MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Implementace OAI-PMH pro český WebArchiv

BAKALÁŘSKÁ PRÁCE

Martin Bella

Brno, 2008

Prehlásenie

Prehlasujem, že táto bakalářská práce je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Vedúci práce: RNDr. Miroslav Bartošek, CSc.

Zhrnutie

Cieľom práce bolo navrhnúť a implementovať rozhranie data-providera protokolu OAI-PMH pre český WebArchiv. Data-provider by sa mal stať súčasného systému WA-CZ. Práca oboznamuje s princípmi protokolu OAI-PMH, navrhuje rôzne modifikácie dátového modelu súčasného systému a popisuje problémy súvisiace s prístupom k dátam v prípade tak veľkého archívu, akým je WebArchiv. Práca tiež predstavuje voľne dostupný softvér na vybudovanie repozitára a ukazuje možnú implementáciu vlastnej aplikačnej vrstvy.

Kľúčové slová

OAI, metadáta, protokol, sťahovanie metadát, data-provider, WebArchiv, Open Archives Initiative, OAI-PMH, Dublin Core

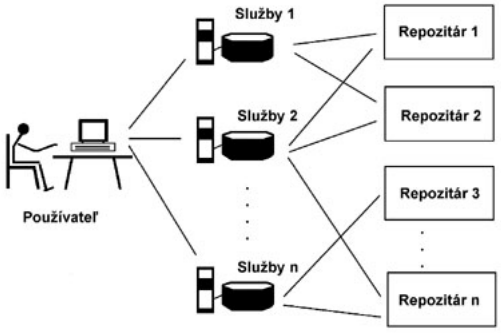
Obsah

Kapitola 1

Úvod

Nárast počtu rozptýlených informačných zdrojov v posledných rokoch zvýšil požiadavky na efektivitu výmeny, tvorby a využívania informácií. Od informačných systémov sa čoraz častejšie požaduje schopnosť spolupráce podľa vopred dohodnutých pravidiel a postupov. Tejto vlastnosti sa tiež hovorí kompatibilita a môžeme ju chápať v dvoch rozmeroch[1]. V minulosti sa jednalo hlavne o priamu kompatibilitu medzi systémami, ktorá si vyžadovala jednotnú technologickú základňu. Moderný rozmer, v oblasti digitálnych knižníc označovaný ako interoperabilita, hovorí o schopnosti technologicky odlišných systémov vymieňať si alebo sprístupňovať svoje dáta jednotným jazykom. Jej cieľom je vybudovanie jednoliateho bloku služieb z rozličných komponentov umiestnených v rôznych organizáciách.

Medzi významné nástroje na dosiahnutie interoperability už niekoľko rokov patrí OAI-PMH (Open Archives Initiative – Protocol for Metadata Harvesting)[2]. Názov napovedá, že je to sieťový protokol pracujúci nad protokolom HTTP a používa sa na sťahovanie metadát z voľne dostupných archívov. Oblasti nasadenia sú tam, kde sa často publikuje alebo kde vzniká nejaký „sklad“ vedeckých alebo odborných článkov, ktoré sú určené na ďalšie šírenie pomocou metadát v rámci komunity. Implementácie teda nájdeme hlavne v knižniciach, múzeách, rôznych archívoch, univerzitách a iných vývojových centrách. V koncepte sa objavujú dva typy účastníkov – poskytovatelia dát a poskytovatelia služieb. Poskytovatelia dát ponúkajú cez jednoduché rozhrania metadáta, ktoré poskytovatelia služieb môžu ďalej obohacovať a vybudovať nad nimi službu s pridanou hodnotou.



Obrázok 1.1: Funkčná schéma OAI (prameň: [1])

Ako už bolo naznačené, základnou činnosťou je metadátová žatva[1], pri ktorej má poskytovateľ služieb vykonávajúci zber možnosť vybrať si záznamy, o ktoré má záujem. Server vystavujúci dáta rozoznáva šesť základných príkazov – *GetRecord*, *Identify*, *ListIdentifiers*, *ListMetadataFormats*, *ListRecords* a *ListSets*. Koncept nehovorí nič o vnútornej implementácii dátového úložiska, jasne však definuje vonkajšie správanie rozhrania. Princíp interoperability podporuje tiež jednotný formát metadátových záznamov. Povinne podporovaný je nekvalifikovaný Dublin Core, ktorý je veľmi jednoduchý a bol navrhnutý ako spoločný metadátový štandard pre rozdielne skupiny užívateľov[3]. Poskytovateľ dát však môže voliteľne svoje metadáta poskytovať aj v ďalších formátoch.

Podporu protokolu OAI-PMH môžeme nájsť vo veľkých archívoch vedeckých prác (arxiv.org[2]), vedeckých inštitúciách (CERN[3], NASA[4]), univerzitách (MIT[5]) i knižniciach (Library of Congress[6]). Na strane poskytovateľov služieb sa najčastejšie vyskytujú rôzne typy súborných katalógov alebo agregačné služby (ARC[7]), ktoré zozbierané dáta opäť sprístupňujú pomocou OAI-PMH. Zaujímavou aplikáciou je služba DP9[8], ktorá zozbieraným metadátam pridáva permanentný odkaz, čím umožňuje prehľadávanie a indexáciu inak nedostupných dokumentov aj webovým robotom. Protokol by si určite našiel svoje miesto aj v českom celoštátnom registri diplomových a dizertačných prác eVŠKP[9] alebo slovenskom ekvivalente ETD[10] či projekte Diplomovka[11].

Myšlienka vytvorenia OAI data-providera nad archívom webu nie je vo svete obvyklá a v súčasnej dobe podobné rozhranie alebo rozhranie nad archívom s porovnateľnou veľkosťou ani nie je v prevádzke. Táto práca dopĺňa o podporu protokolu OAI-PMH systém WA-CZ, ktorý pre český WebArchiv navrhol a implementoval Lukáš Matějka vo svojej diplomovej práci[4]. Druhá kapitola rozoberá princípy protokolu podrobnejšie, kapitola č. 3 sa zamýšľa nad efektívnym fungovaním dátového úložiska, popisuje rôzne modifikácie súčasného dátového modelu a rozoberá ich problémy v súvislosti s OAI. Posledná kapitola navrhuje a implementuje aplikačnú vrstvu systému.

1. z anglického slova harvest. Bežné sú tiež výrazy zber a sťahovanie.
2. http://www.arxiv.org
3. http://cdsweb.cern.ch
4. http://naca.larc.nasa.gov/search.jsp
5. http://dspace.mit.edu
6. http://memory.loc.gov/ammem/index.html
7. http://arc.cs.odu.edu
8. http://dlib.cs.odu.edu/dp9/
9. http://www.evskp.cz
10. http//www.etd.sk
11. http://diplomovka.sme.sk

Kapitola 2

Popis protokolu

Kapitola popisuje druhú verziu protokolu OAI-PMH.

2.1 Základné pojmy

Protokol narába s týmito základnými pojmami:

- *harvester* – zberač, alebo tiež poskytovateľ služieb, ktorý zbiera metadátové záznamy od poskytovateľov dát,
- *repository* – server (repozitár) spravovaný poskytovateľom záznamov, ktorý vystavuje metadáta harvesterom a ktorý dokáže spracovať šesť dotazov protokolu OAI-PMH,
- *resource* – zdroj, o ktorom metadáta sú,
- *item* – metadátový objekt uložený v databázi poskytovateľa,
- *record* – metadátový záznam v konkrétnom formáte,
- *identifier* – unikátny kľúč metadátového objektu v databáze,
- *set* – nástroj pre zoskupovanie objektov.

Je dôležité uvedomiť si rozdiel medzi pojmami resource, item a record. Samotný zdroj (*resource*) nie je pre protokol podstatný a ani sa nezaoberá tým, či je digitálny, alebo či je uložený v databáze. Potrebné sú iba metadáta, ktoré tvoria metadátový objekt (*item*). Tento je po zaslaní požiadavky harvesterom transformovaný na požadovaný formát a hovorí sa mu záznam (*record*).

2.1.1 Jednoznačný identifikátor

Na výber a deduplikáciu metadátového záznamu sa používa jednoznačný identifikátor. Identifikátor sa viaže na metadátový objekt a všetky záznamy z neho odvodené, ktoré môžu byť k dispozícii vo viacerých formátoch, zdieľajú jeden jednoznačný identifikátor.

Organizácie si môžu vyvinúť vlastný systém označovania dokumentov, formát identifikátorov však musí vyhovovať špecifikácii URI[1]. Môžu tiež implementovať oai-identifier[5], ktorý sa skladá z troch častí oddelených dvojbodkami. Prvou časťou je vždy reťazec *oai* vyjadrujúci, že sa jedná o oai identifikátor. Druhou je doménové meno servera hostiaceho repozitár, čo zaručuje celosvetovú jednoznačnosť a treťou jednoznačné označenie v databáze poskytovateľa.

Jednoznačný identifikátor má dva druhy použitia:

- identifikátory sú vrátené príkazmi *ListIdentifiers* a *ListRecords*,
- identifikátor slúži ako parameter príkazu *GetRecord* na získanie príslušného záznamu.

2.1.2 Záznam

Metadátový záznam je séria bajtov zakódovaná vo formáte XML a je určená jednoznačným identifikátorom metadátového objektu, metadátovým formátom a časovou známkou. Záznam obsahuje nasledujúce elementy:

- *header* – hlavička skladajúca sa z
  - elementu *identifier*,
  - elementu *datestamp*,
  - nula alebo niekoľko elementov *setSpec* označujúce príslušnosť k množinám,
  - voliteľného elementu *status* s hodnotou *deleted* označujúceho zmazaný záznam,
- *metadata* – metadátový záznam v požadovanom formáte. Implementácia môže podporovať viacero formátov, povinne však v XML zakódovaný nekvalifikovaný Dublin Core podľa schémy *oai_dc*[2],
- *about* – voliteľný a opakovateľný kontajner XML dát. Organizácie si môžu túto časť prispôsobiť podľa vlastných potrieb, musia však uviesť XML schému pre formálnu kontrolu. Dve základné využitia sú na deklaráciu autorských práv[6] a na uvedenie pôvodu záznamu[7]. To môže byť užitočné najmä v prípade, že server metadáta získal od iného poskytovateľa.

