Универзитет у Новом Саду

Природно – математички факултет

Департман за математику и информатику

Данијела Тешендић

Софтверски систем за циркулацију библиотечке грађе у оквиру библиотечке мреже

- докторска дисертација -

Нови Сад, 2010.

Предговор

Предмет истраживања ове докторске дисертације припада области информационих система. У дисертацији је описан развој једног дела библиотечког информационог система који се бави циркулацијом библиотечке грађе. Истраживање се бави комуникацијом између библиотека унутар конзорцијума с циљем да се омогући праћене коришћена библиотечких фондова на нивоу конзорцијума. За моделирање софтверске архитектуре система за циркулацију коришћен је објектно-оријентисан приступ и језик *UML* 2.0. Имплементација система урађена је у програмском језику Java.

Дисертација садржи следећа поглавља:

1. Увод
2. *NISO Circulation Interchange Protocol*
3. Циркулација у систему БИСИС верзије 4
4. Подсистем за клијент/сервер комуникацију
5. Циркулација у конзорцијуму библиотечке мреже БИСИС
6. Циркулација у конзорцијуму преко *NCIP* протокола
7. Закључак

У **првом** поглављу дата су уводна разматрања која обухватају преглед постојећих резултата истраживања из предметне области, мотивацију, предмет и циљеве дисертације. Поред тога наведени су и описани библиотечки стандарди који су коришћени у истраживању. Описана је изабрана методологија за развој софтвера, као и софтверске архитектуре коришћене за моделирање архитектуре система.

У **другом** поглављу дат је опис *NCIP* протокола (*Z39.83*) који служи за размену података о корисницима. Протоколом су дефинисани

сервиси и поруке које се размењују између апликација да би се остварила функције потребне за задуживање и раздуживање корисника. Протокол је у дисертацији коришћен при реализацији функционалности циркулације унутар конзорцијума библиотека.

У **трећем** поглављу приказани су модел и имплементација система за циркулацију који подржава рад са корисницима у оквиру локалног фонда библиотеке. Систем подржава све стандардне активности за рад са корисницима.

У **четвртом** поглављу описано је моделирање и имплементација подсистема за клијент/сервер комуникацију и његова интеграција у систем за циркулацију. Овом интеграцијом омогућена је комуникација са базом података у различитим мрежним окружењима. Подсистем има патерн оријентисану софтверску архитектуру која је заснована на комбинацији неколико дизајн патерна.

У **петом** поглављу је дато проширење функционалности и модела података постојећег система за циркулацију тако да се омогући циркулација у конзорцијуму библиотека. За размену података са другим библиотекама искоришћен је подсистем за клијент/сервер комуникацију. Ова верзија система ограничена је само на библиотеке које користе БИСИС систем. Поред тога дат је и преглед могућих начина заштите података при размени.

У **шестом** поглављу описан је начин имплементације *NCIP* протокола у оквиру система за циркулацију. Имплементацијом овог протокола омогућена је циркулација у конзорцијуму између библиотека које користе различите библиотечке системе.

У **седмом** поглављу су дата закључна разматрања, анализа резултата и доприноса дисертације, као и могући правци даљег истраживања.

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

Захваљујем се својој породици на неисцрпној подршци.

Нови Сад, 2010. Данијела Тешендић

Садржај

Поглавље 1

Увод

Предмет истраживања ове докторске дисертације припада области информационих система. У дисертацији је описан развој једног дела библиотечког информационог система који се бави циркулацијом библиотечке грађе.

Библиотечки информациони системи покривају различите аспекте библиотечког пословања. Неки од њих су каталогизација, набавка, преузимање записа од других библиотека, циркулација, међубиблиотечка позајмица, корисничко претраживање и др. Преглед постојећих библиотечких система може се наћи на Интернет презентацији Конгресне библиотеке САД [MARCSystems]. На овој презентацији омогућено је произвођачима библиотечког софтвера да оставе информације о својим производима.

Ово истраживање се бави комуникацијом између библиотечких система у сфери библиотечког пословања које се односи на рад са корисницима. Доминантни библиотечки процеси у тој сфери су циркулација, локална и у конзорцијуму, и међубиблиотечка позајмица. У овом поглављу су описане карактеристике и значај процеса циркулације, као и процеса циркулације унутар библиотечког конзорцијума на чему је и акценат ове дисертације. Дат је преглед постојећег стања у области међубиблиотечке позајмице. Такође, дат је и преглед постојећих софтверских решења из ових области и преглед библиотечких стандарда који се користе.

Софтверска архитектура система за циркулацију заснована је на патернима па су овде наведене и неке класификације патерна и примери примене патерна у дизајну софтверске архитектуре. За развој

система коришћена је методологија обједињени процес (*unified process*). Систем је развијан у више итерација и дат је преглед тих итерација.

Систем за циркулацију описан у овој дисертацији развијен је у оквиру библиотечког система БИСИС, те је стога у овом поглављу дат и кратак преглед развоја система БИСИС. Пре наведених прегледа специфицирани су циљеви и предмет истраживања као полазна тачка у истраживању.

1.1 Предмет и циљеви истраживања

Резултати истраживања ове докторске дисертације реализовани су у оквиру библиотечког система БИСИС. Истраживање је рађено у делу библиотечког пословања које се бави услуживањем корисника, односно коришћењем библиотечког фонда. Тренутно актуелна четврта верзија система БИСИС задовољава све потребе електронског пословања библиотеке које се односе на евидентирање и услуживање корисника искључиво у оквиру локалног фонда библиотеке. Рад са корисницима је имплементиран у оквиру система за циркулацију који представља једну од компоненти БИСИС система.

У сценарију какав је конзорцијум библиотека корисници могу да користе фондове свих библиотека у конзорцијуму, а не само фонд библиотеке којој припадају, што је био мотив за ово истраживање. Реализација овог процеса захтева проналажење начина за праћење коришћења фондова на нивоу конзорцијума, што представља предмет истраживања ове дисертације.

Да би овај процес функционисао потребно је обезбедити размену података о корисницима између библиотека. Библиотека којој корисник припада представља његову матичну библиотеку. Када корисник дође у неку библиотеку која није његова матична потребно је тој библиотеци омогућити приступ подацима о том кориснику који се налазе у његовој матичној библиотеци. Тај приступ је потребан да би се, на пример, проверило да ли тај корисник има право да задужује књиге. Такође, када корисник задужи књигу у некој библиотеци та информација би требала да буде доступна и другим библиотекама.

Да би се ово реализовало потребно је проширити постојећу верзију система за циркулацију тако да она омогућава размену података о

корисницима унутар конзорцијума. Проширење има два аспекта. Први се односи на проширење система у улози клијента, односно, систем је потребно проширити тако да има могућност комуникације са другим системима у неком мрежном окружењу, најчешће Интернету. Том комуникацијом омогућава се приступ подацима других библиотека, односно приступ подацима о корисницима који долазе из других библиотека. Други аспект проширења односи се на серверску улогу система, односно, потребно је обезбедити *web* сервисе другим библиотекама преко којих би имале приступ подацима о корисницима који одлазе код њих. Такође, размену података о корисницима је потребно реализовати тако да је независна од конкретних библиотечких система који се користе у конзорцијуму.

Циљ овог истраживања је проширење система за циркулацију тако да се омогући циркулација библиотечке грађе у конзорцијуму библиотека. Реализација овог циља састоји су у следећем:

- Проширење постојећег модела података система за циркулацију тако да он обухвати све потребне податке за функционисање циркулације у конзорцијуму.
- Проширења функционалности система за циркулацију којима би се реализовала размена података са другим библиотекама у конзорцијуму.
- Реализација дела система кроз коју би клијентска апликација комуницирала са серверима, било са сервером своје библиотеке или са серверима других библиотека у конзорцијуму. Овај део система би требао да има следеће особине:
    - Омогућава транспарентну комуникацију клијентске апликације са сервером своје библиотеке без обзира да ли се сервер налази у локалном мрежном окружењу или на Интернету, односно, комуникација би за клијента требала да буде транспарентна у односу на транспортни протокол које се користи за комуникацију.
    - Омогућава транспарентну комуникацију са серверима других библиотека ради функционисања циркулације у конзорцијуму. Библиотеке у конзорцијуму могу да користе различите библиотечке системе тако да комуникација треба да буде транспарентна и у односу на

протокол који се користи за размену података и у односу на транспортни протокол за комуникацију.
    - Независтан је од функционалности система за циркулацију тако да може да се користи за потребе осталих компоненти система БИСИС, као и да се интегрише у друге системе који имају потребу за унифицирањем различитих начина комуникације.
- Реализација *web* сервиса преко ког би друге библиотеке имале приступ подацима потребним за функционисање циркулације у конзорцијуму.

1.2 Циркулација библиотечке грађе

Један од сегмената електронског пословања библиотека је циркулација библиотечке грађе. Циркулација обухвата све активности везане за рад са корисницима као што су евиденција корисника, задуживање и раздуживање корисника, генерисање различитих врста статистика и др. Циркулација као један од основних процеса библиотечког пословања има велики утицај и значај на остале процесе. Један број научних радова из ове области описује различите аспекте циркулације и неке смернице за даљи развој електронске подршке процесу циркулације у циљу развоја библиотечког пословања. У раду (Babafemi, 2002) описана је улога библиотекара у сарадњи са корисницима. Посебно је истакнут значај те сарадње као повратна информација за набавку библиотечке грађе према исказаним потребама корисника. Искуства представљена у радовима (Boone, 2002) и (Joint, 2008) показују да електронски сервиси које библиотеке пружају корисницима омогућавају ефикасније коришћење како папирног тако и електронског фонда библиотеке. У раду (Falk, 2005) описано је неколико таквих сервиса и на који начин ти сервиси унапређују библиотечко пословање. У раду (Bennett, 2007) приказани су резултати истраживања по којима унапређења направљена на корисничком претраживању утичу на повећање коришћења библиотечког фонда и међубиблиотечке позајмице. Повећање ефикасности рада са корисницима коришћењем *RFID* (*Radio Frequency Identification*) технологије описано је у раду (Kern, 2004). У радовима (Adams and Noel, 2008) и (Cheng et al., 2002) је приказана је улога и

значај статистичких извештаја о коришћењу библиотечке грађе за унапређење и проширење фонда библиотеке. Резултати приказани у овој групи радова дају идеје у ком смеру би требао да иде развој електронских система за циркулацију. Интернет оријентисани сервиси према корисницима, као и Интернет оријентисана комуникација и размена података између библиотека представљају главни правац у ком се иде.

Чланови једне библиотеке могу имати права на позајмицу књига и из других библиотека у оквиру конзорцијума библиотека. Конзорцијум представља удружење библиотека на неком подручју које међусобно сарађују што доводи до побољшања услуга које библиотеке пружају својим корисницима. Једна од активности на нивоу конзорцијума је циркулација у конзорцијуму. Циркулација у конзорцијуму је библиотечки процес у ком је корисницима једне библиотеке омогућено да позајмљују књиге из других библиотека унутар конзорцијума. Библиотеке чланице конзорцијума успостављају правила за коришћење библиотечких фондова. Добри примери за конзорцијуме су градске библиотеке које у свом саставу имају више библиотека или факултетске библиотеке унутар једног универзитета. Да би се омогућила ова активност потребно је обезбедити софтверску подршку за циркулацију библиотечке грађе на нивоу конзорцијума библиотека, односно потребно је обезбедити размену података о корисницима на нивоу конзорцијума. Комуникација и размена података дефинисана је протоколом за размену података о корисницима који је описан у даљем тексту.

У оквиру овог истраживања прегледана су и постојећа софтверска решења из ове области. Постоји велики број библиотечких софтверских система који у свом оквиру имају и систем за циркулацију. На Интернет презентацији Конгресне библиотеке за *MARC* стандарде [MARCStandards] постоји посебан одељак на ком се налази преглед библиотечких програмских пакета [MARCSystems]. Неки од тих пакета између осталих функционалности библиотечких информационих система подржавају и циркулацију. Већина софтверских производа је комерцијална и није јавно доступна за преузимање. Могуће је једино преузети корисничка упутства и на основу њих извршити анализу функционалности софтвера. Неки од тих софтвера су *ALEPH 500* [Aleph500], *Voyager* [Voyager], *Atriuum*

[Atriuum], *Concourse* [Concourse], *CyberTools for Libraries* [CyberTools], *EOS.Web* [EOS], *Polaris Library Systems* [Polaris], *SirsiDynix Symphony* [Symphony], *SirsiDynix Unicorn* [Unicorn] и други. Постоје и бесплатни софтверски производи као што су *Evergreen Integrated Library System* [Evergreen] и *Koha* [Koha] чију функционалност је могуће у потпуности анализирати преузимањем инсталационих пакета.

Ови системи омогућавају стандардне активности везане за рад са корисницима које обухватају задуживање и раздуживање корисника, продуживање чланарине корисника, штампање реверса и признаница, претраживање корисника и публикација, као и генерисање статистичких извештаја. Неки од њих укључују и подршку за *UNICODE*, бар код и *RFID* технологију, штампање и слање опомена корисницима за прекорачење рока враћања публикација и резервацију публикација. Клијентске апликације система су имплементиране или као *Windows* апликације подржане само од стране *Windows* оперативних система или као *web* апликације. Од наведених софтверских решења само нека имају подршку за рад у конзорцијуму библиотека и размену података унутар конзорцијума, а то су *ALEPH 500*, *Voyager*, *Polaris* и *SirsiDynix* решења.

1.3 Међубиблиотечка позајмица

Међубиблиотечком позајмицом корисницима се пружа услуга коришћења фондова других библиотека, али уз посредство своје матичне библиотеке. У овом случају библиотеке између себе позајмљују публикације које даље дају корисницима на коришћене на исти начин као и публикације локалног фонда. Да би се омогућио овај процес потребно је обезбедити размену релевантних података између система. Размена података за процес међубиблиотечке позајмице дефинисана је протоколом који је описан у следећем одељку.

Најраспрострањенији и најпопуларнији сервис за међубиблиотечку позајмицу је *OCLC* сервис [OCLC]. *OCLC* (*Online Computer Library Center*) организација је настала 1967. године на универзитету у Охају с циљем да се омогући лакши приступ информацијама у библиотекама. Данас *OCLC* сервисе користи више од 69000 библиотека из 112 земаља. Под окриљем ове организације је настао и *WorldCat* - *OCLC* централни каталог [WorldCat]. У овом каталогу се налази преко 125 милиона

библиотечких записа библиотека које користе овај сервис. Овај каталог је претражив и преко Интернета, а преко *OCLC* сервиса може да се врши позајмица књига које су у овом каталогу. *OCLC* сервис има улогу посредника у трансакцијама међубиблиотечке позајмице. Сервисе међубиблиотечке позајмице могу да користе библиотеке чланице из било ког софтвера којим је имплементиран стандард за међубиблиотечку позајмицу. *OCLC* нуди и свој софтверски систем за позајмицу *VDX* [VDX] који омогућава праћење свих фаза процеса међубиблиотечке позајмице.

Искуства са овим сервисом као и са другим системима који омогућавају међубиблиотечку позајмицу су описана у већем броју радова.

У раду (Fleck, 2000) су описане могућности софтвера за међубиблиотечку позајмицу који се користи на Мичигенском универзитету. Развијен је од стране фирме *Innovative Interfaces* и подржава *ISO ILL* стандард за међубиблиотечку позајмицу. Поред слања стандардних захтева из библиотеке софтвер омогућава и студентима да са свог налога преко Интернета пошаљу захтев за књигом. Такав захтев се прослеђује матичној библиотеци на обраду, одакле се шаље даље у другу библиотеку директно или преко *OCLC* система. Поред управљања самим трансакцијама систем омогућава и генерисање различитих статистичких извештаја.

Пракса у међубиблиотечкој позајмици у библиотекама на универзитетима и колеџима у Израелу је приказана у раду (Porat and Shoham, 2004). Универзитетске библиотеке користе систем *ALEPH 500* преко кога раде и међународну позајмицу. Библиотеке на колеџима највећим делом позајмицу врше од универзитетских библиотека. У раду је описан пројекат повезивања библиотека на колеџима у конзорцијум унутар кога би се вршила позајмица.

У раду (Gatenby, 2003) је приказана улога централних каталога у процесу међубиблиотечке позајмице. *EUCAT* представља индекс европских централних каталога преко ког ови каталози постају доступни сервисима за међубиблиотечку позајмицу. *EUCAT* је претражив и преко *OCLC* система за међубиблиотечку позајмицу.

Развој и имплементација сервиса који обједињује претраживање и позајмљивање књига из националног, регионалних и провинцијалних

каталога Норвешке описани су у раду (Braun et al., 2006). За реализацију и праћење међубиблиотечких трансакција у оквиру овог сервиса користи се *VDX* систем.

Ситуација по питању међубиблиотечке позајмице у Великој Британији је представљена у раду (Froud, 2006). Представљена су два система који су се паралелно развијали. Тренутно постоји сарадња и размена ресурса између њих, а и иницијатива да се направи сервис којим би се омогућио приступ базама оба система са једне локације.

У раду (Missingham, 2007) је дат преглед стања на пољу међубиблиотечке позајмице у библиотекама Аустралије. Дат је историјски осврт на процес аутоматизације библиотека и формирање централног каталога. За потребе међубиблиотечке позајмице у Националној библиотеци је изабран *VDX* систем преко кога је доступан централни каталог и може се користити и од стране других система имплементираних по *ISO* стандарду за међубиблиотечку позајмицу.

У раду (Guerrero and Matte, 2003) су представљена два система за међубиблиотечку позајмицу који се користе у Мексику. Преко ових система позајмица се врши како у Мексику тако и од универзитетских библиотека у Америци.

Ипак, поред великог броја софтверских система и сервиса за међубиблиотечку позајмицу постоје библиотеке које захтеве за међубиблиотечку позајмицу шаљу у папирном облику или путем *e-mail* порука. С циљем да се стандардизује формат *e-mail* захтева и обезбеде сви потребни елементи у захтеву *IFLA* је објавила смернице за писање захтева. Ове смернице, као и резултати анкете о слању *e-mail* захтева су дати у раду (Gould, 2000).

1.4 Библиотечки стандарди

У овом одељку су описани неки од библиотечких стандарда који се користе у развоју библиотечких информационих система.

Формати за опис библиографске грађе који се користе у истраживању за формирање библиотечких записа су *UNIMARC* [Unimarc] и *MARC21* [Marc21]. Оба ова формата поштују међународни стандард *ISO 2709*

[ISO2709] којим је дефинисана структура података библиографског описа. *UNIMARC* формат је развијен од стране организације *IFLA* (*International Federation of Library Associations and Institutions*) и публикован је 1994. године. Настао је с намером да се ускладе различити *MARC* формати и направи један универзалан. *MARC21* формат је развијен од стане Конгресне библиотеке САД и настао је на основу формата *US MARC*. Последњих година је присутна тенденција преласка на *MARC21* услед потребе за разменом библиографских података у оквиру светске библиотечке мреже.

Протоколи који се користе за размену библиографских записа су *Z39.50* [Z3950] и *ebXML* [ebXML]. *Z39.50* је протокол за проналажење и преузимање података у дистрибуираним рачунарским системима. Осмишљен је као општи протокол 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. *Z39.50* је стандард организације *NISO* (*National Information Standards Organization*). Агенција која се бави формалним дефинисањем, унапређивањем и одржавањем стандарда *Z39.50* је *International Standard Z39.50 Maintainance Agency* у оквиру Конгресне библиотеке. *ebXML* је протокол развијен за потребе електронског пословања. То је први протокол у ком је *XML* прихваћен као формат порука које се размењују у информационим системима. Протокол пружа механизам за описивање и извршавање различитих пословних процеса, као и проналажење информација о учесницима у тим процесима. Стандардизован је од стране агенције *OASIS* (*Organization for the Advancement of Structured Information Standards*).

У дисертацији је коришћен протокол *Z39.83* [NCIP] за размену података о корисницима. Овај протокол је стандардизован од стране организације *NISO*. Протоколом су дефинисани сервиси и поруке које се размењују између апликација да би се остварили функције потребне за задуживање и раздуживање корисника и контролисање приступа електронским ресурсима. Области примене овог стандарда су позајмљивање унутар конзорцијума библиотека, размена података о корисницима у процесу међубиблиотечке позајмице и сервиси за само-задуживање корисника без помоћи библиотечког особља. Протокол је описан са два документа. У документу [Z3983-1] је дата сама дефиниција протокола где су описана синтаксна и семантичка

правила порука које се размењују између апликација које комуницирају по овом протоколу. У документу [Z3983-2] је дат имплементациони профил где су специфицирани детаљи практичне имплементације протокола. Структура порука је описана *XML* шемом, за кодни распоред је изабран *Unicode*, а за транспорт порука су предвиђени протоколи *HTTP*, *HTTPS* и *TCP/IP*. Овај протокол је релативно нов и још увек није у тако широкој употреби. Прва верзија је настала 2001. године, а на основу искуства у имплементацији ревидиран је и у новембру 2008. године објављена је тренутно актуелна верзија протокола. Систем *VDX* је имплементирао овај стандард за потребе размене података о корисницима у оквиру међубиблиотечке позајмице. Предности које је овом имплементацијом увидела универзитетска библиотека у Санта Барбари су дате у раду (Johnson, 2009).

Протокол за међубиблиотечку позајмицу *ISO ILL* дефинише правила која се односе на позајмицу публикација између библиотека као што су књиге и часописи или размену неповратних библиотечких јединица као што су фотокопије чланака из часописа. Циљ овог протокола је да се обезбеди потпуна контрола међубиблиотечке трансакције од момента слања захтева, преко достављања публикације, па до враћања публикације, да се омогући позајмица између различитих система и да се минимизују трошкови трансакције. Протокол је стандардизован од стране организације *ISO* (*International Organization for Standardization*) и дефинисан је са три документа. У документу [ISO10160] су дефинисани сервиси међубиблиотечке позајмице, типови трансакција, учесници у трансакцијама, њихове улога и могућа стања. У документу [ISO10161-1] је дата спецификација протокола за међубиблиотечку позајмицу која подржава сервисе дефинисане у претходном документу. Спецификација је дата у *ASN.l* нотацији. Документ [ISO10161-2] представља *PICS* (*Protocol implementation conformance statement*) платформу која служи за опис функционалности конкретне имплементације протокола. Овај протокол је у употреби већ дужи низ година и имплементиран је од стране различитих библиотечких система. У раду (Shuh, 1998) је дат приказ развоја овог протокола. Протокол је настао на основу праксе у канадским и америчким библиотекама. У раду су описане прве имплеметације протокола у Канади и Европи, као и даљи развој тих система који је довео до

унапређења самог протокола и објављивања његове друге, тренутно актуелне, верзије 1997. године.

1.5 Патерн оријентисане софтверске архитектуре

С обзиром на то да су у софтверској архитектури система за циркулацију коришћени патерни у овом одељку су дати примери примене патерна у дизајну софтверске архитектуре, као и неке класификације патерна.

Патерни представљају постојећа добра решења за честе дизајнерске проблеме који настају у одређеном контексту. Сваки патерн се односи на један специфичан проблем у дизајну или имплементацији софтверских система. Идеја патерна је да се дефинишу и категоризују уобичајена дизајнерска решења која су се показала као добра и корисна. Настали су на основу искуства у дизајну софтвера с циљем да се омогући поновна примена стеченог знања и искуства. Патерни не представљају софтверска решења која се користе у развоју софтвера, него се њима дефинише само основна структура решења неког проблема коју је потребно имплементирати у односу на специфичне потребе одређеног проблема. Применом патерна у дизајну софтверске архитектуре добијају се елегантнија софтверска решења лакша за одржавање и даљу надоградњу са могућношћу интеграције у различите софтверске системе. Софтверске архитектуре засноване на патернима називају се патерн оријентисане софтверске архитектуре.

Опис патерна се састоји од три дела: контекста, проблема и решења. Контекстом се описује ситуација у којој настаје проблем. Опис проблема садржи детаље проблема који настаје у датом контексту. Решењем су описани статички и динамички аспекти решења датог проблема. Статички аспект решења обухвата компоненте, класе или објекте, њихове улоге и везе између њих. Динамички аспект решења описује начин комуникације и сарадње између компоненти.

Појам патерна као дизајнерског решења увео је архитекта Christopher Alexander. Он је у својим радовима и књигама описао више од 250 патерна који представљају решења на различитим нивоима, од структуре градова и региона, до декорације вртова и индивидуалних соба. Такође, дефинисао је и појам патерн језика (Alexander, 1979).

Елементи патерн језика су патерни. Патерн језиком је описана структура и међусобне везе патерна, као и зависност патерна од других патерна. Он је дефинисао и контекст/проблем/решење струкутру за опис патерна.

Идеју патерна у развоју софтвера увели су Ward Cunningham и Kent Beck. Први патерни које су направили односили су се на дизајн корисничког интерфејса. У даљем раду Ward се фокусирао на развој пословних система и један од његових резултата је патерн језик *CHECKS* описан у раду (Cunningham, 1994). Kent се бавио имплементацијом патерна у језику *Smalltalk* што је објављено у књизи (Beck, 1997). Употреба патерна у развоју софтвера је шире прихваћена објављивањем прве колекције патерна у књизи (Gamma et al., 1994). Та колекција садржи 23 патерна и њени аутори су је Erich Gamma, Richard Helm, Ralph Johnson и John Vlissides. Решења описана овим патернима су независна од програмског језика.

Да би се патерни лакше користили потребно их је категорисати на основу неког критеријума и описати њихову међусобну зависност. Због ове потребе је и настао појам патерн језика. У књизи (Buschmann, 1996) уведен је појам патерн систем који се користи уместо појма патерн језик. Дефиниција патерн језика подразумева да су покривени сви аспекти домена за који се дефинише језик. Појам патерн система је уведен јер постоје аспекти конструисања софтверске архитектуре који нису покривени патернима. Патерн систем за софтверску архитектуру је дефинисан као колекција патерна за софтверску архитектуру која укључује препоруке за њихову имплементацију, комбинацију и практичну примену у развоју софтвера. У свом систему патерна аутори књиге (Buschmann, 1996) патерне су поделили у три категорије: архитектонски патерни, дизајн патерни и идиоми. Архитектонски патерни су патерни на највишем нивоу и њима се описује структура система, дефинишу подсистеми и њихове одговорности, као и везе међу њима. Дизајн патерни служе за дизајн појединачних подсистема и компоненти. Примена дизајн патерна на делове система не утиче на општу архитектуру система. Дизајн патерни су независни од програмског језика. Идиоми су патерни на најнижем нивоу и везани су за одређени програмски језик. Идиомима је описано како имплементирати неко дизајнерско решење у датом програмском језику. У књизи је описано укупно 17 патерна.

Колекција патерна описана у књизи (Gamma et al., 1994) садржи само дизајн патерне. По систему патерна представљеном у тој књизи патерни су подељени у три категорије на основу њихове намене: *creational, structural* и *behavioral*. *Creational* патерни се баве процесом креирања објеката. *Structural* патернима се описује композиција класа и објеката, а *behavioral* патернима интеракција објеката и њихове одговорности. У књизи (Cooper, 1998) је приказан другачији приступ овој колекцији. За сваки патерн је дат комплетан пример имплементације писан у *Java* програмском језику, чиме се олакшава разумевање патерна и њихове могуће примене.

Поред две наведене, постоје многе колекције патерна, најчешће везане за специфичне домене. Неке од њих су, на пример, *J2EE* дизајн патерни. *J2EE* је *Java* платформа за развој дистрибуираних апликација. Колекција *J2EE* дизајн патерна описана је у књизи (Alur et al., 2001) и садржи 15 патерна. У раду (Hammouda and Koskimies, 2002) приказан је патерн оријентисан фрејмворк (*framework*) за *J2EE* апликације чија архитектура је заснована на *J2EE* патернима. За развој фрејмворка је коришћен алат *FRED* који се користи за развој патерн оријентисаног софтвера.

У великом броју софтверских решења која укључују комуникацију између објеката на различитим локацијама за архитектуру дела за комуникацију изабран је *command* патерн. У случајевима када је потребно из скупа више класа зависно од параметара креирати инстанцу одговарајуће класе највише се користе *abstract factory* и *factory method* патерни. У софтверским решењима где је потребно повезати два различита интерфејса, односно прилагодити један интерфејс другом, користи се често *adapter* патерн. Ова четири патерна припадају колекцији дизајн патерна описаној у књизи (Gamma et al., 1994). У даљем тексту је дат преглед неколико радова где су коришћени ови патерни. Због наведених особина ова четири патерна су коришћена и у архитектури система за циркулацију. Опис *command, abstract factory* и *factory method* патерна, као и њихова примена у систему за циркулацију описани су у четвртом поглављу, док је опис и примена *adapter* патерна описана у шестом поглављу.

У раду (Sawant et al., 2000) је описана архитектура фрејмворка *TIDE* који служи за развој окружења за претраживање и визуализацију

великих скупова података. У овом фрејмворку *command* патерн је искоришћен за комуникацију клијентске и серверске стране, односно за пребацивање корисничког захтева са клијента на сервер одакле се он даље обрађује и резултат обраде се враћа клијенту. Клијентски захтеви укључују обраду великог броја података и рачунање, а оваквим начином ти захтеви могу да се обрађују на серверима са подацима или на више сервера, док клијент врши само приказ већ обрађених података.

Слична идеја примене *command* патерна је дата у раду (Agrawal et al., 2002). *CADDAC* је систем са *web* оријентисаним колаборативним дизајном. Идеја овог система је да се измене од стране једног клијента на моделу кога сви користе врше на серверу и да се те измене пропагирају другим клијентима. *Command* патерн је искоришћен за пренос података и инструкција са клијента на сервер где се те инструкције извршавају и мењају модел, а затим се те промене на моделу шаљу другим клијентима.

Архитектура *command* патерна је погодна и за реализацију *undo*/*redo* операције у апликацијама. Пример једне реализације је дат у раду (O'Brien and Shapiro, 2004). У овом раду је приказан алат за *undo*/*redo* операцију који не поништава акције хронолошки него на основу семантичких веза међу акцијама.

Занимљиве две примене *command* патерна су приказане у радовима (Huang and Chung, 2003) и (Pereira et al., 2006).

У раду (Huang and Chung, 2003) је описана архитектура фрејмворка који подржава *web* сервис оријентисан развој пословних интегрисаних решења. Овај фрејмоворк омогућава развој нових апликација композицијом предефинисаних *web* сервиса. *Command* патерн је у овом решењу искоришћен за део фрејмворка који повезује различите *web* сервисе и води рачуна о њиховој интеракцији.

Arcademis је *Јава* оријентисан фрејмворк за развој и имплементацију средњег слоја система и комуникације у средњем слоју и описан је у раду (Pereira at al., 2006). Подржава развој модуларних и прилагодљивих архитектура средњег слоја система које могу да користе различите протоколе и технологије. У архитектуру фрејмворка је укључено више дизајн патерна. *Command* патерн је искоришћен у делу за размену порука између дистрибуираних

објеката, а предност тога је што је омогућено додавање нових порука без потребе за изменом других делова система. Такође за моделирање фрејмворка *Arcademis* коришћени су и *factory* патерни да би се постигла модуларност система. Поред осталих делова фрејмворка, *factory* патерни су искоришћени и за моделирање комуникације између компоненти чиме је омогућена употреба различитих транспортних протокола на средњем слоју система.

У раду (Zhang et al., 2006) описан је систем који комбинујући анализу услова, поузданости и ограничења предвиђа најпогоднији период за одржавање машина с циљем да се смање трошкови одржавања и утицај на призводни процес. Систем има трослојну патерн оријентисану архитектуру. У архитектури слоја задуженог за рад са базом података коришћени су *factory* и *adapter* патерни да би се омогућио рад са базама података различитих произвођача. Пословна логика система је дизајнирана помоћу *stategy, visitor, decorator* и *facade* патерна чиме се постигла флексибилност и проширивост система. Презентацијски слој задужен за интеракцију са корисником дизајниран је помоћу *stategy, decorator* и *factory* патерна, док је контролер између презентацијског слоја и пословне логике дизајниран *command* патерном. Оваквим дизајном постигла се флексибилност, проширивост и лако одржавање система.

Adapter патерн је чест избор за дизајн софтвера чији је задатак да старе софтвере прилагоде новим интерфејсима или новим архитектурама система. У раду (Hunold et al., 2008) је приказан алат за конверзију апликација са монолитном архитектуром у неколико независних компоненти које се даље могу интегрисати у дистрибуиране апликације или заменити новим компонентама.

У раду (Vergara et al., 2007) *adapter* патерн је коришћен за дизајнирање начина интеграције и повезивања *web* апликација са екстерним апликацијама у *model-driven* сценарију. У раду су наведени проблеми који се јављају при томе и дати су предлози за решења тих проблема. *Adapter* патерн је коришћен у решењима проблема када две апликације користе различита имена или различите структуре података за исте концепте, када користе различите формате података и када користе различите протоколе за комуникацију.

1.6 Библиотечки систем БИСИС

Библиотечки систем БИСИС развија се од 1993. године на универзитету у Новом Саду и преглед тог развоја дат је у монографији (Konjović i Surla, 2004). Тренутно је актуелна четврта верзија система. У овом одељку је дат кратак преглед развоја система.

Прва верзија система је настала као резултат пројекта Систем за формирање и претраживање информација. Резултати истраживања на овом пројекту приказани су у монографији (Лазаревић и др., 1996). Ова верзија реализована је у програмском језику *C*, а као систем за управљање базом података коришћена је *db_VISTA*. У овој верзији дефинисане су базе података библиографске грађе, *UNIMARC* формата, радног окружења библиотекара и циркулације библиотечког фонда. Овако формиране базе података су се уз потребне измене користиле и у наредним верзијама система. За индексирање и претраживање библиографских записа развијен је текст сервер. За обраду библиографске грађе усвојен је *UNIMARC* формат. Функције које подржава ова верзија су: дефинисање стандарда за одржавање библиографске грађе и структуирање записа; дефинисање радног окружења библиотекара; формирање библиографске грађе; библиотекарско извештавање и документовање; корисничко претраживање.

Друга верзија система реализована је у *Oracle* окружењу и програмском језику *Java*. Од *Oracle*-ових производа користи се *Oracle Server v8.0.3* и текст сервер *ConText Cartridge v2.3.6*, која подржава рад са текстом у *Unicode*-у. Систем је предвиђен да ради у Интернет, односно Интранет окружењу. Кориснички интерфејс је имплементиран као *Java* аплет. Овом верзијом су подржане све функције из претходне верзије система.

У трећој верзији развијен је сопствени текст сервер за индексирање и претраживање библиографских записа у *UNIMARC* формату. Главне карактеристике овог сервера су: специјализованост која резултује бољим перформансама, трослојна архитектура, коришћење *Java* платформе и независност од коришћеног релационог система за управљање базом података. Подршка за *Unicode* стандард доследно је спроведена у целом систему. Овом верзијом су такође подржане све функције из претходне верзије система.

Претходно описане верзије БИСИС система задовољавале су потребе факултетских библиотека. Увођењем БИСИС-а верзије 3.1 у градске и специјализоване библиотеке формирали су се нови захтеви проистекли из пословања тих библиотека. На основу анализе захтева и потреба градских и специјализованих библиотека које користе БИСИС, као и трендова у развоју библиотечких система развијена је четврта верзија система БИСИС.

Четврта верзија система базирана је на *XML* технологији. У оквиру ове верзије развијен је *XML* едитор за обраду библиографске грађе по *UNIMARC* и *MARC21* формату, *XML* едитор за претраживање и преузимање библиографских записа по стандарду *Z39.50*, као и систем за циркулацију. Ове три компоненте су резултат истраживања оквиру магистарских теза (Рађеновић, 2006; Димић 2007; Боберић, 2007; Тешендић, 2007). Детаљан опис моделирања, имплементације и коришћења ових компоненти приказан је у монографијама (Рађеновић и др, 2006; Димић и Сурла, 2007; Боберић и Сурла, 2007; Тешендић и Сурла, 2007).

Резултати истраживања у току развоја четврте верзије су објављени у неколико радова. У раду (Dimić and Surla, 2009) приказан је развој *XML* едитора за обраду библиографске грађе по *UNIMARC* и *MARC21* формату. *XML* шема *YUMARC* формата коју користи едитор је дата у раду (Milosavljević and Dimić, 2007). Поред овог едитора развијен је и *web* оријентисан едитор за обраду библиографске грађе по *UNIMARC* формату и приказан је у раду (Belić and Surla, 2008). Конверзија библиографских записа из система БИСИС у формат *MARC21* описана је у раду (Rudić and Surla, 2009). Развој *XML* едитора за претраживање и преузимање библиографских записа по стандарду *Z39.50* је приказан у раду (Boberić and Surla, 2009). Систем за валидацију библиографских записа описан је у раду (Budimir and Surla, 2004). Као софтверска компонента у оквиру система БИСИС развијен је и систем за коришћење библиотечке грађе, а његов развој је приказан у радовима (Tešendić et al., 2009) и (Tešendić, 2007).

1.7 Коришћене методологије и софтверске архитектуре

За моделирање софтверске архитектуре система за циркулацију коришћен је објектно-оријентисани приступ. Моделирање је урађено у

језику *UML* 2.0 и коришћен је *CASE* алат *Sybase PowerDesigner v12.5* [Sybase]. Модел је представљен дијаграмима случајева коришћења, дијаграмима активности, дијаграмима пакета, дијаграмима компоненти, дијаграмима размештаја, дијаграмима класа и дијаграмима секвенци.

Имплементација система је урађена у програмском језику *Java*. У имплементацији су коришћена и готова *open-source* софтверска решења за реализацију неких функција система. Као сервер базе података се користи *MySQL Community Server v5.1* [MySQL]. За *web* апликације коришћен је *web* сервер *Apache Tomcat v6.0* [Tomcat].

Усвојена је трослојна софтверска архитектура система. Састоји се од клијентске апликације, подсистема за клијент/сервер комуникацију и базе података. Клијентска апликација и подсистем за комуникацију имају патерн оријентисану софтверску архитектуру и за дизајн архитектуре је коришћено неколико дизајн патерна.

У развоју система коришћен је методолошки приступ унифицирани процес (*unified process*) (Jacobson, et al., 1999). Основне карактеристике овог приступа у развоју софтвера су:

- итеративни и инкрементални развој (*iterative and incremental develipment*),
- вођен случајевима коришћења (*use case driven*),
- софтверска архитектура заузима централно место (*architecture centric*).

Основна идеја овог приступа је у томе да је развој софтвера подељен у низ мањих пројеката који се називају итерације. Резултат сваке од итерација је један инкремент односно софтверски производ који је тестиран, интегрисан и извршив.

Развој система за циркулацију описаног у овој дисертацији реализован је кроз четири итерације.

Резултат прве итерације развоја је верзија система која подржава само рад са корисницима у оквиру локалног фонда библиотеке. Подржане су све стандардне активности за рад са корисницима. Систем је реализован као независна софтверска компонента која се може интегрисати и у друге библиотечке системе и подржава рад са различитим форматима библиотечких записа. Такође, систем подржава два начина комуникације између клијентске апликације и

базе података тако да је омогућен рад система и у оквиру локалне мреже и на Интернету. Реализација ове комуникације је везана за функционалности система. Ова верзија система описана је у трећем поглављу дисертације.

У другој итерацији развоја реализован је подсистем за клијент/сервер комуникацију и интегрисан је у систем БИСИС. Подсистем омогућава комуникацију са базом података у различитим мрежним окружењима чиме је омогућен рад клијентске апликације са различитих локација. Подсистем је имплементиран тако да је пословна логика система иста без обзира на мрежну конфигурацију. Такође, подсистем није везан за пословну логику система па се може интегрисати и у друге системе. Софтверска архитектура подсистема је патерн оријентисана и заснована је на комбинацији неколико дизајн патерна. Развој и интеграција подсистема описани су у четвртом поглављу.

У трећој итерацији развоја реализована је циркулација у конзорцијуму библиотечке мреже БИСИС, односно циркулација у конзорцијуму у ком све библиотеке користе БИСИС систем. Проширене су функционалности система тако да је омогућен рад са корисницима из других библиотека. За размену података са другим библиотекама искоришћен је подсистем за клијент/сервер комуникацију. Оваквим приступом клијент је независан од начина комуникације са другим библиотекама. Интерфејс за комуникацију који клијент користи је исти и у случају комуникације са својим сервером и у случају комуникације са другим библиотекама. Формат у ком се размењују подаци везан је за имплементацију система те је ова верзија система ограничена на конзорцијум библиотечке мреже БИСИС. Ова верзија система која је резултат треће итерације развоја описана је у петом поглављу дисертације.

У четвртој итерацији развоја реализована је верзија система која подржава размену података о корисницима по *NCIP* протоколу. Овиме је омогућена циркулација у конзорцијуму између библиотека које користе различите библиотечке системе. Протокол је имплементиран у оквиру подсистема за клијент/сервер комуникацију. На овај начин клијент остаје независан од начина комуникације са другим библиотекама, односно клијент размењује податке у истом формату као у случају комуникације са другим БИСИС системом, док

је подсистем за комуникацију задужен за конверзију тог формата у *NCIP* поруке. Такође, у овој верзији имплементиран је и *NCIP* сервис за приступ подацима по *NCIP* протоколу од стране других библиотека. Ова верзија система описана је у шестом поглављу дисертације.

Поглавље 2

NISO Circulation Interchange Protocol

NCIP (*NISO Circulation Interchange Protocol*) или *Z39.83* је протокол за размену података о корисницима стандардизован од стране организације *NISO* (*National Information Standards Organization*). Протоколом су дефинисани сервиси и поруке које се размењују између апликација да би се остварile функције потребне за задуживање и раздуживање корисника и контролисање приступа електронским ресурсима.

Протокол је јаван и може се преузети са званичног сајта *NCIP* протокола [NCIP]. Поред дефиниције протокола на сајту се може наћи пратећа документација за лакше разумевање протокола, новости из области развоја и коришћења протокола, као и информације о имплементацији протокола од стране произвођача софтвера. У овом поглављу је дат преглед основних концепата протокола који је настао на основу званичне дефиниције протокола и препорука за имплементацију. Поред тога је дат и кратак преглед постојећих имплементација протокола од стране произвођача софтвера.

2.1 Развој протокола

Развојем протокола *Z39.50* за проналажење и преузимање библиографских података и *ISO ILL* (*ISO 10160*/*ISO 10161*) протокола за међубиблиотечку позајмицу омогућено је проналажење електронских ресурса, као и слање и праћење захтева за размену библиотечких ресурса. Употребом ових протокола у библиотекама јавила се потреба и за размену података о корисницима који користе те ресурсе. Једини начин за размену ових података међу библиотекама било је коришћење истог софтвера у библиотекама. Размене података између

софтвера различитих произвођача нису биле могуће.

Као одговор на овакву ситуацију *NISO* организација је оформила комитет за развој протокола за размену података о корисницима. Комитет, који се састојао од представника библиотека и представника произвођача библиотечког софтвера, почео је са радом у јануару 1999. године. *NISO* је у јануару 2001. године објавила нацрт стандарда који се састојао од два документа: описа самог протокола и имплементационог профила. Имплементациони профил је дефинисао размену порука у *XML* формату, а структура порука је дефинисана *DTD*-ом. Током 2001. године развијено је и неколико апликационих профила. Коначно, у октобру 2002. године објављена је и прва верзија стандарда која се састојала од описа стандарда и имплементационог профила, а уз њу је објављено неколико апликационих профила, као и прва верзија *DTD* документа и прва верзија *XML* шеме којима је дефинисана структура *XML* порука које се размењују протоколом. Оформљене су и *NCIP Implementers Group* и *NCIP Maintenance Agency* које су задужене за одржавање и даљи развој стандарда.

