x02728850: u'South Georgia ...\u02d6H\x00'>
<Text at 0x02734BE8: u'Turks and Caic...\u02d6H\x00'>
<Text at 0x028F65D0: u'United Kingdom...\u02d6H\x00'>
<Text at 0x02925AF8: u'British Virgin...\u02d6H\x00'>

...>
```

- (XP_3) – čas potřebný k vyhodnocení dotazu = 2,648 s (měřené časy: 3,59 s; 2,43 s; 2,36 s; 2,39 s; 2,47 s)

```
...> 4xpath e:/Data/magda.jml sum(//feature/property[@name='hustota'])div
(count(//feature/property[@name='hustota']))
Result (XPath number):
======================
257.388047809

...>
```

- (XP_4) – čas potřebný k vyhodnocení dotazu = 3,242 s (měřené časy: 4,24 s; 2,99 s; 2,97 s; 3,01 s; 3,00 s)

```
...> 4xpath --namespace=gml=http://www.opengis.net/gml e:/Data/magda.jml
//feature[.//gml:coordinates[contains(.,'152.258246352413,4992.1224416031
89')]]/property[@name='CNTRY_NAME']/text()
Result (XPath node-set):
========================
<Text at 0x01AFBAF8: u'Czech Republic...\u02d6H\x00'>
<Text at 0x01C62C10: u'Germany       ...\u02d6H\x00'>

...>
```

- (XP_5) – čas potřebný k vyhodnocení dotazu = 1,886 s (měřené časy: 2,81 s; 1,67 s; 1,73 s; 1,63 s; 1,59 s)

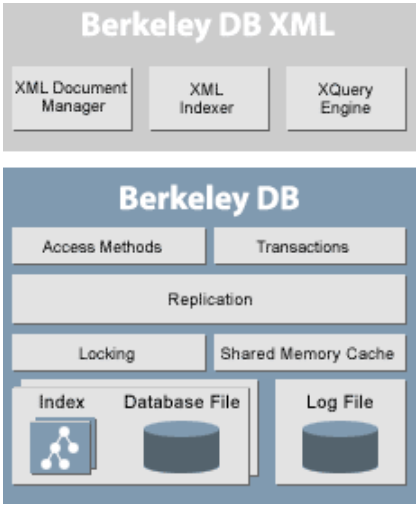
```
...> 4xpath --namespace=gml=http://www.opengis.net/gml e:/Data/magda.jml
count(//feature[boolean(geometry/gml:MultiPolygon)])
Result (XPath number):
======================
114.0

...>
```

5.3 Berkeley DB XML … databáze typu klíč-hodnota

Obecné charakteristiky

Berkeley DB XML je nativní XML databáze, která pracuje s dokumenty uloženými do tzv. kontejnerů[1], v nichž je ve většině případů ještě užité indexování pro snazší přístup k datům, pomocí jazyka XQuery.

Berkeley DB XML byla napsána v jazyce C/C++ a vystavěna nad klasickou (ne XML) databází Berkeley DB, ze které převzala veškeré rysy a vlastnosti (ACID transakce, automatické zotavování, zápis šifrovaných dat, repliky, …), a zdokonalila přidáním určitých rozšíření (parsery, indexování, XQuery,...) její funkci. Stejně jako Berkeley DB je i Berkeley DB XML pouze knihovnou, která poskytuje pro vývojáře programové API a při svém „provozu“ nepotřebuje žádnou lidskou správu.



Obrázek 5.5: Architektura – Berkeley DB XML (převzato z [31])

[1] Kontejner je ve skutečnosti kolekce XML dokumentů a informací o těchto dokumentech.

Obrovskou výhodou tohoto systému je možnost zpracování XML dat s daty, které nevycházejí z XML standardů. Přirozené ukládání a zpětné získávání XML i ne-XML dat je prováděno stejnými operacemi pro oba typy těchto dat – není vyžadováno žádné mapování ani jiné přetváření dat. Také nabízí dvě možná řešení pro ukládání velkých dokumentů. Dokumenty mohou být ponechány neporušené, bez jakéhokoli dělení (zde jsou zachovávány „white space“), nebo mohou být „rozbity“ na jednotlivé uzly, přičemž je umožněno účinné vyhledávání, a také částečná aktualizace dokumentu.

XML dokumenty jsou logicky seskupovány do kontejnerů, ve kterých je možné provést validaci.

V rámci kontejneru umožňuje Berkeley DB XML různé způsoby indexování, které velkou měrou usnadňují přístup k datům. Asi základním druhem je uzlové indexování, které umožňuje snadné vyhodnocování dotazů nad velkými dokumenty. Tento způsob indexování je standardně nastavený, ale jeho užití není povinné. Za zmínku stojí i flexibilní indexování XML uzlů, elementů, atributů a metadat, jež má také velký vliv na kvalitní a rychlé vyhledávání dat. Dalším typem indexu, který Berkeley DB XML užívá, je složený index vytvářený v průběhu probíhání programu.

Dotazování pomocí jazyka XQuery je pro typ databází, které řeší Berkeley DB XML, typické, stejně jako jazyk SQL pro relační databáze. Dotazy mohou být realizovány v rámci jednoho, nebo i více kontejnerů. Všechny dotazy jsou optimalizovány tak, aby potřeba programových aktivit byla co nejnižší.

Berkeley DB XML podporuje XQuery 1.0 a XPath 2.0, XML Namespaces, XML Schema, … a z programovacích jazyků pak C++, Java, Perl, Python, PHP, Tcl, Ruby, atd.

Maximální velikost zpracovávané databáze je 256 TB.

Berkeley DB XML může být využívána v operačních systémech Windows, Linux, BSD, Mac OS (Mac OS X), atd.

Zdroje: [31], [34], [35]

5.3.1 Instalace

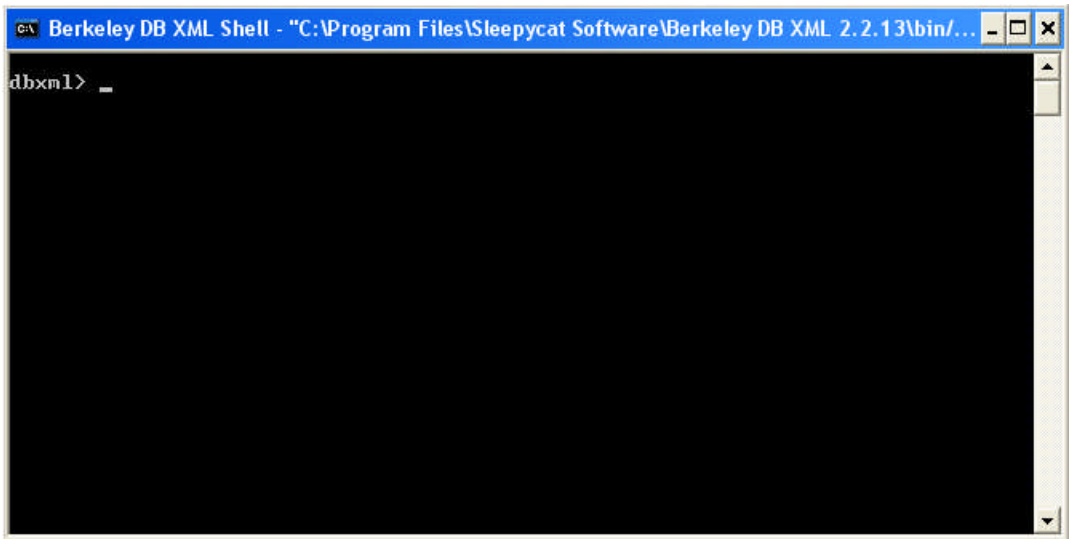
Z adresy http://www.oracle.com/technology/software/products/berkeley-db/xml/index.html stačí stáhnout pouze uživatelem zvolenou verzi programu. Já jsem zvolila přímo instalační

program pro Windows dbxml-2.2.13.msi, tj. verzi 2.2.13, a pomocí instalačního průvodce nainstalovala.

5.3.2 Pracovní prostředí a základní funkce

Pracovním prostředím databáze Berkeley DB XML je pouze příkazová řádka, která je však vzhledem k přehlednosti nápovědy a jednoduchosti příkazů plně postačující. Spuštění aplikace je možné dvojím způsobem:

- Nabídka Start => Programy => Berkeley DB XML 2.2.13 => Berkeley DB XML Shell
- C:\Program Files\Sleepycat Software\Berkeley DB XML 2.2.13\bin\dbxml.exe



Obrázek 5.6: Berkeley DB XML – pracovní prostředí

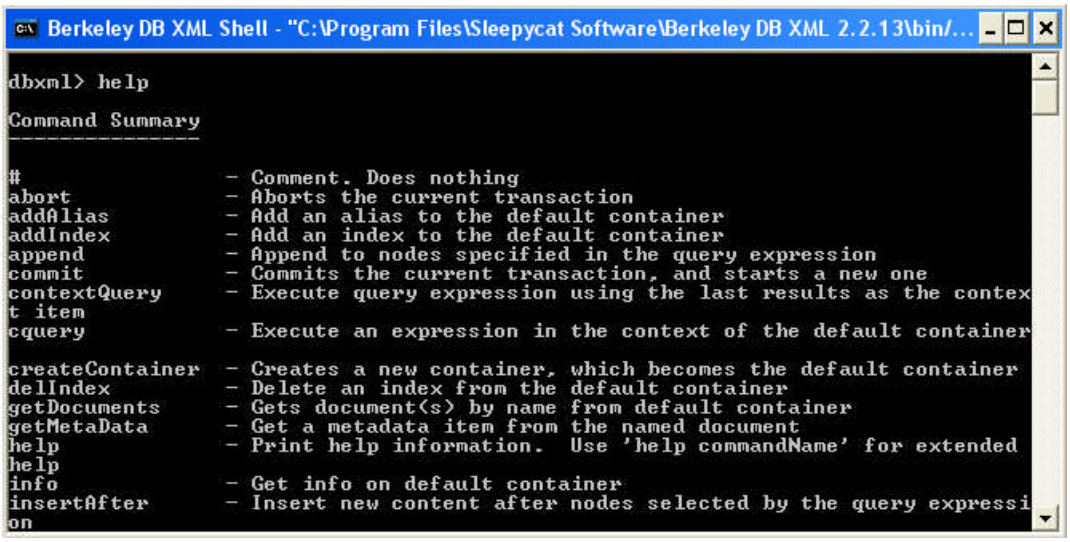
Pomocí nápovědy lze již s databázovým systémem pohodlně pracovat. Základní nápovědu zobrazí příkaz:

```
dbxml> help
```

a jakoukoli další podrobnější nápovědu k jednotlivým příkazům umožňuje pouhé doplnění daného příkazu za výraz `help`:

```
dbxml> help createContainer
```

Základní nápověda zobrazí všechny dostupné funkce (příkazy) databázového systému i s jednoduchým popisem.



Obrázek 5.7: Berkeley DB XML – nápověda

Vytvoření kolekce dokumentů a operace s již existující kolekcí

Pozn. Jak již bylo uvedeno výše Berkeley DB XML označuje jakoukoli kolekci dokumentů jako kontejner (Container).

Vytvoření kontejneru je základní počáteční operací s tímto databázovým systémem. Berkeley DB XML poskytuje čtyři možné způsoby vytvoření kontejneru. Standardním druhem kontejneru je kontejner pro uložení uzlů dokumentů s jejich indexováním (`createContainer nazev_kontejnteru in` nebo `createContainer nazev_kontejneru`), z čehož plyne jasné zaměření na datově orientované dokumenty. Další možností pro datově orientované dokumenty je obdobný kontejner jako standardní ovšem bez indexování (`createContainer nazev_kontejneru n`). Pro dokumentově orientované dokumenty jsou pak určeny především poslední dva druhy kontejnerů, a to kontejnery pro ukládání dokumentově orientovaných dokumentů bez indexování (`createContainer nazev_kontejneru d`) nebo s uzlovým indexování (`createContainer nazev_kontejneru id`).

Při tvorbě kontejneru jakéhokoli druhu si uživatel může zároveň nadefinovat, že všechny soubory do tohoto kontejneru ukládané budou validovány – `createContainer nazev_kontejneru druh_kontejneru(n,in,d,id) validate`.

S existujícím kontejnerem lze provádět jen několik operací:

- Otevření – `openContainer nazev_kontejneru`.
- Přeindexování – `reindexContainer nazev_kontejneru d` (přeindexování kontejneru s indexováním dokumentově orientovaných dokumentů),

`reindexContainer nazev_kontejneru n` (přeindexování kontejneru s uzlovým indexováním).

- Mazání – `removeContainer nazev_kontejneru`.

Vkládání dokumentů do kolekce dokumentů a operace s nimi

I při vkládání dokumentů existuje více způsobů, jak tuto důležitou operaci provést. Základní možností je vložení dokumentu vytvořením dokumentu přímo v prostředí Berkeley DB XML:

```
dbxml> putDocument nazev_dokumentu ‘
... XML dokument ... ‘ s
```

Obdobným způsobem je pak vložení dokumentu získaného z dotazu XQuery:

```
dbxml> putDocument nazev_dokumentu ‘
... XQuery ... ‘ q
```

Posledním možným způsobem vložení dokumentu je samozřejmě vložení již existujícího dokumentu:

```
dbxml> putDocument nazev_dokumentu adresa f
```

S dokumenty, jako takovými (nikoli s daty, jež zahrnují), lze nakládat pouze dvojím způsobem:

- Opětovné získání uloženého dokumentu – `getDocument` (získá všechny dokumenty uložené v otevřeném kontejneru), `getDocument nazev_dokumentu` – získané dokumenty lze zobrazit pomocí příkazu `print`.
- Odstranění dokumentu – `removeDocument nazev_dokumentu`.

Aktualizace dat

Protože Berkeley DB XML nepodporuje jazyk XUpdate, má tento databázový systém vlastní funkce určené k modifikaci a aktualizaci dat uvnitř dokumentů kontejneru:

- Vkládání uzlů – `append XPath_vyraz typ_objektu nazev_objektu obsah_objektu` (vloží „do“ uzlu); `insertBefore XPath_vyraz typ_objektu nazev_objektu obsah_objektu` (vloží „před“ uzel); `insertAfter XPath_vyraz typ_objektu nazev_objektu obsah_objektu` (vloží „za“ uzel).

  Pozn. typ_objektu = (element, attribute, text, comment, processinginstruction)

- Přejmenování uzlů – `renameNodes XPath_vyraz nove_jmeno` .
- Mazání uzlů – `removeNodes XPath_vyraz`.

- Změna obsahu uzlů – `updateNodes XPath_vyraz novy_obsah`.

Dotazování

Podporovaným dotazovacím jazykem v rámci Berkeley DB XML je XQuery, a proto je dotazování v tomto ohledu relativně jednoduché:

```
dbxml> query ‘
... XQuery ... ‘
```

Vlastní indexování

Berkeley DB XML umožňuje uživateli vložit vlastní způsob indexování, jenž může významným způsobem ovlivnit čas potřebný k vyhodnocení dotazu – `addIndex URI_jmenneho_prostoru_uzlu nazev_uzlu deklarace_indexu`.

Indexy (`deklarace_indexu`) jsou specifikovány čtyřmi částmi: typem cesty, typem uzlu, typem klíče a jedinečností. Typ cesty může nabývat hodnot `node` či `edge`, typ uzlu `node` nebo `element`, typ klíče `presence`, `equality` a `substring` a jedinečnost `none`, `boolean`, `decimal`, `double`, `time`, … např. `node-element-presence-node`.

5.3.3 Geodata a Berkeley DB XML

Protože struktura souborů s geodaty je rozhodně bližší datově orientovaným dokumentům, bylo v prvotní fázi při vytváření kolekce dokumentů nutné zvážit pouze to, zda budou uzly v kolekci indexovány či nikoli. Pro velké soubory (cca 50 MB) se indexování kvůli dalšímu zvětšení souboru příliš nedoporučuje, takovou velikost ale zdrojové soubory nemají, a proto jsem zvolila předdefinovaný typ kontejneru s indexováním, čímž se celková velikost kolekce dokumentů vyšplhala na 57 794 560 B.

```
dbxml> createContainer geodata.dbxml
Creating node storage container with nodes indexed

dbxml>
```

Podobně jako v databázovém systému 4Suite nerozlišuje ani Berkeley DB XML neznámé přípony souborů (*.jml). Všechny soubory byly tedy uloženy bez jakékoli změny v časech nepřesahujících ani u velkých souborů 2 s.

```
dbxml> putDocument body_miny1 E:/Data/body_miny1.jml f
Document added, name = body_miny1

dbxml> putDocument body_miny2 E:/Data/body_miny2.jml f
Document added, name = body_miny2

dbxml> putDocument magda E:/Data/magda.jml f
Document added, name = magda
```

```
dbxml> putDocument miny E:/Data/miny.jml f
Document added, name = miny

dbxml> putDocument zbrojeni E:/Data/zbrojeni.jml f
Document added, name = zbrojeni

dbxml> putDocument zbrojeni_popisky E:/Data/zbrojeni_popisky.jml f
Document added, name = zbrojeni_popisky

dbxml>
```

Před vlastním přístupem k dotazování bylo nutné spustit službu, kterou Berkeley DB XML poskytuje, a to výpis protokolu průběhu dotazu, ve kterém je uveden i přesný čas nutný k získání výsledku. Tento čas je uváděn vždy šestimístnou číslicí s oddělovačem desetinných čísel (nebo šestimístnou číslicí bez oddělovače) v milisekundách.

```
dbxml> setVerbose 2 2
```

Protože Berkeley DB XML podporuje mimo XPath také dotazovací jazyk XQuery, bylo možné v této databázi aplikovat všechny dotazy, XQuery i XPath. Automatický výpis výsledku dotazu Berkeley neposkytuje, pro ověření správnosti bylo nutné použít příkaz print. Všechny dokončené dotazy podaly správné výsledky, jediné odlišnosti se vyskytly u dotazů s početními úkony, kde byly výsledné hodnoty uvedeny na mnohem větší počet desetinných míst.

- (XQ_1) – čas potřebný k vyhodnocení dotazu = 2 168,68 ms

```
dbxml> query ‘
collection("geodata.dbxml")//feature[contains(property,”Czech Republic”)
or contains(property,”CZE”)]/property ‘
Query - Starting eager query execution
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - Finished eager query execution, time taken = 2168.68ms
18 objects returned for eager expression '
collection("geodata.dbxml")//feature[contains(property,”Czech  Republic”)
or contains(property,”CZE”)]/property ‘

dbxml>
```

- (XQ_2) – čas potřebný k vyhodnocení dotazu = 2 129,65 ms

```
dbxml> query ‘
collection("geodata.dbxml")//feature[property[@name="SOVEREIGN"]="United
Kingdom               "]/property[@name="CNTRY_NAME"]/ text()‘
Query - Starting eager query execution
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - Finished eager query execution, time taken = 2129.65ms
16 objects returned for eager expression '
collection("geodata.dbxml")//feature[property[@name="SOVEREIGN"]="United
Kingdom               "]/property[@name="CNTRY_NAME"]/ text()‘

dbxml>
```

- (XQ_3) – čas potřebný k vyhodnocení dotazu = 1 898,23 ms

```
dbxml> query '
for $i in collection("geodata.dbxml")//feature
where $i/property[@name="SOVEREIGN"] = "United Kingdom                "
return
$i/property[@name="CNTRY_NAME"]/text()'
Query - Starting eager query execution
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - Finished eager query execution, time taken = 1898.23ms
16 objects returned for eager expression '
for $i in collection("geodata.dbxml")//feature
where $i/property[@name="SOVEREIGN"] = "United Kingdom                "
return
$i/property[@name="CNTRY_NAME"]/text()'

dbxml>
```

- (XQ_4) – čas potřebný k vyhodnocení dotazu = 551,657 ms

```
dbxml> query '
for $i in doc("geodata.dbxml/magda")//feature
where some $j in $i/property[@name ="hustota"] satisfies ($j > 200 and $j
< 300)
order by $i/property[@name ="hustota"]
return
$i/property[@name ="hustota"]/text()'
Query - Starting eager query execution
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - Finished eager query execution, time taken = 551.657ms
13 objects returned for eager expression '
for $i in doc('geodata.dbxml/magda')//feature
... '

dbxml>
```

- (XQ_5) – čas potřebný k vyhodnocení dotazu = 1 473,37 ms

```
dbxml> query '
let $kontinenty := collection("geodata.dbxml")//feature/property[@name="CONTINENT"]
for $kontinent in distinct-values($kontinenty)
order by $kontinent
return
$kontinent'
Query - Starting eager query execution
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - Finished eager query execution, time taken = 1473.37ms
8 objects returned for eager expression '
let $kontinenty := collection("geodata.dbxml")//feature/property[@name="CONTINENT"]
... '

dbxml>
```

- (XQ_6) – čas potřebný k vyhodnocení dotazu = 1 716,18 ms

```
dbxml> query '
<prumer_hustota>{avg(doc("geodata.dbxml/magda")//feature/property[@name="
hustota"])}</prumer_hustota>,
<min_hustota>{min(doc("geodata.dbxml/magda")//feature/property[@name="hus
tota"])}</min_hustota>,
<max_hustota>{max(doc("geodata.dbxml/magda")//feature/property[@name="hus
tota"])}</max_hustota>'
Query - Starting eager query execution
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - Finished eager query execution, time taken = 1716.18ms
3 objects returned for eager expression '
<prumer_hustota>{avg(doc("geodata.dbxml/magda")//feature/property[@name="
hustota"])}</prumer_hustota>,
... '

dbxml>
```

- (XQ_7) – čas potřebný k vyhodnocení dotazu = 46 159,8 ms

```
dbxml> query '
for $i in collection("geodata.dbxml")//feature
where $i/property[@name ="CONTINENT"] = "Europe       "
return
for $j in doc("geodata.dbxml/magda")//feature
where
contains($j/property[@name="CNTRY_NAME"]/text(),substring($i/property[@na
me="CNTRY_NAME"]/text(),1,20))
return
element hustota {
attribute stat{$i/property[@name="CNTRY_NAME"]/text()},
attribute zkr {$j/property[@name="GMI_CNTRY"]/text()},
$j/property[@name="hustota"]/text()}'
Query - Starting eager query execution
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('magda',U) : [1] 4
.
.
.
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - Finished eager query execution, time taken = 46159.8ms
49 objects returned for eager expression '
for $i in collection("geodata.dbxml")//feature
... '
```

```
dbxml>
```

- (XQ_8) – čas potřebný k vyhodnocení dotazu = 226 688 ms

```
dbxml> query `
declare function local:prum_hustota ($kontinent)
{avg(
for $i in doc("geodata.dbxml/zbrojeni")//feature
where $i/property[@name ="CONTINENT"] = $kontinent
return
for $j in doc("geodata.dbxml/magda")//feature
where
contains($j/property[@name="CNTRY_NAME"]/text(),substring($i/property[@na
me="CNTRY_NAME"]/text(),1,20))
return
      $j/property[@name="hustota"])
};

<prumer_hustota kontinent="Afrika">{
local:prum_hustota("Africa       ")}
</prumer_hustota>,

<prumer_hustota kontinent="Antarktida">{
local:prum_hustota("Antarctica   ")}
</prumer_hustota>,

<prumer_hustota kontinent="Asie">{
local:prum_hustota("Asia         ")}
</prumer_hustota>,

<prumer_hustota kontinent="Australie a Oceanie">{
local:prum_hustota(("Australia    ","Oceania      "))}
</prumer_hustota>,

<prumer_hustota kontinent="Evropa">{
local:prum_hustota("Europe       ")}
</prumer_hustota>,

<prumer_hustota kontinent="Jizni a Severni Amerika">{
local:prum_hustota(("South America","North America"))}
</prumer_hustota>'
Query - Starting eager query execution
Query - geodata.dbxml – D('zbrojeni',U) : [1] 6
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('zbrojeni',U) : [1] 6
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('magda',U) : [1] 4
.
.
.
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - Finished eager query execution, time taken = 226688ms
6 objects returned for eager expression '
declare function local:prum_hustota ($kontinent)
```