Poskytovateľ musí deklarovať jeden z troch možných spôsobov zaobchádzania so zmazanými záznamami:

- *no* – nie sú udržiavané žiadne informácie o zmazaných záznamoch,
- *persistent* – poskytovateľ udržiava trvalú informáciu o zmazaných záznamoch,

1. dostupná na http://www.ietf.org/rfc/rfc2396.txt?number=2396
2. dostupná na http://www.openarchives.org/OAI/2.0/oai_dc.xsd

- *transient* – poskytovateľ drží informácie o zmazaných záznamoch, ale negarantuje, po akú dobu.

Nevýhodou neudržiavania informácií o zmazaných záznamoch je, že v prípade inkrementálneho zberu sa harvester nemá možnosť dozvedieť, že záznam sa v databáze servera už nenachádza. Ak server tieto záznamy udržiava, časová známka musí byť zhodná s časom zmazania záznamu. Odpoveď na príkazy *GetRecord* a *ListRecords* musí obsahovať hlavičku s elementom *status* s hodnotou *deleted*. Záznam nesmie obsahovať časti *metadata* a *about*. V prípade selektívneho zberu musia byť do odpovede zahrnuté aj zmazané záznamy.

2.1.3 Množiny

Množina je jednoduchý spôsob zoskupovania záznamov pre účely selektívneho zberu. Implementácia nie je povinná, organizácia môže byť jednoduchá alebo hierarchická. Poskytovateľ, ktorý deklaruje podporu množín, musí uviesť príslušnosť záznamu k množine v odpovediach na príkazy *ListIdentifiers*, *ListRecords* a *GetRecord*.

Každá množina je definovaná identifikátorom *setSpec*, krátkym názvom *setName* a popisom *setDescription*. V prípade hierarchie je *setSpec* dvojbodkami oddelený zoznam podmnožín s koreňovou množinou na začiatku. Každý element zo zoznamu musí byť jedinečný, musí vyhovovať špecifikácii URI a nesmie obsahovať dvojbodky. Jednoduché organizácie množín majú *setSpec* neobsahujúci dvojbodky. *SetDescription* je nepovinný a opakovateľný kontajner ľubovolných dát vo formáte XML.

Metadátový objekt môže byť zaradený v jednej, niekoľkých, ale aj aj žiadnej množine. Harvester by teda nemal predpokladať, že zberom všetkých dát z každej množiny dostane všetky záznamy. Špecifikácia protokolu nedefinuje, ako majú byť dáta v množinách zoradené. Všetky množiny ani nemusia spadať do jedného hierarchického stromu. Server by mal vždy uviesť minimálny počet identifikátorov *setSpec*, aby presne definoval príslušnosť k množine. Ak harvester požiada o záznamy rodičovskej množiny, automaticky získa aj záznamy jej potomkov.

2.2 Využitie protokolu HTTP

Komunikácia medzi harvesterom a data-providerom prebieha pomocou protokolu HTTP. Typická implementácia používa webový prehliadač na jednej strane a web server so špeciálnym softvérom na spracovanie dotazov OAI-PMH na druhej strane. Harvester môže posielať požiadavky metódou GET i POST a server musí podporovať oba spôsoby.

Každý dotaz sa skladá zo základného URL repozitára a zoznamom argumentov s podobou *kľúč=hodnota* oddelených znakom *&*. Práve jeden z týchto párov musí špecifikovať príkaz protokolu OAI-PMH, inak je vrátená chyba *badVerb*. Kľúčom je slovo *verb* a hodnotou buď *Identify*, *GetRecord*, *ListIdentifiers*, *ListMetadataFormats*, *ListRecords* alebo *ListSets*. Obsah ostatných dvojíc závisí na zvolenom príkaze. Vrátený XML dokument obsahuje buď odpoveď na daný dotaz alebo chybové hlásenie.

2.3 Kódovanie odpovedí vo formáte XML

Odpoveďou na dotaz je vždy XML dokument[3] v kódovaní UTF-8 platný voči XML schéme[4], ktorá je súčasťou dokumentácie protokolu. Dokument sa musí držať nasledujúcich zásad:

1. Na začiatku je deklarácia XML dokumentu vo verzii 1.0 a kódovaní UTF-8.
2. Koreňový element má názov OAI-PMH a tri atribúty definujúce menné priestory a XML schému:
   - *xmls* – hodnotou je URI menného priestoru OAI-PMH[5],
   - *xmlns:xsi* – hodnotou je URI menného priestoru XML schémy[6],
   - *xsi:schemaLocation* – hodnotou je pár, kde prvá časť je URI menného priestoru OAI-PMH a druhou URL XML schémy pre formálnu kontrolu dokumentov.
3. Prvé dva synovské elementy každej odpovede sú:
   - *responseDate* – čas vygenerovania odpovede,
   - *request* – element špecifikujúci dotaz. Generovanie sa riadi týmito pravidlami:
     - obsah elementu musí vždy byť URL poskytovateľa bez argumentov,
     - atribútmi elementu a ich hodnotami sú prvky párov *kľúč=hodnota* dotazu,
     - v prípade vygenerovania chyby *badArgument* a *badVerb* ostáva element bez atribútov.

2.4 Časové značky

Dátum a čas dodržiava syntax normy ISO1806[7] a je udávaný v UTC. Povolené sú dve úrovne granularity — dni a sekundy. Je teda vyjadrený vo formáte RRRR-MM-DD resp. RRRR-MM-DDTHH:MM:SSZ, kde *T* oddeľuje dátumovú a časovú zložku a *Z* udáva, že ide o čas na nultom poludníku. Ak data-provider podporuje sekundovú granularitu, mal by tak uviesť v odpovedi na príkaz *Identify*.

Časové značky v tomto formáte sa musia dodržiavať rovnako ako v dotazoch, tak aj odpovediach. V dotazoch sú uvedené ako hodnoty nepovinných argumentov from a until príkazov *ListIdentifiers* a *ListRecords*. Data-provider musí v odpovediach používať najjemnejšiu podporovanú granularitu. Ak harvester žiada jemnejšiu granularitu, ako je serverom podporovaná, je vrátená chyba *badArgument*.

---

3. Príklady niektorých odpovedí sú uvedené v prílohe A
4. dosptupná na http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd
5. URI menného priestoru OAI-PMH je http://www.openarchives.org/OAI/2.0/
6. URI menného priestoru XML schémy je http://www.w3.org/2001/XMLSchema-instance
7. http://www.w3.org/TR/NOTE-datetime

2.5 Metadátové formáty

Protokol OAI-PMH sa kvôli väčšej všeobecnosti neobmedzuje iba na použitie niekoľkých vybraných formátov odpovedí. Metadáta záznamu vráteného príkazmi *GetRecord* a *ListRecords* môžu byť v akomkoľvek formáte, ktorý je označený reťazcom vyhovujúcim URI, má definovanú XML schému na testovanie platnosti a URI menného priestoru. Znamená to, že koreňový element metadátovej časti záznamu obsahuje atribút *xmlns* s hodnotou URI menného priestoru a atribút *xsi:schemaLocation* obsahujúci URL XML schémy.

Pre účely interoperability musí každá implementácia podporovať nekvalifikovaný Dublin Core, ktorého označenie je *oai_dc*. Zoznam všetkých podporovaných formátov je možné získať príkazom *ListMetadataFormats*.

2.6 Riadenie toku dát

Príkazy *ListRecords*, *ListIdentifiers* a *ListSets* vracajú zoznamy, ktoré môžu byť v niektorých prípadoch príliš dlhé a je nepraktické prenášať ich vcelku. Server preto môže vrátiť nekompletný zoznam a tzv. *resumption token*, čo je mechanizmus na získavanie ďalšej série záznamov. Zjednotenie všetkých nekompletných zoznamov tvorí úplný zoznam.

Ak server odpovie nekompletným zoznamom, posledný element koreňového elementu musí byť práve *resumption token.*, Jeho obsahom je ľubovolná krátka postupnosť bajtov, v ktorej sú všetky špeciálne znaky zakódované pomocou únikových sekvencií. V prípade poslednej série záznamov je tento element prázdny. Atribúty nie sú definované, protokol však ponúka voliteľnú implementáciu nasledujúcich:

- *expirationDate* – čas skončenia platnosti tokenu,
- *completeListSize* – celkový počet záznamov, ktoré majú byť vrátené. Toto číslo nemusí byť presné, pretože počas zberu sa záznamy môžu meniť
- *cursor* – počet už vrátených záznamov.

Harvester môže opätovne použiť posledne použitý *resumption token*. Dôvod tohto je dať harvesteru šancu spamätať sa z chýb buď na sieti alebo niekde inde, čo by malo inak za následok začatie zberu všetkých záznamov od začiatku. Medzi prvým a druhým použitím však môže dôjsť k zmenám v záznamoch. Server buď vráti aktuálnu postupnosť záznamov alebo chybu *badResumptionToken*, čo signalizuje, že v databáze došlo k tak výrazným zmenám, že by bolo lepšie začať odznova.

2.7 Selektívny zber

Selektívny zber umožňuje harvesterom lepší výber záznamov, o ktoré majú záujem. Protokol definuje dva typy kritérií na selektívny výber, ktoré môžu byť kombinované – časový rozsah a príslušnosť k množine.

2.7.1 Časový rozsah

Harvester môže zbierať iba záznamy, ktoré boli zmenené v uvedenom časovom rozsahu. Za zmenu sa považuje vytvorenie, modifikácia, v závislosti na stupni podpory zmazaných záznamov zmazanie a tiež pridanie, odobratie a zmena metadátového formátu.

