VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

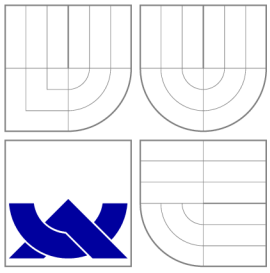
NATIVNÍ XML DATABÁZE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

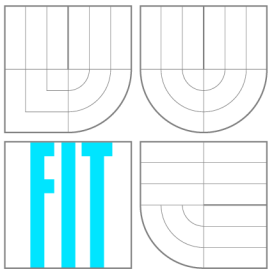
AUTOR PRÁCE
AUTHOR

KAREL PIWKO

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NATIVNÍ XML DATABÁZE

NATIVE XML DATABASES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

KAREL PIWKO

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. LUKÁŠ STRYKA

BRNO 2008

Abstrakt

XML je de facto standardem pro výměnu informací na Internetu, mezi aplikacemi a podniky. Při jeho ukládání máme na výběr techniky umožnující použití relačních databází a nebo nativní XML databáze, jejichž XML datový model lépe odpovídá datům. Tato práce se zabývá tvorbou a porovnáním webových aplikací založených na uvedených technologiích, s využítím open source databáze eXist, webového frameworku Apache Cocoon a open source databáze PostgreSQL, aplikačního frameworku Stripes, Java Persistence API a EJB. Vzorovou aplikací je portál Wikipedie, v obou případě běžící na aplikačním serveru JBoss.

Klíčová slova

XML, nativní XML databáze, eXist, převod XML dokumentů na tabulky relační databáze, porovnání XML nativních a relačních databází, Apache Cocoon, Java Persistence API, Stripes, parser MediaWiki formátu na abstraktní syntaktický strom

Abstract

XML is de facto standard for enchanging information on the Internet, among applications and business companies. While storing XML, we can choose between techniques allowing us to use relation database and native XML databases, whose underlaying XML data model is more appropriate. This study is about creating and comparision of web applications based on mentioned technologies, using open source database eXist, web framework Apache Cocoon and open source database PostgreSQL, application framework Stripes, Java Persistence API and EJB. As a sample, portal of Wikipedia was chosen, in both cases powered by JBoss application server.

Keywords

XML, native XML databases, eXist, converting XML documents to relational tables, comparision of native XML and relational databases, Apache Cocoon, Java Persistence API, Stripes, MediaWiki to abstract syntax tree parser

Citace

Karel Piwko: Nativní XML databáze, bakalářská práce, Brno, FIT VUT v Brně, 2008

Nativní XML databáze

Prohlášení

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

.....
Karel Piwko
12. května 2008

Obsah

Kapitola 1

Úvod

Jazyk XML je v dnešní době de facto standardem pro výměnu informací na Internetu mezi aplikacemi a podniky. Pro valnou většinu aplikací je buďto nativním formátem, nebo umožňují převod proprietárních formátů na XML. Rovněž komunikační protokoly XML-RPC [XML Remote Procedure Call], SOAP [původně Simple Object Access Protocol] a WS [Web Service], sloužící pro komunikaci aplikací na internetu, jsou založeny na XML.

Pro zobrazení XML dat koncovému uživateli lze v výhodou využít XSL [eXtensible Stylesheet Language] transformace, například do formátu PDF nebo XHTML [eXtensible HyperText Markup Language].

Tato práce volně navazuje na ročníkovou práci *Gestionnaire de Données Confidentielles* [Správce důvěrných dat], vypracovanou společně se studenty Islem Rekik a Charlie Nguyen-Ducem, pod vedením Tarika Al Aniho, v akademickém roce 2006/2007 na ESIEE Paris. V této práci jsme implementovali jednoduchou nativní XML databázi, architektu klient-server podporující SSL a uživatelské rozhraní za pomocí JSP a AJAXu. Byla mi inspirací pro volbu tématu bakalářské práce.

Práce nenavazuje na semestrální projekt.

1.1 Řazení kapitol

V kapitole 2 se budeme zabývat samotným jazykem XML a příčinou jeho rozšíření. Rozebereme možnosti použití XML dokumentů jako úložiště informací a potažmo jednoduché databáze. Poté se zaměříme na více komplexní řešení ukládání XML dokumentů do databází s podporou XML a nativních XML databází. Rozdíl mezi těmito technologiemi bude klíčovým tématem pro tuto práci. V souvislosti s ukládáním XML dokumentů si nastíníme jejich dělení podle typu a struktury uložených dat.

Následuje stručný popis ukládání XML dokumentů do relačních databází a popis jedné z nativních XML databází - eXist, která je vzorkem nativní XML databáze v této práci. Kapitolu ukončuje způsob indexace XML dokumentů pro potřeby vyhledávání.

V následující kapitole (3) popíšeme výhody XML databází při ukládání různých typů XML dokumentů. Zároveň se budeme zabývat problémy řešení postavených na relačních databázích, hlavně z pohledu integrace dat z více zdrojů. Ukážeme si, jak nativní rozšiřují transakční model ACID [Atomicity, Consistency, Isolation, Durability] v případě dlouho běžících transakcí a kapitolu uzavřeme případy užití databáze eXist v reálných systémech.

V čtvrtém kapitole (4) navrhneme případovou studii pro porovnání aplikací založených na nativních XML a relačních databázích. Dále popíšeme vhodný vzorek dat a jeho nahrání

do databází, architekturu jednotlivých aplikací a popis konfigurace použitých komponent.

V kapitole 5 se zabýváme čtením dat z databáze, rozdíly mezi způsobem vytváření pohledu na data a výkonem jednotlivých řešení.

Závěrem (6) zhodnotíme přínos celé práce, výhody jednotlivých technologií z hlediska uživatele i programátora a možnosti dalšího rozšíření.

V přílohách následuje velké množství použitých konfiguračních souborů, program pro uložení XML dokumentu do relační databáze, pomocné skripty a diagramy tříd. Zároveň je zde detailnější popis převodu XML Schema na schéma tabulek relační databáze.

1.2 Cíl práce

Tato práce porovnává dva různé přístupy k vyhledávání dat. Uvažuje data ve formátu XML dokumentu, které jsou za pomocí konverzních programů převedena do tabulek relační databáze a zároveň vložena do nativní XML databáze. K datům poté přistupují dvě webové aplikace, které implementují stejné chování.

Výsledkem je popis tvorby referenční aplikace využívající jazyk Java, aplikační framework Stripes, JTA [Java Transaction API], EJB [Enterprise Java Bean], JPA [Java Persistence API] (zde Hibernate) a relační databázi PostgreSQL a aplikace založené na nativní XML databázi eXist, rovněž využívající jazyk Java, webový framework Apache Cocoon a dotazovací jazyk XQuery. Mezi výslednými aplikacemi je poté porovnána složitost jednotlivých řešení, výkon, výhody a nevýhody.

Uživatel, disponující daty ve formátu XML, by je po přečtení této práce měl být schopen zařadit do jedné z kategorií XML dokumentů, určit cíl své aplikace a vhodnost použití jednoho z řešení. Podle návodu v kapitolách 4 a 5, znalosti jazyků XML, Java, XQuery, SQL a XSLT by měl být schopen sestavit webovou aplikaci používající uvedené technologie šitou na míru jeho datům, kterou poté může s využitím aplikačního serveru[1] zpřístupnit na intranetu nebo Internetu.

[1] Tato práce uvažuje aplikační server JBoss, nicméně konfigurace pro jiný aplikační server je podobná. Při omezení na vestavěné JTA lze použít i webový kontejner Tomcat, Jetty nebo jiný. Konfigurace vestavěného JTA není předmětem této práce.

Kapitola 2

XML a databáze

V této kapitole se budeme zabývat jazykem XML [eXtensible Markup Language][6] a jeho přínosem pro výměnu informací. Rozebereme možnost použití XML jako jednoduché databáze, tuto úvahu použijeme na vysvětlení rozdílu mezi databázemi s podporou XML a nativními XML databázemi, možnosti použití relační databáze jako podklad nativní a různé způsoby mapovaní XML dokumentů. V poslední části kapitoly se blíže zaměříme na nativní XML databázi eXist[1] a metody indexování XML dokumentů pro potřeby vyhledávání.

2.1 Jazyk XML

Jazyk byl navržen v roce 1996, jako aplikace principů a formální zjednodušení jazyka SGML [Standard Generalized Markup Language][2]. V současné době je jazyk XML de facto standardem pro výměnu informací a jejich reprezentaci. Shrneme body, které vedly k jeho rozšíření:

Flexibilita: Pomocí jazyka XML lze popsat stromové hierarchie, tabulky (ve smyslu relačních databází), grafy a dokonce rekurzivní struktury. Je dostupný pro prakticky všechny uživatele, protože je schopen pracovat v libovolném kódování znaků a jazyce, navíc je textovým formátem, tedy přímo čitelný bez použití speciálních nástrojů.

Popis a validace: Dokumenty v jazyce XML obecně splňují několik předpokladů, které usnadňují jejich strojové zpracování. Především jsou *well-formed*, což znamená, že splňují předpoklady uvedené v [6]. Dále může být jejich obsah upřesněn metadaty, pomocí různých XML schémat - reprezentovaných například jazyky DTD[6] a nebo XML Schema[5]. Výhodou druhého je, že k popisu používá samotný jazyk XML, výměna dat a jejich významu tedy může probíhat s použitím stejných transportních cest. Navíc podporuje i jmenné prostory a datové typy.

Rychlost vývoje: XML je sám o sobě metajazykem pro popis dat. K dispozici je mnoho nástrojů schopných s ním pracovat. Použitím XML na našich datech docházíme k rychlejší implementaci.

Zacílení: XML byl od začátku navržen pro výměnu informací na internetu[6]. Obsah je oddělen od prezentační logiky. XML je často meziproduktem, který je dále transformovatelný na jiné dokumenty, jako například XHTML nebo PDF.

2.2 XML jako databáze

XML je přirozeným formátem pro ukládání dat. Měli bychom tedy používat samotné XML jako databázi[18][35]?

Porovnejme XML s běžně dostupnými databázemi, například relačními. Pro hovoří následující argumenty:

- XML dokumenty stejně jako databáze obsahují data,
- Hierarchický model XML lze použít pro popis mnoha reálných dat, i když se liší od relačního,
- Pro XML existují jazyky popisující schémata - DDL [Data Definition Language] - například XML Schema nebo RELAX NG[4],
- Dále jsou k dispozici dotazovací jazyky [query language] - například XPath[3][8] a XQuery[9],
- K XML dokumentům lze přistupovat pomocí několik různých API - SAX, DOM, JDOM,
- a další.

Naopak, existuje mnoho důvodů, proč je použití XML jako databáze nevhodné, které převažují:

- XML je neefektivní při využívání diskové kapacity (textový formát), a pomalé při získávání dat - nutnost parsování,
- Neobsahuje věci typické pro databáze - například indexování a víceuživatelský přístup,
- Negarantuje vlastnosti modelu ACID,
- a další.

Ve výsledku lze samotné XML použít jako databázi v jednouživatelském prostředí pro malé objemy dat, například pro konfigurační soubory.

2.3 Typy XML z hlediska dat

Dokumenty XML se podle charakteristiky dat v nich uložených dají rozdělit na dokumentové XML dokumenty [*document-centric*] a datové XML dokumenty [*data-centric*][18].
U druhé varianty navíc můžeme rozlišit podkategorii semistrukturovaných [*semi-structured*].
Typ dat rozhoduje o vhodnosti použití databáze.

2.3.1 Dokumentový XML dokument

Tyto XML dokumenty se vyznačují málo pravidelnou nebo nepravidelnou strukturou dat, většími základními jednotkami dat (až na úroveň dokumentu samotného a složeným obsahem elementů. Pořadí sourozeneckých elementů je téměř vždy důležité. Často jsou určeny pro přímé čtení uživateli a nepochází z databáze, naopak jsou psány ručně.

Těmito dokumenty bývají manuály, knihy, zápisky do blogů (uvažujeme-li formát XHTML) a podobně.

Zdrojový kód 2.1: Příklad dokumentového XML dokumentu

```
<bakalarska-prace>
    <uvod>
    Tato bakalářská práce popisuje <b>nativní XML databáze</b>, jejich a
    aplikaci a srovnává použité technologie s řešením postaveném na
    <b>ORM a JPA</b>
    </uvod>
    <odstavec>
    V následujícím odstavci si popíšeme základní typy XML dokumentů z
    hlediska typu dat v nich uložených. Jak můžeme vidět na příkladu,
    <i>document-centric</i> dokumenty obsahují mnoho textu.
    </odstavec>
    <odstavec>
    Závěrem přidáme, že:
    </odstavec>
    <list>
        <prvek>XML je upovídaný jazyk a</prvek>
        <prvek>pro přenos dat se proto často komprimuje.</prvek>
    </list>
</bakalarska-prace>
```

2.3.2 Datový XML dokument

Tyto XML dokumenty se naopak vyznačují velmi pravidelnou strukturou a malými základními jednotkami dat (na úroveň *PCDATA*-elementů nebo atributů). Pořadí sourozeneckých elementů je většinou zanedbatelné, mimo samotné validace dokumentu. Data často pocházejí přímo z databáze, kde je XML transportním formátem a nebo jsou výstupem jiného programu. Jsou určeny pro strojové zpracování.

Těmito dokumenty jsou například kurzy akcií, faktury, data získaná z vesmírných sond a podobně.

Zdrojový kód 2.2: Příklad datového XML dokumentu

```
<nasa-data>
    <sonda>
        <jmeno>Cassini</jmeno>
        <datum-vypusteni>
            <den>15</den>
            <mesic>říjen</mesic>
            <rok>1997</rok>
        </datum-vypusteni>
    </sonda>
    <mereni>
        <id>1234ABC</id>
        <vzdalenost>
            <hodnota>1000</hodnota>
            <jednotka>km</jednotka>
        </vzdalenost>
        <cil>Titan</cil>
        <data>
```

```
            <voda>70%</voda>
            <albedo>0.23</albedo>
            <teplota>93.7</teplota>
        </data>
    </mereni>
</nasa-data>
```

2.3.3 Semistrukturovaný datový XML dokument

Data, označovaná jako semistrukturovaná[19], mají sice pravidelnou strukturu, nicméně ta se často mění nebo není známa v době návrhu. Data popisují sama sebe, čímž se umožňuje zpracovat data neznámá v době návrhu. Pro shodná data existuje více možných reprezentací v jenom dokumentu, například jméno definované jedním elementem a nebo jako seskupení jména a příjmení. Často jsou data řídká, tedy pro elementy známé již v době návrhu nejsou vyplněny hodnotami.

Typickým příkladem jsou data z genetických výzkumů, lékařských nebo jiných vědeckých přístrojů. Pro tyto data je velmi složité navrhnout schéma pro relační databáze.

2.4 Techniky ukládání XML dat do databáze

Máme-li XML dokumenty, které potřebujeme uložit do databáze, máme v k dispozici tři možnosti:

- XML dokument ukládáme celý,
- XML je pouze transportním formátem. V tomto případě není potřeba ukládat samotný XML dokument, stačí uložit hodnoty v něm obsažené. Vztahy mezi elementy jsou pro nás nedůležité,
- a nebo dokument ukládáme za použití XML datového modelu.

2.4.1 XML dokument

Kandidátem jsou dokumentové XML dokumenty, které ukládáme například do hierarchické struktury v souborovém systému nebo jako objekt CLOB či BLOB do tabulek relační databáze. Tento způsob výrazně omezuje a zpomaluje složitější dotazy, protože dokument musí být před jeho zodpovězením pokaždé znova zpracován.

2.4.2 XML jako transportní formát

Kandidátem jsou datové XML dokumenty, jejichž data je možno uložit do relační databáze. Samotný proces spočívá v následujících krocích:

1. Vytvoříme schéma pro relační databázi odpovídající XML dokumentům,
2. Data rozložíme na fragmenty, které odpovídají řádkům v tabulkách - rozkládání [*shredding*] a uložíme,
3. Při dotazu na data převádíme dotaz na jazyk SQL a odpověď na dotaz vracíme ve formě XML - skládání [*publishing*].

Postup tvorby schématu je popsán v příloze A.1. Různé metody skládání XML jsou popsány v například v [34].

2.4.3 XML jako datový model

Zde jsou kandidátem semistrukturované datové XML dokumenty. Datový model XML je dalším modelem vedle hierarchického, relačního, objektově orientovaného a dalších. Je reprezentován uspořádaným stromem s typovanými a pojmenovanými vnitřními uzly a nepojmenovanými listy, obsahujícími samotné data[35]. Z ostatních používaných modelů je nejblíže hierarchickému modelu. Datový model XML musí obsahovat nejméně elementy včetně jejich pořadí v dokumentu, atributy a *PCDATA*. Příkladem jsou například XPath model, XQuery model a modely založené na DOM nebo SAX. Základní datovou jednotkou uloženou v databázi je XML fragment.

Datový model neklade omezení na fyzický model uložení dat. Může jím být relační, hierarchická nebo objektově orientovaná databáze, indexované soubory nebo jiný, proprietární formát[18].

2.5 Nativní XML databáze a databáze s podporou XML

V závislosti na technice uložení dat do databáze definujeme:

Nativní XML databáze: Používající XML datový model přímo a

Databáze s podporou XML: Používající jiný datový model a XML schéma k mapovaní mezi ním a instancí XML datového modelu.

Databáze s podporou XML často obsahují nástroje pro usnadnění práce s XML, například implementaci SQL/XML standardu[7]. Obecně se nehodí pro uložení všech možných XML dokumentů. Jejich hlavní výhodou je, že se nemusí měnit již existující aplikace, pouze se přidá a nakonfiguruje XML vrstva.

Typicky při použití XML dotazovacích jazyků dochází k překladu na jazyk SQL a ze získaných dat se vytvoří XML dokument. Následuje příklad v jazyce SQL/XML, který je rozšířením jazyka SQL:

Zdrojový kód 2.3: Ukázka jazyka SQL/XML

```
XMLELEMENT(NAME data,
XMLELEMENT(NAME voda, d.voda),
    XMLELEMENT(NAME albedo, d.albedo),
    XMLELEMENT(NAME teplota, d.teplota))
```

a jeho výsledek:

Zdrojový kód 2.4: Výsledek dotazu v jazyce SQL/XML

```
<data>
<voda>hodnota d.voda</voda>
<albedo>hodnota d.albedo</albedo>
<teplota>hodnota d.teplota</teplota>
</data>
```

Nativní XML databáze používají přímo XML datový model. Jejich hlavní výhodou pocítíme při ukládání semistrukturovaných datových XML dokumentů. Při použití nativní XML databáze chceme stejně jako u relační databáze normalizovat uložená data a zajistit referenční integritu.

2.5.1 Normalizace XML dat

Při zpracování dat za účelem uložení v relační databázi se používá několik normálních forem (1NF, 2NF, 3NF, BCNF [Boyce-Coddova normální forma], 4NF a 5NF)[23]. Tyto schémata relace zajišťují několik vlastností dat:

- Eliminace nepřesností ve vyjádření dat,
- Minimalizace redundance (opakovaného vyjádření informace na více místech),
- Znesnadnění udržování integrity dat.

1NF [První normální forma] říká, že data musí být vyjádřeny jako atomické typy[23] a zároveň musí mít každý záznam stejný počet polí[31]. Struktura XML dokumentu je naopak hierarchická, element může být komplexního typu, všechny elementy záznamu nemusí být vyjádřeny (viz 2.3.3) a navíc jsme schopni uložit více hodnot pro jedno pole. Poslední vlastnost má největší vliv na chápání normálních forem definovaných pro relační databáze v konceptu XML databází[31], zároveň ale činí modelovaní dat přirozenějším procesem.

