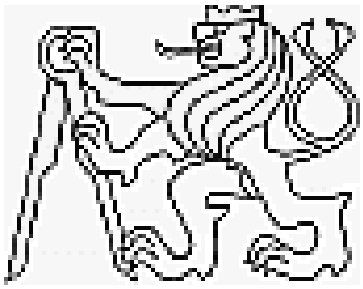
Mapa webové sítě

Martin Řimnáč

Diplomová práce

2004



Vedoucí práce: Ing. Richard Šusta, PhD.
Oponent: Doc. Ing. Zdeněk Kouba, CSc.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze 19. ledna 2004 …….

podpis

Projekt implementuje systém pro stažení dat z webových serverů, jejich crawlerizaci a indexaci, úložiště dat je realizováno pomocí databázového stroje. Primárním zaměřením systému je analýza linků mezi stránkami. Součástí práce je implementace rekonfigurovatelného systému s prioritním řazením vstupních dotazů do fronty a jednoduchá fulltextová báze dat. Vstupním a výstupním formátem dat je XML technologie, konfigurovatelnost je zajištěna pomocí XSLT transformací. Obecně navržený systém bude zkonfigurován pro analýzu webových stránek Katedry řídicí techniky na FEL, ČVUT. Cílem práce je poskytování dat z webových stránek pro generování informačních map a dalších dotazů.

The project implements the system for download, crawlerization and indexation of pages from the web servers, data are stored in a database. The primary system function is analyzation of the links between the pages. The project is designed as easy reconfigured system with the priority sorted queue of input queries. The project also contains basic fulltext base. System input and output format is XML, configurations are in XSLT format. General-purposed designed system is configured for analyzing the pages web at server of Department of Control Engeneering, FEL, CTU. Data are used for the info-map generator and for extended web page analyze.

1 Úvod, motivace

1.1 Zadání

Název tématu: Mapa webové sítě

Prostudujte si realizované metody pro mapování webové sítě.

Navrhněte způsob a pravidla pro mapování katedrální sítě.

Vybranou metodu realizujte.

1.2 Rozbor problematiky

1.2.1 Motivace

Hlavní motivací je automatické generování informačních map. Vedle klasických fulltextových dotazů systém umožnuje zpracovávat další typy dotazů, například hierarchickou mapu webu. Tato práce se zabývá analýzou vstupních dat, tedy získáním dat z webových serverů a jejich transformací a následným uložením do databáze. Zobrazování dat z databáze je předmětem navazující diplomové práce [4].

Hierarchická mapa je graf typu strom, který reprezentuje strukturu webových stránek na serveru. Kořenem stromu je *úvodní stránka* (někdy též homepage), navazující uzly jsou všechny webové stránky, na které je z úvodní stránky odkazováno. Z těchto navazujících uzlů opět vedou hrany připojující k danému uzlu webové stránky, které jsou z uzlů odkazovány a zároveň již nejsou reprezenovány uzlem v jiné části stromu.

Hierarchickou mapu lze použít k celkové propagaci webu nebo v okamžiku, kdy návštěvník webových stránek požaduje stránku s neplatným URL (která neexistuje).

Systém dle zadání musí být schopen generovat hierarchickou mapu webových stránek Katedry řídící techniky na FEL, ČVUT. Původně měl monitorovat i obsah webových stránek studentů v rámci kont na serveru katedry, což bylo prakticky znemožněno absencí těchto stránek po rekonfiguraci sítě v září 2003.

Systém se neomezuje pouze na tuto úlohu, je navržen velmi obecně. Lze jej bez zásahu do zdrojových kódů rozšířit o celou množinu dotazů, jmenujme např. zpřesňování výsledků fulltextového dotazu nebo jejich následná analýza (např. lze zjistit, že u slova systém se velmi často vyskytuje slovo dynamický -pak lze jednoduše rozdělit výsledek fulltextového hledání na ty stránky, které navíc obsahují slovo dynamický, respektive neobsahují; tento postup vede na generování map jiného typu).

Využití systému je dáno předchozím textem. Systém je vhodný pro webmastery, dovede je upozornit na odkazy na neexistující stránky a zároveň slouží jako zdroj pro presentaci webových stránek. Další využití je pro návštěvníky stránek, podle mapy se dovedou rychle zorientovat a nalézt požadované informace.

Součástí systému je i databázově realizovaná fulltextová báze dat, která je určena jako primární zdroj dat pro další analýzy. Nejedná se o další implementaci fulltextového stroje, ta nebyla cílem, fulltextová data v praxi předpokládáme externí. Tato báze dat bude použita pro potřeby

systému. Pokud bychom se zabývali i návrhem fulltextové báze, technologie by byla pravděpodobně jiná (pro tak velký objem dat se používají specializované indexační algoritmy, databázové systémy pouze pro menší objemy dat).

Pro systém bylo navrženo webové rozhraní, které je cílem navazující diplomové práce [4].

1.2.2 Rozbor zadání

Vymezení zadání a základní rozbor vlastností

Zadání lze rozdělit podle bodů do tří skupin. V první, teoretické části, jsou uvedené informace o metodách prohledávání. Vzhledem ke konkurenčnímu prostředí na poli vyhledávačů jsou uvolněné informace pouze částečné, jejich shrnutí obsahuje práce [4]. Některé postupy byly získány metodou inverzního inženýrství.

Druhá část se zabývá pravidly pro mapování sítě. Zde lze diskutovat jednotlivé formáty webových stránek a pravidla vycházející ze standardů těchto formátů. Dále lze definovat pravidla v rámci standardů na straně serverů, která ale nemusejí být obecně splněna. Do třetice se můžeme zabývat implementací pravidel pro mapování obsahu a jejich realizaci.

Část třetí, samotná realizace, je klíčovou částí této práce. Ze zadání implicitně nevyplývá forma realizace, můžeme uvažovat samostatně fungující aplikaci a nebo alternativně modul do systému vzájemně provázaných aplikací.

První varianta pokrývá jednoúčelovou aplikaci, která nejprve stáhne, příp. aktualizuje již stažené webové stránky. Na základě stažených dat provede transformaci z formátů webových stránek na SQL dotazy operujícími nad navrženou databází. V poslední fázi aplikace tyto SQL dotazy zašle databázovému stroji. Navazující webové rozhraní, přes které jsou výsledky zobrazovány, přistupuje výhradně do databáze.

Toto řešení plně odpovídá zadání a je jeho nejjednodušší realizací. Nevýhodou této varianty je netriviální (bezpečné) ovládání aplikace z webového rozhraní, požadavek na multiuživatelské prostředí (tj. více uživatelů může zaslat požadavek na stažení či aktualizaci). Z hlediska programování se jedná o nativní kód použitelný pro tuto konkrétní úlohu, který nebude možné opakovaně použít.

Alternativním řešením, které pokrývá výše uvedené nevýhody řešení prvního, je návrh modulárního systému s centrální správou, vlastní aplikace pak bude modul tohoto systému. Modulární systém je postaven na databázi, tj. ovládání modulů je ekvivalentní úpravám záznamů v tabulkách databáze, jednotlivé požadavky jsou řazeny do prioritní fonty, čímž je zaručeno spouštění pouze jedné instance algoritmu. To vede k odolnosti proti útokům typu odepření (přetížení) služby, tzv. DoS útok (Denial of Service).

Systém je navržen obecně pro jakoukoli problematiku, je jednoduše konfigurovatelný a využívá možností formátu XML a transformací XSLT. Odpadá nutnost psát vše v nativním kódu (akční části aplikace zůstávají v nativním kódu), logika systému je implementována pomocí shellu, čímž zvyšujeme čitelnost algoritmů. Nevýhodou systému je doba prodlení na jednotlivé mezioperace, které by u první varianty nebyly nutné (navíc nativní kód bude vždy rychlejší).

Z nastíněných důvodů je implementována druhá varianta, ač ji zadání nepožaduje. Popis funkčnosti tohoto systému je uveden v kapitole 4.

Realizace je navržena tak, aby byl minimalizován nativní kód, a tam, kde je to možné, je použito standardů (implementace algoritmů nad daty jsou v dotazovacím jazyku SQL, systém používá XML technologii pro vnitřní i vnější komunikaci a XSLT transformací). Některé dílčí aspekty práce byly pokryty volně dostupnými GNU nástroji.

2 Rozbor problematiky

Návrh systému, použité technologie

2.1 Schema systému

Úvodní popis systému a problematiky crawleru

Práce se zabývá analýzou stránek z webových serverů. Abychom mohli stránky analyzovat, potřebujeme je nejprve stáhnout a převést do analyzovatelné formy (v našem případě do databáze). Tento převod nazýváme crawlerizací, uložení těchto dat do formy pak indexací webových stránek. Nástroj pro crawlerizaci nazýváme crawlerem.

V tomto úvodním odstavci jen nastíníme vybrané části problematiky, detailně je crawler popsán v 5. kapitole. Vedle obecných informací není žádná literatura dostupná, implementace crawlerů je předmětem obchodního tajemství jednotlivých technologií.

Předpokládejme, že webové stránky máme staženy nebo aktualizovány z cache. Webovou stránkou obecně rozumíme dokument libovolného typu prezentovaný na webovém serveru, tedy vedle nejčastějších HTML stránek i formáty PDF, Postscript, prostý text a další. Abychom mohli dále tyto stránky různých formátů analyzovat jednotným způsobem, je nutné data stránek pomocí filtrů převést na jednotný formát a dále pracovat s tímto formátem.

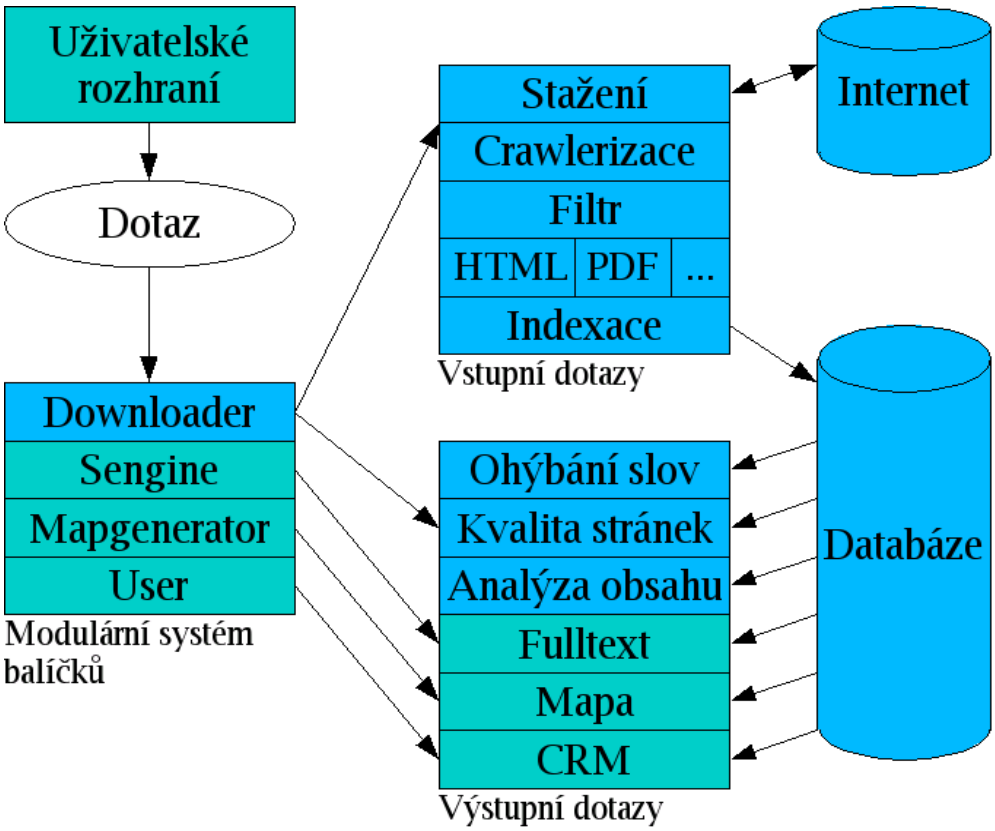
Ukládání dat v tomto formátu (a prakticky zpřístupnění nových informací systému) se nazývá indexace. V našem konkrétním případě představuje SQL dotazy pro databázový server.

S naindexovanými daty lze v rámci možností databázového serveru a databázového modelu provádět další analýzy, např. ohýbání slov (kapitola 6.1), ohodnocení stránek (kapitola 6.2), hledání prefixů a postfixů slov (kapitola 6.3). Dotazy pro fulltextové dotazy a generování map jsou předmětem navazující práce [4].

Podle odstavce 1.2.3 zavádíme pro zpracování dotazů modulární systém balíčků. Balíček udává předpis, jakým způsobem bude dotaz zpracován. Každý balíček běží v separátním procesu a může najednou zpracovávat pouze jeden dotaz.

Webové rozhraní pro zadávání dotazů a prezentaci výsledků dotazů je předmětem citované navazující práce.

Celý systém je zjednodušeně schématicky naznačen na obrázku 2.1, modře označené bloky jsou v dalším textu detailně rozebrány.



Obr 2.1. Zjednodušené schema systému

2.2 Metody generování map

Definice pojmu informační mapa, použití map v praxi.

V dalším textu budeme slovem *mapa* rozumět obecně graf, který bude popisovat vybranou množinu vlastností. Uzly grafu představují stránky, hrany linky mezi stránkami. Účelem mapy je globalizovat informaci obsaženou na stránce a tuto informaci prezentovat v rámci nějakého celku. Mapa má navést uživatele na požadovanou informaci.

Z principu lze mapy rozdělit na mapy tvořené podle obsahu a na mapy tvořené meta informacemi. Meta informacemi rozumíme informace mimo obsah, tedy informace popisující obsah, prakticky mezi ně řadíme klíčová slova stránky, popis stránky a hypertextové odkazy včetně jejich popisů.

Diskutujme nyní první možnost, mapy generované podle obsahu. Tyto mapy jsou určeny k vyhledávání stránek podle obsahu. Dělíme je na mapy se strukturou pevně definovanou nebo nedefinovanou.

Mapy s definovanou strukturou nazýváme katalogy. Katalog lze reprezentovat pomocí stromového grafu. Kořen stromu představuje celý katalog, uzly spojené s kořenovým uzlem se nazývají kategorie příp. sekce. Každá kategorie může obsahovat libovolný počet podkategorií (podsekcí). Podkategorie může obsahovat vedle odkazů na příslušné stránky (které jsou listy grafu) i další podkategorie. Kategorie i podkategorie jsou pevně určeny. Katalog zajišťuje, že stránky s podobným tématem budou v jedné kategorii. Uživatel, který chce svoje stránky zařadit do katalogu, si vybere příslušnou kategorii a podsekci, zvolí popis stránky a uvede adresu úvodní stránky (homepage). Některé katalogy zařadí stránky operativně, některé až po schválení správcem

katalogu. Mezi českými katalogy jmenujme seznam.cz, centrum.cz a atlas.cz, ze zahraničních pak yahoo.com.

Speciálním případem katalogu je i Open Directory Project (http://dmoz.org), kdy každá kategorie má svého správce, který je odpovědný za obsah celé kategorie včetně podkategorií (které mohou mít obecně již jiného správce). Úkolem správce je zařadit a udržovat seznam stránek, příp. i podkategorie. Uživatel, který chce stránky v dané kategorii zaregistrovat, kontaktuje správce kategorie a ten zváží zařazení stránky do své kategorie [11].

Mapy generované obsahem bez pevné struktury jsou odpovědi fulltextových strojů. Fulltextovému stroji zadáme klíčová slova, podle kterých chceme vyhledávat a výsledkem je strom, jehož kořen obsahuje formulaci dotazu a listy jsou nějakým algoritmem ohodnocené stránky, které vyhovují zadanému dotazu. Strom se zpravidla zobrazuje po částech. Registraci stránek většinou není potřeba explicitně provádět [12], fulltextové stroje indexují obsah stránek podle odkazů ze stránek jiných, tedy postačující podmínkou je existnece odkazu z nějaké již naindexované stránky. Narozdíl od katalogů, kde je uvedena úvodní stránka, listy obsahují stránky kdekoliv na webovém serveru (příp. různé stránky na shodném serveru bývají uvedeny v jednom bloku). Příkladem fulltetových vyhledávačů jsou servery seznam.cz, centrum.cz a atlas.cz, ze zahraničních např. google.com. Posledně jmenovaný obsahuje i možnost hledání ve výsledcích, která by prakticky znamenala přidání uzlů mezi kořen a listy grafu, ovšem je realizována pomocí úpravy dotazu, takže výsledek je reprezentován pomocí shodné struktury, jako by se jednalo o klasický dotaz.

Zabývejme se nyní mapami generovanými pomocí meta informací, především linků. Podobný princip užívá i program SiteExpert [13] (trial verze 5.1 z roku 2002) pro generování map na lokálním disku.

První možností je mapa linků, která zobrazuje u každé stránky všechny linky. Vykreslení této mapy (tak, aby byla alespoň trochu čitelná) je výpočetně složité, problém lze přirovnat k problému obchodního cestujícího a ten je NP-úplný. Tuto mapu mohou využít pouze webmasteři, pro návštěvníka stránek obsahuje velké množství informací, které nepotřebuje.

Počet hran lze snížit pomocí hierarchické mapy. Ze známé úvodní stránky (kořen) vybereme pouze ty odkazy (hrany), které odpovídají stránkám (uzlům), jejichž uzly ještě nejsou v grafu uvedeny. Pro každou takto přidanou stránku analyzujeme odkazy a vybíráme pouze ty, které obsahují stránku v grafu ještě neobsaženou. Tento algoritmus je již polynomiálně složitý vůči počtu stránek.

2.3 Pravidla pro mapování webových stránek

Analýza pravidel a jejich použití.

Rozdělme všechna pravidla, která přicházejí v úvahu, na pravidla na straně webových serverů a na pravidla na straně systému, tato pravidla dále na pravidla na vstupní straně systému a na straně výstupní. Pravidla na vstupní straně budou na informace aplikována před uložením do databáze, pravidla na výstupní straně před prezentací mapy.

Pravidla obecně jsou nástroje, které mají pomoci optimalizovat činnost systému tak, aby odpovídala požadavkům uživatele systému a zároveň, aby optimalizovala prováděné operace. Pravidla na straně webových serverů nelze v praxi předpokládat, protože se zabýváme analýzou již existujícího obsahu. Tato pravidla se tedy omezují na obecná pravidla daná normami pro formát přenosu, normy HTML [8] (včetně klíčových slov a popisu obsahu stránky) a norma pro HTTP [9]. Připustíme-li možnost modifikace obsahu na webovém serveru, jednalo by se o pravidla uložená pomocí metainformací v hlavičce HTML webových stránek (a tedy by tato pravidla byla specifikována jen u stránek tohoto formátu). Tuto variantu navržený systém nepoužívá.

Pravidla na vstupní straně systému mají za úkol omezit množství získávaných informací a vyloučit duplicity. Množství informací lze omezit výběrem jen některých stránek, např. podle typu obsahu nebo podle domény, ze které jsou stránky staženy. Další omezení množství informací

spočívá v aktualizaci, tedy u každé stránky (je-li tato informace k dispozici) je zjištěna doba poslední modifikace a je-li tato doba shodná s dobou uloženou v systému, lze tuto stránku přeskočit, protože je již naindexována. Posledním typem pravidel jsou pravidla, která normalizují tvar URL adresy, příp. analyzují parametry u dynamicky generovaných stránek. Tato pravidla jsou detailně popsána v dalším textu.

Typ pravidel na výstupní straně systému závisí na typu generované informační mapy. Obecně lze uvažovat pravidla pro sumaci některých typů obsahu (tedy např. všechny obrázky), pravidla na obsah linků (hierarchická mapa) nebo slov (katalog), někdy je vhodné omezit výstup i adresářovou strukturou webových stránek. Návrh a diskuze těchto pravidel je mimo rozsah této práce a je uvedena v navazující práci [4].

2.4 Použité technologie

Diskuze použitých technologií

V tomto odstavci diskutujme volbu operačního systému, databázového serveru a dalších podpůrných technologií.

Při výběru operačního systému byly uvažovány dvě alternativy, systém Microsoft Windows 2000 nebo libovolná linuxová distribuce. Systém byl navržen jako multiplatformní, lze jej zkompilovat na libovolném operačním systému. Z důvodu jednodušší podpory meziprocesní synchronizace dat (metoda pojmenovaných rour [1]) a cenovým podmínkám byla zvolena druhá varianta, systém byl testován na distribucích SuSE (verze 8.2 a 9), Mandrake (verze 9) a Red Hat (verze 7.1).

Volba databázového serveru odpovídala volbě operačního systému. Na platformě Microsoft Windows byl vybrán Microsoft SQL Server, na linuxové platformě server PostgreSQL, server MySQL (dostupný na obou platformách) nebyl uvažován z důvodu absence možnosti implementace vlastních funkcí. Volbou operačního systému byla implikována volba databázového serveru PostGres (verzi 7.3) [5]. Výhodou této volby je i možnost implementace vlastních funkcí nejen v jazyku odvozeném od SQL - konkrétně PL/pgSQL [7], ale i přímo v jazyce C [6], což umožňuje SQL jazyk pomocí funkcí rozšířit o libovolné vlastnosti. Server rovněž umí pracovat s tabulkovými funkcemi. Instalace serveru je popsána v příloze.

Mezi další použité technologie patří XML a XSLT, které jsou detailněji popsané v následujícím odstavci. Pro naše řešení byl použit XSLT procesor Sablotron standardně dodávaný v distribucích. Poslední použitou technologií je nástroj wget pro stahování obsahu z webových serevů, rovněž standardně dodávaný.

2.5 Technologie XML a XSLT

Základní informace o technologii XML a zpracování dokumentů

Navržený systém hojně využívá technologie XML a transformací XSLT. V tomto odstavci jsou uvedeny základní informace o této technologii. Čtenář znalý problematiky alespoň v rámci [2] nechť tento odstavec přeskočí.

Formát XML (eXtensible Markup Language) je normalizovaný [10] textový formát pro výměnu informací mezi aplikacemi, je rozšířením jazyka SGML. Informace jsou uspořádány do stromové struktury, jsou rozděleny na *elementy* odpovídající listům stromu, element může obsahovat další elementy nebo text, případně kombinaci obého. Navíc u každého elementu je možné specifikovat hodnoty *atributů*.

Element je ohraničen počátečním a koncovým *tagem*. Tag je uvozen znakem '<', následuje

jméno tagu a ukončen znakem '>'. U počátečního tagu lze za jménem tagu a bílým znakem specifikovat atributy a jejich hodnoty, dvojice jsou odděleny mezerou, za jménem atributu je uveden znak '=' následovaný hodnotou atributu uvozenou v jednoduchých nebo dvojitých uvozovkách. Jména tagů a atributů jsou case-sensitive, tj. záleží na velikosti písmen. Koncový tag obsahuje před jménem tagu znak '/'. Je-li počáteční a koncový tag identický, tj. mezi tagy není žádný další tag ani text, lze zápis tagu zkrátit, po uvedení případných atributů je na konec tagu přidán znak '/'.