Након пет година од објављивања прве верзије стандарда почео је рад на ревизији стандарда и формирању друге верзије стандарда. Током петогодишње употребе стандарда уочени су његови недостаци и јавила се потреба за изменама појединих делова. У новембру 2008. године је објављена друга верзија стандарда. Друга верзија има много измена у односу на прву и није компатибилна са првом. У другој верзији структура *XML* порука је дефинисана само *XML* шемом, док се *DTD* више не користи.

NCIP стандард је бесплатан и јаван. Стандард укључује два документа. У документу [Z3983-1] је дата сама дефиниција протокола где су описана синтаксна и семантичка правила порука које се размењују између апликација које комуницирају по овом протоколу. У документу [Z3983-2] је дат имплементациони профил где су специфицирани детаљи практичне имплементације протокола. Поред овог стандард укључује и *XML* шему којом је дефинисана структура порука које се размењују, као и неколико апликационих профила насталих од стране произвођача софтвера током примене стандарда. Сви ови документи су доступни на званичном сајту *NCIP* стандарда

[NCIP]. Поред овога на сајту се могу наћи и информације о активностима и раду групе за имплементацију, као и информације о самој имплементацији стандарда у оквиру различитих библиотечких система.

2.2 Намена и област примене протокола

NCIP протоколом је дефинисана синтакса и семантика порука које апликације размењују да би се омогућиле функције задуживања и раздуживања књига и контролисање приступа електронским ресурсима. Протоколом нису дефинисане ни функције циркулације ни међубиблиотечке позајмице, него само поруке које се размењују између различитих апликација да би се остариле њихове функције. Протоколом су дефинисани скупови објеката, сервиса, порука и елемената у порукама које апликације размењују.

Протокол може бити примењен у различитим областима, али је првенствено намењен за коришћење у три области: циркулација унутар конзорцијума, повезивање циркулације и међубиблиотечке позајмице и самоуслужно задуживање и раздуживање корисника у библиотеци.

Циркулација унутар конзорцијума је библиотечки процес у ком је корисницима једне библиотеке омогућено да позајмљују књиге из других библиотека унутар конзорцијума. У овом процесу *NCIP* стандард обезбеђује размену података о кориснику и о библиотечкој јединици између две различите апликације за циркулацију с циљем да се библиотеци омогући задуживање корисника из других библиотека и да се локалне полисе о задуживању примењују на те кориснике.

NCIP стандард омогућава размену података о корисницима између апликације за циркулацију и апликације за међубиблиотечку позајмицу с циљем да се у корисниковим задужењима чувају и информације о локалним задужењима и задужењима преко међубиблиотечке позајмице. На тај начин се из апликације за циркулацију могу пратити и третирати исто сва корисникова задужења без обзира да ли су локална или преко међубиблиотечке позајмице, што укључује, на пример, слање опомена за прекорачење рока враћања публикација. Такође, библиотеци која је дала

публикацију на позајмицу преко међубиблиотечке позајмице омогућава се праћење задужења публикације из своје локалне апликације за циркулацију.

Апликације за самоуслужно задуживање и раздуживање корисника омогућавају корисницима да задуже или раздуже публикације без помоћи библиотекара. *NCIP* стандард омогућава размену података о корисницима између ових апликација и апликације за циркулацију.

2.3 Дефиниција сервиса и порука

У даљем тексту ће бити дат кратак опис сервиса и порука дефинисаних *NCIP* стандардом. У стандарду је сервис дефинисан као пар порука који се састоји од захтева и одговора којима се размењују подаци потребни за функције циркулације.

Стандардом су дефинисана три типа сервиса: *Lookup, Update* и *Notification*. Сваки тип сервиса се састоји од одређеног броја сервиса.

У стандарду су дефинисане и три врсте објеката: *Agency, Item* и *User*.

Agency представља библиотеку или организацију која позајмљује јединице из свог фонда и пружа друге услуге корисницима. Овај објекат садржи неколико елемената. То су:

- *Agency Address Information*
- *Agency User Privilege Type*
- *Application Profile Supported Type*
- *Authentication Prompt*
- *Consortium Agreement*
- *Organization Name Information*

Item представља физички или електронски ресурс који припада колекцији агенције и који може бити позајмљен или коме корисник може имати приступ. Садржи следеће елементе:

- *Bibliographic Description*
- *Circulation Status*
- *Electronic Resource*
- *Hold Queue Length*
- *Item Description*
- *Item Use Restriction Type*

- *Location*
- *Physical Condition*
- *Security Marker*
- *Sensitization Flag*

Објекат *User* представља особу или организацију која може да користи примерке или сервисе које пружа агенција. Састоји се од следећих елемената:

- *Authentication Input*
- *Block Or Trap*
- *Date Of Birth*
- *Name Information*
- *Previous User Id(s)*
- *User Address Information*
- *User Language*
- *User Privilege*
- *User Id*

Детаљан опис елемента ових објеката, као и елемената који се размењују у порукама сервиса је дефинисан стандардом и налази се у одељку стандарда *Data Dictionary*.

NCIP је дефинисан као *connection-oriented* протокол што значи да се приликом комуникације две апликације отвара конекција преко које се одвија комуникација, односно размењују поруке. Стандардом није забрањено да две апликације у исто време имају више паралелних конекција. Током комуникације апликације могу да буду у више стања. Приликом успостављања комуникације обе апликације се налазе у стању *Idle*. Када једна апликација пошаље захтев она прелази у стање *Waiting* што значи да чека одговор. Када друга апликација добије захтев она прелази у стање *Processing* што значи да обрађује захтев. Након што апликација врати одговор прелази поново у стање *Idle*, као и друга апликација када прими одговор.

NCIP је такође дефинисан као *sessionless* протокол што значи да су сви сервиси независни један од другог, односно да су све информације потребне за извршавање једног сервиса садржане у захтеву и да се подаци добијени извршавањем претходно позваних сервиса не чувају.

2.3.1 Сервиси типа *Lookup*

Сервиси типа *Lookup* омогућавају једној апликацији да од друге апликације добије информације о инстанцама објеката типа *Agency, Item* и *User*. Апликација која одговара на захтев може у поруци послати захтеване податке или одбити захтев у случају да подаци нису доступни или постоји нека забрана. Постоји дефинисано пет сервиса типа *Lookup*. У табели 2.1 су наведени сервиси са описом њихове улоге.

Табела 2.1 Сервиси типа *Lookup*

Сервис	**Опис сервиса**
Lookup Agency Service	Траже се подаци о одговарајућој агенцији. У захтеву се шаље *Id* агенције и листа елемената који се траже. У одговору се шаљу тражени подаци.
Lookup Item Service	Траже се подаци о одговарајућем примерку. У захтеву се шаље *Id* примерка и листа елемената који се траже. У одговору се шаљу тражени подаци.
Lookup Request Service	Траже се подаци о одговарајућем захтеву. У захтеву се шаље *Id* захтева и листа елемената који се траже. У одговору се шаљу тражени подаци.
Lookup User Service	Траже се подаци о одговарајућем кориснику. У захтеву се шаље *Id* корисника и листа елемената који се траже. У одговору се шаљу тражени подаци.
Lookup Version Service	Траже се подаци о верзији *NCIP* стандарда коју апликација подржава.

Сваки сервис се састоји од две поруке. Једна представља захтев, а друга одговор. У табели 2.2 су наведени називи порука сервиса типа *Lookup*.

Стандардом је дефинисано који подаци се размењују у порукама. Неки су обавезни а неки опциони. У табели 2.3 је приказана дефиниција *Lookup Agency* сервиса. Детаљна дефиниција елемената приказаних у

порукама се налази у одељку стандарда *Data Dictionary*. На сличан начин су дефинисана и преостала четири сервиса типа *Lookup*.

Табела 2.2 Захтеви и одговори сервиса типа *Lookup*

Сервис	**Захтев порука**	**Одговор порука**
Lookup Agency Service	*Lookup Agency*	*Lookup Agency Response*
Lookup Item Service	*Lookup Item*	*Lookup Item Response*
Lookup Request Service	*Lookup Request*	*Lookup Request Response*
Lookup User Service	*Lookup User*	*Lookup User Response*
Lookup Version Service	*Lookup Version*	*Lookup Version Response*

Табела 2.3 Дефиниција сервиса *Lookup Agency*

Lookup Agency	
Обавезни подаци	*Agency Id* *Agency Element Type (R)*
Опциони подаци	Нема
Lookup Agency Response	
Обавезни подаци	*Agency Id, or Problem*
Опциони подаци	Нема
Ако је захтевано	*Agency Address Information (R)* *Agency User Privilege Type (R)* *Application Profile Supported Type (R)* *Authentication Prompt (R)* *Consortium Agreement (R)* *Organization Name Information (R)*

2.3.2 Сервиси типа *Update*

Сервиси типа *Update* омогућавају једној апликацији да од друге

апликације захтева креирање или модификацију (додавање или брисање) података инстанци објеката типа *Agency, Item* и *User* за које је надлежна друга апликација. Сервиси такође укључују и креирање и модификацију веза између инстанци објеката. Апликација која одговара на захтев може у одговору послати потврду захтева или одбити захтев у случају да подаци нису доступни или постоји нека забрана. Постоји дефинисано 22 сервиса типа *Update*. У табели 2.4 су наведени сервиси са описом њихове улоге.

Као и код сервиса типа *Lookup* сваки сервис се састоји од две поруке. Једна представља захтев, а друга одговор. Називи порука су формирани у односу на назив сервиса слично као што је приказано у табели 2.2 за сервисе типа *Lookup*. Стандардом је дефинисано који подаци се размењују у порукама, а дефиниције сервиса су сличне примеру дефиниције сервиса *Lookup Agency* приказаног у табели 2.3.

Табела 2.4 Сервиси типа *Update*

Сервис	**Опис сервиса**
Accept Item Service	Захтева се од апликације да прихвати одговарајући примерак да може да циркулише код корисника. Апликација не мора да има никакве податке ни о примерку ни о кориснику. У захтев се поред података о кориснику и примерку могу укључити датум до кад примерак треба да се врати, да ли може да се продужи датум враћања и финансијски подаци везани за примерак. У одговору се шаље *Id* прихваћеног примерка преко кога се може референцирати на примерак у наредим захтевима.
Check In Item Service	Захтева се од апликације да провери примерак. У захтеву се шаље *Id* примерка, а могу да се траже и додатни подаци о примерку или кориснику везаном за тај примерак. У одговору се шаљу тражени подаци.

Сервис	**Опис сервиса**
Check Out Item Service	Захтева се од апликације да задужи примерак кориснику. У захтеву се шаље *Id* примерка и *Id* корисника и евентуално подаци о надокнади за задужење, а могу да се траже и додатни подаци о примерку или кориснику. У одговору се шаљу тражени подаци.
Undo Check Out Item Service	Захтева се од апликације да поништи претходно задужење. Овај сервис се најчешће користи од стране апикација за самоуслужно задуживање и раздуживање.
Create Agency Service	Захтева се од апликације да креира запис о агенцији. Апликацији се прослеђују потребни подаци за креирање објекта типа *Agency.* У одговору се шаље *Id* агенције.
Create Item Service	Захтева се од апликације да креира запис о примерку. Апликацији се прослеђују потребни подаци за креирање објекта типа *Item*. У одговору се шаље *Id* примерка.
Create User Service	Захтева се од апликације да креира запис о кориснику. Апликацији се прослеђују потребни подаци за креирање објекта типа *User*. У одговору се шаље *Id* корисника.
Create User Fiscal Transaction Service	Захтева се од апликације да креира нову фискалну трансакцију за сервис који се пружа кориснику. Апликацији се прослеђују фискални подаци.

Сервис	**Опис сервиса**
Delete Item Service	Захтева се од апликације да обрише запис о примерку. У зависности од правила апликације ово може резултирати и брисањем библиотечког записа. Сврха овог сервиса је брисање непотребних или привремено креираних записа о примерцима. Апликацији се прослеђује *Id* примерка који се брише.
Delete User Service	Захтева се од апликације да обрише запис о кориснику. Сврха овог сервиса је брисање непотребних или привремено креираних записа о корисницима. Апликацији се прослеђује *Id* корисника који се брише.
Recall Item Service	Захтева се од апликације да поново задужи примерак кориснику. Апликацији се прослеђује нови датум за рок враћања примерка.
Cancel Recall Item Service	Захтева се од апликације да поништи претходни захтев за поновно задуживање примерка.
Renew Item Service	Захтева се од апликације да продужи задужење примерка. Апликацији се прослеђује нови датум за рок враћања и евентуално подаци о надокнади за задуживање.
Report Circulation Status Change Service	Захтева се од апликације да означи примерак да му је промењен статус у враћен, недоступан, изгубљен или никад позајмљен. Апликацији се прослеђује *Id* примерка и нови статус.

Сервис	**Опис сервиса**
Request Item Service	Захтева се од апликације да направи захтев за примерком за одређеног корисника без обзира да ли је тај примерак тренутно доступан или не. Апликацији се прослеђује тип захтева и подаци о надокнади. У одговору се враћају информације о томе када ће примерак бити доступан и где се може преузети.
Cancel Request Item Service	Захтева се од апликације да поништи претходни захтев за примерком.
Send User Notice Service	Захтева се од апликације да пошаље обавештење кориснику. Апликацији се прослеђује *Id* корисника, тип и садржај обавештења и евентуално датум слања обавештења.
Update Agency Service	Захтева се од апликације да модификује податке о агенцији. Апликацији се прослеђују подаци који се бришу из инстанце објекта и који се додају. Мењање неког податка се врши тако што се обрише стара вредност и дода нова.
Update Circulation Status Service	Захтева се од апликације да промени статус примерака. Апликацији се прослеђује *Id* примерка и нови статус.
Update Item Service	Захтева се од апликације да модификује податке о примерку. Апликацији се прослеђују подаци који се бришу из инстанце објекта и који се додају. Мењање неког податка се врши тако што се обрише стара вредност и дода нова.

Сервис	Опис сервиса
Update Request Item Service	Захтева се од апликације да модификује податке у захтеву за примерком одговарајућег корисника. Апликацији се прослеђују подаци који се бришу из инстанце објекта и који се додају. Мењање неког податка се врши тако што се обрише стара вредност и дода нова.
Update User Service	Захтева се од апликације да модификује податке о кориснику. Апликацији се прослеђују подаци који се бришу из инстанце објекта и који се додају. Мењање неког податка се врши тако што се обрише стара вредност и дода нова.

2.3.3 Сервиси типа *Notification*

Сервиси типа *Notification* омогућавају једној апликацији да обавести другу апликацију да је дошло до креирања или модификације инстанци објеката типа *Agency, Item* и *User*, као и веза између инстанци објеката. Апликација која одговара у свом одговору шаље потврду о пријему обавештења и да ли је обавештење разумела или не. Постоји дефинисано 20 сервиса типа *Notification*. У табели 2.5 су наведени сервиси са описом њихове улоге.

Као и код претходна два типа сервиса сваки сервис се састоји од две поруке где једна представља захтев, а друга одговор и називи порука су формирани у односу на назив сервиса. Дефиниције сервиса су сличне примеру дефиниције сервиса *Lookup Agency* приказаног у табели 2.3.

Табела 2.5 Сервиси типа *Notification*

Сервис	Опис сервиса
Agency Created Service	Обавештава се апликација да је креирана нова агенција. У поруци се шаље *Id* агенције и подаци о агенцији.
Agency Updated Service	Обавештава се апликација да су модификовани подаци о агенцији. У

Сервис	**Опис сервиса**
	поруци се шаљу подаци који су обрисани и подаци који су додати.
Circulation Status Change Reported Service	Обавештава се апликација да је примерак означен да му је промењен статус у враћен, недоступан, изгубљен или никад позајмљен. У поруци се шаље *Id* примерка и нови статус.
Circulation Status Updated Service	Обавештава се апликација да је статус примерка промењен. У поруци се шаље *Id* примерка и нови статус.
Item Checked In Service	Обавештава се апликација да је примерак проверен. У поруци се шаље *Id* примерка.
Item Checked Out Service	Обавештава се апликација да је примерак задужен одговарајућем кориснику. У захтеву се шаље *Id* примерка и *Id* корисника
Item Created Service	Обавештава се апликација да је креиран нови примерак. У поруци се шаље *Id* примерка и подаци о примерку.
Item Recall Cancelled Service	Обавештава се апликација да је поништен претходни захтев за поновно задуживање примерка. У поруци се шаље *Id* примерка.
Item Recalled Service	Обавештава се апликација да је поново задужен примерак кориснику. У поруци се шаље *Id* примерка и нови датум за рок враћања примерка.
Item Received Service	Обавештава се апликација да је примљен примерак. Ако је примерак примљен у име корисника у поруци се шаљу и подаци о том кориснику.
Item Renewed Service	Обавештава се апликација да је продужено задужење примерка. У поруци

Сервис	**Опис сервиса**
	се прослеђује нови датум за рок враћања.
Item Request Cancelled Service	Обавештава се апликација да је поништен претходни захтев за примерком.
Item Request Updated Service	Обавештава се апликација да су модификовани подаци у захтеву за примерком одговарајућег корисника. У поруци се прослеђују подаци који су обрисани и који су додати.
Item Requested Service	Обавештава се апликација да је направљен захтев за примерком за одређеног корисника без обзира да ли је тај примерак тренутно доступан или не.
Item Shipped Service	Обавештава се апликација да је агенција послала тражени примерак. У поруци су укључени датум слања и адреса на коју је послато. Ако се пошиљка шаље на захтев корисника у поруци могу бити укључени и подаци о кориснику.
Item Updated Service	Обавештава се апликација да су модификовани подаци о примерку. У поруци се прослеђују подаци који су обрисани и који су додати.
User Created Service	Обавештава се апликација да је креиран запис о кориснику. У поруци се прослеђују *Id* корисника и подаци о имену корисника.
User Fiscal Transaction Created Service	Обавештава се апликација да је креирана нова фискална трансакција на корисниковом фискалном налогу.
User Notice Sent Service	Обавештава се апликација да је послано обавештење кориснику. У поруци се прослеђује *Id* корисника, тип обавештења

Сервис	Опис сервиса
	и подаци о обавештењу.
User Updated Service	Обавештава се апликација да су модификовани подаци о кориснику. У поруци се прослеђују подаци који су обрисани и који су додати.

2.4 Имплементација протокола

Као подршка имплементацији протокола уз дефиницију протокола направљен је импементациони профил и неколико апликационих профила. У имплементационом профилу се налазе детаљи потребни за имплементацију протокола. Описано је како се поруке размењују, односно дата је детаљна спецификација порука, енкодинг података и транспортни механизми који се користе за размену порука. Апликациони профили описују како се протокол користи у различитим окружењима и применама. Описано је који сервиси морају да буду подржани, ниво сигурности и догађаји и околности у којима се сервиси користе. Постоји осам апликационих профила. Четири апликациона профила описују како се протокол користи у области циркулације унутар конзорцијума, два описују могућу комуникацију између апликације за циркулацију и апликације за међубиблиотечку позајмицу и још два описују комуникацију између апликације за самоуслужно задуживање и циркулације.

Приликом дефинисања протокола циљ је био да се омогући што лакша имплементација протокола, као и примена протокола у различитим областима. Због тога је одлучено да се при дефиницији имплементације користе само технологије које су у широкој употреби и широко прихваћене. То је условило одбацивање неких веома обећавајућих технологија из разлога што није сигурно да ли ће у будућности бити у широј употреби.

2.4.1 Поруке

За енкодинг порука изабран је *XML*. Упркос чињеници да се у области

библиотекарства највише користи *ASN.1/BER* формат у овом случају је изабран *XML*, јер је *XML* постао доминантан енкодинг метод у Интернет комуникацијама и постоји велики избор алата за рад са *XML*-ом. Употребом *XML*-а у библиотечким апликацијама олакшаће се и повезивање библиотека са другим електронским сервисима доступним на Интернету. Још један разлог за избор *XML*-а је његова проширивост. Различита пракса у библиотекама у области циркулације захтева протокол који је проширив и прилагодљив локалним потребама библиотеке. У дефиницији протокола је остављена могућност библиотекама да саме дефинишу неке елементе додатним *XML* шемама.

Тренутна верзија *NCIP XML* шеме којом су дефинисане поруке налази се на званичном сајту *NCIP* стандарда. За сваку поруку која је дефинисана стандардом постоји један елемент у шеми и поруке које се размењују по *NCIP* стандарду морају да буду валидне у односу на ту шему.

2.4.2 Карактери

За енкодинг карактера изабран је *Unicode* (*UCS-2*) и то енкодинг шема *UTF-8* јер се *UTF-8* користи у Интернет стандардима, а такође омогућава апликацијама које користе *ASCII* шему да остану усаглашене са стандардом. Апликације морају да имају могућност да препознају све *Unicode* карактере као валидне карактере, али свака апликација бира који скуп *Unicode* знакова ће да користи приликом размене порука.

2.4.3 Транспорт порука

За транспорт порука омогућено је апликацијама да користе један од три протокола:

- *HTTP*
- *HTTPS*
- *TCP/IP*

Оваква одлука је донешена на основу нивоа сигурности који је потребно обезбедити и апликативних окружења у којима ће се

имплементирати протокол. Апликација која шаље захтев бира протокол који ће користити, а апликација која одговара користи исти тај протокол. Такође разматрана је употреба *SOAP* протокола у комуникацији. И поред разних предности, употреба овог протокола није одобрена јер *SOAP* није у потпуности стандардизован протокол.

2.5 Апликациони профили

Као подршка имплементацији протокола направљени су апликациони профили. Апликациони профили описују како се протокол користи у различитим окружењима и у односу на окружење дефинишу који сервиси морају бити подржани, транспортни протокол који ће да се користи, ниво сигурности који треба да се обезбеди и др. Постоји осам апликационих профила којима је описана употреба *NCIP* протокола у различитим областима. *NCIP* протокол је намењен за коришћење у три области: циркулација унутар конзорцијума, повезивање циркулације и међубиблиотечке позајмице и самоуслужно задуживање и раздуживање корисника у библиотеци. Четири апликациона профила описују како се протокол користи у области циркулације унутар конзорцијума, два описују могућу комуникацију између апликације за циркулацију и апликације за међубиблиотечку позајмицу и још два описују комуникацију између апликације за самоуслужно задуживање и циркулације. Апликациони профили се могу наћи на званичном сајту *NCIP* протокола. Неки су званично објављени а неки су у *Draft* верзији.

Циркулација унутар конзорцијума омогућава корисницима једне библиотеке да позајмљују књиге из других библиотека унутар конзорцијума. У овом процесу *NCIP* стандард обезбеђује размену података о кориснику и о библиотечкој јединици између две различите апликације за циркулацију с циљем да се библиотеци омогући задуживање корисника из других библиотека. У зависности од тога да ли ће се на корисника из друге библиотеке примењивати локалне полисе за задуживање или полисе из његове библиотеке или ће о трансакцијама да води рачуна нека трећа страна направљена су четири различита апликациона профила за примену *NCIP* протокола у области циркулације унутар конзорцијума. То су профили:

- *Item Agency Manages Transaction* (*DCB-1*)

- *User Agency Manages Transaction* (*DCB-2*)
- *Broker Application Manages Transaction* (*DCB-3*)
- *User Agency Manages Transaction with Proxy Check Out* (*DCB-4*)

2.6 Постојеће имплементације протокола у библиотечким системима

Произвођачи библиотечког софтвера који су у својим софтверским решењима имплементирали *NCIP* протокол имају могућност да на званичном сајту *NCIP* протокола објаве кратак извештај о својој имплементацији. Произвођачи ове извештаје шаљу добровољно и по жељи тако да и детаљност извештаја зависи од одлуке произвођача. Извештаји садрже информације о томе у којој је фази имплементација протокола (развој, тестирање или употреба), које транспортне протоколе подржава, коју улогу у комуникацији има (иницира комуникацију, одговара на захтеве или обе), који сервиси су имплементирани, са којим библиотечким системима размењују поруке и др. У табели 2.6 су приказани детаљи неких извештаја. Приказано је само неколико произвођача чији извештаји су били најдетаљнији, а од детаља имплементације су наведени улога, подржани сервиси и подржани транспортни протоколи.

Табела 2.6 Имплементације протокола

Произвођач	**Улога**	**Транспортни протокол**	**Сервиси**
Ex Libris: Aleph	*Responder*	*TCP -- NCIP Server* *HTTP / HTTPS -- Aleph WWW Server*	*Lookup Item Service* *Lookup User Service* *Lookup Version Service* *Authenticate User Service* *Accept Item Service* *Cancel Request Item Service* *Check Out Item Service* *Check In Item Service* *Request Item Service*

Произвођач	**Улога**	**Транспортни протокол**	**Сервиси**
Ex Libris: Voyager	*Responder*	*TCP/IP*	*LookupVersion* *LookupUser* *LookupItem* *RequestItem* *CancelRequestItem* *CheckOutItem* *CheckInItem* *AcceptItem* *RenewItem* *CreateUser*
Innovative Interfaces	*Initiator*	*HTTPS*	*AcceptItem* *AuthenticateUser* *CheckInItem* *CheckOutItem* *ItemCheckedIn* *ItemCheckedOut* *ItemRecalled* *ItemReceived* *ItemRenewed* *ItemRequestCancelled* *ItemRequested* *ItemShipped* *LookUpUser* *RenewItem*

Произвођач	**Улога**	**Транспортни протокол**	**Сервиси**
Polaris Library Systems	*Initiator* *Responder*	*HTTPS* *TCP*	*Authenticate User* *Lookup Item* *Lookup User* *Lookup Version* *Accept Item* *Cancel Request Item* *Check In Item* *Check Out Item* *Create User* *Create User Fiscal Transaction* *Renew Item* *Request ItemR* *Undo Check Out Item* *Update Request Item* *Item Checked In* *Item Checked Out* *Item Request Cancelled* *Item Requested*
OCLC	*Initiator*	*HTTPS*	*LookupAgency* *LookupUser* *LookupItem* *LookupRequest* *RequestItem* *CancelRequestItem* *UpdateRequestItem* *CheckoutItem* *CheckinItem* *RenewItem* *AcceptItem* *CreateUser* *ItemCheckedOut* *ItemCheckedIn* *ItemRenewed*

Произвођач	**Улога**	**Транспортни протокол**	**Сервиси**
SirsiDynix	*Initiator* *Responder*	*TCP* *HTTP* *HTTPS*	*As Initiator:* *Lookup User* *Lookup Item* *Request Item* *Accept Item* *Checkout Item* *Checkin Item* *Recall Item* *Cancel Request Item* *Renew Item* *Lookup User* *Create Patron* *As Responder:* *Lookup Item* *Request Item* *Accept Item* *Checkout Item* *Checkin Item* *Cancel Request Item* *Renew Item* *Lookup User* *Create Patron* *Authenticate User* *Item Checked out* *Item request cancelled* *Item Checked in* *Item requested* *Lookup Agency*

Поглавље 3

Циркулација у систему БИСИС верзије 4

У овом поглављу приказани су резултати добијени на развоју четврте верзије БИСИС система и то дела који се односи на циркулацију. Систем за циркулацију је резултат истраживања магистарске тезе (Тешендић, 2007), а резултат је објављен и у раду (Tešendić et al., 2009). Детаљан опис моделирања, имплементације и коришћења система дат је у монографији (Тешендић и Сурла, 2007). Будући да је истраживање приказано у дисертацији наставак овог истраживања у овом поглављу је дат текст који је настао на основу садржаја поменуте три публикаиције.

Моделирање и имплементација система за циркулацију је урађено на основу више аспеката. Први је искуство и захтеви библиотека које користе БИСИС, затим анализа постојећих софтверских решења и анализа доступних научних радова у тој области. Преглед постојећих софтверских решења и релевантне литературе је дат у уводном поглављу.

Искуство библиотека које користе БИСИС је довело до формирања захтева за новим функционалностима или проширењем постојећих функционалности у односу на функционалнисти које су постојале у трећој верзији система. Ти захтеви су се највише односили на прегледност екранских форми, перформансе система и извештавање. Што се тиче прегледности екранских форми захтеви су се највише односили на информације које треба да буду приказане на одређеним екранским формама, логичко груписање тих информација и доступност осталих информација које нису ту приказане. Захтеви за бољим перформансама система првенствено су се односили на одзив система, као и разне пречице са тастатуре које омогућавају бржи рад

библиотекара. Захтеви за извештавањем су укључивали одређен број предефинисаних извештаја као и специфичне статистичке информације које се динамички генеришу.

Поред ових захтева који се тичу саме апликације за циркулацију, постојао је и захтев да се омогући рад система у одељењима и огранцима библиотеке који се налазе на физички различитим локацијама. Одељења која се налазе на истој локацији као и сервер, са сервером су обично повезана преко локалне мреже. Одељења која се не налазе на истој локацији, комуникацију са сервером обично остварују преко Интернета.

На основу наведених захтева, као и постојећих софтверских решења и предлога развијен је систем за циркулацију који задовољава све потребе електронског пословања градских и специјализованих библиотека. Овај систем је укључен у систем БИСИС верзије 4.

Систем за циркулацију је направљен као независна софтверска компонента која се може интегрисати у различите библиотечке системе. То значи да систем има подршку за комуникацију са различитим базама библиографских записа, као и подршку за различите *MARC* формате. Клијентска апликација система је реализована као десктоп апликација и може да се извршава на различитим оперативним системима. Комуникација клијентске апликације са базом података је унифицирана коришћењем интерфејса који има две имплементације, за комуникацију у локалном интранету и комуникацију на Интернету. На овај начин омогућен је рад клијентске апликације са физички удаљених локација библиотеке.

Систем подржава све стандардне активности за рад са корисницима које подржавају и софтверска решења наведена у уводном поглављу, а то су: евиденцију, задуживање и раздуживање корисника; претраживање корисника и публикација; генерисање различитих врста извештаја, као и опомена корисницима.

У овом поглављу су приказана софтверска архитектура и имплементација система за циркулацију. Софтверска архитектура је моделирана у обједињеном језику моделирања *UML* 2.0. У првом одељку дата је спецификација функционалних карактеристика система помоћу дијаграма случајева коришћења, а у другом одељку је приказан статички модел система помоћу дијаграма компоненти,

дијаграма пакета и дијаграма класа. Након архитектуре система у трећем одељку овог поглавља приказани су и објашњени делови имплементације система.

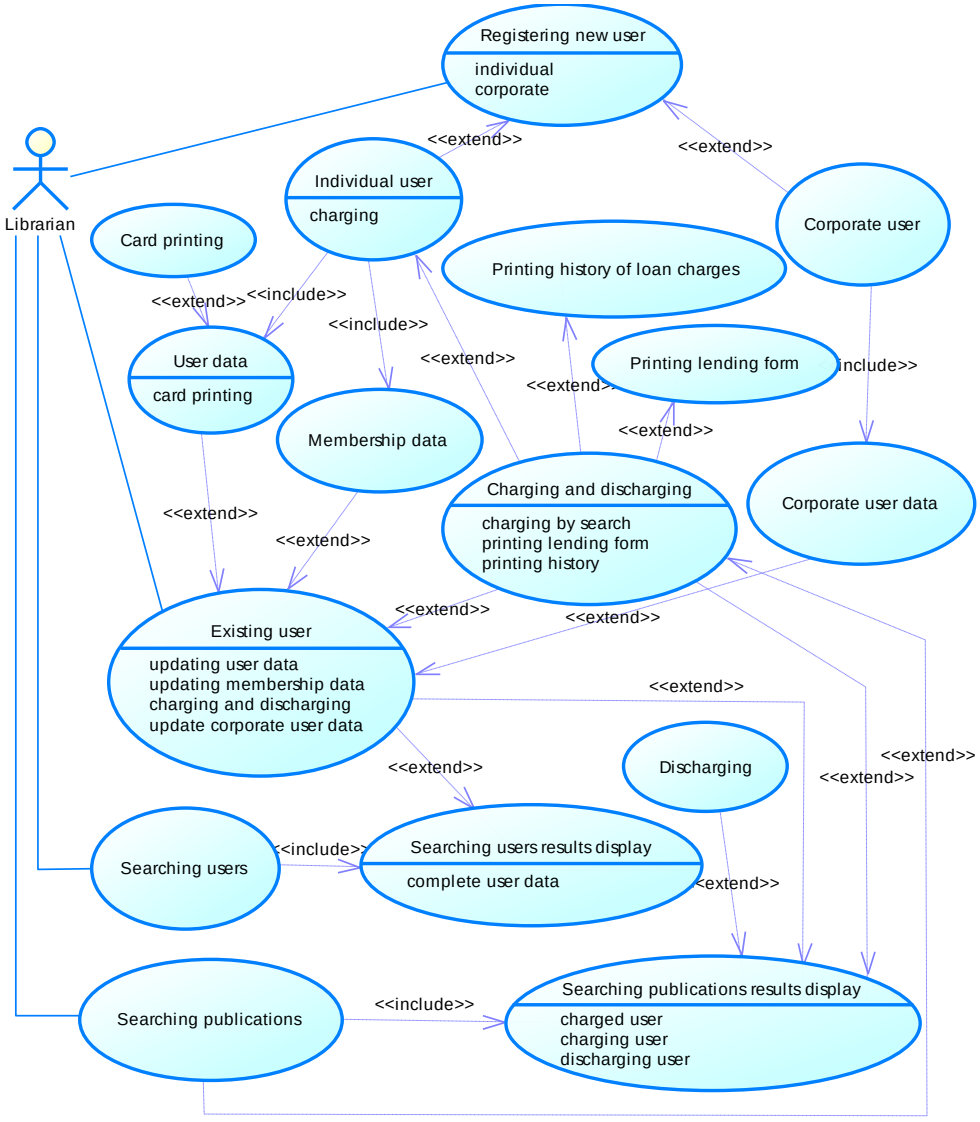
3.1 Опис функционалности система

Спецификација захтева система је представљена са три дијаграма случајева коришћења. Основне функционалности система су представљене на дијаграму случајева коришћења *Рад са корисницима*, који је приказан на слици 3.1. На њему су приказане акције које библиотекар свакодневно врши у раду са корисницима. Ту спадају учлањење нових корисника, продужење чланарине, задуживање и раздуживање корисника и претраживање фонда. Остале функционалности система као што су генерисање извештаја, генерисање опомена и подешавање параметара система за рад у конкретној библиотеци специфициране су на преостала два дијаграмима, који су овде изостављени.

Текстуални опис случајева коришћења са слике 3.1:

Случај коришћења *Registering new user*. Библиотекар уписује новог корисника у библиотеку. Постоје две врсте корисника: индивидуални и колективни корисници. Библиотекар бира врсту корисника и зависно од избора реализује се случај коришћења *Individual user*, односно случај коришћења *Corporate user*.

Случај коришћења *Individual user*. Библиотекар уписује индивидуалног корисника у библиотеку. Индивидуални корисник је појединац који може да користи фонд библиотеке. Појединац може да се упише у библиотеку и преко колектива који је већ уписан у библиотеку као колективни корисник. У том случају постоји информација преко ког колективног корисника је уписан тај индивидуални корисник. У оквиру овог случаја коришћења реализују се случајеви коришћења *User data* и *Membership data*. Ако корисник при упису жели да задужи неке публикације реализује се и случај коришћења *Charging and discharging*.



Слика 3.1 *Дијаграм случајева коришћења Рад са корисницима*

Случај коришћења *User data*. Библиотекар уноси или ажурира податке о индивидуалном кориснику. У те податке спадају: име и презиме, име родитеља, адреса и место становања, број телефона, *e-mail* адреса, ЈМБГ, број документа и место издавања документа, пол, узраст, додатна адреса, степен образовања, занимање, звање, организација у којој је корисник запослен, матерњи језик, интересовања и напомена. Библиотека одређује обавезност уноса ових података. Ако

библиотекар жели да штампа картицу реализује се случај коришћења *Card printing*.

Случај коришћења *Membership data*. Библиотекар уноси или ажурира податке о чланству корисника у библиотеку. У те податке спадају: број корисника, ако се корисник уписује преко колективног корисника тада се повезује са подацима о колективном кориснику, категорија корисника, врста учлањења, датум од кад важи чланарина, датум до кад важи чланарина, одељење на које се корисник уписује, цена, и број признанице која се издаје кориснику за плаћену чланарину. Библиотека одређује обавезност уноса ових података.

Случај коришћења *Charging and discharging*. Приказује се списак тренутних задужења корисника. Ако корисник враћа неку од задужених публикација библиотекар може да га раздужи или може да продужи задужење. Ако корисник жели да задужи неку публикацију библиотекар може задуживање да изврши уношењем инвентарног броја публикације или проналажењем публикације претраживањем. У случају проналажења публикације претраживањем реализује се случај коришћења *Searching publications.* Ако библиотекар жели да штампа реверс кориснику, тада се реализује случај коришћења *Printing lending form.* Ако библиотекар жели да види или штампа претходна задужења корисника, тада се реализује случај коришћења *Printing history of loan charges*.

Случај коришћења *Card printing*. Библиотекару се приказује изглед картице коју може и да штампа.

Случај коришћења *Printing lending form*. Библиотекару се приказује изглед реверса који може и да штампа. Реверс је документ на ком се приказују тренутна задужења корисника. Стављањем потписа на реверс корисник потврђује тачност података приказаних на реверсу.

Случај коришћења *Printing history of loan charges*. Библиотекар задаје временски период за који жели да види историју задуживања. Приказује му се списак задужења корисника у том периоду који може по потреби да штампа.

Случај коришћења *Corporate user*. Библиотекар уписује колективног корисника у библиотеку. Колективни корисник је организација која учлањењем у библиотеку својим припадницима омогућава одређене

погодности при њиховом учлањивању у библиотеку. Те погодности зависе искључиво од уговора између колективног корисника и библиотеке. У оквиру овог случаја коришћења се реализује случај коришћења *Corporate user data.*

Случај коришћења *Corporate user data*. Библиотекар уноси или ажурира податке о колективном кориснику. У те податке спадају: број корисника, назив организације, адреса и место организације, број телефона, број телефакса, *e-mail* адреса, додатна адреса, подаци о контакт особи из организације у које спадају име, презиме и *e-mail* адреса особе. Библиотека одређује обавезност уноса података.

Случај коришћења *Existing user*. До реализације долази у три случаја:

- иницирањем од стране библиотекара и у том случају библиотекар уноси број корисника,
- при реализацији случаја коришћења *Searching users results display* изабран преглед података о неком кориснику из резултата,
- при реализацији случаја коришћења *Searching publications results display* изабран приказ података о кориснику код кога је задужена публикација.

У зависности од тога да ли је корисник индивидуални или колективни реализују се следећи случајеви коришћења:

- у случају индивидуалног корисника случајеви коришћења *User data, Membership data* и *Charging and discharging,*
- у случају колективног корисника случај коришћења *Corporate user data.*

Случај коришћења *Searching users*. Омогућено је претраживање по сваком податку о кориснику који се чува у систему. Претраживање се врши укрштањем до пет података операторима *and, or* и *not.* Поред тога ти подаци се могу укрстити са периодом у ком је корисник задуживан или раздуживан и у ком је вршио продужење чланарине. Након извршеног претраживања реализује се случај коришћења *Searching users results display.*

Случај коришћења *Searching users results display*. Приказује се број погодака и погоци. За сваки погодак, тј. корисника који је задовољио упит приказује се: број корисника, име, презиме, име родитеља,

адреса и место становања. Ако библиотекар жели да види остале податке, тада се реализује случај коришћења *Existing user.*

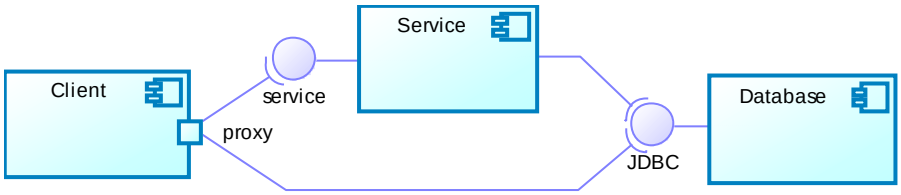
Случај коришћења *Searching publications*. Омогућено је претраживање по дефинисаним префиксима текст сервера библиотечког система БИСИС. Претраживање се врши укрштањем до пет префикса операторима *and, or* и *not.* Поред тога префикси се могу укрстити са периодом у ком је публикација задужена или раздужена. Након извршеног претраживања реализује се случај коришћења *Searching publications results display.*

Случај коришћења *Searching publications results display*. Приказује се број погодака и погоци. Омогућен је приказ погодака на нивоу записа и на нивоу примерака. За сваки примерак је означено да ли може да се задужи или не. Ако је примерак задужив, тада се случај коришћења може проширити реализацијом случаја коришћења *Charging and discharging.* Ако је примерак задужен, тада се случај коришћења може проширити реализацијом случаја коришћења *Discharging.* Ако библиотекар жели да види податке о кориснику код кога је примерак задужен, тада се реализује случај коришћења *Existing user.* За сваки погодак се приказује кратак инфо о публикацији који садржи одређени подскуп информација из записа.

Случај коришћења *Discharging*. Врши се раздуживање публикације изабране при реализацији случаја коришћења *Searching publications results display.*

3.2 Статички модел система

Структура система је приказана на дијаграму компоненти на слици 3.2. Систем се састоји из три дела: клијентске апликације, серверског дела апликације и базе података. Као што је приказано на дијаграму клијентска апликација са базом података може да комуницира на два начина: директно или преко сервиса на серверској страни апликације. Директна комуникација се користи у случају када се клијент и база података налазе у истој локалној мрежи, а комуникација преко сервиса се користи за удаљене клијенте и реализује се преко Интернет протокола.



3.2 Дијаграм компоненти система

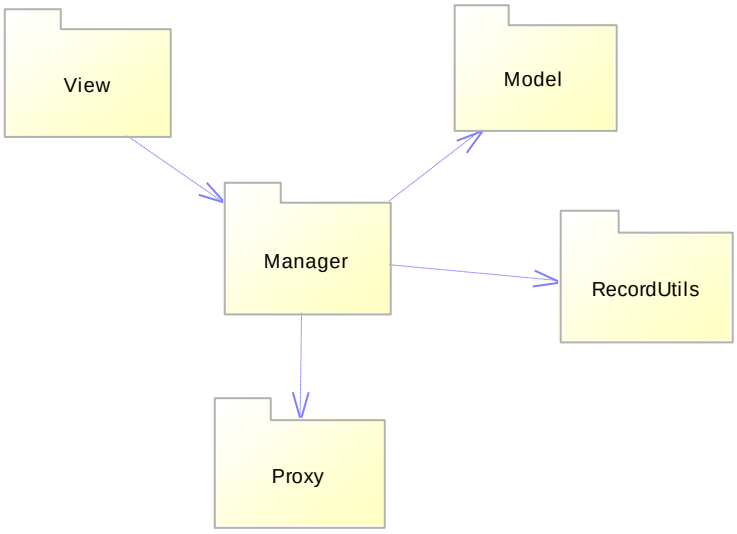
Детаљи модела су у даљем тексту описани на једном дијаграму пакета и четири дијаграма класа. Дијаграм пакета представља структуру клијентске апликације система. На дијаграму класа *Кориснички интерфејс* приказане су класе корисничког интерфејса тј. екранских форми преко којих корисник комуницира са системом. На дијаграму класа *Подаци* приказан је логички модел базе података и на основу тог модела се генерише физички модел базе података. На дијаграму класа *Proxy* су приказане класе које се користе за комуникацију са базом података, а на дијаграму класа *RecordUtils* су приказане класе које представљају модел библиотечког записа у систему за коришћење библиотечке грађе.

3.2.1 Дијаграм пакета клијентске апликације

На слици 3.3 је приказан дијаграм пакета којим је описана структура клијентске апликације система за циркулацију. Апликација има архитектуру *MVC* (*Model–View–Controller*) патерна. *MVC* патерн дели композицију апликације на три дела: презентацијски део, односно кориснички интерфејс, затим модел података с којим апликација ради и део који контролише токове података од корисничког интерфејса до модела и обрнуто. У случају апликације система за циркулацију презентацијски део представља пакет *View*, део за контролу је пакет *Manager*, а модел података чине пакети *Model*, *RecordUtils* и *Proxy*.