K definici schémat lze použít více jazyků (kapitola 2.2), v případě použití XML Schema[5] získáme klíče (element `key`), které se liší od relačních pouze tím, že jejich unikátnost je zaručena na určitém měřítku. Navíc není možné klíč definovat na elementu samotném, ale je nutné ho definovat na jeho předku[32]. Unikátnost v globálním měřítku se zaručuje uvedením cesty k elementu, nápadně připomínající jazyk XPath. Tímto se navíc eliminuje běžná praxe z relačních databází – přiřazování identifikátoru ke každému elementu, častá pro použití v JPA. Používání cesty místo jednoduchého identifikátoru odpovídá hierarchickému modelu XML.

2NF a 3NF, které zajišťují příslušnost dat k primárnímu, resp. kandidátnímu klíči, jsou velmi dobře aplikovatelné na XML. XML Schema obsahuje element `keyref`, prakticky ekvivalent cizího klíče. Přesto je vhodné rozlišovat mezi použitím cizího klíče v relačních a XML databázích. Pokud v relační databázi použijeme vztah klíč - primární klíč ve vztahu *one-to-many* navíc s omezením `ON DELETE CASCADE` u cizího klíče, je vhodnější využít hierarchické struktury XML dat a místo asociace použít kompozici. Uvažujme, že pro jednu zprávu vesmírné sondy umožníme více měření, úsek XML Schema potom bude vypadat takto:

Zdrojový kód 2.5: XML Schema pro měření vesmírné sondy

```
<complexType name="nasa-sonda">
    <element name="sonda">
        ...
    </element>
    <element name="mereni" maxOccurs="unbounded">
        <complexType>
            <sequence>
                <element name="id" type="integer" />
                ...
            </sequence>
        </complexType>
    </element>
</complexType>
```

4NF, která definuje požadavky v rámci multizávislostí, není z hlediska XML zajímavá, protože XML porušuje atomicitu ukládaných dat (1NF). 5NF vynucuje zrušení významových závislostí mezi vícehodnotovými fakty, na XML je aplikovatelná.

Normální formy není nutno aplikovat pouze na dokumenty omezené pomocí XML Schema. V dokumentu [14] je definována XNF [XML Normal Form], která je zobecněním BCNF, definována za pomocí DTD. Na rozdíl od BCNF při použití XNF dekompozice nedochází k potenciální ztrátě funkčních závislostí a omezení [14].

2.5.2 Referenční integrita

Referenční integrita je zajištěna za pomocí ID/IDREF, XML Schema (`key` a `keyref`), XLink nebo jiného, proprietárního, mechanismu. U valné většiny XML databází její vynucení ovšem probíhá pouze v daném dokumentu, nikoliv v globálním měřítku a navíc pouze v době validace dokumentu[18], což vede na aplikační řešení integrity.

2.6 Nativní XML databáze eXist

eXist je nativní XML databáze s otevřeným zdrojovým kódem, aktivně vyvíjený projekt od roku 2001. Je kompletně napsaná v Javě. Dokumenty, k nimž není nutné dodávat schéma, jsou ukládány do hierarchických kolekcí, data jsou uloženy pomocí B+ stromu se stránkováním odpovídajícím fyzickým stránkám na disku [26] [1] pomocí indexů a persistentního DOM stromu.

eXist implementuje specifikaci XQuery[9] jako datový model XML. Současně obsahuje několik rozšíření, například indexované fulltextové vyhledávání pomocí operátorů a funkcí `|=`, `&=` nebo `near()`.

eXist lze použít jako samostatnou aplikaci, knihovnu nebo součást webové aplikace jako servlet. Jeho výhodou je množství API, které lze použít k přístupu: HTTP/REST, XML-RPC, SOAP, WebDAV a nebo programové XML:DB API. Do budoucna se uvažuje o implementaci XQJ [XQuery API for Java], specifikace JSR-225.

2.7 Metody indexovaní XML souborů

Masivní použití indexů výrazně zvyšuje rychlost vyhledávání a sestavování XML dat. eXist indexuje nejen elementy XML dokumentů a jejich vztahy jako persistentní DOM strom, ale navíc i samotný výskyt elementů a jejich atributů. Veškeré indexy jsou řazeny podle kolekcí, což klade výkonové omezení na příliš velké kolekce (2000 a více dokumentů)[26].

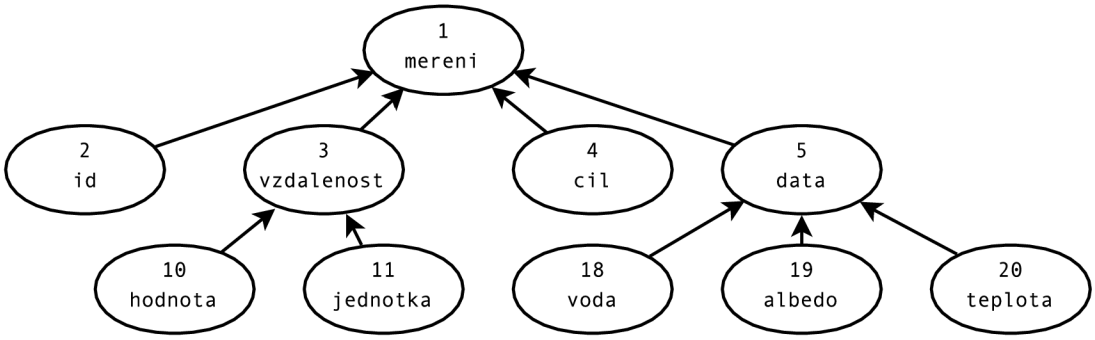
Při ukládání stromu DOM do XML databáze je výhodné umět z indexu samotného zjistit vztahy mezi uzly. Tímto zamezíme načítání uzlů, tedy elementů XML dokumentu z persistentního úložiště a výrazně urychlíme zodpovídání dotazu. V ideálním případě jsme schopni zjistit z unikátních identifikátoru dvou uzlů - UID, zda jsou ve vztahu předek, následník nebo sourozenci.

eXist v současné době používá metodu indexování XML souborů nazvanou dynamické číslování úrovní - DLN [Dynamic Level Numbering][20]. V této podkapitole jej popíšeme současně s původní metodou indexování a bude ilustrovat jeho výhody.

Existují i další metody indexování uzlů, například identifikace uzlu pomocí `document-id`, a pořadí v průchodu *preorder* a *postorder* nebo rekurzivní udílení UID. Podrobnosti lze nalézt v dokumentu [20].

2.7.1 Unikátní identifikátory uzlů průchodem do šířky

Při použití této metody je každému z uzlů přiřazen UID. Aby bylo možné zjistit vztahy mezi uzly, je DOM modelován jako kompletní n-nární strom. Má-li uzel v dané úrovni méně než n uzlů, jsou vloženy virtuální uzly.



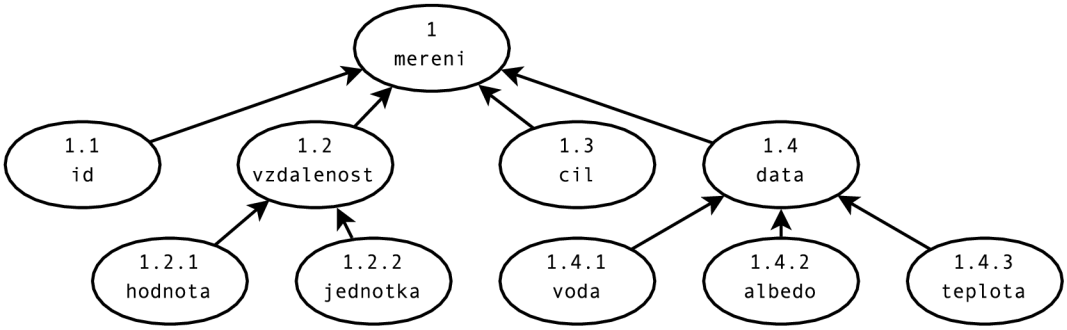
Obrázek 2.1: n-nární strom a UID získaná průchodem do šířky

Tento způsob číslování má několik výrazných nevýhod. Je-li strom nevyvážený, a například na dvacáté úrovní existuje 30 uzlů, jejich UID velmi rychle rostou a snadno se vyčerpá přidělený paměťový rozsah. Kromě omezení velikosti dokumentu je dalším problémem jeho změna. Vložení nového uzlu, jeho výmaz a další operace mohou způsobit přečíslováni uzlů minimálně části dokumentu, výrazně zpomalujíc výkon. Nutnost znát maximální počet uzlů v úrovni předem znemožňuje použití pro streamované XML dokumenty.

eXist používal modifikovaný algoritmus, který pro každou úroveň stromu definuje jeho n-nárnost, ukládajíc ji do pole pro každý indexovaný XML dokument. Tato modifikace výrazně zvyšuje maximální možnou velikost dokumentu[27].

2.7.2 Dynamické číslování úrovní uzlů

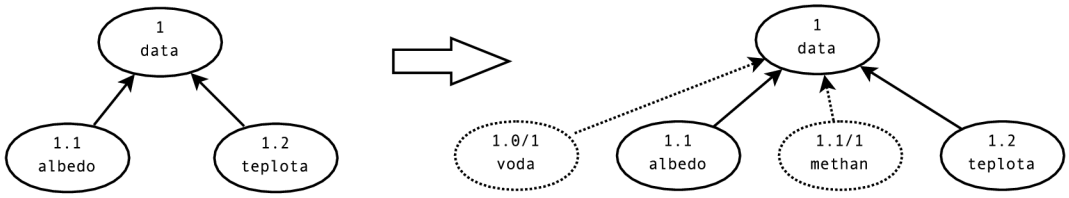
Pro překonání omezení číslování průchodem do šířky byl navržen algoritmus DLN[20]. DLN je hierarchickým číslováním založeným na Deweyově knihovnické soustavě, což je posloupnost čísel oddělených oddělovacím znakem (.). Takto je umožněno zpracovávat XML dokumenty libovolné délky, streamované XML dokumenty a nevyvážené stromy.



Obrázek 2.2: UID získaná dynamickým číslováním úrovní uzlů

Hlavním problémem Deweyovy knihovnické soustavy je způsob zakódování UID tak, aby byly UID binárně porovnatelné, paměťová náročnost uložení UID pro vysoké úrovně byly co nejmenší a modifikace dokumentu nezpůsobovala nutnost přečíslování části stromu[20].

Pro překonání posledního problému DLN využívá systém podhodnot, tedy alternativního oddělovacího znaku jehož hodnota je vyšší než hodnota standardního oddělovače, a zároveň neznamená přechod na další úroveň, tedy mezi elementy *1.1* a *1.2* vložíme element *1.1/1*. Uvedená notace umožňuje vkládat i vlevo před uzel, a to tak, že posledním znakem nikdy nesmí být *0*. Tedy před uzel *1.1* vložíme *1.0/1*, před něj eventuálně *1.0/0/1*. Jedinou nevýhodou je narůstající délka identifikátorů.



Obrázek 2.3: DLN a možnosti přidávání uzlů bez nutnosti je přečíslovat

eXist implementuje DLN pomocí variabilního bitového kódování, které je velmi výhodné pro streamované XML dokumenty. Každá úroveň indexu je zakódována pomocí násobků základní číselné jednotky (momentálně 4 bity), kde nejvyšší bity slouží jako příznaky počtu jednotek. Jako oddělovač úrovní (.) se používá binární *0*, naopak jako alternativní oddělovač (/) binární *1*[27]. Tímto se umožňuje binárně porovnávat číselné hodnoty.

Počet jednotek	Bitový vzor	Číselný rozsah
1	0xxx	0 až 7
2	10xx xxxx	8 až 71
3	110x xxxx xxxx	72 až 583
4	1110 xxxx xxxx xxxx	583 až 4679

Tabulka 2.1: Číselné rozsahy zakódování identifikátorů pomocí jednotek pevné velikosti

Pomocí DLN zakódujeme například uzel *1.33.6.1/3* jako *0001 0 1001 1010 0 0110 0 0001 1 0011*, celkem na 28 bitů. Při použití průchodu do šířky a kompletního n-nárního stromu bychom spotřebovali číselný rozsah $33^0 + 33^1 + 33^2 + 33^3 = 37060$, tedy bychom na použití potřebovali 32bitový datový typ `int`. eXist původně na ukládání používal 64bitový datový typ `long`.

2.8 Shrnutí

Databáze využívající XML můžeme rozdělit na nativní XML databáze, které používají XML datový model, hodící se na ukládání dokumentových XML dokumentů a semistrukturovaných datových XML dokumentů. XML datový model modeluje data jako uspořádaný strom.

Databáze s podporou XML používají vlastní datový model do něhož mapují jednotlivé instance XML datového modelu užitím XML schémat. Je vhodné je použít pro datové XML

dokumenty. Jejich hlavní výhodou je zachování funkčnosti původních aplikací, nevýhodou naopak nemožnost aplikovat je na všechny typy dokumentů.

eXist je nativní XML databáze s otevřeným zdrojovým kódem, několika implementovanými API, indexem s dynamickým číslováním úrovní uzlů a indexem pro full-textové vyhledávání. To ji předurčuje nejen při vyhledáváních v kolekcích libovolné velikosti, ale s přiměřeným výkonem i při úpravách uložených XML dokumentů.

Kapitola 3

Aplikace tvořené nad XML databázemi

V této kapitole si uvedeme ve kterých případech použít nativní XML databáze (dále jen databáze, není-li řečeno jinak), několik příkladů nasazení databází z praxe pro databázi eXist. Kompletní studie lze nalézt například v [21], kapitoly 10 až 16.

Zjednodušeně řečeno, databáze se vyplatí použít pokud potřebujeme spravovat větší množství XML dokumentů, a to typu dokumentových XML dokumentů a nebo semi-strukturovaných datových XML dokumentů. Další možnosti použití jsou ve správě dlouho běžících transakcí, často se měnících schématech dat, práce s velmi velkými dokumenty, integrovaní dat pocházejících z různých zdrojů, práce s hierarchickými daty a v neposlední řadě tvorba webových portálů. Následující příklady čerpají z [19].

3.1 Použití XML databáze pro dokumentové XML dokumenty

Tento případ užití je pro databáze tím nejčastějším[19]. Výhodou databází je zde existence strukturálních dotazovacích jazyků (tato kategorie se překrývá s XML dotazovacími jazyky, řadíme mezi ně tedy například XQuery a XPath) a jejich rozšiřitelnost. Oproti textovým vyhledávačům spojeným s perzistentním úložištěm nabízí XML datový model, snadnou modifikaci uzlů dokumentu, verzování a funkcionalitu typickou pro databáze, tedy řízení přístupu, bezpečnost transakcí, atomicitu operací a zachovávání integrity dat (do určité míry, viz kapitola 2.5.2).

3.1.1 Správa dokumentů

Ukládání a získávání dokumentů z úložiště je pro XML databáze snadné. Každému XML dokumentu může být přiřazen identifikátor, navíc je možné je řadit do kolekcí. Dokumenty uložené v databázi jsou uložené bezztrátově, tedy se zachováním pořadí elementů, včetně procesních instrukcí, komentářů a CDATA sekcí, fyzicky řazené na disku a tudíž jejich získání je velmi rychlou operací.

3.1.2 Vyhledávání dokumentů

Potřebujeme-li například vyhledat XML dokumenty z jedné pobočky naší firmy a vrátit je v původní podobě, v databázích můžeme použít XML dotazovací jazyky.

Navíc při textovém vyhledávání můžeme rozlišovat mezi daty samotnými, elementy [*markup*], a strukturou. Tyto dotazy nemohou být snadno přeloženy do jazyka SQL. Rovněž nevyžadují pevnou strukturu dokumentu, postačuje, když elementy ve vyhledávaných dokumentech mají přibližně stejný význam.

Zdrojový kód 3.1: Vyhledání měření sondy s více než 3 množinami dat, příklad strukturálního dotazu

```
for $nasa in collection("nasa-data")
let $mereni := $nasa//mereni
where fn:count($mereni) > 3
return $nasa
```

3.1.3 Získávání informací a opakované využívaní obsahu

Za pomocí XML dotazovacích jazyků můžeme nejen v dokumentech vyhledávat, ale navíc vytvářet úplně nové dokumenty založené na datech obsažených v databázi. Takto je možné například snadno vytvářet specifické verze dokumentů pro jednotlivé klienty.

3.2 Integrace dat

Integrace dat z nejrůznějších zdrojů je druhým nejčastějším případem užití pro nativní XML databáze[19]. Datový model XML je velmi flexibilní a mnoho nástrojů podporuje přímý export do XML. XQuery je jazyk připravený na kombinování více datových zdrojů a transformaci výsledku do žádaného formátu.

Při zpracovávání dat z více zdrojů je největším problémem rozdílnost schémat. Schémata se mohou lišit strukturálně, tedy stejná data (například adresa) jsou v jednom vyjádřena jedním jednoduchým elementem, ve druhém naopak komplexním elementem; zároveň se však mohou lišit sémanticky: cena je uvedena v jiných měnách, bez započtené daně a podobně.

Uvažujeme-li více datových zdrojů, je třeba rozlišit lokální a distribuované dotazy. Lokální dotazy jsou snazší na implementaci a rychlejší na zodpovězení, ale mohou vést na neaktuální data. Distribuované jsou jejich opakem. eXist momentálně nepodporuje distribuované dotazy.

3.2.1 Integrace dat rozdělených do kolekcí dle schémat

Tento případ je typický pro relační databáze, nicméně může nastat i v nativních XML databázích. Nabízí se následující řešení:

Převod dokumentů na stejné schéma: Data se mohou převádět v době ukládání do databáze nebo za běhu. Toto řešení není vždy možné, viz kapitola 3.4, limituje (hledá se průnik mezi daty) výsledné možnosti vyhledávání, naopak zjednodušuje výslednou aplikaci,

Vyhledávání v každé z kolekcí zvlášť: Pro každou kolekci se vytvoří dotaz, jejich výsledky se potom sdružují do společného formátu. Jazyk XQuery je na tuto variantu velmi dobře připraven,

Vybudování společného indexu: Některé databáze (eXist nevyjímaje) umožňují určit, jak bude vytvořen index. Pokud bychom uvažovali pro měření vesmírné sondy následující XML Schemata, bylo by řešením v obou kolekcích vybudovat index na elementu `mereni`.

Zdrojový kód 3.2: XML Schema pro měření sondy, s a bez rodičovského elementu `kolekce-mereni`

```
...
<element name="mereni"
    maxOccurs="unbounded">
    <complexType>
        <sequence>
            <element name="id"
            type="integer" />
                ...
        </sequence>
    </complexType>
</element>
...
```

```
...
<element name="kolekce-mereni">
    <sequence>
        <element name="mereni"
            maxOccurs="unbounded">
            <complexType>
            <sequence>
                <element name="id"
                type="integer" />
                ...
            </sequence>
            </complexType>
        </element>
    </sequence>
</element>
...
```

3.2.2 Integrace dat bez rozlišených kolekcí

Pokud nejsou data rozděleny, aplikační logika je složitější. V dokumentech je nutné najít společné položky a na nich založit dotazy, popřípadě provést migrace schémat. Není-li to možné, XQuery lze využít na efektivní textové vyhledávání v dokumentech. Poslední možností je rozdíly ignorovat, a tím nechat rozlišení dat na uživateli.

3.3 Použití XML databáze pro semistrukturální datové XML dokumenty

Semistrukturální XML dokumenty (kapitola 2.3.3) lze v nativních XML databázích ukládat snadněji než v relačních databázích. Nativní XML databáze používají datový model XML, účinně ukládají řídká data, umožňují indexování všech elementů dokumentu, podporují XML dotazovací jazyky a textové vyhledávání. Řešení postavené na relačních databázích (index) a kompletních souborech (hierarchická struktura souborového systému, CLOB a podobně) se ukazují jako nedostatečně robustní, neefektivní a nerozšiřitelné[19].

3.4 Dokumenty s rychle se měnícími schématy

V průběhu života každé aplikace se mění data, se kterými pracuje. V případě relační databází tyto změny probíhají pomalu, kvůli technickým problémům (migrace dat na nové schéma a úprava aplikací), které vyžadují důkladné testování. Často je nutné zajistit kompatibilitu aplikací směrem vpřed a vzad.

