VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VYHLEDÁVÁNÍ INFORMACÍ PRO VĚDECKÉ PORTÁLY

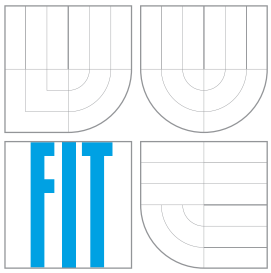
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

JAN ĎULÍK

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VYHLEDÁVÁNÍ INFORMACÍ PRO VĚDECKÉ PORTÁLY

INFORMATION RETRIEVAL IN RESEARCH PORTALS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

JAN ĎULÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MAREK SCHMIDT

BRNO 2010

Zadání diplomové práce/10735/2009/xdulik01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2009/2010

Zadání diplomové práce

Řešitel: **Ďulík Jan, Bc.**
Obor: Informační systémy
Téma: **Vyhledávání informací pro vědecké portály**
Information Retrieval in Research Portals
Kategorie: Web

Pokyny:

1. Seznamte se s problematikou vyhledávání dokumentů s metadaty a problematikou vědeckých portálů.
2. Navrhněte strukturu metadat pro vědecké články a související informace vhodnou pro vědecké portály.
3. Navrhněte uživatelské rozhraní pro vyhledávání nad těmito daty.
4. Implementujte navržený systém jako webovou aplikaci.
5. Shromážděte ukázková data a vyhodnoťte použitelnost výsledného systému.
6. Vytvořte plakát prezentující výsledky práce.

Literatura:

- C. D. Manning, H. Schütze: Introduction to Information Retrieval, Cambridge University Press. 2008
- D. Allen: Seam in Action, Manning. 2008
- D. Smiley: Solr 1.4 Enterprise Search Server, 2009

Při obhajobě semestrální části diplomového projektu je požadováno:

- 1 až 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Schmidt Marek, Ing.**, UPGM FIT VUT
Datum zadání: 21. září 2009
Datum odevzdání: 26. května 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zabývá vyhledáváním informací pro vědecké webové portály se zaměřením na vyhledávání vědeckých publikací. Jsou definovány pojmy související s vyhledáváním informací, klasifikací a reprezentací znalostí. Jsou také představeny existující nástroje pro vyhledávání, z nichž se vychází při vývoji vyhledávače. Dále se práce zabývá návrhem, popisem implementace vyhledávacího rozhraní a shromážděním ukázkových dat. V závěru je provedeno vyhodnocení použitelnosti vytvořené webové aplikace.

Abstract

This paper deals with the information retrieval in research portals with the intention of the retrieval in scientific publications. We define concepts related to the information retrieval, classification and knowledge representation. We also present existing search tools used as the initial inspiration for the design of the search intergace. Futhermore we describe the implementation as well as the process of collecting sample data. In the last chapter we discuss usability of the developed web application.

Klíčová slova

Vyhledávání informací, vědecký webový portál, klasifikace, faseta, fulltextové vyhledávání, uživatelské rozhraní, Java, Solr.

Keywords

Information retrieval, research portal, classification, facet, full text searching, user interface, Java, Solr.

Citace

Jan Ďulík: Vyhledávání informací pro vědecké
portály, diplomová práce, Brno, FIT VUT v Brně, 2010

Vyhledávání informací pro vědecké portály

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Marka Schmidta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Ďulík
24. května 2010

Poděkování

Děkuji Ing. Marku Schmidtovi za podporu během tvorby této práce.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá problematikou vyhledávání informací pro vědecké webové portály. Typický uživatel vědeckého webového portálu je mladý výzkumník nebo doktorand, který hledá relevantní informace v oblasti svého výzkumu. Zájem o informace je uvažován jako dlouhodobý, uživatel by měl být upozorňován na nové publikace, projekty, události atd. Informace z oboru tvořícího rozsah portálu jsou obsaženy v příslušné ontologii [21]. Vědecký webový portál tedy poskytuje informace o všem, co vědec potřebuje pro svou práci: informace o projektech, vědeckých výzkumných skupinách a pracovištích (což mohou být například ústavy nebo univerzity), lidech participujících na výzkumu, publikační činnosti a událostech. Události mohou být různého druhu, uveďme konference, semináře, vědecké kongresy atd. Portál zároveň eviduje vzájemné vztahy a vazby mezi všemi těmito údaji, tak uživatel portálu mohl jednoduše procházet a objevovat informace, které spolu souvisí.

Vědecký webový portál lze rozdělit na více částí. Například na část věnovanou personalizaci, kde každý uživatel portálu má svůj vlastní profil s osobním nastavením, na část, ve které může uživatel měnit údaje poskytované portálem, tedy aktualizovat informace o vývoji výzkumných projektů, zadávat a měnit nové události. Jiná část může sloužit pro zobrazování aktualit. Tato práce se soustředí na část určenou vyhledávání informací ve vědeckých publikacích. Cílem je vytvořit prostředky, které usnadní uživateli nalezení požadované informace, přičemž hlavní důraz je kladen na propracované uživatelské rozhraní využívající fulltextové a fasetové vyhledávání. S tím je také spojena vhodná struktura metadat popisujících publikace. Metadata jsou zde využívána k vyhledávání a propojení publikací s ostatními sekcemi portálu. Uživatel má tak možnost sledovat vazby mezi publikacemi a ostatními informacemi dostupnými na vědeckém webovém portálu.

První kapitola obsahuje definici základních pojmů týkajících se problematiky klasifikace. Dále je v této kapitole uveden souhrnný přehled existujících nástrojů, které jsou určeny k vyhledávání informací ve vědeckých publikacích nebo jejichž uživatelské rozhraní je řešeno inovativním způsobem. Znalosti získané prozkoumáním přístupu k vyhledávání v těchto nástrojích jsou základem pro návrh metadat a uživatelského rozhraní, aby vyvíjený vyhledávač vycházel z osvědčených postupů při vyhledávání informací a zároveň zahrnoval nové prvky zlepšující použitelnost. Následující kapitola seznamuje čtenáře se základními datovými strukturami použitými při vyhledávání informací a také je v této kapitole popsána knihovna Apache Lucene a vyhledávací server Apache Solr. Čtvrtá kapitola se zabývá návrhem metadat a uživatelského rozhraní vyvíjeného vyhledávače. Pátá kapitola seznamuje čtenáře s platformou Java Enterprise Edition a frameworkem Seam, jež byly použity pro implementaci vyhledávače. V šesté kapitole je proveden návrh architektury implementované aplikace a v následující sedmé kapitole je popsána vlastní implementace. Konečně v osmé

kapitole je diskutována použitelnost navržené aplikace a sběr ukázkových dat.

Diplomová práce plně navazuje na předchozí semestrální projekt, z nějž byla převzata druhá až čtvrtá kapitola této práce. Kapitoly věnované návrhu a implementaci webové aplikace vychází z analýzy problematiky, návrhu metadat a uživatelského rozhraní provedených v rámci semestrálního projektu.

Kapitola 2

Existující řešení

Kapitola obsahuje přehled existujících vyhledávacích nástrojů nebo projektů zaměřených na vyhledávání, jejichž vlastnosti jsou buďto typické nebo naopak obsahují inovativní prvky, které netradičním způsobem usnadňují vyhledávání. Nástroje Google Scholar a CiteSeerX jsou zástupci prvé skupiny. Jedná se o prostředky pro vyhledávání informací ve vědeckých článcích. Oproti tomu projekty Flamenco, MSpace a Simile Exhibit nejsou určeny přímo k vyhledávání informací ve vědeckých textech. Jde projekty, které umožňují vyhledávání obecně v libovolných datech, záleží pouze na konfiguraci. Jsou zde uvedeny proto, že přistupují k vyhledávání neobvyklým způsobem a zavádí do uživatelského rozhraní nové prvky zlepšující efektivitu vyhledávání.

Před vlastním přehledem jsou definovány základní pojmy týkající se vyhledávání a klasifikace dokumentů. Na popis funkcionality spojené s vyhledáváním a klasifikací je kladen hlavní důraz v přehledu existujících nástrojů.

2.1 Základní pojmy

Definice 1 *Citační databáze/citační index je označení pro databázi publikací a citací, jež umožňuje určit, které dokumenty cituje určitý dokument nebo naopak k určitému dokumentů vrátí množinu dokumentů, jenž jej citují.*

Definice 2 *Fulltextové vyhledávání je název pro vyhledávání informací v kolekci dokumentů, kdy na základě dotazu ve formě klíčového slova nebo slov je vrácena množina dokumentů obsahující ve svém textu tyto klíčová slova (více viz kapitola 3).*

Definice 3 *Klasifikace znamená třídění, zařazování entit do různých tříd [2].*

Mezi třídami v klasifikaci může být definováno uspořádání. Základní klasifikační struktury jsou taxonomie, folksonomie a fasety.

Definice 4 *Taxonomie je hierarchie s dědičností definovaná nad množinou tříd, tzn. třídy na nižší úrovni jsou specializací tříd na vyšší úrovni a třídy na vyšší úrovni jsou generalizací tříd na nižší úrovni [24].*

Taxonomie je jednodimenzionální klasifikace (klasifikace probíhá na základě jednoho hlediska). Ze zavedené hierarchie vychází omezení, kdy při procházení taxonomie je předem definována posloupnost výběru jednotlivých klasifikací tak, jak jsou definovány v této hierarchii.

Faseta	Hodnoty fasety
Rok vydání	2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009
Typ publikace	článek, příspěvek na konferenci, kniha, diplomová práce, disertační práce
Jazyk	čeština, angličtina

Tabulka 2.1: Příklad faset

Definice 5 *Folksonomie je název pro demokratickou, distribuovanou klasifikační metodu, při které všichni uživatelé systému (nebo mnoho z nich) mohou označit obsah popiskem dle vlastního výběru. Čím více lidí v systému vytváří popisky, tím lépe se ostatním hledají informace [27].*

Pro popisek je častěji používán anglický termín *tag*. Pojem foksonomie vznikl odvozením z taxonomie, jedná se o její opak. Pro foksonomii je typická absence centrální autority a hierarchie. Zpravidla je budována zdola nahoru, tzn. není budována na základě předem dané struktury. Příkladem použití folksonomie v praxi je služba Delicious[1].

Definice 6 *Faseta je množina vzájemně výlučných kategorií, jež dohromady zahrnují všechny entity [10].*

Z jiného úhlu nahlíží na fasetu následující definice:

Definice 7 *Faseta je charakteristická vlastnost nebo aspekt množiny objektů, podle níž může být zdroj klasifikován. Jednotlivé kategorie pak tvoří hodnoty fasety neboli omezení (z ang. constraint) [13].*

Definice 8 *Fasetová klasifikace je označení pro typicky tří a více dimenzionální klasifikaci pomocí faset.*

Definice 9 *Fasetové vyhledávání je způsob vyhledávání, kdy uživatel filtruje množinu entit postupným výběrem vyhovujících hodnot ve fasetové klasifikaci, jinými slovy, definuje omezení. Uživateli je umožněno zadávat omezení faset v libovolném pořadí (na rozdíl od taxonomie) a může také kdykoliv kterékoliv omezení zrušit [13].*

Typické použití fasetového vyhledávání je kombinováno s fulltextovým vyhledáváním. Uživatel nejdříve zadá fulltextový dotaz ve formě klíčových slov a následně vrácenou množinu výsledků dále filtruje pomocí fasetového vyhledávání.

Demonstrujme použití uvedených pojmů na příkladu, kde množina entit je reprezentována kolekcí vědeckých publikací. Tabulka 2.1 obsahuje fasety definované pro účely vyhledávání.

Fasetová klasifikace tvořena fasetami uvedenými v tabulce. Uživatel může provést například následující posloupnost akcí:

1. Uživatel zvolí hodnotu kniha fasety Typ publikace. Množina výsledků se aktualizuje a obsahuje pouze dokumenty, u nichž je typ publikace je kniha.

2. Uživatel zvolí hodnotu 2000 fasety Rok vydání. Množina výsledků se aktualizuje a obsahuje pouze dokumenty, u nichž typ publikace je kniha a zároveň rok vydání je 2000.

[1] `http://del.icio.us`

Aby fasetové vyhledávání poskytovalo skutečně efektivní způsob vyhledávání, musí být splněny následující podmínky [3]:

- kvalitní zpětná vazba: uživatel může interaktivně sledovat, jak definice fasetových omezení ovlivňuje zpracovávanou množinu entit
- neočekávané výsledky a slepý konec při vyhledávání: uživatel dopředu ví, kolik výsledků bude zobrazeno po výběru daného omezení. Pokud by byl po výběru omezení počet výsledků nula, není toto omezení nabídnuto k výběru
- není definována hierarchie: uživatel může přidávat a odebírat omezení v libovolném pořadí (tento požadavek byl uveden už u definice fasetového vyhledávání)

Definice 10 *Metadata jsou data popisující data využívaná v oblastech zabývajících se uchovávám a zpřístupňováním informací [6].*

Typicky jsou metadata spojena s vyhledáváním nebo mohou být nositeli dalších informací spojených s daty. Obvykle bývá uvažováno, že metadata určité entity jsou tvořeny více atributy, kde každý atribut popisuje určitou vlastnost dané entity. Z toho také vychází následující definice:

Definice 11 *Element metadat popisuje informační zdroj nebo pomáhá poskytnout přístup k informačnímu zdroji [9].*

Metadata jsou často automaticky extrahovaná a mohou být tedy chybná. Jestliže jsou použity fasety, je třeba návrh struktury metadat navrhovat s ohledem na plánované fasety. Atributy metadat je vhodné navrhnout tak, aby hodnoty atributů sloužily zároveň také jako hodnoty faset.Hierarchická metadata pak tvoří ontologii.

Definice 12 *Ontologie je explicitní specifikace konceptualizace [11].*

Konceptualizace je pojmenování pro systém pojmů modelující určitou část světa. Explicitní pak znamená to, že sémantika nesmí být skryta, veškeré vztahy jsou popsány formálně a pro jejich pochopení nemusí mít uživatel žádnou předchozí znalost o datech.

2.2 Přehled existující nástrojů a projektů

2.2.1 Google Scholar

Google Scholar[2] slouží k vyhledávání odborné literatury z různých oblastí. Publikace nemusí být explicitně přidávány, kromě ručního zadávaní mohou být také vyhledány crawlerem.

Relevance článků a tím i pořadí zobrazovaných výsledků hledání je stanoveno na základě shody klíčových slov dotazu s textem publikace případně jménem autora. Dále je do hodnocení publikace zahrnuta relevance publikačního zdroje, kde publikace vyšla a kolikrát byla citována v jiné vědecké literatuře. Výsledky vyhledávání lze setřídit také podle nových výzkumů. Třídění pak probíhá na základě času vydání publikace a zároveň je také vzato v potaz hodnocení popsané výše. Výsledky tedy nelze řadit jinak než jedním implicitním způsobem.

[2] `http://scholar.google.com/`

Jedna položka ve výsledcích vyhledávání může reprezentovat více publikací. Může se jednat o soubor článků k určitému výzkumu či projektu nebo se jedná o více dostupných verzí jednoho článku.

U každé položky výsledku je zobrazena řada informací: KWIC (*key word in context*, zobrazování klíčových slov dotazu v relevantních částech nalezených dokumentů) z textu publikace (pokud je volně dostupný), jméno autora, rok publikování, vydavatelská organizace, odkaz na text článku (zpravidla ve formátu PDF). Dále jsou u každého výsledku k dispozici odkazy provádějící nějakou činnost. Je možné zobrazit citaci publikace ve formátu pro import do citačních správců (BibTeX, EndNote aj.) nebo vyhledat související publikace na základě podobnosti s danou publikací. Odkaz s údajem počtu citací odkazujících na danou publikaci zobrazí nové výsledky se seznamem těchto citujících publikací.

Přímo na stránce s výsledky vyhledávání jsou přístupné tři fasety:: rok, typ publikace (články, patenty, právnické texty) a možnost zobrazovat pouze plnohodnotné články nebo také souhrny a nebo i citace. Zahnutí citací znamená, že jako výsledek může být zobrazena pouze citace, na kterou je odkazováno z jiné publikace, a tato citovaná publikace nemusí být dostupná. Hodnoty faset jsou reprezentovány rozbalovacími nabídkami.

2.2.2 CiteSeerX

CiteSeerX[3] je elektronická knihovna s hlavním zaměřením na vědeckou literaturu. Neomezuje se jen na články a publikace, ale zahrnuje i algoritmy, data, služby, software apod. Stejně jako Google Scholar používá fulltextové vyhledávání. U nalezených výsledků zobrazuje KWIC, rok vydání, jméno autora, instituce, publikace nebo konference a počet citací článku z jiných publikací. Opět stejně jako u Google Scholar lze vyhledat publikace, které daný článek citují.

Uživatelské rozhraní neobsahuje fasety, lze jen zvolit způsob řazení výsledků podle relevance, což je hodnota vypočítaná obdobným způsobem jako u Google Scholar, nebo podle roku vydání (sestupně, vzestupně) anebo podle aktuálnosti.

U každé položky výsledků hledání lze zobrazit samostatnou stránku s detailem. Tato stránka obsahuje abstrakt, seznam citací použitých v textu, odkaz na text publikace, graf citací publikace v jednotlivých letech. Kromě toho CiteSeerX podporuje také folksonomii. Uživatel, který může být i anonymní, má možnost přidat k publikaci popisek (tag). Dotazy pak mohou jako jednu ze svých částí obsahovat vyhledání podle popisků.

2.2.3 Flamenco

Flamenco je open-source projekt vyvíjený na Bekeley School of Information[4]. Vyvíjený nástroj umožňuje vyhledávání nad libovolným typem dat (textová, audio, video, obrázky). Vyhledávání je kombinací fulltextového a fasetového a probíhá nad metadaty specifikovanými v konfiguračním nastavení. Každé datové entitě jsou v metadatech přiřazeny kategorie. Množina všech kategorií jednoho atributu metadat tvoří fasetu.

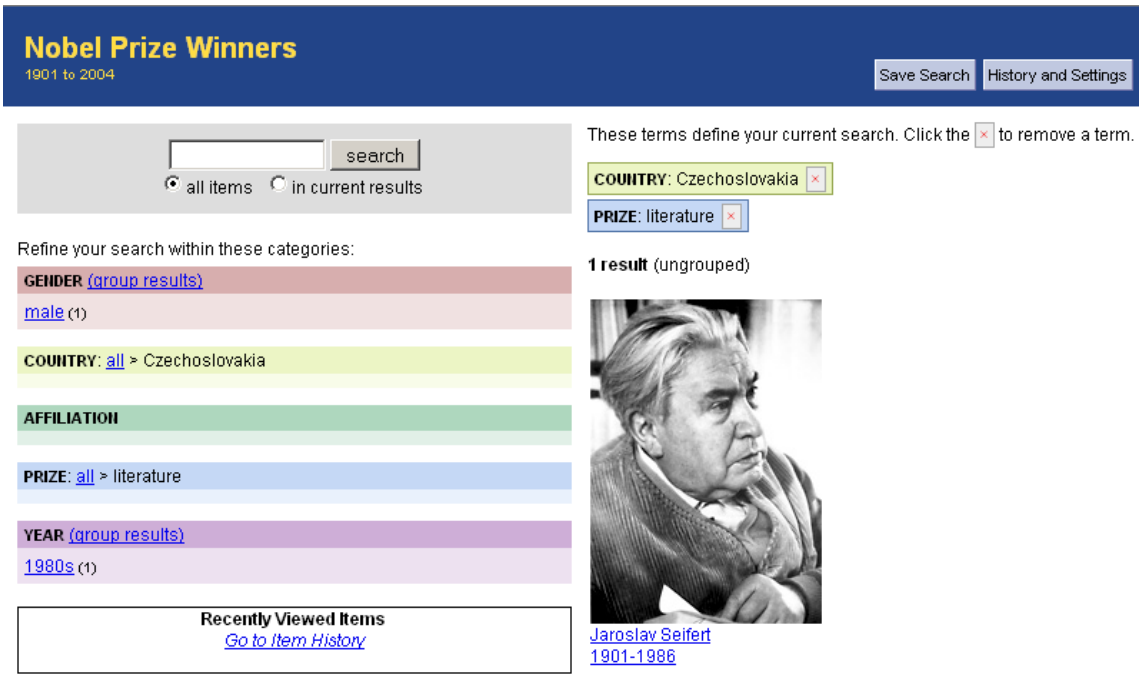
Uživatelské rozhraní vyhledávání je řešeno netradičním způsobem. Stránka pro vyhledávání je rozdělena na část pro zobrazení faset a jejich hodnot, část zobrazující již vybrané omezení a část zobrazující výsledky vyhledávání.

Fasety jsou zobrazeny v barevně odlišených oddílech obsahujících výčet všech hodnot dané fasety spolu s informací o počtu výskytů jednotlivých hodnot. V případě, že hod-

[3] `http://citeseerx.ist.psu.edu`
[4] `http://flamenco.berkeley.edu`

not fasety je více, než je možné zobrazit v prostrou vymezeném této fasetě, je zobrazeno jen omezené množství hodnot a na konci tohoto výčtu je odkaz na samostatnou stránku se seznamem všech hodnot. Hodnoty jsou reprezentovány HTML odkazy, je tedy možné současně vybrat pouze jednu. Fasety mohou být hierarchické. Hierarchické fasety jsou zobrazovány ve stromové struktuře, po zvolení hodnoty fasety na vyšší úrovni hierarchie má uživatel možnost zvolit hodnotu fasety na nižší úrovni).

Vybraná omezení jsou zobrazovány v horní části stránky ve formě obdélníků, uvnitř je vypsán název fasety a vybraná hodnota. Jednotlivá omezení lze zrušit kliknutím na ikonu křížku zobrazovanou u každého obdélníku.



Obrázek 2.1: Flamenco: snímek obrazovky vyhledávacího rozhraní

2.2.4 Mspace

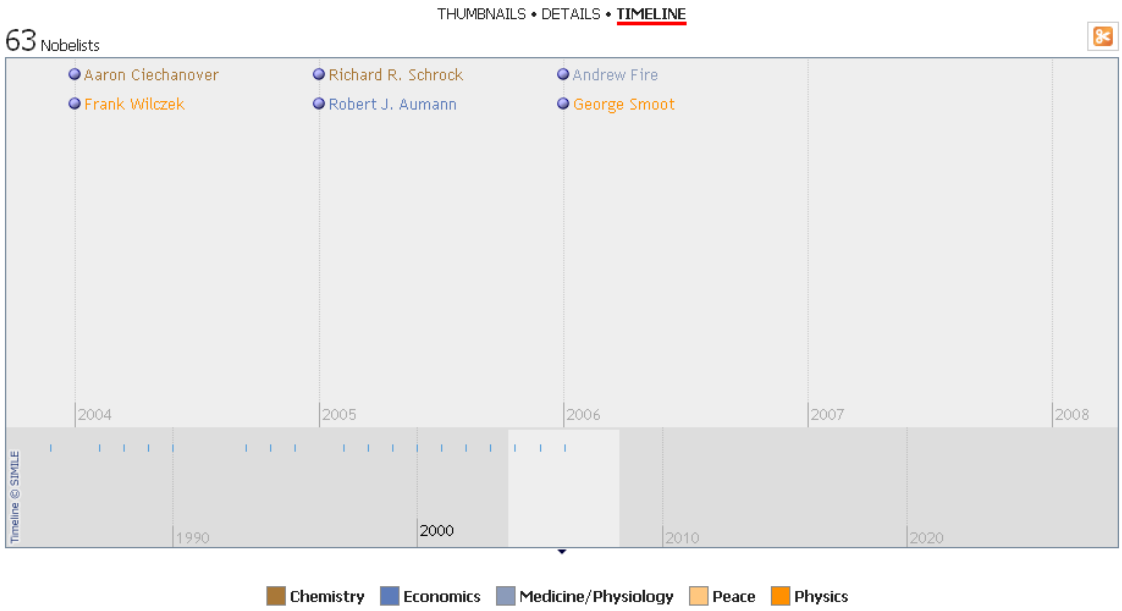
V projektu Mspace[5] jsou, stejně jako u projektu Flamenco, k entitám definovány metadata. Vybrané atributy metadat pak tvoří fasety. Fasetové vyhledávání lze kombinovat s fulltextovým. Fasety jsou zobrazovány v horní části obrazovky, dolní část obrazovky slouží zobrazení výsledků vyhledávání. Přetažením názvu fasety myší do k tomu vyhrazené části obrazovky dojde k aplikaci fasety na výsledky vyhledávání. Faseta je pak v této části obrazovky zobrazena jako seznam hodnot této fasety. Lze přidávat libovolné množství faset. Jednotlivé fasety jsou zobrazovány vodorovně vedle sebe a jejich pořadí zleva doprava určuje pořadí aplikace na výsledky. Změnu pořadí lze provést metodou drag and drop. Kliknutím myši v seznamu lze vybrat hodnotu fasety, při současném stisku s klávesou Ctrl lze vybrat více hodnot dané fasety najednou. Na rozdíl od projektu Flamenco výběr hodnot faset hojně využívá AJAX, po změně výběru hodnot je ihned provedena aktualizace možných hodnot

[5] `http://research.mspace.fm`

faset napravo od seznamu hodnot fasety, kde proběhl výběr. Zároveň jsou aktualizovány i výsledky.

2.2.5 Simile Exhibit

Simile Exhibit[6] je javascriptový framework umožňující třídění, filtrování a vizualizaci dat uvnitř HTML stránek. Veškeré operace jsou prováděny v klientském prohlížeči, z webového serveru jsou stahovány jen zdrojové soubory frameworku, případně předpřipravené stylové předpisy CSS. Při použití autor musí k HTML stránce připojit jeden nebo více souborů v JSON formátu s kolekcí zpracovávaných entit a definovat uživatelské rozhraní. Prvky uživatelského rozhraní jsou reprezentovány HTML značkami div a span, ke kterým jsou definovány rozšiřující atributy s vyhrazeným prefixem. Jako prvky uživatelského rozhraní lze použít fasety, časové osy (viz obrázek níže), mapy Google Maps s vyznačenými údaji, prvky pro zobrazování detailních informací, záložky pro přepínání pohledů atd. Při interakci uživatele s těmito prvky je dynamicky aktualizována HTML stránka, data pro prováděné operace jsou načítány z databáze implementované v jazyce Javascript. Databáze je naplněna entitami z připojených JSON souborů. Protože jsou veškeré údaje zpracovávány na straně klienta, je maximální velikost databáze doporučována řádově ve stovkách entit. Exhibit je uvažován jako aplikace vyhovující požadavkům sémantickému webu. Entity v JSON souborech představují ontologii, která je jednoznačně oddělena od části aplikace určené pro prezentaci [14].



Obrázek 2.2: Simile Exhibit: časová osa jako prvek uživatelského rozhraní

[6] http://www.simile-widgets.org/exhibit/

2.3 Shrnutí

V této kapitole byly uvedeny základní definice a principy týkající se klasifikace. Byl popsán rozdíl mezi taxonomií, která zavádí hierarchii shora dolů, folksonomií zavádějící hierarchii zdola nahoru a fasetovou klasifikací, která není hierarchická. Také byly zavedeny důležité pojmy metadata a ontologie. Stručně bylo představeno fulltextové vyhledávání, jehož detailnímu popisu je věnována následující kapitola. Implementace uvedených principů byla představena v přehledu existujících řešení. Informace získané provedeným průzkumem mohou posloužit jako výchozí bod pro návrh struktury metadat a uživatelského rozhraní vyvíjeného vyhledávače.

Kapitola 3

Fulltextové vyhledávání

Tato kapitola se zabývá fulltextovým vyhledáváním. První jsou uvedeny pojmy spojené s vyhledáváním informací a je objasněna souvislost s fulltextovým vyhledáváním. Dále v kapitole je představen invertovaný index, což je základní datová struktura využívaná ve fulltextovém vyhledávání. Druhá část kapitoly se zabývá implementací. Je zde představena knihovna Apache Lucene a server Apache Solr vystavěný nad touto knihovnou. Velký prostor je věnován způsobu ukládání informací v těchto nástrojích a jak jej lze využít pro uložení dat, k nim připojených metadat. V poslední části je popsán postup pro použití fasetového vyhledávání v nástroji Solr.

3.1 Vyhledávání informací

Tento oddíl byl vypracován na základě knihy [17], odkud byly převzaty také definice níže uvedených pojmů.

Definice 13 *Information retrieval (IR) se zabývá hledáním materiálů (obvykle dokumentů) nestrukturované povahy (obvykle text), které uspokojí potřebu zjištění určité informace, ve velkých kolekcích těchto materiálů (obvykle uložené v počítačích).*

Pojem původně zahrnoval aktivity odborníků, např. knihovníků nebo koncipientů. S rozvojem počítačů a hlavně internetu přichází s IR do styku stále více lidí a vyhledávání je stále více automatizované. IR se stává dominantním způsobem přístupu k informacím oproti tradičnímu databázovému stylu vyhledávání.