Пакет *View* садржи класе корисничког интерфејса и оне су детаљније описане и приказане у одељку 3.2.2.

Пакет *Model* садржи класе које представљају објектни модел базе података и служе за привремено складиштење података с којима систем тренутно ради. Ове класе су описане и приказане у одељку 3.2.3.



Слика 3.3 Дијаграм пакета

Пакет *Manager* садржи класе које управљају током података од корисничког интерфејса до базе података и обрнуто. Овај пакет је описан у одељку 3.2.4.

Пакет *Proxy* садржи интерфејс *Proxy* и његове две имплементације. Преко овог интерфејса класе из пакета *Manager* комуницирају са физичком базом података. Овај пакет је описан у одељку 3.2.5.

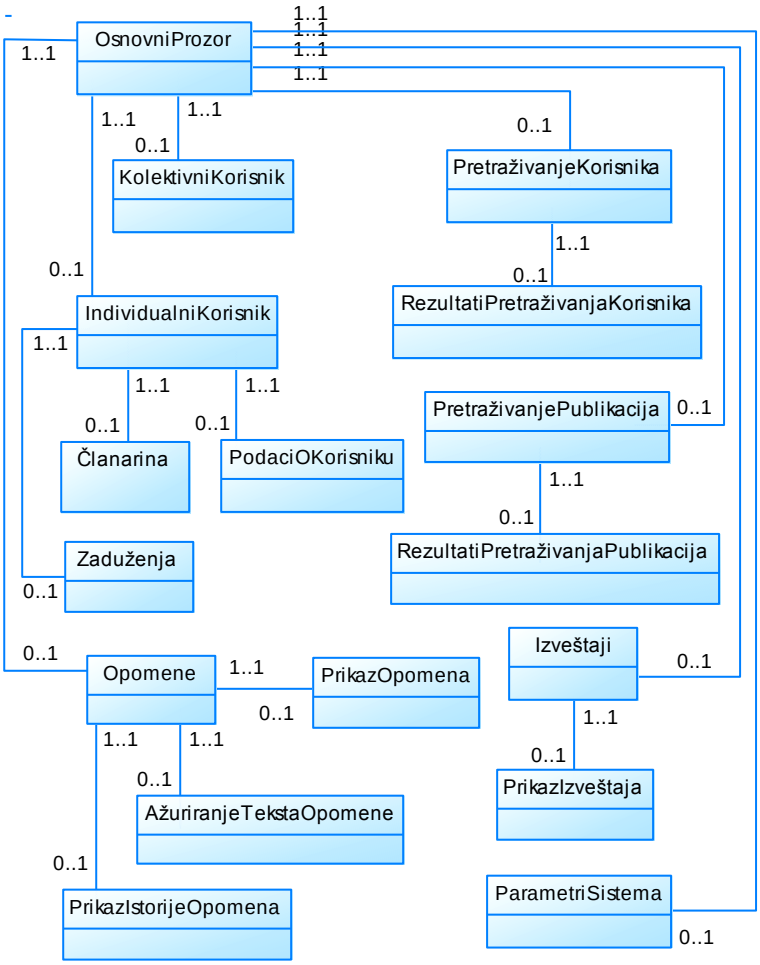
Пакет *RecordUtils* садржи интерфејс који представља модел библиотечког записа у систему за коришћење библиотечке грађе. Овај интерфејс има две имплементације. Пакет је описан у одељку 3.2.6.

3.2.2 Класе пакета *View*

На слици 3.4 је дат дијаграм класа пакета *View.* На дијаграму су приказане основне класе корисничког интерфејса које имају хијерархијску структуру и везе између класа су усмерене асоцијације. Ове класе подржавају функционалности дефинисане дијаграмима случајева коришћења у одељку 3.1. Атрибути ових класа су дефинисани на основу података описаних у случајевима коришћења датих дијаграма.

Преко класе *OsnovniProzor* корисник бира акцију коју жели да изврши. За остваривање функционалности дефинисане на дијаграму случајева

коришћења *Рад са корисницима* који је приказан у одељку 3.1 користе се класе *IndividualniKorisnik, KolektivniKorisnik, PodaciOKorisniku, Članarina, Zaduženja, PretraživanjeKorisnika, RezultatiPretraživanjaKorisnika, PretraživanjePublikacija* и *RezultatiPretraživanjaPublikacija.* Преостале класе се користе за остваривање функционалности са дијаграма случајева коришћења који су овде изостављени.



Слика 3.4 Дијаграм класа *Кориснички интерфејс*

Преко класа *IndividualniKorisnik* и *KolektivniKorisnik* корисник система (библиотекар) има приступ подацима индивидуалних, односно колективних корисника библиотеке и може да иницира акције предвиђене за рад са тим подацима. Класа *IndividualniKorisnik* користи

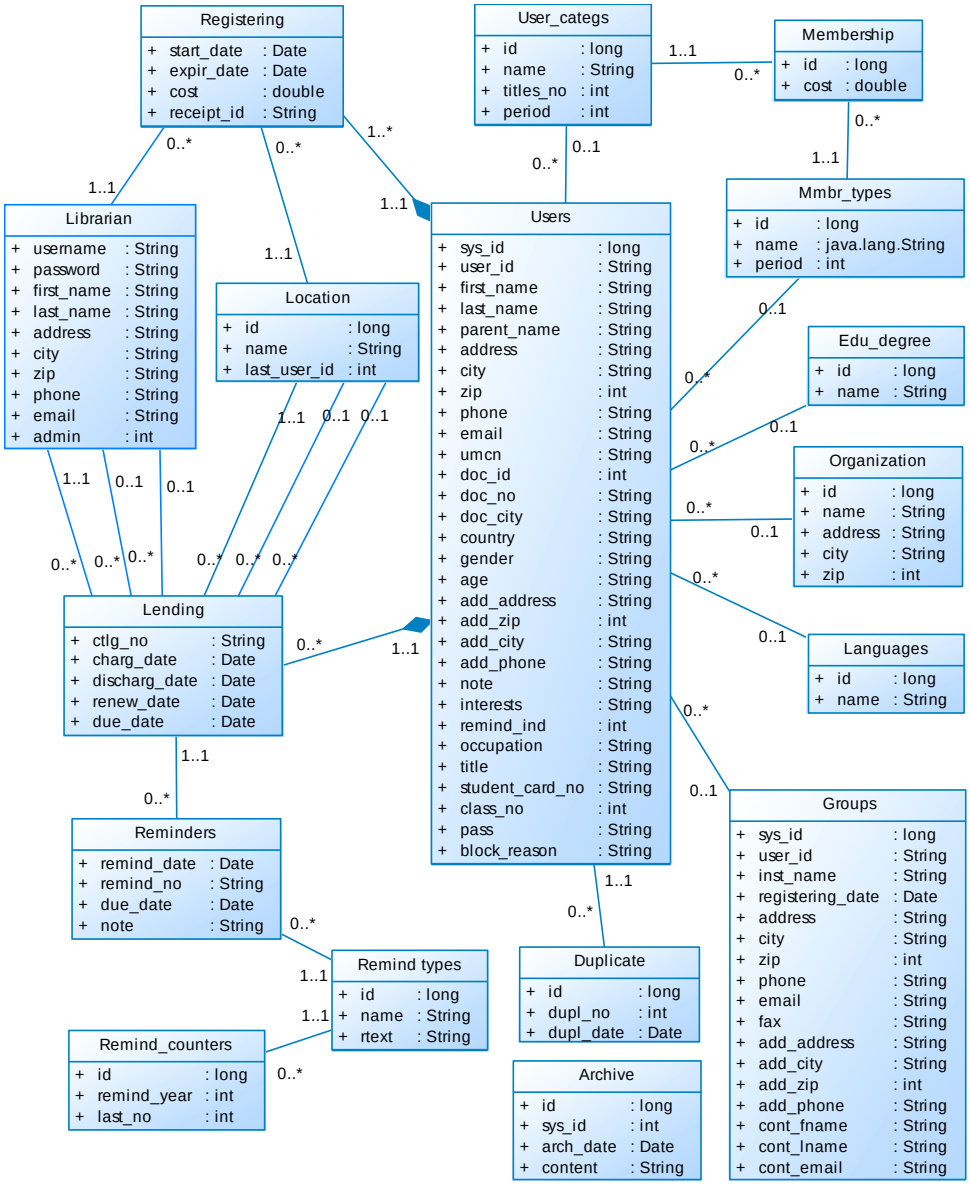
класу *PodaciOKorisniku* за приступ основним подацима о кориснику, класу *Članarina* за приступ подацима о чланству корисника и класу *Zaduženja* за приступ подацима о задужењу корисника.

Преко класе *PretraživanjeKorisnika* корисник система (библиотекар) задаје критеријуме за претраживање корисника библиотеке. Резултати тог претраживања се приказују преко класе *RezultatiPretraživanjaKorisnika.* Слично томе преко класе *PretraživanjePublikacija* корисник система задаје критеријуме за претраживање публикација, а резултати претраживања се приказују преко класе *RezultatiPretraživanjaPublikacija.*

Преко класе *Izveštaji* се врши избор извештаја који систем треба да генерише кориснику. Генерисани извештај се кориснику приказује преко класе *PrikazIzveštaja*. Преко класе *Opomene* корисник приступа генерисању опомена, прегледу историје генерисаних опомена и ажурирању текста опомене. Те акције корисник извршава преко класа *PrikazOpomena, PrikazIstorijeOpomena* и *AžuriranjeTekstaOpomene.* Преко класе *ParametriSistema* администратор система врши подешавање параметара система.

3.2.3 Класе пакета *Model*

На слици 3.5 је дат дијаграм класа пакета *Model*. На њему је приказан објектни модел базе података. Овај модел у систему има две функције. Прва функција је да га апликација користи за привремено складиштење података с којима тренутно ради. Друга функција је да се на основу овог модела генерише физички модел базе података која је приказана као компонента *Database* на дијаграму компоненти на слици 3.2. У раду (Tešendić, 2007) описана је прва верзија овог дијаграма. Развојем система дошло је до промена на дијаграму и крајњи резултат је дијаграм дат на слици 3.5.



Слика 3.5 *Дијаграм класа Подаци*

Класом *Users* је описан индивидуални корисник библиотеке, а класом *Groups* колективни корисник. Атрибути ових класа су настали из спецификације случајева коришћења. Класа *Registering* описује податке о чланарини индивидуалног корисника. За сваког корисника постоји бар један објекат ове класе. Приликом уноса података о чланству води се евиденција на којој локацији су унешени и који

библиотекар је то урадио. Класа *Lending* описује податке о задужењима и раздужењима корисника. Слично као за чланарину за сваку публикацију се води евиденција о локацији и библиотекару који су је задужили, продужили и раздужили. Класа *Duplicate* садржи информације о дупликатима картица корисника. Поред ових ту су и класе које представљају шифарнике из којих се узимају вредности. Класа *Location* је шифарник локација на којима се врши задуживање и учлањење. Класа *User_categs* је шифарник категорија корисника. Класа *Mmbr_types* је шифарник врста учлањења. Класа *Membership* је шифарник цена чланарине. Класа *Edu_degree* је шифарник степена образовања. Класа *Language* је шифарник језика. Класа *Organization* је шифарник организација. Класа *Librarian* је шифарник библиотекара који имају приступ систему.

Класа *Reminders* описује податке о генерисаним опоменама за прекорачење рока враћања публикације. Класа *Remind_types* је шифарник типова опомена. Класа *Remind_counters* садржи податке о бројачима бројева опомена за сваку годину. Класа *Archive* представља архиву података о корисницима.

3.2.4 Пакет *Manager*

Пакет *Manager* садржи две класе: *UsersManager* и *RecordsManager*.

Класа *UsersManager* управља током података о корисницима од корисничког интерфејса до базе података и обрнуто. У основи она обавља две врсте активности:

1. на захтев класа корисничког интерфејса из пакета *View*, класа *UsersManager* добавља и смешта податке из и у базу података преко класа пакета *Proxy*; ти подаци су на клијентској страни привремено смештени у објектима класа из пакета *Model*.
2. пребацује податке из модела на кориснички интерфејс и обрнуто.

Класа *RecordsManager* врши комуникацију између система за коришћење библиотечке грађе и библиотечких записа. У систему за коришћење библиотечке грађе постоји потреба за приказом података из библиотечких записа, као и потреба за променом статуса појединачног примерка из задуженог у слободан и обрнуто. Класа

RecordsManager врши претраживање и добављање библиотечких записа, као и привремено смештање записа у објекте класа пакета *RecordUtils*. Те објекте испоручује класама корисничког интерфејса које их користе за приказ података из записа. Поред тога класа *RecordsManager* врши проверу као и промену статуса примерка у процесу задуживања примерака.

3.2.5 Пакет *Proxy*

На слици 3.6 је приказан дијаграм класа пакета *Proxy*. Овај пакет служи за комуникацију клијентског дела апликације са базом података. Пакет се састоји од једног интерфејса и његове две имплементације.

Интерфејс *Proxy* представља скуп апстрактних операција које су потребне апликацији да би приступила подацима из базе података. То су следеће операције:

- *getUser(userID:String):Users* - за задати број корисника *userID* враћа податке о кориснику са свим задужењима корисника и продужењима чланарине. Ти подаци се смештају у објекте класа из пакета *Model.*

- *saveUser(user:Users):int* - смешта податке о новом или постојећем кориснику у базу података. Подаци се узимају из објеката класа из пакета *Model,* а враћа се број као индикатор успешности извршавања операције. На основу постојања или непостојања вредности атрибута *sys_id* класе *Users* из пакета *Model* се разликује постојећи од новог корисника.

- *getAll(class:Class):List* - за задату класу из пакета *Model* враћа листу свих појава те класе.

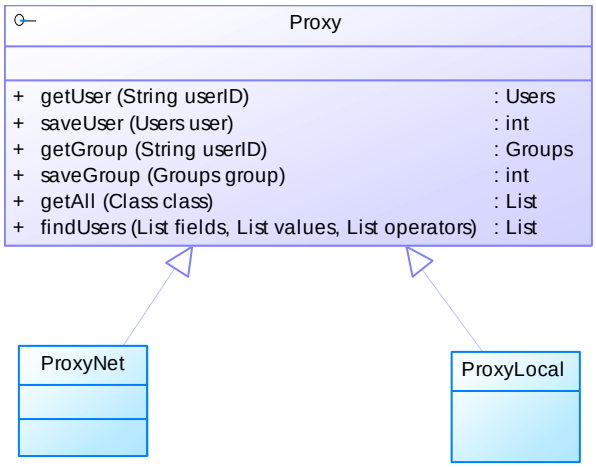
- *getGroup(userID):Group* - за задати број корисника *userID* враћа податке о колективном кориснику. Ти подаци се смештају у објекте класа из пакета *Model.*

- *saveGroup(group:Group):int* - смешта податке о новом или постојећем колективном кориснику у базу података. Подаци се узимају из објеката класа из пакета *Model,* а враћа се број као индикатор успешности извршавања операције. На основу постојања или непостојања вредности атрибута *sys_id* класе *Group* из пакета *Model* се разликује постојећи од новог корисника.

- *findUsers(fields:List, values:List, operators:List):List* - врши претраживање корисника на основу задате листе атрибута корисника, листе вредности тих атрибута и листе оператора *and, or* и *not* који се стављају између свака два операнда типа (атрибут = вредност).

Класа *ProxyLocal* представља имплементацију интерфејса *Proxy* која се користи у случају када се сервер базе налази у локалном интранету. Имплементација користи перманентну конекцију са базом података преко *JDBC* драјвера.

Класа *ProxyNet* представља имплементацију интерфејса *Proxy* која се користи у случају када се сервер базе налази на неком удаљеном серверу коме се приступа путем Интернета. У том случају подаци се шаљу серверској страни апликације представљеном компонентом *Service* на дијаграму компоненти на слици 3.2 преко Интернет протокола, а затим серверска страна успоставља конекцију са базом података. Интерфејс *Proxy* се такође користи за имплементацију серверске стране апликације.

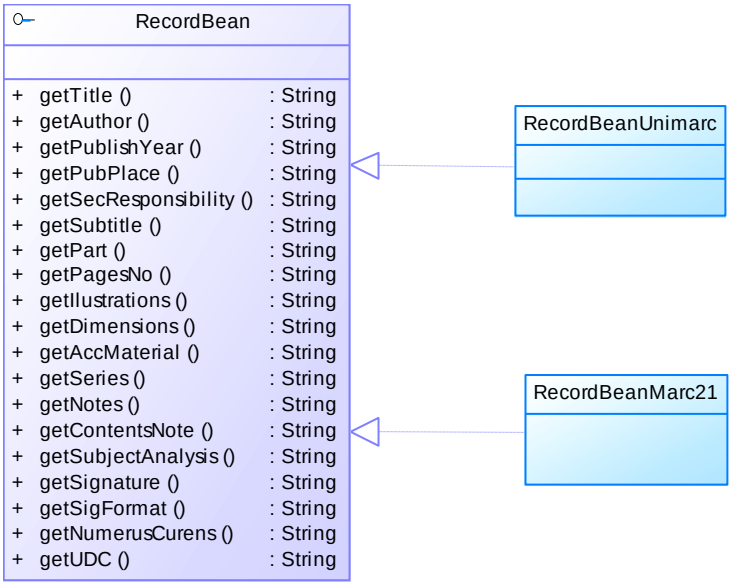


Слика 3.6 Дијаграм класа *Proxy*

3.2.6 Пакет *RecordUtils*

На слици 3.7 је приказан дијаграм класа пакета *RecordUtils.* Овај пакет служи остатку система за добијање одређених садржаја из библиотечког записа који се приказују у систему за коришћење

библиотечке грађе. Пакет се састоји од једног интерфејса и његове две имплементације.



Слика 3.7 *Дијаграм класа пакета RecordUtils*

Интерфејс *RecordBean* представља скуп апстрактних операција које враћају одређене садржаје из библиотечког записа. На основу библиотечких захтева направљен је скуп садржаја из библиотечког записа који су потребни у циркулацији. У зависности од библиографског формата ови садржаји су смештени у различитим пољима и потпољима записа. Класа *RecordBeanUnimarc* је имплементација интерфејса *RecordBean* која се користи у случају када се библиотечки записи налазе у *UNIMARC* формату, а класа *RecordBeanMarc21* у случају када се записи налазе у формату *MARC21*. Моделом су предвиђене имплементације само за ова два формата, али се у случају потребе могу додати имплементације и за неке друге библиографске формате.

3.3 Имплементација система

На основу спецификације описане у претходним одељцима урађена је имплементација система за циркулацију. Систем је имплементиран у *Java* окружењу. Као развојни алат је коришћено окружење *Eclipse v3.2*

[Eclipse]. Систем се састоји из три дела: клијентске апликације, серверског дела апликације и базе података.

Клијентска апликација је имплементирана као класична десктоп апликација чије извршавање се одвија у потпуности на клијенту. Такав начин имплементације је одабран првенствено због функционалности и брзине екранских форми што је био један од основних захтева библиотека. Екранске форме корисничког интерфејса су имплементиране помоћу стандардне *Java* библиотеке *Swing*, а валидација екранских форми је урађена помоћу *Jakarta Commons Validator* [Validator] библиотеке.

Поред екранских форми клијентска апликација садржи и имплементацију објектног модела базе података преко кога остатак апликације приступа подацима из базе података. Део апликације који је задужен за комуникацију између тог објектног модела и базе података је реализован помоћу *ORM* (*Object/Relational Mapping*) технологије, а за имплементацију је коришћен *Hibernate* пакет [Hibernate].

Клијентска апликација има два начина комуникације са базом података чиме је омогућен рад клијента и у случају кад се база података налази у локалном интранету и у случају кад се налази на удаљеном серверу, чиме систем постаје независан од мрежног окружења у ком библиотека ради.

Шема базе података је направљена на основу дијаграма класа *Подаци* приказаног на слици 3.5. Свака класа са дијаграма одговара једној табели шеме базе података. Сва обележја класа су пресликана у одговарајуће колоне табела. На основу веза међу класама су направљени одговарајући референцијални интегритети међу табелама. Као сервер базе података се користи *MySQL Community Server 5.1*.

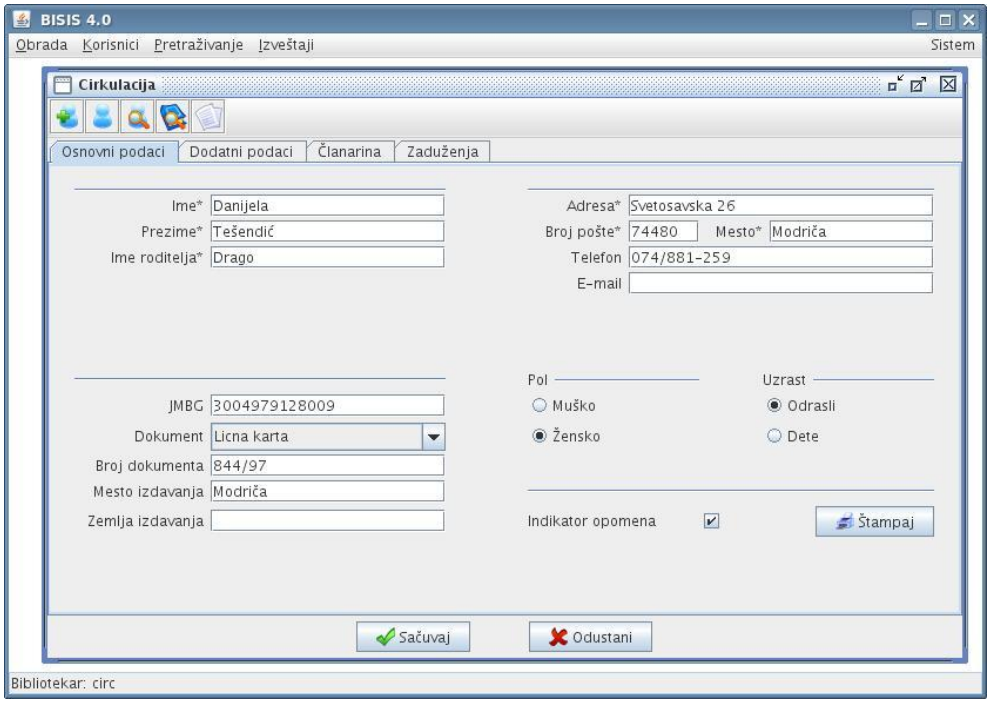
3.3.1 Имплементација корисничког интерфејса

Имплементација корисничког интерфејса је урађена на основу случајева коришћења и дијаграма класа који су наведени у претходним одељцима. Функционалности описане на дијаграмима случајева коришћења су пренесене на екранске форме корисничког интерфејса

преко којих библиотекар комуницира са системом. Имплементација екранских форми је урађена на основу дијаграма класа *Кориснички интерфејс*. Овде су приказане и описане неке од екранских форми на којима су имплементиране функционалности са дијаграма случајева коришћења *Рад са корисницима.* Остале екранске форме су имплементиране на сличан начин.

3.3.1.1 Подаци о кориснику

Приликом покретања апликације отвара се основна екранска форма апликације. Ова екранска форма представља имплементацију класе *OsnovniProzor* са дијаграма класа *Кориснички интерфејс*. Основна екранска форма садржи главни мени апликације и *toolbar* апликације. Све остале форме се отварају унутар основне екранске форме. На слици 3.8 је приказан изглед апликације са отвореном екранском формом за рад са корисницима. Главни мени је подељен на пет делова: *Obrada, Korisnici, Pretraživanje, Izveštaji* и *Sistem*. Мени *Obrada* односи се на обраду библиографске грађе, што није предмет овог рада. Мени *Korisnici* се састоји од два подменија: *Novi* и *Postojeći.* У подменију *Novi* се налазе акције везане за упис новог корисника. Подмени *Novi* има две акције *Individualni* и *Kolektivni.* Акција *Individualni* иницира функцију уписа новог индивидуалног корисника, а акција *Kolektivni* новог колективног корисника. Овај подмени представља реализацију функционалности случаја коришћења *Registering new user* приказаног на дијаграму *Рад са корисницима.* Подмени *Postojeći* садржи акције везане за рад са постојећим корисником у систему. То су: *Podaci, Članarina* и *Zaduženja.* Преко тих акција се иницирају функције промене података корисника, продужења чланарине и задуживања и раздуживања корисника. Овај подмени представља реализацију функционалности случаја коришћења *Existing user*. У менију *Pretraživanje* се налазе две акције: *Korisnici* и *Publikacije.* Акција *Korisnici* иницира функцију претраживања корисника, а акција *Publikacije* претраживање публикација. У менију *Izveštaji* се налазе акције којима се покреће генерисање извештаја који постоје у систему, а у менију *Sistem* акције везане за администрацију система и ажурирање шифарника. На *toolbar*-у апликације се налазе пречице до најчешће коришћених акција из менија.



Слика 3.8 Екранска форма *Osnovni podaci*

Избором акције уписа новог индивидуалног корисника отвара се екранска форма приказана слици 3.8 са празним пољима за унос. Ова екранска форма представља имплементацију класе *IndividualniKorisnik* са дијаграма класа *Кориснички интерфејс* и на њој се реализује функционалност случаја коришћења *Individual user* са дијаграма *Рад са корисницима.* Екранска форма садржи четири картице: *Osnovni podaci, Dodatni podaci, Članarina* и *Zaduženja.* Изглед картице *Osnovni podaci* је приказан на већ поменутој слици 3.8. Картице *Osnovni podaci* и *Dodatni podaci* представљају имплементацију класе *PodaciOKorisniku* са дијаграма класа *Кориснички интерфејс* и на њима је реализована функционалност случаја коришћења *User data.* Садржај картице *Članarina* представља имплементацију класе *Članarina* и реализује функционалност случаја коришћења *Membership data,* а садржај картице *Zaduženja* имплементацију класе *Zaduženja* и функционалност случаја коришћења *Charging and discharging.*

На картицама *Osnovni podaci* и *Dodatni podaci* се уносе подаци о кориснику. На картици *Članarina* се уносе број корисника и остали

подаци о чланству у библиотеку. Пошто се иста екранска форма користи и за новог и за постојећег корисника на овој картици је могуће и продужити чланарину кориснику, као и одштампати признаницу. На картици *Задужења* се врши задуживање и раздуживање корисника. Видљива је табела са задуженим публикацијама. Над сваком публикацијом у табели је могуће урадити две акције: раздуживање и продуживање задужења. Публикације је могуће задужити на два начина: уношењем инвентарног броја публикације или претраживањем библиотечког фонда које води на екранску форму на којој се реализује функционалност случаја коришћења *Searching publications.* Постоји дугме којим се реализује случај коришћења *Printing lending form,* и дугме којим се реализује случај коришћења *Printing history of loan charges.* Поред тога је видљив и кратак инфо о кориснику са упозорењима о истеку чланарине и посланим опоменама.

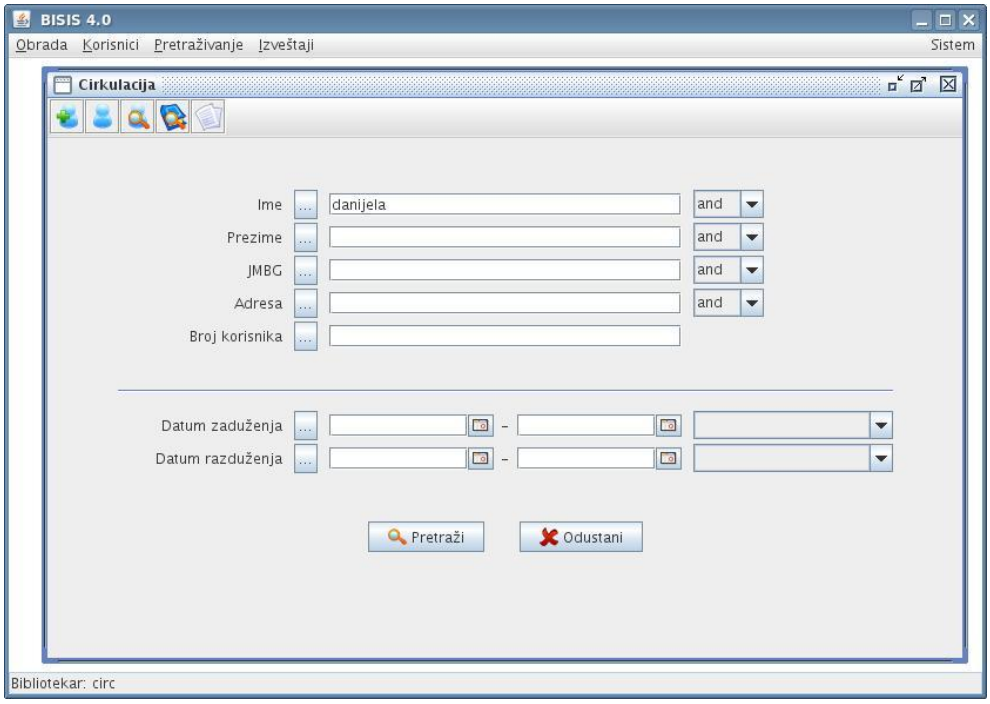
Рад са подацима постојећег индивидуалног корисника је сличан као унос података за новог корисника. Користи се иста екранска форма као за унос новог корисника која је претходно описана. Избором у менију једне од три могуће акције за рад са постојећим корисником (*Podaci, Članarina* и *Zaduženja*) отвара се екранска форма за унос броја корисника. Након уноса броја отвара се екранска форма приказана на слици 3.8 са видљивом картицом *Osnovni podaci, Članarina* или *Zaduženja* у зависности од одабране акције и у случају да унесени број припада индивидуалном кориснику. У случају да број припада колективном кориснику отвара се екранска форма за колективног корисника која представља реализацију случаја коришћења *Corporate user data.* Претходно описане акције представљају реализацију случаја коришћења *Existing user.* У случају да библиотекар не зна број корисника чије податке жели да види са екранске форме за унос броја корисника може отићи на екранску форму за претраживање корисника које је приказано касније у овом одељку.

Ако се у менију изабере акција уписа новог колективног корисника отвара се екранска форма на којој се реализују функционалности случајева коришћења *Corporate user* и *Corporate user data* и која представља имплементацију класе *KolektivniKorisnik*. На форми се уноси скуп података за колективног корисника. Ова екранска форма се користи и за промену података колективног корисника.

3.3.1.2 Претраживање

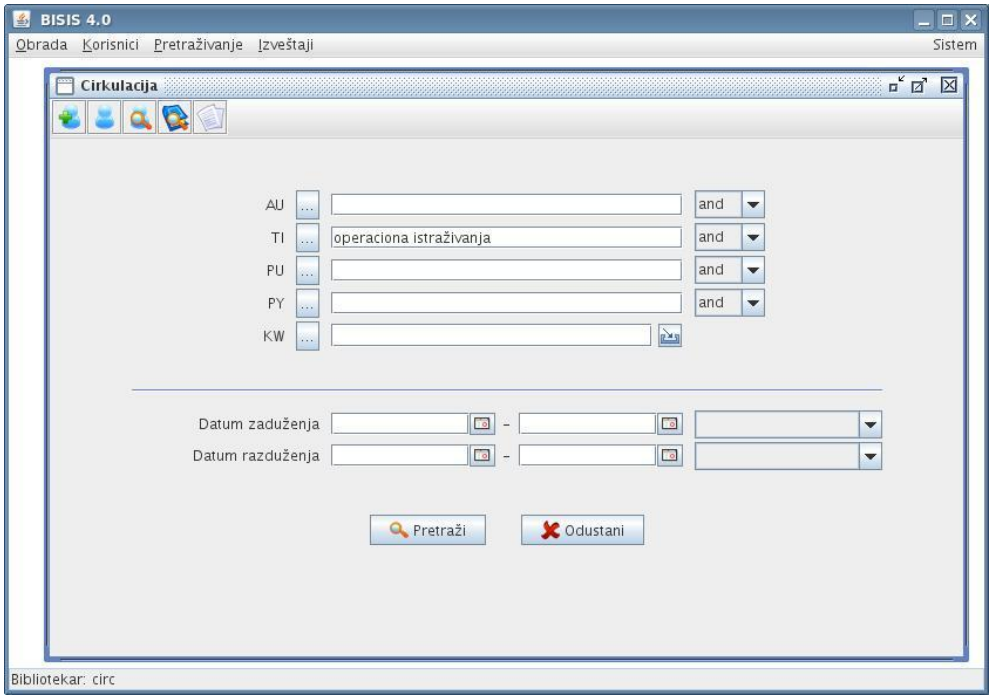
Избором акције претраживања корисника из менија отвара се екранска форма приказана на слици 3.9. Ова екранска форма представља имплементацију класе *PretraživanjeKorisnika* и реализацију функционалности случаја коришћења *Searching users.* На форми се задају критеријуми претраживања. Може се претраживати по свим пољима која су дефинисана у класи *Users* која је приказана на дијаграму *Подаци* у одељку 3.2.3. Критеријуми се могу укрштати између себе са операторима *and, or* и *not.* Може да се зада до пет критеријума типа "поље = вредност поља". Дугме са знаком "..." иницира акцију избора поља по ком се претражује. Назив изабраног поља је видљив са леве стране дугмета. Са десне стране дугмета се налази текстуално поље у које се задаје тражена вредност изабраног поља класе *Users.* Уколико тражено поље узима вредности из неког од шифарника повезаних са класом *Users* уместо текстуалног поља за унос вредности појављује се падајућа листа из које се бира једна од постојећих вредности тог шифарника. Поред тога, на ове критеријуме је могуће додати и временски период за датуме који постоје у класама *Registering* и *Lending* а то су: датум задужења, датум раздужења, рок враћања, датум продужења чланства и датум до кад важи чланство, као и локацију за сваку од тих акција извршених у том временском периоду.

Иницирањем акције претраживања отвара се екранска форма на којој се приказују резултати претраживања корисника. Форма представља имплементацију класе *RezultatiPretraživanjaKorisnika* и реализацију функционалности случаја коришћења *Searching users results display.* Пронађени корисници на основу задатог критеријума приказују се у табели. За одабраног корисника из табеле је могуће отворити екранску форму за рад са подацима корисника која је већ претходно описана и представља реализацију случаја коришћења *Existing user.*



Слика 3.9 Екранска форма за претраживање корисника

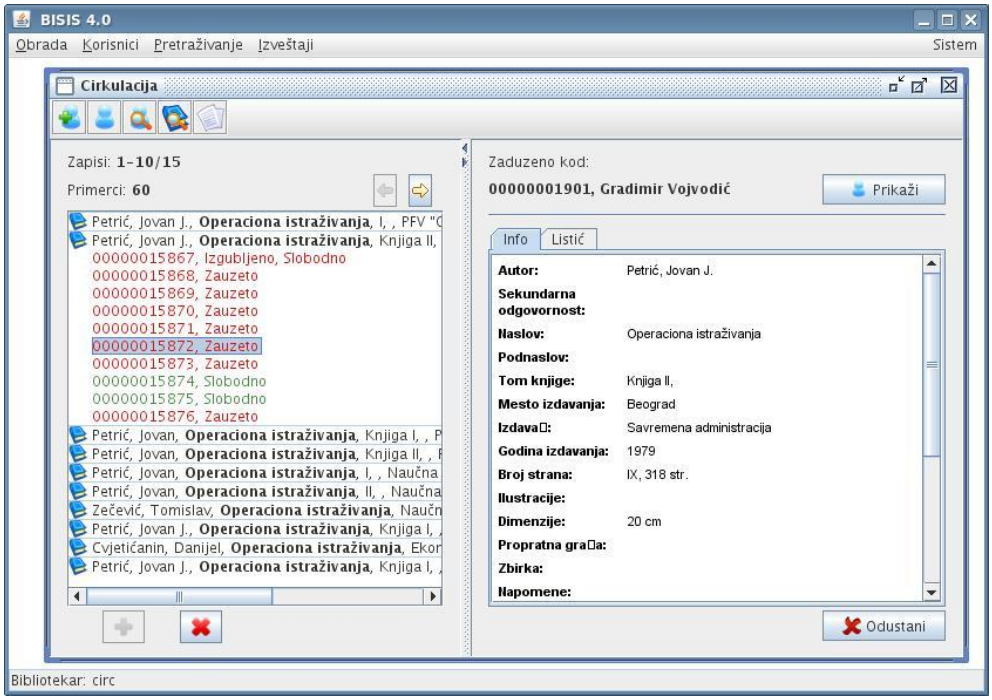
Избором акције претраживања публикација отвара се екранска форма приказана на слици 3.10. Ова екранска форма представља имплементацију класе *PretraživanjePublikacija* и реализацију функционалности случаја коришћења *Searching publications*. На форми се задају критеријуми претраживања. Претраживање се врши по понуђеним префиксима текст сервера БИСИС система и може да се зада до пет критеријума који међу собом могу да се укрсте операторима *and, or* и *not*. Ти префикси се односе на садржај библиотечког записа. Екранска форма за избор префикса се отвара на дугме са знаком "...". Назив префикса је видљив са леве стране дугмета. Са десне стране дугмета се налази текстуално поље у које се задаје тражена вредност изабраног префикса. Уколико префикс означава неки шифриран податак могуће је отворити шифарник из кога се бира шифра. На ове критеријуме је могуће додати и временски период за датум задуживања и датум раздуживања публикације, као и локацију за сваку од тих акција извршених у том временском периоду.



Слика 3.10 Екранска форма за претраживање публикација

Екранска форма за приказ резултата претраживања је приказана на слици 3.11. Екранска форма представља имплементацију класе *RezultatiPretraživanjaPublikacija* и реализацију функционалности случаја коришћења *Searching publications results display*. На левој половини форме се приказује листа резултата. У резултате спадају библиотечки записи који су задовољили критеријуме претраживања. За сваки запис у листи се приказује аутор, наслов, издавач и година издања. Двокликом на сваки елемент листе се отвара подлиста за тај елемент која садржи инвентарне бројеве примерака који припадају том запису и статусе тих примерака. Примерак који је задужен могуће је раздужити и то представља реализацију случаја коришћења *Discharging*. Такође за сваки задужени примерак се у десном горњем углу форме приказује број, име и презиме корисника код кога је задужен. Могуће је отворити екранску форму за рад са подацима тог корисника што представља реализацију случаја коришћења *Existing user*. Примерке који су задуживи је могуће задужити што представља реализацију случаја коришћења *Charging and discharging* којој претходи унос броја корисника који се задужује. На десној страни екранске

форме се приказују детаљи библиотечког записа одабраног у листи погодака. Детаљи се приказују на два начина: у облику кратког инфо-а као сто се види на слици 3.11 или у облику каталошког листића. Жељени начин приказа постаје видљив одабиром одговарајуће картице.



Слика 3.11 Приказ резултата претраживања публикација

3.3.2 Имплементација ажурирања базе података

Део апликације који је задужен за рад са подацима из базе података је имплементиран на основу спецификације пакета *Model*, пакета *Proxy* и класе *UsersManager* која је дата у претходним одељцима. Имплементацију спецификације пакета *Model* и *Proxy* представљају *Java* пакети *Model* и *Proxy* са својим класама, а имплементацију класе *UsersManager Java* класа *UsersManager* која се налази унутар *Java* пакета *Manager*.

3.3.2.1 Имплементација објектног модела базе података

Java класе пакета *Model* представљају имплементацију објектног модела базе података. Имплементација је урађена на основу спецификације класа приказаних на дијаграму класа *Подаци*. Свакој класи са дијаграма одговара једна *Java* класа која има исто име, нпр. класи *Users* са дијаграма одговара *Java* класа *Users.java.*

Сви атрибути специфицирани у класама на дијаграму постоје и у имплементацији. *Java* класе су имплементиране по *JavaBeans* стандарду што значи да за сваки атрибут *xxx* постоје методе *setXxx()* и *getXxx()*. Методе *setXxx()* постављају вредности атрибута и као параметар им се прослеђује та вредност, а *getXxx()* враћају вредности атрибута.

Објекти који су појаве класа, поред вредности атрибута имају и референце на друге објекте с којима су у вези. Тим референцама су имплементиране асоцијације које постоје на дијаграму класа *Подаци.* Тако на пример, асоцијација 1:N између класа *Users* и *Lending* је имплементирана тако да објекат класе *Users* поседује листу референци на објекте класе *Lending* с којима је у вези, а објекти класе *Lending* поседују референцу на одговарајући објекат класе *Users.* Класа *Users* има методе: *getLendings(),* која враћа листу референци на објекте класе *Lending, setLendings(),* којом се поставља вредност за листу референци и *addLending(),* којом се додаје један објекат класе *Lending* у већ постојећу листу. Слично, класа *Lending* има методе: *getUsers()* која враћа референцу на одговарајући објекат класе *Users* и *setUsers()* која поставља вредност референце. На исти начин су имплементиране и остале асоцијације са дијаграма *Подаци.*

3.3.2.2 Имплементација комуникације са базом података

Део апликације који је задужен за комуникацију између претходно описаног објектног модела базе и физичке базе података је реализован помоћу *ORM* (*Object/Relational Mapping*) технологије, а за имплементацију је коришћен *Hibernate* пакет. Овај део апликације представља имплементацију пакета *Proxy* описаног у одељку 3.2.5 и спакован је у *Java* пакет који се такође зове *Proxy.* Класе и интерфејс приказани на дијаграму класа пакета *Proxy* имају своју

имплементацију у облику *Java* класа и интерфејса и то су: интерфејс *Proxy.java,* класа *ProxyLocal.java* и класа *ProxyNet.java*.

Поред тога у *Java* пакету се налазе и фајлови потребни *Hibernate* пакету за чување информација о мапирању објектног модела базе на физичку базу података. Те информације се чувају у *XML* документима. За сваку класу пакета *Model* постоји један *XML* документ са именом *ImeKlase.hbm.xml* у ком се налазе информације о мапирању те класе на одговарајућу табелу базе података.

Класа *ProxyLocal.java* представља имплементацију интерфејса *Proxy.java* која се користи за приступ бази података у случају када се база налази у локалном интранету. Класа имплементира све методе интерфејса које су дефинисане и описане у одељку 3.2.5, као и још неке помоћне за којима се јавила потреба приликом имплементације. Имплементација за комуникацију са базом користи *JDBC* драјвер, што значи да постоји перманентна конекција са базом података. Из тог разлога ова имплементација се користи за локални интранет.

На листингу 3.1 је дат пример имплементације методе *getUser* класе *ProxyLocal.java.* Овај пример илуструје употребу *Hibernate* пакета за имплементацију метода класе *ProxyLocal.java.* На сличан начин су имплементиране и остале методе ове класе.

```
...
private SessionFactory sessions;
...
static{
   sessions = new Configuration().buildSessionFactory();
}
...
public Users getUser(String userID){
   Session session = sessions.openSession();
   Transaction tx = session.beginTransaction();
   Query q = session.createQuery("from Users u where
        u.userId = :userid");
   q.setString("userid", userID);
   Users user = (Users)q.uniqueResult();
   tx.commit();
   session.close();
   return user;
}
```

Листинг 3.1 Имплементације методе *getUser*

Приликом иницијализације класе *ProxyLocal.java* креира се објекат типа `SessionFactory` који управља конекцијама са базом података (приказано на листингу 3.1). Објекат се креира на основу конфигурационог фајла. У случају овог локалног интерфејса у конфигурационом фајлу су наведена подешавања за креирање конекције са базом података преко *JDBC* драјвера.

У случају метода којима се добављају подаци из базе података, какав је приказани метод *getUser*, као резултат се добијају објекти одговарајућих класа пакета *Model.* Такође, приликом ажурирања базе података методама се прослеђују објекти класа пакета *Model*. *Hibernate*, на основу информација о мапирању објектног модела на базу података, које је претходно описано, зна над којим табелама да извршава упите и који атрибути објеката одговарају којим пољима табела базе података.

