УНИВЕРЗИТЕТ У НОВОМ САДУ
ПРИРОДНО-МАТЕМАТИЧКИ ФАКУЛТЕТ
ДЕПАРТМАН ЗА МАТЕМАТИКУ И
ИНФОРМАТИКУ

Данијела Боберић

Софтверски систем за преузимање библиографских записа

- докторска дисертација -

Нови Сад, 2010.

Предговор

Предмет истраживања приказаних у овој докторској дисертацији припада области пројектовања информационих система, односно пројектовању и имплементацији дистрибуираних библиотечких информационих система. Извршено је моделирање и имплементација система који омогућава претраживање и преузимање библиотечких записа по дефинисаним стандардима. Систем је базиран на сервис - оријентисаној архитектури и *mediator/wrapper* шаблонима. Систем је имплементиран у програмском језику *Java*, а модел је приказан у *UML* 2.0 нотацији. У оквиру система развијени су сервиси који представљају серверске стране за протокол Z39.50 и SRU и развијена је посебна софтверска компонента која омогућава интеграцију тих сервиса са постојећим библиотечким системом. Верификација овог система извршена је интеграцијом у софтверски систем БИСИС верзије 4.

Дисертација садржи следећа поглавља:

1. Увод,
2. Стандарди за преузимање информација,
3. Библиотечки систем БИСИС,
4. Упитни језици,
5. Архитектура система за посредовање између софтверских компоненти,
6. Имплементација протокола SRU и Z39.50,
7. Закључак.

У **првом** поглављу дефинисани су циљеви дисертације и полазне претпоставке. Приказан је преглед постојећих истраживања из предметне области.

Систем који је реализован у овој дисертацији обезбеђује пре свега интероперабилност са другим системима. Интероперабилност је постигнута употребом одговарајућих стандарда, и у **другом** поглављу су описани стандарди који се користе за претраживање и преузимање података.

Истраживања у овој дисертацији добијена су у оквиру развоја библиотечког софтверског система БИСИС верзије четири, који је

укратко описан у **трећем** поглављу. Детаљније је описан текст сервер система БИСИС који је заснован на програмском пакету Lucene.

У **четвртом** поглављу описани су упитни језици дефинисани Z39.50 и SRU стандардом. Показано је да се упит креиран у Z39.50 упитном језику може трансформисати у упит формиран по SRU упитном језику, који је у овој дисертацији усвојен као општи језик за претраживање базе библиографских података. Такође је дат предлог мапирања концепата дефинисаних SRU упитним језиком у концепте који су дефинисани *Lucene* упитним језиком.

Пето поглавље представља централно поглавље ове дисертације. У њему су разматрани различити примери софтверских архитектура који се могу применити код реализације система за преузимање библиографских записа. Изабран је *mediator/wrapper* шаблон који је послужио за моделирање и имплементацију софтверске компоненте која је посредник између постојећег библиотечког система и сервиса за претраживање и преузимање. Ова софтверска компонента описана је у овом поглављу.

У **шестом** поглављу дат је модел и имплементација серверске стране протокола SRU. Такође је дат и опис имплементације серверске стране протокола Z39.50. Ради комплетности и верификације система приказне су и имплементације клијентских страна ових протокола. На крају је дат и предлог проширења стандарда SRU. Ово проширење се односи на могућност ажурирања библиографских записа у удаљеним базама података преко Интернета. На овај начин би се омогућило креирање само једне серверске стране система за све библиотеке које су чланице одређене заједнице.

На крају је дат **закључак** рада и наведени су могући правци даљег истраживања.

Захваљујем се ментору и члановима комисије који су својим сугестијама и конкретним предлозима допринели да структура дисертације буде прегледнија и да приказани резултати буду јасније истакнути.

Такође се захваљујем својој породици на разумевању и подршци.

Нови Сад, 2010 Данијела Боберић

Садржај

Предговор 3

Садржај 5

Увод 9

1.1 АРХИТЕКТУРА ДИСТРИБУИРАНИХ СИСТЕМА 9
1.2 SOA У БИБЛИОТЕЧКИМ СИСТЕМИМА 12
1.3 БИБЛИОТЕЧКИ СТАНДАРДИ 13
1.4 ПРЕДМЕТ И ЦИЉ ИСТРАЖИВАЊА 16

Стандарди за преузимање информација 19

2.1 Z39.50 СТАНДАРД 19
2.1.1 Z39.50 сервиси 20
2.2 OPENSEARCH ПРОТОКОЛ 25
2.3 OAI-PMH СТАНДАРД 27
2.4 SRU СТАНДАРД 30
2.4.1 SRU сервиси 31
2.4.1.1 SearchRetrieve сервис 31
2.4.1.2 Scan сервис 33
2.4.1.3 Explain сервис 34
2.4.2 Обрада грешака 40

Библиотечки систем БИСИС 43

3.1.ОБЈЕКТНИ МОДЕЛ БИБЛИОГРАФСКОГ ЗАПИСА У СИСТЕМУ БИСИС 45
3.2 ТЕКСТ СЕРВЕР СИСТЕМА БИСИС 47

3.2.1 Програмски пакет *Lucene* 49
3.2.1.1 Поступак индексирања 49
3.2.1.2 Поступак претраживања 50
3.2.2. Индексирање у систему БИСИС 51
3.2.3 Претраживање у систему БИСИС 55
3.2.3.1 Илустративни примери 57
Упитни језици 61
4.1 Z39.50 УПИТНИ ЈЕЗИЦИ 61
4.1.1 Z39.50 упитни језик типа -1 62
4.1.1.1 Скуп атрибута bib -1 65
4.1.2 Z39.50 упит типа-100 66
4.1.3 Z39.50 упитни језик типа-102 67
4.2 CQL УПИТНИ ЈЕЗИК 72
4.2.1 Context Set 72
4.2.1.1 CQL Context Set 74
4.2.1.2 Dublin Core Context Set 78
4.2.2 Синтакса CQL упитног језика 80
4.3 МАПИРАЊЕ Z39.50 УПИТНОГ ЈЕЗИКА НА CQL УПИТНИ ЈЕЗИК 85
4.4 МАПИРАЊЕ CQL УПИТНОГ ЈЕЗИК НА LUCENE УПИТНИ ЈЕЗИК 91
Архитектура система за посредовање између софтверских компоненти 95
5.1 АРХИТЕКТУРЕ СИСТЕМА ЗА ПРЕТРАЖИВАЊЕ И ПРЕУЗИМАЊЕ ПОДАТАКА 95
5.2 СОФТВЕРСКА КОМПОНЕНТА *INTERMEDIARY* 100

5.2.1 Софтверска компонента *mediator* 103
5.2.1.1 Имплементација софтверске компоненте *mediator* 109
5.2.2 Софтверска компонента *wrapper* 112
5.2.2.1 Трансформација CQL упита 115
5.2.2.2 Имплементација софтверске компоненте *wrapper* 118
Имплементација протокола SRU и Z39.50 125
6.1 СЕРВЕРСКА СТРАНА ПРОТОКОЛА SRU 125
6.1.1 WSDL документ SRU протокола 125
6.1.2 Моделирање SRU web сервиса 130
6.1.3 Имплементација серверске стране протокола SRU 132
6.2 КЛИЈЕНТСКА СТРАНА ПРОТОКОЛА SRU 136
6.3 СЕРВЕРСКА СТРАНА ПРОТОКОЛА Z39.50 139
6.4 КЛИЈЕНТСКА СТРАНА ПРОТОКОЛА Z39.50 140
6.4.1 Конфигурисање параметара библиотеке 142
6.4.2 Претраживање по вредностима *Use* атрибута 145
6.4.3 Претраживање употребом логичких оператора 147
6.4. 4 Напредно претраживање 149
6.5 СЕРВИС ЗА АЖУРИРАЊЕ ЗАПИСА У УДАЉЕНОЈ БАЗИ 152
6.5.1 Проширење SRU стандарда 153
Закључак 159
Литература 163
Биографија 171
КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА 173
KEY WORDS DOCUMENTATION 177

Поглавље 1

Увод

Предмет истраживања приказаних у овој дисертацији припада области пројектовања информационих система, односно пројектовању и имплементацији дистрибуираних библиотечких информационих система. Информационе технологије се мењају и развијају веома брзо и самим тим то захтева прилагођавање пословних процеса новим трендовима. Ове промене захватају све сфере друштва, па тако и пословање библиотека. Потребно је додати нове функционалности постојећим системима а да у исто време то буде и економично и квалитетно. Један од начина да се додају нове функционалности постојећем систему је свакако пројектовање и имплементација система из почетка, што је у већини случајева веома лоше решење јер су са једне стране за изградњу постојећег система већ утрошени одређени ресурси док са друге стране тај систем је постигао одређену зрелост и робусност. У овој дисертацији је посебан акценат дат на архитектури софтверске компоненте која омогућава интеграцију постојећих библиотечких система са одређеним сервисима који имплементирају нове функционалности система. Односно, у дисертацији је описан систем који обезбеђује претраживање и преузимање библиографских записа путем одговарајућих протокола. Описани систем је независан од библиотечког система у који се интегрише и може се једноставно интегрисати и у друге библиотечке системе.

1.1 АРХИТЕКТУРА ДИСТРИБУИРАНИХ СИСТЕМА

Софтверска компонента која је описана у овој дисертацији има задатак да интегрише различите сервисе који се могу користити у библиотекарству, првенствено се мисли на сервисе за претраживање и преузимање података, са постојећим библиотечким системом. Употребом ове компоненте добија се дистрибуирани систем који је базиран на сервис-оријентисаној архитектури и стога је у наставку дат преглед литературе која се односи на архитектуру дистрибуираних система.

Дистрибуирани систем представља систем у коме одређен број софтверских компоненти сарађује међусобно на тај начин што остварује комуникацију преко рачунарске мреже. Предности дистрибуираних система су пре свега:

- могућност дељења хардверских и софтверских ресурса,
- могућност међусобног повезивања хардвера и софтвера различитих произвођача што захтева постојање одређених стандарда за комуникацију,
- могућност паралелног извршавања одређених процеса на различитим рачунарима,
- скалабилност,
- робусност,
- транспарентност, односно могућност да се крајњем кориснику целокупан систем представи као интегрисана целина, без обзира што се одређени процеси могу одвијати на физички одвојеним рачунарима.

Са друге стране мане дистрибуираних система у односу на централизоване системе су:

- сложеност,
- смањена сигурност с обзиром да се подаци размењују путем мреже,
- немогућност утицања на перформансе система које зависе од оптерећености мрежног саобраћаја.

Што се тиче модела за развој дистрибуираних система, може се рећи да се прво користило мрежно програмирање, које је касније замењено *Remote Procedure Call* (RPC) парадигмом. Међутим сви претходни приступи су имали одређене недостатке који су покушани да буду превазиђене појавом *middleware*-а. *Middleware* представља систем који може бити посредник између апликације и операционог система или апликације и базе података. Поред улоге посредника, *middleware* треба да омогући једноставнију интеграцију других компоненти које имају одређену функционалност у дистрибуираном систему. Уколико се правилно испројектује и имплементира употреба *middleware*: олакшава посао програмеру у смислу да не мора да води рачуна о детаљима који зависе од платформе на којој се одређене компоненте дистрибуираног система инсталирају, смањује трошкове имплементације јер постоји могућност поновног коришћења већ постојећих компоненти, пружа одређене функционалности које су заједничке за већину система, као што је аутентификација, заштита, управљање трансакцијама и томе слично.

Природни наставак истраживања и рада на развоју *middleware* је свакако сервис-оријентисана архитектура (SOA). SOA пре свега омогућава интероперабилност између различитих сервиса, при чему сваки сервис енкапсулира одређену функционалност и при томе сервиси могу бити на физички различитим рачунарима. Сваки сервис је одређен добро дефинисаним интерфејсом који може бити објављен, пронађен и позван од стране неког другог сервиса (Lawler and Howell-Barber, 2007). Према томе креирање дистрибуираног система базираног на сервис-оријентисаној архитектури захтева композицију различитих сервиса у једну целину (Milanović and Malek, 2004). Један од начина да се имплементира систем који је базиран на сервис-оријентисаној архитектури је да се сваки сервис представи XML *web* сервисом, при чему је комуникација између два *web* сервиса стандардизована и врши се разменом SOAP порука (Casatio et al., 2004).

Међутим, приликом имплементације система базираних на SOA архитектури могу се појавити и одређени проблеми, као што су проблем прикупљања корисничких захтева јер сада може бити више корисника система који су физички удаљени, проблеми успостављања комуникације између два сервиса, проблем тестирања сервиса на систематичан начин и томе слично, и због тога је потребно имати одговарајући методолошки приступ како би се ти проблеми решили. Методологија која је описана у раду (Lee et al., 2006) комбинује искуства стечена применом агилних методологија и RUP (*Rational Unified Process*) модела са најбољим особинама *web* сервиса. Тренутно се веома пуно ради на проналаску методологија које би се користиле у развоју система базираних на SOA архитектури и у (Ramollari et al., 2007) дат је компаративни преглед неких од методологија које се могу користи за те намене.

Приликом развоја дистрибуираног система, сервис-оријентисана архитектура често се комбинује са развојем заснованим на компонентам (*Component-based development*), тако што се сваки сервис може састојати од више компоненти (Brown et al., 2002). Предност овакве архитектуре је првенствено у повећању продуктивности и смањењу трошкова израде система због могућности поновне употребе неких сервиса, међутим са друге стране појављује се проблем сигурности и поузданости таквих система.

1.2 SOA У БИБЛИОТЕЧКИМ СИСТЕМИМА

Модерни библиотечки системи између осталог имају и следећа два циља: да што више својих услуга путем Интернета учине доступним крајњим корисницима и да омогуће сарадњу са другим системима. Да би се ово остварило потребно је да архитектура библиотечких информационих система буде флексибилна како би се скуп услуга које једна библиотека пружа временом могао проширивати, али и да сервиси које библиотека нуди другим системима буду стандардизовани и с тога се сервис-оријентисана архитектура намеће као природни избор (Lavoie et al., 2006). Сматра се да ће се све већи број пословних процеса које библиотека извршава аутоматизовати и да ће се развој библиотечких система заснивати на комбинацији већ готових сервиса који енкапсулирају одређену функционалност потребну за рад библиотеке. Неки од сервиса које би модерна дигитална библиотека требала да понуди су:

- прикупљање, обрада и архивирање научних радова,
- претрага метаподатака о публикацијама које библиотека поседује,
- отварање сопственог библиотечког фонда за друге претраживаче,
- повезивање са системима за електронско учење.

Тренутно постоје различита софтверска решења која се баве појединачним задацима који су горе наведени, али њихова међусобна интеграција још увек није у потпуности могућа. *Web* сервиси имају велики потенцијал, али библиотеке могу имати користи од њих само ако се користе *web* сервиси који су засновани на стандардима (Wusteman, 2006). SRU стандард [1] је добар пример дефинисања сервиса који служи за претраживање метаподатака и при томе се може имплементирати као *web* сервис. Имплементација SRU *веб* сервиса који се користи у овој дисертавији дата је у шестом поглављу. Још један пример употребе *web* сервиса представља и интеграција дигиталних библиотека у системе који пружају могућност електронског учења и учења на даљину што је описано у раду (Chumbe et al., 2007).

Примера употребе сервис-оријентисане архитектуре у креирању библиотечких система је пуно. *Ncore* је само један од система који је базиран на SOA. Овај систем се развија под окриљем пројекта *National Science Digital Library* (NSDL) (Zia, 2002) и представља платформу која

омогућава интеграцију централног репозиторијума са различитим сервисима неопходним за функционисање библиотеке. Детаљи везани за архитектуру овог система дати су у раду (Krafft et al., 2008). Још један пример примене сервис-оријентисане архитектуре је оквир (*framework*) за кастомизацију дигиталних библиотека који се може интегрисати у постојеће библиотечке системе (Dong et al., 2008). Овај оквир се састоји од пет сервиса који су намењени за аутентификацију и ауторизацију корисника, рад са корисницима, преузимање електронских ресурса, предлагање публикација на основу интересовања појединачног корисника и кастомизацију интерфејса за претраживање за појединачног корисника. Сви сервиси су имплементирани као *web* сервиси и користе заједнички ESB (*Enterprise Service Bus*) који кординира радом сервиса.

Идеја која се крије иза употребе SOA у библиотечким системима је пре свега тенденција да библиотеке не буду пасивни учесници који ће само прикупљати податке а крајњи корисници уколико им требају неке информације приступати тим библиотечким системима. Циљ је да библиотечки системи постану активни учесници у смислу да ће они слати информације кориснику (Fox, 2009). На пример, библиотеке могу обавештавати корисника да ли је књига коју жели слободна, да ли треба да врате књигу, затим да ли су стигле неке нове књиге у библиотеку из области која конкретног корисника интересује и томе слично. Међутим да би се ово остварило потребно је да библиотечки систем не буде монолитан, већ да се састоји од више сервиса који ће моћи да се независно развијају и које ће моћи да користе и неки други системи.

1.3 БИБЛИОТЕЧКИ СТАНДАРДИ

Појава првих облика дигиталних библиотека везује се за почетак 60-тих година прошлог века, међутим праву експанизју дигиталне библиотеке доживљавају почетком 90-тих што се донекле поклапа и са експанзијом Интернета. Циљ дигиталних библиотека је да омогући сакупљање, чување, груписање информација како би се крајњим корисницима омогућио једноставан начин за претраживање, преузимање и каснију обраду тих информација. Постојање интероперабилних система који омогућавају проналажење и размену квалитетних информација електронским путем постало је основа пословања сваке библиотеке. Приликом израде ове дисертације

разматрани су различити библиотечки стандарди који се баве претраживањем и преузимањем података.

Развој библиотечких информационих ситема у последње време првенствено је окренут крајњем кориснику коме је потребно омогућити да на јединствен и једноставан начин приступи фондовима различитих библиотека. У циљу постизања ове интероперабилности осмишљени су различити библиотечки стандарди као што су на пример Z39.50 [2], SRU [1], OAI-PMH [3]. Детаљан опис ових стандарда дат је у другом поглављу ове дисертације.

Стандард Z39.50 је један од првих стандарда који је дефинисао скуп сервиса за претраживање и преузимање података. Овај стандард дефинише само комуникацију између клијента и сервера и не улази у детаље имплементације самог клијента односно сервера. Значајан допринос овог стандарда је пре свега у абстрактном моделу који се користи за претраживање и преузимање. Односно, овај модел дефинише абстрактне префиксе за претраживање који не зависе од имплементације конкретног система који пружа услугу претраживања и такође дефинише формат у коме се подаци могу размењивати.

Ипак Z39.50 има одређених недостатака које стандард новије генерације SRU покушава да превазиђе. SRU стандард покушава да задржи функционалности које су дефинисане Z39.50 стандардом, али да омогући његову имплементацију помоћу тренутно актуелних технологија. Једна од основних предности протокола SRU у односну на Z39.50 је у томе да омогућава размену порука у виду XML документа што није био случај са Z39.50 протоколом.

SRU стандард има две имплементације једна код које се претраживање и преузимање обавља слањем порука путем HTTP GET и POST метода (SRU) и друга која за размену порука користи SOAP протокол (SRW). Основна разлика између SRU и SRW верзије протокола је пре свега у начину слања порука (Morgan, 2004). SRW верзија протокол поруке пакује у SOAP *Envelope* елемент, док SRU верзија протокола поруку представља по принципу параметар/вредност и те парове параметар/вредност укључује у URL адресу. Друга разлика између ове две верзије протокола је да SRU за пренос порука користи само HTTP протокол, док SRW поред HTTP протокола може користити и SSH (*Secure Shell*) и SMTP (*Simple Mail Transfer Protocol*) протокол.

Још један од протокола који се често користи у библиотекарству је OAI-PMH (*The Open Archives Initiative Protocol for Metadata Harvesting*)

протокол. OAI-PMH протокол је намењен за преузимање библиографских записа и складиштење тих података у заједничку архиву. За разлику од SRU стандарда, OAI-PMH стандардом није дефинисан упитни језик и самим тим овај стандард не подржава претраживање већ обезбеђује преузимање одређеног скупа записа које један систем поседује. Поред преузимања метаподатака о ресурсима, OAI-PMH протокол се може користити и за преузимање самих електронских ресурса што је описано у раду (Sompel et al., 2004)

Веома је чест случај да се више библиотечких стандарда користи заједно у различитим пројектима. Једна од могућих заједничких примена је да се направи централни каталог који би се претраживао путем SRU протокола, док би се у исто време тај каталог пунио библиографским записима из различитих фондова путем OAI-PMH протокола (Sanderson et al., 2005). Овај модел искоришћен је у реализацији портала Европских библиотека. Овај портал описан је у раду (Van Veen and Oldroyd, 2004) и циљ портала је да повеже водеће европске библиотеке и формира јединствен централни каталог. Још један пример повезивања дигиталних репозиторијума представља пројекат NORA (Stangeland and Moe, 2006) који се развија у Норвешкој и он представља централни репозиторијум чији циљ је да омогући преузимање података из осталих репозиторијума путем OAI-PMH протокола. NORA је предвиђено да буде сервис коме ће се моћи приступити путем SRU протокола. Сличан приступ у комбиновању OAI-PMH и SRU протокола је искоришћен у пројекту Ockham [4]. Ово је пројекат који се бави дигиталним библиотекама и развија се на *Emory University*, *Oregon State University*, *Virginia Tech*, и *University of Notre Dame*. Основни циљ пројекта је промовисање развоја дигиталних библиотека који је заснован на доступним стандардима и *open-source* софтверским решењима како би што више традиционалних библиотека прихватило нове трендове. У раду (Xiang and Morgan, 2005) описан је систем *Ockham Alerting Service* који служи за претраживање других OAI-PMH репозиторијума. Систем је тако осмишљен да се сваког дана путем OAI-PMH протокола добавља нови садржај из удаљених репозиторијума и смешта у локалну базу. Садржај који је старији од 30 дана се брише из локалне базе и на тај начин база садржи само најактуелније записе. Ова база се може претраживати путем SRU протокола.

С обзиром да је имплементација сервиса који подржавају SRU протокол много једноставнија велики број библиотека имплементира

овај протокол и међу првим библиотекама које су прихватиле овај протокол је Конгресна библиотека у Вашингтону. Међутим, библиотеке које се одлучују за имплементацију сервиса који ће користити SRU протокол, желе да задрже и могућност претраживања по Z39.50 протоколу уколико су већ имали тај сервис имплементиран. Конгресна библиотека подржава претраживање и преузимање података путем Z39.50 протокола и за имплементацију овог сервиса искоришћен је комерцијални систем Voyager [5] који поред ових функционалности обезбеђује и друге функционалности, као што је на пример каталогизација публикација, рад са корисницима и стога је било важно да се задржи постојећи систем и да се само додају нови сервиси. Међутим како је то комерцијалан систем, није било могуће мењати имплементацију тог система па је у раду (Taylor and Dickmeiss, 2005) описан један од начина за интеграцију нових сервиса у постојеће окружење. У том раду описана је компонента YazProxy [6] која представља „*gateway*“, односно ова компонента садржи и серверску страну протокола SRU али и клијентску страну протокола Z39.50, тако да клијент шаље SRU поруку *gateway*-у, а он ту поруку трансформише у одговарајућу Z39.50 поруку и прослеђује је серверској страни протокола Z39.50.

Још један од често примењиваних начина приликом имплементације SRU сервиса у постојеће библиотечке системе је употреба *open-source* компоненти које садрже имплементацију SRU сервиса. Ово решење је могуће уколико је изворни код постојећег библиотечког система могуће прилагођавати како би се интегрисала нова компонента. Пример компоненте која представља имплементацију SRU сервиса је OCLC SRU/W софтверска компонента [7]. Ово је *open-source* решење базирано на Јава програмском језику и садржи интерфејс ка DSpace систему [7] који представља институционални репозиторијум. С обзиром да је то *open-source* решење ова компонента се може прилагодити и неким другим текст-серверима. У радовима (Lam and Chan, 2007; Hoppe et al., 2008) описани су системи који су интегрисали ову компоненту у своје постојеће окружење.

1.4 ПРЕДМЕТ И ЦИЉ ИСТРАЖИВАЊА

Један од сегмената елекронског пословања библиотека је тзв. узајамна каталогизација. Ово је процес обраде библиографске грађе који се састоји у следећем. Библиотекар при обради нове библиографске јединице прво провери да ли је та библиографска јединица већ обрађена

у некој другој библиотеци у свету која има сервис за узајамну каталогизацију. Ако је пронађе, тада библиотекар тај пронађени електронски запис преузима у своју локалну базу библиографских записа. После тога, библиотекару остаје само да у тај запис унесе локацијске податке и та библиографска јединица је обрађена. Дакле, основна идеја је да када се у једној библиотеци формира електронски библиографски запис да је тада тај запис доступан и свим осталим библиотекама за преузимање. Овим се поред вишеструке уштеде рада за обраду добијају и квалитетнији библиографски записи.

Предмет истраживања у овој дисертацији је израда софтверског пакета за узајамну каталогизацију, односно израда електронског сервиса за размену библиографских записа. С обзиром да постоје различити стандарди за размену библиографских записа, циљ истраживања је да се креира такав систем који ће омогућавати интеграцију нових стандарда а да се постојећа архитектура система не мења. Такође, битно је обезбедити да се овако развијен систем може једноставно интегрисати у постојеће библиотечке системе. Резултати истраживања приказаних у овој дисертацији су наставак истраживања која су започета у магистарској тези (Боберић, 2007). У магистраској тези је реализован едитор за претраживање и преузимање библиографских записа путем Z39.50 протокола. Односно реализована је клијентска апликација која користи протокол Z39.50. Едитор је интегрисан у библиотечки систем БИСИС и на тај начин је обезбеђено да библиотеке које користе систем БИСИС преузимају библиографске записе од других библиотечких система. Опис овог едитора дат је у шестом поглављу ове дисертације. Циљ ове дисертације је да се имплементира систем који ће и другим библиотекама омогућити да преузму библиографске записе из система БИСИС. На тај начин библиотеке које користе систем БИСИС ће отворити своје фодове ка другим светским и домаћим библиотекама.

Претпоставке од којих се полази у овој дисертацији су:

- да се може формирати упитни језик који ће садржати све концепте који су дефинисани упитним језицима који се користе у протоколима за размену података,
- да се могу дефинисати трансформације којима би се упит формиран помоћу једног упитног језика трансформисао у еквивалентан упит дефинисан неким другим упитним језиком,

- да је могуће формирати софтверску архитектуру серверске стране система за преузимање записа тако да подржава претраживање и преузимање записа по различитим стандардима.

Резултати истраживања у овој дисертацији су потврдили наведене полазне претпоставке и основни добијени резултати су следећи:

- С обзиром да су два најчешће коришћена стандарда која описују процес размене података Z39.50 и SRU, и да они дефинишу упитне језике тип-1 и CQL, показано је да се упит типа-1 може тансформисати у упит дефинисан CQL упитим језиком (одељак 4.3). CQL садржи све основне концепте који су неопходни за успешно претраживање и зато је он изабран као општи упитни језик који се користи у реализацији система описаног у овој дисертацији.
- Дефинисане су трансформације упитног језика типа-1 у CQL упитни језик, као и трансформације CQL упитног језика у *Lucene* упитни језик (одељак 4.4). Дата је трансформација CQL упитног језика у *Lucene* јер је софтверска компонента која је описана у овој дисертацији интегрисана у систем БИСИС чији текст сервер је базиран на *Lucene*-у и овај текст сервер је описан у одељку 3.2 ове дисертације.
- Креирана је софтверска компонента која је посредник између постојећег библиотечког система и сервиса за претраживање и преузимање записа (поглавље 5). Ова компонента је базирана на *mediator/wrapper* моделу што јој обезбеђује флексибилност у смислу додавања нових сервиса а и једноставну интеграцију у друге библиотечке системе.
- Моделирана и имплементирана је серверска и клијентска страна протокола SRU (поглавље 6).
- Дат је предлог проширења SRU стандарда у циљу да се овај стандард користи и за комуникацију између клијента и сервера када је потребно снимање података у удаљену базу података и на тај начин формирање централног каталога (одељак 6.5).
- Описан је поступак интегрисања реализованог система у библиотечки систем БИСИС верзије 4.

Поглавље 2

Стандарди за преузимање информација

Један од основних проблема са којим се сусрела библиотечка заједница био је како омогућити електронску размену библиографских записа између различитих библиотека с обзиром да библиотеке могу користити системе са различитом софтверском архитектуром. Решење је било у дефинисању стандардног начина за комуникацију који би сви учесници у комуникацији морали да поштују. Тако је још крајем 80-тих година 20. века Конгресна библиотека дефинисала Z39.50 стандард [2] за претраживање и преузимање библиотечких записа. Данас је сервис за претраживање и преузимање библиотечких записа путем Z39.50 протокола есенцијални елемент сваке савремене библиотеке и у раду (McCallum, 2006) је наведено да Конгресна библиотека у току једног дана добије у просеку 75.000 захтева за претраживање њеног каталога путем Z39.50 протокола.

Са развојем нових технологија као што је XML појављују се и нови стандарди и у наредним одељцима поред детаљног описа Z39.50 стандарда биће описани и тренутно актуелни стандарди у домену претраживања и преузимања информација, а то су SRU, OpenSearch и OAI-PMH стандарди. Посебан акценат је стављен на стандарде Z39.50 и SRU пре свега јер софтверски систем који ће бити описан у овој дисертацији подржава комуникацију базирану на овим стандардима и у шестом поглављу дат је и опис имплементације клијентских и серверских страна апликација које су неопходне како би се остварила та комуникација.

2.1 Z39.50 СТАНДАРД

Z39.50 (*ANSI/NISO Z39.50-2003, Information Retrieval (Z39.50): Application Service Definition and Protocol Specification)* је стандард организације NISO (*National Information Standards Organization*). Прихваћен је од стране организације ISO (*International Standards Organization*) као ISO 23950:1998, *Information and documentation- Information retrieval (Z39.50) - Application service definition and protocol*

specification. Актуелна верзија стандарда је верзија 3 и датира из 2003. године. Стандард је осмишљен као општи стандард који се бави проблемима проналажења и преузимања података из удаљених база података, мада је његова најчешћа употреба у системима као што су библиотеке, универзитети и централни каталози. Агенција која се бави формалним дефинисањем, унапређивањем и одржавањем стандарда Z39.50 је *International Standard Z39.50 Maintainance Agency* у оквиру Конгресне библиотеке [9].

Стандард описује сервисе који се користе приликом претраживања и добављања информација и даје спецификацију одговарајућег протокола који се користи при клијент/сервер комуникацији и који подржава дате сервисе.

Сервиси описују активност између две апликације: апликације која иницира активност (Z-клијента) и апликације која шаље одговор (Z-сервера), мада постоје и сервиси у којима је иницијатор акције серверска страна протокола. Z-сервер је повезан са једном или више база података. Сервиси су подељени на процедуре које извршава Z-клијент и процедуре које извршава Z-сервер.

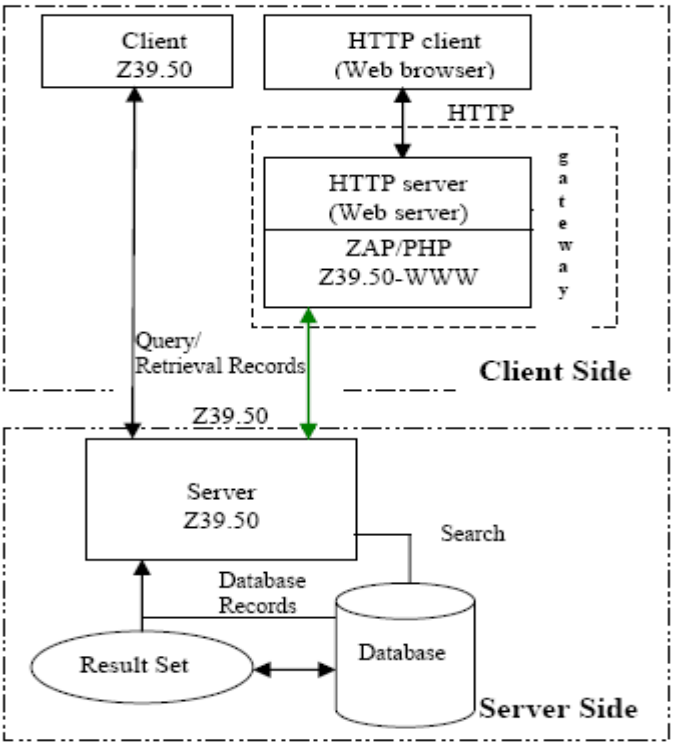
При спецификацији протокола дефинишу се правила за размену информација као и основни захтеви који се морају задовољити приликом имплементације протокола. Стандард не описује начин имплементације сервиса нити протокола у оквиру појединачног система.

Стандард је базиран на OSI моделу (*Open Systems Interconnection Basic Reference Model*) дефинисаног током 80-тих година. У оквиру OSI модела Z39.50 представља протокол на апликационом нивоу и од нижих слојева OSI модела Z39.50 захтева поуздан потпуно двосмеран бинарни транспортни протокол, као што је TCP. За спецификацију садржаја пакета података користи се апстрактна синтаксна нотација – *Abstract Syntax Notation One* (ASN.1) [10], а за серијализацију ASN.1 структура користе се *Basic Encoding Rules (*BER*)*[11].

2.1.1 Z39.50 сервиси

На слици 2.1 која је преузета из (Trichkov et al., 2005) приказана је архитектура система који учествују у комуникацији путем протокола Z39.50. На клијентској страни налази се кориснички интерфејс који омогућава крајњем кориснику система да постави упит и да манипулише са преузетим подацима. Z-клијент представља клијентску страну протокола и он може директно комуницирати са серверском

страном. Међутим, комуникацију је могуће остварити и преко система који се називају капије (*gateway*). Капија је систем који има два интерфејса. Овде су од значаја капије чији је бар један интерфејс Z39.50 клијент или сервер. У том случају крајњи корисник преко одређеног браузера комуницира са капијом која у себи поред *web* сервера садржи и Z-клијента. На слици 2.1 је приказан софтвер ZAP/PHP који представља конкретну имплементацију Z-клијента и преко њега се врши комуникација са Z-сервером.



Слика 2.1 Клијент-сервер архитектура базирана на протоколу Z39.50 [преузето из Trichkov et al., 2005]

На серверској страни, између осталог, налази се и база података. Комуникација између базе и система није дефинисана стандардом и то зависи од самог система. Z-сервер се налази на серверској страни и његов задатак је да реализује серверску страну протокола Z39.50. На серверској страни је потребно извршити и мапирање локалне базе података тако да ти подаци буду доступни Z-серверу. То значи да када Z-сервер прими упит од Z-клијента, тај упит се мора трансформисати тако да се може применити на локалну базу података. Пронађени

подаци се морају прилагодити формату који је погодан за даљу обраду од стране Z-сервера. На слици 2.1 су ове трансформације приказане разменом порука између Z-сервера и базе података и компонентом *Result Set* која представља скуп податак пронађених у бази.

Између Z-клијента и Z-сервера успоставља се конекција, односно Z-асоцијација. Комуникација између Z-клијента и Z-сервера се одиграва разменом порука. Садржај и редослед порука дефинисан је сервисима који су груписани у структурне блокове - *facilities.* Порука може бити захтев (*request*) или одговор (*response*). Сервиси могу бити потврдни, непотврдни и условно-потврдни.

Потврдни сервис је такав сервис који има захтев и одговор. Код непотврдних сервиса постоји само захтев који могу да упуте или Z-клијент или Z-сервер, и не постоји порука о одговору. Условно-потврдни сервис се може посматрати и као потврдни и као непотврдни сервис у зависности да ли се у захтеву специфицира да је потребно проследити и одговор или то није потребно.

Z39.50 стандард дефинише 11 структурних блокова или механизама (*facilities*)*:*

1. Механизам за иницијализацију (*Initialization Facility*)
   - *Init* сервис;
2. Механизам за претраживање (*Search Facility*)
   - *Search* сервис;
3. Механизам за добављање информација(*Retrieval Facility*)
   - *Present* сервис,
   - *Segment* сервис;
4. Механизам за брисање скупа погодака (*Result-set-delete Facility*)
   - *Delete* сервис;
5. Механизам за прегледање (*Browse Facility*)
   - *Scan* сервис;
6. Механизам за сортирање (*Sort Facility)*
   - *Sort* сервис,
   - *Duplicate Detection* сервис;
7. Механизам за контролу приступа (*Access Control Facilty)*
   - *Access Control* сервис;
8. Механизам за контролу ресурса (*Accounting /Resource Control Facility*)
   - *Resource Control* сервис,
   - *Triggger-resource-control* сервис,

- *Resource-report* сервис;

9. Механизам за објашњења (*Explain Facility*)
10. Допунски механизам (*Extended Services Facility*)
    - *Extended-services* сервис;
11. Механизам за прекид конекције (*Termination Facility)*
    - *Close* сервис.

Init сервис омогућава клијенту да успостави Z-асоцијацију. У *Init* захтеву Z-клијент предлаже вредности за иницијалне параметре. У *Init* одговору Z-сервер даје предлог својих вредности за иницијалне параметре, и те се вредности могу разликовати од оних које је предложио Z-сервер. Ако Z-сервер потврдно одговори на Z-клијентов предлог Z-асоцијација се успоставља. Уколико Z-клијент не жели да прихвати параметре које је понудио Z-сервер, он може одустати од даље комуникације. Неки од параметара који се могу поставити су следећи: верзија Z39.50 протокола која се користи у комуникацији, максимална величина слога који ће Z-сервер вратити клијенту, подаци неопходни за аутентификацију Z -клијента, подаци о имплементацији сервера/клијента.

Search сервис омогућава Z-клијенту да дефинише упит који ће се применити на бази података и да прими информације о резултату извршавања упита. Као одговор на Z-клијентов захтев Z-сервер креира скуп резултата који задовољава упит.

Present сервис омогућава Z-клијенту да затражи приказ слогова који су добијени претходном претрагом. Тражени слогови (записи) су груписани у оквиру скупа резултата (*result set*) и имају релативну позицију у оквиру тог скупа. На овај начин је обезбеђено да Z-клијент може тражити један конкретан запис или да дефинише опсег из ког жели преузети записе.

Уколико слог који је захтеван преко *Present* сервиса превазилази величину слога која је договорена у фази иницијализације тада Z-сервер може вратити слог распоређен у више сегмената. Слање оваквих сегмената обезбеђено је *Segment* сервисом.

Delete сервис пружа могућност Z-клијенту да од Z-сервера захтева брисање одређеног скупа резултата или брисање свих скупова резултата који су формирани током Z-асоцијације. Овај сервис може бити веома користан у случају када Z-сервер има ограничен број скупова које може чувати у меморији. Када број скупова резултата пређе дозвољену границу Z-сервер ће по неком свом алгоритму

обрисати неке од скупова, тако да уколико неки од тих обрисаних скупова поново затребају Z-клијенту они више неће бити доступни. Због тога је боље решење да Z-клијент сам избрише скупове резултата који су му непотребни.

Scan сервис представља варијанту претраживања. Омогућава претраживање у оквиру уређене индексиране листе термова (тзв. индекс). Z-клијент треба да наведе префикс, на пример префикс може бити наслов дела, а Z-сервер може вратити који све термови се налазе под тим индексом и који је њихов број појављивања у индексу. На пример, ако корисник жели да погледа у којим све варијантама се јавља реч „Моцарт“ у наслову, он може искористити овај сервис и добиће листу термова као што су: „Моцарт и Бетовен“, „Моцарт у Прагу“, “Моцартово детињство“ и томе слично. Резултат ове операције су само индексирани изрази, тако да би се добио целокупан запис, потребно је позвати сервис *Search*. Показано је да је боље прво позвати сервис *Scan*, па затим извршити претраживање јер се на тај начин смањује могућност да се добију неочекивани резултати.

Sort сервис омогућава Z-клијенту да од Z-сервера захтева сортирање скупа резултата. Z-клијент дефинише правила за сортирање и слогови унутар скупа резултата треба да буду уређени према тим правилима.

Duplicate Detection сервис дозвољава Z-клијенту да од Z-сервера затражи да анализира један или више скупова резултата у циљу проналажења слогова који су дупликати. Z-клијент дефинише правила за препознавање дупликата.

Access-control сервис омогућава Z-серверу да провери Z-клијента. Под провером се подразумева да Z-сервер, уколико је то договорено у фази иницијализације, у сваком тренутку може послати Z-клијенту захтев у коме тражи од Z-клијента да се идентификује. Уколико Z-клијент одговори на начин који је прихватљив Z-серверу комуникација се наставља, у супротном Z-сервер може одбити да изврши операцију која је била у току пре него што је дошло до слања *Access-control* захтева или може затворити Z-асоцијацију.

Resource-control сервис служи за извештавање о стању ресурса. Овај сервис увек иницира Z-сервер. На пример, Z-сервер може обавестити Z-клијента да ће доћи до прекорачења ресурса приликом извршавања операције и од Z-клијента може захтевати потврду о наставку започете операције.

Trigger-resource-control сервис позива Z-клијент као део неке друге активне операције. Овај сервис омогућава Z-клијенту да затражи од Z-сервера извршавање *Resource-control* сервиса или прекидање неке операције. Z-сервер није обавезан да одговори на овакав Z-клијентов захтев због тога и не постоји *Trigger-resource-control* одговор, већ само *Trigger-resource-control* захтев.

Resource-report service омогућава Z-клијенту да затражи *Resource-report* који ће се односити на одређену завршену операцију или целокупну Z-асоцијацију. *Resource-report service* се разликује од *Trigger-resource-control* сервиса по томе што је *Trigger-resource-control* непотврдан сервис и захтева извештај који се односи на тренутно активну операцију, док је *Resource-report* потврдни сервис и извештај који се захтева се односи на операцију која је завршена.

Explain facility не садржи ни један сервис и омогућава Z-клијенту да види детаље о имплементацији Z-сервера: опште могућности (опис, контакт информације, радно време, ограничења, цену коришћења), које су базе података на располагању, детаље о атрибутима и скуповима атрибута, синтаксу слогова. Све ове информације чувају се на Z-серверу у посебној бази којој Z-клијенти могу приступити преко Z39.50 протокола.

Extended Services сервис служи за иницирање посебних додатних задатака који се извршавају ван сесије Z39.50 и чији се напредак може пратити кроз Z39.50 сервисе. Неки од ових специфичних сервиса су: чување скупа резултата, подешавање периодичних упита, експортовање докумената, ажурирање базе података.

Close сервис омогућава Z-клијенту или Z-серверу да оконча све операције које су у току и да иницира затварање Z-асоцијације.

Стандардом су следећи сервиси дефинисани као обавезни приликом имплементације протокола Z39.50: *Init, Search Present* и *Close* сервис, док су остали сервиси опциони и зависе од имплементације.

2.2 OPENSEARCH ПРОТОКОЛ

OpenSearch [12] је протокол за претраживање који се појавио 2005. године и производ је софтверске компаније A9, једне од Амазонових филијала. Овај протокол је базиран на XML технологијама. Протокол је базиран на размени XML докумената путем HTTP протокола. Основни елементи *OpenSearch* протокола су:

- *OpenSearch Description* документ који описује функционалности сервиса који пружа услуге претраживања,
- резултат претраге који је враћен у виду RSS [13] или *Atom* [14] документа.

