



СОФИЙСКИ УНИВЕРСИТЕТ "Св. КЛИМЕНТ ОХРИДСКИ"  
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА

Катедра "Информационни технологии"

# Дипломна работа

**Тема:**

Разпределена система от университетски библиотеки с  
онлайн достъп

**Дипломант:** *Силвия Павлова Павлова*

**Специалност:** *Разпределени системи и мобилни технологии*

**Факултетен номер:** *M21947*

**Научен ръководител:** *доц. д-р Васил Георгиев*

София 2008 г.

# Съдържание

Съдържание.....	0
Увод.....	2
Глава 1. Архитектура на приложението и използвани технологии.....	4
1.1 Общи положения и постановка на задачата.....	4
1.2 WEB сървър.....	9
1.3 MiddleWare .....	11
1.4 База данни .....	21
Глава 2. Съществуващи реализации на онлайн разпределена библиотека .....	25
2.1 Questia.....	25
2.2 Проектът разпределена библиотека (Distributed Library Project).....	27
2.3 ULS Digital Library .....	28
2.4 ЕРА Национална библиотечна мрежа.....	30
Глава 3. Функционално описание .....	34
3.1 Структура на базата от данни.....	34
3.2 Потребителски интерфейс .....	41
3.3 Сървърни компоненти на Middleware.....	50
Глава 4. Инсталация, тестове, бъдещо развитие и надграждане.....	57
Заклучение.....	63

## Увод

Нарастващото значение на Интернет и бурното развитие на информационните технологии през последните 10-15 години от една страна и поевтиняването на компютърните системи и компоненти от друга доведоха до бурното разпространение на ИТ приложения и услуги навсякъде в ежедневието ни. И докато само преди няколко години ролята на домашния персонален компютър беше по-скоро единично и автономно звено (извършващо известен ограничен набор от дейности – като използване на програми за текстообработка, Интернет и електронна поща), то сега съществува тенденция за включване и използване на голям брой различни услуги и възможности свързани с използването на разпределени мрежови приложения. Доколкото може да се съди тази тенденция едва ли ще се промени в близко бъдеще и по-скоро се очаква ролята на разпределените системи и приложенията базирани на използването им не само в случай на бизнес приложения да нараства.

Настоящата дипломна работа има за цел реализация на разпределена библиотечна система с онлайн достъп **UniLib**. Системата трябва да може да осигури едновременна работа на голям брой потребители като им предоставя лесен и интуитивен за използване интерфейс за търсене, получаване на информация за съдържанието и заемане и връщане на книги, списания и други информационни носители. Потребителите на системата след регистрация получават достъп до съдържанието на разпределена система от университетски библиотеки, всяка от които представлява отделна база данни за наличните в нея информационни носители. Системата следва да бъде гъвкава и преносима към максимално голям брой възможни платформи и да има сравнително неголеми изисквания към хардуера на отделните си звена, както и към поддръжката на работата на приложението.

Предложената система се състои от следните части: разпределена база от

данни състояща се от известен брой (предполага се повече от 1) автономни звена (отделните несвързани библиотеки), уеб интерфейс за достъп на потребителите и разпределен междинен слой (middleware) осъществяващ функционалността и комуникациите между крайния клиент и разпределената база. Възможни са множество различни реализации на поставената задача свързани с използването на различни технологии за реализация на всяка от частите на приложението.

В глава 1 е направен преглед на различни възможни технологии (или поне най-известните към момента такива) за реализацията на приложението. Обоснован е избора свързан с използването на java базирани технологии (платформена независимост, използване на компоненти и обектно ориентиран подход при разработката на приложението) като JSP и javaBean - компонентите за презентационния уеб слой (клиентската част), RMI за комуникация между отдалечените клиентска и сървърна части и JDBC базирано взаимодействие между RMI и крайните сървърни звена на базата данни. Направено е и сравнение с алтернативни възможности като PHP, ASP, .NET както и са приведени аргументи за избора на конкретната реализация.

В глава 2 е направен кратък преглед и сравнение на съществуващи вече сходни с UniLib уеб базирани приложения на библиотеки. Направено е сравнение между различните реализации.

Глава 3 представлява функционално и структурно описание на приложението – структурата на звената на разпределената база данни (таблиците и релациите между тях), вида на клиентския интерфейс и възможните действия с него, както и описание на използваните методи, класове и пакети от различните слоеве на сървърния middleware.

Глава 4 съдържа описание на инсталацията на отделните звена от системата и възможностите за бъдещи промени на първоначалната система. Обсъдени са различни възможни тестове както и някои от направените тестове.

# Глава 1. Архитектура на приложението и използвани технологии

## 1.1 Общи положения и постановка на задачата

Съществуват различни дефиниции за разпределена система, но ние ще използваме следната:

**Разпределена система е съвкупност от независими компютри, които изглеждат на потребителя на системата като един компютър.**<sup>1</sup>

Тук има 2 аспекта: машините (звена, хостове) са самостоятелни и са свързани посредством мрежа (локална или Интернет), а що се отнася до софтуера – за потребителя работата на разпределената система е прозрачна (т.е. няма усещане че има някакъв брой звена). Разпределената система функционира като отворена система за достъп до разпределените ресурси. Прозрачността в разпределените системи има няколко аспекта<sup>2</sup> (таблица 1). Прозрачност на местоположението (*Location Transparency*) се обяснява с факта, че в една разпределена система потребителя не може да каже къде е разположен хардуера и софтуера такъв като процесори, принтери, файлове и база от данни се използват. Прозрачност на достъпа (*Access Transparency*) пък се изразява в това, че потребителя не може да различи дали използва (извиква) локален обект (компонент) или отдалечен, вследствие на това че се използват идентични оператори. Прозрачност на миграцията (*Migration Transparency*) означава, че ресурсите трябва да бъдат свободни за да се местят от едно място на друго без да им се променят имената. Прозрачност на репликирането (*Replication Transparency*) има когато операционната система е свободна да прави допълнителни копия на файлове и други ресурси без това да е видимо за потребителя. В случай на едновременен достъп от няколко потребителя на даден ресурс имаме прозрачност на достъпа до ресурси (*Concurrency Transparency*), когато те не могат да усетят това (като пример системата може да заключва

дадения ресурс когато се използва от някой и да го освобождава след това при което ресурса ще бъде достъпен последователно). Съществува още и прозрачност на мащабируемостта (*Scalability transparency*), която се изразява в това че промяната в мащаба на системата (брой хостове или обем ресурси) не изисква промяна в структурата на системата или на използваните алгоритми и софтуер и остава невидима за потребителите и програмистите.

**Таблица 1.** Прозрачности в разпределената система.

Видове	Значение
Прозрачност на местоположението	Потребителя не може да каже къде са разположени ресурсите
Прозрачност на достъпа	Потребителя използва идентични операции за достъп до локални и отдалечени ресурси
Прозрачност на миграцията	Ресурсите могат да се местят без да си променят имената
Прозрачност на репликирането	Потребителя не може да каже колко копия съществуват
Прозрачност на достъпа до ресурси	Много потребители могат да споделят ресурси автоматично
Прозрачност на мащаба	Потребителя не може да установи промяна на големината на системата

В сравнение с централизирана система (някакъв стандартен било файлов било друг вид сървър) разпределената система има редица предимства както и известен брой недостатъци.

### **Предимства на разпределената система над централизираната**

Когато се търси ценово-ефективно решение често се налага използването на голям брой евтини процесори заедно в единна система. По тази причина има тенденция за използване на подобни разпределени системи, тъй като тези системи имат много добро съотношение цена/производителност в сравнение с една голяма (*mainframe*) централизирана система. Освен това подобна съвкупност от

микропроцесори не просто дава по-добро съотношение цена/производителност отколкото аналогична централизирана система, но може да превиши по производителност коя да е от съществуващите днес централизирани системи. Освен бързодействието причина за използването на разпределена система може да е случая когато някои приложения са наследено разпределени (верига от учреждения или голяма корпорация с множество филиали – когато има съвместна работа на група от хора отдалечени един от друг). При този сценарий използването на централизирана система или пък на множество самостоятелни персонални компютри би било или невъзможно или крайно неефективно. Друго голямо потенциално предимство на разпределена система над централизирана е по-високата надеждност. При евентуална повреда на единично звено от разпределената система няма да има почти никаква вреда за останалата част от системата. Това предполага използването на разпределени системи в случаите на приложения свързани с висока надеждност (във военната сфера или за технически приложения като ядрени реактори и т.н.).

**Таблица 2.** Предимства на разпределената система над централизираната

Икономически	Микропроцесорите предлагат по-добро съотношение цена/производителност от големите системи ( <i>mainframes</i> )
Скорост	Разпределената система може да има по-голяма крайна компютърна мощ от голямата система
Наследена разпределеност	Някои приложения заместват пространствено разделените машини
Надеждност	Ако една машина се повреди, системата като цяло може да продължи да работи
Постепенно нарастване	Компютърната мощ може да бъде увеличавана постепенно

Въпреки многото си потенциални предимства разпределените системи имат и своите недостатъци. На първо място това е проблема с избора на софтуер (вида на междинния слой **middleware** реализиращ разпределените алгоритми и комуникацията между отделните звена) и платформа. Към настоящия момент съществуват редица технологии и стандарти за разпределено програмиране, като най-известните специално оптимизирани за целта<sup>3</sup> са **RPC (Remote Procedure Calls** отдалечено извикване на процедури), **RMI (Remote Method Invocation** - отдалечено извикване на метод) на Java и **.NET Remoting**.

Вторият потенциален проблем се дължи на комуникационната мрежа. Тя може да загуби съобщения, които са от критично значение и следва да бъде разработен софтуер, който да не допуска това, без това да доведе до претоварване на мрежата. Сигурността е третия голям проблем на разпределените системи. Файловете и другите ресурси трябва да са защитени от неоторизиран достъп. В едно-процесорните системи достъпа е удостоверен. Системата знае кой потребител се е логнал и може да провери дали всеки достъп е разрешен. В разпределената система, когато съобщението дойде до сървър и искайки някаква услуга, сървърът няма лесен начин да разбере от кого е съобщението. Няма име или идентификационно поле в съобщението на което може да се вярва.

Въпреки тези потенциални проблеми, преобладаващото мнение е че предимствата натежават пред недостатъците и се очаква, че разпределените системи ще станат значително по-разпространени идните години. Всъщност през последните години, повече организации ще свържат компютрите си в голяма разпределена система, за да осигурят по-добри, по-евтини и по-удобни услуги за потребителите си. Персоналните компютри за бизнес употреба вероятно няма да съществуват още дълго.

Настоящата дипломна работа има за цел реализация на разпределена



система от университетски библиотеки (**UniLib**) с онлайн достъп. Всяко едно звено на разпределената система (наричано още възел или хост) представлява базата данни на библиотеката на отделния университет и може да съдържа различни по обем и тип данни. Основно предположение е, че *системата е децентрализирана* и всички възли са равнопоставени. Второто предположение е, че *системата е динамична* – най-малкото защото е неизвестен броя на достъпните (активни) възли в дадения момент – единственото изискване, за да може да функционира цялата система е да има поне един активен възел. Следствие на двете предположения (децентрализирана и динамична система) се явява проблема със синхронизацията на разпределената система. В по-нататъшното изложение ще бъде акцентирано отново появата и решенията на проблема със синхронизацията. Достъпа на системата се осъществява посредством уеб-базиран потребителски интерфейс и дава възможност за регистриране на потребители, търсене, заемане и връщане на книги, както и справки за заетите книги и получаване на информация за намерените при търсенето книги. В известен смисъл централизиран се явява единствено достъпа до системата, тъй като се осъществява през даден уеб сървър. Клиентската част е реализирана на JSP, CSS, JavaScript. Сървърната част е по същество разпределена релационна база данни, състояща се от някакъв брой (най-малко 1) независимо работещи MySQL (или друг вид) сървъри. Всеки един от тези сървъри съхранява данните за наличните му книги и потребители на разпределената система. Общото задължително изискване към разпределената система е еднаква структура на работните таблици във всички звена на разпределената база данни. Междинния слой (**Middleware**) е реализиран в три слоя: JSP+JavaScript+CSS (презентационен слой) - javaBean взаимодействащ с уеб клиента и с Java RMI конзолно приложение базирано на JDBC и осъществяващо връзка между уеб клиента (javaBean) и всеки един от възлите на разпределената

база. Използването на трислойна архитектура увеличава сериозно възможностите за промени на функционалността както и увеличава скалируемостта (мащабируемостта) на системата т.к. позволява изнасянето на всеки слой на различна машина (хост), както и оптимизация и преразпределяне на натоварването. Недостатък на многослойните решения е по-голямата сложност на приложението (в сравнение примерно с една монолитна PHP базирана система) и възможността за комуникационни проблеми между отделните слоеве.