Nativní XML databáze jsou na tyto změny mnohem lépe připraveny:

Relační databáze

- Při změně schématu je nutno migrovat data, což je velmi náročná operace. Pro některé data není migrace možná vůbec (obchodní smlouvy a jiné zákonem vymezené dokumenty),
- Pokud data nelze migrovat, nová data musí být uložena podle nového schématu, SQL dotazy na nezměněné hodnoty nejsou funkční, protože nezahrnují data uložené do nového schématu,
- Data nemohou být uložena dokud není známo jejich schéma, popř. musíme použít obecné schéma.

Nativní XML databáze

- Schéma se může měnit bez nutnosti migrovat data,
- Nové dokumenty můžeme ukládat do stejné kolekce jako původní, XML dotazy jsou funkční přes nová i stará data beze změn,
- Data mohou být uložena do databáze i pokud nemůžou být ihned použity.

3.5 Dlouho běžící transakce

Při zpracování požadavků reálného světa jsou některé transakce dlouho trvající a často vyžadují lidský zásah. Z počítačového hlediska se často dají rozdělit do více jednotlivých částí (ideálně ACID transakcí), čímž se zkracuje doba zamčení jednotlivých částí databáze pro výlučný přístup[22].

Na rozdíl od ACID transakcí se účinek dlouho trvající transakce v případě problému ruší pomocí kompenzací. Kompenzace je akce, která uvede databázi do stavu před provedením transakce, není na ní ovšem možné nahlížet jako na pouhé zrušení akce. V reálném světě existují akce, které není možno jednoduše zrušit. Je-li vydáno příliš mnoho letenek na běžný let, jsou cestující přesvědčováni, aby letěli pozdějším letadlem, například pomocí cenového zvýhodnění, není možné jejich letenky prostě zrušit.

Dlouho trvající transakce je možno rozdělit na několik menších transakcí a k ním příslušejících kompenzací (poslední transakce nemá kompenzaci). Tato sekvence se nazývá sága [*saga*]. Nastane-li při provádění transakce chyba, jsou provedeny odpovídající kompenzace v obráceném pořadí. Sága ovšem porušuje princip izolace modelu ACID. Rozšíření konceptu kompenzace je možné nalézt v [22].

Nativní XML databáze lze využít pro dlouhotrvající transakce jako:

Datový sklad: Dlouho běžící transakce jsou často používají dokumentové XML dokumenty nebo data integrované z více zdrojů,

Frontu zpráv: Databáze mohou provádět směrování zpráv na základě obsahu,

Archiv metadat: Metadata lze ukládat stejně snadno jako data.

Několik systémů SOA [Service Oriented Architecture] používají nativní XML databáze: Sonic ESB, Software AG's Enterprise Service Integrator nebo Openlink's Virtuoso[19].

3.6 Případy použití databáze eXist v praxi

V těchto případech byla nasazena databáze eXist:

Atelier éditorial du Guide Thérapeutique: Vydavatelství lékářských a zdravotnických knih a časopisů.
V databázi je více než 1500 záznamů, přibývá 200 knih ročně. Používá XQuery, XUpdate, XMLDB API.
http://www.masson.fr/

The Tibetan Buddhist Resource Center: Katalog tibetských textů, informací o historickém, sociálním a kulturním původu.
Celkem až 10000 knih, 10 miliónů naskenovaných stran. Používá XQuery, XSLT a automatickou generaci PDF.
http://tbrc.org/

Devon Community Directory: Katalog spolků, organizací a služeb v Devonu, UK.
Přibližně 5000 dokumentů s častými změnami. Používá XQuery, XSLT a Javu.
http://www.devonline.gov.uk/community/

Tyto a další známé případy nasazení databáze eXist jsou uvedeny v [29].

3.7 Shrnutí

Nativní XML databáze se používají převážně pro správu dokumentů, integrování dat a jako úložiště semistrukturálních XML dokumentů. V těchto případech se jiná řešení ukazují jako příliš složitá, neefektivní nebo náročná na údržbu. Dalšími obvyklými případy použití jsou vyhledávání ve velmi velkých dokumentech a tvorba webových portálů. Poslední případ bude naši případovou studií po zbytek publikace.

Hlavními devizami nativních XML databází je datový XML model, schopnost pracovat s daty bez definovaných schémat a dotazovací XML jazyk XQuery. Mnoho nástrojů podporuje XML jako výchozí nebo exportní formát, XML se tak stává de facto standardem pro výměnu informací a databáze jsou přirozeným způsobem jeho uložení. Situaci shrnuje citátem Arun Gaikwad[25]:

> An XML Database System is something which you may think is unnecessary but once you start using it, you wonder how you would survive without it.

Kapitola 4

Případová studie - portál Wikipedie

V této kapitole navrhneme zdroj dat a požadavky na aplikace porovnávající řešení založené na nativní XML databázi (dále jen **NXMLDB**) a na relační databázi (dále jen **ORDB**). Popíšeme architekturu jednotlivých aplikací, základní instalaci a konfiguraci komponent pod systémem specifikovaném v příloze B.1.

Dále zmíníme proces uložení dat do databází, konfiguraci aplikační v aplikačním serveru JBoss. Kapitolu zakončí popis společné komponenty zodpovědné za převod značkovacího jazyka *MediaWiki* na abstraktní syntaktický strom, dále zpracovávaný pro potřeby každé z aplikací.

4.1 Wikipedie

Wikipedie (ostatních jazycích také *Wikipedia*) je mnohojazyčná encyklopedie s otevřeným a svobodným obsahem[16], do níž mohou přispívat prakticky všichni uživatelé internetu. Byla založena v roce 2001. Textová data, která jsou použita jak podklad vytvořených aplikací, jsou šířena pod licencí GNU FDL, což umožňuje jejich volné kopírování a použití za uvedení autorů práce. Ostatní média uložená ve Wikipedii jsou šířena pod různými otevřenými licencemi, nicméně tato nejsou v záběru této práce.

V roce 2007 obsahovala česká verze portálu Wikipedie přibližně 76 000 článků, představujíc tak 0,9% celkového obsahu encyklopedie Wikipedia [16].

V březnu 2008 česká Wikipedie obsahovala přibližně 186 000 článků. Objem dat a licence, pod kterými jsou šířeny, byl hlavním důvodem pro její výběr jako případové studie.

Články uložené ve Wikipedii užívají uživatelsky přívětivý značkovací jazyk *MediaWiki*. Pro tento jazyk momentálně neexistuje formální definice gramatiky[15], jako de facto standard je určen existující parser *MediaWiki*. Základní syntaxe je k dispozici například na stránce http://meta.wikimedia.org/wiki/Help:Wikitext_examples.

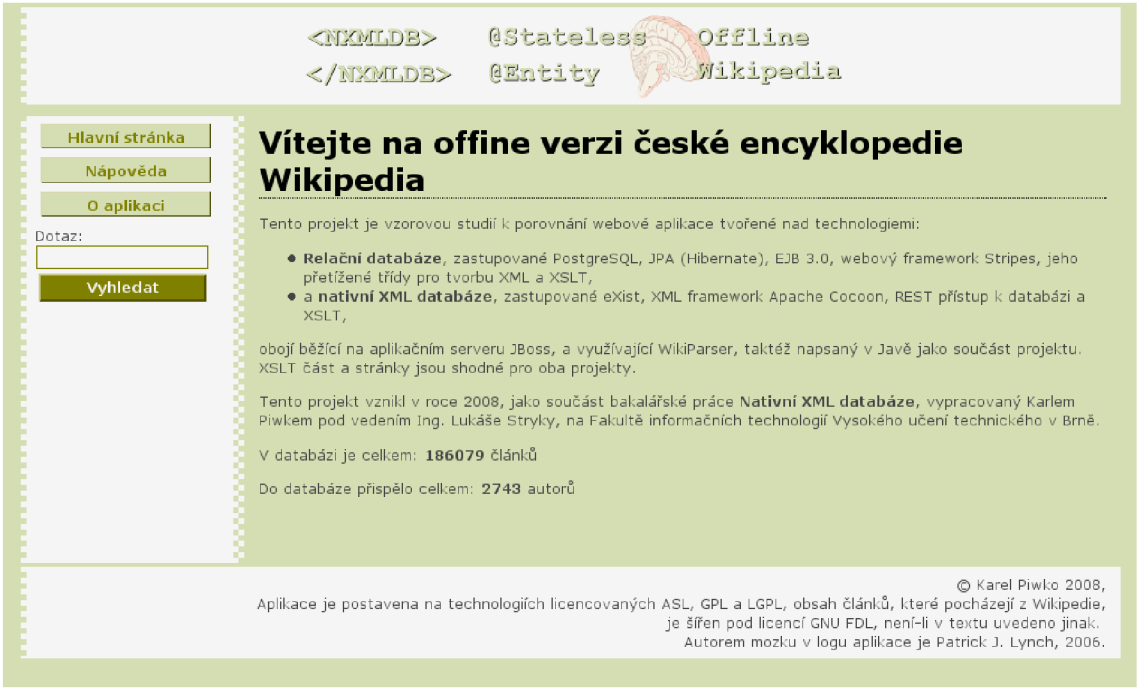
4.1.1 Získání dat

Obsah české Wikipedia je možno stáhnout na stránce http://download.wikimedia.org/cswiki/, soubor `pages-articles.xml.bz2`. Aplikace používá obraz databáze ze dne 20. března 2008. Vybraný soubor obsahuje pouze poslední revize článků. Po rozbalení má velikost 477MiB, což se dá považovat za dostatečnou zátěž pro test obou aplikací.

4.2 Navržená funkcionalita aplikace

Webový portál bude splňovat následující požadavky:

- Jednoduché uživatelské rozhraní,
- Vyhledávání v článcích pomocí formuláře s následujícím chováním:
  - je-li nalezen článek se shodným názvem, je zobrazen uživateli,
  - je-li nalezeno více článků, které v názvu obsahují vyhledávaný řetězec, je vrácen seznam dostupných článků,
  - není-li nalezen žádný článek s vyhledávaným řetězcem, je o tom uživatel informován,
- ze syntaxe *MediaWiki* bude zachováno dělení na odstavce, nadpisy, odkazy mezi články a externí odkazy, šablony budou odstraněny,
- a vzhled a chování aplikace založené na nativní XML databázi a aplikace založené na relační databázi bude shodný.



Obrázek 4.1: Úvodní stránka aplikace

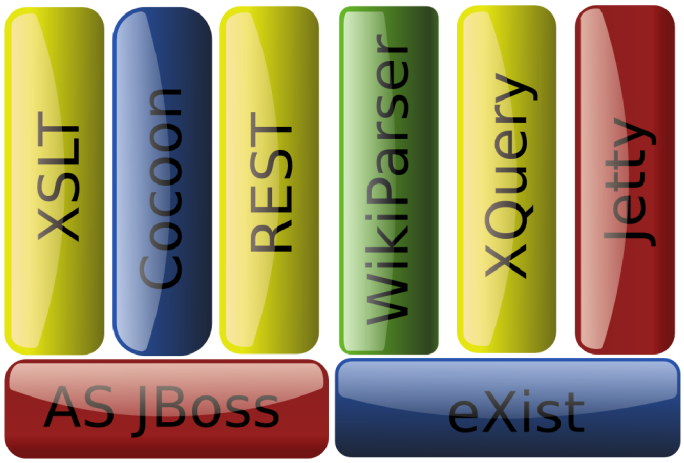
4.2.1 Společné rysy aplikací

Obě aplikace jsou napsány v jazyce Java a běží pod aplikačním serverem JBoss. Další použité programovací jazyky jsou XML, XSL, XQuery, XPath a CSS. Pro kompilaci, sestavení, nahrání na server a zálohování aplikace používají nástroj Apache Ant.

Společným prvkem obou aplikací je komponenta *Wikiparser*, která slouží pro převod textu uloženém ve formátu *MediaWiki* do datové struktury zpracovatelné aplikacemi. Dále obě aplikace sdílejí statický obsah a XSL transformační šablony.

4.3 Aplikace NXMLDB

Aplikace je tvořena WAR archivem, ve kterém jsou XML fragmenty stránek (menu a podobně), statický obsah, konfigurační soubory a javové knihovny frameworku Apache Cocoon. Do databáze je přistupováno přes HTTP/REST, v databázi jsou uloženy dotazy v jazyce XQuery, které jsou při přístupu přes HTTP/REST automaticky vyhodnoceny. Komponenty aplikace jsou zobrazeny na obrázku 4.2.



Obrázek 4.2: Stavební komponenty aplikace NXMLDB

4.3.1 Konfigurace databáze eXist

Instalace databáze eXist probíhá po stažení z [1] za pomocí grafického nebo textového rozhraní ze souboru JAR. Během instalace se zadává heslo pro superuživatele a umístění databáze. eXist byl nainstalován do adresáře `/var/lib/eXist`, dále označovaný jako `$EXIST_HOME`. V naší instalaci běží databáze eXist běží samostatně na výchozím portu `8088` ve vestavěném webovém kontejneru Jetty, pod dedikovaným uživatelem `exist`.

eXist se v samostatném módu spouští pomocí souboru `start.jar`, s parametrem `standalone` a nastavenou proměnnou prostředí Java `exist.home`. Jako konfigurační soubor se v tomto případě používá `conf.xml`. K vypnutí databáze se používá podobný postup, mění se pouze parametr `standalone` na `shutdown`, a následují další s uživatelským jménem, heslem a identifikátorem instance řetězcem URI, ve výchozí hodnotě `xmldb:exist://localhost:8088/xmlrpc`.

Běžící instance ve výchozí konfigurací poskytuje rozhraní XML-RPC, WebDAV a REST. Podrobnosti a spouštěcí skript jsou uvedeny v příloze B.2.

Vzhledem k tomu, že eXist používá kontejner Jetty, který porušuje specifikaci Java Servlet 2.3, respektive 2.4, tím, že mění kódování kontejneru z výchozího ISO-8859-1 na UTF-8, a v aplikaci dochází k interakci mezi Jetty a Tomcat (součást AS JBoss), bylo

v konfiguračním souboru databáze eXist změněno výchozí kódování kontejneru Jetty na ISO-8859-1.

Zdrojový kód 4.1: Fragment konfiguračního souboru `$EXIST_HOME/conf.xml`

```
...
<rest enabled="yes" context="/*">
    <!--
        Special params: set form-encoding and container-encoding. If they
        are set to different encodings,
        eXist may need to recode form parameters.
    -->
    <param name="form-encoding" value="UTF-8"/>
    <param name="container-encoding" value="ISO-8859-1"/>
</rest>
...
```

Dále byla do adresáře určeného pro uživatelské knihovny `$EXIST_HOME/lib/user` nahrána knihovna *Wikiparser*u a potřebné závislosti, aby mohla být syntaxe *MediaWiki* zpracována během vyhodnocování dotazů XQuery. Podrobnosti jsou v kapitole 4.5.

4.3.2 Uložení dat do databáze

Pro uložení a indexaci dat je nejsnazší použít klienta, který se spouští opět pomocí souboru `start.jar` s parametrem `client`. V následujícím okně vyplníme identifikátor instance, uživatelské jméno a heslo a zvolíme type připojení *Remote*.

V databázi vytvoříme dvě nové kolekce - první bude obsahovat samotná data a druhá XQuery dotazy ve formě souborů XQL. Dále vytvoříme nového uživatele, vlastníka obou kolekcí. V daném nastavení pro jednoduchost povolíme čtení kolekcí všem uživatelům (výchozí stav), abychom se nemuseli autentizovat při vykonávání dotazů HTTP/REST, případně zadat výchozího uživatele pro dotazy HTTP/REST v konfiguračním souboru.

Zdrojový kód 4.2: Ukázka definice výchozího uživatele pro dotazy rozhraním REST

```
<rest enabled="yes" context="/*">
    ...
    <param name="use-default-user" value="true" />
    <param name="default-user-username" value="username" />
    <param name="default-user-password" value="password" />
    ...
</rest>
```

Vytvoříme tedy kolekce `nxmldb` a `nxmldb-pages`. Do první kolekce umístíme soubor `cswiki-datum-pages-articles.xml` [1], do druhé potom dotazy XQuery jako typ *Binary resources*. Tímto jsou při volání HTTP/REST automaticky dotazy vyhodnoceny [28], čehož budeme využívat v použitém webovém frameworku. Indexy databáze jsou uloženy v adresáři `$EXIST_HOME/webapp/WEB-INF/data/`.

[1] Při vkládání souboru o velikosti 477MiB eXist naalokoval přibližně 1 300MiB operační paměti (měřeno programem `top`). Na testovací konfiguraci B.1 vzhledem k swapování nebyla indexace dokončena ani za 24 hodin. Soubor byl indexován na stroji s 4GiB RAM, procesorem Intel Core 2 Duo E6750 a diskem SATA II za 20 minut reálného času. Adresář s databází byl poté přemístěn.

4.3.3 Konfigurace webového frameworku Apache Cocoon

Apache Cocoon[10], dále jen Cocoon, je webový framework, který je založen na principu komponent, které se mohou vyvíjet nezávisle na sobě a jejich skládáním vzniká webový portál. Složení komponent se nazývá *pipeline*. Tyto je možné opět použít jako stavební komponenty.

Ve výchozí konfigurace Cocoon podporuje různé datové zdroje: XML soubory, XML databáze, relační databáze, webové služby, LDAP a další. Databáze eXist není podporována ve výchozí konfiguraci, přístup přes protokol XMLDB je zde zastoupen databázi Apache Xindice. Podporu eXist lze relativně snadno přidat, jelikož ale budeme využívat samostatnou instanci databáze a volání HTTP/REST, nebudeme muset Cocoon modifikovat [2].

Cocoon se kompiluje ze zdrojového kódu. Po rozbalení souboru nalezneme v adresářové struktuře kompilační skript `build.sh` a konfigurační soubory `blocks.properties` a `build.properties`. Poslední dva zmiňované zkopírujeme s prefixem `local.` na `local.build.properties` a `local.blocks.properties` a v nim zrušíme nepotřebné moduly. Výsledné soubory jsou v příloze B.3[3].

Poté Cocoon zkompilujeme a necháme si vytvořit kostru webové aplikace příkazem `./build.sh webapp`. Knihovny v adresáři `build/cocoon/WEB-INF/lib` budou součástí naší aplikace, dále použijeme soubory `cocoon.xconf`, `logkit.xconf`, `web.xml` a `sitemap.xmap` (první tři z adresáře `build/cocoon/WEB-INF`, poslední z `build/cocoon`).

4.3.4 Konfigurace webové aplikace

Webový popisovač aplikace `web.xml` s výchozí konfigurací Cocoonu nemusíme příliš upravovat, postačí, zakomentujeme-li servlet Jetty, protože používáme jiný kontejner a zkontrolujeme, je-li kódování znaků kontejneru nastaveno na ISO-8859-1 a kódování znaků formuláře na UTF-8. Výsledný soubor je v příloze B.4.

Poslední součástí konfigurace je vytvoření skriptu, který z adresářové struktury vytvoří WAR archiv a nahraje ho na server. K tomuto účelu je používán nástroj Apache Ant a jeho příkazů [*task*].