Класа *ProxyNet.java* представља имплементацију интерфејса *Proxy.java* која се користи за приступ бази података у случају када се база налази на удаљеном серверу. Класа имплементира све методе интерфејса које су дефинисане и описане у одељку 3.2.5. Ова имплементација комуникацију са базом података остварује преко серверског дела апликације који се налази на истом или неком другом удаљеном серверу. Серверски део апликације је имплементиран као *XML web* сервис да би се комуникација између сервера и клијента одвијала преко *HTTP* протокола.

Операције које *web* сервис нуди су еквивалентне методама које поседује интерфејс *Proxy.java.* Класа *ProxyNet.java* представља *proxy* објекат тог *web* сервиса. Методе класе *ProxyNet.java* позивају одговарајуће операције *web* сервиса и прослеђују им објекте које су добиле, а даље операције *web* сервиса извршавају функционалност метода која је дефинисана интерфејсом *Proxy.java.* Овај *web* сервис није јаван и служи искључиво за употребу од стране клијентске апликације система за коришћење библиотечке грађе.

Серверска страна апликације садржи имплементацију *web* сервиса и објектни модел базе података имплементиран већ описаним пакетом *Model.* За имплементацију операција *web* сервиса је искоришћен *Hibernate* пакет и имплементација ових операција је иста или врло слична имплементацији метода класе *ProxyLocal.java* које су описане у

претходном одељку. Разлике у односу на локални интерфејс постоје највећим делом у начину на који *Hibernate* приступа бази података. У случају локалног интерфејса *Hibernate* је подешен да бази приступа преко *JDBC* драјвера. Тај начин приступа је изабран због једноставности одржавања система јер у том случају систему на серверској страни не треба ништа друго осим базе података. У случају интерфејса за удаљени приступ већ само постојање *web* сервиса условљава постојање серверске стране апликације која се извршава на неком *web* серверу. Из тог разлога је изабрано да се за комуникацију са базом података искористе погодности које пружа *web* сервер уместо *JDBC* драјвера, па је у овом случају *Hibernate* подешен да користи *JTA* приступ бази података кроз *web* сервер уместо *JDBC* драјвера. Због транспарентности *Hibernate* интерфејса задуженог за трансакције са базом података и у случају када се користи *JDBC* и у случају *JTA* приступа измене у коду које користе тај интерфејс су незнатне. Због тога је имплементација операција *web* сервиса врло слична или иста имплементацији метода класе *ProxyLocal.java.*

3.3.2.3 Имплементација менаџера за комуникацију са базом података

Класа *UsersManager.java* која се налази унутар *Java* пакета *Manager* имплементира функционалности које управљају током података о корисницима од корисничког интерфејса до базе података и обрнуто. Ова класа својим функционалностима остварује везу између пакета *View, Model* и *Proxy.*

Те функционалности обухватају:

- добављање података из базе података преко класа пакета *Proxy* који се привремено смештају у објекте класа из пакета *Model,*
- чување података из објектног модела у базу података преко класа пакета *Proxy,*
- пребацивање података из објектног модела на кориснички интерфејс и обрнуто.

Извршавање метода ове класе се иницира из корисничког интерфејса на акције корисника. Укупно је имплементирано 17 метода. Неке од њих су:

- *loadUser(User userForm, Users userModel)* – из објектног модела *userModel* пребацује податке о кориснику на екранску форму за рад са корисницима коју представља параметар *userForm.*

- *loadGroup(Group groupForm, Groups groupModel)* – из објектног модела *groupModel* пребацује податке о колективном кориснику на екранску форму за рад са колективним корисницима коју представља параметар *groupForm.*

- *getUser(User userForm, Group groupForm, String userID)* – за задати број корисника *userID* преко класа пакета *Proxy* добавља податке о кориснику из базе података који су смештени у класама пакета *Model,* а затим позивањем једне од две претходно описане две методе (у зависности од тога да ли је корисник индивидуални или колективни) пребацује податке из објектног модела на одговарајућу екранску форму *userForm* или *groupForm.*

3.3.3 Имплементација рада са библиотечким записима

Део апликације који је задужен за рад са библиотечким записима је имплементиран на основу спецификације пакета *RecordUtils* и класе *RecordsManager* детаљно описаним у претходним одељцима. Имплементацију спецификације пакета *RecordUtils* представљају *Java* пакет *RecordUtils* са својим класама, а имплементацију класе *RecordsManager Java* класа *RecordsManager.java* која се налази унутар *Java* пакета *Manager.*

3.3.3.1 Имплементација интерфејса библиотечког записа

Систем за коришћење библиотечке грађе садржаје из библиотечких записа добија преко имплементације класа пакета *RecordUtils* чија је спецификација описана у одељку 3.2.6. Тај пакет садржи један интерфејс и две класе који имају своју имплементацију у облику *Java* класа и интерфејса и то су: интерфејс *RecrodBean.java,* класа *RecordBeanUnimarc.java* и класа *RecordBeanMarc21.java*. Ове класе и интерфејс имплементирају све методе описане у одељку 3.2.6. Класа *RecordBeanUnimarc.java* је имплементација интерфејса *RecordBean.java* која се користи у случају када се библиотечки записи налазе у

UNIMARC формату, а класа *RecordBeanMarc21* у случају када се записи налазе у формату *MARC21*. Оваквим начином имплементације систем за коришћење библиотечке грађе постаје независан од *MARC* формата у коме се налазе библиотечки записи јер је интерфејс преко кога се приступа садржајима из записа увек исти.

Као формат за обраду библиотечких записа у оквиру система БИСИС користи се варијанта *UNIMARC* формата названа *YUMARC* формат. Због тога је направљена имплементација интерфејса *RecordBeanYumarc.java* која се користи за рад са записима у *YUMARC* формату какав је случај у БИСИС систему. И ова класа имплементира све методе интерфејса *RecrodBean.java.* Конструктору класе се прослеђује објектни модел записа система БИСИС. Објектни модел записа је имплементиран пакетом *Record* који се налази у оквиру БИСИС система, а детаљно је описан у монографији (Димић и Сурла, 2007).

На листингу 3.2 је дат пример имплементације методе *getPublishYear* класе *RecordBeanYumarc.java* као илустрација начина имплементирања метода ове класе. Пре саме имплементације методе је приказан конструктор класе коме се прослеђује објектни модел записа *Record*. Метода *getPublishYear* по спецификацији из одељка 3.2.6 враћа годину издавања публикације. Метода је имплементирана тако да из записа враћа вредност потпоља *100c* у које се по *YUMARC* формату смешта вредност за годину издавања публикације. Остале методе ове класе на сличан начин враћају одговарајуће вредности из библиотечког записа. Такође, на сличан начин су имплементиране и класе *RecordBeanUnimarc* и *RecordBeanMarc21* у складу са форматом записа.

```
...
private Record rec;
...
public RecordBeanUnimarc(Record rec){
    this.rec = rec;
}
...
public String getPublishYear(){
    return rec.getSubfieldContent("100c");
}
```

Листинг 3.2 Имплементација методе *getPublishYear*

3.3.3.2 Имплементација менаџера записа

Класа *RecordsManager.java* имплементира функционалности везане за комуникацију између система за коришћење библиотечке грађе и библиотечких записа. У оквиру БИСИС система записима се приступа преко текст сервера имплементираног унутар система. Класа *RecordsManager.java* је задужена за комуникацију са текст сервером.

Прва функционалност за коју је класа задужена је промена статуса појединачног примерка. Приликом задуживања примерка потребно је проверити да ли је примерак у неком од задуживих статуса да би се дозволило задуживање тог примерка. Такође при задуживању је потребно променити статус примерка у задужен, као што је потребно при раздуживању статус променити у слободан. Ова функционалност је реализована кроз следеће методе класе *RecordsManager.java*:

- *lendBook(String ctlgno)* – проверава статус примерка са инвентарним бројем *ctlgno* и ако је примерак задужив мења га у заузет.

- *returnBook(String ctlgno)* – статус примерка са инвентарним бројем *ctlgno* мења у слободан.

Друга функционалност за коју је класа задужена је претраживање и добављање библиотечких записа. Претраживање се врши креирањем упита на језику текст сервера и прослеђивањем тих упита текст серверу. Текст сервер као поготке враћа записе у објектном моделу записа имплементираним пакетом *Record* који је претходно описан. Класа има задатак да те записе остатку система за коришћење библиотечке грађе испоручи у облику објеката класа пакета *RecordUtils* чија је имплементација такође претходно описана. Систем даље те објекте користи за приказ података из библиотечких записа. Методе класе *RecordsManager.java* којима се комуницира са текст сервером су следеће:

- *getRecord(String ctlgno)* – добавља запис који садржи примерак са задатим инвентарним бројем *ctlgno*.

- *getRecords(Query q)* – добавља записе који задовољавају упит *q*; упит се креира помоћном методом *makeLuceneAPIQuery()* која од одговарајућих параметара креира упит на језику текст сервера.

На листингу 3.3 је дата имплементација методе *getRecord* као илустрација коришћења текст сервера из система за коришћење библиотечке грађе. Текст серверу се приступа позивањем методе *getRecordManager()* класе *BisisApp*. Прво се текст серверу прослеђује упит методом *select2* која враћа листу бројева записа који задовољавају упит, а затим се методом *getRecord* којој се прослеђује број записа добија запис у објектном моделу. Од тог објектног модела се креира објекат класе *RecordBeanYumarc.java* преко кога остатак система за коришћење библиотечке грађе добија садржаје из записа. Упит на језику текст сервера се креира позивом методе *makeQueryTerm* помоћне класе *QueryUtils.* На сличан начин су имплементиране и остале наведене методе класе *RecordsManager.java* које користе текст сервер.

```
public RecordBean getRecord(String ctlgno){
  Record record = null;
  int[] hits =BisisApp.getRecordManager()
         .select2(QueryUtils.makeQueryTerm
                       ("IN", ctlgno, "", null), null);
  if (hits!=null && hits.length != 0){
    record = BisisApp.getRecordManager()
          .getRecord(hits[0]);
  }
  return new RecordBeanYumarc(record);
}
```

Листинг 3.3 Имплементација методе *getRecord*

Поглавље 4

Подсистем за клијент/сервер комуникацију

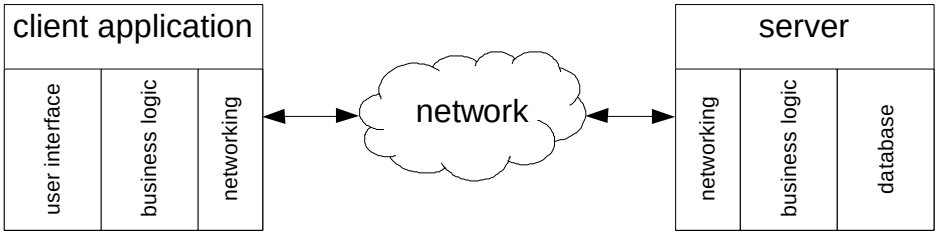
У претходном поглављу дат је опис моделирања и имплементације система за циркулацију библиотечке грађе који је интегрисан у библиотечки систем БИСИС верзије 4. Систем је независна софтверска компонента са могућношћу интегрисања у различите библиотечке системе, јер има подршку за комуникацију са различитим репозиторијумима библиографских записа, као и подршку за различите *MARC* формате. Клијентска апликација је реализована као десктоп апликација али тако да може да се извршава на различитим оперативним системима. Комуникација клијентске апликације са базом података у систему је реализована преко интерфејса. Тај интерфејс има две имплементације. Једна имплементација се користи у случају кад се база података налази у локалној рачунарској мрежи, односно интранету, а друга у случају кад се налази на удаљеном серверу на Интернету. На овај начин је омогућен рад клијентске апликације са физички удаљених локација библиотеке. Систем такође поседује и интерфејс преко кога добија садржаје из библиотечког записа. Тај интерфејс има две имплементације. Једна се користи у случају кад су записи смештени у *UNIMARC* формату, а друга када се налазе у формату *MARC21.* Коришћењем овог интерфејса систем постаје независан од формата у који се смештају записи у оквиру система БИСИС (или других библиотечких система). Систем подржава све стандардне активности за рад са корисницима, а то су: евиденцију, задуживање и раздуживање корисника; претраживање корисника и публикација; генерисање различитих врста извештаја, као и опомена корисницима.

Решење интерфејса за комуникацију са базом података које је приказано у претходном поглављу у одељку 3.2.5 је везано искључиво за систем у оквиру кога је имплементирано, односно за систем

циркулације. Методе интерфејса представљају функционалности циркулације и свака нова функционалност захтева проширење интерфејса и имплементацију нових метода, што је недостатак тог решења. Из тог разлога је направљено другачије решење.

У овом поглављу је приказано ново решење проблема комуникације клијентске апликације са базом података. Направљен је подсистем за комуникацију клијентске и серверске стране апликације кроз који је омогућена и комуникација са базом података која се налази на серверској страни. У раду (Milosavljević and Tešendić, 2010) описана је првобитна верзија овог подсистема која је била интегрални део система за циркулацију. Даљим развојем подсистем је постао независна компонента и крајњи резултат његовог развоја дат је у овом поглављу.

Једна од карактеристика овог подсистема је да он не зависи од функционалности саме апликације. Клијент се, према томе, састоји из три основне компоненте: корисничког интерфејса, имплементације пословне логике и подсистема за клијент/сервер комуникацију. Подсистем за комуникацију је у том случају независна софтверска компонента која може да се интегрише и у друге системе, а не само у систем циркулације. Поред тога што подистем може да се користи за комуникацију са базом података на серверској страни као у случају система за циркулацију, он може да се користи за различите потребе комуникације са сервером јер не зависи од пословне логике система који га користи.



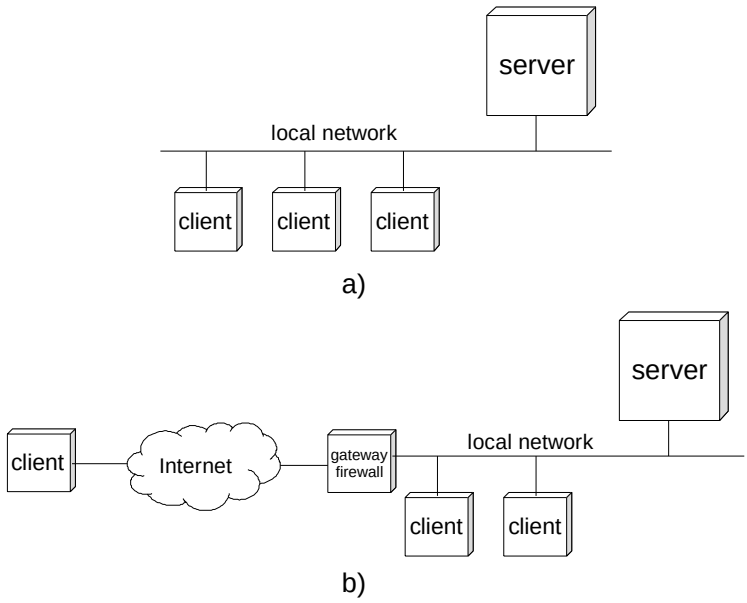
Слика 4.1 Софтверска архитектура *network-agnostic* клијента

Поред тога, подсистем за комуникацију представља апстракцију са више начина за комуникацију са серверском страном, чиме се омогућава рад система у различитим мрежним окружењима (локална

мрежа, удаљени клијенти повезани путем Интернета, па чак и мобилни клијенти). Слика 4.1 илуструје концепт софтверске архитектуре са *network-agnostic* клијентом. Оваква софтверска архитектура клијента где подсистем за комуникацију не зависи од пословне логике система омогућава развој пословне логике (нових функција система) који не утиче на подсистем за комуникацију. С обзиром да подсистем омогућава комуникацију у различитим мрежним окружењима исти програмски код целокупне клијент апликације, без измена, може се користити за клијенте који серверу приступају у локалној мрежи (у двослојној или трослојној софтверској архитектури), као и за удаљене клијенте који имају ограничене могућности повезивања са сервером, на пример путем изнајмљених линија или јавног Интернета. Начин приступа серверу подешава се у конфигурацији клијентске апликације. Програмски код који имплементира пословну логику користи се у неизмењеном облику без обзира на мрежну конфигурацију система. Слика 4.2 илуструје два различита *deployment* сценарија истог клијента – рад у двослојној архитектури која подразумева директан приступ серверу (у локалној мрежи) и приступ са удаљене мреже путем технологије *web* сервиса.

У одељку 4.1 дата је софтверска архитектура и имплементација подсистема за клијент/сервер комуникацију. Софтверска архитектура је заснована на комбинацији неколико дизајн патерна. Моделирање је урађено у обједињеном језику моделирања *UML* 2.0 и модел је представљен са једним дијаграмом класа и два дијаграма секвеци. Имплементација је урађена у програмском језику *Java*, а при имплементацији су коришћени програмски пакети *Hibernate* (за објектно – релационо мапирање) и *Hessian* (за комуникацију дистрибуираних компоненти).

У одељку 4.2 приказана је интеграција подсистема за комуникацију у систем за циркулацију, као и цео систем БИСИС. Приказане су измене на моделу система за циркулацију у односу на модел који је дат у претходном поглављу.



Слика 4.2 Различити *deployment* сценарији:
а) локална мрежа и двослојна клијент-сервер архитектура,
б) повезивање удаљених клијената путем јавног Интернета

4.1 Подсистем за клијет/сервер комуникацију

Подсистем за клијент/сервер комуникацију има патерн оријентисану архитектуру засновану на комбинацији неколико дизајн патерна. Дизајн патерни представљају опис различитих објектно оријентисаних дизајнерских решења и архитектура којима се решавају специфични дизајнерски проблеми. Применом дизајн патерна у софтверским решењима добијају се елегантнија решења лакша за одржавање и даљу надоградњу са могућношћу интеграције у различите софтверске системе.

У великом броју софтверских решења која укључују комуникацију између објеката на различитим локацијама за архитектуру тог дела изабран је *command* патерн. У уводном поглављу у одељку 1.5 је дат приказ неколико таквих софтверских решења. У приказаним решењима могу да се уоче неке од погодности *command* патерна. У неким решењима је постојала потреба да се неке акције изврше на другом месту од места на ком су инициране. На пример, извршавање акције је било потребно пребацити са клијента на сервер да би се

растеретио клијент или акције извршавати над подацима које користе и други клијенти. У овом случају *command* патерн је искоришћен да би се акције енкапсулирале у објекте и као објекти слале на удаљене локације, а затим се извршавале на тим удаљеним локацијама. У другом случају *command* патерн је искоришћен због погодности лаког додавања нових акција без потребе за изменама на постојећој имплементацији. Због наведених погодности *command* патерн је искоришћен и у софтверској архитектури подсистема за комуникацију описаног у овом поглављу.

Такође, у софтверским решењима приказаним у уводном поглављу у одељку 1.5 да би се постигла модуларност система коришћени су *factory* патерни. *Factory* патерни се користе у случајевима када је потребно из скупа више класа зависно од параметара креирати инстанцу одговарајуће класе. У једном од поменутих решења *factory* патерни су искоришћени за моделирање комуникације између компоненти чиме је омогућена употреба различитих транспортних протокола на средњем слоју. На сличан начин *factory* патерни су употребљени и у софтверској архитектури подсистема приказаног у овом поглављу за моделирање транспарентне комуникације између клијента и сервера.

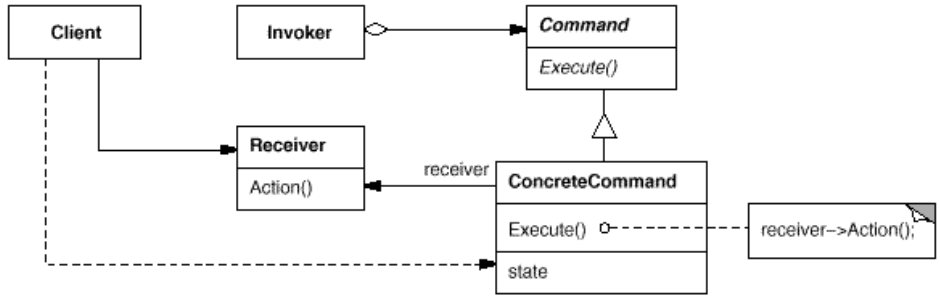
4.1.1 *Command* патерн

Command патерн је дизајн патерн код кога се објекти користе за представљање акција. Основна идеја овог дизајна је да се акција и њени параметри енкапсулирају у објекат. Такви објекти могу да се чувају или шаљу као било који други објекти. На дијаграму класа на слици 4.3 је приказана структура *command* патерна (дијаграм преузет из (Gamma et al., 1994)).

Основни концепт овог патерна је интерфејс *Command*. Интерфејс *Command* представља интерфејс за класе којима су представљене акције и има један апстрактан метод *Execute()*. Конкретна класа овог интерфејса, на дијаграму приказана класом *ConcreteCommand*, имплементира операције потребне за извршавање акције. Објекти конкретних класа се називају командама. Команде унутар себе енкапсулирају и апликативну логику, и улазне и излазне параметре. Команде се инстанцирају од стране клијента, на дијаграму приказано

класом *Invoker*, и прослеђе се класи *Receiver* која не мора да буде део клијента. Та класа иницира извршавање акције команде, тј метода *Execute()*. По завршетку операције класа *Receiver* враћа објекат клијенту, одакле клијент добија резултате извршене акције. Имплементацијом овог патерна могу да се у потпуности раздвоје објекти који иницирају акцију од објеката који извршавају акцију.

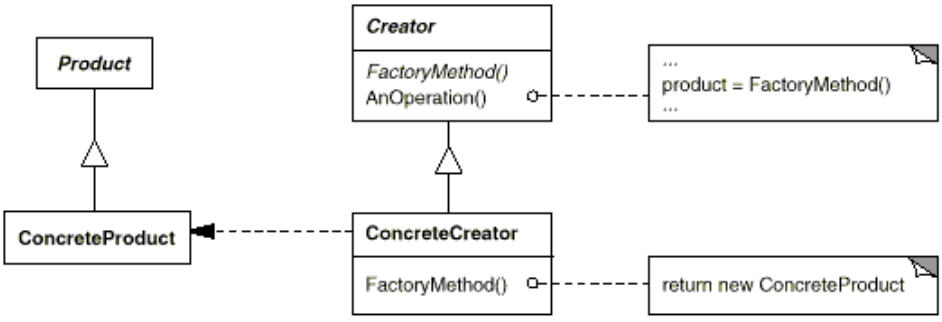
Command патерн је у случају подсистема за клијент/сервер комуникацију искоришћен за имплементацију акција које представљају пословну логику (нпр. регистрација члана, задуживање књиге).



Слика 4.3 Дијаграм класа *command* патерна

4.1.2 *Factory* патерни

Factory method и *abstract factory* патерни спадају у групу *creational* патерна. *Creational* патернима се постиже апстракција процеса инстанцирања, односно омогућава се независност система од начина на који се објекти креирају и представљају.

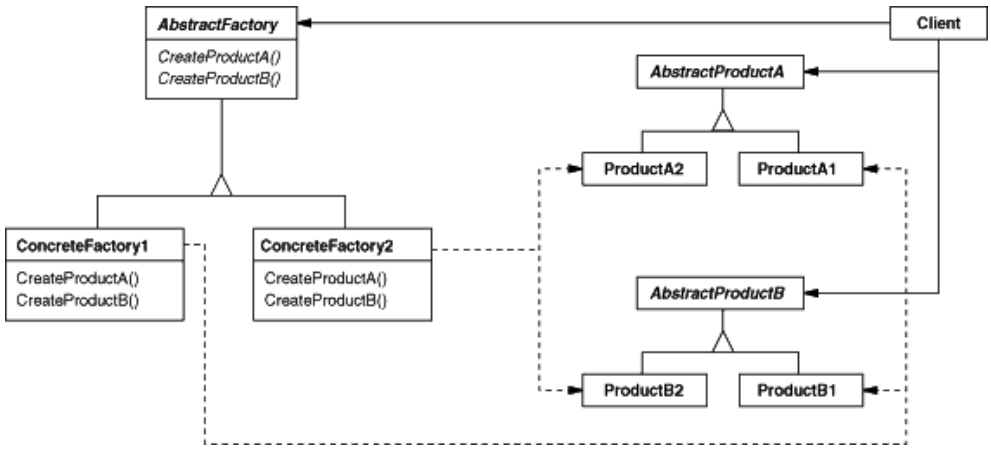


Слика 4.4 Дијаграм класа *factory method* патерна

Factory method патерн дефинише механизам добијања инстанце одговарајуће класе зависно од параметара који су му прослеђени. Обично се користе апстрактне класе да се дефинишу везе међу објектима. Класе чије инстанце се враћају су подкласе тих апстрактних класа и имају исте методе али различите имплементације тих метода. На дијаграму класа на слици 4.4 је приказана структура *factory method* патерна (дијаграм преузет из (Gamma et al., 1994)).

Product представља интерфејс објекта кога је потребно креирати у систему. *ConcreteProduct* је једна конкретна имплементација тог интерфејса. Објекти се у систему креирају преко интерфејса *Creator*. *Creator* има метод *FactoryMethod()* који креира и враћа објекат типа *Product*. *ConcreteCreator* представља имплементацију интерфејса *Creator* која креира и враћа објекат типа *ConcreteProduct*.

Abstract factory патерн је на једном нивоу апстракције више од *factory method* патерна. Он враћа инстанцу одговарајуће класе која је и сама *factory*. Користи се у случају када је потребно креирати фамилију објеката без специфицирања конкретних класа тих објеката. На дијаграм класа на слици 4.5 приказана је структура *abstract factory* патерна (дијаграм преузет из (Gamma et al., 1994)).



Слика 4.5 Дијаграм класа *abstract factory* патерна

Интерфејс *AbstractFactory* дефинише операције за креирање објеката који су дефинисани интерфејсима *AbstractProductA* и *AbstractProductB*. Клијент користи ове интерфејсе, а не конкретне имплементације. Ови

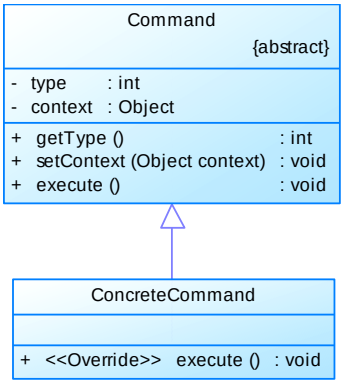
интерфејси представљају дефиницију за конкретне фамилије класа. Класа *ConcreteFactory1* представља *factory* класу која служи за креирање објеката класа *ProductA1* и *ProductB1*, а класа *ConcreteFactory2* за креирање објеката класа *ProductA2* и *ProductB2*.

У случају подсистема за клијент/сервер комуникацију ови патерни се користе за инстанцирање одговарајућег начина комуникације зависно од конфигурације клијента. На овај начин клијентска апликација не мора да зна на који начин се одвија комуникација.

4.1.3 Софтверска архитектура подсистема

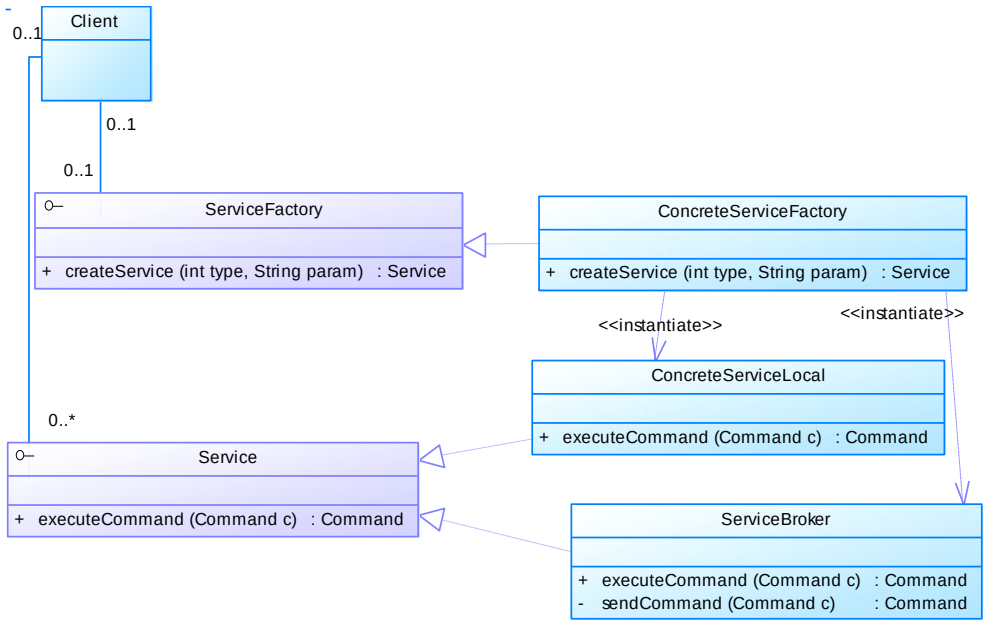
На сликама 4.6, 4.7 и 4.8 приказани су дијаграми класа подсистема за клијент/сервер комуникацију. На дијаграмима је приказана архитектура и клијентског и серверског дела подсистема. Архитектура се заснива на комбинацији *command* патерна, *apstract factory* патерна и *factory method* патерна.

Апстрактна класа *Command* (слика 4.6) представља апстракцију објеката којима се имплементирају различите акције (као код *command* патерна). Класа има само један апстрактан метод *execute()*. Свака конкретна класа која наслеђује класу *Command* има различиту имплементацију метода *execute()*. Метод *execute()* треба да имплементира акцију коју та класа представља. Конкретне имплементације класе које су на дијаграму приказане класом *ConcreteCommand* се називају командама.



Слика 4.6 Интерфејс за имплементацију акција

У зависности од акције која је имплементирана, команда може да захтева неко окружење или контекст за своје извршавање. На пример, ако акција укључује комуникацију са базом података, команди је потребно обезбедити конекцију са базом података у окружењу у ком се извршава. На основу параметра *type*, односно метода *getType()*, онај ко извршава команду добија информацију шта је потребно команди обезбедити за извршавање, а методом *setContext()* команди се прослеђује параметар потребан за извршавање.



Слика 4.7 Дијаграм класа клијентске стране подсистема за клијент/сервер комуникацију

На слици 4.7 приказан је дијаграм класа клијентске стране подсистема за комуникацију. У архитектури *command* патерна приказаној у одељку 4.1.1 класа која извршава команде је представљена *Receiver* класом. У овом случају ту улогу имају интерфејс *Service* и његове имплементације. Интерфејс *Service* служи за извршавање команди и има један апстрактан метод *executeCommand()* коме се као параметар прослеђује конкретна команда. Различите имплементације овог интерфејса омогућавају рад система у различитим мрежним окружењима. Оваквом архитектуром клијентска апликација која користи подсистем постаје независна од конфигурације мреже и не

захтева измене у случају различитих начина комуникације са сервером.

На дијаграму су приказане две имплементације интерфејса *Service*. Имплементација *ConcreteServiceLocal* се користи у случају када се команда извршава у локалном окружењу, а имплементација *ServiceBroker* у случају када се команда извршава на неком удаљеном серверу на Интернету. У случају извршавања команде на удаљеном серверу команду је потребно пребацити са клијента на сервер. Позивањем метода *executeCommand* класе *ServiceBroker* иницира се извршавање метода *sendCommand* који команду шаље на одговарајући сервер. Сценарији употребе описане две имплементације интерфејса *Service* приказани су на дијаграмима секвенци на сликама 4.9 и 4.10 и биће описани даље у тексту.

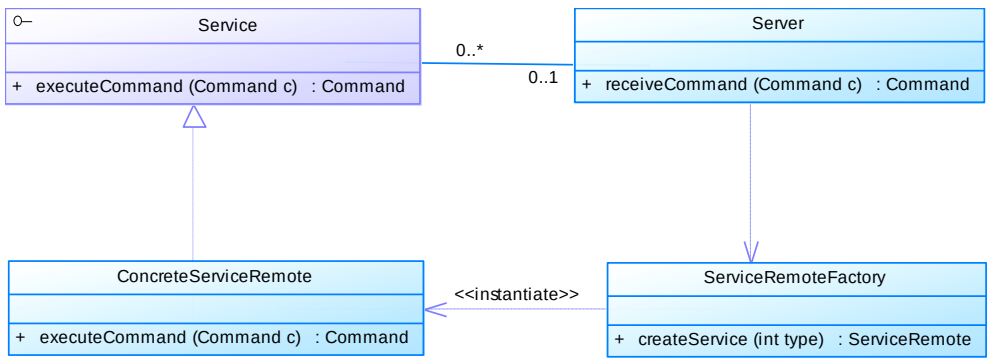
Креирање одговарајуће инстанце једне од имплементација интерфејса *Service* дефинисано је интерфејсом *ServiceFactory* по *factory method* патерну. Овај интерфејс има један метод *createService* који у зависности од параметара који му се проследе враћа инстанцу одговарајуће имплементације интерфејса *Service* (приказано на дијаграмима на сликама 4.9 и 4.10).

Различите имплементације интерфејса *Service* омогућавају рад система у различитим мрежним окружењима. Поред тога, као што је већ речено, у зависности од акције која је имплементирана, команди је потребно обезбедити окружење за извршавање (на пример, ако акција укључује комуникацију са базом података, команди је потребно обезбедити конекцију са базом података). У зависности од потребног окружења, по *abstract factory* патерну, постоје различите фамилије класа које су имплементације описаних интерфејса. Тако да за свако потребно окружење, постоји имплементација интерфејса *ServiceFactory* која служи за креирање одговарајуће инстанце интерфејса *Service* која обезбеђује потребно окружење за извршавање команде. На дијаграму на слици 4.7 је приказана само једна таква фамилија класа и њу чине класе *ConcreteServiceFactory*, *ConcreteServiceLocal* и *ServiceBroker*. У зависности од имплементације команде и тога шта јој је потребно за извршавање, клијентска апликација бира једну од имплементација *ConcreteServiceFactory* класе која креира одговарајућу инстанцу интерфејса *Service.*

Окружење за извршавање команде потребно је обезбедити само у тренутку извршавања команде. Из тога следи да имплементација интерфејса *Service* која шаље команду на удаљени сервер (класа *ServiceBroker*) нема потребу да обезбеди окружење за извршавање команде на клијенту. Због тога се та иста класа може користити за било коју фамилију класа која се креира у зависности од окружења, а коју чине претходно описане класе *ConcreteServiceFactoy*, *ConcreteServiceLocal* и *ServiceBroker*. За команду која се шаље на удаљени сервер окружење за извршавање се обезбеђује на серверској страни.

На слици 4.8 приказан је дијаграм класа серверске стране подсистема за комуникацију. Класа *Server* је класа која добија послату команду од класе *ServiceBroker* на клијентској страни. Класа *Server* добијену команду прослеђује интерфејсу *Service* који као и на клијентској страни служи за извршавање команди. Пошто је добијеној команди потребно обезбедити окружење за извршавање, као и на клијентској страни, постоје различите имплементације интерфејса *Service* које су на дијаграму представљене класом *ConcreteServiceRemote*.

Класа *Server* одговајућу инстанцу интерфејса *Service* креира преко класе *ServiceRemoteFactory* по *factory method* патерну. Класа *ServiceRemoteFactory* има један метод *createService* који у зависности од параметра који му се проследи враћа инстанцу одговарајуће имплементације интерфејса *Service*. Параметар који је потребно проследити налази се у добијеној команди у пољу *type*.

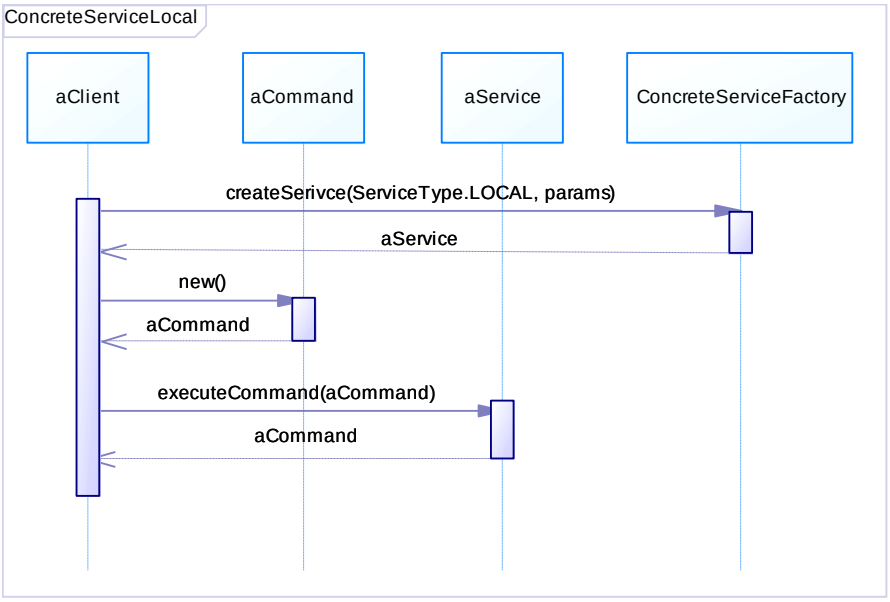


Слика 4.8 Дијаграм класа серверске стране
подсистема за комуникацију

На дијаграму секвенци приказаном на слици 4.9 је дат сценарио извршавања команде у случају када се користи имплементација *ConcreteServiceLocal* интерфејса *Service*. Имплементација *ConcreteServiceLocal* се користи у случају када се команда извршава у локалном окружењу и циљ дијаграма је да прикаже тај поступак. Класа *ConcreteServiceLocal* представља било коју имплементацију интерфејса *Service* која се користи за локално извршавање команде без обзира на окружење које је команди потребно обезбедити за извршавање.

Објекти који учествују у интеракцији су:

- *aClient* – објекат клијентске апликације која користи подсистем и иницира извршавање команде,
- *aCommand* – команда; објекат конкретне класе која наслеђује и имплементира апстрактну класу Command,
- *aService* – објекат класе *ConcreteServiceLocal*,
- *ConcreteServiceFactory* – једна од имплементација интерфејса *ServiceFactory*, изабрана у односу на окружење потребно команди, која креира објекат *aService*.



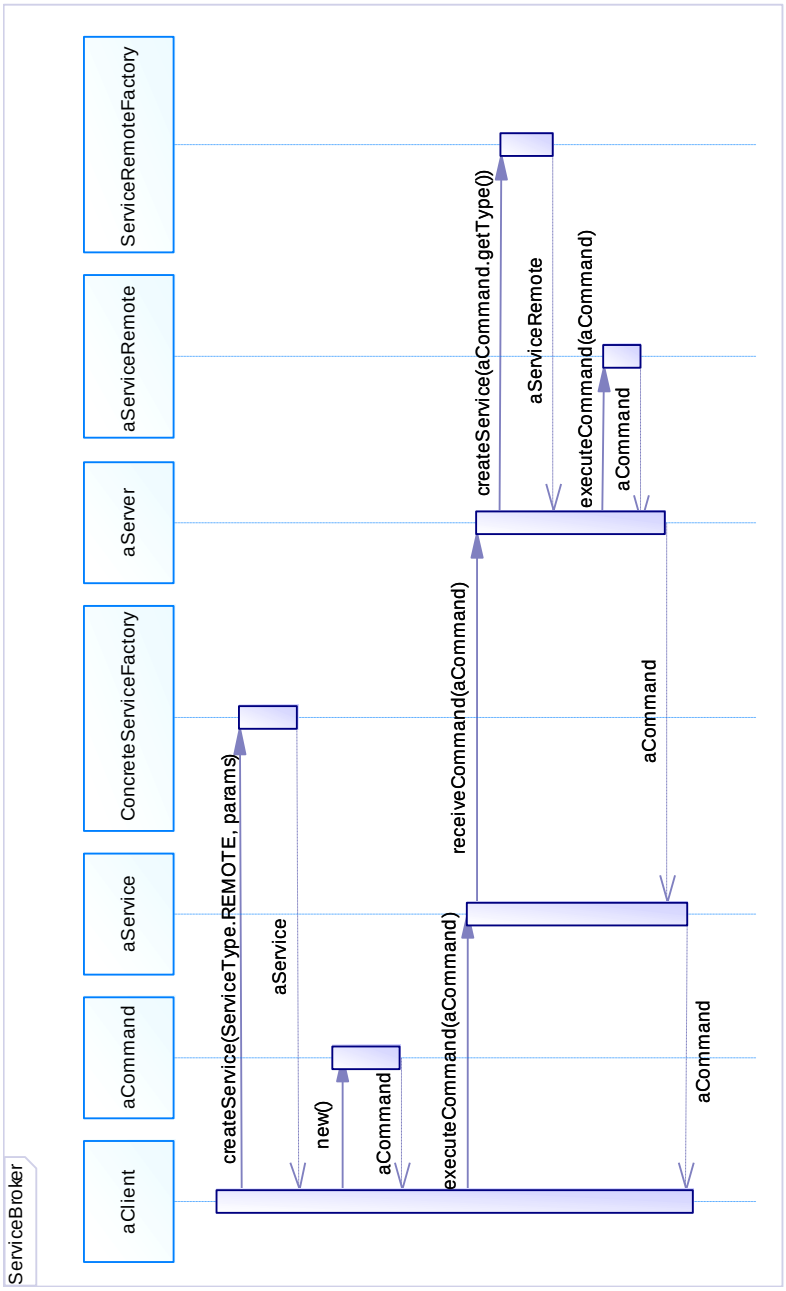
Слика 4.9 Дијаграм секвенци *ConcreteServiceLocal*

Клијентска апликација поруком *createService()* класи *ConcreteServiceFactory* иницира креирање објекта *aService*. На основу параметара који се прослеђују у поруци *createService()* класа *ConcreteServiceFactory* за објекат *aService* креира одговарајућу инстанцу класе *ConcreteServiceLocal*. Такође, клијентска апликација поруком *new()* креира објекат *aCommand*. Извршавање команде *aCommand* се иницира од стране клијента поруком *executeCommand()* објекту *aService* коме клијент прослеђује објекат *aCommand*. Објекат *aService* извршава команду позивајући метод *execute()* објекта *aCommand*. По завршетку извршавања акције објекат *aService* враћа клијенту објекат *aCommnad* у коме се налазе резултати извршавања акције одакле их клијент узима и даље користи.

На дијаграму секвенци приказаном на слици 4.10 је дат сценарио извршавања команде у случају када се користи имплементација *ServiceBroker* интерфејса *Service*. Имплементација *ServiceBroker* се користи у случају када се команда извршава на неком удаљеном серверу на Интернету и тада се команда шаље на сервер где се иницира њено извршавање. Позивањем метода *executeCommand* класе *ServiceBroker* иницира се извршавање метода *sendCommand* који команду шаље на одговарајући сервер. Без обзира на окружење потребно команди за извршавање, за слање команде се увек користи класа *ServiceBroker*. На серверској страни се на основу параметра команде *type* креира одговарајућа инстанца класе *ConcreteServiceRemote* која команди обезбеђује потребно окружење.

Објекти који учествују у интеракцији су:

- *aClient* - објекат клијентске апликације која користи подсистем и иницира извршавање команде,
- *aCommand* – команда; објекат конкретне класе која наслеђује и имплементира апстрактну класу Command,
- *aService* – објекат класе *ServiceBroker*,
- *ConcreteServiceFactory* – једна од имплементација интерфејса *ServiceFactory*, изабрана у односу на окружење потребно команди, која креира објекат *aService* (у случају слања команде на сервер све имплементације креирају објекат класе *ServiceBroker*),



Слика 4.10 *Дијаграм секвенци ServiceBroker*

- *aServer* – објекат класе *Server*; серверска страна система која је инсталирана на удаљеном серверу,
- *aServiceRemote* – инстанца одговарајуће класе *ConcreteServiceRemote,* изабрана у односу на окружење потребно команди,
- *ServiceRemoteFactory* – класа која креира објекат *aServiceRemote*.