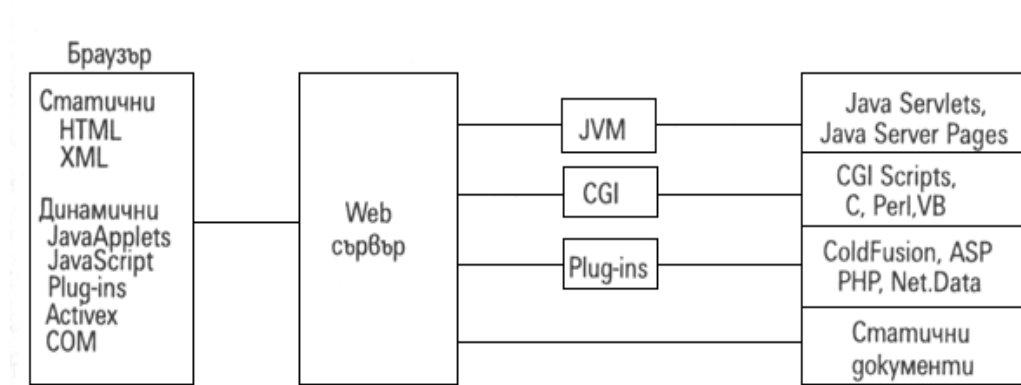
## **1.2 WEB сървър**

Избора на уеб сървър както и на останалите компоненти се обуславя от постановката на конкретната задача – изграждане на разпределена университетска библиотечна система. Впрочем избора на уеб сървър е не особено сложен – по данни от Wikipedia понастоящем (към септември 2007<sup>4</sup>) се използват главно 2 WEB сървъра – Apache (с около 50% пазарен дял) и IIS на Microsoft (приблизително 35%). Варианти с използване на далеч по-малко разпространените GWS (на Google 5% дял), lighttpd (1%), Sun-ONE-Web-Server (Sun Microsystems - по-малко от 2%) едва ли има смисъл да се коментират. Въпреки нарастващия дял на IIS на Microsoft той трудно може да бъде приемлив избор най-малкото поради факта, че работи върху една единствена операционна система (MS Windows). От друга страна Apache е свободен, мултиплатформен и с поддръжка за множеството сървърни скриптов езици, както и със сравнително неголеми изисквания към хардуера.

Ролята на уеб сървъра е да следи за клиентски заявки на даден порт (80) и да връща резултати (файлове). Взаимодействието клиент (браузър или приложение) – сървър се осъществява посредством HTTP (HyperText Transfer Protocol) протокола. HTTP заявките обикновено са под формата GET име на даден файл (може да съдържа текст, форматиращи

инструкции и графика). При получаването на такива заявки HTTP сървърът претърсва дървото на документите за искания файл и го връща на заявлия го клиент. В повечето случаи сървърът обработва едновременно множество заявки от различни клиенти. При тази схема обаче уеб сървърът връща съдържанието на статични документи. В много случаи генерирането само на статична информация е неприемливо и трябва да бъде генерирано динамично съдържание в зависимост от клиентската заявка. Съществуват няколко начина за генериране на динамично съдържание, които са свързани най-често с използване на скрипт (било от клиентската – например JavaScript, Java аплети или различни plugins като Macromedia Flash; или сървърната страна).

Apache осигурява поддръжка за работата на CGI-базирани perl, C/C++ или скриптове от обвивката (AWK, bash/tcsh и др.), също и на PHP скриптове както и java сървлети (servlets – java приложения използващи идеологията на CGI-скриптовите) и JSP (използвайки **Tomcat** контейнера на *Apache Software Foundation*).



**Фигура 1** Генериране на статично и динамично съдържание от уеб сървър с помощта клиентски plug-in и сървърни скриптове (базирани на виртуалната машина на java - **JVM**, или на интерпретатор от обвивката или на модул/plugin)

Както ще стане ясно по-нататък избора на използваната скриптова сървърна част е от особена важност за работата на цялото приложение.

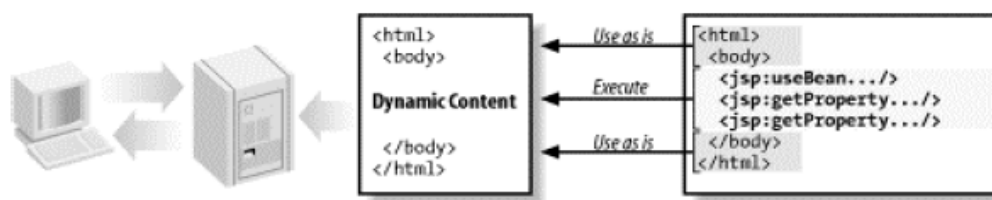
## 1.3 MiddleWare

Избора на реализацията на междинния слой между крайния клиент (уеб браузър) и отдалечените възли на разпределената база от данни на университетските библиотеки има решаващо значение за работата на цялата система. Образно казано MiddleWare се явява сърцето на разпределената система, тъй като това е реализацията на всички клиент-сървър комуникации, обработката на събитията (свързани с избора на клиентите на системата както и произтичащите резултатите), анализа и валидацията на данните и резултатите. Доста вероятно е точно работата на сървърната част да бъде тясното място в разпределената система. От друга страна избора на клиентската част на приложението също има своето значение за успеха (или неуспеха) на проекта. Използвайки клиент-сървър модела работата на цялата система може схематично да се представи като следната поредица от двойки-взаимодействия:

- уеб-клиент (броузер) – JSP + Java Bean компонента със сървър Apache в комбинация с уеб-контейнера (или приложен сървър) TomCat
- Java Bean компонента явяваща се RMI-клиент правеща заявки към отдалечен RMI-сървър (или няколко такива) при нужда от обръщане към базата данни
- Java RMI-конзолно приложение явяващо се на свой ред JDBC-клиент на наличните възли разпределената база данни. Взаимодействието между клиента и разпределения сървър (т.к. при всички случаи имаме обход в цикъл на всички „видими“ в дадения момент възли, всеки от които е отделен университет) става с помощта на JDBC

JSP е Java технология за разработване на динамични уеб страници (html, XHTML, XML или друг вид). За разлика от HTML страниците, които съдържат статично съдържание, при JSP страниците може да се променя съдържанието. JSP страниците съдържат HTML тагове, такива като при

обикновените уеб страници. Обаче, JSP страниците съдържат и специални JSP елементи, които позволяват на сървъра да вмъква динамично съдържание в страницата. JSP елементите могат да се използват за редица цели, такива като получаване на информация от базата данни, прихващане и обработка на данни въведени от потребителя и анализ и разклонения в зависимост от случая и логиката на приложението. Когато уеб-клиентът се обръща към JSP страница, сървърът изпълнява JSP елемент, обединява резултатите със статичните части на страницата, и праща динамичната страница отново към браузъра (фигура<sup>5</sup> 2).



**Фигура 2.** Създаване на динамично съдържание с JSP елементи

JSP дефинира известен брой стандартни елементи, които са полезни за уеб приложенията, такива като Java Beans компоненти, текущ контрол между страниците и споделяната информация, между заявките, страниците и потребителите. Програμισите могат да разширят синтаксиса на JSP чрез добавяне на приложно-специфични елементи, които изпълняват задачи такива като достъп до базата данни и EJB, изпращане на електронна поща, и генериране на HTML за представяне на приложно-специфични данни.

Елементите на JSP могат да бъдат разделени в следните групи:

- статично съдържание (най-често текст или графики) HTML
- JSP директиви – това са page, include и taglib които се използват за предаване на информация при обработването на страницата от JSP-контейнера (приложния сървър Tomcat в нашия конкретен случай) както и за включване на допълнителни файлове (било със

статичен код или динамични java/servlet/jsp библиотеки).

- JSP скрипт-елементи (скриптлети) и променливи. Целта им е да осъществяват контрол или съединяване на отделни части – по-същество те са фрагменти от „чист“ java код (т.е. вътре може да има например декларации, изчисления, цикли или валидация на съдържанието на jsp страницата).
- JSP действия (actions) са XML тагове които използват вградената в уеб-сървъра функционалност. Някои от стандартните action-тагове са: **<jsp:usebean>** (декларира употребата на java bean компонент), **<jsp:getProperty>** (прочита свойство на bean елемент), **<jsp:setProperty>** (задава свойство на bean елемент), **<jsp:forward>** (препраща заявката на клиента към друг jsp, сървлет или html-страница), и др. Освен стандартните тагове разработчика има възможност да дефинира и използва свои.

Важна особеност е че JSP страниците винаги се компилират до java сървлети преди да бъдат пуснати от сървъра и след като бъдат стартирани остават в кеша в резултат, на което се повишава бързодействието в сравнение например с технология като CGI, при която се изисква сървърът да зарежда интерпретатор (Perl или друг скрипт от обвивката) и скрипт всеки път когато страницата поиска. Тъй като Java сървърните страници са построени върху Java сървлети те имат достъп до целия набор от java API (Приложен Програмен Интерфейс - Application Program Interface) като например:

- JDBC (Java DataBase Connectivity)
- RMI и Corba (Common Object Request Broker Architecture)
- JNDI (Java Naming and Directory Interface)
- EJB (Enterprise Java Beans)
- JAXP (Java API for XML Processing - java API за обработка на XML)

- JMS (Java Message Service)

### Алтернативи решения на JSP

Нека да разгледаме някои други решения при създаване на динамично уеб съдържание. Някои от тези решения са подобни на JSP, докато други са потомък на по-стари технологии.

- **ASP** – е популярна технология за разработване на динамични уеб страници. Подобно на JSP в ASP се очаква разработчика на страницата да имплементира логика под формата VBScript или Jscript код, за да генерира динамична уеб страница. За по-сложен код, COM компонентите написани на езика C++ могат да бъдат извикани чрез скриптов код. Стандартната дистрибуция включва компоненти за достъп до база данни и други компоненти. Когато има заявка към ASP страница, кода от страницата се изпълнява от сървър. Комбинацията от статично и динамично съдържание се зарежда в браузър.
- **ASP.NET** последната версия на ASP, добавя известен брой нови черти. Като алтернатива на скрипта, динамичното съдържание може да бъде генерирано от уеб-форми които са HTML/XML елементи подобни на JSP таговете. Като сървърни езици могат да бъдат използвани C#, VB.NET, J# и много други. С помощта на ADO.NET може лесно да бъде реализирана връзка към голям набор от сървъри за бази данни.
- **PHP** – е уеб сървърен скриптов език. Създаден е в средата на 90-те години като проект с отворен код. Към настоящия момент има огромна популярност сред уеб-разработчиците, особено в комбинация с уеб-сървър Apache (където поддръжката се осигурява от модула *mod\_php*) и базата данни MySQL (известни като платформата XAMPP<sup>6</sup>). Както JSP и ASP, PHP позволява на автора на страницата да включва скриптов код в страниците за да

генерира динамично съдържание. PHP има синтаксис подобен на C с някои черти взаймствани от Perl, C++ и Java. Програмният код може да бъде реализиран и посредством функции, както и обектно-ориентирано (с класове). Големият брой предефинирани функции са част от PHP, такива като достъп до база данни, работа с XML, обработка на символни низове, създаване на pdf документи и изображения, криптиране и декриптиране на данни. PHP е мултиплатформен (работи върху MS Windows, Mac и практически всички UNIX/Linux системи).