Ilustrujme předcházející odstavec příkladem. Mějme položku adresáře osob uchovanou pomocí formátu XML. Každý element polozka obsahuje jeden element osoba, který dále obsahuje elementy jmeno a prijmeni a několik elementu adresa s atributem typ a vnořenými elementy ulice a mesto. Element mesto obsahuje atribut psc.

```
<polozka>
        <osoba>
                <jmeno>Karel</jmeno>
                <prijmeni>Novák</prijmeni>
        </osoba>
        <adresa typ='domu'>
                <ulice>U pomníku 123</ulice>
                <mesto psc='123 45'>Nová Ves</mesto>
        </adresa>
</polozka>
```

Má-li každý počáteční tag odpovídající koncový tag (tj. všechny elementy jsou uzavřené) a jednotlivé elementy jsou uzavírány v inverzím pořadí, než byly otvírány počátečními tagy (tj. platnost elementů nelze křížit - nebyla by zachována hierarchická struktura), XML dokument obsahuje pouze jeden kořenový tag a všechny tagy odpovídají popisu z předešlého textu, při splnění všech těchto podmínek dokument nazýváme *XML validní*.

V rámci XML můžeme definovat libovolné tagy. Pro zjednodušení analýzy je žádoucí pro tagy specifikovat pravidla. Pravidla mohou být typu element může/musí/nemusí obsahovat nepovinně/jeden/alespoň jeden element nebo text, u elementů navíc lze specifikovat přípustné povinné/nepovinné atributy. K tomu slouží DTD (Document Type Definition) nebo XML schémata. Odpovídá-li daný dokument pravidlům, označujeme jej za *DTD, nebo schématicky validní.*

Nespornou výhodou XML formátu je čitelnost dokumentu jako textového souboru. XML dokument obsahuje pouze data, nikoli informace o jejich formátování. XML formát je vhodný k veřejné reprezentaci dat, protože každý uživatel si může XML soubor sám analyzovat, neexistují tedy omezení vyplývající z používání proprietárních, většinou ani nestandardizovaných, formátů jedné konkrétní aplikace.

XML dokumenty se zpracovávají pomocí analyzátorů, které podle principu je dělíme na dva druhy, SAX (Simple API for XML) a DOM (Document Object Model).

SAX analyzátor je řízen událostmi. Při otevření analyzovaného XML dokumentu je vyvolána událost documentBegin. Podle struktury následují další události, pro ukázku jmenujme detekování počátečního nebo koncového tagu elementu včetně atributů v případě počátečního tagu, textu nebo konce dokumentu. Každá taková událost je analyzátorem ošetřena. Výhodou SAX parseru je, že umožňuje proudové zpracování dokumentu (tj. znak po znaku, bez nutnosti ukládat celý dokument do paměti). Používá se pro analýzu rozsáhlých dokumentů, klasické použití je pro vyhledávání relevantních informací v rozsáhlém dokumentu.

Naopak DOM analyzátory využívají hierarchické struktury elementů, podle XML dokumentu vytvoří v paměti daty instanciované objekty a další operace jsou prováděny nad touto objektovou strukturou. Výhodou je rychlost přístupu, nevýhodou paměťová náročnost u rozsáhlých dokumentů.

Speciálním případem DOM analyzátoru jsou XSLT procesory. XSLT procesor zpracovává XML dokument podle šablony, která je popsána jiným XML dokumentem (XSLT šablonou). Tím je možné transformovat XML dokumenty buď na XML dokumenty s jinou strukturou nebo na libovolný textem reprezentovatelný formát (např. PDF, PostScript, HTML).

Ilustrativním příkladem je situace, kdy potřebujeme výstupní data jedné aplikace převést na vstup aplikace jiné, přičemž vstupní a výstupní struktury dat nejsou totožné. Budou-li výstupní data

ve formátu XML, použitím XSLT procesoru je převedeme na formát vstupních dat druhé aplikace a problematika je vyřešena. U proprietárních formátů s chybějící dokumentací je tato úloha téměř neřešitelná.

O problematice XML bylo napsáno mnoho tlustých knih, tuto kapitolu je nutné chápat jako stručné nutné minimum pro pochopení následně popsaného systému, který této technologie hojně využívá.

3 Popis databázového modelu

Model systému, popis mechanizmu zpracování dotazů v databázi a databázový model.

V této kapitole navrhneme implementaci systému jako celku a datové struktury popsané pomocí databázového modelu.

3.1 Popis modelu, požadavky

Při návrhu struktury systému je nutné vedle návrhu struktur rovněž navrhnout a zahrnout omezení, která do datového modelu přináší použití webového rozhraní. Z implementačního hlediska musíme rozhodnout, které operace nad databázovými strukturami lze provést v reálném čase, a které nikoli. Pro časovou optimalizaci je vhodné navrhnout cache výsledků, protože některé operace předpokládáme značně časově náročné. Budeme pracovat s velkými objemy dat, je vhodné navrhnout strukturu, která splňuje co nejvyšší třídu normalizace.

Nyní rozebereme výše uvedené požadavky podrobněji. Uvažujme použití webového rozhraní. Všechny dotazy nad databázovým modelem budou zadávány koncovým uživatelem pomocí webového rozhraní a výsledek bude zobrazen v různých formátech opět přes toto rozhraní. Uvážíme-li množství zpracovávaných dat a fakt, že systém navrhujeme jako centrální, některé operace mohou trvat delší dobu, nežli je očekáváno koncovým uživatelem. Z těchto důvodů navrhujeme efektivně využívanou cache. Dalším důvodem je i předpoklad opakování se zadávaných dotazů.

Pokud bychom umožnili libovolnému uživateli zadávat časově neomezeně náročné dotazy, systém by nebylo těžké přetížit, což by vedlo k náchylnosti na útok přetížení a následné odmítnutí služby (DoS). Systém by proto měl obsluhovat v jednom časovém okamžiku omezený počet dotazů. Počet dotazů lze omezit v nastavení databázového serveru, avšak při tomto řešení ztrácíme možnost provádět časově nenáročné databázové dotazy v případě, kdy již budou všechna možná připojení k databázovému serveru obsazena. Jako vhodnější varianta se jeví možnost uživatelův dotaz uložit do fronty dotazů a pak jej provést podle dostupnosti prostředků a nastavených priorit. Výsledek dotazu bude uložen do cache a uživatel bude o dostupnosti výsledku vyrozuměn. Tento postup omezuje možnost zahlcení systému vstupními daty.

Pokud rozdělíme dotazy do balíčků a pro každý balíček definujeme vlastní frontu, dosáhneme možnosti paralelizace systému. Navíc se zavedením balíčků zjednoduší konfigurace a administrace dotazů.

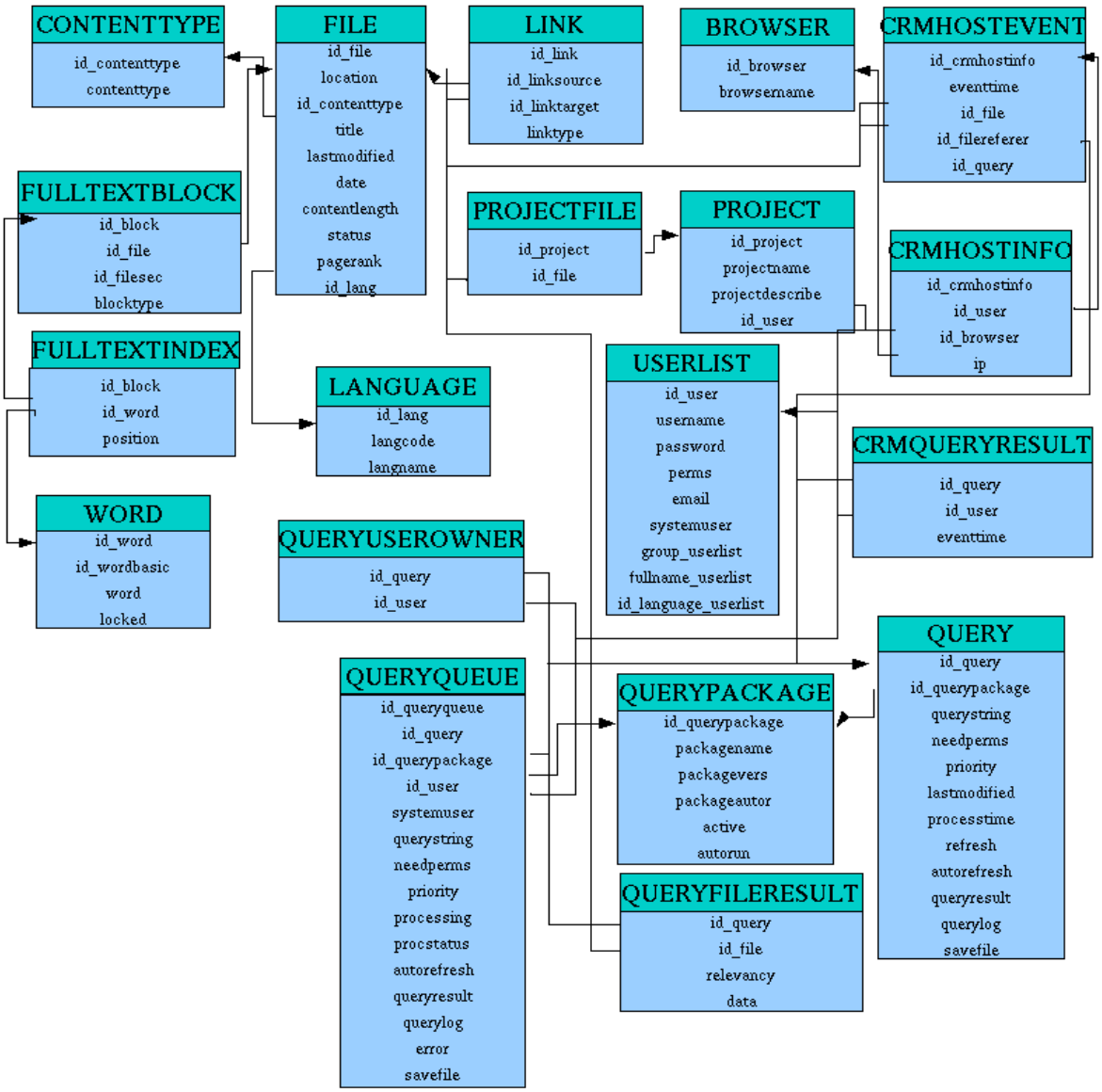
Z tématického uspořádání tabulek rozdělujeme databázový model rozdělit na 5 logických bloků.

- **Blok URI adres** Blok obsahuje informace o každé URI adrese a o obsahu na této adrese. Hypertextové odkazy mezi jednotlivými stránkami jsou uloženy v tabulce linků. Součástí bloku je i číselník typů obsahu (content-type).
- **Uživatelské rozhraní** Budeme-li uvažovat přístup k multiuživatelskému rozhraní, navrhneme autentifikaci uživatelů. Uživatele bude vhodné rozdělit do skupin. První skupinu označíme hosté; tito uživatelé budou moci zobrazovat výsledky nenáročných dotazů. Druhou skupinou budou uživatelé, kteří navíc budou připravovat dotazy složitějšího charakteru (download, mapy), budou mít plný přístup k webovému rozhraní (resp. přístup může být v principu omezen na některé komponenty webového rozhraní). Třetí skupinou uživatelů budou systémoví démoni, kteří poběží na pozadí s nízkou prioritou a budou automaticky provádět aktualizační dotazy, příp. další předvolené dotazy.
- **Blok dotazů** Blok tvoří specifikace balíčků dotazů, tabulka dotazů, prioritní fronta dotazů jednotlivých balíčků a cache výsledků dotazů. Každý uživatel podle příslušného oprávnění

může zadávat dotazy a vybrané dotazy zařadit do svých oblíbených (obecně dotazy, o kterých chce mít přehled). Každý dotaz je přiřazen některému z balíčků. Fronta dotazů je rozšíření tabulky dotazů o atributy související se zpracováním, umožňuje vložit dotaz, který není uložen v tabulce dotazů. Výsledky jednotlivých dotazů jsou uloženy v tabulce dotazů, obsahují log a výsledek dotazu ve formátu XML, v některých případech je vhodné navrhnout reprezentaci výsledku dotazu přímo v databázi.

- **Fulltextová báze dat** Skládá se z tabulky slov (slovník), bloků stránek a indexu. Obsah stránky je rozdělen do bloků podle významu (ten odpovídá skupinám elementů H1-H3, TH a TD, LI, P a DIV a metatag keywords a description definovaných v jazyce HTML). Tyto bloky jsou indexovány. Index je relace M:N mezi slovníkem a blokem.
- **CRM blok** Blok sleduje činnost koncového uživatele a ukládá ji do tabulek. Získaná data se dají využít pro zpřesnění výsledku z pohledu uživatele. Vyšší četnost zobrazení dotazu může vést k jeho častější aktualizaci nebo další, podrobnější, analýze. Četnost výběru daného linku daného dotazu může vést ke zvýšení jeho relevance.

Logický model celého systému je uveden na následujícím obrázku.



Obr 3.1. Logický model systému

3.2 Blok URL informací

3.2.a Informace o obsahu URI adresy

Tabulka FILE a obslužné funkce

V tabulce FILE jsou uvedeny klíčové informace o obsahu na URI adrese. Záznam má svoje identifikační číslo (atribut id_file) a URI adresu (location). Pro vnitřní zpracování obsahuje atribut status, který nabývá jedné z hodnot E (Error), L (Loaded), I (Indexed), R (Registered), F (403/AccessForbidden), N (404/NotFound), i (Ignored), e (Chyba crawleru) a E (jiná chyba). Typ obsahu specifikuje atribut id_contenttype, titulek stránky je uveden v atributu title (duplicitně vůči fulltextové bázi dat). Dalšími uchovávanými atributy jsou hodnoty z HTTP hlavičky lastmodified, date a contentlength. Lze-li zjistit, atribut id_lang obsahuje identifikátor jazyka dokumentu. U každého záznamu je uveden i atribut pagerank, který udává hodnotu PageRanku dané stránky.

SQL dotaz pro vytvoření tabulky FILE

```
CREATE SEQUENCE seq_file START 1 INCREMENT 1;
CREATE TABLE file (
            id_file int4 UNIQUE NOT NULL DEFAULT nextval('seq_file'),
            location varchar(255) UNIQUE NOT NULL,
            id_contenttype int2,
            title varchar(255),
            lastmodified timestamp,
            date timestamp,
            contentlength int4,
            status char DEFAULT 'R',
            pagerank float,
            id_lang int2 REFERENCES language,
      PRIMARY KEY (id_file));
```

Definujeme index varcející funkci integer indexoffile(text), která vrátí hodnotu primárního klíče pro stránku se zadanou URI adresou. Pokud odpovídající záznam neexistuje, je vytvořen nový záznam s implicitními hodnotami ostatních atributů a je vrácena hodnota primárního klíče tohoto záznamu.

Funkce integer indexoffile(text)

```
CREATE FUNCTION indexoffile(text) RETURNS integer AS '
DECLARE
   searchvalue ALIAS FOR $1;
   row         RECORD;
   ret         int4;
BEGIN
      SELECT INTO row id_file AS id FROM file WHERE location=searchvalue;
      IF NOT FOUND THEN
            ret:=nextval("seq_file");
            INSERT INTO file(id_file,location) VALUES(ret,searchvalue);
      ELSE
            ret:=row.id;
      END IF;
      RETURN ret;
END;' LANGUAGE 'plpgsql';
```

Pro tabulku FILE definujeme trigger pro mazání záznamu. Při mazání záznamu budou smazány všechny související záznamy ve fulltextovém bloku, bloku informací o uživateli a všechny záznamy o hypertextových odkazech, kde mazaný záznam je buď zdrojovou nebo cílovou URL adresou.

Trigger pro mazání záznamu z tabulky FILE

```
CREATE FUNCTION trigger_file_delete() RETURNS TRIGGER AS 'BEGIN
      DELETE FROM fulltextblock WHERE id_file=OLD.id_file OR id_file_sec=OLD.id_file;
      DELETE FROM link WHERE link.id_linksource=OLD.id_file OR link.id_linktarget=OLD.id_file;
      DELETE FROM projectfile WHERE id_file=OLD.id_file;
      DELETE FROM queryfileresult WHERE id_file=OLD.id_file;
      RETURN OLD;
END;' LANGUAGE 'plpgsql';

CREATE TRIGGER trigger_file_delete BEFORE DELETE ON file FOR EACH ROW EXECUTE PROCEDURE trigger_file_delete();
```

Dále deklarujeme trigger pro update záznamu. Je-li u starého záznamu status souboru nastaven na indexovaný (I) nebo nahraný (L) a dojde-li ke změně hodnoty tohoto statusu, jsou zároveň smazány všechny odkazy vedoucí z toho souboru a všechny fulltextové bloky, které mají tento soubor jako zdrojový.

Trigger pro aktualizaci tabulky FILE

```
CREATE FUNCTION trigger_file_update() RETURNS TRIGGER AS 'BEGIN
IF OLD.status IN ("L","I") AND ((OLD.lastmodified IS NOT NULL AND OLD.lastmodified<NEW.lastmodified)
            OR OLD.status<>NEW.status) THEN
            RAISE NOTICE "Deleting child records";
            DELETE FROM fulltextblock WHERE (id_file=OLD.id_file AND id_file_sec IS NULL) OR id_file_sec=OLD.id_file;
            DELETE FROM link WHERE link.id_linksource=OLD.id_file;
      END IF;
      RETURN NEW;
END;' LANGUAGE 'plpgsql';

CREATE TRIGGER trigger_file_update BEFORE UPDATE ON file FOR EACH ROW EXECUTE PROCEDURE trigger_file_update();
```

3.2.b Číselník typu obsahu

Tabulka CONTENTTYPE a obslužné funkce

Tabulka CONTENTTYPE je číselník typů obsahu souboru. Primárním klíčem tabulky je identifikační číslo typu (atribut id_contenttype), samotný název typu je pak uložen v atributu contenttype.

SQL dotaz pro vytvoření tabulky CONTENTTYPE

```
CREATE SEQUENCE seq_contenttype START 1 INCREMENT 1;
CREATE TABLE contenttype (
            id_contenttype int2 not null unique,
            contenttype varchar(48) not null,
            PRIMARY KEY (id_contenttype));
```

Obslužnou funkcí je index vracející funkce integer indexofcontenttype(text), která vrací hodnotu primárního klíče odpovídající typu obsahu (parametr funkce).

Funkce integer indexofcontenttype(text)

```
CREATE FUNCTION indexofcontenttype (text) RETURNS int2 AS 'DECLARE
   searchvalue ALIAS FOR $1;
   ret        int2;
   row         RECORD;
BEGIN
      SELECT INTO row id_contenttype AS id FROM contenttype WHERE contenttype=searchvalue;
      IF NOT FOUND THEN
            ret:=nextval("seq_contenttype");
            INSERT INTO contenttype(id_contenttype,contenttype) VALUES(ret,searchvalue);
      ELSE
            ret:=row.id;
      END IF;
      RETURN ret;
END;' LANGUAGE 'plpgsql';
```

3.2.c Hypertextové odkazy

Tabulka LINK a obslužná funkce

Mezi jednotlivými stránkami mohou existovat linky - hypertextové odkazy. Typy linků (atribut linktype) odvozujeme z množiny typů linků, které známe z jazyka HTML. Z této množiny vybíráme typy ahref (A), img (I), a link (L) a script (S), zpracování formulářů, appletů a analýza skriptovacích jazyků (javascipt, vbscript) není podporována. Typ linku je uložen v atributu linktype. Link chápeme jako hranu orientovaného grafu, přičemž počáteční vrchol je specifikován pomocí atributu id_linksource (identifikátor zdrojové URI adresy) a koncový vrchol (identifikátor cílová adresa) pomocí atributu id_linktarget.

SQL dotaz pro vytvoření tabulky LINK

```
CREATE TABLE link (
        id_link serial unique,
        id_linksource int4 REFERENCES file NOT NULL,
        id_linktarget int4 REFERENCES file NOT NULL,
        linktype char NOT NULL,
        PRIMARY KEY (id_link));
```

Pro optimalizaci vyhledávání v tabulce linků definujeme index na atributy id_linksource a id_linktarget.

Index pro tabulku LINK

```
CREATE INDEX idx_link ON link USING btree (id_linksource,id_linktarget);
```

Deklarujeme funkci text addlink(int4 source,text target, char type, int typedescription, text description, int typetitle, text title), která vloží záznam do tabulky LINK a vrátí zprávu ve formátu XML. Parametry funkce jsou identifikátor zdrojové a cílové URL adresy, typ linku a obsah popisu a textu odkazu.

Funkce text addlink(int4 idsource,text target, char type, int typedescription, text description, int typetitle, text title)

```
CREATE FUNCTION addlink(int4,text,char,integer,text,integer,text) RETURNS text AS '
DECLARE
   idsource    ALIAS FOR $1;
   target      ALIAS FOR $2;
   linktp      ALIAS FOR $3;
   tplinkdesc  ALIAS FOR $4;
   linkdesc    ALIAS FOR $5;
   tplinktit   ALIAS FOR $6;
   linktitle   ALIAS FOR $7;

   idtarget    int4;
   row         RECORD;
BEGIN
   SELECT INTO row urlto FROM temp_urltransform WHERE urlfrom=target;
   IF NOT FOUND THEN
      idtarget:=indexoffile(target);
   ELSE
      idtarget:=indexoffile(row.urlto);
   END IF;
   INSERT INTO link(id_linksource,id_linktarget,linktype) VALUES (idsource,idtarget,linktp);

   RETURN "<link target=""||idtarget||"">"|| CASE WHEN linkdesc IS NOT NULL THEN
fulltext_addblock(idtarget,idsource,tplinkdesc,linkdesc) ELSE "" END || CASE WHEN linktitle IS NOT NULL THEN
fulltext_addblock(idtarget,idsource,tplinktit,linktitle) ELSE "" END || "</link>";
END; ' LANGUAGE 'plpgsql';
```

3.3 Tabulky uživatelského prostředí

3.3.a Správa uživatele

Tabulka USERLIST a obslužné funkce

Tabulka USERLIST reprezentuje uživatele. Každý uživatel má pomocí bitové masky definována práva (atribut perms) na jednotlivé komponenty webového rozhraní. Uživatel se do systému hlásí pomocí uživatelského jména (username) a hesla (password). Heslo je z důvodu bezpečnosti zakódováno pomocí algoritmu MD5. V některých případech je vhodné uvést svůj kontaktní email (email) a plné jméno (fullname_userlist). Pro potřeby webového rozhraní dále atributy určující skupinu (group_userlist), do které uživatel patří a jazyková mutace webového rozhraní(id_language_userlist). Systémový démon má nastaven atribut systemuser.

SQL dotaz pro vytvoření tabulky USER

```
CREATE SEQUENCE seq_userlist START 1 INCREMENT 1;
CREATE TABLE userlist (
        id_user int2 UNIQUE NOT NULL DEFAULT nextval('seq_userlist'),
        username varchar(32) UNIQUE NOT NULL,
        password char(32),
        perms int2 NOT NULL DEFAULT 1,
        email char(64),
        systemuser bool NOT NULL DEFAULT false,
        group_userlist varchar(20),
        fullname_userlist varchar(200),
        id_language_userlist int2 REFERENCES language,
    PRIMARY KEY (id_user));
```

Pro zpřehlednění zápisu SQL dotazů využívající jméno uživatele definujeme funkci integer indexofuser(text). Na rozdíl od předchozích variant funkcí je nežádoucí, aby v případě neexistence záznamu byl vytvořen nový uživatelský účet (záznam v tabulce).