Definice 14 *Nestrukturovaná data jsou data, jež nemají čistou, zřetelnou, strojově snadno zpracovatelnou sémantickou strukturu. Příkladem strukturovaných dat je např. relační databáze, nestrukturovaných dat např. prostý text.*

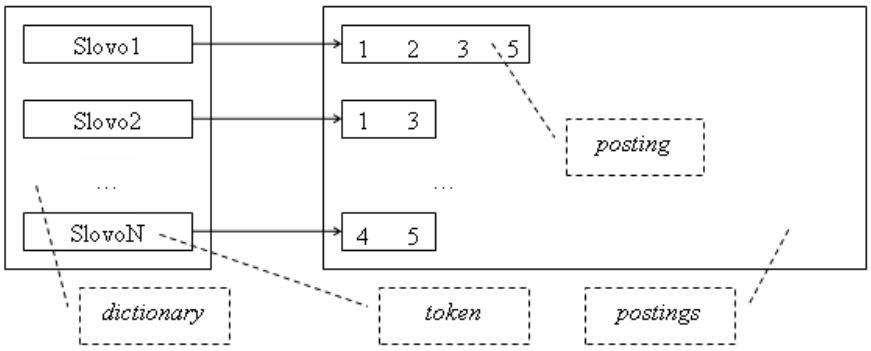
IR zahrnuje také procházení, shlukování, třídění a filtrování kolekcí dokumentů nebo nalezené množiny dokumentů. Tato problematika je popsána v předchozí kapitole.

Převažující úlohou IR systémů je *ad hoc* vyhledávání (*fulltextové vyhledávání*). Uživatel se formou dotazu snaží specifikovat téma, o kterém se snaží získat informace. Systém odpoví množinou relevantních dokumentů, tzn. dokumentů obsahujícími informace o daném tématu.

3.1.1 Vyhledávání informací

Nejjednodušší způsob vyhledávání je postupné lineární procházení celé kolekce dokumentů s hledáním řetězců shodných s řetězcem dotazu, nejčastěji na základě regulárního výrazu (anglicky *grepping* z Unioxého příkazu grep). Tento přístup je však použitelný pouze u kolekcí dokumentů velmi malého rozsahu.

Převažující strukturou použitou v ad hoc vyhledávání je *invertovaný index* (též invertovaný soubor, v dalším textu bude používán pouze pojem invertovaný index). Tato datová struktura mapuje slova (přesněji řečeno tokeny, viz dále v kapitole) na dokumenty, v nichž se vyskytují. Idea invertovaného indexu je zobrazena na obrázku 3.1.



Obrázek 3.1: Schéma invertovaného indexu

Všechna slova tvoří slovník, anglicky *dictionary*. Ke každému slovu je pak přiřazen seznam identifikátorů dokumentů, ve kterých se dané slovo vyskytuje. Pro tento seznam se používá anglický výraz *posting*, kolekce seznamů všech slov je pak nazývána *postings*. Např. invertovaný index uvedený na obrázku 1 nese informaci, že slovo2 se nachází v dokumentech identifikovaných čísly 1 a 3. Kromě identifikátorů dokumentů je v posting často také uložena informace o pozici slova v dokumentu. Invertovaný index je vytvářen následujícím způsobem

1. Vytvoření kolekce dokumentů
2. Tokenizace jednotlivých dokumentů. Při tokenizaci jsou dokumenty rozděleny na tokeny, je odstraněna interpunkce a další znaky. V principu můžeme uvažovat, že má výraz token stejný význam jako pojem slovo.
3. Dalším krokem může být lingvistické předzpracování, výsledkem jsou normalizované tokeny.
4. Vytvoření invertovaného indexu.

Při vyhledávání jsou pak ze zpětného indexu čteny postings jednotlivých slov dotazu. Výsledná množina dokumentů je získána jako průnik postings, to v případě hledání dokumentů obsahující všechna klíčová slova, nebo jako sjednocení, pokud hledané dokumenty musí obsahat alespoň jedno z hledaných slov.

Název pole	Obsah
Obsah	vlastní text článku
Autor	jméno autora článku
Datum	datum vydání článku

Tabulka 3.1: Příklad dokumentu uloženého v intertovaném indexu

3.2 Apache Lucene

Apache Lucene[1] je knihovna pro indexování a fulltextové vyhledávání implementovaná v programovacím jazyce Java. Aplikační rozhraní knihovny obsahuje třídy nutné k vytvoření indexu, indexování dokumentů a následné vyhledávání.

Z pohledu aplikačního rozhraní se index skládá z dokumentů [25]. Dokumenty se dále dělí na pole (fields). Každé pole je pojmenováno a je specifikováno, jak je uložena jeho textová hodnota. Hodnota pole může být indexována (uložena v invertovaném indexu) a může tedy nad ní probíhat vyhledávání nebo může být hodnota v indexu uložena v původní textové podobě. Hodnota pole může být zároveň indexována i uložena v původní podobě. Pokud je hodnota pole indexována, lze určit, zda má nebo nemá být hodnota tokenizována a také lze volbou vhodného analyzátoru zvolit způsob tokenizace. Každý dokument v indexu může obsahovat rozdílný počet polí a tyto pole mohou být libovolně pojmenovány a pole může rovněž nést více hodnot najednou.

Dělení dokumentů na pole lze použít pro uložení metadat. V principu je pak vlastní text dokumentu uložen v jednom poli, ostatní slouží pro uložení metadat dokumentu. Tabulka 3.1 je příkladem, jak by dokument reprezentující vědecký článek mohl být navržen.

V prvním poli je uložen vlastní text článku a další dvě slouží jako popisná metadata. Více se čtenář dozví v kapitole 4, která se podrobně zabývá problematikou návrhu metadat.

3.3 Solr

Solr[2] představuje vyhledávací platformu založenou na knihovně Apache Lucene. Knihovna je využívána jako jádro pro vyhledávání a indexování. Solr běží jako samostatný server, pro svůj běh vyžaduje servletový kontejner. Rozhraní serveru je dostupné jako webová služba. Komunikace probíhá nad protokolem HTTP, v XML nebo JSON formátu. Existují klienti pro použití v různých programovacích jazycích. Pro Javu existuje knihovna Jsolr. Při použití knihovny Jsolr je třeba pouze definovat způsob připojení k serveru a ten je poté ve zdrojovém kódu přístupný jako objekt návrhového vzoru Singleton. Komunikace se serverem probíhá voláním metod tohoto objektu.

3.3.1 Schéma

Schéma definuje, jaká pole může obsahovat dokument a jaké jsou vlastnosti těchto polí. V předchozí podkapitole byla pole uvažována pro uložení metadat. Schéma tudíž zároveň popisuje metadata. Představuje metadata metadat, nazývaná také metadata[2]. Schéma je definováno v souboru schema.xml v konfiguračním adresáři serveru (conf/schema.xml).

[1] `http://lucene.apache.org/java/docs/`
[2] `http://lucene.apache.org/solr/`

Pole dokumentu jsou silně typovaná, což znamená, že každé pole musí mít určen datový typ. Nejdříve je nutno deklarovat vlastní datové typy a poté je použít při definici polí dokumentu.

Pro deklaraci datových typů slouží sekce vymezená značkou `<types>`. Datový typ je definován ve značce `<fieldtype>`. Povinnými atributy jsou jméno typu a jméno třídy, která daný typ implementuje. Tato třída musí být potomkem třídy FieldType z balíku solr.schema. Tento přístup, při kterém je třeba nejdříve definovat vlastní datové typy, umožňuje implementovat a použít vlastní datový typ bez zásahů do zdrojového kódu Solru. Solr implementuje třídy pro základní datové typy: celočíselné, s plovoucí řádovou čárkou, datum, řetězec. Například značka

```
<fieldType name="integer" class="solr.IntField"/>
```

definuje celočíselný datový typ `integer`, jenž je implementován třídou solr.IntField. V definici typu lze také specifikovat, jaký analyzátor a filtr má být na pole použit při tokenizaci (více informací o analyzátorech a filtrech lze nalézt v dokumentaci Apache Lucene [25]).

Sekce `<fields>`, která následuje za sekcí `<types>`, slouží pro vlastní definici polí dokumentu. Pro definici pole slouží značka `<fieldType>`. Atributy této značky určují vlastnosti definovaného pole: jméno, zda bude hodnota pole indexována a zda bude uložena v indexu v původní podobě. Například

```
<field name="cena" type="integer" indexed="true" stored="true"/>
```

Definuje pole cena typu `integer`. Toto pole je indexováno a zároveň uloženo v indexu i v původní podobě.

Schéma je platné pro všechny dokumenty v indexu. Solr však umožňuje během indexování vytvořit i dynamická pole, která jsou uložena jen v daném dokumentu. Více podrobností lze nalézt v dokumentaci projektu Solr [26].

3.3.2 Přidávání, změna a odebírání dokumentů

Změny indexu jsou prováděny zasíláním HTTP požadavků na URL *adresaServeru/update*. Příkaz je připojen k požadavku metodou POST, pro reprezentaci je použit formát XML. Kořenová značka připojeného XML kódu určuje typ příkazu.

Pro přidávání a změnu dokumentů slouží značka `<add>`. To, zda se tímto příkazem vloží do indexu nový dokument, nebo se změní existující, souvisí se schématem indexu. V souboru schema.xml lze značkou `<uniqeKey>` určit název pole, které jednoznačně identifikuje daný dokument. Pokud je pak vkládán dokument s identifikátorem, který je již obsažen v indexu, je původní dokument přepsán novými hodnotami. Schéma XML značek je následující:

```
<add>
  <doc>
    <field name="pole1">hodnota</field>
      ...
    <field name="poleN">hodnta</field>
  </doc>
  <doc>
    ...
```

```
  </doc>
  ...
</add>
```

Pro mazání dokumentů z indexu slouží značka `<delete>`. Dokumenty, jež mají být smazány, lze specifikovat hodnotou pole jednoznačného identifikátoru nebo lze smazat všechny dokumenty vyhovující zadanému dotazu. Například:

```
<delete><id>hodnota</id></delete>
<delete><query>id:hodnota</query></delete>
```

3.3.3 Vyhledávání

Vyhledávání probíhá pomocí HTTP požadavků na URL *adresaServeru/select/*. Parametry dotazu jsou zaslány metodou GET. Povinný je pouze parametr *q*, který obsahuje text dotazu. Ostatní parametry specifikují formát odpovědi serveru. Parametry mohou být hierarchické, kdy jméno parametru se skládá s více částí oddělených tečkami. Část před tečkou pak reprezentuje určitý aspekt vyhledávání a části za tečkou jeho atributy toho aspektu. Například parametrem *hl* je nastavováno KWIC a atributy *hl.fl* a *hl.snippset* definují pole dokumentu pro KWIC resp. počet vrácených úseků pro každý dokument. Kromě KWIC lze definovat parametry pro fasetové vyhledávání, řazení, která pole dokumentu má odpověď obsahovat atd.

Odpověď serveru je zaslána ve formátu XML, případně JSON (lze určit parametrem dotazu) a obsahuje statistické údaje o výsledku hledání a vlastní kolekci nalezených dokumentů.

3.3.4 Fasety

Použití fasetového vyhledávání v Solru nevyžaduje žádnou explicitní konfiguraci. Jako faseta může být použito libovolné pole, které je indexováno. Název fasety tedy odpovídá názvu pole v indexu. Solr nabízí dvě základní fasetové operace:

- získání počtu výskytů jednotlivých hodnot požadovaného pole (tzn. získání všech hodnot dané fasety a poštu výskytů hodnot této fasety)
- získání dokumentů, u nichž faseta nabývá požadované hodnoty

Pro nastavení fasetového vyhledávání slouží parametr *facet*. Hodnota tohoto parametru musí být definována (hodnota může být libovolná, obvykle true nebo on). Další atributy pak slouží pro nastavení způsobu použití faset. Výše uvedené operace se vždy provádí až nad kolekcí dokumentů získanou dotazem, specifikovaným parametrem *q*. Pro použití faset na kolekcí všech dokumentů je třeba použít parametr *q* s hodnotou *:*. Například *adresaServeru/search/?q=*:*&facet=true&facet.field=název* vrátí všechny hodnoty fasety s názvem název počet výskytů těchto hodnot a požadavek *adresaServeru/search/?q=*:*&facet=true &fq=název:hodnota* vrátí dokumenty, u nichž faseta s názvem název nabývá hotnoty hodnota.

Solr umožňuje také vytvořit fasety nad poli nesoucími číselné nebo časové hodnoty. Hodnoty fasety pak představují určité rozsahy hodnot číselného pole. Toho je dosaženo příkazem *facet.query*. Tento příkaz umožňuje explicitně specifikovat dotaz, pro hodnotu

fasety. Místo unikátních hodnot pole lze použít konstrukci *facet.query=názevPole:[spodní mez TO hodní mez]*. Tato problematika je podrobně rozebrána v článku [3].

Kapitola 4

Návrh struktury metadat a uživatelského rozhraní

Vědecký webový portál, jak byl popsán v úvodu této práce, poskytuje uživateli informace spojené s výzkumem. Souhrn všech informací poskytovaných vědeckým portálem představuje bázi znalostí.

Definice 15 *Báze znalostí je databáze obsahující informace o lidských zkušenostech a odborných znalostech, dále zahrnuje indukční, dedukční a interferenční pravidla a pokud je použita v automaticky se učících systémech, obsahuje informace odvozené z dříve řešených problémů [1].*

Báze znalostí může být reprezentována různým způsobem, například pomocí RDF, což je specifikace konsorcia W3C pro reprezentaci metadat[1] nebo jako relační databáze doplněná ontologickými pravidly. Otologie je v těchto dvou případech chápána jako formální sémantický popis, který tvoří základ báze znalostí a je vyjádřen vhodnou formální strukturou na sémantické úrovni [12]. Tzn. ontologie definuje strukturu uložených dat, typy uložených entit a vztahy mezi nimi.

V nástrojích pro reprezentaci báze znalostí mají informace definovánu strukturu a vzájemné vztahy. Uživateli nebo správci jsou dostupné prostředky pro aktualizaci: vložení, změnu, mazání. Pro získávání informací jsou k dispozici nástroje pro přesné dotazování, kde je předpokládáno, že struktura dotazovaných dat a samotné dotazy jsou známy již při implementaci. Například u relačních databází, kde data jsou uložena zpravidla v normalizované podobě s důrazem na rozdělení dat na tabulky propojené cizími klíči, je tvorba dotazů za běhu většinou komplikovaná. Při nevhodně vytvořeném dotazu, dotazu využívajícím mnoho tabulek nebo vyhledávání nejasně specifikované informace může být provedení dotazu velice náročné na dostupné zdroje. Oproti tomu invertovaný index je datová struktura určená primárně pro ad-hoc dotazování, neumožňuje však tak efektivní správu uložených entit a jejich vzájemných vazeb jako nástroje určené pro reprezentaci báze znalostí.

Pokud je vyžadována možnost ad-hoc dotazování a zároveň je vyžadováno i přesné dotazování a efektivní správa (což je případ vědeckého webového portálu), nabízí se možnost oba způsoby uložení báze znalostí zkombinovat. Část báze znalostí je pak uložena redundantně. V použitém nástroji pro reprezentaci báze znalostí je uložena celá báze znalostí, v invertovaném indexu jsou uloženy údaje, na nichž může být prováděno hledání při ad-hoc dotazování. Invertovaný index lze tedy uvažovat jako pohled na bázi znalostí s možností

[1] `http://www.w3.org/RDF/`

rychlého vyhledávání [20]. V navrhovaném portálu je pak postup pro vykonání dotazu uživatele následující:

1. v uživatelském rozhraní portálu zadá uživatel dotaz pro ad-hoc vyhledávání
2. vyhledávací subsystém portálu postoupí dotaz serveru Solr
3. Solr provede dotaz, z invertovaného indexu získá množinu výsledků a vrátí vyhledávacímu subsystému portálu
4. vyhledávací subsystém vypíše výsledky. U každé nalezené položky mohou být vypsány dodatečné detailní informace. Ty jsou v případě potřeby získány vyhledávacím subsystémem z báze znalostí

Dále v této kapitole je diskutováno, jaké údaje o publikacích a jakým způsobem mají být uloženy v invertovaném indexu. Tyto údaje budou dále v textu nazývány metadata, v souladu s definicí tohoto pojmu uvedenou v kapitole 2.

4.1 Návrh struktury metadat

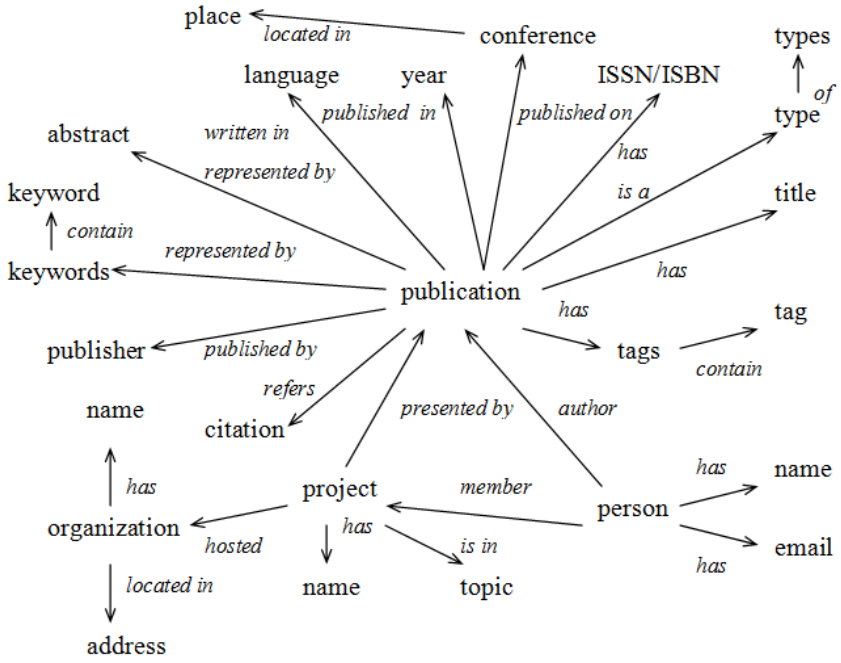
Při návrhu schématu metadat je potřeba zohlednit, jaké informace jsou nebo by měly být dostupné v bázi znalostí. Dále je třeba posoudit, které z těchto informací o dokumentech může uživatel zahrnout do vyhledávacího dotazu a jaké fasety může uživatel posléze vyžadovat při filtrování výsledků fulltextového vyhledávání. Důležitým hlediskem je také zřetel na výkon a rychlost vyhledávače.

Pro potřeby návrhu je uvažována báze znalostí uvedená na obrázku 4.1. Báze znalostí je znázorněna jako orientovaných graf. Uzly představují entity, hrany tvoří vazby, kde uzel na počátku orientované hrany je podmětem a uzel na konci je předmětem vazby. V uvedené bázi znalostí je uvažována jen ta část znalostí, jenž se přímo týká publikací nebo bude využita při návrhu uživatelského rozhraní.

Jak bylo uvedeno výše, invertovaný index si lze představit jako pohled na bázi znalostí s možností rychlého vyhledávání a další podrobnosti pak načítat z báze znalostí. Je tedy potřeba jednoznačně definovat vztah mezi publikací v bázi znalostí a dokumentem v invertovaném indexu, který představuje tuto publikaci. Tzn. je nutné mít k dispozici jednoznačný identifikátor. Jako tento identifikátor se nabízí URI entity publication s tím, že bude uložen jej jako jeden z atributů metadat.

4.1.1 Atributy významné pro fulltextové vyhledávání

Je potřeba stanovit, podle kterých vlastností publikace může uživatel vyhledávat, tzn. které vlastnosti lze zahrnout do fulltextového dotazu. Z výše uvedené báze znalostí byly vybrány text publikace (je dostupný jako dokument na URI entity publikace), titulek, abstrakt, rok vydání, autor, klíčová slova a název organizace. Tyto atributy musí být v invertovaném indexu indexovány a zároveň také uloženy v původní podobě a to ze dvou důvodů: za prvé zobrazování KWIC v Solru vyžaduje, aby pole dokumentu, které je použito pro KWIC, bylo uloženo v původní podobě a zadruhé z výkonnostních důvodů. Při zobrazování výsledků vyhledávání (jak bude ukázáno dále v této kapitole), je vždy zobrazeno několik nalezených dokumentů najednou a u každého jsou zobrazeny podrobnosti, např. rok vydání, jméno autora/autorů, popisky, organizace apod. Získávání těchto údajů jednotlivě z báze znalostí by bylo velice náročné, je proto vhodné uložit je přímo do invertovaného indexu a z báze



Obrázek 4.1: Báze znalostí

znalostí zjišťovat pouze rozšiřující informace, které jsou zobrazovány až v reakci na další činnost uživatele (např. zobrazení detailu vybrané publikace v seznamu s výsledky hledání).

4.1.2 Atributy významné pro fasetové vyhledávání

Pro možnost filtrování jsou navrženy tyto fasety: typ publikace, rok vydání, jméno autora/autorů, geografická lokalita organizace, jméno konference, vydavatel, oblast výzkumu a jazyk.

Typ publikace vychází z typů záznamů citačního manažeru BibTeX a může nabývat hodnot article, book, booklet, conference, incollection, inproceedings, manual, masterthesis, misc, phdthesis, proceedings, techreport a unpublished.

Faseta pro oblast výzkumu je hierarchická. Pro její taxonomii je použit ACM Computing Classification System (ACM CCS). ACM CSS je tříúrovňová klasifikace vědeckých oborů v oblasti počítačů. ACM CCS je využíván hlavně pro anotaci a vyhledávání vědeckých publikací [19]. Aby Solr správně prováděl operace nad touto fasetou, musí být při indexování atribut pro tuto fasetu uložen jako vícehodnotový se všemi hodnotami, které se nachází v ACM CCS na vyšších úrovních hierarchie. Například pro publikaci z oblasti Machine translation ponese tento atribut kromě hodnoty Machine translation také hodnoty Natural Language Processing, Artifical Intelligence a Computing Methodologies. Hodnota fasety v uživatelské rozhraní je pak zobrazena s ohledem na hierarchii a na to, kterou úroveň této hierarchie uživatel zvolil.

Faseta pro jazyk publikace ovlivňuje strukturu celého dokumentu uloženého v invertovaném indexu. Pro všechny atributy, které se mění pro každý jazyk (text publikace, nadpis, klíčová slova, abstrakt), musí být definována samostatná varianta těchto atributů pro každý podporovaný jazyk. Důvodem je možnost použít odlišné jazykové analyzátory při tokenizaci

Název	Popis
URI	jednoznačný identifikátor publikace
Type	typ publikace
Language	jazyk
Text	text
Year	rok vydání
Author	autor
Title	titulek
Keywords	klíčová slova
Abstract	abstrakt
Conference	jméno konference
Location	geografická lokalita
Publisher	vydavatel
Topic	vědecký obor podle ACM CCS
Project-name	název projektu
Tags	uživatelské popisky

Tabulka 4.1: Vlastnosti publikace uložené jako metadata

během indexování a vyhledávání. Hodnoty atributu fasety pro jazyk jsou odvozeny z normy ISO 639-1 [4]. Atributy jednotlivých jazykových variant jsou odlišeny postfixem skládajícím se z pomlčky a názvu jazyka dle zmíněné normy. Před vlastním vyhledáváním pak musí být dotaz upraven tak, aby vyhledávání probíhalo ve všech variantách atributů nebo v té variantě, která je určena hodnotou fasety pro jazyk.

Atributy sloužící pro fasetové vyhledávání musí být indexovány, avšak bez tokenizace (a tedy i bez zpracování analyzátory nebo filtry). Aby mohly tyto atributy sloužit také pro fulltextové vyhledávání, musí být uloženy v další kopii, ve které jsou během indexování také tokenizovány.

Posledním atributem, využívaným pro filtrování výsledků, jsou popisky (tags). V indexu jsou reprezentovány vícehodnotovým atributem, který je stejně jako atributy určené pro fasety, indexován bez tokenizace. Pro účely výpisu jsou všechny atributy pro fasetové vyhledávání uloženy v indexu také v původní podobě.

4.1.3 Výsledný návrh struktury metadat

Tabulka 4.1 obsahuje seznam vlastností, které jsou součástí navrhovaných metadat publikace. Tabulka 4.2 obsahuje atributy metadat tak, jak budou uloženy v invertovaném indexu. Tzn. pro každý atribut je vyhrazeno jméno, datový typ a způsob, jakým bude v indexu uložen.

4.2 Návrh uživatelského rozhraní

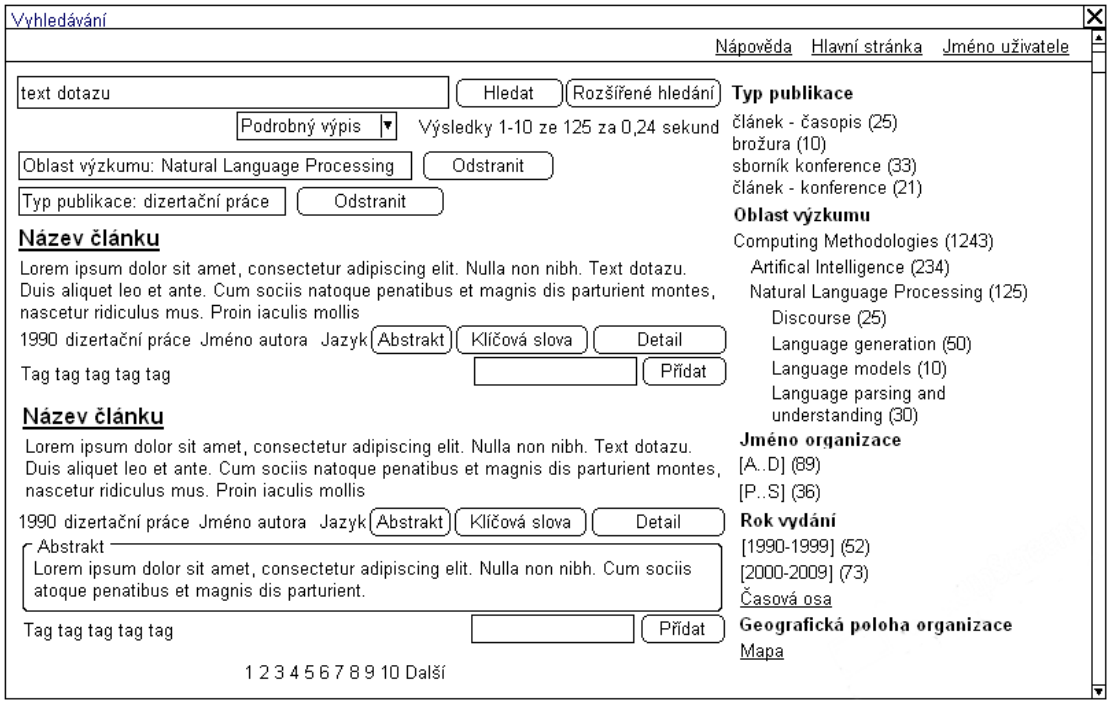
Při návrhu uživatelského rozhraní je třeba dbát na to, aby bylo jednoduché a snadno použitelné. Fasety by měly být co nejvíce interaktivní a vyhledávací dotaz by mělo jít vytvořit nejen textovým zápisem, ale také uživatelsky přívětivější formou pomocí tomu určeného formuláře. Výpis výsledků hledání by měl uživateli dynamicky nabízet detailní informace o nalezených publikací a také možnost jednoduše zobrazit další informace, které souvisí

Název typ	Datový	Indexován	Uložen	Více-hodnotový	Tokenizován	Jazyková analýza
uri	string	ano	ano	ne	ne	ne
type	string	ano	ano	ne	ne	ne
language	string	ano	ano	ano	ne	ne
text-en	string	ano	ano	ne	ano	ano
title-en	string	ano	ano	ne	ano	ano
abstract-en	string	ano	ano	ne	ano	ano
keywords-en	string	ano	ano	ano	ano	ano
year	integer	ano	ano	ne	ne	ne
author	string	ano	ano	ano	ano	ne
author-facet	string	ano	ne	ano	ne	ne
conference	string	ano	ano	ne	ano	ne
conference-faset	string	ano	ne	ne	ne	ne
location	string	ano	ano	ne	ano	ne
location-faset	string	ano	ne	ne	ne	ne
publisher	string	ano	ano	ne	ano	ne
publisher-faset	string	ano	ne	ne	ne	ne
topic	string	ano	ano	ano	ano	ne
topic-faset	string	ano	ne	ano	ne	ne
project-name	string	ano	ano	ne	ano	ne
tags	string	ano	ano	ano	ne	ne

Tabulka 4.2: Atributy metadat a jejich způsob uložení v invertovaném indexu

s publikacemi (např. detail autora, projektu atd.). Jinými slovy, vyhledávání by mělo být integrováno s ostatními částmi vědeckého webového portálu.