- Сървлетите са java програми, които се изпълняват от уеб сървър, действат като среден слой между една заявка, идваща от Web браузър или друг HTTP клиент, и бази данни или приложения на HTTP сървър, и са на практика аналог на CGI реализиран от Java технологията. От една страна те трябва да прочетат данните, изпратени от потребителя, които обикновено са въведени в някоя уеб-форма, но също могат да идват и от Java аplet или потребителски HTTP клиент. След това те трябва да направят справка или допълнителна обработка на информацията от заявката. Друга тяхна задача е генерирането на резултатите. Този процес може да изисква комуникация с база данни, изпълнение на RMI или CORBA обръщение, изпълнение на налично приложение или директно изчисляване на отговора. След което получените резултати се форматират в документ, т.е. се поставят в HTML страница и се указва на браузъра какъв вид документ се връща (т.е. HTML). Последната стъпка е изпращането на документа обратно към клиента.

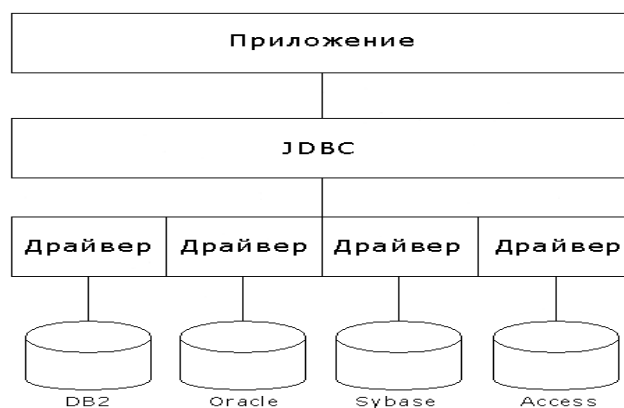
Предимства на JSP пред алтернативните решения са свързани главно с използването на Java. ASP и ASP.NET са конкурентни технологии от Microsoft и до голяма степен са еквивалентни като възможности и като



идеология за употреба (реализация на многослойни приложения с разделяне на клиентската част на скрипта от бизнес логиката на приложението и осъществяването на достъп до базата данни). Все пак към момента Java технологиите (сървлети, JSP, JSF и Struts) могат да се похвалят с по-добра преносимост както към различни операционни системи така и към различни уеб-сървъри (макар че .NET технологията вече може да се ползва върху повече от една платформа примерно с помощта на проекта Mono <http://www.mono-project.com>). В сравнение с PHP Java технологиите (а също и .NET) вероятно биха загубили като бързодействие т.к. това е цената която се плаща при използването на виртуална машина. Обаче при реализацията на големи корпоративни приложения работещи разпределено в хетерогенна среда бързодействието, което предлага PHP (което е може би най-атрактивната му черта в сравнение примерно с jsp) ще се окаже далеч не най-важно. Възможностите които дават както Java така и .NET технологиите за повторно използвани компоненти и промяна и разширяване на функционалността на уеб-приложенията биха били решаващи при избора на технология за реализация.

Следваща стъпка в конкретната реализация на приложението е използването на JavaBeans компонентите. JavaBeans (в последствие и EJB) на Sun и Component Object Model и Distributed Component Object Model (COM / DCOM) на Microsoft са конкуриращи се компонентни модели. Идеята на компонентния модел представлява когато части от софтуера разработени от различни програмисти могат да работят заедно. Компонентите също така могат да се използват многократно. Bean-елементът представлява клас, който следва специфични правила за именуване на методите си, може да поддържа собственото си съществуване и е пакетирен, така че да може лесно да се разпространява и използва. Чрез поставянето на компонентите bean-елементи на едно място, с добре дефиниран интерфейс, може лесно да се изграждат

приложения с използването на компонентите и свързването им чрез събития и допълнителен съединителен Java код. Ако разберем функциите, които трябва да се предоставят и функциите, предоставени от компонентите, проектирането и изграждането на системата се свежда до свързването на частите в правилната последователност. Нуждата от запазване на състоянието на уеб-приложението (в конкретния случай – разпределената система от университетски библиотеки) води до използването на база данни. JDBC е съвкупност от API, които дават възможност на Java приложения да се свържат със стандартни отраслови и собствени системи за управление на бази данни. Използвайки JDBC, приложенията могат да извличат и съхраняват информация използвайки оператори на езика за структурно запитване (**SQL** - Structured Query Language), както и самата машина-сървър за база данни. Отчитайки наличието на огромно множество бази данни, архитектурното предизвикателство е да се предостави прост интерфейс за свързване дори със сложни бази данни. За програмиста, интерфейса към бази данни трябва да бъде същият независимо от вида на базите данни, към които искаме да се свържем. Фигура 3<sup>7</sup> показва основна архитектура на JDBC приложение.



**Фигура 3** Архитектура на JDBC за връзка посредством различни драйвери към различни сървъри за бази данни

JDBC е Java базиран интерфейс който може да осъществява достъп до какъвто и да е вид таблични данни и особено ако е използвана

релационна база данни. JDBC работи в следната поредица от стъпки:

1. Осъществява връзка с базата данни (без да се интересува от конкретния и вид – използва се конкретен драйвер от класа ***DriverManager*** и се инициира начало на сесия чрез интерфейса ***Connection***)
2. Изпраща заявка тип SQL ***SELECT/INSERT/UPDATE/DELETE*** (за извличане, въвеждане, промяна или изтриване на данни в дадена таблица от базата данни)
3. Получават и се обработват резултатите получени от базата данни в отговор на заявката

### Обектът ***DriverManager***

В сърцето на JDBC стои DriverManager. След като един драйвер е инсталиран той се използва от Java обекта посредством DriverManager. DriverManager групира драйверите заедно, така че множество бази данни да могат да бъдат използвани по еднотипен начин. Той предоставя общ интерфейс за обекта на JDBC драйвера без да е нужно потопяване в спецификата на конкретната база данни.

Драйверът е отговорен за създаването и реализирането на обектите ***Connection***, ***Statement*** и ***ResultSet*** за специфичната база данни. Като прави това, приложенията написани използвайки DriverManager са изолирани от различните детайли на отделните бази данни, както и от бъдещи подобрения и промени в самите реализации (фигура 4).



**Фигура 4** Разделяне на приложението от интерфейсите ***Connection***, ***Statement*** и ***ResultSet*** с помощта на DriverManager

***Connection*** интерфейс за база данни.

Обектът **Connection** е отговорен за създаване на връзка между Системата за управление на база данни (**RDBMS** - relational database management system) и Java приложението. Чрез отделянето му от **DriverManager**, драйверът може да изолира базата данни от специфични части на реализацията. Той също така, дава възможност на програмиста да избере правилния драйвер за изискваното приложение.

Методът *Connection.getConnection* приема като параметър URL и дава възможност на JDBC обекта да използва различни драйвери в зависимост от ситуацията и дава на приложението средства, с които да определи специфичната база данни, към която трябва да си свърже.

Обект **Statement** на базата данни.

Statement съдържа запитване (заявка), написано на SQL и дава възможност на JDBC обекта да състави серии от стъпки за търсене на информация в база данни. Използвайки Connection, Statement може да изпрати към базата данни и да получи ResultSet.

**ResultSet** е контейнер за серии от редове и колони, получени в резултат от извикване на **Statement**. Използвайки методите на **ResultSet**, JDBC обектът може да премине през всеки ред в резултатната съвкупност. Отделни полета на колоните могат да се извлекат използвайки **get** методите от **ResultSet**. Колоните могат да се специфицират по името на полето или техния индекс (например *getInt(„id“)* или *getInt(1)*).

Технологията RMI

RMI е Java интерфейс за предоставяне на обектен еквивалент на RPC. RMI позволява описание и извикване на отдалечени разпределени обекти използвайки Java. Тези обекти могат да бъдат нови Java обекти или могат да бъдат само обвивка около съществуващото API. RMI е базиран на използването на интерфейс (**remote interface** наследник на *java.rmi.Remote*) чрез който клиента взаимодейства със сървър.

Извикването на отдалечен обект (сървър) става с помощта на RMI регистър (*java.rmi.registry*), който съдържа информация (под формата на уникални имена) за предлаганите сървърни обекти от даден хост. Комуникацията между сървъра и клиента става с помощта на бланки (*stubs*) от страна на клиента и скелети (*skeletons*) от страна на сървъра. Използването на RMI улеснява много реализирането на разпределено програмиране, тъй като извикването на локални и отдалечени обекти става абсолютно аналогично с единствената разлика, че се използва междинен интерфейс описващ методите които предоставя отдалечения сървър. RMI има няколко предимства пред традиционните RPC системи тъй като той е част от обектно-ориентираните системи. Освен лесната употреба RMI предлага и пълна обектна сериализация (не се предоставя от технологиите RPC и CORBA) и преносимост от произволна към произволна JVM (за JDK не по-ниска от версия 1.2).

Основни предимства на RMI:

- обектно-ориентиран – може да се предават като параметри и да се връщат като стойности обекти от произволен тип (т.е. поддържа се сериализация на обекти за разлика примерно от RPC където клиента и сървъра трябва да декомпонират обектите до примитиви) без писане на никакъв допълнителен код
- мобилно поведение – клиента и сървъра могат да си разменят ролите използвайки един и същ *remote interface*
- позволява използването на шаблони за обектно-ориентиран дизайн и реализирането на многослойни приложения
- сигурност – използва вградения в java модел на сигурността (с използването на стандартния java SecurityManager или производния RMISecurityManager) при пълна свобода независимо от страна на клиента и сървъра за различни политики (security

policy) съобразно нуждите на конкретната инсталация

- връзка към съществуващи системи – взаимодействие с помощта на **JNI (Java Native Interface)** позволява стартиране на външни приложения за осигуряване на платформено-зависими решения – например драйвери на C/C++ и др.)
- лесен за използване и писане и написан веднъж, работи навсякъде вследствие преносимостта осигурена от java – RMI приложение може да бъде стартирано върху произволна JVM. В комбинация с JNI – RMI може да се използва върху произволна сървърна система, а в комбинация с JDBC може да се използва за връзка към коя да е база данни
- разпределен събирач на боклука (**garbage collector**) – след като бъде създадена и използвана референцията към отдалечен обект тя се разрушава автоматично от JVM без клиентското приложение да се грижи за това

## 1.4 База данни

База от данни се нарича всяка организирана колекция (съвкупност) от данни<sup>8</sup>. За да се представят данните от реалния свят в паметта на компютрите, се използват модели за описание на данните. Най-разпространените модели използвани при базите данни са<sup>9</sup>:

йерархичен, мрежови, релационен, обектно-релационен и др. При йерархичния модел представянето на данните може да се опише с помощта на дървовидна структура (пример за този модел е файловата система състояща се от директории и файлове). Йерархичния модел е бил широко използван в миналото, но вече няма особено приложение. Мрежовия модел представлява обобщение на дървовидния при което базата от данни се представя като ориентиран граф – пример, за което е WWW (World Wide Web). Релационният модел е предложен през 1969 г.

от д-р Едгар Ф. Код и може да се смята за общоприет към настоящия момент. При релационния модел, данните се съхраняват в таблици, като е възможно между отделните таблици да се задават **релации** (връзки). Всяка таблица е съставена от записи, които представляват редовете на таблицата. Записите се състоят от полета (клетки от таблицата), които са най-малкото количество информация, което може да бъде обработвано в дадена база от данни. Обработката и управлението на релационните бази данни става с помощта на програмния език **SQL** (Structured Query Language). В момента има няколко стандарта (първите са **ANSI** от 1986 и **ISO** от 1987), като най-разпространените са – **SQL-92** (ISO), който се поддържа от всички релационни бази от данни и **SQL-99**, който все повече навлиза в употреба. SQL предлага конструкции за създаване, промяна, изтриване на таблици и други обекти (индекси, тригери) в базата от данни. Тази част от езика се нарича **DDL** (Data Definition Language) и обхваща команди като **CREATE**, **ALTER**, **DROP** на таблици. Другата част от езика е наречена **DML** (Data Manipulation Language) и предлага команди за търсене извличане, добавяне и изтриване на данни. Към нея спадат команди като **SELECT**, **INSERT**, **UPDATE** и **DELETE**. В допълнение на SQL съществуват и различни разширения на езика които осигуряват възможности за писане на процедури – така например PL/SQL (на Oracle), Transact-SQL (на Microsoft/Sybase) и др. Някои от най-разпространените системи за управление на бази данни (Relational Database Management System – RDBMS) или другояче казано сървъри за управление на бази от данни (СУБД) са:

- **Oracle**
- **IBM DB2**
- **Microsoft SQL Server**
- **PostgreSQL**
- **MySQL**

Първите три са комерсиални продукти, докато последните два са с

отворен код. От изброените само MS SQL-сървър не е мултиплатформен и следователно неподходящ за университетска система. За много надежден (и близък по архитектура до Oracle) се смята PostgreSQL, а за най-разпространен и сравнително надежден може да се приеме MySQL. Изискванията към разпределената база данни е да е максимално мощна (поради което тук не се разглеждат бази данни от типа на MS Access, Excel, FoxPro) и гъвкава, както и да е независима от платформата (предполага се че в различните университети може да имат различни както операционни сървърни системи така и хардуерни/софтуерни решения) и лесна за поддръжка. Конкретната реализация се базира на MySQL 5.x, като обаче това не е особено ограничение, понеже използването на JDBC технологията позволява миграция към някоя от другите СУБД поддържащи SQL и много потребители (например PostgreSQL или дори Oracle).