```
... ’

dbxml>
```

- (XQ_9) – čas potřebný k vyhodnocení dotazu = 2 414,64 ms

```
dbxml> query ‘
declare namespace gml="http://www.opengis.net/gml";
let $souradnice := "152.258246352413,4992.122441603189"
for $i in collection("geodata.dbxml")//feature
where some $j in $i[property/@name="CONTINENT"]//gml:coordinates
satisfies contains($j/text(),$souradnice)
return
element stat{$i/property[@name!="kod"]}’
Query - Starting eager query execution
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - Finished eager query execution, time taken = 2414.64ms
2 objects returned for eager expression '
declare namespace gml="http://www.opengis.net/gml";
... ’

dbxml>
```

- (XQ_10) – čas potřebný k vyhodnocení dotazu = 1 054,47 ms

```
dbxml> query ‘
declare namespace gml="http://www.opengis.net/gml";
element multipolygony {
attribute celkovy_pocet {
count(doc("geodata.dbxml/magda")//feature[boolean(geometry/gml:MultiPolyg
on)])},
for $i in doc("geodata.dbxml/magda")//feature[boolean(geometry/gml:Mu
ltiPolygon)]
return
element stat {
attribute pocet_polygonu {count($i//gml:Polygon)},
$i/property[@name="CNTRY_NAME"]/text()
}}’
Query - Starting eager query execution
Query - geodata.dbxml – D(‘magda’,U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D(‘magda’,U) : [1] 4
Query - geodata.dbxml – D(‘magda’,U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D(‘magda’,U) : [1] 4
Query - Finished eager query execution, time taken = 1054.47ms
1 objects returned for eager expression '
declare namespace gml="http://www.opengis.net/gml";
... ’

dbxml>
```

- (XQ_11) – tento dotaz pro nedostatek paměti nemohl proběhnout

```
dbxml> query ‘
declare namespace gml="http://www.opengis.net/gml";
<sousedni_staty nazev="Ceska republika">
{
```

```
let $retezec := doc("geodata.dbxml/magda")//feature[property="CZE"]//gml:
coordinates/text()
let $souradnice1_0 := substring($retezec,0,30)
for $k in (0 to string-length($retezec) idiv 40)
let $souradnice1 := substring(substring-after(substring($retezec,$k*40,15
0),"                      "),0,30)
let $souradnice2 := substring(substring-after(substring($retezec,($k+1)*4
0,150),"                       "),0,30)
for $i in doc(("geodata.dbxml/magda")//feature
where some $j in $i//gml:coordinates satisfies (
if($k = 0) then (contains($j/text(),$souradnice1_0) and not(contains($j/t
ext(),$souradnice2 )))
else(contains($j/text(),$souradnice1) and not(contains($j/text(),$souradn
ice2))))
return
element nazev_statu {$i/property[@name ="CNTRY_NAME"]/text()}
}
</sousedni_staty>'
Query - Starting eager query execution
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('magda',U) : [1] 4
.
.
.
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('magda',U) : [1] 4
stdin:256: query failed, Error: BaseMemoryManager::allocate(): Out of memory

dbxml>
```

- (XQ_12) – tento dotaz pro nedostatek paměti nemohl proběhnout

```
dbxml> query '
declare namespace gml="http://www.opengis.net/gml";
declare function local:porovnani($retezec, $X_0 ,$Y_0 )
{
for $k in (0 to string-length($retezec) idiv 40)
let $souradnice := substring(substring-after(substring($retezec,$k*40,150
),"                      "),0,35)
let $X := number(substring-before($souradnice,","))
let $Y := number(substring-after($souradnice,","))
where (($X - $X_0)*($X - $X_0) + ($Y - $Y_0)*($Y - $Y_0))<(1000*1000)
return
element souradnice{
attribute x {$X},
attribute y {$Y}}
};
declare function local:oblast($i, $X_0 ,$Y_0 )
{
for $LinearRing in $i/geometry//gml:LinearRing
let $retezec := $LinearRing//gml:coordinates/text()
return
local:porovnani($retezec,$X_0 ,$Y_0 )
};
let $X_0 := 500
```

```
let $Y_0 := 5000
return
element okruh_1000km{
attribute X_0 { $X_0 },
attribute Y_0 { $Y_0 },
for $i in doc("geodata.dbxml/magda")//feature
return
if (boolean(local:oblast($i,$X_0 ,$Y_0 )))
then(
<nazev_statu>{$i/property[@name ="CNTRY_NAME"]/text()}</nazev_statu>
)
else ()
}'
Query - Starting eager query execution
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('magda',U) : [1] 4
stdin:256: query failed, Error: BaseMemoryManager::allocate(): Out of memory

dbxml>
```

Aby byl průběh dotazování pomocí jazyk XPath korektní vůči databázovému systému 4Suite, byly všechny XPath dotazy převedeny na jednoduché XQuery dotazy, ke změně došlo pouze definováním souboru, nad kterým tento dotaz má proběhnout. Pokud by k této úpravě nedošlo, neproběhl by žádný dotaz, protože databázový systém Berkeley DB XML nerozpozná, k jakým datům se dotaz bez udání zdrojových dat vztahuje – nelze u něho tedy předpokládat, že bez udání zdroje by byl dotaz aplikován na celou aktuální kolekci.

Další korekce, která musela být v XPath dotazech provedena byla změna apostrofů, označující řetězce, na uvozovky. Apostrof je v rámci Berkeley DB XML užíván k vymezení pole dotazu, proto použití apostrofu uvnitř zdrojového kódu dotazu tento systém nepovoluje.

- (XP_1) – čas potřebný k vyhodnocení dotazu = 792,896 ms

```
dbxml> query '
doc("geodata.dbxml/magda")//feature[property="CZE"]/property/text()'
Query - Starting eager query execution
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D('magda',U) : [1] 4
Query - Finished eager query execution, time taken = 792.896ms
11 objects returned for eager expression '
doc("geodata.dbxml/magda")//feature[property="CZE"]/property/text()'

dbxml>
```

- (XP_2) – čas potřebný k vyhodnocení dotazu = 730,648ms

```
dbxml> query '
doc("geodata.dbxml/magda")//feature[contains(property[@name="SOVEREIGN"],
"Kingdom")]/property[@name="CNTRY_NAME"]/text()'
```

```
Query - Starting eager query execution
Query - geodata.dbxml – D(‘magda’,U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D(‘magda’,U) : [1] 4
Query - Finished eager query execution, time taken = 730.648ms
16 objects returned for eager expression '
doc("geodata.dbxml/magda")//feature[contains(property[@name="SOVEREIGN"],
"Kingdom")]/property[@name="CNTRY_NAME"]/text()’

dbxml>
```

- (XP_3) – čas potřebný k vyhodnocení dotazu = 1 406,54ms

```
dbxml> query ‘
sum(doc("geodata.dbxml/magda")//feature/property[@name="hustota"])div(cou
nt(doc("geodata.dbxml/magda")//feature/property[@name="hustota"]))’
Query - Starting eager query execution
Query - geodata.dbxml – D(‘magda’,U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D(‘magda’,U) : [1] 4
Query - geodata.dbxml – D(‘magda’,U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D(‘magda’,U) : [1] 4
Query - Finished eager query execution, time taken = 1406.54ms
1 objects returned for eager expression '
sum(doc("geodata.dbxml/magda")//feature/property[@name="hustota"])div(cou
nt(doc("geodata.dbxml/magda")//feature/property[@name="hustota"]))’

dbxml>
```

- (XP_4) – čas potřebný k vyhodnocení dotazu =1 735,60 ms

```
dbxml> query ‘
declare namespace gml="http://www.opengis.net/gml";
doc("geodata.dbxml/magda")//feature[.//gml:coordinates[contains(.,"152.25
8246352413,4992.122441603189")]]/property[@name="CNTRY_NAME"]/text()’
Query - Starting eager query execution
Query - geodata.dbxml – D(‘magda’,U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D(‘magda’,U) : [1] 4
Query - Finished eager query execution, time taken = 1735.60ms
2 objects returned for eager expression '
declare namespace gml="http://www.opengis.net/gml";
doc("geodata.dbxml/magda")//feature[.//gml:coordinates[contains(.,"152.25
8246352413,4992.122441603189")]]/property[@name="CNTRY_NAME"]/text()’

dbxml>
```

- (XP_5) – čas potřebný k vyhodnocení dotazu = 593,325ms

```
dbxml> query ‘
declare namespace gml="http://www.opengis.net/gml";
count(doc("geodata.dbxml/magda")//feature[boolean(geometry/gml:MultiPolyg
on)])’
Query - Starting eager query execution
Query - geodata.dbxml – D(‘magda’,U) : [1] 4
Query - geodata.dbxml - U : [6] 2 3 4 5 6 7
Query - geodata.dbxml – D(‘magda’,U) : [1] 4
Query - Finished eager query execution, time taken = 593.325ms
1 objects returned for eager expression '
```

```
declare namespace gml="http://www.opengis.net/gml";
count(doc("geodata.dbxml/magda")//feature[boolean(geometry/gml:MultiPolygon)])'