OpenSearch Description документ је XML документ који садржи опис интерфејса одређеног претраживача (*search engine*). Овај документ описује начин на који клијент може да изврши претрагу и даје додатне информације о самом претраживачу, као што је кратак опис, подаци о особи која је задужена за администрацију претраживача и томе слично. Документ садржи URL адресу до претраживача и опис упита који се може направити. Не постоји дефинисан упитни језик него сваки претраживач дефинише упитни језик који жели да користи. Идеја која се крије иза овог протокола је да уколико неко жели да свој претраживач учини доступан и другима, потребно је само да дефинише *OpenSearch Description* документ и да га објави на неком од директоријума који садржи листу претраживача који подржавају *OpenSearch* протокол. Пример оваквог директоријума је [15]. У наставку је дат пример *OpenSearch Description* документа (листинг 2.1) који описује претраживач часописа *New York Times*.

```
<?xml version="1.0" encoding="UTF-8" ?>
  <OpenSearchDescription xmlns=
                  "http://a9.com/-/spec/opensearch/1.1/">
   <ShortName>New York Times</ShortName>
   <Description>
     Searches the New York Times  newspaper
   </Description>
   <Url type="text/html"
        template="http://query.nytimes.com/search/query?
                     query={searchTerms}" />
   </OpenSearchDescription>
```

Листинг 2.1 Пример *OpenSearch Description* документа

Из овог документа најбитнија је URL адреса претраживача која је представљена атрибутом `template` у оквиру `Url` елемента. Анализом садржаја атрибута `template` види се да је адреса претраживача `http://query.nytimes.com/search/query` и да овај претраживач користи само један префикс `query` по коме се може претраживати.

Да би извршио претраживање клијент шаље HTTP захтев претраживачу са дефинисаним упитом и добија одговор у виду RSS

или *Atom* документа. HTTP захтев заједно са упитом који клијент шаље претраживачу има облик:

```
http://query.nytimes.com/search/query?query=New York history
```

Пример одговора сервера на упит који је клијент послао дат је на листингу 2.2 у виду RSS 2.0 документа.

```
<?xml version="1.0" encoding="UTF-8"?>
 <rss version="2.0"
      xmlns:opensearch="http://a9.com/-
      /spec/opensearch/1.1/"
      xmlns:atom="http://www.w3.org/2005/Atom">
   <channel>
     <title>Example.com Search: New York history</title>
     <link>http://example.com/New+York+history</link>
     <description>
      Search results for "New York history" at  Example.com
     </description>
     <opensearch:totalResults> 42300
     </opensearch:totalResults>
     <opensearch:startIndex>21</opensearch:startIndex>
     <opensearch:itemsPerPage>10</opensearch:itemsPerPage>
     <atom:link rel="search"
      type="application/opensearchdescription+xml"
      href="http://example.com/opensearchdescription.xml"/>
     <opensearch:Query role="request" searchTerms="New York
         History" startPage="1" />
     <item>
       <title>New York History</title>
       <link>
          http://www.columbia.edu/cu/lweb/eguids/amerihist/
          nyc.html
       </link>
       <description>
          ... Harlem.NYC - A virtual tour and information on
         businesses ...  with historic photos of Columbia's
        own New York neighborhood ... Internet Resources
        for the City's History. ...
       </description>
     </item>
   </channel>
 </rss>
```

Листинг 2.2 Пример RSS 2.0 документа

2.3 OAI-PMH СТАНДАРД

OAI-PMH (*Open Archives Initiative Protocol for Metadata Harvesting*) [3] је стандард који је намењен прикупљању метаподатака о

библиографским записима и смештање тих података у једну заједничку архиву. Треба напоменути да овај стандард не подржава претраживање већ само преузимање свих расположивих метаподатака и снимање у једну заједничку архиву којој се може приступити путем неког од протокола за претраживање и преузимање (на пример путем Z39.50 протокола).

OAI-PMH стандардом дефинисана су два типа учесника у комуникацији: *Data Provider* и *Service Provider*. *Data Provider* (репозиторијум) обезбеђује слободан приступ метаподацима док *Service Provider* путем OAI-PMH протокола приступа *Data Provider*-у и преузима метаподатке.

OAI-PMH протокол је заснован на HTTP протоколу. *Service Provider* шаље захтеве у виду GET и POST захтева, при чему је обавезно дефинисан параметар *verb*, док у зависности од вредности параметра *verb* могу бити дефинисани и додатни параметри. Параметар *verb* може имати једну од следећих шест предефинисаних вредности:

- *Identify* – ова вредност служи за идентификацију *Data Provider-a*. Наиме, *Data Provider* као одговор на овај захтев шаље информације као што је назив репозиторија, коју верзију протокола користи, адресу контакт особе и томе слично. Пример употребе ове вредности је:

  ```
  http://archive.org/oai-script?verb=identify
  ```

- *ListMetadataFormats* – користи се за добијање информација о доступним форматима за опис метаподатака. Обавезан формат који морају да подрже сви репозиторијуми је *Dublin Core* формат [16]. Опциони параметар који се може комбиновати са овом вредношћу је *identifier*.

- *ListSets* – сваки репозиторијум може креирати логичке целине (скупове) које могу бити хијерархијски организовани и у којима су смештени метаподаци. Употребом ове вредности добијају се сви дефинисани скупови.

- *ListIdentifiers* – сваки запис поред метаподатака који су у неком дефинисаном формату мора имати и заглавље у коме је дат идентификатор записа, датум крирања/модификовања записа, путања до скупа у коме се налази запис. Употребом ове вредности добијају се само заглавља без метаподатака.

- *ListRecords* – служи за преузимање записа из репозиторијума. Може се дефинисати и да ли се преузимају записи који припадају тачно одређеном скупу или сви записи без обзира на припадност скуповима. Такође, може се дефинисати и преузимање само оних записа који су обрађени у одређеном периоду.
- *GetRecord* – користи се за преузимање појединачног записа и при томе се обавезно мора навести и параметар који јединствено идентификује запис.

Одговор који *Data Provider* шаље *Service Provider*–у је у виду HTTP одговора при чему је садржај дефинисан као *text/xml*. XML документ који се размењује има коренски елемент OAI-PMH и он има три обавезна поделемента:

- *responseDate* – садржи датум и време када је добијен одговор,
- *request* – садржи захтев који је прослеђен *Data Provider*-у и самим тим проузроковао одговор,
- трећи поделемент има назив као и вредност *verb* параметра која је наведена у захтеву. На пример ако је *Service Provider* послао захтев за идентификацију *Data Provider-a*, елемент ће се звати *Identify*. Овај елемент може имати своје поделементе који су дефинисани шемом [17]. У случају да се деси грешка овај трећи елемент може бити замењен елементом *error*.

Пример XML документа који описује одговор *Data Provider*-а на клијентов захтев:
`http://eprints.iisc.ernet.in/cgi/oai2?verb=Identify`
дат је на листингу 2.3.

```
<?xml version="1.0" encoding="UTF-8" ?>
<OAI-PMH
   xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
   http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd"
   xmlns="http://www.openarchives.org/OAI/2.0/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <responseDate>2010-01-04T13:07:29Z</responseDate>
  <request verb="Identify">
     http://eprints.iisc.ernet.in/cgi/oai2
  </request>
  <Identify>
    <repositoryName>ePrints@IISc</repositoryName>
   <baseURL>http://eprints.iisc.ernet.in/cgi/oai2</baseURL>
    <protocolVersion>2.0</protocolVersion>
    <adminEmail>admin@eprints.iisc.ernet.in</adminEmail>
```

```
    <earliestDatestamp>
        0001-01-01T00:00:00Z
    </earliestDatestamp>
    <deletedRecord>persistent</deletedRecord>
    <granularity>YYYY-MM-DDThh:mm:ssZ</granularity>
    <description>
     <oai-identifier
       xsi:schemaLocation="http://www.openarchives.org/OAI
       /2.0/oai-identifier
       http://www.openarchives.org/OAI/2.0/oai-
       identifier.xsd"
       xmlns="http://www.openarchives.org/OAI/2.0/oai-
       identifier">
       <scheme>oai</scheme>
       <repositoryIdentifier>
            generic.eprints.org
       </repositoryIdentifier>
       <delimiter>:</delimiter>
       <sampleIdentifier>
            oai:generic.eprints.org:23
       </sampleIdentifier>
     </oai-identifier>
   </description>
  </Identify>
</OAI-PMH>
```

Листинг 2.3 Пример одговора *Data Provider*-а

2.4 SRU СТАНДАРД

Z39.50 је у протеклих двадесет година био један од најчешће коришћених протокола у размени библиотечких записа, међутим уочено је да овај протокола поред свих својих предности има и неких недостатака који се углавном односе на потешкоће које се јављају приликом имплементације овог протокола. Због тога су корисници Z39.50 протокола покренули различите иницијативе у циљу да се Z39.50 протокол учини прихватљивијим већем броју произвођача софтвера, испоручиоца софтверских система и корисника. Један од пројеката који има за циљ да се смањи сложеност протокола Z39.50, али да се очувају његове основне функционалности је стандард SRU (*Search and Retrieve URL Service*) [1].

SRU протокол је базиран на технологијама које су широко доступне - XML, SOAP, HTTP, URI. SRU је стандард који дефинише два транспортна механизма. Један механизам је пренос порука употребом HTTP GET или POST метода док други механизам подразумева

размену порука путем SOAP протокола. Приликом слања поруке путем HTTP GET метода мора се водити рачуна о енкодирању *Unicode* карактера који нису дозвољени у URI идентификатору. Слање порука путем HTTP POST метода има неколико предности у односу на слање поруке путем HTTP GET методе пре свега јер се не мора водити рачуна о енкодирању карактера и не постоји ограничење на број параметара који ће се појавити у URL адреси.

SRU може користити SOAP протокол као транспортни механизам и у том случају и захтев и одговор су дати у виду XML документа. Предности употребе овог механизма је у томе што може постојати више SOAP *intermediary* чворова који могу бити посредници између клијента и неког другог сервиса и прослеђивати захтеве клијента другим сервисима. Такође, Web сервис пружа могућност стандардизоване аутентификације клијента. У наставку ће акценат бити дат само на SRU сервисима који користе SOAP као транспортни механизам.

2.4.1 SRU сервиси

За разлику од Z39.50 стандарда који је дефинисао 11 различитих сервиса, по SRU стандарду су дефинисана само три сервиса, пре свега јер је уочено да се само одређен број сервиса дефинисаних Z39.50 стандардом стварно користи у пракси. Сервиси дефинисани SRU стандардом су следећи:

- *SearchRetrieve,*
- *Scan,*
- *Explain.*

2.4.1.1 SearchRetrieve сервис

Сервис *SearchRetrieve* задужен је за претраживање и добављање података на тај начин што клијент пошаље *SearchRetrieveRequest* поруку и од сервера добија одговарајућу *SearchRetrieveResponse* поруку. Порука *SearchRetrieveRequest* састоји се од више параметара од којих су само параметар који дефинише верзију протокола и параметар који дефинише упит обавезни док су остали параметри опциони и уколико клијент не дефинише вредности за те параметре сервер ће сам поставити вредности за те параметре. Тренутно су актуелне две верзије стандарда 1.1 и 1.2, и клијент када шаље поруку мора да дефинише верзију стандарда по којој ради, уколико сервер не подржава ту верзију вратиће одговарајућу грешку. Параметар који

дефинише верзију протокола је обавезан за све сервисе који су дефинисани SRU стандардом.

Подаци које клијент добија од сервера могу бити у форми XML документа што клијенту омогућава да употребом XSLT трансформација веома једноставно и са минимално обраде прикаже резултате претраге. Уколико клијент жели да подаци буду у форми XML документа он може у захтеву дефинисати параметар *recordSchema* којим је дефинисао XML шему по којој очекује да му сервер врати записе. Постоји скуп регистрованих XML шема које су прихваћене од Конгресне библиотеке и свака шема има назив и URI идентификатор који је јединствен. На пример записи могу бити у *Dublin Core* формату и шема која је дефинисана за овај формат има назив `dc` и идентификатор је `info:srw/schema/1/dc-v1.1`. Уколико сервер не може да врати записе у специфицираној шеми вратиће одговарајућу поруку о грешци. Међутим, подаци не морају бити у форми XML документа, већ могу бити дати у виду слободног текста. Да ли ће подаци бити враћени као слободан текст или као XML документ клијент специфицира одговарајућим вредностима параметра *recordPacking*, који је обавезан елемент *SearchRetrieveRequest* поруке. Уколико је корисник дефинисао по којој XML шеми жели да добије записе онда може дефинисати и XPath израз којим ће добити тачно одређен део из записа, а не цео запис. Овај параметар се може користити само у верзији протокола 1.1. У случају да сервер не може да изврши XPath израз вратиће одговарајућу грешку.

Клијент приликом слања поруке може дефинисати и максималан број записа које жели да добије од сервера. Ово је одређено параметром *maximumRecord*. Сервер не сме вратити већи број записа од оног броја који је клијент специфицирао. Клијент такође дефинише позицију првог записа (параметар *startRecord*) који ће бити враћен из скупа пронађених записа. Овај параметар омогућава клијенту да преузме део по део пронађених записа, а не цео скуп.

Сервер по пријему *SearchRetrieveRequest* поруке процесира упит који је добио од клијента и враћа *SearchRetrieveResponse* поруку која као обавезан параметар садржи ознаку верзије протокола и број пронађених записа. Уколико нема записа који одговарају задатом упиту сервер ће вратити вредност 0 за број погодака.

Сервер може имплементирати опцију чувања пронађених резултата и тада се сваком скупу резултата мора доделити јединствено име.

Приликом слања одговора сервер може клијенту послати идентификатор скупа резултата што је дефинисано параметром *resultSetId*. На овај начин је омогућено да клијент и после одређеног времена може поново приступити скупу погодака које је раније пронашао. Клијент може дефинисати временски период у ком би одређени скуп резултата требао да буде доступан, али сервер не мора то да прихвати и може сам дефинисати период доступности скупа резултата. Период доступности представљен је параметром *resultSetIdleTime*.

Основни параметар *SearchRetrieveRequest* поруке је параметар *records* који представља скуп појединачних записа. Сваки запис садржи информацију о XML шеми по којој је запис форматиран, затим позицију тог записа у скупу пронађених погодака као и сам запис. У верзији стандарда 1.2 додат је још један параметар (*recordIdentifier*) који сваком запису додељује јединствени идентификатор који се може користити у наредним упитима и операцијама.

У случају да сервер не може да одговори на клијентов захтев, порука коју сервер шаље ће садржати параметар *diagnostics* који служи да се у њега сместе поруке о грешкама. Постоји листа дефинисаних грешака и за сваку грешку постоји идентификатор грешке, опис грешке и додатно објашњење које се може приказати крајњем кориснику.

2.4.1.2 Scan сервис

Овај сервис омогућава клијенту да за одређени индекс по ком се претражује добије скуп вредности тог индекса и евентуално уколико сервер то подржава за сваку вредност из индекса може се дати број погодака који би се добио уколико би се у претрази користио тај индекс са том вредношћу. На пример, овај сервис омогућава да погледамо које све наслове књига имамо у нашој бази података а да при томе не морамо знати конкретан наслов књиге. *Scan* сервис се обично користи у комбинацији са *SearchRetrieve* сервисом како би се избегло непотребно претраживање и добијање превеликог опсега података. Што значи да корисник прво позове сервис *Scan* за одређени индекс, добије листу могућих вредности за тај индекс и онда позове сервис *SearchRetrieve* са упитом у коме је дефинисана једна од пронађених вредности.

ScanRequest порука коју клијент шаље серверу састоји се од обавезног параметра *scanClause* који садржи информацију о индексу који се претражује. Вредност параметра *scanClause* дата је у облику CQL

упита који ће бити описан у четвртом поглављу. Поред овог параметра клијент може дефинисати и параметар *maximumTerms* који означава макисималан број вредности које клијент може прихватити.

Сервер враћа поруку *ScanResponse* која садржи листу вредности које су пронађене у датом индексу што је представљено параметром *terms*. Параметар *terms* се састоји од секвенце параметара *term*, при чему сваки параметар *term* описује појединачну вредност која је пронађена у датом индексу. Параметар *term* садржи информацију о вредности и позицији те вредности у листи осталих вредности за изабрани индекс. Вредности за позицију су предефинисане и позиција може бити на почетку листе, на крају листе и слично. Параметар *term* садржи и информацију о броју погодака у којима се посматрана вредност налази за задати индекс.

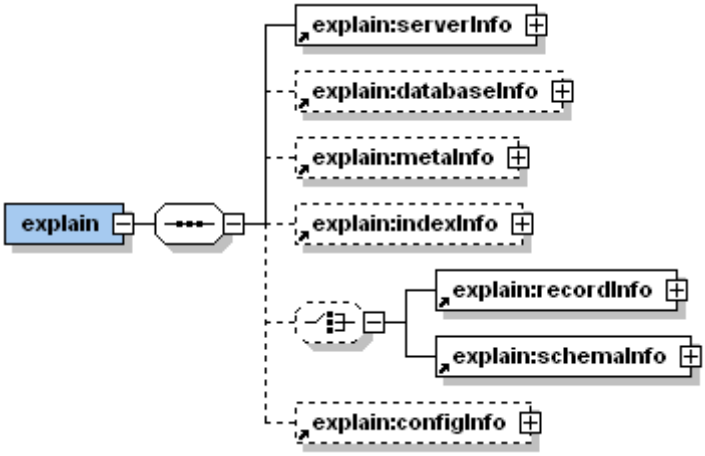
2.4.1.3 Explain сервис

Овај сервис омогућава клијенту да добије информацију о елементима стандарда које сервер подржава, пошто сервер не мора да имплементира све детаље стандарда. *ExpalainRequest* порука коју клијент шаље је веома једноставна и поред обавезног параметра који означава верзију протокла не мора имати ни један други параметар.

Одговор сервера у виду *ExpalainResponse* поруке садржи обавезан параметар *record* који представља XML документ у складу са *ZeeRex* спецификацијом [18].

ZeeRex (*Z39.50 Explain, Explained and Re-Engineered in XML*) спецификација служи за опис функционалности сервера који подржава SRU стандард и клијент може користити добијену спецификацију сервера за самоконфигурисање корисничког интерфејса. Односно на тај начин ће спречити корисника да користи неке од елемената стандарда које посматрани сервер не подржава.

Првобитна намена *ZeeRex* спецификације била је опис сервера који користе Z39.50 стандард, међутим убрзо је уочено да се ова спецификација може користити и за опис других протокола за преузимање и претраживање информација. Коренски елемент *explain* дефинисан *ZeeRex* XML шемом дат је на слици 2.2 која представља графички приказ XML шеме генерисан у алату *Altova XMLSpy* [19].

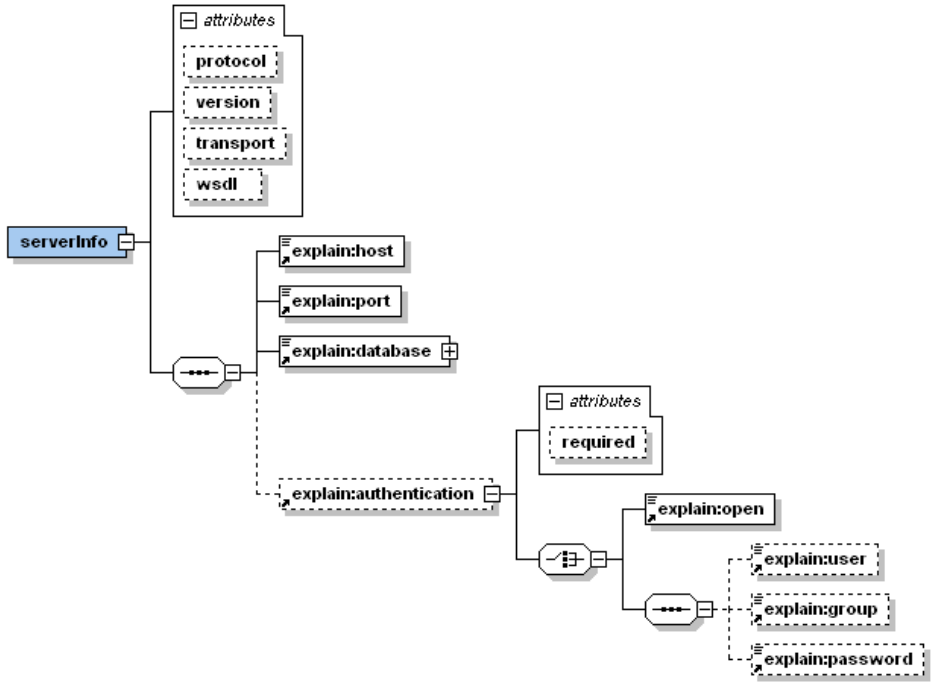


Слика 2.2 Коренски елемент *explain*

Коренски елемент састоји се од шест поделемената:

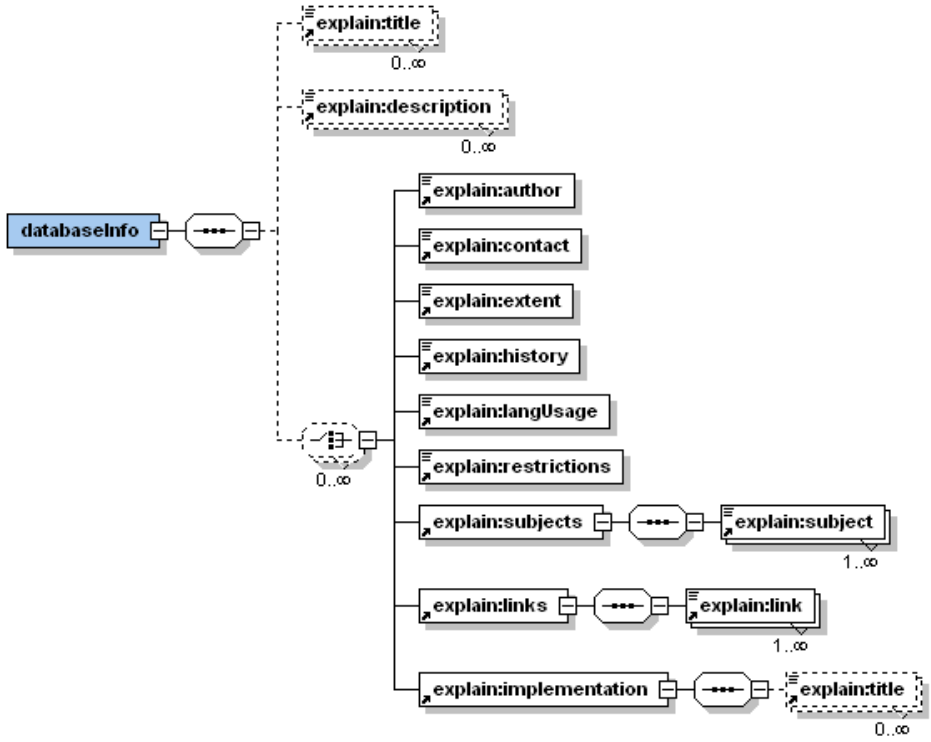
- *serverInfo,*
- *databaseInfo,*
- *metaInfo,*
- *indexInfo,*
- *recordInfo* или *schemaInfo* што зависи од протокола који се описује,
- *configInfo.*

Елемент *serverInfo* садржи основне информације неопходне да се успостави комуникација са сервером. Односно дата је адреса сервера, порт, назив базе која се претражује и подаци потребни за аутентификацију клијента. Овај елемент садржи и атрибут *protocol* који дефинише који протокол се користи у комуникацији са сервером. Предефинисане вредности за овај атрибут су Z39.50, SRU, SRW и SRU/W. Уколико је изабрана вредност SRU/W то значи да су обе верзије сервиса доступне на истој URL адреси. Графички приказ овог елемента дат је на слици 2.3.



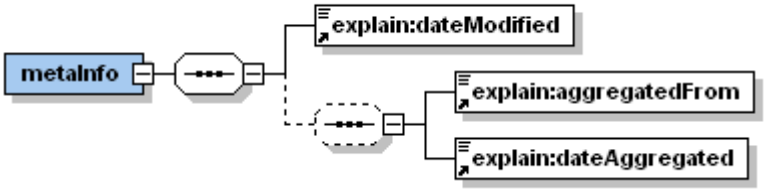
Слика 2.3 Елемент *serverInfo*

Елемент *databaseInfo* садржи информације о бази података која се претражује. Те информације обухватају назив базе, податке о особи која је задужена за одржавање базе и кратак опис базе намењен крајњем кориснику. Може се дати и информација о језицима који су заступљени у бази или информације о рестрикцијама над базом, у смислу да је база доступна само у одређеном периоду и слично. Графички приказ овог елемента дат је на слици 2.4.



Слика 2.4 Елемент *databaseInfo*

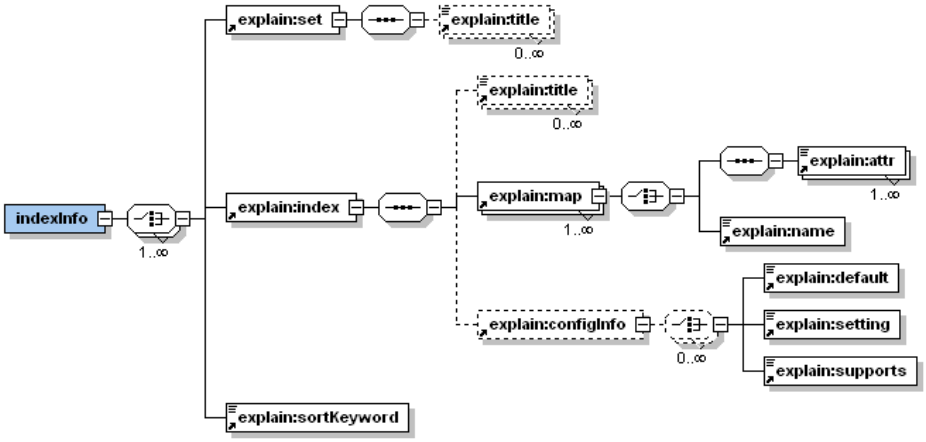
Елемент *metaInfo* служи за опис самог *ZeeRex* XML документа. Овај елемент садржи информације о датуму када је документ креиран или модификован и на који начин је добијен садржај. Садржај документа се може добити претраживањем посебних *Explain* база података које садрже само *ZeeRex* XML документе и служе за претраживање сервера који подржавају Z39.50 и SRU протоколе. Графички приказ овог елемента дат је на слици 2.5.



Слика 2.5 Елемент *metaInfo*

Елемент *indexInfo* садржи информације о префиксима који се могу користити у претраживању. Овај елемент се разликује у зависности да ли се описује сервер који користи Z39.50 протокол или сервер који

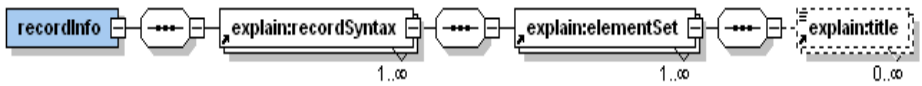
користи SRU протокол. У случају да се описује Z39.50 сервер овај елемент садржи скуп свих атрибута који се могу користити у претрази и такође је дефинисано ком скупу атрибута припадају дати атрибута. На пример, може бити дефинисано да сервер подржава скуп атрибута *bib-1* [20] и из тог скупа подржава атрибут *title* и *author*. Када је у питању SRU сервер потребно је дефинисати *ContextSet* који сервер подржава и које све префиксе подржава из посматраног *ContextSet*-а. Разлике између упитних језика које су дефинисане Z39.50 стандардом и SRU стандардом биће детаљније описане у четвртом поглављу. Графички приказ овог елемента дат је на слици 2.6.



Слика 2.6 Елемент *indexInfo*

У зависности да ли *ZeeRex* XML документ описује Z39.50 сервер или SRU сервер могу бити дефинисани респективно или *recordInfo* елемент или *schemaInfo*. Графички приказ елемената *recordInfo* и *schemaInfo* дати су респективно на сликама 2.7 и 2.8. Према томе за опис Z39.50 сервера користиће се *recordInfo* елемент који садржи информације о синтакси у којој ће се запис преузимати. Библиотечки записа може бити представљен, на пример у MARC21[21] или UNIMARC [22] формату или било ком другом формату који је дефинисан од стране Конгресне библиотеке. Када *ZeeRex* XML документ служи за опис SRU сервера тада се користи елемент *schemaInfo* и садржи информације о XML шемама по којима могу бити форматирани записи које ће сервер слати клијенту као одговор на упит. SRU сервер, на пример, може дефинисати да подржава размену записа по DublinCore [23], MARCXML шеми [24] и UNIMARC XML шеми [25], док клијент у

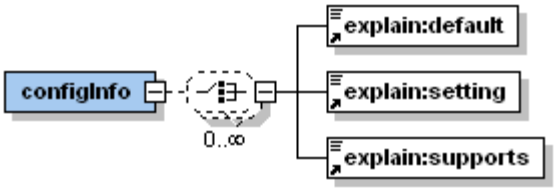
свом захтеву може специфицирати по којој конкретно шеми жели да добија записе.



Слика 2.7 Елемент *recordInfo*

Слика 2.8 Елемент *schemaInfo*

Елемент *configInfo* дефинише конфигурацију сервера, односно који све концепти дефинисани стандардом су подржани. На пример, SRU сервер може подржавати само *SearchRetrieve* сервис, док не мора подржавати *Scan* сервис. Слично, може подржавати само упите са једним префиксом, а да не подржава упите са логичким операторима, или да не подржава именоване скупове резултата итд. Односно овај елемент служи да се дефинише које све параметре, који су стандардом дефинисани као опциони, сервер подржава. Овај параметар има три поделемента којим су описани елементи стандарда који се сматрају подразумеваним уколико другачије није дефинисано, али се ове вредности могу заменити уколико клијент другачије дефинише (елемент *default*), затим описују се одређена подешавања чије вредности се не могу заменити вредностима које клијент предложи (елемент *setting*) и дат је опис елемената који су подржани од стране сервера (елемент *supports*). Графички приказ овог елемента дат је на слици 2.9.



Слика 2.9 Елемент *configInfo*

2.4.2 Обрада грешака

Све грешке које се десе приликом комуникације клијента и сервера описане су стандардом и представљене су елементом *diagnostics*. Сервер раликује две врсте грешака: критичне и некритичне. Када се деси критична грешка сервер ће вратити само поруку о грешци и неће ни покушати да врати поготке. Пример критичне грешке је када клијент пошаље упит чија синтакса није исправна и тада сервер не може ништа да уради осим да врати поруку о грешци. Некритичне грешке су оне које, када се десе, могу да утичу на резултат претраге али сервер може да настави са радом. Пример овакве грешке је када клијент пошаље синтаксно исправан упит али упит садржи неке елементе које сервер не подржава. Сервер тада може одлучити да настави са радом а да одређене делове упита изостави приликом претраге. Некритичне грешке се даље могу поделити у две подгрупе: *surrogate* и *non-surrogate*. Грешке типа *surrogate* представљају замену за запис. Ове грешке се могу десити на пример када сервер подржава именоване скупове резултата и уколико клијент после неког времена затражи записе из тог скупа може се десити да је у међувремену неки од записа избрисан из базе и тада ће сервер вратити све записе који су и даље доступни а уместо оних који су избрисани вратиће грешку типа *surrogate*. Грешке типа *non-surrogate* служе да означе да је дошло до одређене грешке али да је сервер ипак успео да врати погодке који можда нису у складу са клијентовим очекивањима. На пример може се десити да је клијент захтевао да резултати претраге буду сортирани по одређеном критеријуму, а да сервер то није могао да изврши.

Грешке које сервер може да врати дефинисане су стандардом и свака грешка одређена је са три параметра: *uri*, *details* и *message*. Параметар *uri* представља јединствени идентификатор грешке. На пример ако је вредност *uri* параметра `info:srw/diagnostic/1/10` то значи да је грешка дефинисана SRU стандардом под редним бројем 10. У листи регистрованих грешака може се видети да грешка број 10 значи *'синтаксно неисправан упит'*. Параметар *details* може садржати додатне информације које клијент може да искористи како не би поново направио исту грешку. Обично је садржај овог параметра дефинисан самом врстом грешке. На пример уколико се деси грешка где параметар *uri* има вредност `info:srw/diagnostic/1/38`, што значи да је клијент формирао упит који има превише операнада, сервер као вредност параметра *details* може вратити максималан број дозвољених операнада у упиту. Параметар *message* садржи опис

грешке и евентуално предлог како да се та грешка избегне и намењен је крајњем кориснику. Једино је параметар *uri* обавезан док остали параметри су опциони.

Поглавље 3

Библиотечки систем БИСИС

Библиотечки систем БИСИС развија се на Природно-математичком факултету и Факултету техничких наука у Новом Саду од 1993. године и до сада је имао неколико верзија. Тренутно је актуелна четврта верзија система, која је базирана на XML технологијама.

У оквиру развоја четврте верзије система БИСИС могу се издвојити одређене функционалне целине:

- коришћење библиотечке грађе, односно циркулација,
- каталогизација библиографских записа,
- индексирање и претраживање библиографских записа,
- преузимање библиографских записа путем протокола Z39.50,
- креирање каталошких листића,
- генерисање статистичких извештаја.

Коришћење библиотечке грађе. Подсистем за коришћење библиотечке грађе резултат је истраживања у оквиру магистарске тезе (Тешендић, 2007), а детаљи везани за пројектовање и имплементацију овог подсистема дати су и у монографији (Тешендић и Сурла, 2007) као и у радовима (Tešendić et al., 2009; Milosavljević and Tešendić, 2010). Подсистем за коришћење библиотечке грађе задовољава све информационе захтеве електронског пословања факултетских, градских и специјалних библиотека. То се посебно односи на рад како у интранет окружењу тако и на Интернету. Овај подсистем омогућава следеће функционалности:

1. унос податке о кориснику библиотеке,
2. унос податке о чланству,
3. унос податке о коришћењу библиотечке грађе,
4. креирање опомена корисницима библиотеке,
5. кастомизација корисничког интерфејса.

Каталогизација библиографских записа. Развој едитора за каталогизацију у оквиру четврте верзије система БИСИС описан је у магистарској тези (Димић, 2007) као и у монографији (Димић и Сурла,

2007) и радовима (Dimić and Surla, 2009; Dimić et al., 2010). Едитор за обраду библиографске грађе реализован је по угледу на савремене графичке апликације али уводи и специфичности које се односе на библиотечку област. Едитор за обраду библиографске грађе заснован је на XML технологијама чиме је постигнута независност од библиотечког формата по ком се врши обрада библиогарфске грађе и независност од софтверског система у који се едитор интегрише.

Индексирање и претраживање библиографских записа. У четвртој верзији система БИСИС креиран је текст сервер за индексирање и претраживање библиографских записа коришћењем програмског пакета *Lucene* [25]. Опис овог текст сервера за индексирање и претраживање библиографских записа дат је у раду (Milosavljević et al., 2010). Због потреба ове дисертације у одељку 3.2 биће детаљније описана ова компонента.

Преузимање библиографских записа путем протокола Z39.50. У циљу да се обезбеди комуникација система БИСИС са другим системима путем неког од стандардних начина комуникације у оквиру развоја четврте верзије система БИСИС али и као део истраживања на магистарској тези (Боберић, 2007) развијена је клијентска апликација која обезбеђује претраживање и преузимање библиографских записа од других система путем протокола Z39.50. Детаљи везани за пројектовање и имплементацију ове апликације дати су у шестом поглављу.

Креирање каталошких листића. Креирање разних врста каталошких листића на основу библиографских записа важан је сегмент у развоју библиотечких система и овај аспект је значајно унапређен у четвртој верзији система БИСИС. У монографији (Рађеновић и др, 2006) и раду (Rađenović et al., 2009) предложено је решење за креирање каталошких листића у облику софтверског пакета *Report.* У оквиру овог пакета генерисање каталошких листића извршено је формирањем темплејта у алату FreeMarker [26]. Формирањем темплејта на основу којих се врши генерисање каталошких листића избегнуте су измене апликације приликом додавања нових листића, јер се додавање нових листића и концепата врши додавањем нових темплејта односно изменом постојећих темплејта. На овај начин могуће је ажурирати каталошке листиће без поновног компајлирања изворног кода.

Генерисање статистичких извештаја. Једна од функционалности библиотечког информационог система је свакако и могућност приказивања и штампања различитих статистичких извештаја. Постоје

различити извештаји који су библиотекарима потребни у свакодневним пословима. Подсистем за генерисање статистичких извештаја је прилагођен потребама сваке појединачне библиотеке. За сваку библиотеку постоји посебан конфигурациони фајл у коме се могу унети специфичности везане за извештавање у тој библиотеци. Поред дефинисања различитих врста извештаја могу се дефинисати временски периоди за који се генеришу извештаји. У циљу ефикаснијег рада библиотекара генерисање извештаја је предвиђено да се извршава у периоду када је систем мање оптерећен, на пример ноћу. Детаљи везани за имплементацију овог подсистема дати су у раду (Boberić and Milosavljević, 2009).

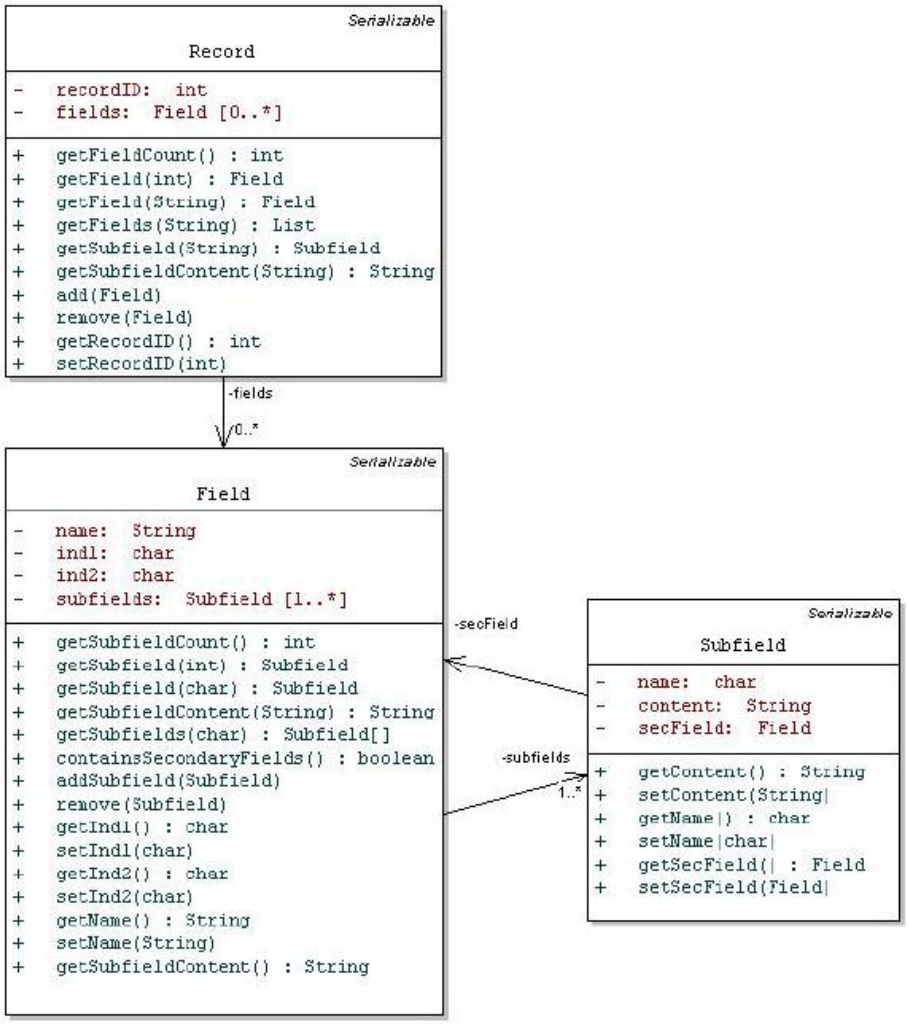
У наставку су детаљније објашњени објектни модел библиографског записа и текст сервер система БИСИС, јер су те две компоненте битне у архитектури софтверске компоненте која је описана у овој дисертацији. Наиме, та компонента комуницира са текст сервером система БИСИС и подаци које добија од система БИСИС су представљени описаним објектним моделом библиографског записа.

3.1.ОБЈЕКТНИ МОДЕЛ БИБЛИОГРАФСКОГ ЗАПИСА У СИСТЕМУ БИСИС

У четвртој верзији система БИСИС каталогизација библиографске грађе се врши употребом UNIMARC формата. У циљу имплементације самог подсистема за каталогизацију формиран је објектни модел једног записа. Основна улога објектних модела библиографских записа јесте репрезентација података о запису у апликацији. Подаци о запису се могу учитати у објектни модел из XML докумената. Такође, предвиђено је да се подаци о запису складиште у те XML документе. Да би се извршило правилно мапирање података из XML докумената на објектни модел, и обрнуто, потребно је да објектни модел одговара моделу XML шеме према којој се креирају XML библиографски записи.

На слици 3.1 дат је дијаграм класа који описује објектни модел записа. Ова слика је преузета из монографије (Димић и Сурла, 2007) где је и детаљно описан и сам објектни модел записа.

Класа `Record` моделира библиографски запис. Запис је одређен својим идентификационим бројем који је садржај атрибута `recordId`. Сваки запис се састоји од више поља. Класа `Record` садржи одговарајуће операције за додавање, селектовање односно брисање поља из записа.



Слика 3.1 Објектни модел библиографског записа

Класа `Field` представља поље записа. Поље је одређено својим именом које је представљено атрибутом `name`. Поред тога поље садржи и атрибуте `ind1` и `ind2` које представљају вредности индикатора.

Поље се може састојати од више потпоља. Свако потпоље представљено је класом `Subfield`. Класа `Subfield` има атрибут који означава име потпоља (`name`) и атрибут `content` у коме се налази вредност тог потпоља. Потпоље може имати и листу поља и тада се та поља називају секундарна.

3.2 ТЕКСТ СЕРВЕР СИСТЕМА БИСИС

Текст сервер је софтверски систем чија је основна функција да омогући ефикасно претраживање текстуалних докумената. У циљу омогућавања претраживања текст сервер генерише интерну структуру података која се назива текст индекс. Процес креирања текст индекса назива се индексирање. Текст сервер омогућава претраживање по речима у оквиру документа и стога је приликом индексирања потребно извршити анализу полазног документа и његово растављање на речи. У зависности од врсте докумената који се индексирају текст сервери се могу поделити на оне које претражују документе на Интернету, оне које претражују документе у фајл систему и на оне који претражују документе смештене у бази података. Неки од *open-source* текст сервера су: Xapian, Swish++, Senga, ht://Dig, MG, Isearch, Lemur, Oasis и они су описани у раду (MacFarlane, 2003).