Funkce int2 indexofuser(text)

```
CREATE FUNCTION indexofuser (text) RETURNS int2 AS '
        SELECT id_user AS id FROM userlist WHERE username = $1;' LANGUAGE 'sql';
```

3.3.b Správa projektu

Tabulky PROJECT, PROJECTFILE a obslužné funkce

Každý uživatel může založit a následně spravovat jeden nebo více projektů. Projekt je určen primárním klíčem id_project a unikátním jménem projektu (projectname). Informace o vlastníkovi projektu je uložena v atributu id_user, atribut projectdescribe obsahuje nepovinný popis projektu. Projekt slouží ke specifikování skupiny stránek.

SQL dotaz pro vytvoření tabulky PROJECT

```
CREATE TABLE project (
        id_project serial UNIQUE NOT NULL,
        projectname varchar(255) UNIQUE NOT NULL,
        id_user int2 REFERENCES userlist NOT NULL,
        projectdescribe text,
        PRIMARY KEY (id_project));
```

Nadeklarujme obslužnou index vracející funkci integer indexofproject(text), která vrátí identifikaní číslo k zadanému jménu projektu. Pokud nevyhovuje žádný záznam, funkce vrátí NULL.

Funkce integer indexofproject(text)

```
CREATE FUNCTION indexofproject (text) RETURNS int4 AS '
        SELECT id_project AS id FROM project WHERE projectname = $1;' LANGUAGE 'sql';
```

Každý soubor může, ale nemusí, patřit do jednoho projektu nebo více projektů. Tato relace je reprezentována pomocí tabulky PROJECTFILE. Jedná se o relaci N:M mezi tabulkami FILE (id_file) a PROJECT (id_project). Této relace je využito při vyhledávání, kdy omezujeme výsledek dotazů jen na soubory z daného projektu.

SQL dotaz pro vytvoření tabulky PROJECTFILE

```
CREATE TABLE projectfile (
        id_file int4 REFERENCES file NOT NULL,
        id_project int4 REFERENCES project NOT NULL,
        PRIMARY KEY (id_file,id_project));
```

3.4 Blok dotazů

Tento blok zajišťuje zadávání, přípravu a obsluhu dotazů, je propojen na procesy jednotlivých balíčků, které provádějí samotný výpočet. Blok zajišťuje paralelizaci výpočtů (každý balíček má vlastní proces na zpracování dotazů, při sdílení disků může běžet i na více strojích), prioritně řazenou frontu a prostředky pro správu, zobrazení a automatickou aktualizaci výsledků dotazů. Vstupní i výstupní data jsou ve formátu XML.

3.4.a Balíčky dotazů

Tabulka QUERYPACKAGE a obslužná funkce

Nadeklarujeme tabulku balíčků dotazů, tabulku QUERYPACKAGE. Tabulka obsahuje základní informace o balíčku; identifikační číslo balíčku (id_querypackage), název balíčku (packagename), verzi balíčku (packagevers) a atribut (autorun) určující, zda balíček bude automaticky zaveden při startu systému, volitelně informaci o autoru balíčku (packageauthor) a stavový atribut active indikující aktivitu balíčku (výkonná jednotka je připravena provádět dotazy tohoto balíčku).

SQL dotaz pro vytvoření tabulky QUERYPACKAGE

```
CREATE SEQUENCE seq_querypackage START 1 MINVALUE 1;
CREATE TABLE querypackage (
        id_querypackage int2 UNIQUE DEFAULT nextval('seq_querypackage'),
        packagename varchar(32) NOT NULL UNIQUE,
        packagevers varchar(16) NOT NULL,
        packageauthor varchar(255),
        active bool DEFAULT false,
        autorun bool DEFAULT false,
        PRIMARY KEY (id_querypackage));
```

Pro tabulku QUERYPACKAGE definujeme index vracející funkci integer indexofquerypackage(text), která k danému jménu balíčku vrátí jeho identifikační číslo.

Funkce int2 indexofquerypackage(text)

```
CREATE FUNCTION indexofquerypackage(text) RETURNS int2 AS '
        SELECT id_querypackage FROM querypackage WHERE packagename = $1;' LANGUAGE 'sql';
```

3.4.b Tabulka dotazů

Tabulka QUERY a obslužné funkce

Definici dotazů umožňuje tabulka QUERY, která rovněž slouží jako cache dotazů. Dotaz je specifikován pomocí identifikačního čísla dotazu (id_query), následuje balíček (id_querypackage), ve kterém bude dotaz zpracován, text dotazu (querystring), bitová maska práv potřebných ke spuštění dotazu (needperms), priorita dotazu ve frontě (priority) a volitelná možnost specifikování souboru (savefile), do kterého bude uložena kopie výsledku dotazu. Následují statistické atributy poslední aktualizace dotazu (lastmodified), doba potřebná k poslední aktualizaci dotazu (processtime) a volitelně nastavitelný interval aktualizace (refresh). Je-li navíc nastaven atribut autorefresh na true, je tento dotaz zařazen do skupiny dotazů, u kterých je doba aktualizace dynamicky vypočítávána podle počtu zobrazení výsledků. Poslední atributy představují cache výsledků, výsledek je rozdělen na vlastní obsah (queryresult) a na log (querylog), obsah obou atributů je ve formátu XML.

SQL dotaz pro vytvoření tabulky QUERY

```
CREATE SEQUENCE seq_query START 1 MINVALUE 1;
CREATE TABLE query (
        id_query int4 UNIQUE DEFAULT nextval('seq_query'),
        id_querypackage int2  NOT NULL REFERENCES querypackage,
        querystring text NOT NULL,
        needperms int8 DEFAULT 1,
        priority int2 DEFAULT 0,
        lastmodified timestamp,
        processtime interval,
        refresh interval,
        autorefresh bool DEFAULT false,
        queryresult text,
        querylog text DEFAULT '',
        savefile varchar(128),
        PRIMARY KEY (id_query));
```

Pro tuto tabulku definujeme trigger, který smaže všechny relevantní záznamy referencující mazaný záznam tabulky.

Trigger pro mazání záznamů z tabulky QUERY

```
CREATE FUNCTION trigger_query_delete() RETURNS TRIGGER AS '
BEGIN
    DELETE FROM crmhostevent WHERE id_query=OLD.id_query;
    DELETE FROM crmqueryresult WHERE id_query=OLD.id_query;
    DELETE FROM queryqueue WHERE id_query=OLD.id_query;
    DELETE FROM queryuserowner WHERE id_query=OLD.id_query;
    RETURN OLD;
END; ' LANGUAGE 'plpgsql';

CREATE TRIGGER trigger_query_delete BEFORE DELETE ON query FOR EACH ROW EXECUTE PROCEDURE
trigger_query_delete();
```

Pro potřeby jednotného přístupu k zadávání dotazů z webového rozhraní definujeme funkci integer insertquery(int2 id_package, text querystring, int8 needperms, integer priority), která vrací identifikační číslo zadaného dotazu. Pokud zadaný dotaz neexistuje v tabulce dotazů, je vložen záznam s vyplněnými hodnotami podle parametrů zadaných funkci. První parametr je číslo balíčku, který bude dotaz zpracovávat, druhý je řetězec obsahující dotaz a následují maska požadovaných práv uživatele k volání dotazu a priorita dotazu.

Funkce integer insertquery(int2 id_package, text querystring, int8 needperms,integer priority)

```
CREATE FUNCTION insertquery(integer,text,int8,integer) RETURNS integer AS '
DECLARE
   inspackage        ALIAS FOR $1;
   insquerystring    ALIAS FOR $2;
   insneedperms      ALIAS FOR $3;
   inspriority       ALIAS FOR $4;
```

```
   row              RECORD;
   ret              integer;
BEGIN
   SELECT INTO row id_query FROM query WHERE querystring=insquerystring;
   IF NOT FOUND THEN
      ret:=nextval("seq_query");
      INSERT INTO query (id_query,id_querypackage,querystring,needperms,priority) VALUES
(ret,inspackage,insquerystring,insneedperms,inspriority);
   ELSE
      ret:=row.id_query;
   END IF;
   RETURN ret;
END;
' LANGUAGE 'plpgsql'
```

K tabulce QUERY nadefinujeme funkci text getqueryresult(int2 iduser, int4 idquery), která vrací výsledek zadaného dotazu ve formátu XML. Během zpracování je provedena kontrola práv (nenulový logický součin bitových masek práv uživatele a dotazu), je-li test vyhodnocen negativně nebo jeden z parametrů je neplatný identifikátor, je vráceno NULL. V kladném případě je uložen záznam o zobrazení dotazu do příslušející CRM tabulky a je vrácen příslušný řetězec.

Funkce text getqueryresult(int2 iduser, int4 idquery)

```
CREATE FUNCTION getqueryresult (integer,integer) RETURNS text AS '
DECLARE
      row RECORD;
BEGIN
      SELECT INTO row query.needperms & userlist.perms = 0 as failture FROM userlist, query WHERE id_user = $1 AND id_query =
$2;
      IF NOT FOUND THEN
            RETURN "NULL - not found";
      END IF;
      IF row.failture THEN
            RETURN "NULL - failture" ;
      END IF;
      INSERT INTO crmqueryresult (id_user,id_query) VALUES ($1,$2);
      SELECT INTO row "<?xml version="1.0" encoding="iso-8859-2"?><senginequery id=""||id_query||"" package=""||packagename||""
packageversion=""||packagevers
            || CASE WHEN lastmodified IS NOT NULL THEN "" lastmodified=""||lastmodified ELSE "" END
            || CASE WHEN processtime IS NOT NULL THEN "" processtime=""||processtime ELSE "" END
            || ">"|| querystring
            || CASE WHEN querylog IS NOT NULL THEN "<log>"||querylog||"</log>" ELSE "" END
            || CASE WHEN queryresult IS NOT NULL THEN "<result>" ||queryresult ||"</result>" ELSE "" END
            || "</senginequery>" AS result
            FROM query NATURAL JOIN querypackage WHERE id_query= $2;
      RETURN row.result;
END;
' LANGUAGE 'plpgsql';
```

3.4.c Vlastnictví dotazu

Tabulka QUERYUSEROWNER

V rámci databázového modelu webové rozhraní umožňuje uživateli vybrat si dotazy, o jejíž výsledcích chce být informován (oblíbené dotazy, statistiky pro uživatelův projekt,...). Pro uchování této relace definujeme tabulku QUERYUSEROWNER představující relaci N:M mezi tabulkami QUERY(id_query) a USERLIST (id_user). Uživatel si může dotaz pojmenovat (atribut queryname).

```
CREATE TABLE queryuserowner (
            id_query int4 REFERENCES query NOT NULL,
            id_user int2 REFERENCES userlist NOT NULL,
            queryname varchar(40),
            PRIMARY KEY (id_user,id_query));
```

3.4.d Fronta dotazů

Tabulka QUERYQUEUE, obslužné funkce a automatické předzpracování dotazů

Fronta dotazů ke zpracování je reprezentována tabulkou QUERYQUEUE. Obsahuje většinu atributů z tabulky QUERY. Dalšími atributy jsou identifikační číslo položky fronty (id_queryqueue) a uživatele zadávajícího dotaz (id_user), atributy pro dočasné uchování výsledku (queryresult, querylog, error), stavové atributy pro indikaci právě prováděného dotazu (processing), flag automatické aktualizace (autorefresh) a indikaci, že dotaz zadal systémový uživatel (systemuser). Poslední atribut udává čas zadání dotazu (není-li dosud zpracováván) nebo započetí výpočtu (starttime).

SQL dotaz pro vytvoření tabulky QUERYQUEUE

```
CREATE SEQUENCE seq_queryqueue START 1 MINVALUE 1 MAXVALUE 32767 CYCLE;
CREATE TABLE queryqueue (
        id_queryqueue int2 UNIQUE NOT NULL DEFAULT nextval('seq_queryqueue'),
        id_query int4,
        id_querypackage int2 NOT NULL REFERENCES querypackage,
        id_user int4 NOT NULL REFERENCES userlist,
        querystring text,
        needperms int8 DEFAULT 1,
        priority int2 DEFAULT 0,
        savefile varchar(64),
        queryresult text DEFAULT '',
        querylog text DEFAULT '',
        error text,
        processing bool DEFAULT false,
        procstatus float,
        autorefresh bool DEFAULT false,
        systemuser bool,
        starttime timestamp DEFAULT now(),
        PRIMARY KEY (id_queryqueue));
```

Popišme nyní možnosti přidání dotazu do fronty. První možností je přidání dotazu registrovaném v tabulce QUERY. Do fronty dotazů postačuje vložit záznam s vyplněnými atributy id_user a id_query, zkopírování ostatních atributů z tabulky QUERY zajistí trigger trigger_queryqueue_insert, který nadeklarujeme později v tomto odstavci. Druhou alternativou je zařazení do fronty dotazu dosud neregistrovaného dotazu. V tomto případě je nutné vložit identifikační číslo uživatele, balíčku dotazů, text dotazu a prioritu, 0 bit masky práv je defaultně nastaven.

První alternativa je realizována pomocí funkce integer queryintoqueue(integer iduser,integer idquery, text savefile), kde prvním parametrem je identifikační číslo uživatele a druhým číslo dotazu a třetí udává jméno souboru, do kterého bude výsledek uložen. Funkce vrací identifikační číslo ve frontě dotazů.

Funkce integer queryintoqueue(integer id_user, integer id_query, varchar(128) savefile)

```
CREATE FUNCTION queryintoqueue(integer,integer,varchar(128)) RETURNS integer AS '
DECLARE
    idqueryqueue    integer;
    iduser          ALIAS FOR $1;
    idquery         ALIAS FOR $2;
    filename        ALIAS FOR $3;
BEGIN
    idqueryqueue:=nextval("seq_queryqueue");
    INSERT INTO queryqueue(id_queryqueue,id_user,id_query,savefile) VALUES (idqueryqueue,iduser,idquery,filename);
    RETURN idqueryqueue;
END;
' LANGUAGE 'plpgsql';
```

Alternativně pro druhou možnost lze použít funkci integer querytoqueue(integer iduser, integer idquerypackage, text querystring, integer priority), maska práv je nastavena podle masky práv uživatele, který dotaz zadal. Funkce vrací identifikátor položky ve frontě dotazů.

Funkce integer querytoqueue(int2 iduser, int2 idquerypackage, text querystring,int8 priority)

```
CREATE FUNCTION queryintoqueue(int2,int2,text,int8) RETURNS integer AS '
DECLARE
    idqueryqueue    integer;
```

```
    iduser              ALIAS FOR $1;
    idquerypackage  ALIAS FOR $2;
    insquerystring      ALIAS FOR $3;
    inspriority             ALIAS FOR $4;
    userperms           RECORD;

BEGIN
    SELECT INTO userperms perms FROM userlist WHERE id_user = iduser;
    IF NOT FOUND THEN
        RETURN NULL;
    END IF;
    idqueryqueue:=nextval("seq_queryqueue");
    INSERT INTO queryqueue(id_queryqueue,id_user,id_querypackage,querystring,priority,needperms)
        VALUES (idqueryqueue,iduser,idquerypackage,insquerystring,inspriority,userperms.perms);
    RETURN idqueryqueue;
END;' LANGUAGE 'plpgsql';
```

Nyní nadefinujeme mechanizmus přijímání dotazů do fronty, který je realizován pomocí triggeru trigger_queryqueue_insert.

V první fázi je zkontrolována identifikace uživatele, neexistuje-li, je do atributu error uloženo chybové hlášení a ukončeno zpracování triggeru. V opačném případě na základě těchto informací je nastavena hodnota atributu systemuser.

Pokud je nastaven atribut id_query, nastav atributy querystring, id_querypackage, priority, needperms a savefile podle záznamu tabulky QUERY.

Následně se provede kontrola oprávnění (součin bitové masky dotazu a uživatele), pokud nevyhovuje (součin ke nulový), je do atributu error uloženo chybové hlášení a ukončeno zpracování triggeru.

Poslední kontrolou je, zda-li je zadán balíček dotazů, který dotaz bude zpracovávat. Pokud je kontrola neúspěšná, je do atributu error uloženo chybové hlášení a ukončeno zpracování triggeru.

Nyní je dotaz úspěšně vložen do fronty.

Trigger pro vkládání do fronty dotazů

```
CREATE FUNCTION trigger_queryqueue_insert() RETURNS trigger AS '
DECLARE
   query      RECORD;
   user       RECORD;
 BEGIN
   RAISE NOTICE "Insert Query";
   SELECT INTO user * FROM userlist WHERE id_user=NEW.id_user;
   IF NOT FOUND THEN
      NEW.error := "Unknown user id";
      RETURN NEW;
   END IF;
   NEW.systemuser := user.systemuser;
   IF NEW.id_query IS NOT NULL THEN
      SELECT INTO query * FROM query WHERE id_query=NEW.id_query;
      IF NOT FOUND THEN
         NEW.error := "Unknown query id";
         RETURN NEW;
      END IF;
      NEW.querystring := query.querystring;
      NEW.id_querypackage := query.id_querypackage;
      NEW.priority := query.priority;
      NEW.needperms := query.needperms;
      NEW.savefile := query.savefile;
   END IF;
   IF NEW.needperms & user.perms = 0 THEN
      NEW.error := "Specified user has not permisions to activate this query";
      RETURN NEW;
   END IF;

   IF NEW.id_querypackage IS NULL THEN
      NEW.error := "No package specified.";
      RETURN NEW;
   END IF;
      RETURN NEW;
END;
' LANGUAGE 'plpgsql';

CREATE TRIGGER trigger_queryqueue_insert BEFORE INSERT ON queryqueue FOR EACH ROW EXECUTE PROCEDURE
trigger_queryqueue_insert();
```

Dalším triggerem je implementován výběr dotazu z vrcholu fronty, tj. dotazu, který bude zpracováván. Tento trigger je volán při vkládání do fronty a při mazání záznamu z fronty (dotaz byl již zpracován a je z fronty smazán).

Trigger zjistí seznam balíčků, které momentálně žádný dotaz nezpracovávají. Pro každý takový balíček se vybere z tabulky QUERYQUEUE záznam, který má nevyplněný atribut error a má nejvyšší prioritu (priorita dána systémovým/nesystémovým uživatelem, číselnou prioritou (maximální nejvyšší), automatickou aktualizací/uživatelským dotazem a časem vložení dotazu, v tomto pořadí).

Pomocí jména balíčku dotazu je vytvořeno jméno pojmenované roury [1] balíček.pipe v dočasném adresáři a do této roury poslán požadavek tvořený hlavičkou a atributem querystring. Atribut starttime je nastaven na aktuální čas a atribut processing je nastaven na true. Pokud není zadáno identifikační číslo dotazu, je do atributu error uložena o tomto zpráva, čím že je zajištěno, že dotaz ve frontě nebude po ukončení smazán a bude možné přečíst výsledek. Tímto zpracování dotazu přebírá výkonná jednotka (viz kapitola 4), která se při ukončení zpracování dotazu pokusí smazat záznam z fronty, čímž je opětovně volán tento trigger.

Pokud je trigger volán při mazání a fronta dotazů pro některý balíček je prázdná, provede se test, zda-li nevypršel časový interval platnosti výsledku automaticky aktualizovaných dotazů. Všechny pozitivní případy dotazů jsou vloženy do fronty s nastaveným atributem autorefresh a uživatelem sengine.

Trigger pro skenování fronty dotazů

```
CREATE FUNCTION trigger_queryqueue_scan() RETURNS trigger AS '
DECLARE
   query      RECORD;
   queryresult RECORD;
   package           RECORD;

BEGIN
   RAISE NOTICE "Scan query ";
   FOR package IN SELECT id_querypackage as id FROM querypackage WHERE active AND id_querypackage NOT IN (SELECT
DISTINCT id_querypackage FROM queryqueue WHERE processing) LOOP
      SELECT INTO query * FROM queryqueue NATURAL JOIN querypackage
         WHERE active AND error IS NULL AND NOT processing AND id_querypackage=package.id
         ORDER BY systemuser,priority DESC,autorefresh,starttime LIMIT 1;
      IF FOUND THEN
         RAISE NOTICE "Send querystring";
         IF NOT savetofile("temp/" || query.packagename || ".pipe","<!DOCTYPE entities SYSTEM
"/home/sengine/sengine.conf/entity.dtd">
            <senginequery queryqueue="" || query.id_queryqueue || "" idquery=""||CASE WHEN query.id_query IS NULL THEN "none"
ELSE text
            (query.id_query) END || "" iduser=""||query.id_user||"">"||query.querystring||"</senginequery>") THEN
            RAISE EXCEPTION "Package Event Pipe open failture";
         END IF;
         UPDATE queryqueue SET processing=true,starttime=now() WHERE id_queryqueue = query.id_queryqueue;
         IF query.id_query IS NULL THEN
            UPDATE  queryqueue SET error="Temperatory result" WHERE id_queryqueue = query.id_queryqueue;
         END IF;
      ELSE
         IF TG_OP = "DELETE" THEN
            INSERT INTO queryqueue(id_query,autorefresh,id_user)
               SELECT id_query,true,indexofuser("sengine") FROM query
                  WHERE lastmodified IS NOT NULL AND refresh IS NOT NULL AND lastmodified+refresh<now() AND id_query NOT IN
                     (SELECT DISTINCT id_query FROM queryqueue);
         END IF;
      END IF;
   END LOOP;
   IF TG_OP ="INSERT" THEN
      RETURN NEW;
   ELSE
      RETURN OLD;
   END IF;
END;
' LANGUAGE 'plpgsql';

CREATE TRIGGER trigger_queryqueue_scan AFTER INSERT OR DELETE ON queryqueue FOR EACH ROW EXECUTE
PROCEDURE trigger_queryqueue_scan();
```

Posledním triggerem je obhospodařeno mazání záznamu z fronty dotazů. Trigger trigger_queryqueue_delete nejprve testuje mazaný záznam na prázdný atribut error a id_query při resetovaném atributu processing (stav, kdy dotaz u přímo do fronty vloženého dotazu byl načten výsledek, nulován atribut error a záznam může být smazán).

Druhým testem je test na prázdný atribut error a processing nastavený na true. V tomto případě se jedná o zakončení zpracování dotazu. Výsledek je uložen do cache tabulky dotazů (existuje-li záznam, je nahrazen). Je-li dále specifikováno jméno souboru, je do tohoto souboru uložena XML zformátovaná kopie výsledku dotazu. Nelze-li dotaz při specifikovaném jménu souboru uložit, je uložena chybová zpráva do atributu error a stornováno mazání záznamu.

Pokud je druhý test negativní, je stornováno mazání záznamu. Z toho vyplývá, že je-li vyplněn atribut error, záznam nelze smazat. Pokud chceme záznam smazat, je nutné nejdříve atribut error nastavit na NULL, pak při mazání projede první testem tohoto triggeru a je smazán.