Проектирането на базата от данни е етапа, при който след анализ на структурата и логическата свързаност на данните те се представят под формата на таблици и връзки (релации) между тях. Релациите се реализират с помощта ключове. Първичен ключ (primary key) е колона от таблицата, която уникално идентифицира даден неин ред(запис). Два записа са различни тогава когато им съответстват различни стойности на първичния ключ. Външен ключ (foreign key) се нарича копие на първичен ключ на дадена таблица, което е включено в структурата на друга таблица. Имаме дефинирана релация между две таблици когато стойността на външния ключ от едната съответства на първичния ключ от другата. Връзките между таблици могат да бъдат 1 към 1, 1 към много или много към много (така наречената множественост на връзките показваща колко записа от една таблица съответстват на запис в друга). За описание на дадена база данни се използва релационна схема, която е съвкупността от всички таблици и релациите между тях. Релационните схеми се изобразяват графично чрез т.нар. E/R (Entity/Relationship –



същност/връзка) диаграми и са общоприетия начин за визуализация и моделиране на представеното съдържание. Релационният модел налага изискване за нормализация (с цел оптимизация при работа на СУБД) на структурата на базата данни.

## Глава 2. Съществуващи реализации на онлайн разпределена библиотека

### 2.1 Questia

Questia (<http://www.questia.com/aboutQuestia/about.html>) е първата онлайн библиотека осигуряваща 24 часа в денонощието, 7 дни в седмицата достъп до голямата световна онлайн колекция от книги и научни списания в хуманитарната и социална сфера. Всеки потребител може да търси по дадена фраза или всяка една дума в заглавията (или авторите, съдържанието, издателството или ключови думи) във всички книги, списания и научни статии в колекцията (фигура 5).

The screenshot displays the Questia website's search interface. At the top, the Questia logo is accompanied by the tagline "faster, easier research!". Navigation tabs include HOME, SEARCH, READ, WORK, and CLASSROOM. A "FREE Trial!" button and a "Login" button are also present. Below the navigation bar, a "Search the Library:" section features a search input field and a "Search" button. A note advises users to "Put exact phrases in quotes". Below this, a "Search in:" section lists various media types with checkboxes: Books, Journals, Magazines, Newspapers, Encyclopedia, Research Topics, and an "Uncheck All" option. A banner for "WATCH VIDEOS" promotes the use of the Videos Toolbar, with a "Get Started" button. The "Advanced Search" section includes fields for Title, Author, Subject, Publisher, and Contents, each with a dropdown menu set to "All the words". It also includes fields for Publication Year (After, Before, Exact) and Lexile (Minimum, Maximum, Exact). A "Media Types" section at the bottom has checkboxes for Books, Journals, Magazines, Newspapers, Encyclopedia, and Research Topics, along with an "Uncheck All" option. "Clear" and "Search" buttons are located at the bottom of the Advanced Search section. On the right side, a "Search Tips" box provides instructions on using quotation marks and Boolean operators. Below the tips, a promotional box encourages users to "Register today for FREE access to top books in your selected subject" with a "Click to Continue" button.

**Фигура 5** Разширено търсене в Questia по име, автор, тема, издателство, година на издаване и тип на изданието

Това богато научно съдържание, подбрано от професионални

библиотекари, не може да се срещне на друго място в Интернет. Студенти, ученици от средните училища и Интернет потребители от всички възрасти намират библиотеката Questia за безценен онлайн ресурс. Всеки потребител, който прави проучване или се интересува от теми като хуманитарни и социални науки ще намери заглавия тук.

Какво ви предлага библиотеката Questia:

- в най-голямата световна онлайн колекция от книги, научни списания и статии (<http://www.questia.com/aboutQuestia/exploreLibrary.html>) може да се търси по дума, фраза, заглавие, автор или тема.
- съдържа книги и списания на повече от 300 одобрени издателства (<http://www.questia.com/aboutQuestia/partnersPub.html>) в хуманитарната и социалната област.
- неограничен достъп до книгите и статиите в колекцията, като няма значение колко други потребители четат същите материали
- на разположение 24 часа в денонощието, 7 дни в седмицата
- лесна за използване като средството за създаване на библиография

Части от уеб интерфейса на библиотеката са реализирани на JSP. Като цяло Questia е отлична онлайн библиотечна система що се отнася до съдържание и начин на употреба. Обаче по всичко изглежда, че тя не е разпределена система – за разлика от предложения проект за университетска разпределена система от библиотеки UniLib. Все пак създаването на UniLib до голяма степен е вдъхновено от Questia, с уговорката, че ще обслужва група от самостоятелни университетски библиотеки. Търсенето в UniLib е доста сходно с това на Questia (във вариант просто и разширено търсене с множество параметри като ключови думи, издател, теми и т.н.). Разлика има в това, че UniLib по замисъл работи и като заемна – тоест съхранява информация за потребителите и заетите от тях материали в разпределената система. Приликите са по-скоро визуални, като може да се отбележи че Questia предлага разширено търсене с повече възможности от UniLib.

## 2.2 Проектът разпределена библиотека (Distributed Library Project)

Проектът разпределена библиотека (<http://library.burngreave.net/>) е библиотека за книги, аудио и видео. Това е експеримент в създаването на общност и споделянето на информация. Потребителите си създават акаунт с пълен списък от интереси, след което правят списък с предпочитаните от тях книги, аудио и видео. След това потребителят има достъп до множеството от ресурси в колекцията. Има възможности за търсене (<http://library.burngreave.net/search.php?type=book>) по автор и заглавие; по автор, заглавие и рецензия; само по автор или заглавие, както и да разглеждат индивидуални колекции и да се намират хора със сходни интереси (фигура 6).



User	Author	Title	Section	Status	Distance	
thomman1	a. a	Learning Perl	Computers	Available	0.00mi	Show
silvipavlova	a. a	Learning Perl	Computers	Available	0.00mi	Show
thomman1	a. w	802.11 Wireless Networks: The Definitive Guide	Computers	Pending	0.00mi	Show
Damian	AAA...	Free eBook reader + 13,000 free eBooks! (see notes)	Other	Pending	3692.41mi	Show
dionian		Advisory Service for Squatters, Squatters Handbook (12th Edition)	Other	Available	3694.05mi	Show
Damian	Alexander, Games	The Essential Spike Milligan	Humor	Available	3692.41mi	Show
Damian	Altman, Nathaniel	The Nonviolent Revolution	Philosophy	Available	3692.41mi	Show
sheffieff	Anderson, Luke	Genetic Engineering, Food and Our Environment: A Brief Guide	Politics	Available	3694.69mi	Show
sheffieff	Athanasios, Tom	Slow Reckoning	Politics	Available	3694.69mi	Show
aland	Austen, Jane	Sense and Sensibility (Wordsworth Classics)	Literature	Pending	3694.29mi	Show
sheffieff	Baarschers, William H.	Eco-facts and Eco-fiction: Understanding the Environmental Debate	Politics	Available	3694.69mi	Show
sovereignborg	bach, richard	illusions	Other	Available	0.00mi	Show
sheffieff	Bach, Richard	Illusions: The Adventures of a Reluctant Messiah	Religion	Available	3694.69mi	Show
tdenta	bach, richard	jonathan livingston seagull	Philosophy	Pending	3631.52mi	Show
silvipavlova	Back, Iaian	The Bridge	Other	Available	0.00mi	Show
ntahall	Back, Ken	Assertiveness at Work: A Practical Guide to Handling Awkward Situations	Other	Available	3694.30mi	Show
Damian	Banks, Iain	The Bridge	Fiction	Available	3692.41mi	Show
Damian	Banks, Iain	The Player of Games	Sci-Fi	Available	3692.41mi	Show
Damian	Banks, Iain	Consider Phlebas (Orbit Books)	Sci-Fi	Available	3692.41mi	Show
cygnus_at_tfr	Banks, Iain	the wasp factory	Other	Available	0.00mi	Show

Фигура 6 Търсене в проекта Разпределена библиотека

Съществува и възможност за разширено търсене (Basic/Advanced search) по автор, заглавие, рецензия и собственик в комбинация с жанр. Всеки

има достъп до потребителските рецензии и всеки може да напише своя такава. Всеки материал (книга, аудио или видео) може да бъде заеман за период от 2, 7, 14 или 30 дни.

Особеност на интерфейса е разделението в търсенето на книги, аудио и видео – в проекта Unilib типа на носителя на информация е реализиран по различен начин. Проекта разпределена библиотека е реализиран с помощта на Apache, PHP 4, Perl и MySQL 4 (технологии с отворен код – като версиите на PHP и MySQL са вече леко остарели). Проектът определено е добър пример за истински разпределено приложение базирано на жизнеспособни (и при това свободни!) технологии. Все пак в сравнение с предложението от нас проект за университетска библиотека реализацията на търсенето и заемането определено отстъпва като простота и функционалност. Съществуващата възможност за рецензиране на предоставените материали е сериозен плюс, т.к. никой от останалите примери (в това число и Unilib) не предлага подобна възможност. Въпреки лекотата и простота на разгледания пример, в случай на едно бъдещо разширяване (или промяна) на функционалността вероятно би било по-трудно за реализация в сравнение с Unilib, за което ще стане дума и по-нататък.

## **2.3 ULS Digital Library**

Университета в Питсбърг е основан през 1787 като малко, частно училище. 220 години по-късно университета еволюира в международен център за познание и изследване. Университетската библиотечна система е 26-тата най-голяма академична библиотека в Северна Америка и 16-тата сред по-престижните народни библиотеки, членки на съюза на Американските университети. Здравната библиотека поддържа образователните, изследователските и клиничните дейности на Медицинското училище и университет в Питсбърг като предлага

актуална информация. Правната библиотека на университета се използва както от студенти и преподавателите на правното училище, така и от всеки, който има интерес в областта. Университетската онлайн библиотечна система **ULS** (University Library System <http://pittcat.pitt.edu/>; <http://www.library.pitt.edu/>) поддържа обучението и изследователската дейност на университета и обслужва потребителите чрез създаване и предаване на уеб-достъпна дигитална колекция. Като начало, дигиталната библиотека се фокусира върху създаването на текст-базирани колекции, чиито дигитализация и достъпност в уеб пространството би подкрепило изследователската дейност на учени, юристи, преподаватели и студенти. Библиотеката предлага също така фотографии, карти, ръкописи, пощенски картички, аудио-визуални материали и биографични каталози. Търсене може да се реализира по няколко начина: по автор, заглавие и тема или по ключови думи (фигура 7). Има предложени търсене на нови материали и на материали за курсовете които предлага университета. Част от предлаганите услуги са предназначени само за студентите на университета и изискват удостоверяване на самоличността (аутентификация).

**PITTCat**  
Online Catalog of the University of Pittsburgh Libraries

E-Z Borrow | Help  
Find Articles | Other Libraries  
Ask-A-Librarian | Library Home Page

- **Search**  
By author name, title, subject, or call number.
- **Keyword Search**  
Using forms and drop-down menus.
- **Course Reserves**  
Search course reserve materials by instructor's name, department, or course number.
- **New Materials**  
View a list of library materials recently added to PITTCat.