Na upresnenie sa používajú argumenty *from* a *until* príkazov *ListRecords* a *ListIdentifiers*, pričom je možné uviesť oba, jeden, ale aj žiadny z nich. Ich význam je väčší alebo rovný resp. menší alebo rovný ako zadaná časová značka. Argument *from* musí byť menší alebo rovný argumentu *until*, inak server vráti chybu *badArgument*. Povinne podporovaná granularita je na úrovni dní, voliteľne na úrovni sekúnd.

2.7.2 Skupiny

Druhou možnosťou výberového zberu je príslušnosť k množine. Harvester argumentom set príkazov *ListRecords* a *ListIdentifiers* špecifikuje požadovanú množinu.

2.8 Príkazy protokolu

2.8.1 Identify

Tento príkaz slúži na získanie základných informácií o poskytovateľovi. Nemá žiadne argumenty, v prípade porušenia tohto pravidla vracia chybu *badArgument*.

Povinné elementy odpovede:

- *repositoryName* – názov poskytovateľa (nie identifikátor),
- *baseURL* – základná URL poskytovateľa,
- *protocolVersion* – verzia podporovaného protokolu,
- *earliestDatestamp* – dolný limit časových známok,
- *deletedRecord* – deklarácia úrovne podpory zmazaných záznamov. Je to práve jedna z hodnôt no, persistent, transient,
- *granularity* – úroveň rozlišovania časových známok (dni alebo sekundy),
- *adminEmail* – e-mailová adresa administrátora. Povolených je viacero inštancií tohto elementu.

Nepovinné elementy odpovede:

- *compression* – podpora kompresie na úrovni HTTP,

- *description* – popis poskytovateľa, kontajner ľubovolných XML dát. Popis protokolu ukazuje vhodné použitia tohto poľa. Poskytovateľ môže implementovať napríklad schému *oai-identifier* popisujúcu štruktúru jednoznačného identifikátora. Schéma *eprints* zas špecifikuje oprávnenie nakladania s dátami a schéma friends umožňuje odkázať harvester na ďalšie známe repozitáre, čím je možné budovať sieť spolupracujúcich poskytovateľov.

2.8.2 GetRecord

Príkaz vráti záznam v požadovanom formáte.
Argumenty:

- *identifier* – povinný argument, ktorého hodnotou je jednoznačný identifikátor záznamu,
- *metadataPrefix* – povinný argument označujúci formát, v ktorom má byť vrátená metadátová časť záznamu.

Chybové hlásenia:

- *badArgument* – dotaz používa ďalšie alebo duplicitné argumenty,
- *cannotDisseminateFormat* – metadátový objekt nie je v danom formáte k dispozícii,
- *dDoesNotExist* – je použitý neplatný identifikátor záznamu.

2.8.3 ListIdentifiers

Príkaz sa používa na získanie hlavičiek záznamov.
Argumenty:

- *from* – voliteľný argument upresňujúci dolnú hranicu časových známok pre účely selektívneho zberu,
- *until* – voliteľný argument upresňujúci hornú hranicu časových známok pre účely selektívneho zberu,
- *set* – voliteľný argument upresňujúci množinu pre účely selektívneho zberu,
- *metadataPrefix* – povinný argument označujúci formát, v ktorom má byť vrátená metadátová časť záznamu,
- *resumptionToken* – exkluzívny argument, ktorý sa používa na získanie ďalšej série záznamov.

Chybové hlásenia:

- *badArgument* – dotaz používa ďalšie alebo duplicitné argumenty,
- *badResumptionToken* – použitý token je nesprávny alebo vypršal,
- *cannotDisseminateFormat* – metadátový objekt nie je v danom formáte k dispozícii,
- *noRecordsMatch* – kombinácia argumentov from, until a set nevracia žiadne záznamy,
- *noSetHierarchy* – množiny nie sú podporované.

2.8.4 ListMetadataFormats

Príkaz vráti zoznam všetkých podporovaných metadátových formátov vrátane odkazu na ich špecifikáciu.

Argumenty:

- *identifier* – voliteľný argument, ktorý vracia zoznam metadátových formátov pre konkrétny záznam.

Chybové hlásenia:

- *badArgument* – dotaz používa ďalšie alebo duplicitné argumenty,
- *idDoesNotExist* – je použitý neplatný identifikátor záznamu,
- *noMetadataFormats* – pre daný záznam neexistujú metadátové záznamy.

2.8.5 ListRecords

Príkaz sa používa na získanie záznamov.

Argumenty:

- *from* – voliteľný argument upresňujúci dolnú hranicu časových známok pre účely selektívneho zberu,
- *until* – voliteľný argument upresňujúci hornú hranicu časových známok pre účely selektívneho zberu,
- *set* – voliteľný argument s hodnotou setSpec upresňujúci množinu pre účely selektívneho zberu,
- *metadataPrefix* – povinný argument označujúci formát, v ktorom má byť vrátená metadátová časť záznamu,

- *resumptionToken* – exkluzívny argument, ktorý sa používa na získanie d'alšej série záznamov.

Chybové hlásenia:

- *badArgument* – dotaz používa d'alšie alebo duplicitné argumenty,
- *badResumptionToken* – použitý token je nesprávny alebo vypršal,
- *cannotDisseminateFormat* – metadátový objekt nie je v danom formáte k dispozícii,
- *noRecordsMatch* – kombinácia argumentov from, until a set nevracia žiadne záznamy,
- *noSetHierarchy* – množiny nie sú podporované.

2.8.6 ListSets

Odpoveď'ou je zoznam všetkých množín.
Argumenty:

- *resumptionToken* – nepovinný argument, ktorý sa používa na získanie d'alšej série záznamov.

Chybové hlásenia:

- *badArgument* – dotaz používa d'alšie alebo duplicitné argumenty,
- *badResumptionToken* – použitý token je nesprávny alebo vypršal,
- *noSetHierarchy* – množiny nie sú podporované.

Kapitola 3

Dátový model

V súčasnosti systém WA-CZ spravuje približne 150 miliónov dokumentov a tento počet rýchlo rastie. Základnými požiadavkami sú efektívne vyhľadávanie a poskytovanie dokumentov na základe danej URL a času, schopnosť spracovať archívny formát ARC, jednoduché pridávanie nových záznamov, možnosť vytvárania štatistík a napojenie na Java web technológie.

Ako dátová základňa bola zvolená open-source databáza MySQL[1]. Kvôli rýchlejšiemu prístupu boli použité tabuľky typu MyISAM[2] a hlavná tabuľka docs bola navrhnutá tak, aby všetky riadky mali rovnakú dĺžku. Daňou za túto efektivitu je tabuľka resturls so zvyškami URL, ktoré sa do hlavnej tabuľky nezmestili a absencia cudzích kľúčov, čo nabúrava integritu dát.

Medzi základné požiadavky na nový systém patrí členenie na množiny podľa kolekcií a domén druhej úrovne. Súčasný dátový model poskytuje možnosti pre jednoduchú správu kolekcií, ktorá slúži na zaradenie dokumentu do tematického zberu, zberu seriálov alebo celoplošného zberu národnej domény. Nedostatkom je chýbajúci mechanizmus na jednoduchšie vystavovanie domén ako množín a neschopnosť systému priradiť dokument do viacerých kolekcií. Nasledujúce štyri podkapitoly hľadajú kompromis medzi ich počtom a schopnosťou databázového servera vyrovnať sa s nárastom veľkosti databázy.

3.1 Model s jednou kolekciou

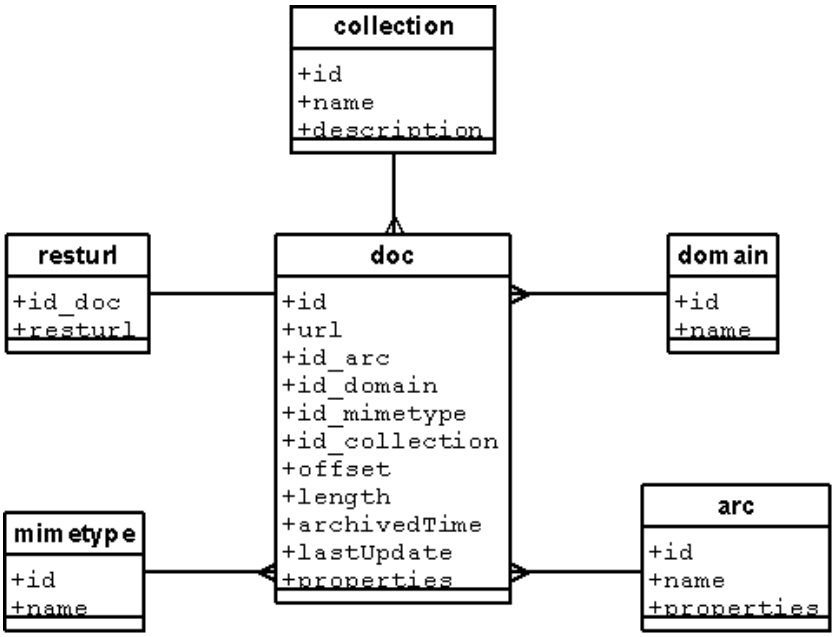
Model s jednou kolekciou vznikol malou úpravou súčasného modelu. Do hlavnej tabuľky doc bol pridaný atribút *lastUpdate*, do ktorého spúšť (trigger) pri vkladaní alebo modifikovaní záznamu doplní aktuálny čas. Tento atribút je nevyhnutný v prípade selektívneho zberu na základe časového rozsahu a pre správne generovanie hlavičiek záznamov v OAI.