Trigger pro mazání dotazu z fronty

```
CREATE FUNCTION trigger_queryqueue_delete ()  RETURNS trigger AS '
DECLARE
   querysave   RECORD;
   queryresult RECORD;
BEGIN
      IF NOT OLD.processing THEN
            RAISE NOTICE "Empty not active query";
            RETURN OLD;
      END IF;
      IF OLD.processing AND OLD.id_query IS NOT NULL AND OLD.error IS NULL THEN
            RAISE NOTICE "Save query result";
            UPDATE query
                  SET lastmodified=now(),processtime=now()-OLD.starttime,
                  querylog="<logsess from=""||OLD.starttime||"" to=""||now()||"">"||OLD.querylog||"</logsess>"||querylog,
                  queryresult=CASE WHEN OLD.error IS NOT NULL THEN "<error>"||OLD.error||  "</error>"ELSE OLD.queryresult END
                   WHERE id_query=OLD.id_query;
            IF OLD.id_query IS NOT NULL AND OLD.savefile IS NOT NULL AND NOT
savetofile(OLD.savefile,getqueryresult(OLD.id_user,OLD.id_query)) THEN
                  UPDATE queryqueue SET error="Output result file open failture", processing=false WHERE id_queryqueue =
OLD.id_queryqueue;
                  RAISE NOTICE "Error saving temperatory result";
                  RETURN NULL;
            END IF;
            RAISE NOTICE "Query done";
            RETURN OLD;
      END IF;
      IF OLD.id_query IS NULL AND length(OLD.queryresult)=0 THEN
            RAISE NOTICE "Empty result, dropping";
            RETURN OLD;
      END IF;
      UPDATE queryqueue SET processing=false, starttime=now() WHERE id_queryqueue = OLD.id_queryqueue;
      RAISE NOTICE "Result stored at queue";
      RETURN NULL;
END;
' LANGUAGE 'plpgsql';

CREATE TRIGGER trigger_queryqueue_delete BEFORE DELETE ON queryqueue FOR EACH ROW EXECUTE PROCEDURE
trigger_queryqueue_delete();
```

3.4.e Cache výsledků vyhledávání

Tabulka QUERYFILERESULT

Tabulka QUERYFILERESULT je pomocná cache výsledků, kterou lze reprezentovat relací mezi tabulkou dotazů QUERY a tabulkou FILE. Tato relace je využívána při uchování výsledků vyhledávání a při generování složitějších map podle obsahu slov.

SQL dotaz pro vytvoření tabulky QUERYFILERESULT

```
CREATE TABLE queryfileresult (
        id_query int4 REFERENCES query NOT NULL,
        id_file int4 REFERENCES file NOT NULL,
        relevancy float,
        data text,
        PRIMARY KEY (id_query,id_file));
```

3.5 Blok lexikálních znalostí

Lexikální znalosti omezujeme na postupně se generující slovník slov. Blok obsahuje číselník jazyků. Uvažuje se ohýbání slov, každý tvar odkazuje na základní tvar, je-li tento detekován apriorně zadanými příponami. Hlubší lexikální analýza se neprovádí.

3.5.a Slovník

Tabulka WORD a obslužné funkce

Seznam slov je implementován pomocí tabulky WORD. Vedle identifikačního čísla slova (id_word) a slova (word) obsahuje atribut locked, který určuje, zda-li je u tohoto slova povoleno automatické zjišťování základního tvaru. Byl-li základní tvar zjištěn, je v atributu id_basicword uloženo identifikační číslo slova, ze kterého je slovo odvozeno. Více podrobnějších informací o této problematice naleznete v popisu příslušného dotazu v kapitole 6.1.

SQL dotaz pro založení tabulky WORD

```
CREATE SEQUENCE seq_word START 1 INCREMENT 1 ;
CREATE TABLE word (
        id_word int4 UNIQUE NOT NULL,
        word varchar(48) NOT NULL,
        id_basicword int4 REFERENCES word,
        locked bool DEFAULT false,
    PRIMARY KEY (id_word));
```

Pro tabulku vytvoříme indexy, první idx_word unikátní na atribut word a druhý na atribut id_basicword.

Indexy pro tabulku WORD

```
CREATE UNIQUE INDEX idx_word ON word USING btree (word);
CREATE INDEX idx_basicword ON word USING btree(id_basicword);
```

Pro zjednodušení vyhledávání identifikačního čísla slova nadeklarujeme funkci int4 indexofword(text,bool), která vrací hodnotu primárního klíče z tabulky WORD záznamu obsahujícího specifikované slovo. Neexistuje-li takovýto záznam, vytvoří jej a vrátí hodnotu primárního klíče tohoto záznamu. Uvažujme case-insensinitive přístup. Je-li druhý parametr nastaven na true, bude vrácena hodnota identifikátoru slova v základním tvaru, při vkládání nového slova nemá druhý parametr definován význam.

Funkce integer indexofword(text word, bool basicfigure)

```
CREATE FUNCTION indexofword (text, bool) RETURNS int4 AS '
DECLARE
   searchvalue text;
   row        RECORD;
   ret        int4;

BEGIN
      searchvalue:=$1;
      SELECT INTO row id_word, id_basicword FROM word WHERE word=searchvalue;
      IF NOT FOUND THEN
            ret:=nextval("seq_word");
            INSERT INTO word(id_word,word) VALUES(ret,searchvalue);
      ELSE
            ret:=CASE WHEN $2 AND row.id_basicword IS NOT NULL THEN row.id_basicword ELSE row.id_word END;
```

```
    END IF;
    RETURN ret;
END;' LANGUAGE 'plpgsql';
```

Pro tabulku WORD nadeklarujeme trigger pro případ, že dané slovo bude z tabulky odstraněno. Trigger odstraní všechny relevantní záznamy ve fulltextovém indexu.

Trigger pro rušení slov

```
CREATE FUNCTION trigger_deleteword() RETURNS TRIGGER AS '
BEGIN
    DELETE FROM fulltextindex WHERE id_word=OLD.id_word;
    RETURN OLD;
END;' LANGUAGE 'plpgsql';

CREATE TRIGGER trigger_deleteword AFTER DELETE ON word FOR EACH ROW EXECUTE PROCEDURE trigger_deleteword();
```

3.5.b Číselník jazyků

Tabulka LANGUAGE a obslužná funkce

Číselníku jazyků odpovídá tabulka LANGUAGE, vedle identifikačního čísla jazyku (id_lang) obsahuje kód jazyka (langcode) a název jazyka (langname).

SQL dotaz pro založení tabulky LANGUAGE

```
CREATE SEQUENCE seq_language START 1 INCREMENT 1;
CREATE TABLE language (
        id_lang int2 DEFAULT nextval('seq_language'),
        langcode varchar(16) NOT NULL,
        langname varchar(32),
        PRIMARY KEY (id_lang));
```

Pro tuto tabulku nadefinujeme funkci integer indexoflang(text langcode), která k zadanému kódu jazyku vrátí příslušné identifikační číslo. Pokud jazyk se zadaným kódem v tabulce neexistuje, je vložen nový záznam s příslušným kódem a vráceno identifikační číslo nově vzniklého záznamu.

Funkce int2 indexoflang(text)

```
CREATE FUNCTION indexoflang(text) RETURNS int2 AS '
DECLARE
    slangcode ALIAS FOR $1;
    ret    integer;
    row    RECORD;
BEGIN
    SELECT INTO row id_lang FROM language WHERE langcode=slangcode;
    IF NOT FOUND THEN
        ret:=nextval("seq_language");
        INSERT INTO language(id_lang,langcode)VALUES(ret,slangcode);
    ELSE
        ret:=row.id_lang;
    END IF;
    RETURN ret;
END;' LANGUAGE 'plpgsql';
```

3.6 Fulltextová báze dat

Fulltextová báze dat je reprezentována pomocí tabulek FULLTEXTBLOCK a FULLTEXTINDEX. Definice bloku je v první jmenované, obsah je pak uložen v druhé.

Tato fulltextová báze slouží pro systém jako primární zdroj dat. Nejedná se ale o implementaci vhodnou pro větší objem dat, implementace na databázovém stoji se v praxi většinou z důvodu časové a paměťové náročnosti nepoužívají. Pro naše účely je ale postačující.

3.6.a Blok báze

Tabulka FULLTEXTBLOCK a obslužné funkce

Indexovaný soubor je rozdělen do bloků podle tagů, mezi nimiž se text nachází. Hlavičky bloků jsou uloženy v tabulce FULLTEXTBLOCK. Bloku je přiřazeno identifikační číslo (atribut id_block), specifikována indexovaná stránka (atribut id_file) a podle ohraničujícího tagu typ bloku (atribut blocktype). Jedná-li se o popis linku mezi soubory, je vyplněn atribut id_file_sec (zdrojový soubor) a atribut id_file ukazuje na identifikační číslo cílového souboru.

Pro tabulku FULLTEXTBLOCK vytvoříme sekvenci seq_fulltextblock.

SQL dotaz pro vytvoření tabulky FULLTEXTBLOCK

```
CREATE SEQUENCE seq_fulltextblock START 1 INCREMENT 1;
CREATE TABLE fulltextblock (
            id_block int4 UNIQUE NOT NULL,
            id_file int4 REFERENCES file NOT NULL,
            id_file_sec int4 REFERENCES file,
            blocktype int2,
      PRIMARY KEY (id_block));
```

Index pro tabulku FULLTEXTBLOCK

```
CREATE INDEX idx_fulltextblock ON fulltextblock USING btree (id_file,id_file_sec);
```

Pro tabulku FULLTEXTBLOCK deklarujeme trigger trigger_fulltextblock_delete, který při mazání bloku smaže i všechna indexovaná slova patřící tomuto bloku.

Trigger pro mazání fulltextových bloků

```
CREATE FUNCTION trigger_fulltextblock_delete() RETURNS TRIGGER AS '
BEGIN
      DELETE FROM fulltextindex WHERE id_block=OLD.id_block;
      RETURN OLD;
END;' LANGUAGE 'plpgsql';

CREATE TRIGGER trigger_fulltextblock_delete BEFORE DELETE ON fulltextblock FOR EACH ROW EXECUTE PROCEDURE
trigger_fulltextblock_delete();
```

3.6.b Index

Tabulka FULLTEXTINDEX

Vlastní fulltextový index představuje tabulka FULLTEXTINDEX. Tabulka obsahuje identifikační číslo bloku (atribut id_block), pozici slova v bloku (atribut position) a identifikační číslo slova (id_word).

SQL dotaz pro vytvoření tabulky FULLTEXTINDEX

```
CREATE TABLE fulltextindex (
            id_block int4 REFERENCES fulltextblock NOT NULL,
            position int2 NOT NULL,
            id_word int4 REFERENCES word NOT NULL,
            PRIMARY KEY (id_block,position));
```

Pro vkládání dat do fulltextové báze dat nadefinujme funkci text fulltext_addblock(int4 idfile1, int4 idfile2,integer blocktype, text content). První dva parametry jsou identifikátory URI adresy, u textového bloku první udává adresu obsahu a druhý je nastaven na NULL, u linků je první cílová URI a druhý zdrojová URL linku. Třetí parametr udává typ bloku a čtvrtý je řetězec obsahující mezerou oddělená slova, která se budou indexovat. Funkce vrací informaci o vloženém bloku ve formátu XML.

Funkce nejdříve založí blok s novým identifikačním číslem, hodnoty ostatních atributů jsou

dány parametry funkce. V druhé části je provedena separace slov, přiřazení identifikačních čísel jednotlivým slovům pomocí funkce integer indexofword(text, bool) a uložení takto vytvořeného záznamu do indexační tabulky fulltextové báze dat. Z výše popsaného vyplývá, že tato funkce je dost časově náročná (volání funkce indexofword na každý záznam).

Funkce text fulltext_addblock(int4 idfile_src,int4 idfile_trg,integer blocktype, text content)

```
CREATE FUNCTION fulltext_addblock(int4,int4,integer,text) RETURNS text AS '
DECLARE
      idfile  ALIAS FOR $1;
      idfile2 ALIAS FOR $2;
      blocktp ALIAS FOR $3;
      blockcnt       ALIAS FOR $4;
      idblock int8;

BEGIN
      idblock:=nextval("seq_fulltextblock");
      INSERT INTO fulltextblock(id_block,id_file,id_file_sec,blocktype) VALUES (idblock,idfile,idfile2,blocktp);
      INSERT INTO fulltextindex(id_block,id_word,position)
            SELECT idblock,indexofword(word,false),pos-32675 FROM wordsepare(lower(blockcnt)) AS (pos int4, word varchar) WHERE
pos<32677;
      RETURN "<block id=""||idblock||""/>";
END;' LANGUAGE 'plpgsql';
```

Pro demostrační vyhledávání ve fulltextové bázi dat zaveďme tabulkovou funkci SETOF fulltextindex fulltextsearchword(int4 idword, bool figures, int4 blocktypemask), která vrací všechny odpovídající záznamy. Prvním parametrem je identifikátor slova, které hledáme, následuje logický atribut určující, zda-li hledáme i ohýbané tvary zadaného slova. Poslední parametr specifikuje masku typů bloku, ve výsledku budou obsaženy jen ty bloky, jejichž logický součin typu bloku s třetím parametrem zadané masky je nenulový. Funkce předpokládá při hledání ohebných slov identifikátor slova v základním tvaru.

Funkce SETOF fulltextindex fulltextsearchword(int4 idword, bool figures, int4 blocktypemask)

```
CREATE FUNCTION fulltextsearchword(int4,bool,int4) RETURNS SETOF fulltextindex  AS '
            SELECT fulltextindex.* FROM fulltextindex NATURAL JOIN fulltextblock
            WHERE (blocktype & $3)<>0 AND (id_word= $1 OR $2 AND id_word IN
                        (SELECT id_word FROM word WHERE id_basicword = $1));
' LANGUAGE 'sql';
```

Při zobrazování výsledků je někdy vhodné uvést slova v okolí hledaného slova. Pro tuto rekonstrukci definujeme funkci text fulltextrestorecontent(int8 idblock,int2 position,int2 distance), která rekonstruuje blok daný hodnotou prvního parametru na pozici (druhý parametr) a vypisuje (třetí parametr) slov před a za.

Funkce text fulltextrestorecontent(int8 idblock,int2 position,int2 distance)

```
CREATE FUNCTION  fulltextrestorecontent(int4,int4,int4) RETURNS text AS '
DECLARE
   row RECORD;
   ret text;
BEGIN
   ret:="";
   FOR row IN SELECT word FROM word NATURAL JOIN fulltextindex WHERE id_block=$1 AND position>$2-$3 AND position<$2+$3
ORDER BY position LOOP
      ret:=ret || row.word || " ";
   END LOOP;

   IF length(ret)=0 THEN
      ret:=NULL;
   END IF;
   RETURN "<part>"||ret||"</part>";
END; ' LANGUAGE 'plpgsql';
```

3.7 CRM blok

V tomto odstavci nadefinujeme tabulky pro statistické zpracování chování koncových uživatelů. Pokud uživatel zvolí dotaz, je vložen záznam do tabulky CRMQUERYRESULT. Pokud uživatel v rámci zobrazeného dotazu klikne na některý z odkazů, je analogicky vložen záznam do tabulky CRMHOSTEVENT. Tabulka CRMHOSTINFO obsahuje základní informace o připojeném uživateli a tabulka BROWSER představuje číselník prohlížečů.

SQL dotaz pro vytvoření tabulek CRM

```
CREATE SEQUENCE seq_browser START 1 MINVALUE 1;

CREATE SEQUENCE seq_crmhostinfo START 1 MINVALUE 1 CYCLE;
CREATE TABLE browser (
        id_browser int2 UNIQUE NOT NULL,
        browsername varchar(128) UNIQUE NOT NULL,
        PRIMARY KEY (id_browser));
CREATE TABLE crmhostinfo (
        id_crmhostinfo int4 UNIQUE NOT NULL,
        id_user int4 REFERENCES userlist NOT NULL,
        id_browser int2 REFERENCES browser NOT NULL,
        ip varchar(17),
        PRIMARY KEY (id_crmhostinfo));
CREATE TABLE crmhostevent (
        id_crmhostinfo int4 REFERENCES crmhostinfo NOT NULL,
        eventtime timestamp NOT NULL,
        id_file int4 NOT NULL,
        id_file_referer int4,
        id_query int4 REFERENCES query);
CREATE TABLE crmqueryresult (
        id_query int4 REFERENCES query NOT NULL,
        id_user int2 REFERENCES userlist NOT NULL,
        eventtime timestamp DEFAULT now());
```

Pro tabulku BROWSER nadeklarujeme funkci integer indexofbrowser(text), které k zadanému názvu vrátí identifikační číslo. Není-li příslušný záznam v tabulce uveden, je založen nový záznam a vráceno identifikační číslo tohoto záznamu.

SQL dotaz pro vytvoření tabulek CRM

```
CREATE FUNCTION indexofbrowser(text) RETURNS int2 AS 'DECLARE
    searchvalue     ALIAS FOR $1;
    ret             int2;
    row              RECORD;
BEGIN
    SELECT INTO row id_browser AS id FROM browser WHERE browsername=searchvalue;
    IF NOT FOUND THEN
        ret:=nextval("seq_browser");
        INSERT INTO browser(id_browser,browsername) VALUES (ret,searchvalue);
    ELSE
        ret:=row.id;
    END IF;
    RETURN ret;
END; ' LANGUAGE 'plpgsql';
CREATE FUNCTION insertcrmhostinfo(integer,integer,varchar(17)) RETURNS int4 AS 'DECLARE
    ret             int2;
BEGIN
    ret:=nextval("seq_crmhostinfo");
    INSERT INTO crmhostinfo(id_crmhostinfo,id_user,id_browser,ip) VALUES (ret,$1,$2,$3);
    RETURN ret;
END;' LANGUAGE 'plpgsql';
```

3.8 Omezení vycházející z databázového modelu

Omezení, která přináší takto nadefinovaný databázový model.

Každý databázový model může popisovat jen určitou část reality, navíc je omezen implementačními a hardwarovými (případně i časově optimalizačními) prostředky. Všechna tato omezení, která jsou zahrnuta v implementační části systému, jsou uvedena v tomto oddílu.

Prvním nedostatkem tohoto databázového modelu je, že neuvažuje možnost přesměrování na jiný soubor na straně serveru. Pokud zašleme serveru požadavek na URL adresu url1 a server bude mít nastaveno, že má vrátit obsah jiného souboru, vrátí URL adresu url2 tohoto souboru v HTTP hlavičce. Tato nová lokace se ale v bloku URL adres neprojeví, systém se stále odkazuje na původní URL (url1). Tento problém lze vyřešit přidáním tabulky, která bude reprezentovat tuto relaci mezi soubory. Datový model na takového dvě adresy nahlíží jako na dva odlišné soubory. Tento fakt se projeví za předpokladu, že jedna část stránek se odkazuje na url1 a druhá na url2.

Dalším omezením je maximální počet slov v jednom bloku. Každý blok může obsahovat maximálně 65535 slov. Zvýšení maximálního počtu by vedlo k větší paměťové náročnosti.

4 Mechanizmus zpracování dotazů

Prioritně řazená fronta dotazů

Jak již bylo popsáno v kapitole 2, systém navrhujeme jako multiuživatelský. Každý uživatel má možnost zadat dotaz a následně zobrazit výsledek tohoto dotazu. Systém musí zabránit zpracování neomezeného počtu dotazů najednou, aby nemohlo dojít k jeho přetížení.

Výše popsané kritérium splňuje řazení příchozích dotazů do fronty. Zpracování fronty je cyklické, z fronty se vybere dotaz s největší prioritou a pošle se ke zpracování. Po zpracování je dotaz z fronty vyjmut a cyklus se uzavírá. Je-li fronta prázdná, cyklus čeká na další příchozí dotaz.

Nevýhodou takto navrženého algoritmu je, že umožňuje současně zpracovávat pouze jeden dotaz. Při zpracovávání dlouho trvajícího dotazu systém dojde k časovému prodlení všech dotazů ve frontě, což není akceptovatelné, protože uvažujeme zpracování dotazů z webového rozhraní a uživatel očekává odpověď na svůj dotaz pokud možno v reálném čase. Aby k této situaci nedocházelo, je nutné dotazy rozdělit na dlouhodobé a krátkodobé a udržovat dvě nezávislé fronty. V obecnosti lze rozdělit dotazy na n skupin podle délky trvání. Skupinu dotazů nazveme balíčkem dotazů.

Uvažujeme, že jeden dotaz bude součástí pouze jednoho balíčku. tj. bude definován v rámci jednoho balíčku. To umožňuje, aby každý balíček dotazů měl vlastní konfigurační soubor. S využitím přístupových práv získáváme velmi volně konfigurovatelný systém, kdy jednotlivým administrátorům systému umožňujeme spravovat konkrétní balíčky dotazů. Kvůli zjednodušení správy balíčků požadujeme dotazy ve formátu XML a konfigurační soubor ve formátu XSLT, který zadaný dotaz převede do spustitelné formy.

Dalším požadavkem na frontu dotazů je, aby u zvolených dotazů zajistila automatickou aktualizaci, tj. aby přidala do fronty dotaz s nastavenou aktualizací v okamžiku, kdy doba od poslední aktualizace přesáhla stanovenou mez. Aktualizované dotazy budou mít menší prioritu nežli uživatelem zadané, protože výsledek aktualizovaného dotazu nikdo nečeká.

4.1 Implementace

Popišme konkrétní implementaci mechanizmu zpracování dotazů s databázovým pozadím podle popsaného modelu.

Při spuštění systému je pro každý balíček vytvořen proces, který bude dotazy daného balíčku zpracovávat. Standardní výstup z tohoto procesu je pomocí roury přesměrován na databázového klienta. Tím je zaručeno, že časově náročné připojení klienta k databázi bude provedeno pouze v průběhu inicializace skriptu.

Systém obsahuje tabulku dotazů QUERY (viz kapitola 3.4), kde jsou v záznamech uloženy dotazy a vlastnosti dotazů (automatická aktualizace, poslední aktualizace, bitová maska práv). Prioritně řazená fronta je realizována pomocí tabulky QUERYQUEUE, přičemž vkládání a mazání záznamu do/z tabulky je ošetřeno triggery. Do fronty lze zadat dotaz definovaný v tabulce QUERY, v tomto případě se zkopírují všechny relevantní informace do fonty, nebo lze zadat přímo dotaz s vyplněnými požadovanými vlastnostmi.

Záznam s dotazem je vložen do fronty jedním ze dvou popsaných způsobů. Před operací fyzického vložení záznamu je vyvolán trigger, který zajistí zkopírování informací o dotazu v případě, že dotaz je uložen v tabulce QUERY, autentifikuje uživatele a porovná práva uživatele s bitovou maskou práv dotazu. Neprojde-li jeden z testů, je dotaz odmítnut, je sice vložen do tabulky fronty, ale v poli error je uloženo příslušné chybové hlášení. Záznamy s vyplněným chybovým hlášením jsou dále ignorovány.