Na obrázku 4.2 je zobrazen mock-up stránky pro vyhledávání. V horní části se nachází lišta s odkazy na ostatní části portálu. Protože tématem této práce je pouze vyhledávací část, jsou zde předběžně navrženy odkazy pro nápovědu, návrat na hlavní stránku portálu a odkaz se jménem aktuálně přihlášeného uživatele, který vede na stránku s osobním nastavením. Pod touto lištou je umístěno pole pro zadání fulltextového dotazu. Vedle tohoto pole je tlačítko pro spuštění vyhledávání a tlačítko pro zadání podrobného dotazu (stisk tlačítka přesměruje uživatele na další stránku, kde může parametry dotazu zadat pomocí formuláře, viz obrázek 4.3). V pravé části se nalézá část vyhrazená pro fasety, nastavení řazení a seznam popisků. Levá část slouží k zobrazování výsledků hledávání.

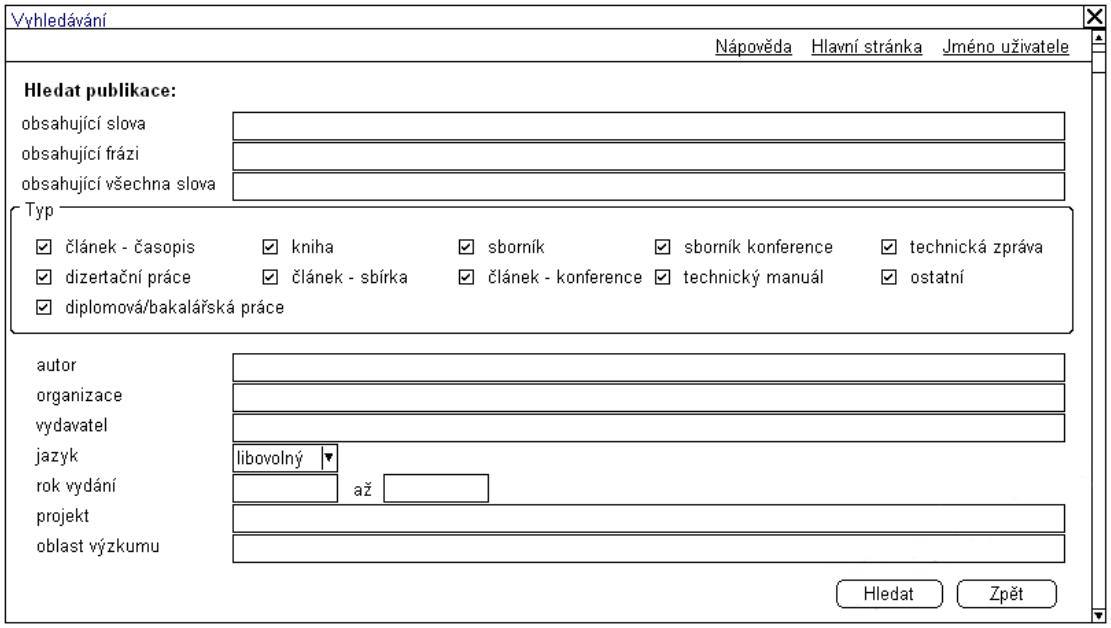


Obrázek 4.2: Mockup stránky pro vyhledávání

4.2.1 Fasety, nastavení řazení a seznam popisků

Jak bylo uvedeno, pravá část stránky má hned tři úlohy. Zobrazuje nastavení řazení, fasety a seznam popisků. Nabízí se možnost zobrazit každou část v samostatné záložce s možností přepínání v horní části stránky (viz výřez na obrázku 4.4). Avšak to by bylo v protikladu s požadavkem na jednoduchost uživatelského rozhraní, protože uživatel by neměl možnost vidět všechny parametry vyhledávání a nastavení zobrazení výsledků najednou. Všechny části jsou tedy zobrazeny společně v pořadí: fasety, seznam popisků, nastavení vyhledávání.

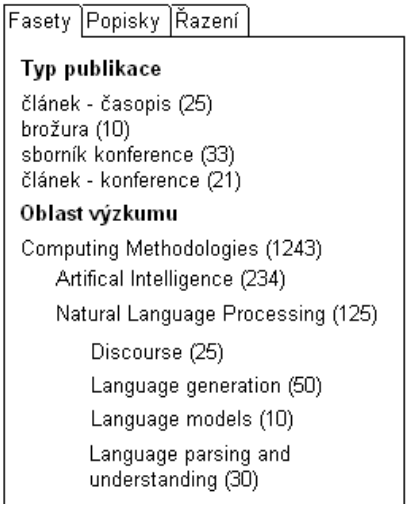
V případě, že faseta nabývá velkého množství hodnot, je možné vypsat pouze omezený počet těchto hodnot a na konci výčtu zobrazit odkaz pro zobrazení úplného seznamu. Seznam je po výběru odkazu uživatelem zobrazen buďto na zvláštní stránce nebo v dynamicky zobrazovaném rolovacím seznamu na původní stránce (pomocí AJAX). Druhou



Obrázek 4.3: Mockup formuláře pro podrobné zadání dotazu

možností je využití hierarchických faset. U hierarchické fasety jsou hodnoty shlukovány do skupin podle určitého hlediska. Poté, co uživatel vybere některou ze skupin, jsou zobrazeny hodnoty z této skupiny (což již mohou být vlastní jednotlivé hodnoty fasety nebo další skupiny). Zároveň je také provedena aktualizace výsledků vyhledávání a zobrazených hodnot ostatních faset. Hledisko pro seskupování může být různé. U číselné fasety Rok vydání (atribut Year) jsou to skupiny podle desetiletí, do něhož rok vydání náleží. U textových faset se nabízí počáteční písmeno hodnoty (skupiny jsou pak např. „A..D“, „E..H“, atd.). Solr nativně nepodporuje tento typ shlukování. Je potřeba zavést v invertovaném indexu pole, které nese hodnotu pro danou skupinu (např. „A..D“) a zjišťovat tuto hodnotu při indexování za pomoci analyzátoru podporujícího parsování řetězců pomocí regulárních výrazů [23]. Jak již bylo uvedeno, u fasety Oblast výzkumu je hierarchie definována podle ACM CCS. Hierarchický způsob zobrazování je přehlednější než zobrazení zvláštního seznamu všech hodnot fasety. Nabízí také uživateli více informací o hierarchii resp. ontologii v bázi znalostí. Na druhou stranu je nutné zvážit, zda zavedení hierarchie u některé z faset naopak nepovede k nižší srozumitelnosti a použitelnosti fasetového vyhledávání (např. hierarchická faseta Autor s členěním podle organizací, pro které autoři pracují, by vyžadovala po uživateli znalost příslušnosti autora k organizaci). Je nutné rozpoznávat, kdy by bylo zobrazeno příliš velké množství hodnot fasety a je tedy potřeba zavést hierarchii, a kdy zobrazit přímo výčet hodnot bez hierarchie. Kromě zobrazení hodnot fasety jako výčtu hodnot lze fasety Rok vydání a Geografická poloha zobrazit ve formě časové osy resp. mapy za využití projektu Simile Exhibit. Příslušný grafický prvek se dynamicky zobrazí po kliknutí na tomu určený odkaz, obdobně jako u zobrazování seznamu všech hodnot fasety.

Druhým problémem je, jak zobrazit, že uživatel vybral hodnotu fasety. Jako první východisko se nabízí zvýraznit tuto skutečnost přímo v seznamu faset v pravé části obrazovky. Způsobů, jak toho dosáhnout, je více. Lze zobrazit pouze hodnotu vybranou uživatelem a ostatní hodnoty skrýt. Nevýhodou je, že uživatel může vybrat pouze jednu hodnotu

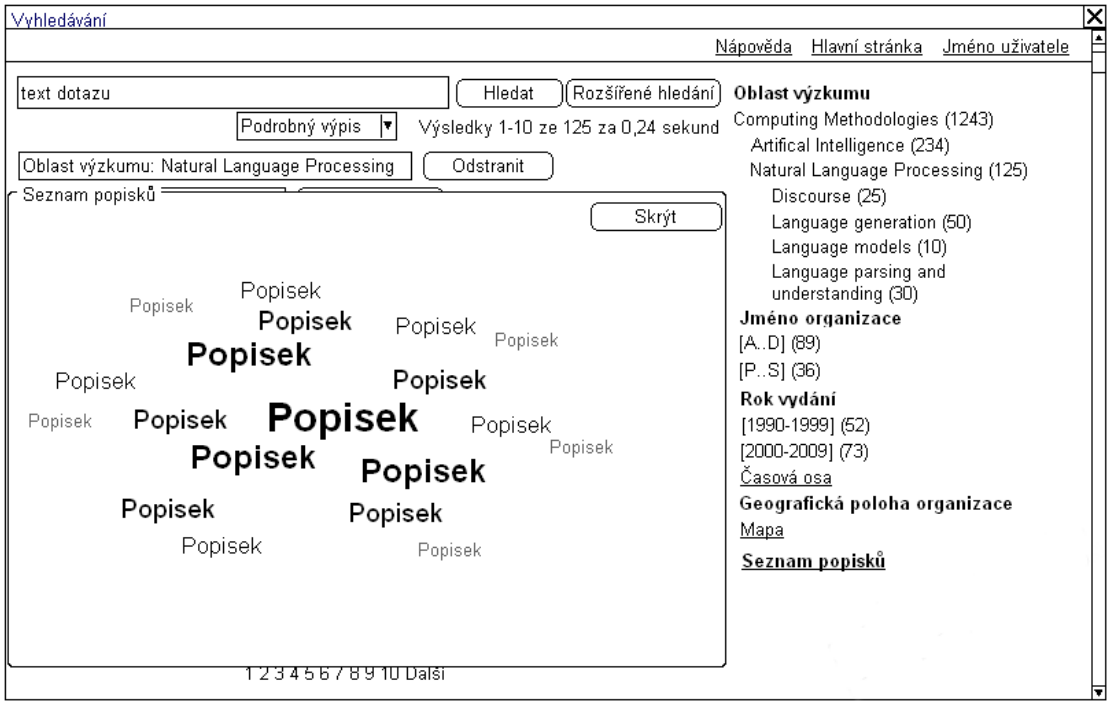


Obrázek 4.4: Výřez návrhu zobrazení faset, popisků a řazení

fasety, protože aktualizace výsledků, zobrazovaných hodnot ostatních faset a odstranění dalších hodnot dané fasety probíhá bezprostředně pro kliknutí na hodnotu v seznamu. Jiný způsob je zvýraznit vybranou hodnotu, například použít jinou barvu písma nebo větší tučnost písma. U obou těchto způsobů je problematické rušení fasetového omezení. Je potřeba buďto přidat speciální hodnotu každé fasety rušící všechna omezení dané fasety (např. Google Scholar změní při nastavení omezení nadpis fasety na odkaz „Všechny hodnoty“) nebo přidat ke každé vybrané hodnotě tlačítko pro zrušení omezení. Prostor ve vyhrazené části obrazovky je omezený, tlačítko tedy musí být malé a proto je jeho použití dost nepohodlné. Odlišný způsob, jak informovat uživatele o nastavených omezeních, je odebrat vybranou hodnotu fasety z nabídky v pravé části obrazovky a omezení vypisovat pod polem pro zadání textového dotazu (v levé části obrazovky). Omezení jsou vypisována pod sebou ve tvaru faseta:hodnota1, hodnota2, …, hodnotaN s dostatečně velkým tlačítkem nebo odkazem pro odstranění. Poslední způsob je vypisovat omezení přímo do textového dotazu. Z pohledu implementace je tento způsob nejjednodušší, avšak z pohledu použitelnosti uživatelského rozhraní se jeví lépe předchozí způsob. Ten je také použit v mock-upu znázorněném na obrázku 4.2.

Pod seznamem faset následuje tlačítko pro zobrazení *tag cloud*. Tag cloud je seznam popisků (tagů), kde významnější popisky jsou vypsány výraznějším písmem (větší velikost, tučnost a nebo barva písma). Významnost jednotlivých popisků je stanovena podle počtu dokumentů, jimž jsou přiřazeny. Tag cloud je zobrazován dynamicky až po kliknutí na příslušné tlačítko, vyhrazená oblast v pravé části stránky je příliš malá pro tento druh seznamu. Za seznamem faset a tag cloud jsou umístěny dvě přepínací tlačítka pro řazení podle relevance (implicitní volba) a podle roku vydání.

Fasety, seznam popisků a nastavení řazení jsou zobrazeny pouze až spolu s výsledky vyhledávání. Tzn. fasetová omezení jsou aplikována vždy na množinu výsledků fulltextového vyhledávání, nelze filtrovat všechny dokumenty uložené v invertovaném indexu. Důvodem je implementace fasetového vyhledávání v Solru, kdy fasety jsou použity na výsledky fulltextového dotazu. Ačkoliv Solr umožňuje vytvořit dotaz, který vrací všechny dokumenty v indexu (text dotazu *:*), není toto implicitně z výkonových důvodů povoleno (uživatel



Obrázek 4.5: Mockup stránky se zobrazeným tag cloud

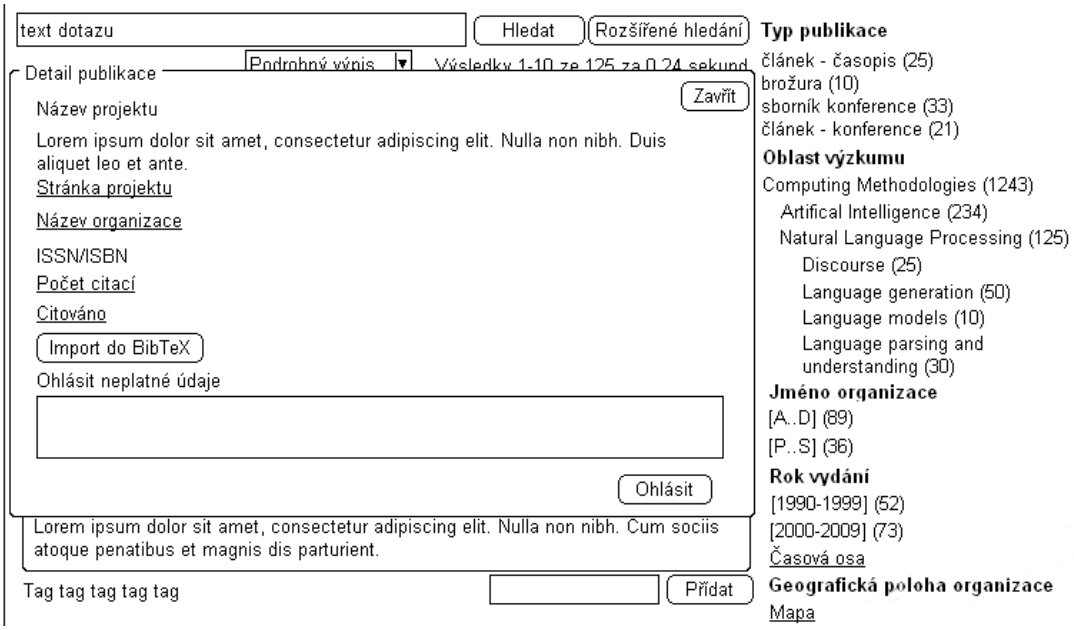
musí explicitně zadat dotaz *:*).

4.2.2 Zobrazování výsledků vyhledávání

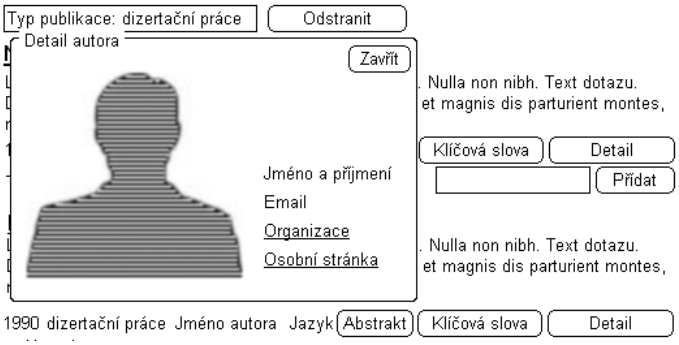
Poslední oblast obrazovky, kterou zbývá popsat, je část vyhrazená pro zobrazení výsledků vyhledávání. V pravém horním rohu je vypsán čas, za který byl výsledek vyhledávání získán, pozice a počet nalezených dokumentů a přepínač pro zkrácený nebo podrobný výpis. Zkrácený výpis vypisuje o každém dokumentu méně informací, zato však na jedné stránce zobrazí více nalezených dokumentů.

Při podrobném výpisu je u každého nalezeného dokumentu v záhlaví zobrazen nadpis, jenž zároveň slouží jako odkaz vedoucí na soubor s původním textem publikace (na její URI, jenž reprezentuje publikaci v invertovaném indexu). Dále KWIC publikace, rok vydání, typ publikace, autor/autoři (po kliknutí na jméno autora je dynamicky zobrazen krátký popis osoby autora), jazyk a tlačítka pro zobrazení abstraktu a klíčových slov. Tyto tlačítka po kliknutí zobrazí, bez obnovení načtené stránky, roletový panel. Uživatel může pro snadnější porovnávání výsledků tyto panely otevřít u více nalezených dokumentů najednou. Poslední tlačítko slouží pro zobrazení dalších detailů o publikaci. Nachází se zde informace o projektu, v rámci nějž byla publikace vydána, organizace, ISSN/ISBN, seznam citací publikace a citovaných publikací atd. Dále tlačítko pro automatické generování záznamu pro citační manažer BibTeX a možnost ohlásit nepřesné nebo neplatné údaje (jak bylo uvedeno v první kapitole, metadata mohou být extrahována automaticky a nemusí být vždy platná). Údaje o projektu, organizaci a citacích slouží odkazy na další části portálu s informacemi o daném tématu. Ve spodní části každého výsledku na stránce s výsledky vyhledávání je zobrazen seznam popisků přiřazených k dokumentu (nebo alespoň jejich část) a pole s tlačítky pro

přidání nového popisku a zobrazení všech popisků. Podrobný výpis je použit na obrázku 4.2.



Obrázek 4.6: Výřez stránky se zobrazeným detailem publikace



Obrázek 4.7: Výřez stránky se zobrazeným detailem osoby autora

U zkráceného výpisu je zobrazen u každé položky pouze nadpis, rok vydání, typ publikace, autor/autoři a jazyk. Ostatní údaje jsou dostupné v detailu publikace.

Kapitola 5

Platforma Java EE a framework Seam

Tato kapitola obsahuje stručný popis frameworku Seam, který je použit pro implementaci navrhovaného vyhledávače. Kromě samotného frameworku Seam se kapitola zabývá platformou Java EE 5.0, s níž je tento framework úzce spojen.

JBoss Seam je framework pro vývoj webových aplikací na platformě Java EE 5.0. Pro vývoj webových aplikací jsou v Java EE 5.0 nejvýznamnější technologie JSF a EJB [7]. V dalším textu jsou postupně tyto technologie představeny, včetně jejich zaměření a problémů, které jejich použití přináší. Poslední čast kapitoly pak popisuje, jak se framework Seam snaží problémy vznikající při kombinaci těchto technologií vyřešit sjednocením jejich komponentového a programovacího modelu.

5.1 Java EE 5.0

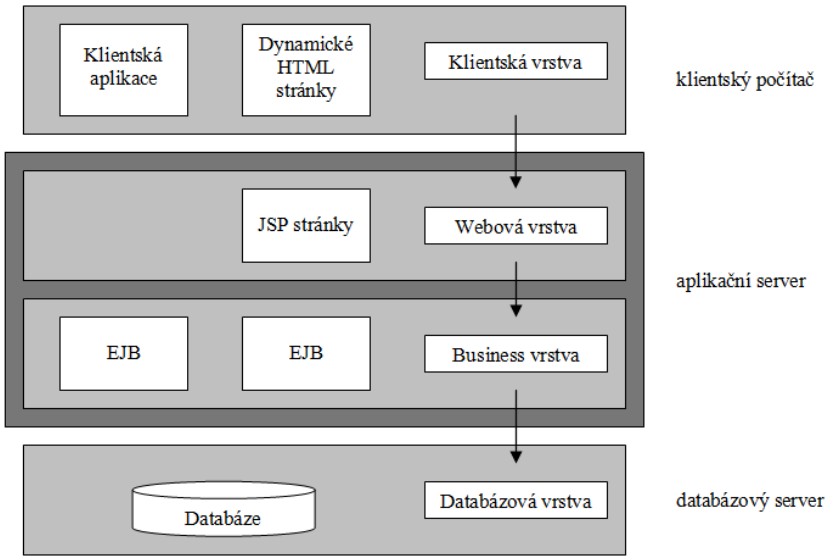
Java Platform Enterprise Edition (Java EE, dříve označovaná J2EE) je platforma určená pro vývoj a provoz podnikových aplikací a informačních systémů (enterprise aplikace). Jejím základem je platforma Java Standard Edition (Java SE), nad kterou jsou definovány součásti Javy EE poskytující prostředky pro tvorbu distribuovaných, vícevrstvých aplikací s garantovanou dostupností (fault-tolerant aplikace) [15].

Jednotlivé součásti jsou definovány pomocí dílčích specifikací. Specifikace jsou vyvíjeny v rámci Java Community Process, což je formalizovaný proces, který umožňuje různým zájmovým skupinám podílet se na vývoji platformy Java EE.

Aplikace pracující na platformě Java EE jsou tvořeny z modulárních komponent, vyvíjených na základě API a dalších fragmentů definovaných v jednotlivých specifikacích. Běhovým prostředím pro tyto komponenty je aplikační server. Specifikace zajišťují koncept přenositelnosti, kdy komponenty mohou teoreticky běžet v prostředí aplikačního serveru dodaného třetí stranou. Ovšem pouze za podmínky, že dodaný aplikační server splňuje vyžadované specifikace. Ve skutečnosti výrobci doplňují své aplikační servery dalšími vlastnostmi a aplikace, které používají tyto rozšířené vlastnosti, pak nejsou přenositelné [15]. Mezi nejznámější aplikační servery paří Sun Java System Application Server, GlassFish (Sun), Geronimo (Apache), WebSphere (IBM) a JBoss (Red Hat). Poslední z jmenovaných aplikačních serverů, JBoss společnosti Red Hat, je použit pro implementaci navrhovaného vyhledávače.

Jak bylo uvedeno, Java EE umožňuje tvorbu vícevrstvých aplikací. Samotný aplikační

model Java EE je definován jako vícevrstvý. Vývojář aplikace nebo služby vytváří komponenty na příslušné vrstvě, přičemž tyto komponenty implementují prezentační a business pravidla (tedy vlastní logiku vyvíjené aplikace). Ostatní služby nutné pro běh aplikace jsou poskytovány platformou Java EE skrze příslušný kontejner. Jinými slovy, kontejner poskytuje běhové prostředí pro komponenty na dané vrstvě [15]. Na obrázku 5.1 jsou zobrazeny základní vrstvy aplikace v prostředí Java EE.



Obrázek 5.1: Vrstvy aplikačního modelu

Aplikace sestává ze čtyř vrstev:

- *Klientská vrstva.* Na této vrstvě jsou uživateli zobrazovány dynamicky generované HTML stránky nebo je použita aplikace běžící v klientském kontejneru. Vrstva se nachází na klientských počítačích.
- *Webová vrstva.* Obstarává interakci mezi business vrstvou a klientskou vrstvou (jedná se o tzv. front-end). Běhové prostředí pro tuto vrstvu je webový kontejner, použitá technologie je Java Server Faces využívající JSP nebo Facelets.
- *Business vrstva.* Implementuje chování aplikace (business logiku, back-end). Chování je zapouzdřeno do Enterprise Java Beans komponent (EJB). Běhovým prostředím je EJB kontejner. Vrstva může být potencionálně distribuovaná. Webový kontejner a EJB kontejner jsou poskytovány aplikačním serverem běžícím na samostatném stroji [15].
- *Databázová vrstva.* Je reprezentována systémem pro řízení báze dat (SŘBD), zpravidla také běží na samostatném stroji [15].

5.2 Enterprise Java Beans

Enterprise Java Beans (EJB) poskytují standardizovaný způsob implementace business logiky. Snaží se eliminovat problémy, na které vývojáři enterprise aplikací opakovaně naráží.

EJB poskytují služby jako perzistence databázových entit, transakce, integritu dat nebo zabezpečení. Vývojář se může soustředit na implementaci vlastní business logiky, ostatní potřebnou funkcionalitu poskytuje EJB kontejner. EJB jsou zvláště vhodné u aplikací, u kterých je vyžadována vysoká škálovatelnost, aplikace je distribuována na více strojů nebo musí být schopná běžet na různých typech klientů anebo transakce musí dodržovat integritu dat.

EJB se dělí na dva základní druhy: session bean, která reprezentuje konverzaci s klientem, a message driven mean, jež umožňuje přijímat zprávy asynchronně.

Session bean se dále dělí na stateful a stateless. Stateful session beans zastupují klienta na straně serveru, respektive představují stav konverzace s jedním klientem a provádí akce požadované klientem. Stav stateful session bean je udržován po dobu interakce s klientem. Naproti tomu stateless session beans neuchovávají stav po celou dobu konverzace, stav je udržován pouze po dobu vykonávání metody volané klientem.

V Java EE 5.0 je zavedena nová specifikace EJB 3.0, která z pohledu vývojáře zavádí některé významné vlastnosti [7]:

- Zadávání metadat popisujících EJB komponenty je zjednodušeno pomocí anotací přímo ve zdrojovém kódu. Tím je odstraněna nutnost vytvářen deployment deskriptory popsaných v XML.
- Pomocí Dependency Injections lze jednoduchým způsobem definovat, jak jsou komponenty provázány mezi sebou. Referenci na jinou komponentu lze vložit deklarativním způsobem za pomocí anotací ve zdrojovém kódu.
- K přístupu k perzistentním objektům je zavedeno Java Persistence Api (JPA), které spravuje data pomocí objektově-relačního mapování [15]. JPA nahrazuje dřívější Entity beans.

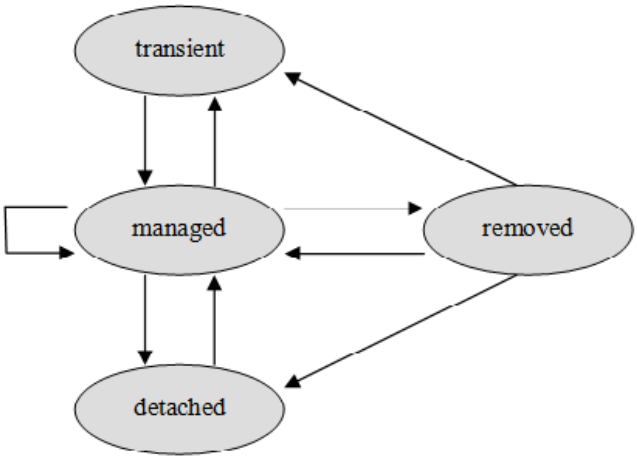
5.3 Java Persistence Api

Java Persistence Api (JPA) specifikuje způsob správy perzistentních dat pomocí objektově-relačního mapování. Základním účelem je odstínit implementaci aplikace od SQL při přístupu k databázi a nahradit jej manipulací s objekty tak, aby databázové operace byly prováděny transparentně a reflektovaly změny stavu objektů [5].

Objekty reprezentující data jsou POJO (Plain Old Java Objects), se kterými se pracuje jako s běžnými objekty. Tyto POJO jsou pomocí anotací doplněny o metadata, která popisují jejich mapování na tabulky relační databáze. V terminologii JPA se objekty s perzistentními daty nazývají entity, jejich definice pak entitní třídy. Obvykle jedna entita odpovídá jednomu řádku v databázové tabulce. Atributy entity, jež jsou anotacemi označeny jako perzistentní (respektive nejsou označeny jako transient – neperzistentní), jsou mapovány na sloupce databázové tabulky. Atributy entity, které jsou typu kolekce nebo jsou referencí na jinou entitu, představují vazby mezi databázovými tabulkami. Každá entita musí mít definován jednoznačný identifikátor ve formě jednoho nebo více atributů. Identifikátor je obvykle mapován na primární klíč databázové tabulky.