**Quick Search**

Find:

By:

**My Account**

Log in to renew books and see a record of what you have checked out.

Questions? Contact us by phone at 412-648-3330 or by email at [feedback@library.pitt.edu](mailto:feedback@library.pitt.edu)

© 2005 University Library System, University of Pittsburgh

**Фигура 7** Търсене в библиотеката на университета в Питсбърг

Системата **ULS** определено е университетска и осъществява търсене в разпределена система от библиотеки. По това тя има съществена

прилика с предложения проект Unilib. Разликите са в реализацията и отчасти в подхода които е използван. Търсенето в **ULS** е реализирано с помощта на CGI (вече доста остаряла технология) и демонстрира доста впечатляващо (поне на вид) бързодействие. Като цяло обаче **ULS** е доста нехомогенна система – така например търсенето е пръснато в много страници и липсва единично разширено търсене по набор от параметри (присъщо на повечето подобни системи), както и няма добра структура на представянето на съдържанието – има пръснати части които съдържат както информация за звената от библиотеката така и многобройни различни варианти за търсене. **ULS** по-скоро е създадена за общо улеснение на студентите на университета в Питсбърг, отколкото като специализирана библиотечна система.

## **2.4 EPA Национална библиотечна мрежа**

Националната мрежа от библиотеки EPA (Environmental Protection Agency – националната агенция за защита на околната среда <http://www.epa.gov/natlibra/>) на САЩ осигурява ефективен достъп до информацията в областите наука, техника, управление и политика. Библиотеката се състои от няколко релационни бази данни, които съдържат голямо количество издания. Като използвате онлайн EPA библиотеката може да намерите голямо разнообразие от книги, сведения, научни списания, аудио-визуални материали. Материалите се обновяват на всеки две седмици и може да се търси по много начини – по комбинация от заглавие, автор, ключова дума. Към основният каталог има допълнителна специална колекция:

- Национален каталог, съдържащ списък от книги, документи и списания, се използва от съкровищницата на EPA и други специализирани библиотеки.
- GroundWater Ecosystems Restoration е колекция намираща се в Ада,



Окалхома, която съдържа научна и специализирана информация отнасяща се до опазване на околната среда. Информацията включва стратегии и технологии за опазване на околната среда.

- База данни съдържаща исторически издания в Бостън, Денвър и Сан Франциско

Всеки може да търси в онлайн библиотеката (**OLS** - Online Library System) и да намери информация по специфична тема (в конкретна библиотека която е част от цялата система), или да търси в общия каталога (<http://www.epa.gov/natlibra/ols.htm> т.е. от всички библиотеки фигура 8). Търсенето е реализирано в два варианта – основно и аналог на вече разгледаните тип „разширено търсене“ - като и двата варианта са комбинация от различни параметри – автор, заглавие, ключови думи, година на издаване и издателство и др. Освен това в случая на разширено търсене потребителя може да избере вида на подредбата на резултатите, както и различни типове търсене (с включване на логическо И/ИЛИ и отрицание).

U.S. ENVIRONMENTAL PROTECTION AGENCY

**Libraries and Repositories**

Contact Us Search:  All EPA  This Area

You are here: [EPA Home](#) » [EPA National Library Network](#) » Online Library System (OLS)

**OLS : Search EPA National Catalog**

[OLS Home](#) [Help](#) [Search By Command](#) [Serial Search](#)

Sort By:  Year Published  Sort Order:  Descend

Limit Search to a Specific Library:  Search all libraries

Collections: ☐ all ☐ books ☐ documents ☐ journals ☐ reference ☐ audiovisual

Limit to a Specific Database:  Library Holdings and NTIS Microfiche

Search:  Title  FOR  all the words

AND  Keywords  FOR  all the words

AND  Author  FOR  all the words

AND  Report Number  FOR

AND  Year Published  FOR  =

AND  Date Modified  FOR  =

You Can Also Search EPA Special Collections Via OLS:

[National Service Center for Environmental Publications](#) | [GroundWater Ecosystems Restoration](#) | [Air Pollution Technical Info Center](#)  
[Region 1 Information](#) | [Region 9 Information \(Historical Material\)](#)

**Фигура 8** Разширено търсене в **OLS** по комбинация от множество параметри и набор от варианти за сортиране на резултатите

Когато потребителя намери желания от него материал, има няколко начина да го получи. Ако публикацията е собственост на библиотеките



ЕРА, онлайн библиотечната система ще съдържа информация. В полето Holdings има четири-буквен код, който показва собственика на библиотеката. Името на библиотеката също е написано. В някои случаи търсените издания могат да бъдат достъпни в онлайн вариант (например като връзка към pdf-файл).

Като цяло онлайн версията на ЕРА е отличен пример (вероятно най-добрия от приложените тук) за разпределена библиотечна система. Това е впечатляващ с мащабите си проект за онлайн базирана библиотечна система (организацията ЕРА се състои от над 17000 човека и се финансира от правителството на САЩ) със сериозни амбиции за бъдещо развитие. Трудно може да бъде направено сравнение между OLS на ЕРА и проекта за университетска библиотека UniLib, но все пак може да се каже че са сходни като възможности за търсене (OLS има само еквивалента на разширено търсене който е приблизително съизмерим като възможности с UniLib). При OLS няма заемане (което всъщност освен в UniLib го има само в проекта разпределена библиотека), но пък има вариант за търсене в отнапред зададено звено от разпределената система (конкретна избрана библиотека) – което разбира се би могло да бъде съществен плюс за потенциалния потребител. Принципно това би могло да се добави като бъдеща функционалност и в случая на UniLib (в момента не е направено), с уговорката, че разпределената система в случая на UniLib е „по-динамична“ - докато възлите на ЕРА са винаги достъпни (тоест системата е статична по отношение на броя на възлите си) при UniLib броя на възлите е променлива величина (т.е. има възможност част от библиотеките да се „изключват“ в някои моменти от време). Иначе изобщо казано като мащаб UniLib и OLS са несравними, най-малкото защото втората има звена в огромна част от територията на САЩ, докато проекта UniLib има за цел да обслужва някакъв брой университетски библиотеки (разделени и географски отдалечени една от друга) с цел удобство при търсене (като се предполага че вероятния

потребител е студент).

**Таблица 3:** Сравнение на разглежданите реализации

Библиотека	Разпределена	Регистрация	Търсене	Заемане / Четене онлайн	Информация за изданието	Статус	Рецензии
Questia	не	не изисква	основно и разширено	четене; няма заемане	не	не	не
DLP	да	изисква	основно и разширено	заемане и четене	не	да	да
ULS	да	не изисква	основно и разширено	не поддържа	да	не	не
OLS	да	не изисква	основно и разширено	четене	да	не	не
Unilib	да	изисква	основно и разширено	заемане	да	да	не

Предложения преглед на реализации на разпределени библиотеки в никакъв случай не претендира за изчерпателност, а по-скоро има информативен характер за наличните днес системи и мястото на UniLib между тях. По наше мнение проекта UniLib определено може да се конкурира със съществуващите в момента подобни системи.

## Глава 3. Функционално описание

### 3.1 Структура на базата от данни

Проектирането на базата от данни е етапа, в който се преминава от логическото представяне на данните към представяне във формата на таблици и релациите между тях (ясно е че използваме релационния модел). В конкретния случай се проектира модел за единичен възел на разпределената система, които съдържа данни за книги или изобщо по-общо казано информационни носители – могат да бъдат още списания или друг вид носители като например хипервръзки или оптични носители – CD или DVD; както и данни за потребителите на въпросния възел. Следователно основните обекти се явяват книгите (таблица **books**) и регистрираните потребителите (таблица **users**). Двете таблици са в релация много към много, т.к. дадена книга може да е взета от няколко потребителя както и един потребител може да е заел известен брой (в общия случай повече от една) книги. Това налага използването на междинна таблица **loans** реализираща релацията много към много.

Необходимите таблици за които и да е от възлите на системата са:

- books
- users
- loans
- mediatypes
- languages
- db\_options

Възприета е следната конвенция за наименованията таблиците и техните полета:

- ✓ Имената на таблиците са винаги с множествено число понеже се отнасят за множество записи
- ✓ Името на първичния ключ се е обозначава с “id”

Структура на таблиците

➤ **Таблица “books”:**

Тази таблица съдържа наличните към момента книги (или по-общо информационни носители от даден тип) за всеки един възел от системата. Полетата са следните:

*id int(10) unsigned NOT NULL PRIMARY KEY*

първичен ключ на таблицата

*title varchar(100) NOT NULL*

заглавието на книгата – задължително поле

*author varchar(50)*

автора на произведението

*mediaType int(10) unsigned NOT NULL*

задължително поле явяващо се външен ключ за типа носител – книга, вестник, списание и т.н.; реализира релация едно към много между **mediatypes** и **books**

*genre varchar(100)*

жанра (в случай на книги и списания) - незадължително

*publisher varchar(512)*

името на издателството – незадължително поле

*keywords varchar(512)*

съдържа ключови думи по които може да се извършва търсене – незадължително поле

*language int(10) unsigned NOT NULL (default 1)*

външен ключ за езика на които е носителя – свързва **books** и **languages**; съответно е задължително и има стойност по подразбиране отговаряща на английски език (id = 1)

Всички стрингови полета (*varchar*) използват кодировка (encoding) UTF-8 от една страна явяваща се естествена за java и от друга даваща удобство за многонационална поддръжка на езици в базата данни. Всички първични ключове (полета *id* от тип *int(10) unsigned* за всички таблици, в случая на **RDBMS MySQL** са **auto\_increment** – т.е. са с

автоматично нарастване; а в случай на **Oracle** например биха нараствали с помощта на *sequences*).

Следва да се отбележи, че полетата отговарящи за жанр и издателство биха могли да се превърнат във външни ключове към нови таблици (особено е в сила за жанр) при една бъдеща реализация.

➤ **Таблица “users”:**

Основното предназначение на тази таблица е да съхранява информация за регистрираните потребители на библиотечната система. Таблицата се състои от следните полета:

*id int(10) unsigned NOT NULL PRIMARY KEY*

първичен ключ на таблицата

*login varchar(20) UNIQUE NOT NULL*

уникалното име на потребителя което го идентифицира (в комбинация с паролата му) – няколко потребителя могат да имат еднакви пароли или собствени/фамилни имена но не и еднакви **login** (полето е глобален за системата идентификатор на потребителя, докато **users. id** е „локален“ за дадения възел)

*password varchar(20) NOT NULL*

тук се записва паролата на потребителя – задължително поле

*firstName varchar(50) NOT NULL*

собствено име на потребителя – задължително поле

*lastName varchar(50) NOT NULL*

фамилия на потребителя – задължително поле

*Address varchar(100)*

адрес на потребителя – незадължително поле

*Email varchar(30)*

полето съдържа адреса на електронната поща на потребителя – незадължително поле

➤ **Таблица “loans”:**

Чрез тази таблица се осъществява релацията “много към много” между таблици “**users**” и “**books**”. Съдържа първичен ключ и два външни ключа за връзка между потребител и заета от него книга, както и поле в което се запазва информацията за момента на заемането (от тип *datetime*). Това е таблицата която отразява заемането и връщането на предоставените от съответния възел на разпределената библиотека материали (книги, списания и т.н.)