Po fyzickém vložení záznamu do fronty je volán další trigger -skenovací. Ten zjistí seznam všech aktivních balíčků, které momentálně nezpracovávají žádný dotaz. Pro každý takový balíček je

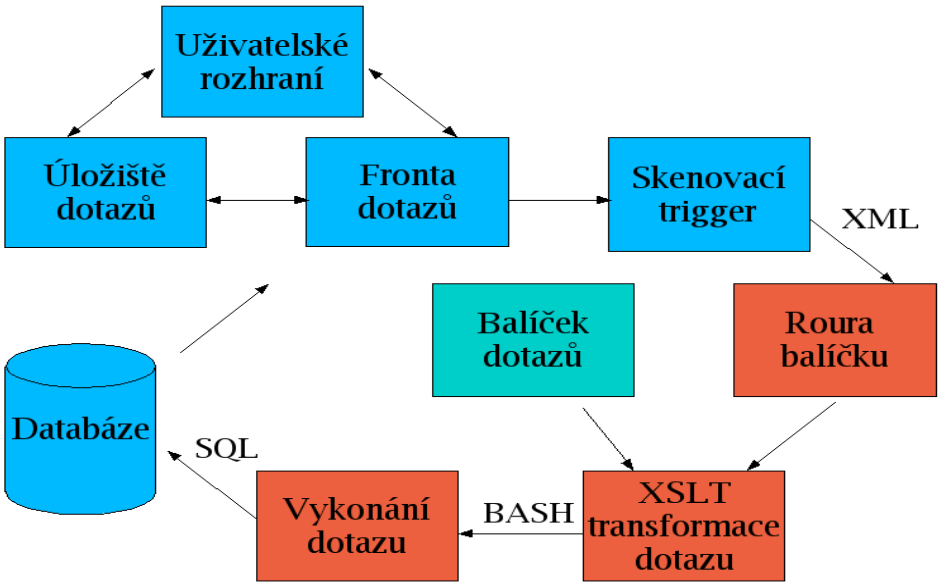
proveden výběr z fronty dotazů na dotaz s největší prioritou. Existuje-li takový dotaz (tj. fronta obsahuje nějaký dotaz daného balíčku určený ke zpracování), je specifikace dotazu a informace o záznamu poslána ve formátu XML pojmenovanou rourou zpracovávajícímu skriptu balíčku.

Zpracovávající skript balíčku vytvoří v dočasném adresáři pracovní prostor dotazu a z pojmenované roury přijme požadavek ze scanovacího triggeru a provede XSLT transformaci tohoto požadavku na dávku bash shellu. Protože databázový server PostgreSQL nesynchronizuje všechny připojené klienty najednou, dochází k situaci, že již transformovaný dotaz odkazoval na záznam, který již vložen byl (vložení provedl jiný klient), ale klient zpracovávající balíček ještě nebyl synchronizován a příslušný záznam nenalezl. Příčina této chyby nebyla v dokumentaci nalezena, řešením je vložení čekající smyčky, kdy se testuje, zda-li zadaný záznam již existuje.

Výsledek XSLT transformace je uložen do souboru. Je-li tento soubor nenulové délky, je předáno řízení tomuto souboru (skriptu). Ten provede operace požadované dotazem a na konci smaže záznam v tabulce fronty. Operací mazání je volán trigger, který v případě, že dotaz byl zadán prostřednictvím tabulky dotazů, modifikuje čas a dobu poslední aktualizace, výsledek a k stávajícímu logu přidá logovací informace z právě proběhnuvšího algoritmu. Následně je volán skenovací trigger popsaný výše a algoritmus zpracování se cyklicky uzavírá.

Ukončení cyklu je dáno speciálním dotazem, který nastaví daný balíček jako neaktivní. Tím při volání skenovacího triggeru nedojde k výběru tohoto balíčku (vybírají se jen aktivní balíčky) a tím je znemožněno poslání dalšího požadavku.

Celý systém včetně propojení s databázovým modelem popisuje následující obrázek 4.1.



Obr 4.1. Schema mechanizmu zpracování dotazů

5 Crawler

Crawler je nástroj pro získání dat z prostředí internetu a uložení reprezentace těchto dat na lokální souborový systém. Crawler lze logicky rozdělit na tři části. První provede stažení obsahu, druhá část převede tento obsah na požadovaný formát (crawlerizace) a třetí provede vintegrování obsahu do stávajícího systému (indexace).

Pro vstup do crawleru lze využít libovolná na internetu veřejně přístupná data. S daty z jednotlivých internetových serverů lze manipulovat pomocí protokolů. Omezme se pouze na základní, na protokol HTTP u webových (někdy též WWW) serverů nabízejí webové stránky a FTP (File Transmition Protocol) u serverů nabízející soubory. Pro úplnost ještě jmenujme protokoly HTTPs a FTPs, které jsou bezpečnými (šifrovanými) alternativami zmíněných protokolů. Ze zadání práce se dále omezíme pouze na webové servery, tedy na HTTP protokol.

Webová stránka je soubor, který zpřístupňuje webový server. Každá webová stránka je jednoznačně určena *URI adresou* (Uniform Resource Identificator), která se skládá ze jména serveru a jména souboru včetně případné adresářové struktury. Webové stránky někdy bývají lokalizovány pomocí *URL adresy* (Unifrom Resource Locator) [2]. Narozdíl od URI adresy nemusí být specifikováno jméno souboru, tedy adresa http://server/adresar/ je URL adresa a http://server/adresar/soubor je URI adresa stránky. Na serveru je URL adresa podle konfigurace jednotlivých serverů převedena na URI a je prohlížeči vrácen požadovaný soubor.

Webové stránky rozdělujeme na *statické* a *dynamicky generované*. Statické stránky jsou na serveru uloženy jako soubor, webový server klientovi pouze přepošle obsah této stránky. Naopak u dynamicky generovaných stránek je obsah stránky generován pomocí skriptu (PHP, ASP) nebo jiných technologií (CGI, JSP, Cocoon), které jsou s volanou stránkou svázány.

Jak již bylo nastíněno, webový prohlížeč komunikuje se serverem pomocí protokolu HTTP. Protokol HTTP je bezstavový, při ukončení spojení klienta se serverem nejsou informace dále udržovány. Tam, kde je toto chování nežádoucí, lze využít různých mechanizmů (session, cookies), které tuto vlastnost obcházejí.

Komunikaci zahajuje klient odesláním HTTP požadavku serveru [9]. Požadavek se skládá z příkazu (jmenujme vedle jiných příkaz GET sloužící k získání obsahu stránky a příkaz HEAD sloužící k získání HTTP hlavičky stránky) a URL nebo lépe URI adresy požadované stránky. K požadavku lze připojit některé informace z prostředí klienta (např. typ a verzi prohlížeče). Server požadavek zpracuje a zašle příkazu odpovídající výsledek. Výsledek se skládá ze dvou částí, HTTP hlavičky a samotného obsahu, je-li příkazem vyžadován. HTTP hlavička začíná stavovým řádkem, kde je uvedena verze protokolu, kód stavu a popis stavu vyřízení požadavku. Seznam stavů je uveden v citované normě. Následují informace o stránce, její typ, velikost obsahu a další údaje. Každý řádek hlavičky představuje jeden atribut informace, informace jsou uvedeny ve tvaru název: hodnota. Seznam informací je rovněž uveden v normě. HTTP hlavička a případný následující obsah je oddělen prázdným řádkem. Uveďme příklad odpovědi webového serveru na požadavek o získání textového souboru.

```
HTTP/1.0 200 OK
Date: Sat, 03 Jan 2004 11:04:40 GMT
Server: Apache/1.3.28 (Unix) PHP/4.3.3 mod_ssl/2.8.15 OpenSSL/0.9.7b
Last-Modified: Wed, 08 Oct 2003 16:37:28 GMT
ETag: "43ff1-31a-3f843d48"
Accept-Ranges: bytes
Content-Length: 794
Content-Type: text/plain
Age: 3777
X-Cache: HIT from proxy.felk.cvut.cz
X-Cache-Lookup: HIT from proxy.felk.cvut.cz:80
Proxy-Connection: close

Zde je obsah souboru.
```

První část crawleru tedy pošle požadavek webovému serveru a ten podle výše nastíněného postupu odpoví. Nyní je nutné analyzovat serverem vrácený výsledek. Zpracujme HTTP hlavičku, podle vraceného kódu stavu budeme dále analyzovat obsah, jedná-li se o chybový stav, nemá další analýza smysl. Uchováme vybrané informace z HTTP hlavičky (čas poslední modifikace, ...). Podle v hlavičce vráceného typu obsahu (content-type) je na vrácený obsah aplikován filtr, který převede obsah na jednotný formát stroje.

U některých typů (formát HTML (text/html) nebo PDF (application/pdf)) obsahu je možnost pomocí hypertextových odkazů ukazovat na další stránky. Tyto odkazy dále nazýváme linky. Linky rozdělujeme na relativní a absolutní. Relativní link je link v rámci jednoho webu, jsou vyjádřeny relativní pozicí vůči zdrojové webové stránce nebo vůči explicitně zadané URL adrese. Naopak absolutní linky zpravidla ukazují na externí zdroje, formát externího linku odpovídá URL adrese.

Jsou-li v obsahu uvedeny linky, je žádoucí crawlerizovat i tyto linky. Tím dochází k zacyklení procesu. Protože je nutné proces crawlerizace webu řídit, zavádíme pojem hloubky stahování. Hloubka stránky je definována jako počet stránek mezi touto stránkou a stránkou na začátku cyklu. Aby se crawler nezacyklil, používáme limit pro tuto hloubku, tedy crawlerizujeme pouze stránky do určité hloubky.

V poslední části se zabývejme indexací. Existují různé modely fulltextových bází [4], jmenujme booleovský, vektorový a plný.

Booleovský model je reprezentován dvojdimenzionálním polem, první index přestavuje označení stránky a druhý pak označení slova. Pro kažký prvek platí, že hodnota prvku je rovna 1, pokud se indexem dané slovo v daném souboru vyskytuje, 0 v opačném případě (odtud název). Hledání zadaného sousloví probíhá tak, že se hledají všechny stránky, které mají slovům příslušící prvek nenulový, kritériem je součet těchto prvků.

Nevýhodou booleovského modelu je, že nelze seřadit výsledek dotazu podle počtu výskytů jednotlivých slov. To řeší *model vektorový*, který vychází z modelu boolovského, modifikací je, že hodnota každého prvku pole odpovídá četnosti výskytu slova na stránce (každá stránka je reprezentována vektorem, kde i-tý prvek vektoru odpovídá i-tému slovu). Hledání probíhá tak, že pro hledané sousloví je vytvořen vektor, nenulové prvky odpovídají hledaným slovům (tedy slova lze vážit) a pro každou stránku se provede skalární součin stránce odpovídajícího vektoru s vektorem slov hledaných. Použitím relevance získáváme lépe seřazené výsledky.

Nevýhodou vektorového modelu je nemožnost zpětné rekonstrukce textu. To řeší *model plný*, kdy každému slovu na stránce odpovídá jeden záznam. Nevýhodou tohoto modelu je paměťová náročnost.

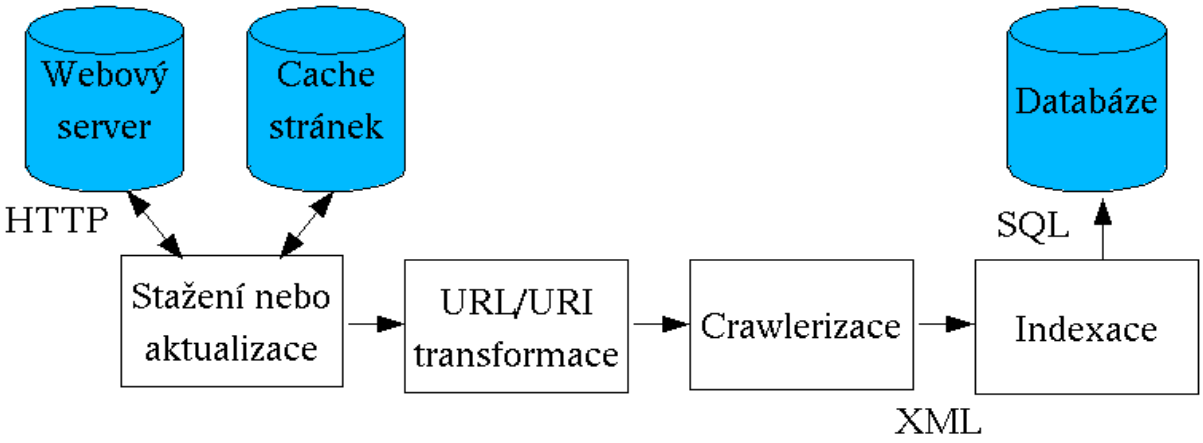
5.1 Implementace

Shrňme nyní požadavky na crawler. Ze zadání se omezujeme pouze na webové servery (HTTP protokol), výstup crawleru je ve formátu SQL dotazu pro databázi vytvořenou podle v kapitole 3 uvedeného databázového modelu. Crawler rozdělíme do tří částí podle předchozího odstavce. Další požadavek je, abychom data dokázali efektivním způsobem aktualizovat, tedy aby nebyla provedena crawlerizace u stránek, které se od poslední doby aktualizace nezměnily. U druhé a třetí části je vhodné, aby filtry bylo možné podle potřeby přidávat s minimálními změnami konfigurace.

Pro potřeby složitých dynamicky generovaných webů je nutné umožnit použití apriorní informace o struktuře parametrů jednotlivých skriptů.

Posledním krokem je volba fulltextového modelu. Navržený databázový model implementuje model plný, přičemž stránky navíc dělí do bloků podle významu, např. titulek stránky, klíčová slova stránky, nadpis, text atd.

Návaznost jednotlivých kroků je uvedena na obrázku 5.1.



Obr 5.1. Schéma crawleru

5.1.a Stažení stránek

Stažení stránek je provedeno pomocí GNU nástroje wget, který implementuje komunikaci pomocí HTTP protokolu. Nástroj je ovládán pomocí parametrů z příkazové řádky. Umožňuje uložit do pracovního prostoru dotazu stažené stránky včetně HTTP hlaviček. Lze jej používat v tzv. rekurzivním módu, kdy se provede analýza linků HTML stránek, a pokud cíle těchto linků ještě nebyly staženy a hloubka těchto stránek je menší nežli zadaná, je provedeno stažení těchto stránek. U rekurzivního módu lze omezit domény, ze kterých stránky mohou, nebo nesmějí být staženy.

Nástroj rovněž umožňuje zachovat dobu poslední modifikace stránky tak, že u lokálně staženého souboru nastaví dobu poslední modifikace na požadovanou hodnotu. Pokud před započetím stahování adresář již obsahuje některé soubory, je nástrojem testována doba poslední modifikace stránky a pokud tato doba odpovídá době poslední modifikace souboru, je stahování pozastaveno (tj. rovná se o identický soubor a tudíž není nutná jeho aktualizace).

U dynamicky generovaných stránek je obsah generován v okamžiku požadavku, parametr poslední změny stránky pozbývá smyslu, v HTTP hlavičce není ani uveden. Tento parametr je možné doplnit na začátku volaného skriptu, ovšem tuto eventualitu používá mizivé množství stránek. Další možností je u dynamicky generovaných stránek sledovat změnu nepřímo, např. pomocí fyzické délky souboru (délka obsahu stránky není v HTTP hlavičce uvedena, protože v době odesílání hlavičky nemusí být známa) . To je komplikováno faktem, že soubory obsahují vedle obsahu i HTTP hlavičku. Z těchto důvodů se dynamicky generované stránky neaktualizují jen v případě, že mají vyplněnou dobu poslední aktualizace, prakticky tedy téměř vždy.

Druhým problémem je možnost uvedení URL adresy namísto URI adresy. Ilustujme tento případ. Z prohlížeče zadáme URL adresu, např. http://server a odešleme požadavek na stažení. Server podle své vlastní konfigurace provede transformaci na URI adresu. Řekněme, že na tomto serveru běží serverová aplikace Apache. V konfiguraci aplikace je uveden seznam možných doplnění adresy URL na URI. Server postupuje od začátku seznamu a testuje, zda-li URL doplněný o řetězec ze seznamu neodpovídá stránce uložené v prostoru určeném pro webové stránky. Pokud ano, je zvolena tato URI adresa, neodpovídá-li, je testován následující prvek seznamu. Byl-li prvek posledním, je podle konfigurace serveru vrácen chybový stav, nebo je dynamicky vygenerována HTML stránka s obsahem adresáře, který URL odpovídá. V našem konkrétním příkladu tedy server nalezl soubor odpovídající URI adrese http://server/index.php, vykonal skript a výsledek vrátil prohlížeči.

Problém nastává v prohlížeči, který požadoval stránku http://server, ale vrácena mu byla

stránka http://server/index.php. Tento fakt se ale prohlížeč nemusí dovědět. Tento principiální problém nástroj wget řeší tak, že pokud byla zadána URL adresa, která nekončí jménem souboru, a stránka vrácená serverem neobsahuje informaci o URI (pole location), stáhne tuto stránku jako stránku s URI vytvořenou přidáním řetězce "index.html" k zadané URL adrese. V našem případě tedy označí stránku s URI http://server/index.php jako stránku s URI http://server/index.html.

Tato problematika je řešena až při transformaci obsahu stránek, řešením se budeme zabývat v následující části.

Rekapitulujme tedy tuto část crawleru. Z textu vyplývá, že v první fázi je vhodné otestovat, zda-li zadané stránky již nebyly staženy (a jsou uloženy v cache stránek), pokud ano, zkopírovat je do pracovního adresáře. Tím zajistíme, že nebudeme stahovat stránky, u kterých nebyla prokázána modifikace. Následuje stažení stránek v rekurzivním módu do zadané hloubky ze serveru do pracovního adresáře. U aktualizovaných nebo nových souborů je žádoucí provést kopii do cache stránek pro další stahování (aktualizaci). V této fázi máme stránky staženy v pracovním adresáři a můžeme je začít analyzovat.

Nevýhodou použitého nástroje je, že v rekurzivním módu nevytváří v pracovním adresáři soubory odpovídající URI, u kterých server vrátil chybový kód stavu.

5.1.b Transformace URI adres

V některých případech je vhodné začlenit apriorní informaci o URI stránek. Především se jedná o transformaci URL adresy, která není URI adresou, tj. v minulém odstavci diskutovaný případ typu index.php/index.html. Použitím pravidla lze zajistit, že stránka odpovídající souboru index.html je uvažována jako index.php (na úrovni souborového systému pomocí symbolického odkazu).

Dalším dobrým důvodem pro vložení transformace je pořadí argumentů u dynamicky generovaných stránek, kdy URL s parametry (a,b) odpovídá stejné URI adrese jako URL s parametry (b,a) v tomto pořadí. Tato nejednoznačnost je odstraněna seřazením parametrů podle abecedy a nahrazením souboru odpovídající neseřazenému pořadí parametrů symbolickým linkem na soubor se seřazenými parametry.

Poslední implementovanou transformací je přípustnost parametrů. U URI lze definovat parametry, které jsou nutné pro identifikaci stránky. Definujeme seznam parametrů, ostatní parametry jsou ignorovány. Parametry jsou rozděleny do následujících kritérií.

- atribut může být uveden, ale nemusí
- atribut je uveden
- atribut je uveden a nabývá zadaných hodnot

Stránky, které dané požadavky na parametry nesplňují, jsou označeny jako ignorované.

Vlastní transformace je prováděna na úrovni databáze při vkládání linků.

5.1.c Transformace obsahu

Předpokládejme, že máme stránky staženy v pracovním adresáři pomocí výše zmíněného postupu. Nyní začne analýza stažených stránek. Analýza je navrhnuta tak, aby umožnila zpracovat výstup různými způsoby (např. pouze linky na generování map, s fulltextovým indexováním, s benchmarkem) a aby formát tohoto výstupu bylo možné modifikovat bez zásahu do zdrojových kódů (např. při dílčí změně databázového modelu). To lze zaručit použitím mezikroku, kdy výstupem analýzy je XML dokument, a pomocí různých šablon XSLT lze generovat SQL dotaz, který uloží data do databáze.

Nyní k samotné analýze. V první fázi zjistíme, zda-li byly vůbec nějaké stránky staženy. Pokud je pracovní adresář prázdný, v předešlé fázi nastala chyba a proces je ukončen.

V principu máme dvě možnosti, jak postupovat. První možností je vytvořit seznam stažených

souborů a na každý soubor z tohoto seznamu aplikovat filtr, výsledný XML dokument transformovat na SQL dotaz a ten spustit. U tohoto přístupu je jednoduché spočítat počet analyzovaných souborů (a koncovému uživateli zobrazovat kolik stránek z kolika je již analyzováno). Nevýhodou tohoto způsobu je, že nelze průběžně zjišťovat mezivýsledky u jiných balíčků (např. generování map do již indexované hloubky).

Zmíněné nedostatek odstraňuje princip druhý, kdy jsou stažené soubory procházeny hierarchicky. Stránky jsou rozděleny do úrovní podle hloubky a každá úroveň se analyzuje zvlášť. Vstupem každé úrovně je seznam a stav stránek, které se mají analyzovat. Každá úroveň má vlastní dočasný soubor pro uchování SQL dotazu.

Seznam první úrovně obsahuje pouze stránku s URI adresou, která byla zadána jako počáteční pro stahování stránek, a její stav je nastaven na aktualizaci. Nyní zavoláme proceduru analýzy úrovně s tímto seznamem. Procedura postupně prochází soubory ze seznamu.

Nejprve je testován stav stránky. Je-li stránka na seznamu již analyzovaných stránek, přechází se na další stránku v seznamu. Není-li nutné stránku aktualizovat, je tato stránka vložena do seznamu analyzovaných stránek a je vybrána další stránka ze seznamu. Je-li stránku potřeba aktualizovat, provede se test, zda-li zadaná stránka odpovídá URL adrese bez jména souboru. V kladném případě je v souladu s nástrojem pro stažení URL adresa doplněna na URI, avšak v databázi je uložena pod URL adresou (není-li předefinováno transformací adres).

Je testována existence souboru v pracovním adresáři odpovídající URI adrese. Pokud soubor neexistuje, je pomocí nástroje wget proveden pokus získat tuto konkrétní stránku znovu, nástroj je volán s parametrem určujícím, že obsah HTTP hlavičky je zkopírován do logovacích informací. Zpětnou analýzou těchto informací lze získat kód stavu. Kód je transformován do stavu odpovídajícímu databázovému modelu (není-li kód chybový, je stránka označena jako registrovaná). Tato stránka je zařazena do seznamu analyzovaných stránek a je vybrána další stránka ze seznamu.

V opačném případě, tedy pokud soubor odpovídající URI adrese existuje, je na něj aplikován filtr na převod do XML dokumentu, podle požadovaného módu je provedena XSLT transformace tohoto dokumentu na SQL dotaz. Stránka je zařazena do seznamu analyzovaných stránek a postup se opakuje pro další stránku v seznamu.

Je-li dosaženo konce seznamu, je procedurou vygenerovaný SQL dotaz pro danou úroveň zaslán databázovému stroji, který dotaz zpracuje. Není-li seznam stránek aktuální úrovně prázdný, je k vygenerovanému SQL skriptu přidán dotaz, který vrátí seznam stránek odkazovaných ze stránek z aktuální úrovně. Pro každý soubor z tohoto seznamu je testováno, zda doba poslední změny souboru v pracovním adresáři (tj. doba poslední modifikace stránky) je shodná s dobou poslední modifikace uloženou v databázi. Podle toho je určeno, zda-li je nutné danou stránku aktualizovat či nikoli. Seznam stránek je pomocí pojmenované roury [1] poslán jako vstup pro proceduru zpracovávající další úroveň hloubky, čímž je zajištěna synchronizace mezi databázovým klientem a zpracováním dotazu.

Algoritmus je ukončen v okamžiku, kdy je proceduře vrácen prázdný seznam, tedy v případě, kdy všechny stránky z předchozí úrovně jsou buď registrované (registrovaná stránka neobsahuje informace o linkách, tj. seznam cílových stránek linků z registrované stránky je prázdný) nebo je u nich nastaven chybový status. Algoritmus tedy požaduje zápis linků do databáze a zabraňuje cyklické aktualizaci stránek, z čehož plyne, že algoritmus ukončí svoji činnost (z předchozího textu to explicitně nevyplývá).