Medzi tabuľkami *collection* a *doc* je zachovaná väzba 1:N a teda predpokladá, že každý záznam patrí do práve jednej kolekcie. Nová je tabuľka *domain*, do ktorej sa ukladá identifikátor a názov domény, z ktorej bol dokument získaný. Záznam teda okamžite patrí do dvoch množín – kolekcia a doména, pričom je dobré tieto dve skupiny navzájom odlíšiť. Dá sa to urobiť pridaním nejakého prefixu k názvu kolekcie a iného k názvu domény, napríklad *collection-volby06* pre kolekciu a *domain-www.webarchiv.cz* pre dokumenty

1. http://www.mysql.com
2. bližší popis tabulike typu MyISAM je dostupný na http://dev.mysql.com/doc/refman/5.0/en/myisam-storage-engine.html

v doméne *www.webarchiv.cz*. Iný spôsob počíta s vytvorením množiny *collection*, kde podmnožinami sú kolekcie a množiny *domain*, kde sú podmnožinami názvy domén. Vznikajú tu teda dva navzájom oddelené hierarchické stromy.

S tvorbou hierarchie sa dá ísť aj ďalej. Pre zdroje, z ktorých dokumenty môžeme voľne sprístupňovať, sa vžil názov seriály[3]. Majme napríklad kolekciu s názvom *serials* a každý zber dokumentov z týchto zdrojov je zaradený do kolekcie, ktorá patrí pod *serials*. Taká hierarchia potom môže mať tvar *collection:serials:harvest_november07*. U domén sa zas nemusíme obmedzovať iba na druhú úroveň. Dokument zo stránok WebArchívu bude zaradený v množine *domain:cz:webarchiv:www*. Ďalším oddeleným stromom množín môžu byť súborové formáty (mimetype). Pekný príklad využitia je služba pracujúca nad OAI, ktorá vytvára zbierku videí a hudby.



Obrázok 3.1: dátový model s jednou kolekciou

Ponúka sa niekoľko možností vytvorenia jednoznačného identifikátora. Prvou je zreťazenie URL a času, čo je však nepraktické pri dlhých a komplikovaných URL. Druhou možnosťou je kombinácia ARC súboru, v ktorom je dokument uložený a ofsetu. Problém nastane pri prechode na iný spôsob uloženia stiahnutého dokumentu. Najjednoduchším riešením je číslo, ktoré sa generuje pričítaním jednotky k predchádzajúcemu. K tomuto číslu je možné pridať prefix a vytvoriť tým identifikátor podľa odporúčaní iniciatívy Open Archives.

Všeobecne sa hierarchia v tabuľke rieši väzbou 1:N na samú seba. Do tabuľky je pridaný atribút *parent*, ktorý obsahuje identifikátor rodiča. V tomto prípade je nutné na zistenie príslušnosti k množinám vykonať dotaz na formát súboru, doménu, kolekciu a rekurzív-

3. zoznam partnerov je dostupný na http://www.webarchiv.cz/partneri/

ne na ich rodičov. Pri príkazoch *ListIdentifiers* a *ListRecords* sa tento postup musí opakovať pre každý záznam, čo môže byť veľký nápor na databázu. Iné riešenie môže spočívať v budovaní tejto stromovej hierarchie už pri vkladaní záznamu, kedy názov množiny tvorí dvojbodkami oddelený zoznam podmnožín. Spolu s podmnožinou sa do tabuľky rekurzívne vložia aj nadmnožiny. Napríklad pre množinu *collection:serials:harvest_november07* sa do tabuľky *collection* vloží množina *serials* a *serials:harvest_november07*. Dotaz na všetky dokumenty patriace do množiny *collection:serials* môže byť

```
SELECT doc.id FROM doc, collection
WHERE id_collection=collection.id AND
      (name='serials' OR name LIKE 'serials:%')
```

Jednou z požiadaviek na nový systém je podpora zmazaných záznamov. V praxi k mazaniu dát nedochádza, nutnosť považovať dokument za zmazaný však môže nastať pri neočakávaných situáciách, ako je napríklad havária dátového úložiska. Vlastnosti záznamov sú uložené v atribúte *properties*, na ktorý sa dá tiež pozerať ako na bitovú masku, kde jednotkový bit označuje hodnotu *true* danej vlastnosti. Ďalšou, v poradí piatou vlastnosťou bude *deleted*.

Prechod zo starého dátového modelu na nový je jednoduchý, zachováva zložitosť vyhľadávania v triede log n a priradenie dokumentu do jednej kolekcie. Každý dokument je tak zaradený v troch množinách – podľa kolekcie, domény a formátu súboru. Nový dátový model nezväčšuje pôvodný nijak dramaticky a slúži ako odrazový mostík pre ďalšie pokusy.

3.2 Model s pevným počtom kolekcií

Model s pevným počtom kolekcií je opäť len jednoduchým rozšírením predchádzajúceho dátového modelu a snahou mať možnosť priradiť dokument do viacerých kolekcií. Do hlavnej tabuľky *doc* sú pridané atribúty *collection_1* až *collection_m*. Pri návrhu však musí byť známe maximum priradení. Ak dokument priradíme do menšieho počtu kolekcií, hodnoty niektorých atribútov s identifikátormi kolekcií budú *NULL*. Hoci je takýto stav v praxi možný, nezodpovedá definícii relačnej algebry. Zvýšenie alebo zníženie tohto maxima nie je triviálne, pretože si vyžaduje zásahy do štruktúry tabuľky *doc*.

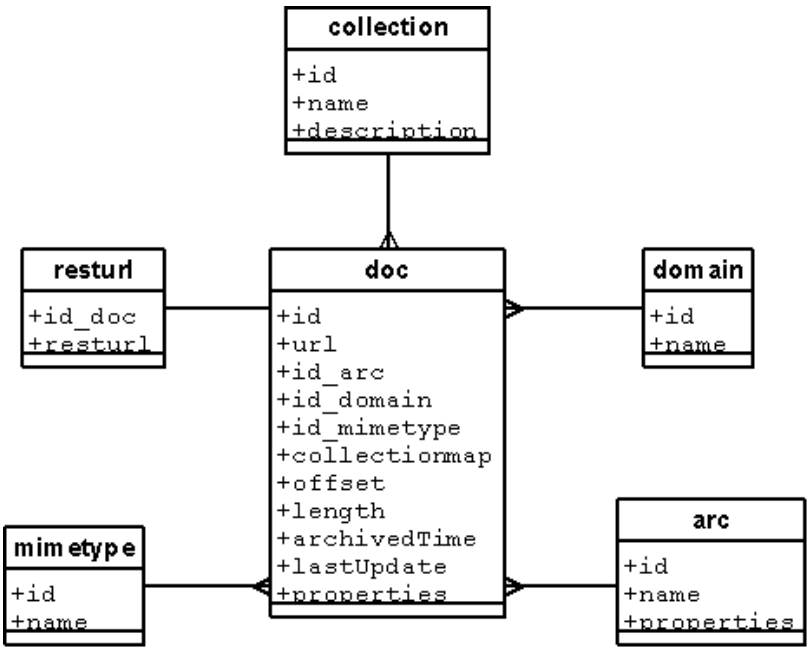
Zložitosť vyhľadávania dokumentu patriaceho do danej kolekcie je v triede m x log n, kde m je počet maximálny možný počet priradení a n je celkový počet záznamov. Tento model je teda o niečo horší a nie je vhodný na reálne použitie.

3.3 Binárna mapa

Zaujímavým riešením, ako zachovať štruktúru tabuliek a mať možnosť priradiť dokument do veľkého počtu kolekcií, je binárna mapa. Identifikátor kolekcie tvoria nuly a jedna jednotka, pričom každá kolekcia má túto jednotku na inej pozícii. Hodnota atribútu *collection_map* tabuľky doc vznikne XORovaním všetkých identifikátorov kolekcií, do ktorej do-

kument patrí. Maximálne množstvo množín je dané dátovým typom identifikátora. Použitím typu *bigint* je to 64, pri niektorom z textových dátových typov závisí od technických možností databázy.

Veľké množstvo informácie obsiahnuté v jednom atribúte je síce výhodou, získať však informáciu o tom, či dokument patrí do danej kolekcie, je extrémne zložité. Je nutné testovať všetky variácie jednotiek a núl okrem tých, ktoré majú nulu na pozícii, kde má daná kolekcia jednotku. Ak teda na identifikátor použijeme 64 bitový *bigint*, je nutné v tabuľke vyhľadávať až 2^{63} krát. Tento model teda môže byť použitý iba v prípade malého počtu kolekcií, do ktorých bude dokument zaradený.



Obrázok 3.2: dátový model s binárnou mapou

3.4 Väzba M:N

Model M:N dovoľuje každému dokumentu aby bol zaradený do ľubovolného počtu kolekcií. Dokonca sa ponúka možnosť zjednotiť tabuľky *mimetype*, *domain* a *collection* do tabuľky *set* a jednotlivé typy záznamov rozlišovať podľa názvu koreňovej množiny. V tomto prípade však tabuľka set rastie rýchlejšie a teda rastie aj čas na vyhľadávanie.