Има следните полета:

*id int(10) unsigned NOT NULL PRIMARY KEY*

първичен ключ на таблицата

*bookId int(10) unsigned NOT NULL*

външен ключ към таблицата “books” определящ еднозначно книгата в дадения възел

*userId int(10) unsigned NOT NULL*

външен ключ към таблицата “users” указващ id на потребителя (уникално в рамките на конкретния възел, но безполезно в глобалната разпределена система където такова е *users.login*)

*loaned datetime NOT NULL*

съдържа датата и часа на заемането – взема се стойността на системното време в момента на въвеждането (в MySQL се използва функцията *NOW()*, а примерно в ORACLE *SYSDATE*) и се използва за справка кога е заета дадена книга

➤ **Таблица “db\_options”:**

Тази таблица е системна. Тя няма директно отношение към връзките между книги и потребители, но има съществено значение за работата на цялата разпределена система. Смисъла на таблицата е да идентифицира конкретния възел на разпределената система (по идея тази таблица се прочита в момента на инициализация на

разпределената система и се запазва името на съответния възел в масив от имената на активните възли). За разлика от останалите таблици тази таблица съдържа само един запис с името и информация за възела. Съдържа следните полета:

*id int(10) unsigned NOT NULL PRIMARY KEY*

*name varchar(100) NOT NULL*

тук се съхранява името на възела (библиотеката), което се визуализира при показването на намерените книги от конкретната библиотека - полето е задължително и се визуализира в страниците свързани с търсене или заемане/връщане на книги и е идентификатор за дадения възел от гледна точка на външните слоеве на приложението (уеб и RMI слоевете)

*server varchar(100) NOT NULL*

би могло да съдържа различна информация (например описание – на операционната система или версията на СУБД или информация за хардуера) за сървъра

*serverip varchar(30)*

съдържа IP адреса на сървъра – незадължително поле

*note varchar(1024)*

съдържа незадължителни общи бележки за конкретния възел

➤ **Таблица “mediatypes”:**

В тази таблица се записва типа на носителя – книги, вестници, списания, енциклопедии или други. Съдържанието в тази таблица трябва да кореспондира с формата за разширено търсене в уеб-интерфейса и следователно при промяна на съдържанието на тази таблица (добавяне на нови видове носители в **някой** от възлите или премахване на някои видове за **всички** възли) би трябвало да доведе до промяна вида на уеб-интерфейса (както и на функционалността понеже типовете търсени носители се явяват

параметри подавани от презентационния слой към следващите слоеве на приложението). Полетата са:

*id int(10) unsigned NOT NULL PRIMARY KEY*

първичен ключ на таблицата

*code varchar(5) NOT NULL UNIQUE*

уникален код за съответния тип (например book, jour, enc съответно за книги, списания и енциклопедии) – задължително и уникално поле

*name varchar(100) NOT NULL*

тук се съхранява името на типа на носителя – задължително поле

*note varchar(1024)*

съдържа общи бележки за дадения тип – незадължително поле

➤ **Таблица “languages”:**

В тази таблица се записва езика на носителя. Полетата са:

*id int(10) unsigned NOT NULL PRIMARY KEY*

първичен ключ на таблицата

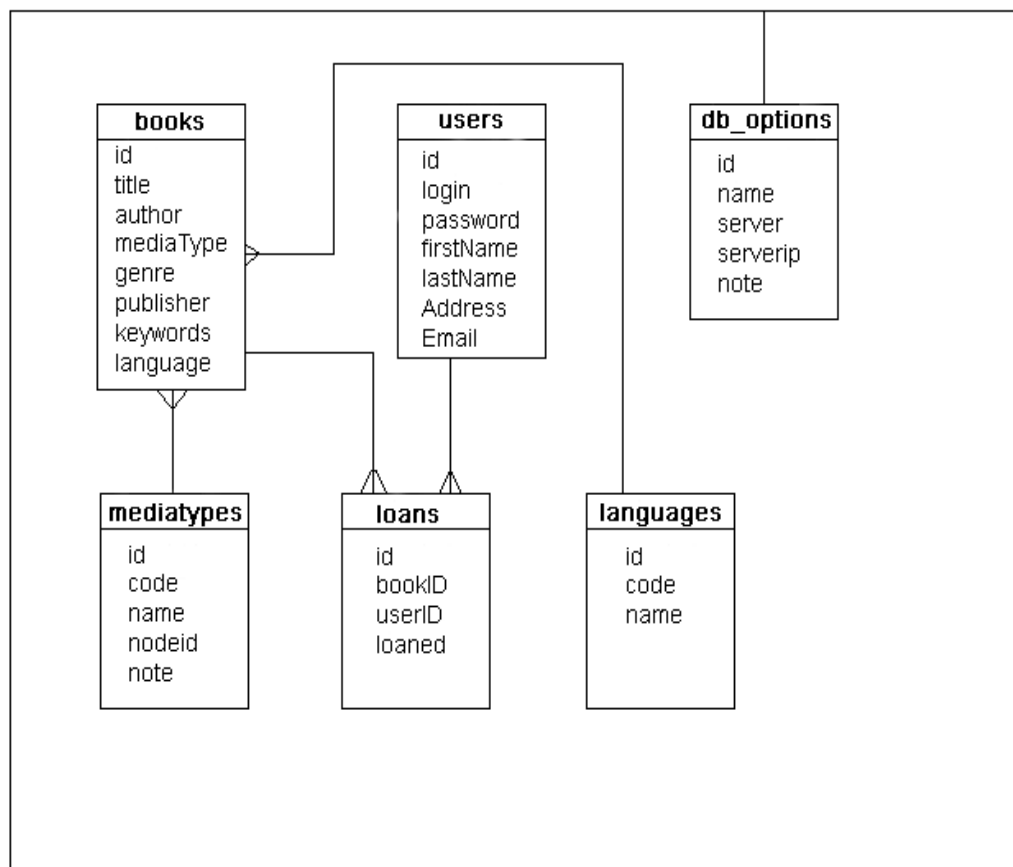
*code varchar(3) NOT NULL UNIQUE*

уникален код за съответния език съгласно ISO 639 стандарта – задължително и уникално поле

*name varchar(100) NOT NULL*

тук се съхранява наименованието на езика – задължително поле





**Фигура 9** Схема на структурата на базата данни

### 3.2 Потребителски интерфейс

Началната страница на онлайн библиотеката UniLib представлява уеб форма (*index.jsp*) за идентификация на потребителя. Очаква се той да се представи въвеждайки потребителското си име и парола, за да бъде допуснат в системата.

The image shows a login form titled "Login:" in bold black text. Below the title, there are two input fields. The first is labeled "User name:" and contains the text "silvi". The second is labeled "Password:" and contains ten black dots. Below these fields is a yellow button with a gradient and the word "login" in red. At the bottom of the form, there is a link labeled "Registration" in bold black text. The entire form is set against a light yellow background with a subtle gradient.

**Фигура 10** Форма за влизане в приложението

В случай, че потребителя не е регистриран той може да използва връзка “**Registration**”, която води към формата за регистрация (*registration.jsp*). Задължително условие е да бъдат въведени и двете полета (потребителско име и парола). Случаите, които се третираят като грешки (валидацията се извършва от *checkLogin.jsp*) са когато някое от полетата не е въведено (тоест първоначално се проверяват за съдържание полетата и едва след което се прави заявка за проверка в разпределената база чрез обход на всички налични звена) или когато е въведена двойка име-парола, които не могат да бъдат намерени в таблицата *users* в никой от достъпните възли на системата. В случай на успешна идентификация (т.е. въвеждане на „правилна“ двойка потребителско име – парола) се създава сесиен параметър (*userLogin* съответстващ на полето *login* от

таблицата **users**) съдържащ потребителското име, чиято стойност е уникален идентификатор за дадения потребител на системата. Така създаденият сесиен параметър се препредава между отделните jsp страници до приключването на сесията на потребителя. Това се прави с две цели – от една страна някои от следващите страници може да имат нужда да идентифицират логналия се потребител (в случай че се наложи заемане или връщане на книги) и от друга страна всяка от следващите страници проверява за съдържанието на параметъра и ако няма такова прекратява работата и извежда съобщение което препраща обратно към формата за идентификация, с цел предпазване от не оторизиран достъп (което по правило се реализира или с използване на параметри от сесията или с използване на бисквитки – *cookies*).

Стартовата страница има и още една „невидима“ за клиента функция – а именно да инициализира java bean (като се укаже името на класа и пакета които се използват) отговарящ за връзката към разпределената система:

```
<%@page language="java" contentType="text/html"
    pageEncoding="UTF-8" session="true" import="libClient.*"%>
<jsp:useBean id="lib" scope="session" class="libClient.unilib"/>
```

При първоначално зареждане на страницата извикването на <jsp:useBean> довежда до стартиране на конструктора по подразбиране на java bean компонента (*unilib*), който осъществява връзка със следващите слоеве на приложението (отдалечения сървър комуникиращ с базата данни с помощта на JDBC). Обхвата на bean елемента е в рамките на сесията, т.е. до момента в който потребителя е логнат. Инициализирания в началната страница javabean елемент (който всъщност е специален java клас с полета за съхраняване на данни и методи за реализиране на функционалност) с *id "lib"* се препредава във всички последващи в сесията страници и реално осъществява по-голямата част от работата на приложението свързана с осъществяването

на заявки към базата данни (търсене, заемане, връщане с помощта на имплементираните методи за взаимодействие със следващите слоеве), както и съхранение на част от данните свързани или с потребителя или в резултат от комуникацията с разпределената база (например резултати от търсене; имена на библиотеките – възли на разпределената система и т.н.). След инициализацията на lib-елемента се проверява дали има активни възли и в случай, че няма се променя вида на страницата и се извежда съобщение, че системата временно не функционира (това не би следвало да се случва никога в реална ситуация, но би могло да се случи ако евентуално бъде прекъсната комуникацията между уеб-сървър и разпределената система – RMI сървър или към всички отделни звена на базата данни).



**Registration form:**

First name: \*

Last name: \*

User name: \*

Pass: \*

Repeat Pass: \*

Address:

e-mail:

**Фигура 11** Регистрация на потребителите

За да бъде регистриран нов потребител той трябва да въведе името и фамилията си, електронната си поща и адреса си, както и избрано от него потребителско име и парола (Фигура 11 - използва се формата *regform.jsp*, която на свой ред се обработва от *registration.jsp*). Полетата

отбелязани със звездички са задължителни – всички освен адрес и e-mail. Потребителското име може да съдържа латиница, кирилица, „\_“ и цифрите от 0 до 9, а паролата трябва да бъде не по-дълга от 20 символа и не по-къса от 6. Валидацията се извършва с помощта на JavaScript от клиентската страна (във формата *regform.jsp*, с цел ако има явна грешка да не се прави излишно обръщение към следващите слоеве, което се осъществява от *registration.jsp*) и при коректно въведени задължителни полета се прави добавяне на нов потребител в разпределената база данни. В случаи на некоректно въведени (или невъведени) данни се извежда съобщение за полетата, които не са попълнени правилно. Ако бъде въведен e-mail адрес той трябва да бъде с валиден формат.

След проверката на полетата на *regform.jsp* управлението се предава към *registration.jsp* и се извиква процедура за добавяне на нов потребител **lib.addUser**. Преди да се добави новия потребител се проверява дали вече няма друг регистриран със същото потребителско име. В случай че потребителското име вече е използвано (т.е. има запис в таблиците *users* в някое от звената на Unilib) се извежда съобщение, че потребителското име вече е регистрирано и следва да се избере различно.

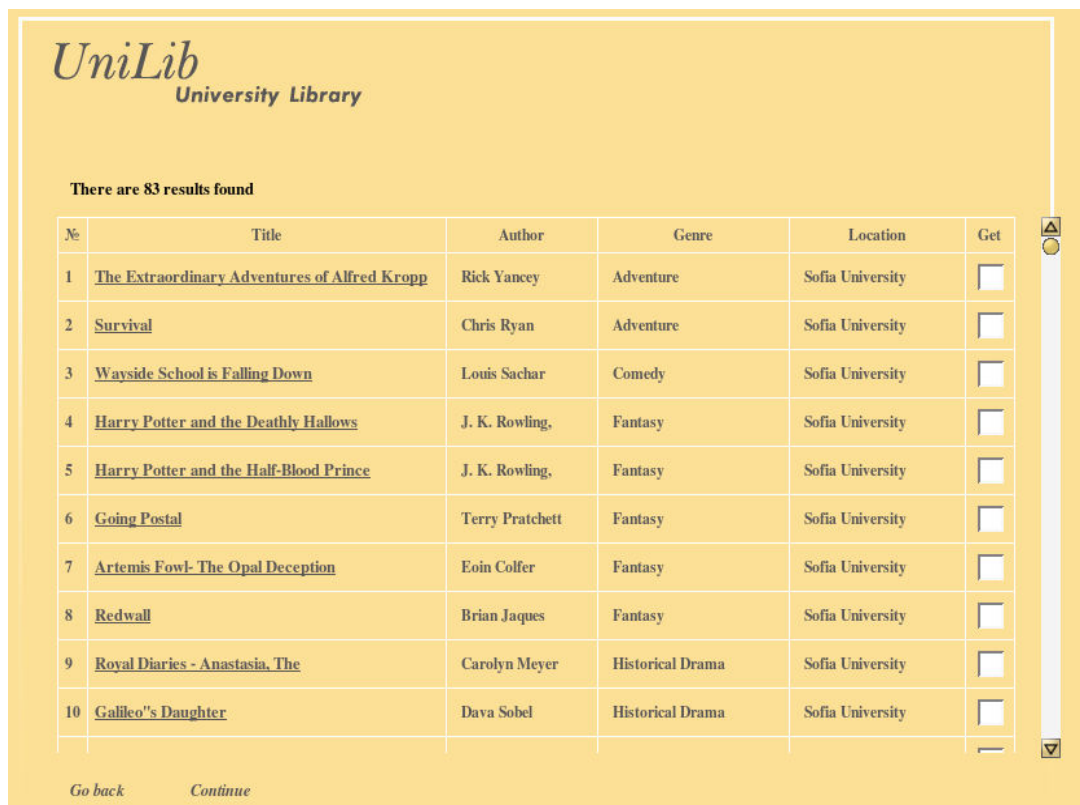
При успешна регистрация, потребителят влиза направо в страницата за търсене (*search.jsp*) на приложението (фигура 12).