4.4 Aplikace ORDB

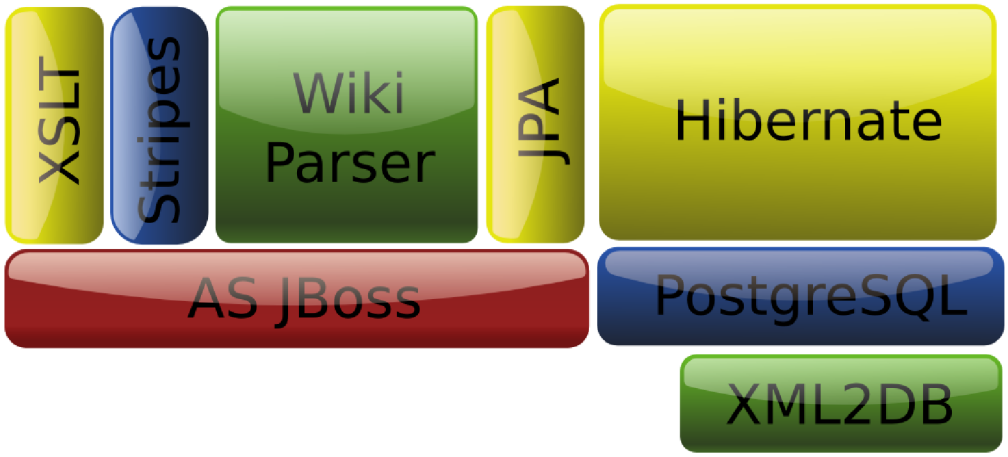
Aplikace je tvořena EAR archivem, jehož součástí je popisovač aplikace, EJB komponenta a WAR archiv, obsahující XML fragmenty stránek, statický obsah a konfigurační soubory. Přístup do databáze je řešen přes lokální zdroj dat, nakonfigurovaným v aplikačním serveru, JPA a EJB beany. Komponenty aplikace jsou zobrazeny na obrázku 4.3.

Převodem dat z XML dokumentu do tabulek relační databáze se zabývá příloha A.1, výsledné schéma je uvedeno v příloze A.2 a konečně převod XML dokumentu za pomocí parseru SAX a JDBC do tabulek databáze řeší příloha A.4.

Knihovnami JAR potřebnými pro běh aplikace jsou aplikační framework Stripes včetně závislostí, *Wikiparser* a jeho závislosti; pro kompilaci knihovny JPA a EJB3. Pokud bychom chtěli používat JSP stránky, což není náš případ, je vhodné zahrnout také knihovny s elementy JSTL.

[2] Pro použití databáze eXist společně s Cocoonem a protokolem XMLDB je třeba nahradit knihovny XMLDB a XML-RPC distribuovanými s Cocoonem verzemi přítomnými v instalaci databáze eXist.

[3] Zde uvedená konfigurace je stále zredukovatelná, nicméně umožňuje rozšíření funkcionality například stránkami XSP bez nutnosti rekompilace.



Obrázek 4.3: Stavební komponenty aplikace ORDB

4.4.1 Zdroj dat

V databázi PostgreSQL jsme vytvořili nového uživatele `nxmldb`, novou databázi `nxmldb` a zároveň i tabulky, například skriptem uvedeným v příloze A.3 a poté jsme je naplnili daty. Zbývá informovat aplikační server JBoss o existenci datového úložiště a nahrát ovladač JDBC pro databázi PostgreSQL mezi knihovny serveru.

Zdroj dat je nakonfigurován v souboru `ordb-ds.xml`, který je nahrán na server stejně jako samotný EAR archiv. V tomto souboru je uveden ovladač k databázi, její JNDI identifikátor a uživatelské jméno a heslo.

Zdrojový kód 4.3: Popisovač zdroje dat `ordb-ds.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>ordb_ds</jndi-name>

    <connection-url>jdbc:postgresql://localhost:5432/nxmldb</connection-url>
    <driver-class>org.postgresql.Driver</driver-class>
    <user-name>nxmldb</user-name>
    <password>password</password>
  </local-tx-datasource>
</datasources>
```

4.4.2 Konfigurace modulů v prostředí aplikačního serveru

Pro Java EE aplikace používáme speciální popisovač `application.xml`, který obsahuje popis a konfiguraci jednotlivých modulů EAR archivu. Pro naši aplikaci obsahuje pouze dva moduly, EJB a samotný WAR archiv.

Zdrojový kód 4.4: Popisovač Java EE modulů pro aplikaci ORDB

```
<?xml version="1.0" encoding="UTF-8"?>

<application xmlns="http://java.sun.com/xml/ns/j2ee"
```

```
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
            http://java.sun.com/xml/ns/j2ee/application_1_4.xsd"
            version="1.4">

  <display-name>ORDB</display-name>
  <description>Object Relational Database</description>

  <module>
    <ejb>ordb-ejb.jar</ejb>
  </module>

  <module>
    <web>
      <web-uri>ordb.war</web-uri>
      <context-root>ordb</context-root>
    </web>
  </module>
</application>
```

4.4.3 Přístup k datovému zdroji z aplikace

Nutnou součástí modulu EJB je soubor `persistence.xml`, který obsahuje název jednotky, typ zdroje dat (v našem případě zdroj dat nakonfigurovaný v kapitole 4.4.1), popřípadě doplňující parametry pro poskytovatele JPA [Java Persistence API].

Soubor `persistence.xml` se povinně musí nacházet v adresáři `META-INF` EJB modulu.

Zdrojový kód 4.5: Konfigurační soubor `persistence.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
       http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">

    <persistence-unit name="ordb">
        <jta-data-source>java:/ordb_ds</jta-data-source>
        <properties>
            <property name="hibernate.archive.autodetection" value="class"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="hibernate.dialect"
        value="org.hibernate.dialect.PostgreSQLDialect"/>
        </properties>
    </persistence-unit>
</persistence>
```

Dále musíme aplikaci informovat používaném JNDI kontextu souborem `jndi.properties`, umístěným do adresáře `WEB-INF` WAR archivu.

Zdrojový kód 4.6: Konfigurační soubor `jndi.properties`

```
java.naming.factory.inital=org.jnp.interfaces.LocalOnlyContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
```

4.4.4 Konfigurace aplikačního frameworku Stripes

Stripes[13] je aplikační framework pro tvorbu webových aplikací za použití Javy, jehož hlavní devizou je nulová externí XML konfigurace, namísto toho jsou používány anotace. V aplikaci je používán společně s rozšířením na automatickou injekci EJB beanů [33].

Stripes používá koncept ActionBeanů, tříd zodpovědných za vygenerování pohledu v závislosti na požadavku. ActionBean je mapovaný na určitou URL, jsou mu předány případné parametry GET nebo POST. Výsledkem pohledu je objekt typu `Resolution`. V aplikaci se používá vlastní typ, `XMLResolution`, který vytváří pohled v XML instanciací XML šablony a dále transformuje na XHTML, které je zobrazené uživateli. Podrobnosti jsou v kapitole 5.2.

Jediná konfigurace se tedy provádí ve webovém popisovači aplikace `web.xml`, kde nastavíme pro webový filtr Stripes rozšiřující balíčky (`Extension.Packages`), balíčky obsahující ActionBeany (`ActionResolver.Packages`) a zároveň jeho mapování na servlet *StripesDispatcher*. Zároveň nastavíme mapování servletu na vzory URL, tak aby mohly být volány ActionBeany. Výsledný soubor se nachází v příloze B.5.

I v aplikaci ORDB se používá Apache Ant ke kompilaci, sestavení EJB modulu, WAR archivu, EAR archivu a nahrání aplikace na server.

4.5 Wikiparser

Tato komponenta provádí převod syntaxe *MediaWiki* na abstraktní syntaktický strom, tvořen elementy `ASTNode`. Každý z těchto elementů obsahuje `Content` s vlastním obsahem a dynamické pole potomků. Jednotlivé podtřídy `ASTNode` odpovídají elementům *MediaWiki*, které parser umí rozpoznat.

4.5.1 Parsování dat

Data jsou získána z databáze. Protože bylo původně počítáno pouze s rozdělením na odstavce (element `Paragraph`), probíhá rozpoznávání na dva průchody. Základem je Mealyho stavový automat, implementující rozhraní `WikiParser`, s využitím pomocné parametrizovatelné struktury `ParseStack`.

Komponenta používá návrhový vzor *Visitor* pro zpracování vzniklého abstraktního syntaktického stromu průchodem *preorder*. Za pomocí jeho implementace, třídy `ParagraphParseVisitor` je výsledek parsování automatu `SimpleWikiParser` znovu zpracován automatem `ParagraphParser` do výsledné podoby.

4.5.2 Zpracování abstraktního syntaktického stromu

Komponenta dále využívá další dvě implementace vzoru Visitor, a sice `DOMVisitor`, používaný aplikací ORDB pro vytvoření DOM stromu, který je vložen do nadřazeného DOM stromu během instanciace XML šablony a dále `ExistDOMVisitor`, který vytváří DOM strom během volání funkce jazyka XQuery definované v přídavném modulu.

Kompletní popis je ve zdrojových kódech komponenty. Diagram tříd je v nahlédnutí v příloze C.1.

4.6 Shrnutí

V této kapitole jsme navrhli webovou aplikaci pro porovnání řešení postaveném na nativní XML databázi a relační databázi. Výsledné řešení předpokládá nakonfigurovaný aplikační server JBoss, popřípadě jiný aplikační server.

Pro aplikaci NXMLDB jsme popsali postup konfigurace, spuštění a indexace dokumentů v databázi eXist. Zároveň jsme popsali konfiguraci a kompilaci webového frameworku Apache Cocoon, který je slouží ke skládání statického a dynamického obsahu. V neposlední řadě jsme popsali webový popisovač aplikace, základní součást všech webových aplikací na platformě Java.

Pro aplikaci ORDB jsme si po převodu dat popsali vytvoření zdroje dat, jeho registraci v aplikačním serveru a jeho konfiguraci v aplikaci. Letmo jsme popsali popisovač aplikace pro EAR archiv a stručně komentovali obsah webového popisovače aplikace.

Pro zpracování obsahu *MediaWiki* jsme navrhli *Wikiparser*, který byl navržen tak, aby spolupracoval s oběma aplikacemi.

Kapitola 5

Vzájemné porovnání aplikací

V této kapitole se budeme zabývat rozdíly mezi aplikacemi NXMLDB a ORDB. Nejprve popíšeme mapovaní URL na ActionBeany frameworku Stripes a toto porovnáme se schopnostmi frameworku Cocoon. Následně se budeme zabývat vlastním rozšířením `XMLResolution` pro instanciaci XML šablon během generování XML pohledu v Stripes a odpovídajícímu řešení ve frameworku Cocoon.

Poté se zaměříme na samotné vyhledávání v databázi a konstrukci odpovědi v XQuery a XML šabloně s využitím EJB a JPA. Toto vyhledávání bude předmětem výkonového testu.

5.1 Mapování URL

Javové aplikace mapují ve webovém popisovači určitý vzor URL na příslušný servlet. Servlet je dále zodpovědný za zpracování požadavku a zobrazení odpovědi. Toto mapovaní (v popisovačích element `servlet-mapping` a jeho následník `url-pattern`) je pro obě aplikace velmi podobně, liší se pouze v tom, že Cocoon ve výchozí konfiguraci obsluhuje i statické zdroje, jako šablony CSS a obrázky. Webové popisovače aplikací jsou v přílohách B.4 a B.5.

V aplikacích budeme využívat dvojí mapovaní. Prvním je mapování dotazů s příponou *html*, které slouží pro zobrazení statických stránek a k vyhledávání pomocí formuláře, druhým je zobrazení článků a obsluha vnitřních odkazů[1] Wikipedie.

5.1.1 Mapování ve frameworku Stripes

Stripes automaticky mapuje veškeré třídy umístěné v balíčcích `ActionResolver.Packages` (viz. kapitola 4.4.4) na název ActionBeanu s odstraněným s odstraněnými řetězci „Action“, „Bean“, a názvy balíčků „www“, „stripes“, „web“ a „action“ na URL s příponou *action*[12]. Chceme-li toto chování změnit, použijeme u třídy anotaci `@UrlBinding()`, kde přímo nastavíme URL.

V aplikaci ORDB používáme verzi Stripes 1.5 beta 1 kvůli podpoře Clean URL, připomínajících Apache `mod_rewrite`. Část URL ve složených závorkách odpovídá parametru GET či POST je automaticky dostupná v proměnné, která je součástí ActionBeanu. Clean URL jsou součástí vnitřních odkazů Wikipedie.

[1] Vnitřní odkaz je odkaz na jiný článek umístěný na Wikipedii.

Zdrojový kód 5.1: Příklady mapovaní ActionBeanu

```
...
// mapování pro stránku obsahující informace o aplikaci
@UrlBinding("/about.html")
public class AboutAction extends AbstractAction {
...
// mapování Clean URL s výchozí hodnotou parametru
@UrlBinding("/art/{q=Vysoké učení technické v Brně}")
public class ArticleAction extends AbstractAction {
...
```

Protože jeden ActionBean může obsluhovat více událostí (uvažujme například formulář s více tlačítky), je vhodné anotací určit výchozí metodu, která je spuštěna, nebo přímo přiřadit určité události metodu. Stripes se ve výchozí konfiguraci snaží rozpoznat metodu podle stisknutého tlačítka a názvu metody[11]. Více ke konfiguraci Stripes anotacemi k nalezení v [11].

Zdrojový kód 5.2: Mapování metod na události

```
...
// výchozí metoda
@DefaultHandler
public Resolution list() throws Exception {
...
// výchozí metoda a zároveň metoda pro událost 'vyhledat'
@DefaultHandler
@HandlesEvent("vyhledat")
public Resolution search() throws Exception {
...
```

Dodejme ještě, že třída `AbstractAction` je abstraktní implementací rozhraní Stripes `net.sourceforge.stripes.action.ActionBean`.

5.1.2 Mapování ve frameworku Cocoon

Cocoon pro mapování URL používá konfigurační soubor `sitemap.xmap` (viz kapitola 4.3.3), který umístíme do kořenového adresáře WAR archivu. Cocoon je výrazně flexibilnější než Stripes, pro mapovaní na URL používá více tříd *matcher* schopných rozpoznat například výrazy s podporou *wildcard*, regulární výrazy nebo konkrétní jazykové nastavení uživatele. My si v naší aplikaci vystačíme s výchozím typu `wildcard`.

Veškeré nastavení umístíme do elementu `<map:pipeline>`. Element, zodpovědný za rozpoznání URL a následné vykonání akce se nazývá `<map:match>`. Konfigurace je velmi variabilní a případným zájemcům doporučuji literaturu [30]. V při kolizi možných *matcher*ů má přednost první uvedený, je tedy vhodné umisťovat ty více obecné na konec elementu `<map:pipeline>`.

Zdrojový kód 5.3: Mapování URL v `sitemap.xmap`

```
<map:pipeline>
    <!-- je-li URL 'article.html', proveď následující akci -->
    <map:match pattern="article.html">
    ...
```

```
    </map:match>
    ...
<!-- mapuj URL vnitřního odkazu Wikipedie
<map:match pattern="*">
    <!-- zde je obsah '*' dostupný v proměnné {1} -->
</map:match>
...
</map:pipeline>
```

Parametrů '*' může být více než jeden, používá se také '**', který pokryje libovolný řetězec včetně lomítek. *Matcher*y je vhodné používat nejen pro stránky, ale například pro centrální uložení šablon, multimediálního obsahu a podobně, bez ohledu úroveň zanoření samotného URL.

5.2 Zobrazení XML pohledu

Pro udržení konsistence vzhledu a ovládání generují obě aplikace výstup v XML, který je transformační šablonou dále zpracován na XHTML, které je zobrazeno uživateli. Jednotlivé stránky jsou skládány ze statických částí (například menu), částí odpovídajích statické stránce (například stránka „O aplikaci“) a výsledku dotazu předzpracovaného komponentou *Wikiparser*. Obě aplikace provádějí XSL transformaci na serverové části, ať už programově (ORDB) nebo jako součást *pipeline* v Cocoonu.

5.2.1 XML šablony v aplikaci ORDB

Stripes používá na zobrazení obsahu stránky JSP nebo Freemarker. Ačkoliv bychom je mohli používat k vytvoření XML, která by byla transformována na klientovi, vytvořili jsme další implementaci rozhraní `net.sourceforge.stripes.action.Resolution`, `XMLResolution`, která používá koncept šablon pro vygenerování XML stránky na základě použité šablony a obsahu ActionBeanu. Detailní diagram tříd je v příloze C.2.

Šablona `XMLTemplate` je složena s jednotlivých elementů, implementací rozhraní `TemplatePart`. Tyto elementy jsou uloženy do seznamu v daném pořadí. Šablona implementuje návrhový vzor *Visitor*, který v závislosti na typu `TemplatePart` generuje výstup. V aplikaci se používá `DOMVisitor`, který generuje DOM strom. Element `WikiElement` interně používá *Wikiparser*, a výsledek vkládá do existujícího DOM stromu. DOM strom je poté transformován na XHTML, které je zobrazeno uživateli.

Zdrojový kód 5.4: Vytvoření a použití XML šablony pro článek v metodě ActionBeanu `ArticleBean`

```
public Resolution list() throws Exception {
...
    // Vytvoření šablony s použitím ActionBeanu
    XMLTemplate t = new ArticleTemplate(this);
    // umístění XSLT šablony
    String stylesheet = getPrefix() + Configuration.BASIC_STYLESHEET;
    // pro Visitor nastavujeme objekt pro výstup a XSLT šablonu
    Visitor v = new DOMVisitor(ctx.getResponse().getWriter(),stylesheet);
    // XMLResolution obsahuje šablonu, Visitor a content-type
    return new XMLResolution(t,v, "text/html");
}
```

5.2.2 Agregace více zdrojů v aplikaci NXMLDB

Uvnitř elementu `<map:match>` obvykle v Cocoonu následují elementy `<map:generate>` sloužící jako zdroj dat, `<map:transform>` provádějící transformaci a jako poslední `<map:serialize>`, který serializuje transformovaný dokument pro zobrazení uživateli a nebo zpracování v jiné *pipeline*.

Velkou výhodou Cocoonu je implementace více protokolů a pseudoprotokolů jako možného zdroje generátoru. Můžeme tak používat *http://*, pro volání služeb HTTP/REST, *file://* pro získání souborů v souborovém systému počítače, *cocoon://* pro přístup k jiným *pipeline*, *context://* pro zdroje umístěné v kontextu servletu, *xmldb://* pro čtení z nativních XML databází a další[30]. Potřebujeme-li spojit více *pipeline* do jednoho generátoru, jednoduše použijeme element `<map:aggregate>`.

Zdrojový kód 5.5: Agregace zdrojů a volání REST pro článek

```
<map:match pattern="article.html">
    <!-- zde nastavujeme podporu čtení argumentů
         získaných z formuláře metodou GET nebo POST -->
<map:act type="request">
        <map:parameter name="parameters" value="true"/>
        <!-- agregace do elementu „page\
        <map:aggregate element="page">
            <!-- statické menu -->
            <map:part src="context:/components/menu.xml"/>
            <!-- dotaz REST na databázi -->
<map:part src="http://localhost:8088/db/nxmldb-pages/article.xql?q=q"/>
        </map:aggregate>

        <!-- transformace -->
        <map:transform src="resources/stylesheets/basic.xsl">
            <!-- případné parametry transformace předané šabloně -->
        </map:transform>
        <!-- serializace -->
        <map:serialize encoding="UTF-8" type="xhtml"/>
    </map:act>
</map:match>
```

5.3 Vyhledávání v databázi

Nyní když máme připravené šablony, zbývá získat obsah z databáze. Zde se obě aplikace asi nejvýrazněji liší. NXMLDB používá jazyk XQuery, který data vrací ve formě XML, ta jsou v Cocoonu rovnou transformována a zobrazena uživateli.

Pro ORDB je postup poněkud složitější. ActionBean nejprve za pomocí JNDI vyhledá EJB bean zodpovědný za získání dat, voláním jeho metody JPA provádí JDBC dotaz na databázi. Nalezená data jsou transformována na persistentní entitu, objekt POJO [Plain Old Java Object] a takto jsou vrácena ActionBeanu. Ten je použije společně s XML šablonou, která je zpracována, transformována a zobrazena uživateli. Výkonovým rozdílem se budeme zabývat v kapitole 5.4.