dbxml>
```

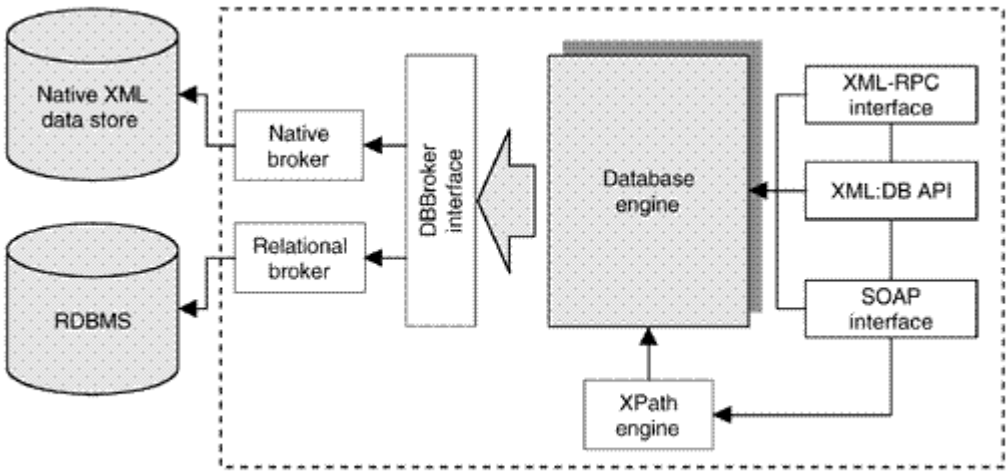
5.4 eXist … databáze s vlastním modelem

Obecné charakteristiky

Databáze eXist je celá napsána v programovacím jazyce Java. Může běžet jako nezávislá databáze, nebo být vložena do jiné aplikace (knihovna Javy, webová aplikace).

Největší důraz je kladen na účinné indexově založené dotazování. eXist byl navržen k řešení XPath dotazů při používání indexů pro všechny uzly typu element, text a atribut.

Vývoj této databáze začal v lednu 2001 a stále je zdokonalována.



Obrázek 5.8: Architektura – eXist (převzato z [4])

Data jsou ukládána do B+ stromů[1] nebo stránkovacích souborů. Uzly jsou uchovávány v DOM.

U dokumentů, se kterými eXist pracuje, není požadováno, aby splňovaly nějaké určité schéma nebo přesný typ dokumentu. Toto opatření vychází i z toho, že mnoho dokumentů nemá přesné vymezení, nebo má DTD, ale bohužel, psaný pouze pro sebe.

[1] B+ stromy jsou speciálním případem B stromů. Na rozdíl od B stromů mají všechna data uložená v listech, vnitřní uzly obsahují jen klíče a ukazatele. Všechny listy leží na stejné, nejnižší, úrovni, zároveň jsou také propojené spojovým seznam, což umožňuje snadné vyhledávání.

Dokumenty jsou hierarchicky uspořádané v kolekcích, jejich celkový počet je, dalo by říci, neomezený, poněvadž maximální počet dokumentů ukládaný do databáze je 2^{31}. Každá kolekce může obsahovat různé typy dokumentů o libovolné velikosti. Uživatelé se pak mohou dotazovat na kolekci pomocí XPath, buď podle jejího hierarchického uspořádání, nebo na všechny dokumenty v ní obsažené.

Pro kvalitní dotazování používá eXist numerické indexování. Toto indexování umožňuje odkazovat přímo na DOM uzly, ale také dokáže velice rychle odvodit vztah mezi jednotlivými uzly, jako např. vztah rodič-dítě, předek-potomek. Indexy jsou přiřazeny všem prvkům v dokumentu, tj. elementům, atributům, textů i komentářů. Všechny indexy jsou pak spravovány v databázi. Samozřejmě je také možné omezit full-textové indexování přímo na nějakou část dokumentu.

Dotazy jsou řešeny přímo podle XPath výrazu, který odkazuje přímo na určitý uzel dokumentu, na rozdíl od obvyklých přístupů, kdy se strom dokumentu prochází zdola nahoru nebo shora dolů.

eXist také umožňuje pomocí různých rozšíření full-textové dotazování, určené především pro dokumentově orientované dokumenty.

Aktualizace je prováděna pomocí XUpdate a aktualizačními rozšířeními XQuery.

Databáze si vede „deník“ o provedených transakcích a pokud dojde k havárii, jsou provedeny zapsané transakce a nedokončené se odstraní.

Používané XML standardy, které tento systém využívá, jsou XQuery 1.0, XPath, XInclude (některé prvky chybí), XPointer (z části), XSLT (přes rozšíření XQuery), XUpdate. K vývoji aplikací slouží XML DB:API.

Vzdálený přístup podporuje eXist pomocí HTTP/REST, XML-RPC, SOAP, WebDAV.

Operačním systémem, nad kterým by eXist měl pracovat, může být jakýkoli. Testy byly prováděny v operačních systémech Linux, FreeBSD (verze 5.4 a 6.0), Solaris (Sun Solaris 10), Windows (Windows 2000, Windows XP, Windows XP Server Edition).

Zdroje: [4], [43]

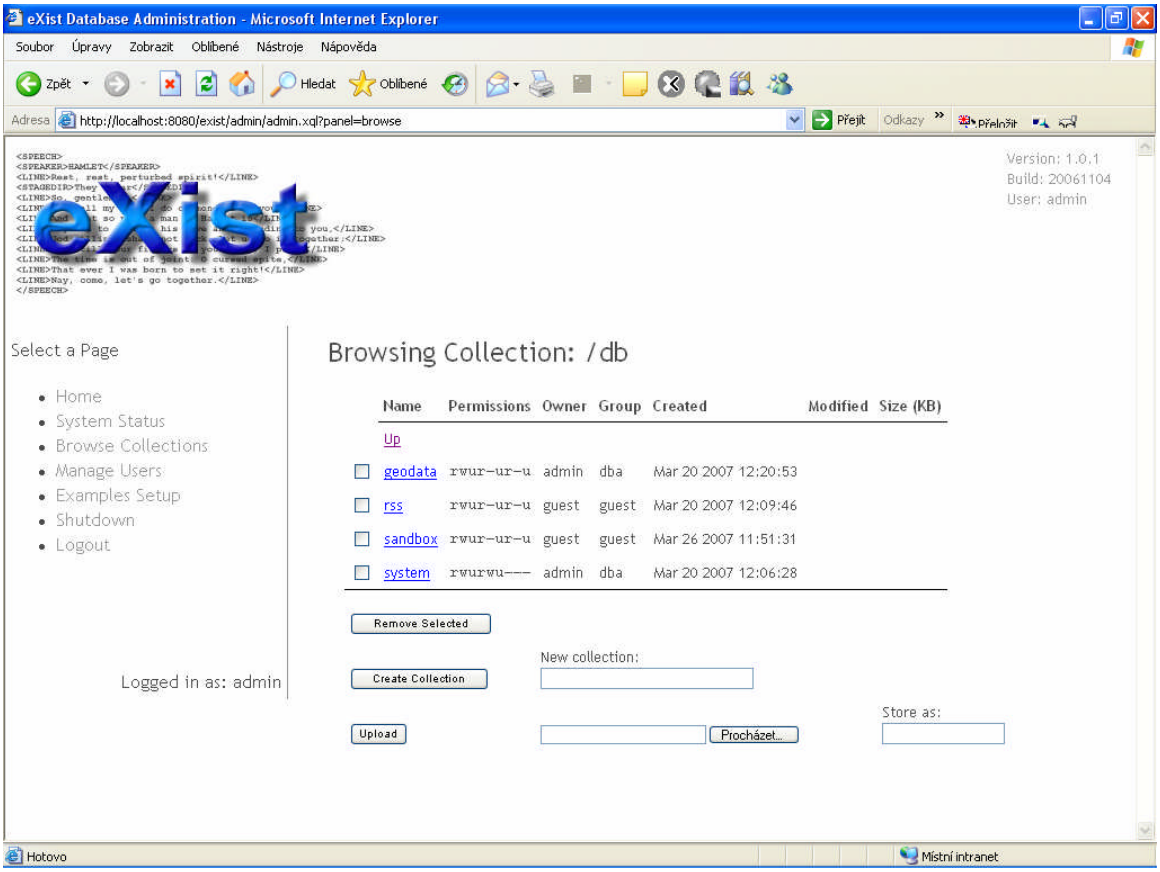
5.4.1 Instalace

Na internetové adrese http://exist.sourceforge.net/index.html#download si uživatel může zvolit verzi programu. Já jsem se rozhodla pro eXist verze 1.0 a proto jsem stáhla soubor eXist-1.0.1-build4311.jar. (Pro spuštění instalačního programu je nutné mít Java 2 SDK 1.4.2 nebo vyšší.). Dvojklikem na stažený soubor se spustí instalační průvodce, pomocí něhož se jednoduše eXist nainstaluje.

Pro správné fungování programu jsem si nainstalovala Apache server. Využila jsem jednoduchého způsobu: Na adrese http://www.slunecnice.cz/product/PHP-Triad/ jsem si stáhla phptriad2-2-1.exe, což je instalační program, který nainstaluje webserver Apache s podporou PHP a zároveň také databázový server MySQL.

5.4.2 Pracovní prostředí a základní funkce

eXist poskytuje uživatelům dva druhy pracovního prostředí. Prvním jednodušším pracovním prostředím je webové rozhraní na http://localhost:8080/exist/, kde po přihlášení jako administrátor databáze může uživatel vykonávat základní správu databáze a jednoduché dotazování.

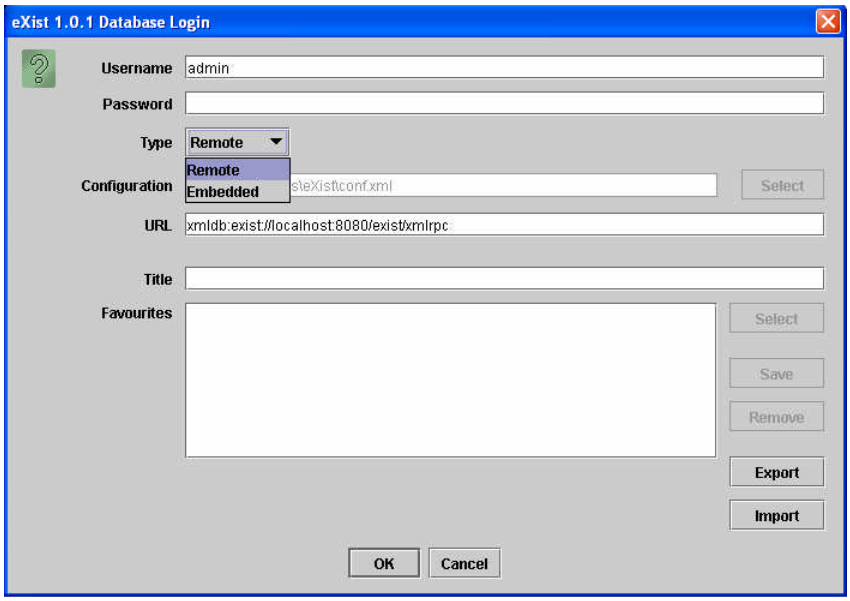


Obrázek 5.9: eXist – webové rozhraní

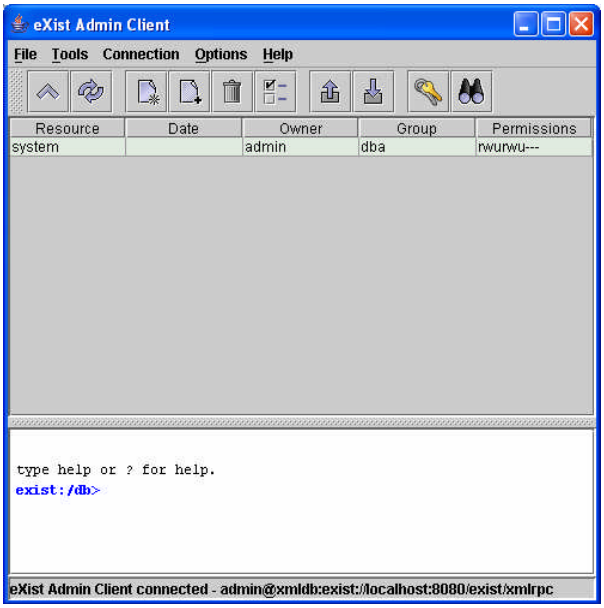
Druhým pracovním prostředím je administrátorské Java rozhraní, spustitelné přes:

-  na http://localhost:8080/exist/index.xml,
- Nabídka Start => Programy => eXist XML Database => eXist Client Shell,
- C:\Program Files\eXist\start.jar,

které umožňuje více administrátorských činností, jako například nastavení zabezpečení, správu uživatelů či zálohování a obnovu dat. Po spuštění si uživatel (klient) musí určit způsob připojení k databázi (obrázek 5.10). Jednou možností je vzdálené připojení k serveru (`Remote`) určeného URI v poli `URL`. Druhým způsobem je připojení k databázi v místním systému (`Embedded`), tedy přímo v systému klienta.



Obrázek 5.10: eXist – připojení k databázi



Obrázek 5.11: eXist – Java rozhraní

V dalším popisu se zaměřím pouze na administrátorské Java rozhraní, poněvadž operace přes webové rozhraní jsou velmi podobné.

Vytvoření kolekce dokumentů a operace s již existující kolekcí

Kolekci dokumentů vytváří eXist třemi možnými způsoby:

- příkazem v příkazové řádce – `mkcol „nazev_kolekce“`,
- pomocí ikony,
- pomocí nástroje Create Collection v nabídce File.

Operace nad již vytvořenou kolekcí, tj. mazání, přejmenování, kopírování a vkládání, jsou stejné jako operace s uloženými dokumenty. Otevření kolekce se provádí „dvojklikem“ na příslušnou kolekci v poli s jednotlivými kolekcemi (popřípadě soubory) nebo pomocí příkazu `cd „nazev_kolekce“`.

Vkládání dokumentů do kolekce dokumentů a operace s nimi

Princip vkládání dokumentů je stejný jako při vytváření nové kolekce dokumentů, tj:

- příkazem v příkazové řádce – `put „adresa_dokumentu“`,
- pomocí ikony,
- pomocí nástroje Store files/directories v nabídce File.

Ostatní operace jako mazání, přejmenování, kopírování a vkládání lze realizovat přes nabídku File a příslušné nástroje nebo také pomocí příkazů zjištěných pomocí „?“ v příkazové řádce.

Stručný popis ikon

 – umožňuje přechod na kolekci „rodič“.

 – obnoví pohled na kolekci dokumentů.

 – vytvoří novou kolekci dokumentů.

 – vloží dokument/-y do kolekce dokumentů.

 – odstraní vybrané dokumenty, kolekce dokumentů.

 – umožňuje editaci vlastností vybraných dokumentů, kolekcí dokumentů (oprávnění k užívání, vlastník, …).

 – vytvoří zálohu.

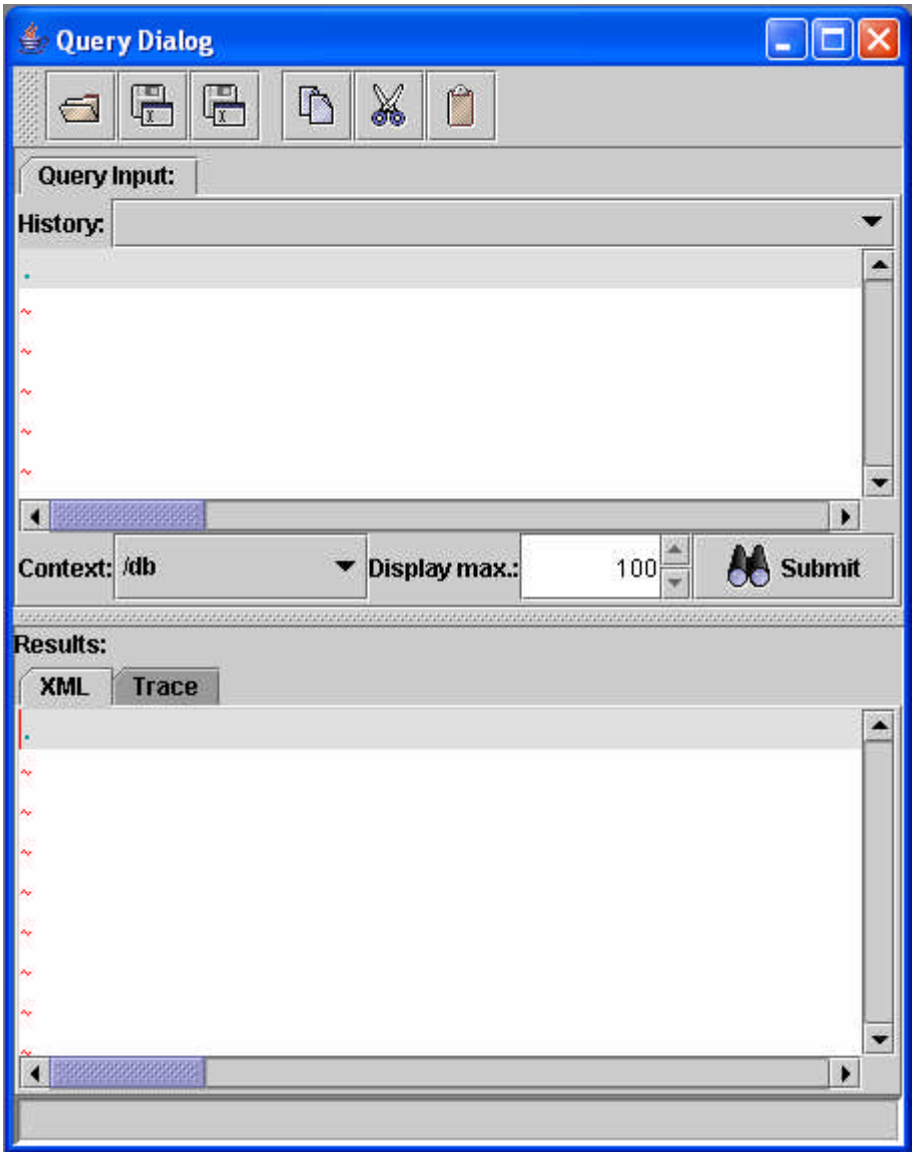
 – obnoví dokumenty ze zálohy.

 – spravuje uživatele.

 – zobrazí dotazovací dialog.

Dotazování a aktualizace dat

Podporovanými dotazovacími jazyky v eXist jsou XQuery a samozřejmě XPath. Pro vlastní aplikaci těchto dotazovacích jazyků je nejvhodnější využití dotazovacího dialogu (obrázek 5.12), který mimo jiné umožňuje otevření již existujícího souboru s dotazem.



Obrázek 5.12: eXist – dotazovací dialog

Dotazovací dialog lze také aplikovat při aktualizaci pomocí rozšíření nad XQuery, které umožňuje:

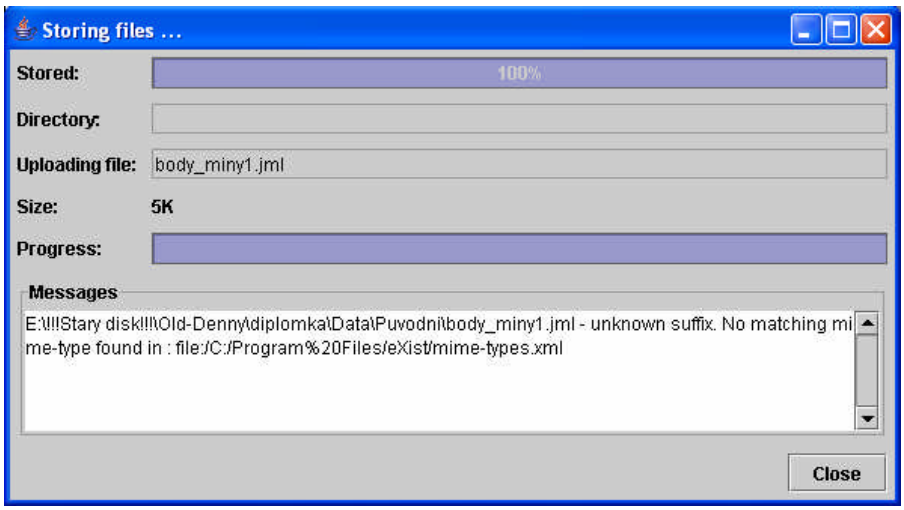
- vkládání – `update insert „objekt“ (into | following | preceding) „XPath_vyraz“`,
- přejmenování – `update rename „XPath_vyraz“ as „nove_jmeno“`,
- nahrazování – `update replace „XPath_vyraz“ with „objekt“`,
- aktualizaci obsahu – `update value „XPath_vyraz“ with „XPath_vyraz“`,
- mazání – `update delete „XPath_vyraz“`.