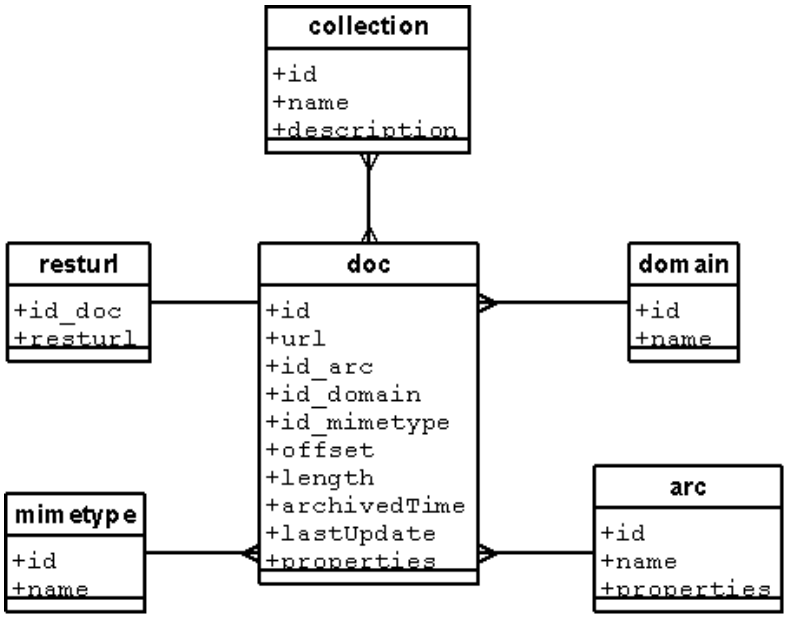
Dátový model bol testovaný na počítači s procesorom Intel Pentium 4 3.0 GHz a 1 GB RAM. Databáza bola naplnená takmer 140 miliónmi záznamov, mnohé boli priradené do niekoľkých kolekcií. Selekcia jedného dokumentu, či už s danou URL, časom vloženia do databázy alebo patriaceho do danej kolekcie, trvala niekoľko desiatok milisekúnd. Databázový stroj si teda dokázal poradiť s pridaním veľkej väzbovej tabuľky medzi tabuľkami

doc a *collection*. Testované boli aj dotazy, ktoré by boli použité pre OAI. Sú to dotazy typu

```
SELECT <atribúty> FROM <tabul'ky> WHERE <podmienky> LIMIT n OFFSET m
```

Tento dotaz simuluje výber veľkého množstva dát so stránkovaním, čiže postupným vracaním záznamov v sériách použitím resumption tokenu. Databáza MySQL v tomto prípade pracuje tak, že vyhľadáva záznamy splňujúce klauzulu *WHERE*, napočíta prvých m + n záznamov a ako výsledok zobrazí iba posledných n. S rastúcim ofsetom sú tu dva problémy. Prvým je, že databáza prehľadáva strom vždy od koreňa namiesto pokračovania od miesta, kde skončila. Druhým je rastúci čas potrebný na vyhľadanie prvých m záznamov a teda aj preťažovanie stroja. Pri m rádovo v desiatkach miliónov trvá vykonanie dotazu niekoľko minút.

Okamžite sa ponúkajú dve riešenia. Data-provider môže po dobu načítavania záznamov odpovedať prázdnym zoznamom dokumentov a resumption tokenom. Vďaka tomuto komunikácia na prvý pohľad funguje podľa pravidiel protokolu. Ďalším riešením je odpovedať HTTP kódom 403 alebo 503, nastaviť hodnotu *Retry-After* na čas potrebný na vyhľadanie záznamov a vyhľadávať na pozadí. Nedostatkom oboch postupov je, že neriešia ani jeden z problémov databázy, sú veľmi neštandardné a je veľmi pravdepodobné, že by harvester pri zbere havaroval.



Obrázok 3.3: dátový model s väzbou M:N

Problémom so stránkovaním sa vyhľadávacie nástroje vyhýbajú tým, že zobrazovanie výsledkov od niektorej stránky úplne zakážu[4]. Tento postup je odôvodnený tým, že uží-

4. napríklad Google zobrazí prvých 93 stránok

vateľ si pozrie iba niekoľko prvých výsledkov a to, čo ohodnocovací (ranking) algoritmus ohodnotil horšie, už nikoho nezaujíma.

Bol vykonaný pokus rozštiepiť tabuľku doc a väzbovú tabuľku medzi *doc* a *collection* na menšie, každú s 10 miliónmi záznamov. Vrámci jednej tabuľky je síce vyhľadávanie jediného dokumentu efektívne, v najhoršom prípade je však nutné prehľadať všetky tabuľky a čas odpovede sa blížil k hranici 1 sekunda. Odpoveď na dotaz s ofsetom v najhoršom prípade prišla v desiatkach sekúnd. Hoci by bolo možné túto hodnotu znížiť prechodom na výkonnejší produkčný server, proti tomuto riešeniu hovorí komplikovanejší prístup k dátam a náročnosť udržateľnosti konzistencie dát.

3.5 Použitie kurzora

Databázový kurzor je mechanizmus, pomocou ktorého je možné prechádzať záznamy z výsledku dotazu a nad každým záznamom vykonať nejakú operáciu. V MySQL fungujú v uložených procedúrach, kde sú inicializované, spracované a nakoniec uzatvorené[8]. Databázový systém PostgreSQL[5] ide ďalej a umožňuje odkaz na kurzor z uloženej procedúry vyviesť von a vrátiť klientskej aplikácii[9]. Toto riešenie je zaujímavé v najmä prípade veľkého počtu záznamov vo výsledku dotazu, pretože aplikácia nie je okamžite zahltená dátami, ale môže si postupne žiadať ďalšie riadky podľa potreby. Vytvorenie funkcie, ktorá vracia odkaz na kurzor, môže vyzerať nasledovne:

```
CREATE FUNCTION refcursor_function() RETURNS refcursor AS '
DECLARE
    ref refcursor;
BEGIN
    OPEN ref FOR SELECT * FROM test_table;
    RETURN ref;
END;
' LANGUAGE plpgsql;
```

Klientská aplikácia ďalej s odkazom pracuje ako klasickou množinou výsledkov dotazu.

Pri dotaze s ofsetom bol problém s rastúcim časom odpovede pri rastúcom počte stránok a prehľadávanie stromu vždy od koreňa. Kurzor zas načíta celú množinu výsledkov hneď na začiatku, čo znamená, že čím je táto množina väčšia, tým dlhšie trvá čakanie na prvú odpoveď systému. Rieši sa tým však problém so stránkovaním, pretože systém nemusí opäť vyhľadávať ďalšiu sériu záznamov, ale má ju pripravenú. Najväčšou nevýhodou kurzorov sú zvýšené požiadavky na systémové zdroje, pretože databázový systém si musí pamätať vybraté záznamy, ktoré sú navyše zamknuté pre zápis. Je preto nutné znížiť maximálny možný počet harvesterov, ktoré môžu naraz vykonávať zber a obmedziť maximálnu dobu nečinnosti, po ktorej dôjde k uzavretiu spojenia a tým aj k uvoľneniu kurzora z pamäte a odomknutiu zámkov.

Použitie kurzora predpokladá, že harvester bude pri zbere využívať možnosti selektívneho zberu a obmedzovať svoj dotaz tak, aby počet záznamov vo výsledku nebol príliš

5. http://www.postgresql.org

vysoký. Poskytovateľ by zas pri každom vložení nových záznamov mal dať návod na to, ako takýto zber uskutočniť. Testovací stroj bohužiaľ nemal dostatočný výkon na lepšie odskúšanie chovania kurzora pri vyššej záťaži.

PostgreSQL má na rozdiel od MySQL len jeden typ tabuliek, do ktorého môžeme pridať cudzie kľúče. Zvýši sa tým konzistencia dát a nie je potrebné používať externé nástroje na jej zaisťovanie. Zmenu prinášajú aj reťazce. Kým MySQL pracuje efektívnejšie s reťazcami s pevnou dĺžkou, systém PostgreSQL medzi pevnou a premenlivou dĺžkou nevidí rozdiel[10]. Vďaka tomu je možné úplne zrušiť tabuľku *resturl* a ukladať celú URL do atribútu *url* tabuľky *doc*. Na uchovávanie bitových masiek slúži typ *bit*[11], ktorý môže mať pevnú alebo premenlivú dĺžku. Dá sa s ním pracovať ako s reťazcom, kde znaky môžu nadobúdať hodnoty buď 0 alebo 1. Atribút *properties* teda môžeme previesť na typ *bit*.

názov atribútu	typ	popis
id	integer	jednoznačný identifikátor dokumentu; maximum typu integer je viac ako 2 miliardy, čo bude v najbližších rokoch stačiť
id_arc	integer	identifikátor ARC súboru – cudzí kľúč do tabuľky arc; v databáze ich momentálne je okolo 50 tisíc
id_domain	integer	identifikátor domény – cudzí kľúč do tabuľky domain
id_mimetype	smallint	identifikátor formátu súboru – cudzí kľúč do tabuľky mimetype; počet sa pohybuje rádovo v stovkách, smallint bude teda stačiť
url	varchar	url dokumentu; dĺžka je premenlivá a maximum závisí od technických možností databázy
doc_offset	integer	pozícia dokumentu v ARC súbore; maximum je viac ako 2 GB, čo postačuje
length	integer	veľkosť dokumentu v bytoch
archivedTime	bigint	čas stiahnutia dokumentu vo formáte RRRRMMDDHHMMSS
lastUpdate	bigint	čas vloženia dokumentu do databázy vo formáte RRRRMMDDHHMMSS
properties	bit(5)	vlastnosti dokumentu

Tabuľka 3.1: detailnejší popis tabuľky doc

3.6 Optimalizácia dotazov

Sekvenčné vyhľadávanie v tabuľkách je príliš pomalé a preto je nevyhnutné vytvoriť pomocné štruktúry.

Primárnym kľúčom hlavnej tabuľky *doc* bude atribút *id*. Zložitosť vyhľadávania napríklad pri príkaze *GetRecord* sa tým dostane do triedy log n. Kvôli rýchlejšiemu dohľadaniu

dokumentu podľa ARC súboru, domény alebo súborového formátu musí byť vytvorený index na ich identifikátory.

Dôležitým požiadavkom je rýchle vyhľadávanie dokumentu s danou url a časom. Na tento účel postačí jediný index definovaný na atribútoch *url* a *archivedTime*. Dá sa využiť aj na vyhľadávanie pomocou ľavých predpon. Starý systém využíval napríklad dotazy

```
SELECT * FROM doc WHERE url LIKE 'http://www.webarchiv.cz/%'
```

a

```
SELECT * FROM doc WHERE url LIKE 'http://www.webarchiv.cz/%'
                        AND archivedTime LIKE '20070101%'
```

V databázovom systéme PostgreSQL druhý z nich pracuje tak, že vyberie všetky riadky s daným prefixom URL, časový atribút prevedie do textovej podoby a záznamy s nevyhovujúcim časom vylúči. Efektívnejší spôsob je zadať dotaz nasledovne:

```
SELECT * FROM doc WHERE url LIKE 'http://www.webarchiv.cz/%'
                        AND archivedTime >= 20070101000000
```

Systém v tomto prípade nemusí prehľadávať všetky riadky s daným prefixom URL, ale dokáže efektívne vyhľadať prvý záznam vyhovujúci aj časovej značke.