Ústředním prvkem JPA je Persistence Context, který představuje množinu entit uložených v perzistentním úložišti. Pro přístup k Persistence Context slouží Persistence Manager, který využívá metadata získaná z anotací k převodu entit z prostředí běžící Java aplikace do databáze, resp. z databáze do běžící aplikace. To znamená, Persistence Manager spravuje entity, je zodpovědný za jejich čtení z databáze a ukládání do databáze. Každá entita

prochází životním cyklem, který vyjadřuje stav entity během její správy Persistence Managerem [5].



Obrázek 5.2: Životní cyklus entity

Obrázek 5.2 ukazuje možné stavy entity a přechody mezi nimi. Entita může být ve stavu:

- *Transient* – entitě není přirazen Persistence Context
- *Managed (též persistent)* – entita má přiřazen Persistence Context. V rámci tohoto kontextu má entita přidělenu identitu a korespondující řádek v databázové tabulce, je synchronizována s databází
- *Removed* – entita byla odebrána z Persistence Context. Synchronizace s databází v tomto případě znamená smazání korespondujícího řádku z databázové tabulky.
- *Detached* – entitě byl přiřazen persistence context, avšak tento kontext byl uzavřen. Entita má přiřazenu identitu a korespondující řádek v databázové tabulce.

Ve vývojovém prostředí je Persistence Manager prezentován rozhraním *EntityManager*. Instanci Persistence Manageru lze v EJB zpřístupnit pomocí Dependency Injection (anotace *PersistenceContext*). Persistence Manager poskytuje veškeré prostředky potřebné pro správu entit. Pro dotazování je k dispozici jazyk JPQL. Jedná se o deklarativní jazyk podobný SQL. Na rozdíl od SQL, kde výsledkem dotazu je tabulka, JPQL vrací jako výsledek množinu entit vyhovujících dotazu.

Operace nad entitami s přiřazeným Persistence Contextem by měly být vždy prováděny v transakcích zaručujících zachování konzistence dat v databázi, protože z pohledu databáze je Persistence Manager obyčejný klient, jež přistupuje k SQL rozhraní. V prostředí EJB komponent je integrita zajištěna automaticky, metody EJB komponent, které manipulují s entitami s přiřazeným Persistence Contextem, jsou implicitně prováděny v transakcích.

5.4 Hibernate

Hibernate je open-source framework pro objektově-relační mapování, který implementuje JPA specifikaci. Poskytuje však také vlastní nativní rozhraní, které rozšiřuje funkcionalitu nabízenou JPA. Často využívanou vlastností je možnost manuálně provést synchronizaci entit spravovaných Persistence Contextem s databází (tzv. flushing). Obecně lze říci, že nativní rozhraní Hibernate dává vývojáři větší kontrolu nad operacemi nad Persistence Contextem. Na druhou stranu je si potřeba uvědomit, že nativní API není standardizováno a není jej možné jednoduše nahradit jinou implementací objektově relačního mapování.

5.5 Java Server Faces

Java Sever Faces (JSF) je prezentační framework, který umožňuje vytvářet uživatelská rozhraní. Definuje flexibilní model pro visuální komponenty (často nazývané widgets). Kromě toho definuje programovací model pro backing beans, což jsou komponenty obsahující aplikační logiku. JSF definuje způsob interakce mezi uživatelským rozhraním a aplikační logikou a způsob, jak jsou tyto dva celky spolu propojeny [7]. Navigace v uživatelském rozhraní je definována deklarativně pravidly popsanými v XML formátu.

JSF framework je založen na architektonickém vzoru MVC.

- Model je tvořen EJB, které provádí business pravidla a poskytují přístup k perzistentním datům.
- View je zodpovědný za vytváření stromu vizuálních komponent. Základní knihovny komponent jsou Core a HTML Render Kit. K vlastnímu zobrazení stromu ve formě HTML a obslužného JavaScriptu je využívána technologie JSP a nebo v současnosti preferované Facelets. Facelets je v podstatě šablonovací systém, který přináší řadu vylepšení oproti JSP. Jako příklad uveďme možnost vytvářet šablony stránky, tvorba složených komponent, vylepšené možnosti ladění aj.
- Controller je tvořen backing beany (nazývány také managed bean). Jedná se o POJO komponenty, které JSF spravuje ve vlastních kontextech (Stateless, Request, Session) a které řídí běh aplikace. Tyto komponenty jsou konfigurovány pomocí XML deskriptorů. S View je Controller svázán pomocí Expression Language (EL).

JSF je MVC pull-based framework, což znamená, že pohled získává data, která požaduje z obecně libovolného počtu kontrolerů (data „vytahuje“ z kontrolerů). Liší se tedy od push-based frameworku, kde jeden kontroler provádí zpracování dat a ty pak předá vybranému pohledu [18]. Obsluha stránky je rozdělena na jednotlivé backing bean komponenty, které zprostředkovávají informace zobrazované v různých částech stránky. Z pohledu provádění operací uživatelského rozhraní to znamená, že JSF je událostmi řízený framework. Při použití ovládacího prvku na HTML stránce vzniká událost, která je zaslána registrovaným příjemcům. Těmito příjemci jsou backing beans komponenty, resp. jejich metody. V JSF není možné z pohledu přistupovat přímo k modelu, přístup musí být vždy proveden přes baking beans komponenty. JSF tak dodržuje vícevrstvou architekturu Java EE, kdy View a Controller JSF frameworku odpovídají webové vrstvě a Model business vrstvě.

Výhodou JSF je efektivita při základním použití. Problém nastane, jestliže je vyžadováno jiné chování než implicitní. JSF je robustní framework, umožňující rychlý vývoj

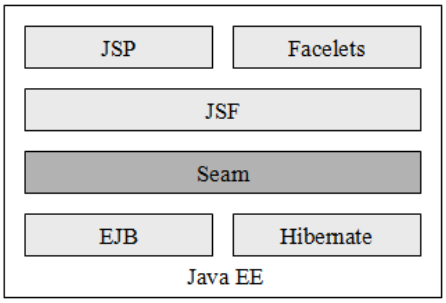
aplikací s velkou mírou obecnosti, kdy je možné nahrazovat použité technologie za jiné (například zmíněné použití Facelets místo JSP). To přináší složitost ukrytou pod povrchem JSF a je obtížné upravit framework tak, aby se choval jinak než implicitně. Další často kritizovanou vlastností je, že navigace je založena pouze na POST parametrech a neumožňuje vytvářet RESTful URL.

RESTful URL je URL ve formátu, který permanentně reprezentuje určitý stav aplikace. Tento stav je možné kdykoliv obnovit pouze s využitím tohoto URL a lze je na něj odkazovat HTML odkazy.

5.6 Seam

Jak bylo uvedeno na začátku kapitoly, dvě nejdůležitější součásti Java EE 5.0 pro vývoj webových aplikací jsou EJB a JSF. EJB je standardní programovací model pro tvorbu bezpečných, škálovatelných business komponent, které přistupují ke zdrojům transakčním způsobem. JSF je standardní prezentační framework založený na událostech. Ani jedna z těchto dvou technologií nepokrývá všechny požadavky na webové aplikace. Každá technologie má navíc rozdílný model a specifikace Java EE 5.0 neobsahuje standardizovaný způsob, jak je propojit dohromady.

Seam využívá toho, že EJB i JSF umožňují rozšíření a spolupráci s dalšími frameworky. Seam propojuje programovací a komponentní modely obou technologií do jednoho unifikovaného a eliminuje nutnost vytvářet vlastní „glue“ kód, který by toto propojení implementoval [5].



Obrázek 5.3: Pozice frameworku Seam v Java EE

Důsledkem této unifikace je, že v Seamu není rozdělení na webovou a business vrstvu. Je možné navrhnout libovolnou architekturu aplikace, backing beans nejsou povinné. Business komponenty jsou schopny využívat data uložená v kontextu webové vrstvy nebo přistupovat ke stromu visuálních komponent JSF. Je tím odbourána nutnost programování tradičního Controlleru v architektonickém vzoru MVC. JPA entity mohou být přímo zobrazeny ve stránce generované pomocí JSF a EJB komponenty mohou tvořit Controller. Veškeré možnosti jsou k dispozici použitím příslušných anotací bez nutnosti vytvářet jakýkoliv další kód [22].

Důraz na konfiguraci využitím anotací přímo ve zdrojovém kódu je jednou ze základních vlastností frameworku Seam. Stále však existuje také možnost definovat vlastnosti komponent ve formě XML souborů [7]. Pomocí anotací se libovolná třída může stát Seam komponentou. Je tedy zcela na uvážení programátora používajícího Seam, jaké komponenty a v jakých kontextech navrhne a zda se bude jednot o POJO nebo EJB. V Seamu

je totiž v POJO komponentách možné využívat služby deklarativním způsobem pomocí Dependency Injections, stejně jako v EJB. Dostupné jsou služby zajišťující transakce nebo perzistenci, a to prostřednictvím JPA nebo Hibernate. Při použití pouze POJO komponent aplikace nevyžaduje aplikační server a může běžet i v běžném servlet kontejneru [5].

Seam řeší za programátora problém bezestavovosti HTTP protokolu takzvaným State Managementem. Není potřeba používat statelss session beans s uchováváním kompletního stavu aplikace v HTTP session a databázi (případně přenášen v parametrech mezi jednotlivými HTTP požadavky). Seam minimalizuje ukládání dat do HTTP session a kvůli výkonnosti také do databáze. Preferované jsou stateful session beans. Seam rovněž řeší konkurenční přístup ke komponentám. Samotná specifikace Java EE nevyžaduje serializovaný přístup k stateful session beans a ponechává riziko konkurenčního přístupu a ošetření případného race condition na programátora komponenty.

Seam obsahuje centrální správu kontextů a zavádí nové kontexty, které lépe odpovídají logice webových aplikací – conversation a business. Konverzace umožňuje programovat v Seamu webové aplikace bez ohledu na bezstavový HTTP protokol. Konverzací (Long Running Conversation) je nazývána souslednost několika požadavků, které v konečném důsledku tvoří celek mající nějaký cíl [16]. Seam obhospodařuje také konkuretní konverzace. V jednom prohlížeči je na více záložkách možné v téže webové aplikaci vést více konverzací. Toto je možné díky přesunu stavovosti z bezstavové HTTP session do stavových EJB komponent. Business kontext je speciální perzistentní kontext, jehož životnost může přesahovat dobu běhu aplikace.

Mezi další funkce Seamu patří transparentní podpora AJAXu (standardně knihovny RichFaces, IceFaces, Ajax4Jsf). AJAX lze využívat bez nutnosti vytvářet vlastní JavaScript obsluhy a nebo naopak lze přistupovat ke komponentám z prostředí klienta prostřednictvím JavaScriptu. Dále Seam obsahuje podporu pro generování CRUD formulářů pomocí Seam Application Framework. V klasické webové aplikaci tvoří velkou část kódu CRUD formuláře (formuláře umožňující operace vytvoření – Create, čtení – Read, změny – Update a smazaní – Delete entit). Tato úloha je obvykle řešena návrhovým vzorem DAO (Data Access Object). Protože se v DAO opakují stejné kusy kódu, zavádí Seam předpřipravenou komponentu EntityHome. DAO třídu pro perzistentní entitu lze vytvořit pouhým děděním ze třídy EntityHome nebo definicí ve formě XML [16].

Díky unifikaci programovacího a komponentního modelu se framework Seam prosadil jako prototyp budoucí specifikace Java EE [5]. Na druhou stranu je třeba říci, že Seam porušuje vícevrstvou strukturu Java EE. Pravidlem bývá, že závislosti jdou jen jedním směrem. Nižší vrstvy by měly být nezávislé na vyšších vrstvách, což v případě Seamu není splněno. Je například možné injektovat JSF komponentu do EJB) [16].

Kapitola 6

Architektura a návrh aplikace

Tato kapitola se zabývá architekturou a návrhem vyvíjeného vyhledávače. Vychází z kapitoly 4, pro názornost obsahuje množství diagramů. Vlastní implementace je podrobně rozebrána v následující kapitole.

V kapitole 4 bylo uvedeno, že informace poskytované vědeckým portálem tvoří bázi znalostí. Nástroje pro reprezentaci báze znalostí obvykle obsahují prostředky pro definici struktury, správu poskytovaných dat, navigaci a přesné dotazování. Pro ad-hoc dotazování je vhodnější invertovaný index, primárně sloužící pro vyhledávání v nestrukturovaných datech. Vyvíjená aplikace má umožňovat vyhledávání nad bází znalostí vědeckého portálu, kombinací obou způsobů dotazování. V invertovaném indexu budou redundantně uloženy ty informace z báze znalostí, které jsou důležité pro fulltextové a fasetové vyhledávání.

Jako implementační platforma byla zvolena Java EE a framework Seam, báze znalostí bude reprezentována relační databází. Fulltextové a fasetové vyhledávání bude zajišťovat vyhledávací server Solr.

6.1 Architektura

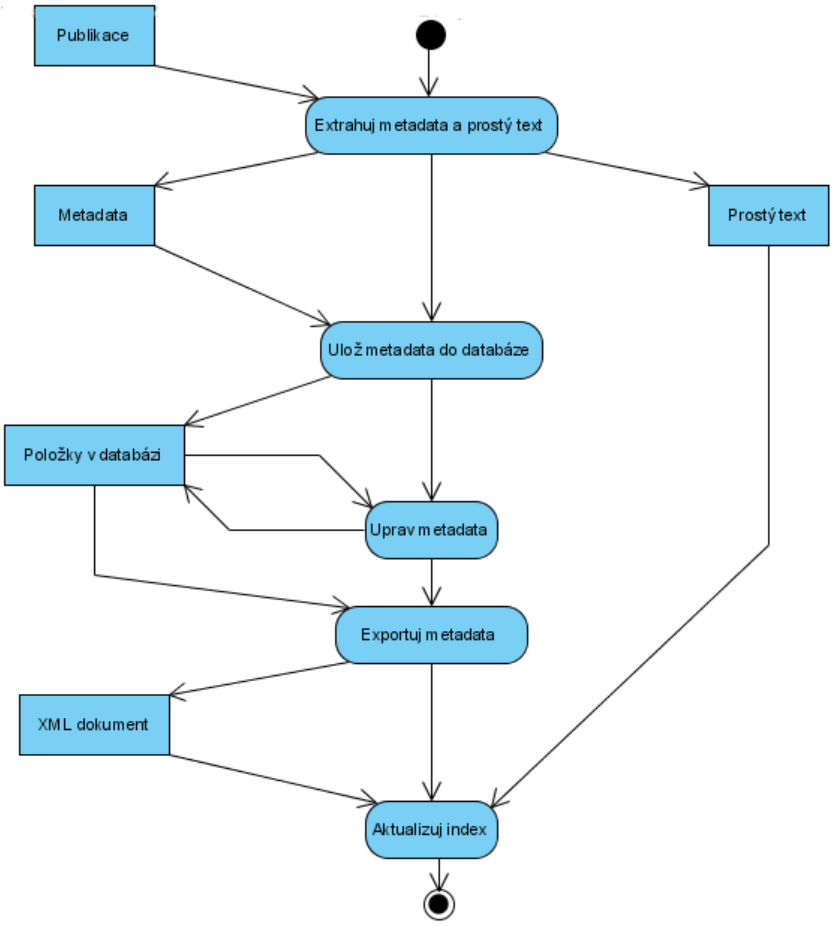
Na základě návrhu struktury metadat a uživatelského rozhraní provedeného v kapitole 4 můžeme rozdělit základní funkční celky vyhledávače následovně:

- Vyhledávací rozhraní. Tato část bude tvořena uživatelským rozhraním zpřístupňujícím vyhledávací služby poskytované serverem Solr. V souladu s návrhem provedeným v kapitole 4 bude toto uživatelské rozhraní nabízet možnost zobrazovat další informace souvisejících s nalezenými publikacemi, přičemž tyto informace budou získávány z databáze.

- Rozhraní pro správu báze znalostí. Tato část bude sloužit pro správu dat uložených v databázi. Uživatelské rozhraní bude tvořeno formuláři provádějícími CRUD operace nad perzistentními entitami.

- Automatizovaný import do databáze. Rozhraní bude umožňovat dávkovým způsobem naplnit databázi daty z externího zdroje.

- Export databáze do invertovaného indexu. Je potřeba také vytvořit nástroj, který na základě obsahu databáze vytvoří XML dokument pro aktualizaci invertovaného indexu spravovaného serverem Solr tak, aby data v invertovaném indexu odpovídaly datům uloženým v databázi.



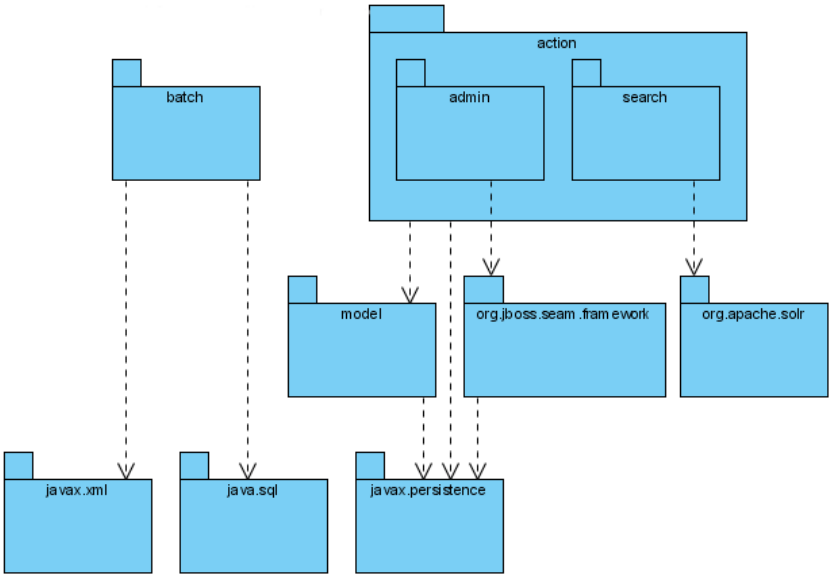
Obrázek 6.1: Diagram aktivity pro import dat

UML diagram aktivity na obrázku 6.1 ukazuje průchod dat výše uvedenými funkčními celky při importu publikací a jejich zpřístupnění pro vyhledávání. Jako první aktivita v procesu, jež je znázorněn na diagramu, je uvedena extrakce metadat a prostého textu. Tyto operace nejsou součástí vyvíjeného vyhledávače. V tomto případě však uvažujeme scénář, kdy je potřeba zpracovat již existující data, např. ve formě publikací v PDF formátu. Takovýto scénář bude pravděpodobně nejčastější způsob použití automatického importu.

V následující aktivitě jsou metadata uložena do databáze. Tím je rozšířena množina údajů poskytovaných bází znalostí. Prostý text je uchováván zvlášť, např. ve formě textových souborů v souborovém systému.

V aktivitě *Uprav metadata* má uživatel možnost procházet a upravovat importovaná metadata. Aplikace dává uživateli možnost zkontrolovat úspěšnost importu a případně provést úpravy nepřesných nebo neplatných údajů. To může být důležité u automatizované extrakce metadat. Jak bylo uvedeno v první kapitole, strojově získaná metadata nemusí být vždy platná.

Poslední dvě aktivity jsou *Exportuj metadata*, která vytvoří XML dokument popisující aktualizaci Solr indexu a aktivita *Aktualizuj index*, která představuje odeslání vygenerovaného XML dokumentu serveru Solr.



Obrázek 6.2: Struktura balíčků

Diagram na obrázku 6.2 popisuje strukturu balíčků implementujících všechny funkční celky vyhledávače v prostředí platformy Java. Balíček *action* obsahuje komponenty webové aplikace (ve smyslu Java EE nebo Seam komponent). Balíček *search* obsahuje komponenty poskytující vyhledávací služby a balíček *admin* obsahuje komponenty rozhraní pro správu obsahu databáze. Kromě těchto dvou balíčků obsahuje balíček *action* další pomocné komponenty, jako například komponentu zajišťující autentizaci uživatelů. Balíček *model* obsahuje definici perzistentních tříd ukládaných do databáze. Balíčky *org.jboss.seam.framework*, *javax.persistence*, *org.apache.solr* a *javax.sql* znázorňují závislosti na použitých nástrojích a technologiích. Balíček *org.jboss.seam.framework* zde neznamená celý framework Seam, jedná se pouze o jeden z balíčků Seam Application Framework. Tato knihovna s poněkud matoucím názvem poskytuje programátorovi pomocné prostředky pro rychlý vývoj aplikací (RAD – Rapid Application Development).

Dalším balíčkem je balíček *batch*, který implementuje import a export dat do resp. z databáze. Balíček je uvažován jako balíček obyčejných Java SE aplikací pracujících z příkazové řádky. Jelikož se jedná o SE aplikace, není možné využít objektově-relačního mapování JPA, což je služba poskytovaná pouze v protředí Java EJB kontejneru (i když Seam teoreticky dokáže Java EE služby poskytovat i ve webovém kontejneru, jak bylo uvedeno v předchozí kapitole) a je nutné použít kontektor na SQL rozhraní relační databáze. To přináší jistou nekonzistentnost, kdy pezistentní třídy jsou definovány balíčku *model* a zároveň v balíčku batch jsou používány SQL příkazy, jejichž definice nezávisí implicitně na balíčku *model*. Navíc v balíku batch se pracuje s konkrétním mapováním entit na databázové tabulky, jež se může v různých imlementacích JPA lišit. Na druhou stranu ovšem příkazová řádka umožňuje snadné provádění dávkových úloh. Pokud by byl vyžadován konzistentní model, mohl by být balík batch nahrazen Java EE řešením v podobě nového balíčku v balíku *action*, výměna dat by probíhala protokolem HTTP.

6.2 Návrh

6.2.1 Struktura databáze

Při použití objektově-relačního mapování není nutné navrhovat přímo databázové schéma. Strukturu stačí popsat skrze definici entitních tříd (entit) a jim připojených anotací, na základě kterých pak Hibernate automaticky vygeneruje databázové schéma.

Definice entitních tříd je umístěna v balíčku *model* a určuje v podstatě strukturu uložených metadat. Vychází z návrhu metadat, uživatelského rozhraní a též z báze znalostí vymezených v kapitole 4 (viz graf na obrázku 4.1) a dále byla vypracována také na základě schématu databáze pro systém vědeckého portálu vyvíjeného v rámci projektu ReResearch při Výzkumné skupině pro zpracování přirozeného jazyka [1].

Výsledný diagram tříd (perzistentních tříd) je zachycen na obrázku 6.3. Ústřední entitou modelu je entita *Publication*, s níž jsou provázány ostatní entity. Entita *Publication* je, kromě číselného identifikátoru, jednoznačně identifikovatelná atributem *Uri*. Obsah tohoto atributu je URI originální publikace. Všechny ostatní atributy jsou textové a mohou nést hodnotu null. Nemá smysl klást omezení na nutnost zadání některé hodnoty, protože vždy je potřeba uvažovat, že metadata mohou být extrahována automaticky a vyžadovaná hodnota nemusí být k dispozici nebo nemusí být extrahována správně. Zmiňme ještě atribut *plaintext*, který slouží pro uložení cesty k souboru obsahujícímu vlastní text publikace ve formě prostého textu. Tento atribut je využit při aktualizaci Solr indexu, kdy jsou indexovány kromě metadat také samotné texty publikací.

Další entity v diagramu jsou *Conference*, *Person*, *Project* a *Organization*. *Conference* obsahuje informace o konferencích, v rámci kterých mohou být publikace zveřejňovány. *Person* reprezentuje osoby, ve vztahu k publikacím v roli autorů. U této entity si lze také všimnout, že její definice přihlíží k možnosti automatické extrakce metadat. Neobsahuje tradiční dělení jména osoby na křestní jméno a příjmení. Místo toho zavádí pouze jeden atribut pro jméno, protože u pojmenování osob může být v mnoha případech složité určit, které slovo je křestním jménem a které příjmením. Entita *Project* a *Organization* zachycují členění osob a publikací. Na projektu se může podílet více osob a zároveň jedna osoba se může angažovat ve více projektech. To samé platí o organizacích, organizace může spolupracovat na více projektech a zároveň v jednom projektu může být zahrnuto více organizací. U organizace jsou také definovány atributy pro umístění organizace ve formě adresy. Druhou možností by bylo definovat lokaci zeměpisnou šířkou a délkou, pokud by to nástroj pro grafické znázornění fasety Lokalita vyžadoval (v návrhu uživatelského rozhraní je pro tento účel uvažován Simile Exhibit). Pro úplnost ještě uveďme, že osoba může mít přiřazenu domovskou organizaci a publikace může být vypracována v rámci některého z projektů.

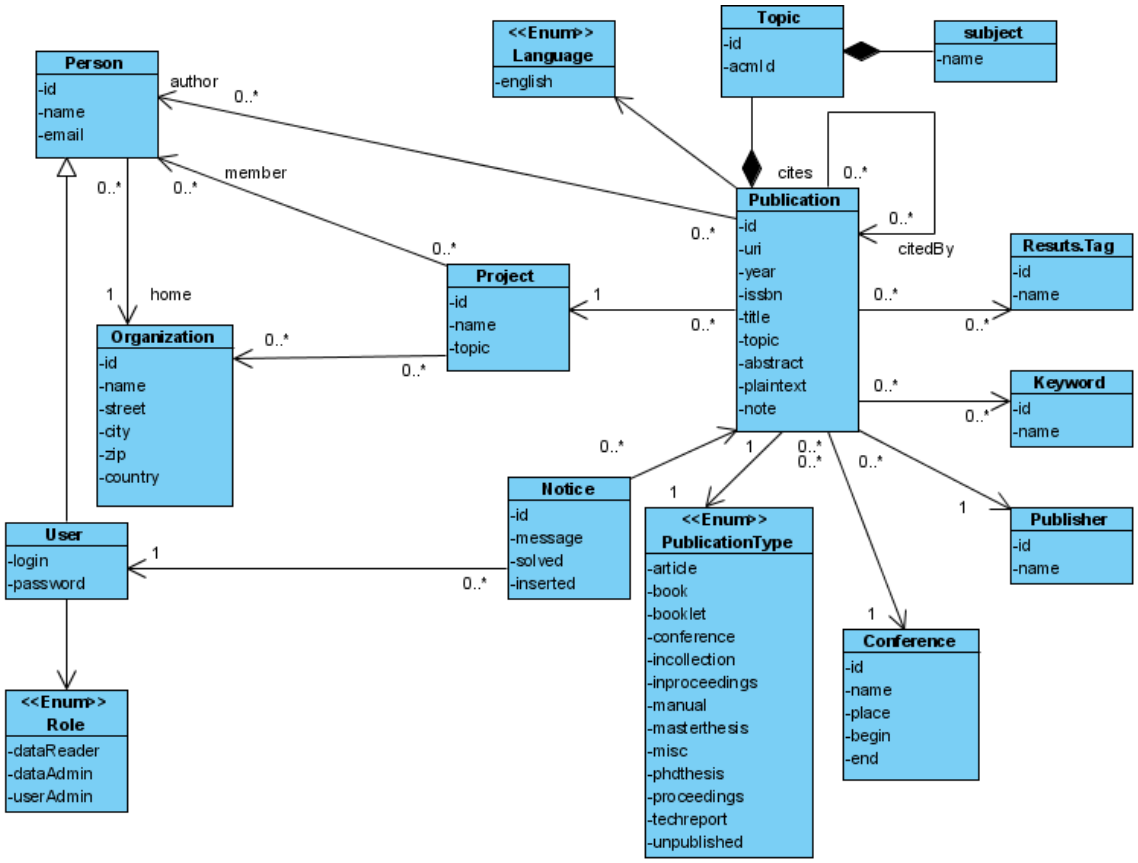
Entity *Tag*, *Keyword* a *Publisher* slouží pro uložení zbylých vlastností publikací. Jedná se o jednoduché entity, obsahující pouze atribut *id* pro identifikaci a *name* pro vlastní hodnotu entity. Citace jsou reprezentovány vztahem *cites* resp. *citedBy*, který vede z entity *Publication* na sebe samou.

Entita *Topic* slouží k uložení informace o oblasti výzkumu, do které publikace spadá. Pro oblasti je využívána čtyřstupňová klasifikace ACM CCS, která se skládá z cesty a seznamu předmětů v dané oblasti (subjects). Z definice této klasifica může mít publikace přiděleno více ACM kategorií v různých úrovních.