Основни елемент сваког библиотечког информационог система је свакако и текст сервер и тренутно је на тржишту велики број система који на различите начине решавају проблеме индексирања и претраживања. Један од текст сервера који се често користи за индексирање и претраживање докумената је *Zebra* [27]. *Zebra* је софтверски продукт компаније *Index Data* који омогућава индексирање структуираних докумената (e-mail, XML, MARC формат) и претраживање докумената путем протокола Z39.50 и SRU. Пошто је на почетку свог развоја *Zebra* била намењена индексирању библиографских записа, велики број библиотечких система је искористио *Zebru* за свој текст сервер и неки од тих система су описани у радовима (Baranov et al., 2000; Watkins, 2003; Leveling and Veiel, 2007).

Такође, интересантан систем је *DSpace* [8] који представља централни репозиторијум за електронска документа и у њему је стављен акценат на архивирању различитих врста докумената. Детаљан опис овог система дат је у раду (Tansley et al., 2003) али битно је истаћи да овај систем користи *Lucene* програмски пакет и особине *Lucene*-а које су искоришћене у имплементацији тог система дате су у раду (ARD Prasad and Patel, 2005). Основна улога *Lucene*-а је у индексирању, односно архивирању различитих врста докумената, док је претраживање сведено на миминум и омогућено је претраживање само по наслову, аутору или кључној речи.

Софтверски систем *Greenston* (Witten et al., 2000) који се развија на универзитету *Waikato*, Нови Зеланд је још један систем за складиштење и претраживање библиотечке грађе и овај систем као текст сервер користи алат *MG* [29] . *Greenstone* је *open-source* софтвер, подржава *full-text* претраживање докумената, модуларан је и проширив у смислу да се може прилагодити за различите врсте докумената.

У трећој верзији библиотечког система БИСИС развијен је текст сервер (Милосављевић, 1999) који је као језик за претраживање користио подскуп језика *Dialog* [30]. Претраживање библиографских записа се изводило коришћењем префикса. Сваки префикс је једнозначно одређен називом од два знака. Садржај префикса је текст који је наведен у неком од потпоља библиографског записа дефинисаног UNIMARC форматом. Мапирање поља и потпоља библиографског записа на префиксе је прецизно дефинисано. Један префикс може да укључује садржај који је дефинисан у више различитих потпоља библиографског записа.

Развојем четврте верзије система БИСИС уочено је да каталогизатори библиотечке грађе имају потребу за софистицираније претрагом библиотечких записа како би што више сузили скуп погодака. Њихови захтеви су се односили на:

- Реконфигурисање префикса и додавање нових,
- Побољшавање перформанси индексирања и претраживања,
- Претраживање по само једном конкретном потпољу библиографског записа,
- Претраживање шифрираних потпоља,
- Претраживање по секундарним пољима,
- Могућност дефинисања позиције терма који се претражује,
- Могућност претраживања у задатом опсегу.

На основу изнетих захтева имплементиран је нови текст сервер који је базиран на програмском пакету *Lucene*. Разлози за избор *Lucene*-а су вишеструки. Пре свега, систем БИСИС је имплементиран у Јава окружењу па је због тога изабран *Lucene* који је такође имплементиран у истом окружењу. Због специфичних потреба система БИСИС веома је битно да постоји могућност кастомизације (то се у првом реду односи на модификацију лексичког анализатора) што је омогућено код *Lucene*-а. У раду (Smith, 2000) дат је преглед функционалности везаних за претраживање библиотечке грађе које би требало да задовољи један информациони систем. Програмски пакет *Lucene* пружа могућност имплементације свих поменутих функционалности. Поред тога, *Lucene*

се показао као ефикасан и користе га и многе друге апликације чији је списак дат на адреси [31].

3.2.1 Програмски пакет *Lucene*

Lucene је софтвер за индексирање и претраживање података (*Information Retrieval Library*). *Lucene* је бесплатан, *open-source* пројекат имплементиран у Јави. *Lucene* обезбеђује једноставно, али моћно језгро API функција, које захтева минимално разумевање индексирања и претраживања текста. *Lucene* омогућава да се у апликацији бавимо проблемима карактеристичним за ту апликацију, док индексирање и претраживање скрива иза једноставних API функција.

Модел података који *Lucene*, у својој интерној репрезентацији, користи за опис докумената које индексира представљен је *Lucene* документом (*Document*), при чему се сваки документ састоји од поља (*Field*). За појединачно поље обавезно се дефинише назив поља и садржај који се налази у том пољу. Назив *Lucene* поља користиће се приликом претраживања по самом индексу.

3.2.1.1 Поступак индексирања

Поступак индексирања се састоји из низа релативно комплексних операција, које се могу поделити у три главне групе, које се разликују по задацима које обављају. То су: конверзија података у текст, анализа текста и прављење индекса.

Да би се индексирали подаци помоћу *Lucene*-а прво се морају конвертовати у ASCII текст, јер је то једини формат који *Lucene* може да прихвати. Узевши у обзир ту чињеницу, јасно је да индексирање докумената у форматима као што су PDF, HTML, XML, ... подразумева проналажења начина да се извуче текст из тих формата да би се направили *Lucene* документи и њихова поља. Када је то урађено, *Lucene* прво анализира податке (текст) да би их прилагодио индексирању. Да би то постигао, дели текст у токене и изводи додатне операције на њима, на пример пребацује сва слова у мала да претраге не би правиле разлику између малих и великих слова. Такође, пожељно је отарасити се честих, а непотребних токена (стоп-речи) као што су у енглеском језику речи *a*, *an*, *the*, *in*, *on* и др. које нису битне у претрагама које се врше. Након што је унос анализиран, спреман је да се дода у индекс. *Lucene* чува улазне податке у структури познатој као инвертовани индекс.

3.2.1.2 Поступак претраживања

Приликом претраживања индекса, повратна информација је скуп резултата (*hits*) поређаних у неком редоследу. Резултати су поређани по бодовима (*score*). *Lucene* израчунава бодове за сваки документат. Резултати као такви нису одговарајући документи, него референце на одговарајуће документе. У већини апликација које приказују резултате претраге, корисници прегледају првих неколико резултата, тако да није потребно вратити све одговарајуће документе, него само првих неколико.

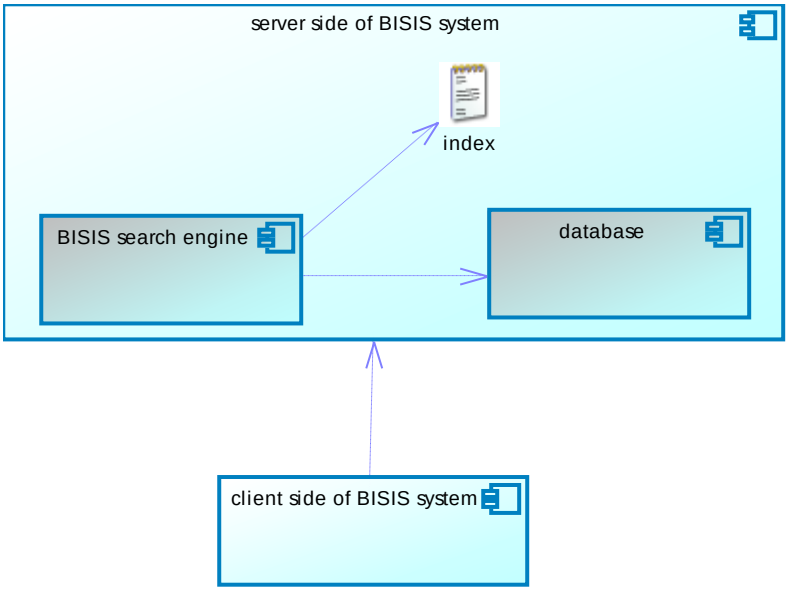
IndexSearcher је главна класа која се користи за претраживање докумената у индексу. *Lucene* омогућава неколико типова претраге и сходно томе садржи одговарајуће класе којима су дефинисани различити типови упита. Класе које се користе за креирање упита у *Lucene*-у су:

- *TermQuery* - Класа која је задужена за креирање најједноставнијег упита који се састоји само од једног операнда. Операнд се састоји од назива *Lucene* поља које се претражује и терма који се тражи.
- *RangeQuery* - Терми су у индексу поређани лексикографски, дозвољавајући ефикасно претраживање по терму у неким опсезима. *Lucene*-ов *RangeQuery* олакшава претрагу од почетног до крајњег терма, с тим да први и последњи терм могу бити укључени или искључени из претраге.
- *WildcardQuery* - Ова класа омогућава креирање упита код ког су појединачни карактери у терму који се тражи замењени специјалним знаком. Користе се два специјална знака: * замењује ниједан или више карактера, и ? замењује ниједан или један карактер. На овај начин добијају се резултати који садрже терм који се од траженог терма разликује у неколико карактера.
- *PhraseQuery* - Индекс садржи информације о позицијама терма. *PhraseQuery* користи ове информације да лоцира документат у коме су одређени изрази, на некој раздаљини једни од других. Ова класа омогућава претраживање по одређеним фразама.
- *FussyQuery* - Ова класа омогућава креирање упита којим се проналазе терми који су слични терму дефинисаном у упиту. Сличност два терма одређује се *Levensthein distance* алгоритмом.

- *SpanNearQuery* - ова класа служи за реализацију близинске претраге, односно да се креира упит у коме ће бити специфицирана удаљеност између два терма.
- *BooleanQuery* - Ова класа омогућава креирање упита који садрже логичке операторе AND, OR и NOT. Операнди у таквим упитима су представљени одговарајућим класама (*TermQuery, PhraseQuery, WildcardQuery, …*) .

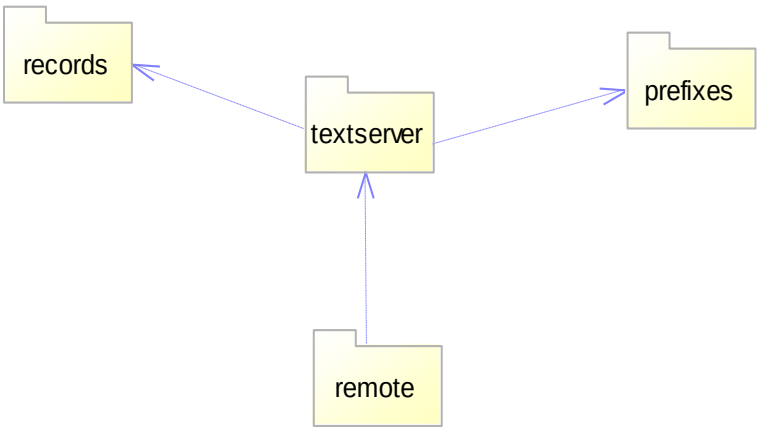
3.2.2. Индексирање у систему БИСИС

Процес индексирања у систему БИСИС је реализован на тај начин да се у самој бази чувају библиотечки записи док се у индексу налазе само оне вредности по којима се претражује. То значи да клијент прво приступа текст серверу, који након што је извршио претрагу из базе добавља записе и приказује их кориснику. Односно када корисник жели да обради нови запис тај запис се прво смешта у базу а након тога се шаље на индексирање. Индексирање се одвија у позадини како би корисник могао несметано да настави да ради. На овај начин су побољшане перформансе система јер корисник мора да сачека само док се сними запис у базу, а индексирање се асинхроно одвија на серверу и оно је независно од клијента. Ова архитектура система БИСИС приказана је на слици 3.2.



Слика 3.2. Архитектура система БИСИС

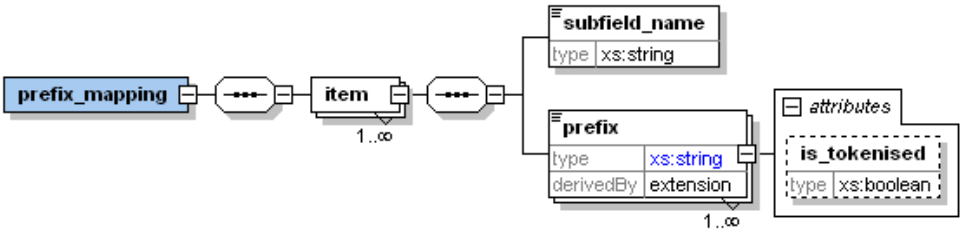
Архитектура текст сервера у систему БИСИС приказана је на дијаграму пакета (слика 3.3.).



Слика 3.3. Архитектура текст сервера у систему БИСИС

Пакет *records* садржи објектни модел библиотечког записа по UNIMARC формату који је укратко описан у одељку 3.1.

Пакет *prefixes* садржи објектни модел којим је описано мапирање између префикса по којима се претражује и потпоља из UMIMARC формата. Информације о мапирању чувају се у XML документу који је инстанце XML шеме приказане на слици 3.4.



Слика 3.4. XML шема за мапирање потпоља на префиксе

Коренски елемент *prefix_mapping* се може састојати од више поделементата *item*. Сваки елемент *item* садржи поделемент *subfield_name* који представља назив потпоља из UNIMARC формата. Елемент *item* може садржати више елемената *prefix*. Елемент *prefix* садржи назив *Lucene* поља које ће се креирати приликом индексирања записа и по коме ће се касније моћи претраживати. Овај елемент има атрибут *is_tokenised* који дефинише да ли се на садржај префикса

примењује неко посебно парсирање или не. Пример дела XML документа који представља инстанцу дате XML шеме приказан је на листингу 3.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<prefix_mapping>
.............
      <item>
            <subfield_name>200a </subfield_name>
            <prefix>TI</prefix>
            <prefix>KW</prefix>
      </item>
      <item>
            <subfield_name>606a </subfield_name>
            <prefix>TN</prefix>
            <prefix>KW</prefix>
            <prefix>SB</prefix>
      </item>
      <item>
            <subfield_name>101a </subfield_name>
            <prefix>101a</prefix>
      </item>
.............
</ prefix_mapping >
```

Listing 3.1 Deo XML documenta za mapiranje

Приказани XML документ дефинише правила мапирања за три потпоља. То су потпоља 200a које по UNIMARC стандарду представља стварни наслов дела, затим потпоље 606a које представља први елемент уноса за тематску предменту одредницу и потпоље 101a који представља језик текста публикације. Из документа се види да ће, на пример, садржај потпоља 606a бити индексиран у оквиру три префикса: TN (ознака за тематску предметну одредницу), KW (ознака за кључну реч) и SB (ознака за све предметне одреднице без обзира на врсту). Индексирање у систему БИСИС је тако организовано да се могу индексирати и појединачна потпоља. На листингу 3.1 је приказан случај када се потпоље 101a мапира само на префикс који се такође зове 101a.

Овако креирани XML документ се учитава у објектни модел из кога је помоћу одговарајућих *get* метода могуће добити скуп свих префикса који припадају једном потпољу. Предност овакве архитектуре је у томе што се могу додавати нови префикси и мењати правила за мапирање у оквиру XML документа који је улазна информација у пакет *prefixes*.

Измена правила за мапирање повлачи реиндексирање индекса, што са *Lucene*-ом не траје дуго.

Пакет *remote* садржи интерфејс `RecordManager` који прописује основне операције за рад са записима, као што су:

- `public boolean add(Record rec)`- додаје нови запис у индекс,
- `public Record update(Record rec)`- врши ажурирање постојећег записа у индексу,
- `public boolean delete(int recID)`- брише запис из индекса,
- `public Record getRecord(int recID)`- враћа један конкретан запис,
- `public String lock(int recID, String user)`- закључава запис за оговарајућег корисника система,
- `public void unlock(int recID)`- откључава закључани запис.

Такође, у овој класи се налазе и методе које су намењене за претраживање и о њима ће више бити речи у следећем одељку. Овај интерфејс има две имплементације, једна која омогућава да текст сервер ради у локалу, односно да се и клијентска страна система БИСИС и серверска страна система налазе на истом рачунару. Друга имплементација овог интерфејса омогућа да се текст серверу приступи путем HTTP протокола.

Методе дефинисане интерфејсом `RecordManager` позивају одговарајуће методе из пакета *textserver*. Пакет *textserver* је у ствари задужен за рад са *Lucene*-ом и обезбеђује сву потребну функционалност за индексирање и претраживање. Предност овакве архитектуре је у томе да уколико се промени програмски пакет за индексирање и претраживање потребно је имплементирати само класе из овог пакета у складу са новоизабраним окружењем.

Класе из пакета *textserver* директно позивају методе из библиотеке *Lucene* и креирају одговарајуће документе у *Lucene* индексу. Архитектура система је таква да једном библиотечком запису одговара један *Lucene* документ. Приликом индексирања једног библиотечког записа пролази се кроз објектни модел тог записа и за свако потпоље записа се позива одговарајући *get* метод из објектног модела дефинисаног у пакету *prefixes* који враћа листу префикса који одговарају датом потпољу. Након тога потребно је направити поље у *Lucene* документу које представља уређени пар. Први елемент је назив

префикса, док је други елемент садржај одговарајућег потпоља. То конкретно значи да ако у библиотечком запису имамо потпоље 606a чији је садржај *programming language* на основу правила за мапирање тог потпоља приказаних у листингу 3.1 у *Lucene* документу ће бити креирана три *Lucene* поља: (TN, *programming language*), (KW, *programming language*) и (SB, *programming language*).

За потребе претраживања у систему БИСИС приликом индексирања укључене су и неке специфичности. Пре свега цео садржај записа се конвертује у латинично писмо, уз обавезну транслитерацију ћирилице у латиницу по правилима српског језика, у циљу да се обезбеди равноправност латиничног и ћириличног писма приликом претраживања. Затим, вредности у *Lucene* пољу су претворена у мала слова како би се при претраживању изједначила велика и мала слова. Такође за потребе претраживања приликом индексирања садржаја из неког потпоља на почетку садржаја се додаје реч *0start0* а на крају садржаја реч *0end0*. Ово је било потребно урадити како би се омогућило да корисник приликом претраживања дефинише да ли се тражени терм налази на почетку индексираног текста или на крају и на тај начин корисник има могућност да сузи скуп пронађених погодака. Један од битних аспеката индексирања у *Lucene*-у је избор одговарајућег анализатора. У систему БИСИС искоришћен је *WhitespaceAnalyser*, његов задатак је да садржај неког *Lucene* поља испарсира на токене и да при томе као сепаратор користи знак бланко и да остатак садржаја остане неизмењен. Овај анализатор је искоришћен приликом парсирања садржаја префикса који садрже бројеве и интерпункцијске знаке (на пример UDK, ISBN, ISSN број). За парсирање осталих префикса је имплементирана метода која прво избацује интерпункцијске знаке а затим се тако припремљен садржај парсира помоћу *WhitespaceAnalyser*.

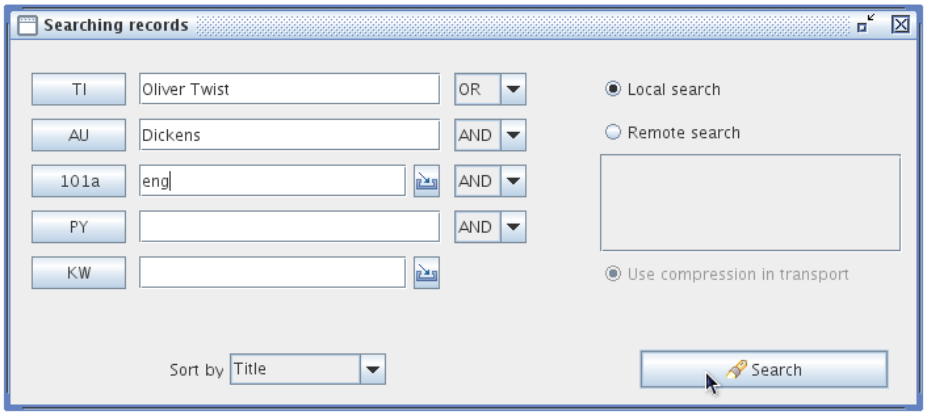
3.2.3 Претраживање у систему БИСИС

При претраживању публикација у систему БИСИС кориснику се отвара екранска форма која је приказана на слици 3.5. Кориснику су на располагању следеће могућности претраживања:

- претраживање по једном изабраном префиксу,
- претраживање фраза,
- употреба џокер знакова,
- могућност дефинисања позиције појединачног израза у оквиру једног префикса,

- повезивање више префикса са логичким операторима.

Приликом претраживања не разликују се велика и мала слова, као и латинично и ћирилично писмо, чиме је омогућено да се без обзира у ком формату (велика и мала слова) или ком писму је унет садржај у једнолинијски едитор пронађу сви записи који садрже тражену реч или фразу.

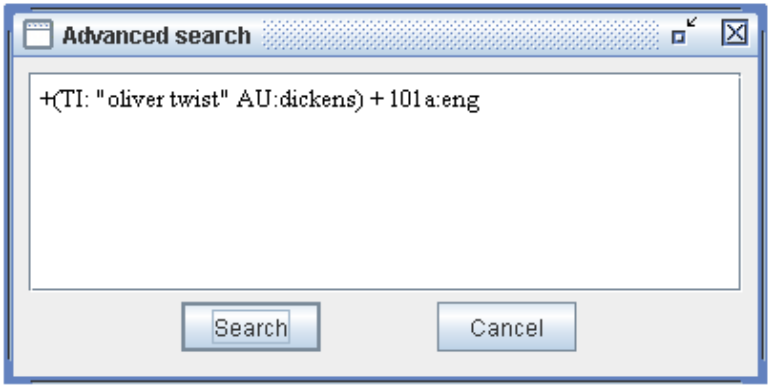


Слика 3.5 Прозор за претраживање

Поред ове екранске форме постоји и екранска форма приказана на слици 3.6 која је намењена за напредну претрагу и омогућава следеће акције:

- груписање операнада коришћењем заграда,
- проналажење погодака који су слични са унетим термом,
- претраживање у опсегу,
- дефинисање удаљености између две речи.

У овој екранској форми да би се поставио упит корисник мора користити синтаксу упитног језика која је дефинисана у *Lucene*-у. На слици 3.6 дат је пример упита постављеног у *Lucene* синтакси. Овај упит има три операнда повезана логичким операторима. Знаком + је представљен оператор AND док уколико нема никаквог знака између два операнда онда то значи да су они повезани логичким оператором OR. Такође од корисника се очекује да називе префикса уноси великим словима а садржај који претражује малим латиничним словима.



Слика 3.6 Прозор за претраживање у *Lucene* синтакси

3.2.3.1 Илустративни примери

У наредном одељку дати су илустративни примери који описују карактеристике претраживања у систему БИСИС.

Претраживање по појединачним потпољима

Пример: Пронаћи све записе који у наслову имају фразу *Oliver Twist* или је аутор *Dickens* и језик текста је енглески.

Креирани упит приказан је на слици 3.5. Као што се на слици види искоришћена су два стандардна префикса TI (наслов) и AU (аутор) и још једна додатан префикс 101a. По UNIMARC стандарду у потпољу **a** поља **101** налази се информација о језику тексту, из тих разлога је креиран овај префикс **101a** који садржи само вредности из овог потпоља. Могућност претраживања по појединачним пољима веома је значајна за рад каталогизатора јер они на тај начин могу значајно да сузе скуп погодака, што посебно долази до изражаја када се ради о библиотеци са већим фондом. Исти овај упит приказан у *Lucene* синтакси дат је на слици 3.6.

Претраживање по фрази

У систему БИСИС омогућено је претраживање по одређеној фрази. Односно у текстуално поље за унос упита уноси се фраза и систем ће вратити записе који садрже наведене речи у наведеном редоследу.

Пример: Пронаћи записе који у наслову садрже израз *20000 mile under*

У циљу креирања овог упита потребно је изабрати префикс TI и унети тражени израз. У случају да је приликом обраде записа обрађивач унео за наслов *20.000 mile under*, записи са таквим називом ће бити

пронађени јер се интерпункцијски знаци не узимају у обзир осим за префиксе где они имају посебно семантичко значење као што су UDK, ISBN, ISSN број. Исти овај упит представљен у *Lucene* синтакси, који би се унео на екранску форму са слике 3. 6, изгледа овако:

```
TI:"20000 mile under"
```

Знаци навода одређују да се ради о фрази која се тражи и део су *Lucene* синтаксе.

Близинска претрага

Близинска претрага омогућава кориснику да дефинише удаљеност између две речи у неком изразу и креирање оваквог упита у систему БИСИС је могуће ако се користи само екранска форма за напредно претраживање (слика 3.6).

Пример: Пронађи записе у којима наслов садржи речи *values* и *change* и између њих постоје још три речи.

Да би се поставио овај упит потребно је унети текст `TI:"values change"~3`, односно мора се креирати фраза што омогућавају наводници и додати још специјалан знак ~ и иза њега број који означава растојање између две речи. Као резултат оваквог упита добијају се записи који у наслову имају на пример следећу фразу: *Values, attitudes and behaviour change.*

Претраживање употребом џокер знакова

Приликом претраживања кориснику су на располагању специјални знаци који имају одређено значење. У наставку су дати примери употребе ових знакова у систему БИСИС.

Специјални знаци * и ?

Знак "*" замењује нула или више знакова, а знак "?" највише један знак.

Пример: Наћи записе код којих је датум издавања у периоду од 1990 до 1999.године.

Да би се креирао овај упит потребно је изабрати префикс PY и унети израз 199? на овај начин обухваћене су све године у задатом периоду јер упитник замењује било коју цифру од 0 до 9. Исти упит приказан у *Lucene* синтакси изгледа: `PY:199?`

Специјални знак ~

У појмовима за претраживање може се наћи и специјални знак тилда (~) и то као први или као последњи знак. Уносом тилде испред речи која се тражи претражује се садржај префикса који почиње том речју. Слично, тилда на крају речи која се претражује омогућава претраживање садржаја префикса који се завршава том речју. Овај знак се не може користити у овом значењу ако користимо екранску форму приказаној на слици 3.6, јер тада она има другачије значење у *Lucene* синтакси и користи се за близинску претрагу.

Пример: Наћи записе чији наслов почиње са речју *love*.

Да би се креирао овај упит потребно је изабрати префикс TI и да би се истакао услов да наслов почиње са датом речју испред речи *love* налази се знак (~). У овом случају добијају се записи који у наслову имају следеће фразе: *Love and Ehile*, *Love is Eternal*, *Love is not enough* и слично. Као што се види реч *love* је увек на почетку фразе. Исти упит приказан у Lucene синтакси изгледа:

```
TI:"0start0 love"
```

Као што се види приликом директне употребе *Lucene* синтаксе се не може користити знак ~ јер он у том случају има специјално значење због тога се уноси израз *0start0* који је у процесу индексирања додат на почетак садржаја наслова. Тако да се у овом случају тражи фразу *0start0 love* и на тај начин обезбеђује да реч *love* буде на почетку наслова.

Поглавље 4

Упитни језици

Основни циљ истраживања у овој дисертацији је да се креира софтверска компонента која ће прихватати различите врсте упита дефинисане одређеним стандардима и онда те упите трансформисати у упит који је подржан од постојећег библиотечког система. Идеја је да се сви упити које ова компонента прима трансформишу у упит који је дефинисан једним општим упитним језиком. Када се добије упит дефинисан општим упитним језиком потребно је још дефинисати трансформацију која ће тај упит превести у упит који је подржан од стране постојећег система.

У овом поглављу описана су два најчешће коришћена упитна језика за претраживању и преузимању података. То су упитни језик типа-1 дефинисан Z39.50 стандардом као и CQL упитни језик дефинисан SRU стандардом. Детаљном анализом оба упитна језика утврђено је да упитни језик типа-1 представља подскуп упитног језика CQL и дат је предлог мапирања упита дефинисаног упитним језиком типа-1 у упит дефинисан CQL-ом. Односно у имплементацији софтверске компоненте која је описана у шестом поглављу усвојен је CQL упитни језик као општи упитни језик. Из тог разлога је у овом поглављу дат и предлог мапирања CQL упитног језика у *Lucene* упитни језик који се користи у имплементацији текст сервера система БИСИС.

4.1 Z39.50 УПИТНИ ЈЕЗИЦИ

Z39.50 стандардом дефинисано је више различитих типова упитних језика и у овом поглављу су објашњени неки од типова. У пракси се највише користи упитни језик типа-1, јер је он обавезан приликом имплементације Z39.50 протокола. Неки од Z39.50 сервера такође подржавају упитни језик типа-101, који представља уопштење упитног језика типа-1. Упитни језик типа-101 је независтан од верзије имплементације Z39.50 протокола, тренуто су актуелне верзија 2 и верзија 3, док упитни језик типа-1 је стриктно везан за верзију 3. У досадашњем истраживању на овом раду нису пронађени Z39.50

сервери који пружају могућност претраживања помоћу осталих типова упитних језика.

Стандард дефинише употребу шест типова упитних језика од који нису сви дефинисани самим стандардом Z39.50:

1. Тип-0 се може користити ако постоји претходни договор ван оквира стандарда.
2. Тип-1 стандардом Z39.50 је дефинисан као обавезан тип упитног језика**.**
3. Тип-2 је упитни језик дефинисан у стандарду ISO8777.
4. Тип-100 је упитни језик познат под називом *Common Command Language* и није дефинисан у оквиру стандарда Z39.50.
5. Тип-101 представља проширење упитног језика тип-1.
6. Тип-102 (*Ranked List Query*) треба да буде дефинисан у наредним верзијама овог стандарда

4.1.1 Z39.50 упитни језик типа -1

Синтакса упитног језика типа-1 дата је помоћу Бакус-Наурове форме и има следећу структуру:

```
RPN-Query ::= Argument | Argument + Argument  Operator

Argument ::= Operand | RPN-Query

Operand ::= AttributeList + Term | ResultSetId |  Restriction

Restriction ::= ResultSetId + AttributeList

Operator ::= AND | OR | AND-NOT | Prox
```

Упит дефинисан овим упитним језиком се може састојати од једног операнда или више операнада повезаних логичким операторима. За операнд постоји више могућности:

- *AttributeList + Term* - у овом случају операнд представља скуп погодака који се добија применом скупа одређених атрибута и одговарајуће вредности терма за те атрибуте
- *ResultSetId* - овај операнд означава именовани скуп резултата који је добијен у некој ранијој претрази
- *Restriction* - овај операнд представља скуп резултата који су добијени тако што је на неки именовани скуп примењена рестрикција са одређеним атрибутима. На пример, ако је дат

неки именовани скуп резултата *скуп1* у коме се реч Хамлет појављује у:

a) наслову
b) наслову и у аутору
c) наслову, аутору и одредници

тада ће скуп резултата који се добијају после примене рестрикције са атрибутом *аутор* обухватати само слогове који припадају случају b) и c).

Атрибути асоцирани с термом који се тражи припадају одређеном скупу атрибута. Дефиниција скупа атрибута је регистрована, тј. додељен јој је јединствен, глобално препознатљив идентификатор скупа атрибута. Сваки атрибута представља уређени пар, где је први елемент пара тип атрибута, а други елеменат конкретна вредност за тај тип атрибута. За сваки тип атрибута постоје предефинисане вредности које атрибут може имати. Улога јединственог идентификатора скупа атрибута је да омогући разликовање оваквих уређених парова који могу имати исте вредности али припадати различитим скуповима атрибута.

За различите системе развијени су и различити скупови атрибута. Тако на пример, постоје следећи скупови атрибута: STAS-1 (*Scientific and Technical Attribute Set*), који је намењен за претраживање техничких и научних информација, CIMI-1 који је намењен за претраживање музејске грађе, *bib-1* који специфицира разне атрибуте који су корисни за претраживање библиографских записа и многи други који су дати на адреси [32]. У наставку је детаљније објашњен скуп атрибута *bib -1*.

Оператори који се користе при формирању упита су бинарни оператори AND, OR, AND-NOT и Prox. Оператори AND и OR су стандардни логички оператори, док оператор AND-NOT означава везу између два операнда при чему се врши негација другог операнда. Према томе ако постоје два операнда A и B тада A AND-NOT B означава све слогове који задовољавају операнд A и не задовољавају операнд B. Оператор *Prox* даје информацију о удаљености два операнда, на пример да ли су термови дефинисани у операндима у истом параграфу и томе слично.

Ако претпоставимо да имамо два операнда O_1 и O_2 и ако респективно са S_1 односно са S_2 означимо скупове који они представљају тада:

- O_1 AND O_2 - представља пресек скупова S_1 и S_2

- O_1 OR O_2 - представља унију скупова S_1 и S_2
- O_1 AND-NOT O_2 - представља разлику скупа S_1 и скупа S_2
- O_1 Prox O_2 –
  Prox оператор служи да ближе одреди везе између два операнда и параметри које *Prox* може имати су следећи:

  - *Distance*: Означава удаљеност између два операнда. На пример, ако је вредност параметра *Unit* параграф и ако је вредност параметра *Distance* нула, то означава да се оба терма наведена у оквиру операнада морају наћи у истом параграфу. Вредност за овај параметар не може бити негативна.
  - *Relation*: Означава релацију између два терма у операндима. Вредности за овај параметар могу бити мање, мање или једнако, једнако, веће, веће или једнако или неједнако.
  - *Unit*: Вредности које може имати овај параметар су на пример карактер, реч, реченица, параграф, одељак, поглавље, документ или нешто што је дефинисано ван ових могућности.
  - *Ordered flag*: Ако је постављен то значи да се терм наведен у левом операнду мора налазити пре терма наведеног у десном операнду.
  - *Exclusion flag*: Ако је овај параметар постављен то значи да треба да се примени негација свих параметара наведених за оператор *Prox.*

  Према томе O_1 *Prox* O_2 је подскуп од скупа који се добија применом O_1 AND O_2 и важи да је O_1 *Prox* O_2 тачно, односно све вредности параметра за оператор *Prox* су задовољене. Ово важи само ако су оба операнда типа *AttributeList + Term* у супротном сервер може пријавити грешку.

Употреба *Prox* оператора биће разумљивија на конкретном примеру. Претпоставимо да *A* и *B* означавају операнде "*аутор=Андрић*" и "*аутор= Нушић*" и нека су параметри за *Prox* постављени на следећи начин:

- *Distance* је нула,
- *Relation* је једнако,
- *Unit* је параграф,
- *Ordered flag* је искључен,
- *Exclusion flag* је искључен.

Тада, скуп резултата чине слогови у коме се оба имена аутора појављују у истом параграфу. Ако у истом примеру укључимо *Exclusion flag*, тада резултат чине слогови у коме се оба имена аутора никада не појављују у истом параграфу. Ако укључимо *Ordered flag* а Exclusion flag је искључен онда резултат чине слогови у коме се име аутора *Андрић* појављује у истом параграфу, али пре имена аутора *Нушић*.

4.1.1.1 Скуп атрибута bib -1

Za мапирање атрибута ка логичком дизајну базе података задужен је сервер који имплементира Z39.50 протокол. Један од проблема код употребе скупова атрибута је да различити имплементатори Z-сервера различито интерпретирају дате атрибуте. На овај начин умањује се интероперабилност, која би требала да буде основна предност овог стандарда.

Скуп атрибута *bib-1* [20] развијен је од стране библиографске заједнице и најчешће се користи при формирању упита дефинисаних упитним језиком типа-1. *Bib-1* скуп атрибута састоји се од шест типова атрибута:

- *Use Attributes*
- *Relation Attributes*
- *Truncation Attributes*
- *Structure Attributes*
- *Completeness Attributes*
- *Position Attributes*

***Use* атрибут** - Дефинише улазне критеријуме претраге (наслов, аутор, одредница, итд). Да би се избегла различита тумачења атрибута Конгресна библиотека је дефинисала правила за мапирање *Use* атрибута на одговарајућа поља MARC21 формата [21]. Ако се на пример за *Use* атрибут изабере вредност 4, то значи да је изабрана вредност која означава наслов и та вредност се приликом претраживања тражи у следећим пољима MARC21 библиографског записа: 130, 21X -24X, 440, 490, 730, 740, 830, 840 и у потпољу $t у следећим пољима: 400, 410, 410, 600, 610, 611, 700, 710, 711, 800, 810, 811. Ознака X приликом означавања поља, као што је то у случају 21X -24X значи да ће се претраживати по свим пољима (из задатог опсега) која на почетку имају задате две цифре а трећа цифра је опциона. На конкретном примеру то би значило да ако корисник жели да пронађе књигу са насловом „*На Дрини ћуприја*“ он би изабрао вредност 4 за Use

атрибут и добиће записе које у неком од наведених поља садрже тражени израз.

***Relation* атрибут** - Дефинише релацију (на пример: мање, веће, једнако, итд) која треба да важи између изабраног Use атрибута и израза који се тражи. На пример, ако се жели истаћи да се траже књиге код којих је година издања мања од терма који је задат, користи се вредност 1 за *Relation* атрибут и она означава релацију мање.

***Truncation* атрибут** - Дефинише који део вредности критеријума претраге ће бити коришћен током претраге (почетак речи, крај речи, итд). Улога овог атрибута може се упоредити са џокер знацима који се користе у стандардним претраживањима. Овај тип атрибута дефинише да ли ће један или више карактера из терма који се тражи бити изостављени.

***Structure* атрибут** - Дефинише формат терма који се тражи, односно да ли терм представља реч, фразу, датум или нешто друго. Уколико се, на пример израз „Лелејска гора“, жели тумачити као фраза а не као листа речи које могу бити у произвољном редоследу тада ће се за *Structure* атрибут изабрати вредност 1, која означава фразу.

***Position* атрибут** - Дефинише где се у оквиру поља налази тражени термин. Атрибутом *Position* дефинише се локација терма који се тражи у оквиру поља или потпоља у којима се терм јавља.

***Completeness* атрибут** - Дефинише да ли терм за претрагу мора или не мора бити једина вредност у датом пољу/потпољу. Овај тип атрибута дефинише да ли терм који се тражи представља целокупан садржај поља, односно потпоља или не. Атрибут *Completeness* специфицира да ли се и неке друге речи осим тражених налазе у пољу/потпољу.

4.1.2 Z39.50 упит типа-100

Овај упитни језик је познат под називом *Common Command Language* (Klemperer, 1987) и дефинисан је у оквиру стандарда ANSI/NISO Z39.58 *Common Command Language for Online Interactive*. Првобитни циљ овог упитног језика био је да се омогући унос упита путем командне линије на основу чега је и добио име.

Синтакса овог типа упита је следећа:

```
CCL-Find ::= CCL-Find Op Elements |Elements

Op ::= "and" | "or" | "not"

Elements ::= '(' CCL-Find ')' |
```

```
                    Set |
                    Terms |
                    Qualifiers Relation Terms |
                    Qualifiers Relation '(' CCL-Find ')'|
                    Qualifiers '=' string '-' string
Set ::= 'set' = string
Terms ::= Terms Prox Term | Term
Term ::= Term string | string
Qualifiers ::= Qualifiers ',' string | string
Relation ::= '=' | '>=' | '<=' | '<>' | '>' | '<'
Prox ::= '%' | '!'
```

Слично као и код упитног језик типа-1, упит дефинисан овим упитним језиком се може састојати од више елемената повезаних логичким операторима. Оператори могу бити AND, OR и NOT. Поред ових оператора појављује се и оператор *Prox* који детаљније објашњава везу између два терма која се претражују. Може се дефинисати редослед речи које се претражују или удаљеност једног израза од другог.

Операнд може бити раније добијен скуп резултата који има своје име. Затим, може бити низ стрингова, без икаквих префикса, што је новина у односу на упитни језик типа-1. Код оваквог типа операнда, корисник система за претраживање наводи само изразе које жели да пронађе, али не наводи никакве префиксе. Систем који имплементира овај тип упита дефинише предефинисане префиксе и тада се претрага врши по тим префиксима. Следећа могућност за операнд је квалификовано име које је одређеном релацијом повезано са термом или новим упитом. Квалификовано име представља неки предефинисани префикс, као што је на пример име аутора, наслов и друго. Код овог типа упитног језика могуће је као операнд поставити и одређени опсег вредности, али само за одговарајућа квалификована имена, што није било могуће у упитном језику типа-1. На пример, може се поставити упит који ће пронаћи све књиге из периода од 1990 до 2000. године.

4.1.3 Z39.50 упитни језик типа-102

Други назив под којим је овај упитни језик познат је *Ranked Query List*, јер је заснован на рангирању добијених резултата. Ово је омогућено тако што се сваком операнду додели нека тежина у смислу важности тог операнда. Овај упитни језик тренутно није део Z39.50 стандарда и

постоје само предлози како би он требао да изгледа. Један од тих предлога [33] описан је у наставку.

Синтакса упитног језика дата је у наставку помоћу *Abstract Syntax Notation One* (ASN1).