Pro aktualizace pomocí jazyka XUpdate, který je podle [43] podporován, samozřejmě nelze použít dotazovací dialog, protože jazyk XUpdate je od XQuery velmi vzdálený, proto se využívá klasické příkazové řádky:

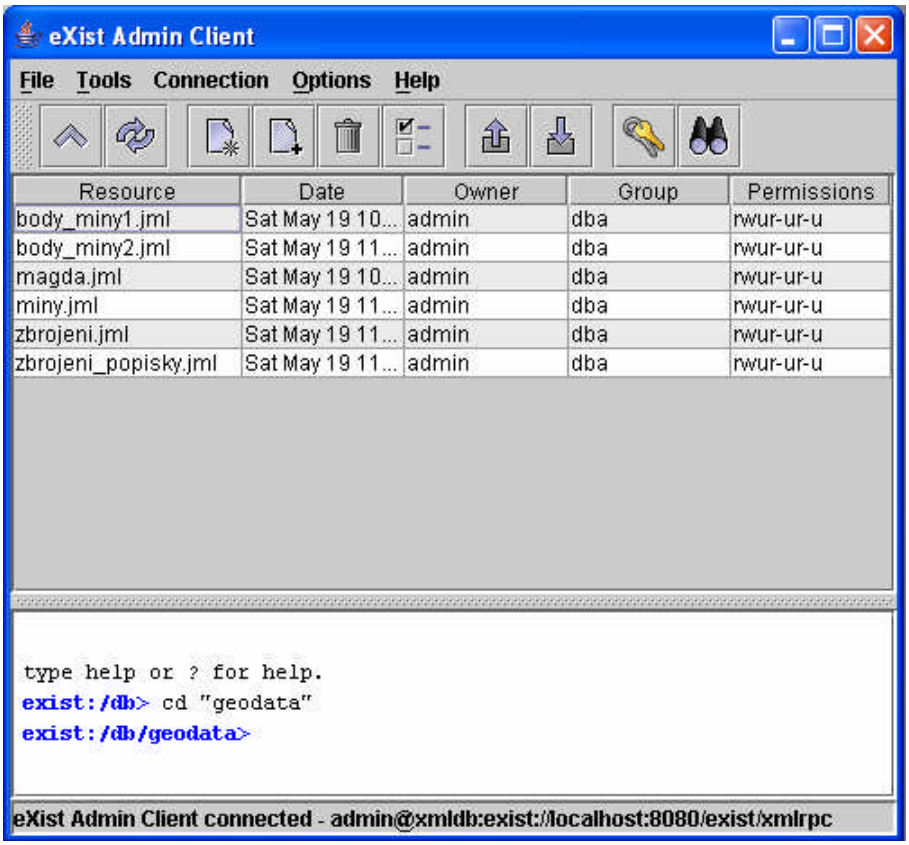
```
bin/client.bat –c /db/nazev_kolekce/nazev_dokumentu –X xupdate.xml
```

5.4.3 Geodata a eXist

Po připojení k databázovému systému a následném vytvoření kolekce `geodata` jsem již mohla v klidu přistoupit k uložení dokumentů do kolekce. Protože je pracovním prostředím GUI, které využívá běžných zvyklostí při vybírání, načítání a otevírání souborů, neočekávala jsem žádné větší komplikace. Bohužel se ale vyskytly při pokusu načtení souboru s příponou *.jml (obrázek 5.13). Tuto příponu nebyl eXist ochoten akceptovat, protože nepovažal tento soubor za XML dokument. Pro odstranění této chyby stačilo pouze uvést příponu *.jml v souboru „mime-types.xml“. Následně už nebylo žádných překážek a všechny soubory byly v pořádku vloženy do připravené kolekce.



Obrázek 5.13: eXist – chyba při ukládání souboru



Obrázek 5.14: eXist – kolekce dokumentů

Čas potřebný k uložení různých souborů nebyl tak jednoznačný jako v případě předchozích dvou databázových systémů – čas uložení menších souborů (body_miny1.jml, body_miny2.jml, zbrojeni_popisky.jml) nepřekročil 2 s, ale větší soubory (magda.jml, miny.jml, zbrojeni.jml) potřebovaly ke svému uložení cca 30 s.

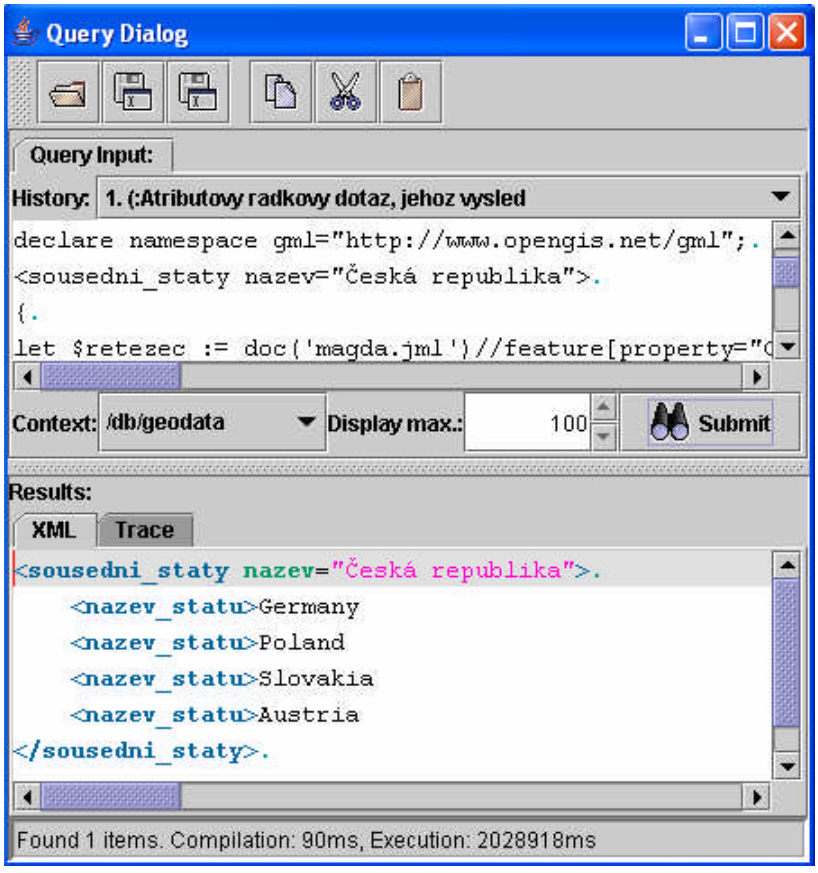
Způsob dotazování byl v prostředí eXist velice triviální. V dotazovacím dialogu (obrázek 5.12) stačilo vždy jen načíst příslušný xquery-soubor s dotazem a dotaz nechat proběhnout. Ve spodním poli dialogu se objevil výsledek dotazu a čas bylo možné zjistit na spodní liště dialogu (obrázek 5.14). Všechny XQuery dotazy proběhly a výsledky byly v naprostém pořádku.

- (XQ_1) – čas potřebný k vyhodnocení dotazu = 2 454 ms.
- (XQ_2) – čas potřebný k vyhodnocení dotazu = 240 ms.
- (XQ_3) – čas potřebný k vyhodnocení dotazu = 68 538 ms.
- (XQ_4) – čas potřebný k vyhodnocení dotazu = 4 286 ms.
- (XQ_5) – čas potřebný k vyhodnocení dotazu = 661 ms.
- (XQ_6) – čas potřebný k vyhodnocení dotazu = 271 ms.
- (XQ_7) – čas potřebný k vyhodnocení dotazu = 289 816 ms.

- (XQ_8) – čas potřebný k vyhodnocení dotazu = 1 257 899 ms.
- (XQ_9) – čas potřebný k vyhodnocení dotazu = 9 543 ms.
- (XQ_10) – čas potřebný k vyhodnocení dotazu = 531 ms.
- (XQ_11) – čas potřebný k vyhodnocení dotazu = 2 028 918 ms.
- (XQ_12) – čas potřebný k vyhodnocení dotazu = 24 762 647 ms.

Pro aplikaci XPath dotazů bylo nutné stejně jako v Berkeley DB XML upřesnit zdrojový soubor pro dotaz, ale v tomto případě to bylo nutné z důvodu, že eXist XPath dotaz zpracoval nad celou kolekcí, tudíž se výsledné hodnoty lišily od předpokládaných hodnot pro jeden soubor. Po zavedení definice žádoucího dokumentu byly všechny výsledky již korektní.

- (XP_1) – čas potřebný k vyhodnocení dotazu = 74 ms.
- (XP_2) – čas potřebný k vyhodnocení dotazu = 108 ms.
- (XP_3) – čas potřebný k vyhodnocení dotazu = 138 ms.
- (XP_4) – čas potřebný k vyhodnocení dotazu = 58 ms.
- (XP_5) – čas potřebný k vyhodnocení dotazu = 13 389 ms.



Obrázek 5.15: eXist – ukázka vyhodnocení dotazu

5.5 Zhodnocení použitých databází

Na první pohled by se mohlo zdát, že všechny použité nativní XML databázové systémy by bylo možné, samozřejmě bez velkých nároků, využít pro ukládání jakýchkoli dat, tedy i geodat. Všechny tři databázové systémy umožňovaly snadné vytvoření kolekcí dokumentů, podporovaly alespoň jeden dotazovací jazyk, aktualizace byla v systémech také zajištěna (v některých databázích lépe, v jiných hůře) a ani správa dokumentů a samotné databáze nebyla příliš zanedbána. Bohužel ale, tyto základní ukazatele neukazují vhodnost dané databáze k využití pro ukládání geodat.

5.5.1 Obecné poznatky a závěry o databázi 4Suite

Databáze 4Suite byla první z aplikovaných databázových systémů. Její základní nedostatek byl hned z počátku zřejmý – databáze 4Suite nepodporuje dotazovací jazyk XQuery ani žádný jiný obdobně vyvinutý dotazovací jazyk, proto nemůže při dotazování poskytovat ani trochu podobné výsledky jako při dotazování v prostorových databázích.

Podporovaným dotazovacím jazykem je jazyk XPath, který by ale mohl být při jednoduchých úkonech dostačující a ve spojení s 4Suite Serverem by mohl získávat pohodlné výsledky. 4Suite Server sice umožňoval vytvoření kolekce dokumentů a z části i správu uložených dokumentů, ale bohužel nebylo možné této nadstavby využít při vlastní aplikaci dotazu, a tak mohly být dotazy aplikovány pouze na dokumenty uložené v běžných souborových úložištích.

Pro práci umožňoval 4Suite společně s 4Suite Serverem používat tři pracovní prostředí: klasickou příkazovou řádku, pracovní prostředí programovacího jazyka Python a GUI serveru. Protože podle mého názoru nemusí být uživatel ani správce databáze zdárným programátorem, testovala jsem 4Suite pouze v prostředí příkazové řádky a GUI. Příkazová řádka byla pro práci, i přes chudou nápovědu, příjemným pracovním prostředím bez zjevných nedostatků. V GUI se vyskytovaly nesčetné problémy, jako jeden z nejnápadnějších bych uvedla nemožnost mazání uložených dokumentů (tuto operaci bylo možné provést pouze z příkazové řádky), proto nebylo možné toto prostředí příliš využívat.

Obrovskou předností tohoto systému je podpora XSLT a XUpdate. Právě díky této velké výhodě bych 4Suite pro ukládání geodat nezavrhovala, a pokud by bylo možné načítat soubory i z kolekcí dokumentů na serveru (tuto skutečnost znemožňovala serverová

chyba), bych spíše věřila, že by měl tento systém velkou budoucnost alespoň v oblastech transformací XML souborů s geodaty.

5.5.2 Obecné poznatky a závěry o databázi Berkeley DB XML

Dalším vybraným zástupcem z nativních XML databázových systémů byl Berkeley DB XML. Tato databáze je velice dobře uzpůsobená práci s různými XML i ne XML soubory uloženými v kolekci dokumentů, což by mohlo být velmi vhodné například ve spojení geodat (ve formátu XML) s rastry.

Protože z dotazovacích jazyků podporuje Berkeley jak XPath, tak XQuery, může teoreticky v dotazování, právě díky jazyku XQuery, podat také velmi dobré výsledky.

Z ostatních XML technologií, které se týkají nějakým způsobem přímo dokumentů, nepodporuje Berkeley DB XML bohužel žádný, tedy ani XSLT ani XUpdate. Aktualizace je zajištěna vestavěnými funkcemi a pro transformaci dat do jiné podoby musí každý uživatel sáhnout do jiného systému. Pro geografická data je nemožnost využití XSLT dost nevýhodná, ale pro celkové využití Berkeley DB XML pro ukládání těchto dat zase není prioritou, tudíž i přes tento nedostatek by mohla být databáze Berkeley DB XML pro ukládání geodat snadno, a myslím si, že vcelku úspěšně, využita.

Pracovní prostředí by se mohlo mnohým uživatelům jevit jako velice strohé, však se také jedná pouze o prostředí příkazové řádky, ale z vlastní zkušenosti mohu s klidným svědomím tvrdit, že GUI je zde zcela zbytečné, protože díky kvalitní nápovědě a jednoduchým věcným příkazům si každý uživatel brzy osvojí základní pracovní postupy a zdánlivá počáteční nepřehlednost zcela zmizí.

5.5.3 Obecné poznatky a závěry o databázi eXist

Poslední využitou nativní XML databází byla databáze eXist. Tento systém je jistě pro uživatele na první pohled z hlediska pracovního prostředí nejpříjemnější. Dokumenty uložené v kolekcích jsou dobře viditelné, editovatelné, a způsob jejich načítání či odstraňování se ničím neliší od způsobu v běžných programových vybaveních. Nedostatečná nápověda systému je nahrazena obrovským zázemím na webových stránkách, kde se dá zjistit téměř cokoli.