**Фигура 12.** Търсене по име на творбата

Съществува възможност да търси по два начина – използвайки точката

от менюто **SEARCH** (формата *search.jsp*) или разширено търсене (**ADVANCED SEARCH** – формата *adsearch.jsp*). Когато се въведе желаното заглавие на книга, при натискане на бутон **Search** излиза списък с намерените резултати както и библиотеките (възлите на UniLib), от които могат да бъдат заемани те (фигура 13 - *searchResults.jsp*). В случай че не бъде открито заглавие което да отговаря на избора на потребителя се извежда съобщение и линк за връщане към формата за търсене.



There are 83 results found

№	Title	Author	Genre	Location	Get
1	<a href="#">The Extraordinary Adventures of Alfred Kropp</a>	Rick Yancey	Adventure	Sofia University	<input type="checkbox"/>
2	<a href="#">Survival</a>	Chris Ryan	Adventure	Sofia University	<input type="checkbox"/>
3	<a href="#">Wayside School is Falling Down</a>	Louis Sachar	Comedy	Sofia University	<input type="checkbox"/>
4	<a href="#">Harry Potter and the Deathly Hallows</a>	J. K. Rowling,	Fantasy	Sofia University	<input type="checkbox"/>
5	<a href="#">Harry Potter and the Half-Blood Prince</a>	J. K. Rowling,	Fantasy	Sofia University	<input type="checkbox"/>
6	<a href="#">Going Postal</a>	Terry Pratchett	Fantasy	Sofia University	<input type="checkbox"/>
7	<a href="#">Artemis Fowl: The Opal Deception</a>	Eoin Colfer	Fantasy	Sofia University	<input type="checkbox"/>
8	<a href="#">Redwall</a>	Brian Jaques	Fantasy	Sofia University	<input type="checkbox"/>
9	<a href="#">Royal Diaries - Anastasia, The</a>	Carolyn Meyer	Historical Drama	Sofia University	<input type="checkbox"/>
10	<a href="#">Galileo's Daughter</a>	Dava Sobel	Historical Drama	Sofia University	<input type="checkbox"/>

Go back      Continue

**Фигура 13.** Намерени резултати от търсенето (просто или разширено)

В случая на разширено търсене (фигура 14) потребителите могат да търсят по заглавие, автор, жанр, издателство и ключови думи в комбинация с типа на информационните носители - дали да са книги, списания, вестници, енциклопедии или оптични носители (CD/DVD). Вида на формата за разширено търсене кореспондира на съдържанието на таблиците *books* и *mediatypes*, следователно при добавяне на нов тип носители в базата данни – това трябва да се отрази във вида на уеб-

интерфейса за разширено търсене (*adsearch.jsp*). И двете форми за търсене използват една и съща универсална сървърна процедура (*searchBook*), като разликата е че при разширеното търсене се използва пълния набор от възможни параметри за търсене (наименованието, автора и т.н.).

UniLib  
University Library

Welcome: silvi

SEARCH ADVANCED SEARCH CURRENT STATUS

Advanced Search

Title: Love ☒

Author: ☐

Subject: ☐

Publisher: ☐

Keywords: ☐

Media Types

☒ Books ☐ Journals ☐ Magazines

☐ Newspapers ☐ Encyclopedia ☐ CD/DVD

search clear

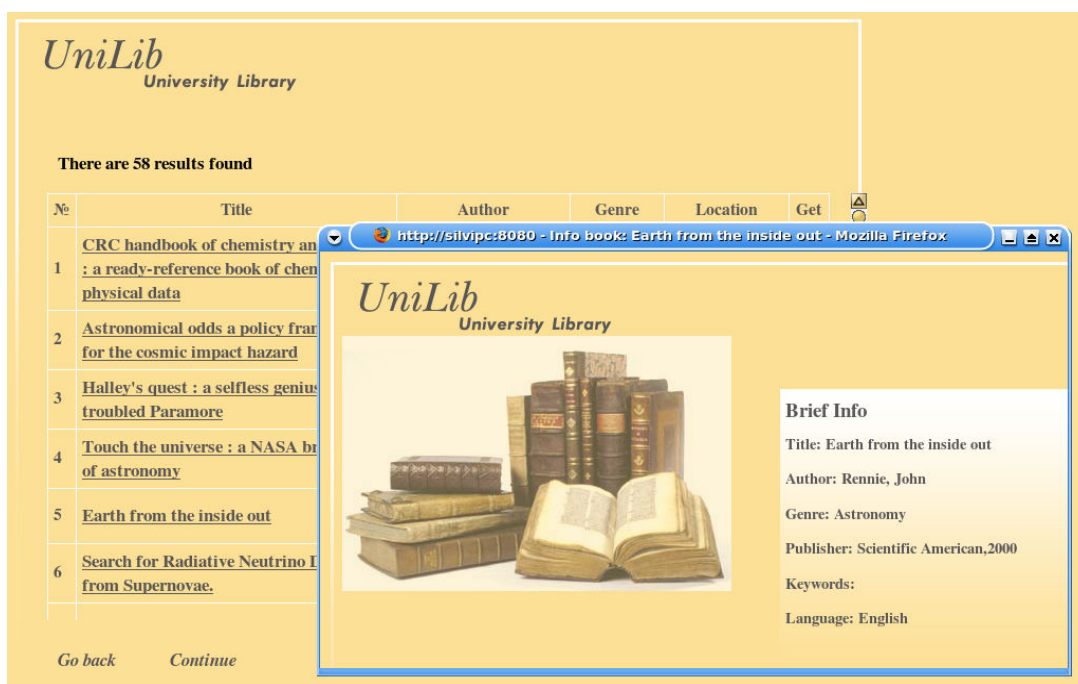
logout

**Фигура 14** Разширено търсене по заглавие, автор, тема, издателство и ключови думи.

След като потребителят осъществи някой от двата вида търсене и получи и прегледа резултата от търсенето (резултата генериран от *searchResults.jsp* изглежда подобно на фигура 13 независимо от вида на търсенето и съдържа информация за наименованието, автора, жанра и възела-библиотека в който се намира) потребителят може да реши да направи ново търсене или да заеме някаква част от намерените резултати от търсенето. За представянето на резултатите е използвана JavaScript функция за реализиране на скролер в случай че резултатите са повече от 10 (обичайна практика в подобни ситуации е да се показват някакъв

отнапред фиксиран брой заглавия – в случая това не е реализирано по този начин).

В случай че желае по-детайлна информация за някое заглавие (*searchResults.jsp* предоставя само малка част от съдържанието на базата) е предвидена възможност потребителя да може да отвори информационен прозорец (фигура 15), в който се извежда по-детайлна налична информация от базата данни за избраното издание. Реализацията е с помощта на JavaScript функция, която приема за входни параметри стойности на променливи генерирани от *searchResults.jsp*. По този начин в този случай се генерира динамично съдържание в резултат от *jsp* + *JavaScript* взаимодействие.



**Фигура 15.** Получаване на детайлна информация за издание


Потребителите имат две възможности – да заемат част от материалите (тогава *searchResults.jsp* е входна форма за следващата *doGetBooks.jsp*) или да изберат ново търсене. В случай, че желае ново търсене потребителя може да се върне с помощта на линка ***Go back***, при което той ще бъде върнат във вида търсене от който е дошъл – за целта се използва сесийна променлива, която „помни“ коя е последната посетена



jsp форма. В случай че няма намерени резултати от търсенето се извежда съобщение и линк за връщане към формата от която потребителя е дошъл (отново с използването на сесийна променлива).

В случай, че желае да заеме някоя (или няколко) от намерените книги потребителя трябва да ги маркира (използвайки съответния чекбокс от колоната *Get* – фигура 13) и да използва ***Continue***. Заемане се осъществява в случай, че са маркирани книги (ако е използван ***Continue*** без да е маркирано нищо от списъка след проверка на сесийна променлива се извежда предупредително съобщение с помощта на JavaScript функцията *alert()*) след като се направи проверка дали вече не са заети от потребителя (от дадения възел на библиотеката).

Потребителите имат възможност да следят текущият си статус (фигура 16) – т.е. да проверяват какви книги вече са заели използвайки точката от менюто CURRENT STATUS, която извиква *status.jsp*. В този случай сървърната процедура (***userLoans***) има за входен параметър името на потребителя което е негов глобален идентификатор за цялата система (вследствие на факта че полето *login* от таблицата ***users*** е уникално глобално за системата; докато *uID* от ***loans*** е уникално само за конкретния възел). Работата на процедурата се изразява в обход на системата възел по възел и проверка дали има запис в таблицата ***loans*** запис съответстващ на *uID* за дадения потребител. В случай, че не бъдат открити записи се извежда съобщение че потребителя няма заети книги. Тук е място където може да възникне проблем свързан с липсата на синхронизация – ако част от възлите на разпределената система, от които потребителя е заел книги в предишен момент от време, са неактивни няма да има информация за заетите от тези звена на разпределена система материали (част от записите в таблиците ***loans*** ще бъдат недостъпни).



№	Title	Author	Loaned from	Location	Return
1	Bored of the Rings	Henry N Beard	26/02/08 14:51	Sofia University	<input type="checkbox"/>
2	Moby Dick	Herman Melville	26/02/08 14:51	Sofia University	<input type="checkbox"/>
3	At the Sign of the Sugared Plum	Mary Hooper	26/02/08 13:31	Sofia University	<input type="checkbox"/>
4	The Tale of Peter Rabbit	Beatrix Potter	08/01/08 16:38	USA University Blagoevgrad	<input type="checkbox"/>
5	Lord of the Rings	J.R.Tolkein	08/01/08 16:39	Tech.Uni - city of Gabrovo	<input type="checkbox"/>
6	The Hobbit	J.R.Tolkein	14/12/07 17:29	Tech.Uni - city of Gabrovo	<input type="checkbox"/>
7	About a boy	Nick Hornby	08/01/08 16:40	Tech.Uni - city of Gabrovo	<input type="checkbox"/>
8	Men at Arms	Terry Pratchett	26/02/08 14:51	Varna Technical University	<input type="checkbox"/>

[Go back](#)
[Continue](#)

**Фигура 16.** Списък със заетите от потребителя издания (*status.jsp*)

В случай че потребителя желае да върне дадено издание може да го направи докато се намира във формата статус (фигура 16 – *status.jsp* е входна форма за *doGetBooks.jsp* която реализира връщане – т.е. изтрива кореспондиращия запис от таблицата *loans*) като маркира съответния чекбокс от колонката *Return*. Аналогично на случая при заемането се прави проверка за маркирани записи (и се извежда предупредително съобщение с *alert* ако няма). След като бъде реализирано връщането потребителя се връща в статус където се опреснява състоянието му (върнатите издания изчезват от списъка и евентуално излиза съобщение че няма заети материали). В момента който реши да напусне системата потребителя може да го направи с линка *Logout*. При напускането на системата се изтрива съдържанието на сесийната променлива *userLogin* - *session.setAttribute("userLogin", new String())*. По този начин, за да влезе отново в системата потребителя трябва да се идентифицира с името и паролата си (и не се допуска в рамките на сесията след напускане, друг човек от същия компютър да влезе непозволено в системата).