```
RankedQuery ::= SEQUENCE {
needList            [1] IMPLICIT SEQUENCE OF NeedStatement,

combineNeedLists    [2] SEQUENCE {
     combinePreference [1] CHOICE {
                useAlgorithm [1] IMPLICIT NULL,
                 recommended  [2] IMPLICIT NULL,
                 serverChoice [3] IMPLICIT NULL },

     combineAlgorithm  [2] CHOICE {
                  addWeight    [1] IMPLICIT NULL,
                   ext         [2] IMPLICIT EXTERNAL}}
                              OPTIONAL,

     attributeSet      [3]   IMPLICIT AttributeSetId,

  searchOutputRequest  [4]   IMPLICIT SearchOutputRequest
                              OPTIONAL,
  clientServerInfo     [5]   IMPLICIT ClientServerInfo
                              OPTIONAL,
  serverClientInfo     [6]   IMPLICIT  ServerClientInfo
                              OPTIONAL
                     }
                   }
NeedStatement ::= SEQUENCE {
  restrictSet  [1]   RestrictSet  OPTIONAL,
  feedbackInfo [2]   FeedbackInfo  OPTIONAL,
  rQuery       [3]   IMPLICIT OperandPlusWeight OPTIONAL,
   weight      [4]   IMPLICIT IntUnit OPTIONAL
                   }

RestrictSet  ::= SEQUENCE {
  databaseNames [1]  CHOICE {
           dbExclude [1] IMPLICIT SEQUENCE OF DatabaseName,
           dbOnly    [2] IMPLICIT SEQUENCE OF DatabaseName
                     } OPTIONAL,
  query          [2]  IMPLICIT RPNQuery
                  }

FeedbackInfo       ::= SEQUENCE OF SEQUENCE {
   documentId    [1]    CHOICE {
          localDocid  [1]   IMPLICIT OCTET STRING,
    feedbackText      [2]   IMPLICIT HumanString,
    otherFeedbackInfo [3]   EXTERNAL
```

```
                        },
    relevance    [2]    IntUnit
                 }
OperandPlusWeight ::= SEQUENCE {
      operand     [1] CHOICE{
         attrTerm          AttributesPlusTerm,
         sOperand    [1]   StructuredOperand
                    },
      weight       [2] IMPLICIT IntUnit OPTIONAL,
clientServerInfo   [3] IMPLICIT ClientServerInfo OPTIONAL,
serverClientInfo   [4] IMPLICIT ServerClientInfo OPTIONAL
                   }

StructuredOperand ::= SEQUENCE {
  rqOperator      [1] IMPLICIT RQOperator,
  rqOperandList   [2] IMPLICIT SEQUENCE OF OperandPlusWeight,
 rqProximity      [3] CHOICE {
                  prox  [1] IMPLICIT ProximityOperator,
                  ext   [2] IMPLICIT EXTERNAL} OPTIONAL
                  }

SearchOutputRequest ::= SEQUENCE {
   doSearch       [1] IMPLICIT BOOLEAN,
   returnReformulatedQuery   [2] IMPLICIT BOOLEAN,
   mData           [3] IMPLICIT SEQUENCE OF SEQUENCE{
                   tagType  [1]   IMPLICIT INTEGER OPTIONAL,
                   tagValue [2]    StringOrNumeric} OPTIONAL
                   }

ClientServerInfo ::= SEQUENCE {
   reformClause      [1] IMPLICIT BOOLEAN,
    recallImportance  [2] IMPLICIT IntUnit OPTIONAL,
    reformMethod      [3] IMPLICIT EXTERNAL OPTIONAL,
   resultSetDesc     [4] IMPLICIT SEQUENCE {
       numRecordsWanted  [1] IMPLICIT INTEGER OPTIONAL,

       rsvThresholdValue [2] IMPLICIT IntUnit OPTIONAL
                       } OPTIONAL
                    }

ServerClientInfo ::= SEQUENCE {
   intUnit     [1]   IMPLICIT IntUnit OPTIONAL,
   humanString [2]   IMPLICIT HumanString OPTIONAL,
   metaData          OtherInformation OPTIONAL
                }

RQOperator ::=  SEQUENCE {
 operator      [1] CHOICE {
    rqIndep          [1] IMPLICIT NULL,
```

```
    rqAND            [2] IMPLICIT IntUnit,
    rqOR             [3] IMPLICIT IntUnit,
    rqANDNOT         [4] IMPLICIT IntUnit,
    rqHeadRelation   [5] IMPLICIT SEQUENCE {
                tagType  [1]   IMPLICIT INTEGER OPTIONAL,
                tagValue [2]   StringOrNumeric
                      },
    other            [6] IMPLICIT EXTERNAL
                    },
allowedReform [2] IMPLICIT BOOLEAN
            }
```

Упит дефинисан овим упитним језиком се састоји од више операнада, где се за сваки овај операнд може дефинисати скуп рестрикција, затим упит и тежина овог једног операнда. Под рестрикцијама се подразумева да се могу навести имена база које се желе претражити, али се исто тако могу навести и имена база података која не треба да учествују у претраживању. За сваки операнд дефинише се тежина која има вредности између 0 и 1, и њен задатак је да одреди важност овог операнда.

Операнд може бити листа атрибута или да представља структуирани операнд. У случају да операнд представља листу атрибута тада се као и код упитног језика типа-1 наводи тип атрибута и његова вредност, само што се у овом случају уведени нови типови атрибута. Атрибути који су уведени су следећи:

- ***locationInRecord*** - овај атрибут садржи информацију о делу библиографског записа који ће се претраживати, на пример поље или потпоље. Овај атрибут има сличну улогу као и *Use attribute* у упитном језику типа-1, само што у овом случају серверска страна не мора да води рачуна о мапирању префикса на конкретна поља записа.
- ***semanticClass*** - атрибут пружа информацију о значењу терма дефинисаног у оквиру операнда. На пример да ли је то корпоративно тело, датум или можда монетарна јединица.
- ***contentAuthority*** - атрибут служи да специфицира опсег вредности које операнд може имати. Опсег вредности може бити дефинисан помоћу референце на одређени стандард, на пример "NISO Z39.53-1994 - Шифарник за језике" или одређен договором између учесника у комуникацији.

- ***contentFormat*** - атрибут специфицира енкодинг који треба користити приликом обраде операнда. На пример, овим атрибутом може се специфицирати да је садржај операнда писан на арапском језику.
- ***Relation*** - овај атрибут је еквивалентан истоименом атрибуту из скупа *bib*-1, али овај атрибут може имати само вредности *мање од, мање или једнако, једнако, веће или једнако, веће и различито*.

Према томе за сваки операнд се наводи листа атрибута по којима треба вршити претраживање и терм који се претражује.

Када операнд представља структуирани операнд, тада се он може састојати од више нових операнада који су повезани одређеним операторима. Код овог типа упита нису дефинисани стандардни логички оператори AND, OR и NOT већ су дефинисани нови оператори rqAND, rqOR, rqANDNOT чија вредност може бити децималан број између 0 и 1. Ако су два операнда повезана оператором rqAND који има вредност 0 то значи да упит представља негацију првог операнда и негацију другог операнда, односно резултат упита су погодци који не задовољавају ни први али ни други операнд. Уколико оператор rqAND има вредност 1, то значи да оба операнда морају бити у потпуности задовољена, што је исто као да су операнди повезани стандардним AND оператором.

Битне разлике које се могу уочити између упитног језика типа-1 и упитног језика типа-102 су следеће:

- Тип-1 подржава само бинарне операторе, док тип-102 подржава комбинације са више од два операнда.
- Резултати добијени упитом типа-102 су рангирани на основу нивоа задовољавања постављених критеријума, док са упитом типа-1 то није случај
- У оквиру типа-102 могу се дефинисати правила, која објашњавају серверу како да интерпретира постављени упит.
- Тип-102 обезбеђује могућност дефинисања одређених рестрикција у оквиру добијеног скупа резултата. На пример, могуће је приказати само први 20 резултата, а не цео скуп.

4.2 CQL УПИТНИ ЈЕЗИК

Упитни језик који се користи за формирање упита када се користи SRU протокол, без обзира на транспортни механизам, назива се CQL (*Contextual Query Language*) [34] језик. CQL омогућава писање људских читљивих, интуитивних упита. Овај упитни језик подржава веома једноставне упите али поседује и експресивност сложенијих језика, који обезбеђују могућност постављања произвољно сложених упита. Традиционално, упитни језици се грубо могу поделити у две категорије: изразито моћни и експресивни језици који нису лако схватљиви крајњим корисницима (SQL, XQuery) са једне стране, и на другој страни интуитивни и једноставни језици, али који су недовољни за изражавање комплексних упита (на пример синтакса упита на *Google*-u). Циљ CQL-a је да се постигне комбинација једноставности и интуитивности с једне стране, али и спектар могућности које нуди Z39.50 упитни језик типа-1 с друге стране.

Пре него што се објасни синтакса CQL упитног језика потребно је дефинисати појам *Context Set*-a.

4.2.1 Context Set

CQL упитни језик уводи као концепт појам *Context Set* и одатле и потиче његов назив *Contextual Query Language*. Наиме, сви префикси по којима се може претраживати су груписани у *Context Set*-ове и ови скупови су регистровани у оквиру Конгресне библиотеке. Увођењем концепта *Context Set* омогућено је да одређене заједнице односно корисници дефинишу своје префиксе, релације, модификаторе без бојазни да ће назив њиховог префикса бити идентичан имену префикса из другог скупа. Односно, могуће је дефинисати два префикса са истим називом али да при томе они припадају различитим скуповима и самим тим имају различиту семантику.

Сваки *Context Set* има јединствени URI идентификатор и за сваки скуп се може дефинисати скраћени назив тог скупа. Скраћени назив скупа се може користити приликом формирања упита. На пример ако је дефинисан *Dublin Core Context Set* његов скраћени назив је *dc*, тада се у упиту може садржати префикс *dc.title*, што значи да се претражује по префиксу *title* који је дефинисан у оквиру *Dublin Core* скупа. Примери најчешће коришћени *Context Set*-ова са њиховим идентификаторима и скраћеним називима дати су у табели 4.1 .

Назив скупа	**URI идентификатор**	**Скраћени назив**
CQL context set Version 1.1	info:srw/cql-context-set/1/cql-v1.1	cql или srw
CQL context set Version 1.2	info:srw/cql-context-set/1/cql-v1.2	cql
Dublin Core Context Set Version 1.1	info:srw/cql-context-set/1/dc-v1.1	dc
Bibliographic Context Set Version 1.0	info:srw/cql-context-set/1/bib-v1	bib
Bath Context Set	http://zing.z3950.org/cql/bath/2.0/	bath
GILS Context Set	info:srw/cql-context-set/14/gils-v1.0	gils

Табела 4.1 Примери *Context Set*-ова

Приликом креирања новог *Context Set-a* поред дефинисања URI идентификатора и скраћеног назива скупа могу се дефинисати још и вредности за концепте који се користе у креирању упита. Концепти који постоје у CQL упитном језику, а могу се дефинисати у различитим *Context Set*-овима су:

- Индекси,
- Релације,
- Модификатори релација,
- Модификатори логичких оператора.

Према томе сваки од *Context Set*-а може дефинисати нове вредности за неки од ових концепта и при томе се не морају дефинисати вредности за све концепте. Приликом дефинисања скупова најчешће се дефинишу вредности за индексе, вредности за модификаторе релација и евентуално релације.

Подразумевани *Context Set* који се користе при формирању CQL упита је CQL *Context Set* без обзира на верзију скупа. Понекада се као

скраћени назив за CQL *Context Set* верзије 1.1 може наћи и ознака *srw* али постоји тенденција да се користи само скраћени назив *cql* за обе верзије. Уколико у упиту није експлицитно дефинисан *Context Set* за префиксе који се појављују у упиту сматра се да припадају CQL *Context Set*-у. У наставку ће детаљно бити описани CQL *Context Set* верзија 1.2 и *Dublin Core Context Set* верзија 1.1 који су и подржани у имплементацији серверске стране протокола SRU приказане у овој дисертацији у шестом поглављу.

4.2.1.1 CQL Context Set

CQL Context Set је подразумевани скуп приликом креирања упита и њиме су дефинисани индекси који имају општу примену, затим релације као и модификатори.

Индекси

Од индекса *CQL Context Set* верзије 1.2 дефинише следеће индексе:

resultSetId – овај индекс се користи кад се у упиту користе именовани скупови, тада је вредност овог индекса у ствари назив именованог скупа који је сервер креирао у неком од претходних упита. Овај индекс се може комбиновати само са релацијом '=' док са осталим релацијама поретка нема дефинисано значење. Такође, овај атрибут се не може користити са *Scan* сервисом.

allRecords – овај индекс се користи уколико је потребно вратити све записе који постоје у бази и при томе није битно који терм се тражи. Препоручена синтакса за употребу овог индекса је *allRecords=1* при чему у самој претрази терм 1 се неће узети у обзир, уместо броја 1 могао је да стоји било који други терм. Овај индекс се не може користити са *Scan* сервисом.

allIndexes – овај индекс је у верзији 1.1 CQL Context Set био дефинисан под називом *anywhere*. Ово је специјалан индекс који омогућава да се дефинише упит при чему ће дати терм бити тражен по свим индексима које је сервер дефинисао. На пример, ако сервер има дефинисана само три индекса *title*, *author* и *keyword* тада је упит *allIndexes=fish* еквивалентан са упитом *title= fish or author=fish or keyword=fish*.

anyIndexes – овај индекс је у верзији 1.1 CQL Context Set био дефинисан под називом *serverChoice*. Уколико корисник дефинише упит који садржи индекс *anyIndexes* то значи да сервер има слободу да претражи по оним индексима које је он изабере и при томе не мора приликом сваког упита користити исте индексе као подразумеване. У

колико корисник приликом креирања упита не дефинише индекс сервер ће сматрати да је дефинисан индекс *anyIndexes*.

keywords – ово је индекс који сервер дефинише и који садржај ће бити индексиран под овим индексом зависи од сервера. Према томе два различита сервера могу на различити начин да имплементира овај индекс, али обично овај индекс представља неки општи садржај или цео документ или нешто слично.

Релације

Стандардне релације поретка (=, >, <,...) су дефинисане самом синтаксом CQL упитног језика, међутим сваки *Context Set* може дефинисати и још неке додатне релације. CQL *Context Set* дефинише следеће релације:

adj – ова релација се користи приликом претраживања фраза. Односно све речи које се појављују у терму се у истом редоследу морају наћи и у запису.

all, any – ове релације се могу користити када терм садржи више речи и тада се са релацијом *all* може дефинисати да све речи морају бити пронађене у изабраном индексу али да редослед речи у индексу није битан. Слично, релација *any* дефинише да индекс може садржати било коју од датих речи. Релација *all* се може заменити одговарајућим упитом при чему сви операнди морају бити повезани логичким оператором AND, док се релација *any* може заменити одговарајућим упитом при чему сви операнди морају бити повезани логичким оператором OR.

within – ова релација се користи када је потребно дефинисати неки опсег ком може припадати терм. Посебно може бити корисна уколико се не зна тачна вредност за неки индекс али се зна опсег вредности, на пример потребно је пронаћи књигу чија је година издавања између 2002 и 2006.

encloses – ова релација се користи када се жели истаћи услов да резултати претраге укључују дати терм. На пример, уколико се претражују неки географски подаци и постоји индекс *area*, тада би упит *area encloses 45.3,19.0* где терм представља координате неке тачке, требао да врати све резултате код којих вредност индекса *area* укључује дату тачку.

Модификатори релација

Модификатори се користе да релацији дају ново значење, у циљу да се скуп резултата претраге што више сузи како би корисник добио само оне резултате који су њему релевантни. Модификатори релација се могу поделити на функционалне модификаторе и модификаторе који одређују формат терма који се тражи.

Функционални модификатори су:

stem – овај модификатор указује да би сервер приликом претраге требао да врати и све резултате који имају исти корен речи као и тражени терм.

relevant – употребом овог модификатора сервер би требао да врати све резултате који су релевантни са траженим термом. На пример ако је корисник задао упит са термом 'риба' тада би као резултат требало вратити и погодке који садрже реч 'пастрмка', 'шаран', и слично. Алгоритам којим ће се одредити релевантност дефинише сервер.

phonetic – уколико је дефинисан овај модификатор, сервер враћа све резултате код којих се тражени терм изговара на исти начин иако се можда другачије пише. Овај модификатор има смисла уколико се претражује база са подацима на неком од страних језика код којих постоји могућност да две речи које се различито пишу и имају различито значење ипак имају исти изговор.

fuzzy – употребом овог модификатора сервер враћа све погодке који садрже израз сличан траженом терму. Алгоритам за одређивање сличности дефинише сервер.

partial – овај модификатор се може користи са релацијама *within* и *encloses*. У том случају сервер би требао да врати и оне податке који делимично задовољавају дату релацију.

ignoreCase, respectCase – овим модификаторима се серверу дају упутства да ли да занемари или не, величину слова приликом претраге.

ignoreAccents, respectAccents – овим модификаторима се серверу дају упутства да ли да занемари или не, дијакритике који се могу појавити у неким писмима (на пример српска латиница, немачки алфабет и слично).

Модификатори који одређују формат терма су:

word – уколико се терм састоји од више речи треба тако и посматрати (као скуп речи).

string – уколико се терм састоји од више речи треба га посматрати као једну јединствену реч, односно као фразу.

isoDate – терм представља датум по ISO 8601 спецификацији.

number – терм представља број. Овај модификатор посебно долази до изражаја када се користе релације поретка <, >,>=,>=.

uri – терм представља URI идентификатор

oid – терм представља идентификатор објекта дефинисан ISO форматом.

substring – овај модификатор се може користити за дефинисање првог и последњег карактера који се посматра у индексу који се тражи. Односно тражени терм може бити подскуп вредности индекса по ком се претражује. Синтакса модификатора је: *substring=почетак : крај*, при чему су почетак и крај цели бројеви који означавају респективно први и последњи карактер који се посматрају у датом индексу. На пример ако је постављен упит `marc.008 =/substring="1:6" 920102`, при чему је `920102` терм који се тражи, то значи да се претражује по индексу `008` који припада `marc` *Context Set*-у али се посматра само првих 6 карактера садржаја у датом индексу.

masked – ово је подразумевани модификатор релације и означава да специјални знаци '*', '?' и '^' имају посебно значење. Знак '*' замењује нула или више карактера у терму, а знак '?' замењује тачно један карактер. Знак '^' се користи уколико се терм састоји од више речи а дата је једна од релација *all*, *any*, или *adj*. Може се појавити на почетку речи или на крају речи и означава да се реч уз коју стоји мора појавити на почетку индексираног садржаја у индексу, односно на крају. На пример ако је дата упит `dc.title all ^cat dog`, то значи да се траже они записи који у наслову садрже и реч `cat` и реч `dog` али се реч `cat` мора наћи на почетку наслова, ниједна друга реч не може бити испред ње.

unmasked – овај модификатор се користи уколико је потребно да се специјални знаци '*', '?' и '^' тумаче као и обични карактери, значи без посебног значења. Исти ефекат се може постићи уколико се испред сваког специјалног знака стави знак '\'.

regexp – модификатор означава да је дати терм представљен у виду регуларног израза и сервер би требало да поседује алгоритам који је у стању да парсира тај регуларни израз.

Модификатори логичких оператора

CQL Context Set дефинише само четири модификатора логичких оператора и ови модификатори се могу користити само са логичким оператором *Prox*.

distance – означава удаљеност између два терма

unit – дефинише формат терма. Предефинисане вредности за овај модификатор су *paragraph*, *sentence*, *word* и *element*, стим да је *word* подразумевана вредност модификатора. Сваку од ових вредности сервер може интерпретирати на себи својствен начин.

unordered – овим модификатором је дефинисано да редослед термова који се траже није битан.

ordered – овим модификатором је дефинисано да редослед термова који се траже мора бити исти као и у упиту.

4.2.1.2 Dublin Core Context Set

Dublin Core је један од најчешће коришћених скупова приликом претраживања употребом SRU протокола. Овај скуп дефинише само индексе, с тога се најчеће комбинује са *CQL Context Set*-ом. Индекси *Dublin Core Context Set* су преузети из спецификације *Dublin Core* стандарда за опис метаподатака [35]. *Dublin Core* стандард за опис метаподатака се може користити за опис различитих ресурса, и један од тих ресурса могу бити и публикације. С тога *Dublin Core* не описује везу између *Dublin Core* формата записа и MARC21 односно UNIMARC формата записа. Веза између *Dublin Core* и MARC21 формата описана је документом који је прописала Конгресна библиотека [36]. Веза између *Dublin Core* и UNIMARC формата описана је документом [37].

У наставку је дата табела 4.2 са индексима који припадају *Dublin Core* скупу као и њихова семантика када се *Dublin Core* користи за опис публикације, и одговарајућа мапирања на поља и потпоља MARC21 и UNIMARC формата библиографског записа. У првој колони дат је назив индекса дефинисан по *Dublin Core* стандарду, друга колона садржи опис, односно семантику индекса, трећа и четврта колона садрже поља и потпоља MARC21 односно UNIMARC библиографског

записа. Свако поље се састоји од три цифре иза кога следе ознаке индикатора и након тог вредност потпоља испред кога стоји знак $. Уколико су индикатори недефинисани тада уместо конкретне вредности стоји ознака #. За неке индексе наведене су само ознаке поља што значи да се приликом претраживања по тим индексима претражује по свим потпољима наведених поља без обзира на вредности индикатора.

Индекс	**Опис**	**MARC21**	**UNIMARC**
title	наслов публикације	245 00$a	200 ##$a, 200 ##$e, 517 ##$a
creator	особа или организација примарно одговорна за настанак публикације	720 ##$a при чему је вредност потпоља 720 ##$e author	700 ##$a 701 ##$a 710 ##$a 711 ##$a 200 ##$f
subject	предметна одредница	653 ##$a	610 ##$a 606 675 676 680 686
description	опис ресурса (може обухватати напомене, абстракт, садржај и слично)	520 ##$a	330 ##$a
publisher	издавач	260 ##$b	210 ##$c
contributor	особа или организација која је такође допринели настанку публикације	720 ##$a	701 ##$a 711 ##$a 200 ##$g
date	датум издавања	260 ##$c	210 ##$d
type	тип садржаја ресурса (може бити на пример слика, звук, текст...)	655 #7$a при чему је вредност потпоља 655 #7$2 local	608
format	физички формат	856 ##$q	336 ##$a

Индекс	**Опис**	**MARC21**	**UNIMARC**
	ресурс (на пример електронски ресурс)		
identifier	једнозначна референца на конкретан ресурс који је описан метаподацима (на пример URL, DOI…)	Уколико терм почиње са http:// тада се мапира на 856 40$u, иначе на 024 8#$a	001 010 011 020 300 ##$a
source	референца на ресурс од кога је настао ресурс који се описује (на пример референца на папирну верзију публикације на основу које је настала електронска верзија)	786 0#$n	324
language	језик публикације	546 ##$a	101 300
relation	референца на релевантне ресурсе	787 0#$n	300
coverage	период или локација на коју се односи ресурс	500 ##$a	300
rights	особа или организација која полаже право на ресурс	540 ##$a	300

Табела 4.2 Индекси дефинисани *Dublin Core Context Set*-ом

4.2.2 Синтакса CQL упитног језика

Синтакса упитног језика у ABNF (*Augmented Backus-Naur Form*) нотацији [38] приказана је на листингу 4.1. Знак '=' одваја елемент од његове дефиниције, знак '/' раздваја алтернативне могућности за елемент, угластим заградама '[ ]' су уоквирени опциони елементи док је знаком '*' представљено вишеструко појављивање неког елемента.

Кључне речи које су део синтаксе уоквиране су двоструким наводницима.

```
cql-query = query [sort-spec]
query = *prefix-assignment search-clause-group
search-clause-group = search-clause-group Boolean-modified subquery / subquery
subquery = "(" query ")" / search-clause
search-clause = [index relation-modifed] search-term
search-term = simple-string / quoted-string
sort-spec = sort-by *index-modified
sort-by = "sortby"
prefix-assignment = ">" [prefix "="] uri
prefix = simple-name
uri = quoted-uri-string
index-modified = index [modifier-list]
index = simple-name / prefix-name
relation-modified = relation [modifier-list]
relation = relation-name / relation-symbol
relation-name = simple-name / prefix-name
relation-symbol = "=" / ">" / "<" / ">=" / "<=" / "<>" / "=="
Boolean-modified = Boolean [modifier-list]
Boolean = "and" / "or" / "not" / "prox"
modifier-list = *modifier
modifier = "/" modifier-name [modifier-relation]
modifier-name = simple-name
modifier-relation = relation-symbol modifier-value
modifier-value = simple-string / quoted-string
prefix-name = prefix "." simple-name
quoted-uri-string = URI под наводницима
quoted-string = секвенца карактера под наводницима. Унутар наводника се може појавити знак за наводнике али онда испред тог знака мора стојати знак \.
simple-name = simple-string
simple-string = Било која секвенца карактера која не садржи знаке (,),=,>,<,",/, или празан карактер.
```

Листинг 4.1. Синтакса CQL упитног језика

У даљем тексту ће бити објашњени основни концепти CQL упитног језика и биће дати конкретни примери упита.

CQL упитни језик се може састојати или од једног операнда или од више операнада повезани логичким операторима. У оба случаја може се навести и критеријум сортирања навођењем кључне речи *sortby*. Операнд се може састојати од индекса, релације и терма који се тражи или само од терма.

Индекс припада одређеном *Context Set*-у, и стога индекс може бити квалификован префиксом, односно испред индекса може стојати скраћени назив *Context Set*-а коме припада. Индекс и префикс су одвојени знаком '.'. С обзиром да је *Context Set* јединствено одређен идентификатором испред упита се могу дефинисати сви *Context Set*-ови који се користе у упиту на тај начин што се наводи скраћен назив скупа и његов идентификатор и знак '>'.

CQL упитни језик дефинише стандардне релације поретка које представљају везу између индекса и терма. Поред стандардних релација могу се појавити и релације које су дефинисане *Context Set*-овима и оне такође могу бити квалификоване одговарајућим префиксом. Уколико нема префикса испред релације сматра се да она припада CQL *Context Set*-у. Релације се могу модификовати употребом једног или више релационих модификатора. Релација и модификатори су одвојени знаком '/'.

CQL упит се може састојати од операнада повезаних логичким операторима и дефинисана су четири бинарна логичка оператора: AND, OR, NOT, *Prox*. Оператори AND и OR су стандардни Булови оператори док оператор NOT није унарни оператор, већ бинарни и уколико постоје два операнда А и Б повезана оператором NOT онда је резултат претраге по овом упиту скуп погодака који задовољавају операнд А али не задовољавају операнд Б. Оператор *Prox* дефинише удаљеност два терма у оквиру посматраног индекса. Оператор *Prox* може имати логичке модификаторе који још детаљније дефинишу семантику овог оператора. Оператор *Prox* може имати више логичких модификатора и они су међусобно раздвојени знаком '/'.

У наставку дати су неки од типичних примера упита дефинисаних помоћу CQL упитног језика.

Пример 1:

CQL упит: `Шекспир`

Значење: Ово је упит који се састоји само од терма који се тражи и сервер сам одређује индекс по ком претражује.

Пример 2:

CQL упит: `creator = Шекспир`

Значење: Овај упит се састоји од индекса creator, релације '=' и терма Шекспир, односно траже се публикације код којих је за примарну одговорност унет терм Шекспир.

Пример 3:

CQL упит: `title = Xamlet sortBy date/ascending`

Значење: Овим упитом се тражи терм Хамлет у индексу *title* и дефинисан је услов за сортирање по индексу *date* у растућем редоследу, при чему *ascending* представља модификатор индекса.

Пример 4:

CQL упит: `dc.title = Xamlet`

Значење: Ово је упит у коме је дефинисан квалификован индекс *title* са префиксом *dc*, што представља скраћени назив за *Dublin Core Context Set.*

Пример 5:

CQL упит: `dc.title any Јулија Ромео`

Значење: Ово је пример упита где индекс припада *Dublin Core Context Set*-у, а релација *any* пошто је некваликована припада *CQL Context Set*-у и траже се они резултати који у наслову имају или реч Јулија или реч Реомео.

Пример 6:

CQL упит:
```
>dc="info:srw/context-sets/1/dc-1.1" dc.title any fish
```

Значење: Ово је пример упита у коме је експлицитно дефинисан *Context Set* навођењем индикатора и скраћеног назива скупа. Ознака > је део синтаксе упита и указује на то да после ње следи дефинисање *Context Set*-а.

Пример 7:

CQL упит: `keyword =/relevant рачунар`

Значење: Ово је упит у коме је употребљен модификатор релације *relevant* и тражи се скуп погодака који у индексу *keyword* садрже појмове релевантне са појмом *рачунар*.

Пример 8:

CQL упит :
`(dc.title=Хамлет and dc.creator=Шекспир)or keyword=Хамлет`

Значење: Ово је пример упита у коме је више операнада повезано логичким операторима *and* и *or*. Такође ово је пример да се у једном упиту могу комбиновати индекси из различитих *Context Set*-ова (индекс *title* и *creator* припадају *Dublin Core* скупу док *keyword* припада *CQL* скупу).

Пример 9:

CQL упит :
`dc.title=санprox/unit=word/distance=1 dc.title=/stem ноћ`

Значење: Ово је пример упита са логичким оператором *prox* при чему су наведени и логички модификатори *unit* и *distance.* Поред тога дат је и релациони модификатор *stem*. Резултат овог упита су сви погодци који у наслову садрже речи *сан* и корен речи *ноћ* и удаљеност између те две речи је једна реч.

CQL упитни језик дефинише различите концепте и велики спектар могућности за формирање упита, али SRU стандардом није прописано да сервер мора да имплементира све те концепте. С тога су дефинисана три нивоа који се односе на степен сложености упита који сервер може да парсира и обради.

Ниво 0 дефинише формирање упита који се састоји само од терма који се тражи, односно клијент не дефинише ни индекс ни релације већ то сервер сам одређује. Сервер мора бити у могућности да у случају да клијент пошаље упит који сервер не подржава врати одговарајућу поруку о грешци.

Ниво 1 обухата правила дефинисана нивом 0 и проширује та правила тако да сервер може да парсира:

а) упите који се састоје од индекса, релације и терма,

б) упите код којих су операнди повезани логичким операторима.

Битно је истаћи да сервер мора да подржи парсирање упита наведеног под а) и упита под б), али да не мора да подржи обраду оба типа упита већ може подржати само један од та два типа.

Ниво 2 обухвата ниво 1 и још је у могућности да парсира било који упит дефинисан CQL упитним језиком, али не мора да имплементира све концепте и у таквим случајевима може вратити поруку о грешци.

4.3 МАПИРАЊЕ Z39.50 УПИТНОГ ЈЕЗИКА НА CQL УПИТНИ ЈЕЗИК

У овом одељку анализиране су особине упитних језика CQL и упитног језика тип-1 дефинисаног у стандарду Z39.50 са циљем да се утврди веза између њих. Односно да ли постоји еквиваленција између ова два упитна језика или је један упитни језик подскуп другог или су то два независна језика. Крајњи циљ због ког се врши ова анализа је да се изабере један упитни језик који ће се користити у имплементацији софтверског система који треба да омогући претраживање по различитим упитним језицима. Тачније, идеја је да се за сваки упит који подржава тај систем дефинише одговарајућа трансформација у општи језик, и да онда систем даље ради само са тим упитним језиком.

У наставку је прво анализирано да ли се упит формиран помоћу Z39.50 упитног језика може трансформисати у CQL упитни језик.

Z39.50 упитни језик и CQL упитни језик имају доста сличних особина. Оба упитна језика подржавају формирање упита који се састоји од једног операнда или од више операнада повезаних логичким операторима. У оба случаја се користе исти логички оператори са истом семантиком. Међутим једна од основних разлика између ова два типа упита је у том што је CQL упитни језик заснован на концепту *Context Set*-а док је упит дефинисан по Z39.50 стандарду заснован на концепту *Attribute Set*. Због овога није могуће директно мапирање операнда дефинисаног у једном упиту на операнд дефинисан у другом упиту. Z39.50 упитни језик дефинише три врсте операнда:

Z1. листа атрибута са термом који се тражи,
Z2. именовани скуп као операнд,
Z3. именовани скуп са рестрикцијом на одређене атрибуте.

Са друге стране, према CQL упитном језику операнд може бити:

C1. само терм,

C2. именовани скуп,
C3. операнд који је дефинисан индексом, релацијом и термом који се тражи.

На основу изнетих чињеница следи да оба упита подржавају именоване скупове као операнде и то неће бити проблем приликом конверзије упита. Тип операнда у Z39.50 упитном језику наведеном под Z3. уствари представља комбинацију операнада наведених под Z1. и Z2. Односно, он се може тумачити као упит који се састоји од два операнда повезана логичким оператором AND, при чему је први операнд облика наведеног под Z2., а други операнд је облика наведеног под Z1. Према томе, проблем конверзије Z39.50 упитног језика на CQL упитни језик се своди на проблем конверзије операнда наведеног под Z1 у одговарајући CQL упит.

Основни задатак је мапирати елементе *Attribute Set* на одговарајуће концепте дефинисане CQL упитним језиком. С обзиром да постоји више различитих *Attribute Set*-ова који се могу користити при формирању упита потребно је дефинисати посебно мапирање за сваки појединачан скуп. У овом одељку биће дато мапирање скупа *bib-1* на одговарајуће концепте CQL упитног језика првенствено јер је то један од најчешћих скупова који се користи за претраживање библиотечког садржаја.

Bib-1 скуп се састоји од шест типова атрибута:

- Use,
- Relation,
- Structure,
- Truncation,
- Position,
- Completness.

Use атрибут садржи предефинисане вредности по којима се може претраживати и ове вредност ће се мапирати на одговарајуће вредности индекса дефинисане у *Context Set*-овима. Вредности *Use* атрибута су изражене нумеричким вредностима. На пример *Use* атрибута са вредношћу 4 представља наслов публикације. Пример мапирања неких вредности *Use* атрибут у индексе дефинисане *Context Set*-овима дат је у табели 4.3. Најчешће коришћене вредности *Use* атрибута се углавном могу мапирати на елементе *Dublin Core Context Set*-а, мада се могу искористити и индекси дефинисани у другим *Context Set*-овима.

Use атрибут	**Индекс**	**Context Set**
4 (наслов)	title	DublinCore
8 (ISSN)	issn	Bath Context Set
14 (УДК број)	subject	DublinCore
31 (датум издавања)	date	DublinCore
1003 (аутор)	creator	DublinCore
1018 (издавач)	publisher	DublinCore

Табела 4.3 Примери мапирања вредности *Use* атрибута на индексе

Relation атрибут садржи релације поретка и оне се једнозначно мапирају на релације дефинисане CQL у *Context Set*-ом. Постоје још и вредности *phonetic, stem* и *relevance* које би се мапирале на истоимене модификаторе релација дефинисане у CQL *Context Set*-у, а основна релација би била релација једнакости. Вредност *Relation* атрибута *AlwaysMatches* би приликом мапирања игнорисала терм који се тражи и тумачило би се као да је задат CQL упит који садржи индекс *allRecords*.

Вредности *Structure* атрибута би се мапирали на одговарајуће модификаторе релација који одређују формат терма.

Вредности *Truncation* атрибута би биле замењене специјалним знаком '*' на одговарајућим позицијама. На пример, уколико је вредност *Truncation* атрибута *left and right* упит формиран помоћу CQL упитног језика би садржао терм који на почетку и на крају има знак '*'.

Вредости *Position* атрибута дефинишу да ли се терм налази на почетку садржаја који је индексиран или позиција није битна. CQL упитни језик има специјални знак '^' који у потпуности задовољава семантику *Position* атрибута. Овај знак би се могао искористити и за мапирање *Completness* атрибута који дефинише да ли терм који се тражи треба да буде једини у индексираном садржају или се могу појављивати и друге речи.

Приликом креирања упита помоћу Z39.50 упитног језика уколико се користи *bib-1* скуп потребно је дефинисати вредности за све атрибуте

из тог скупа и у табели 4.4 су дати примери Z39.50 упита који се састоје само од операнда који садржи листу атрибута и терм, и њихово мапирање на еквивалентне упите дефинисане CQL упитним језиком.

Семантика упита	**Z39.50**		**CQL**
Пронађи библиографске записе који у наслову имају реч *Hamlet*	*Use*	4- наслов	dc.title=Hamlet
	Relation	3 - =	
	Structure	2 - word	
	Truncation	100 - do not truncate	
	Position	3 - any position	
	Completness	1- incomplete subfield	
	терм	Hamlet	
Пронађи библиографске записе који у наслову имају **само** реч *Hamlet*	*Use*	4- наслов	dc.title=^Hamlet^
	Relation	3 - =	
	Structure	2 - word	
	Truncation	100 - do not truncate	
	Position	3 - any position	
	Completness	3- complete field	
	терм	Hamlet	
Пронађи библиографске записе код којих наслов садржи реч чији је корен речи реч *дан*	*Use*	4- наслов	dc.title=/stem дан
	Relation	101 - stem	
	Structure	2 - word	
	Truncation	100 - do not truncate	
	Position	3 - било где	
	Completness	1- incomplete subfield	
	терм	дан	
Пронађи библиографске	*Use*	4- наслов	dc.title= ^рад*
	Relation	3 - =	
	Structure	2 - word	

Семантика упита	**Z39.50**		**CQL**
записе чији наслов почиње са низом карактера *'рад'*	*Truncation*	1 - right Truncation	
	Position	1 - first in field	
	Completness	1- incomplete subfield	
	терм	рад	
Пронаћи библиографске записе који у наслову садрже само фразу *lord of the rings*	*Use*	4- наслов	dc.title adj ^lord of the rings^
	Relation	3 - =	
	Structure	1 - phrase	
	Truncation	100 - do not truncate	
	Position	3 - било где	
	Completness	3- complete field	
	терм	lord of the rings	

Табела 4.4 Мапирање Z39.50 упита на CQL упите

На основу изнетих чињеница може се закључити да се упити формирани у складу са Z39.50 упитним језиком могу трансформисати у упите дефинисане CQL упитним језиком.

Поставља се питање да ли се може извршити трансформација у обрнутом смеру, односно да ли се сваки упит дефинисан CQL упитним језиком може трансформисати у одговарајући упит дефинисан Z39.50 упитним језиком. Као што се може видети из претходно написаног Z39.50 и CQL имају доста сличних концепата. Оно по чему се CQL на први поглед разликује је појам модификатора, међутим пажљивијим разматрањем се долази до закључка да се модификатори могу трансформисати у одговарајуће атрибуте из скупа атрибута *bib-1*. Што се тиче индекса, они могу припадати различитим *Context Set*-овима, али са друге стране Z39.50 упитни језик омогућава да се дефинише подразумевани скуп атрибута чиме се истиче да сви атрибути припадају том скупу, али је могуће исто тако да се за појединачне атрибуте дефинише скупу ком припадају. Према томе индекси се могу мапирати на одговарајуће атрибуте, уколико се дефинише добро мапирање између *Context Set* – а у одговарајући *Attribut Set*. Једино што

може представљати проблем приликом мапирања су упити у коме је наведен услов сортирања. Наиме, CQL омогућава да се дефинише критеријум по ком ће резултати претраге бити сортирани док Z39.50 упитним језиком тај концепт није подржан и према томе такви упити се не би могли у потпуности трансформисати у одговарајуће Z39.50 упите.

С обзиром да је показано да се упит дефинисан упитним језик типа-1 може пресликати у упит дефинисан CQL упитним језиком а да се при томе не изгуби на квалитету упита, као општи језик који ће се користити у систему за претраживање који је описан у овој дисертацији изабран је CQL упитни језик. Још један од разлога због чега је у овој дисертацији изабрано да се Z39.50 упитни језик трансформише у CQL упитни језик је тај што је CQL интуитивнији и читљивији. Односно у случају трансформације CQL упита у Z39.50 упит морало би се додатно водити рачуна о синтакси упита која је дефинисана ASN1 нотацијом. Сложеност Z39.50 синтаксе приказана је на следећем примеру.

Пример: Нека је дат најједноставнији упит у CQL упитном језику, који садржи само терм који се тражи, на пример упит је: *компјутер*, тада еквивалентан упит у Z39.50 упитном језику има следећи облик:

```
{type-1{
       attributeSet 1.2.840.10003.3.1,
       rpn{
         op{
          attrTerm{
           attributes{
                       {attributeType  1,attributeValue{numeric
1035}}
                       {attributeType  2,attributeValue{numeric
3}}
  {attributeType 3,attributeValue{numeric 3}}
  {attributeType 4,attributeValue{numeric 1}}
  {attributeType 5,attributeValue{numeric 100}}
  {attributeType 6,attributeValue{numeric 1}}
                      },
                      term{general "компјутер" }
                  }
            }
          }
       }
}
```

Софтверски систем који је описан у овој дисертацији подржава претраживање употребом CQL и Z39.50 упитног језика. Међутим уколико би било потребно обезбедити подршку за неки други упитни језик морала би се извршити анализа тог упитног језика и његова компатибилност са CQL упитним језиком. Уколико би тај нови језик био подскуп од CQL језика, онда би било потребно само дефинисати одговарајућу трансформацију и сама архитектура система се не би променила. У случају да тај нови упитни језик садржи концепте који нису дефинисани CQL упитним језиком, било би потребно проширити CQL и опет дефинисати одговарајућу трансформацију за тај нови упитни језик.

4.4 МАПИРАЊЕ CQL УПИТНОГ ЈЕЗИК НА LUCENE УПИТНИ ЈЕЗИК

Оба упитна језика и CQL и тип-1 дефинисан Z39.50 стандардом уведени су са циљем да клијент, односно крајњи корисник, не мора да познаје архитектуру серверске стране система са којим комуницира. То значи да клијент не зна ништа о томе гда сервер чува податке (то може бити у релационој бази, XML бази података или пак у неким фајловима) нити каква је структура тих података. Међутим, са серверске стране мора да постоји компонента која ће да изврши мапирање упита формираних помоћу CQL или Z39.50 упитног језика на конкретну структуру података.

При имлементацији софтверског система који је предмет истраживања у овој дисертацији коришћен је програмски пакет *Lucene* који има свој упитни језик и у овом одељку је описано мапирање CQL упитног језика на *Lucene* упитни језик. *Lucene* садржи једноставан али веома експресиван упитни језик који омогућава:

- Употребу логичких оператора
- Претрагу по опсегу вредности
- Употреба џокер знакова
- Fazzy претрагу
- Близинску претрагу
- Повећање значаја терма

У наставку дат је предлог на који начин се концепти дефинисани CQL упитним језиком могу мапирати на *Lucene* упитни језик.

CQL концепт: `Индекс`

Lucene омогућава дефинисање поља по којима се може претраживати, тако да се може за сваки индекс из одређеног *Context Set* дефинисати *Lucene* поље.

CQL концепт: `Логички оператори`

У *Lucene*-у су подржани AND, OR и NOT оператор са истом семантиком као што је дефинисано и CQL упитним језиком. Пандан оператору *Prox* је близинска претрага коју *Lucene* подржава.

CQL концепт: `Релације поретка (=, <, >, <=, >=, < >)`

Lucene подржава само релацију једнакости, али се остале релације поретка <, >, <=, >=, < > могу имплементирати као претраге по опсегу вредности. На пример, CQL упит `date < 2009` се може представити претрагом по опсегу где вредност префикса *date* припада интервалу између 0 и 2009. Уколико је потребно да се дефинише упит `date > 2009` тада се такође дефинише претрага по опсегу само што је доња граница интервала 2009 док је горња граница недефинисана, односно Lucene дозвољава вредност *null* уколико је потребно истаћи да је граница интервала недефинисана. Што се тиче релације неједнакости (< >) њена интерпретација захтева мало више труда. Нека је претпоставка да је упит *`date < > 2009`* тада ће се овај упит трансформисати у упит који садржи два операнда повезана логичким оператором OR, односно *`date`* `∈(null, 2009) OR` *`date`* `∈ (2009, null)`, при чему границе нису укључене.

CQL концепт: `Релације` *`any,all,adj, within, encloses`*

Релације *any* и *all* се могу представити помоћу упита где су операнди међусобно повезан са логичким операторима OR, односно AND. Релација *adj* се може представити упитом који претражује фразе. Релација *within* се може имплементирати као претрага по опсегу вредности. Релација *encloses* би се могла имплементирати као претрага по терму који садржи на почетку и на крају џокер знак '*'.

CQL концепт: `Релациони модификатори (`*`stem, relevant, fuzzy, phonetic, ignoreCase, respectCase, ignoreAccents, respectAccsent`*`)`

Lucene подржава претрагу по корену речи (*stem*), претрагу по релевантности (*relevant*), претрагу по сличности (*fuzzy*) и претрагу по звучности (*phonetic*), међутим потребно је за сваку врсту претраге

дефинисати одговарајући алгоритам по коме би се одредило, на пример, да ли две речи имају исти корен и слично.

Претрага по звучности није употребљива за српски језик јер се речи изговарају онако како се и пишу, ова претрага има смисла уколико се претражује садржај на неким другим језицима и тада би требало имплементирати одговарајући алгоритам.

Један од начина да се имплементирају модификатори којима се одређује да ли се величина слова односно дијакртици узимају у обзир приликом претраге или не је да се приликом индексирања самог садржаја направи *Lucene* анализатор који ће садржај индексирати и са дијакритицима и без њих, односно претвориће сва слова у мала али ће задржати и оргинални облик садржаја.

CQL концепт: Модификатори који одређују формат терма који се тражи (*word, string, isoDate, number, uri , oid, masked,unmasked, regexp, substring*)

Уколико се користи модификатор *word* за сваку реч направиће се по један операнд и међусобно повезати операнде логичким операторима AND или OR у зависности која релација је дата. Подразумевани оператор је OR. Модификатори *string, uri, oid* ће се тумачити као да се претражује фраза.

У случају да је модификатор *isoDate* могао би се направити алгоритам који би трансформисао терм у формат датума који је подржан од стране сервера. Исто важи и за модификатор *regexp*, где би било потребно направити алгоритам који ће трансформисати регуларни израз у терм који садржи само џокер знаке који су подржани у *Lucene*-у. Модификатор *masked* је подразумевани модификатор и тада ће се џокер знаци и тумачити у складу са њиховом семантиком, међутим уколико је модификатор *unmasked* тада ће се креирати упит који подразумева претрагу по фрази и специјални знаци ће се третирати као и остали карактери. За имплементацију модификатора *substring* се такође могу искористити џокер знаци ? и * који су дефинисани у *Lucene*-у.

Поглавље 5

Архитектура система за посредовање између софтверских компоненти

У овом поглављу дат је преглед најчешће коришћених софтверских архитектура при имплементацији система који омогућавају претраживање и преузимање записа. Такође дат је и предлог софтверске компоненте која би омогућила имплементацију протокола SRU и Z39.50 у систему БИСИС.

5.1 АРХИТЕКТУРЕ СИСТЕМА ЗА ПРЕТРАЖИВАЊЕ И ПРЕУЗИМАЊЕ ПОДАТАКА

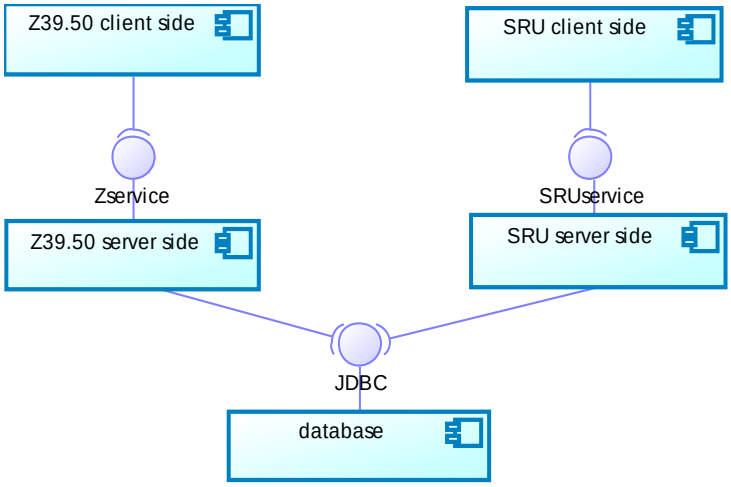
Појам дигитална библиотека почео је да се интезивно користи 90-тих година прошлог века, мада постоје одређене несугласице око његове интерпретације. Библиотекари дигиталну библиотеку виде само као један нови облик библиотеке који је настао са појавом нових технологија као што је Интернет, према томе дигитална библиотека је и даље институција. Међутим, са друге стране компјутерски стручњаци дигиталну библиотеку виде као скуп електронских докумената. Без обзира на тумачење овог појма и једни и други се слажу да је један од основни задатака дигиталних библиотека да омогуће пружање информација крајњем кориснику путем Интернета.

У циљу да се библиотечки фонд учини доступан корисница путем Интернета развијени су различити стандарди за претраживање и преузимање информација о којима је било речи у другом поглављу. У наставку је дат преглед софтверских архитектура које се користе у имплементацији система за претраживање и преузимање података путем одговарајућих протокола.

Прегледом објављених научних радова који су се бавили имплементацијом серверске стране оваквих система уочени су одређени типови софтверске архитектуре које су примењивани у имплементацији. Основни проблем који је углавном решаван је како искористити постојећу софтверску архитектуру система који

подржавају Z39.50 протокол, а да се при томе омогуће и функционалности SRU протокола. Паралелно подржавање оба протокола је веома битно јер појединачне библиотеке се неће тако лако одлучити на прелазак на нови протокол док цела библиотечка заједница не почне да подржава комуникацију путем новог протокол, а са друге стране библиотечку заједницу чине појединачне библиотеке.

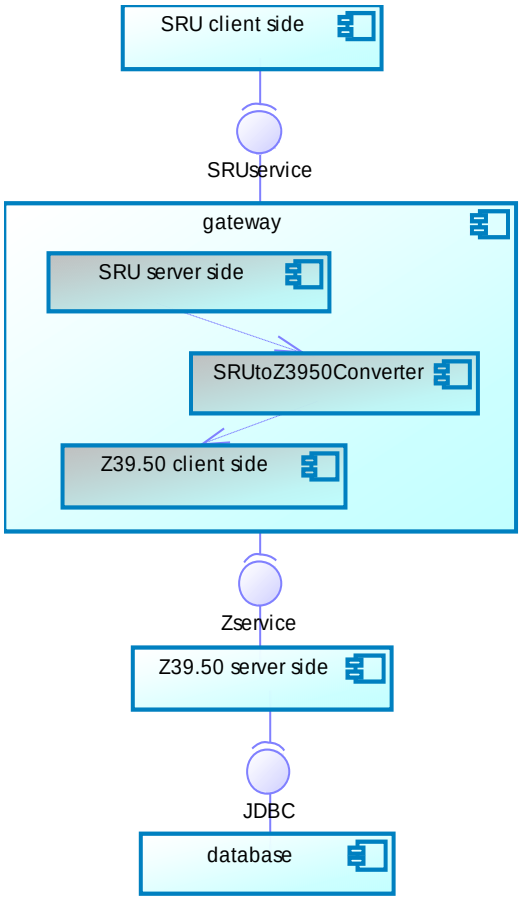
Један од приступ у имплементацији система за претраживање и преузимање података подразумева постојање две самосталне имплементације Z39.50 и SRU серверске стране протокола, при чему обе софтверске компоненте приступају само једној бази података. Овај приступ подразумева креирање серверских имплементација од почетка без искоришћавања постојеће архитектуре. Овакав приступ је добар уколико не постоје ни Z39.50 нити SRU серверска страна протокола, или уколико постоји библиотечки информациони систем који, на пример, подржава Z39.50 протокол али је програмски кôд отворен и дозвољава измене, односно постоји могућност приступа бази података па је стога могуће имплементирати и серверску страну SRU протокола. Архитектура система који је заснован на овом приступу дата је на слици 5.1 у виду дијаграма компоненти. Јасно су издвојене софтверске компоненте које представљају имплементацију клијентске и серверске стране протокола за сваки појединачан протокол, док је база података заједничка.



Слика 5.1. Софтверска архитектура система са одвојеним имплементацијама серверске стране протокола

Међутим, велики број библиотека је купио комерцијалне информационе системе који садрже серверску страну Z39.50 протокола, али имплементација система није доступна и на њој се не могу вршити преправке. Тада једно од решења је да се направи такозвани „*gateway*“ (капија). *Gateway* је софтверска компонента која са једне стране има имплементацију SRU серверске стране протокола, а са друге стране имплементацију Z39.50 клијента који ће путем Z39.50 протокола приступати постојећем Z39.50 серверу. Односно, уколико неки SRU клијент пошаље захтев за претраживање, тај захтев ће прихватити *gateway,* трансформисати SRU захтев у захтев дефинисан Z39.50 стандардом и проследити тај захтев Z39.50 серверу. Слично томе, када добије одговор од Z39.50 сервера *gateway* ће тај одговор трансформисати у складу са SRU стандардом и проследити га SRU клијенту. На овај начин клијент ће имати утисак да комуницира директно са SRU сервером, док ће постојећи Z39.50 сервер мислити да добија захтеве од Z39.50 клијента. На слици 5.2 дат је дијаграм компоненти који представља архитектуру система који је заснован на овом приступу.

Софтверска архитектура приказана на слици 5.2 је један од најчешће коришћених приступа који је за имплементацију свог система искористила и Конгресна библиотека што је описано у раду (Taylor и Dickmeiss, 2005). Конгресна библиотека користи комерцијални библиотечки информациони систем *Voyager* [5] који омогућава претраживање по Z39.50 протоколу, али да би омогућили да и SRU клијенти претраже њихов фонд, компанија *IndexData* [39] је развила за њихове потребе софтверску компоненту *YazProxy* [6] која представља SRU-Z39.50 *gateway*. Иста идеја искоришћена је и у имплементацији портала „*The European Library*“ [40] који за циљ има да обједини фондове свих европских националних библиотека и да омогући кориснику да путем јединственог портала претражује све европске библиотеке. Софтверска архитектура примењена у реализацији овог пројекта описана је у раду (Van Veen and Oldroyd, 2004).

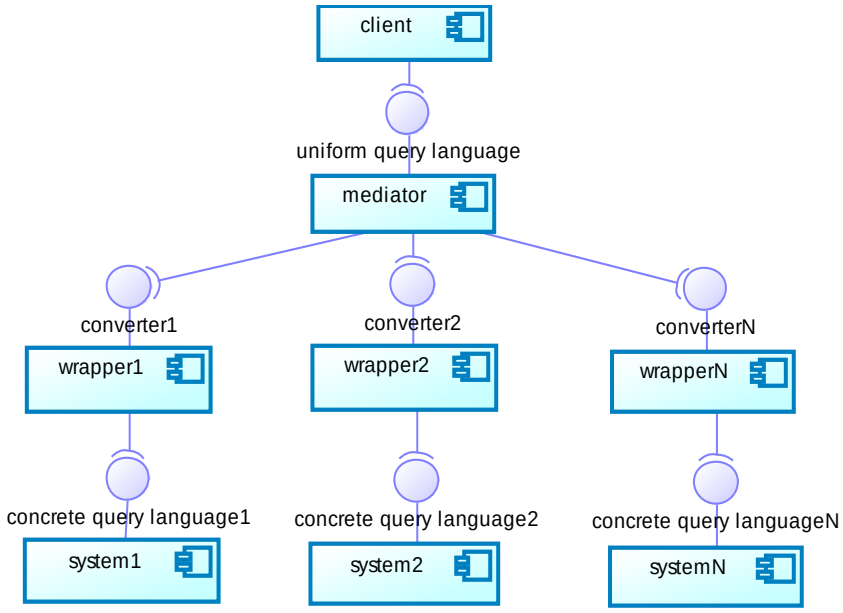


Слика 5.2. Софтверска архитектура система са *gateway*-ом

Још један од интересантних приступа реализацији софтверске архитектуре система намењених за претраживање и преузимање информација уочен је код система који се баве претраживањем хетерогених извора информација. Архитектура оваквих система приказана је на слици 5.3.

Основна идеја код већине оваквих система је да се кориснику обезбеди јединствен интерфејс за претраживање различитих система, а да постоји посебна компонента која ће прихватати кориснички упит и трансформисати га у упит који подржава конкретан систем који нуди услуге претраживања и преузимања података. Ова компонента се у литератури назива *mediator* док се за сваки систем креира посебна софтверска компонента која конвертује кориснички упит у упит који је

дефинисан од стране конкретног система и она се назива *wrapper* (Wiederhold, 1992).



Слика 5.3. Архитектура система са *mediator*/*wrapper* приступом

На слици 5.3 приказана је архитектура система који омогућава комуникацију са три различита система (*system1*, *system2* и *system3*), сваки од тих система може користити различите упитне језике па стога се морају имплементирати и три различите *wrapper* компоненте (*wrapper* 1, *wrapper* 2 и *wrapper* 3). Наравно, у оваквој архитектури сваки од система са којим се комуницира може представљати нову *mediator* компоненту која ће комуницирати са неким другим системима. Односно, *wrapper* компонента може комуницирати или са системом или са *mediator*-ом.

Улога *mediator*-а је да прихвати упит дефинисан од стране корисника и да тај упит пошаље свим *wrapper* компонентама. *Wrapper* компонента зна да трансформише упит који је послао *mediator* у конкретан упит који подржава систем са којим комуницира *wrapper*. Такође, *wrapper* мора да трансформише податке које је добио од система у формат који је прописао *mediator*. Комуникација између клијентске апликације и софтверске компоненте *mediator* може бити преко неког од протокола за претраживање и преузимање информација, на пример преко SRU или Z39.50 протокола. Системи чија је архитектура заснована на

mediator -у описани су у више радова (Chumbe et al., 2007; Coiera et al., 2005; Melnik et al., 2000).

Оваква архитектура погодна је за укључивање постојећих систем у јединствен портал, при чему сваки појединачни систем имплементира претраживање и складиштење информација и може да функционише и независно од портала чији је део. Основна предност овакве архитектуре је у томе што је могуће једноставно укључити нове системе који ће пружити услуге претраживања, потребно је додати нови *wrapper* који ће вршити одговарајуће трансформације упита добијеног од клијента у упит који појединачан систем може да процесира. Проблем који се може јавити код оваквих хетерогених система је проблем скупљања резултата и креирања јединственог скупа резултата у коме ће погодци бити рангирани према релевантности и неће бити дуплих погодака.

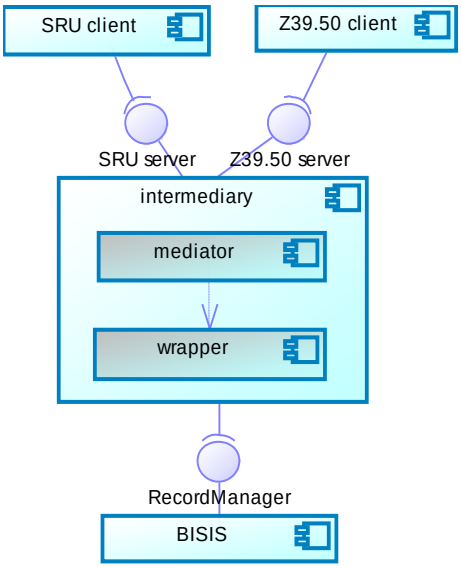
5.2 СОФТВЕРСКА КОМПОНЕНТА *INTERMEDIARY*

Информациони систем БИСИС представља интегрисано окружење за рад библиотекара, односно библиотекар може да обрађује податке о публикацијам, да их складишти и након тога да врши претраживање по различитим критеријумима. Тренутно овај информациони систем користи више градских, факултетских и специјалних библиотека у Србији и у раду (Zarić, 2006) дат је опис софтверске компоненте која омогућава да различите библиотеке које користе информациони систем БИСИС међусобно претражују и размењују библиографске записе. Тренута архитектура система БИСИС је заснована на *Peer-To-Peer* архитектури где сваки учесник у комуникацији може да претражује ресурсе других учесника, а у исто време нуди и своје ресурсе другим учесницима. Протокол који се користи у комуникацији заснован је на *ebXML* спецификацији [41] и размени одговарајућих XML докумената, тако да је на овај начин омогућено да комуницирају само они учесници који користе информациони систем БИСИС. Према томе, у тренутној имплементацији система БИСИС ресурси библиотека које користе овај систем нису доступни другим библиотекама које користе неке друге информационе системе, осим уколико се претходно не направи неки интерни договор око формата XML докумената који се размењују преко овог протокола у систему БИСИС.

Основни задатак у овој дисертацији је моделирање и имплементација софтверске компоненте која ће омогућити прихватање различитих врста упита од различитих клијената и добављање података из система

БИСИС. Потребно је обезбедити да систем БИСИС комуницира са другим системима путем неких од стандардних протокола за комуникацију као што су на пример Z39.50 и SRU. На основу прегледаних радова који се баве сличном тематиком дошло се до закључка да је за имплементацију такве софтверске компоненте најбоље применити архитектуру *mediator/wrapper*. Односно, да би се обезбедила комуникација других система са системом БИСИС путем одређених протокола потребно је у постојећи систем БИСИС интегрисати софтверске компоненте које садрже имплементацију серверских страна одговарајућих протокола. Међутим ове компоненте би садржале само имплементације конкретних протокола и са системом БИСИС би комуницирале преко компоненте *mediator*, док би *mediator* преко одговарајућег *wrapper*-а комуницирао са БИСИС системом. На овај начин би се комуникација са системом БИСИС могла остварити преко различитих протокола, а успостављање комуникације по неком другом протоколу би захтевало имплементирање одговарајуће серверске стране тог протокола. Предност овакве архитектуре је у томе што већ постоје *open-source* компоненте које имплементирају одређене протоколе и потребно је те компоненте интегрисати у предложену архитектуру система БИСИС.

Предлог архитектуре система која би омогућила претраживање и преузимање библиотечких записа у систему БИСИС од стране других информационих система дат је на слици 5.4.



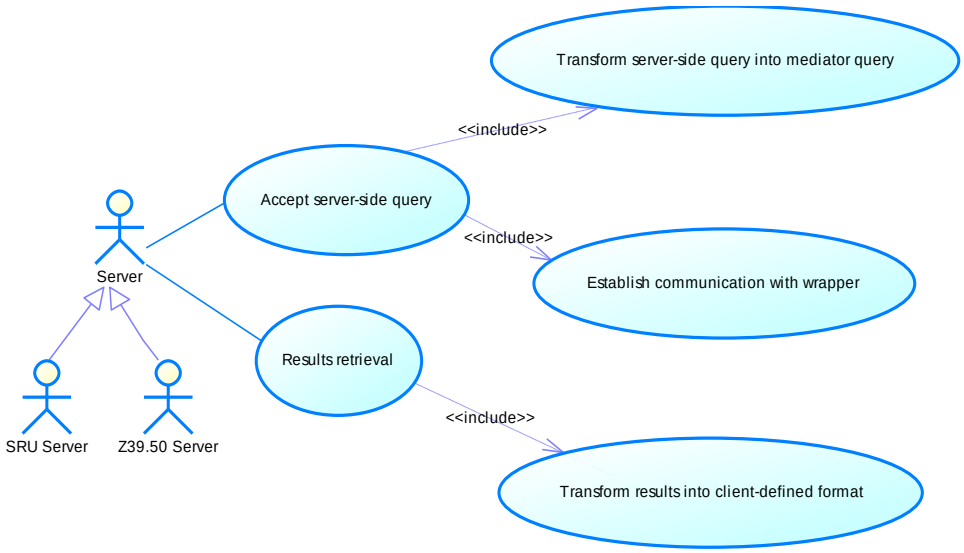
Слика 5.4. Архитектура система за преузимање и претраживање библиографских записа

Софтверска компонента која ће повезивати систем БИСИС са серверским странама протокола у даљем тексту ће се називати *intermediary* и функције ове софтверске компоненте приказане су на дијаграму случајева коришћења који је дат на слици 5.5. Ова компонента се састоји од две софтверске компоненте *mediator* и *wrapper*.

Учесници у случајевима коришћења представљају софтверске компоненте које имплементирају серверске стране одговарајућих протокола. Они су на дијаграму приказани као спецификација једног апстрактног учесника *Server*.

Компонента *mediator* мора да обезбеди прихватање упита које учесници прослеђују компоненти, што је приказано случајем коришћења *Accept server-side query*. Компонента *mediator* за даљи рад користи своју интерну репрезентацију упита и зато је прво потребно трансформисати упит добијен од учесника у одговарајућу интерну репрезентацију упита што је представљено случајем коришћења *Transform server-side query into mediator query*. Након тога компонента *mediator* може успоставити комуникацију са *wrapper* компонентом и проследити јој упит који треба да се изврши у систему БИСИС. Ова активност представљена је случајем коришћења *Establish*

communication with wrapper. Основна улога *wrapper* компоненте у систему БИСИС је да упит добијен од *mediator*-а трансформише у упит дефинисан упитним језиком који је подржан од информационог система БИСИС. Резултате претраге које компонента *mediator* добије од *wrapper* компоненте (приказано случајем коришћења *Results retrieval*) потребно је пре него што се проследе одговарајућем учеснику трансформисати у формат који је дефинисан од стране учесника и то је на дијаграму представљено случајем коришћења *Transform results into client-defined format*.



Слика 5.5. Случајеви коришћења софтверске компоненте *intermediary*

5.2.1 Софтверска компонента *mediator*

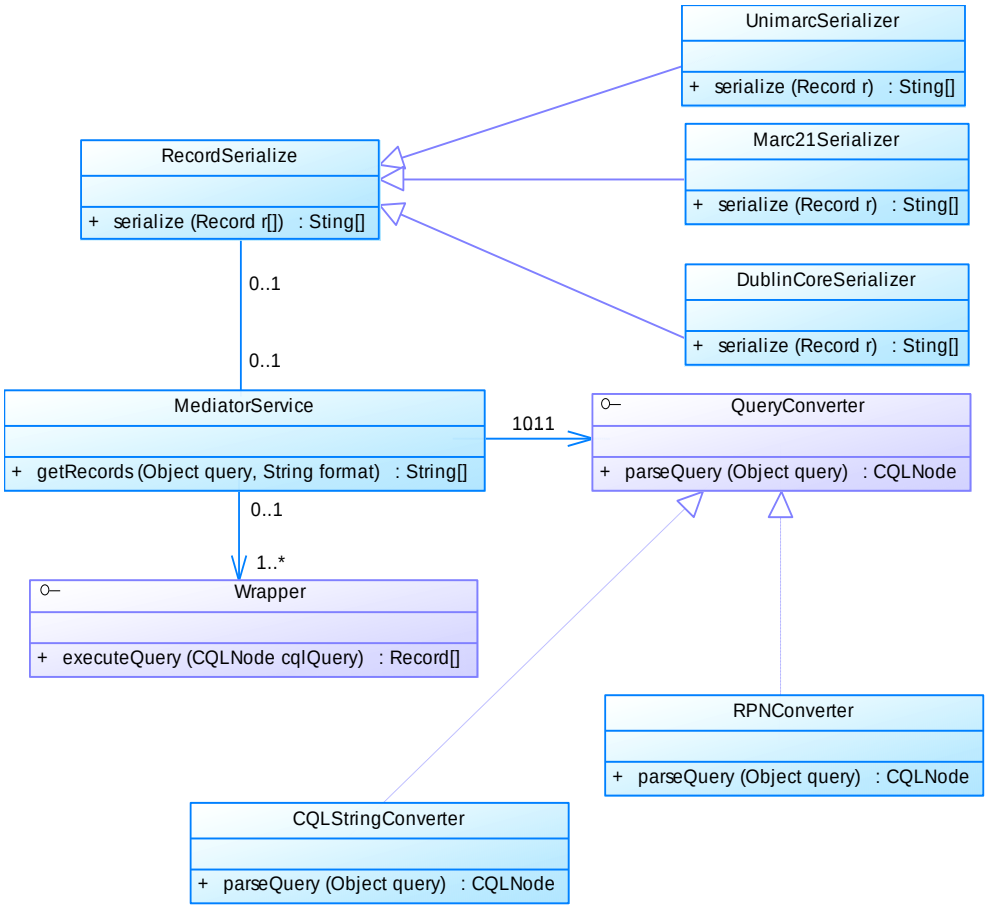
Mediator је софтверска компонента која би требало да омогући јединствен интерфејс за различите клијентске апликације, међутим у овој дисертацији је изабрано мало другачије решење. Наиме, уместо да *mediator* комуницира директно са клијентском апликацијом, што би у случају протокола за размену података била клијентска страна тог протокола, он у ствари комуницира са компонентама које имплементирају серверске стране одговарајућих протокола, док клијентске апликације размењују поруке са одговарајућим серверским странама протокола. На основу овога следи да компонента *mediator* мора да обезбеди интерфејс ка серверским странама протокола којим

ће се обезбедити прихватање упита од серверских страна и враћање резултата претраге.

На слици 5.6 дат је дијаграм класа који описује софтверску компоненту *mediator*.

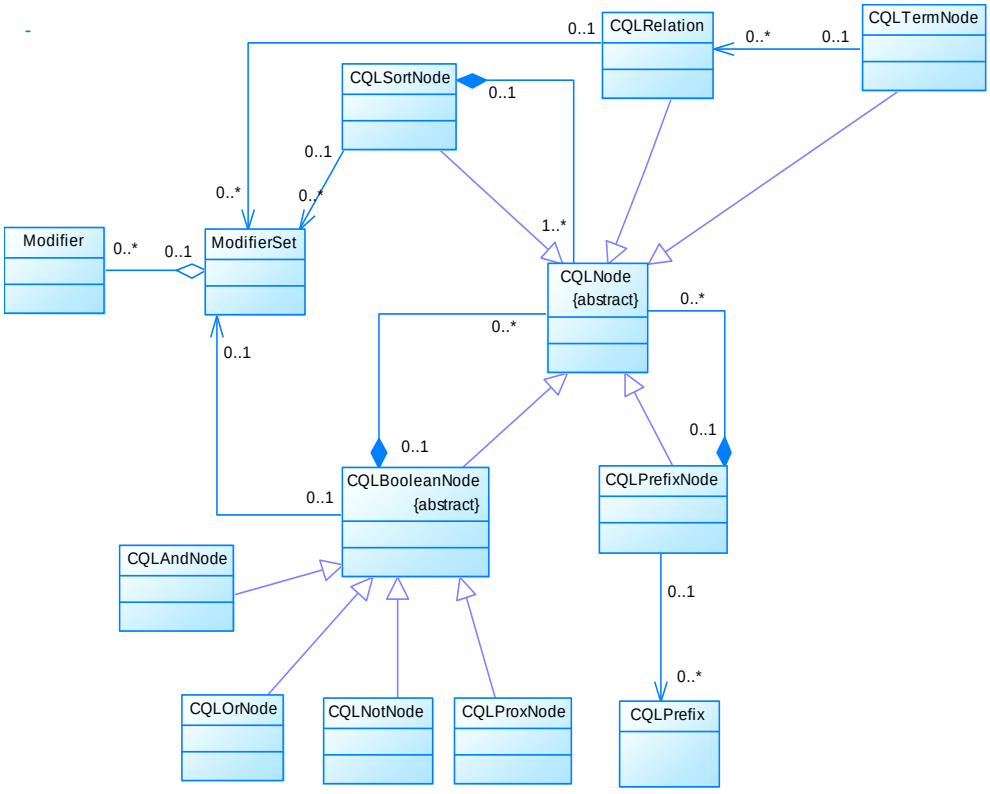
Класа `MediatorService` је задужена за комуникацију са серверским странама протокола Z39.50 и SRU. Она садржи само једну методу `getRecords(Object query, String format):String[]` која на основу добијеног упита серверским странама протокола враћа низ библиографских записа у формату који је дефинисан параметром `format`. Класом `MediatorService` реализовани су случајеви коришћења *Accept server-side query* и *Results retrieval*.

Компонента *mediator* је посредник између серверске стране одређеног протокола и система БИСИС. Компонента *mediator* може примити упите дефинисане различитим упитним језицима и њен задатак је да све те упите трансформише у један изабрани упитни језик који ће се даље прослеђивати *wrapper* компонентама. У овој имплементацији је изабрано да се сви упити које компонента *mediator* добије трансформишу у објектну репрезентацију CQL упитног језика. Један од разлога за избор CQL упитног језика као упита који ће бити прослеђиван свим *wrapper* компонентама је тај што је у поглављу 4 показано да се концепти дефинисани у упитном језику типа-1 могу пресликати на одговарајуће концепте дефинисане CQL упитним језиком. У случају да се појави нови упитни језик потребно је извршити мапирање тог новог упитног језика на CQL упитни језик или ако то није изводљиво проширити објектни модел CQL упитног језика са новим концептима.



Слика 5.6 Дијаграм класа компоненте *mediator*

На слици 5.7 дат је дијаграм класа који представља објектни модел CQL упитног језика. Овај модел је преузет из пројекта CQL-Java [42] чији је аутор Mike Taylor. Целокупан упит представљен је абстрактном класом `CQLNode`. У зависности да ли се упит састоји од операнада повезаних логичким операторима или представља само један операнд који има индекс, релацију и терм, биће креирани респективно објекти класа `CQLBooleanNode` и `CQLTermNode` који наслеђују апстрактну класу `CQLNode`. *CQL* упит који садржи критеријум за сортирање представљен је класом `CQLSortNode`, док је упит који садржи спецификацију префикса који означавају коришћене *ContextSet*-ове представљен класом `CQLPrefixNode`.



Слика 5.7 Објектни модел CQL упитног језика

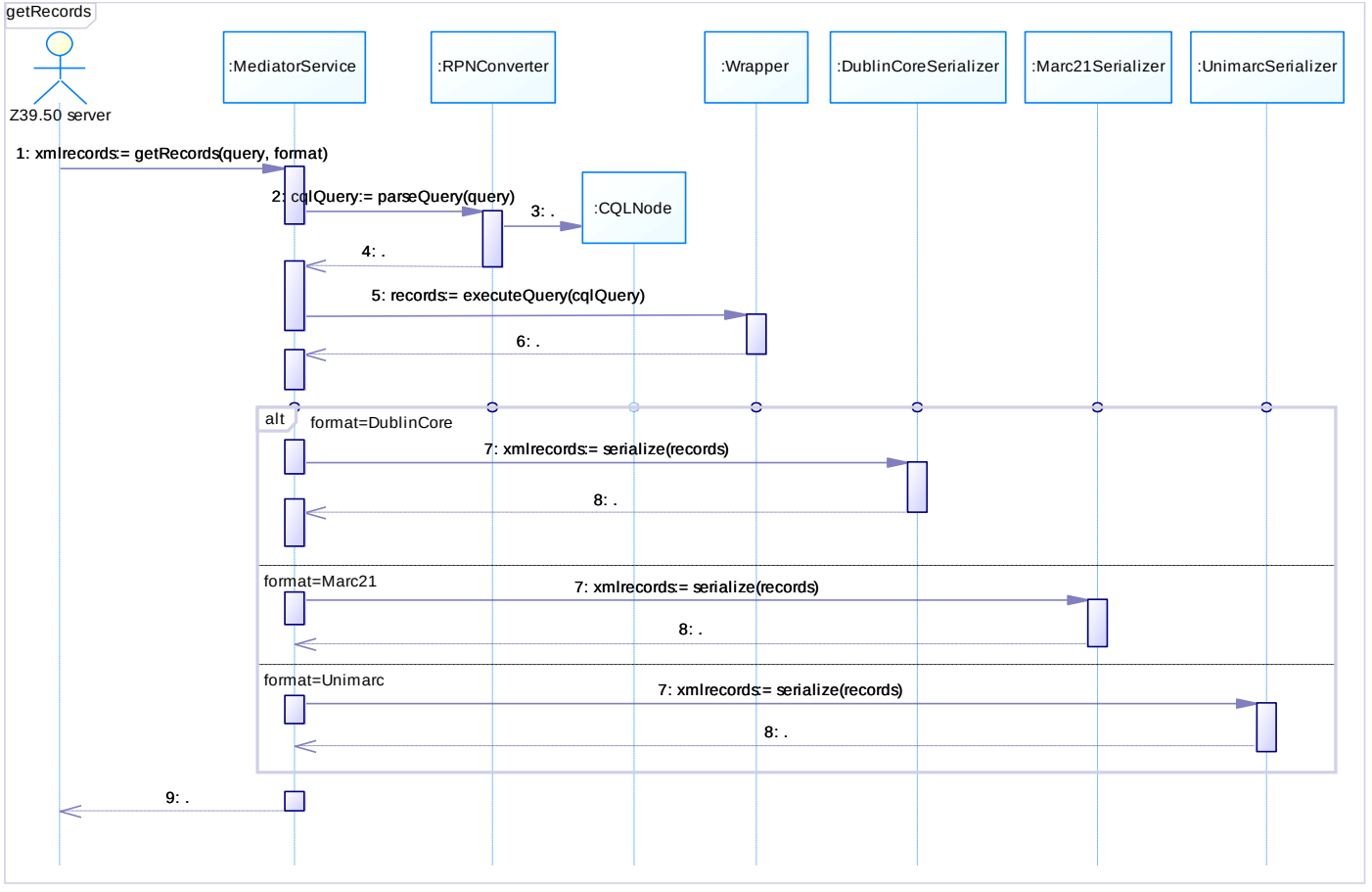
Према томе случај коришћења *Transform server-side query into mediator query* реализован је помоћу интерфејса `QueryConverter`. Овај интерфејс има две имплементације представљене класама `RPNConverter` и `CQLStringConverter`. Интерфејс садржи једну операцију `parseQuery(Object query):CQLNode` чији задатак је да упит добијен од серверске стране одговарајућег протокола трансформише у објекат класе `CQLNode`. Уколико је упит послао SRU сервер, трансформацију упита ће извршити класа `CQLStringConverter`, док уколико је упит послао Z39.50 сервер трансформацију упита ће извршити класа `RPNConverter`.

Један од задатака *mediator* компоненте је да клијенту који му је послао захтев да пронађе одговарајуће записе, врати те записе у оном формату који је клијент дефинисао. С обзиром да *mediator* комуницира са Z39.50 и SRU серверском страном, њихов задатак ће бити да провере

да ли је формат записа који клијент захтева подржан и уколико није подржан неће ни слати *mediator*-у захтев за добављање записа, иначе ће медиатор добавити записе и обезбедити трансформацију записа у захтевани формат. Компонента *mediator* библиографске записе од система БИСИС добија у облику објектног модела `Record` који је описан у трећем поглављу. Према томе, када класа `MediatorService` добије низ записа који одговарају постављеном упиту, она ће позвати методу `serialize(Record[]r):String[]`класе `RecordSerializer` која ће извршити потребну трансформацију записа. Класама `UnimarcSerializer`, `Marc21Serializer` и `DublinCoreSerializer` представљене су трансформације објектног модела `Record` у одговарајуће библиотечке формате.

На слици 5.8 дат је дијаграм секвенци који описује комуникацију између Z39.50 сервера и компоненте *mediator* у циљу добијања записа из система БИСИС. Комуникацију започиње Z39.50 сервера који шаље поруку *getRecord* објекту класе `MediatorService`. Ова порука садржи параметре који представљају упит и формат у коме се желе добити резултати претраге. По пријему поруке класа `MediatorService` прво шаље поруку *parseQuery* објекту класе `RPNConverter` који је задужен да упите добијене од Z39.50 сервера трансформише у CQL упите који су представљени објектом класе CQLNode. Након тога добијени CQL упит класа `MediatorService` шаље објекту класе `Wrapper` као аргумент поруке *executeQuery*. Класа `Wrapper` је део софтверске компоненте *wrapper* која ће бити описана у наставку и задужена је да добави записе који одговарају постављеном упиту. У зависности који формат записа је дефинисао Z39.50 сервер класа `MediatorService` ће слати поруку *serialize* објекту класе `UnimarcSerializer`, `Marc21Serializer` или `DublinCoreSerializer` који ће добијене записе трансформисати у XML документе. Према томе Z39.50 сервер добија записе у XML формату и даље их може серијализовати у бинарни формат како је то дефинисано Z39.50 стандардом.

На сличан начин се одвија и комуникација између SRU сервера и компоненте *mediator*, само што се тада за трансформацију добијеног упита уместо класе `RPNConverter` користи класа `CQLStringConverter`.



Слика 5.8 Дијаграм секвенци *getRecords*

5.2.1.1 Имплементација софтверске компоненте *mediator*

Софтверска компонента *mediator* чији је дијаграм класа дат на слици 5.6 имплементирана је у Јава програмском језику и имплементирана је као *web* апликација. Комуникација између серверске стране протокола и компоненте *mediator* реализована је употребом програмског пакета *Hessian* [43]. *Hessian* нуди имплементацију два једноставна протокола за комуникацију са web сервисима, један бинарни протокол и њему одговарајући XML протокол, при чему се оба ослањају на HTTP транспортни протокол. Помоћу *Hessian* пакета се на једноставан начин креира сервлет на серверској страни, а на клијентској страни *proxy* објекат преко кога се комуницира са тим сервлетом. Комуникација између *proxy* објекта и сервлета може да се одвија по једном од два протокола које имплементира *Hessian*. Односно, за потребе имплементације компоненте *mediator* направљени су интерфејс `MediatorService` и класа `MediatorServiceImpl` који представљају имплементацију сервиса на серверској страни и на основу њих је креиран сервлет. Имплементација серверске стране протокола SRU и Z39.50 описана је у наредном поглављу где је описан и начин на који се остварује комуникација са сервлетом који је део *mediator* компоненте.

Интерфејс `MediatorService` има методу `getRecords` која служи да серверска страна протокола пошаље упит и да добије резултат претраживања у формату који је дефинисан. Имплементација класе `MediatorServiceImpl` дата је на листингу 5.1.

Задатак методе `getRecords` је да утврди ког типа је упит и да на основу тога позове одговарајуће класе (`CQLStringConverter` или `RPNConverter`) које ће извршити конверзију упита. Имплементација методе `getRecords` очекује упит који је типа `String` што значи да је добијен CQL упит или може добити упит типа `RPNQuery` што значи да је добијен упит типа-1 дефинисан Z39.50 стандардом. Тип `RPNQuery` представља објектни модел упитног језика типа-1. Овај објектни модел је коришћен у имплементацији серверске стране протокола Z39.50 у оквиру пројекта JAFER што ће бити детаљније описано у шестом поглављу.

```
public class MediatorServiceImpl extends HessianServlet
implements MediatorService{

 public String [] getRecords(Object query, String format){
   Record [] records=null;
   String [] XMLrecords;
   RecordSerializer serializer;
   Wrapper wrapper=new LuceneWrapper();
   CQLNode root = null;
   if (query instanceof String){
     root = CQLStringConverter.parseQuery((String)query);
   }else if (query instanceof RPNQuery){
     root = RPNConverter.parseQuery((RPNQuery)query);
   }
   records=wrapper.executeQuery(root);
   if (records!=null) {
    serializer=RecordSerializerFactory.getSerializer(format);
     XMLrecords = serializer.serialize(records);
   }
   return XMLrecords;
  }

}
```

Листинг 5.1 Имплементација класе *MediatorServiceImpl*

Након тога, упит ће се проследити `LuceneWrapper` класи која ће извршити трансформацију CQL упита у *Lucene* упит и вратити записе који одговарају упиту. Имплементација класе `LuceneWrapper` представља део *wrapper* компоненте и детаљи њене имплементације дати су у одељку 5.2.2.1.

Уколико постоје записи у систему БИСИС који одговарају прослеђеном упиту класа `RecordSerializerFactory` ће на основу дефинисаног формата креирати одговарајућу инстанцу класе `RecordSerializer`, након чега ће се извршити трансформација записа у одговарајући формат. Класа `RecordSerializer` је апстрактна класа и њу наслеђују класе `UnimarcSerializer`, `Marc21Serializer` и `DublinCoreSeraializer`.

Објектни модел `Record` се може трансформисати у XML документ који описује UNIMARC формат. Шема документа по овом формату дата је у раду (Димић, 2007). Класа `UnimarcSerializer` имплементира трансформацију записа дефинисаног објектним моделом `Record` у XML документ по UNIMARC формату. Објектни модел записа могуће је трансформисати и у XML документ који

описује MARC21 формат, XML шема је описана у раду (Димић, 2007). Библиографски записи се у систему БИСИС чувају у UNIMARC формату па је стога прво потребно трансформисати записе у MARC21. У раду (Rudić and Surla, 2009) дат је предлог конверзије UNIMARC формата у MARC21 формат и дате су XSLT трансформације којима се запис описан UNIMARC шемом може конвертовати у документ валидан на MARC21 шему. Ове трансформације искоришћене су у имплементацији класе `Marc21Serializer`.

Записе је могуће трансформисати и у XML документ који представља библиографски запис у складу са *Dublin Core* форматом. На листингу 5.2 дата је XML шема која описује овај формат.

Ова шема се може користити за опис колекције записа и у том случају у једном XML документу се налазе сви записи и коренски елемент је `dcCollection` који садржи секвенцу елемената `dc` који представљају појединачан запис. У овој тези сваки појединачан запис је описан једним XML документом па је самим тим коренски елемент `dc`. Сваки елемент `dc` састоји се од поделемената који су дефинисани *Dublin Core* форматом.

Имплементација трансформације објектог модела `Record` у XML документ у складу са *Dublin Core* форматом представљена је класом `DublinCoreSeraializer`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema targetNamespace="info:srw/schema/1/dc-schema"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:srw_dc="info:srw/schema/1/dc-schema"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">
<import namespace="http://purl.org/dc/elements/1.1/"
 schemaLocation="http://dublincore.org/schemas/xmls/simpledc20
 021212.xsd" />
 <element name="dc" type="srw_dc:srw_dcType" />
<element name="dcCollection" type="srw_dc:dcCollectionType" />
 <complexType name="srw_dcType">
  <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="dc:title" />
    <element ref="dc:creator" />
    <element ref="dc:subject" />
    <element ref="dc:description" />
    <element ref="dc:publisher" />
    <element ref="dc:contributor" />
    <element ref="dc:date" />
```

```
    <element ref="dc:type" />
    <element ref="dc:format" />
    <element ref="dc:identifier" />
    <element ref="dc:source" />
    <element ref="dc:language" />
    <element ref="dc:relation" />
    <element ref="dc:coverage" />
    <element ref="dc:rights" />
  </choice>
</complexType>
<complexType name="dcCollectionType">
  <sequence>
    <element ref="srw_dc:dc" maxOccurs="unbounded" />
  </sequence>
</complexType>
</schema>
```

Листинг 5.2 *Dublin Core Record Schema*

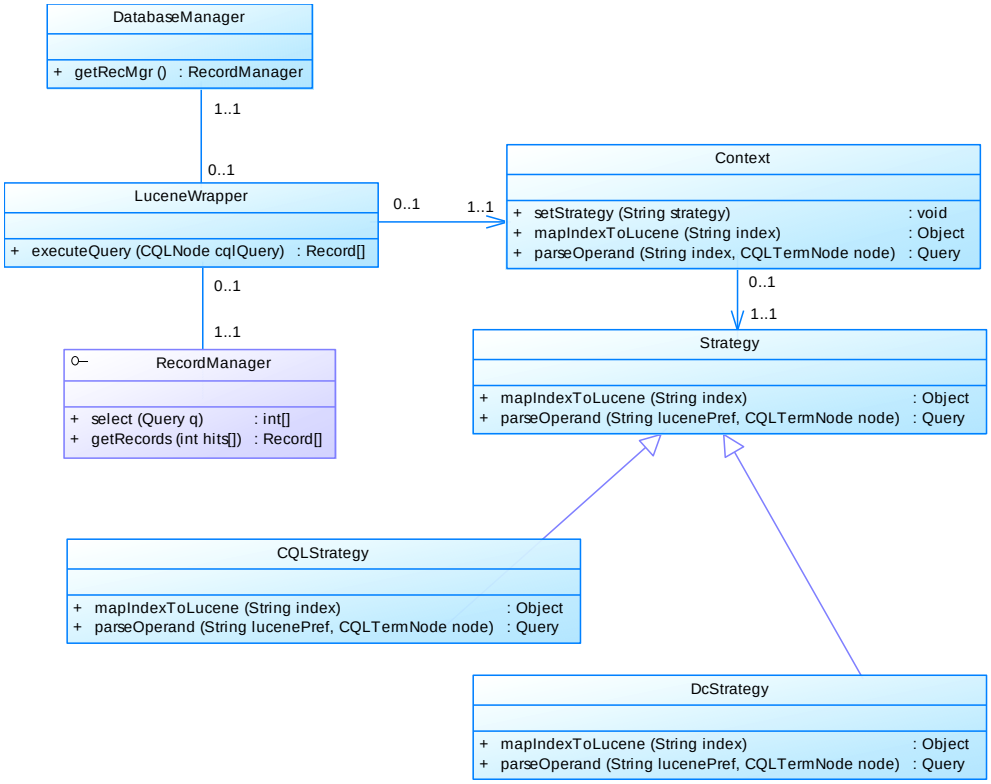
5.2.2 Софтверска компонента *wrapper*

Софтверска компонента *wrapper* има задатак да обезбеди комуникацију са системом БИСИС. Као што је описано у трећем поглављу индексирање и претраживање података у систему БИСИС врши се помоћу програмског пакета *Lucene* који има свој дефинисан упитни језик и *wrapper* компонента треба да упит типа `CQLNode` трансформише у *Lucene* упитни језик. У четвртом поглављу су анализирани концепти CQL и *Lucene* упитног језика и дат је предлог трансформације елемената дефинисаних CQL-ом у елементе дефинисане *Lucene*-ом док је у овом одељку акценат стављен на моделирање и имплементацију компоненте која ће да изврши дефинисане трансформације и да преузме записе из система БИСИС.