I pro uživatele pracujícího s geodaty by mohl být tento systém velice atraktivní, a to nejen díky příjemnému pracovnímu prostředí. Jsou zde opět podporovány dva dotazovací jazyky, XQuery a XPath, přičemž XQuery má v rámci eXistu další podstatná rozšíření umožňující například aktualizace či transformace dokumentů. Jazyk XSLT je však tímto způsobem zcela nahrazen a tedy bohužel dále již není nijak podporován.

5.5.4 Výsledky dotazování

Základním měřítkem kvality nativního XML databázového systému pro ukládání geodat je schopnost správně, a pokud možno v co nejkratším čase, vyhodnocovat atributové a obzvláště pak prostorové dotazy.

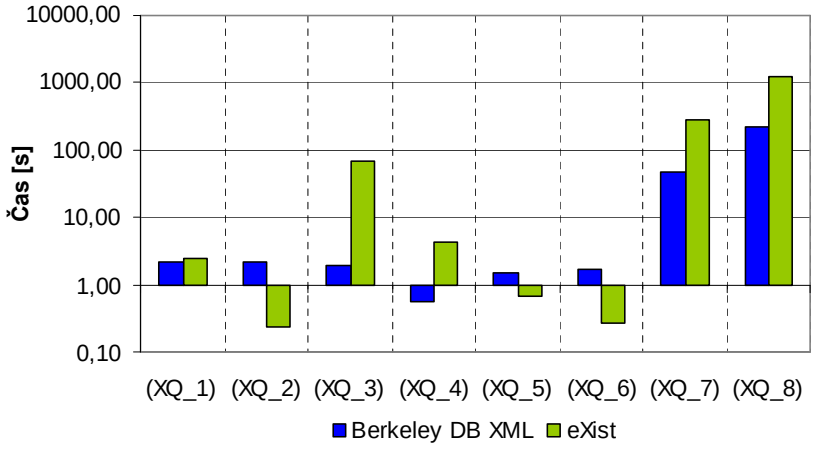
Při vytváření dotazů jsem získala přibližnou představu o tom, jak dlouho by mohl průběh každého dotazu trvat. Nemohla jsem tedy očekávat, že výsledky dotazů (XQ_11) a (XQ_12) budou známy v několika málo sekundách, když v nedatabázovém programu vyhodnocení těchto dotazů bylo měřeno na minuty či desítky minut. Přesto však zastávám názor, že vyhodnocení dotazu, které trvá více než 10 sekund, není optimální. Bohužel ale, jak je patrné z tabulky 5.2 uvádějící všechny časy XQuery dotazů odzkoušených v databázích Berkeley DB XML a eXist (4Suite dotazovací jazyk XQuery nepodporuje!), dotazů překračujících 10vteřinou hranici není málo.

	Berkeley DB XML	eXist
(XQ_1) atrib_dotaz_CR_radka.xquery	2,169	2,454
(XQ_2) atrib_dotaz_UK_radka.xquery	2,130	0,240
(XQ_3) atrib_dotaz_UK.xquery	1,898	68,538
(XQ_4) atrib_dotaz_hustota_200_300.xquery	0,552	4,286
(XQ_5) atrib_dotaz_kontinenty.xquery	1,473	0,661
(XQ_6) atrib_dotaz_hustoty.xquery	1,716	0,271
(XQ_7) atrib_dotaz_hustota_Evropa.xquery	46,160	289,816
(XQ_8) atrib_dotaz_prum_hustota_svet.xquery	226,688	1 257,899
(XQ_9) prostor_dotaz_souradnice.xquery	2,415	9,543
(XQ_10) prostor_dotaz_multipolygony.xquery	1,054	0,531
(XQ_11) prostor_dotaz_sousedi.xquery	"out of memory"	2 028,918
(XQ_12) prostor_dotaz_okruh_1000km.xquery	"out of memory"	24 762,647

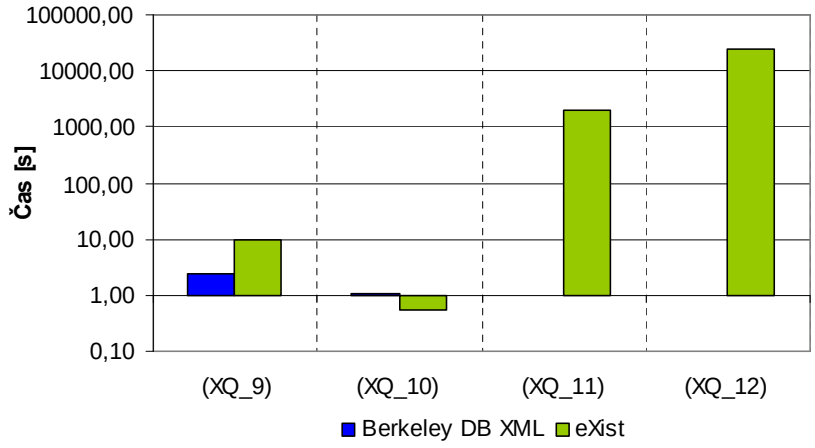
Tabulka 5.2: Výsledné časy vyhodnocení XQuery dotazů v sekundách

Protože každý dotaz má jinou strukturu a obě databáze pracovaly při jejich zpracovávání jiným způsobem, není možné najisto říci, která z databází si poradila s vyhodnocením XQuery dotazů lépe. Databáze Berkeley DB XML zpracovávala v konečném výsledku všechny dotazy v rychlejším čase (tabulka 5.4) a rovnoměrněji (obrázky 5.15, 5.16), ale nebyla schopna pro nedostatek paměti vyhodnotit složité prostorové dotazy. Naproti tomu databáze eXist ve svých výsledcích velice kolísala (obrázky 5.15, 5.16), ale dokázala vyhodnotit dotazy všechny – otázkou však zůstává, zda výsledný čas dotazu (XQ_12) 6 h 52 min 42,647 s (= 24 762,647 s) je v praxi akceptovatelný.

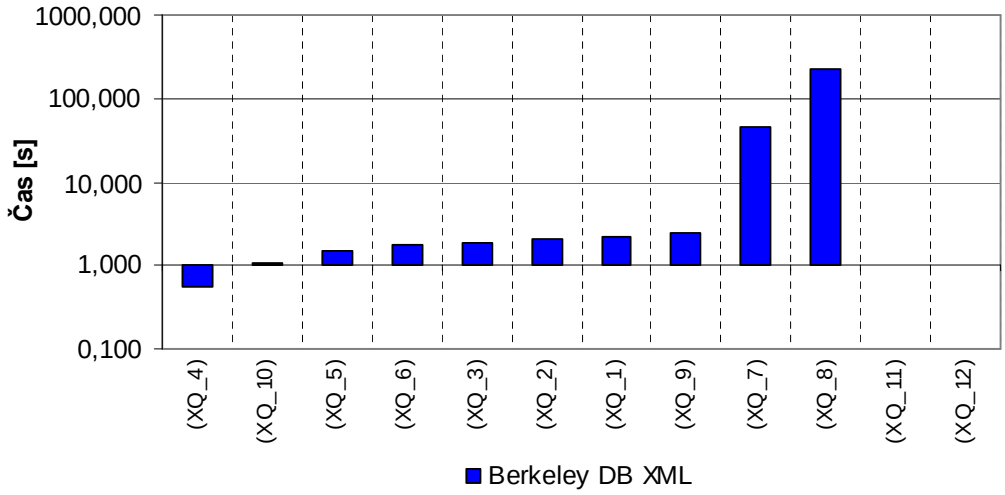
Pozn. Pro grafické znázornění časů vyhodnocení dotazů jsem z důvodu velkých rozdílů jednotlivých hodnot použila pro větší přehlednost na ose *y* logaritmické měřítko.



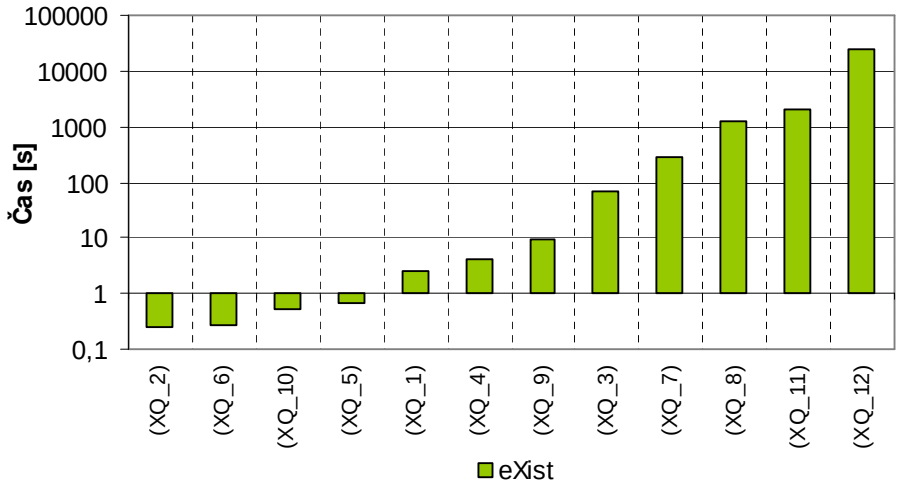
Obrázek 5.16: Grafické znázornění času vyhodnocení atributových XQuery dotazů jednotlivých databází



Obrázek 5.17: Grafické znázornění času vyhodnocení prostorových XQuery dotazů jednotlivých databází



Obrázek 5.18: Grafické znázornění času vyhodnocení XQuery dotazů dabáze Berkeley DB XML – seřazeno vzestupně

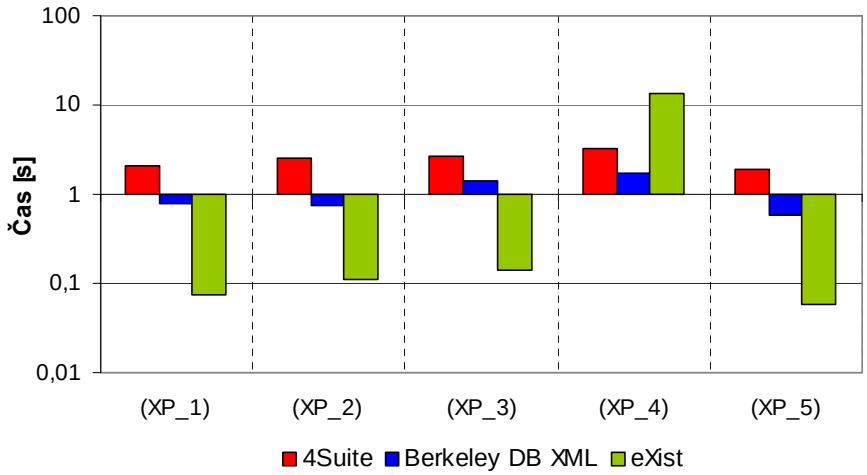


Obrázek 5.19: Grafické znázornění času vyhodnocení XQuery dotazů databáze eXist – seřazeno vzestupně

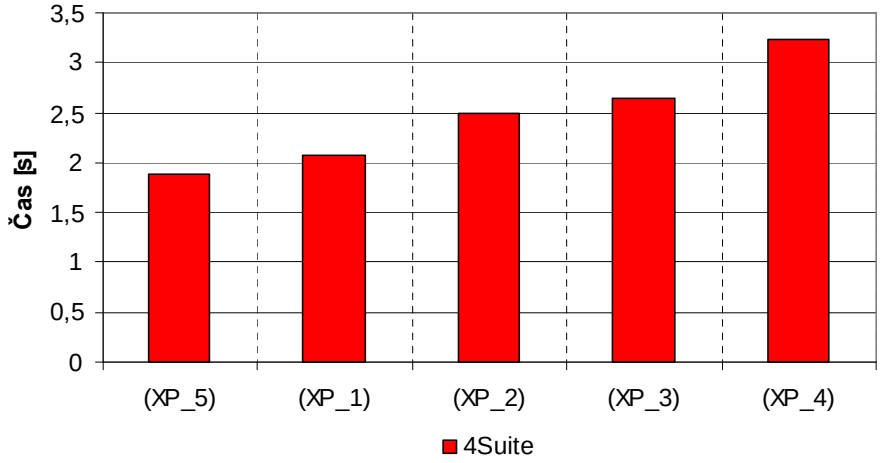
Vytvořené XPath dotazy byly vyzkoušeny ve všech použitých databázových systémech. Protože struktura těchto dotazů si je mnohem bližší, než tomu bylo u dotazů XQuery, bylo jejich vyhodnocení v rámci jednotlivých databází velice podobné – z hlediska času však kvůli různým přístupům jednotlivých databází opět odlišné. Například u databází 4Suite a eXist bylo po srovnání časových hodnot podle velikosti dosaženo totožných výsledků (obrázky 5.20, 5.22), tj. pořadí dotazů se shoduje. U těchto dotazů již nedocházelo k výkyvům, pouze databáze eXist vyhodnotila dotaz (XP_4) nad 10vteřinovou hranici, což byl celkově nejhorší dotaz celého testování XPath dotazů (13,389 s), který tomuto databázovému systému velmi ovlivnil celkový čas potřebný k vyhodnocení všech XPath dotazů (tabulka 5.4).

	4Suite	Berkeley DB XML	eXist
(XP_1) XPath_atrib_dotaz_CR_magda.txt	2,078	0,793	0,074
(XP_2) XPath_atrib_dotaz_UK_magda.txt	2,484	0,731	0,108
(XP_3) XPath_atrib_dotaz_prum_hustota_sveta_magda.txt	2,648	1,407	0,138
(XP_4) XPath_prostor_dotaz_souradnice_magda.txt	3,242	1,736	13,389
(XP_5) XPath_prostor_dotaz_multipolygony_magda.txt	1,886	0,593	0,058

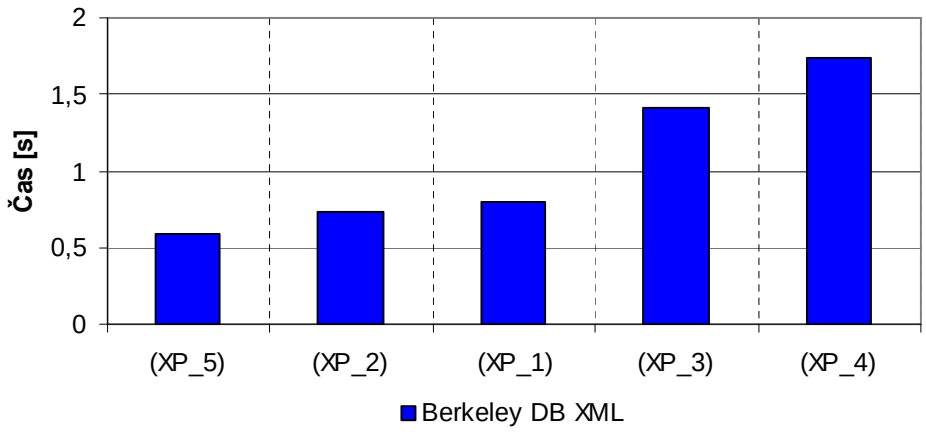
Tabulka 5.3: Výsledné časy vyhodnocení XPath dotazů v sekundách



Obrázek 5.20: Grafické znázornění času vyhodnocení XPath dotazů v jednotlivých databázových systémech

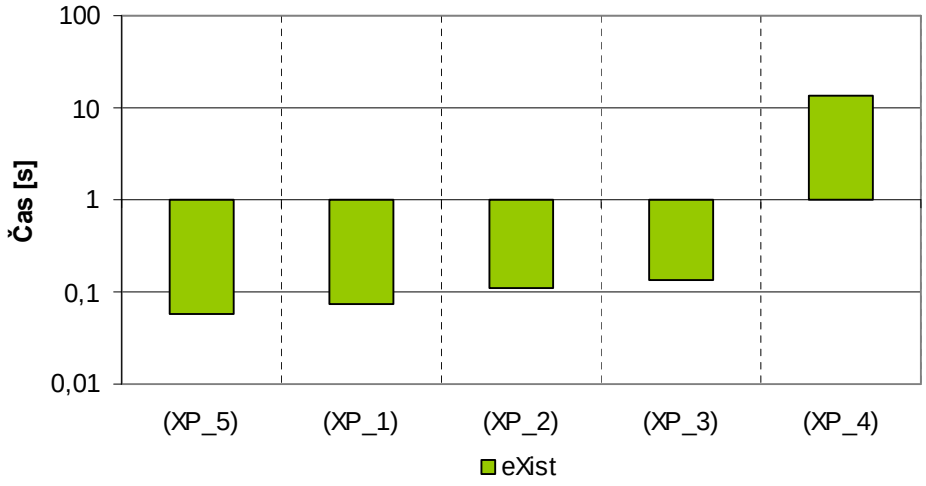


Obrázek 5.21: Grafické znázornění času vyhodnocení XPath dotazů databáze 4Suite – seřazeno vzestupně



Obrázek 5.22: Grafické znázornění času XPath dotazů databáze Berkeley DB XML – seřazeno vzestupně

Pozn. Pro grafické znázornění časů vyhodnocení dotazů na obrázcích 5.20 a 5.21 nemělo logaritmické měřítko přílišnou vypovídací hodnotu, proto jsem pouze v těchto dvou případech zvolila měřítko klasické.



Obrázek 5.23: Grafické znázornění času XPath dotazů databáze eXist – seřazeno vzestupně

Po dokončení testování XQuery a XPath dotazů jsem dospěla k závěru, že ze všech použitých databází pracovala nejrychleji a nejstabilněji databáze Berkeley DB XML. Pro ukládání geodat by však tato databáze zřejmě musela pracovat na výkonnějším počítači, aby dokázala vyhodnotit i složitější dotazy. Databáze eXist sice byla schopná dokončit všechny dotazy, ale celkový potřebný čas k jejich vyhodnocení je pro praktické využití (alespoň na počítači s podobnou konfigurací) příliš vysoký, což je patrné z tabulky 5.4.

Databáze 4Suite dokázala vyhodnotit XPath dotazy velice spolehlivě a v podobných časech, její využití bych ale pro ukládání geodat pro nepodporu XQuery nevolila.

	4Suite	Berkeley DB XML	eXist
atributové XQuery dotazy	x	4 min 42,786 s	27 min 4,165 s
prostorové XQuery dotazy	x	"nelze vyhodnotit"	7 h 26 min 41,639 s
Xpath dotazy	12,338 s	5,259 s	13,768 s

Tabulka 5.4: Sumarizace časů jednotlivých databází po vyhodnocení všech dotazů

ZÁVĚR

Vzhledem k výsledkům, ke kterým jsem dospěla, považuji využití nativních XML databází pro ukládání geodat na počítači s podobnou konfigurací (1,8 GHz, 768 RAM) jako ne příliš vhodné. Pevně věřím, že na výkonnějších počítačích by například databáze Berkeley DB XML byla schopná vyhodnotit všechny dotazy a databáze eXist by složitější dotazy řešila poněkud rychleji.

Negativní výsledky však neovlivnila pouze konfigurace počítače, hlavní příčinou byla podle mého názoru špatná vnitřní struktura zdrojových dokumentů, kde obrovský počet hraničních souřadnic jednotlivých států byl uveden v jednom elementu s nadbytečnými mezerami (pro jednu dvojici souřadnic cca 25 mezer), proto k problémům docházelo především při prostorových dotazech, kde bylo nutné jednotlivé souřadnice porovnávat.

Vlastní velikost souboru samozřejmě také hrála významnou roli. S velkými velikostmi souborů se však v oblasti geověd musí počítat. Pro zmenšení velikostí by bylo možné využít například binárního XML. Další možností by mohlo být nahrazení názvů elementů kratšími názvy, čímž by se zmenšil počet znaků v souboru, a tím by klesla i jeho velikost.