Zbylé dvě entity jsou spíše provozního rázu a slouží pro ukládání dat využitých pro funkce samotné aplikace. První je entita *User*, která představuje uživatele aplikace. Entitní

[1] `http://www.fit.vutbr.cz/research/groups/nlp/`

třída *User* dědí z entitní třídy *Person*. Uživatel je definován ve vztahu k některé osobě, uživatelé jsou tedy přímo propojeni s metadaty, která prohledávají. Kromě osoby má uživatel přiřazeno jednoznačné přihlašovací jméno, heslo a roli. Jsou uvažovány tři typy rolí – role umožňující používat vyhledávací rozhraní, role pro správu databáze metadat a role pro správu uživatelských účtů. Poslední entita je entita *Notice*. Tato entita slouží pro nahlašovaní chyb, na které uživatelé narazili během vyhledávání (tato funkce je popsána v kapitole 4). Entita obsahuje atributy pro datum a čas zadání chyby, vlastní text chyby a příznak nastavovaný z rozhraní pro správu databáze a který značí, zda již byla chyba opravena. Vztah s publikací je povinný, ohlášení chyby se vždy váže k některé publikaci. Povinnost vztahu s entitou *User*, který nese informaci o tom, kdo chybu ohlásil, není jednoznačná. Vztah by mohl být povinný pouze v případě, že vyhledávací rozhraní budou moci používat pouze přihlášení uživatelé. Pokud však bude vyhledávaní veřejně přístupné nebude možné tento vztah ustavit u všech hlášení chyb.



Obrázek 6.3: Diagram perzistentních tříd

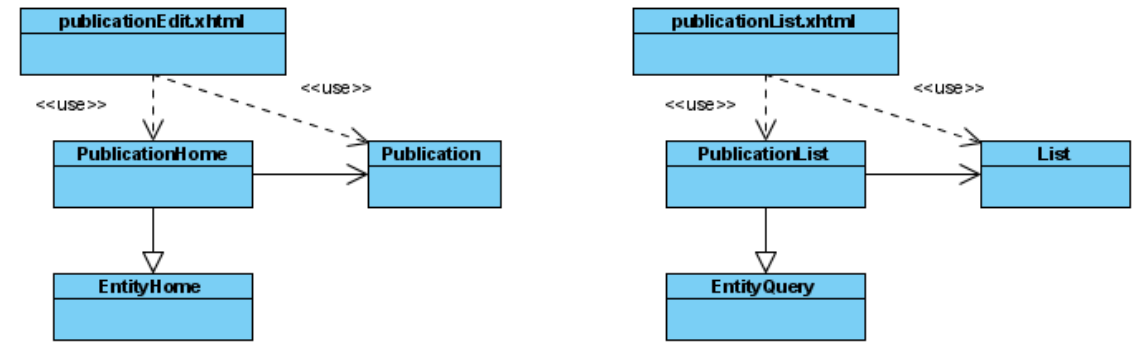
6.2.2 Správa báze znalostí

Implementace uživatelského rozhraní pro správu databáze se nachází v balíčku *action.admin*. Základ rozhraní tvoří formuláře pro správu každého typu entity z diagramu tříd na obrázku 6.3. Pro každou entitu je definována stránka pro listování seznamem entit a stránkou s formulářem pro provádění CRUD operací nad danou entitou. Pro implementaci budou

použity třídy *EntityQuery* a *EntityHome* ze Seam Application Framework. Tyto dvě třídy jsou generickými typy – jedná se o šablony DAO objektů, které pracují s obecně libovolným typem entity. Při použití stačí definovat novou třídu děděním jedné z těchto generických tříd a z této třídy vytvořit anotací *Name* Seam komponentu.

Šablona *EntityQuery* slouží pro zobrazování výsledků zadaného JPQL dotazu. Poskytuje zobrazování výsledků po stránkách, řazení a filtrování výsledků. Šablona *EntityHome* zajišťuje CRUD operace nad entitou, přičemž typ entity je definován v parametru generické třídy. Doplnit stačí aplikační logiku pro zadávání vazeb na jiné entity. Aby toto zadávání bylo uživatelsky přívětivé, je nejjednodušší využít stránku se seznamem entit, na kterou má vést reference. Z této stránky přenést jako parametr HTTP požadavku identifikátor uživatelem vybrané entity a entitu s tímto identifikátorem pak vložit do modifikované entity v původním formuláři. Ke komponentám vzniklým na základě šablon je ještě potřeba vytvořit Facelets stránky, provázat je s vytvořenými komponentami pomocí Expression Language (EL) a definovat navigační pravidla. Vše ostatní poskytnou komponenty, která obsahuje instanci modifikované entity a zapouzdřují databázové operace s touto entitou. Komponenty jsou umístěny v konverzačním kontextu. Interakce uživatelem s vytvořeným formulářem tvoří Seam konverzaci a zároveň také transakci. Změny jsou promítnuty do databáze až při opouštění formuláře, kdy končí konverzace (v souladu s principem Seam Long Running Conversation, který se snaží omezit přístup do HTTP session a databáze).

Na obrázku 6.4 je znázorněn diagram tříd komponent pro správu entity *Publication*. Komponenta *publicationHome* je tvořena třídou *PublicationHome*, která vznikla děděním z generické třídy *EntityHome*. Zobrazení obstarává stránka *publicationEdit.xhtml*, která skrz EL volá metody komponenty *PublicationHome* a přistupuje k vlastnostem entity *Publication*. Komponenta *publicationList* zobrazuje seznam všech komponent. Komponentu tvoří třída *PublicationList*, která dědí z třídy *EntityQuery*. Pro zobrazení slouží Facelts stránka *publicationList.xhtml*.

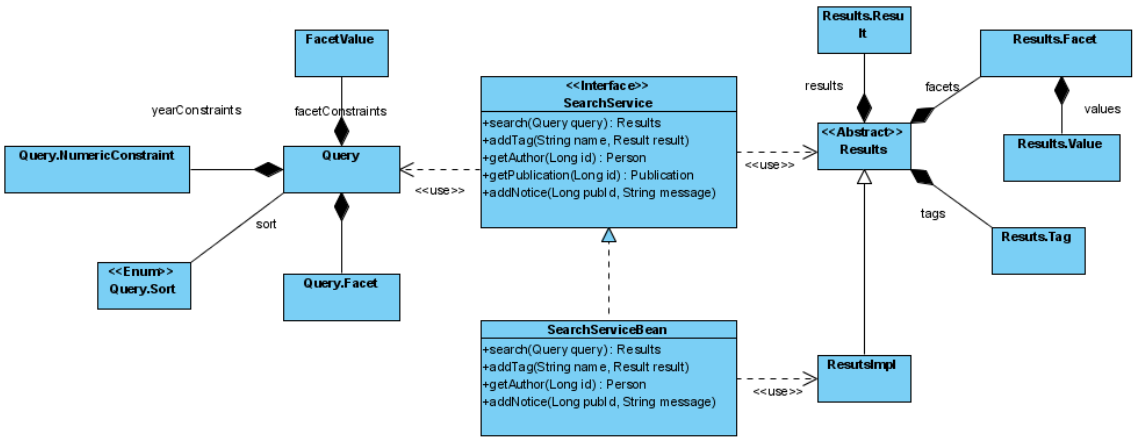


Obrázek 6.4: Použití DAO šablon

6.2.3 Správa báze znalostí

Vyhledávací rozhraní se skládá z vyhledávací služby umístěné v balíčku *action.search* a komponenty uživatelského rozhraní v balíčku *action*. Vyhledávací služba je reprezentována stateful session bean komponentou *searchService*. Tato EJB komponenta je současně také Seam

komponentou umístěnou v aplikačním kontextu. Komponenta vzniká při spuštění aplikace a svůj stav udržuje po celou dobu běhu aplikace.



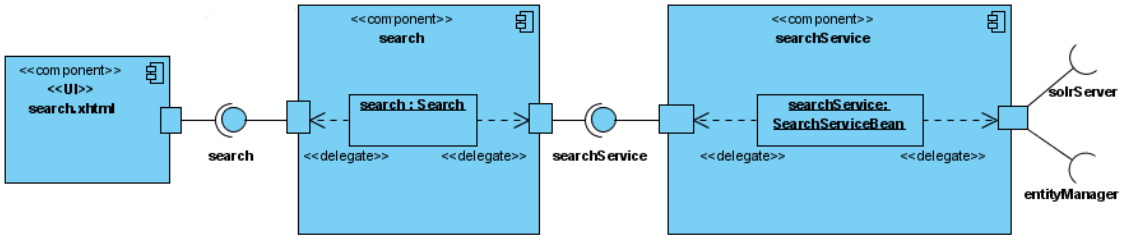
Obrázek 6.5: Diagram tříd implementujících vyhledávací službu

Komponenta *searchService* přijímá dotazy ve formě objektů třídy *Query* a vrací výsledky v podobě objektu, který vznikl děděním z abstraktní třídy *Results*. Komponenta transformuje objekt třídy *Query* na obecný dotaz, jenž je odeslán vyhledávacímu serveru Solr. Spojení se serverem zajišťuje knihovna Solrj, která komunikuje se serverem přes HTTP protokol nebo může i přímo přistupovat k indexu v lokálním souborovém systému. Obecný dotaz je reprezentován objektem třídy *SolrQuery* z knihovny Solrj. Jako odpověď je vrácen objekt třídy *QueryResponse*, který je transformován na objekt třídy Results. Výsledky jsou transformovány do tvaru, který umožňuje snadné použití pro různé prvky uživatelského rozhraní, jako například textové a numerické fasety, hierarchické fasety, popisky (tagy), zobrazované vlastnosti publikací s možností navigace na další informace atd. Poslední z jmenovaných prvků uživatelského rozhraní vyžaduje pro svou funkci přístup k perzistentním entitám uložených v databázi. Kromě vyhledávání komponenta *searchService* zajišťuje také získávání metadat přímo z databáze a to nezávisle na vyhledávání v Solr indexu.

Na obrázku 6.5 je zobrazen diagram tříd souvisejících s komponentou. Třída *Query* poskytuje prostředky pro snadné definování všech vlastností dotazu. Struktura dotazu je popsána také několika vnitřními třídami. Abstraktní třída *Results* slouží jako kontejner pro výsledky. Opět definuje několik vnitřních tříd, které výsledky strukturují. Třída *ResultsImpl*, jež dědí této abstrakní třídy, k tomuto kontejneru dodává metody pro naplnění tohoto kontejneru hodnotami. Třídy *Query*, *Results* a jejich vnitřní třídy obsahují řadu atributů a metod. Kvůli zachování přehlednosti diagramu byly uvedeny jen názvy těchto tříd, pouze u třídy *SearchServiceBean*, implementující komponentu vyhledávací služby, a jí poskytovaného rozhraní *SearchService* jsou uvedeny také hlavičky veřejných metod.

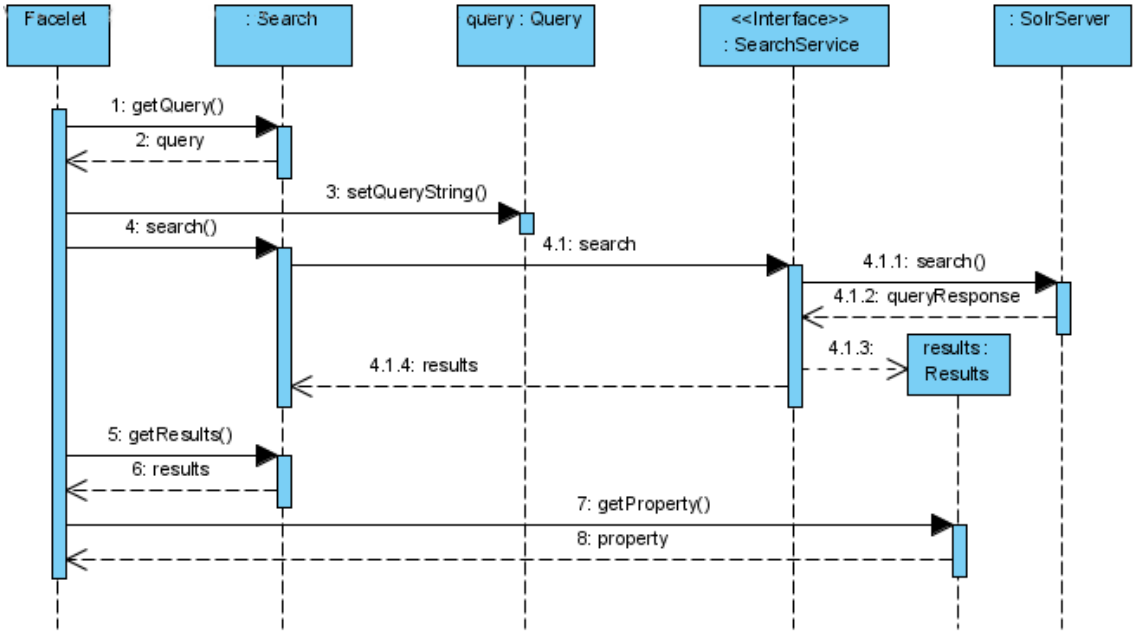
Komponentu *searchService* používá Seam komponenta *search*, tvořená třídou *Search*, která obstarává vazbu mezi touto vyhledávací službou a uživatelským rozhraním. Komponenta search tedy vystupuje v roli kontroleru návrhové vzoru MVC. Model je tvořen komponentou *searchService* a pohled Facelets stránkou.

Na obrázku 6.7 je znázorněna interakce jednotlivých komponent během vyhledávání. Diagram začíná odesláním HTTP požadavku z prohlížeče klienta. Jako první je provedena aktualizace hodnot svázaných pomocí EL s vizuálními komponentami na HTML stránce (více podrobností o životním cyklu JSF požadavku lze nalézt např. v knize [5]). V diagramu



Obrázek 6.6: Diagram komponent vyhledávání

je tato činnost pro jednoduchost ilustrována voláním metody *getQuery* komponenty *search*, která získá referenci na objekt třídy *Query* a následně je v tomto objektu voláním metody *setQueryString* aktualizována vlastnost *queryString*. Po dokončení aktualizace je zavolána metoda *search* komponenty *search*, která předá požadavek vyhledávací službě reprezentované komponenotu *searchService*. Vyhledávací služba transformuje objekt třídy *Query* na objekt třídy *SolrQuery* a odešle požadavek Solr serveru. Po obdržení odpovědi transformuje tuto odpověď na objekt třídy *Results* a tento objekt vrátí. Jako poslední je v diagramu naznačena aktualizace hodnot ve stromu vizuálních komponent před renderováním HTML stránky. Z komponenty *search* je získána reference na v ní uložený objekt s výsledky (volání metody *getResults*) a pak jsou postupně čteny atributy tohoto objektu a aktualizovány hodnoty vizuálních komponent.



Obrázek 6.7: Interakce komponent během vyhledávání

6.2.4 Import a export z/do databáze

Import a export metadat poskytují třídy umístěné v balíčku *batch*. Pro import je zvolen formát XML, který umožňuje snadné proudové zpracování.

Při importu vzniká problém, jak popsat strukturovaná data v serializovaném XML formátu. Navrhovaný XML formát musí dokázat importovat vlastnosti všech entit a zároveň i vztahy mezi nimi. Je potřeba, aby importované entity měly přiřazenu identitu, na kterou se půjde odkazovat z jiných importovaných entit. Tvar identifikátoru může být přitom libovolný, po importování bude entitě přidělena nová identita, která bude jedinečná v rámci celé databáze. Kvůli zjednodušení importu je ovšem vhodné zavést také možnost zadat hodnotu přímo bez nutnosti odkazovat se na jinou entitu, i když je daný atribut referencí. Ne vždy jsou importovaná metadata komplexní a pokud jsou některé hodnoty dostupné pouze jako textové řetězce, musely by pro ně být vytvářeny prázdné entity jen s jedním zadaným atributem pro jméno a do odkazující publikace by musel být vložen odkaz v podobě identifikátoru nově vytvořené entity. Tuto činnost může provádět přímo aplikace pro import a není nutné se jí zabývat při přípravě XML dokumentu s daty pro import. Tuto vlastnost lze využít např. při automatické extrakci metadat z existujících publikací, při které získaná metadata popisují publikace pouze textovými hodnotami bez vzájemných vazeb.

Následuje příklad XML formátu pro import. Úplná definice formátu bude přiložena ke zdrojovým kódům aplikace v podobě XML Schema. V navrhovaném formátu jsou hodnoty zadávány mezi značky se jmény atributů databázových entit. Výše popsaný problém referencí je vyřešen pomocí atributu *ref*. U XML značek, které reprezentují atribut s referencí na jinou entitu, je možné zadat tento atribut. Pokud nese hodnotu *external*, znamená to, že mezi značkami je zadán identifikátor jiné entity z daného XML dokumentu. V případě hodnoty *internal* je mezi značkami zadán identifikátor entity, která již existuje v databázi. Jiná hodnota nebo jestliže je atribut *ref* nedefinován znamená, že mezi značkami je přímo uvedena hodnota atributu name anonymní entity. V tom případě bude v databázi vytvořena nová entita s daným jménem. Pokud se v importovaném XML dokumentu vyskytuje více anonymních entit se stejným jménem, je v databázi vytvořena pouze jedna entita, na kterou vedou všechny reference

```
<import>
  <person>
    <id>idOsoby1</id>
    <name>Jméno osoby</name>
    <organization ref="internal">1</organization>
  </person>
  <organization>
    <id>idOrganizace</id>
    <name>Jméno organizace</name>
    <street>Ulice</street>
    <city>Město</city>
    <zip>PSČ</zip>
    <country>Země</country>
  </organization>
  </project>
    <id>idProjektu1</id>
    <name>Jméno projektu</name>
    <topic>Oblast výzkumu</topic>
    <organization ref="external">idOrganizace</organization>
    <person>Jméno jiné osoby</person>
  </project>
```

```
  <publisher>
    <name>VydavatelA</name>
  </publisher>
  <conference>
    <name>Jméno konference</name>
    <begin>2010-01-30</begin>
  </conference>
  <publication>
    <id>idPublikace</id>
    <uri>URI poblikace</uri>
    <language>jazyk</language>
    <year>2000</year>
    <issbn>ISBN</issbn>
    <topic>
      <id>B.1.1.2</id>
      <subject>Předmět článku 1</subject>
      <subject>Předmět článku 2</subject>
    </topic>
    <abstract>text abstraktu publikace</abstract>
    <note>poznámka</note>
    <plaintext>cesta k souboru s textem publikace</plaintext>
    <author ref="external">idOsoby1</author>
    <keyword>klíčové slovo</keyword>
    <tag>popisek</tag>
    <publisher>VydavatelB</publisher>
    <project ref="external">idProjektu1</project>
    <conference>Jméno konference</conference>
  </publication>
</import>
```

V uvedeném příkladu lze vidět, že například na entitu importující vydavatele *VydavatelA* se nelze odkazovat, protože nemá definovánu identitu. U importované publikace je vydavatel zadán přímo, je vytvořena nová entita vydavatele s názvem *VydavatelB*. Naproti tomu importovaná osoba (první importovaná entita) se v domácí organizaci osoby odkazuje na v databázi již existující entitu s identifikátorem 1. Ostatní reference jsou v rámci uvedeného XML dokumentu.

Aplikace, zpracovávající XML dokumenty výše ve výše popsaném formátu, využívá SAX XML parser. Při parsování postupně načítá entity a jakmile je některá entita načtena celá, je SQL příkazem INSERT uložena do databáze. Při zpracování XML dokumentu je potřeba uchovávat mapování mezi identifikátorem importované entity v XML dokumentu a identifikátorem přiděleným po vložení do databáze kvůli možným referencím. Aby bylo mapování prováděno dostatečně rychle, jsou datové struktury zajišťující mapování umístěny v operační paměti ve formě vyhledávacích stromů nebo hashovaní tabulky. Velikost dostupné operační paměti u dnešních počítačů by měla být dostatečná i pro rozsáhlé soubory importovaných entit. U mapování je také nutné řešit dopředné reference na entity, které ještě nebyly importovány. Pokud je dopředná reference použita, je potřeba vytvořit v databázi prázdnou entitu a zapamatovat si mapování z identifikátoru použitého v referenci na iden-

tifikátor nově vloženého záznamu. Při pozdějším importu entity, na niž reference vedla, je pak v databázi naplněna dříve vytvořená prázdná entita.

Aplikace pro export metadat vytváří XML dokument pro aktualizaci Solr indexu (Formát tohoto dokumentu byl popsán v kapitole 3). Aplikace postupně zpracovává všechny entity typu *Publication*. Hodnoty vlastností této entity zapisuje jako hodnoty polí dokumentů v Solr indexu. U vlastností, které jsou referencí na jinou entitu, je zapsána vybraná vlastnost této entity (zpravidla je to vlastnost *name*).

Kapitola 7

Implementace

V této kapitole je rozebrána implementace navržených komponent. Uvedeny jsou také problémy, které se během implementace vyskytly, a popsány změny v návrhu, které byly při řešení těchto problémy provedeny.

7.1 Služba pro vyhledávání

Vyhledávací služba odděluje uživatelské rozhraní od vlastního použití knihovny Solrj. K tomuto účelu je definováno rozhraní *SolrService* s deklarací poskytovaných metod a dále třídy *Query* a *Results* pro reprezentaci strukturovaných dotazů a výsledků vyhledávání.

Jak již bylo uvedeno při návrhu, objekt třídy *Query* je naplněn komponentou *search* na základě obsluhy událostí z HTML stránky. V dotazu, reprezentovaném třídou *Query*, se nacházejí vlastnosti dotazu ve formě jednoduchých datových typů a řetězců (pozice ve výsledcích, text fulltextového dotazu) nebo seznamů (seznam fasetových omezení). Třída také pro reprezentaci dotazu používá několik vnitřních tříd. Definuje také celou řadu metod pro snadné zadávání, změnu, vyhledávání a porovnávání vlastností dotazu. Služba pro vyhledávání, tedy komponenta searchService, převede dotaz reprezentovaný objektem třídy *Query* na objekt třídy *SolrQuery* a odešle jej prostřednictvím knihovny Solrj Solr serveru. Odpověď serveru ve podobě objektu třídy *QueryResponse* je transformována na objekt třídy *Results*, který nese výsledky. Třída *Results* je abstraktní, komponenta *searchService* používá objekt třídy *ResultsImpl*, který vznikl děděním z třídy *Results*. Vlastní naplnění probíhá v konstruktoru této třídy, objekt třídy *QueryResponse*, který má být transformován, je konstruktoru předán jako parametr.

Výsledky v objektu třídy *Results* lze rozdělit na dvě části – vlastní kolekci nalezených dokumentů a kolekce hodnot faset. Kromě toho jsou ve výsledcích zapsány další informace, např. pozice ve výsledcích pro stránkování, celkový počet nalezených dokumentů apod.

7.1.1 Fulltextové vyhledávání

Implementace tohoto vyhledávání je nejjednodušší – fulltextový dotaz je řetězec, ve třídě Query uložený v atributu queryString. V komponentě searchService se tento řetězec jen zkopíruje do objektu třídy SolrQuery. Syntaxi dotazu není potřeba řešit, je použita implicitní syntaxe, mírně lišící se od původní syntaxe dotazu v Lucene (specifikaci lze nalézt v Solr Wiki[1].

[1] `http://wiki.apache.org/solr/SolrQuerySyntax`

7.1.2 Fasetové vyhledávání

U fasetového vyhledávání je třeba implementovat dvě základní činnosti – získání hodnot faset, jež jsou poté zobrazeny uživateli, a použití některé z těchto hodnot jako fasetového omezení. V aplikaci byly implementovány tyto fasety:

- textové: autor, oblast výzkumu, typ publikace, vydavatel, konference
- numerické: rok vydání

Autor, oblast výzkumu a rok vydání jsou zároveň také hierarchické fasety. Dále bylo implementováno filtrování výsledků na základě klíčových slov a popisků, což jsou, jak je ukázáno dále v textu, v podstatě jen jiným způsobem zobrazené fasety.

Oproti návrhu není implementována faseta pro jazyk publikace. Vyhledávač, jak byl vypracován v rámci diplomové práce, podporuje pouze anglický jazyk (zde je míněna podpora jazyka při využití indexu, kde je přirozený jazyk zpracováván pomocí analyzátorů, pracujících jako filtry).

7.1.3 Textové fasety

Jak už bylo dříve uvedeno, fasety jsou zobrazeny až spolu s výsledkem fulltextového dotazu. Obě akce lze tedy provádět současně – u každého dotazu zaslaného Solr serveru se zároveň získají hodnoty faset, které jsou po transformaci vráceny v objektu třídy *Results*. V tomto objektu je každá faseta reprezentována objektem vnitřní třídy *Facet*, která obsahuje seznam hodnot a počtem jejich výskytů. Všechny fasety jsou pak uloženy v seznamu faset. Tento seznam je implementován pomocí standardní třídy *Map*, kde klíčem je jméno fasety. Jestliže uživatel vybere některou hodnotu fasety jako fasetové omezení, je aktualizován příslušný objekt třídy *Query* a zavolána metoda search komponenty *searchService*, která vrátí aktualizované výsledky.

Ve třídě *Query* je pro každou fasetu vyhrazeno pole použitých fasetových omezení. Jedné fasetě tedy může být přiřazeno více omezení. Nalezené dokumenty jsou filtrovány na průnik těchto omezení (tzn. dokument musí vyhovovat všem definovaným fasetovým omezením). Fasetová omezení mohou být na různých úrovních hierarchie. Omezení je jednoznačně definováno fasetou, ke které patří, hodnotou a identifikátorem této hodnoty, pokud je definována a příznakem skupiny (viz hierarchické fasety).

Fasety pracují nad poli v Solr indexu, jež musí být indexována a neměla by být pro správnou funkčnost tokenizována. Pro textové fasety je potřeba definovat vlastní pole, který vyhovuje těmto podmínkám. Při návrhu metadat byly tyto pole označeny postfixem *-facet*. Obecně jsou tyto pole v souboru *schema.xml*, který definuje strukturu dokumentů uložených v indexu, vytvářeny kopií původního pole pomocí značky *copyfield*. V komponentě *searchService* je mapování jména fasety na použité pole v indexu definováno v objektu *mapping*. Problém nastává u víceslovných hodnot, například jméno autora se skládá z křestního jména a příjmení. Solr pracuje jen s jednoslovnými hodnotami faset. Řešením je nahrazovat znak mezery jiným vyhrazeným znakem. Při implementaci byl zvolen znak _. Dalším problémem je, jak zjistit, u kolika dokumentů není hodnota některé fasety zadána, tzn. pole nesoucí hodnotu dané fasety je nedefinované. Byla zavedena speciální hodnota *_empty* značící prázdnou hodnotu. Tato hodnota nemůže být v konfliktu s existující hodnotou fasety, protože u existujících hodnot je znak _ nahrazován mezerou. To, že hodnoty polí budou v tomto formátu, je zajištěno při exportu metadat z databáze. Nepoužívá se tedy výše uvedená metoda, kdy pole pro fasetu kopíruje Solr, hodnota je ve speciálním tvaru vytvořena již při

zápisu XML dokumentu pro aktualizaci indexu. Ve třídách *Query* a *Results* jsou hodnoty uloženy ve své původní podobě, převod na tvar použitý v indexu zajišťuje komponenta *searchService*.

7.1.4 Numerické fasety

Faseta rok vydání je implementována jako numerická faseta. Implementace je provedena tak, že umožňuje případné rozšíření o další numerické fasety. Faseta pracuje s indexovým polem *year* a je hierarchická – hodnoty jsou zobrazovány po desetiletích, po výběru desetiletí pak po jednotlivých letech. To, která úroveň má být zobrazena, je určeno na podle toho, zda je definováno alespoň jedno omezení na tuto fasetu. Pokud ano, jsou získány hodnoty v rozsahu a v intervalech definovaných fasetovým omezením, tedy v rozsahu jednoho desetiletí (omezení na první úrovni fasety) nebo jednoho roku (omezení na druhé úrovni fasety) a s intervalem o velikosti jednoho roku. V opačném případě je rozsah zobrazovaného období omezen na roky 1950 až 2020 s intervalem deset let. Hodnoty přesahující daný rozsah jsou zobrazeny všechny společně v hodnotě pro starší resp. novější publikace.

V třídách Query a Results jsou numerické fasety reprezentovány vnitřními třídami *NumericFacet*. V dotazu zaslaném Solr serveru je použit princip využívající parametr *facet.query*, který byl popsán v kapitole 3. Intervaly hodnot fasety se převádějí z číselné reprezentace v třídě *NumericFacet* na textovou reprezentaci, pro každý interval je vytvořen samostatný dotaz, který představuje jednu hodnotu fasety (Solr parametr *facet.query*). V získaných výsledcích je naopak převedena textová hodnota na číselný interval.

7.1.5 Hierarchická faseta Autor publikace

Jako hierarchická faseta je implementována také textová faseta pro jméno autora. Tato faseta na své první úrovni zobrazuje jména autorů ve skupinách po třech písmenech abecedy podle příjmení. Počet tří písmen byl vybrán proto, že anglická abeceda obsahuje 28 písmen a při třípísmenných skupinách nepřesahuje počet hodnot na první úrovni hierarchie velikost deset.