Дијаграм класа компоненте *wrapper* дат је на слици 5.9. Класа `MediatorService` библиографске записе из система БИСИС добија тако што упит типа `CQLNode` проследи класи `LuceneWrapper`. Класа `LuceneWrapper` представља једну имплементацију интерфејса `Wrapper` који је дат на слици 5.6. Класа `DatabaseManager` задужена је за успостављање комуникације са БИСИС системом. Интерфејс `RecordManager` је део система БИСИС и омогућава претраживање и преузимање записа из система БИСИС. Библиографски записи у систему БИСИС су представљени објектним моделом `Record` који је описан у трећем поглављу. Интерфејс `RecordManager` има две операције, једну којом се извршава упит и враћа број погодака

(`select(Query q):int[]`) и другу операцију којом се добијају библиографски записи (`getRecords(int hits[]):Record[]`).

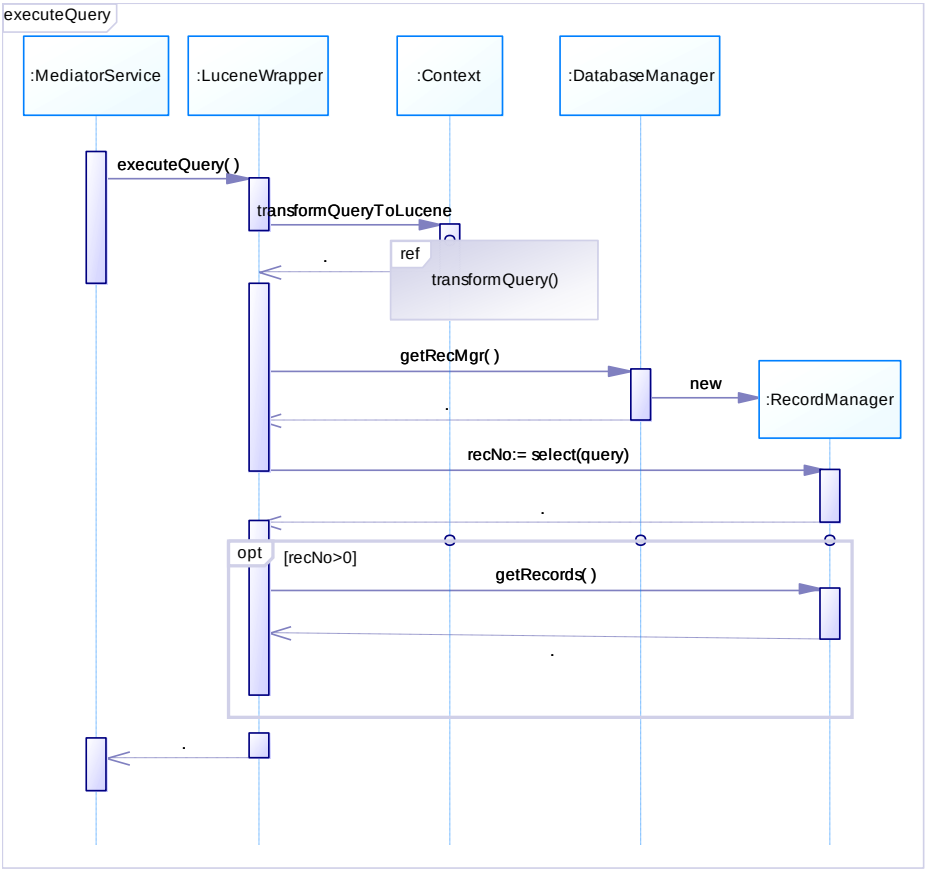
Класе `Context`, `Strategy`, `CQLStrategy` и `DcStrategy` задужене су за мапирање концепата дефинисаних CQL упитним језиком у *Lucene* језик. Класа `Context` задужена је за избор одговарајуће стратегије за парсирање у зависности коме *Context Set*-у припада елемент који треба да се трансформише. Класе `CQLStrategy` и `DcStrategy` задужене су за мапирање елемената који припадају респективно CQL односно *Dublin Core Context Set*-у у одговарајуће елементе дефинисане *Lucene* упитним језиком.



Слика 5.9 Дијаграм класа компоненте *wrapper*

На слици 5.10 дат је дијаграм секвенци којим је описано извршавање операције `executeQuery(CQLNode cqlQuery):Record[]` која је дефинисана у класи `LuceneWrapper`. Извршавање ове операције иницира објекат класе `MediatorService` слањем поруке `executeQuery` објекту класе `LuceneWrapper` који прво мора да изврши трансформацију добијеног упита у одоварајући упит дефинисан *Lucene*-ом. Класа `Context` задужена је за ову

трансформацију, што је на дијаграму приказано референцом на дијаграм *transformQuery()*, и у одељку 5.2.2.1 описан је алгоритам за трансформисање CQL упита. Након што је добијени упит трансформисан, објекат класе `LuceneWrapper` шаље поруку `getRecMgr` класи `DatabaseManager` која је задужена да успостави комуникацију са системом БИСИС, односно са текст сервером система БИСИС. Текст сервер је репрезентован преко интерфејса `RecordManager`. Класа `LuceneWrapper` прво шаље поруку *select* интерфејсу `RecordManager` који ће извршити претраживање и вратити број пронађених записа. Уколико има записа који одговарају упиту класа `LuceneWrapper` шаље поруку *getRecords* која служи за добављање записа из система БИСИС.

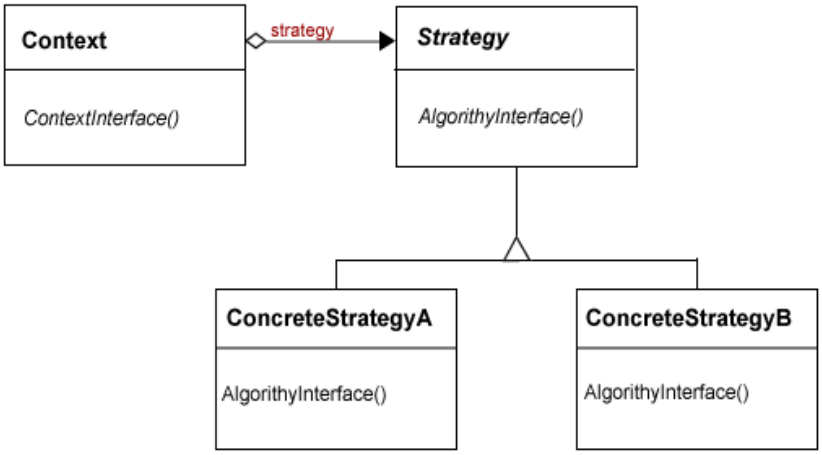


Слика 5.10 Дијаграм секвенци *executeQuery*

5.2.2.1 Трансформација CQL упита

Један од основних проблема који се јавио код трансформације CQL упитног језика у *Lucene* упитни језик је могућност да се у једном упиту комбинују елементи који су дефинисани у различитим *ContextSet*-овима. Приликом парсирања упита потребно је проверити ком *ContextSet*-у припада одређени елемент и онда применити мапирање тог *ContextSet*-а на одговарајући елемент дефинисан *Lucene*-ом. За решавање овог проблема искоришћен је дизајн патерн *strategy*. Дијаграм класа који описује овај дизајн патерн дат је на слици 5.11, која је преузета из (Gamma et al., 1994).

Дизајн патерн *strategy* припада групи патерна који описују понашање објеката (*Behavioral patterns)*, односно баве се одређивањем одговорности појединих објеката и начином на који објекти међусобно комуницирају. Основни задатак *strategy* патерна је да омогући једноставну промену алгоритма који објекат примењује у одређеном тренутку.



Слика 5.11 Дизајн патерн *strategy*
[слика преузета из Gamma et al., 1994]

Strategy патерн дефинише фамилију алгоритама и при томе сваки појединачан алгоритам је енкапсулиран у појединачни објекат. Основни елементи у овом патерну су класе `Context`, `Strategy` и `ConcreteStrategyA` и `ConcreteStrategyB`. Класа `Context` задужена је за избор и промену алгоритма на тај начин што ће креирати одговарајућу инстанцу класе која имплементира интерфејс `Strategy`. Интерфејс `Strategy` дефинише методу коју треба да имплементирају све класе које имплементирају овај интерфејс. Класа

`ConcreteStrategyA` садржи имплементацију једног конкретног алгоритма.

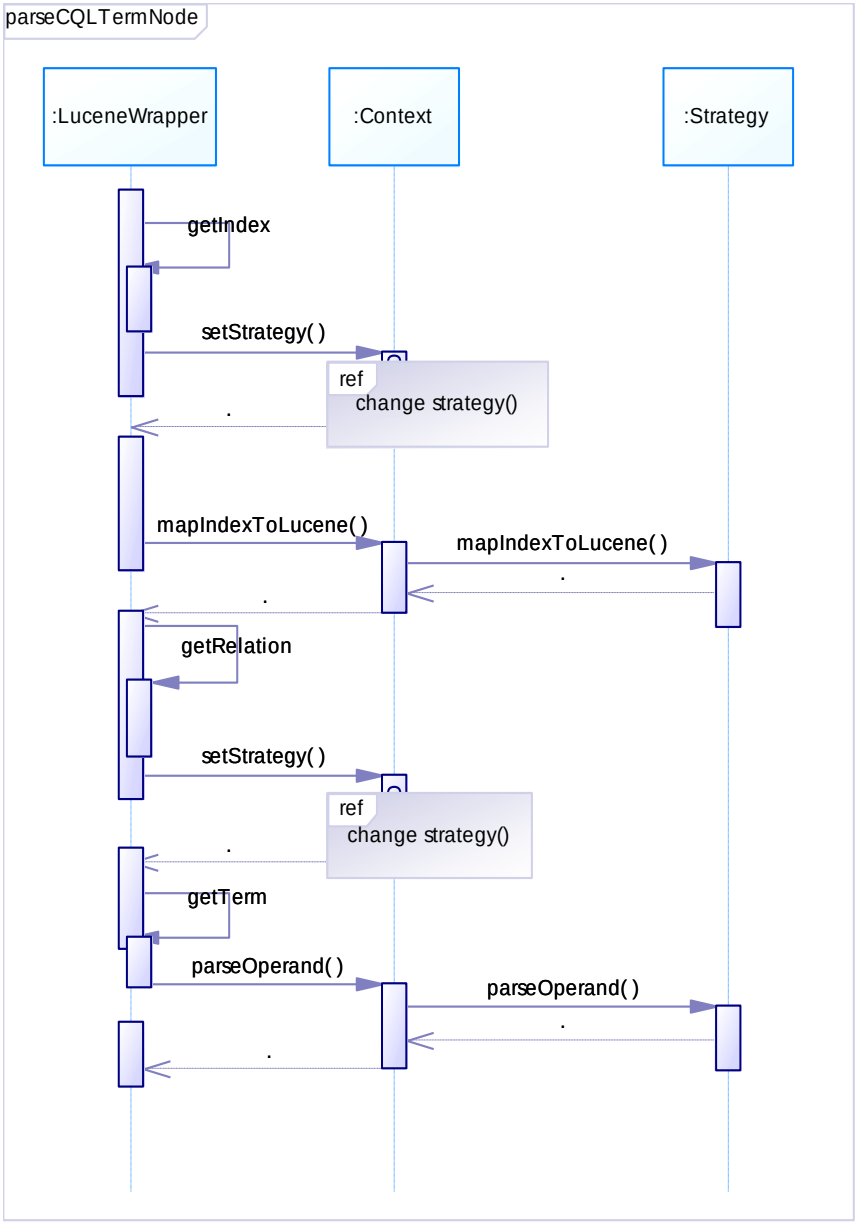
Овај дизајн патерн искоришћен је приликом трансформисања CQL упитног језика у *Lucene* упитни језик првенствено јер постоје различити *ContextSet*-ови чији елементи се различито интерпретирају.

Објектна репрезентација CQL упитног језика има структуру стабла где коренски елемент може бити један од објекат чије класе наслеђују класу `CQLNode`, а са друге стране деца коренског чвора поново могу бити објекти класе `CQLNode`, стога се приликом парсирања CQL упита рекурзивно позива операција `makeQuery(CQLNode node, Query query)` која је дефинисана у оквиру класе `LuceneWrapper`. Ова операција позиваће се рекурзивно све док чвор који треба да се обради не буде објекат класе `CQLTermNode`. Односно за сваки елемент који није типа `CQLTermNode` и самим тим садржи децу елементе типа `CQLNode` узимају се елементи који представљају његову децу и над тим елементима се примењује операција `makeQuery(CQLNode node, Query query)`. Резултат извршавања ове операције је објекат класе Query који представља објектну репрезентацију упита дефинисаног у *Lucene*-у. Добијени објекат се након изласка из рекурзије може комбиновати са осталим објектима добијеним у рекурзији и на тај начин формирати сложенији упит.

На дијаграму секвенци, приказаном на слици 5.12, описан је поступак парсирања објекта класе `CQLTermNode`. Објекат ове класе представља упит који се састоји од индекса, релације, терма и евентуално модификатора релације. Према томе класа `LuceneWrapper` прво треба да одреди индекс по ком се претражује и пошто индекс може припадати различитим *ContextSet*-овима потребно је да пошаље поруку *setStrategy*() објекту класе `Context` који ће изабрати одговарајућу стратегију за мапирање индекса на тај начин што креира инстанцу одговарајуће класе. Ово је описано дијаграмом секвенци *changeStrategy* који је дат на слици 5.13.

Објекат класе `Context` поставља одговарајућу стратегију и објекат класе `LuceneWrapper` на даље комуницира само са класом `Context` тако што шаље поруку *mapIndexToLucene,* а класа `Context` ће проследити ту поруку објекту класе `Strategy`. Класа `Strategy` је апстрактна и у ствари класа `Context` шаље поруку *mapIndexToLucene* објекту класе која наслеђује класу `Strategy` и која је креирана приликом слања поруке *setStrategy*. Након што је индекс мапиран на

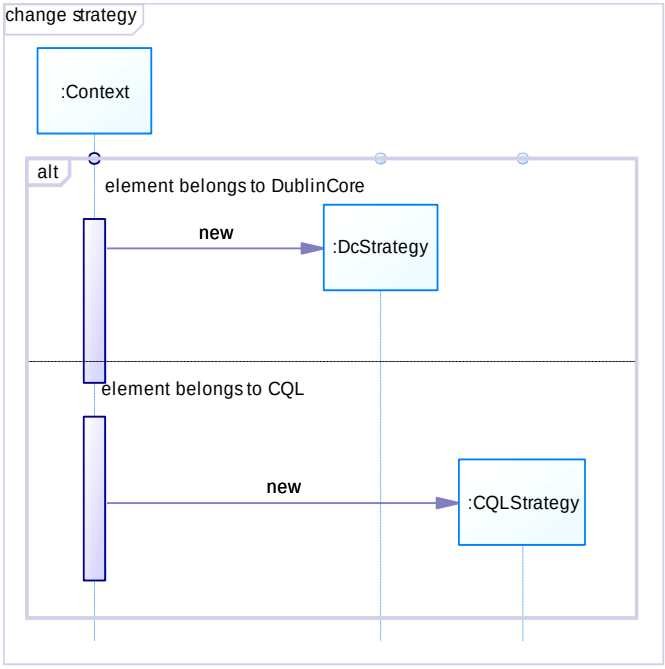
одговарајуће поље у *Lucene*, потребно је сличан поступак поновити и за релацију која може припадати неком другом *ContextSet*-у или може имати модификаторе који припадају другим *ContextSet*-овима.



Слика 5.12 Дијаграм секвенци *parseCQLTermNode*

Због тога је потребно да објекат класе `LuceneWrapper` поново пошаље поруку *setStrategy* како би се променила стратегија уколико има

потребе за тим. Након тога може се извршити мапирање релације, узимање терма и формирање операнда слањем поруке *parseOperand*.



Слика 5.13 Дијаграм секвенци *changeStrategy*

Резултат овог парсирања упита је објека класе `Query` који класа `LuceneWrapper` прослеђује интерфејсу `RecordManager` који садржи методе којима се из система БИСИС добијају подаци о библиографским записима.

5.2.2.2 Имплементација софтверске компоненте *wrapper*

На основу дијаграма класа компоненте *wrapper* извршена је имплементација у Јава програмском језику. Све класе приказане на дијаграму класа имају своју репрезентацију у виду Јава класе.

Основни задаци *wrapper* компоненте су да:

1. трансформише CQL упит у *Lucene* упит,
2. успостави комуникацију са системом БИСИС

Трансформација CQL упит у *Lucene* упит

У четвртом поглављу је описано како се основни концепти *CQL* упитног језика могу трансформисати у упитни језик дефинисан *Lucene*-ом а овде ће бити описани неки детаљи саме имплементације.

Класа `LuceneWrapper` је задужена за добављање записа из система БИСИС и она има методу

`executeQuery(CQLNode cqlQuery):Record[]`

која парсира део по део упита и за сваки елемент се проверава ком *ContextSet*-у припада и тада се позива метода `setStrategy(String strategy)` класе `Context` која креира инстанцу одговарајуће класе `Strategy` и позива њене методе које ће посматрани елемент трансформисати у складу са алгоритмом за мапирање који је имплементиран. Имплементација класе `Context` дата је на листингу 5.3.

```
public class Context {
  private Strategy strategy=null;
  public void setStrategy(String setName) {
    if (setName.equals("cql")){
        strategy=new CQLStrategy ();
    }else if (setName.equals("dc")){
        strategy=new DcStrategy ();
    }
  }
  public Object mapIndexToLucene(String index){
        return strategy.mapIndexToLucene(index);
  }
  public Query parseOperand(String index,CQLTermNode node){
        return strategy. parseOperand (index,node);
  }
}
```

Листинг 5.3 Имплементација класе `Context`

Уколико елемент припада CQL *ContextSet*-у креира се инстанца класе `CQLStrategy` или уколико припада *DoublinCore ContextSet*-у креира се објекат класе `DcStrategy`. Ове класе наслеђују класу `Strategy` која има методе на основу којих се одговарајући индекс мапира на *Lucene* префикс (метода `mapIndexToLucene(String index)`) и одговарајући операнд дефинисан CQL упитним језиком трансформише у операнд подржан *Lucene* упитним језиком (метода `parseOperand(String lucenePref,CQLTermNode node):Query`).

Увођењем патерна *strategy* омогућено је да се приликом извршавања методе `executeQuery(CQLNode cqlQuery):Record[]` и по неколико пута промени алгоритам по ком ће се парсирати упит што је био и циљ с обзиром на могућност комбиновања различитих *ContextSet*-ова у упиту. Описана имплементација *wrapper* компоненте

омогућава парсирање упита који садрже само елементе који припадају CQL *ContextSet*-у и/или *DoublinCore ContextSet*-у, мада би обезбеђивање подршке за неки нови *ContextSet* захтевало нову имплементацију интерфејса `Strategy` који би садржао алгоритам за парсирање елемената дефинисаних тим новим скупом.

Приликом мапирања индекса на одређен префикс у *Lucene* правила за мапирање се читају из одговарајућег XML документа који је дефинисан за сваки појединачан *Context Set*. На листингу 5.4 дат је пример XML документа који представља део правила за мапирање елемената *Doublin Core ContextSet*-а на префиксе у *Lucene*-у. Елементом `index` представљено је мапирање једног индекса из *Context Set*-а при чему је са елементом `name` дефинисан назив индекса, а елемент `mappingElement` представља префикс дефинисан у *Lucene*-у на који се индекс мапира.

```
<?xml version="1.0" encoding="UTF-8"?>
<contextSet identifier="info:srw/cql-context-set/1/dc-v1.1"
           name="dc">
  <indexes>
    <index>
      <name>title</name>
      <mappingElement>TI</mappingElement>
    </index>
    <index>
      <name>creator</name>
      <mappingElement>AU</mappingElement>
    </index>
    <index>
      <name>subject</name>
      <mappingElement>SB</mappingElement>
    </index>
  </indexes>
</contextSet>
```

Листинг 5.4. XML документ са правилима за мапирање *DoublinCore ContextSet*-а

Успостављање комуникације са системом БИСИС

Систем БИСИС има свој текст сервер који је базиран на *Lucene* програмском пакету и комуникација БИСИС клијента са БИСИС текст сервером може бити у локалу, односно да се обе апликације налазе на истом рачунару, или је могуће да БИСИС клијент путем HTTP протокола комуницира са текст сервером, у том случају се БИСИС клијент и текст сервер налазе на различитим рачунарима. На сличан

начин може да комуницира и *wrapper* компонента са БИСИС системом, односно са његовим текст сервером.

Наиме, интерфејс `RecordManager` има две имплементације, једна која се користи уколико се текст сервер налази на истом рачунару где и *wrapper* компонента и друга која се користи када је текст сервер на неком удаљеном рачунару. Уколико је текст сервер на неком удаљеном рачунару онда интерфејс `RecordManager` има своју имплементацију у виду Јава сервлета који је имплементиран помоћу програмског пакета *Hessian*. У оба случаја потребно је да *wrapper* компонента има информације о адреси на којој се налази база података, параметрима за приступ бази (корисничко име и лозинка), као и информацију да ли се текст серверу приступа у локалу или не, и уколико се текст сервер налази на неком удаљеном рачунару потребна је адреса тог рачунара. Ове информације чувају се у конфигурационом фајлу *config.ini* који је дат на листингу 5.4

```
[database]
driver = com.mysql.jdbc.Driver
url = jdbc:mysql://archimedes.is.im.ns.ac.yu/bisis
username = bisis
password = bisis

[textsrv]
standalone =no

recmgr = http://10.5.8.9/bisis/RecMgr
```

Листинг 5.4 Конфигурациони фајл *config.ini*

Класа задужена за успостављање комуникације је класа `DatabaseManager` и она има само једну методу `getRecMgr()` чији задатак је да утврди на који начин се приступа текст серверу и да на основу тога врати одговарајућу инстанцу интерфејса. На листингу 5.5 дат је део имплементације методе `getRecMgr`().

```
public static RecordManagerImpl getRecMgr(){
....
RecordManager recMgr;
if (standalone) {
  try {
    Class.forName(iniFile.getString("database", "driver"));
    GenericObjectPool connectionPool =
                              new GenericObjectPool(null);
    DriverManagerConnectionFactory connectionFactory =
          new DriverManagerConnectionFactory(
              iniFile.getString("database", "url"),
```

```
                iniFile.getString("database", "username"),
                iniFile.getString("database", "password"));
    PoolableConnectionFactory poolableConnectionFactory =
         new PoolableConnectionFactory(connectionFactory,
                connectionPool, null, null, false, false);
    PoolingDataSource poolingDataSource =
                    new PoolingDataSource(connectionPool);
    RecordManagerImpl recMgr1 = new RecordManagerImpl();
    String osname = System.getProperty("os.name");
    if (osname.equals("Linux"))
        recMgr1.setIndexPath("/opt/lucene-index");
    else
        recMgr1.setIndexPath("C:/lucene-index");
    recMgr1.setStandalone(true);
    recMgr1.setDataSource(poolingDataSource);
    recMgr = recMgr1;
  } catch (Exception ex) {
      log.error(ex);
  }
} else {
  try {
    Class.forName(iniFile.getString("database", "driver"));
    String tekstsrv= iniFile.getString("textsrv", "recmgr");
    HessianProxyFactory proxyFactory =
                                  new HessianProxyFactory();
    recMgr =(RecordManager)proxyFactory.create(
                           RecordManager.class, tekstsrv);
  } catch (Exception ex) {
     log.error(ex);
  }

 }

 return recMgr;

}
```

Листинг 5.5 Метода `getRecMgr`

Предложена архитектура заснована на *mediator/wrapper* приступу у систему БИСИС се може искористити за креирање једног заједничког електронског портала, чије би чланице могле бити све библиотеке које користе систем БИСИС. У том случају постојала би једна заједничка компонента *mediator* и више *wrapper* компоненти (за сваку библиотеку по један *wrapper*). *Mediator* би слао упите *wrapper* компонентама, док би сваки *wrapper* био у стању да успостави комуникацију са конкретном библиотеком. Након добијања резултата компонента *mediator* би требало да споји резултате добијене од свих *wrapper* компоненти, затим да избаци дупликате и евентуално сортира

резултате тако да крајњи корисник има утисак да је претражио једну заједничку базу података.

Поглавље 6

Имплементација протокола SRU и Z39.50

У претходном поглављу описана је софтверска компонента *intermediary* која је посредник између система БИСИС и серверске стране протокола за размену података. Међутим да би се подаци могли преузимати од система БИСИС потребно је имплементирати серверску страну одређеног протокола. Два најчешће коришћена протокола у области библиотекарства су свакако SRU и Z39.50 и у овом поглављу су описане имплементације серверских страна ових протокола.

6.1 СЕРВЕРСКА СТРАНА ПРОТОКОЛА SRU

SRU протокол има два транспортна механизма за пренос порука, један механизам је размена порука путем URL адресе а други механизам је размена порука помоћу SOAP протокола. У овој дисертацији имплементиран је само транспортни механизам који подразумева коришћење SOAP протокола. Спецификација SRU протокола је дата у виду WSDL (*Web Service Desrption Language*) докумената али тако да је сваки елемент WSDL спецификације представљен појединачним XML документом.

6.1.1 WSDL документ SRU протокола

WSDL представља XML документ који се користи за опис web сервиса, односно за опис операција које се могу позивати и структура података која се размењују у комуникацији. WSDL документ се састоји од седам елемената:

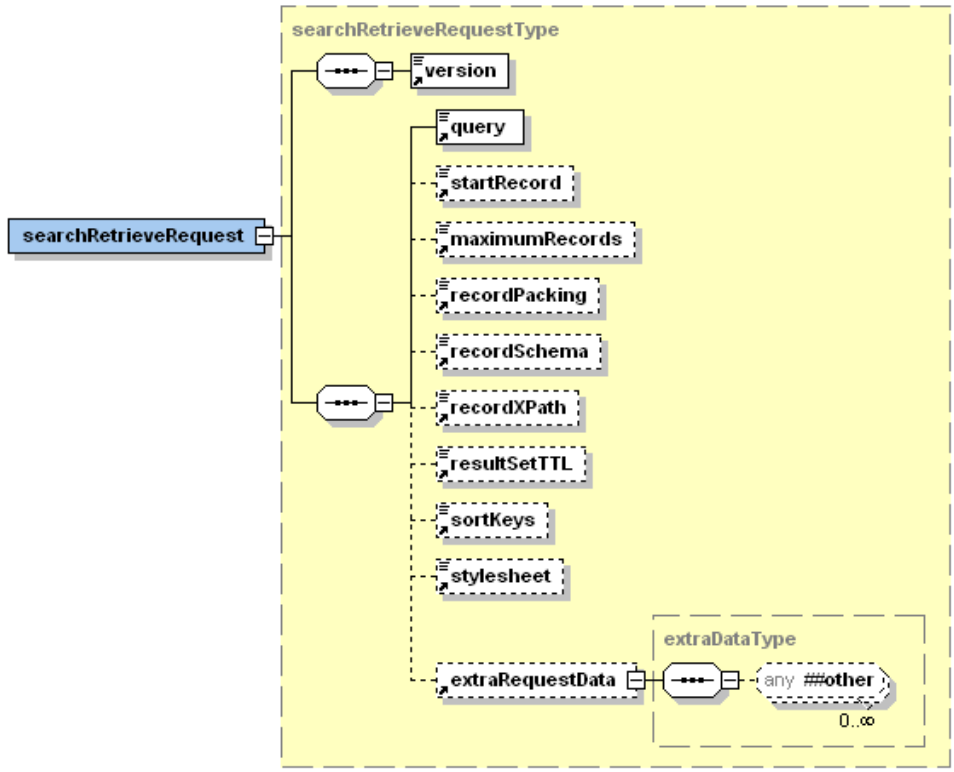
- *types*
- *message*
- *operation*
- *portType*
- *binding*
- *port*
- *service*

Елемент *types* који је дефинисан WSDL спецификацијом користи се за опис типова или структура података који се користе приликом размене порука. Пошто је изабран *document/literal wrapped* патерн за размену порука свака порука представљена је одговарајућим XML елементом који је специфициран XML шемом. XML шемом дефинисано је шест елемената за сваки сервис који је предвиђен SRU стандардом по два елемента, порука која представља захтев и порука која представља одговор. Према томе дефинисани су следећи елементи:

- *searchRetrieveRequest,*
- *searchRetrieveResponse,*
- *scanRequest,*
- *scanResponse,*
- *explainRequest,*
- *explainResponse*.

Због обимности XML шеме која описује типове порука које се размењују, на слици 6.1 и слици 6.2 дати су респективно само графичке репрезентације елемената *searchRetrieveRequest* и *searchRetrieveResponse* које су генерисане помоћу алата XMLSpy [19]. Комплетна XML шема дата је на адреси [44].

Поделементи елемента *searchRetrieveRequest* представљају параметре које порука *searchRetrieveRequest* може да садржи и они су детаљно описани у другом поглављу ове дисертацији. Слично, поделементи елемента *searchRetrieveResponse* представљају параметре које може садржати порука *searchRetrieveResponse*.

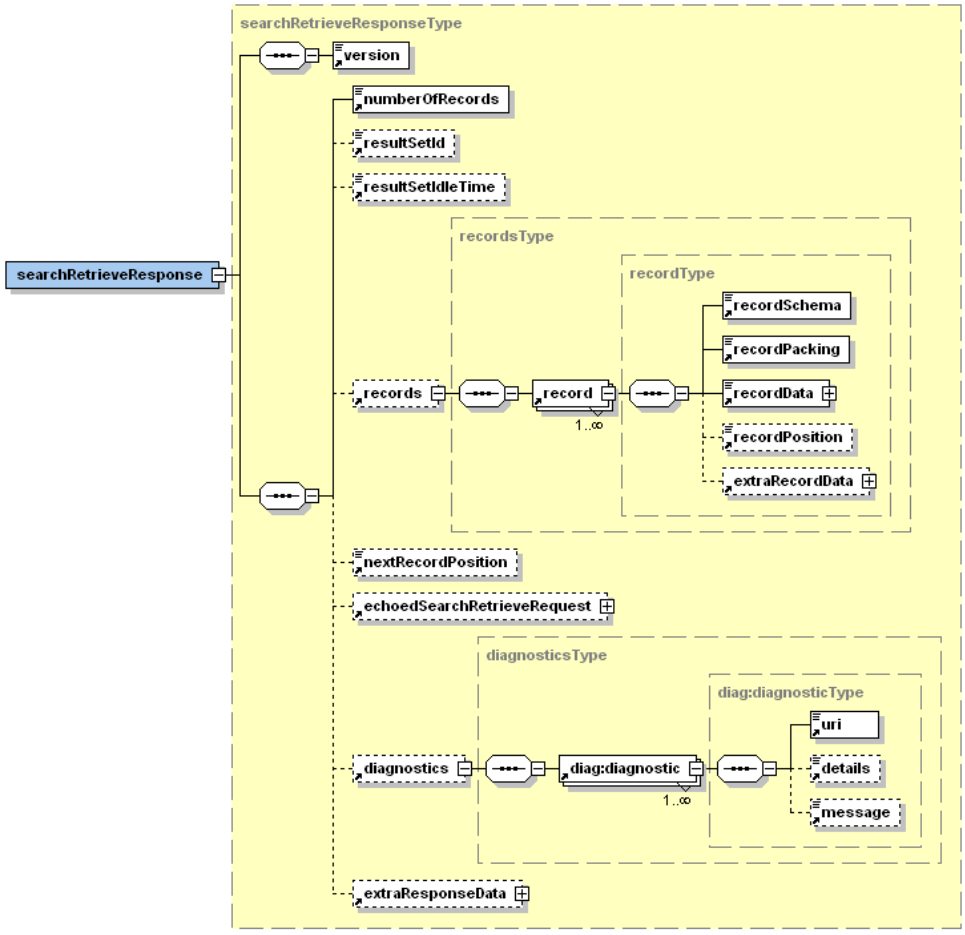


Слика 6.1 XML Елемент *searchRetrieveRequest*

Елемент *message* дефинише поруке које се размењују између клијентске и серверске стране протокола и на листингу 6.1 дат је део WSDL документа који описује поруке које се размењују када се позива сервис *SearchRetrieve* дефинисан SRU стандардом. Обе поруке које су дате на листингу садрже по један елемент *part*, при чему сваки елемент *part* указује на одређени тип који је дефинисан елементом *types*.

```
<message name="SearchRetrieveRequestMessage">
   <part name="body" element="srw:searchRetrieveRequest"/>
</message>
<message name="SearchRetrieveResponseMessage">
   <part name="body" element="srw:searchRetrieveResponse"/>
</message>
```

Листинг 6.1 Елемент *message* за опис порука сервиса *SearchRetrieve*



Слика 6.2 XML Елемент *searchRetrieveResponse*

Елемент *portType* састоји се од скупа операција, представљених елементом *operation* при чему се за сваку операцију дефинишу улазни и излазни параметри помоћу елемента *message*. На листингу 6.2 дат је део XML документа који описује операције које су подржане SRU стандардом док је комплетан документ дат на адреси [45].

```
<portType name="SRWPort">
  <operation name="SearchRetrieveOperation">
    <input message="SearchRetrieveRequestMessage"/>
    <output message="SearchRetrieveResponseMessage"/>
  </operation>
  <operation name="ScanOperation">
      <input message="ScanRequestMessage"/>
      <output message="ScanResponseMessage"/>
```

```
  </operation>
</portType>
<portType name="ExplainPort">
  <operation name="ExplainOperation">
    <input message="ExplainRequestMessage"/>
    <output message="ExplainResponseMessage"/>
  </operation>
</portType>
```

Листинг 6.2 Елемент *portType* за операције подржане SRU стандардом

Елемент *binding* служи за дефинисање протокола који ће се користити приликом размене порука, као и за дефинисање формата и енкодинга поруке. За операцију `SearchRetrieveOperation`, чији елемент *binding* је приказан на листингу 6.3, изабран је HTTP протокол за транспорт SOAP поруке и *document/literal* стил. Комплетан листинг елемента *binding* за SRU протокол дат је на адреси [46].

```
<binding name="SRW-SoapBinding" type="SRWPort">
  <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="SearchRetrieveOperation">
    <soap:operation soapAction="" style="document"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>...
</binding>
```

Листинг 6.3 Елемент *binding* за операцију SearchRetrieveOperation

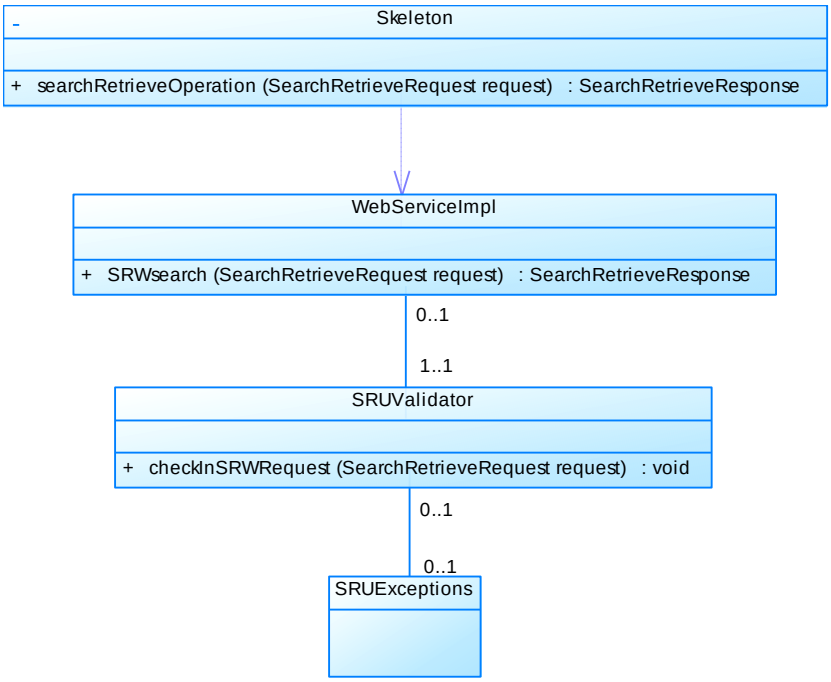
Елемент *service* дефинише физичку локацију web сервиса и на листингу 6.4 дат је пример дефиниције web сервиса који је инсталиран на локалном рачунару.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions .....name="SRW"> ...
 <service name="SRWSampleService">
   <port name="SRW" binding="SRW-SoapBinding">
      <soap:address location="http://localhost:8080/"/>
   </port>
   <port name="ExplainSOAP" binding="Explain-SoapBinding">
      <soap:address location="http://localhost:8080/"/>
    </port>
  </service>
</definitions>
```

Листинг 6.4 Елемент *service*

6.1.2 Моделирање SRU web сервиса

Дијаграм класа серверске стране протокола SRU дат је на слици 6.3. Класа `Skelteon` представља класу која је задужена за прихватање и слање SOAP порука клијенту. Имплементација пословне логике *web* сервиса представљена је класом `WebServiceImpl`. Када класа `Skelteon` добије SOAP поруку, она ту поруку прослеђује класи `WebServiceImpl`, односно операцији `SRWsearch` која је задужена за обраду поруке и формирање нове поруке која представља одговор.



Слика 6.3 Дијаграм класа SRU web сервиса

Приликом обраде поруке, потребно је проверити да ли је та порука валидна, односно да ли сервер може да процесира тај захтев. То значи да се проверава верзија SRU протокола коју клијент користи, затим да ли упит садржи само оне елементе које сервер подржава и на крају да ли сервер може да врати записе у формату који је клијент дефинисао у захтеву. Класа `SRUValidator` задужена је за проверу валидности поруке. Елементи прописани SRU стандардом које серверска страна подржава описани су XML документом који је инстанца *Explain* шеме описане у другом поглављу.

На листингу 6.5 дат је део XML документ који описује елементе који су подржани од серверске стране протокола SRU у имплементацији описаној у овој дисертацији. Елемент `indexInfo` садржи два поделемента На основу листинга може се закључити да серверска страна подржава `set` којим је дефинисано да серверска страна протокола SRU подржава *CQL* и *DublinCore Context* Set. Елементом `schema` у оквиру елемента `schemaInfo` дефинисано је да је подржан само *DublinCore* формат записа који је дефинисан одговарајућом шемом.

```
<?xml version="1.0" encoding="UTF-8"?>
<explain xmlns="http://explain.z3950.org/dtd/2.0/">
  <serverInfo protocol="SRW" version="1.2">
    <host>localhost</host>
    <port>8080</port>
    <database>axis/services/SRW</database>
  </serverInfo>
  <indexInfo>
    <set name="cql" identifier="info:srw/cql-context-
             set/1/cql-v1.2">
      <title>CQL</title>
    </set>
    <set name="dc" identifier="info:srw/cql-context-set/1/dc-
             v1.2">
      <title>Dublin Core</title>
    </set>
    <index>
      <title>Naslov dela</title>
      <map>
        <name set="dc">title</name>
      </map>
    </index>
    <index>
      <title>Bilo koje polje</title>
      <map>
        <name set="cql">anyIndexes</name>
      </map>
    </index>
  </indexInfo>
  <schemaInfo>
    <schema identifier="info:srw/schema/1/dc-v1.1"
            sort="true" retrieve="true" name="dc">
      <title>Dublin Core</title>
    </schema>
  </schemaInfo>
  <configInfo>
    <default type="NumberOfRecords">10</default>
```

```
    <default type="contextSet">cql</default>
    <setting type="retrieveSchema">
      info:srw/schema/8/unimarcxml-v0.1
    </setting>
    <setting type="resultSetTTL">1000</setting>
    <setting type="recordPacking">xml</setting>
    <supports type="maskingCharacter">*</supports>
    <supports type="maskingCharacter">?</supports>
   ..............
  </configInfo>
</explain>
```

Листинг 6.5 Део XML документ који описује подржане елементе дефинисана SRU стандардом

Овај XML документ се користи за проверу валидности поруке коју је клијент послао, али се исто тако може користити за описивање SRU сервера путем *Explain* сервиса.

Уколико серверска страна не подржава неки од параметара наведених у поруци, класа `SRUValidator` креираће инстанце класе `SRUException` којом су описане поруке о одговарајућој грешци. У другом поглављу је било речи о обради грешака приликом слања порука путем SRU протокола и класа `WebServiceImpl` је задужена за креирање одговарајуће поруке на основу добијеног објекта `SRUException`.

6.1.3 Имплементација серверске стране протокола SRU

Серверска страна протокола SRU имплементирана је као *web* сервис у програмском језику Јава помоћу програмског пакета Apache Axis2 [47]. Apache Axis2 је *open-source*, XML *web* сервис *framework*, развијен од стране *Apache Software Foundation*. Axis2 садржи имплементацију SOAP протокола у Јави и C++, као и разне алате и API-је за имплементацију *web* сервиса.

Axis2 садржи алат *WSDL2Java* којим се могу генерисати серверска и клијенска страна web сервиса на основу WSDL документа. Приликом генерисања серверске стране *web* сервиса генеришу се Јава класе које представљају Јава бинове којима су описани XML елементи дефинисаних у оквиру елемента *types* у WSDL документу. Тако на пример за XML елемент *searchRetrieveResponse* приказан на слици 6.2 генерише се одговарајућа Јава класа са истим именом. Поред тога генеришу се још две класе:

- `SRWSampleServiceMessageReceiverInOut.java`,
- `SRWSampleServiceSkeleton.java`.

Класа `SRWSampleServiceMessageReceiverInOut` задужена је за обраду SOAP порука, односно за њихово прихватање и формирање одговора. Ова класа позива одговарајуће методе класе `SRWSampleServiceSkeleton`. Класа `SRWSampleServiceSkeleton` представља имплементацију класе `Skelteon` која је дата на дијаграму класа на слици 6.3 и она садржи операције *web* сервиса које клијент може да позива. Пословна логика *web* сервиса имплементирана је у класи `WebServiceImpl` и класа `SRWSampleServiceSkeleton` садржи само позиве одговарајућих метода које су дефинисане у класи `WebServiceImpl`. Изглед SOAP поруке коју *web* сервис прима од клијента дат је на листингу 6.6.