5.1.d Filtr obsahu stránky

V předchozím odstavci byl při popisu procesu použit filtr pro transformaci stránky do formátu XML. Rozeberme nyní činnost toho filtru.

Vstup pro tento filtr tvoří URI adresa, pod kterou bude stránka uložena v databázi, číslo fronty dotazu a název projektu. Poslední dva parametry slouží pro interní funkcionalitu systému. Výstupem je XML dokument, který popisuje crawlerizovanou stránku.

Filtr lze rozdělit na dvě části, na analýzu HTTP hlavičku a na analýzu obsahu.

Problematika a struktura HTTP hlavičky byla popsána v teoretické kapitole 5. Filtr ve zdrojovém kódu obsahuje pole atributů, jejichž hodnoty z HTTP hlavičky se mají zachovat v XML dokumentu. U některých atributů je nutné provést konverzi. V současné konfiguraci seznam obsahuje tyto atributy: location, content-type, content-length, content-language,date, lastmodified, etag. Tento seznam je rozšířen ještě o atributy title, keywords, description a crawler, hodnoty těchto atributů jsou poskytovány dílčími filtry obsahu.

Je-li kód stavu vrácené stránky roven hodnotě 200 (odpovídá vše OK), provede se druhá fáze -analýza obsahu stránky, v opačném případě je přeskočena. Podle typu obsahu (atribut content-type) je vybrán dílčí filtr, existuje-li takový, který obsah stránky převede do XML formátu. Dílčí filtr je spuštěn s parametrem URL adresy, obsah je zapsán do standardního vstupu filtru a výsledek je očekáván na standardním výstupu filtru. Existence filtru je testována za běhu, takže lze velmi jednoduše přidávat filtry pro další formáty obsahu stránek.

Výstup dílčího filtru lze rozdělit na dvě části. První, formátem shodná se seznamem atributů a jejich hodnot v HTTP hlavičce a druhá ve formátu XML.

V první části je možné vyplnit, a nebo předefinovat (umožňuje-li to daný typ obsahu) hodnoty atributů z HTTP hlavičky stránky. Každý filtr by zde měl uvést svůj název a verzi v atributu crawler. Mohou následovat další hodnoty, např. název stránky, klíčová slova nebo popis stánky (v HTML specifikované pomocí obsahu elementu <title> nebo elementu <meta> s atributem name) nebo předefinované hodnoty HTTP atributů (v HTML element <meta> s atributem http-equiv). První část končí prázdným řádkem.

Dále následuje druhá část ve formátu XML. Dílčí filtr převedl obsah stránky do XML formátu a vrací řízení filtru.

Pomocí shodných mechanizmů, jako byly analyzovány atributy HTTP hlavičky, jsou do paměti uloženy hodnoty atributů vrácené dílčím filtrem. Existuje-li v paměti již nadefinovaná hodnota atributu a první fáze filtru, pak je přepsána.

Ve třetí, poslední fázi je vygenerován požadovaný XML dokument. Z paměti jsou vygenerovány vstupní interní parametry a podle hodnot HTTP atributů i příslušné elementy. Vrátil-li dílčí filtr nějaký XML dokument, je tento dokument vložen do výstupního XML dokumentu do elementu <body>. Jelikož HTTP atributy jsou ve výstupním XML dokumentu uvedeny pomocí elementů, je nutné tyto elementy definovat pomocí příslušného DTD. To ospravedlňuje uvedení seznamu HTTP atributů přímo ve zdrojovém kódu.

5.1.e HTML formát a implementace filtru HTML stránek

Mezi dílčí filtry popsané v předchozím odstavci patří i implementovaný filtr pro převod HTML stránek. Protože formát HTML je pro webové prezentace hojně užíván, popišme filtr detailněji.

Historie jazyka HTML

Jazyk HTML (HyperText Markup Language) vychází, stejně jako jazyk XML, z rodiny jazyků SGML. Je to jazyk navržený pro formátování textu. V současné době existuje několik norem [8], které upravují pravidla pro HTML jazyk. Jmenujme alespoň HTML 3.2, která byla prvním masově používaným formátem, následovanou HTML 4.0, která vedle rozšíření obsahovala i první restrikce.

Bohužel, dodržování norem bylo velmi volné, především nejpoužívanějším komerčním prohlížečem (Internet Explorer firmy Microsoft), a nebyly nástroje pro vynucení používání norem. Argumentem tvůrců webových stránek byl fakt, že normu není potřeba dodržovat, protože to funguje i bez ní (alespoň ve nejrozšířenějším prohlížeči) a oni ji ke své práci nepotřebují. Některé aspekty z normy v praxi zcela vypadly, např. používání koncových tagů tam, kde to neovlivňovalo design stránky. Některé přetrvávají i v současných produktech, např. validní stránka uložená standardním způsobem v Internet Exploreru (verze 6.0 s nejnovějšími záplatami; prosinec 2003) je na lokálním disku nevalidní, protože zmizí vybrané koncové tagy, atd.

S vývojem možností sítě internetu začal vznikat požadavek na změnu trendu. Po serverech internetu se lze pohybovat po různých dalších protokolech (WAP) pomocí různých zařízení (např. mobilní telefon). Zde je výhodné, aby data byla uložena odděleně od formátování. Tato zařízení pochopitelně umí zpracovat i HTML, ale z důvodu omezených HW prostředků potřebují dodržovat nějakou závaznou normu. Tento fakt vedl k znovuobjevení jazyka XML a ustanovení norem jazyka XHTML, který vychází z normy HTML 4.0, ale přísně vyžaduje, aby byla stránka validní. Dnes existuje přechodná norma HTML 4.01 a podle restrikcí rozdělené normy XHTML ve verzích 1.0 a 1.1 a mutací Strict, Transitional a Frameset.

Druhým aspektem je požadavek na strojové zpracování HTML stránek. Právě nedodržování norem je problém, se kterým se potýkají stroje indexující obsah stránek. Nedodržením norem vzniká neurčitost obsahu, nemožnost přímé transformace na jiné formáty (např. odstranění formátování a následná extrakce dat). Problém je i používání kontextových značek namísto značek formátovacích (tabulkový layout).

Právě vyhledávače se stávají důvodem (vedle výše zmíněných), proč je dokumenty vhodné psát čitelné nejen pro člověka, ale i pro stroj. Vyhledávače nyní začínají penalizovat nevalidní HTML stránky [12], tím se propadají stránky s kvalitním obsahem ale nevalidní strukturou do nižší pozice. To se zpětně dotýká majitelů stránek a proto požadují nové stránky validní.

Filtr pro HTML

Nejjednodušším způsobem, jak HTML stránku převést do XML, je XSLT transformace. Tento způsob nelze prakticky využít, z textu výše vyplývá, že značné množství stránek není validní a žádný vyhledávač si nemůže dovolit takové procento stránek neindexovat.

Odbočme ještě ke způsobu analyzování HTML stránky, můžeme použít XSLT procesor nebo SAX parser. XSLT procesor by vyhovoval pro většinu stránek (malý rozsah dokumentu), SAX je obecnější a jeho implementace je jednodušší. Obě dvě technologie ale vyžadují validní vstup.

Co tedy s nevalidními stránkami? První možnost je validace a provedení nastíněné transformace. Validace je poměrně náročný proces, nástroje pro validaci jsou většinou komerční a vyžadují apriorně zadané informace (GNU nástroj tidy). Druhou cestou je použití analyzátoru, který umožňuje analyzovat i nevalidní stránky. Standardně dodávaný a používaný xmllib tuto možnost nepodporuje.

Pro aplikaci byl napsán prázdný objektový SAX analyzátor včetně podpůrných modulů (ošetření různých typů vstupů), který umožňuje parsovat i XML nevalidní dokumenty bez záruky, že výstup parseru je XML validní. Validitu výstupu je nutné zaručit nativním kódem, který bude nevalidním dokumentem generovanými událostmi generovat XML validní výstup.

Vedle problematiky chybějících koncových tagů (kterou z principu SAX parseru nelze odstranit) je nutné řešit chyby v zápisu atributů a jejich hodnot. Zpravidla se jedná po dva problémy.

- **Absence hodnoty** Za jménem atributu není znak = a hodnota (tento problém vychází z norem HTML 3.2 a nižších). Řešením je oprava na stávající normu, kdy se v tomto případě použije pro hodnotu shodný řetězec jako pro atribut, tedy počáteční tag <option value="1" selected> je převeden na <option value="1" selected="selected">.
- **Absence uvození hodnoty** Hodnota atributu není uvozena jednoduchými (apostrofy) nebo dvojitými uvozovkami. Tuto vlastnost podporoval výše citovaný komerční prohlížeč. Zde ponecháme řešení, které bylo akceptováno prohlížečem, tedy jako hodnota je uvažována posloupnost znaků mezi rovnítkem a prvním bílým znakem.

Výše zmíněné problémy vypadají na první pohled triviálně, ale jedná se o zejména na ladění náročnou implementaci, uvážíme-li možné kombinace.

Samotný filtr HTML stránek představuje objekt zděděný z objektu SAX parseru. Výstupní XML dokument je generován jak koncovými, tak počátečními tagy (v případě, že koncový tag chybí a následující počáteční tag neodpovídá kontextu). Takto navržená implementace neuvažuje vnoření identických elementů, i když norma tento aspekt umožňuje (např. u tagu norma vnoření povoluje, u tagu <p> nikoli).

Hlavním úkolem je rozdělit obsah HTML stránky na části a analyzovat linky. Podle tagů zadaných v ošetření událostí SAX parseru (tagy pro nadpis, odstavec, tabulku) je stránka rozdělena do bloků. Pokud typ dvou po sobě následujících bloků je shodný, dojde ke spojení těchto bloků.

Dalším zaměřením je analýza linků. Relativní linky je nutné převést na absolutní, buď pomocí aktuální URL nebo pomocí specifikované jiné URL adresy (element <base>). Po převedení je nutné testovat, zda-li nelze vytvořenou adresu zkrátit, někdy může být uvedena značka pro návrat do nadřazeného adresáře uprostřed linku (např. http://server/dir1/../dir2/soubor).

Filtr musí některé informace (z elementu <head>) uložit před samotný XML dokument, ale některé i do XML dokumentu. To je vyřešeno přes buffer, pokud není instanciován XML výstup, všechny výstupy do XML dokumentu se ukládají do paměti a do XML dokumentu jsou uloženy až v okamžiku, kdy je výstupní objekt instaciován.

5.1.f Ostatní filtry

Pro účely systému byly ještě implementovány filtry pro prostý text (text/plain) a pro PDF formát (application/pdf).

5.1.g Omezení filtrů

V tomto odstavci diskutujme známé problémy související s problematikou filtrů.

- **Nezadaným kódování** Týká se všech filtrů, které pracují s textovými formáty. U některých formátů (HTML) může být kódování explicitně definováno, u starších českých webů (psaných převážně v kódování windows-1250) tomu tak ale nebývá. Řešením je použití defaultní hodnoty. To ovšem předpokládá apriorní informace o stránce, které systém neuvažuje. Alternativním řešením je automatická detekce kódování, podle zkušeností [4] je velmi nespolehlivá. Stránky neprojdou vstupní validací (ty vyžadují informaci o kódování) a tedy nebudou zaindexovány.
- **Chybná identifikace typu obsahu** U filtru prostý text je problém se správnou identifikací typu obsahu. Pokud webový server odpovídá na požadavek o soubor s neznámou příponou, vrací defaultní hodnotu typu obsahu. Ta bývá nastavena právě na typ text/plain. Příkladem jsou externí funkce pro program Matlab, tzn. m-file a některé méně známé koncovky. Částečným řešením je analýza přípon souboru.

5.2 Volání funkcí crawleru

Zde uvedeme syntax volání funkcí popsaných v předchozím odstavci. Všechny dotazy jsou definovány v rámci balíčku downloader.

Kompletní příkaz pro stažení a následnou indexaci odpovídá dotazu, který se skládá ze kopie z cache webových stránek, downloadu (resp. aktualizace) stránek z webového serveru, uložení aktualizovaných souborů do cache, transformace URL/URI adres a vlastní indexace. Jednotlivé části na sebe navazují, některé lze vynechat.

5.2.a Kopie stránek z cache

Pokud již stránky byly jednou zpracovány a byly uloženy v cache stránek, je vhodné je před novým stažením načíst do pracovního prostoru dotazu. Tím zaručíme, že nebudeme znovu stahovat a indexovat stránky, které se od posledního stažení nezměnily (změna stránek je definována jako změna atributu Last-Modified v HTTP hlavičce nebo jeho neuvedení u dynamicky generovaných stránek).

Popis syntaxe

downloader: <copyfromcache><domain>domain</domain> [<domain>...</domain> [...]]</copyfromcache>

Všechny stránky, které odpovídají v elementech <domain> zadaným doménám, budou zkopírovány do pracovního prostoru dotazu. Domén lze v rámci jednoho dotazu uvést více. Absence domény v cache nezpůsobí chybu, taková doména je ignorována.

Ukázka

Zkopíruj do pracovního prostoru dotazu stránky z domény dce.felk.cvut.cz.
downloader: <copyfromcache><domain>dce.felk.cvut.cz</domain></copyfromcache>

5.2.b Stažení stránek

Pro další zpracování stáhneme stránky do pracovního prostoru dotazu. Je-li před stažením provedena kopie z cache stránek, jedná se o aktualizaci.

Popis syntaxe

downloader: <download maxlevel="@maxlevel"><url>url</url> [<url>...</url> [...]]</download>

Atribut @maxlevel udává maximální hloubku stažení stránek. Vnořený element <url> specifikuje URL adresu (je ale vhodnější použít URI adresu) úvodní stránky. Adresa je uvedena bez označení protokolu. V rámci jednoho příkazu download může být stahováno více úvodních stránek.

Ukázka

Dotaz pro stažení stránek do úrovně 3 z katedrálního webu http://dce.felk.cvut.cz
downloader: <download maxlevel="3"><url>dce.felk.cvut.cz/pub/index.php</url></download>

Poznámky

Jedná-li se o aktualizaci, stažení stránek probíhá jen v rámci zadaných domén.

5.2.c Kopie stránek do cache

Jak již bylo uvedeno v odstavci 5.2.a, je vhodné udržovat cache stránek. Následující dotaz provede zkopírování stránek z pracovního prostoru dotazu do cache stránek

Popis syntaxe

downloader: <copytocache/>

Element nemá žádné atributy.

5.2.d Transformace adres

Podle odstavce 5.1.b definujme pravidla pro transformaci URL adres na URI adresy. Pravidla typu redirect provádí přesměrování, pravidlo typu varlist pak výčet přípustných parametrů.

Popis syntaxe

```
downloader: <urltransform>
      <redirect base="@base">uri</redirect> [...]
      <varlist base="@base">
            [<var name="@name1"/>]
            [<var name="@name2"> <value>val1</value> [<value>valn</value>]</var>]
            [<var name="@name3">optional</var>]
            [...]
      </varlist>
      [...]
</urltransform>
```

Atribut @base udává URL (URI) adresu stránky, které se transformace týká.

U pravidla typu redirect probíhá přesměrování z URL adresy zadané atributem @base na URL adresu zadanou hodnotou elementu. Pravidel může být uvedeno v rámci jednoho dotazu více.

Pravidlo varlist podle citovaného odstavce pokrývá 3 alternativy. První, parametr @name1 musí být u URL adresy odpovídající skriptu @base uveden, může nabývat libovolné hodnoty. Druhá alternativa, parametr @name2 musí být uveden a musí nabývat hodnot zadaných v elementech value, tedy 'val1' až 'valn'. Poslední alternativou je nepovinný výskyt parametru @name3.

Použije-li se dotaz urltransform bez dalších elementů, provede se jen seřazení parametru URL adres podle abecedy.

Ukázka

V rámci katedrálního webu http://dce.felk.cvut.cz/pub proveď přesměrování ze zadané URL adresy (která je dle 5.1.a automaticky doplněna na index.html) na index.php a pro tento skript definuj pravidla tak, že parametr idpage je povinný, nabývající libovolné hodnoty, povinný parametr language nabývající hodnot en a cz a parametr version, který musí být roven hodnotě 1 (parametr udává, zda-li se jedná o textovou nebo grafickou verzi stránek). Volitelným parametrem je parametr include, který specifikuje příp. externí skripty

```
downloader: <urltransform>
      <redirect base="dce.felk.cvut.cz/pub/index.html">dce.felk.cvut.cz/pub/index.php</redirect>
      <varlist base="dce.felk.cvut.cz/pub/index.php">
            <var name="idpage"/>
            <var name="language"> <value>cz</value><value>en</value></var>
            <var name="version"> <value>1</value></var>
            <var name="include">optional</var>
      </varlist>
</urltransform>
```

Tento příklad u URI adres provede

- **.../index.php?idpage=1&language=cz&version=1** vyhovuje, bude ponechána.
- **.../index.php?idpage=1&language=us&version=1** bude označena jako ignorovaná (neplatná hodnota parametru language).
- **.../index.php?idpage=1&language=cz** rověž ignorovaná (chybějící parametr version).
- **.../index.php?idpage=1&language=cz&version=1&menu=12** bude přesměrována do stránky .../index.php?idpage=1&language=cz&version=1 (parametr menu je ignorován).

Poznámky

Diskutujme poslední případ z příkladu, kdy parametr menu není uveden v seznamu. Pak stránce s URL bez tohoto parametru odpovídají URL se shodnými hodnotami parametrů, ale bez parametru menu. Problém nastává v případě, kdy parametr ovlivňuje obsah stránky (tj. stránky nejsou shodné). V tomto případě bude naindexována libovolná stránka odpovídající kombinaci v seznamu zadaných parametrů.

Jako sekundární výstup dotazu je orientační počet souborů, které se budou indexovat.

5.2.e Crawlerizace a indexace

Poslední částí je transformace stránek stažených do pracovního prostoru dotazu. Dotaz orientačně spočte počet stránek, které se budou indexovat, tento krok je přeskočen v případě, kdy byla provedena trasformace URL adres (kdy je počet souborů počítán). Stránky se prochází hierarchicky od zadané stránky (stránek), každá stránka s novějším datem, než který je uložen v databázi, se zcrawlerizuje a pro každou hloubku se vytvoří SQL dotaz, který se následně provede. Pokud stránka je označena jako ignorovaná, uloží se tato informace do databáze.

Popis syntaxe

downloader: <crawler mode="@mode"><url>url</url>.... </crawler>

Pomocí atributu @mode lze specifikovat, které vlastnosti stránek se budou ukládat. Každému módu odpovídá soubor v konfiguračním adresáři systému, v adresáři crawler. Atribut může nabývat hodnot basic (titulek stránky a odkazy), full (plná indexace obsahu a odkazů) a benchmark (plná indexace obsahu a odkazů s časovými značkami v logu).

V elementu <crawler> je nutné do elementu <url> zadat URI (ve tvaru, jaký vrací případná transformace nebo ve tvaru, v jakém bylo zadáno v dotazu download, sekce 5.2.b), jednu nebo více adres. Adresa je uvedena bez protokolu.

Ukázka

Crawlerizujme a indexujme stránky katedrálního webu.

```
downloader: <crawler mode="full"><url>dce.felk.cvut.cz/pub/index.php</url></crawler>
```

Poznámky

V případě, že používáme transformaci URL adres a užíváme filtru parametrů, je při zadávání adres nutné zvolit takovou URL adresu, která není ignorovaná. V opačném případě není indexována žádná stránka, zadaná stránka je označena jako ignorovaná a protože ignorovaná stránka nemá žádné informace o odkazech, je tato stránka i stránkou poslední.

5.2.f Příklad použití

V předchozích odstavcích byla definována syntaxe jednotlivých dotazů. Vytvořme nyní dotaz, který zašleme systému ke zpracování. Dotaz provede aktualizaci stránek Katedry řídící techniky (http://dce.felk.cvut.cz) a Katedry kybernetiky (http://cyber.felk.cvut.cz) a to do hloubky 4. U první jmenované katedry navíc provede omezení na parametry. Oba weby budou plně zaindexovány.

```
downloader:    <copyfromcache>
      <domain>dce.felk.cvut.cz</domain><domain>cyber.felk.cvut.cz</domain>
</copyfromcache>
<download maxlevel="4">
      <url>dce.felk.cvut.cz/pub/index.php</url><url>cyber.felk.cvut.cz/index.html</url>
</download>
<urltransform>
      <redirect base="dce.felk.cvut.cz/pub/index.html">dce.felk.cvut.cz/pub/index.php</redirect>
      <varlist base="dce.felk.cvut.cz/pub/index.php">
            <var name="idpage"/>
            <var name="language"> <value>cz</value><value>en</value></var>
            <var name="version"> <value>1</value></var>
            <var name="include">optional</var>
      </varlist>
</urltransform>
<crawler mode="full">
      <url>dce.felk.cvut.cz/pub/index.php</url>
      <url>cyber.felk.cvut.cz/index.html</url>
</crawler>
```

5.3 Časová optimalizace

Během vývoje systému došlo k časové optimalizaci SQL dotazů. Zásadním poznatkem, který není uveden v dokumentaci databázového systému [5], je časová optimalizace při volání funkce v podmínce WHERE. Velmi častou konstrukcí u SQL dotazů byla podmínka na shodnost hodnoty identifikátoru s hodnotou vrácenou index vracející funkcí, tedy např. `WHERE id_file = indexoffile('urladresa');`. Ač funkce vrací na záznam nezávislou hodnotu, je volána pro každý záznam. Uvážíme-li, že funkce ve svém těle provádí SELECT, pak testování každého záznamu odpovídá jednomu SELECTu. Řešením je použití vnořeného SELECTu, tj. konstrukci `WHERE id_file = (SELECT indexoffile('urladresa'));`. Touto změnou lze docílit velmi znatelného zrychlení.

6 Další implementované funkce

Popis implementovaných balíčků, dotazů těchto balíčků a algoritmů zpracovávajících tyto dotazy.

Není-li uvedeno jinak, dotazy jsou implementovány v balíčku Downloader. Popis generování map a fulltextového vyhledávání jsou uvedeny u popisu webového rozhraní [4] .

Balíček Downloader sdružuje dotazy týkající se získávání vstupních dat pro další balíčky. Balíček obsahuje vedle dotazů uvedených v předchozí kapitole i dotazy pro následné zpracování informací a dotazy pro ohýbání slov.

Balíček Sengine obsahuje dotazy na práci s fulltextovými daty, především na vyhledávání. Vyjma níže uvedeného dotazu jsou popsány u webového rozhraní.

Balíček Mapgenerator je balíček pro generování map. Popis balíčku je opět uveden u webového rozhraní.

6.1 Ohýbání slov

Dotaz wordfigure

Dotaz wordfigure prohledává seznam slov končící zadanou posloupností znaků (koncovkou), která nemají vyplněný atribut udávající základní tvar slova, ale mají povolůenu jeho automatickou detekci. Testuje, zda-li neexistuje záznam o slově bez této koncovky (příp. bez této koncovky, ale s explicitně definovanou základní koncovkou). Existuje-li, je u tohoto záznamu modifikováno identifikační číslo základního tvaru slova.

Popis syntaxe

```
downloader:<wordfigure extension="@extension" [basicextension="@baseextension"] />
```

Dotaz vyhledá všechna slova končící na koncovku `@extension` a slova začínajících shodným základem (příp. základ rozšířený o základní koncovku `@baseextension`). U každé takové dvojice nastaví u slova s příponou identifikátor základního tvaru slova na nalezené slovo.