Клијентска апликација као и на претходном дијаграму поруком *createService* иницира креирање објекта *aService*. На основу параметара који се прослеђују у тој поруци класа *ConcreteServiceFactory* у овом случају креира инстанцу класе *ServiceBroker*. Такође, клијентска апликација креира објекат *aCommand*, а затим иницира извршавање те команде поруком *executeCommand* објекту *aService* коме прослеђује објекат *aCommand*. Али у овом сценарију објекат *aService* не извршава команду, него је шаље серверској страни, класи *aServer*, кроз поруку *receiveCommand*. Серверска страна, односно објекат *aServer*, иницира креирање објекта *aServiceRemote* поруком *createService* класи *ServiceRemoteFactory.* У тој поруци као параметар прослеђује се параметар *type* добијеног објекта *aCommand* на основу ког се креира одговарајућа инстанца *aServiceRemote.* Извршавање команде је слично као на клијентској страни. Објекат *aServer* иницира извршавање добијене команде, односно објекта *aCommand,* поруком *executeCommand* објекту *aServiceRemote* коме се у поруци прослеђује команда. Објекат *aServiceRemote* извршава команду позивајући метод *execute* објекта *aCommand*, а затим тај објекат се са резултатима извршавања акције враћа клијентској страни, односно објекту *aService*. Објекат *aService* враћа објекат *aCommand* клијенту одакле клијент добија резултате извршавања акције.

Оваквом архитектуром направљена је подела функција у оквиру система. Пословна логика система је имплементирана *Command* класама, подсистем за комуникацију је задужен за извршавање команди без обзира на мрежно окружење, док је клијентска апликација задужена за интеракцију са корисником и слање команди подсистему. У случају проширења функционалности система додају се нове команде и евентуално праве измене у клијентској апликацији док подсистем за комуникацију остаје неизмењен.

4.1.4 Имплементација подсистема

На основу описане софтверске архитектуре у претходном одељку урађена је имплементација подсистема за клијент/сервер комуникацију. Имплементација је урађена у програмском језику *Java*. Све класе и интерфејси приказани на дијаграму класа на сликама 4.6, 4.7 и 4.8 имају своју имплементацију у облику *Java* класа и интерфејса. Класе *ConcreteCommand, ConcreteServiceFactory, ConcreteServiceLocal* и *ConcreteServiceRemote* на дијаграму су приказане само као илустрација конкретних имплементација одговарајућих интерфејса. Имплементирано је више таквих класа и имена су им дата у складу са конкретном имплементацијом.

За имплементацију комуникације између клијентске и серверске стране подсистема коришћен је програмски пакет *Hessian* [Hessian]. *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*.

Подсистем за клијент/сервер комуникацију интегрише се у неки конкретан систем тако што се акције пословне логике тог система имплементирају као класе које наслеђују апстрактну класу *Command*. Подсистем обезбеђује да је имплементација ових класа иста без обзира на то где се налази сервер и преко ког протокола му се приступа. Апстрактна класа *Command* је дата на листингу 4.1.

У зависности од акције која је имплементирана, команда може да захтева неко окружење или контекст за своје извршавање, као што је на пример конекција са базом података. Приликом имплементације команде у параметру *type* наводи се које окружење је потребно команди. У параметру *type* налази се једна од вредности дефинисана у класи *CommandType*, која је приказана на листингу 4.2. Овом класом су дефинисана окружења која подсистем може да обезбеди команди за извршавање. Описана имплементација подсистема може команди да обезбеди *JDBC* конекцију или *Hibernate* сесију за комуникацију са

базом података и локацију у фајл систему. Због саме архитектуре подсистема на лак начин може да се имплементира извршавање команди које захтевају било које друго окружење. Преко метода *getType()* онај ко извршава команду може да добије информацију шта је потребно команди обезбедити за извршавање, а методом *setContext()* команди се прослеђује параметар потребан за извршавање. Конкретне имплементације неких команди ће бити приказане у наредним одељцима који описују интеграцију подсистем у систем БИСИС.

```
public abstract class Command {
   protected Object context;
   protected int type;

   public void setContext(Object context){
         this.context = context;
   }
   public int getType(){
         return type;
   }
   public void setType(int type){
         this.type = type;
   }

  public abstract void execute();
}
```

Листинг 4.1 Апстрактна класа *Command*

```
public class CommandType {
   public static final int NONE = 0;
   public static final int HIBERNATE = 1;
   public static final int JDBC = 2;
   public static final int DIR = 3;
}
```

Листинг 4.2 Класа *CommandType*

Интерфејс *Service* служи за извршавање команди. Различите имплементације овог интерфејса омогућавају комуникацију у

различитим конфигурацијама мреже и обезбеђују командама различито окружење за извршавање. Креирање одговарајуће инстанце једне од имплементација интерфејса *Service* дефинисано је интерфејсом *ServiceFactory*. Постоји више имплементација интерфејса *ServiceFactory* и оне су на дијаграму класа на слици 4.7 представљене класом *ConcreteServiceFactory.* Различите имплементације интерфејса постоје у односу на окружење потребно команди за извршавање. Као што је већ речено, тренутна имплементација подсистема може команди да обезбеди *JDBC* конекцију или *Hibernate* сесију за комуникацију са базом података и локацију у фајл систему. То значи да тренутно постоје четири имплементације интерфејса *ServiceFactory* и то су:

- *HibernateServiceFactory* – креира одговарајућу инстанцу интерфејса *Service* која команди обезбеђује *Hibernate* сесију,

- *JdbcServiceFactory* - креира одговарајућу инстанцу интерфејса *Service* која команди обезбеђује *JDBC* конекцију,

- *FileServiceFactory* - креира одговарајућу инстанцу интерфејса *Service* која команди обезбеђује референцу на директоријум у фајл систему,

- *PlainServiceFactory* - креира одговарајућу инстанцу интерфејса *Service* која команди не обезбеђује никакво додатно окружење.

За сваку имплементацију интерфејса *ServiceFactory* на клијентској страни подсистема постоје две имплементације интерфејса *Service* које, поред тога што командама обезбеђују одговарајуће окружење, омогућавају извршавање команде у два различита мрежна окружења. Једна имплементација се користи у случају када се команда извршава у локалном окружењу, а друга у случају када се команда извршава на неком удаљеном серверу на Интернету. На дијаграму класа на слици 4.7 ове имплементације су илустративно представљене класама *ConcreteServiceLocal* и *ServiceBroker.* Одговарајућа инстанца интерфејса *Service* креира се унутар метода *createService*. Као параметри том методу се прослеђује тип сервиса који се креира и *URL* адреса сервера на ком се налази серверска страна подсистема у случају да се креира инстанца за удаљени приступ. У случају кад је вредност првог параметра *ServiceType.LOCAL* креира се инстанца за локално извршавање команди, а у случају да је вредност параметра *ServiceType.REMOTE* креира се инстанца за удаљени приступ која комуницира са сервером

на адреси датој у другом параметру. Креирана инстанца се враћа као повратна вредност метода *createService*. На листингу 4.3 приказана је имплементација овог метода класе *HibernateServiceFactory*. Одговарајуће инстанце интерфејса *Service* које креира ова класа су *HibernateServiceLocal* и *ServiceBroker*. И за преостале две имплементације интерфејса *ServiceFactory* постоје одговарајуће имплементације интерфејса *Service*, с тим што се у свим случајевима за удаљени приступ користи класа *ServiceBroker* као што је већ објашњено у претходном одељку. У класи *ServiceType* су дефинисани начини извршавања команди. Описана имплементација подсистема има претходно описана два начина.

```
public class HibernateServiceFactory implements ServiceFactory {
   public Service createService(int type, String param){
       CirculationService service;
       if (type == ServiceType.LOCAL){
          service = new HibernateServiceLocal();
       } else if (type == ServiceType.REMOTE) {
          service = new ServiceBroker(param);
       };
       return service;
   }
}
```

Листинг 4.3 Имплементација класе *HibernateServiceFactory*

Класа *HibernateServiceLocal* представља имплементацију интерфејса *Service* која се користи у случају када команда може да се изврши на истом месту одакле је иницирано њено извршавање, односно на клијенту. Поред тога ова класа команди обезбеђује *Hibernate* сесију, што значи да се користи за извршавање команди које за комуникацију са базом података користе *Hibernate.*

У случају када се сервер базе података налази у интранету са њим је могуће остварити перманентну конекцију директно из клијентске апликације, односно омогућена је комуникација *Hibernate*-а са базом података директно из клијента. То значи да је у овом случају могуће команде извршити на клијенту и да ће комуникација са базом података бити успешно остварена.

Из овога следи да се имплементација *HibernateServiceLocal* користи за извршавање команди које за комуникацију са базом података користе *Hibernate* при чему су и сервер базе података и клијент смештени у локалном интранету.

На листингу 4.4 приказана је имплементација метода *executeCommand* класе *HibernateServiceLocal*. Приликом извршавања овог метода позива се метод *execute* објекта *command* који му је прослеђен као параметар. Пре позива метода *execute* објекту *command* се обезбеђује *Hibernate* сесија која му се прослеђује позивом метода *setContext*.

```
public Command executeCommand(Command command){
    .... dobavi Hibernate sesisju
    command.setContext(session);
    command.execute();
    command.setContext(null);
    session.close();
    return command;
}
```

Листинг 4.4 Имплементација метода *executeCommand* класе *HibernateServiceLocal*

Класа *ServiceBroker* представља имплементацију интерфејса *Service* која се користи у случају када је команду потребно извршити на некој другој локацији различитој од локације где се иницира извршавање команде. У случају извршавања команде на удаљеном серверу команду је потребно пребацити са клијента на сервер. Позивањем метода *executeCommand* класе *ServiceBroker* команда се шаље серверској страни подсистема.

У случају када се извршава команда која укључује комуникацију са базом података, а при томе се база података налази на неком удаљеном серверу са њом није могуће остварити перманентну конекцију. У том случају команду је потребно пребацити на тај удаљени сервер или неки други сервер који је у интранету са сервером на ком је база података. Извршавање команде на том удаљеном серверу ће остварити успешну комуникацију са базом података. При извршавању команде на серверској страни команди је потребно

обезбедити конекцију са базом података слично као у случају претходно описаних класа за локално извршавање команди.

Из овога следи да се имплементација *ServiceBroker* може користи за извршавање команди које укључују комуникацију са базом података при чему је сервер базе података смештен на удаљеном серверу на Интернету. Поред тога, ова имплементација се користи и за све друге врсте команди које је потребно пребацити на серверску страну пре извршавања.

За имплементацију комуникације између клијента и сервера је коришћен програмски пакет *Hessian* и *Burlap XML* протокол који је имплементиран у оквиру тог пакета. Помоћу овог пакета на серверској страни је креиран сервлет, а на клијентској страни *proxy* објекат преко кога се комуницира са тим сервлетом.

За креирање сервлета помоћу *Hessian* пакета било је потребно направити интерфејс и класу који дефинишу и имплементирају функција сервлета. Функције овог сервлета су описане класом *Server* на дијаграму класа на слици 4.8 у претходном одељку. За потребе имплементације направљени су интерфејс *Server* и класа *ServerImpl* који представљају имплементацију сервиса на серверској страни и на основу њих је креиран сервлет. Интерфејс *Server* има метод *receiveCommand* кроз који се команда прослеђује са клијента на серверску страну.

```
public class ServiceBroker(String param){
   try {
       BurlapProxyFactory proxyFactory = new BurlapProxyFactory();
      service = (Server)proxyFactory.create(Server.class, param);
    } catch (MalformedURLException e) {}
.............
   public Command executeCommand(Command command){
      return  service.receiveCommand(command);
   }
}
```

Листинг 4.5 Имплементација метода *executeCommand* класе *ServiceBroker*

Proxy објекат преко ког се комуницира са серверском страном користи се на клијентској страни у имплементацији класе *ServiceBroker*. Пошто се *proxy* објекат креира на основу интерфејса *Server* он има један метод *receiveCommand* са параметром типа *Command*. Позивањем тог метода параметар односно команда се кроз *Burlap* протокол шаље сервлету на сервер, одакле се иницира извршавање команде. *Proxy* објекат се креира у класи *ServiceBroker* чија имплементација је приказана на листингу 4.5. Приликом креирања *proxy* објекта као параметар прослеђује му се адреса на којој се налази сервлет.

На серверској страни класа *ServerImpl* по добијању команде иницира њено извршавање. Да би се команда извршила потребно је претходно креирати одговарајућу инстанцу интерфејса *Service* која ће да изврши ту команду. Слично као на клијенту, и на серверу постоје различите имплементације интерфејса *Service* у зависности од окружења које је потребно обезбедити команди за извршавање. Те имплементације су на дијаграму класа на слици 4.8 у претходном одељку илустративно представљене класом *ConcreteServiceRemote*. Као што је већ речено, тренутна имплементација подсистема може команди да обезбеди *JDBC* конекцију или *Hibernate* сесију за комуникацију са базом података и локацију у фајл систему. То значи да тренутно на серверској страни постоје четири имплементације интерфејса *Service* и то су: *HibernateServiceRemote*, *JdbcServiceRemote*, *FileServiceRemote* и *PlainServiceRemote*.

Имплементација ових класа је врло слична имплементацији класа за локално извршавање команди на клијентској страни. Једина разлика је у начину на који се обезбеђује окружење за команду. Претходно је на листингу 4.4 приказана имплементација метода *executeCommand* класе *HibernateServiceLocal*. На исти начин је имплементиран метод *executeCommand* класе *HibernateServiceRemote*. Једина разлика између имплементације класа *HibernateServiceLocal* и *HibernateServiceRemote* је у начину на који се остварује *Hibernate* сесија. У случају класе *HibernateServiceLocal*, *Hibernate* се конфигурише тако да креира конекцију са базом података преко *JDBC* драјвера, а у случају класе *HibernateServiceRemote* конекција се добавља из серверовог *JNDI datasource*-а. Након добављања *Hibernate* сесије команде се у случају обе класе извршавају на исти начин.

```
public class ServiceRemoteFactory{
    private static Service hibernateservice;
    private static Service jdbcservice;
    private static Service fileservice;
    private static Service service;
…….
     public static Service createService(int type){
          if (type == CommandType.HIBERNATE){
              return getHibernateService();
           } else if (type == CommandType.JDBC){
              return getJdbcService();
          } else if (type == CommandType.DIR){
              return getFileService();
           } else if (type == CommandType.NONE){
              return getService();
           }
           return null;
     }
……
    private static Service getHibernateService(){
           if (hibernateservice == null){
                hibernateservice = new HibernateServiceRemote();
           }
           return hibernateservice;
   }
……..
}
```

Листинг 4.6 Имплементација класе ServiceRemoteFactory

Инстанца одговарајуће класе на серверској страни се креира преко класе *ServiceRemoteFactory*. Ова класа има један метод *createService* који у зависности од параметра који му се проследи враћа инстанцу једне од претходно описане три имплементације интерфејса *Service*. Параметар који је потребно проследити налази се у добијеној команди у пољу *type* и има једну од дефинисаних вредности у класи *CommandType* претходно приказаној на листингу 4.2. На листингу 4.6 је приказана имплементација метода *createService* класе *ServiceRemoteFactory*. Једном

креирана инстанца интерфејса *Service* чува се у класи и користи поново при наредним позивима.

Описана имплементација подсистема омогућава извршавање команди на клијенту и извршавање команди на удаљеном серверу при чему је комуникација са сервером имплементирана помоћу програмског пакета *Hessian* кроз *HTTP* протокол. Додавањем нових имплементација интерфејса *Service* може се имплементирати комуникација и кроз друге протоколе.

Такође, описана имплементација може да извршава команде које не захтевају додатно окружење за своје извршавање, команде које комуницирају са базом података и при томе користе *Hibernate* програмски пакет или *JDBC API* и команде које користе ресурсе смештене у фајл систем. Из архитектуре и имплементације се може видети да је за извршавање команди које захтевају неко друго окружење потребно имплементирати нову фамилију класа која је у архитектури описана класама *ConcreteServiceFactory*, *ConcreteServiceLocal* и *ConcreteServiceRemote.* У описаној имплементацији постоје следеће фамилије:

- *HibernateServiceFactory*, *HibernateServiceLocal* и *HibernateServiceRemote* – за извршавање команди које за комуникацију са базом података користе *Hibernate* пакет,
- *JdbcServiceFactory*, *JdbcServiceLocal* и *JdbcServiceRemote* – за извршавање команди које за комуникацију са базом података користе *JDBC API*,
- *FileServiceFactory*, *FileServiceLocal* и *FileServiceRemote* – за извршавање команди које користе ресурсе смештене у фајл систем,
- *PlainServiceFactory*, *PlainServiceLocal* и *PlainServiceRemote* – за извршавање команди које не захтевају додатно окружење за своје извршавање.

4.2 Интеграција подсистема у БИСИС систем

У претходним одељцима овог поглавља приказана је софтверска архитектура и имплементација подсистема за клијент/сервер комуникацију. Подсистем је направљен са идејом да се његовом

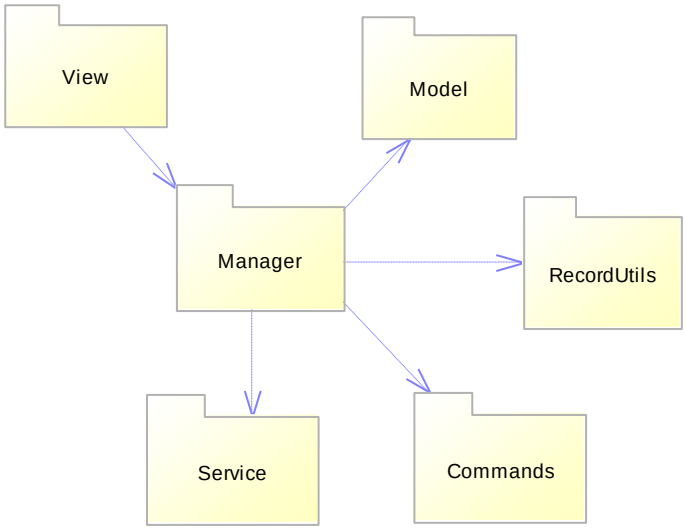
употребом омогући транспарентна комуникација клијентске и серверске стране система који га користи, односно комуникација у различитим мрежним окружењима.

Разлог за интеграцију овог подсистема у БИСИС систем су библиотеке које имају одељења и огранке на различитим локацијама. Нека од тих одељења се физички налазе на истој локацији на којој је библиотечки сервер, а нека не. Одељења која се налазе на истој локацији као и сервер, са севером су обично повезана преко локалне мреже. Одељења која се не налазе на истој локацији, комуникацију са сервером обично остварују преко Интернета. Због тога је било потребно омогућити рад клијентске апликације БИСИС система у различитим мрежним окружењима. Имплементација подсистема која је направљена омогућава рад у локалној мрежи и рад на Интернету преко *web* сервиса што задовољава потребе БИСИС система.

4.2.1 Интеграција у систем за циркулацију

За рад са базом података у систему циркулације се користи *ORM* технологија и *Hibernate* пакет. *ORM* (*Object/Relational Mapping*) технологија представља начин мапирања објектно-оријентисане логике апликације на релациони модел базе података. *Hibernate* пакет је једна од имплементација *ORM* технологије. *Hiberanate* омогућава како мапирање класа апликације на табеле базе података тако и перзистенцију тих класа. Класе апликације са којима *Hiberante* ради представљају објектни модел базе података и имплементиране су као *POJO* класе. Податке из базе података са којима апликација тренутно ради *Hiberante* привремено смешта у објекте тих класа одакле их апликација даље користи, а такође из тих објеката их поново враћа у базу података.

Дијаграм пакета система за циркулацију је приказан у трећем поглављу у одељку 3.2.1. Подсистем за комуникацију који је приказан у овом поглављу је интегрисан у систем за циркулацију тако што је постојећи пакет *Proxy* који је коришћен за комуникацију са базом података замењен са подсистемом. Нови дијаграм пакета система за циркулацију је приказан на слици 4.11.



4.11 Дијаграм пакета система за циркулацију

У систем за циркулацију су укључена два нова пакета: *Service* и *Commands*. У овим пакетима се налазе класе подсистема за комуникацију. У пакету *Commands* се налазе интерфејс *Command* и све конкретне имплементације интерфејса (приказано и описано на дијаграму класа на слици 4.6 у одељку 4.1.3). У пакету *Service* се налази клијентска страна подсистема за клијент/сервер комуникацију, односно интерфејси и класе којима је имплементиран сервис за извршавање команди, а приказани су на дијаграму класа на слици 4.7 у одељку 4.1.3.

Да би се подсистем за комуникацију интегрисао у систем циркулације било је потребно све акције циркулације које укључују комуникацију са базом података имплементирати као команде. То је урађено тако што су све методе дефинисане интерфејсом *Proxy* приказаном на дијаграму у одељку 3.2.5 имплементиране као команде. Атрибути тих команди су објекти објектног модела базе података који је приказан у трећем поглављу у одељку 3.2.3. Пример једне такве команде приказан је на листингу 4.7. Ова команда представља акцију проналажења података о кориснику у бази података за задати чланski број корисника. Класа *User* је *POJO* класа која представља део објектног модела базе података која се користи за трајно смештање података о кориснику. На овај начин су све функционалности пакета *Proxy* пренесене на подсистем за комуникацију. Конкретне команде које су

имплементације функционалности пакета *Proxy* су смештене у пакет *Commands*.

Систем циркулације за комуникацију са базом података користи *Hibernate* пакет. То значи да је командама при извршавању потребно обезбедити *Hibernate* сесију. Та информација се приликом имплементације смешта у команду тако што се у поље *type* поставља вредност *CommandType.HIBERNATE.* Да се то не би понављало приликом имплементације сваке команде у систему за циркулацију је направљена апстрактна класа *HibernateCommand* која наслеђује класу *Command* и поставља одговарајућу вредност. Конкретне команде наслеђују ову класу што се види и у приказаној команди на листингу 4.7.

Команде се извршавају кроз подсистем за клијент/сервер комуникацију. За извршавање команди се користи фамилија класа која обезбеђује *Hibernate* сесију, а то су *HibernateServiceFactory*, *HibernateServiceLocal* и *HibernateServiceRemote.*

Начин на који се извршавају команде, односно начин на који клијент приступа бази података, подешава се у конфигурационом фајлу клијентске апликације. У конфигурационом фајлу се налази информација да ли се користи локални или удаљени приступ и у случају удаљеног приступа адреса на којој се налази сервис. Једна библиотека може имати локалне клијенте који директно приступају бази података, као и удаљене клијенте које бази података приступају путем Интернета. За обе врсте клијената користи се идентична клијентска апликација која се разликује само по конфигурацији. Коришћењем истог програмског кода за све клијенте олакшава се одржавање и даљи развој апликације.

Поред тога што је пакет *Proxy* замењен пакетима *Service* и *Commands*, измене су направљене и у класама пакета *Manager* описаног у одељку 3.2.4. Класе пакета *Manager* управљају током података од корисничког интерфејса до базе података и у својим функционалностима су користиле пакет *Proxy* за комуникацију са базом података. Интеграцијом подсистема за комуникацију класе пакета *Manager* су постале задужене за креирање команди и иницирање њиховог извршавања, односно прослеђивање команди сервису за комуникацију са базом података.

```
public class GetUserCommand extends HibernateCommand {
    String userID;
    User user = null;

    public void setUserID(String userID){
        this.userID = userID;
    }

    public User getUser(){
        return user;
    }

    public void execute() {
        session.beginTransaction();
        Query q = session.createQuery("from User u
                                        where u.userId = :userid");
        q.setString("userid", userID);
        user = (User)q.uniqueResult();
        session.commitTransaction();
    }
}
```

Листинг 4.7 Имплементација акције проналажења корисника

4.2.2 Интеграција у остатак БИСИС система

На претходно описан начин подсистем за клијент/сервер комуникацију је интегрисан у систем за циркулацију чиме је омогућен рад система у различитим мрежним окружењима. Клијентска апликација циркулације је део клијентске апликације БИСИС система, која поред циркулације садржи и обраду библиотечке грађе, извештавање о обради, претраживање библиотечког фонда и администрацију система.

Поред система за циркулацију још неки делови клијентске апликације БИСИС система комуницирају директно са базом података, на пример, логовање на систем, учитавање генералних параметара за клијента, део за администрацију система и др. Ови делови система

користе различиту технологију за комуникацију са базом података у односу на циркулацију. Они су имплементирани тако да са базом података комуницирају преко *JDBC* драјвера. Комуникација преко *JDBC* драјвера није погодна за рад у Интернет окружењу јер јој је потребна перманентна конекција са базом података. Због тога је комуникација са базом података и у овим деловима система имплементирана преко подсистема за клијент/сервер комуникацију.

Подсистем је у ове делове интегрисан слично као у систем за циркулацију. Акције које укључују комуникацију са базом података имплементиране су као команде и те команде се прослеђују подсистему, а он даље обезбеђује конекцију са базом података и извршава команде. Будући да се за комуникацију са базом података користи *JDBC API,* командама је при извршавању потребно обезбедити *JDBC* конекцију. Због тога се за извршавање команди користи фамилија класа која обезбеђује *JDBC* конекцију, а то су *JdbcServiceFactory*, *JdbcServiceLocal* и *JdbcServiceRemote.* Поред тога та информација се при имплементацији смешта у команду тако што се у поље *type* поставља вредност *CommandType.JDBC.*

На овај начин су сви делови БИСИС клијента који комуницирају са базом података прилагођени да раде у различитим мрежним окружењима.

Поред базе података неки подаци у оквиру БИСИС система се смештају у фајлове у фајл систему. Делови БИСИС система који приступају фајловима су подсистем за генерисање извештаја и подсистем за претраживање библиографских записа.

Подсистем за генерисање статистичких извештаја о библиотечком фонду у оквиру БИСИС система описан је у раду (Boberić і Milosavljević, 2009). Подсистем има два начина генерисања извештаја: у реалном времену и периодично. За случај периодичног генерисања извештаја на серверској страни система постоји апликација која генерише податке за извештаје у одређеним временским интервалима. Те податке чува у *XML* фајловима на серверу. На серверској страни се налази и сервлет који на захтев клијента враћа одговарајући *XML* фајл на основу кога се на клијентској страни креира и приказује извештај.

Подсистем за клијент/сервер комуникацију је у овај део интегрисан и искоришћен за захтеве за *XML* фајловима од стране клијентске стране

која креира и приказује извештаје. Акције које су директно од сервлета тражиле појединачне фајлове или списак фајлова имплементиране су као команде. Извршавање ових команди захтева приступ фајловима који се налазе у фајл систему. То значи да је командама пре извршавања потребно обезбедити референцу на одговарајући директоријум у фајл систему где се налазе фајлови. Због тога се за извршавање команди кроз подистем за клијент/сервер комуникацију користи фамилија класа *FileServiceFactory*, *FileServiceLocal* и *FileServiceRemote.* Такође, при имплементацији команди, у поље *type* поставља се вредност *CommandType.DIR.*

С обзиром на то да се *XML* фајлови са припремљеним подацима за извештаје генеришу и чувају на серверу, а команде се креирају и иницирају од стране клијента, за извршавање ових команди се користи интерфејс за удаљени приступ. Команде се извршавају локално само у случају када су и сервер и клијент смештени на истом рачунару.

На сличан начин је подсистем за клијент/сервер комуникацију интегрисан у део система за претраживање библиографских записа који је описан у раду (Milosavljević i dr, 2010). Компонента за претраживање је базирана на програмском пакету *Lucene*. *Lucene* је *search engine* који омогућава ефикасно индексирање и претраживање текстуалних докумената. Индексирани подаци се чувају у фајлу који се назива индекс. Компонента за претраживање БИСИС система реализована је тако да се библиографски записи физички чувају у бази података, док се у индексу за претраживање налазе само вредности по којима се претражује и референца на одговарајући запис. Претраживање се врши по префиксима. Сваки префикс обухвата одређен скуп поља и потпоља библиографског записа. Индекс за претраживање, као и база података, налази се на серверу. Компонента за претраживање има две врсте операција, за претраживање и за рад са записима. Операцијама за претраживање се прослеђују упити који се извршавају над индексом. Операције за рад са записима комуницирају са базом података где су смештени записи.

Подсистем за клијент/сервер комуникацију је у овај део интегрисан тако што су операције за претраживање и операције за рад са записима имплементиране као команде. Команде којима је имплементирано претраживање за своје извршавање захтевају

приступ индексу који је смештен у фајл у фајл систему. Због тога се за извршавање ових команди кроз подсистем за клијент/сервер комуникацију користи фамилија класа *FileServiceFactory*, *FileServiceLocal* и *FileServiceRemote,* као у случају подсистема за генерисање извештаја. Такође, при имплементацији команди, у поље *type* поставља се вредност *CommandType.DIR.*

Команде којима је имплементиран рад са записима за своје извршавање захтевају конекцију за базом података, па се за извршавање ових команди користи фамилија класа која обезбеђује *JDBC* конекцију, а то су *JdbcServiceFactory*, *JdbcServiceLocal* и *JdbcServiceRemote.* Поред тога та информација се при имплементацији смешта у комаду тако што се у поље *type* поставља вредност *CommandType.JDBC.*

Како се и индекс и база података налазе на серверу, а команде се креирају и иницирају од стране клијента и у овом случају се за извршавање команди користи интерфејс за удаљени приступ. Будући да се за слање команди на сервер унутар подсистема увек користи иста класа *ServiceBroker*, на клијенту је довољно креирати једну инстанцу интерфејса за извршавање команди за обе врсте команди. На серверској страни се, у зависности од вредности у поље *type,* команди обезбеђује одговарајуће окружење за извршавање.

На овај начин се комуникација између клијентске и серверске стране целог БИСИС система врши преко подсистема за клијент/сервер комуникацију.

Поглавље 5

Циркулација у конзорцијуму библиотечке мреже БИСИС

У претходна два поглавља описан је систем за циркулацију који је интегрисан у библиотечки систем БИСИС верзије 4. Систем за циркулацију има подршку за комуникацију са различитим репозиторијумима библиографских записа, као и подршку за различите *MARC* формате тако да се може интегрисати и у друге библиотечке системе. Систем се састоји из три дела: клијентске апликације, подсистема за клијент/сервер комуникацију и базе података. Ова архитектура је приказана на дијаграму 3.2 у трећем поглављу. Клијентска апликација је имплементирана као десктоп апликација. Клијентска апликација комуникацију са базом података остварује преко подсистема за комуникацију који је детаљно описан у четвртом поглављу. Подсистем омогућава комуникацију са базом података у различитим мрежним окружењима чиме је омогућен рад клијентске апликације са различитих локација. Подсистем је имплементиран тако да је пословна логика система иста без обзира на мрежну конфигурацију. Такође, подсистем није везан за пословну логику система па се може интегрисати и у друге системе. Претходно описан систем за циркулацију подржава активности везане за евидентирање и услуживање корисника искључиво у оквиру локалног фонда библиотеке.

Библиотеке имају потребу да корисницима омогуће коришћење фондова других библиотека. Библиотечки процес којим се најчешће решава овај проблем је међубиблиотечка позајмица у ком библиотеке између себе размењују публикације, а затим корисник те публикације користи на исти начин као и локални фонд библиотеке.

У неким случајевима као што су мрежа факултетских библиотека на универзитету или чак и мрежа градских библиотека јавља се потреба за могућношћу коришћења фондова свих библиотека у мрежи од стране једног корисника, али на локацијама тих фондова. Те библиотеке се организационо повезују у конзорцијуме. Унутар конзорцијума библиотеке чланице конзорцијума успостављају правила за коришћење библиотечких фондова. Овај начин коришћења библиотечких фондова се назива циркулација унутар конзорцијума. У односу на међубиблиотечку позајмицу циркулација унутар конзорцијума у наведеним случајевима може знатно да поједностави и убрза процес коришћења фондова библиотека.

Циркулација унутар конзорцијума је библиотечки процес у ком је корисницима једне библиотеке омогућено да позајмљују књиге из других библиотека унутар конзорцијума. Да би овај процес функционисао потребно је обезбедити размену података о корисницима између библиотека. Библиотека којој корисник припада представља његову матичну библиотеку. Када корисник дође у неку библиотеку која није његова матична потребно је тој библиотеци омогућити приступ подацима о том кориснику који се налазе у његовој матичној библиотеци. Тај приступ је потребан да би се, на пример, проверило да ли тај корисник има право да задужује књиге. Такође, када корисник задужи књигу у некој библиотеци та информација би требала да буде доступна и другим библиотекама.

У овом поглављу је описано проширење система за циркулацију које обухвата функционалности потребне за процес циркулације унутар конзорцијума. Те функционалности укључују комуникацију и размену података са другим библиотекама. У овом поглављу је описан случај комуникације и размене података када се на обе стране налази библиотечки систем БИСИС. Комуникација са другим библиотечким системима је описана у наредном поглављу.

5.1 Функционалности система за рад са корисницима из других библиотека

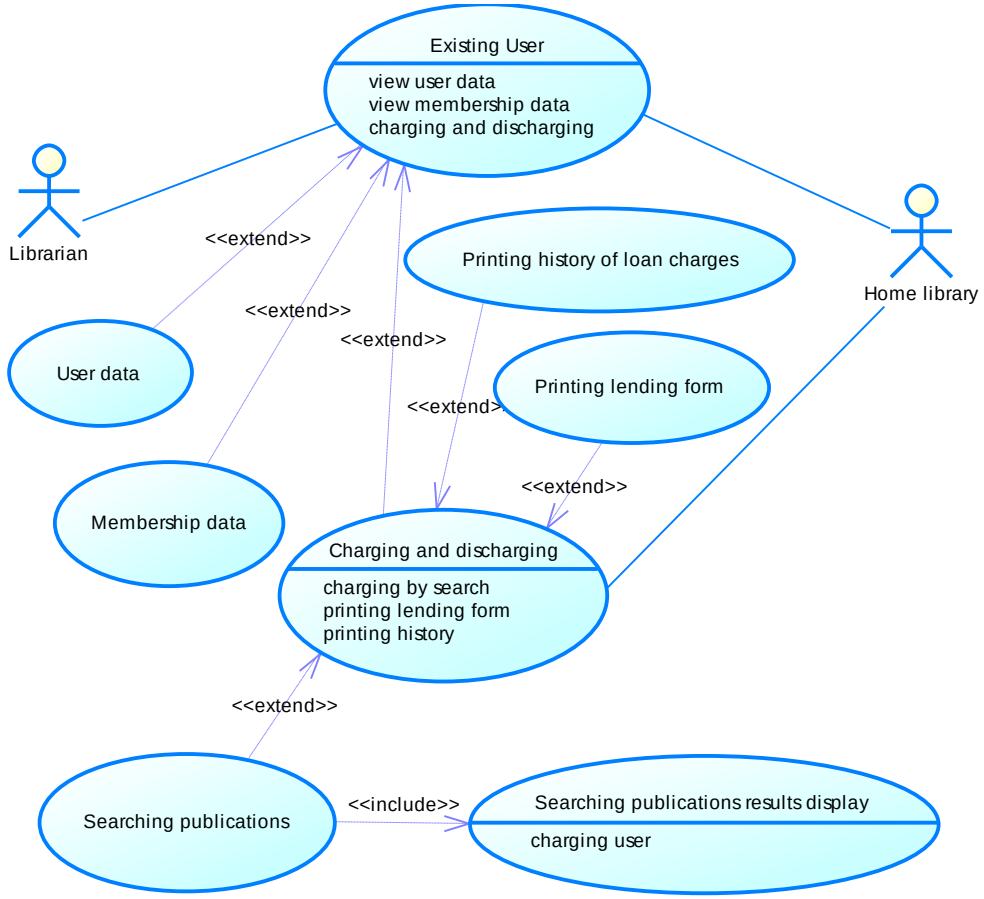
У трећем поглављу на дијаграму случајева коришћења, на слици 3.1, приказане су функционалности система за рад са корисницима библиотеке. На њему су приказане акције које библиотекар

свакодневно врши при услуживању корисника. Ту спадају учлањење нових корисника, продужење чланарине, задуживање и раздуживање корисника и претраживање корисника и фонда. У случају када се ради о услуживању корисника који је дошао из друге библиотеке скуп функционалности система за услуживање тог корисника је много мањи, а неке од тих функционалности су проширене. На дијаграму случајева коришћења на слици 5.1 издвојен је скуп функционалности система за рад са корисницима из других библиотека који је подскуп скупа функционалности дефинисаног на слици 3.1. Функционалност неких од приказаних случајева коришћења су проширене да би се у систему омогућило услуживање корисника из других библиотека. Као учесник на дијаграму се појављује и систем матичне библиотеке корисника јер функционалности укључују комуникацију са тим системом због података о кориснику. У текстуалном опису случајева коришћења датом у даљем тексту наведена су само проширења функционалности случајева коришћења у односу на функционалности дефинисане дијаграмом на слици 3.1.

Текстуални опис случајева коришћења:

Случај коришћења *Existing User*. На основу чланског броја корисника и информације којој библиотеци припада библиотекар проналази податке о кориснику у библиотечкој мрежи. Спецификација реализације овог случаја коришћења је приказана и описана дијаграмом активности на слици 5.2.

Случај коришћења *User data*. Библиотекару се приказују подаци о кориснику који постоје у матичној библиотеци. У податке о кориснику спадају сви подаци дефинисани случајем коришћења на слици 3.1 а то су: име и презиме, име родитеља, адреса и место становања, број телефона, *e-mail* адреса, ЈМБГ, број документа и место издавања документа, пол, узраст, додатна адреса, степен образовања, занимање, звање, организација у којој је корисник запослен, матерњи језик, интересовања и напомена. Библиотекар нема могућност измене ових података.



Слика 5.1 *Дијаграм случајева коришћења за рад са корисницима из других библиотека*

Случај коришћења *Membership data*. Библиотекару се приказују подаци о чланству корисника у библиотеци. У те податке спадају сви подаци дефинисани случајем коришћења на слици 3.1 а то су: чланski број корисника, ако се корисник уписује преко колективног корисника тада се повезује са подацима о колективном кориснику, категорија корисника, врста учлањења, датум од кад важи чланарина, датум до кад важи чланарина, одељење на које се корисник уписује, цена, и број признанице која се издаје кориснику за плаћену чланарину. Библиотекар нема могућност измене ових података.

Случај коришћења *Charging and discharging*. Библиотекару се приказује списак тренутних задужења корисника. Библиотекар може да задужи или раздужи корисника или продужи задужење. Задуживање се врши

на исти начин као што је дефинисано случајем коришћења на слици 3.1 за локалне кориснике, а то значи да се публикација може задужити уношењем инвентарног броја публикације или проналажењем публикације претраживањем. У случају проналажења публикације претраживањем реализује се случај коришћења *Searching publications.* Детаљна спецификација реализације овог случаја коришћења је приказана и описана на дијаграму активности на слици 3.3. Ако библиотекар жели да штампа реверс кориснику, тада се реализује случај коришћења *Printing lending form.* Ако библиотекар жели да види или штампа претходна задужења корисника, тада се реализује случај коришћења *Printing history of loan charges.*

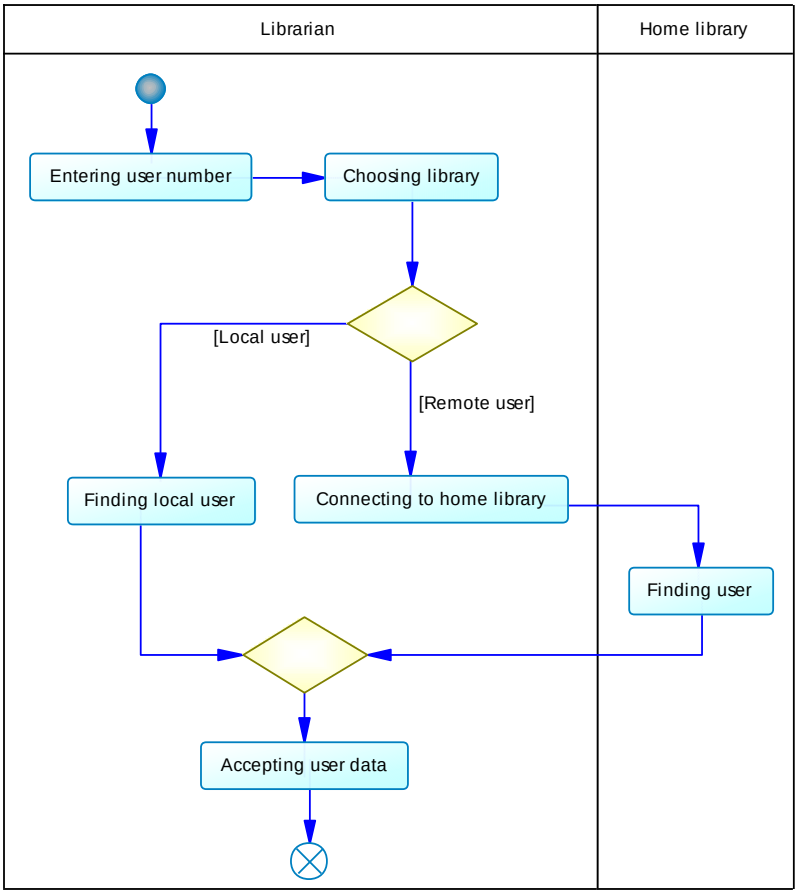
Случајеви коришћења *Searching publications, Searching publications results display, Printing lending form* и *Printing history of loan charges* немају додатна проширења у односу на дијаграм на слици 3.1 и због тога њихов опис овде није наведен.

5.2 Дијаграми активности случајева коришћења

Да би се приказале разлике у функционисању система у случају када систем ради са подацима локалног корисника и у случају када ради са подацима корисника из друге библиотеке у овом одељку су приказане функционалности заједничких случајева коришћења приказаних на сликама 3.1 и 5.1.

На дијаграму активности на слици 5.2 приказана је функционалност случаја коришћења *Existing User*. Овај случај коришћења је приказан на оба наведена дијаграма случајева коришћења. На дијаграму на слици 3.1 њиме је дефинисана активност проналажења података постојећег локалног корисника у систему, а на дијаграму на слици 5.1 његова функционалност је проширена проналажењем података корисника у библиотечкој мрежи. На дијаграму активности је приказана комплетна функционалност овог случаја коришћења где се јасно могу уочити разлике у функционисању система у случају локалних корисника и у случају корисника из других библиотека. Активност *Entering user number* представља унос чланског броја корисника, а активност *Choosing library* избор библиотеке којој корисник припада. Извршавањем ових активности задају се параметри систему на основу којих се проналазе подаци о кориснику.