Pro implementaci hierarchie je zavedeno v indexu pomocné pole *author-bucket*. Toto pole obsahuje jméno skupiny, ke které patří příjmení autora. V kapitole 4 byl na základě knihy [3] uveden doporučený postup pro naplnění hodnoty tohoto pole – použít v souboru *schema.xml* značku *copyField*, která zkopíruje hodnotu pro pole *author*, nově vytvořenému poli přiřadit analyzátor regulárních výrazů, který získá první písmeno příjmení a na základě souboru s definicí synonym určí skupinu. V souboru se synonymy by ke každému písmenu byla přiřazena příslušná skupina. Zde byl použit jednodušší postup, hodnota pole *author-bucket* se určí při exportu metadat z databáze. Problémem je, že jméno i příjmení autora je v entitě *Person* uloženo v jednom atributu *name*. V programu pro export je třeba hodnotu atributu rozdělit na jednotlivá slova a zjistit první písmeno posledního slova (obvykle je jméno a příjmení zadáno v tomto pořadí). Současně je také odstraněno případné diakritické znaménko. Poté je na základě tohoto písmena přidělena hodnota pole *author-bucket*. Zavedení pole *author-bucket* tedy přináší změnu schématu metadat definovaného v kapitole 4.

Úroveň, na které byla hodnota fasety vybrána, je určena na základě příznaku ve třídě *FacetValue*. Tato třída slouží pro reprezentaci jednoho fasetového omezení. Kromě zmíněného příznaku úrovně obsahuje tato třída vlastní hodnotu a identifikátor této hodnoty (účel identifikátoru je vysvětlen dále v textu). Uživatelem vybrané fasetové omezení tedy obsahuje informaci o vlastní hodnotě fasety a zároveň také informaci, na které úrovni byla

tato hodnota vybrána. U faset, které nepoužívají hierarchii, je příznak úrovně a všechny na něj vázané akce jednoduše ignorovány. To dovoluje použít jeden zdrojový kód pro všechny textové fasety. Navíc použití příznaku skrývá způsob implementace pomocí dvou polí v indexu.

Zbývá ještě definovat pravidlo pro určení úrovně, ze které mají být získány hodnoty fasety pro zobrazení. Pokud není specifikováno žádné omezení pro danou fasetu, tzn. žádná hodnota fasety nebyla vybrána, je úroveň stanovena na základě počtu hodnot fasety. Součástí dotazu zaslaného Solr serveru je požadavek na fasetové hodnoty pole *author-facet* i *author-bucket*. Pokud server vrátí větší počet hodnot pro pole *author-facet*, než je stanovený limit, bude použita první úroveň hierarchie, tedy skupiny. V opačném případě budou zobrazeny hodnoty pole *author-facet*, protože zobrazení ve skupinách by při tak malé počtu hodnot vnášelo pouze nepřehlednost. Přesažení limit se indikuje tak, že se maximální počet hodnot vracený Solr serverem nastaví na velikost limitu plus jedna, což je nejmenší hodnota větší než limit a nejsou tedy zbytečně přenášeny další hodnoty pole author-facet, které by zůstaly nevyužity.

Pokud je specifikováno alespoň jedno omezení dané fasety, je zobrazena druhá úroveň hierarchie, protože první úroveň už byla zadána v některém z existujících omezení nebo v případě malého počtu hodnot nebyla první úroveň uživateli ani nabídnuta.

Omezení hodnot na nižší úrovni, aby byly pouze z této úrovně, je automatické. Při výběru hodnoty na vyšší úrovni vytvoří tato hodnota nové fasetové omezení, které je v objektu třídy *SolrQuery* přidáno jako parametr *facet.filter* a v aktualizovaných výsledcích jsou již jen hodnoty patřící do příslušné skupiny. Problém nastává u vícehodnotových polí, což jsou i pole obsahující jméno autora. Výběr fasetového omezení specifikuje nové podmínky dotazu, jehož výsledkem je množina publikací. Při vybrání některé skupiny jsou tedy v nových výsledcích obsaženy publikace, jméno jejichž autora patří do dané skupiny. Ovšem autorů může mít publikace více a jména těchto autorů mohou patřit do dalších skupin. Fasetové hodnoty pole *author-facet* jsou hodnoty jmen všech autorů nalezených publikací, tedy i těch, kteří nepatří do skupiny specifikované fasetovým omezení na první úrovni hierarchie. Jinými slovy, výsledkem dotazu je vždy množina publikací a faseta autor jen popisuje určitou vlastnost této množiny. Aby hierarchická faseta bylo pro uživatele intuitivní, musí být hodnoty fasety na nižší úrovni filtrovány, aby byly zobrazeny jen jména autorů patřící do vybrané skupiny.

Nabízí se i další méně vhodná řešení tohoto problému. Jen pro úplnost je uveďme. První možností by bylo zavést novou vlastnost publikace hlavní autor. Faseta autor by pak pracovala nad touto vlastností, která by mohla nabývat nejvýše jedné hodnoty a odpadnul by tak výše popsaný problém. Tato vlastnost by však byla značně umělá, u vědeckých publikací jsou zpravidla všichni autoři uváděni jako stejně významní a případné zobrazení méně autorů je jen z typografických důvodů. Druhá možnost by byla nepoužít hierarchii a zobrazovat všechny autory s možností stránkování. Solr pro tuto možnost nabízí prostředky, stejně jako u definice pozice v nalezených dokumentech lze stanovit pozici v hodnotách faset a to pro každou fasetu zvlášť. Toto řešení by však přinášelo pouze nepřehlednost uživatelského rozhraní. Fasetové vyhledávání by mělo umožňovat filtrovat výsledky rychlým způsobem, navíc pro navigaci ve stránkách hodnot by bylo na velmi malém prostoru. Spíše než aby uživatel zdlouhavě listoval seznamem hodnot, specifikuje lépe fulltextový dotaz. Listování v seznamu hodnot faset bylo implementováno, avšak bylo shledáno jako nevhodné. Byly odstraněny prvky uživatelského rozhraní s tlačítky pro listování, vlastní implementace byla ponechána a lze ji jednoduše zpřístupnit doplněním odstraněných tlačítek.

7.1.6 Hierarchická faseta Oblast výzkumu

Poslední hierarchickou fasetou je faseta Oblast výzkumu. Hierarchie je vytvářena na základě klacifikace ACM. Na rozdíl od fasety Autor není hierarchie vytvářena až za běhu v závislosti na počtu zobrazovaných hodnot, ale je již dána specifikací ACM CCS.

V indexu je hierarchie vyjádřena tím, že pro každou úroveň náleží vlastní typ pole – jsou to pole *topic-facet-l0* až *topic-facet-l3*. Hodnoty polí jsou odvozeny během vytváření XML dokumentu pro aktualizaci indexu. V ACM CCS jsou identifikátory na jednotlivých úrovních odděleny tečkou. Identifikátor hodnoty v hierarchi je definován idetifikátory na všech úrovních od kořene hierarchie (např. hodnota A.3.1 je složena z identifikátorů A, 3 a 1). Pro každou hodnotu lze tedy jednoduše získat všechny její nadřízené kategorie a ty pak tvoří cestu stromem hierarchie ACM CCS. V indexu jsou také uloženy názvy použitých kategorií. Celá hierarchie ACM CCS je na začátku načtena z XML dokumentu se specifikací do paměti a je přístupný ve formě stromu skrz rozhraní Document Object Model (DOM). Stačí tedy pouze projít stromem získanou cestou a v každém navštíveném uzlu přečít název kategorie.

V samotné aplikaci je pak každé hodnotě přiřazena úroveň, na které se tato hodnota nachází (atribut *level* ve třídách *Facet* a *FacetValue*). Na počátku je zobrazena nejvyšší úroveň, při výběru fasetového omezení z této úrovně je zobrazena nižší úroveň atd. Avšak ne všechny podstromy obsahují hodnoty na všech stupních hierarchie, listové uzly se obecně nachází na libovolné úrovni. Index umožňuje uložit pouze jednotlivá pole, je tedy nutné zvolit reprezentaci hodnot polí *topic-facet-l0* až *topic-facet-l3* takovou, aby odrážela strukturu stromu kategorií. Byl zvolen tento formát: *idKategorie_názevPředmětuKategorie#názevKategorie*. To, zda daná hodnota je listem či ne, lze odvodit z toho, že listové kategorie nemají dle specifikace ACM CCS přiřazen název (hodnoty jsou ve tvaru *idKategorie_názevPředmětuKategorie*) a uzly na vyšších úrovních namají definovány předměty kategori (hodnoty jsou ve tvaru *idKategorie#názevKategorie*). Díky tomu lze odvodit, zda daná hodnota je listovým uzlem či ne a zda tedy mají být uživateli zobrazeny kategorie na nižší úrovni fasety.

7.1.7 Klíčová slova

Klíčová slova jsou v podstatě jen jinak zobrazenou fasetou. Hodnoty této fasety nejsou získány ze Solr serveru jako kolekce hodnot fasety a počtu výskytů. Jsou použity přímo hodnoty u konkrétní publikace. Jako fasetová omezení však klíčová slova fungují stejně jako u kterékoliv jiné textové fasety.

7.1.8 Popisky

Jako uživatelská navigace kombinují popisky (tagy) vlastnosti klíčových slov a standardních textových faset. Jsou zobrazovány u každého dokumentu a zároveň je možné zobrazit všechny popisky pro danou kolekci výsledků, jako u ostatních textových faset. Rozdíl je v tom, že z hodnot není vypisován číselný počet výskytů, ale v závislosti na počtu výskytů je změněna velikost písma položky v seznamu (tzv. tag cloud popsaný v čtvrté kapitole). Velikost písma je vypočítána na základě vzorce [8].

$$s_i = \left\lceil \frac{f_{max} \cdot (t_i - t_{min})}{t_{max} - t_{min}} \right\rceil$$

kde

s_i	velikost písma
f_{max}	maximální velikost písma
t_i	počet výskytů hodnoty
t_{min}	minimální počet výskytů v dané kolekci hodnot
t_{max}	maximální počet výskytů v dané kolekci hodnot

U velkého počtu indexovaných dokumentů je výskyt jednotlivých popisků popsatelný mocninnou řadou. Pak by bylo lepší nahradit použitý lineární vzorec logaritmickým [28].

Jedna hodnota pro tag cloud je pak je reprezentována objektem třídy *Results.Tag*, ve které je mimo názvu hodnoty uložena vypočítaná velikost.

7.1.9 Řazení výsledků

Výsledky vyhledávání jsou řazeny buďto podle relevance (automatické hodnocení Solru) nebo podle počtu citací. Použitý způsob řazení je určen v parametru *sortFiled* v objektu třídy *SolrQuery*.

7.1.10 Zpracování kolekce dokumentů ve výsledcích hledání

Dokumenty vrácené Solr serverem jsou reprezentovány jako kolekce objektů třídy *Results.Result*. Při vytváření prvků této kolekce jsou čteny hodnoty polí v odpovědi Solr serveru. Ten vrací pole dokumentu v asociativním poli, s mapováním jména pole na jeho hodnotu. Hodnoty polí jsou zapsány do atributů struktury *Result*. Oproti návrhu metadat je v indexu uložen i identifikátor publikace a to kvůli snadnému napojení na databázi při dalších doplňujícím dialogu na detail publikace. Problém nastává u dialogu s detailem autora. Je potřeba získat entitu *Person*, která patří danému autoru. Avšak jméno autora neidentifikuje jednoznačně entitu *Person*. V indexu je potřeba mít společně se jménem autora také identifikátor jeho entity *Person*. Nelze zavést další pole pro identifikátor, protože pole *author* v indexu je vícehodnotové a mezi hodnotami vícehodnotového pole není definováno uspořádání. Jméno i identifikátor musí být uloženy v jednom poli. Bylo využito pole *author-facet*, které je již uloženo ve speciálním formátu. Tento formát byl rozšířen o identifikátor entity *Person* daného autora. Identifikátor je zapsán za jméno autora a je od něj odděleno znakem #. Jména autorů pak v třídě *Results* nejsou uloženy jako řetězce, nýbrž jako objekty třídy *FacetValue*.

7.1.11 Získání citací

Aby vyhledávač poskytoval funkce citační databáze, musí dokázat vrátit kolekci publikací, které citují vybranou publikaci, a také kolekci publikací, které jsou citovány určitou publikací. V databázi jsou tyto informace uloženy vztahy *citedBy* a *cites*, v indexu reprezentované vícehodnotovými poli *citedby* a *cites*. V těchto polích jsou uložena URI odkazovaných publikací. Výše zmíněné typy kolekcí lze získat fulltextovým dotazem nad těmito poli. Uvažujme publikaci s URI *uriPublikace*. Kolekci publikací citujících uvažovanou publikaci získáme dotazem *cites:uriPublikace*, pro kolekci publikací citovaných uvažovanou publikací využijeme dotaz *citedby:uriPublikace*.

Aby uživatel nebyl nucen tyto dotazy psát ručně, nabízí uživatelské rozhraní u každého zobrazeného výsledku tlačítka pro jejich generování a aktualizaci výsledků. V popisku tlačítek je zároveň uveden počet citací, jenž je v indexu uložen v polích *citedby-cnt* a *cites-cnt*.

7.2 Zobrazování výsledků

Pro zobrazování výsledků slouží komponenta *search* a Facelets stránka *search.xhtml*. Komponenta zajišťuje aplikační logiku fulltextového a fasetového vyhledávání, navigaci pro výsledcích hledání, zobrazování detailů a přidávání popisků. Komponenta je umístěna do konverzačního kontextu. To zajišťuje existenci dotazu ve formě objektu třídy *Query* po celou domu interakce s uživatelem, který tak může postupně měnit podmínky dotazu a procházet výsledky hledání. Většina informací, která je vypisována z výsledků hledání je ve formě kolekcí – nalezené publikace, hodnoty faset a také fasetová omezení. Výpis je usnadněn pomocí visuálních komponent *repeat* z knihovny frameworku Facelets nebo Ajax4jsf (tato verze komponenty pracuje s kolekcemi načítanými na pozadí pomocí AJAXu), *dataList* z knihovny RichFaces nebo v některých případech také *forEach* z knihovny JSTL. Posledně jmenovaná komponenta by měla být používána omezeně, nespolupracuje se stromem vizuálních komponent v JSF.

7.2.1 Fasety

Fasetová omezení jsou zobrazena v seznamu pod polem pro zadání fulltextového dotazu. Jednotlivá omezení jsou vypisována v podobě tlačítek, vždy pro jednu fasetu v jednom řádku. Hodnoty fasetových hodnocení se čtou z objektu třídy *Query*, pro každou fasetu je v této třídě definován seznam omezení. Po kliknutí na tlačítko je dané fasetové omezení zrušeno a výsledky jsou aktualizovány. To je možné díky tomu, že každé fasetové omezení je jednoznačně identifikováno. Tlačítko pro odebrání je vykresleno také v pravé části obrazovky u hodnot faset. V kapitole 4 byl tento způsob vyloučen jako nevhodný kvůli malé velikosti. Při implementaci byl však ponechán, uživatel tak má možnost volby.

Všechna omezení tvoří průnik. Původně bylo implementována omezení tak, že se při vyhledávání vytvářelo sjednocení omezení každé fasety. Bylo to však značně neintuitivní a nepraktické – místo toho, aby se s přibývajícími omezeními množina vyhovujících publikací zmenšovala, byla stále větší.

7.2.2 Dynamicky zobrazované detaily publikací

Při kliknutí na jméno autora nebo odkaz Detail u některé z nalezených publikací jsou zobrazeny další informace, které jsou získány z databáze. Informace jsou načteny na pozadí, neprovádí se aktualizace celé HTML stránky. Toto chování je implementováno pomocí komponent z knihovny Ajax4jsf. Vyskakovací dialog se získanými detaily je HTML prvek div, jehož kaskádové styly tento div zobrazí, aniž by měl vliv na rozložení ostatních HTML prvků na stránce. Požadované informace jsou z databáze získány prostřednictvím metod komponenty *searchService*, která vrátí požadovanou entitu *Publication* nebo *Person* na základě její identifikace.U dialogu s detailem publikace je navíc k dispozici formulář pro možnost nahlášení chybných dat. Původně bylo zamýšlelo použít pro vyskakovací dialogy komponentu ToolTip z knihovny RichFaces. Tato komponenta však ve verzi pracující s použitou verzí frameworku Seam obsahuje chybu, kdy komponenta nedokáže získat svůj obsah ze serveru na pozadí. Správně pracuje pouze pokud je obsah získán ihned při renderování stránky, což je z výkonnostních důvodů v použitém případě nevhodné.

Zobrazení abstraktu a seznamu klíčových slov je implementováno pomocí komponenty TogglePanel z knihovny RichFaces. Zobrazené hodnoty jsou načteny z příslušného objektu třídy *Results.Result*.

U každé publikace je ve spodní části zobrazen seznam popisků (tagů) připojených k publikaci. V pravé části tohoto seznamu je umístěn dialog pro přidání nového popisku. Po kliknutí na tlačítko dialogu se na pozadí zavolá metoda komponenty *searchService* pro přidání popisku. Nový popisek je uložen do databáze a vypsán v seznamu. Solr neumožňuje aktualizovat jednotlivá pole dokumentu, pouze celý dokument. Při aktualizaci jsou původní pole dokumentu smazána a uložena pole definovaná v příkazu pro aktualizaci. Pro přidání popisku by bylo nutné získat z databáze všechny vlastnosti publikace, vytvořit reprezentaci publikace ve formě Solr dokumentu a teprvé poté dokument aktualizovat. Proto prozatím, je popisek přidán pouze do databáze, a do indexu se promítne při příštím exportu databáze a aktualizaci indexu.

7.2.3 Navigace ve výsledcích

Navigace v nalezených publikacích je možná díky atributům *rows* a *start* v třídě *Query*, které stanovují počet vypsaných publikací resp. pozici v nalezených publikacích, a atributu *numFound* ve třídě *Results*, jenž nese informaci o celkovém počtu nalezených publikací. Navigace je implementována ve třídě *NavigationPanel*. Třída obsahuje metody pro generování vlastností tlačítek panelu a jim příslušných akcí. Tlačítka slouží pro přechod na začátek, konec, posun o jedna dopředu nebo dozadu a pro přímý přechod na některou stránku (popiskem tlačítka je pořadové číslo stránky). Při konstrukci objektu třídy *NavigationPanel* jsou zadány vlastnosti panelu – celkový počet zobrazovaných položek, počet položek na stránku, aktuální pozice a počet tlačítek pro přímý přechod.

Po stisknutí některého z tlačítek panelu je aktualizován stav panelu, přičemž stavem panelu se rozumí aktuální pozice. Stav je zkopírován do atributu *start* objektu třídy *Query* a je provedena aktualizace výsledků

7.3 Správa databáze

Rozhraní pro správu bylo implementováno s využitím Solr Application Framework, přesněji řečeno pomocí generických tříd *EntityQuery* a *EntityHome*, jak bylo popsáno v kapitole Architektura a návrh.

Textové atributy editované entity se váží přímo na formulářové prvky ve Facelets stránce, kolekce entit, jež představují vazby, jsou zobrazovány ve formě tabulek. Pro přidání nové entity do některé z kolekcí se využívá stránka pro zobrazování seznamu entit (stránka se jménem *název přidávané entity + List.xhtml*). Stránka pro zobrazení seznamu entit u každé vypsané entity nabízí buďto odkaz vedoucí na stránku s možností editace (*název entity + Edit.xhtml*) nebo odkaz pro návrat na předchozí stránku, jako parametr je předán identifikátor vybrané entity. Na předchozí stránce, což v popisovaném případě byla stránka pro editaci entity, je automaticky obnoven obsah všech formulářových prvků, i když změny v entitě nebyly uloženy. To je možné díky tomu, že Seam komponenta pro editaci (*název entity + Home*) je umístěna v konverzačním kontextu a přechod na jinou stránku proběhl v rámci této konverzace. Podmínkou ovšem je, že pro navigaci mezi stránkami jsou použity JSF komponenty CommandButton nebo CommandLink, které zajistí, že při postbacku proběhne celý životní cyklus JSF požadavku s aktualizací stromu vizuálních komponent a na ně navázaných atributů Seam komponent).

U některých atributů je třeba implementovat omezení na unikátní hodnotu. Jedná se o URI publikace a přihlašovací jméno uživatele. U popisků a klíčových slov publikace, ač se jedná o vztah s jinou entitou, by bylo nevhodné vybírat hodnoty ze seznamu. Popisky

a klíčová slova se proto k publikaci přidávají přímým zápisem. Také zde je nutné testovat, aby byl každý popisek resp. klíčové slovo svázán s danou publikací nejvýše jedenkrát a aby název popisků byl jednoslovný. Možnost přidávat popisky je v rozhraní pro správu databáze spíše pro úplnost, popisky by měly být přidávány samotnými uživateli vyhledávacího rozhraní a tím tak tvořit uživatelskou navigaci.

Při návrhu v předchozí kapitole bylo také uvedeno, že změny v editované entitě se do databáze promítnou při ukončení konverzace. Ve vytvořených formulářích je konverzace ukončena stisknutím tlačítek *Save*, *Remove* nebo *Cancel*. Aby při stisknutí tlačítka Cancel nebyly změny uloženy, je před samotným ukončením konverzace volána metoda clear() komponenty entityManager, která entitu odstraní z perzistentního kontextu (ta přejde do stavu detached).

Při implementaci formulářů pro správu uživatelů se vyskytl problém ve vztahu mezi entitou User a Person. Původní návrh byl, že entita User vzniká děděním z entity Person. Problém nastává v objektově-relačním mapování. Objektově-relační mapování není zcela transparentní a vždy přináší jistá omezení, podrobnosti lze nalézt v knize[1]. JPA nabízí tři možné způsoby mapování dědičnosti [7]:

- jedna společná tabulka pro strom dědičnosti – v každém řádku tabulky jsou vlastnosti všech tříd plus diskriminátor typu entity
- jedna tabulka pro konkrétní třídu – pro každou entitu, která není abstraktní třídou, je vytvořena jedna tabulka. V této tabulce jsou kromě atributů entity také atributy rodičovské entity
- jedna tabulka pro každou entitu – v každé tabulce odpovídají sloupce atributům entity, primární klíč v tabulce děděné entity je zároveň cizím klíčem na záznam tabulky rodičovské entity

První způsob mapování nelze použít, protože vyžaduje, aby sloupce reprezentující atributy děděných entit mohly nést hodnotu null (rodičovská třída tyto atributy nezná, příslušné sloupce tabulky by tedy měly být nedefinovány). V entitě *User* jsou však povinné atributy *login*, *password* a *role*. Další dva způsoby užít lze, jsou však určeno pro komplexnější stromy dědičnosti a do jednoduchého vztahu mezi dvěma entitami *Person* a *User* vnáší značnou složitost, hlavně na úrovni formulářů pro správu. Dědičnost také zavádí umělou podmínku, která vyžaduje, aby každý uživatel byl současně osobou. To ovšem nemusí být nutně pravda, například správce databáze nebo uživatelských účtů se nemusí být vědeckým pracovníkem. Proto místo dědičnosti byla použita běžná asociace, kdy entita *User* definuje atribut typu *Person*, který ukazuje na entitu osoby přiřazené uživateli.

7.4 Autentizace a autorizace

Autentizace a autorizace uživatelů je implementována pomocí komponenty *identity*, která je součástí frameworku Seam. Tato komponenta nese informaci o identitě uživatele a jeho rolích. V deskriptoru každé stránky (soubory *jméno stránky + .page.xml*) je stanoveno, jaké musí mít uživatel role, aby mohl danou stránku použít. V šabloně HTML stránek (soubor *view/layout/template.xhtml*) je menu, zobrazované v horní části všech stránek, zobrazováno tak, aby skrývalo odkazy na stránky, ke kterým role přihlášeného uživatele neumožňují přístup. Vlastní ověření identity uživatele je implementováno v komponentě *authenticator*.

Jedná se o Seam komponentu a zároveň také o stateless session bean. Nastavení, aby komponenta identity používala pro autentizaci právě komponentu *authenticator*, je definováno v konfiguračním souboru *resources/WEB-INF/components.xml*. Uživatelské rozhraní pro zadávání přihlašovacích údajů je umístěno v souboru *login.xhtml*. Pro změnu hesla slouží komponenta *changePassword* spolu se stránkou *changePassword.xhtml*, které nabízí obvyklý dialog (vyžádání současného hesla a zadání hesla nového). Pro úplnost zmiňme, že hesla jsou v databázi ukládána ve jako hash hodnoty.

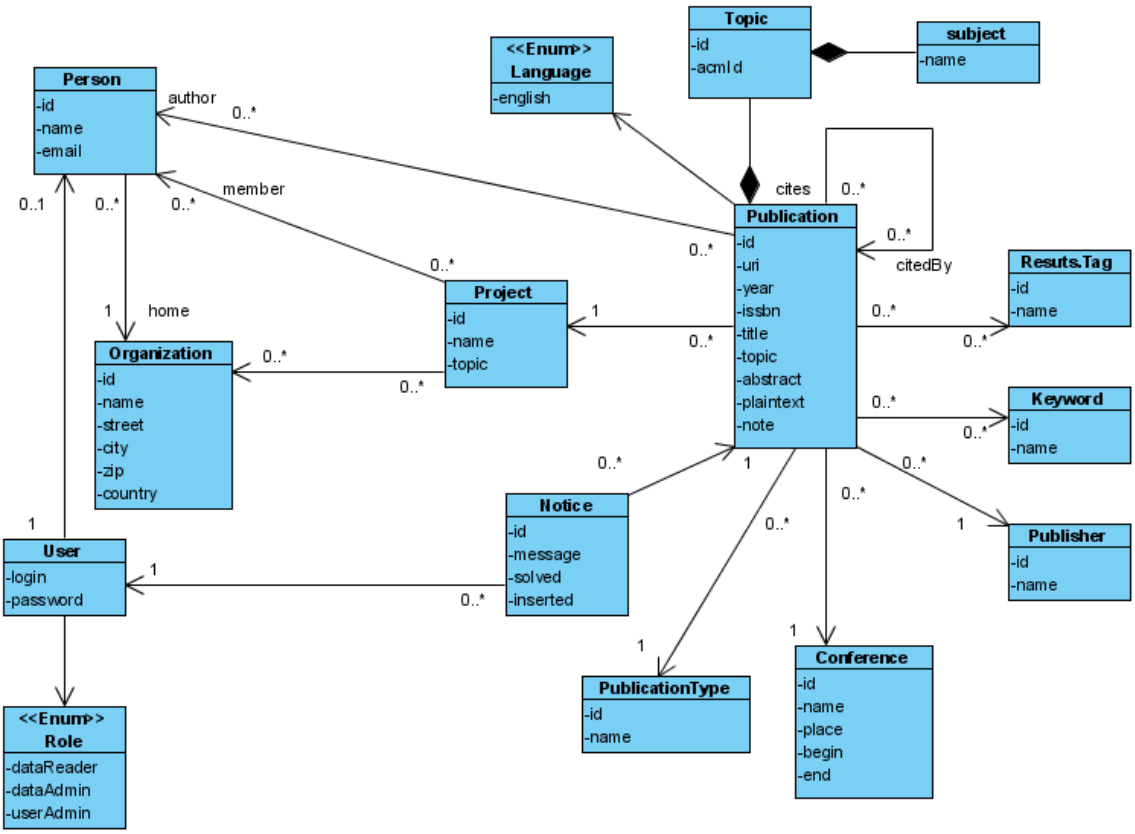
7.5 Import

Aplikace pro import byla implementována podle návrhu popsaného v předchozí kapitole. Parsování vstupního dokumentu zajišťuje SAX parser, vlastní funkcionalita je umístěna v handleru *ImportHandler*. Zápis do databáze zajišťuje třída *DbWriter*. Tato třída přistupuje prostřednictví SQL příkazů přímo k relačním tabulkám v databázi, pro připojení je použit SQL konektor, při implementaci byl použit konektor pro spojení s databázovým serverem MySQL). Pro mapování identifikátorů je použita třída Hashtable, pro každou entitu je definována jedna instance. Identifikátor entity zapsané do databáze je bezprostředně po vložení příkazem *INSERT* získán příkazem *SELECT LAST_INSERT_ID*.

Během implementace a hlavně během přípravy ukázkových dat se ukázalo, že omezit typ publikace pouze na typy definované v citačním manažeru BibTeX je zbytečně svazující (importu ukázkových dat se podrobně věnuje následující kapitola). Importovaná data mohou mít definován i jiný typ. Model perzistentních tříd byl doplněn o novou entitu *PublicationType*, která umožňuje zavést libovolný typ publikace. Výsledný diagram perzistentních tříd reflektující změny popsané v této kapitole je zobrazen na obrázku 7.1.