Teorie

Pro zjednodušení výkladu definujme množinu všech slov **W**, symbol `(z,k)`, který reprezentuje slovo se základem `z` a koncovkou `k`.

Ohebná slova (slova příbuzného charakteru) mají u pravidelných slov shodný základ slova a liší se pouze koncovkou. Některá slova mohou mít základní tvar se základní koncovkou, pak při porovnávání musíme tuto koncovku uvažovat.

Vytvořme tedy dvojice slov $[w_1,w_2]$, $w_{1,2} \in \mathbf{W}$, kde $w_1 = (z,k_1)$ a $w_2 = (z,k_2)$ (slovo má v základním tvaru koncovku) nebo $w_2 = z$ (základ slova odpovídá slovu v základním tvaru).

Algoritmus pak pro všechny tyto dvojice nastaví u slova w_1 identifikátor základního tvaru na w_2.

Některé dvojice slov ovšem významově neodpovídají, odpovídají pouze tvarem. U takových slov w_1 je nutné automatické zjišťování základního tvaru zakázat.

Implementace

Popsaný algoritmus na úrovni databáze zpracovává funkce text wordwithextension (text,text). Podle nutnosti zpracovávat navíc základní příponu se algoritmus z důvodu časové optimalizace (operace s přidávanou základní příponou je pomalejší) dělí do dvou větví. Vstupní slova s hledanou příponou musí mít identifikátor základního tvaru nevyplněný a musí mít povolenu automatickou detekci.

Pro všechna slova splňující výše popsaný test je u slova v rozšířeném tvaru nastaveno identifikační číslo základního tvaru.

Funkce text wordwithextension (text,text)

```
CREATE FUNCTION wordwithextension (text,text) RETURNS text AS '
DECLARE
   ext_find    ALIAS FOR $1;
   ext_bas     ALIAS FOR $2;
   lext_bas    integer;
   lext_find   integer;
   row         RECORD;
   cnt         integer;
   ret         text;
BEGIN
   lext_bas:=length(ext_bas);
   lext_find:=length(ext_find);
   cnt:=0;

   IF ext_bas IS NOT NULL THEN
      FOR row IN SELECT basic.id_word AS basic, extend.id_word AS extend
         FROM word basic JOIN word extend ON substring (basic.word,1,length(basic.word)-lext_bas) || ext_find  = extend.word
         WHERE
            basic.id_word IN (SELECT id_word from word WHERE substring (word,length(word)-lext_bas+1,lext_bas)=ext_bas)
            AND extend.id_basicword IS NULL
      LOOP
         UPDATE word SET id_basicword=row.basic WHERE id_word=row.extend;
         cnt:=cnt+1;
      END LOOP;
      ret:= "" basictype=""|| ext_bas;
   ELSE
      FOR row IN SELECT basic.id_word AS basic, extend.id_word AS extend
         FROM word basic JOIN word extend ON basic.word || ext_find  = extend.word
         WHERE extend.id_basicword IS NULL
      LOOP
         UPDATE word SET id_basicword=row.basic WHERE id_word=row.extend;
         cnt:=cnt+1;
      END LOOP;
      ret:="";
   END IF;
   RETURN "<extension type=""||ext_find||ret||"" count="" || cnt ||""/>";
END;' LANGUAGE 'plpgsql';
```

Poznámky

Z takto navržených podmínek pro algoritmus vyplývá, že neuvažujeme hierarchii základů slov, uvažujeme jeden základ pro všechna odvozená slova.

Pro vytvoření pravidel pro všechny koncovky daného jazyka byla implementována untilita ispell2query, která ze zadaného konfiguračního souboru jazyka pro program Ispell (GNU nástroj)

vykopíruje pravidla a převede je do dotazu pro navržený systém.

6.2 Page Rank

Výpočet PageRanku.

PageRank je numerická hodnota [3], která udává kritérium kvality obsahu souboru na základě odkazů na tento soubor a PageRanků těchto souborů.

Popis syntaxe

```
downloader:<pagerank [coeficient="@coeficient"] />
```

Dotaz provede jeden interakční krok výpočtu hodnoty PageRanku. Koeficient je zadán pomocí atributu @coeficient, není-li zadán, uvažuje se hodnota koeficientu podle 0.85.

Teorie

Definujme výpočet PageRanku. PageRank souboru je hodnota odvozená z hodnot PageRanků všech souborů, které na daný soubor ukazují, dělených počtem souborů, na které tyto soubory ukazují. Není-li hodnota PageRanku specifikována, lze zvolit libovolné reálné číslo.

U algoritmu výpočtu PageRanku definujeme koeficient, který udává poměr mezi statickou a okolím ovlivnitelnou částí PageRanku. Definujme tedy PageRank.

Mějme množinu stránek **P** a orientovaný graf **G** uspořádávající tuto množinu tak, že hypertextovému odkazu ze stránky p_1 na stránku p_2 odpovídá orientovaná hrana grafu **G**. Pak hodnotu PageRanku stránky p_i označenou $PR(p_i)$ vypočteme podle vzorce

$$PR(p_i) = (1-d) + d\sum PR_L(p_j),$$

kde d je koeficient, p_j jsou stránky, které mají orientovanou hranu směřující do uzlu stránky p_i. Symbol $PR_L(p_j)$ představuje hodnotu PageRanku stránky $PR(p_j)$ dělenou počtem hran z uzlu p_j vystupujících. Není-li hodnota $PR(p_j)$ dosud definována, je nastavena na pevné libovolné číslo (např. 1).

Průměrná hodnota PageRanku počítaná přes všechny soubory konverguje v interakčních krocích k 1. Aby průběh konvergence byl optimální, je doporučeno volit hodnotu koeficientu d 0.85.

Základní myšlenkou algoritmu je, že na dobrý obsah existuje větší množství (nezávislých) linků a že soubory s dobrým obsahem budou odkazovat na jiné soubory s dobrým obsahem. Soubory, které ukazují na jiné soubory, rozdrobí rovnoměrně své příspěvky k PageRankům souborů, na které ukazují.

Algoritmus PageRanku poprvé užil Google [12], dnes existuje spousta algoritmů založených na tomto původním algoritmu. Existuje několik způsobů, jak lze podmnožinu souborů zvýhodnit nad ostatními (link farmy), většina implementací PageRanku se těmto praktikám brání a pokutuje tyto stránky. Tyto techniky už jsou nad rámec této práce.

Protože hodnota PageRanku je velmi důležitá pro pořadí souborů vrácených fulltextovým strojem, jsou konkrétní implementace algoritmů předmětem obchodních tajemství. Proto je v systému implementována pouze základní verze PageRanku plně postačující pro tyto účely.

Z výše uvedeného také vyplývá, že pořadí souborů vrácených externím fulltextovým strojem spíše závisí na hodnotě PageRanku než na míře splnění specifikovaného dotazu. To byl jeden z důvodů vytvoření vlastní fulltextové báze dat.

Implementace

Aktualizace hodnot PageRanku celá probíhá na straně databáze, je implementována pomocí databázové funkce float pagerank(integer,float). Tato funkce vypočte novou hodnotu PageRanku stránky dané prvním parametrem, druhý parametr je hodnota koeficientu d. Výpočet probíhá podle vzorce uvedeného v teoretické části.

Funkce float pagerank(integer,float)

```
CREATE FUNCTION urlpagerank(integer,float) RETURNS float AS '
SELECT (1 - $2) + $2*SUM(CASE WHEN sourcefile.pagerank IS NULL THEN 1 ELSE sourcefile.pagerank END
    /(SELECT count(*) FROM link WHERE id_linksource=sourcefile.id_file))
    FROM file file JOIN link ON id_linktarget=file.id_file JOIN file sourcefile ON id_linksource=sourcefile.id_file WHERE file.id_file= $1;
' LANGUAGE 'sql';
```

6.3 Word prefix

Pro další analýzy je vhodné zjistit četnost výskytu slov v nejbližším okolí slova. Toho lze využít při strukturializaci výsledku. Uveďme modelový příklad z kapitoly 1.2.1, kdy hledáme slovo systém, a na základě analýzy výskytu slov v okolí tohoto slova rozdělíme stránky vrácené fulltextovým strojem na stránky, které navíc obsahují i slovo dynamický, a nebo slovo statický. To vede na informační mapu na základě obsahu.

Dotaz pro četnost výskytu slov v okolí slova je implementován v rámci balíčku Sengine.

Popis syntaxe

```
sengine:<wordprefix between="@between">word<wordprefix>
```

Dotaz vrátí seznam slov doplněný o relevanci, který se vyskytuje v maximální vzdálenosti dané atributem @between od slova uvedeném v elementu. Relevance je vypočtena jako součet inverzních hodnot vzdáleností od zadaného slova.

Teorie

Z fulltextové báze dat vyber takové dvojice slov $[w,w_i]$, kde w je slovo podrobené analýze a w_i jsou slova, která jsou v záznamech fulltextové báze vzdálena nanejvýše o vzdálenost m (parametr @between). Nejsou-li slova v základním tvaru, jsou na něj převedena. Vzdálenost $L(w,w_i)$ definujeme jako počet slov mezi slovy w a w_i zvětšený o jedna.

Výsledkem dotazu je množina dvojic $[w_i,R(w_i)]$, kde $R(w_i)$ je relevance. Dle předchozího textu

$$R(w_i) = \sum \frac{1}{L(w, w_i)}.$$

Implementace

Dotaz je implementován pomocí tabulkové funkce fulltextwordprefix, která vrátí v teoretické části popsaný výsledek, který je převeden do formátu XML.

Funkce setof(int4, numeric) fulltextprefixword(integer,integer)

```
CREATE FUNCTION fulltextprefixword(integer,integer) RETURNS SETOF RECORD AS 'SELECT
      CASE WHEN word.id_basicword IS NULL THEN word.id_word ELSE word.id_basicword END AS returnid,
      SUM(1.0/abs(basicword.position-prefword.position)) as relevancy
   FROM fulltextsearchword($1, true,~1) basicword
      JOIN fulltextindex prefword ON basicword.id_block=prefword.id_block AND abs(basicword.position-prefword.position) < $2 AND
basicword.position<>prefword.position
```

```
    LEFT JOIN word word ON prefword.id_word=word.id_word
GROUP BY returnid;' LANGUAGE 'sql';
```

Ukázka

V kapitole 5.2.f je uveden dotaz pro aktualizaci stránek Katedry řídící techniky a kybernetiky. Nyní najdeme prefixy slova katedra do maximální vzdálenosti 5.

```
sengine:<wordprefix between="5">katedra<wordprefix>
```

Dle očekávání dataz vrátil slova řídící, kybernetika a technika. (slova jsou v základním tvaru).

6.4 Cleaner

Protože se během provozu hromadí data jak v dočasných pracovních adresářích, tak v databázi (např. chybová hlášení), je vhodné tato data průběžně mazat. K tomu slouží dotaz cleaner, který odstraní veškerá dočasná již nepotřebná data, která jsou starší nežli čas posledního spuštění dotazu.

Teorie

Každý dotaz si v adresáři do pracovních adresářů vytvoří soubory, které jsou pro výpočet potřebné. Po ukončení dotazu jsou tyto soubory v pracovním adresáři ponechány, což prakticky umožňuje administrátorovi (obecně uživateli s přístupem k souborovému systému) zpětnou analýzu dotazu z logovacích souborů.

Naopak ve frontě dotazů se hromadí záznamy s výsledky přímých dotazů (neuložených v cache dotazů) a dotazy, které skočily chybovým stavem. To vede k procházení více záznamů, nežli je bezpodmíněčně nutné.

Na straně druhé potřebujeme uchovat dostatečně dlouhou historii dotazů (aby se uživatel dozvěděl, že jím volaný dotaz byl ukončen chybovým stavem). Řešením je mazání jen těch dat, která odpovídají době před posledním volání mazacího dotazu.

Popis syntaxe

```
downloader:<cleaner/>
```

Dotaz nemá žádné parametry.

Poznámky

Z výše uvedeného je patrné, že dotaz je navržen na použití jako periodicky se aktualizující.

7 Závěr, benchmarky

V poslední kapitole je zrekapitulován rozsah implementace systému, diskutovány vlastnosti a prezentovány výsledky pomocí bechmarku na testovaných strojích.

Protože významnou část tvoří dotazy vytvořené v navazující diplomové práci [4], byl systém navržen jako modulární, což se v praxi osvědčilo. Tato práce zajišťuje stažení a analýzu stránek, dotazy jsou implementovány v balíčku Downloader (funkce popsány v kapitole 5 a 6) a vybrané dotazy v jiných balíčcích (vzhledem k podobné funkcionalitě). Dalším aspektem je jednoduchá konfigurovatelnost systému, které bylo využito v závěru práce, kdy bylo potřeba přidat možnost apriorní informace o struktuře katedrálního webu, protože některé stránky se díky různým URI objevovaly vícekrát.

7.1 Stránky Katedry řídící techniky

Dle zadání systém musí umět vytvořit mapu katedrálního webu. Webové stránky Katedry řídící techniky strukturou patří k velmi složitým. Jsou plně dynamicky generované, navíc se rozpadají v rámci předmětů vypsaných katedrou na podweby. Obsahují textovou a grafickou verzi a českou a anglickou mutaci stránek, přičemž vztah mezi mutacemi není triviální (např. v jednom jazyce může být stránka HTML dokumentem a v druhé PDF). Indexaci ztěžuje i fakt, že některé hodnoty parametrů skriptů primárně neovlivňují obsah (např. podmenu v textové verzi).

Z výše uvedených příkladů byl ve finální fázi vytvořen dotaz na pravidlovou transformaci URL adres, pomocí kterého byly nežádoucí vlivy potlačeny. S pravidly byl seznámen i webmaster katedry, který pravděpodobně bude v budoucnu systém spravovat.

Systém byl použit i během restrukturalizace katedrálních stránek, kdy pomohl analyzovat neplatné odkazy.

Jediným přetrvávajícím problémem indexace katedrálního webu je absence atributu, který udává poslední změnu obsahu stránky. Ten standardně není u žádných dynamicky generovaných stránek, ale tam, kde je to možné, je vhodné jej explicitně uvést. Absence tohoto atributu způsobuje, že se během aktualizace indexuje celá dynamicky generovaná část webu znovu.

Některé nejednoznačnosti, které byly na katedrální webu systémem objeveny, byly opraveny (autor byl součástí týmu webmastera).

V příloze D je uvedena hierarchická mapa webových stránek Katedry řídící techniky získaná pomocí dat z této práce. Mapu vygeneroval systém ve webovém rozhraní [4].

7.2 Benchmarky

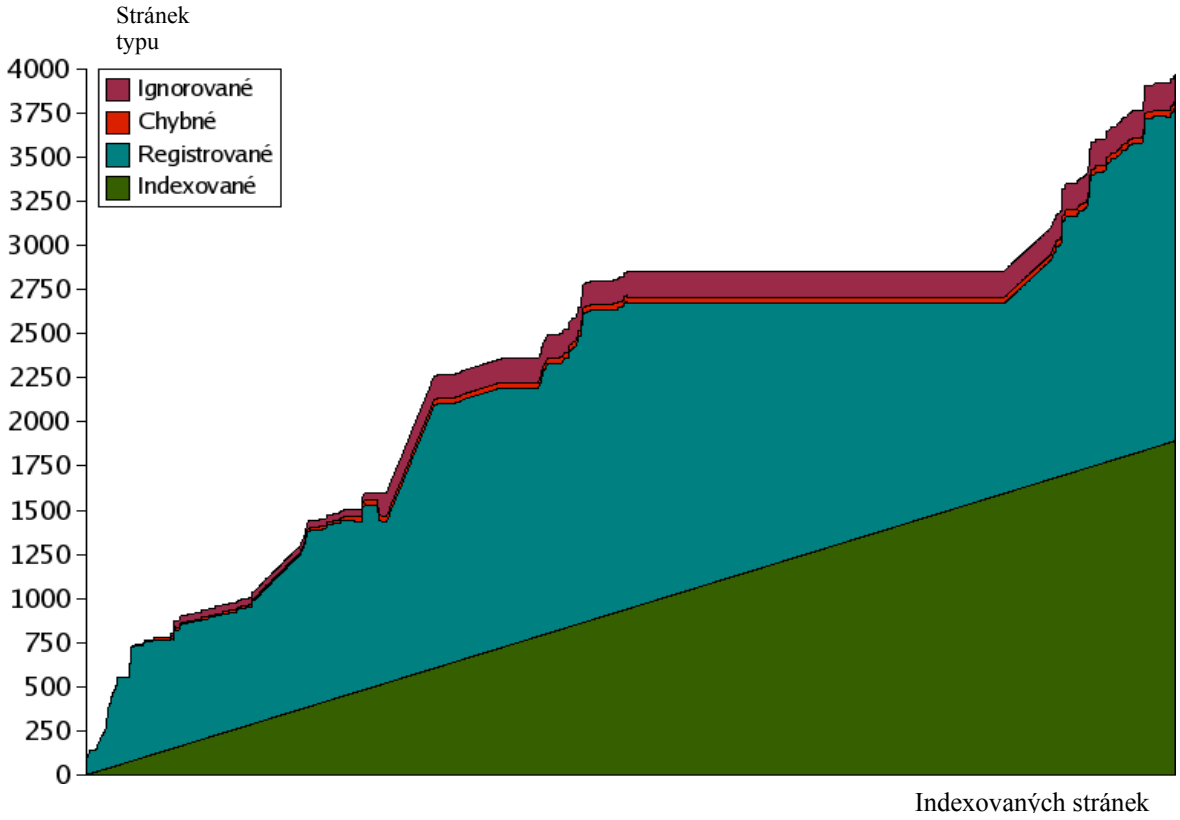
Systém byl testován na několika strojích a několika linuxových systémech. Na všech konfigurací se ukázal funkční.

Test probíhal na těchto počítačích:

- Pentium, 75MHz; 24MB RAM; Red Hat 7.3
- Pentium II, 330MHz; 128MB RAM; SuSE Linux 8.1
- Athlon, 1GHz; 256MB RAM, DDR 133MHz; SuSE Linux 9.0
- Pentium 4, 2.4 GHz; 1024MB RAM, DDR; Mandrake

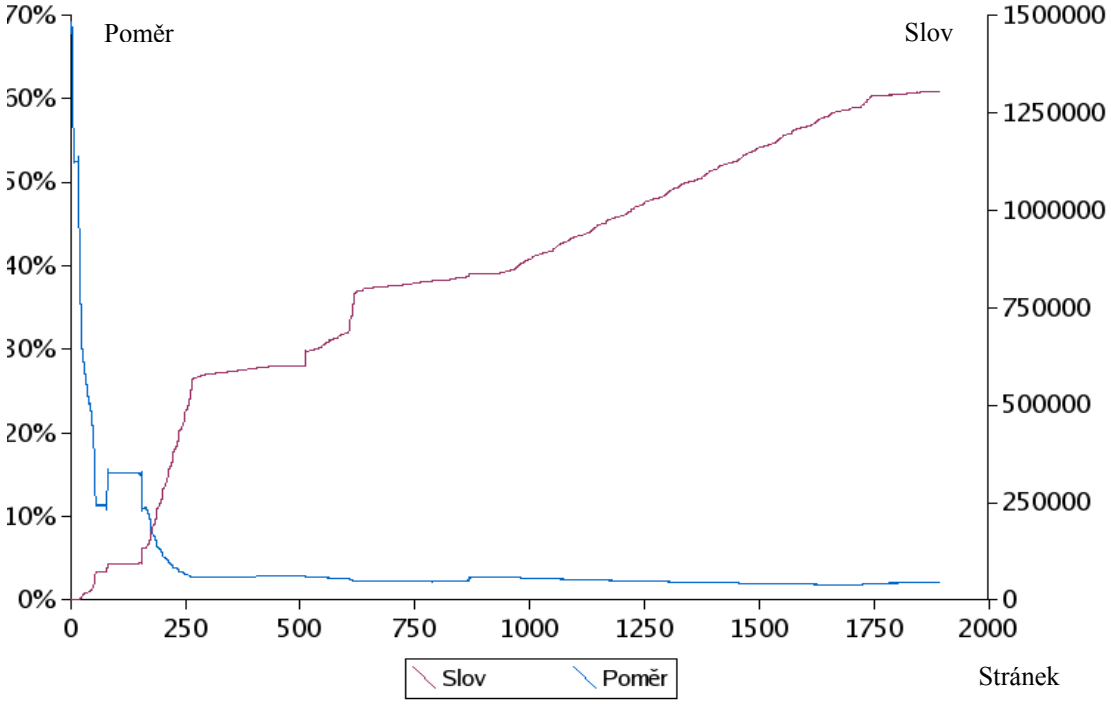
Předmětem testu byla crawlerizace a indexace stránek Katedry řídící techniky na všech uvedených strojích. Pro prezentaci benchmarků byl vybrán třetí stroj, Athlon 1GHz.

Na grafu 7.1 je uveden průběh poměru stránek podle jejich statusu. Z obrázku je vidět nárůst počtu registrovaných stránek. Počet ignorovaných stránek se od určité hranice dále nezvětšuje, jedná se o textové verze dynamicky generovaných stránek. Na katedrálním webu je i několik nenaindexovatelných stránek, pravděpodobná příčina byla popsána v kapitole 5.1.g.



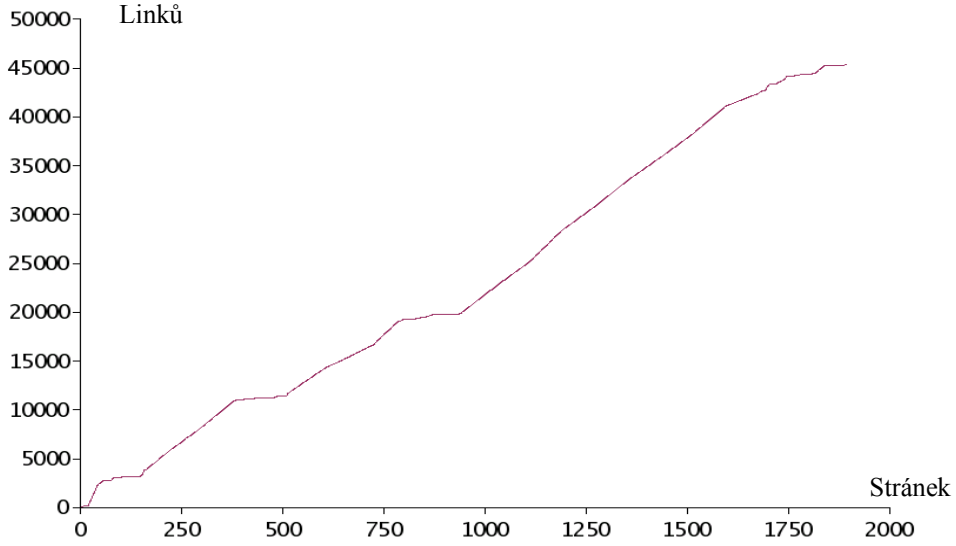
Obr 7.1. Průběh poměrů stránek podle jejich statusu.

Graf 7.2 ukazuje vývoj celkového počtu slov a poměru jedinečných slov k této veličině. Z grafu je patrné, že jednotlivé stránky obsahují různý počet slov, v opačném případě by byl nárůst lineární. Zajímavý je vývoj poměru jedinečných slov k počtu slov indexovaných, dle předpokladu se slova velmi často opakují a poměr stále klesá. Opačný trend by nastal při indexování stránek napsaných v jazyce, jehož slova zatím indexována nebyla.