5.3.1 Přístup do relační databáze přes JTA a JPA

Pro práci s JPA musíme nejprve vytvořit jednotlivé persistentní entity budoucího EJB archivu. Persistentní entita je označena anotací `@Entity` a implementuje rozhraní `java.io.Serializable`, přístup k atributům může být přímý nebo s využitím *getter*ů a *setter*ů. Její atributy jsou mapovány na sloupce relační databáze. Dále je nutné určit primární klíč (`@Id`)a definovat vztahy mezi tabulkami (`@OneToOne`, `@OneToMany` a další). Názvy atributů můžeme pomocí dalších anotací upravovat, například můžeme přiřadit `@Id` sekvenci uloženou v databázi, ze které se budou získávat hodnoty. Celkové možnost JPA výrazně přesahují tuto práci a čtenáři je doporučenou nahlédnout do [17].

Zde si předvedeme entitu `Page`, odpovídají tabulce `Page` relační databáze.

Zdrojový kód 5.6: Část entity `Page`

```
@Entity
// vytvoření generátoru vázaného na sekvenci uloženou v DB
@SequenceGenerator(name="PageSeq", sequenceName="PageSeq")
// pojmenované dotazy, jedna s variant vyhledávání v DB
@NamedQueries({
    @NamedQuery(name="total-pages",
        query="SELECT COUNT(*) FROM Page"),
    @NamedQuery(name="page-by-title",
        query="FROM Page WHERE title LIKE :title")
})
public class Page extends BaseBean {
    // nastavení identifikátoru a generátoru na sekvenci
    @Id @GeneratedValue(strategy=GenerationType.AUTO, generator = "PageSeq")
    private Long id;
    ...
    // vazba na entitu Revision
    @OneToMany(mappedBy="page", fetch=FetchType.LAZY)
    private Set<Revision> revisions;
    ...
```

Dále vytvoříme rozhraní pro *Stateless* EJB bean a jeho implementaci. EJB za použití JTA automaticky získá objekt *Session* nebo v našem případě *EntityManager*. Dotaz se připojí k již probíhající transakci, je-li to možné. Používají se *cache* a *pool* pro získání připojení. To vše za nás automaticky řeší aplikační server.

Zdrojový kód 5.7: Část EJB pro práci s entitou `Page`

```
@Stateless
public class ManagePageBean implements ManagePage {

    // injekce EntityManageru z JTA
    @PersistenceContext
    private EntityManager em;

    // celkový počet stránek v databázi získáme voláním pojmenovaného dotazu
    public Long totalPages()
        return (Long) em.createNamedQuery("total-pages")
                .getSingleResult();
    }
...
```

Poslední částí, která nám zbývá, je injekce EJB beanu do ActionBeanu. Využijeme k tomu rozšíření Stripes pro EJB3 beany[33].

Zdrojový kód 5.8: Injekce EJB beanu do ActionBeanu

```
@UrlBinding("/art/{q=Vysoké učení technické v Brně}")
public class ArticleAction extends AbstractAction {
    ...
    @EJBBean("ordb/ManagePageBean/local")
    protected ManagePage mp;
    ...
}
```

Poté už nám zbývá pouze ve obslužné metodě události zavolat metodu EJB beanu, naplnit hodnoty ActionBeanu a na správné místě XML šablony vypsat hodnoty v něm obsažené.

5.3.2 Dotaz v nativní XML databázi v jazyce XQuery

Jazyk XQuery[9] použijeme nejen na vyhledání dat podobně jako XPath[3], ale pomocí výrazového prostředku FLWOR [For, Let, Where, Order by, Return] můžeme snadno konstruovat fragmenty XML. Interpret jazyka XQuery obsažený v eXist nám umožňuje snadno rozšířit funkcionalitu moduly psanými v Javě.

Dotazy samotné jsme uložili do databáze (viz kapitola 4.3.2) a používáme automatického vyhodnocování dotazů HTTP/REST. Ekvivalent vyhledání počtu stránek v relační databázi bude v XQuery vypadat takto:

Zdrojový kód 5.9: Vyhledání počtu stránek v XMLDB

```
(: soubor uložený v databázi obsahuje XML Schema s definicí namespace:)
declare namespace mw="http://www.mediawiki.org/xml/export-0.3/";

(: počet všech mw:page v kolekci :)
let $pages := count(collection("/db/nxmldb")//mw:page)
return
    <query>
        <total-articles>{$pages}</total-articles>
    </query>
```

Zde můžeme vidět, že ačkoliv práce s XML databází vyžaduje navíc ovládání jazyka XQuery, samotná integrace dotazů do webové aplikace je ve spojení s Cocoonem snazší a křivka rychlosti vývoje aplikace je minimálně ze začátku strmější.

5.4 Rychlost zpracování dotazů

V aplikaci jsme provedli několik testů, které měli určit, jak dlouho trvá vyhledání dat. Měření dat proběhlo v pěti etapách:

1. Měření času vyhledání dat v relační databázi,

2. Měření času od inicialize ActionBeanu po vrácení `XMLResolution`, což odpovídá zpracování požadavku a předání dat webovému prohlížeči,
3. Měření času vyhledání dat v nativní XML databázi s rozšířením *Wikiparser*,
4. Měření času vyhledání dat v nativní XML databázi bez rozšíření *Wikiparser*,
5. a konečně doba zpracování požadavku frameworku Cocoon až do předání dat webovému prohlížeči.

Všechny měření byly provedeny desetkrát v řadě a zde je použit aritmetický průměr. Pro měření 1. byla použita direktiva `\timing` relační databáze PostgreSQL klienta `psql`. Následující měření využívá koncept *Interceptor*u Stripes, umožnující provést libovolný kód v určité fázi zpracování požadavku. Časy 3. a 4. jsou odečteny z grafického klienta databáze eXist a konečně poslední čas pochází z logu frameworku Cocoon. Jednotkami jsou milisekundy.

Zdrojové kódy dotazů jsou uvedeny v příloze B.6.

Typ dotazu	Relační DB	Stripes	eXist(Wikiparser)	eXist	Cocoon
Vyhledání článku Brno v databázi	1 615	3 237	2 838	2 700	3 795
Počet článků v databázi	113	222	-	586	704
Počet příspěvších autorů	10	127	-	27 209	26 130

Tabulka 5.1: Rychlost vyhodnocení dotazů

Výsledky ukazují, že full-textové rozšíření vyhledávání `near()` a odpovídající regulární výraz v databázi PostgreSQL jsou jedinou kategorií, ve které je výkon obou řešení srovnatelný. Ačkoliv je dotaz proveden v relační databázi přibližně o sekundu rychleji, způsob mapování JPA vede na vytvoření mnoha objektů typu `Page` a ve výsledném čase zpracování požadavku se již tolik databáze neliší.

Pro zjištění počtu článků v databázi se již databáze výrazně liší, nicméně celkový rozdíl zobrazení stránky není pro uživatele viditelný. Tento jednoduchý dotaz ukazuje, že sečíst počet řádků v tabulce je skutečně rychlejší než zjištění počtu potomků elementu.

Skutečný problém ovšem nastává pokud potřebujeme nejen zjistit počet potomků, ale zároveň zajistit jejich unikátnost, je v tomto případě nutné provést porovnávání všech nalezených elementů. V našem případě vybíráme textové porovnávání uživatelského jména. V relační databázi za nás tuto práci obstaral převodní program (viz příloha A.4), který každému uživateli přiřadil nový primární klíč a zamezil existenci duplicit.

5.5 Shrnutí

Během porovnávání aplikací jsme si ukázali způsob mapování URL na události a *pipeline* a následné zpracování požadavku. Ačkoliv je Stripes javovým frameworkem s minimální konfigurací, počáteční spojení všech komponent je výrazně náročnější na čas a znalosti programátora než použití frameworku Cocoon.

Co ovšem ORDB ztrácí na eleganci řešení, dohání svým výkonem. Pokud bychom potřebovali články pouze zobrazovat, dala by se aplikace NXMLDB považovat za rovnocennou, v jiných dotazech prohrává řádem třídy.

Kapitola 6

Závěr

Nativní XML databáze nejsou technologií, která by měla převzít dominanci na trhu s daty nad nejrozšířenějšími relačními databázemi. V této práci jsme si ukázali, že je možné navrhnout aplikaci tak, aby využívala jedné ze dvou databázových technologií, aniž by uživatel zpozoroval rozdíl.

6.1 Zhodnocení výsledků práce

Během vypracovávání práce se technologie okolo XML databází ukázala jako dostatečně vyspělá. Ačkoliv jsme v aplikaci ORDB použili framework Stripes s minimální konfigurací, aplikační server se postaral o databázovou část a navíc jsme použili koncept Clean URL, aplikace NXMLDB se vyvíjela výrazně rychleji.

Aplikace NXMLDB navíc měla více oddělené komponenty, které mohly být individuálně testovány. Ačkoliv framework Stripes podporuje objekty typu *mock*, jejich použití k testování by ještě více prodloužilo dobu nutnou na vývoj aplikace.

Uvažujeme-li, že databáze eXist nemusí běžet pouze v módu *standalone*, ale i s použitím web kontejneru Jetty a vestavěným Cocoonem, kde se nutnost konfigurace ještě snižuje, stávají se webové aplikace založené na nativních XML databázích ideálním řešením pro rychlý vývoj aplikace pracující s daty, kde výkon není primárním cílem.

Programovací jazyk Java umožnil snadné rozšíření funkcionality komponent obou aplikací, s využitím anotací navíc redukoval potřebu externích konfiguračních souborů na minimum. Java je ovšem i společným problémem, protože dostupnost hostování na serverech je v porovnání s jinými jazyky velmi špatná.

6.1.1 Volba případové studie

Volba *Wikipedie* jako vzorových dat se ukázala dvousečnou zbraní. Sice jsme tím získali representativní vzorek dat, nicméně tvorba parseru pro formát *MediaWiki* ubrala času samotné realizace funkčnosti aplikace.

Po problémech s indexací se nativní XML databáze ukázala být schopná pracovat s velkým objemem dat. Data ovšem neodpovídaly tomu, co by bylo pro nativní XML databáze ideální. Struktura dat byla pravidelná a nedefinovaných dat bylo poskrovnu. Po převedení dat do relační databáze nenastal problém měnícího se schématu.

Pokud by nastala změna ve schématu, XML databáze by ukázala svou světlou stránku. Jelikož jsou samotné dotazy XQuery uložené přímo v databázi, není potřeba zasahovat do aplikace běžící na serveru. XML výstup bychom uzpůsobili tak, aby odpovídal použité

šabloně, XML dokument dle nového schématu bychom uložili do databáze a modifikovali vyhledávaní tak, aby podporovalo i nové XML Schema. Celková změna by zahrnovala pouze změnu XQL souborů.

Naopak, uvedená operace by v aplikaci ORDB znamenala vytvoření nového schématu, převodní aplikace a eventuálně migrace původních dat. Dále bychom museli modifikovat transportní entity POJO a pravděpodobně i EJB beany. Rovněž by mohla být nutná modifikace XML šablon a ActionBeanu, stručně, museli bychom zrevidovat celou aplikaci.

6.2 Náměty na rozšíření

Jedním z cílů do budoucnosti je naprogramovat aplikaci ORDB s využitím nové verze Apache Cocoon (2.2), která je založena na frameworku Spring. Takto bychom mohli měnit pouze databázový *back-end*, což je obě aplikace ještě více sjednotilo a dalo lépe vyniknout rozdílu mezi databázemi.

Do aplikací by bylo vhodné přidat další funkcionalitu:

Správa revizí Aplikace jsou po technické stránce připraveny, zbývá tedy implementace rozhodování mezi různými revizemi,

Editace Se správou revizí přichází v úvahu editace. Jak dotazy HTTP/REST, tak Stripes jsou dobře připraveny. Bylo by vhodné použít uživatelsky přívětivý editor v JavaScriptu. Zároveň bychom tím získali možnost otestovat rychlost vkládání do databáze a rozšíření XInclude.

Wikiparser Komponenta Wikiparser by mohla navíc umět:

- Rozpoznávat seznamy,
- pasovat jedním průchodem za cenu složitějšího automatu a
- pamatovat si nejen pořadí, ale i umístění potomků uvnitř elementu.

6.3 Osobní přínos

Při tvorbě aplikace jsem se naučil tvorbě XSLT šablon a jazyku XQuery. Ujasnil jsem si rozdíl mezi XML dokumenty a porozuměl jazyku pro definici schémat. Zároveň jsem nabyl znalosti, jak tyto schémata použít při ukládání dat do relační databáze.

Při používání databáze eXist jsem si vyzkoušel práci s nativní XML databází a koncept HTTP/REST. Přesvědčil jsem se, že práce s XML databázemi je velmi pohodlná, zároveň jsou snadno rozšiřitelné v Javě. Nativní XML databáze nyní považuji za perspektivní technologii, a v menších webových aplikacích jí budu nahrazovat řešení Stripes s JPA.

Využil jsem dřívějších znalostí frameworku Stripes a JPA, abych je poprvé použil na aplikačním serveru v kombinaci s EJB a JTA, a naučil jsem se základně nakonfigurovat aplikační server JBoss. U nahrávání aplikací na server jsem zužitkoval Apache Ant a rozšířil si v něm znalosti ohledně tvorby balíčků WAR a EAR.

6.4 Návaznost na jiné práce

Tuto práci lze použít jako výchozí bod pro bakalářskou práci „*Formát XML pro značkování slovníků*“. Čtenář by čtením kapitol 2 a 3 měl identifikovat svá data a nemá-li zájem zvolit

proprietární řešení, vybrat si mezi relační a nativní XML databázi jako úložiště. Dále by mohl svou aplikaci rozšířit o webový *front-end*.

Dalším užitím této studie je použití nativní XML databáze nebo databáze s podporou XML jako zdroj dat pro bakalářskou práci „*Interaktivní vizualizace XML*“.

Dalším využití má komponenta *Wikiparser*, která má vzhledem ke konstrukci abstraktního stromu a koncepci *Visitor*ů potenciál být použita v pracích pod vedením doc. RNDr. Pavla Smrže, Ph.D., například v práci zaobírající se sémantickou Wiki.

Literatura

[1] eXist: Open Source Native XML database. http://exist.sourceforge.net/, ze dne 18. března 2008.

[2] ISO 8879:1986(E). Information processing — Text and Office Systems — Standard Generalized Markup Language (SGML), 1986.

[3] XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath/, ze dne 12. března 2008, 1999.

[4] RELAX NG Specification. http://relaxng.org/spec-20011203.html, ze dne 12. března 2008, 2001.

[5] XML Schema Part 0. http://www.w3.org/TR/xmlschema-0/, ze dne 12. března 2008, 2004.

[6] Extensible Markup Language (XML) 1.0. http://www.w3.org/TR/REC-xml/, ze dne 12. března 2008, 2006.

[7] SO/IEC 9075-14:2003 Information technology – Database languages – SQL – Part 14: XML-Related Specifications (SQL/XML), 2006.

[8] XML Path Language (XPath) Version 2.0. http://www.w3.org/TR/xpath20/, ze dne 12. března 2008, 2007.

[9] XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/, ze dne 12. března 2008, 2007.

[10] Apache Cocoon. http://cocoon.apache.org/2.1/, ze dne 7.5.2008, 2008.

[11] Stripes - Annotation Reference. http://www.stripesframework.org/display/stripes/Annotation+Reference, ze dne 7.5.2008, 2008.

[12] Stripes - Quick Start Guide. http://www.stripesframework.org/display/stripes/Quick+Start+Guide, ze dne 7.5.2008, 2008.

[13] Stripes - Stripes Framework. http://www.stripesframework.org/, ze dne 7.5.2008, 2008.

[14] Marcelo Arenas. Normalization Theory for XML. *ACM SIGMOD Record*, 35(4):57–64, 2006.

[15] Komunita autorů. MediaWiki. http://en.wikipedia.org/wiki/MediaWiki, ze dne 6. května 2008, 2008.

[16] Komunita autorů. Wikipedie, otevřená encyklopedie. http://cs.wikipedia.org/wiki/Wikipedie, ze dne 6. května 2008, 2008.

[17] Christian Bauer and Gavin King. *Java Persistence With Hibernate*. Manning, 2007. ISBN 1-932394-88-5.

[18] Ronald Bourret. XML and Databases. http://www.rpbourret.com/xml/XMLAndDatabases.htm, ze dne 12. března 2008, 2005.

[19] Ronald Bourret. Going native: Use cases for native XML databases. http://www.rpbourret.com/xml/UseCases.htm, ze dne 12. března 2008, 2007.

[20] Timo Böhme and Erhard Rahm. Supporting efficient streaming and insertion of XML data in RDBMS, 2004.

[21] Akmal B. Chaudhri, Awais Rashid, and Roberto Zicari. *XML Data Management: Native XML and XML-Enabled Database Systems*. Addison-Wesley Professional, 2003. ISBN 0-201-84452-4.

[22] Mandy Chessell, Catherine Griffin, David Vines, Michael Butler, Carla Ferreira, and Peter Henderson. Long Duration Transaction in Software Design Projects. *IBM SYSTEMS JOURNAL*, 41(4), 2002.

[23] doc. Ing. Jaroslav Zendulka and Ing. Ivana Rudolfová. Studijní opora předmětu Databázové systémy IDS, 2006.

[24] Daniela Florescu and Donald Kossman. Storing and Querying XML Data using an RDMBS. *IEEE Data Engineering Bulletin*, 22(3):27–34, 1999.

[25] Arun Gaikwad. Introduction to Xindice. http://www.ibm.com/developerworks/web/library/wa-xindice.html, ze dne 13. dubna 2008, 2002.

[26] Wolfgang Meier. *Web, Web-Services, and Database Systems – eXist: An Open Source Native XML Database*. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2002.

[27] Wolfgang Meier. Index-driven XQuery processing in the eXist XML database. http://exist.sourceforge.net/xmlprague06.html, ze dne 23. března 2008, 2006.

[28] Wolfgang Meier. Developer's Guide: Writing Web Applications Using XQuery. http://exist.sourceforge.net/devguide_xquery.html#storedxq, ze dne 7. května 2008, 2008.

[29] Wolfgang Meier. Powered By eXist. http://demo.exist-db.org/apps/applications.xml, ze dne 13. dubna 2008, 2008.

[30] Lajos Moczar and Jeremy Aston. *Cocoon Developer's Handbook*. Sams, 2002. ISBN 0672322579.

[31] Will Provost. Normalizing XML, Part 1. http://www.xml.com/pub/a/2002/11/13/normalizing.html, ze dne 1. dubna 2008, 2002.

[32] Will Provost. Normalizing XML, Part 2. http://www.xml.com/pub/a/2002/12/04/normalizing.html, ze dne 1. dubna 2008, 2002.

[33] Samuel Santos. Stripes - EJB3 with Stripes. http://www.stripesframework.org/display/stripes/EJB3+with+Stripes, ze dne 7.5.2008, 2008.

[34] Jayavel Shanmugasundaram, Eugene Shekita, Jerry Kiernan, Rajasekar Krishnamurthy, Efstratios Viglas, Jeffrey Naughton, and Igor Tatarinov. A general technique for querying XML documents using a relational database system. *ACM SIGMOD Record*, 30(3):20–26, 2001.

[35] Bart Steegmans, Ronald Bourret, Owen Cline, Olivier Guyennet, Shrinivas Kulkarni, Stephen Priestley, Valeriy Sylenko, and Ueli Wahli. *XML for DB2 Information Integration*. IBM Redbooks, 2004. ISBN 0738490032.