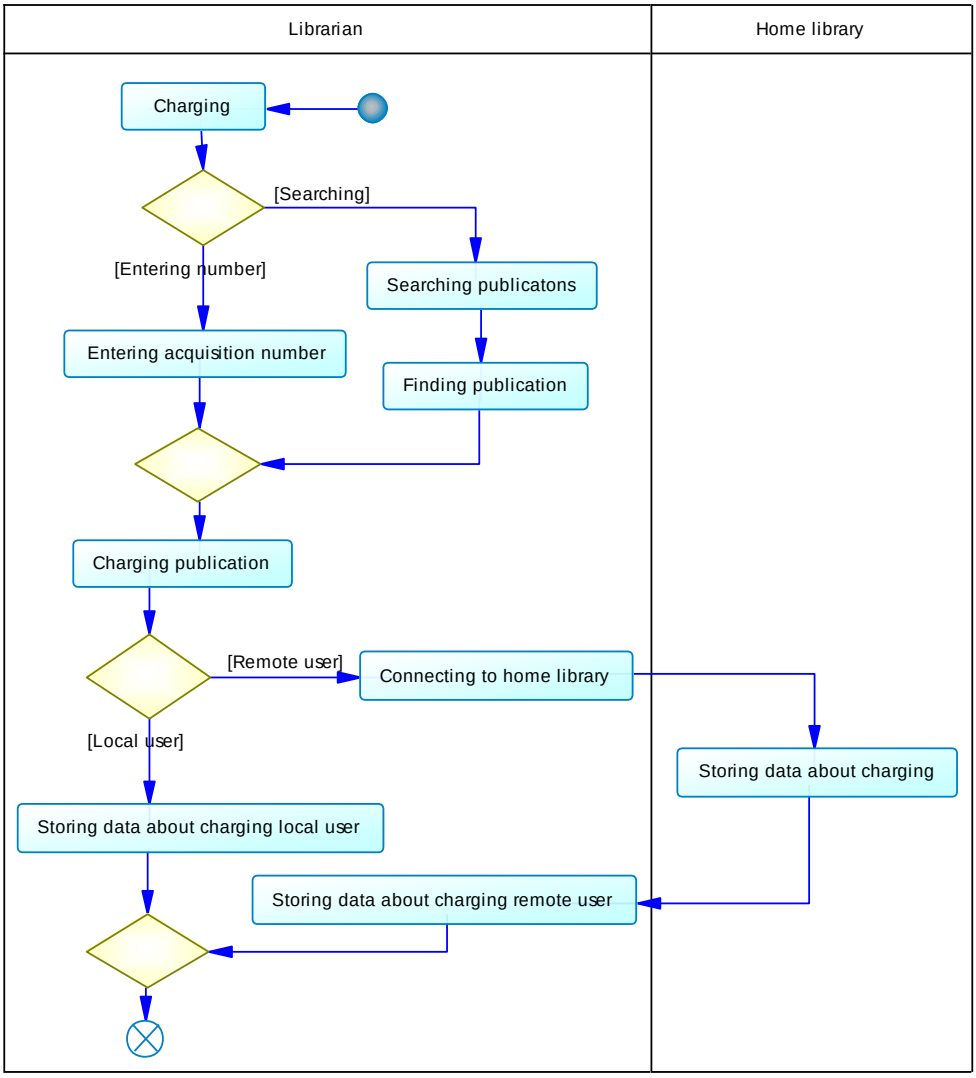
У зависности од тога да ли је при избору библиотеке изабрано да се ради о локалном кориснику или о кориснику из неке друге библиотеке извршава се један од два тока контроле. У случају да се ради о локалном кориснику подаци о кориснику се проналазе у локалном систему што је представљено активношћу *Finding local user*. У случају да се ради о некој другој библиотеци у конзорцијуму апликација приступа систему те библиотеке и тамо проналази податке о кориснику. Ово је представљено активностима *Connecting to home library* и *Finding user*. Након проналажења корисника било у локалном систем или у систему неке друге библиотеке подаци о кориснику се враћају апликацији која их даље процесира што је представљено активношћу *Accepting user data*.



Слика 5.2 Дијаграм активности случаја коришћења *Existing User*

На дијаграму активности на слици 5.3 приказана је функционалност случаја коришћења *Charging and discharging*. Као и претходно описан случај коришћења и овај је приказан на оба наведена дијаграма случајева коришћења. На дијаграму на слици 3.1 његова функционалност обухвата задуживање и раздуживање локалних корисника, а на дијаграму на слици 5.1 та функционалност је проширена задуживањем и раздуживањем корисника из других библиотека у конзорцијуму. На дијаграму активности који описује функционалност случаја коришћења приказане су разлике у задуживању ове две врсте корисника. На дијаграму је због једноставности и јасности приказана само активност задуживања корисника, а изостављене су преостале две активности дефинисане случајем коришћења, а то су раздуживање и продужење задужења. Начин извршавања активности раздуживања и продужења задужења не разликује се суштински од начина извршавања активности задуживања па су због тога те активности изостављене са дијаграма. Задуживање корисника представљено активношћу *Charging* може се вршити на два начина: уношењем инвентарног броја или претраживањем фонда библиотеке. У зависности од тог избора извршава се један од два тока контроле. Активност *Entering acquisition number* представља уношење инвентарног броја. Задуживање претраживањем фонда је представљено активностима *Searchig pubicatons* и *Finding publication*. Приликом претраживања фонда библиотекар из резултата претраживања бира инвентарни број публикације којом задужује корисника. Задуживање изабране публикације је представљено активношћу *Charging publication*. У зависности од тога да ли се ради о задуживању локалног корисника или корисника из друге библиотеке извршава се један од два тока контроле. У случају задуживања локалног корисника подаци о задужењу се смештају у локални систем што је представљено активношћу *Storing data about charging local user*. У случају задуживања корисника из неке друге библиотеке подаци о задужењу се смештају у систем те библиотеке што је представљено активностима *Connecting to home library* и *Storing data about charging*. Тиме се обезбеђује да се подаци о задужењима корисника у било којој библиотеци налазе на једном месту одакле су доступни свим библиотекама у конзорцијуму. Поред тога део података о задужењу и кориснику се смешта и у локални

систем ради евиденције задужених публикација у библиотечком фонду што представља активност *Storing data about charging remote user*.

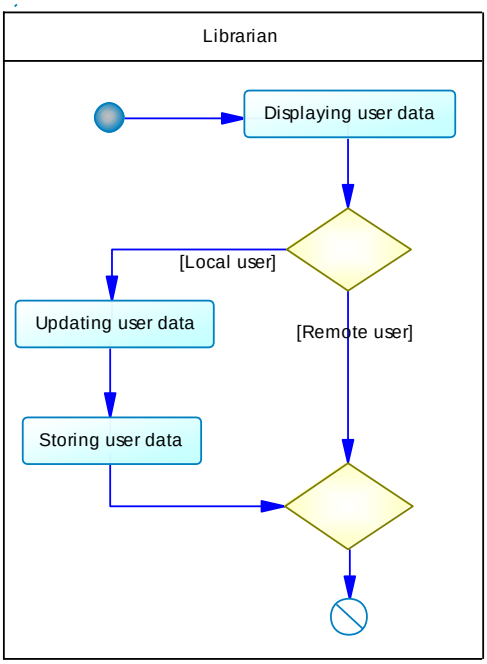


Слика 5.3 Дијаграм активности случаја коришћења *Charging and discharging*

На дијаграму активности на слици 5.4 приказана је функционалност случаја коришћења *User data.* Овај случај коришћења обухвата приказ података о кориснику. У случају када се ради о подацима локалног корисника библиотекар може да ажурира те податке и поново их сачува што је представљено активностима *Updating user data* и *Storing*

user data. У случају корисника из друге библиотеке библиотекар нема могућност измене података.

Функционалност случаја коришћења *Membership data* је иста као функционалност случаја коришћења *User data.* Разлика је само у подацима који се приказују. Због тога је дијаграм активности овог случаја коришћења изостављен.



Слика 5.4. Дијаграм активности случаја коришћења *User data*

5.3 Проширење модела података

У трећем поглављу у одељку 3.2.3 на слици 3.5 приказан је дијаграм класа модела података система за циркулацију. Овај модел података задовољава све функције система које се тичу услуживања локалних корисника библиотеке. На моделу се налазе сви подаци о кориснику који се чувају у систему, подаци о чланству корисника у библиотеку, као и подаци о задужењима корисника. Што се тиче задужене публикације моделом је предвиђено да се у задужењима корисника налази једино инвентарни број те публикације. Пошто инвентарни број јединствено одређује публикацију, на основу њега се може доћи

до осталих информација о тој публикација односно до библиотечког записа те публикације. Начин на који се из система за циркулацију долази до библиотечких записа је описан у трећем поглављу у одељку 3.2.4. Поред инвентарног броја у задужењима корисника се налази податак о томе ког дана и на којој локацији у библиотеци је задужена та публикација, рок враћања публикације, као и податак о библиотекару који је извршио задужење. Податак о датуму, локацији и библиотекару постоји и за продужење задужења и за раздужење публикације.

На дијаграму активности на слици 5.3 приказана је функционалност задуживања корисника. У случају када се ради о задуживању локалних корисника претходно описан модел задовољава све потребе за извршавање те активности. Да би се омогућило задуживање корисника из других библиотека у конзорцијуму претходно описан модел је потребно проширити. На дијаграму активности је предвиђено да се приликом задуживања корисника из других библиотека подаци о задужењу смештају у систем библиотеке којој корисник припада. На тај начин се подаци у конзорцијуму организују тако да се све информације о једном кориснику налазе на једном месту одакле су доступни другим библиотекама у конзорцијуму. Да би се ово омогућило потребно је решити неколико задатака у вези модела података.

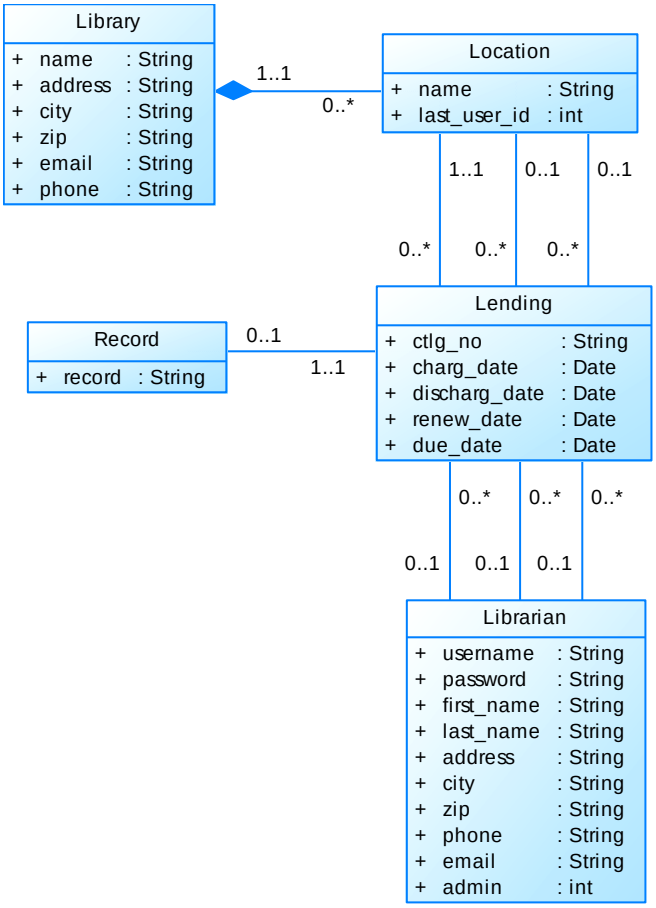
Као што је већ речено у досадашњем моделу се за свако задужење чува инвентарни број, локација, датум задужења и библиотекар. Први задатак је везан за инвентарни број. У систему у ком се чувају подаци о задужењу (систем библиотеке којој корисник припада) не постоји информација ком запису припада тај инвентарни број, односно о којој публикацији се ради, јер је та информација у систему библиотеке у којој се врши задуживање. Чак шта више, тај инвентарни број може бити додељен некој другој публикацији која припада том систему. То значи да је приликом смештања података у систем библиотеке којој корисник припада потребно у тај систем сместити и податке о самој публикацији. Овај задатак је решен тако што је модел проширен класом *Record* у којој се налази библиотечки запис о публикацији коју је корисник задужио ван библиотеке којој припада. Ово проширење је приказано на дијаграму класа на слици 5.5. Изабрана је варијанта да се у класу *Record* смешта цео запис а не само неки подаци из записа јер

систем за циркулацију већ има интерфејс за рад са библиотечким записима па у овој варијанти нема потребе за проширењем система које би радило само са неким подацима из записа.

Други задатак који је потребно решити да би се подаци о задужењу сместили у систем библиотеке којој корисник припада је везан за локацију. У досадашњем моделу класом *Location* су описане локације унутар библиотеке на којима се врши задуживање. Да би се у модел унела информација и о самој библиотеци у којој је извршено задуживање модел је проширен класом *Library* којом је описана библиотека унутар конзорцијума. Свака библиотека има локације на којима се врши задуживање и те локације се и даље описују класом *Location*. Ово проширење је такође приказано на дијаграму класа на слици 5.5. Овакво решење захтева да подаци о библиотекама у конзорцијуму и њиховим локацијама буду заједнички за све системе да би се за свако задужење једнозначно могло утврдити у којој библиотеци и на којој локацији унутар те библиотеке је извршено то задужење.

Трећи задатак који је потребно решити је информација о библиотекару који је извршио задужење. Ако би се овај задатак решавао на сличан начин као претходни, односно тако да се за свако задужење зна који библиотекар из које библиотеке у конзорцијуму је извршио то задужење, то би значило да и подаци о библиотекарима морају бити заједнички на нивоу конзорцијума. Оваква организација података у конзорцијуму не би била сврсисходна, па је задатак решен тако што се за задужења која су извршена на локацијама других библиотека не чува информација о библиотекару него се та информација чува у систему библиотеке која је извршила задужење. Начин чувања те информације ће бити објашњен у даљем тексту.

На дијаграму класа на слици 5.5 је приказано проширење модела података са слике 3.5 којим се омогућава смештање задужења и из других библиотека, односно омогућава се смештање свих задужења корисника на нивоу конзорцијума у систем библиотеке којој тај корисник припада. На слици 5.5 су приказане нове класе објашњене у претходном тексту, као и класе *Lending*, *Location* и *Librarian* са дијаграма класа са слике 3.5.

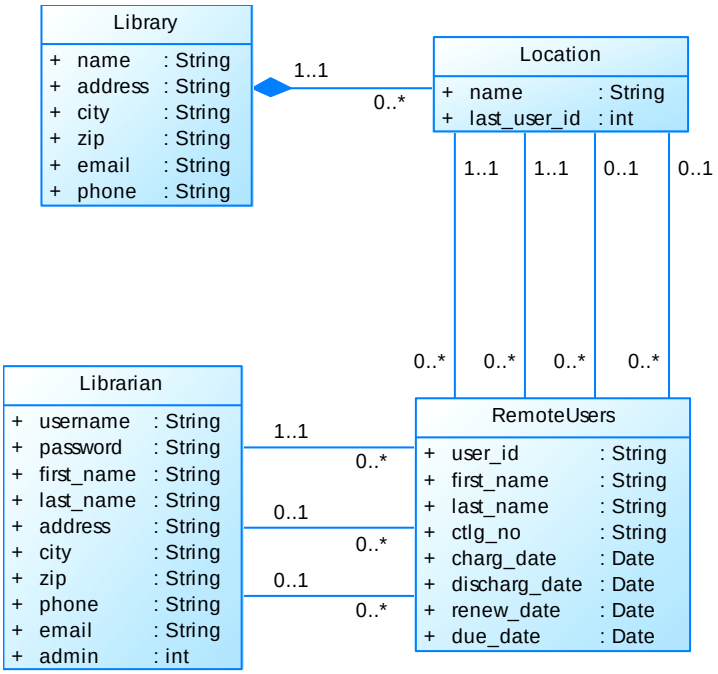


Слика 5.5 Проширење модела у сврху смештања задужења из других библиотека

Поред тога што се подаци о задужењу смештају у систем библиотеке којој корисник припада на дијаграму активности на слици 5.3 предвиђено је да се део података о задужењу и кориснику смешта и у локални систем у ком се врши задужење. Разлог за то је евиденција задужених публикација у библиотеци и информација о томе код кога је публикација задужена.

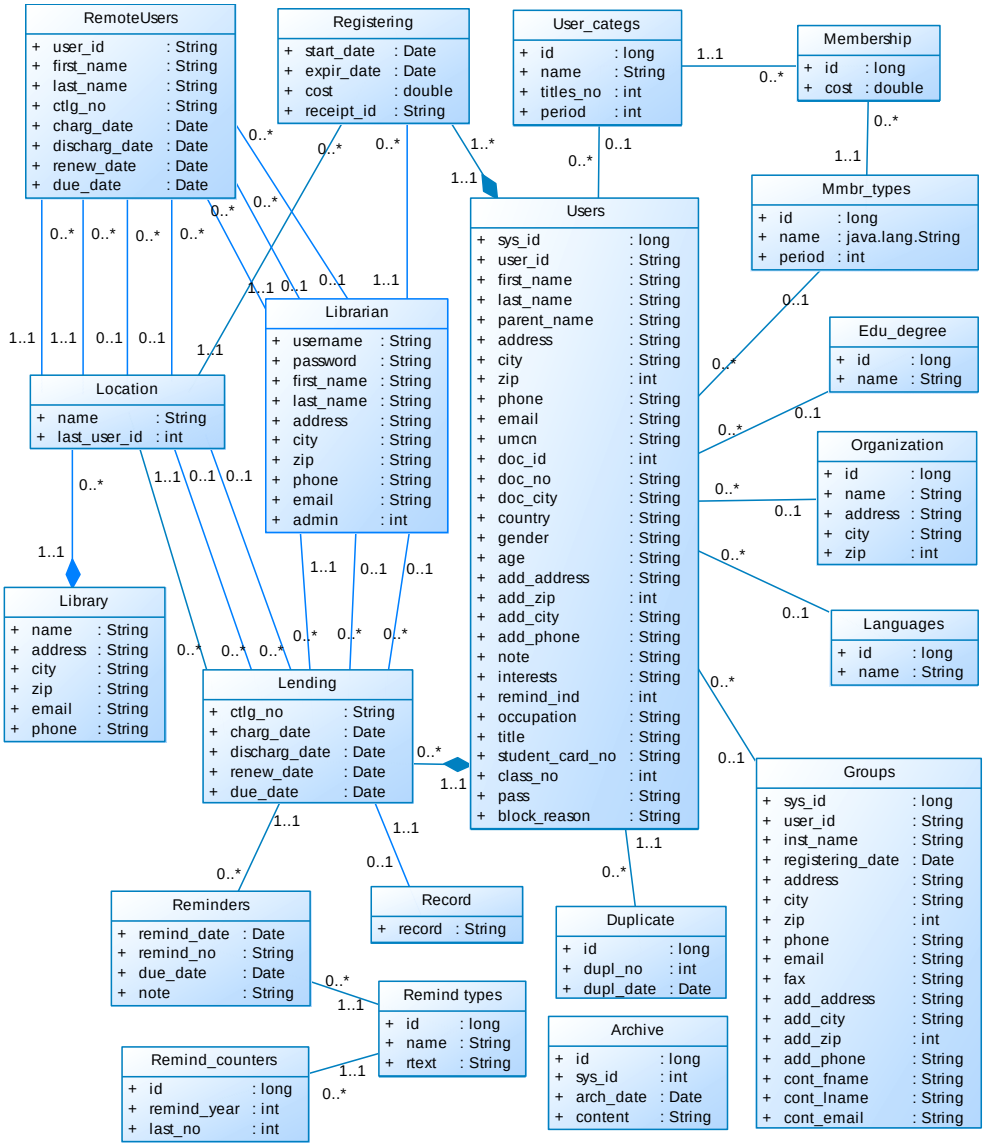
На дијаграму класа на слици 5.6 приказано је проширење модела података са слике 3.5 које омогућава смештање података о задужењу у локални систем. Класом *RemoteUsers* описано је задужење корисника из друге библиотеке. Осим података о задужењу ова класа садржи чланski број, име корисника и библиотеку којој корисник припада. На основу ових података могуће је доћи до осталих података о кориснику

који се налазе у систему корисникове библиотеке. Поред ових података за свако задужење се чува инвентарни број публикације, локација на којој је публикација задужена, датум задужења, рок враћања и библиотекар који је извршио задужење. Такође, постоји податак о датуму, локацији и библиотекару за продужење задужења, као и за раздужење публикације. Класе *Location* и *Librarian* су класе са дијаграма на слици 3.5, а класа *Library* је претходно објашњена.



Слика 5.6 Проширење модела у сврху смештања података о задужењу у локални систем

На слици 5.7 је приказан целокупан дијаграм класа модела података система за циркулацију који поред класа приказаних на слици 3.5 обухвата и претходно наведена проширења. Овакав модел података задовољава све функције система које се тичу услуживања како локалних корисника тако и корисника из других библиотека у конзорцијуму.



Слика 5.7 Модел података

5.4 Проширења имплементације система

У претходним одељцима су описана проширења функционалности система за циркулацију која обухватају процес циркулације у конзорцијуму. Такође приказана су и проширења модела података где су детаљније објашњени разлози и потребе за тим проширењима.

Начин на који су ова проширења реализована у различитим деловима система приказан је кроз имплементацију у овом одељку. Приказано је како се приступа системима других библиотека, које су нове функције система, на који начин се у конзорцијуму размењују заједнички подаци о библиотекама и њиховим локацијама, као и промене на екранским формама.

5.4.1 Имплементација нових функција система

У трећем поглављу је описана имплементација система за циркулацију. Клијентска апликација система се састоји из три дела: корисничког интерфејса, затим модела података с којим апликација ради и дела који контролише токове података од корисничког интерфејса до модела и обрнуто. На слици 3.3 у трећем поглављу је приказан и објашњен дијаграм пакета апликације.

У четвртом поглављу систем за циркулацију је проширен са подсистемом за клијент/сервер комуникацију и нови дијаграм пакета апликације је дат на слици 4.11. Подсистем за комуникацију омогућава рад система у различитим мрежним окружењима. Архитектура подсистема је базирана делом на *command* патерну, па је пословна логика апликације која користи овај подсистем имплементирана у командама које се прослеђују подсистему. Због овакве архитектуре подсистем не зависи од функционалности саме апликације и нове функционалности се додају имплементацијом нових команди. У систему за циркулацију акције које укључују комуникацију са базом података су имплементиране као команде.

Да би се имплементирала проширења описана у претходним одељцима било је потребно додати неке нове команде у систем и модификовати неке постојеће. Неке од тих команди су:

- *GetUserCommand* – модификована тако да ако за неко задужење корисника везана појава класе *Record* учитава и те податке из базе података који се касније користе при приказивању задужења корисника; представља активности *Finding local user* и део активности *Finding user* са дијаграма активности на слици 5.2.

- *GetRemoteUserCommand* – додата; користи се за учитавање података о задужењу у локалном систему за корисника из друге библиотеке у

случају када је неко задужење тог корисника задужење из локалне библиотеке; представља део активности *Finding user* са дијаграма активности на слици 5.2.

- *SaveLendingCommand* – додата; користи се за смештање података о задужењу у систем библиотеке којој корисник припада при чему се креира или модификује инстанца класе *Lending*, као и инстанца класе *Record* кроз коју се у тај систем смешта и запис коме припада задужена публикација; представља имплементацију активности *Storing data about charging* са дијаграма на слици 5.3.

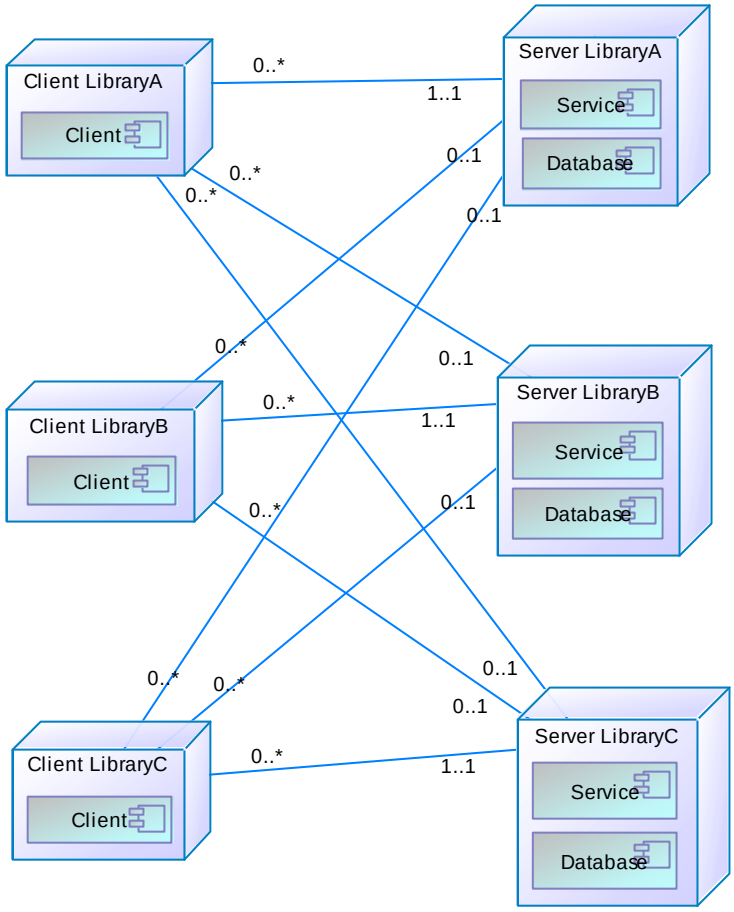
- *SaveRemoteUserCommand* – додата; користи се за смештање података о задужењу у локални систем при чему се креира или модификује инстанца класе *RemoteUser*; представља имплементацију активности *Storing data about charging remote user* са дијаграма активности на слици 5.3.

5.4.2 Приступ библиотечком сервису

Подсистем за клијент/сервер комуникацију описан у четвртом поглављу омогућава рад система у различитим мрежним окружењима. Кроз подсистем се извршавају команде које представљају делове пословне логике система. У случају система за циркулацију акције које укључују комуникацију са базом података су имплементиране као команде. Подсистем омогућава пренос тих команди кроз мрежу и извршавање на удаљеним серверима.

Због ових карактеристика подсистем је искоришћен да би се омогућила размена података између система различитих библиотека у конзорцијуму библиотечке мреже БИСИС. Свака библиотека у конзорцијуму на серверској страни свог система има инсталиран серверски део подсистема за клијент/сервер комуникацију и клијенти из других библиотека могу да му приступе. Сваки клијент има могућност креирања инстанци клијентске стране подсистема које комуницирају са серверима различитих библиотека. Ова комуникација је приказана на дијаграму размештаја на слици 5.8. На дијаграму је приказан случају када су у конзорцијуму три библиотеке: *Library1*, *Library2* и *Library3*. Свака библиотека има један сервер и више

клијената. Сваки клијент осим са својим сервером може да комуницира и са серверима других библиотека.



5.8 Конзорцијум библиотечке мреже БИСИС

Комуникација између клијента и сервера различитих библиотека подразумева слање команди на одговарајући сервер и извршавање тих команди над базом података тог сервера. На тај начин клијент добавља податке из друге библиотеке и смешта их у базу података друге библиотеке чиме се омогућава рад са корисницима из других библиотека.

Инстанца клијентске стране подсистема за комуникацију која комуницира са сервисом неке библиотеке креира се помоћу класе *CirculationServiceFactory*, на исти начин као што се креира инстанца за комуникацију са локалним сервером. Приликом креирања инстанце

класи *CirculationServiceFactory* се прослеђује адреса сервиса библиотеке којој се приступа. На листингу 5.1 је приказан начин креирања објекта за приступ библиотеци чији је сервер на адреси *serviceAddress.* Након креирања, објекту *service* могу се прослеђивати команде које ће се слати на одговарајући сервер и тамо извршавати. Креирање и коришћење овог објекта представља реализацију активности *Connecting to home library* са дијаграма активности приказаних на сликама 5.2 и 5.3.

```
CirculationService service = CirculationServiceFactory.createService(
                                ServiceType.REMOTE, serviceAddress));
```

Листинг 5.1 Креирање објекта за комуникацију
са другом библиотеком

5.4.3 Менаџер за комуникацију са другим библиотекама

Поред претходно описаних команди у пакету *Commands*, проширење је направљено и у пакету *Manager* (одељак 3.2.4) где је додата нова класа *RemoteUserManager*. Ова класа је задужена за управљање подацима корисника из других библиотека. Класа је задужена за комуникацију са системом библиотеке којој корисник припада, као и смештање одговарајућих података у локални систем. Начин комуникације са другим библиотекама је објашњен у претходном одељку. Класа *RemoteUserManager* је задужена за креирање и коришћење инстанце клијентске стране подсистема за комуникацију која комуницира са сервисом неке библиотеке (листинг 5.1). Неке од метода ове класе су:

- *createService(String serviceAddress)* – креира објекат *service* као што је приказано на листингу 5.1 који служи за комуникацију са сервисом одговарајуће библиотеке који је на адреси *serviceAddress.*

- *getUser(String userID)* – за задати чланski број корисника *userID* добавља податке о кориснику из система библиотеке којој корисник припада; креира се инстанца команде *GetUserCommand* која се кроз објекат *service* шаље на сервер одговарајуће библиотеке где се извршава и са резултатима враћа назад; резултат извршавања команде су подаци о кориснику смештени у објектима класа пакета *Model*.

- *loadUser(User userForm, Users userModel)* - из објектног модела *userModel* добијеног извршавањем команде *GetUserCommand* пребацује податке о кориснику на екранску форму за рад са корисницима коју представља параметар *userForm.*

- *saveLending(Lending lending)* – чува податке о задужењу; креира објекат класе *Record* у који смешта запис коме припада публикација која се задужује и креира инстанцу команде *SaveLendingCommand* којој прослеђује објекат *lending* и креирани објекат класе *Record*, а затим команду извршава кроз објекат *service* чиме се подаци о задужењу смештају у систем одговарајуће библиотеке; креира објекат класе *RemoteUser* који кроз креирану инстанцу команде *SaveRemoteUserCommand* смешта у локалну базу података кроз постојећи објекат класе *UserManager* који управља током података о локалним корисницима.

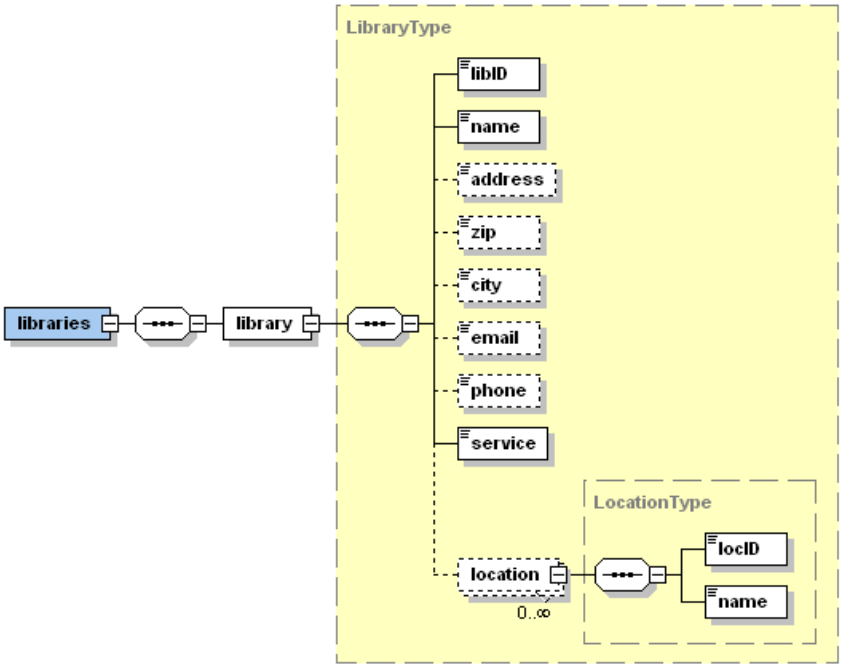
5.4.4 Подаци о библиотекама и локацијама

Као што је описано у одељку 5.3 подаци о библиотекама у конзорцијуму и њиховим локацијама треба да буду заједнички за све системе да би се за свако задужење једнозначно могло утврдити у којој библиотеци и на којој локацији унутар те библиотеке је извршено. На слици 5.5 су приказане класе *Libray* и *Location* којима су моделиране библиотеке и локације унутар њих.

Будући да подаци описани овим класама треба да буду заједнички за све библиотеке одлучено је да се подаци чувају на једном месту одакле би били доступни свим системима. На овај начин се поједностављује процес ажурирања ових података. За чување података је изабран *XML* формат као најпогоднији за размену података на мрежи. *XML* фајл са подацима о библиотекама и локацијама стоји на *HTTP* серверу на мрежи одакле је преко *HTTP* протокола доступан свим клијентима у конзорцијуму. Графички приказ *XML* шеме која описује овај фајл је дат на слици 5.9. Пример овог *XML* фајла који садржи податке о једној библиотеци са њене две локације је приказан на листингу 5.2.

Овакво централизовано решење би могло да онемогући рад конзорцијума услед недоступности сервера на ком се налази *XML* фајл. Због тога је предвиђено да се копија тог фајла приликом преузимања

сачува у оквиру локалног система, тако да у ситуацији у којој фајл не може да се преузме постоји копија фајла која може да се користи.



Слика 5.9. *XML* шема којом су описани подаци
о библиотекама и њиховим локацијама

На основу података који се налазе у овом *XML* фајлу креира се листа библиотека из које библиотекар бира библиотеку којој припада корисник. Такође у фајлу се налазе и адресе библиотечких сервиса који се користе за приступ систему одговарајуће библиотеке из класе *RemoteUserManager* која је описана у претходном одељку.

5.4.5 Проширења на екранским формама

У трећем поглављу су приказане и описане екранске форме апликације система за циркулацију. За рад са подацима корисника из других библиотека користи се иста екранска форма као за локалне кориснике. Та екранска форма садржи четири картице: *Osnovni podaci,*

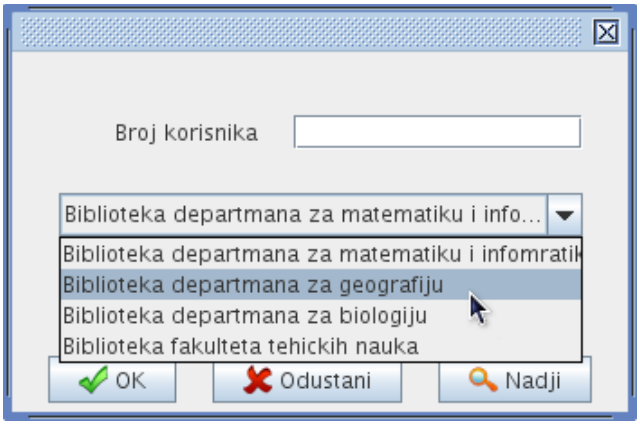
Dodatni podaci, Članarina и *Zaduženja.* Изглед екранске форме са отвореном картицом *Osnovni podaci* приказан је на слици 3.8 у трећем поглављу. У случају рада са корисницима из других библиотека није могуће мењати податке на форми. Могуће је само задуживати и раздуживати корисника. Задуживање и раздуживање тих корисника се ради на исти начин као задуживање и раздуживање локалних корисника које је објашњено у трећем поглављу.

```
<?xml version="1.0" encoding="UTF-8"?>
<libraries>
 <library>
  <libID>dmi</libID>
  <name>Biblioteka departmana za matematiku i infomratiku</name>
  <address>Trg Dostieja Obradovića 4</address>
  <zip>21000</zip>
  <city>Novi Sad</city>
  <email>goga@uns.ac.rs</email>
  <phone>021/485-2818</phone>
  <service>http://libsrv.dmi.uns.ac.rs/CircService</service>
  <location>
    <locID>dmi1</locID>
    <name>Biblioteka</name>
  </location>
  <locations>
    <locID>dmi2</locID>
    <name>Čitaonica</name>
  </locations>
 </library>
</libraries>
```

Листинг 5.2 XML фајл са подацима о библиотекама и локацијама

Да би библиотекар добио податке о кориснику потребно је претходно да унесе чланski број корисника. Да би се омогућио рад са корисницима из других библиотека поред чланског броја потребно је омогућити и избор библиотеке којој корисник припада. Због тога је

екранска форма за унос чланског броја корисника проширена са падајућом листом из које се бира библиотека којој корисник припада. На слици 5.10 приказан је изглед екранске форме за унос чланског броја корисника где се при уносу броја бира којој библиотеци корисник припада. У падајућој листи су приказана имена библиотека у конзорцијуму која су добијена из преузетог *XML* фајла који је описан у претходном одељку.



Слика 5.10 Унос чланског броја корисника

5.5 Заштита података унутар конзорцијума

У претходним одељцима је описан начин рада циркулације у конзорцијуму у оквиру библиотечке мреже БИСИС. Циркулација у конзорцијуму је реализована помоћу подсистема за клијент/сервер комуникацију који омогућава размену података између система различитих библиотека. Те библиотеке обично комуникацију остварују кроз јавну мрежу. С обзиром на то да се у процесу циркулације размењују лични подаци корисника потребно је обезбедити механизме заштите тих података како при самом преносу података тако и у приступу подацима односно библиотечким системима.

У имплементацији подсистема која је описана у четвртом поглављу нису укључене никакве мере заштите података који се размењују кроз подсистем. Комуникација између клијентског и серверског дела

подсистема остварује се кроз *HTTP* протокол који не пружа никакве механизме заштите података који се преносе. Такође при имплементацији серверске стране није укључена провера права приступа сервису. У овом одељку је дат преглед механизама заштите који би могли да се примене при употреби подсистема за клијент/сервер комуникацију. Избор механизма заштите зависи од ширег контекста у ком се користи подсистем, односно од потреба и могућности оног ко користи подсистем. У овом случају то је конзорцијум библиотека унутар ког се размењују подаци о корисницима.

У књизи (Stallings, 2007) су описани различити механизми заштите и услуге које ти механизми пружају. Стандардом *X.800* дефинисано је пет врста сервиса које механизми заштите треба да обезбеде систему. То су: аутентификација, контрола приступа, поверљивост, интегритет, непорецивост и доступност. Постоје различити приступи у пружању заштите системима који се разликују у односу на локацију њихове примене у оквиру *TCP/IP* протокол стека. Када се ради и обезбеђивању сигурне комуникације у дистрибуираним системима обично се механизми заштите обезбеђују на мрежном, транспортном или апликативном нивоу *TCP/IP* протокол стека. Сваки од ова три приступа има своје добре и лоше особине, и у различитој мери обезбеђује дефинисане сервисе. На мрежном нивоу заштита се обично обезбеђује коришћењем *IPSec* протокола. На транспортном нивоу уобичајена је употреба *SSL/TLS* протокола, док су на апликативном нивоу механизми сигурности обично имплементирани у оквиру апликација и прилагођени су специфичним потребама апликације. У даљем тексту је дат кратак опис ова три начина обезбеђивања заштите.

У зависности од потреба и могућности конзорцијума било који од ова три начина могао би да се користи у процесу циркулације у конзорцијуму. Из досадашњег искуства у раду са библиотекама закључено је да би механизме заштите било најпогодније уградити на четвртом нивоу употребом *SSL/TLS* протокола. Разлог за то је што се овај механизам заштите имплементира само у крајњим тачкама комуникације, односно само унутар клијентске и серверске стране подсистема за клијент/сервер комуникацију, при чему нема никакве потребе за администрацијом међутачака на мрежи чиме се олакшава посао администратора мреже у библиотекама. Поред тога,

имплементација овог механизма захтева само минималне измене у имплементацији подсистема за комуникацију, док се пословна логика команди не мења.

5.5.1 *IPSec*

IPSec (*IP Security*) представља проширење *IP* протокола којим се пружа заштита на мрежном нивоу *TCP/IP* протокол стека. На овај начин заштита се пружа свим протоколима и апликацијама на вишим нивоима, а не само појединачним. *IPSec* протоколи могу да буду подржани од стране приступних тачака некој мрежи (рутер, *firewall*), чиме се обезбеђује заштита комплетног саобраћаја који пролази кроз ту тачку, а такође могу бити подржани од стране појединачних радних станица. *IPSec* обухвата три области: аутентификацију, поверљивост података и управљање кључевима. Аутентификацијом се обезбеђује да примљени подаци долазе од стране оног ко се идентификује као пошиљалац. Поверљивост података се реализује шифровањем података да би се очувала њихова тајност при комуникацији. Управљање кључевима обезбеђује сигурну размену кључева у комуникацији.

IPSec укључује два протокола: *AH* (*Authentication Header*) и *ESP* (*Encapsulated Security Payload*). *AH* протокол обезбеђује сервисе за аутентификацију и интегритет података. Базиран је на принципу рачунања вредности која се уписује у *AH* заглавље и на основу које прималац може да провери аутентичност и интегритет података. *ESP* протокол обезбеђује сервис поверљивости податка. Поред ове постоји и варијанта *ESP* протокола са аутентификацијом чиме се онда добијају и сервиси аутентификације и интегритета података. *ESP* протокол користи механизме шифрирања података да би се обезбедила тајност података.

Протоколи подржавају два режима рада: преносни режим и тунеловање. У случају преносног режима рада заштита се примењује само на податке који се преносе, односно на податке протокола са виших нивоа. На заглавља пакета се не примењују механизми заштите тако да садржаји заглавља остају видљиви при транспорту. Тунеловање је режим рада у коме се механизми заштите примењују на целе пакете који се преносе, при чему се они енкапсулирају у нове пакете за

потребе преноса. Заглавља тих нових пакета којима се преносе полазни могу да буду потпуно различита од заглавља полазних пакета. На овај начин комплетан садржај пакета који се размењују остаје заштићен у транспорту.

Тунеловање је механизам који се користи за прављење виртуелних приватних мрежа (*VPN, virtual private network*). *VPN* представља логичку рачунарску мрежу која се креира унутар веће постојеће мреже с циљем да се тој логичкој рачунарској мрежи обезбеде додатни сервиси као што су сервиси заштите података. Употребом *IPSec* протокола и тунеловања креирају се *VPN* мреже које обезбеђују сигурну комуникацију и контролу приступа. Један од могућих сценарија је повезивање више локалних мрежа у једну *VPN* мрежу у којој је омогућена сигурна размена података. Овај сценарио се реализује инсталирањем *IPSec* протокола у приступним тачкама локалних мрежа. Механизмом тунеловаља обезбеђује се сигурна комуникација између приступних тачака локалних мрежа која се одвија кроз јавну мрежу, док се унутар локалних мрежа комуникација одвија без *IPSec* протокола. Могући су и сценарији у којима се рачунари као крајње тачке комуникације повезују у *VPN* мрежу и у том случају *IPSec* протокол мора да буде подржан од стране тих рачунара.

Употреба *IPSec* протокола за обезбеђивање заштите података на мрежном нивоу има неколико предности. *IPSec* је транспарентан према крајњим корисницима и апликацијама, односно укључивање механизама заштите на мрежном нивоу не захтева измене на апликацијама. Такође, *IPSec* представља једно решење опште намене где сви транспортни протоколи користе исту инфраструктуру са механизмима заштите. Обезбеђивањем сервиса заштите које користе транспортни протоколи обезбеђена је заштита и свим апликацијама. Инсталирањем *IPSec* механизама заштите на приступним тачкама некој мрежи (рутер, *firewall*) обезбеђује се заштита комплетног саобраћаја који пролази кроз ту тачку. Недостатак употребе *IPSec* протокола је већа потрошња ресурса. За операције шифровања података се троши додатно процесорско време. Такође је повећан и мрежни саобраћај због додатних заглавља пакета и већих садржаја шифрованих пакета (у одређеним случајевима).

5.5.2 SSL/TLS

SSL и *TLS* протоколи су један од начина којима се обезбеђује заштита на транспортном нивоу *TCP/IP* протокол стека. Ови протоколи пружају сервисе заштите апликацијама које користе *TCP* протокол и смештени су између *TCP* протокола и апликација. Прва верзија ових протокола која је коришћена била је *SSL v2*. Даљим развојем и отклањањем безбедоносних недостатака настала је верзија *SSL v3*. Ни једна од ових верзија није усвојена као званични стандард. Као Интернет стандард је усвојен протокол *TLS* који представља мало унапређену верзију протокола *SSL v3* и компатибилан је с њим, тако да се *TLS* протокол сматра стандардизованом верзијом протокола *SSL v3*. Ови протоколи могу бити имплементирани као додатак транспортног протокола одакле су доступни свим апликацијама или могу бити укључени у саме апликације, као у случају *web browser*-а.