Po optimalizaci struktury i velikosti jednotlivých souborů věřím, že nativní XML databáze by na výkonnějších počítačích bylo možné v praxi využít. Jejich vlastnímu využití by však musel předcházet složitý výběr z existujících databází, protože pro geodata, jejich správu a zpracování, by měla databáze podporovat co možná nejvíce XML technologií, a to především dotazovací jazyky (zejména XQuery), transformační jazyky a jazyky pro aktualizaci.

Seznam použité literatury a zdrojů

1. Použitá literatura

[1] XML kompletní průvodce / Neil Bradley ; [přeložil Jiří Bráza]. -- 1. vyd. -- Praha : Grada, 2000. -- 537 s. :. -- Kniha byla napsána a zformátována v programu Microsoft Word 7 pro Windows 95. -- Slovníček. -- ISBN 80-7169-949-7 (brož.)

[2] XML Bible / Elliotte Rusty Harold. -- Foster City : IDG Books Worldwide, c1999. -- xxxiii, 1015 s. : il. ISBN 0-7645-3236-7 (brož.)

[3] XML v kostce : pohotová referenční příručka / Elliotte Rusty Harold, W. Scott Means ; [překlad Martin Blažík]. -- Vyd. 1. -- Praha : Computer Press, 2002. -- xvi, 439 s. ;. -- ISBN 80-7226-712-4 (brož.)

[4] XML data management : native XML and XML-enabled database systems / Akmal B. Chaudhri, Awais Rashid, Roberto Zicari, editors. -- Boston : Addison-Wesley, c2003. -- xxxvi, 641 s. : il. ISBN 0-201-84452-4 (brož.)

[5] Technologie XML / Irena Mlýnková ... [et al.]. -- 1. vyd.. -- Praha : Karolinum, 2006. -- 186 s. :. -- (Učební texty Univerzity Karlovy v Praze). -- 500 výt.. -- ISBN 80-246-1272-0 (brož.) :

[6] Přednášky z předmětu Databázové systémy 1 (KIV/DB1).

[7] Přednášky z předmětu Počítačová kartografie (KMA/POK).

[8] Přednášky z předmětu Úvod do GIS (KMA/UGI).

2. Internetové odkazy

[9] Bourret, Ronald. *XML and Databases* [online]. c2006 [cit. 2006-08-15]. <http://www.rpbourret.com/index.htm>

[10] Braeutigam, Falko; Mueller, Gerd; Nyfelt, Per; Mekenkamp, Leo. *Ozone Users Guide*. c2002,2003. <http://sourceforge.net/project/showfiles.php?group_id=39695&package_id=143551&release_id=303748 ... user-guide.html>

[11] Bříza, Petr. *Kompletní průvodce XSLT – Úvod do problematiky* [online]. c2004. [cit. 2007-01-13]. <http://interval.cz/clanky/kompletni-pruvodce-xslt-uvod-do-problematiky/>

[12] Crews, Nathan. *LandXML.org 2006.* <http://www.landxml.org/Workshops/Workshops.htm ... LandXML.org_2006.ppt >

[13] CRS4. *The compact GML for mobile devices* [online]. c2003-2005 [cit. 2007-01-23]. <http://www.crs4.it/nda/cgml/index.html>

[14] Čerba, Otakar. *Kapitola 1. XML formáty pro práci s geografickými daty* [online]. c2005 [cit. 2007-01-23]. <www.gis.zcu.cz/studium/pok/Materialy/xml_geodata.html>

[15] Český úřad zeměměřický a katastrální. *Základní báze geografických dat ZABAGED®*.[cit. 2007-03-15]. <http://www.cuzk.cz/Dokument.aspx?PRARESKOD=998&MENUID=0&AKCE=DOC:30-ZU_ZABAGED>

[16] Dorninger, Peter. *XML Technologies and Geodata.* <www.ipf.tuwien.ac.at/MarsExpress/docs/corp03/article0302.pdf >

[17] GBorg. *The xpsql Project* [online]. C2000-2006 [cit. 2006-10-11]. < http://gborg.postgresql.org/project/xpsql/projdisplay.php>

[18] Institute for System Programming RAS. *SEDNA – Native XML Database System* [online]. c2003-2006 [cit. 2006-10-11]. < http://modis.ispras.ru/sedna/>

[19] ITER. *ITER*[online].c2000[cit. 2006-09-05]. <http://dbdom.sourceforge.net/>

[20] Japan Information Processing Development Corporation/ Database Promotion Center, Japan. *G-XML Milestones* [online]. c1999-2006 [cit. 2007-01-23]. <http://www.dpc.jipdec.jp/gxml/contents-e/history.htm>

[21] Japan Information Processing Development Corporation/ Database Promotion Center, Japan. *G-XML PROJECT HOMEPAGE* [online]. c1999-2006 [cit. 2007-01-23]. <http://www.dpc.jipdec.jp/gxml/contents-e/>

[22] Jupitermedia Corporation. *Bowerbird Computing Release XDBM v1.0* [online]. c2006 [cit. 2006-10-11]. < http://www.internetnews.com/dev-news/article.php/278811>

[23] Kosek, Jiří. *Seriál o XML pro Sortwarové noviny* [online]. c2000-2001 [cit. 2007-01-13]. <http://www.kosek.cz/clanky/swn-xml/index.html>

[24] Kosek, Jiří. *XSLT v příkladech* [online]. c2000-2001 [cit. 2007-01-13]. <http://www.kosek.cz/xml/xslt/index.html>

[25] Murthi, Vijay. *Native XML Databases – A Review.* <http:// www-d0.fnal.gov/computing/grid/xml_database.pdf >

[26] myXMLDB. *myXMLDB*[online]. c2005 [cit. 2006-10-11]. <http://myxmldb.sourceforge.net>

[27] Nguyen Viet Cuong. *XML Native Database Systems - Review of Sedna, Ozone, NeoCoreXMS.* <http://swing.felk.cvut.cz/index.php?option=com_docman&task=doc_view&gid=5&Itemid=62>

[28] Особенности СУБД Sedna. *XML-СУБД Sedna: технические особенности и варианты использования* [online]. c1992-2006 [cit. 2006-10-11]. < http://www.osp.ru/text/302/185085/>

[29] Ogbuji, Uche; Ogbuji, Chimezie. *Develop Python/XML with 4Suite, Part 5: The repository features*. c2002. <https://www6.software.ibm.com/developerworks/education/x-4suite5/x-4suite5-a4.pdf >

[30] OpenDEN. *Ozone* [online]. c2006 [2006-10-11]. <http://www.openden.com/viewprod.php?pid=66206319950&mcatid=466517362138&scatid=73478219916&set=>

[31] Oracle. *Sleepycat Products: Berkeley DB XML*[online]. c2006 [cit. 2006-08-15]. <http://sleepycat2.inetu.net/products/bdbxml.html >

[32] O'Reilly Media, Inc. *Introduction to dbXML* [online]. c2006 [cit. 2006-10-11]. <http://www.xml.com/pub/a/2001/11/28/dbxml.html>

[33] SGML Users' Group. *A Brief History of the Development of SGML* [online]. c1990. [cit. 2007-03-15]. <http://www.sgmlsource.com/history/sgmlhist.htm>

[34] Sleepycat Software. *Introduction to Berkely DB XML*. <http://www.oracle.com/technology/documentation/berkeley-db/xml/intro_xml/BerkeleyDBXML-Intro.pdf>

[35] Stylus Studio® and DataDirect XQuery™. *Berkeley DB XML* [online]. c2004-2007 [cit. 2006-08-15]. <http://www.stylusstudio.com/dbxml.html>

[36] Stylus Studio® and DataDirect XQuery™. *SQL/XML Tutorial: SQL/XML, XQuery and Native XML Programming Languages* [online]. c2004-2007 [cit. 2007-01-15]. <http://www.stylusstudio.com/sqlxml_tutorial.html>

[37] The Apache Software Foundation. *4SUITE.org* [online].c2000 [cit. 2006-09-05]. <http://4suite.org/index.xhtml >

[38] The Apache Software Foundation. *Apache Xindice* [online]. c2001-2003 [cit. 2006-10-11]. < http://xml.apache.org/xindice/index.html>

[39] The dbXML Group, L.L.C.*dbXML (Native XML Database)*[online].c2000-2003[cit. 2006-09-05]. <http://www.dbxml.com/product.html>

[40] The XML:DB Initiative. *XUpdate – Working Draft* [online]. c2000-2003 [cit. 2007-01-13]. <http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html>

[41] Yang, Yu. *Benchmarking of Native XML Database Systems.* <http://www.library.uow.edu.au/adt-NWU/uploads/approved/adt-NWU20051004.101038/public/01Front.pdf >

[42] *dbXML 2.0 - An Introduction* [online]. [cit. 2006-09-05] <http://www.xml-training-guide.com/dbxml.html>

[43] *eXist – Open Source Native XML Databáze*[online].[cit. 2006-09-05]. <http://exist.sourceforge.net/>

[44] *ISO/TC 211/WG 4/PT 19136: Geographic information – Geography Markup Language (GML)* <http://portal.opengeospatial.org/modules/admin/license_agreement.php?suppressHeaders=0&access_license_id=3&target=http://portal.opengeospatial.org/files/index.php?artifact_id=4700 ... GML-3.1.pdf>

[45] *Ozone* [online].[cit. 2006-10-11]. <http://www.ozone-db.org/noframes/ozonies/ozonexml.html>

[46] *Timber* [online]. Poslední aktualizace: 2006-05-09 [cit. 2006-10-11]. <http://www.eecs.umich.edu/db/timber/>

[47] *WIKIPEDIA* [online]. [cit. 2007-05-11]. <http://www.wikipedia.org/>

Příloha A

Konfigurační soubor „4ss.conf“