Obr 7.2. Počet naindexovaných slov a poměr jedinečných ku indexovaných v závislosti na počtu naindexovaných stránek.

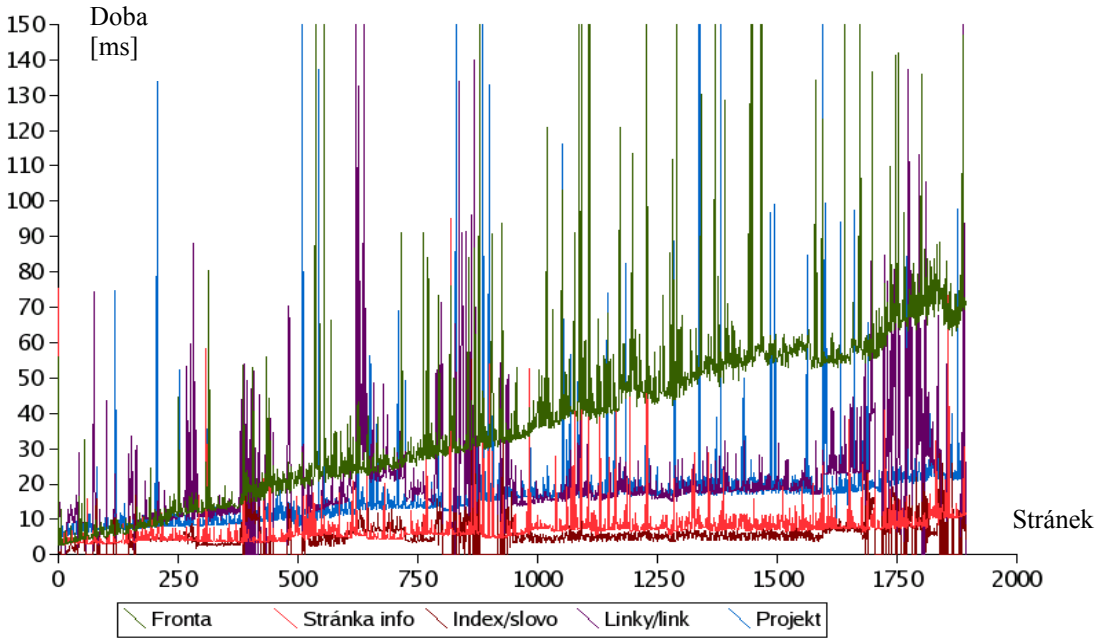
Pro analýzu linků je důležitý vývoj jejich počtu v závislosti na počtu indexovaných a nahraných stránek. Přibližně lineárně rostoucí průběh je zobrazen v grafu 7.3. U určování výpočetní složitosti algoritmů pracující s linky lze místo počtu linků pracovat s počtem naindexovaných stránek.



Obr 7.3. Počet linků v závislosti na počtu naindexovaných stránek.

Poslední graf (7.4) ukazuje dobu potřebnou na provedení indexace stránek. Z grafu je patrné velké zašumění, které je způsobeno různým aktuálním vytížením stroje, na stroji běžely jiné aplikace, byť nebyly po dobu testování využívány. Doba na vytvoření linku a zaindexování slova je vztáhnuta na jeden element, v grafu je tedy zobrazena průměrná doba potřebná na vložení jednoho záznamu.

Dolní mez doby operací má lineární charakter. Největší nárůst je u doby aktualizace záznamu ve frontě, kdy je ukládán údaj o počtu již zpracovaných stránek. Tato hodnota bude shodná i pro ostatní dotazy. Průběh ukazuje nutnost použití čístících mechanizmů popsaných v kapitole 6.4.



Obr 7.4. Bechmark indexace stránek. Doba v ms.

7.3 Celkové shrnutí

Použití systému se v praxi již jednou osvědčilo při restrukturalizaci stránek katedry v září 2003. Systém byl navržen obecně, během testování byly indexovány i další weby, statické i dynamické, v jednom případě i vícejazyčné (cca 20 jazyků). Během testování nebyly zjištěny žádné nedostatky.

Většina testů byla provedena na stroji s procesorem Pentium II s taktovací frekvencí 333MHz, operace trvaly delší dobu, ale vždy končily úspěchem. Propojení databázového modelu s akční částí (kapitola 4) bylo testováno a laděno na stroji s procesorem Pentium (takt 75 MHz), opět bez havárií. Lze tedy usuzovat, že systém je dostatečně robustní.

Systém je navržen i s ohledem na bezpečnost, omezuje možnost útoku typu přetížení služby (DoS), na který je z principu náchylný. Celková bezpečnost je dána konfiguračními balíčky dotazů, jsou-li vhodně nastavena přístupová práva k dané části souborového systému, lze systém považovat za bezpečný.

7.4 Další využití systému

Předpokládá se uvedení systému do praxe, jiná služba generování map přístupná přes serverové řešení mám není známa. Použitím XML technologie se výrazně zjednodušuje možnost integrace mapy do designu stránek.

Systém bude patrně dále rozšiřován, ve zkratce jmenujme alespoň některé možnosti.

- **Automatické generování RSS souborů** RSS soubory obsahují informace o novinkách na webových stránkách. Pokud budou stránky v systému pravidelně aktualizovány, lze RSS soubor automaticky generovat v zadaném časovém intervalu.
- **Automatický katalog** Systém obsahuje základní nástroje pro vytvoření automaticky generovaného katalogu na základě výskytu klíčových slov.
- **CRM systém** Při propojení webového serveru lze plně využít navržený CRM blok a poskytnout majiteli stránek informace o návštěvnosti jednotlivých sekcí stránek a případně na základě analýz přizpůsobit stránky koncovým uživatelům.
- **Zdroj dat pro další systém** Jelikož komunikace probíhá ve formátu XML, je jednoduché systém připojit na webové služby a poskytovat výsledky dotazů jiným aplikacím.
- **Převod dynamických stránek na statické** Většina dnešních vyhledávačů obchází dynamické stránky, proto je vhodné je s použitím apriorní informace o parametrech převést na stránky statické.

A Instalace

Popis instalace databázového serveru PostGreSQL, XSLT procesoru Sablotron a vlastního systému.

Pro správnou funkci celého systému je nutné mít nainstalován databázový server PostgreSQL verze 7.3 a vyšší. Server musí být nainstalován se všemi hlavičkovými soubory, protože jsou nutné při kompilaci přídavných modulů k databázi. Pro konverzi formátů XML na jiné je nutné mít nainstalován XSLT procesor, v demonstrační instalaci je použit Sablotron verze 0.95. Lze použít jakýkoli jiný, pak v konfiguračních souborech nahradíme jméno parseru z sabcmd (Sabloton) za příslušné jiné.

Návod předpokládá alespoň základní dovednosti v systému Linux.

A.1 Instalace PostgreSQL

Zde je popsán postup instalace databázového serveru PostgreSQL ze zdrojových kódů.

Archív obsahující zdrojové kódy lze stáhnout ze serveru http://www.postres.org v sekci download, stáhneme aktuální verzi na lokální počítač.

Nejprve se zdrojové kódy rozbalí do předpřipraveného adresáře. Před začátkem instalace je doporučeno si přečíst návod na instalaci (soubor INSTALL) a seznámit se s možnými rozšiřujícími nastaveními (příkaz ./configure --help).

Nejdříve je potřeba zkonfigurovat soubory pro řízení průběhu kompilace (Makefile) pomocí příkazu ./configure. Příklad může vypadat následovně:

```
[bash] ./configure --prefix=/usr/local/pgsql --datadir=/data/postgres --enable-nls=cz
```

Pokud příkaz proběhne bez chybových hlášení, můžeme pokračovat kompilací aplikace:

```
[bash] make
```

Nepovinně můžeme provést kontrolu konfigurace databázového serveru. Otestují se základní databázové operace.

```
[bash] make check
```

Kompilované části jsou zkopírovány do zadaných adresářů.

```
[bash] make install
```

Posledním krokem souvisejícím s kompilací je kopie hlavičkových souborů potřebných pro další vývoj.

```
[bash] make install-all-headers
```

Nyní je kompilace hotova. Protože démon databázového serveru nemůže běžet pod privilegovaným uživatelem, je nutné založit nového uživatele, pod jehož privilegii démon poběží. Vytvoříme tedy ve skupině daemon uživatele postgres.

```
[bash] adduser -g daemon postgres
```

Nyní připravíme prostor na ukládání dat. Při konfiguraci lze specifikovat adresář, ve kterém data budou uložena (--datadir=/data/postgres). Neexistuje-li, vytvoříme jej.

```
[bash] mkdir /data/postgres
```

Protože databázový server musí být vlastníkem těchto adresářů, změníme uživatele.

```
[bash] chown -R /data/postgres postgres:daemon
```

Nalogujeme se jako uživatel postgres.

```
[bash] su postgres
```

Vytvoříme primární databázové struktury pomocí příkazu initdb.

```
[bash] /usr/local/pgsql/bin/initdb -D /data/postgres
```

Nyní je databázový server připraven ke startu. Spustíme jej na pozadí pomocí

```
[bash] /usr/local/pgsql/bin/postmaster -D /data/postgres &
```

Nyní můžeme doladit drobnosti. Výstup démonu je vhodné přesměrovat do logovacího souboru a je vhodné nastavit inicializaci systému tak, aby se démon databázového serveru spouštěl automaticky při startu systému.

A.2 Instalace XSLT procesoru

Instalaci XSLT procesoru Sablotron provedeme z RPM balíčku se zdrojovými kódy nebo přímo ze zdrojových kódů, které lze najít na http://www.gingerall.com. Celou kompilaci lze uskutečnit pomocí trojice příkazů

```
[bash] ./configure
[bash] make
[bash] make install
```

A.3 Instalace systému

Systém je distribuován jako archív obsahující dokumentaci, bashové untility, zdrojové kódy a implicitní nastavení systému.

Systém je navržen jako démon běžící pod vlastním uživatelem. Uživatel musí patřit do stejné skupiny, do jaké patří uživatel postgres. Nalogujeme se jako privilegovaný uživatel a založíme uživatele pro démona systému.

```
[bash] adduser -g deamon sengine
```

Příhlásíme se jako uživatel sengine a do domovského adresáře rozpakujeme archív obsahující systém.

```
[bash] su sengine
```

Nyní z archívu nainstalujeme systém, pokud není domovský adresář démona /home/sengine, je nutné modifikovat příslušnou volbu v souboru Makefile. Tím zavedeme adresářovou strukturu s předdefinovanou konfigurací, zkompilujeme potřebné nástroje a inicializujeme databázi.

```
[bash] make install
```

Nyní můžeme systém spustit, příp. ukončit.

```
[bash] ~/sengine start
[bash] ~/sengine stop
```

B Databázové funkce

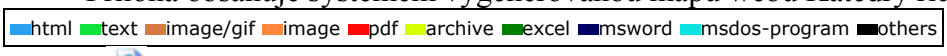
Systém využívá volání databázových funkcí napsaných přímo v jazyce C. Zde jsou deklarovány.

Funkce psané v jazyce C

```
CREATE FUNCTION savetofile(text, text) RETURNS bool AS '/home/sengine/lib/xmllize.so' LANGUAGE 'c';
    CREATE FUNCTION loadfile(text) RETURNS text AS '/home/sengine/lib/xmllize.so' LANGUAGE 'c';
    CREATE FUNCTION filetime(text) RETURNS text AS '/home/sengine/lib/xmllize.so' LANGUAGE 'c';
    CREATE FUNCTION xmlselect (text) RETURNS  text AS '/home/sengine/lib/xmllize.so' LANGUAGE 'c';
    CREATE FUNCTION sengine() RETURNS bool AS '/home/sengine/lib/sengine.so' LANGUAGE 'c';
    CREATE FUNCTION wordsepare(text) RETURNS SETOF RECORD AS '/home/sengine/lib/sengine.so' LANGUAGE 'c';
```

C Webová mapa

Příloha obsahuje systémem vygenerovanou mapu webu Katedry řídící techniky.



- **DCE** (cs) 19.43 kB
  - **DCE** (cs) 19.43 kB
  - **DCE Adresy** (cs) 22.91 kB
  - **DCE Úvod** (cs) 21.33 kB
  - **DCE Struktura** (cs) 22.42 kB
  - **DCE Lidé** (cs) 25.83 kB
    - **DCE Doc Ing Michael Šebek DrSc homepage** (cs) 9.3 kB
    - **DCE Doc Ing Jiří Bayer CSc homepage** (cs) 9.23 kB
    - **DCE Doc Ing Jan Bílek CSc homepage** (cs) 9.35 kB
    - **DCE Dr Ing Zdeněk Hanzálek homepage** (cs) 9.31 kB
    - **DCE Ing Jindřich Fuka homepage** (cs) 9.28 kB
    - **DCE Petra Stehlíková homepage** (cs) 9.3 kB
    - **DCE Prof Ing Jan Štecha CSc homepage** (cs) 9.36 kB
    - **DCE Ing Pavel Burget homepage** (cs) 9.33 kB
    - **DCE horacek homepage** (cs) 6.78 kB
    - **DCE Ing Roman Bartosinski homepage** (cs) 9.21 kB
    - **DCE Ing Miguel Bernal homepage** (cs) 9.23 kB
    - **DCE Ing Milan Cepák homepage** (cs) 9.2 kB
    - **DCE Ing Josef Čapek Ph D homepage** (cs) 9.28 kB
    - **DCE RNDr Sergej Čelikovský CSc homepage** (cs) 9.24 kB
    - **DCE Ladislav Čmelík homepage** (cs) 9.26 kB
    - **DCE Ing Pavel Deutsch homepage** (cs) 9.23 kB
    - **DCE Ing Ondřej Dolejš homepage** (cs) 9.24 kB
    - **DCE Mgr Helena Doležílková homepage** (cs) 9.4 kB
    - **DCE Ing Jiří Fajt homepage** (cs) 9.13 kB
    - **DCE Ing Miroslava Fenclová CSc homepage** (cs) 9.27 kB
    - **DCE Ing Karel Foltýn homepage** (cs) 9.16 kB
    - **DCE Ing Petr Haba homepage** (cs) 9.26 kB
    - **DCE Ing Petr Havel homepage** (cs) 9.2 kB
    - **DCE Doc Ing Vladimír Havlena CSc homepage** (cs) 9.27 kB
    - **DCE Ing Didier Henrion Ph D homepage** (cs) 9.26 kB
    - **DCE Ing Martin Hlinovský homepage** (cs) 9.35 kB
    - **DCE Ing Jaroslav Honců CSc homepage** (cs) 9.29 kB
    - **DCE Ing Petr Honzík homepage** (cs) 9.14 kB
    - **DCE Ing Robert Hornych homepage** (cs) 9.22 kB
    - **DCE Ing Zdeněk Hurák homepage** (cs) 9.19 kB
    - **DCE Ing Petr Hušek Ph D homepage** (cs) 9.27 kB
    - **DCE Doc Ing Kateřina Hyniová CSc homepage** (cs) 9.33 kB
    - **DCE Ing Bernard Jaroš homepage** (cs) 9.22 kB
    - **DCE Doc Ing Jan John CSc homepage** (cs) 9.3 kB
    - **DCE Ing Petr Jurčík homepage** (cs) 9.21 kB

- DCE Ing Jiří Kadlec CSc homepage (cs) 9.22 kB
- DCE Doc Ing Bohuslav Kirchmann CSc homepage (cs) 9.29 kB
- DCE Ing Tomáš Klecker homepage (cs) 9.17 kB
- DCE Ing Jan Krákora homepage (cs) 9.24 kB
- DCE Ing Michal Krákora homepage (cs) 9.2 kB
- DCE Ing Štěpán Kroupa homepage (cs) 9.22 kB
- DCE Ing Aleš Kruczek homepage (cs) 9.2 kB
- DCE Ing Tomáš Kučera homepage (cs) 9.22 kB
- DCE Prof Ing Vladimír Kučera DrSc Dr h c homepage (cs) 9.37 kB
- DCE Ing Martin Langer homepage (cs) 9.15 kB
- DCE Ing Miroslav Líčko homepage (cs) 9.21 kB
- DCE Ing Daniel Marčev homepage (cs) 9.25 kB
- DCE Jaroslava Matějková homepage (cs) 9.29 kB
- DCE Ing Pavel Němeček homepage (cs) 9.24 kB
- DCE Ing Ondřej Novotný homepage (cs) 9.23 kB
- DCE Ing Daniel Pachner homepage (cs) 9.31 kB
- DCE Ing Michal Pajr homepage (cs) 9.17 kB
- DCE Doc Ing Zdislav Pech CSc homepage (cs) 9.32 kB
- DCE Ing Jaroslav Pekař homepage (cs) 9.27 kB
- DCE Ing Pavel Píša homepage (cs) 9.27 kB
- DCE Ing Zdeněk Pohl homepage (cs) 9.13 kB
- DCE Zita Pračková homepage (cs) 9.29 kB
- DCE Ing Renata Pytelková homepage (cs) 9.35 kB
- DCE Ing Mgr Branislav Rehák homepage (cs) 9.34 kB
- DCE Ing Jiří Roubal homepage (cs) 9.27 kB
- DCE Ing Pavel Růžička homepage (cs) 9.22 kB
- DCE Ivan Rybníček homepage (cs) 9.24 kB
- DCE Ing Igor Savič homepage (cs) 9.13 kB
- DCE Ing Petr Smolík homepage (cs) 9.25 kB
- DCE Ing Michal Sojka homepage (cs) 9.21 kB
- DCE Ing Martina Svádová homepage (cs) 9.25 kB
- DCE Ing Zdeněk Šebek homepage (cs) 9.25 kB
- DCE Doc Ing Tomáš Šimek CSc homepage (cs) 9.24 kB
- DCE Ing Radek Šindelář homepage (cs) 9.24 kB
- DCE Ing Přemysl Šůcha homepage (cs) 9.21 kB
- DCE Ing Richard Šusta Ph D homepage (cs) 9.31 kB
- DCE Ing Milan Tichý homepage (cs) 9.14 kB
- DCE Ing Jiří Trinkewitz homepage (cs) 9.25 kB
- DCE Ing Petr Urban homepage (cs) 9.28 kB
- DCE Ing František Vacek homepage (cs) 9.27 kB
- DCE Ing František Vaněk homepage (cs) 9.28 kB
- DCE Jan Veselý homepage (cs) 9.23 kB
- DCE Ing Zdeněk Vlček homepage (cs) 9.22 kB
- DCE Doc Ing Ondřej Vysoký CSc homepage (cs) 9.29 kB
- DCE Ing Libor Waszniowski homepage (cs) 9.23 kB
- DCE Ing Pavel Zezula homepage (cs) 9.21 kB

- DCE Doc Ing Antonín Stříbrský CSc homepage (cs) 9.3 kB
- DCE pilc homepage (cs) 6.75 kB

- DCE Všeobecně (cs) 26.95 kB
- DCE Projekty (cs) 14.54 kB
- DCE Semináře (cs) 51.75 kB
- DCE Knihovna (cs) 19.75 kB
- DCE Všeobecně (cs) 18.85 kB
- DCE Skripta (cs) 20.91 kB
- DCE Pracoviště (cs) 22.39 kB
- DCE Laboratoře Přehled (cs) 23.64 kB
- DCE Publikace 2000 2003 (cs) 140.69 kB
- DCE Publikace 2003 (cs) 31.82 kB
- DCE Publikace 2002 (cs) 38.46 kB
- DCE Publikace 2001 (cs) 36.47 kB
- DCE Publikace 2000 (cs) 33.54 kB
- DCE Publikace 1999 (cs) 37.47 kB
- DCE Publikace 1998 (cs) 27.49 kB
- DCE Publikace 1997 (cs) 22.87 kB
- DCE Publikace 1996 (cs) 21.26 kB
- DCE Publikace 1995 (cs) 17.49 kB
- DCE Publikace 1994 (cs) 17.09 kB
- DCE Publikace 1993 (cs) 14.54 kB
- DCE MCs studium Denní studium (cs) 20.57 kB
- DCE MCs studium Předměty (cs) 42.95 kB
- DCE Práce BP DP sem Pokyny k BP DP (cs) 22.42 kB
- DCE Práce BP DP sem prostř pro říz (cs) 18.77 kB
- DCE Práce BP DP sem řídící systémy (cs) 19.28 kB
- DCE Práce BP DP sem teorie řízení (cs) 18.76 kB
- DCE PhD studium Všeobecně (cs) 20.28 kB
- DCE PhD studium Předměty (cs) 52.83 kB
- DCE PhD studium Směrnice (cs) 48.09 kB
- DCE PhD studium Témata (cs) 45.52 kB
- DCE PhD studium Zkouška (cs) 43.64 kB
- DCE NET info Web DCE (cs) 27.02 kB

- phpinfo 28.66 kB

Seznam literatury

[1] Matthew, N.*Linux - začínáme programovat.* Praha: Computer Press, 2000. Autorizovaný překlad Beginning Linux Programming. 2. vydání. Wrong Press, 1999. ISBN 80-7226-307-2.

[2] Marchal, B.*XML v příkladech.* Praha: Computer Press, 2000. Autorizovaný překlad XML by Example. Que Corporation/Benoit Marschal, 2000. ISBN 80-7226-332-3.

[3] Henzinger, M.R.. Hyperlink Analysis for the Web . *IEEE Internet Computing*. 2001 (January/February). Stránky 45-50.

[4] Živnůstka, J.*PHP rozhraní pro vyhledávání a správu mapy sítě.* Katedra řídící technuky, FEL, ČVUT, 2004. Diplomová práce.

[5] Postgres, Inc.. *PostgreSQL 7.4 Documentation* [online]. Poslední revize 12.11.2003 [citace 08.01.2004]. <http://www.postgres.org/docs/7.4/static/index.html>

[6] Sněhule, P.. *Rozšiřování PostgreSQL v C - Funkce* [online]. Poslední revize 29.11.2002 [citace 08.01.2004]. <http://www.root.cz/clanek/1425>

[7] Sněhule, P.. *Jemný úvod do PL/pgSQL PostgreSQL* [online]. Poslední revize 28.08.2002 [citace 08.01.2004]. <http://prosgres.ok.cz/doc/plpgsql.html>

[8] W3C. *Hypertext Markup Language* [online]. Poslední revize 04.11.2003 [citace 08.01.2004]. <http://www.w3c.org/MarkUp/>

[9] Yves Lafon. *Hypertext Transfer Protocol* [online]. Poslední revize 16.04.2003 [citace 08.01.2004]. <http://www.w3c.org/Protocols/>

[10] Liam Quin. *Extensible Markup Language (XML)* [online]. Poslední revize 20.08.2003 [citace 08.01.2004]. <http://www.w3c.org/XML/>

[11] Mižoch, L. *Open Directory Project neboli DMOZ* [online]. Poslední revize 11.12.2003 [citace 08.01.2004]. <http://interval.cz/clanek.asp?article=2896>

[12] Google. *Aboat Google* [online]. [citace 08.01.2004]. <http://www.google.com/intl/cs/about.html>

[13] Němec, L . *SiteExpert 5.1* [online]. Poslední revize 17.12.2001 [citace 26.08.2002]. <http://www.webtip.cz/art/wt_tech_xmlpdfsvg/sitexpert51.html>