Často používaným spôsobom prístupu k záznamom pomocou protokolu OAI-PMH je na základe časovej značky poslednej modifikácie. Kvôli urýchleniu bude index vytvorený aj na atribúte *lastUpdate*. Identifikátory v tabuľkách *domain*, *collection* a *mimetype* sa stanú primárnymi kľúčmi. Názov každej množiny musí byť jedinečný a kvôli častému prístupu k záznamom na základe príslušnosti k množine bude atribút name týchto tabuliek označený ako *UNIQE*. Znamená to vytvorenie indexu, kde nie je možné dva krát vložiť tú istú hodnotu.

Väzbová tabuľka medzi tabuľkami doc a collection obsahuje iba dva atribúty – identifikátor kolekcie a identifikátor dokumentu. Primárny kľúč bude definovaný na usporiadanej dvojici *id_doc* a *id_collection* a kvôli rýchlejšiemu dohľadaniu kolekcie bude vybudovaný index aj na atribúte *id_collection*.

3.7 Vkladanie dát

Na vkladanie dát do databázy boli vyvinuté dve aplikácie. Obe prechádzajú ARC súbormi, extrahujú z nich metadáta a ukladajú ich do databázy. Pri vkladaní množiny, či už ide o formát súboru, doménu alebo kolekciu, sa vkladajú aj rodičia. Vyhľadáva sa čo najdlhší možný zoznam podmnožín, ktoré sú oddelené dvojbodkami. Ak neexistuje, vloží sa do databázy a rekurzívne sa vyhľadáva rodičovská množina.

Rozdiel medzi oboma aplikáciami je v spôsobe, akým sa do databázy vkladajú nové záznamy. Prvá používa SQL príkaz *INSERT*. Obyčajné vkladanie po riadkoch je v systéme PostgreSQL síce flexibilnejšie, má však značnú réžiu a naplniť jednoduchú tabuľku jedným miliónom záznamov môže trvať aj hodiny. Druhá aplikácia využíva príkaz *COPY*[12], ktorý

je optimalizovaný na vkladanie veľkého množstva dát. Tento spôsob vyžaduje použitie ďalšej databázy, napríklad MySQL, ktorá je pri obyčajnom vkladaní rýchla a s ktorou aplikácia počas prechádzania ARC súbormi pracuje. Na začiatku vkladania obsahuje kópiu pôvodných tabuliek *mimetype* a *domain*, na konci aj nové dáta. Každá tabuľka, okrem tabuľky *collection*, je reprezentovaná súborom, do ktorého sa ukladajú nové riadky. Názvy súborov slúžia ako argumenty pre príkaz *COPY* a ich obsah sa do databázového systému vkladá v jednej transakcii. Ak teda dôjde k chybe pri vkladaní ľubovolného riadku ľubovolného súboru, nahrávanie dát sa odvolá.

Vkladanie veľkého množstva dát v transakcii má jednu nevýhodu. Atribút lastUpdate obsahuje časovú značku poslednej modifikácie dokumentu. Pri spracovávaní v transakcii je to čas jej spustenia. Ak trvá vkladanie dát príliš dlho, časová značka nezodpovedá času, od ktorého je dokument skutočne k dispozícii. Vkladanie 1 milióna riadkov pomocou novej aplikácie trvá približne 45 minút[6]. Napríklad počet záznamov zberu celej národnej domény v roku 2006 je približne 71 miliónov, čiže vloženie zaberie 53 hodín. Ak by nejaký harvester vykonával zber denne, pričom by svoj dotaz obmedzil na predchádzajúci deň, nemal by možnosť túto kolekciu zachytiť. Teoreticky môže harvseter robiť zber trebárs v hodinových intervaloch (alebo aj menších), preto je veľmi dôležité dáta vkladať oddelene po niekoľkých ARC súboroch. Na druhej strane je tento spôsob pracnejší a tiež menej efektívny.

6. Prechádzanie ARC súborov a vloženie 1 milióna záznamov do MySQL databázy trvá asi 30 minút. K tomu treba pripočítať vytvorenie kópie tabuliek pre MySQL a vkladanie dát do PostgreSQL, ktorý je pri veľkom objeme efektívnejší. Kopírovanie tabuliek je našťastie možné vynechať v prípade, ak dáta do PostgreSQL neboli predtým vkladané iným spôsobom (napríklad prvou zmienenou aplikáciou).

Kapitola 4

Programové riešenie

4.1 Voľne dostupné nástroje

Zoznam na domovskej stránke iniciatívy Open Archives[1] zahŕňa množstvo data-providerov, harvesterov a utilít pre OAI-PMH.

Najzaujímavejšou skupinou sú tzv. inštitucionálne repozitáre[13], ktoré tvoria kompletnú produkčnú platformu. Medzi takéto repozitáte patrí Archiméde[2] vyvíjaný na Laval University v Quebecku, kde spravuje diplomové, dizertačné a iné elektronické práce. Medzi hlavné výhody patrí podpora jazykov a automatická full-textová indexácia. Holandský projekt ARNO[3], rovnako ako nemecký OPUS[4], slúžia na výmenu e-printov medzi poprednými univerzitami vo svojich domovských krajinách. DSpace[5] vznikol spoločným úsilím MIT a firmy HP a okrem spravovania a výmeny sa zameriava aj na dlhodobú archiváciu. K najväčším implementáciám môžeme zaradiť CDS Invenio[6] vyvíjaný Európskou organizáciou pre jadrový výskum (CERN) s počtom záznamov viac ako 900 000 a Fedoru[7] s efektívnou podporou pre približne 1 milión záznamov. Medzi najčastejšie inštalácie patrí systém Eprints[8].

Zvyšné repozitáre môžeme rozdeliťdo dvoch skupín – statické a dynamické. V statických nedochádza k zmenám v záznamoch a metadáta sú väčšinou uložené na disku v podobe XML súborov. K takýmto implementáciám patrí napríklad Rapid Visual OAI Tool[9]. Dynamické väčšinou spolupracujú s databázou a sú rôzne konfigurovateľné. PHP OAI Data Provider[10] neponúka takmer žiadne možnosti konfigurácie, má napevno zakódované typy tabuliek a atribútov. Pri iných, napríklad OAIbiblio Data Provider[11], je možné editovať dotazy tak, aby zodpovedali štruktúre databázy. Ale, ako priznávajú sami autori, úprava nemusí byť jednoduchá.

Vynikajúcim nástrojom je OAICat[12], ktorý používa napríklad Austrálska národná kniž-

1. dostupný na http://www.openarchives.org/pmh/tools/tools.php
2. http://www.bibl.ulaval.ca/archimede/index.en.html
3. http://www.uba.uva.nl/arno
4. http://elib.uni-stuttgart.de/opus/index.php?la=en
5. http://www.dspace.org/
6. http://cdsware.cern.ch/invenio/index.html
7. http://www.fedora.info/
8. http://www.eprints.org/software/
9. http://rvot.sourceforge.net/
10. http://physnet.uni-oldenburg.de/oai/
11. http://www.ibiblio.org/oaibiblio/
12. http://www.oclc.org/research/software/oai/cat.htm

nica na sprístupnenie kolekcie hudby, obrázkov, rukopisov, kníh a seriálov[13]. Základ tejto Java web servlet aplikácie tvoria abstraktné triedy *ServerVerb* a *AbstractCatalog*. *ServerVerb* reprezentuje všeobecný príkaz OAI-PMH. Potomkovia túto triedu pretvárajú na jeden z konkrétnych príkazov protokolu. Trieda *AbstractCatalog* je zas všeobecným predstaviteľom dát. Balík obsahuje ukážkové triedy dediace z abstraktného katalógu, ktoré majú dáta uložené na disku v adresároch, v XML súboroch a v relačnej databáze. Vytvorenie programovej časti data-providera teda môže znamenať iba vytvorenie nového potomka triedy *AbstractCatalog*. Situáciu to však uľahčovať nemusí. A to práve kvôli tomu, čo je najväčšou výhodou tejto aplikácie, teda všeobecná použiteľnosť.

Ja som nakoniec rozhodol vytvoriť vlastnú aplikáciu, ktorá sa bude dať jednoduchšie prispôsobiť podmienkam českého WebArchivu.

4.2 Vlastné riešenie

Aplikácia je napísaná v jazyku Java a pracuje pod kontajnerom Apache Tomcat[14]. Pre jednoduchšiu správu je rozdelená do niekoľkých balíkov. Balík *cz.webarchiv.arcwb.oai.verb* predstavuje príkazy, *cz.webarchiv.arcwb.oai.exception* zas všetky chybové stavy protokolu OAI-PMH, *cz.webarchiv.arcwb.oai.util* obsahuje pomocné triedy slúžiace na vytvorenie tokenov a časových značiek *cz.webarchiv.arcwb.oai.record* zahŕňa triedy na vytvorenie metadátového záznamu v požadovanom formáte a balík *cz.webarchiv.arcwb.oai* obsahuje iba servlet OAI.java obsluhujúci komunikáciu medzi aplikáciou a harvesterom. Diagram tried je uvedený v prílohe C.