```
<?xml version='1.0' encoding='UTF-8'?>
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns='http://xmlns.4suite.org/4ss/properties#'
                                xml:base=''
>
  <Core rdf:ID='Core'>

    <SystemContainer>ftss</SystemContainer>

    <!-- Use filesystem for repository -->
    <Driver rdf:parseType='Resource'>
      <rdf:type
resource='http://xmlns.4suite.org/4ss/properties#FlatFile'/>
      <Root>xmlserver</Root>
    </Driver>

    <!-- Use PostGreSQL for repository
    <Driver rdf:parseType='Resource'>
      <rdf:type
resource='http://xmlns.4suite.org/4ss/properties#Postgres'/>
      <DbName>xmlserver</DbName>
      <User>dbusername</User> -->
      <!--
        User, Password, Host, Port are optional.
        SECURITY NOTE: All info is stored here in plain text!
        Omit whatever optional fields you would rather be prompted for.
      -->
      <!--
      <Password>dbpwd</Password>
      <Host>dbhost</Host>
      <Port>dbport</Port>
    </Driver> -->

    <!-- Controller PID and log file locations -->
    <PidFile>c:/Python23/Scripts/tmp/4ss.pid</PidFile>
    <LogFile>c:/Python23/Scripts/tmp/4ss.log</LogFile>

    <!-- Controller log level (optional; default: notice) -->
    <!-- one of emerg|crit|error|warning|notice|info|debug -->
    <LogLevel>debug</LogLevel>

  </Core>
</rdf:RDF>
```

Příloha B

Struktura přiloženého CD

- **Geodata** ... šest zdrojových souborů s geodaty ve formátu JML
- **NXD_instalace**
  - **4Suite** ... instalační programy NXD 4Suite a programovacího jazyka Python 2.3
  - **Berkeley_DB_XML** ... instalační program NXD Berkeley DB XML
  - **eXist** ... instalační program NXD eXist
- **Text_DP** ... vlastní text diplomové práce ve formátu PDF
- **XPath_dotazy**
  - **atributove_dotazy** ... tři atributové XPath dotazy ("*.txt")
    - **vystupy_dotazu** ... výsledky atributových XPath dotazů ("*.txt")
  - **prostorove_dotazy** ... dva prostorové XPath dotazy ("*.txt")
    - **vystupy_dotazu** ... výsledky prostorových XPath dotazů ("*.txt")
- **XQuery_dotazy**
  - **atributove_dotazy** ... osm atributových XQuery dotazů ("*.xquery")
    - **vystupy_dotazu** ... výsledky atributových XQuery dotazů ("*.txt")
  - **prostorove_dotazy** ... čtyři prostorové XQuery dotazy ("*.xquery")
    - **vystupy_dotazu** ... výsledky prostorových XQuery dotazů ("*.txt")