SSL протокол се састоји од четири протокола који су распоређени у два слоја. Ти протоколи су: *Record, Handshake, Alert* и *Change Cipher Spec*. *SSL Record* протокол се налази на нижем слоји и ослања се на *TCP* протокол. Он обезбеђује основну сигурност протоколима виших нивоа где спадају остали *SSL* протоколи који се налазе на вишем слоју у односу на *SSL Record*, као и апликативни протоколи као што је на пример *HTTP* протокол. *SSL Record* протокол обезбеђује поверљивост и интегритет података који се размењују. Интегритет података се обезбеђује применом одређених функција на податке при чему се добија вредност која се шаље заједно са подацима. Прималац података применом исте функције треба да добије исту вредност што представља проверу интегритета добијених података. Поверљивост података се обезбеђује коришћењем механизама за шифровање података. *SSL Handshake* протокол служи за успостављање комуникације између две стране које размењују податке. Користи се пре било какве размене апликативних података и кроз њега се потврђује идентитет учесника у комуникацији и договарају се параметри комуникације као што су алгоритми за шифровање података, кључеви који се при томе користе и друго. *SSL Change Cipher Spec* протокол служи за слање порука које на крају процеса успостављања комуникације означавају почетак примене договорених алгоритама за шифровање података. *SSL Alert* протокол служи за размену порука упозорења током комуникације.

HTTPS (*Hypertext Transfer Protocol Secure*) је комбинација *HTTP* и *SSL*/*TLS* протокола чиме се обезбеђује сигурна размена података кроз *HTTP* протокол. *HTTP* протокол као апликативни протокол користи сервисе заштите *SSL*/*TLS* протокола да би се обезбедила заштита података који се даље шаљу кроз *TCP* протокол. На овај начин апликације које користе *HTTP* протокол за размену података могу да обезбеде заштиту података које размењују.

SSL/*TLS* протокол за заштиту података на транспортном нивоу представља једно опште решење којим се пружају сервиси заштите свим апликацијама и протоколима на вишем нивоу који на транспортном нивоу користе *TCP* протокол. На апликативном нивоу, у зависности од потреба апликација, бира се да ли ће поједине апликације да користе сервисе заштите или не. *SSL*/*TLS* протокол се имплементира у крајњим тачкама комуникације тако да нема потребе за имплементацијом и администрацијом у међутачкама (рутер, *firewall*). Недостатак *SSL*/*TLS* решења је у томе што његова употреба захтева да се на апликацијама које га користе направе измене којима се захтевају сервиси сигурности од транспортног нивоа.

5.5.3 Апликативни ниво

Један од приступа у обезбеђивању сервиса заштите података је уградња механизама заштите на апликативном нивоу, односно у оквиру самих апликација. Предност овог приступа је у томе што се механизми заштите могу у потпуности прилагодити специфичним потребама апликације. Такође, уградњом механизама у саму апликацију не захтева се од окружења у ком се апликација извршава обезбеђивање сервиса заштите, као што је случај са имплементацијом заштите на мрежном и транспортном нивоу. Недостатак овог приступа је што се механизми заштите уграђују посебно у сваку апликацију. То захтева веће измене постојећих апликација, а и велика је вероватноћа прављења грешака и пропуста у безбедности.

Поглавље 6

Циркулација у конзорцијуму преко *NCIP* протокола

Конзорцијум представља удружење библиотека на неком подручју које међусобно сарађују што доводи до побољшања услуга које библиотеке пружају својим корисницима. Једна од активности на нивоу конзорцијума је циркулацију у конзорцијуму. Циркулација у конзорцијуму је библиотечки процес у ком је корисницима једне библиотеке омогућено да позајмљују књиге из других библиотека унутар конзорцијума. Библиотеке чланице конзорцијума успостављају правила за коришћење библиотечких фондова. Да би се омогућила ова активност потребно је обезбедити размену података о корисницима на нивоу конзорцијума.

У претходном поглављу приказано је решење циркулације у конзорцијуму у оквиру библиотечке мреже БИСИС. То решење подразумева да све библиотеке чланице конзорцијума користе библиотечки систем БИСИС. Такође, у том решењу, користи се модел циркулације у конзорцијуму где корисник задужује публикације на локацијама где се те публикације налазе. Случај постојања посредника који би могао да обезбеди кориснику књигу са било које локације у конзорцијуму није обухваћен тим решењем.

Циркулација у конзорцијуму библиотечке мреже БИСИС представља део функционалности система за циркулацију. У систему за циркулацију комуникација између клијентске апликације и базе података реализована је кроз подсистем за клијент/сервер комуникацију чиме је омогућен рад система у различитим мрежним окружењима. Архитектура подсистема за комуникацију је базирана делом на *command* патерну, па је пословна логика апликације која

користи овај подсистем имплементирана у командама које се прослеђују подсистему. У систему за циркулацију акције које укључују комуникацију са базом података су имплементиране као команде. Подсистем за комуникацију омогућава локално извршавање команди и пренос тих команди кроз мрежу и извршавање на удаљеним серверима.

Због могућности извршавања команди на удаљеним серверима, подсистем за клијент/сервер комуникацију је искоришћен да би се омогућила размена података између система различитих библиотека у конзорцијуму библиотечке мреже БИСИС. Свака библиотека у конзорцијуму на серверској страни свог система има инсталиран серверски део подсистема за комуникацију и клијенти из других библиотека могу да му приступе. Клијенти креирају инстанце клијентске стране подсистема које комуницирају са серверима различитих библиотека. Комуникација између клијента и сервера различитих библиотека подразумева слање команди на одговарајући сервер и извршавање тих команди над базом података тог сервера. На тај начин клијент добавља податке из друге библиотеке и/или их смешта у базу података друге библиотеке. Оваквим принципом размене података реализована је циркулација у конзорцијуму библиотечке мреже БИСИС.

Као што је већ речено ово решење подразумева да све библиотеке чланице конзорцијума користе библиотечки систем БИСИС, јер је размена података везана за имплементацију система. Да би се обезбедила интероперабилност са другим библиотечким системима потребно је размену података имплементирати преко стандардизованог протокола за размену података. Тиме би се омогућила циркулација у конзорцијуму и између библиотека које користе различите библиотечке системе.

Стандардизован протокол за размену података о корисницима је *NCIP* (*NISO Circulation Interchange Protocol*) или *Z39.83* протокол који је детаљно описан у другом поглављу. Протокол је стандардизован од стране *NISO* организације. По овом протоколу подаци се размењују у *XML* формату, а структура порука које се размењују је дефинисана *XML* шемом. У овом поглављу описана је имплементација *NCIP* протокола у оквиру система за циркулацију. Протокол је

имплементиран и на клијентској страни, чиме је омогућено иницирање порука ка другим системима, и на серверској страни, чиме је омогућен одговор на поруке од стране других система.

6.1 *NCIP* протокол

NCIP протокол је протокол за размену података о корисницима. Протоколом су дефинисани сервиси и поруке које се размењују између апликација да би се остварile функције потребне за задуживање и раздуживање корисника и контролисање приступа електронским ресурсима. Протокол је дефинисан тако да може бити примењен у различитим библиотечким областима, али је првенствено намењен за коришћење у три области: циркулација унутар конзорцијума, повезивање циркулације и међубиблиотечке позајмице и самоуслужно задуживање и раздуживање корисника у библиотеци.

У другом поглављу ове дисертације дата је спецификација протокола. Наведени су сервиси дефинисани протоколом и дат је кратак опис сваког сервиса. Сервис је дефинисан као пар порука који се састоји од захтева и одговора којима се размењују подаци. Укупно је дефинисано 47 сервиса.

Такође, протоколом су дефинисане и три врсте објеката: *Agency, Item* и *User. Agency* представља библиотеку или организацију која позајмљује јединице из свог фонда и пружа друге услуге корисницима. *Item* представља физички или електронски ресурс који припада колекцији агенције и који може бити позајмљен или коме корисник може имати приступ. Објекат *User* представља особу или организацију која може да користи примерке или сервисе које пружа агенција. За сваку врсту објекта дефинисан је скуп елемената које садржи.

Постоје три типа сервиса: *Lookup, Update* и *Notification*. Сваки тип сервиса се састоји од одређеног броја сервиса. Сервиси типа *Lookup* омогућавају да се добије информација о инстанцама објеката типа *Agency, Item* и *User*. Сервиси типа *Update* омогућавају једној апликацији да од друге апликације захтева креирање или модификацију (додавање или брисање) података инстанци објеката типа *Agency, Item* и *User* за које је надлежна друга апликација. Сервиси типа *Notification* омогућавају једној апликацији да обавести другу апликацију да је

дошло до креирања или модификације инстанци објеката типа *Agency, Item* и *User*.

NCIP је дефинисан као *connection-oriented* протокол што значи да се приликом комуникације две апликације отвара конекција преко које се одвија комуникација, односно размењују поруке. *NCIP* је такође дефинисан као *sessionless* протокол што значи да су сви сервиси независни један од другог, односно да су све информације потребне за извршавање једног сервиса садржане у захтеву и да се подаци добијени извршавањем претходно позваних сервиса не чувају.

За енкодинг порука изабран је *XML* као доминантан енкодинг метод у Интернет комуникацијама. Структура порука које се размењују је дефинисана *XML* шемом. Тренутна верзија *NCIP XML* шеме којом су дефинисане поруке налази се на на званичном сајту *NCIP* протокола. За сваку поруку која је дефинисана стандардом постоји један елемент у шеми и поруке које се размењују по *NCIP* протоколу морају да буду валидне у односу на ту шему. За транспорт порука дефинисано је да се користи један од три протокола: *HTTP, HTTPS* и *TCP/IP*.

С обзиром на различите намене протокола, а као подршка имплементацији протокола, направљено је више апликационих профила којима је описано како се протокол користи у различитим окружењима и који сервиси при томе морају бити подржани. Укупно постоји осам апликационих профила. У области циркулације у конзорцијуму *NCIP* протокол обезбеђује размену података о кориснику и о библиотечкој јединици између две различите апликације за циркулацију с циљем да се библиотеци омогући задуживање корисника из других библиотека. У зависности од тога да ли ће се на корисника из друге библиотеке примењивати локалне полисе за задуживање или полисе из његове библиотеке или ће о трансакцијама да води рачуна нека трећа страна направљена су четири различита апликациона профила за примену *NCIP* протокола у овој области. То су профили:

- *Item Agency Manages Transaction* (*DCB-1*)
- *User Agency Manages Transaction* (*DCB-2*)
- *Broker Application Manages Transaction* (*DCB-3*)
- *User Agency Manages Transaction with Proxy Check Out* (*DCB-4*)

6.2 Имплементација *NCIP* протокола у систему за циркулацију

Анализом постојећих апликационих профила утврђено је да је за имплементацију протокола у БИСИС систему најпогоднији апликациони профил *User Agency Manages Transaction* (*DCB-2*). Овим профилом описана је употреба *NCIP* протокола у окружењу у ком трансакцијама циркулације управља библиотека којој корисник припада, односно на корисника се не примењују полисе библиотеке у којој се врши задуживање него полисе његове матичне библиотеке. Матична библиотека одређује и ниво доступности података корисника, односно одређује којим подацима ће друге библиотеке имати приступ. Овим профилом је предвиђена директна комуникација две апликације циркулације без посредства неке треће апликације. Такође, профил обухвата размену података између библиотеке којој припада корисник и библиотеке којој припада библиотечка јединица. Друге библиотеке у конзорцијуму не добијају информације о овој комуникацији.

Поред могућности да корисник задужује публикације у другим библиотекама унутар конзорцијума овај профил обухвата и фискалне трансакције корисника на другим локацијама као што су продужење чланарине или надокнада за задужење. Такође, подржано је и враћање публикација на различитим локацијама, односно кориснику се може омогућити да врати публикације на локацији различитој од оне где их је узео. За сваку од ових функционалности профилом је дефинисано који сервиси *NCIP* протокола су обавезни, а који опциони за имплементацију. Поред тога дефинисано је који су обавезни, а који опциони елементи у порукама, као и догађаји који иницирају поруке. С обзиром да овај профил укључује сервисе и поруке којима се размењују лични подаци корисника као транспортни протокол за овај профил је изабран *HTTPS* протокол.

За остваривање функционалности задуживања корисника унутар конзорцијума са следећим особинама:

- публикације се задужују на локацији на којој се налазе (не шаљу се на друге локације),
- публикације се враћају на истој локацији где су узете,
- нема финансијске надокнаде за задужење,

анализом профила закључено је да је потребно имплементирати 6

сервиса. У табели 6.1 приказани су ови сервиси, као и догађаји који захтевају од апликације да иницира одговарајући сервис и улоге апликација током извршавања тог сервиса. Овакав модел циркулације у конзорцијуму прихваћен је за БИСИС систем и имплементирано је приказаних 6 сервиса.

Табела 6.1 Сервиси имплементирани у систему БИСИС

Догађај	**Апликација која иницира**	**Сервис**	**Апликација која одговара**
Корисник долази у библиотеку да задужи публикацију	*Item Agency*	*Lookup User*	*User Agency*
Корисник задужује публикацију	*Item Agency*	*Check Out Item*	*User Agency*
Корисник продужава задужење	*User Agency*	*Item Renewed*	*Item Agency*
Корисник продужава задужење	*Item Agency*	*Renew Item*	*User Agency*
Корисник раздужује публикацију	*Item Agency*	*Check Item In*	*User Agency*
Библиотека жели податке о публикацији	*User Agency*	*Lookup Item*	*Item Agency*
Библиотека жели податке о кориснику	*Item Agency*	*Lookup User*	*User Agency*

Приликом дефинисања протокола водило се рачуна о различитој пракси библиотека у области циркулације. Те различитости су у контексту размене података већином обухваћене коришћењем различитих вредности за поједине елементе у подацима о корисницима или публикацијама. Зато је протокол дефинисан тако да буде проширив у односу на потребе библиотеке или конзорцијума. Библиотекама је остављено да за одређене елементе објеката *Item, User*

и *Agency* дефинишу листе вредности које се могу наћи у тим елементима. Те листе су доступне библиотекама које учествују у размени и јединствено су одређене *URI* адресом. Такође је библиотекама остављена могућност да додају и нове елементе објектима у односу на оне дефинисане протоколом.

За потребе имплементације *NCIP* протокола у оквиру БИСИС система било је потребно дефинисати неколико листа вредности. Те листе су дефинисане у односу на шифарнике који постоје у моделу података.

Сервиси дефинисани *NCIP* протоколом се састоје од две поруке, захтева и одговора, којима се размењују подаци. Поруке се размењују у *XML* формату, а синтакса порука дефинисана је *NCIP XML* шемом. За сваку поруку која је дефинисана стандардом постоји један елемент у шеми и поруке које се размењују по *NCIP* стандарду морају да буду валидне у односу на ту шему.

На примеру сервиса *Lookup User* биће илустрован начин имплементације *NCIP* протокола у оквиру система БИСИС. Приказан је изглед и садржај порука које се размењују у оквиру овог сервиса. Овај сервис се иницира од стране библиотеке која није корисникова матична библиотека, а са циљем да се добију подаци о кориснику.

Захтев порука која се шаље овим сервисом дефинисана је елементом *NCIP XML* шеме *LookupUser.* Овај елемент је приказан на листингу 6.1. Елемент `LookupUser` је комплексног типа и представља секвенцу елемената комплексног типа. Први елемент који се појављује у секвенци је `InitiationHeader` који предстаља заглавље поруке и у њему се наводи која библиотека шаље захтев и коме се шаље захтев. Ово заглавље се појављује у свим захтев порукама. Након заглавља налази се један од два елемента за идентификацију корисника чији подаци се захтевају: `UserId` или `AuthenticationInput`. Након тога наводе се елементи типа `UserElementType` којима се специфицира који подаци се захтевају. Затим се наводе елементи `LoanedItemsDesired`, `RequestedItemsDesired` и `UserFiscalAccountDesired` у зависности од тога да ли се захтевају подаци о задужењима корисника, резервацијама и фискалним трансакцијама корисника. Елемент `Ext` који може да се наведе након свих претходних елемената служи као проширење протокола за размену података који нису обухваћени протоколом.

```
<xs:element name="LookupUser">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="InitiationHeader" minOccurs="0"/>
   <xs:choice>
    <xs:element ref="UserId"/>
    <xs:element ref="AuthenticationInput"
                          maxOccurs="unbounded"/>
   </xs:choice>
   <xs:element ref="UserElementType" minOccurs="0"
                           maxOccurs="unbounded"/>
   <xs:element ref="LoanedItemsDesired" minOccurs="0"/>
   <xs:element ref="RequestedItemsDesired" minOccurs="0"/>
   <xs:element ref="UserFiscalAccountDesired"
                          minOccurs="0"/>
   <xs:element ref="Ext" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Листинг 6.1 Дефиниција поруке LookupUser

На листингу 6.2 је приказан *XML* документ који представља захтев поруку сервиса *Lookup User* и креиран је на основу претходно описаног елемента *NCIP XML* шеме. У *XML* документу се може уочити структура и садржај описане секвенце елемената. Елемент `InitiationHeader` дефинисан је као секвенца елемената и у овом случају има два елемента: `FromAgencyId` и `ToAgencyId` у којима се налази информација која библиотека шаље захтев и коме се шаље захтев. Оба ова елемента у себи садрже елемент `AgencyId` у коме се налази информација о конкретној библиотеци. Елемент `AgencyId` је типа `SchemeValuePair`. Овај тип је дефинисан за потребе прилагођавања протокола локалним праксама библиотека. Елементи овог типа имају атрибут `Scheme` где се наводи *URI* адреса листе вредности из које се узимају вредности елемента. У случају елемента `AgencyId`, који је овог типа, и система БИСИС, на адреси `http://bisis.ns.ac.rs/NCIPschemes/AgencyID.scm` налази се листа вредности скраћених назива библиотека које су чланице конзорцијума и које размењују поруке. У приказаном примеру на листингу уочава се да поруку шаље библиотека за ознаком `ftnns`, а прима је библиотека са ознаком `dmins`.

Након елемента `InitiationHeader` налази се елемент `UserId` на основу ког се идентификује корисник чији подаци се траже. Овај елемент

представља секвенцу елемената AgencyId, UserIdentifierType и UserIdentifierValue. У овом случају елемент AgencyId одређује којој библиотеци корисник припада. Елемент UserIdentifierType дефинише који податак се користи за идентификацију корисника. Овај елемент је типа SchemeValuePair и има атрибут Scheme где се наводи *URI* адреса листе вредности у којој су дефинисани подаци који могу да се користе за идентификацију корисника. У приказаном примеру то је податак Patron ID који представља број корисника. Елемент UserIdentifierValue представља конкретну вредност податка дефинисаног у елементу UserIdentifierType, у овом случају број конкретног корисника. Након елемента UserId наводе се елементи UserElementType којима се дефинише који подаци о кориснику се траже. Елемент UserElementType је такође типа SchemeValuePair и вредности ових елемената се узимају из дефинисане листе вредности. У листи вредности за овај елемент се налазе елементи објекта *User* дефинисаног протоколом. Након елемената UserElementType у приказаном примеру наведен је и елемент LoanedItemsDesired који означава да се захтевају и подаци о задужењима корисника.

```
<?xml version="1.0" encoding="UTF-8"?>
<ncip:LookupUser xmlns:ncip="http://www.niso.org/2008/ncip"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.niso.org/2008/ncip
       ncip_v2_0.xsd ">
 <ncip:InitiationHeader>
  <ncip:FromAgencyId>
   <ncip:AgencyId ncip:Scheme="http://bisis.uns.ac.rs/
                               NCIPschemes/AgencyID.scm">
      ftnns
   </ncip:AgencyId>
  </ncip:FromAgencyId>
  <ncip:ToAgencyId>
   <ncip:AgencyId ncip:Scheme="http://bisis.uns.ac.rs/
                               NCIPschemes/AgencyID.scm">
      dmins
   </ncip:AgencyId>
  </ncip:ToAgencyId>
 </ncip:InitiationHeader>
 <ncip:UserId>
  <ncip:AgencyId ncip:Scheme="http://bisis.uns.ac.rs/
                              NCIPschemes/AgencyID.scm">
     dmins
  </ncip:AgencyId>
```

```
  <ncip:UserIdentifierType ncip:Scheme="http://bisis.uns.ac.
                      rs/NCIPschemes/UserIdentifierType.scm">
     Patron ID
  </ncip:UserIdentifierType>
  <ncip:UserIdentifierValue>
     00000001369
  </ncip:UserIdentifierValue>
 </ncip:UserId>
 <ncip:UserElementType ncip:Scheme="http://bisis.uns.ac.rs/
                           NCIPschemes/UserElementType.scm">
     Name Information
 </ncip:UserElementType>
 <ncip:UserElementType ncip:Scheme="http://bisis.uns.ac.rs/
                           NCIPschemes/UserElementType.scm">
     User Address Information
 </ncip:UserElementType>
 <ncip:UserElementType ncip:Scheme="http://bisis.uns.ac.rs/
                           NCIPschemes/UserElementType.scm">
     User Language
 </ncip:UserElementType>
 <ncip:UserElementType ncip:Scheme="http://bisis.uns.ac.rs/
                           NCIPschemes/UserElementType.scm">
     User Privilege
 </ncip:UserElementType>
 <ncip:UserElementType ncip:Scheme="http://bisis.uns.ac.rs/
                           NCIPschemes/UserElementType.scm">
     Block Or Trap
 </ncip:UserElementType>
 <ncip:LoanedItemsDesired/>
</ncip:LookupUser>
```

Листинг 6.2 Порука LookupUser

Као одговор на ову поруку у оквиру сервиса *Lookup User* креира се порука дефинисана елементом *NCIP XML* шеме *LookupUserResponse*. Овај елемент је приказан на листингу 6.3. Елемент је комплексног типа и представља секвенцу елемената. Први елемент у секвенци је елемент `ResponseHeader` који представља заглавље поруке и дефинисан је исто као елемент `InitiationHeader` у захтев поруци, што значи да садржи информацију о томе ко шаље и ко прима поруку. Ово заглавље се налази у свим одговор порукама. Након заглавља у поруци се наводи или елемент `Problem` или секвенца елемената са траженим подацима о кориснику. Елемент `Problem` се наводи у случају када систем не може да одговори на захтев и у оквиру њега се наводе разлози за то. У случају да систем може да одговори на захтев у поруци се наводи секвенца елемената са подацима. Први у секвенци се наводи елемент

`UserId` који је дефинисан на исти начин као у претходно описаној захтев поруци *LookupUser.* Затим, у случају да су захтевани подаци о фискалним трансакцијама корисника, наводи се елемент `UserFiscalAccount` са тим подацима. У случају да су захтевани подаци о задужењима корисника, наводе се елементи `LoanedItemsCount` и `LoanedItem` који садрже податке о броју задужених публикација и податке о појединачним задужењима. Такође, ако су захтевани подаци о резервацијама наводе се елементи `RequestedItemsCount` и `RequestedItem` који садрже број резервисаних публикација и податке о појединачној резервацији. Након тога наводи се елемент `UserOptionalFields` чији садржај зависи од тога шта је од података о кориснику тражено. Такође, елемент `Ext` може да се наведе након свих претходних елемената као проширење протокола за размену података који нису обухваћени протоколом.

```
<xs:element name="LookupUserResponse">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="ResponseHeader" minOccurs="0"/>
   <xs:choice>
    <xs:element ref="Problem" maxOccurs="unbounded"/>
    <xs:sequence>
     <xs:element ref="UserId"/>
     <xs:element ref="UserFiscalAccount" minOccurs="0"
                                maxOccurs="unbounded"/>
     <xs:element ref="LoanedItemsCount" minOccurs="0"
                                maxOccurs="unbounded"/>
     <xs:element ref="LoanedItem" minOccurs="0"
                                maxOccurs="unbounded"/>
     <xs:element ref="RequestedItemsCount" minOccurs="0"
                                maxOccurs="unbounded"/>
     <xs:element ref="RequestedItem" minOccurs="0"
                                maxOccurs="unbounded"/>
     <xs:element ref="UserOptionalFields" minOccurs="0"/>
    </xs:sequence>
   </xs:choice>
   <xs:element ref="Ext" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Листинг 6.3 Дефиниција поруке LookupUserResponse

На основу претходно описаног елемента шеме креира се *XML* документ који представља поруку *LookupUserResponse* и пример једне

такве поруке је приказан на листингу 6.4. Ова порука представља одговор на поруку приказану на листингу 6.2. Због обимности неки делови поруке су изостављени. Заглавље поруке `ResponseHeader` има исту структуру као заглавље захтев поруке `InitiationHeader` приказано на листингу 6.2 и из тог разлога није приказано. Након заглавља у поруци се наводи елемент `UserId` који је исти као у већ приказаној захтев поруци. Будући да су у захтеву тражени подаци о задужењима корисника у поруци се налазе елементи `LoanedItemsCount` и `LoanedItem`. Елемент `LoanedItemsCount` садржи податке о броју задужених публикација и представља секвенцу елемената `ItemUseRestrictionType` и `LoanedItemCountValue`. Елемент `ItemUseRestrictionType` представља врсту задужења и типа је `SchemeValuePair` што значи да узима вредност из дефинисане листе вредности на датој *URI* адреси. Елемент `LoanedItemCountValue` садржи податак о броју задужених публикација одговарајуће врсте задужења. Након елемента `LoanedItemsCount` за свако задужење се наводи по један елемент `LoanedItem` који садржи податке о појединачним задужењима. У приказаном примеру то су два задужења. `LoanedItem` представља секвенцу елемената и први елемент у тој секвенци је `ItemId` на основу ког се идентификује публикација. Елемент `ItemId` садржи елементе `AgencyId`, `ItemIdentifierType` и `ItemIdentifierValue`. На основу елемента `AgencyId` идентификује се библиотека којој публикација припада, као и у претходно приказаним употребама овог елемента. Елемент `ItemIdentifierType` дефинише који податак се користи за идентификацију публикације и типа је `SchemeValuePair`. У приказаном примеру то је податак `accession number` који представља инвентарни број публикације. Елемент `ItemIdentifierValue` садржи конкретну вредност инвентарног броја задужене публикације. Након елемента `ItemId` наводе се елементи `ReminderLevel`, `DateDue`, `Amount` и `Title` који садрже податке о броју послатих опомена за дату публикацију, року враћања публикације, финансијској надокнади за задужење и наслову публикације. После података о задужењима у поруци се наводе подаци о кориснику. Садржај тих података зависи од тога шта је тражено у захтев поруци. У захтев поруци приказаној на листингу 6.2 тражени су следећи подаци: `Name Information`, `User Address Information`, `User Language`, `User Privilege` и `Block Or Trap`. За сваки овај податак у одговор поруци се наводи истоимени елемент у оквиру елемената

`UserOptionalFields` са одговарајућим садржајем. На пример, подаци о имену се налазе у елементу `NameInformation` где је име дефинисано као *лично име* елементом `PersonalNameInformation` и подаци о имену су дати структуирано у елементу `StructuredPersonalUserName`. На сличан начин су наведени и подаци о адреси корисника, матерњем језику, привилегијама корисника у матичној библиотеци и блокирању корисничког налога, што се види у поруци приказаној на *листингу* 6.4.

```
<?xml version="1.0" encoding="UTF-8"?>
<ncip:LookupUserResponse .........  ">
 <ncip:ResponseHeader>
    ..................
 </ncip:ResponseHeader>
 <ncip:UserId>
    ............
 </ncip:UserId>
 <ncip:LoanedItemsCount>
  <ncip:ItemUseRestrictionType ncip:Scheme="http://bisis.
        uns.ac.rs/NCIPschemes/ItemUseRestrictionType.scm">
    normal loan period
  </ncip:ItemUseRestrictionType>
  <ncip:LoanedItemCountValue>2</ncip:LoanedItemCountValue>
 </ncip:LoanedItemsCount>
 <ncip:LoanedItem>
  <ncip:ItemId>
   <ncip:AgencyId ......>
   <ncip:ItemIdentifierType ncip:Scheme="http://bisis.uns.
              ac.rs/NCIPschemes/UserIdentifierType.scm">
     accession number
   </ncip:ItemIdentifierType>
   <ncip:ItemIdentifierValue>
       00000023310
   </ncip:ItemIdentifierValue>
  </ncip:ItemId>
  <ncip:ReminderLevel>1</ncip:ReminderLevel>
  <ncip:DateDue>2010-03-27T17:49:50</ncip:DateDue>
  <ncip:Amount>
   <ncip:CurrencyCode>none</ncip:CurrencyCode>
   <ncip:MonetaryValue>0</ncip:MonetaryValue>
  </ncip:Amount>
  <ncip:Title>
      Network security essentials
  </ncip:Title>
 </ncip:LoanedItem>
 <ncip:LoanedItem>
  <ncip:ItemId>
```

```
   <ncip:AgencyId ......>
   <ncip:ItemIdentifierType ncip:Scheme="http://bisis.uns.
                ac.rs/NCIPschemes/UserIdentifierType.scm">
    accession number
   </ncip:ItemIdentifierType>
   <ncip:ItemIdentifierValue>
      00000023311
   </ncip:ItemIdentifierValue>
  </ncip:ItemId>
  <ncip:ReminderLevel>1</ncip:ReminderLevel>
  <ncip:DateDue>2010-04-08T12:15:04</ncip:DateDue>
  <ncip:Amount>
   <ncip:CurrencyCode>none</ncip:CurrencyCode>
   <ncip:MonetaryValue>0</ncip:MonetaryValue>
  </ncip:Amount>
  <ncip:Title>
     Data and computer communications
  </ncip:Title>
 </ncip:LoanedItem>
 <ncip:UserOptionalFields>
  <ncip:NameInformation>
   <ncip:PersonalNameInformation>
    <ncip:StructuredPersonalUserName>
     <ncip:GivenName>Danijela</ncip:GivenName>
     <ncip:Surname>Tešendić</ncip:Surname>
    </ncip:StructuredPersonalUserName>
   </ncip:PersonalNameInformation>
  </ncip:NameInformation>
  <ncip:UserAddressInformation>
   <ncip:UserAddressRoleType ncip:Scheme="http://bisis.uns.
                ac.rs/NCIPschemes/UserAddressRoleType.scm">
      home
   </ncip:UserAddressRoleType>
   <ncip:PhysicalAddress>
    <ncip:StructuredAddress>
     <ncip:Street>Ilije Bircanina 29</ncip:Street>
     <ncip:Locality>Novi Sad</ncip:Locality>
     <ncip:PostalCode>21000</ncip:PostalCode>
    </ncip:StructuredAddress>
    <ncip:PhysicalAddressType ncip:Scheme="http://bisis.uns.
                ac.rs/NCIPschemes/PhysicalAddressType.scm">
      street address
    </ncip:PhysicalAddressType>
   </ncip:PhysicalAddress>
  </ncip:UserAddressInformation>
  <ncip:UserLanguage ncip:Scheme="http://bisis.uns.
                   ac.rs/NCIPschemes/UserLanguage.scm">
      scr
  </ncip:UserLanguage>
  <ncip:UserPrivilege>
```

```
    <ncip:AgencyId ..........>
    <ncip:AgencyUserPrivilegeType ncip:Scheme="http://bisis.
          ns.ac.rs/NCIPschemes/AgencyUserPrivilegeType.scm">
        assistant
    </ncip:AgencyUserPrivilegeType>
   </ncip:UserPrivilege>
   <ncip:BlockOrTrap>
    <ncip:AgencyId ...........>
    <ncip:BlockOrTrapType>
       none
    </ncip:BlockOrTrapType>
   </ncip:BlockOrTrap>
 </ncip:UserOptionalFields>
</ncip:LookupUserResponse>
```

Листинг 6.4 Порука *LookupUserResponse*

6.3 Проширење клијентске стране система за циркулацију

Да би се библиотеци која користи систем БИСИС омогућило да иницира поруке и прима одговоре по *NCIP* протоколу ка другим системима потребно је проширити клијентску страну система за циркулацију. Клијентска страна, како са својим сервером, тако и са серверима других библиотека, комуницира преко подсистема за клијент/сервер комуникацију. Подсистем омогућава различите начине комуникације са серверима. Задатак клијентске апликације је да креира и иницира команде са пословном логиком, а за начин извршавања команди брине се подсистем за комуникацију. У складу са оваквом архитектуром имплементација *NCIP* протокола треба да буде смештена у подсистем за клијент/сервер комуникацију. Подсистем за комуникацију је детаљно описан у четвртом поглављу, а начини комуникације са другим библиотекама у петом.

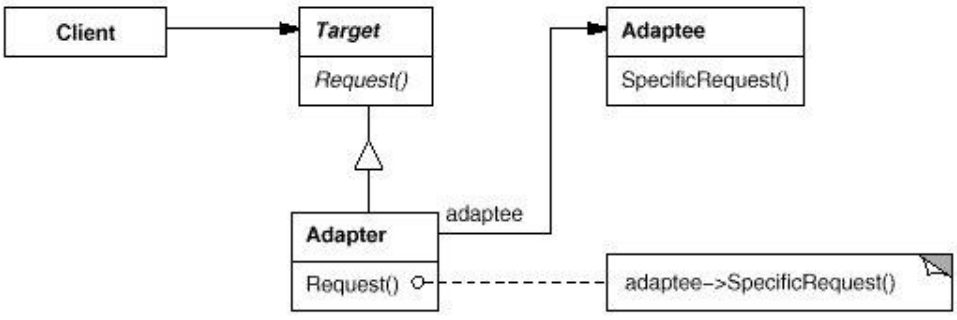
Клијентска апликација подсистему за комуникацију прослеђује команде које подсистем по извршавању враћа апликацији. У командама су енкапсулирани подаци потребни за извршавање команди као и резултати тог извршавања. У случају система за циркулацију командама су имплементиране акције којима се добављају подаци из базе података или смештају у њу. Да би се те акције са базом података трансформисале у комуникацију преко *NCIP* протокола са неким другим библиотечким системом потребно је податке из команди трансформисати у одговарајуће *NCIP* поруке, као

и податке из порука поново енкапсулирати у команде као резултате извршавања. За ову трансформацију је изабран *adapter* патерн.

6.3.1 *Adapter* патерн

Adapter патерн спада у групу *structural* патерна. *Structural* патерни описују различите начине повезивања класа и објеката да би се формирале веће структуре. *Adapter* патерн омогућава повезивање интерфејса две класе који нису компатибилни. Користи се за коверзију интерфејса једне класе у интерфејс друге који клијент очекује. Постоје две врсте *adapter* патерна: *class adapter* и *object adapter*. *Class adapter* користи вишеструко наслеђивање да би прилагодио интерфејс једне класе другом интерфејсу. *Object adapter* користи композицију објеката за конверзију једног интерфејса у други.

На дијаграму класа на слици 6.1 приказана је структура *object adapter* патерна (дијаграм преузет из (Gamma et al., 1994)). *Target* представља интерфејс који клијент очекује. *Adaptee* је класа коју је потребно прилагодити клијенту на месту где очекује интерфејс *Target*. *Adapter* представља класу која прилагођава, или адаптира, класу *Adaptee* на интерфејс *Target*. Како се ради о *object adapter* патерну класа *Adapter* имплементира интерфејс *Target* и има референцу на објекат класе *Adaptee.* Клијент тако користи инстанцу класе *Adapter* на месту где очекује интерфејс *Target*, а класа *Adapter* даље позива методе објекта класе *Adaptee* да би реализовала одговарајући захтев.



Слика 6.1 Дијаграм класа *object adapter* патерна

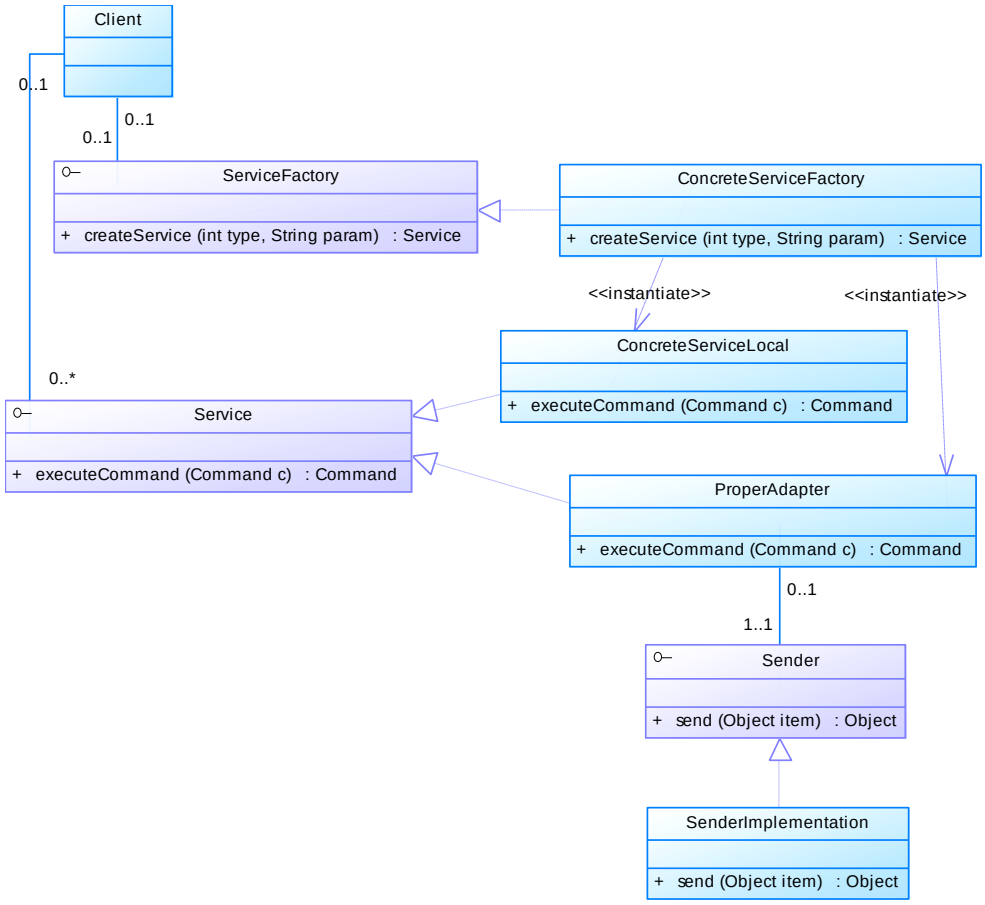
6.3.2 Проширење клијентске стране подсистема за клијент/сервер комуникацију

Патерн оријентисана софтверска архитектура подсистема за клијент/сервер комуникацију, приказана у одељку 4.1.3, заснована је на комбинацији три дизајн патерна, и то *command* патерна, *apstract factory* патерна и *factory method* патерна. *Command* патерн је искоришћен да би се пословна логика енкапсулирала у објекте, који даље могу на различите начине да се шаљу на одговарајуће локације и извршавају. Коришћењем *factory* патерна добила се апстракција креирања и коришћења објеката задужених за различите начине комуникације са сервером. На слици 4.6 приказан је начин имплементације акција пословне логике у објекте, односно команде, по принципу *command* патерна. На слици 4.7 приказана је софтверска архитектура клијентске стране подсистема за клијент/сервер комуникацију. Клијентска апликација представљена класом *Client* користи подсистем преко два интерфејса: *Service* и *ServiceFactory*. Интерфејс *Service* и његове имплементације служе за извршавање команди. Различите имплементације интерфејса третирају команде на различите начине (извршавају их локално или шаљу на удаљени сервер при чему могу да постоје различите имплементације за слање кроз различите протоколе). Интерфејс *ServiceFactory* служи за креирање одговарајуће инстанце једне од имплементација интерфејса *Service.*

Класа *ServiceBroker* је имплементација интерфејса *Service* која се користи у случају када је команду потребно извршити на неком удаљеном серверу на Интернету и тада се команда шаље серверској страни подсистема за комуникацију који даље извршава команду. За имплементацију комуникације између клијента и сервера је коришћен програмски пакет *Hessian.* Класа *ServiceBroker* креира *proxy* објекат преко кога комуницира са сервером, односно кроз који команде добијене од клијента шаље серверу.

Да би се омогућила комуникација преко *NCIP* протокола потребно је направити имплементацију сличну класи *ServiceBroker* која би команду добијену од клијента конвертовала у *NCIP* поруку и проследила на одговарајући сервер кроз *HTTPS* протокол. За ове потребе архитектура клијентске стране подсистема за комуникацију је

модификована и проширена. У овим изменама је искоришћена архитектура *adapter* патерна.



Слика 6.2 Дијаграм класа клијентске стране подсистема за клијент/сервер комуникацију

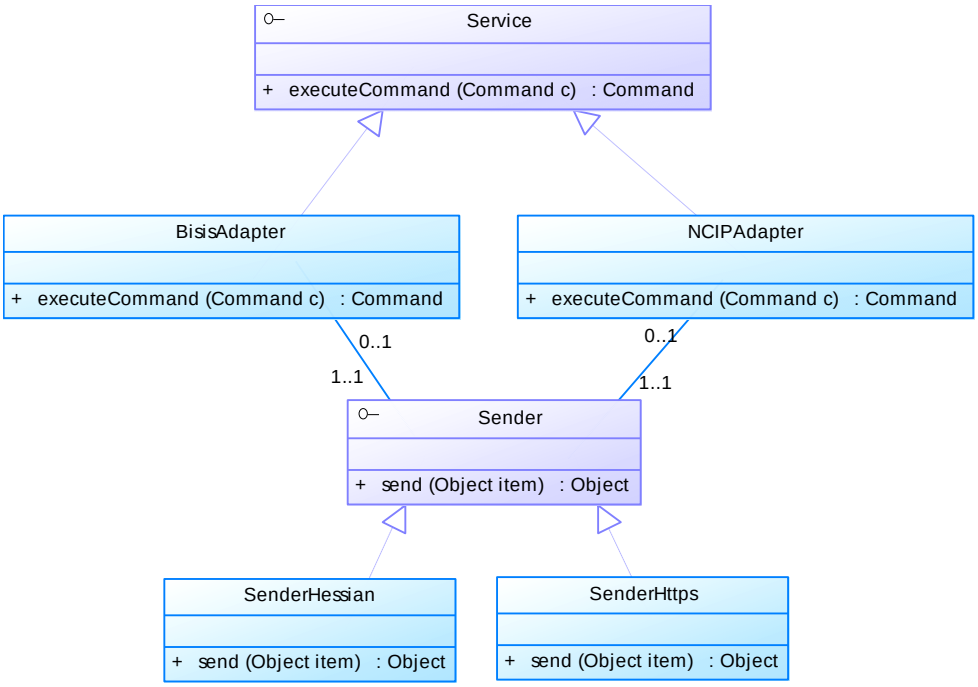
На слици 6.2 приказан је дијаграм класа клијентске стране подсистема за клијент/сервер комуникацију. Функционалности интерфејса и класа *Client*, *Service*, *ServiceFactory*, *ConcreteServiceFactory* и *ConcreteServiceLocal* непромењене су у односу на опис подсистема дат у четвртом поглављу и на слици 4.7. Класа *ServiceBroker* је замењена комбинацијом класа и интерфејса *ProperAdapter*, *Sender* и *SenderImplementation* које обухватају и функционалности класе *ServiceBroker* и функционалности потребне за комуникацију преко *NCIP* протокола. Интерфејсом *Sender* је

дефинисано слање објеката на удаљене сервере. Интерфејс има један метод *send()* коме се као параметар прослеђује објекат који се шаље. Различите имплементације овог интерфејса користе различите протоколе за слање. Те имплементације су на дијаграму илустративно представљене једном класом *SenderImplementation*. Класа *ProperAdapter* има улогу класе *Adapter* приказане у архитектури *adapter* патерна. Класа *ProperAdapter* је адаптер који прилагођава интерфејс *Sender* на интерфејс *Service*. Ова класа представља имплементацију интерфејса *Service* коју клијент користи за комуникацију са удаљеним серверима. Клијент прослеђује адаптеру команду коју је потребно извршити, коју он даље конвертује у одговарајући формат и преко референце на одговарајућу имплементацију интерфејса *Sender* шаље на удаљени сервер. У зависности од формата који се шаље постоје различите имплементације адаптера. Класом *ProperAdapter* на дијаграму су илустративно представљене све имплементације, односно сви адаптери.