Pre dobrú funkčnosť systému je veľmi dôležitá efektívna práca s databázou. Klasické riešenie by vyzeralo tak, že aplikácia nadviaže spojenie, zadá dotaz, spracuje výsledky a spojenie uzavrie. V prípade väčšej vyťaženosti je neefektívne spojenie neustále otvárať a zatvárať. Lepšie riešenie využíva tzv. *connection pooling*, frontu spojení. Aplikačný server si drží niekoľko otvorených spojení vo fronte a prideľuje ich aplikáciám podľa potreby. Aplikácia po skončení výpočtu spojenie do fronty vráti. Každý aplikačný server alebo servlet kontajner túto metódu rieši trochu inak. Pod systémom Apache Tomcat vo verzii 5.x alebo 6.x sa dá využiť *Java naming and directory interface* (JNDI)[14]. Globálne nastavenia sú uložené v súbore *$TOMCAT_HOME/conf/context.xml*. Ako už bolo spomenuté, je dôležité obmedziť počet klientov a čas, po ktorom sa spojenie s databázou aplikácii odoberie. Pre ostatné aplikácie je najmä výrazné obmedzovanie počtu klientov nežiadúce a preto musí byť konfigurácia pre OAI-PMH oddelená od globálnej. Dá sa to urobiť vytvorením súboru *context.xml* v lokálnom adresári *WEB-INF* aplikácie. Jeho obsah vyzerá nasledovne:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/OAI">
  <Resource auth="Container"
            driverClassName="org.postgresql.Driver"
            logAbandoned="True"
```

13. kolekcia je dostupná na http://www.nla.gov.au/digicoll/oai/
14. http://tomcat.apache.org/

```
            maxActive="2"
            maxWait="10000"
            name="db"
            password="some_password"
            removeAbandoned="True"
            removeAbandonedTimeout="60"
            type="javax.sql.DataSource"
            url="jdbc:postgresql://localhost:5432/arcrepos"
            username="some_username"/>
</Context>
```

Popísaná štruktúra je typu *DataSource* a obsahuje základné nastavenia databázového spojenia, napríklad driver, URL, užívateľské meno a heslo. Atribútom *maxActive* obmedzíme maximálny počet súčasne otvorených spojení, a tým aj maximálny možný počet harvesterov vykonávajúcich zber. *MaxWait* definuje dobu v sekundách, po ktorú bude aplikácia na spojenie čakať v prípade, že fronta je vyčerpaná. Dôležitú úlohu zohrávajú atribúty *removeAbandoned* a *removeAbandonedTimeout*. Ak je otvorené spojenie nevyužité viac ako 60 sekúnd, je aplikácii odobrané a vrátené späť do fronty. Vo všeobecnosti sa tento mechnizmus využíva ako prevencia pred chybami alebo haváriami aplikácií, ktoré spojenie nestihnú uzavrieť a už nikdy ho do fronty nevrátia. Rieši tiež problém starých resumption tokenov. V prípade, že harvester nepožiada o ďalšie záznamy včas, zdroje sa uvoľnia pre ďalšieho klienta. Toto nastavenie má však aj ďalší efekt – ak databázový server nestihne v danej dobe záznamy vyhľadať, spojenie sa uzavrie a vykonávanie aplikácie skončí s chybovým hlásením. Nasledujúci fragment kódu slúži na získanie spojenia z fronty:

```
InitialContext ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("java:comp/env/db");
Connection conn = ds.getConnection();
```

Základ aplikácie tvorí abstraktná trieda *Verb*, ktorá predstavuje všeobecný príkaz protokolu OAI-PMH. Dokáže vytvoriť hlavičku XML dokumentu, otestovať správnosť zadaných parametrov, získať databázové spojenie z fronty, poslať výsledný XML dokument na výstup a uložiť svoj stav do tokenu. Dediace triedy musia prepísať abstraktné triedy na inicializáciu parametrov a vytvorenie tela odpovede.

Aplikácia začína svoj beh vykonávaním súboru index.jsp, ktorý zakáže vytvorenie nového sedenia a spustí servlet *OAI.java*. Harvester môže svoje dotazy posielať metódou GET i POST, a preto musí *OAI.java* implementovať metódu *doGet()* aj *doPost()*. Servlet podľa obsahu atribútu verb vyberie jednu z tried implementujúcu príslušný príkaz, zavolá metódu na vytvorenie XML dokumentu s odpoveďou a nakoniec metódu na vypísanie obsahu dokumentu na výstup. Servlet tiež zachytáva výnimky a snaží sa rozlíšiť chyby vzniknuté vyčerpaním fronty spojení od ostatných. V prípade prázdnej fronty odpovie HTTP kódom 503, preťaženie servera, v opačnom prípade kódom 500, vnútorná chyba servera.

Resumption token je mechanizmus, pomocou ktorého si server pamätá stav komunikácie s harvesterom. Návrh tried *Token* a *TokenCareTaker* je inšpirovaný návrhovým vzorom Memento[15], ktorý rieši problém uloženia a obnovy stavu objektu. Na uloženie slúži trieda

Token, ktorá rozlišuje, či už bol token použitý alebo je úplne nový. Do nového tokenu sa vložia databázové premenné potrebné na získavanie ďalších záznamov z databázy a odkaz na predchádzajúci token. Starý token obsahuje iba už vygenerovaný element *ListIdentifiers*, *ListRecords* alebo *ListSets*. Trieda *TokenCareTaker* je typu jedináčik (singleton) a zodpovedá za správu tokenov, ktoré si ukladá do mapy ako usporiadanú dvojicu identifikátor, token. Identifikátor je vygenerovaný ako počet milisekúnd od 1.1.1970. Protokol OAI-PMH totiž nedefinuje sedenia, a preto je možné, že zber začne jeden harvester a dokončí ho iný. Server by však mal vytvárať identifikátory tokenov tak, aby nebolo úplne jednoduché uhádnuť ďalší identifikátor, čo by sa dalo zneužiť na prekazenie zberu inému klientovi. Staré tokeny sú podľa časovej značky vyhľadávané a následne mazané pri každom požiadavku o vrátenie tokenu.

Všetky dokumenty sú tzv. e-born a metadáta pochádzajú z jedinej databázy. Pridanie podpory pre ďalší metadátový formát vďaka tomu znamená, že v tomto formáte je možné vrátiť každý záznam. Jediným povinne podporovaným formátom je Dublin Core. Šikovným spôsobom, ktorým sa dá podpora pre ďalšie formáty sa dá vyriešiť, je pomocou XSLT transformácie XML dokumentu s metadátovými časťami záznamov zakódovanými v *oai_dc*. Nakoniec implementované riešenie využíva abstraktnú triedu *RecordCreator*, ktorá ako vstup dostane metadátový element a dáta. Potomkovia túto triedu pretvoria tak, aby naplnila element poskytnutými dátami v požadovanom formáte.

4.3 Konfigurácia

Systém bol testovaný na nasledujúcom softvérovom vybavení:

- operačný systém Debian GNU/Linux lenny a Microsoft Windows XP
- Java verzia 1.5.0 a 1.6.0
- Apache Tomcat verzia 5.5.15 a 6.0.14
- MySQL server verzia 5.0
- PostgreSQL verzia 8.2
- MySQL-JDBC driver mysql-connector-java-5.0.7-bin.jar
- PostgreSQL-JDBC driver postgresql-8.2-506.jdbc3.jar a postgresql-8.2-506.jdbc4.jar

Pre správny beh je nutné upraviť už spomínaný súbor *context.xml* a súbor *oai.properties*, ktorý je zobrazený a okomentovaný v prílohe C.

Kapitola 5

Záver

Táto práca sa zaoberá návrhom a implementáciou rozhrania data-providera protokolu OAI-PMH. Pojem interoperability a úloha konceptu OAI v nej je v komunite (digitálnych) knižníc známy, preto úvod iba pripomína základné princípy. Druhá kapitola rozoberá protokol podrobnejšie a pre lepšiu ilustráciu sú niektoré výstupy uvedené aj prílohe A. Praktickú časť tvorí výstavba dátového úložiska a implementácia aplikačnej vrstvy. Tretia kapitola hľadá vhodnú schému uloženia dát a približuje chovanie navrhovaných dátových modelov. Aplikácia je postavená na Java enterprise technológiách a jej návrh popisuje kapitola č.4.

Najväčšie archívy a systémy podporujúce OAI-PMH (CDS Invenio, Fedora) sú optimalizované pre jeden až dva milióny záznamov. Táto práca sa snaží o vybudovanie rozhrania nad archívom, ktorého veľkosť pri podobných projektoch nie je úplne bežná. Napriek tomu je to možné, prináša to však určité komplikácie. Prvou je vkladanie dát, ktoré trvá o čosi dlhšie, je pracnešie a poskytovateľ by mal zverejňovať aj určitý postup, ako zadávať dotazy tak, aby nedošlo k preťaženiu servera. Repozitár je menej flexibilný a nespráva sa ako menšie implementácie. Harvester by mal využívať metódy selektívneho zberu a nesťahovať príliš veľké objemy dát naraz.

Veľkým problémom pri písaní práce bola októbrová havária hlavného dátového úložiska v Prahe, ktorá nastala akurát počas plnenia databázy. Náhradný testovací server[1], ktorý by mal byť v blízkej budúcnosti vyradený, neposkytoval dostatok výkonu na otestovanie „limitov“ vo veľkosti a rýchlosti pri sťahovaní dát. Až po úplnom odstránení závad havárie bude možné zistiť, akú záťaž systém znesie.

Nás ešte čaká skutočné nasadenie, systém sa pravdepodobne stane súčasťou jednotnej informačnej brány Národní knihovny. Prvé využitie je už známe, popri písaní práce bol vytvorený jednoduchý harvester, ktorý na základe metadát sťahuje z archívu aj súbory a ukladá ich na disk. Aplikácia môže poslúžiť na čiastočné obnovenie webov zo zálohy. Po spustení pravdepodobne budú rozhranie využívať najmä spolupracujúci partneri, ktorí budú mať záujem vidieť, aké dáta od nich máme. Ďalšie využitie je hlavne na užívateľoch.

1. v čase písania práce bolo rozhranie nad malou kolekciou dát dostupné na testovanie na http://war.webarchiv.cz:8080/OAI/

Literatúra

[1] Šušol, J.: *Otvorené archívy – model systémovej interoperability*, ITlib. Informačné technológie a knižnice, č. 3/2002, Dokument dostupný na URL<http://www.cvtisr.sk/itlib/itlib023/susol.htm>. 1, 1.1

[2] *The Open Archives Initiative Protocol for Metadata Harvesting* <http://www.openarchives.org/OAI/openarchivesprotocol.html>, Open Archives Initiative, 2004-10-12. 1

[3] *Dublin Core* <http://en.wikipedia.org/wiki/Dublin_Core>, Wikipedia, the free encyclopedia. 1

[4] Matějka, L.: *Zpřístupnění archivu českého webu [diplomová práce]*, Brno, 2006. 1

[5] *Specification and XML Schema for the OAI Identifier Format* <http://www.openarchives.org/OAI/2.0/guidelines-oai-identifier.htm>, Open Archives Initiative, 2006-03-09. 2.1.1

[6] *Conveying rights expressions about metadata in the OAI-PMH framework* <http://www.openarchives.org/OAI/2.0/guidelines-rights.html>, Open Archives Initiative, 2005-05-03. 2.1.2

[7] *XML schema to hold provenance information in the 'about' part of a record* <http://www.openarchives.org/OAI/2.0/guidelines-provenance.htm>, Open Archives Initiative, 2002-07-02. 2.1.2

[8] *MySQL AB :: MySQL 5.1 Reference Manual :: 18.2.9 Cursors* <http://dev.mysql.com/doc/refman/5.1/en/cursors.html>. 3.5

[9] *PostgreSQL 8.2.5 Documentation, chapter 37.8. Cursors* <http://www.postgresql.org/docs/8.2/static/plpgsql-cursors.html>. 3.5

[10] *PostgreSQL 8.2.5 Documentation, chapter 8.3. Character Types* <http://www.postgresql.org/docs/8.2/static/datatype-character.html>. 3.5

[11] *PostgreSQL 8.2.5 Documentation, chapter 8.9. Bit String Types* <http://www.postgresql.org/docs/8.2/static/datatype-bit.html>. 3.5

[12] *PostgreSQL 8.2.5 Documentation, COPY* <http://www.postgresql.org/docs/8.2/static/sql-copy.html>. 3.7

[13] *A Guide to Institutional Repository Software*, Open Society Institute, 400 West 59th Street, New York, NY 10019, 2004, Dokument dostupný na URL (november 2007)<http://www.soros.org/openaccess/pdf/OSI_Guide_to_IR_Software_v3.pdf>. 4.1

[14] *The Apache Tomcat 5.5 Servlet/JSP Container, JNDI Datasource HOW-TO* <http://tomcat.apache.org/tomcat-5.5-doc/jndi-datasource-examples-howto.html>. 4.2

[15] *Návrhové vzory (design paterns), Memento Pattern* <http://objekty.vse.cz/Objekty/Vzory-Memento>. 4.2

Dodatok A

Odpovede protokolu OAI-PMH

Odpoveď na dotaz *http://war.webarchiv.cz:8080/OAI/OAI?verb=Identify*