Seznam použitých zkratek

ACID	Atomicity, Consistency, Isolation, Durability
BCNF	Boyce-Codd Normal Form
BLOB	Binary Large Object
CDATA	Character Data
CLOB	Character Large Object
DDL	Data Definition Language
DLN	Dynamic Level Numbering
DOM	Document Object Model
DTD	Document Type Definition
EAR	Enterprise Archive
EE	Enterprise Edition
EJB	Enterprise Java Bean
ESIEE	École Supérieure d'Ingénieurs en Électronique et Électrotechnique
FDL	Free Documentation License
FLWOR	For, Let, Where, Order by, Return
GNU	GNU's Not Unix
HTTP	Hypertext Transfer Protocol
JDBC	Java Database Connectivity
JDOM	Java Document Object Model
JNDI	Java Naming and Directory Interface
JPA	Java Persistence API
JSP	Java Server Pages
JSR	Java Specification Request
JSTL	JavaServer Pages Standard Tag Library
JTA	Java Transaction API
LDAP	Lightweight Directory Access Protocol
NF	Normal Form
ORM	Object-Relational Mapping
PCDATA	Parsed Character Data
POJO	Plain Old Java Object
RELAG NG	Regular Language for XML Next Generation
REST	Representational State Transfer
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
SOA	Service Oriented Architecture
SOAP	původně Simple Object Access Protocol
SQL	Structured Query Language

URI	Uniform Resource Identifier
WAR	Web Application Archive
WebDAV	Web-based Distributed Authoring and Versioning
WS	Web Service
XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language
XMLDB	XML Database
XML-RPC	XML Remote Procedure Call
XNF	XML Normal Form
XQJ	XQuery for Java
XQL	XQuery Language
XSL	eXtensible Stylesheet Language
XSP	eXtensible Server Pages

Seznam příloh

Tato práce obsahuje následující přílohy:

Dodatek A

Převod XML na tabulky relační databáze

A.1 Vytváření schématu relační databáze z XML dokumentu

Mapování XML dokumentu na schéma relační databáze je prvním krokem nutným pro jeho uložení. Dalšími kroky je potom rozložení samotného dokumentu na fragmenty odpovídajícím řádkům v relacích vzniklého schématu [*shredding*] a nakonec převod XML dotazovacího jazyka na jeho ekvivalent v jazyce SQL, a zodpovězení ve formě XML [*publishing*].

Existuje více způsobů, jak vytvořit toto schéma a dokonce existují automatizované prostředky jako doplňky či přímé součásti databází s podporou XML. Většina z těchto postupů ovšem vyžaduje speciální preprocesor nebo procesor pro převod mezi dotazovacími jazyky typu XML a SQL. V této příloze si uvedeme algoritmus uvedený v [34], pro nějž odpadá nutnost speciálního preprocesoru. Další algoritmus, *hranový*, je uvedený například v [24], který zachovává vztahy mezi elementy XML dokumentu je tedy vhodný spíše pro použití v relačních databázích sloužící jako podklad nativních XML databází.

Podmínkou zpracování dokumentu tímto jsou:

1. Mapování XML na relace je bezztrátové, tedy je zde dostatek informací o struktuře dokumentu,
2. a XML elementy a atributy jsou mapovány do sloupců relací, čímž usnadňujeme tvorbu SQL dotazů.

Jako příklad si uvedeme zjednodušené XML Schema pro MediaWiki, pro XML dokument určený na export článků. Vychází z http://www.mediawiki.org/xml/export-0.3.xsd a je zjednodušené pro pokrytí potřeb aplikace. Algoritmus [34] je ilustrován pomocí XML Schema namísto původního DTD:

Zdrojový kód A.1: Zjednodušené XML Schema MediaWiki

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:mw="http://www.mediawiki.org/xml/export-0.3/"
    targetNamespace="http://www.mediawiki.org/xml/export-0.3/"
    elementFormDefault="qualified">

    <annotation>
```

```
    <documentation xml:lang="en">
        MediaWiki's page export format
    </documentation>
</annotation>

<!-- Need this to reference xml:lang -->
<import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd"/>

<!-- Our root element -->
<element name="mediawiki" type="mw:MediaWikiType"/>

<complexType name="MediaWikiType">
    <sequence>
        <element name="page" type="mw:PageType"
            minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="version" type="string" use="required"/>
    <attribute ref="xml:lang" use="required"/>
</complexType>

<complexType name="PageType">
    <sequence>
        <!-- Title in text form.
        (Using spaces, not underscores; with namespace ) -->
        <element name="title" type="string"/>

        <!-- optional page ID number -->
        <element name="id" type="positiveInteger" minOccurs="0"/>

        <!-- current revision -->
        <element name="revision" type="mw:RevisionType" />
    </sequence>
</complexType>

<complexType name="RevisionType">
    <sequence>
        <element name="id" type="positiveInteger" minOccurs="0"/>
        <element name="timestamp" type="dateTime"/>
        <element name="contributor" type="mw:ContributorType"/>
        <element name="minor" minOccurs="0" />
        <element name="comment" type="string" minOccurs="0"/>
        <element name="text" type="mw:TextType" />
    </sequence>
</complexType>

<complexType name="TextType">
    <simpleContent>
        <extension base="string">
            <attribute ref="xml:space" use="optional"
                default="preserve" />
        </extension>
    </simpleContent>
</complexType>
```

```
    <complexType name="ContributorType">
        <sequence>
            <element name="username" type="string" minOccurs="0"/>
            <element name="id" type="positiveInteger" minOccurs="0" />
            <element name="ip" type="string" minOccurs="0"/>
        </sequence>
    </complexType>
</schema>
```

Z definice vidíme, že kořenovým elementem dokumentu je `mediawiki` typu `mw:MediaWikiType`. Protože v aplikaci očekáváme pouze jeden XML dokument obsahující aktuální verzi článků Wikipedie, nebudeme tento element v aplikaci modelovat, podobně jako v ER diagramu nemodelujeme entitní množinu, která representuje celý modelovaný systém [23].

Kořenový element obsahuje libovolný počet elementů `page` typu `mw:PageType`. Jak je ukázáno v kapitole 2.5.1, definice v tomto případě odpovídá kompozici a byla by modelována vztahem 1:M.

Element `page` je podle XML Schema komplexním typem, obsahujíc elementy `title` typu `string`, volitelně `id` typu `positiveInteger` a `revision` typu `mw:RevisionType`, přesně v uvedeném pořadí, vynuceném elementem `sequence`.

Volitelnost elementu rozpoznáme velmi jednoduše. Pro každý element mimo kořenového je možné definovat atributy `minOccurs` s výchozí hodnotou `1` a `maxOccurs` s výchozí hodnotou rovněž `1`. Nejsou-li tedy definovány, element se musí v XML dokumentu vyskytovat právě jednou. Minimální hodnota `minOccurs` je `0`, element se nemusí vyskytovat a je tedy volitelný. Minimální hodnota `maxOccurs` je vždy větší nebo rovna hodnotě `maxOccurs` a zároveň `1`. Maximální hodnota `maxOccurs` je neomezená, označená slovem `unbounded`.

Pro element `page` máme teoreticky k dispozici kandidátní klíč `title`, jelikož je velice pravděpodobné, že nebudou existovat dvě stránky se shodným názvem, přesto přidáme automaticky generovaný primární klíč, `id`, a element `id` XML dokumentu přejmenujeme na `wid`, což nám v budoucnu usnadní práci s objektově relačním mapováním. Navrhneme tedy následující relaci:

Zdrojový kód A.2: Relace pro element mw:PageType XML Schema

```
CREATE TABLE Page(
    id BIGINT NOT NULL,
    wid BIGINT DEFAULT NULL,
    title VARCHAR(255) NOT NULL,
    revision BIGINT NOT NULL,
    PRIMARY KEY(id),
    CONSTRAINT RevFK FOREIGN KEY(revision) REFERENCES Revision
);
```

Podobně bude postupovat pro typ `mw:RevisionType` a `mw:ContributorType`, pro type `mw:TextType` s výhodou využijeme délkově neomezeného datového typu databáze PostgreSQL `text`. Za povšimnutí stojí, že pro element `contributor` nejsou povinné žádné vnořené elementy, zde je tedy dodaný primární klíč nutností.

Protože budeme uvažovat více revizí k jedné stránce, budeme vztah mezi `page` a `revision` ve skutečnosti modelovat jako 1:M, za využití pomocné tabulky. Výsledné schéma se nachází v následující kapitole.

A.2 Výsledné schéma XML dokumentu pro relační databázi

Aplikací metody uvedené v příloze A.1 získáme výsledný skript pro generaci tabulek relační databáze PostgreSQL:

Zdrojový kód A.3: Tabulky relační databáze PostgreSQL pro zjednodušené XML Schema

```
DROP TABLE Contributor;
DROP TABLE Page;
DROP TABLE Revision;
DROP TABLE PageRevision;

DROP SEQUENCE ContributorSeq;
DROP SEQUENCE RevisionSeq;
DROP SEQUENCE PageSeq;

CREATE SEQUENCE ContributorSeq INCREMENT BY 1 START WITH 1;
CREATE SEQUENCE RevisionSeq INCREMENT BY 1 START WITH 1;
CREATE SEQUENCE PageSeq INCREMENT BY 1 START WITH 1;


CREATE TABLE Contributor(
        id BIGINT NOT NULL,
                wid BIGINT DEFAULT NULL,
        username VARCHAR(255) DEFAULT NULL,
                ip VARCHAR(80) DEFAULT NULL,
                PRIMARY KEY(id)
);
-- ALTER TABLE Contributor id SET DEFAULT NEXTVAL('ContributorSeq');

CREATE TABLE Revision(
                id BIGINT DEFAULT NULL,
        wid BIGINT DEFAULT NULL,
        contributor BIGINT NOT NULL,
        created TIMESTAMP NOT NULL,
        comment VARCHAR(4096) DEFAULT NULL,
                minor VARCHAR(4096) DEFAULT NULL,
        content TEXT NOT NULL,
        PRIMARY KEY(id),
        CONSTRAINT ContFK FOREIGN KEY(contributor)
                    REFERENCES Contributor
);
-- ALTER TABLE Revision id SET DEFAULT NEXTVAL('RevisionSeq');

CREATE TABLE Page(
                id BIGINT NOT NULL,
        wid BIGINT DEFAULT NULL,
        title VARCHAR(255) NOT NULL,
        PRIMARY KEY(id)
);
-- ALTER TABLE Page id SET DEFAULT NEXTVAL('PageSeq');

CREATE TABLE PageRevision(
                page BIGINT NOT NULL,
```

```
                    revision BIGINT NOT NULL,
                    PRIMARY KEY(page, revision),
                    CONSTRAINT PageRevisionFK1 FOREIGN KEY(page)
                        REFERENCES Page,
                    CONSTRAINT PageRevisionFK2 FOREIGN KEY(page)
                        REFERENCES Revision
);
```

A.3 Vytvoření uživatele, databáze a tabulek pro aplikaci ORDB

Pro všechny data uvažujeme výchozí kódování UTF-8.

Zdrojový kód A.4: Vytvoření uživatele, databáze a tabulek pro aplikaci ORDB

```
#!/bin/sh

USERNAME=nxmldb
DBNAME=nxmldb
FILELIST="tables.sql"

dropdb $DBNAME
dropuser $USERNAME
createuser -D -P $USERNAME
createdb -E UTF8 -O $USERNAME $DBNAME

for i in $FILELIST
do
    psql -U $USERNAME -d $DBNAME -f $i
done
```

A.4 Převod XML dokumentu do relační databáze PostgreSQL

Za využití znalosti JDBC, SAX, PostgreSQL a postupu pro tvorbu relací z XML Schema, lze pomocí následujícího programu uložit XML dokument do relační databáze PostgreSQL:

Zdrojový kód A.5: Převod dat XML dokumentu relační databáze

```
/*
 * Stores MediaWiki XML format into relational database
 */
package xml2db;

import java.io.FileInputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Properties;
import org.xml.sax.Attributes;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.helpers.XMLReaderFactory;

/**
 *
 * @author kapy
 */
public class XML2DBHandler extends DefaultHandler {

    public static final String CONF_FILE_PROP = "xml2db.conffile";
    public static final String CONF_FILE = "xml2db.properties";
    public static final String TIME_FORMAT = "yyyy-MM-dd'T'HH:mm:ss'Z'";
    public static final String FIND_CONTRIBUTOR =
            "SELECT DISTINCT id FROM Contributor WHERE " +
            "(wid IS NOT NULL AND wid = ?) " +
            "OR (username IS NOT NULL AND username = ?) " +
            "OR (ip IS NOT NULL AND ip = ?)";
    public static final String INSERT_CONTRIBUTOR =
            "INSERT INTO Contributor(id,wid,username,ip) " +
            "VALUES(NEXTVAL('ContributorSeq'),?,?,?)";
    public static final String LAST_ID =
            "SELECT LASTVAL()";
    public static final String INSERT_REVISION =
            "INSERT INTO Revision(id,wid,created," +
            "contributor,comment,minor,content) " +
            "VALUES(NEXTVAL('RevisionSeq'),?,?,?,?,?,?)";
```

```
public static final String INSERT_PAGE =
        "INSERT INTO Page(id,wid,title) " +
        "VALUES(NEXTVAL('PageSeq'),?,?)";
public static final String INSERT_PAGE_REVISION =
        "INSERT INTO PageRevision(page,revision) VALUES(?,?)";
protected Connection con;

// parsed data
private WikiPage page;
private WikiState state;
private StringBuffer sb;

public XML2DBHandler(Connection con) {
    this.con = con;
    state = WikiState.BEGIN;
    sb = new StringBuffer();
}

/* ********************************************************** */
/* XML handler methods */

@Override
public void startElement(String uri, String name,
        String qname, Attributes attrs)
        throws SAXException {

    switch (state) {
        case BEGIN:
            if ("page".equals(qname)) {
                page = new WikiPage();
                state = WikiState.PAGE;
            }
            break;
        case PAGE:
            if ("revision".equals(qname)) {
                state = WikiState.REVISION;
                page.setRevision(new WikiRevision());
            }
            break;

        case REVISION:
            if ("contributor".equals(qname)) {
                state = WikiState.CONTRIBUTOR;
                page.getRevision().setContributor(
                        new WikiContributor());
            }
            break;
    }

    // character inside element are stored here,
    // so create new buffer inside every element
    sb = new StringBuffer();

}
```

```
@Override
public void endElement(String uri, String name, String qname)
        throws SAXException {

    switch (state) {
        case PAGE:
            if ("title".equals(qname)) {
                page.setTitle(sb.toString());
            } else if ("id".equals(qname)) {
                page.setWid(Long.valueOf(
                        sb.toString()).longValue());
            }
            break;
        case REVISION:
            if ("id".equals(qname)) {
                page.getRevision().setWid(Long.valueOf(
                        sb.toString()).longValue());
            } else if ("text".equals(qname)) {
                page.getRevision().setText(sb.toString());
            } else if ("comment".equals(qname)) {
                page.getRevision().setComment(sb.toString());
            } else if ("minor".equals(qname)) {
                page.getRevision().setMinor(sb.toString());
            } else if ("timestamp".equals(qname)) {

                // parse data string and store it as sql timestamp
                final DateFormat dt =
                        new SimpleDateFormat(TIME_FORMAT);
                final String timestamp = sb.toString();
                try {
                    java.util.Date date = dt.parse(timestamp);
                    page.getRevision().setTimestamp(
                            new java.sql.Timestamp(date.getTime()));
                } catch (ParseException e) {
                    System.err.println("Unable to parse timestamp: "
                            + timestamp);
                }
            }
            break;
        case CONTRIBUTOR:
            if ("id".equals(qname)) {
                page.getRevision().getContributor().setWid(
                        Long.valueOf(sb.toString()).longValue());
            } else if ("username".equals(qname)) {
                page.getRevision().getContributor()
                        .setUsername(sb.toString());
            } else if ("ip".equals(qname)) {
                page.getRevision().getContributor()
                        .setIp(sb.toString());
            }
            break;
    }
```

```
    if ("contributor".equals(qname)) {
        state = WikiState.REVISION;
    } else if ("revision".equals(qname)) {
        state = WikiState.PAGE;
    } else if ("page".equals(qname)) {
        try {
            storeWikiPage();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        state = WikiState.BEGIN;
    }

}

@Override
public void characters(char[] ch, int start, int len)
        throws SAXException {
    sb.append(ch, start, len);
}

/* ********************************************************** */
// database methods
private void storeWikiPage() throws SQLException {

    PreparedStatement s = null;
    /* ********************************************************** */
    // store contributor

    s = con.prepareStatement(FIND_CONTRIBUTOR);
    s.setLong(1, page.getRevision().getContributor()
            .getWid());
    s.setString(2, page.getRevision().getContributor()
            .getUsername());
    s.setString(3, page.getRevision().getContributor()
            .getIp());

    // there is no contributor with given id, we can store him
    ResultSet rs = s.executeQuery();
    long contributor_id;
    if (rs.next() == false) {
        s = con.prepareStatement(INSERT_CONTRIBUTOR);
        s.setLong(1, page.getRevision().getContributor()
                .getWid());
        s.setString(2, page.getRevision().getContributor()
                .getUsername());
        s.setString(3, page.getRevision().getContributor()
                .getIp());

        s.executeUpdate();

        // get contributor id
        s = con.prepareStatement(LAST_ID);
        rs = s.executeQuery();
```

```
            rs.next();
            contributor_id = rs.getLong(1);

        } // contributor found, get his id
        else {
            contributor_id = rs.getLong(1);
        }

        /* ********************************************************* */
        // store revision

        s = con.prepareStatement(INSERT_REVISION);
        s.setLong(1, page.getRevision().getWid());
        s.setTimestamp(2, page.getRevision().getTimestamp());
        s.setLong(3, contributor_id);
        s.setString(4, page.getRevision().getComment());
        s.setString(5, page.getRevision().getMinor());
        s.setString(6, page.getRevision().getText());

        s.executeUpdate();

        /* ********************************************************* */
        // store page

        // get revision id
        s = con.prepareStatement(LAST_ID);
        rs = s.executeQuery();
        rs.next();
        long revision_id = rs.getLong(1);

        s = con.prepareStatement(INSERT_PAGE);
        s.setLong(1, page.getWid());
        s.setString(2, page.getTitle());

        s.executeUpdate();

        // get page id
        s = con.prepareStatement(LAST_ID);
        rs = s.executeQuery();
        rs.next();
        long page_id = rs.getLong(1);

        // store page & revision
        s = con.prepareStatement(INSERT_PAGE_REVISION);
        s.setLong(1, revision_id);
        s.setLong(2, page_id);
        s.executeUpdate();

    }

    /* ********************************************************** */
    /* static methods */
    /**
     * @param args the command line arguments
```

```
 */
public static void main(String[] args) {

    // load configuration
    Properties conf = new Properties();
    String conffile = System.getProperty(CONF_FILE_PROP);
    if (conffile == null || conffile.equals("")) {
        conffile = CONF_FILE;
    }
    try {
        conf.load(new FileInputStream(conffile));
    } catch (Exception e) {
        System.err.println("Configuration file at: "
                + conffile + " not found or not accesible");
        printUsage();
        System.exit(1);
    }

    // prepare driver
    String driver = conf.getProperty("driver");
    try {
        Class.forName(driver);
    } catch (ClassNotFoundException e) {
        System.err.println("JDBC SQL driver "
                + driver + " not found!");
        System.exit(1);
    }

    // create db connection
    Connection con = null;
    String url = conf.getProperty("url");
    try {
        con = DriverManager.getConnection(url, conf);
    } catch (SQLException e) {
        System.err.println("Unable to connect to DB at: "
                + url);
        e.printStackTrace();
        System.exit(1);
    }


    XMLReader parser = null;
    XML2DBHandler handler = new XML2DBHandler(con);

    // check arguments
    if (args.length == 0) {
        printUsage();
        System.exit(1);
    }

    // create XML reader
    try {
        parser = XMLReaderFactory.createXMLReader();
        parser.setContentHandler(handler);
```

```
            parser.setErrorHandler(handler);
        } catch (SAXException e) {
            System.err.println("Unable to create default " +
                    "XMLParser from factory.");
            System.exit(1);
        }

        // parse files
        for (String fileName : args) {
            try {

                long timeBefore = System.currentTimeMillis();
                parser.parse(new InputSource(fileName));
                long timeAfter = System.currentTimeMillis();