7.6 Export

Zápis XML dokumentu je prováděn pomocí třídy *XMLStreamWriter*. Metadata jsou zapisována tak, aby byla ve formátu pro správnou funkcionalitu všech operací komponenty searchService. Jedná se o speciální reprezentaci prázdné hodnoty polí určených pro fasetové vyhledávání, záměnu mezer na znak _, přidání identifikátoru ke jménům autorů, určení skupiny, do které náleží první písmeno příjmení autora) a naplnění polí s kategoriemi oblasti výzkumu.



Obrázek 7.1: Diagram perzistentních tříd reflektující změny provedené při implementaci

Kapitola 8

Shromáždění ukázkových dat a vyhodnocení použitelnosti

8.1 Příprava ukázkových dat

Původní záměr bylo importovat část databáze vědeckého portálu vyvíjeného v rámci projektu ReResearch. Do dokončení této diplomové práce nebyla databáze ještě naplněna, bylo tedy nutné navrhnout náhradní řešení. Jako ukázková data byla použita podmnožina článků extrahovaných z citační databáze Citeseerx, které byly dostupné na serveru minerva1 (jedná se o jeden z počítačů Výzkumné skupiny pro zpracování přirozeného jazyka). Problémem ovšem je, že články ve svých metadatech obsahují jen omezený počet atributů. K dispozici je pouze identifikátor, název, rok vydání, abstrakt, vydavatel, URI původního PDF dokumentu, jména autorů, citace. Typ publikace je definován pouze jeden a to „text“. Metadata také nejsou nijak dále strukturována, např. autoři jsou specifikováni pouze svými jmény, a tak většina údajů uložených v databázi zůstane po importu nedefinována. Jako potíž se ukázalo také to, že podmnožina článků byla z původní množiny vybrána principielně náhodně. Pro uchování veškerého provázání článků citacemi by bylo potřeba vybrat podmnožinu tvořící shluk. To však by však přesahovalo téma a rozsah této diplomové práce a vyžadovalo by to více času na vypracování, než bylo k dispozici.

Články byly rozděleny do více souborů: metadata jsou uložena v XML dokumentech se syntaxí Dublin Core. Text článků byl uložen v textových souborech, pro každý článek jeden soubor. S metadaty byly texty propojeny pomocí mapovacího souboru, který mapoval URI publikace na jméno souboru (pouze pro některé publikace v metadatech). Jména souborů byly automaticky generované hash hodnoty.

Převod článků do tvaru vhodného pro import byl proveden tak, že nejdříve bylo ručně zkopírováno 10000 souborů s textem článků. Dále byl vytvořen program pracující z příkazové řádky, který k těmto 10000 souborům najde příslušná metadata. Program je implementován třídou *Citeseerx* v balíku *batch.app*. V programu jsou nejdříve načteno mapování URI na hash 10000 vybraných souborů. V dalším kroku je toto mapování invertováno. Poté jsou parsovány XML dokumenty s metadaty. U každé popisované publikace je ověřeno, zda se její URI nenachází v načteném mapování. Pokud ne, je záznam ignorován. V opačném případě je do výstupního XML dokumentu pro import zapsán nový záznam pro tuto publikaci, do atributu plainText je uložena hash hodnota jména souboru s textem publikace.

Poté byl proveden dříve popsaný postup – vytvořený XML dokument byl importován do databáze, poté byl obsah databáze exportován a na základě výsledného XML dokumentu

počet dokumentů	10 000
celková velikost prostého textu	657 036 KB
průměrná velikost indexovaného dokumentu	65,7 KB
velikost XML dokumentu pro import metadat	14 741 KB
velikost metadat uložených v MySQL databázi	15 469 KB
velikost optimalizovaného Solr indexu	750 372 KB
počet termů v indexu	910 721

Tabulka 8.1: Vlasnosti ukázkových dat

import metadat do databáze	58.406 sekund
export metadat z databáze	3 minuty 29.534 sekund

Tabulka 8.2: Rychlost zpracování ukázkových dat

byl aktualizován index.

8.1.1 Rychlost zpracování

Měření rychlosti zpracování bylo provedeno na počítači s procesorem Intel Core Duo 2,26 Ghz 3MB L2 Cache, RAM 4GB 1067 Mhz, pevný disk 5400 ot/min., rozhraní SATA, souborový systém ext3, operační systém GNU/Linux, distribuce OpenSuse 11.0, Java verze 1.6.0_19 64bit.

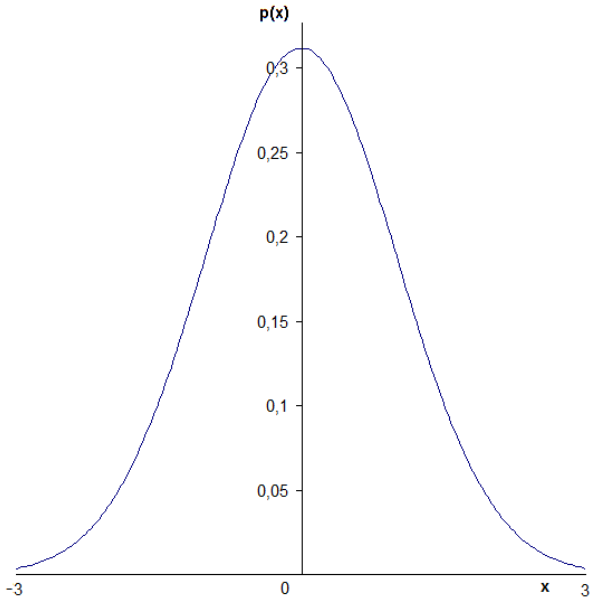
8.1.2 Generátor náhodných metadat

Pro potřeby vývoje a průběžného testování vytvořeného vyhledávače byl vytvořen generátor testovacích dat. Tento generátor vytváří náhodná metadata k podmnožině článků anglické verze Wikipedie. Byla použita pouze podmnožina, aby bylo možné rychle generovat testovací data odpovídající aktuálním potřebám.

Z Wikipedie jsou převzaty texty článků, jejich nadpisy, identifikátory a URI. Pro ostatní atributy jsou hodnoty vybírány náhodně. Pro každý atribut se nabízí možnost zvolit soubor se zdrojovou kolekcí náhodných hodnot, maximální počet vybraných hodnot, zda může být hodnota nedefinovaná a rozložení pravděpodobnosti pro generování náhodných hodnot. Různá rozložení pravděpodobnosti jsou použita proto, aby bylo ověřeno, že rozhraní faset a dalších prvků uživatelského rozhraní je použitelné pro různé distribuované hodnoty. Na výběr je mezi uniformním a normálním rozložením. U normálního rozložení jsou hodnoty generovány v intervalu -3 až 3. Hodnoty se pak transformují na celočíselné hodnoty v požadovaném intervalu.

8.2 Vyhodnocení použitelnosti

Uživatelské rozhraní vyhledávače kombinuje prvky a funkce různých existujících vyhledávačů a snaží se převzít ty, které jsou efektivní a ověřené, a zároveň používá prvky a funkce, které rozšiřují obvyklé možnosti existujících vyhledávačů. Základní myšlenkou je poskytnout uživateli více cest k nalezení požadované informace a současně zachovat uživatelské rozhraní přehledné a intuitivní.



Obrázek 8.1: Gaussova křivka normálního rozložení použitého v generátoru náhodných hodnot

Těžištěm je vyhledávání postupným filtrováním výsledků podle různých kritérií odvozených z vlastností publikací. K tomu jsou poskytovány funkce popsané ve druhé kapitole – fulltextové vyhledávání, vyhledávání na základě klasifikace publikací (taxonomie, folksonomie), na základě vícedimenzionální fasetové klasifikace kombinované s hierarchickou klasifikací a také je k dispozici funkce citační databáze.

Pro implementaci byly použity platformy Java EE a Solr. Obě tyto platformy poskytují prostředky k dosažení vysoké škálovatelnosti. Při velkém počtu dat a dotazů je možné vyvinutou aplikaci transparentně distribuovat na více strojů.

8.2.1 Přehled vyhledávacích funkcí

Pro fulltextové vyhledávání lze použít následující atributy publikace: URI, typ, jazyk, text publikace, název, abstrakt, klíčová slova, popisky, rok vydání, jména autorů, jméno vydavatele, identifikátor a předmět oblasti výzkumu, jméno projektu, citace a jméno konference. Tedy všechny vlastnosti publikace, které lze využít při fasetovém vyhledávání, lze zadat i přímo ve fulltextovém dotazu. Fasetové vyhledávání slouží pro hledání podle nepřesně specifikovaných vyhledávacích kritérií, kdy uživatel až postupným procházením množiny nalezených publikací a jejím filtrováním tato kritéria upřesňuje. Naproti tomu atributy zadané ve fulltextovém dotazu jsou uživatelem použity v případě, že uživatel má přesnou představu o tom, co hledá.

Pro fasetové vyhledávání lze použít hierarchické fasety:

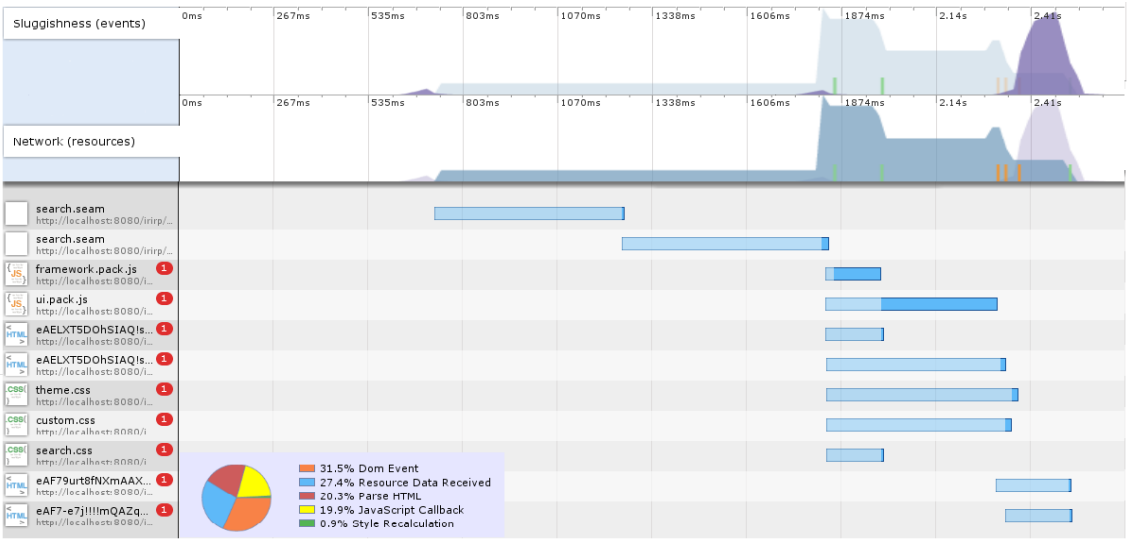
- oblast výzkumu (taxonomie podle ACM CCS)
- autor (explicitní hierarchie podle příjmení autora)
- rok vydání (hierarchie podle desetiletí)

- popisky (folksonomie, hierarchie tvoří uživatelskou navigaci ve formě přidávaných popisků) Dále lze pro fasetové vyhledávání využít jednoúrovňové fasety pro vydavatele, typ publikace a názvů konferencí.

8.2.2 Rychlost

Měření odezev vyhledávače bylo provedeno pomocí pluginu SpeedTracer pro prohlížeč Google Chrome. První měření bylo provedeno na fulltextovém dotazu *+information +retrieval +speed*. Nalezeno bylo 734 výsledků. Na obrázku 8.2 jsou znázorněna odezva aplikace. Graf označený štítkem Network(resources) zobrazuje počet přenášených dat mezi webovým serverem a prohlížečem v čase, graf označený štítkem Sluggishness (events) zpracování stránky a dalších souborů v prohlížeči. Z grafu odečtená doba odezvy na daný dotaz je zhruba 1,7 sekundy. Z grafu lze také vyčíst, že vlastní vyhledávání, renderování html stránky search.seam a její přenos tvoří pouze přibližně polovinu času odezvy, zbytek je přenos dalších souborů potřebných pro funkci frameworku JSF, případně přídavných stylových předpisů.

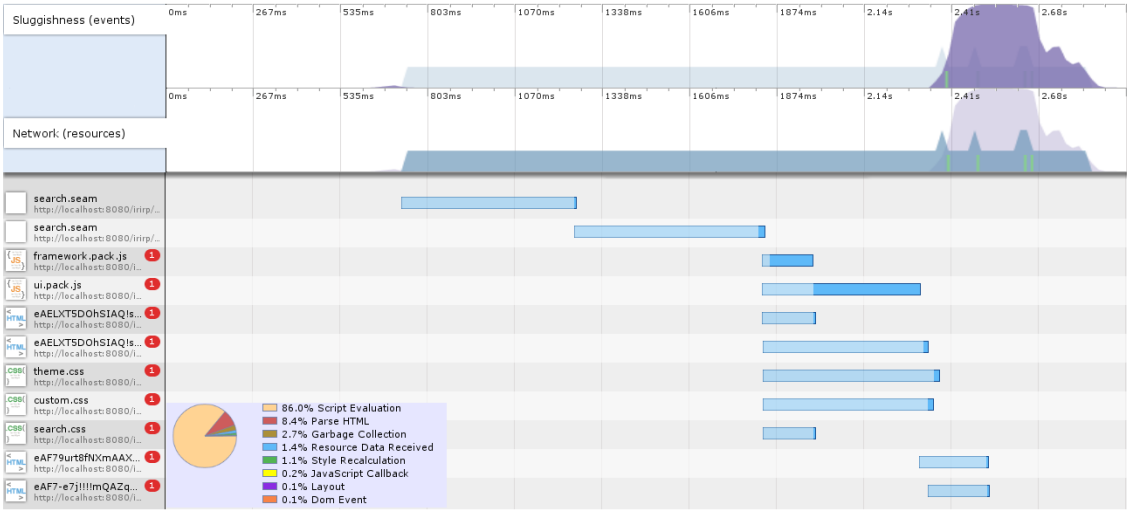
Druhé měření bylo provedena na fulltextovém dotazu *+information +retrieval +speed*, s omezením hierarchické fasety *Author* na hodnotu *DEF*. Nalezeno bylo 280 publikací, faseta *Author* měla na nižší úrovni hierarchie 268 hodnot. Na obrázku 8.3 je uvedena odezva aplikace na druhý dotaz. Odezva je přibližně 2 sekundy. Lze usoudit, že přenos většího množství hodnot fasety Author a jejich filtrování, aby vyhovovaly skupině na vyšší úrovni hierarchie (podrobnosti viz v kapitole Implementace) nezpůsobuje výrazné zpomalení. Měření bylo provedeno také na řádově větších kolekcích výsledků a časy byly srovnatelné.



Obrázek 8.2: Doba odezvy na fulltextový dotaz

8.2.3 Ukázkové dotazy

Jako poslední jsou k tématu použitelnost uvedeny ukázkové dotazy pro demonstraci možností vyhledávacího rozhraní. Kromě prvního kroku, tvořeného fulltextovým dotazem, je pořadí vykonávání jednotlivých bodů libovolné. Postupy jsou jen orientační, ve skutečnosti



Obrázek 8.3: Doba odezvy na fulltextový dotaz kombinovaný z fasetovým omezením

se uživatel při hledání s využitím faset rozhoduje o dalším kroku až podle aktuálně zobrazených výsledků.

Dotaz 1

Zadání: najdi publikace napsané Janem Svobodou.

1. fulltextový dotaz *author:(Jan Svoboda)*

Dotaz 2

Zadání: najdi nejcitovanější příspěvky na konferenci pojednávajících o rychlosti ve vyhledávání informací, uvažuj konference v devadesátých letech.

1. fulltextový dotaz *Information Retrieval speed*, řazení podle citací
2. faseta *Year*, hodnota *1990-1999*
3. faseta *Type*, hodnota *Proceedings*

Dotaz 3

Zadání: najdi články některého známého autora, které se zabývají pravděpodobností v teorii front.

1. fulltextový dotaz *Probability*
2. facesta *Topic*, hodnota *Mathematics of Computing*
3. faseta *Topic*, hodnota *Probability and Statistics*
4. faseta *Topic*, hodnota *Queue Theory*
5. faseta *Author*, výběr jméno některého známého člověka. Při více autorech procházení po skupinách

Dotaz 4

Zadání: najdi publikace citující některou knihu z nakladatelství Adison, která se zabývá serverem Solr.

1. fulltextový dotaz *title:Solr* nebo jen dotaz *Solr*
2. faseta *Type*, hodnota *Book*
3. faseta *Publisher*, hodnota *Adison*
4. vylistování nalezených publikací, při nalezení požadované publikace stisknutí tlačíka *Cited by*

Dotaz 5

Zadání: najdi publikace zabývající se rychlostí v oblasti Information Retrieval, které jsou nejvíce populární u ostatních uživatelů a o kterých ostatní uživatelé tvrdí, že se zabývají Open Source softwarem.

1. fulltextový dotaz *information retrieval speed*
2. seznam popisků (tag cloud), výběr popisku s největší velikostí písma
3. seznam popisků (tag cloud), nalezení popisku obsahujícího slovo speed

Dotaz 6

Zadání: najdi příspěvky na konference, které se konaly v roce 2009 a které byly věnována robotice.

1. fulltextový dotaz *robot*
2. faseta *Type*, hodnota *proceedings*
3. fasta *Year*, hodnota *2009*, případně nejdříve skupina *2000-2009*
4. faseta *Conference*, výběr vyhovující konference

Kapitola 9

Závěr

Cílem této práce je implementace rozhraní pro vyhledávání ve vědeckých publikacích, přičemž toto rozhraní je koncipováno jako jedna ze součástí vědeckého webového portálu. Byla navržena struktura metadat popisujících vědecké publikace a dále bylo navrženo a implementováno uživatelské rozhraní vyhledávače.

Informace poskytované vědeckým webovým portálem jsou reprezentovány bází znalostí, jejíž součásti jsou také údaje o vědeckých publikacích. Pro účely vyhledávání jsou vybrané údaje o publikacích uloženy v reverzním indexu, který lze chápat jako pohled na bázi znalostí s možností rychlého vyhledávání. Navrhnutá struktura metadat tedy zároveň tvoří strukturu reverzního indexu. Při návrhu byla brána na zřetel možnost kombinace různého způsobu vyhledávání (fulltextové, fasetové), výkonnostní hledisko a způsob propojení reverzního indexu s bází znalostí.

Uživatelské rozhraní je navrženo tak, aby bylo připraveno na integraci s ostatními částmi vědeckého webového portálu. Návrh je proveden tak, aby vyhledávač poskytoval prostředky pro inteaktivní filtrování výsledků, zobrazoval na požádání dodatečné informace o nalezených publikacích a současně zůstal jednoduchý a intuitivně ovladatelný. Použitelnost vytvořeného vyhledávače byla otestována na testovací sadě článků, která byla převzata z citační databáze Citeseerx.

I když je vyhledávač implementován jako samostatná aplikace a nabízí prostředky pro automatizovaný import a správu metadat, v dalším vývoji by měla následovat integrace s existujícím vědeckým webovým portálem. Z dalších možných rozšíření funkcionality vyhledávače se nabízí implementace zbylých navrhovaných prvků uživatelského rozhraní, kterými jsou grafické fasety a zkrácený výpis výsledků. Další rozšíření uživatelského rozhraní, které nebylo zahrnuto v návrhu, by mohlo být našeptávání během zapisování fulltextových dotazů. U tohoto prvku je potřeba vyhodnotit, jaké informace má zobrazovat – buďto položky z indexu s daným prefixem (funkce podporována serverem Solr) nebo zda má uživateli nabízet historii dříve zadaných dotazů. Ve druhém případě by bylo nutné historii reprezentovat vlastním formátem, např. s využitím datové struktury trie. Návrh metadat byl proveden také s ohledem na možnost vyhledávání ve více jazycích. Prostor pro další rozvoj se tedy otevírá v oblasti implementace Lucene analyzátorů pro další jazyky. Jako další možné rozšíření uveďme ještě podporu importu metadat ve standardu Dublin Core.

Při řešení zadaného tématu si autor prohloubil znalosti problematiky vyhledávání a klasifikace informací, seznámil se způsoby jejich elektronické reprezentace a nástroji pro jejich vyhledávání. Při implementaci autor dále získal zkušenosti s vývojem v prostředí platformy Java Enterprise Edition a naučil se pracovat s frameworky Seam a Hibernate.

Literatura

[1] *Computer Science and Communications Dictionary*. Springer US, 2001, ISBN 978-1-4020-0613-5.

[2] Definition of Classifying at Dictionary.com [online]. `http://dictionary.reference.com/browse/classifying`, [cit. 2009-12-09].

[3] Faceted Search with Solr [online]. `http://www.lucidimagination.com/Community/Hear-from-the-Experts/Articles/Faceted-Search-Solr`, [cit. 2009-12-09].

[4] Codes for the Representation of Names of Languages [online]. `http://www.loc.gov/standards/iso639-2/php/code_list.php`, [cit. 2009-12-21].

[5] Allen, D.: *Seam in Action*. Manning, 2009, ISBN 1933988401.

[6] Bartošek, M.: Vyhledávání v Internetu a DUBLIN CORE. *Zpravodaj ÚVT MU*, ročník 9, č. 4, 1999: s. 1–4, ISSN 1212-0901.

[7] Bauer, C.; King, G.: *Java Persistence with Hibernate*. Manning, 2007, ISBN 1-932394-88-5.

[8] Bye, K.: Tag Cloud Font Distribution Algorithm [online]. `http://www.echochamberproject.com/node/247`, [cit. 2010-05-20].

[9] Cathro, W.: Metadata: an overview. In *Matching Discovery and Recovery Seminar*, 1997.

[10] Denon, W.: How to Make a Faceted Classification and Put It On the Web [online]. `http://www.miskatonic.org/library/facet-web-howto.html`, 2009-03-28 [cit. 2009-12-09].

[11] Gruber, T. R.: A Translation Approach to Portable Ontology Specifications. Technická Zpráva 5, Knowledge Acquisition, 1993.

[12] Guariono, N.; Giaretta, P.: *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, kapitola Ontologies and Knowledge Bases: Towards a Terminological Clarification. Towards Very Large Knowledge Base. Amsterodam: IOS Press, 1995.

[13] Hostetter, C.: Faceted searching with Apache Solr. In *Apachecon US 2006*, 2006.

[14] Huynh, D. F.; Karger, D. R.; Miller, R. C.: Exhibit: Lightweight Structured Data Publishing. In *Proceedings of the 16th international conference on World Wide Web*, May 2007.

[15] Jendrock, E.; Ball, J.; Carson, D.; aj.: The Java EE 5 Tutorial [online]. http://java.sun.com/javaee/5/docs/tutorial/doc/, [cit. 2010-05-11].

[16] Kubásek, L.: Představení aplikačního frameworku JBoss Seam [online]. http://www.kubasek.cz/blog/2008/02/17/predstaveni-aplikacniho-frameworku-jboss-seam/, [cit. 2010-05-11].

[17] Manning, C. D.; Raghavan, P.; Schütze, H.: *Information Retrieval*. Cambridge University Press, 2009.

[18] Mesbah, A.; van Deursen, A.: A component- and push-based architectural style for ajax applications. *Journal of Systems and Software*, ročník 81, č. 12, 2008: s. 2194–2209.

[19] Mirkin, B.; Nascimento, S.; Pereira, L. M.: Representing a Computer Science Research Organization on the ACM Computing Classification System. In *Supplementary Proceedings of the 16th International Conference on Conceptual Structures (ICCS-2008), CEUR*, 2008.

[20] Nithianandan, A.: Solr and RDBMS: The basics of designing your application for the best of both [online]. http://www.lucidimagination.com/Community/Hear-from-the-Experts/Articles/Solr-and-RDBMS-design-basics, [cit. 2009-12-18].

[21] Nováček, V.; Smrž, P.; Pomikálek, J.: Text Mining for Semantic Relations as a Support Base of a Scientific Portal Generator. In *Proceedings of the 5th International Conference on Language Resources and Evaluation*, European Language Resources Association, 2006, ISBN 2-9517408-2-4, s. 1338–1343.

[22] SeamFramework.org: Introduction to JBoss Seam [online]. http://docs.jboss.com/seam/latest/reference/en-US/html/Book-Preface.html, [cit. 2010-05-11].

[23] Smiley, D.; Pugh, E.: *Solr 1.4 Enterprise Search Server*. 2009, ISBN 1847195881.

[24] Svátek, V.: Ontologie a WWW. In *Sborník konference Datakon*, Brno, 2002, str. 6.

[25] The Apache Software Foundation: Apache Lucene: Overview [online]. http://lucene.apache.org/java/docs/, [cit. 2009-12-11].

[26] The Apache Software Foundation: SchemaXml [online]. http://wiki.apache.org/solr/SchemaXml, [cit. 2009-12-13].

[27] Vaishar, A.: Folksonomie. *Inflow: information journal*, ročník 1, č. 1, 2008, ISSN 1802-9736.

[28] Voss, J.: Collaborative thesaurus tagging the Wikipedia way. Technická zpráva, Wikimedia Deutschland e.V, 2006.

Dodatek A

Uživatelská příručka

A.1 Webová aplikace

V horní části všech stránek webové aplikace se nachází lišta s menu. Na pravém konci této lišty se nachází tlačítko pro přihlášení. Při přihlašování uživatel musí zadat přihlašovací jméno a heslo. V levé části lišty se nachází položky menu:

- *Home* – úvodní stránka aplikace
- *Search* – vyhledávací rozhraní
- *User* – operace nad účtem uživatele
- *Administration* – rozhraní pro správu databáze

A.1.1 Vyhledávací rozhraní

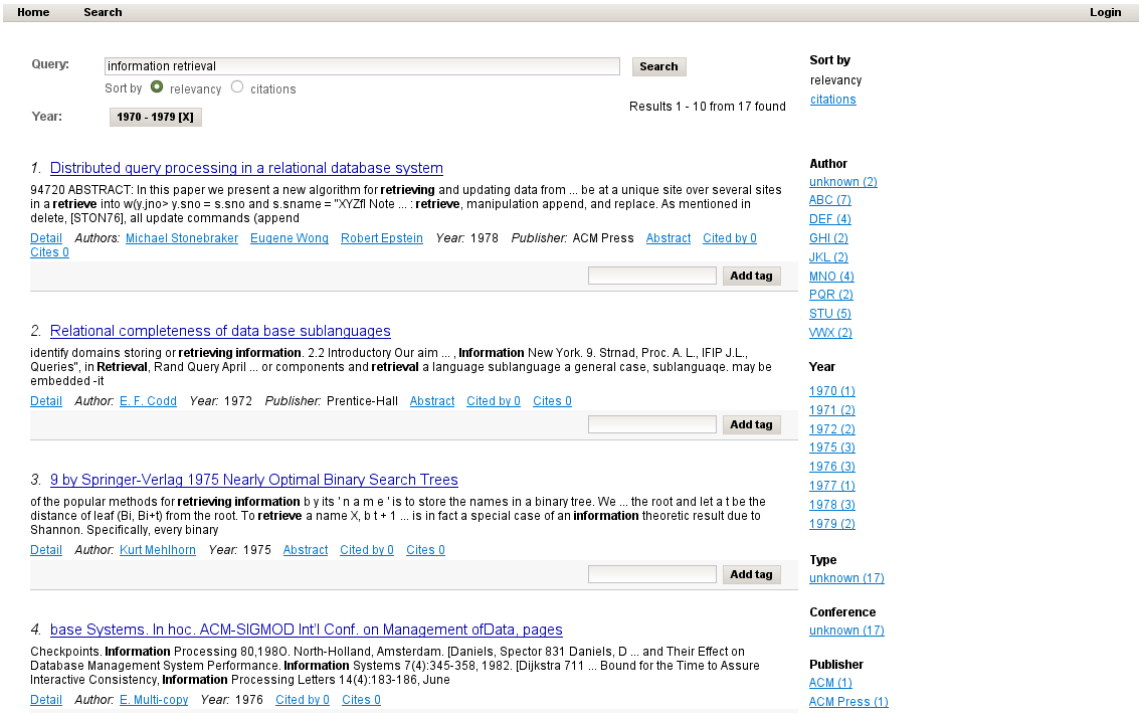
Stránka pro vyhledávání je zobrazena na obrázku A.1. Lze rozdělit na následující oblasti:

- oblast pro zadání fulltextového dotazu
- oblast pro zobrazení dalších vybraných podmínek dotazu (fasetová omezení aj.)
- oblast pro zobrazení výsledků vyhledávání
- oblast pro zobrazení faset

Oblast pro zadání fulltextového dotazu

V této části stránky se nachází pole pro zadání fulltextového dotazu. Zároveň je možné vybrat způsob řazení nalezených publikací podle relevance k zadanému dotazu nebo podle počtu citací. Stisknutím tlačítka Search je provedeno vlastní vyhledávání. Stisk tohoto tlačítka vytvoří nový dotaz a zruší všechny ostatní podmínky původního dotazu (fasetová omezení, klíčová slova, popisky).