```
<OAI xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
 http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
  <responseDate>2008-01-06T19:16:55Z</responseDate>
    <request verb="Identify">
      http://war.webarchiv.cz:8080/OAI/OAI
    </request>
  <Identify>
    <repositoryName>WebArchiv</repositoryName>
    <url>http://war.webarchiv.cz:8080/OAI/OAI</url>
    <protocolVersion>2.0</protocolVersion>
    <adminEmail>admin@webarchiv.cz</adminEmail>
    <earliestDatestamp>2000-01-01T12:00:00Z</earliestDatestamp>
    <deletedRecord>persistent</deletedRecord>
    <granularity>YYYY-MM-DDThh:mm:ssZ</granularity>
  </Identify>
</OAI>
```

Odpoveď na dotaz *http://war.webarchiv.cz:8080/OAI/OAI?verb=GetRecord&identifier=10&metadataPrefix=oai_dc*

```
<OAI xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
 http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
  <responseDate>2008-01-06T19:32:38Z</responseDate>
  <request identifier="10" metadataPrefix="oai_dc"
   verb="GetRecord">
    http://war.webarchiv.cz:8080/OAI/OAI
  </request>
  <GetRecord>
    <header>
      <identifier>10</identifier>
      <datestamp>2008-01-06T16:21:20Z</datestamp>
      <setSpec>col:culture:estonia</setSpec>
      <setSpec>dom:ee:etv24:www</setSpec>
      <setSpec>mim:text:html</setSpec>
    </header>
    <metadata>
      <oai_dc:dc xsi:schemaLocation="
```

```
       http://www.openarchives.org/OAI/2.0/oai_dc/
       http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
        <dc:format>text/html</dc:format>
        <dc:format>computerFile</dc:format>
        <dc:format>466 bytes</dc:format>
        <dc:date>2006-04-05T14:22:18Z</dc:date>
        <dc:source>http://www.etv24.ee/robots.txt</dc:source>
      </oai_dc:dc>
    </metadata>
  </GetRecord>
</OAI>
```

Odpoveď na dotaz *http://war.webarchiv.cz:8080/OAI/OAI?verb=ListMetadataFormats*

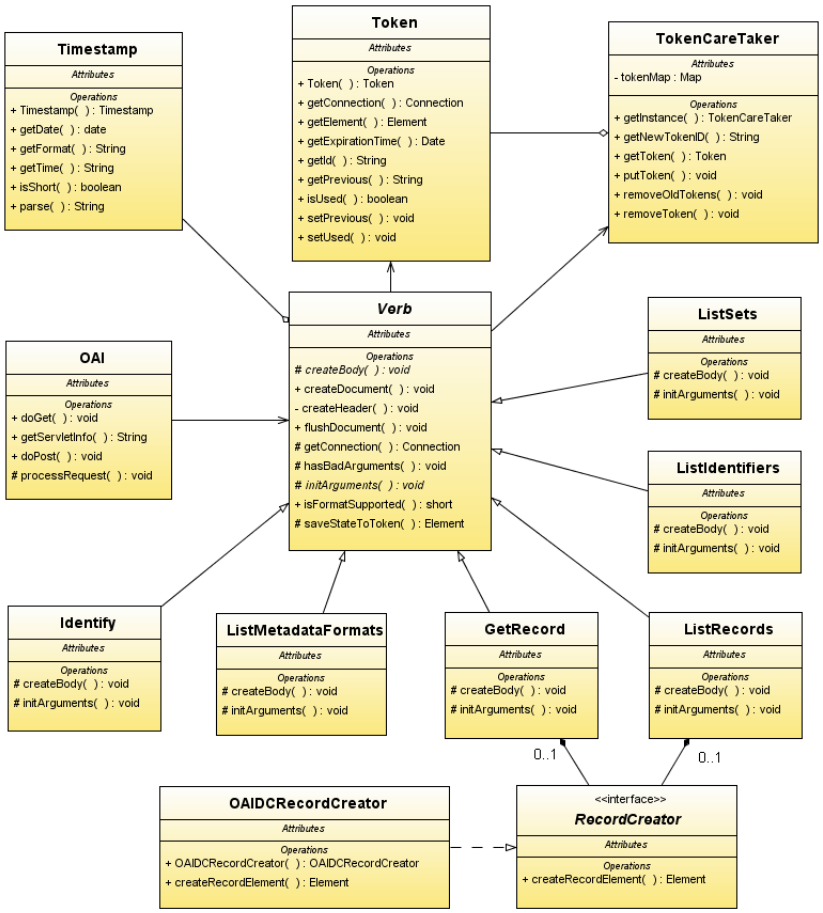
```
<OAI xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
 http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
  <responseDate>2008-01-06T20:43:21Z</responseDate>
  <request verb="ListMetadataFormats">
    http://war.webarchiv.cz:8080/OAI/OAI
  </request>
  <ListMetadataFormats/>
  <metadataFormat>
    <metadataPrefixElement>oai_dc</metadataPrefixElement>
    <schema>
      http://www.openarchives.org/OAI/2.0/oai_dc.xsd
    </schema>
    <metadataNamespace>
      http://www.openarchives.org/OAI/2.0/oai_dc/
    </metadataNamespace>
  </metadataFormat>
</OAI>
```

Odpoveď na dotaz *http://war.webarchiv.cz:8080/OAI/OAI?verb=ListRecords*

```
<OAI xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
 http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
  <responseDate>2008-01-06T20:47:50Z</responseDate>
  <request verb="ListRecords">
    http://war.webarchiv.cz:8080/OAI/OAI
  </request>
  <error code="badArgument">
    The request includes illegal arguments, is missing required
    arguments, includes a repeated argument, or values for
    arguments have an illegal syntax.
  </error>
</OAI>
```

Dodatok B

Diagram tried



Obrázok B.1: diagram tried

Dodatok C

Konfiguračný súbor oai.properties

```
# názov repozitára
repositoryName = WebArchiv

# verzia protokolu
protocolVersion = 2.0

# najskoršia časová značka
earliestDatestamp = 2000-01-01T12:00:00Z

# stupeň podpory zmazaných záznamov
deletedRecord = persistent

# názov metadátového formátu (oai_dc je povinný)
metadataFormat[0] = oai_dc

# trieda implementujúca metadátový formát
metadataFormatClass[0] = cz.webarchiv.arcwb.oai.record.OAIDCRecordCreator

# XML schéma metadátového formátu
metadataFormatSchema[0] = http://www.openarchives.org/OAI/2.0/oai_dc.xsd

# menný priestor metadátového formátu
metadataFormatNamespace[0] = http://www.openarchives.org/OAI/2.0/oai_dc/

# emaily - aspoň jeden je povinný
adminEmail[0] = admin@webarchiv.cz

# obsah nepovinného elementu description príkazu Identify
description[0] =

# počet záznamov, ktoré v jednej dávke vráti ListIdentifiers, ListRecords
# a ListSets
numOfRecords = 100

# doba, po ktorú si systém pamäté tokeny [minúty]
liveTime = 1
```

Dodatok D

Obsah priloženého CD

Výpis adresárov a ich obsach:

- *ARCInsert* – aplikácia na vkladanie dát do databázy využívajúca MySQL
- *ARCInsert2* – aplikácia na vkladanie dát do databázy pracujúca bez pomoci d'alšej databázy
- *ddl* – skripty na vytvorenie tabuliek a funkcií pre PostgreSQL
- *OAI* – OAI-PMH rozhranie
- *text* – text bakalárskej práce