                System.out.println("Total time spent: "
                        + (timeAfter - timeBefore) + "ms");
            } catch (SAXException e) {
                System.err.println("Failed when parsing file: "
                        + fileName);
                e.printStackTrace();
            } catch (IOException e) {
                System.err.println("Unable to open file: "
                        + fileName);
                e.printStackTrace();
            }
        }

        // close DB connection
        try {
            con.close();
        } catch (SQLException e) {
            System.err.println("Unable to close SQL connection");
            System.exit(1);
        }
    }

    private static void printUsage() {
        System.out.println(
                "Stores XML file(s) with mediawiki XML " +
                "schema in relational database.\n" +
                "Configuration is stored in properties file, " +
                "by default: " + CONF_FILE + "\n" +
                "Location can be changed throught "
                + CONF_FILE_PROP + " system's property\n" +
                "Properties used are:\n" +
                "\tdriver - for jdbc driver class\n" +
                "\turl - for jdbc connection url\n" +
                "\tuser - with common meaning\n" +
                "\tpassword - with common meaning" +
                "Usage:\n" +
                "java -jar XML2DB.jar <filename>*");
    }
}
```

```
/* ****************************************************************** */
/* Automaton states */
enum WikiState {

    BEGIN,
    PAGE,
    REVISION,
    CONTRIBUTOR,}

/* ****************************************************************** */
/* JavaBeans */

class WikiPage {

    private long id;
    private long wid;
    private String title;
    private WikiRevision revision;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public long getWid() {
        return wid;
    }

    public void setWid(long wid) {
        this.wid = wid;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public WikiRevision getRevision() {
        return revision;
    }

    public void setRevision(WikiRevision revision) {
        this.revision = revision;
    }
}

class WikiRevision {
```

```
private long id;
private long wid;
private java.sql.Timestamp timestamp;
private WikiContributor contributor;
private String minor;
private String comment;
private String text;

public long getId() {
    return id;
}

public void setId(long id) {
    this.id = id;
}

public long getWid() {
    return wid;
}

public void setWid(long wid) {
    this.wid = wid;
}

public java.sql.Timestamp getTimestamp() {
    return timestamp;
}

public void setTimestamp(java.sql.Timestamp timestamp) {
    this.timestamp = timestamp;
}

public WikiContributor getContributor() {
    return contributor;
}

public void setContributor(WikiContributor contributor) {
    this.contributor = contributor;
}

public String getMinor() {
    return minor;
}

public void setMinor(String minor) {
    this.minor = minor;
}

public String getComment() {
    return comment;
}

public void setComment(String comment) {
```

```
        this.comment = comment;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }
}

class WikiContributor {

    private long id;
    private long wid;
    private String username;
    private String ip;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public long getWid() {
        return wid;
    }

    public void setWid(long wid) {
        this.wid = wid;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getIp() {
        return ip;
    }

    public void setIp(String ip) {
        this.ip = ip;
    }
}
```

Dodatek B

Konfigurace a zdrojové kódy

B.1 Konfigurace testovacího stroje

B.1.1 Hardware

- Intel Pentium M 1500MHz,
- 1024 MiB DDR 333MHz RAM,
- disk ATA 133, 5400 ot./min

B.1.2 Software

- eXist 1.2, SVN revize 7235
- Apache Cocoon 2.1.11
- Java Developement Kit 1.5.0_14
- JBoss 4.2.2GA
- PostgreSQL 8.2.6
- Ovladač JDBC postgresql-8.2-508.jdbc3.jar
- Stripes 1.5 beta 1
- operační systém openSUSE 10.3, jádro Linux 2.6.22.17-0.1-default
- webový prohlížeč Epiphany 2.20.0
- souborový systém XFS

B.2 Spouštěcí skript pro databázi eXist

Pro potřeby automatického spouštěni samostatné databáze byl napsán následující skript. Databáze je automaticky spuštěna pod uživatelem `$EXIST_USER`. Nejdůležitějšími parametry jsou výchozí (`-Xms`) a maximální (`-Xmx`) hodnota paměti přidělené virtuálnímu stroji Javy. Během indexace vzorového dokumentu musela být maximální hodnota zvýšena až na 1536MiB.

Zdrojový kód B.1: Spouštěcí skript databáze eXist

```
#!/bin/sh
#
# Exist Control Stript
#
#
### BEGIN INIT INFO
# Provides: eXist
# Default-Start: 3 5
# Default-Stop: 0 1 2 6
# Description: Start the eXist XML database.
### END INIT INFO

# define where exist is - this is the directory containing
# directories log, bin, conf etc
EXIST_HOME=${EXIST_HOME:-"/var/lib/eXist"}

EXIST_INSTANCE=${EXIST_INSTANCE:-"xmldb:exist://localhost:8088/xmlrpc"}

EXIST_CONSOLE=${EXIST_CONSOLE:-"/var/log/eXist/eXist.log"}

# Shell functions sourced from /etc/rc.status:
#      rc_check         check and set local and overall rc status
#      rc_status        check and set local and overall rc status
#      rc_status -v     ditto but be verbose in local rc status
#      rc_status -v -r  ditto and clear the local rc status
#      rc_failed        set local and overall rc status to failed
#      rc_reset         clear local rc status (overall remains)
#      rc_exit          exit appropriate to overall rc status
. /etc/rc.status

# First reset status of this service
rc_reset

# Return values acc. to LSB for all commands but status:
# 0 - success
# 1 - misc error
# 2 - invalid or excess args
# 3 - unimplemented feature (e.g. reload)
# 4 - insufficient privilege
# 5 - program not installed
# 6 - program not configured
#
# Note that starting an already running service, stopping
# or restarting a not-running service as well as the restart
```

```
# with force-reload (in case signalling is not supported) are
# considered a success.

#define the user under which exist will run,
#or use RUNASIS to run as the current user
EXIST_USER=${EXIST_USER:-"exist"}

JAVA_OPTS="-Xms128m -Xmx768m \
    -Djava.endorsed.dirs=\"$EXIST_HOME/lib/endorsed\"\
      -Dexist.home=\"$EXIST_HOME\""

CMD_START="java $JAVA_OPTS -jar \"$EXIST_HOME/start.jar\"\
         standalone"

# eXist shutdown
# additional parameters
# -u username
# -p password
# -l identification of instance
CMD_STOP="java $JAVA_OPTS -jar \"$EXIST_HOME/start.jar\"\
        shutdown -u admin -p password -l \"$EXIST_INSTANCE\""

if [ "$EXIST_USER" = "RUNASIS" ]; then
  SUBIT=""
else
  SUBIT="su - $EXIST_USER -c "
fi

# check home directory
if [ ! -d "$EXIST_HOME" ]; then
  echo EXIST_HOME does not exist as a valid directory : $EXIST_HOME
  exit 1
fi

case "$1" in
start)
    echo -n "Starting eXist database: "
    cd $EXIST_HOME/bin
    if [ -z "$SUBIT" ]; then
        eval $CMD_START >${EXIST_CONSOLE} 2>&1 &
    else
        $SUBIT "$CMD_START >${EXIST_CONSOLE} 2>&1 &"
    fi

    # Remember status and be verbose
    rc_status -v
    ;;
stop)
    echo -n "Shutting down eXist database: "
    if [ -z "$SUBIT" ]; then
        $CMD_STOP
    else
        $SUBIT "$CMD_STOP"
    fi
```

```
    # Remember status and be verbose
    rc_status -v
    ;;
restart)
    $0 stop
    $0 start

    # Remember status and be quiet
    rc_status
    ;;
*)
    echo "usage: $0 (start|stop|restart|help)"
esac
```

B.3 Konfigurační soubory frameworku Apache Cocoon

Během kompilace byla použita následující konfigurace:

Zdrojový kód B.2: Konfigurační soubor `local.build.properties`

```
#  Licensed to the Apache Software Foundation (ASF) under one or more
#  contributor license agreements.  See the NOTICE file distributed with
#  this work for additional information regarding copyright ownership.
#  The ASF licenses this file to You under the Apache License, Version 2.0
#  (the "License"); you may not use this file except in compliance with
#  the License.  You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
#  Unless required by applicable law or agreed to in writing, software
#  distributed under the License is distributed on an "AS IS" BASIS,
#  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#  See the License for the specific language governing permissions and
#  limitations under the License.
#
#-------------------------------------------
#  Cocoon Build Properties
#-------------------------------------------

# NOTE: don't modify this file directly but copy the properties you need
# to modify over to a file named 'local.build.properties' and modify that.
# The build system will override these properties with the ones in the
# 'local.build.properties' file.

# ---- Webapp ------------------------------------------------------------------

exclude.webapp.samples=true
exclude.webapp.test-suite=true

# ---- Build Exclusions --------------------------------------------------------

exclude.deprecated=true
exclude.javadocs=true
# Include Java source code into the binary jar files
#include.sources-in-jars=true
# Include Java source code into separate, source only jar files
#include.sources-jars=true

# ---- Configuration -----------------------------------------------------------

#include.driver.oracle=true
#include.driver.postgre=true
#include.driver.odbc=true
config.allow-reloads=true
config.enable-uploads=true
#config.enable-instrumentation=true

# ---- Validation --------------------------------------------------------------
```

```
#exclude.validate.config=true
#exclude.validate.jars=true

# ---- Anteater ------------------------------------------------------------

#anteater.home = /default-from-build.properties/anteater-0.9.16
#anteater.target.host = localhost
#anteater.target.port = 8888
#anteater.target.base.path = /
#anteater.option.haltonerror = true

# disable some long-running tests by default
# anteater.test.bug26186InternalRequestMemoryLeak.enabled = true

# ---- htmlUnit ------------------------------------------------------------

# htmlunit can be downloaded from http://htmlunit.sourceforge.net/
# Tests currently require htmlunit 1.13, please change this
# comment and the default value below if moving to another version

#htmlunit.home = /default-from-build.properties/htmlunit-1.13
#htmlunit.test.baseurl=http://localhost:8888/

# for serious leak testing increase iteration count to 10000
#htmlunit.test.Bug26186InternalRequestMemoryLeak.iterations=1

# This can be changed to run only a subset of the tests
# The mask is used in ant <fileset>/<include> elements
htmlunit.test.include=**/*TestCase.class

# ---- JUnit ---------------------------------------------------------------

#junit.test.debugport=8000
#junit.testcase=org.apache.cocoon.util.test.NetUtilsTestCase
#junit.test.loglevel=0

# ---- IDE -----------------------------------------------------------------

#ide.eclipse.outputdir=${build.root}/eclipse/classes
#ide.eclipse.export.libs=false

# ---- Build ---------------------------------------------------------------

build.root=build
build=${build.root}/${name}
build.dest=${build}/classes
build.mocks=${build}/mocks
build.test=${build}/test
build.test.output=${build.test}/output
build.test.report=${build.test}/report
build.test.htmlunit=${build.test}/htmlunit
build.test.htmlunit.output=${build.test.htmlunit}/output
build.test.htmlunit.report=${build.test.htmlunit}/report
build.javadocs=${build}/javadocs
```

```
build.context=${build}/documentation
build.blocks=${build}/blocks
build.deprecated=${build}/deprecated
build.samples=${build}/samples
build.temp=${build}/temp
build.mounttable=../../mount-table.xml

# ---- Webapp Build Properties ----------------------------------------------

build.webapp=${build.root}/webapp
build.webapp.webinf=${build.webapp}/WEB-INF
build.webapp.classes=${build.webapp.webinf}/classes
build.webapp.lib=${build.webapp.webinf}/lib
build.webapp.samples=${build.webapp}/samples
build.webapp.test-suite=${build.webapp}/test-suite
build.webapp.loglevel=INFO
build.webapp.logappend=false
# Change the sample's hsqldb server port to run multiple Cocoons on a single
# machine
#build.webapp.hsqldb-server-port=9003
build.war=${build}/${name}.war

# ---- Standalone-demo Build Properties -------------------------------------

build.standalone.demo=${build.root}/standalone-demo

# ---- Compilation ----------------------------------------------------------

compiler=modern
compiler.debug=on
compiler.optimize=on
compiler.deprecation=off
compiler.nowarn=on
source.vm=1.5

# ---- System Properties ----------------------------------------------------

# WARNING: you shouldn't need to modify anything below here since there is a
# very high change of breaking the build system. Do it only if you know what
# you're doing.

packages=org.apache

# Project descriptor
gump.descriptor=gump.xml

# Directory Layout
src=src
java=${src}/java
mocks=${src}/mocks
test=${src}/test
resources=${src}/resources
resources.styles=${resources}/styles
resources.logos=${resources}/logos
```

```
resources.javadoc=${resources}/javadoc
blocks=${src}${file.separator}blocks
samples=${src}/samples
webapp=${src}/webapp
webapp.samples=${webapp}/samples
webapp.test-suite=${webapp}/test-suite
customconf=${src}/confpatch

# Deprecated Stuff
deprecated=${src}/deprecated
deprecated.src=${deprecated}/java
deprecated.conf=${deprecated}/conf

# Tools
tools=tools
tools.lib=${tools}/lib
tools.src=${tools}/src
tools.tasks.src=${tools.src}/anttasks
tools.tasks.dest=${tools}/anttasks
tools.loader.src=${tools.src}/loader
tools.loader.dest=${tools}/loader
tools.jetty=${tools}/jetty

# Libraries
lib=lib
lib.core=${lib}/core
lib.endorsed=${lib}/endorsed
lib.optional=${lib}/optional
lib.local=${lib}/local

# Distribution Directories
dist.root=dist
dist=${dist.root}/${name}-${version}
dist.name=${name}-${version}
dist.target=${dist.root}

# Site Directory
site=../cocoon-site/site/2.1

# Legal
legal=legal
```

Zdrojový kód B.3: Konfigurační soubor `local.blocks.properties`

```
#  Licensed to the Apache Software Foundation (ASF) under one or more
#  contributor license agreements.  See the NOTICE file distributed with
#  this work for additional information regarding copyright ownership.
#  The ASF licenses this file to You under the Apache License, Version 2.0
#  (the "License"); you may not use this file except in compliance with
#  the License.  You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
#  Unless required by applicable law or agreed to in writing, software
```

```
#  distributed under the License is distributed on an "AS IS" BASIS,
#  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#  See the License for the specific language governing permissions and
#  limitations under the License.
#

#-------------------------------------------------------------------------------#
# ***** DO  NOT edit blocks.properties yourself! ********                       #
# This file is generated from gump.xml - to keep it in sync when that file is  #
# modified, use the generate-blocks.properties build target.                    #
#-------------------------------------------------------------------------------#

#-------------------------------------------------------------------------------#
#                              Cocoon Blocks                                    #
#-------------------------------------------------------------------------------#

# Remove blocks from your cocoon distribution by setting the corresponding
# include property to true or false. The blocks are included by default, i.e. if
# no property was set.

# NOTE: Don't modify this file directly but make a copy named
# 'local.blocks.properties' and modify that. The build system will first load
# 'local.blocks.properties' and properties are immutable in Ant.

# For most cases it is enough that you exclude all blocks and include only those
# few you want, example:
# exclude.all.blocks=true
# include.block.forms=true
# include.block.template=true

# The opposite is also allowed:
# include.all.blocks=true
# exclude.block.scratchpad=true

# If there is a conflict on the same level of granularity:
# include.block.template=true vs. exclude.block.template=true,
# include.all.blocks=true vs. exclude.all.blocks=true
# it is always resolved in favour of include.* properties.

# NOTE: "[dependency]" indicates blocks that are required by other blocks.
# Disabling batik, for example, will result in a RuntimeException when using
# fop. Dependencies only needed for the block's samples are marked explicitely.
# This latter information was introduced only short time ago, so do not expect
# it to be complete.

# NOTE: (to Cocoon committers): blocks.properties is generated from gump.xml
# using "build generate-blocks.properties". Any changes to blocks definitions
# must be made in gump.xml, not here.

# All blocks ------------------------------------------------------------------

# Use this property to exclude all blocks at once
exclude.all.blocks=true
```

```
# Use this property to include all blocks at once
# include.all.blocks=true


# Stable blocks ------------------------------------------------------------

# Stable blocks are those that can be considered ready for production and
# will contain components and API that will remain stable and where
# developers are committed to back compatibility. In short, stuff that you
# can depend on.

#-----[dependency]: "authentication-fw" depends on "session-fw".
#-----[dependency]: "authentication-fw" is needed by "portal", "portal-fw".
#include.block.authentication-fw=false
#-----[dependency]: "batik" is needed by "fop", "tour".
#include.block.batik=false
#include.block.bsf=false
include.block.chaperon=true
#-----[dependency]: "databases" depends on "xsp".
#-----[dependency]: "databases" is needed by "hsqldb", "jms", "ojb", "petstore",
#       "repository", "xmldb".
include.block.databases=true
#-----[dependency]: "fop" depends on "batik", "xsp" (for samples).
#-----[dependency]: "fop" is needed by "tour".
#include.block.fop=false
#-----[dependency]: "forms" depends on "ajax", "template" (for samples).
#-----[dependency]: "forms" is needed by "apples", "javaflow", "ojb", "petstore",
#       "portal", "querybean", "tour".
include.block.forms=true
#-----[dependency]: "hsqldb" depends on "databases".
#-----[dependency]: "hsqldb" is needed by "jms", "ojb", "petstore".
#include.block.hsqldb=false
#include.block.html=false
#-----[dependency]: "itext" depends on "xsp" (for samples).
#include.block.itext=false
#include.block.jfor=false
#include.block.jsp=false
#-----[dependency]: "linkrewriter" depends on "xsp".
#include.block.linkrewriter=false
#-----[dependency]: "lucene" is needed by "querybean".
#include.block.lucene=false
#include.block.midi=false
#include.block.naming=false
#-----[dependency]: "ojb" depends on "databases" (for samples),
#   "forms" (for samples), "hsqldb" (for samples), "xsp" (for samples).
#-----[dependency]: "ojb" is needed by "javaflow", "portal", "querybean".
#include.block.ojb=false
#include.block.paranoid=false
#include.block.poi=false
#-----[dependency]: "portal" depends on "auth", "authentication-fw",
#   "cron", "forms", "ojb", "session-fw".
#-----[dependency]: "portal" is needed by "faces".
#include.block.portal=false
#-----[dependency]: "profiler" depends on "auth".
```

```
#include.block.profiler=false
#-----[dependency]: "python" depends on "xsp".
#include.block.python=false
#-----[dependency]: "session-fw" depends on "xsp".
#-----[dependency]: "session-fw" is needed by "authentication-fw", "portal",
#   "portal-fw".
include.block.session-fw=true
#-----[dependency]: "velocity" is needed by "petstore".
#include.block.velocity=false
#include.block.web3=false
#-----[dependency]: "xmldb" depends on "databases".
include.block.xmldb=true
#-----[dependency]: "xsp" is needed by "axis", "databases", "fop", "itext",
#   "linkrewriter", "ojb", "python", "session-fw", "woody".
include.block.xsp=true

# Deprecated blocks ------------------------------------------------------------

# Although some of these blocks may have been stable, they are now deprecated
# in favour of other blocks and therefore are excluded by default from the build.
# For including one of them you have to set the exclude property into comment in
# blocks.properties.

include.block.php=false
#-----[dependency]: "portal-fw" depends on "authentication-fw", "session-fw".
include.block.portal-fw=false
include.block.swf=false
#-----[dependency]: "woody" depends on "xsp" (for samples).
include.block.woody=false

# Unstable blocks --------------------------------------------------------------

# Unstable blocks are currently under development and do not guarantee that the
# contracts they expose (API, xml schema, properties, behavior) will remain
# constant in time. Developers are not committed to back-compatibility just yet.
# This doesn't necessarily mean the blocks implementation is unstable or
# the code can't be trusted for production, but use with care and watch
# its development as things might change over time before they are marked
# stable.