Као што је већ речено постоје различите имплементације интерфејса *Sender* које су на дијаграму на слици 6.2 представљене класом *SenderImplementation*, као и различите имплементације класе *ProperAdapter*. Имплементације *SenderImplementation* користе различите протоколе за слање, а имплементације *ProperAdapter* конвертују команду у различите формате за слање. У овој итерацији развоја система за циркулацију овим класама је потребно имплементирати функционалности класе *ServiceBroker* из претходне итерације, као и функционалности за комуникацију преко *NCIP* протокола. Због тога су у имплементацији подсистема направљене по две имплементације сваке класе. Ове имплементације су приказане на дијаграму класа на слици 6.3.

Класа *ServiceBroker* добијену команду прослеђује кроз *Hessian* протокол серверској страни подсистема која даље ту команду извршава. У ту сврху класа *ServiceBroker* креира *proxy* објекат преко кога комуницира са сервером, односно кроз који шаље команде. Наведене функционалности класе *ServiceBroker* су у овој итерацији развоја система имплементиране у две класе: *BisisAdapter* и *SenderHessian*. Класа *SenderHessian* омогућава комуникацију са одговарајућим сервером кроз *Hessian* протокол, односно креира *proxy* објекат преко кога комуницира са сервером. Класа *BisisAdapter* добија команде од

клијента и прослеђује их класи *SenderHessian* која их даље шаље на сервер.



Слика 6.3 Два начина комуникације са удаљеним серверима

Функционалности за комуникацију преко *NCIP* протокола су имплементиране са преостале две класе са дијаграма на слици 6.3: *NCIPAdapter* и *SenderHttps*. Класа *SenderHttps* омогућава слање објеката кроз *HTTPS* протокол. Класа *NCIPAdapter* конвертује добијену команду у *NCIP* поруку и ту поруку прослеђује класи *SenderHttps* која је даље шаље на одговарајући сервер.

Из наведеног следи да подсистем за клијент/сервер комуникацију омогућава два начина комуникације са удаљеним серверима:

- слање команди на удаљене сервере (чиме је реализована циркулацију у конзорцијуму унутар библиотечке мреже БИСИС),
- размену *NCIP* порука кроз *HTTPS* протокол (чиме је реализована циркулација у конзорцијуму између различитих библиотечких система).

У случају потребе за комуникацијом на неки други начин, било да се ради о другом апликативном протоколу или другом транспортном протоколу потребно је обезбедити одговарајуће адаптер класе или имплементације интерфејса *Sender.* Из тог разлога подсистем може да се користи за комуникацију по различитим протоколима и у различитим применама.

6.3.3 Имплементација проширења подсистема за клијент/сервер комуникацију

На основу описане архитектуре у претходном одељку имплементирано је проширење подсистема за клијент/сервер комуникацију. Проширење обухвата имплементацију интерфејса и класа приказаних на слици 6.3, као и измене на неким од постојећих класа чија је имплементација описана у одељку 4.1.4.

Постојећа класа *ServiceType* дефинише типове сервиса, односно начине извршавања команди. У одељку 4.1.4 дефинисана су два типа сервиса: *LOCAL* и *REMOTE*. За потребе комуникације преко *NCIP* протокола дефинисан је још један тип сервиса *NCIP*.

Имплементације интерфејса *ServiceFactory* креирају одговарајућу инстанцу интефејса *Service* на основу типа сервиса који се прослеђује као параметар. На листингу 4.3 у одељку 4.1.4 приказана је једна од имплементација. Ове имплементације су измењене тако да се у случају типа сервиса *REMOTE* креира инстанца класе *BisisAdapter* (уместо инстанце *ServiceBroker*), а у случају типа сервиса *NCIP* креира се инстанца класе *NCIPAdapter*.

Интерфејси и класе приказани на слици 6.3 имплементирани су у облику *Java* интерфејса и класа. Класе *BisisAdapter* и *SenderHessian* имплементирају функционалност класе *ServiceBroker* чија је имплементација дата на листингу 4.5 у одељку 4.1.4. Класа *SenderHessian* креира *proxy* објекат преко кога шаље команде на сервер, а класа *BisisAdapter* добијене команде од клијента прослеђује инстанци класе *SenderHessian*.

Класа *SenderHttps* омогућава слање објеката кроз *HTTPS* протокол. Поруке се шаљу кроз *POST* захтев као тело *HTTP* поруке. За имплементацију комуникације коришћен је програмски пакет

HttpClient. *HttpClient* је развијен у оквиру пројекта *HttpComponents* [HttpComponents] од стране *Apache* заједнице. Пакет *HttpClient* представља имплементацију клијентске стране *HTTP* комуникације и омогућава слање и примање *HTTP* порука. Такође, пакет омогућава креирање *SSL* конекције кроз коју се обавља *HTTP* транспорт што је коришћено приликом имплементације класе *SenderHttps*.

```
public class NCIPAdapter implements Service {
.......
public Command executeCommand(Command command) {
    String ncipRequest = null;
    String ncipResponse = null;
    if (command instanceof GetUserCommand){
        ncipRequest = NCIPSerializer.toLookupUser
                        ((GetUserCommand)command, param);
        ncipResponse = (String)sender.send(ncipRequest);
        NCIPSerializer.fromLookupUserResponse
                        ((GetUserCommand)command, ncipResponse);
        } else if .....
            .....
        }
        return command;
    }
}
```

Листинг 6.5 Имплементација метода *executeCommand* класе *NCIPAdapter*

Класа *NCIPAdapter* има задатак да добијену команду конвертује у *NCIP* поруку и ту поруку проследи инстанци класе *SenderHttps* која је даље шаље на одговарајући сервер. Класа *NCIPAdapter* команду добија кроз позив метода *executeCommand*. На листингу 6.5 дат је део имплементације метода *executeCommand*. По добијању команде класа *NCIPAdapter* проверава коју команду је добила и у односу на то позива одговарајући метод помоћне класе *NCIPSerializer* која команду конвертује у *NCIP* захтев. Након тога захтев се прослеђује инстанци класе *SenderHttp sender* која захтев шаље на одговарајући сервер и враћа добијени одговор. Одговор се помоћу одговарајућег метода класе

NCIPSerializer конвертује у команду и враћа клијенту одакле клијент чита тражене податке.

Помоћна класа *NCIPSerializer* служи за конверзију команде у *NCIP* захтев и конверзију *NCIP* одговора у команду. Класа *NCIPSerializer* за рад са *XML* документима користи програмски пакет *XMLBeans* [XMLBeans]. Овај пакет је такође развијен од стране *Apache* заједнице и служи за рад са *XML* документима из *Java* програмског кода. На основу *XML* шеме *XMLBeans* компајлира скуп *Java* интерфејса и класа. Ови интерфејси и класе се у програмском коду користе за приступ и модификацију инстанци *XML* шеме. Будући да су поруке *NCIP* протокола дефинисане *XML* шемом *XMLBeans* је врло погодан за креирање и модификовање *NCIP* порука.

Класа *NCIPSerializer* за сваки имплементирани *NCIP* сервис има по два метода који врше конверзију. Један метод *NCIP* захтев конвертује у команду, а други резултате извршавања команде конвертује у *NCIP* одговор. На листингу 6.5 су коришћени методи који врше пребацивање података између сервиса *Lookup User* и команде *GetUserCommand*. То су методи *toLookupUser* и *fromLookupUserResponse*. У табели 6.2 је дат приказ имплементираних сервиса у БИСИС систему и одговарајућих команди на које се конвертују методама класе *NCIPSerializer.*

Табела 6.2 Парови одговарајућих сервиса и команди

Сервис	**Команда**
Lookup User	*GetUserCommand*
Check Out Item	*SaveLendingCommand*
Item Renewed	*SaveRemoteUserCommand*
Renew Item	*SaveLendingCommand*
Check Item In	*SaveLendingCommand*
Lookup Item	*GetRecordCommand*

На листингу 6.6 је приказана имплементација метода *toLookupUser* класе *NCIPSerializer* која добијену команду конвертује у *LookupUser* захтев. Пример *LookupUser* захтева је претходно приказан на листингу

6.2. Приликом креирања поруке из одговарајућег *XML* документа се учита темплејт поруке у коме су дефинисани елементи које порука садржи као и вредности које су исте за све поруке тог типа (на пример, *URI* адреса листе вредности за неки елемент). Парсирањем овог темплејта помоћу *XMLBeans*-а добија се објектни модел поруке који даље може да се модификује. У датом листингу парсирањем темплејта добио се објекат *doc*. Темплејт садржи већину елемената и вредности које су наведене у поруци на листингу 6.2. Оно што темплејт не садржи су вредности за ознаку библиотеке која шаље захтев, ознаку библиотеке која прима захтев, ознаку библиотеке којој корисник припада и чланski број корисника. Ове вредности се у поруку смештају преко објекта *doc*. Чланski број корисника се узима из команде. Ознаке библиотеке се добијају из помоћне класе *MappingValues*. Класа *MappingValues* садржи мапирања листа вредности које се користе у *NCIP* порукама и које су заједничке за конзорцијум на вредности дефинисане у локалним шифарницима библиотеке. У приказаном примеру преко метода *getMyAgencyID* добија се ознака за локалну библиотеку која се користи у конзорцијуму, а преко метода *getNCIPAgenciID* добија се ознака за библиотеку чији сервис се налази на адреси наведеној у параметру *library*. Након постављања свих вредности из објекта *doc* креира се текст поруке који представља повратну вредност метода.

Метода *fromLookupUserResponse* коришћена у листингу 6.5 конвертује одговор *LookupUserResponse* у команду, односно податке добијене у одговору смешта у команду одакле их клијент користи. И у овом случају парсирањем одговора помоћу *XMLBeans*-а добија се објектни модел поруке преко кога се приступа подацима добијеним у поруци. Као што је приказано у одељку 4.2.1 у процесу извршавања команде *GetUserCommand* подаци о кориснику се смештају у објекте класа објектног модела базе података. Због тога се и приликом конверзије креирају унутар команде објекти тог објектног модела у које се смештају подаци из поруке. На тај начин клијенту је омогућено да и у случају комуникације преко *NCIP* протокола податке користи на исти начин као у случају када се команда извршава.
На сличан начин су имплементиране и остале методе класе *NCIPSerializer* које обрађују поруке других сервиса.

```
public static String toLookupUser(GetUserCommand command,
                                                    String library){
    LookupUserDocument doc = LookupUserDocument.Factory.parse
        (NCIPSerializer.class.getResource("/com/gint/app/bisis4/
         commandservice/templates/LookupUser.xml"));
    doc.getLookupUser().getInitiationHeader().getFromAgencyId().
        getAgencyId().setStringValue(MappingValues.getMyAgencyID());
    doc.getLookupUser().getInitiationHeader().getToAgencyId().
        getAgencyId().setStringValue
        (MappingValues.getNCIPAgenciID(library));
    doc.getLookupUser().getUserId().getAgencyId().
        setStringValue(MappingValues.getNCIPAgenciID(library));
    doc.getLookupUser().getUserId().setUserIdentifierValue
        (command.getUserID());
    StringWriter sw = new StringWriter();
    XmlOptions xmlOptions = new XmlOptions();
    xmlOptions.setSavePrettyPrint();
    doc.save(sw,xmlOptions);
    return sw.toString();
}
```

Листинг 6.6 Имплементација метода toLookupUser класе NCIPSerializer

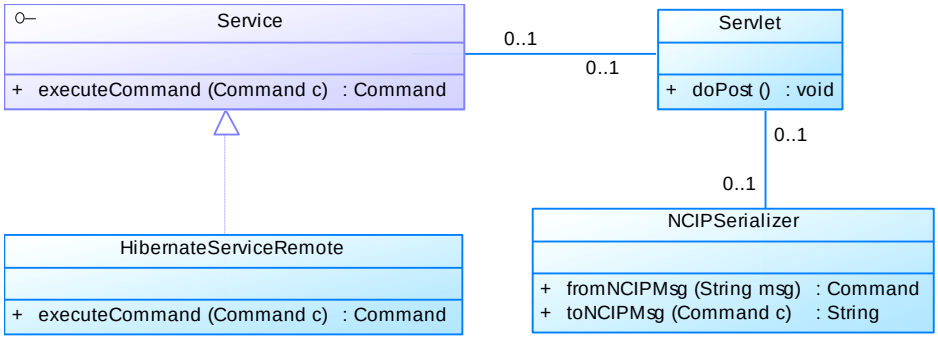
6.4 *NCIP* сервис система БИСИС

У претходним одељцима описана је имплементација *NCIP* протокола на клијентској страни система за циркулацију чиме је омогућено да корисници БИСИС система буду иницијатори *NCIP* комуникације, односно да шаљу *NCIP* захтеве и примају *NCIP* одговоре. Да би се омогућила циркулација у конзорцијуму потребно је и другим системима омогућити да БИСИС систему шаљу захтеве и добијају одговоре, чиме се успоставља обострана размена података. Да би се ово реализовало потребно је у оквиру БИСИС система направити *NCIP* сервер који би примао захтеве од других система и слао им одговоре.

Клијентска страна је реализована тако да клијент креира команде, иницира њихово извршавање и чита резултате из њих. Приликом

извршавања команди подсистем конвертује те команде у *NCIP* поруке преко којих се обавља комуникација, а затим податке из порука поново смешта у команде које враћа клијенту. Слична идеја је искоришћена за реализацију *NCIP* сервера. *NCIP* сервер добија поруке које затим конвертује у одговарајуће команде имплементиране у оквиру система за циркулацију које извршава над базом података система циркулације. Резултате извршавања из команди конвертује у поруке које враћа као одговор.

На слици 6.4 приказана је архитектура *NCIP* сервера система БИСИС. Класа *Servlet* је класа која добија *NCIP* захтеве и шаље *NCIP* одговоре. Њен задатак је да по добијању *NCIP* захтева конвертује ту поруку у команду. Конверзија се врши позивањем одговарајућег метода класе *NCIPSerializer*. Поред тога класа *Servlet* има референцу на инстанцу интерфејса *Service* преко које се извршава команда. Интерфејс *Service* и класа *HibernateServiceRemote* су преузети из имплементације серверске стране подсистема за клијент/сервер комуникацију чији модел и имплементација су описани у одељцима 4.1.3 и 4.1.4. Класа *HibernateServiceRemote* је имплементација интерфејса *Service* која служи за извршавање команди које за комуникацију са базом података користе *Hibernate* пакет. Како команде система циркулације за комуникацију са базом података користе *Hibernate,* у реализацији *NCIP* сервера за извршавање команди користи се имплементација *HibernateServiceRemote*. Након извршавања команде она се са резултатима извршавања враћа класи *Servlet.* Класа *Servlet* даље ту команду конвертује у *NCIP* одговор позивањем одговарајућег метода класе *NCIPSerializer* и враћа га пошиљаоцу захтева.



Слика 6.4 Дијаграм класа *NCIP* сервера система БИСИС

6.4.1 Имплементација *NCIP* сервиса

На основу описане архитектуре урађена је имплементација *NCIP* сервера система БИСИС. Имплементација је урађена у програмском језику *Java*. Имплементација интерфејса *Service* и класе *HibernateServiceRemote* дата је у одељку 4.1.4. Класа *Servlet* имплементирана је као *Java Servlet*. Класа кроз *HTTP* захтев добија *NCIP* поруке које обрађује унутар *doPost* метода. Очекује се да је *NCIP* порука смештена у телу *HTTP* поруке.

Добијена порука се конвертује у команду преко класе *NCIPSerializer*. Класа *NCIPSerializer* има исту функцију као истоимена класа имплементирана на клијентској страни. Њен задатак је да пристигли *NCIP* захтев конвертује у одговарајућу команду, као и да резултате извршавања команде конвертује у одговарајући *NCIP* одговор. На дијаграму на слици 6.4 илустративно су приказана два метода који представљају ове две врсте конверзије. Имплементација класе *NCIPSerializer* за сваки имплементирани *NCIP* сервис има по два метода који врше конверзију. На пример, за сервис *LookupUser* то су методи *fromLookupUser* и *toLookupUserResponse*. Слично као у имплементацији на клијентској страни ова два метода врше пребацивање података између сервиса *LookupUser* и команде *GetUserCommand*. Метод *fromLookupUser* захтев поруку сервиса *LookupUser* конвертује у команду *GetUserCommand*. Метод *toLookupUserResponse* резултате извршавања команде *GetUserCommand* конвертује у одговор поруку сервиса *LookupUser*.

Као и на клијентској страни и у овој имплементацији класа *NCIPSerializer* за рад са *XML* документима користи програмски пакет *XMLBeans* помоћу кога добија објектни модел *NCIP* поруке. Пребацивање података између порука и команди је урађено на сличан начин као у имплементацији на клијентској страни што је приказано у примеру на листингу 6.6. Такође и у овом случају користи се помоћна класа *MappingValues* која садржи мапирања листа вредности које се користе у *NCIP* порукама на вредности дефинисане у локалним шифарницима библиотеке.

Поглавље 7

Закључак

Истраживање чији резултати су приказани у овој дисертацији рађено је у делу библиотечког пословања које се бави циркулацијом библиотечког фонда. Ово истраживање је наставак истраживања чији резултати су приказани у магистарској тези (Тешендић, 2007). У магистраској тези је реализован систем за циркулацију који задовољава потребе електронског пословања библиотеке које се односе на евидентирање и услуживање корисника у оквиру локалног фонда библиотеке. Опис овог система је дат у трећем поглављу дисертације.

Истраживање је настављено у правцу комуникације између библиотека унутар конзорцијума с циљем да се омогући праћење коришћена фондова на нивоу конзорцијума. Предмет истраживања је процес циркулације у конзорцијуму библиотека где корисници могу да користе фондове свих библиотека у конзорцијуму, а не само фонд матичне библиотеке којој припадају. Разултат овог истраживања је проширење система за циркулацију тако да он омогућава размену података између библиотека, односно, рад са корисницима из других библиотека. Резултати су презентовани у четвртом, петом и шестом поглављу.

За моделирање софтверске архитектуре система за циркулацију коришћен је објектно-оријентисани приступ и језик UML 2.0. Имплементација система је урађена у програмском језику Java. Систем има трослојну архитектуру. Састоји се од клијентске апликације, подсистема за клијент/сервер комуникацију и базе података. Софтверска архитектура клијентске апликације и подсистема за комуникацију је патерн оријентисана.

Коришћен је методолошки приступ унифицирани процес развоја софтверског система. Развој система за циркулацију реализован је кроз

четири итерације. У првој итерацији је реализована верзија система за циркулацију у оквиру локалног фонда библиотеке која је као што је већ речено резултат магистарске тезе (Тешендић, 2007). Верзије система реализоване у другој, трећој и четвртој итерацији су резултат истраживања дисертације и описане су у четвртом, петом и шестом поглављу дисертације, респективно.

У четвртом поглављу је описано моделирање и имплементација подсистема за клијент/сервер комуникацију и његова интеграција у систем БИСИС. Интеграцијом подсистема у систем за циркулацију омогућена је комуникација са базом података у различитим мрежним окружењима. Подсистем има патерн оријентисану софтверску архитектуру која је заснована на комбинацији *command* патерна, *apstract factory* патерна и *factory method* патерна. Интеграцијом овог подсистема у систем БИСИС добијени су следећи резултати:

- Клијенту је омогућена транспарентна комуникација са сервером у односу на транспортни протокол који се користи за комуникацију. У случају система за циркулацију на овај начин је омогућена комуникација са базом података и у случају када је база у локалном окружењу и у случају када је на Интернету, при чему је интерфејс за комуникацију за клијента исти у оба случаја.
- Пословна логика система је имплементирана командама које су независне од подсистема за комуникацију. Измена функција система подразумева додавање нових или измену постојећих команди, при чему се подсистем не мења.
- Будући да подсистем не зависи од пословне логике система који га користи интегрисан је и у остале делове БИСИС система.

У петом поглављу је дата спецификација функционалности за проширење система за циркулацију тако да се омогући циркулација у конзорцијуму библиотека. Проширен је модел података и имплементиране су нове функционалности система којима је омогућен рад са корисницима из других библиотека. За размену података са другим библиотекама искоришћен је подсистем за клијент/сервер комуникацију. Употребом подсистема у ову сврху добијени су следећи резултати:

- Клијентска апликација користи исти интерфејс за комуникацију и у случају комуникације са својим сервером и у случају комуникације са серверима других библиотека.
- У комуникацији између библиотека размењују се команде са пословном логиком, тако да је ова размена ограничена само на библиотеке које користе систем БИСИС.

Такође, у петом поглављу је дат преглед механизама заштите података који би могли да се примене при употреби подсистема за клијент/сервер комуникацију.

У шестом поглављу описан је начин имплементације *NCIP* протокола у оквиру система за циркулацију. Протокол је имплементиран унутар подсистема за клијент/сервер комуникацију чиме је клијенту омогућен приступ другим библиотекама по *NCIP* протоколу. У проширењу подсистема коришћен је *adapter* патерн. Такође, имплементиран је и *NCIP* сервис који служи за приступ подацима по *NCIP* протоколу од стране других библиотека. Оваквом имплементацијом протокола добијени су следећи резултати:

- Клијентска апликација користи исти интерфејс за комуникацију без обзира да ли се комуникација одвија преко *NCIP* протокола или не. Клијент размењује податке кроз команде, док је подсистем за комуникацију задужен за конверзију команди у *NCIP* поруке.
- Омогућена је циркулација у конзорцијуму између библиотека које користе различите библиотечке системе.

Наведени добијени резултати у потпуности испуњавају циљеве истраживања дефинисане у уводном поглављу. Посебан научни значај у овом истраживању има патерн оријентисана софтверска архитектура подсистема за клијент/сервер комуникацију. Погодном комбинацијом неколико дизајн патерна добила се архитектура која омогућава комуникацију клијента и сервера у различитим мрежним окружењима и кроз различите транспортне протоколе, а такође омогућава и конверзију података који се размењују у различите формате у зависности од потребе. Клијенти који користе подсистем су независни од начина комуникације јер податке подсистему шаљу увек у истом формату, док се начин комуникације подешава у конфигурацији која се прослеђује подсистему. Подсистем је независан

од пословне логике система који га користи тако да се може интегрисати у било који систем који има потребу за клијент/сервер комуникацијом. Измене функционалности система не захтевају измене на подсистему за комуникацију.

Интеграцијом подсистема за комуникацију у систем циркулације добио се једнообразан начин комуникације клијентске апликације са серверском страном и у случају кад се сервер налази у локалном окружењу и у случају кад је на Интернету. Такође, исти тај начин комуникације се користи и када је у питању комуникација са другим библиотекама, било да се ради о библиотечкој мрежи БИСИС или да се комуникација одвија по стандардизованом протоколу. Подсистем је интегрисан и у друге делове БИСИС система где је било потребе за клијент/сервер комуникацијом.

Тренд развоја Интернет сервиса пружа нове могућности, с којима расту и очекивања корисника библиотека. Сервиси који корисницима омогућавају коришћење електонских докумената, као и сервиси за претраживање и лоцирање ресурса као што је *WorldCat*, постају све више популарни. Као и у случају папирних ресурса библиотеке имају потребу за евидентирањем коришћења електронских ресурса. Праћење коришћења електронског фонда библиотеке је нови задатак који није обухваћен резултатима ове дисертације и може да представља смерницу за нека даља истраживања. Такође, сервиси који корисницима пружају неке услуге на основу њиховог чланства у библиотеци размењују податке са библиотекама. За те потребе користи се *NCIP* протокол. Проширење *NCIP* сервиса описаног у дисертацији у сврхе размене података са различитим сервисима остаје, такође, као могућност за нека даља разматрања.

Литература

Adams, B., Noel, B. (2008), "Circulation statistics in the evaluation of collection development", Collection Building, Vol. 27, No. 2, pp. 71 – 73, www.emeraldinsight.com/10.1108/01604950810870227

Agrawal, A., Ramani, K., and Hoffmann, C. (2002), "CADDAC: Multi-Client Collaborative Shape Design System with Server-Based Geometry Kernel", Proceedings of ASME 2002 Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Montreal, Canada Paper No. DETC20020CIE-344

Alexander, C. (1979), "The Timeless Way of Building", Oxford University Press

Alur, D., Crupi, J., Malks, D. (2001), "Core J2EE Patterns: Best Practises and Design Strategies", Prentice Hall

Babafemi, G. O. (2002), "Public relations activities in an academic library: the roles of the circulation librarian", Library Review, Vol. 51, No. 9, pp. 464-468, www.emeraldinsight.com/10.1108/00242530210446944

Beck, K. (1997), "Smalltalk Best Practice Patterns", Prentice-Hall

Belić K., Surla D. (2008), "User Friendly Web Application for Bibliographic Material Processing", The Electronic Library, Vol. 26, No. 3, pp. 400 – 410, www.emeraldinsight.com/10.1108/02640470810879536

Bennett, M. J. (2007), "OPAC Design Enhancements and Their Effects on Circulation and Resource Sharing within the Library Consortium Environment", Information Technology and Libraries, Vol. 26, No. 1, pp. 36 (11 pages), http://proquest.umi.com/pqdlink?did=1269339471&sid=1&Fmt=3&clientId=57708&RQT=309&VName=PQD

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 – 495, www.emeraldinsight.com/10.1108/02640470910966916

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

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

Boone, M. D. (2002), "Taking FLITE: how new libraries are visioning their way into the future", Library Hi Tech, Vol. 20, No. 4, pp. 464 – 468, www.emeraldinsight.com/10.1108/07378830210452668

Braun, P., Hörnig, L., Visse, F. (2006), "A new approach towards a national inter-library loan system in The Netherlands: introducing VDX", Interlending & Document Supply, Vol. 34, No. 4, pp. 152-159, www.emeraldinsight.com/10.1108/02641610610714713

Budimir, G., Surla, D. (2004), "Quality control system of XML bibliographic records", Novi Sad Journal of Mathematics, Vol 34, No 1, pp 107-130, http://www.im.ns.ac.yu/NSJOM/

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. (1996), "Pattern-oriented software architecture: a system of patterns", John Wiley & Sons, Inc., New York

Cheng, R., Bischof, S., Nathanson, A. J. (2002), "Data collection for user-oriented library services: Wesleyan University Library's experience", OCLC Systems & Services, Vol. 18, No. 4, pp. 195 – 204, www.emeraldinsight.com/10.1108/10650750210450130

Cooper, J.W. (1998), "The Design Patterns: Java Companion", Addison-Wesley

Cunningham, W. (1994), "The CHECKS Pattern Language of Information Integrity", Proceedings of PLOP '94, pp. 145-155

Dimić B., Surla D. (2009), "XML Editor for UNIMARC and MARC21 cataloguing", The Electronic Library, Vol. 27, No. 3, pp. 509 – 528, www.emeraldinsight.com/10.1108/02640470910966934

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

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

Falk, H. (2005), "Temple of the computer", The Electronic Library, Vol. 23, No. 2, pp. 244 – 248, www.emeraldinsight.com/10.1108/02640470510592951

Fleck, N.W. (2000), "Interlibrary loan – a new frontier!", Library Hi Tech, Vol. 18, No. 2, pp. 172-176, www.emeraldinsight.com/10.1108/07378830010333563

Froud, R. (2006), "Unity reaps rewards: an integrated UK ILL and resource discovery solution for libraries", Interlending & Document Supply, Vol. 34, No. 4, pp. 164-166, www.emeraldinsight.com/10.1108/02641610610714731

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

Gatenby, J. (2003), "Inter-library loans and document delivery via EUCAT: a new PICA/OCLC initiative", Interlending & Document Supply, Vol. 31, No. 2, pp. 123-129, www.emeraldinsight.com/10.1108/02641610310460673

Gould, S. (2000), "Sending ILL requests by e-mail: a discussion and IFLA guidelines", Interlending & Document Supply, Vol. 28, No. 2, pp. 73-78, www.emeraldinsight.com/10.1108/02641610010331525

Guerrero, E. M., Matte, D. (2003), "Interlibrary loan in Mexico: two solutions to an age-old problem", Interlending & Document Supply, Vol. 31, No. 1, pp. 12-14, www.emeraldinsight.com/10.1108/02641610310460691

Hammouda, I., Koskimies, K. (2002),"A pattern-based J2EE application development environment", Nordic Journal of Computing, Vol. 9, No. 3, pp. 248-260

Huang, Y, Chung, J.Y. (2003), "A Web Services-Based Framework for Business Integration Solutions". Electronic Commerce Research and Applications, Vol. 2, No. 1, pp. 15–26

Hunold, S. , Korch, M., Krellner, B., Rauber, T., Rechel, T., Runger, G.. (2008), "Transformation of Legacy Software into Client/Server Applications through Pattern-based Rearchitecturing", Proc. the 32nd IEEE International Computer Software and Applications Conference, pp. 303 - 310

Jacobson I., Booch G. and Rumbaugh, J. (1999), "The Unified Software development process", Addison-Wesley

Johnson, G, (2009), "Engineering to the rule, not the exception: an interlibrary loan case study at The University of California, Santa Barbara", Interlending & Document Supply, Vol. 37, No. 1, pp. 4 - 10

Joint, N., (2008), "Is digitisation the new circulation?: Borrowing trends, digitisation and the nature of reading in US and UK libraries", Library Review, Vol. 57, No. 2, pp. 87 – 95, www.emeraldinsight.com/10.1108/00242530810853973

Kern, C. (2004), "Radio-frequency-identification for security and media circulation in libraries", The Electronic Library, Vol. 22, No. 4, pp. 317 – 324, www.emeraldinsight.com/10.1108/02640470410552947

Konjović, Z., Surla, D. (eds) (2004), "Overview of the development of the library information system BISIS", conference proceedings, International Conference on Distributed Library Information Systems, Ohrid

Лазаревић, Б., Бендер, М., Цветановић, С., Дикановић, В., Ђуричић, Д., Конечни, А., Коруновић, Д., Миливојевић, Љ., Нешковић ,С., Пауновић, Ђ., Сурла, Д. (1996), "Формирање и претраживање База података у систему научних и технолошких информација Србије", монографија, Министарство за науку и технолошки развој Републике Србије, Београд

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

Milosavljević, B., Dimić, B. (2007), "XML Schema of UNIMARC format variant and bibliographic record in BISIS software system", Novi Sad Journal of Mathematics, Vol. 37, No. 1, pp 115-128, http://www.im.ns.ac.yu/NSJOM/

Milosavljević, B., Tešendic, D. (2010), "Software Architecture of Distributed Client/Server Library Circulation System", The Electronic Library (u štampi)

Missingham, R. (2007), "Networking a nation: ILL developments in Australia", Library Hi Tech, Vol. 25, No. 2, pp. 188-196, www.emeraldinsight.com/10.1108/07378830710754956

O'Brien, J., Shapiro, M. (2004), "Undo for anyone, anywhere, anytime", Proceedings of the 11th workshop on ACM SIGOPS European workshop, Leuven, Belgium

Pereira, F. M. Q., Valente, M. T. O., Bigonha, R. S., Bigonha, M. A. S. (2006), "Arcademis: a Framework for Object-Oriented Communication Middleware Development", ACM Software-Practice & Experience, Vol. 36, No. 5, pp. 495 –512

Porat, L., Shoham, S. (2004), "Israeli college interlibrary loan practices: implications for Israeli universities", Interlending & Document Supply, Vol. 32, No. 4, pp. 219-226, www.emeraldinsight.com/10.1108/02641610410567962

Рађеновић Ј., Сурла, Д. и Милосављевић Б. (2006), "Софтверски пакет за генерисање библиотечких каталошких листића", монографија, Природно-математички факултет, Департман за математику и информатику, Нови Сад

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

Rudić, G., Surla, D. (2009), "Conversion of bibliographic records to MARC 21 format", The Electronic Library, Vol. 27, No. 6, pp 950 – 967, www.emeraldinsight.com/10.1108/02640470911004057

Sawant, N., Scharver, C., Leigh, J., Johnson, A., Reinhart, G., Creel, E., Batchu, S., Bailey, S., Grossman, R. (2000),"The Tele-Immersive Data Explorer: A Distributed Architecture for Collaborative Interactive Visualization of Large Data-sets", 4th International Immersive Projection Technology Workshop, Ames, Iowa, http://evlweb.eecs.uic.edu/cavern/TIDE/tide_ipt2000.pdf

Shuh, B. (1998), "The renaissance of the interlibrary loan protocol: developments in open systems for interlibrary loan message management", Interlending & Document Supply, Vol. 26, No. 1, pp. 25-33., www.emeraldinsight.com/10.1108/02641619810369662

Stallings, W. (2007), "Network security essentials", Pearson, Prentice Hall

Tešendić, D. (2007), "A database model for library material usage",Novi Sad Journal of Mathematics, Vol. 37, No.1, 2007, pp 155-162, http://www.im.ns.ac.yu/NSJOM/

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, http://www.emeraldinsight.com/10.1108/02640470910934669

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

Vergara, N.M., Linero, J.M.T., Moreno, A.V. (2007), "Model-driven component adaptation in the context of Web Engineering", European Journal of Information Systems, Vol. 16, No. 4, pp. 448-459

Zhang, L., Mathew, A., Zhang, S., Ma, L. (2006) "Implementation of Asset Health Assessment System with Pattern-Oriented Design and Practice", 3rd International IEEE Conference on Intelligent Systems, London

***Web* странице**

[Aleph500] Aleph 500, http://www.exlibrisgroup.com/aleph.htm (прегледано 21. фебруара 2010.)

[Atriuum] Atriuum, http://www.booksys.com/v2/products/atriuum/ (прегледано 21. фебруара 2010.)

[Concourse] Concourse, http://www.booksys.com/v2/products/concourse/ (прегледано 21. фебруара 2010.)

[CyberTools] Cybertools for Libraries, http://www.cybertoolsforlibraries.com/ (прегледано 21. фебруара 2010.)

[ebXML] Electronic Business using eXtensible Markup Language http://www.ebxml.org/specs/index.htm (прегледано 21. фебруара 2010.)

[Eclipse] Eclipse IDE, http://www.eclipse.org/ (прегледано 21. фебруара 2010.)

[EOS] EOS.Web, http://www.eosintl.com/ (прегледано 21. фебруара 2010.)

[Evergreen] Evergreen Integrated Library System, http://open-ils.org/ (прегледано 21. фебруара 2010.)

[Hessian] Hessian binary web service protocol, http://hessian.caucho.com/ (прегледано 21. фебруара 2010.)

[Hibernate] Hibernate project, http://www.hibernate.org/ (прегледано 21. фебруара 2010.)

[HttpComponents] HttpComponents, http://hc.apache.org/ (прегледано 21. фебруара 2010.)

[ISO10160] ISO 10160:1997 Information and documentation -- Open Systems Interconnection -- Interlibrary Loan Application Service Definition, http://www.iso.ch/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22247 (прегледано 21. фебруара 2010.)

[ISO10161-1] ISO 10161-1:1997 Information and Documentation - Open Systems Interconnection - Interlibrary Loan Application Protocol Specification - Part 1: Protocol Specification, http://www.iso.ch/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22248 (прегледано 21. фебруара 2010.)

[ISO10161-2] ISO 10161-2: Information and Documentation - Open Systems Interconnection - Interlibrary Loan Application Protocol Specification - Part 2: PICS Proforma, http://www.iso.ch/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=21389 (прегледано 21. фебруара 2010.)

[ISO2709] ISO 2709:2008 Information and documentation - Format for information exchange, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=41319 (прегледано 21. фебруара 2010.)

[Koha] Koha Integrated Library System, http://koha.org/ (прегледано 21. фебруара 2010.)

[Marc21] Mark21, Marc Standards, Library of Congress, Network Development and Marc Standards Office, http://www.loc.gov/marc/ (преглеdано 21. фебруара 2010.)

[MARCStandards] MARC Standards, Network Development and MARC Standards Office, Library of Congress, http://www.loc.gov/marc/ (преглеdано 21. фебруара 2010.)

[MARCSystems] MARC Records, Systems and Tools, http://www.loc.gov/marc/marcservice.html (преглеdано 21. фебруара 2010.)

[MySQL] MySQL open source database, http://www.mysql.com/ (преглеdано 21. фебруара 2010.)

[NCIP] NCIP Implementation Group website, http://www.ncip.info/ (преглеdано 21. фебруара 2010.)

[OCLC] *OCLC, Online Computer Library Center*, http://www.oclc.org/ (преглеdано 21. фебруара 2010.)

[Polaris] Polaris Library Systems, http://www.polarislibrary.com/ (преглеdано 21. фебруара 2010.)

[Sybase] Sybase, http://www.sybase.com/ (преглеdано 21. фебруара 2010.)

[Symphony] SirsiDynix Symphony, http://www.sirsidynix.com/ (преглеdано 21. фебруара 2010.)

[Tomcat] Apache Tomcat, http://tomcat.apache.org/ (преглеdано 21. фебруара 2010.)

[Unicorn] SirsiDynix Unicorn, http://www.sirsidynix.com/ (преглеdано 21. фебруара 2010.)

[Unimarc] UNIMARC Manual : Bibliographic Format 1994, IFLA Universal Bibliographic Control and International MARC Core Programme (UBCIM), http://www.ifla.org/VI/3/p1996-1/sec-uni.htm (преглеdано 21. фебруара 2010.)

[Validator] Apache Commons Validator, http://jakarta.apache.org/commons/validator/ (преглеdано 21. фебруара 2010.)

[VDX] VDX, http://www.oclc.org/vdx/default.htm (прегледано 21. фебруара 2010.)

[Voyager] Voyager Integrated Library System, http://www.exlibris-usa.com/voyager.htm (прегледано 21. фебруара 2010.)

[WorldCat] WorldCat, http://www.oclc.org/worldcat/default.htm (прегледано 21. фебруара 2010.)

[XMLBeans] Apache XMLBeans, http://xmlbeans.apache.org/ (прегледано 21. фебруара 2010.)

[Z3950], ANSI/NISO Z39.50-2003, Information Retrieval (Z39.50): Application Service Definition and Protocol Specification, http://www.loc.gov/z3950/agency/document.html (прегледано 21. фебруара 2010.)

[Z3983-1] ANSI/NISO Z39.83-1 - NISO Circulation Interchange Part 1: Protocol (NCIP), http://www.niso.org/kst/reports/standards?step=2&gid%3Austring%3Aiso-8859-1=&project_key%3Austring%3Aiso-8859-1=2d46d484a625029ef698b96b7537c334348c8eb8 (прегледано 21. фебруара 2010.)

[Z3983-2] ANSI/NISO Z39.83-2 - NISO Circulation Interchange Protocol (NCIP) Part 2: Implementation Profile 1, http://www.niso.org/kst/reports/standards?step=2&gid%3Austring%3Aiso-8859-1=&project_key%3Austring%3Aiso-8859-1=599708d764b8a1cccb7fad45d74ec70c1b7cb235 (прегледано 21. фебруара 2010.)

Биографија

Данијела Тешендић је рођена 30.04.1979. године у Градачцу, БиХ. На Природно-математички факултет Универзитета у Новом Саду, одсек за математику, смер дипломирани информатичар уписала се школске 1998/99. године. У периоду од 1998-2004. године положила је све испите предвиђене планом и програмом са просечном оценом 8,90 (осам и 90/100). Дипломски рад *Имплементација процеса Овера семестра у ORACLE окружењу* одбранила је у јуну 2004. године са оценом 10 (десет).

На последипломске студије, смер информатика, уписала се школске 2004/05 године на Природно-математичком факултету у Новом Саду. Положила је све испите предвиђене планом и програмом са просечном оценом 10 (десет). Магистарску тезу под насловом *Систем за коришћење библиотечке грађе* одбранила је у септембру 2007. године и стекла академски назив магистра информатичких наука.

У децембру 2005. године засновала је радни однос на Природно-математичком факултету у Новом Саду на радном месту истраживач-приправник и ангажована је на пројекту *Апстрактни методи 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.

Држала је вежбе из предмета *Информациони системи, Одабрана поглавља хардвера и софтвера* и *Рачунарксе мреже* на основним студијама.

Има девет објављених научних радова од којих су два рада у међународном часопису са SCI листе, један рад у часопису националног значаја, два рада са међународног скупа штампана у целини, једна монографија од националног значаја и три рада 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 Тешендић
Ментор: **МН**	др Душан Сурла, професор емеритус, ПМФ, Нови Сад
Наслов рада: **НР**	Софтверски систем за циркулацију библиотечке грађе у оквиру библиотечке мреже
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**	(7/194/92/8/36/0/0) (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**	Циркулација, конзорцијум, UML, патерн оријетисана софтверска архитектура, дизајн патерни, NCIP
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 софтверског система за циркулацију који омогућава праћење коришћења библиотечког фонда на нивоу конзорцијума библиотека. Коришћен је методолошки приступ унифицирани процес развоја система. У моделирању архитектуре коришћени су дизајн патерни, а модел је приказан у *UML* 2.0 нотацији. Систем је имплементиран у програмском језику *Java*. У оквиру система развијен је подсистем за

клијент/сервер комуникацију који омогућава транспарентну комуникацију клијента и сервера у односу на транспортни протокол које се користи. Подсистем има патерн оријентисану софтверску архитектуру која је заснована на комбинацији неколико дизајн патерна. Његовом интеграцијом у софтверски систем БИСИС омогућен је рад система у различитим мрежним окружењима.

Такође, подсистем је искоришћен и за комуникацију са другим библиотекама. У оквиру подсистема имплементиран је *NCIP* протокол чиме је омогућена размена података са библиотекама које користе различите библиотечке софтверске системе. Подсистем омогућава једнообразан начин комуникације клијентске апликације, било са сервером своје библиотеке или серверима других библиотека. Имплементиран је и *NCIP* сервис који служи за приступ подацима по *NCIP* протоколу од стране других библиотека.

Датум прихватања теме од НН већа: 14. 04.2009.

ДП

Датум одбране:

ДО

Чланови комисије:

КО

Председник:	др Милош Рацковић, редовни професор, ПМФ, Нови Сад
члан:	др Душан Старчевић, 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 Tešendić
Mentor/comentor: **MN**	Dušan Surla, Ph. D., professor emeritus, Faculty of Sciences, Novi Sad
Title: **TI**	Circulation system for direct consortial borrowing
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:	Author's reprint

PU

Publication place: **PP**	Faculty of Sciences, Trg Dositeja Obradovića 4, Novi Sad
Physical description: **PD**	(7/194/92/8/36/0/0) (chapters/pages/literature/tables/ pictures/graphs/appendix)
Scientific field: **SF**	Informatics
Scientific discipline: **SD**	Information Technology and Systems
Subject/ Key words: **SKW** **UC**	Circulation, library consortium, UML, pattern oriented software architecture, design patterns, NCIP
Holding data: **HD**	Library of Department of Mathematics and Informatics, Trg Dositeja Obradovića 4
Note: **N**	None
Abstract: **AB**	Modeling and implementation of circulation software system with support for direct consortial borrowing has been done. Unified software development process is used. Software architecture modeling is done using design patterns and it is shown in UML 2.0 notation. System implementation is realized in programming language Java. Subsystem for client/server communication is developed as part of circulation system. Subsystem enables transparent communication between client and server in accordance with used transport protocol. Software architecture of this subsystem is pattern oriented and it is based on combination of several design patterns. By integrating subsystem into system

BISIS, it is allowed operation of system in different network environments.

Also, subsystem is used for communication with other libraries. NCIP protocol is implemented inside the subsystem by which exchange data with different library software systems is enabled. Subsystem provides unique way of communication between client application and server, no matter whether it is its own library server or servers of other libraries. NICP service used by other libraries to access data according to NICP protocol is implemented, as well.

Accepted by the Scientific Board: **ASB**	April 14, 2009.
Defended on: **DE**	
Thesis defend board: **DB**	
President:	Miloš Racković, Ph. D., full. prof., Faculty of Sciences, Novi Sad
Member	Dušan Starčević, 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