```
<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ns3:searchRetrieveRequest
        xmlns:ns3="http://sruserver/srw/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="ns3:searchRetrieveRequestType">
      <ns3:version>1.2</ns3:version>
      <ns3:query>dc.title=analiza</ns3:query>
      <ns3:startRecord>1</ns3:startRecord>
      <ns3:maximumRecords>1</ns3:maximumRecords>
      <ns3:recordPacking>xml</ns3:recordPacking>
      <ns3:recordSchema>
         info:srw/schema/8/unimarcxml-v0.1
      </ns3:recordSchema>
    </ns3:searchRetrieveRequest>
  </soapenv:Body>
</soapenv:Envelope
```

Листинг 6.6 SOAP порука - захтев

Из SOAP поруке коју је клијент послао види се да је клијент поставио упит `dc.title=analiza`, ова информација налази се у елементу `query`. Затим елементом `maximumRecords` дефинисано је да клијент може да прими само максимално 1 записа који су у XML формату (дефинисано елементом `recordPacking`). Записи које клијент очекују треба да буду формирани у складу са UNIMARC XML шемом што је дефинисано елементом `recordSchema`.

Web сервис задужен је само да прими поруку од клијента и да формира одговор у складу са SRU стандардом, док је за добављање записа задужена компонента *intermediary* која је описана у петом поглављу. Компонента *intermediary* је имплементирана као *web* апликација и за комуникацију са њом искоришћен је програмски пакет *Hessian*. Програмски пакет *Hessian* креира `proxy` објекат преко кога се комуницира са сервлетом (класа `MediatorService`) који је имплементиран у оквиру компоненте *intermediary*. Тај `proxy` објекат представљен је инстанцом класе `HessianProxyFactory` која има методу `create` помоћу које успоставља комуникацију са сервлетом `MediatorService` који се налази на дефинисаној URL адреси. Након успостављене конекције, w*eb* сервис позива методу `getRecords` класе `MediatorService` којој прослеђује CQL упит (параметар `query`) који је добио од клијента и информацију о формату у ком клијент очекује резултате. Резултат извршавања ове методе је низ XML докумената.

На листингу 6. 7 дат је део програмског кода који описује на који начин *web* сервис успоставља комуникацију са компонентом *intermediary*.

```
String url = "http://localhost:8083/mediator/MediatorServ";
HessianProxyFactory proxy = new HessianProxyFactory();
MediatorService mediator =(MediatorService)
             proxy.create(MediatorService.class, url);
 String []records= mediator.getRecords(query,format);
```

Листинг 6.7 Успостављање комуникације *Web* сервис са компонентом *intermediary*

Након добијања записа од система БИСИС класа `WebServiceImpl` креира одговор који је представљен објектом класе `SearchRetrieveResponse`. На основу овог објеката класа `SRWSampleServiceMessageReceiverInOut` креираће SOAP поруку која је дата на листингу 6.8.

```
<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
     <ns3:searchRetrieveResponse
         xmlns:ns3="http://sruserver/srw/"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:type="ns3:searchRetrieveResponseType">
       <ns3:version>1.2</ns3:version>
```

```
<ns3:numberOfRecords>338</ns3:numberOfRecords>
<ns3:records>
   <ns3:record>
     <ns3:recordSchema>
        info:srw/schema/8/unimarcxml-v0.1
     </ns3:recordSchema>
     <ns3:recordPacking>xml</ns3:recordPacking>
     <ns3:recordData>
      <rec>
        <lab>*****iam0#2200000*##450#</lab>
        <cf t="001">187</cf>
        <df t="100" i1=" " i2=" ">
          <sf c="a">
           20051223d1995*********scr***********
          </sf>
        </df>
        <df t="101" i1="0" i2=" ">
          <sf c="a">scr</sf>
        </df>
        <df t="102" i1=" " i2=" ">
          <sf c="a">yug</sf>
          <sf c="a">sr</sf>
        </df>
        <df t="200" i1="0" i2=" ">
          <sf c="a">Funkcionalna analiza</sf>
          <sf c="f">Aleksandar Torgašev</sf>
        </df>
        <df t="210" i1=" " i2=" ">
          <sf c="a">Beograd</sf>
          <sf c="c">A. Torgašev</sf>
          <sf c="d">1995</sf>
        </df>
        <df t="215" i1=" " i2=" ">
          <sf c="a">225 str.</sf>
          <sf c="d">24 cm</sf>
        </df>
        <df t="700" i1="1" i2="1">
          <sf c="4">070</sf>
          <sf c="a">Torgašev</sf>
          <sf c="b">Aleksandar</sf>
        </df>
        <df t="992" i1=" " i2=" ">
          <sf c="b">crolga9611-old</sf>
        </df>
      </rec>
     </ns3:recordData>
     <ns3:recordPosition>1</ns3:recordPosition>
   </ns3:record>
 </ns3:records>
```

```
    </ns3:searchRetrieveResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

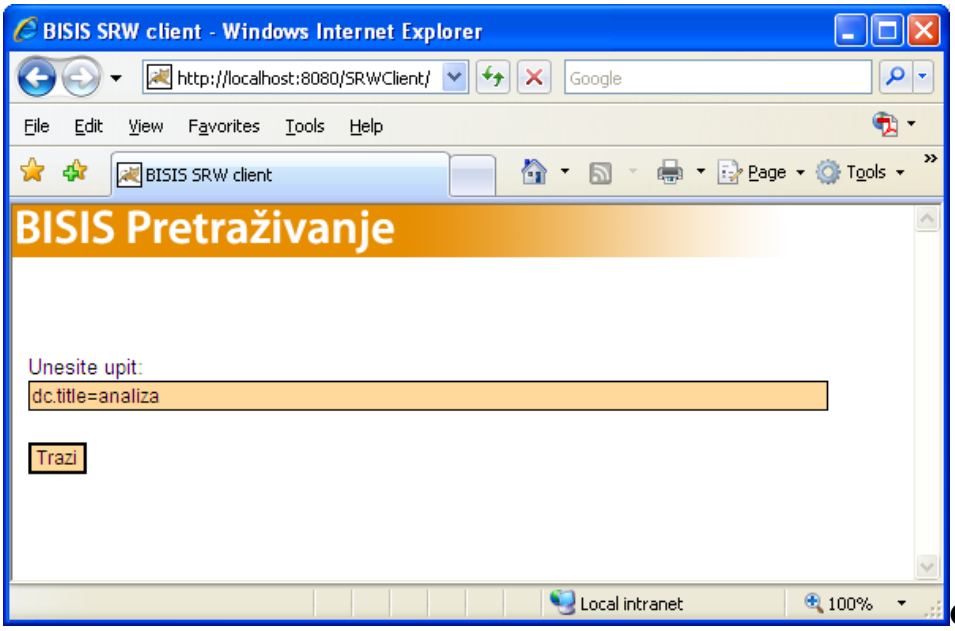
Листинг 6.8 SOAP порука – одговор

Ова SOAP порука садржи информацију о броју записа који одговарају упиту, али садржи само један запис јер је тако клијент специфицирао и тај запис је у формату дефинисаним шемом коју је клијент навео.

6.2 КЛИЈЕНТСКА СТРАНА ПРОТОКОЛА SRU

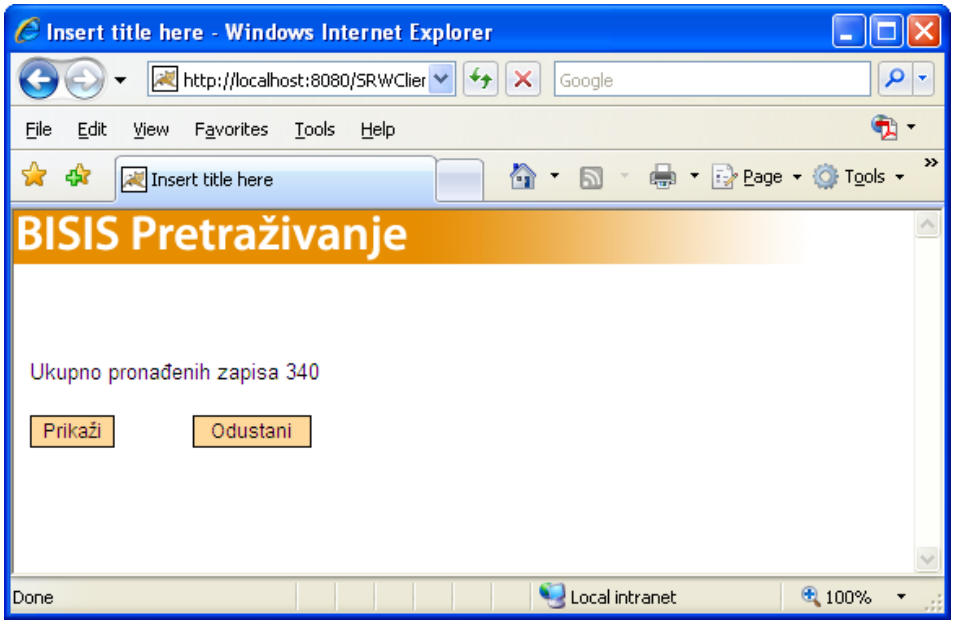
Клијентска страна протокола SRU имплементирана је као *web* апликација. За имплементацију корисничког интерфејса искоришћена је JSP технологија, док је клијентска страна која служи за позивање *web* сервиса изгенерисана помоћу пакета Axis 2.0. Изглед почетне *web* странице дат је на слици 6.8.

Ова страница намењена је за унос упита и корисник може да унесе упит у једнолиниски едитор и при томе мора да поштује синтаксу CQL упитног језика. Овде треба напоменути да је ова клијентска апликација првенствено намењена за тестирање SRU *web* сервиса а самим тим и тестирање софтверске компоненте *intermediary* и њене интеграције у систем БИСИС. Из наведених разлога није се водило рачуна о корисничком интерфејсу и уколико би овај систем требао да користи просечан корисник, морало би се посветити више пажње корисничком интерфејсу.



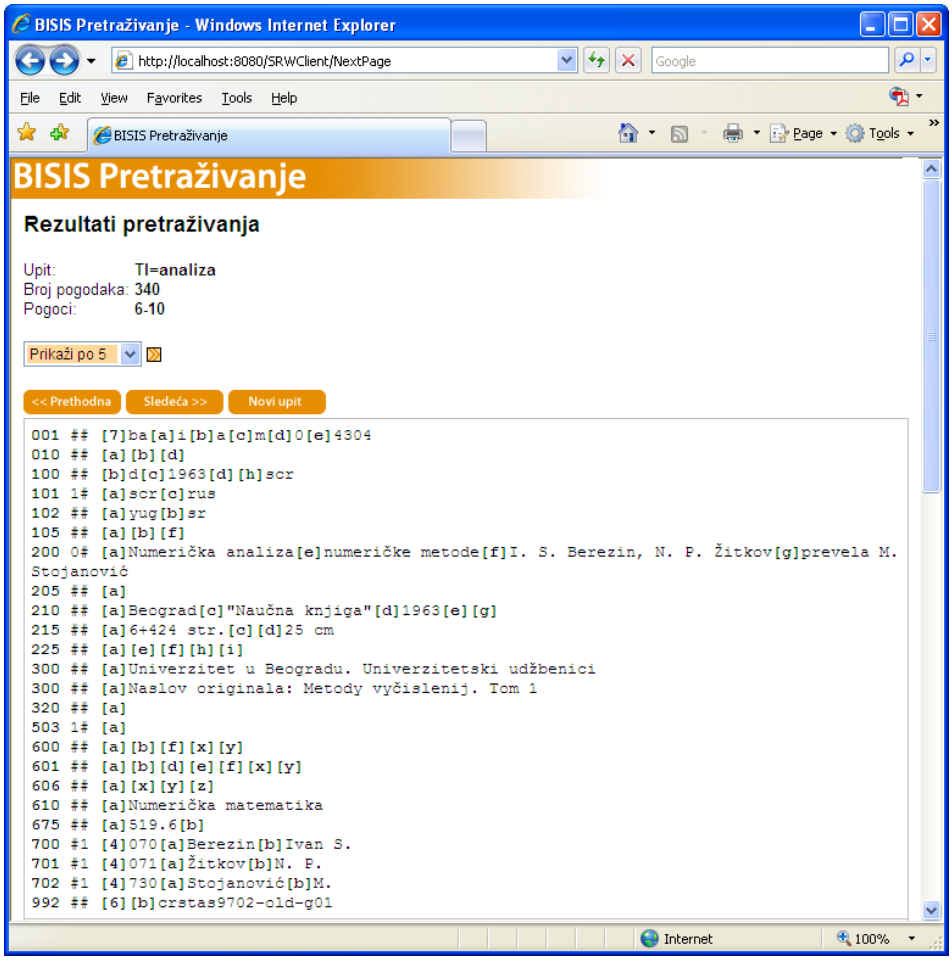
лика 6.8 Почетна страница за унос упита

SRU протокол омогућава крајњем кориснику да дефинише формат у ком жели да преузме пронађене записе и колико записа жели да преузме, али због једноставности у имплементацији ове клијентске апликације претпостављено је да клијент захтева записе по UNIMARC XML шеми и да максимално може да прихвати 10 записа. Након извршеног упита уколико није било неких грешака корисник ће добити информацију о броју пронађених погодака што је приказано на слици 6.9.



Слика 6.9 Страница са приказаним бројем погодака

Уколико су пронађени записи који одговарају задатом упиту корисник може изабрати да погледа те записе. Записи се могу приказивати по на пример 5 на једној страници и са притиском на дугме *Prethodni* и *Sledeći* омогућена је навигација кроз скуп погодака. Приказ записа намењен крајњем кориснику остављен је клијентској апликацији. Клијентска апликација на пример може помоћу XSLT трансформација добијене записе који су представљени XML документима трансформисати у формат који је примеренији крајњем кориснику. Тај формат може садржати само неке основне податке из записа, као што су аутор, наслов књиге и издавач или може бити у форми каталошког листића или као што је случај са овом клијентском апликацијом то може бити тзв. *full* формат записа. Изглед једног пронађеног записа дат је на слици 6.10.



Слика 6.10 Страница са пронађеним записом

6.3 СЕРВЕРСКА СТРАНА ПРОТОКОЛА Z39.50

У циљу илустрације функционалност компоненте *intermediary*, описаној у претходном поглављу, илустрована је комуникација серверске страна протокола Z39.50 са том компонентом. За имплеметацију серверске стране протокола Z39.50 искоришћен је пројекат JAFER [48] који садржи компоненту Z-сервер имплементирану у програмском језику Јава. JAFER пројекат је базиран на XML технологијама и омогућава једноставну имплементацију серверске стране без превеликог уплитања у детаље самог протокола. Z-сервер имплементиран у JAFER пројекту пружа подршку за прихватање и слање порука дефинисане Z39.50 стандардом и

омогућава мапирање записа представљених OAI MARC XML шемом [49] у одговарајући бинарни формат. Детаљи везани за пројекат JAFER дати су у раду (Corfield et al., 2002). Z-сервер реализован у оквиру пројекта JAFER искоришћен је и у имплементацији неких других библиотечких информационих система. Један од таквих система је описан и у раду (Cousins and Sanders, 2006) где је JAFER-ов Z-сервер интегрисан у већ постојећи виртуелни каталог како би се омогућило да клијент путем Z39.50 протокола комуницира са виртуелним каталогом који ће извршити паралелно претраживање и добављање записа из свих библиотека које чине тај каталог.

Интеграција Z-сервера у постојеће окружење захтева само прихватање упита од Z-сервера и прослеђивање тог упита компоненти *intermediary*. Компонента *intermediary* од Z-сервера добија упит типа-1 који је представљен одговарајућом објектном репрезентацијом упита која је у складу са ASN1спецификацијом. Класа `RPNConverter` која је део компоненте *mediator* задужена је за конверзију овог упита у CQL упитни језик. Записи које Z-сервер добија од компоненти *intermediary* су у XML формату, и серијализација тих записа у бинарни формат препуштена је компоненти Z-сервер.

Као што се види архитектура компоненте *intermediary* је флексибилна тако да је могуће једноставна интеграција и других имплементација Z-сервера, то не мора стриктно бити Z-сервер који је развијен у оквиру JAFER пројекта. Компонента која представља имплементацију Z-сервера треба само да омогући комуникацију са *intermediary* компонентом и да пошаље упит у оном формату како је то дефинисано у тој имплементацији Z-сервера, док је задатак *intermediary* компоненте да добијени упит трансформише у CQL упитни језик.

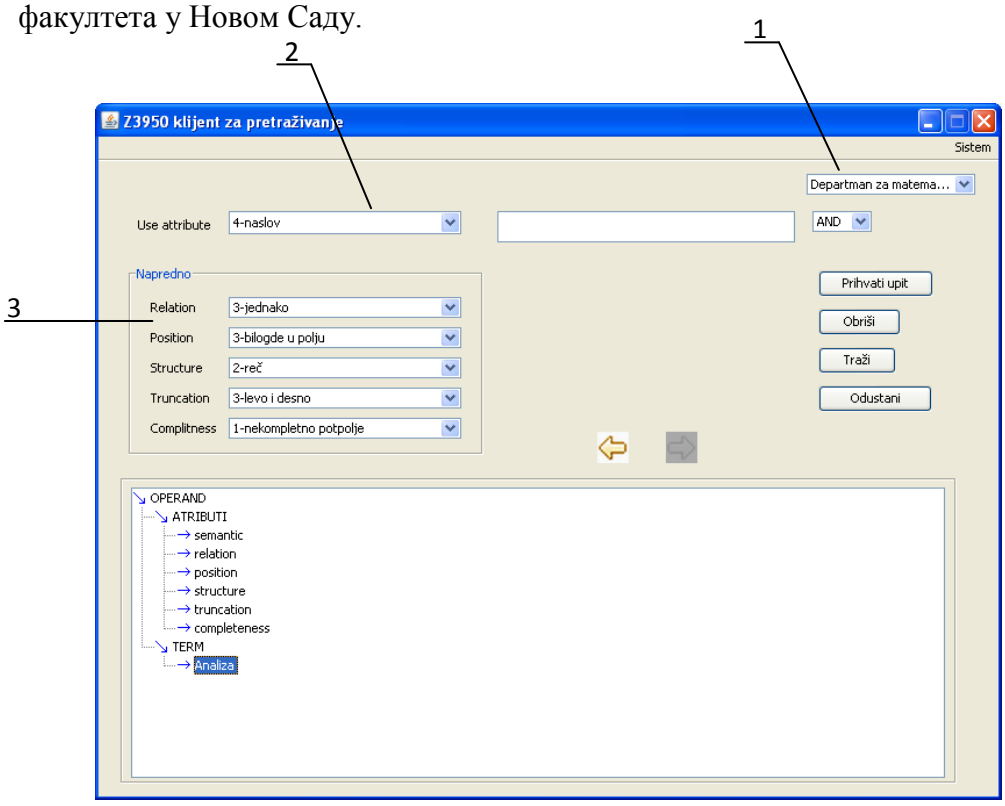
6.4 КЛИЈЕНТСКА СТРАНА ПРОТОКОЛА Z39.50

Већ је наведено да је у оквиру истраживања и рада на магистарској тези (Боберић, 2007) развијена софтверска компонента која представља Z-клијента. Резултати тог истраживања објављени су и у монографији (Боберић и Сурла, 2007) као и у раду (Boberić and Surla, 2009). У овом одељку дати су основни детаљи пројектовања и имплементације те компоненте првенствено јер је тестирање компоненте *intermediary* извршено слањем упита који је формиран помоћу тог Z-клијента. Овај едитор ће послужити као Z-клијент који ће претраживати и преузимати записе из библиотеке која ради под системом БИСИС. Једна од таквих

библиотека је библиотека Департмана за математику и информатику Природно-математичког Факултета у Новом Саду. Односно, Z-клијент пошаље поруку Z-серверу који комуницира са компонентом *intermediary* и преко ње добија записе из система БИСИС.

Едитор за претраживање и преузимање библиографских записа је имплементиран у програмском језику Јава, а за реализацију клијентске стране протокола Z39.50 такође је искоришћен пројекат JAFER.

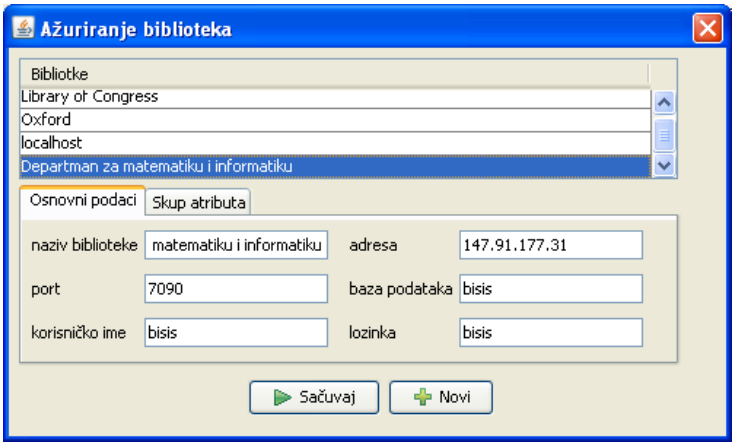
Приликом покретања апликације кориснику се отвара почетна екранска форма која је приказана на слици 6.4. На овој екранској форми корисник може изабрати библиотеку коју жели да претражи избором из падајуће листе означене бројем 1. У примерима који су приказани у наставку као тестна база искоришћена је база библиотеке Департмана за математику и информатику, Природно-математичког факултета у Новом Саду.



Слика 6.4 Почетна екранска форма едитора

6.4.1 Конфигурисање параметара библиотеке

Да би се омогућило претраживање по библиотечком фонду неке библиотеке, корисник прво мора изабрати библиотеку из падајуће листе. Уколико библиотека није дата у падајућој листи тада је корисник мора додати избором менија *Sistem* на слици 6.4. На слици 6.5 приказана је екранска форма за додавање нове библиотеке и ажурирање параметара већ постојећих библиотека која се отвара након избора из менија *Sistem*.



Слика 6.5 Екранска форма за ажурирање параметара библиотеке

Ова екранска форма има две картице и на слици 6.5 приказана је картица *Osnovni podaci*. Притиском на дугме *Novi* кориснику се омогућава унос потребних параметара за конекцију као што су назив библиотке, адреса, назив базе података и томе слично. Име библиотеке јединствено означава одређену библиотеку. Z-сервер који је интегрисан у систем БИСИС има следеће параметре неопходне за успостављање конекције:

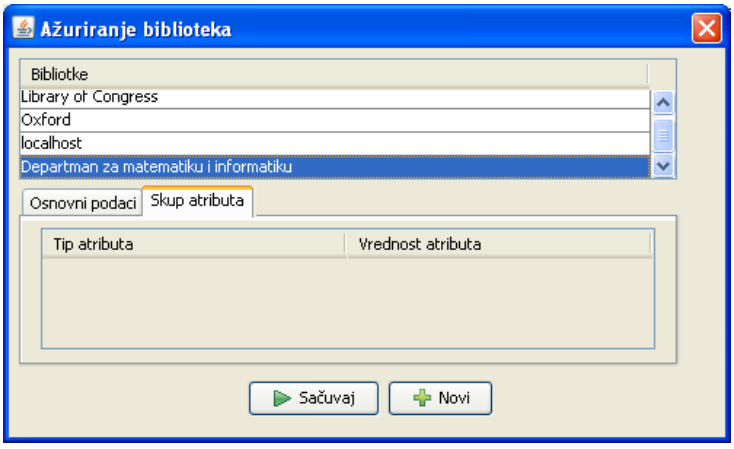
```
назив библиотеке:Библиотека департмана за математику и
информатику
адреса: 147.91.177.31
порт:7090
корисничко име: бисис
лозинка:бисис
база:бисис
```

Уколико корисник није задовољан унетим параметрима притиском на дугме *Novi* поништавају се све унете вредности. Притиском на дугме

Sačuvaj сви унети подаци се чувају у XML фајлу који садржи за сваку библиотеку која се може претраживати помоћу овог едитора.

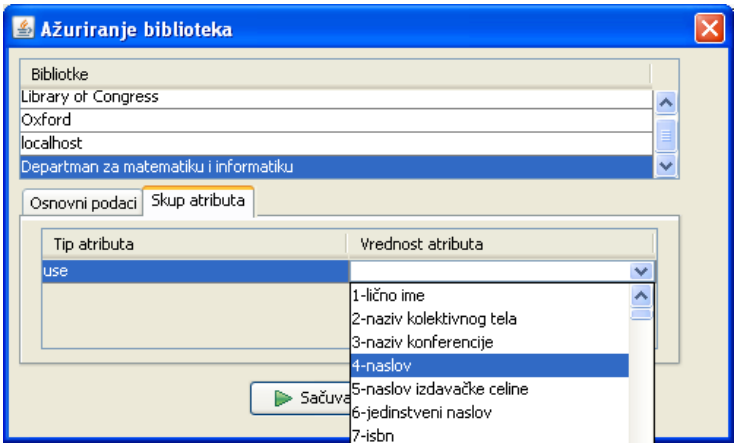
Кориснику је такође омогућена и промена постојећих података везаних за одређену библиотеку. Селектовањем једне од библиотека из табеле приказане у горњем делу екранске форме (слика 6.5) приказују се и сви остали параметри везани за ту библиотеку. Корисник може да измени неке од параметара и притиском на дугме *Sačuvaj* извршиће се ажурирање. Слично, селектовањем неке библиотеке и притиском типке *Delete* на тастатури врши се брисање селектоване библиотеке.

Картицом *Skup atributa* дефинисани су атрибути које изабрана библиотека подржава. За правилно конфигурисање Z-клијента потребно је унети вредности атрибута који су подржани од стране сервера, како корисник не би био у могућности да формира упит који садржи атрибуте који нису подржани. На овај начин смањује се број грешака које могу настати због лоше формираног упита. Екранске форме за додавање нових атрибута приказан је на слици 6.6 и на њој се види да је селектована библиотека Департмана за математику и информатику и за коју није додат ни један атрибут.



Слика 6.6 Екранска форма за ажурирање параметара библиотеке-картица *Skup atributa*

Притиском на дугме *Novi* додаје се нови атрибут. Корисник је потребно да из падајуће листе изабере врсту атрибута (*Use, Relation, Stucture, Position,Truncation, Completness*) и за изабрану врсту атрибута изабере одређену вредност из падајуће листе као што је приказано на слици 6.7. Притиском на дугме *Sačuvaj* врши се чување унетих вредности.



Слика 6.7 Картица *Skup atributa* - додавање атрибута

Z-сервер библиотеке Департмана за математику и информатику подржава подскуп атрибута дефинисаних у скупу *bib-1* и подржане вредности атрибута из тог скупа дати су у табели 6.4.

Назив атрибута	**Вредности атрибута**
Use	4 - наслов
	7 - ISBN
	8 - ISSN
	14 - УДК број
	21 - предметна одредница
	31 - датум издавања
	54 - шифра језика
	59 - место издавања
	1003 - аутор
	1018 - издавач
Relation	1 - мања
	2 - мање или једнако
	3 - једнако
	4 - веће или једнако
	5 - веће
	6 - различито
Structure	1 - фраза
	2 - реч
Position	2 - први у потпољу
	3 - било где у пољу

Назив атрибута	**Вредности атрибута**
Truncation	1 - десно одсецање
	2 - лево одсецање
	3 - лево и десно одсецање
	100 - без одсецања
Completness	1 - непотпуно потпоље
	2 - потуно потпоље

Табела 6.4 Подржане вредности атрибута из скупа *bib-1*

6.4.2 Претраживање по вредностима *Use* атрибута

У овом одељку на следећем примеру описана је употреба едитора приликом претраживања по вредностима *Use* атрибута.

Пример 1: Пронаћи све библиографске записе који у наслову садрже реч *Analiza*.

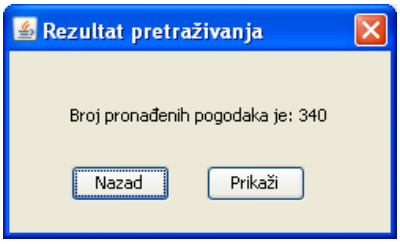
Да би се задовољио постављени задатак корисник прво мора да изабере одговарајући *Use* атрибут из падајуће листе означене бројем 2 на слици 6.4 чија вредност је *4-naslov*, и служи за претраживање по наслову. Ефикасно претраживање обавезује корисника да познаје семантику вредности *Use* атрибута. .

Бројем 3, на слици 6.4 означене су падајуће листе из којих се бирају напредни атрибути из скупа *bib-1* и то ће бити детаљније објашњено у одељку 6.3.4. Атрибути који су груписани у напредне атрибуте су опциони и уколико их корисник не селектује апликација ће узети подрезумеване вредности тих атрибута које су дефинисане у оквиру саме апликације. У овом конкретном примеру корисник не мора да селектује напредне атрибуте јер постављени упит то не захтева. Терм *Analiza* који се тражи уноси се у једнолинијски едитор.

Притиском на дугме *Prihvati upit* врши се формирање упита који је приказан у доњем делу екранске форме на слици 6.4. Формирани упит је графички репрезентован као стабло где у овом примеру имамо коренски елемент *OPERAND* који у себи садржи поделементе којима су описане вредности изабраних атрибута (поделемент *ATRIBUTI*) и вредност терма (поделемент TERM). Уколико корисник није задовољан креираним упитом притиском на дугме *Odustani* обрисаће се целокупни садржај упита који је до тада креиран.

Након притиска на дугме *Traži*, могу се јавити одређене грешке типа одбијена конекција на сервер, сервер не подржава неке атрибуте и томе

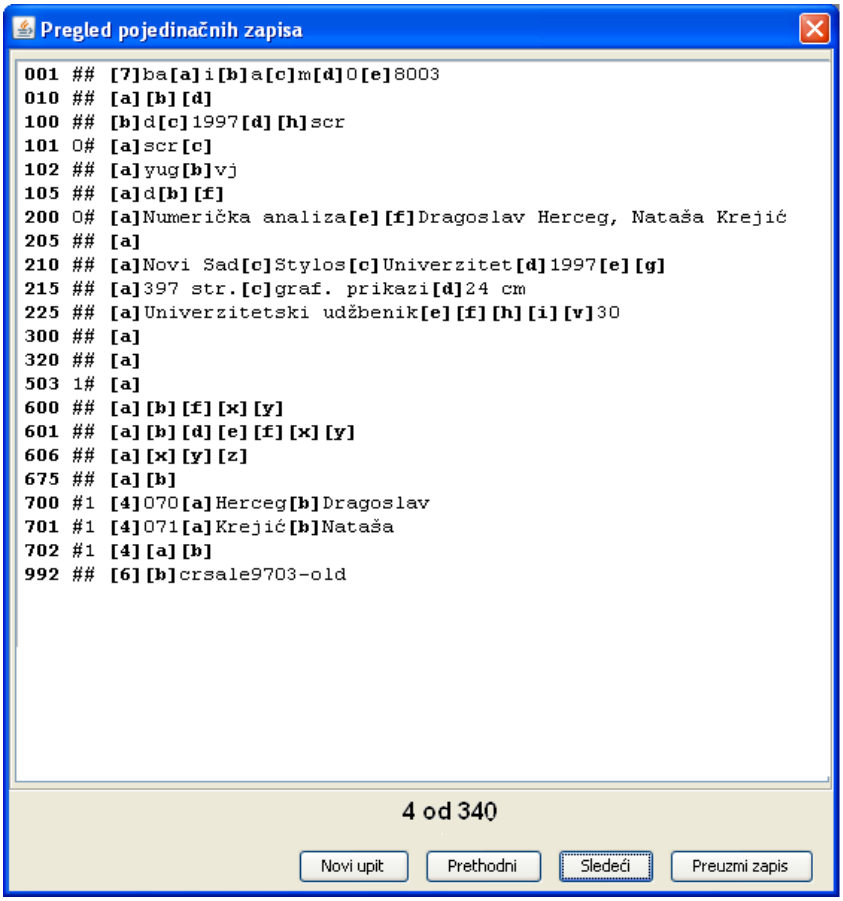
слично и у том случају ће систем генерисати изузетак а кориснику приказати одговарајући дијалог са поруком о грешци. Уколико није дошло до грешке појављује се прозор (слика 6.8) са информацијом о броју погодака.



Слика 6.8 Прозор са бројем погодака

Притиском на дугме *Nazad* корисник одустаје од прегледа записа и враћа се на почетну екранску форму (слика 6.4). Притиском на дугме *Prikaži* отвара се нова екранска форма за преглед резултата (слика 6.9).

Централни део екранске форме представља део за приказ пронађеног записа у пуном формату. Такође постоји информација о редном броју записа који се тренутно гледа. Кориснику је омогућена навигација кроз скуп добијених резултата притиском на дугме *Prethodi* и *Sledeći*, а притиском на дугме *Preuzmi zapis*, библиографски запис се снима у фајл систем у виду XML документа. Притиском на дугме *Novi upit* корисник се враћа на почетну екранску форму (слика 6.4) за постављање новог упита.



Слика 6.9 Прозор за преглед појединачних записа

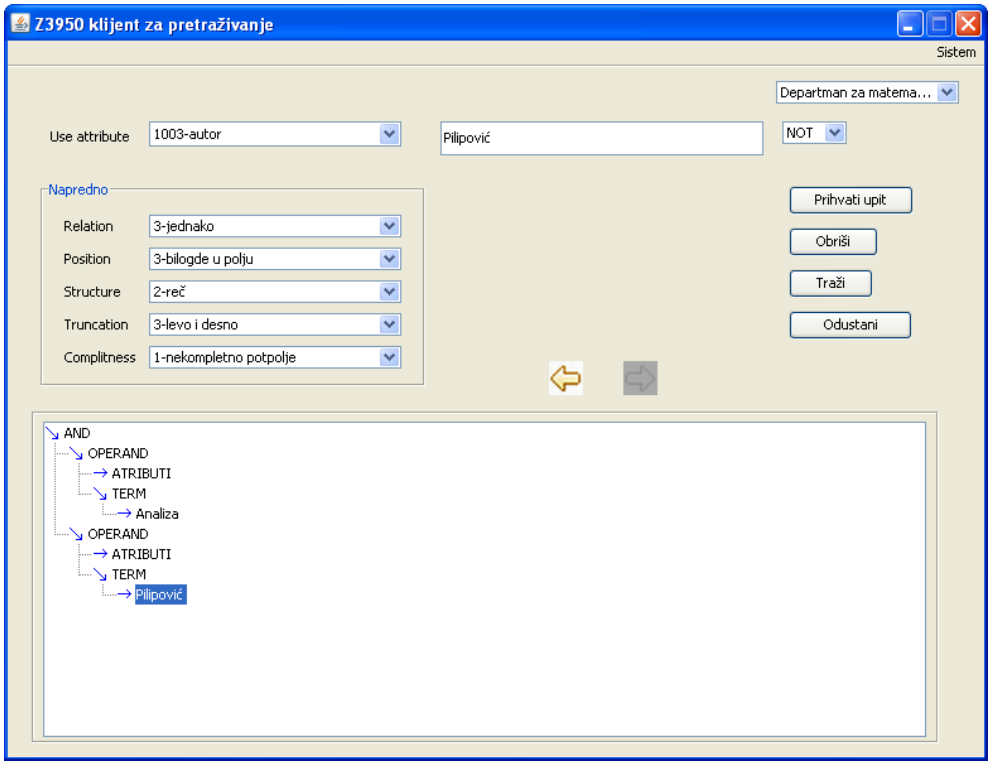
6.4.3 Претраживање употребом логичких оператора

Имплементирани едитор омогућава употребу логичких оператора *AND, NOT* и *OR* и њихов избор врши се избором из падајуће листе.

Пример 2: Наћи све записе који у наслову садрже реч *Analiza*, аутор је *Pilipović* и нису на енглеском језику.

Да би се формирао упит којима се решава постављени задатак, корисник прво мора креирати операнд где је *Use* атрибут *4-naslov,* а терм који се тражи је *Analiza*. Ово је до сада исто као и у примеру 1, једино што још треба да се уради је да се из падајуће лист изабере оператор AND којим ће се повезати следећи операнд. Када је формиран први операнд (притиском на дугме *Prihvati upit*), корисник треба да креира нови операнд. Потребно је да изабере *Use* атрибут *1003-autor*, који означава аутора књиге и унесе терм *Pilipović*. Пошто

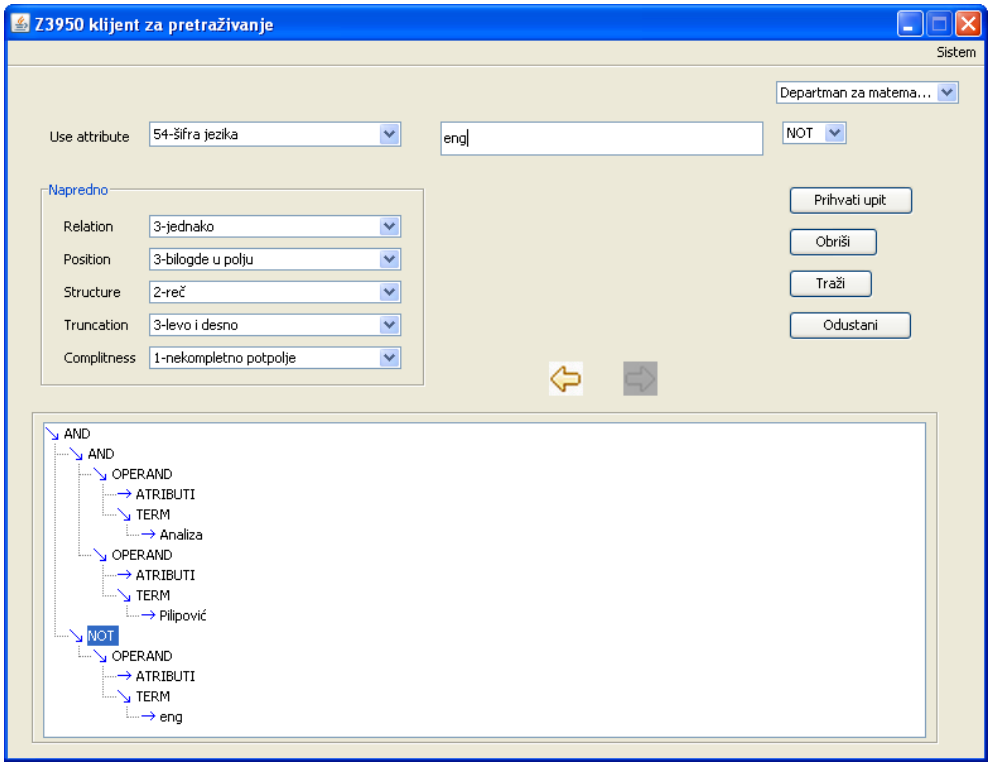
корисник не жели књиге на енглеском језику приликом формирања другог операнда било је неопходно да се изабере логички оператор NOT. Формирани операнд приказан је на слици 6.10.



Слика 6.10 Други операнд упита

На сличан начин формира се и трећи операнд, потребно је селектовати *Use* атрибут *54-šifra jezika*, који представља шифру језика на ком је написана књига, по шифарнику језика који се примењује за обраду записа по MARC 21 формату шифра за енглески језик је *eng*. На слици 6.11 приказан је формирани операнд.

Кориснику је омогућено кретање кроз формиране операнде притиском на дугмад ⇦ и ⇨ . На тај начин могуће је изменити већ креирани операнд и притиском на дугме *Prihvati upit* измењени операнд се ажурира. Такође, притиском на дугме *Obriši* брише се операнд који је тренутно селектован.



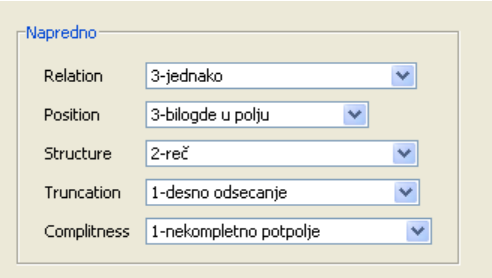
Слика 6.11 Трећи операнд упита

Графичи приказ упита дат је у доњем делу екранске форме на слици 6.11. Графички приказ одговара префиксној нотацији упита. Односно прво следи оператор па затим операнди. Тако у овом примеру коренски елемент у стаблу означава логички оператор који повезује операнде а поделементима OPERAND приказани су конкретни операнди са вредностима изабраних атрибута (поделемент ATRIBUTI) и изразом који се тражи (поделемент TERM).

Притиском на дугме *Traži* врши се претраживање изабране библиотеке и даљи принцип рада едитора је описан у првом примеру.

6.4. 4 Напредно претраживање

У овом одељку су приказани примери упита који укључују напредне атрибуте из скупа *bib*-1, и који су груписани у оквиру секције *Napredno* (слика 6.12). Сваки атрибут из овог дела допуњује вредност *Use* атрибута, што значи да се не могу селектовати само вредности ових атрибута а да при томе није селектована и вредност за *Use* атрибут.



Слика 6.12 Напредни атрибути из скупа *bib*-1

Пример 3: Пронаћи библиографске записе где је аутор *Сурла*, а књиге су објављене у периоду од 2000. до 2007. године, укључујући и године које представљају границе интервала.

Опис упит који корисник треба да постави дат је у табели 6.1.

Из табеле се види да упит има три операнда и сви су повезани логички оператором AND. Први операнд има само *Use* атрибут чија вредност означава аутора. Пошто треба да се дефинише период у ком су књиге издате, искоришћена су два операнда који имају *Use* атрибут *031-datum izdavanja* који представља датум издавања књиге. Међутим, уколико се не би за сваки операнд дефинисао и атрибут *Relation*, добио би се следећи упит: „Нађи све књиге које је написао Сурла и које су издате 2000. и 2007. године“ (што је различито од постављеног упит).

	тип атрибута	**вредност атрибута**	**терм**
Први операнд	Use	1003-autor	Сурла
оператор	AND		
Други операнд	Use	031-datum izdavanja	2000
	Relation	4-veće ili jednako	
оператор	AND		
Трећи операнд	Use	031- datum izdavanja	2007
	Relation	2-manje ili jedanko	

Табела 6.1 Табеларни приказ упит за пример 3

Из тих разлога за други операнд је дефинисан *Relation* атрибут са вредношћу *4-veće ili jednako*. *Relation* атрибут има задатак да допуни значење *Use* атрибута и у овом конкретном примеру то значи да година издања мора бити већа или једнака од 2000. Слично је за трећи операнд

дефинисан *Relation* атрибут са вредношћу *2- manje ili jedanko* и на тај начин је реализован критеријум да је година издања мања или једнака од 2007.

Пример 4: Пронаћи књиге које у наслову садрже фразу *Thinking in Java* (табела 6.2).

Овај упит се састоји само од једног операнда који има два атрибута: *Use* атрибут са вредношћу *4-naslov* и *Structure* атрибут са вредношћу *1-fraza*. Атрибутом *Structure* је дефинисан услов да треба наћи тачну фразу, да је уместо ове вредности селектована вредност *2-reč*, тада би били пронађени и записи који у наслову садрже све наведене речи, али које не морају бити у наведеном редоследу. Такође овим упитом су пронађени и записи који садрже тачну фразу у наслову али имају поред тога и неке друге изразе.