Pro fulltextové dotazy je používána implicitní Solr syntaxe. Názvy polí se od hodnoty oddělují dvojtečkou (např. dotaz *type:(article book)* nalezne publikace typu článek nebo kniha).



Obrázek A.1: Celkový pohled na vyhledávací rozhraní

Oblast pro zobrazení podmínek dotazu

Tato oblast se nachází pod polem pro zadání fulltextového dotazu. Jsou zde postupně pod sebou vypisovány další zadané podmínky dotazu – což mohou být fasetová omezení, klíčová slova nebo popisky.

V každém řádku je vlevo je zobrazen název podmínky resp. název fasety a pak vybraná omezení dané podmínky ve formě tlačítek. Kliknutím na některé z tlačítek je dané omezení zrušeno. Celkově všechny podmínky dohromady tvoří průnik (tzn. každý nalezený dokument vyhovuje všem zde zobrazeným podmínkám).

Oblast pro zobrazení výsledků vyhledávání

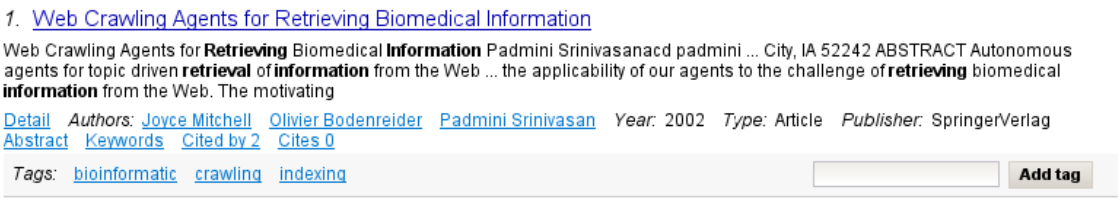
Výsledky vyhledávání jsou zobrazovány pod oblastí pro zobrazování podmínek dotazu. U každé zobrazené publikace je uvedeny tyto údaje (obrázek A.2):

- název, který slouží zároveň jako odkaz na vlastní dokument publikace
- KWIC – tři výseky z textu se zvýrazněnými slovy fulltextového dotazu
- tlačítko pro zobrazení detailu s dalšími informacemi o publikaci
- jména autorů, která po kliknutí taktéž zobrazí detail dané osoby
- rok vydání
- jméno vydavatele

uri	URI
type	typ
language	jazyk
text	text publikace (implicitní vyhledávací pole)
title	název
abstract	abstrakt
keyword	klíčové slovo
tag	popisek
year	rok vydání
author	jméno autora
publisher	jméno vydavatele
topic	identifikátor nebo předmět oblasti výzkumu
project	jméno projektu
cites	publikace citované zadanou publikací
citedby	publikace citující zadanou publikaci
conference	jméno konference

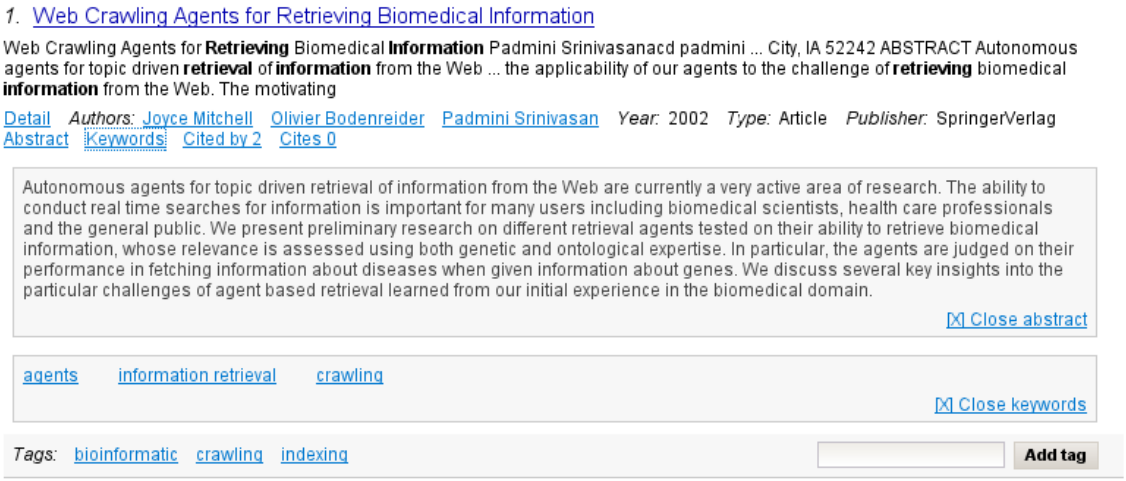
Tabulka A.1: Atributy fulltextového vyhledávání

- tlačítko pro zobrazení abstraktu. Abstrakt je zobrazen ve vysunovacím panel (viz obrázek A.3).
- tlačítko pro zobrazení klíčových slov. Seznam klíčových slov je zobrazen ve vysunovacím panelu (obrázek A.3). Po kliknutí na některé z klíčových slov je toto slovo přidáno mezi podmínky dotazu.
- počet publikací, jež citují tuto publikaci. Po kliknutí jsou tyto publikace vyhledány a zobrazeny jako výsledky nového dotazu
- počet citovaných publikací. Po kliknutí jsou tyto publikace vyhledány a zobrazeny jako výsledky nového dotazu.
- Lišta se seznamem přiřazených popisků. Po kliknutí na některý z popisků je tento přidán do podmínek dotazu. V pravé části lišty je umístěno pole pro zadání nového popisku a tlačítko pro jeho přidání.



Obrázek A.2: Výsledek vyhledávání

V detailu publikace jsou zobrazeny další informace o vybrané publikaci. Nachází se zde také formulář pro možnost ohlášení nepřesných informací (obrázek A.4).



Obrázek A.3: Výsledek vyhledávání s otevřenými panely abstrakt a klíčová slova

V detailu autora jsou zobrazeny podrobné informace o osobě autora (obrázek A.5).

Na stránce je zobrazeno nejvýše 10 výsledků. Ve spodní části stránky se nachází lišta pro procházení všech stránek s výsledky vyhledávání (obrázek A.6).

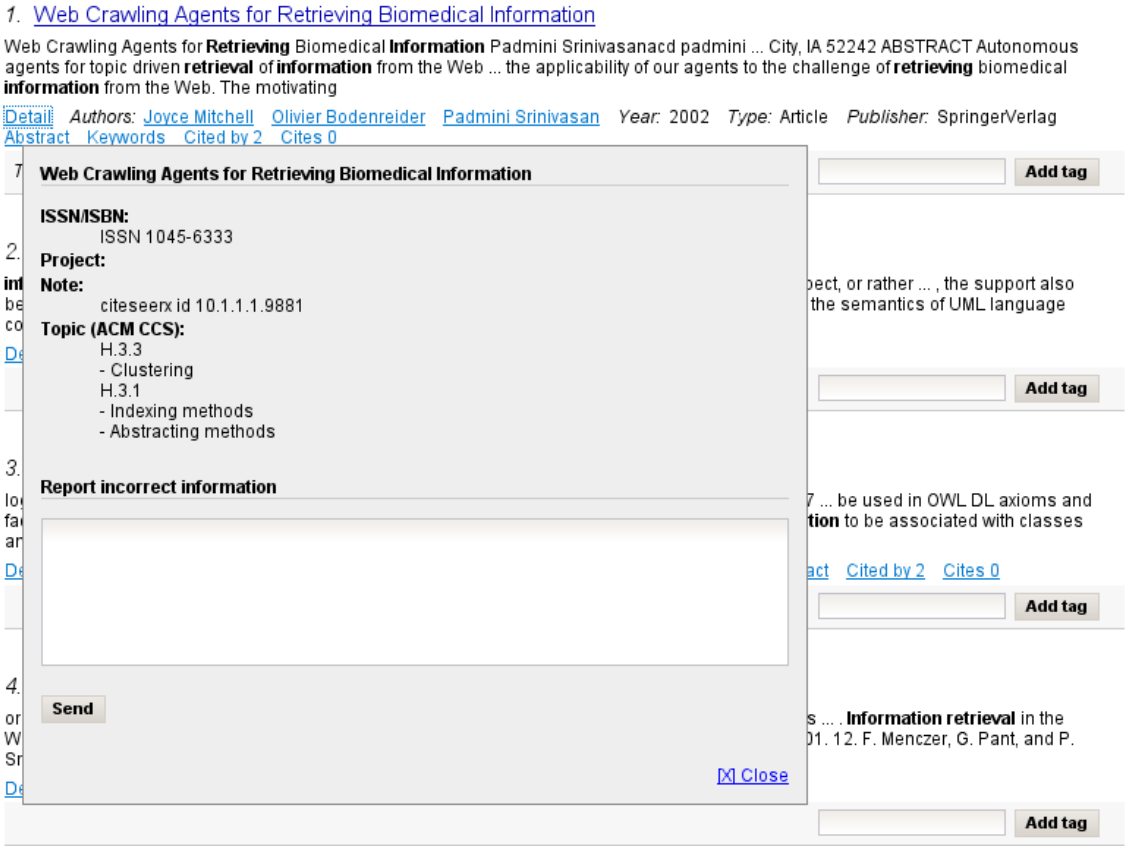
Oblast pro zobrazení faset

Tato oblast se nachází v pravé části stránky. V horní části je umístěn přepínač řazení a odkaz pro zobrazení okna se seznamem popisků (obrázek A.7). Dále jsou zde zobrazeny hodnoty faset autor, oblast výzkumu, typ publikace, vydavatel a konference. Při výběru některé z hodnot fasety je aktualizován seznam nalezených publikací a vybrané omezení je zobrazeno v seznamu podmínek dotazu. U hierarchických faset jsou zobrazeny hodnoty z leva doprava od vyšší postupně k nižším úrovním hierarchie.

A.1.2 Oblast pro zobrazení faset

Formuláře pro správu databáze se nachází v menu *Administration*. K dispozici jsou formuláře pro správu:

- organizací
- projektů
- publikací
- typů publikací
- vydavatelů
- klíčových slov
- popisků
- upozornění na chybné údaje



Obrázek A.4: Detail publikace

- osob
- uživatelů

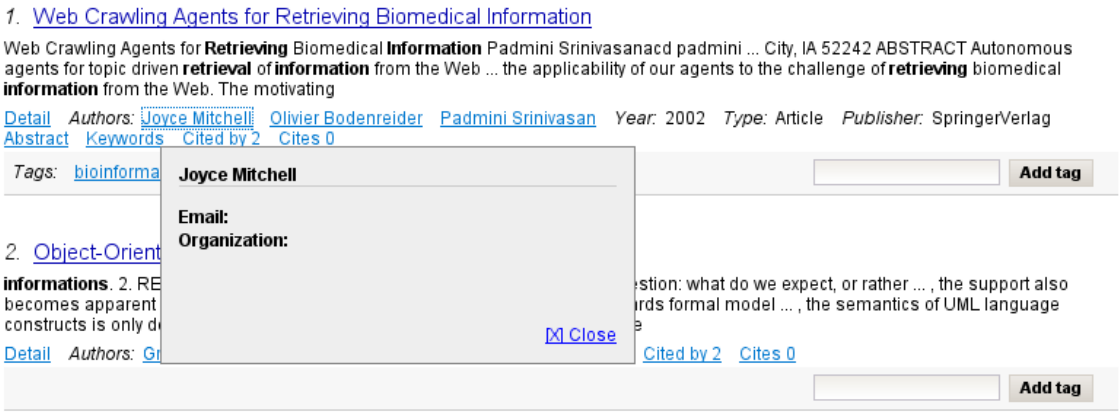
U každého formuláře je nejdříve zobrazen seznam všech výskytů s možností filtrování a stránkování (obrázek A.10).

Vlastní formuláře umožňují všech hodnot dané entity. Textové hodnoty jsou zadávány přímo, pro hodnoty, které ukazují na jinou entitu, slouží pro zadání hodnoty seznamy možných hodnot. V případě klíčových slov a popisků, je možné zadat přímo nový název.

A.2 Dávkové úlohy

Programy pro provádění dávkových úloh využívají konfigurační soubory, ve kterých jsou vlastnosti definovány ve tvaru *názevVlasnosti=hodnota*, znakem # jsou uvozeny komentáře. Například soubor s konfigurací připojení k databázovému serveru je v následujícím formátu:

```
# Database connection
url=jdbc:mysql:///irirp
user=root
passwd=
```



Obrázek A.5: Detail autora

A.2.1 Import

Formát XML dokumentu pro import do databáze je přiložen ke zdrojovým kódům aplikace. Program pro import se spustí příkazem

```
java cz.vutbr.fit.irirp.batch.Import xmlFile --config configFile [--delete]
```

kde `xmlFile` je cesta k importovaném souboru a v parametru `--config` je specifikována cesta k souboru s konfigurací databázového připojení volitelný parametr `--delete` před vlastním importem smaže všechna data v databázi. Soubor s konfigurací databázového připojení obsahuje vlastnosti *url*, *user* a *passwd*.

A.2.2 Export

Program pro export se spustí příkazem

```
java cz.vutbr.fit.irirp.batch.ExportIndex outputFile --config configFile
```

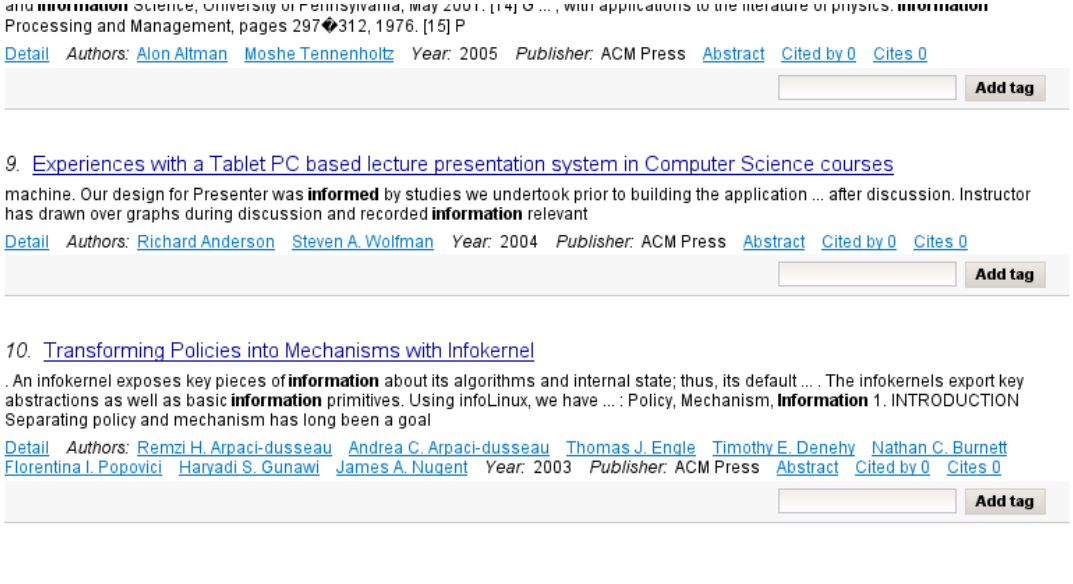
kde `outputFile` je název výstupního souboru a parametr `--config` specifikuje cestu k souboru s konfigurací. Tento soubor obsahuje vlastnosti pro připojení k databázovému serveru (*url*, *user*, *passwd*) a dále vlastnost *path*, ve které je zadána cesta k adresáři s texty publikací, a vlastnost *acm*, která obsahuje cestu XML dokumentu s ACM CCS hierarchií.

A.2.3 Generátor náhodných metadat

Program pro generování náhodných metadat se spustí příkazem

```
java cz.vutbr.fit.irirp.batch.app.TestGenerator confFile
```

kde `confFile` je cesta k souboru s konfigurací. Soubor obsahuje vlastnosti pro zadání vstupních a výstupních souborů a dále nastavení způsobu generování jednotlivých atributů metadat. Vlastnosti se skládají z názvu entity, jména atributu a jména nastavení, které může být pro jednohodnotové atributy:



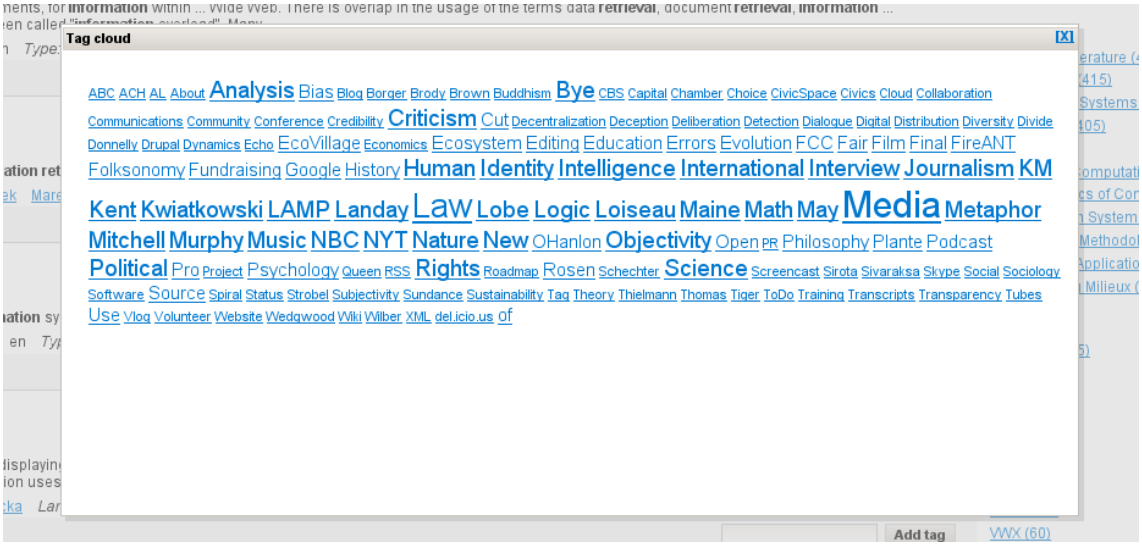
Obrázek A.6: Lišta pro procházení výsledků po stránkách

- *allowNull* – pokud je true, je genenerována také prázdná hodnota
- *gaussian* – pokud je true, bude použito normální rozložení namísto uniformního

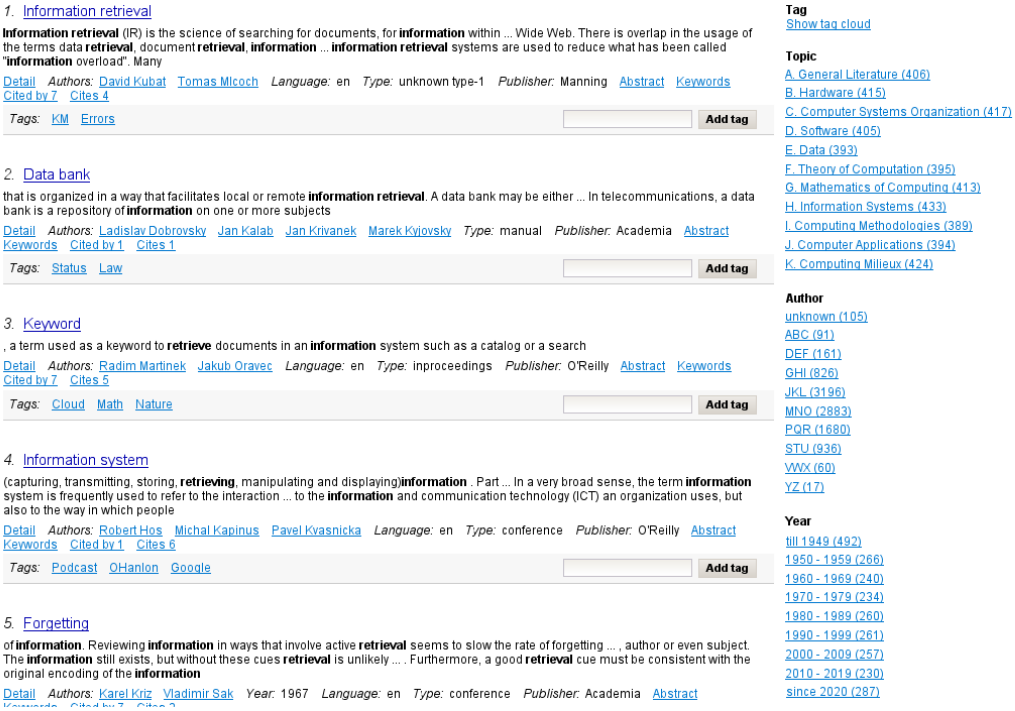
a pro vícehodnotové atributy:

- *max* – maximální počet vygenerovaných hodnot
- *itemGaussian* – pokud je true, hodnoty jsou generovány s normálním rozoložením namísto uniformního
- *sizeGaussian* – pokud je true, počet hodnot je generován s normálním rozložením namísto uniformního

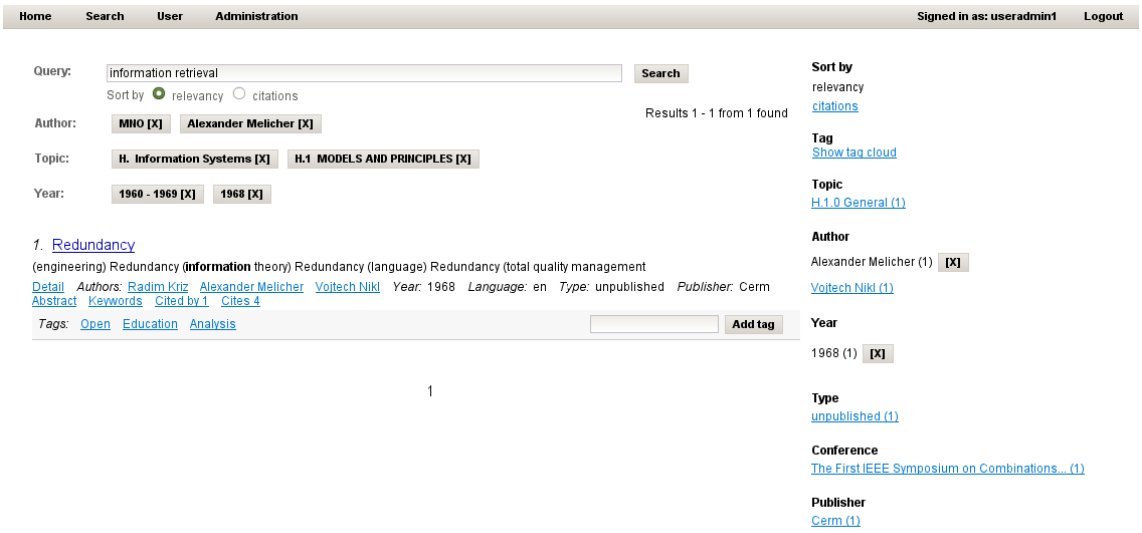
Např. *publication.type.gaussian=false* znamená, že atribut pro typ publikace bude generován s normálním rozložením. Ukázkový soubor s výčtem všech vlastností je přiložen ke zdrojovým kódům.



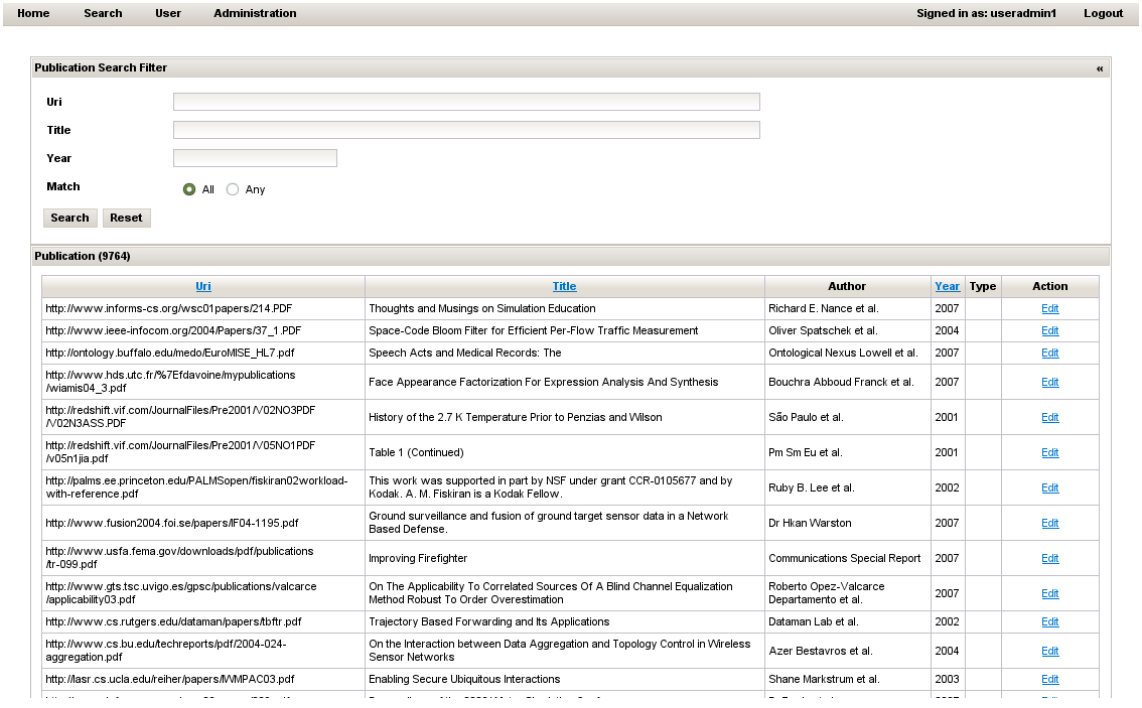
Obrázek A.7: Seznam popisků



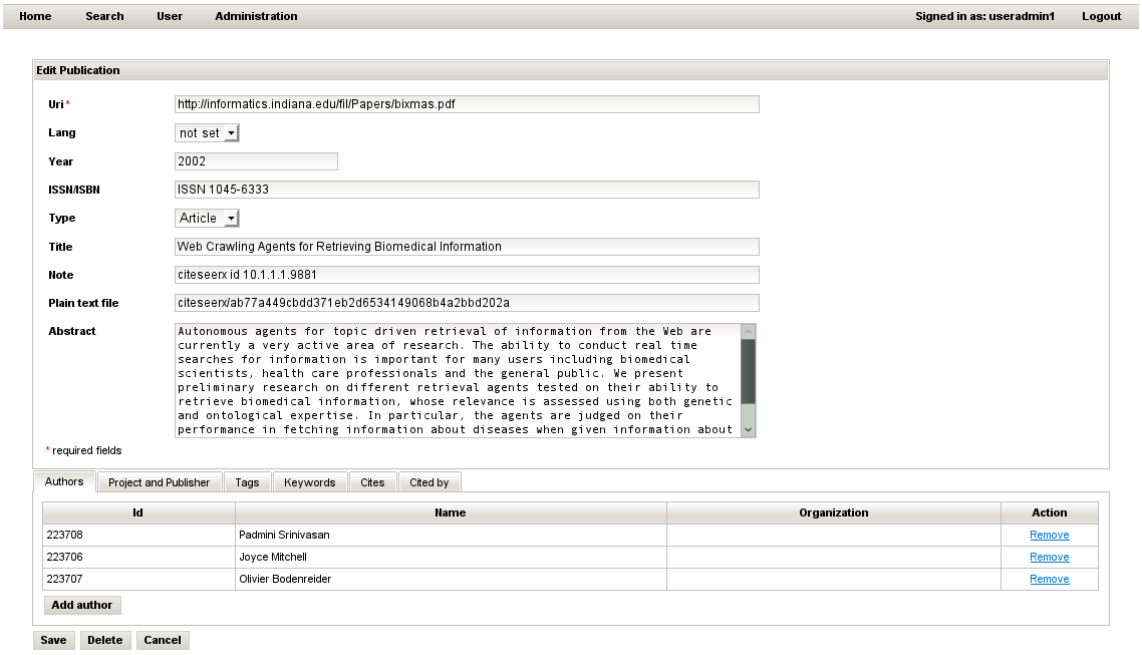
Obrázek A.8: Ukázka hodnot na nejvyšší úrovni strukturovaných faset



Obrázek A.9: Ukázka vybraných omezení strukturových faset



Obrázek A.10: Administrátorské rozhraní



Obrázek A.11: Administrátorské rozhraní