#-----[dependency]: "ajax" depends on "template" (for samples).
#-----[dependency]: "ajax" is needed by "forms".
include.block.ajax=true
#-----[dependency]: "apples" depends on "forms" (for samples).
#include.block.apples=false
#-----[dependency]: "asciiart" is needed by "mail".
#include.block.asciiart=false
#-----[dependency]: "auth" is needed by "portal", "profiler".
#include.block.auth=false
#-----[dependency]: "axis" depends on "xsp" (for samples).
#include.block.axis=false
#-----[dependency]: "captcha" depends on "template" (for samples).
#include.block.captcha=false
#-----[dependency]: "cron" is needed by "jms", "portal".
```

```
#include.block.cron=false
#include.block.deli=false
#-----[dependency]: "eventcache" depends on "jms".
#-----[dependency]: "eventcache" is needed by "repository", "webdav".
#include.block.eventcache=false
#-----[dependency]: "faces" depends on "portal", "taglib".
#include.block.faces=false
#include.block.imageop=false
#-----[dependency]: "javaflow" depends on "forms", "ojb".
#include.block.javaflow=false
include.block.jcr=false
#-----[dependency]: "jms" depends on "cron", "databases" (for samples), "hsqldb".
#-----[dependency]: "jms" is needed by "eventcache", "slide".
#include.block.jms=false
#include.block.linotype=false
#-----[dependency]: "mail" depends on "asciiart" (for samples).
#include.block.mail=false
#-----[dependency]: "petstore" depends on "databases", "forms", "hsqldb",
#    "velocity".
#include.block.petstore=false
#include.block.proxy=false
#include.block.qdox=false
#-----[dependency]: "querybean" depends on "forms" (for samples), "lucene",
#   "ojb".
#include.block.querybean=false
#-----[dependency]: "repository" depends on "databases", "eventcache".
#-----[dependency]: "repository" is needed by "slide", "webdav".
#include.block.repository=false
#include.block.serializers=false
#-----[dependency]: "slide" depends on "jms", "repository".
#include.block.slide=false
#-----[dependency]: "slop" is needed by "tour".
#include.block.slop=false
#include.block.stx=false
#-----[dependency]: "taglib" is needed by "faces".
#include.block.taglib=false
#-----[dependency]: "template" is needed by "ajax", "captcha", "forms".
include.block.template=true
#-----[dependency]: "tour" depends on "batik", "fop", "forms", "slop".
#include.block.tour=false
include.block.validation=true
#-----[dependency]: "webdav" depends on "eventcache", "repository".
#include.block.webdav=false
include.block.xsltal=true
```

B.4 Webový popisovač aplikace NXMLDB

Pro detailní popis funkce inicializačních parametrů a další dostupné parametry je vhodné konzultovat výchozí popisovač vygenerovaný při kompilaci frameworku Cocoon.

Zdrojový kód B.4: Webový popisovač aplikace NXMLDB

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>

  <!-- Servlet Configuration ======================================== -->
  <servlet>
    <servlet-name>Cocoon</servlet-name>
    <display-name>Cocoon</display-name>
    <description>Cocoon</description>
    <servlet-class>org.apache.cocoon.servlet.CocoonServlet</servlet-class>

    <init-param>
      <param-name>init-classloader</param-name>
      <param-value>false</param-value>
    </init-param>

    <init-param>
      <param-name>configurations</param-name>
      <param-value>/WEB-INF/cocoon.xconf</param-value>
    </init-param>

    <init-param>
      <param-name>logkit-config</param-name>
      <param-value>/WEB-INF/logkit.xconf</param-value>
    </init-param>

    <init-param>
      <param-name>servlet-logger</param-name>
      <param-value>access</param-value>
    </init-param>

    <init-param>
      <param-name>cocoon-logger</param-name>
      <param-value>core</param-value>
    </init-param>

    <init-param>
      <param-name>log-level</param-name>
      <param-value>WARN</param-value>
    </init-param>

    <init-param>
      <param-name>forbidden-deprecation-level</param-name>
      <param-value>ERROR</param-value>
    </init-param>
```

```
  <init-param>
    <param-name>manage-exceptions</param-name>
    <param-value>true</param-value>
  </init-param>

  <init-param>
    <param-name>enable-instrumentation</param-name>
    <param-value>false</param-value>
  </init-param>

  <!--
    Set encoding used by the container. If not set the ISO-8859-1 encoding
    will be assumed.
    Since the servlet specification requires that the ISO-8859-1 encoding
    is used (by default), you should never change this value unless
    you have a buggy servlet container.
  -->
  <init-param>
    <param-name>container-encoding</param-name>
    <param-value>ISO-8859-1</param-value>
  </init-param>

  <!--
    Set form encoding. This will be the character set used to decode request
    parameters. If not set the ISO-8859-1 encoding will be assumed.
  -->
  <init-param>
    <param-name>form-encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>

  <!--
    This parameter allows you to startup Cocoon2 immediately after startup
    of your servlet engine.
  -->
  <load-on-startup>1</load-on-startup>
</servlet>

<filter>
  <filter-name>SetCharacterEncoding</filter-name>
  <filter-class>org.apache.servlet.SetCharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>ISO-8859-1</param-value>
  </init-param>
</filter>


<filter-mapping>
  <filter-name>SetCharacterEncoding</filter-name>
  <servlet-name>Cocoon</servlet-name>
</filter-mapping>
```

```
<!-- URL space mappings ============================================= -->

<!--
  Cocoon handles all the URL space assigned to the webapp using its sitemap.
  It is recommended to leave it unchanged. Under some circumstances though
  (like integration with proprietary webapps or servlets) you might have
  to change this parameter.
-->
<servlet-mapping>
  <servlet-name>Cocoon</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

<!--
  Some servlet engines (Tomcat) have defaults which are not overriden
  by '/' mapping, but must be overriden explicitly.
-->
<servlet-mapping>
  <servlet-name>Cocoon</servlet-name>
  <url-pattern>*.jsp</url-pattern>
</servlet-mapping>
<!--
  Some servlet engines (WebLogic) have defaults which are not overriden
  by '/' mapping, but must be overriden explicitly.
-->
<servlet-mapping>
  <servlet-name>Cocoon</servlet-name>
  <url-pattern>*.html</url-pattern>
</servlet-mapping>

<!-- various MIME type mappings ===================================== -->

<mime-mapping>
  <extension>css</extension>
  <mime-type>text/css</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>xml</extension>
  <mime-type>text/xml</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>xsl</extension>
  <mime-type>text/xml</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>xconf</extension>
  <mime-type>text/xml</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>xmap</extension>
```

```
    <mime-type>text/xml</mime-type>
  </mime-mapping>

  <mime-mapping>
    <extension>ent</extension>
    <mime-type>text/plain</mime-type>
  </mime-mapping>

  <mime-mapping>
    <extension>grm</extension>
    <mime-type>text/plain</mime-type>
  </mime-mapping>

</web-app>
```

B.5 Webový popisovač aplikace ORDB

Zdrojový kód B.5: Webový popisovač aplikace ORDB

```
<?xml version="1.0" encoding="UTF-8" ?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
         version="2.4">

    <display-name>ORDB</display-name>
    <description>Object-Relational DB concept</description>

    <filter>
        <display-name>Stripes Filter</display-name>
        <filter-name>StripesFilter</filter-name>
        <filter-class>
        net.sourceforge.stripes.controller.StripesFilter
    </filter-class>
        <init-param>
            <param-name>Extension.Packages</param-name>
            <param-value>
                com.samaxes.stripes.integration.ejb
            </param-value>
        </init-param>
        <init-param>
            <param-name>ActionResolver.Packages</param-name>
            <param-value>nxmldb.actionbean</param-value>
        </init-param>
    <!--
        <init-param>
            <param-name>ExceptionHandler.Class</param-name>
            <param-value>hda.control.ExceptionHandler</param-value>
        </init-param>
    -->
        <init-param>
            <param-name>LocalePicker.Locales</param-name>
            <param-value>cs_CZ:UTF-8,en_US:UTF-8</param-value>
        </init-param>
    </filter>

    <filter-mapping>
        <filter-name>StripesFilter</filter-name>
        <url-pattern>*.jsp</url-pattern>
        <dispatcher>REQUEST</dispatcher>
    </filter-mapping>

    <filter-mapping>
        <filter-name>StripesFilter</filter-name>
        <servlet-name>StripesDispatcher</servlet-name>
        <dispatcher>REQUEST</dispatcher>
    </filter-mapping>
```

```
    <servlet>
        <servlet-name>StripesDispatcher</servlet-name>
        <servlet-class>
        net.sourceforge.stripes.controller.DispatcherServlet
    </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>StripesDispatcher</servlet-name>
        <url-pattern>/art/*</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>StripesDispatcher</servlet-name>
        <url-pattern>/raw/*</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>StripesDispatcher</servlet-name>
        <url-pattern>*.html</url-pattern>
    </servlet-mapping>

    <context-param>
        <param-name>javax.servlet.jsp.jstl.fmt.localizationContext</param-name>
        <param-value>/StripesResources</param-value>
    </context-param>
</web-app>
```

B.6 Zdrojové kódy pro vyhledávání v databázích

Zdrojové kódy jsou vždy uvedeny nejprve v jazyce SQL, dialektu PostgreSQL, poté následuje kód v jazyce XQuery. Poslední uvedené výsledky rovnou upravují pro potřeby aplikace. V první ukázce je vidět použití XQuery modulu *Wikiparser*.

Zdrojový kód B.6: Vyhledání článku Brno v databázi

```
SELECT * FROM Page WHERE title
    ~* E'^([^[:space:]]+[[:space:]\(\)-]+)*Brno([^[:alnum:]]+[^[:space:]]+)*$';

SELECT * FROM Page, Revision, PageRevision WHERE Page.id=2813
    AND Page.id = PageRevision.page
    AND PageRevision.revision = Revision.id
-- zde se uvazovala pouze jedna revize, nebo reseni aktualnosti revize
-- mimo databazi


xquery version "1.0" encoding "UTF-8";
declare namespace mw="http://www.mediawiki.org/xml/export-0.3/";

declare namespace test="http://www.kapy.info/nxmldb/util";

declare function test:max-string ( $strings as xs:anyAtomicType* )
    as xs:string? {
   max(for $string in $strings return string($string))
};

let $q := 'Brno',
    $articles :=  collection("/db/nxmldb")//mw:page[near(mw:title,$q)],
    $article := $articles[upper-case(mw:title) eq upper-case($q)],
    $count-narrow := count($article),
    $count := count($articles),
    $newest := $article/mw:revision[test:max-string(
            $article/mw:revision/mw:timestamp)]
return
    <query>
    { (: no such result :)
    if($count-narrow = 0 and $count = 0) then
        <no-results>
            <query-string>{$q}</query-string>
        </no-results>
    (: multiple articles, but no exact match :)
    else if ($count-narrow = 0 and $count > 0) then
        <multiple-results>
            <query-string>{$q}</query-string>
            {
                for $result in $articles/mw:title
                return
                    <result-match>{$result/text()}</result-match>
            }
        </multiple-results>
    (: single or multiple revisions :)
    else
        <article>
```

```
            { if($count-narrow > 1) then <multiple-revisions/> else ()}
            <query-string>{$q}</query-string>
            <title>{$article/mw:title/text()}</title>
            {wikiparser:parse($article/mw:revision/mw:text/text(),
                    "http://localhost:8080/xmldb/")}
        </article>
    }
    </query>
```

Zdrojový kód B.7: Počet článků v databázi

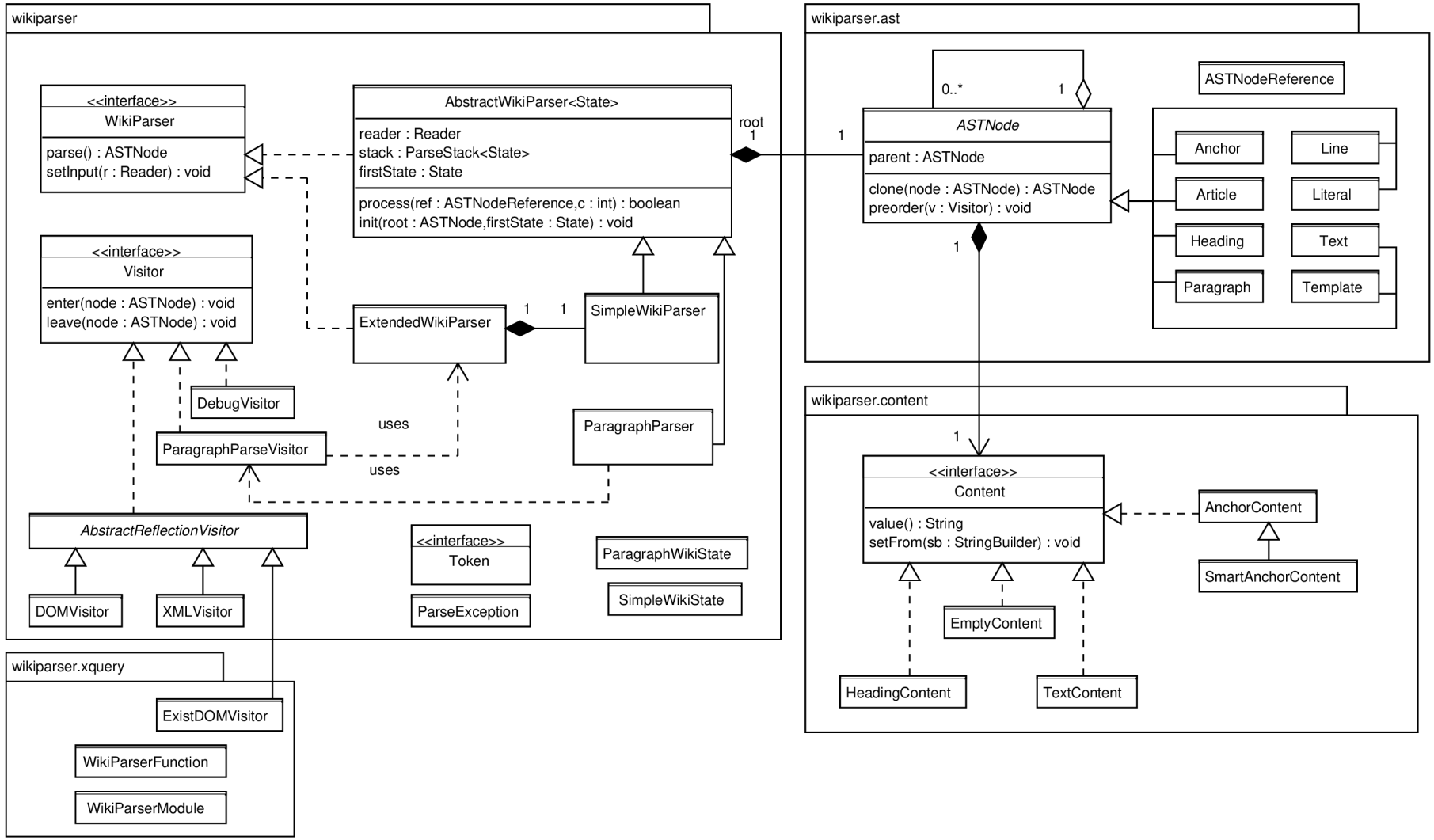
```
SELECT COUNT(*) FROM Page;

xquery version "1.0" encoding "UTF-8";
declare namespace mw="http://www.mediawiki.org/xml/export-0.3/";

let $pages := count(collection("/db/nxmldb")//mw:page)
return
    <query>
        <total-articles>{$pages}</total-articles>
    </query>
```

Zdrojový kód B.8: Počet příspěvších autorů

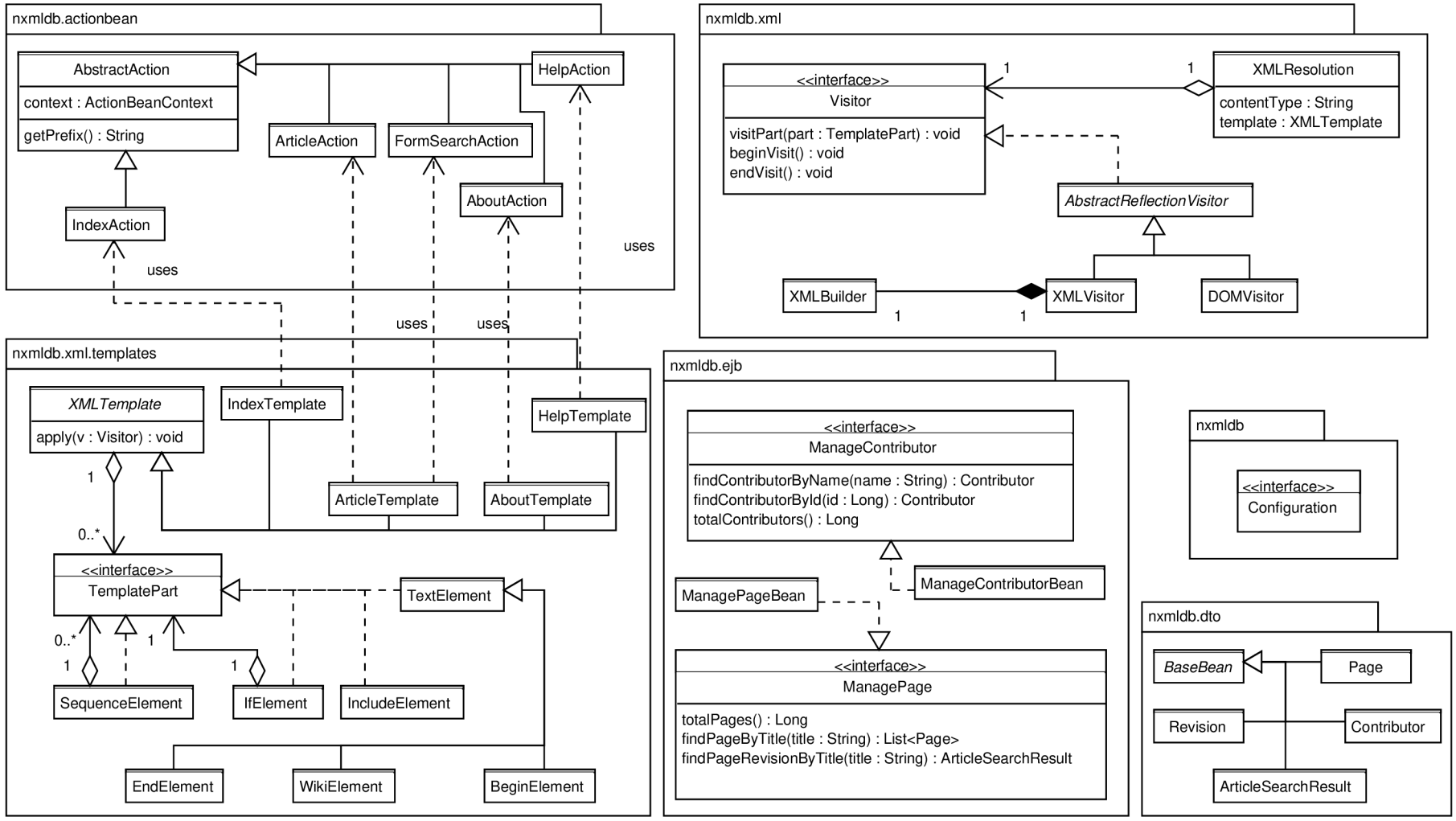
```
SELECT COUNT(username) FROM Page;

xquery version "1.0" encoding "UTF-8";
declare namespace mw="http://www.mediawiki.org/xml/export-0.3/";

let $contributors := count(
        distinct-values(
            collection("/db/nxmldb")//mw:contributor/mw:username))
return
    <query>
        <total-contributors>{$contributors}</total-contributors>
    </query>
```

Dodatek C

Diagramy tříd



Obrázek C.1: Diagram tříd pro komponentu Wikiparser



Obrázek C.2: Diagram tříd pro aplikaci ORDB