	тип атрибута	**вредност атрибута**	**терм**
Први операнд	Use	4-naslov	Thinking in Java
	Structure	1-fraza	

Табела 6.2 Табеларни приказ упит за пример 4

Уколико корисник жели да пронађе само записе који садрже тачан израз потребно је да се постави вредност *Truncation* атрибута на *100-bez odsecanja*.

Пример 5: Пронаћи књиге које у предметној одреднице садрже изразе које почињу са речју *Метод* (табела 6.3).

Тражени упит има само један операнд при чему *Use* атрибут има вредност *21-predmetna odrednica* која означава предметну одредницу, и додат је и атрибут *Truncation* са вредношћу *1- desno odsecanje*. Атрибут *Truncation* служи да омогући претраживање по почетку речи, крају речи и слично. У овом примеру је искоришћена вредност *1- desno odsecanje* што значи да предметна одредница књиге мора да почне низом карактера *Метод*. Формирани упит приказан је у табели 6.3.

Овако постављеним упитом биће пронађени и записе који у предметној одредници садрже и речи као што су: *Методика*, *Методологија*, *Методе* и томе слично. Да је било потребно да се пронађе само запис који садржи тачну реч *Метод* тада би *Truncation* атрибут имао вредност *100-bez odsecanja*.

	тип атрибута	вредност атрибута	терм
Први операнд	Use	21-predmetna odrednica	*Метод*
	Truncation	1- desno odsecanje	

Табела 6.3 Табеларни приказ упит за пример 5

Са одговарајућим променама вредности атрибута *Truncation* могу се добити записи код којих на пример наслов почиње са задатом речју, наслов се завршава са задатим низом карактера или се задати низ карактера налази у средини израза који представља наслов.

6.5 СЕРВИС ЗА АЖУРИРАЊЕ ЗАПИСА У УДАЉЕНОЈ БАЗИ

Идеја сервиса за преузимање библиографских записа је да се записи пронађу у удаљеним базама и да се након тога записи незнатно измене, у смислу додавања локацијских података, и сниме у локалну базу записа чиме би се смањило време обраде једног записа. На овај начин, уколико посматрамо библиотеку која има више огранака или било који конзорцијум библиотека, добијамо децентрализовану мрежу библиотека где свака библиотека има своју локалну базу података.

Насупрот томе стоји приступ са централизованом мрежом библиотека, односно приступ где постоји само једна база података и све библиотеке које су чланице неког конзорцијума снимају и ажурирају записе у тој бази података. Предности формирања оваквог централног каталога су посебно евидентне код библиотека које имају више физичких одвојених огранака. Код оваквих библиотека набавка нових књига се обично врши у матичној библиотеци и ту се врши и обрада записа а одређени примерци нових књига се распоређују по огранцима и сваки пут када се врши прилив нових књига огранци морају да ажурирају своје локалне базе података новоунетим библиографским записима. То значи да се врши копирање базе података у матичној библиотеци и онда се те копије дистрибуирају огранцима. Увођењем централног каталога елиминише се посао око дистрибуције ажурних копија базе података, а и смањује се редундантност података. Централни каталози имају предности поготово када се ради о пружању услуга члановима библиотеке, јер је претраживањем једне јединствене базе података могуће одговорити на питања да ли постоји одређена књига и у којим све огранцима се може задужити.

Према томе формирање централног каталога би значило да све библиотеке имају могућност да путем Интернета приступају јединственој бази података и да на тај начин снимају, ажурирају, претражују и преузимају библиографске записе. Будући да већ постоје стандардизовани протокли за претраживање и преузимање података једно од решења како да се обезбеди слање и ажурирање записа јесте да се дефинишу или нови стандарди за те намене или да се постојећи стандарди прошире потребним функционалностима. У наредном одељку изабрано је да се SRU протокол користи за слање записа ка централном каталогу и описана су неопходна проширења стандарда како би се та идеја о централном каталогу могла реализовати.

6.5.1 Проширење SRU стандарда

Стандард SRU намењен је првенствено за претраживање и преузимање библиографских записа, међутим очигледно је да би се овај стандард могао проширити тако да омогућава снимање, ажурирање и брисање записа. Односно, идеја је да се стандард тако прошири да клијент који шаље захтев у том захтеву може да упакује библиографски запис и да се он онда путем SRU протокола пренесе до неког удаљеног сервера где ће бити сачуван и индексиран. Уколико би се ово проширење имплементирало тако да се као транспортни механизам користи SOAP протокол, онда је потребно проширити постојећи WSDL документ. Проширење *web* сервиса захтева дефинисање две нове операције:

- *SaveUpdate* - служи за снимање новог записа и за ажурирање постојећег записа,
- *Delete* - служи за брисање постојећег записа.

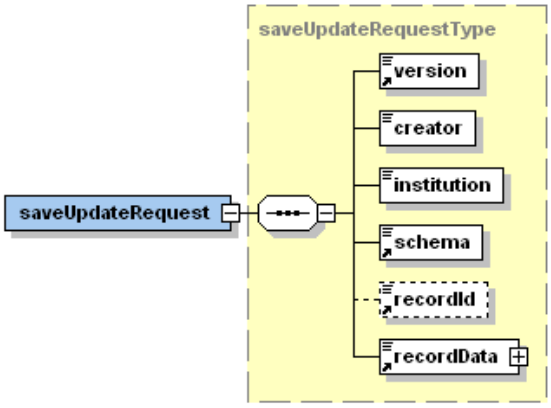
Операција *SaveUpdate*

Ова операција би се састојала од две поруке *SaveUpdateRequest* *SaveUpdateResponse*. Поруку *SaveUpdateRequest* би слао клијент и она би имала следеће параметре и сви наведени параметри су обавезни:

- *version* - овај параметар имају све операције дефинисане SRU стандардом, и означава верзију стандарда која се користи. Сервер може одбити да изврши операцију уколико се верзије стандарда не поклапају на клијентској и серверској страни протокола.
- *creator* - подаци о особи која је креирала запис.

- *institution* - овај параметар дефинише институцију, односно библиотеку којој припада креирани запис
- *schema* - записи би се размењивали у XML формату па је потребно да клијент дефинише по којој шеми је форматиран запис. Клијент може да шаље записе по оним шемама које је сервер дефинисао да подржава код сервиса *SearchRetrieve*. Уколико клијент пошаље запис форматиран по шеми коју сервер не подржава, сервер ће вратити одговарајућу грешку и неће се извршити снимање записа.
- *recordData* - овај параметар садржи сам запис и идентичан је параметру *recordData* дефинисаном у оквиру *SearchRetrieveResponse* поруке. Уколико се ради о ажурирању постојећег записа садржај постојећег записа ће бити замењен овим новим записом.
- *recordId* - овај параметра је обавезан само уколико се ради о ажурирању постојећег записа, тада се мора навести овај параметар како би се знало који запис треба да се ажурира. Иначе уколико се ради о снимању новог записа тада се не наводи овај параметар.

Пошто је изабран *document/literal wrapped* патерн за размену порука *SaveUpdateRequest* порука представљена је одговарајућим XML елементом који је специфициран XML шемом и који је дат на слици 6.4.

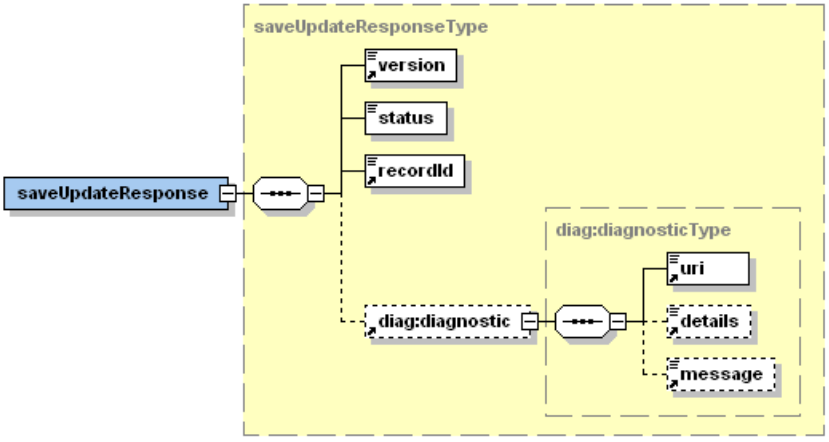


Слика 6.4 Елемент *saveUpdateRequest*

Када сервер прими ову поруку он може да изврши снимање записа и клијенту враћа поруку *SaveUpdateResponse* која садржи следеће параметре:

- *version* - верзија стандарда
- *status* - овај параметар дефинише да ли је сервер успео да изврши снимање и постоје две предефинисане вредности за овај параметар: *Succsessful* (уколико је успешно извршено снимање), *Not successful* (уколико снимање записа није извршено).
- *recordId* - овај параметар садржи идентификациони број записа, и касније се овај параметар може користити при ажурирању и брисању записа. Будући да *SearchRetrieveResponse* поруке не садржи параметар *recordId* било би потребно модификовати и ову поруку како би корисник знао идентификациони број записа који је пронашао уколико исти жели касније да ажурира или брише.
- *diagnostic* - овај параметар је идентичан параметру *diagnostic* дефинисаном у оквиру *SearchRetrieveResponse* поруке и служи за детаљније описивање грешке која се десила. Увођење овог параметара захтевало би и дефинисање додатног скупа грешака као што је то предвиђено и за сервис *SearchRetrieve*. Параметар *diagnostic* је опционалан.

XML елемент који описује ову поруку дат је на слици 6.5.



Слика 6.5 Елемент *saveUpdateResponse*

У складу са WSDL спецификацијом потребно је још дефинисати елементе *message*, *portType*, *binding* и *service* за ову операцију.

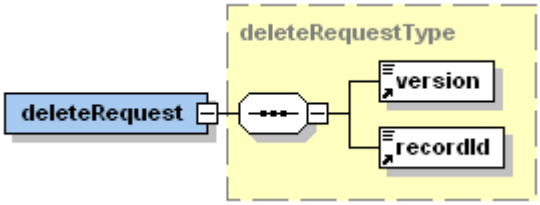
```
<message name="SaveUpdateRequestMessage">
    <part name="body" element="saveUpdateRequest"/>
</message>
<message name="SaveUpdateResponseMessage">
    <part name="body" element="saveUpdateResponse"/>
</message>
```

Листинг 6.9 Елемент *message* за опис порука операције *SaveUpdate*

На листингу 6.9 приказан је само елемент *message* док ће остали елементи бити приказани након што се опише операција *Delete*.

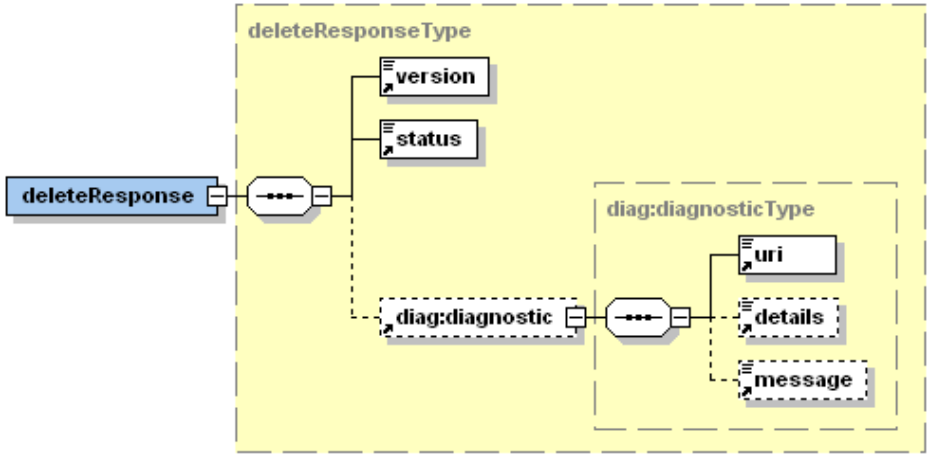
Операција *Delete*

Ова операција се такође састоји од две поруке *DeleteRequest* и *DeleteResponse*. Порука *DeleteRequest* садржи само параметре који означавају верзију стандарда и идентификациони број записа који се брише. XML елемент који описује ову поруку дат је на слици 6.6.



Слика 6.6 Елемент *deleteRequest*

Порука *DeleteResponse* од параметара садржи верзију стандарда коју сервер користи, затим параметар који описује статус извршене операција и уколико је дошло до грешке при брисању записа параметар *diagnostic* садржи детаље грешке. XML елемент који описује ову поруку дат је на слици 6.7.



Слика 6.7 Елемент *deleteResponse*

Елемент *message* за опис порука операције *Delete* дат је на листингу 6.10, док су на листинзима 6.11, 6.12 и 6.13 дати и остали елементи прописани WSDL спецификацијом.

```
<message name="DeleteRequestMessage">
  <part name="body" element="deleteRequest"/>
</message>
<message name="DeleteResponseMessage">
  <part name="body" element="deleteResponse"/>
</message>
```

Листинг 6.10 Елемент *message* за опис порука операције *Delete*

```
<portType name="UpdatePort">
  <operation name="SaveUpdateOperation">
    <input message="SaveUpdateRequestMessage"/>
    <output message="SaveUpdateResponseMessage"/>
  </operation>
  <operation name="DeleteOperation">
      <input message="DeleteRequestMessage"/>
      <output message="DeleteResponseMessage"/>
  </operation>
</portType>
```

Листинг 6.11 Елемент *portType* за операције *SaveUpdate* и *Delete*

```
<binding name="SRUExstension" type="UpdatePort">
<soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="SaveUpdateOperation">
    <soap:operation soapAction="" style="document"/>
    <input>
```

```
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
  <operation name="DeleteOperation">
    <soap:operation soapAction="" style="document"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
```

Листинг 6.12 Елемент *binding* за операције *SaveUpdateOperation* и *DeleteOperation*

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions .....name="SRUExtension">
...
 <service name="SRUExstensionService">
   <port name="SRUExtension" binding="SRUExtension ">
      <soap:address location="http://localhost:8080/"/>
   </port>
  </service>
</definitions>
```

Листинг 6.13 Елемент *service*

Један од основних проблема који се јавља код свих оваквих сервиса који пружају снимање, ажурирање и брисање података је проблем аутентификације и ауторизације, као и проблем сигурности податка који се преносе. Будући да би ове операције биле имплементиране као *web* сервиси, сама технологија *web* сервиса садржи уграђене механизме који обезбеђују аутентификацију и енкрипцију који се могу искористити у те сврхе.

Закључак

Истраживања која су описана овој дисертацији представљају развој система који ће омогућити да и други библиотечки системи могу да преузимају библиографске записе од система БИСИС. Основна идеја је да се реализује независна софтверска компонента која ће се интегрисати у постојећи библиотечки систем, при чему ће серверске стране одговарајућих протокола комуницирати путем јединственог интерфејса са овом компонентом. У циљу тестирања развијене софтверске компоненте извршена је интеграција те компоненте у библиотечки систем БИСИС, мада је сама компонента независна од система у који се интегрише.

За моделирање софтверске архитектуре система за преузимање библиографских записа коришћен је објектно-оријентисани приступ и CASE алати који подржавају UML 2.0 нотацију. Имплементација система је урађена у програмском језику Java. Систем је базиран на сервис-оријентисаној и *mediator/wrapper* архитектури. Серверске стране протокола имплементиране су као сервиси и они комуницирају са компонентом која представља *mediator*, док се за сваки библиотечки систем креира посебна *wrapper* компонента која има задатак да успостави комуникацију са библиотечким системом и да трансформише упит добијен од *mediator* компоненте у упит који подржава библиотечки систем.

Оргинални резултати истраживања приказани су у четвртом, петом и шестом поглављу.

У четвртом поглављу су анализирана два најчешће коришћена упитна језика када је реч о стандардима за претраживање и преузимање библиографских записа. Описани су концепти оба упитна језика и разматране су могућности трансформације једног упитног језика у други. Добијени резултати приказани у четвртом поглављу су следећи:

- Показано је да се упити формирани помоћу Z39.50 упитног језика могу трансформисати у еквивалентне упите формиране помоћу CQL упитног језика који се користи при употреби SRU протокола.

- Дат је предлог трансформације упита формираног помоћу CQL упитног језика у упитни језик *Lucene*. Трансформације које су разматране су касније искоришћене за имплеметацију *wrapper* компоненте за систем БИСИС чији текст сервер користи програмски пакет *Lucene*. У случају да је потребно софтверску компонтенту која је описана у овој дисертацији интегрисати у неки други библиотечки систем који користи неки други упитни језик потребно је извршити само одговарајуће трансформације CQL упитног језика у тај нови упитни језик и имплементирати одговарајућу компоненту *wrapper* док остатак софтверске компоненте остаје непромењен.

У петом поглављу су разматрани различити примери софтверских архитектура који се могу применити код реализације система за преузимање библиографских записа. Изабран је *mediator/wrapper* модел који је послужио за моделирање и имплементацију софтверске компоненте која је посредник између постојећег библиотечког система и сервиса за претраживање и преузимање. Добијени резултати приказани у овом поглављу су следећи:

- Моделирана је и имплементирана софтверска компонента која је независна од библиотечког система и протокола за преузимање података.
- Разматрана је могућност примене добијене софтверске компоненте у процесу интеграције више библиотечких система у један заједнички портал.

У шестом поглављу дат је модел и имплементација серверске и клијентске стране протокола SRU. Такође, ради верификације независности описане софтверске компоненте од конкретног протокола дат је опис серверске и клијентске стране протокола Z39.50, при чему је тестирана интеграција серверске стране протокола Z39.50 са овом компонентом и комуникација путем Z39.50 протокола. У овом поглављу анализиране су и могућности проширења постојећих стандарда у циљу креирања сервиса који ће обезбеђивати и претраживање и преузимање библиографских записа из удаљене базе али исто тако и њихово снимање и ажурирање у удаљену базу. Добијени резултати приказани у овом поглављу су следећи:

- Моделиран је и имплементиран сервис за преузимање податка путем SRU протокола.

- Имплементирана је клијентска страна SRU протокола.
- Дат је предлог проширења SRU протокола како би се омогућило слање и снимање записа на удаљеном серверу и на тај начин се формирао централни каталог.

Наведени добијени резултати у потпуности испуњавају циљеве истраживања дефинисане у уводном поглављу. Моделирана је и имплементирана софтверска компонента која има такву архитектуру да се може интегрисати у различите библиотечке системе. Са друге стране, ова компонента је флексибилна јер омогућава интеграцију различитих протокола за претраживање и преузимање података. Употребом ове софтверске компоненте могуће је формирати електронски портал који би кориснику или неком другом систему омогућавао да путем различитих протокола претражују фондове библиотека које су окупљене око портала. У случају портала који би окупљао библиотеке које користе различите библиотечке системе било би потребно имплементирати одговарајуће *wrapper* компоненте за сваки појединачни библиотечки систем.

Добијени резултати истраживања могу се искористити за развој јавног електронског библиотечког сервиса који је један од обавезних јавних сервиса у Европској Унији. Поред тога могу бити и полазна основа за развој интелигентних софтверских система за креирање, претраживање и коришћење дигиталних електронских садржаја.Такође, с обзиром на све већу тенденцију повезивања библиотека и система за учење на даљину, развој сервиса који би омогућио такву интеграцију може представљати смерницу за даља истраживања.

Литература

ARD Prasad and Patel, D., (2005), "*Lucene Search Engine: An Overview* ", DRTC-HP International Workshop on Building Digital Libraries using DSpace , 7th – 11th March, 2005 DRTC, Bangalore.

Baranov, V., Plemnek, A., Riabev, V., Sokolova, N., Sova. D, Usmanov, R., (2000), "*Review of Z39.50 servers and Z39.50 environment in Russia*", Library Hi Tech, vol. 18, no. 4, p.p 304-314

Boberić, D., Milosavljević, B. (2009), "*Generating library material reports in software system BISIS*", Proceedings of the 4th international conference on engineering technologies - ICET 2009, Novi Sad April 28-30, pp. 73-77

Boberić, D., Surla, D. (2009), "*XML Editor for Search and Retrieval of Bibliographic Records in the Z39.50 Standard*", The Electronic Library, Vol.27, No 3, pp. 474-95

Боберић Д. (2007), "*XML едитор за преузимање библиографских записа*", магистарска теза, Природно-математички факултет, Нови Сад

Боберић, Д. и Сурла, Д. (2007), "*Преузимање библиографских записа по Z39.50 стандарду*", монографија, Природно-математички факултет, Департман за математику и информатику, Нови Сад

Brown, A., Johnston, S., Kelly, K., (2002) "*Using Service-Oriented Architecture and Component- Based Development to Build Web Service Applications",* A Rational Software White Paper

Casatio, F., Kuno, H., Machiraju, V., (2004) "*Web services: Concepts, Architectures and Applications*", Springer-Verlag Berlin Heidelberg, New York

Chumbe, S., MacLeod, R., Kennedy, M., (2007), "*Building Bridges with Blocks: Assisting Digital Library and Virtual Learning Environment Integration through Reusable Middleware",* Proceedings ELPUB2007 Conference on Electronic Publishing, Austria

Coiera, E.,Walther,M., Nguyen, K.,Lovell, H. N., (2005),"*Architecture for Knowledge-Based and Federated Search of Online* Clinical Evidence*",* Journal of Medical Internet Research, Vol. 7, No.5

Corfield, A., Dovey, M., Mawby, P., Tatham, C., (2002),"*JAFER ToolKit project: interfacing Z39. 50 and XML"*, JCDL'02, Portland, Oregon, USA

Cousins, S., Sanders, A., (2006), "*Incorporating a virtual union catalogue into the wider information environment*", Journal of Documentation, Vol. 62 No. 1, pp. 120-144

Dimić, B., Milosavljević, B., and Surla, D. (2010) "*XML schema for UNIMARC and MARC 21 formats*", The Electronic Library (in press)

Dimić, B., Surla, D. (2009), "*XML Editor for UNIMARC and MARC21 cataloguing*", The Electronic Library, Vol. 27., No 3, 509-28

Димић, Б. (2007), "*XML едитор за обраду библиографске грађе", магистарска теза*, Природно-математички факултет, Нови Сад

Димић, Б., Сурла, Д.(2007), "*Обрада библиографске грађе у софтверском систему БИСИС*", монографија, Природно-математички факултет, Департман за математику и информатику, Нови Сад

Dong, L., Xing, C., Lin, J., Wang, K., (2008), "*Personalized Digital Library Framework Based on Service Oriented Architecture*", Digital Libraries: Universal and Ubiquitous Access to Information, Springer Berlin / Heidelberg

Fox, R., (2009), "*The advent of twenty-first century library services*", OCLC Systems & Services, Vol. 25 No. 1, pp. 8-15

Gamma, E., Helm, R., Johnson, R., Vlissides, J., (1994), "*Design Patterns: Elements of Reusable Object-Oriented Software*", Addison–Wesley

Hoppe, M., Razum, M., (2008), "*A SRW/U-compliant Search Service for Fedora*", Third International Conference on Open Repositories, United Kingdom,Southampton

Klemperer, K. (1987), "*Common Command Language for on-line interactive information retrieval*", Library Hi Tech 5(4), 7-12

Krafft, B.D., Birkland, A., Cramer, J.E, (2008), "*Ncore: architecture and implementation of a flexible, collaborative digital library"*, In JCDL '08: Proceedings of the 8th ACM/IEEE-CS joint conference on digital libraries, pages 313–322, New York, NY, USA, 2008. ACM.

Lam, K., Chan L.H.D., (2007), *"Building an institutional repository: sharing experiences at the HKUST Library"*, OCLC Systems & Services, Vol. 23, No.3, pp.310-323

Lavoie, B., Henry, G., Dempsey, L. (2006), "*A Service Framework for Libraries*", *D-Lib Magazine*, Vol. 12, No. 7/8

Lawler. P.J, Howell-Barber, H., (2007) „*Service-Oriented Architecture: SOA Strategy, Methodology, and Technology*", AUERBACH

Lee S.P., Chan L.P., Lee E.W., (2006) „*Web Services Implementation Methodology for SOA Application*", Proceedings of the International Conference on Industrial Informatics, IEEE, pp. 335 – 340.

Leveling, J., Veiel, D., (2007), "*Experiments on the Exclusion of Metonymic Location Names from GIR*", Lecture Notes In Computer Science, Springer Berlin / Heidelberg, vol.4730, pp. 901–904

MacFarlane, A., (2003), "*On Open Source IR*", Aslib Proceedings: New information perspectives, vol. 55, no.4, p.p 217-222

McCallum, H. S, (2006), "*A Look at New Information Retrieval Protocols: SRU, OpenSearch/A9, CQL, and XQuery",* World Library And Information Congress: 72nd Ifla General Conference And Council,Seoul, Korea

Melnik, S., Garcia-Molina, H., Paepcke, A., (2000), "*A mediation infrastructure for digital library services",* Proceedings of the fifth ACM conference on Digital libraries, p.p. 123 - 132 , Texas, United States

Milosavljević, B., (1999), "*Text server for UNIMARC records*", Master thesis, Faculty of Technical Sciences, Novi Sad, Serbia. (in Serbian)

Milosavljević, B., Boberić, D., Surla, D. (2010), "*Retrieval of Bibliographic Records Using Apache Lucene*", The Electronic Library (in press)

Milosavljević, B., Tesendić, D. (2010), "*Software Architecture of Distributed Client/Server Library Circulation*", *The Electronic Library* (in press)

Milanović, N., Malek, M., (2004), "*Current Solutions for Web Service Composition*", IEEE Internet Computing

Morgan, L. E., (2004), "*An Introduction to the Search/Retrieve URL Service (SRU)*", Ariadne, issue 40.

Rađenović, J., Milosavljević, M. and Surla, D. (2009), "*Modelling and implementation of catalogue cards using FreeMarke*r" *Program-electronic libarary and information systems*, Vol. 43, No.1, pp. 63-76

Рађеновић, Ј.(2006), "*Моделирање и имплементација библиографских каталошких листића у софтверском пакету FreeMarker*", магистарска теза, *Природно-математички факултет*, Нови Сад

Ramollari, E., Dranidis, D., Simons, J. H.,A., (2007), „*A Survey of Service Oriented Development Methodologies*", The 2nd European Young Researchers Workshop on Service Oriented Computing

Rudić, G., Surla, D., (2009), "*Conversion of bibliographic records to MARC 21 format*", The Electronic Library

Sanderson, R., Young, J., Le Van, R., (2005), "*SRW/U with OAI, Expected and Unexpected Synergies "*, D-Lib Magazine,Vol. 11, No. 2.

Smith, Alstair G., (2000), "*Search features of digital libraries*", Information Research, 5(3)

Stangeland, E., Moe, M., (2006), "*How and why is the NORA project adding value to the institutional repositories established in Norway?"*, Leuven University Press

Tansley, R., Bass, M., Stuve, D., Branschofsky, M., Chunov, D., McClellan, G., Smith, M., (2003), "*The DSpace Institutional Digital Repository System: Current Functionality*", Proceedings of the 3rd ACM/IEEE-CS joint conference on Digital libraries

Taylor, M., Dickmeiss, A. (2005), "*Delivering MARC/XML records from the Library of Congress catalogue using the open protocols SRW/U and Z39.50"*, World Library and Information Congress: 71st IFLA General Conference and Council, Oslo, Norway.

Tešendić, D., Milosavljević, B., Surla, D. (2009), "*A Library Circulation System for City and Special Libraries*", The Electronic Library, Vol. 27, No. 1, pp 162-186

Тешендић, Д. и Сурла, Д. (2007), "*Коришћење библиотечке грађе у софтверском систему БИСИС*", монографија, Природно-математички факултет, Департман за математику и информатику, Нови Сад

Тешендић, Д. (2007), "*Систем за коришћење библиотечке грађе*", магистарска теза, Природно-математички факултет, Нови Сад

Trichkov, K., Tsenov, M., Trichkova, E., (2005), "*Search and Retrieval Using Non-Proprietary Standards-Based Communications Protocol Z39.50*", Proceedings of First International Conference on Information Systems & Datagrids, Sofia, Bulgaria

Van Veen, T., Oldroyd, B., (2004), "*Search and Retrieval in The European Library*", D-Lib Magazine,Vol. 10, No. 2.

Van Veen, T.,(2006), "*Serving Services in Web 2.0"*, Ariadne, issue 47

Watkins, S. G. (2003), "*The IAMSLIC Z39.50 Distributed Library: facilitating international resource sharing through linked systems and services*", In: Bridging the Digital Divide, Proceedings of the 28th Annual Conference of the International Association of Aquatic and Marine Science Libraries and Information Centers (ed. by J. W. Markham and A. L. Duda). IAMSLIC: Fort Pierce, Florida. pp. 33-42 .

Wiederhold, G., (1992), "*Mediators in the Architecture of Future Information Systems"*, The IEEE Computer Magazine, Vol. 25, No. 3, pp. 38-49

Witten, H. I., McNab, J. R., Boddie, J. S., Bainbridge, D., (2000), "*Greenstone: A Comprehensive Open-Source Digital Library Software System*", Proceedings of the fifth ACM conference on Digital libraries
Wusteman, J., (2006), "*Realising the potential of web services*", OCLC Systems & Services: International digital library perspectives, Vol. 22 No. 1, pp. 5-9

Xiang, X., Morgan, L. E. ,(2005), *"Exploiting "Light-weight" Protocols and Open Source Tools to Implement Digital Library Collections and Services",* D-Lib Magazine, Vol. 11, No.10
Van de Sompel, H., Nelson, M., Lagoze, C., Warner, S., (2004), "*Resource Harvesting within the OAI-PMH Framework*", D-Lib Magazine.

Zarić, M., (2006), "*Elektronska razmena bibliografskih zapisa u XML formatu*", Fakultet tehničkih nauka, magistarska teza

Zia, L., (2002),"*The NSF National Science, Mathematics, Engineering, and Technology Education Digital Library (NSDL) Program : New Projects in Fiscal Year 2002*", D-Lib Magazine

Web странице

[1] Search/Retrieval via URL, http://www.loc.gov/standards/sru/ , погледано 22.01.2010

[2] ANSI/NISO Z39.50-2003, http://www.loc.gov/z3950/agency/Z39-50-2003.pdf, погледано 22.01.2010

[3] *Open Archives Initiative Protocol for Metadata Harvesting*, http://www.openarchives.org/OAI/openarchivesprotocol.html, погледано 22.01.2010

[4] Ockham, http://ockham.org/, погледано 08.03.2010

[5] Voyager Integrated Library System , http://www.exlibrisgroup.com/category/Voyager, погледано 25.01.2010

[6] YazProxy, http://www.indexdata.com/yazproxy , погледано 25.01.2010

[7] OCLC's SRU/W, http://code.google.com/p/oclcsrw, погледано 08.03.2010

[8] Dspace, http://www.dspace.org/ , погледано 08.03.2010

[9] International Standard Z39.50 Maintainance Agency, http://www.loc.gov/z3950/agency/, погледано 22.01.2010

[10] Abstract Syntax Notation One, http://www.obj-sys.com/asn1tutorial/asn1only.html, погледано 22.01.2010

[11] Basic Encoding Rules, http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf, погледано 22.01.2010

[12] Open Search protocol, http://www.opensearch.org/Home, погледано 22.01.2010

[13] Really Simple Syindication, http://cyber.law.harvard.edu/rss/rss.html, погледано 22.01.2010

[14] The Atom Syndication Forma, http://tools.ietf.org/html/rfc4287, погледано 22.01.2010

[15] OpenSearchList, http://www.opensearchlist.com/index.aspx, погледано 22.01.2010

[16] Dublin Core format, http://dublincore.org/, погледано 22.01.2010

[17] XML Schema for Validating Responses to OAI-PMH Requests, http://www.openarchives.org/OAI/openarchivesprotocol.html#OAIPMHschema, погледано 22.01.2010

[18] *Z39.50 Explain, Explained and Re-Engineered in XML*, http://explain.z3950.org/dtd/commentary.html , погледано 22.01.2010

[19] Altova XMLSpy, http://www.altova.com/xmlspy.html, погледано 29.01.2010

[20] Attribute Set bib-1, http://www.loc.gov/z3950/agency/defns/bib1.html , погледано 22.01.2010

[21] Marc Standards, http://www.loc.gov/marc/, погледано 22.01.2010

[22] UNIMARC Manual : Bibliographic Format 1994, http://archive.ifla.org/VI/3/p1996-1/sec-uni.htm, погледано 22.01.2010

[23] Dublin Core XML šema, http://www.loc.gov/standards/sru/resources/dc-schema.xsd , погледано 22.01.2010

[24] MARCXML: The MARC 21 XML Schema, http://www.loc.gov/standards/marcxml/schema/MARC21slim.xsd , погледано 22.01.2010

[25] UNIMARC XML šema, http://www.bncf.firenze.sbn.it/progetti/unimarc/slim/documentation/unimarcslim.html, погледано 22.01.2010

[26] Lucene, http://lucene.apache.org/java/docs/, погледано 08.02.2010

[27]FreeMarker, http://freemarker.sourceforge.net/ , погледано 08.02.2010

[28] IndexData, Zebra, http://www.indexdata.dk/zebra/, погледано 08.02.2010

[29]Managing gigabytes-MG, http://ww2.cs.mu.oz.au/mg/, погледано 08.02.2010

[30] Successful Searching with Dialog Command Language, http://support.dialog.com/searchaids/success/, погледано 6.03.2010

[31] Applications and web applications using Lucene, http://wiki.apache.org/jakarta-lucene/PoweredBy, погледано 08.02.2010

[32] Attribute Sets, http://www.loc.gov/z3950/agency/defns/oids.html#3, погледано 22.01.2010

[33] Ranked Query List, http://itl.nist.gov/iaui/894.02/projects/rlq/RLQ_spec_Feb96_ZIG.html, погледано 22.01.2010

[34] Contextual Query Language, http://www.loc.gov/standards/sru/specs/cql.html, погледано 22.01.2010

[35] Dublin Core elements, http://dublincore.org/documents/dces/, погледано 22.01.2010

[36] Dublin Core to MARC Crosswalk, http://www.loc.gov/marc/dccross.html, погледано 22.01.2010

[37] Mapping Dublin Core to UNIMARC, http://www.ukoln.ac.uk/metadata/interoperability/dc_unimarc.html, погледано 22.01.2010

[38] Augmented Backus-Naur Form, http://tools.ietf.org/pdf/rfc5234.pdf ,погледано 22.01.2010

[39] Index Data, http://www.indexdata.com/ , погледано 25.01.2010

[40] The European Library, www.theeuropeanlibrary.org, погледано 25.01.2010

[41] Electronic Business using eXtensible Markup Language, http://www.ebxml.org/, погледано 26.01.2010

[42] CQL-Java: a free CQL compiler for Java, http://zing.z3950.org/cql/java/, погледано 26.01.2010

[43] Hessian, http://hessian.caucho.com/doc/hessian-overview.xtp, погледано 26.01.2010

[44] SRU WSDL data types specification, http://www.loc.gov/standards/sru/xml-files/srw-types.xsd, погледано 29.01.2010

[45] SRU portType element, http://www.loc.gov/standards/sru/sru1-1archive/xml-files/srw-ports.wsdl, погледано 30.01.2010

[46] SRU binding element, http://www.loc.gov/standards/sru/sru1-1archive/xml-files/srw-bindings.wsdl, погледано 30.01.2010

[47] Apache Axis2/Java, http://ws.apache.org/axis2/, погледано 29.01.2010

[48] JAFER, http://www.jafer.org/ , погледано 29.01.2010

[49] OAIMarc Schema,http://www.openarchives.org/OAI/oai_marc.xsd, погледано 02.02.2010

Биографија

Данијела Боберић је рођена 25.02.1982. године у Зрењанину. Школске 2001/2002. године уписује Природно-математички факултет у Новом Саду, одсек за математику, смер дипломирани информатичар. Дипломирала је септембра 2005. године са просечном оценом 9,41 (девет и 41/100). Дипломски рад *Обрада XML докумената наставних планова* одбранила је са оценом 10 (десет). Носилац је награде „*Милева Марић Ајнштај*“ за најбољег студента у школској 2004/05 години.

На последипломске студије, смер Информатика уписала се школске 2005/2006 на Природно-математичком факултету у Новом Саду. Положила је све испити предвиђене планом и програмом са просечном оценом 10 (десет). Магистарску тезу под насловом *XML едитор за претраживање и преузимање библиографских записа* одбранила је у јуну 2007. године и стекла академски назив магистра информатичких наука.

У периоду од фебруара 2006. до фебруара 2007. године била је стипендиста Министарства за науку и заштиту животне средине. У марту 2007. године засновала је радни однос на Природно-математичком факултету у Новом Саду на радном месту истраживач-приправник и ангажована је на пројекту *Апстрактни модели и примена у рачунарским наукама*, који финансира Министарство науке и заштите животне средине Републике Србије. Изабрана је у звање истраживач-сарадник за ужу научну област Информациони системи на Природно-математичком факултету у Новом Саду на три године почевши од 01. јануара 2008. године. У периоду од 2007 до 2010 године држала је вежбе из предмета Информациони системи 1 и Информациони системи 2 на основним студијама, а школске 2006/2007. и на мастер студијама.

Има десет објављених научних радова од којих су два рада у међународном часопису са SCI листе, два рада са међународног скупа штампана у целини, једна монографија од националног значаја и пет радова са скупа националног значаја штампана у целини. Сви радови припадају области дисертације.

Од страних језика говори енглески.

УНИВЕРЗИТЕТ У НОВОМ САДУ
ПРИРОДНО-МАТЕМАТИЧКИ ФАКУЛТЕТ

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број: **РБР**	
Идентификациони број: **ИБР**	
Тип документације: **ТД**	Монографска документација
Тип записа: **ТЗ**	Текстуални штампани материјал
Врста рада: **ВР**	Докторска дисертација
Аутор: **АУ**	Данијела Боберић
Ментор: **МН**	др Душан Сурла, професор емеритус, ПМФ, Нови Сад
Наслов рада: **НР**	Софтверски систем за преузимање библиографских записа
Језик публикације: **ЈП**	српски (ћирилица)

Језик извода: **ЈИ**	српски/енглески
Земља публиковања: **ЗП**	Србија
Уже географско подручје: **УГП**	Војводина
Година: **ГО**	2010
Издавач: **ИЗ**	Ауторски репринт
Место и адреса: **МА**	Природно-математички факултет, Трг Доситеја Обрадовића 4, Нови Сад
Физички опис рада: **ФО**	(7/180/105/8/47/0/0) (број поглавља/страна/лит.цитата/ табела/слика/графика/прилога)
Научна област: **НО**	Информатика
Научна дисциплина: **НД**	Информационе технологије и системи
Предметна одредница/ кључне речи: **ПО** **УДК**	Претраживање и преузимање, Z39.50, SRU, SOA, mediator/ wrapper архитектура
Чува се: **ЧУ**	Библиотека Департмана за математику и информатику ПМФ-а у Новом Саду

Важна напомена:
ВН

Нема

Извод:
ИЗ

Извршено је моделирање и имплементација система који омогућава претраживање и преузимање библиотечких записа по дефинисаним стандардима. Систем је базиран на сервис-оријенисаној архитектури и mediator/wrapper шаблону. Систем је имплементиран у програмском језику *Java*, а модел је приказан у *UML* 2.0 нотацији. У оквиру система развијени су сервиси који представљају серверске стране за протокол Z39.50 и SRU и развијена је посебна софтверска компонента која омогућава интеграцију тих сервиса са постојећим библиотечким системом. Верификација овог система извршена је интеграцијом у софтверски систем БИСИС верзије 4.

Такође, показано је да се упит формиран помоћу Z39.50 упитног језика може трансформисати у упит који је дефинисан SRU упитним језиком. Дата је и трансформација SRU упитног језика у *Lucene* упитни језик. Дат је предлог проширења SRU стандарда у циљу да се овај стандард користи и за комуникацију између клијента и сервера када је потребно снимање података у удаљену базу података.

Датум прихватања теме од НН већа:
ДП

15. 05.2009.

Датум одбране:
ДО

Чланови комисије:

КО

Председник:	др Милош Рацковић, редовни професор, ПМФ, Нови Сад
члан:	др Зоран Марјановић, редовни професор, ФОН, Београд
члан:	др Зора Коњовић, редовни професор, ФТН, Нови Сад
члан:	др Бранко Милосављевић, ванредни професор, ФТН, Нови Сад
члан:	др Душан Сурла, професор емеритус, ПМФ, Нови Сад

UNIVERSITY OF NOVI SAD
FACULTY OF SCIENCES

KEY WORDS DOCUMENTATION

Accession number: **ANO**	
Identification number: **INO**	
Document type: **DT**	Monograph publication
Type of record: **TR**	Textual printed material
Content code: **CC**	Doctoral dissertation
Author: **AU**	Danijela Boberić
Mentor/comentor: **MN**	Dušan Surla, Ph. D., professor emeritus, Faculty of Sciences, Novi Sad
Title: **TI**	System for retrieval of bibliographic records
Language of text: **LT**	Serbian (Cyrilic)

Language of abstract: **LA**	English
Country of publication: **CP**	Serbia
Locality of publication: **LP**	Vojvodina
Publication year: **PY**	2010
Publisher: **PU**	Author's reprint
Publication place: **PP**	Faculty of Sciences, Trg Dositeja Obradovića 4, Novi Sad
Physical description: **PD**	(7/180/105/8/47/0/0) (chapters/pages/literature/tables/ pictures/graphs/appendix)
Scientific field: **SF**	Informatics
Scientific discipline: **SD**	Information Technology and Systems
Subject/ Key words: **SKW** **UC**	Information retrieval, Z39.50, SRU, SOA, mediator/ wrapper architecture
Holding data: **HD**	Library of Department of Mathematics and Informatics, Trg Dositeja Obradovića 4
Note: **N**	None

Abstract:
AB

Modeling and implementation of software system for retrieval of bibliographic records using defined standard has been done. System is based on service – oriented architecture as well as on mediator/wrapper architecture. System implementation is realized in programming language Java and modelling of system is performed using UML 2.0. Also, services presenting server side of protocols Z39.50 and SRU have been developed. In addition, software component based on mediator approach used for connecting services for retrieval with legacy system is developed. Verification of described system is done by integration of that system into library system BISIS, version 4.

Moreover, it is proved that transformations of Z39.50 query into SRU query are possible, and it has been made a suggestion how to transform SRU query into Lucene query. Also, it has been made suggestion how to extend existing SRU standard in order to use that extension when it is necessary to update bibliographic records on remote databases via Internet.

Accepted by the Scientific Board:
ASB

May 15, 2009.

Defended on:
DE

Thesis defend board:
DB

President:	Miloš Racković, Ph. D., full. prof., Faculty of Sciences, Novi Sad
Member	Zoran Marjanović, Ph. D., full. prof., Faculty of Organizational Sciences, Novi Sad
Member:	Zora Konjović, Ph. D., full. prof., Faculty of Technical Sciences, Novi Sad
Member:	Branko Milosavljević, Ph. D., associate professor, Faculty of Technical Sciences, Novi Sad
Member:	Dušan Surla, Ph. D., professor emeritus, Faculty of Sciences, Novi Sad