### 3.3 Сървърни компоненти на Middleware

Програмирането от сървърната страна на приложението е реализирано в няколко слоя и съответно се описва от няколко групи класове. Частта, която взаимодейства с разпределената база използва JDBC и реализира SQL за DML (Data Manipulation Language) операции като заявки за извличане на записи (***SELECT***) или пък за промяна на данните (тип ***INSERT*** и ***DELETE***). Типа операции за извличане се реализират с помощта на ***executeQuery(String query)*** метода на Statement, а типа промяна (или изтриване или въвеждане) с помощта на ***executeUpdate(String statement)***. За функционалността на Unilib е необходимо да може да се извличат данни от повечето от таблиците и да може да се въвеждат нови записи или да се трият записи от таблицата **loans** (INSERT /DELETE – когато се взема нова книга се прави INSERT, а когато се връща книга се изтрива кореспондиращия запис; за момента функционално няма изискване за тип UPDATE на някоя от таблиците). За целта са създадени методите:

- **public int getNumber**(*String query, int nodeNumb, StringBuffer errorMessage*)
- **public String getString**(*String query, int nodeNumb, StringBuffer errorMessage*)
- **public void updateLib**(*String query, int nodeNumb, StringBuffer errorMessage*)
- **public void search**(*String bookName, String authorName, String gen, String publ, int[] mediaType, int nodeNumb, StringBuffer errorMessage*)

Първите 2 метода извличат данни (съответно целочислени и символни), а третия извършва произволна манипулация от тип INSERT/UPDATE/DELETE и съответно не връща стойност. Параметрите на методите са SQL-стринг за изпълнение, идентификатор (*nodeNumb*)

на възела – отличителна черта на тази група методи (прилагат се в цикъл по всички възли) и евентуално върнато съобщение за грешка.

Използването на тези методи е крайния етап (от гледна точка на крайния сървър – отделните бази данни по възлите на системата), когато като параметър на методите се предава директно SQL-стринг за изпълнение в който е заложена логиката от слоевете свързани с клиентската част и идентификация (ID) на възела.

Четвъртият метод реализира операция търсене на книга и връща неявно масив (или по-точно списък *ArrayList* които е инстанция на *java.util.ArrayList* от обекти *<bookSet>*) от специално дефиниран тип обекти - класа ***bookSet***, който има за цел да дефинира структура от данни (с полета заглавие, автор, жанр, издателство, ключови думи, идентификатор за възела в който се намира и т.н.; както и конвенционалните методи за bean компонентите – *get/set* методи) съдържащи параметрите (свойствата) на търсените книги. Решението да се избере такъв вариант се дължи от една страна на многообразието на входните параметри (заглавие или автор или ключови думи както и тип на носителя или комбинация от всичко изброено) както на възможностите за различен набор от резултати за различните частни случаи. Съществува и допълнителен аргумент за направения избор и той, е че **свойствата** на категорията книга (както и на **критериите за търсене!**) могат лесно да бъдат променяни понеже са капсулирани съответно в отделен обект (и съответно метод ***search***). Тук ясно може да се усети мощта на обектно-ориентирания подход, при който може да се раздели вида (типа и количеството) на данните от конкретно реализираната функционалност.

Всички обръщания към разпределената база от данни се осъществяват посредством някои (или комбинация от няколко) от тези методи. Спомагателни методи за търсене и манипулиране в базата данни са още:

- **public int getUserID** (*String login, int nodeNumb, StringBuffer errorMessage*)
- **public void getBook** (*String login, int nodeID, int bookID, StringBuffer errorMessage*)
- **public void returnBook** (*String login, int nodeID, int bookID, StringBuffer errorMessage*)

Първият метод връща уникалния идентификатор на даден потребител (*uID* – първичния ключ от таблицата **users**), за съответния възел на разпределената система, по зададено име на потребителя (т.е. по зададеното уникално за цялата система име намира уникалното *id* на потребителя в дадената библиотека). Вторият и третият метод реализират операциите заемане и връщане на книга от потребител (зададен с уникалното си потребителско име - *login*), като за целта използват **getUserID** за да установят „локалния“ идентификатор който е първичния ключ *uID*. Техни параметри се явяват името на потребителя и *id*-то на книгата (от таблицата **books**) което вече е известно (или в резултат на предходно търсене или ако вече е заета се съдържа в таблицата **loans**). Вземането и връщането на книга реализира INSERT или DELETE в таблицата **loans** с помощта на **updateLib** – като в двата различни случая се композират съответните SQL заявки чиито параметри са *bookID* и *uID*. В случая на заемане на книга от даден възел се проверява дали вече дадената книга не е заета от дадения потребител (т.е. преди да се направи INSERT в **loans** се проверява дали няма запис за конкретните *uID* и *bookID* и ако има не се допуска повторно заемане на книгата).

Друга група методи са методите описващи взаимодействието между клиента (достъпващ през първия презентационен front-end слой на централизирания уеб графичен интерфейс) и разпределената сървърна част (третия слой back-end отговарящ за съхранението на данни), която

вече беше разгледана. Това взаимодействие е центъра на междинния слой и може да бъде реализирано като RMI, Corba клиент-сървър или като взаимодействие между два компонента (entity/session javabeen или EJB и се явява средния бизнес слой). Във всички случаи клиент-сървър взаимодействията (jsp bean-елемент – отдалечен сървър) е удобно да се представят под формата на интерфейс (*remote interface*):

```
public interface LibInterface{  
  
    public boolean validUser(String loginName, String pass);  
  
    public boolean checkUserName(String loginName, StringBuffer  
errorMessage);  
  
    public void addUser (String loginName, String pass, String firstName,  
String lastName, String address, String email, StringBuffer errLog);  
  
    public void userLoans (String userLogin, StringBuffer errMsg);  
  
    public void searchBook(String mName, String mAuthor, int[] mType,  
String mGenre, String mPublisher, StringBuffer errorMessage);  
  
    public void getBook (String login, int nodeID, int bookID,  
StringBuffer errorMessage);  
  
    public void returnBook (String login, int nodeID, int bookID,  
StringBuffer errorMessage);  
  
    public String[] nodesNames();  
  
    public ArrayList <bookSet> getResult();  
  
}
```

Предложения интерфейс осигурява връзката между RMI клиента, който е част от уеб интерфейса и методите за манипулация на базата данни (back-end слоя). Интерфейса реализира обръщение от един клиент (логнатия през интернет потребител) към множеството (всичките

достъпни в дадения момент) възли на разпределената система, с изключение на случаите на вземане или връщане на книга когато е ясно в кой възел се намира книгата (тоест вече е реализирано търсене в цикъл по всички възли). Следователно само в този случай обръщението на клиента е към конкретен възел (идентифициран с помощта на параметъра `nodeID`). Първият метод установява идентичността на даден потребител – а именно дали двойката потребителско име и парола може да бъде открито в някой активен възел на системата. Метода има и „скрита“ функционалност – в случай че потребителското име бъде открито само в част от активните възли (а не във всички, което се установява с проверка на стойността на брояч при обхода на възлите) се стартира процедурата **replicateUser**(*loginName*, *pass*), която добавя валидното потребителско име (както и останалите данни за потребителя – парола, имена и адрес и email ако има въведени такива) във възлите където то не е намерено. По този начин се цели постигането на синхронизация на разпределената система – макар да е невъзможна пълна синхронизация, в случай на лоша свързаност на цялата система (ако има често редуване на едни активни с други неактивни възли). Втория метод проверява дали дадено потребителско име е свободно за използване от нов потребител на системата. Метода се използва в случай на регистрация на нов потребител и се извиква като спомагателен от **addUser**. Последния добавя потребител в случай че желаното потребителско име не е вече използвано (обаче ако се окаже, че е използвано в неактивните възли на разпределената система – това би довело до нежелан конфликт – евентуално двама потребители с еднакво име). Пътищата за по-добра синхронизация вероятно следва да се търсят в допълнителна осигурителна система (примерно регулярна размяна на SQL скриптове между всички възли – активни и неактивни през определен период време).

Метода **userLoans** осъществява търсене на заети книги от даден



потребител (тоест осигурява се функционалността на *status.jsp*). Отново се използва обход в цикъл по възлите **getUserID** за да се направи съответствие между глобалното потребителско име *login* и локалното за даден възел *id* на потребителя и се проверява дали има записи в **loans** за дадения потребител. В случай че бъдат намерени записи те се записват в списък (*ArrayList <bookSet>*) от обекти, които може да бъде получен от клиента с помощта на метода **getResult()** (чиято единствена цел е да осигурява комуникация). Тук е от съществено значение факта, че RMI позволява обектна сериализация (в предвид конкретната задача оптимално е да се прехвърлят колекция от обекти от тип *bookSet*).

Метода **searchBook** реализира търсене на книги по различен критерий (осигурен от *search.jsp* или *adsearch.jsp*) отново в цикъл по наличните възли. Има за входни параметри заглавието, автора, типа на носителя (масив отговарящ на съдържанието на таблицата *mediatypes*), издателя и евентуално ключови думи. Метода проверява кои от подадените му параметри съдържат стойности (понеже потребителя може и да не използва всички параметри, а и има вариант на търсене само по заглавие - *search.jsp*) и композира *SQL SELECT* със съответната *WHERE* клауза съответстваща на параметрите.

Метода **nodesNames()** извлича имената на възлите (библиотеките от разпределената система) който сървърът зарежда при инициализацията на системата.

Методите **getBook** и **returnBook** осигуряват заемане на избрани от потребителя книги (след като той е реализирал или търсене или проверка на статуса си – тоест е извикал **searchBook/userLoans** и е определил кои заглавия от кои възли ще заема или връща).

Реализацията на отделните компоненти е разделена в три пакета – за клиентската част (*javaBean* – RMI клиент) *libClient*, за сървърната *libServer* (реализиращ средния слой) и за *remote* интерфейса (съдържащ



описания вече *LibInterface* както и класовете *bookSet* и *unilibUser* които са обектното представяне на данните свързани с книгите и потребителите и имплементират интерфейса *Serializable*) осигуряващ взаимодействието клиент – отдалечен сървър. Инсталацията на системата изисква дефинирането на известен брой параметри свързани с местоположението на сървъра (параметъра *java.rmi.server.codebase*) и използването на RMI регистъра и системата за именуване (*java.rmi.Naming*), както и конфигурирането на достъпа на сървъра до възлите на разпределената база. От гледна на потребителите на системата най-удобно би било конфигурацията да става с помощта на файлове. За целта е създаден клас *configReader* (използващ потока *java.io.FileReader*, и класовете *java.io.BufferedReader* и *java.util.Scanner*) който осъществява прочитане на ASCII файл по зададен шаблон. С негова помощ може динамично да се зареждат необходимите параметри (това се отнася особено до конфигурацията на отделните университетски бази данни – *DBNodes.conf*, която може да търпи чести изменения – примерно смяна на паролите за достъп или добавяне/промяна на имена на възли) при стартирането на сървърите вместо да се спират и прекомпилират при всяка една промяна на параметър на системата.

## Глава 4. Инсталация, тестове, бъдещо развитие и надграждане

В първоначалната реализация UniLib е инсталирана на 2 до 3 домашни персонални компютъра със сравнително скромни хардуерни възможности (RAM памет 1-2 GB и процесори AMD Athlon/Athlon64 1 – 2.5 GHz). Използвани са операционни системи Linux (Slackware11/12, Debian Lenny 4.0 и Slax server 5.x) с уеб сървър Apache 1.3.37 и jsp/servlet контейнер Tomcat 5.5.23/6.0.14 и Sun Java SDK 5.0 (или 6.0). Сървър за базите данни беше използван MySQL 5.0.x (5.0.24, 5.0.37) както и като алтернатива PostgreSQL 8.2.4 (с тестова цел). Беше направен и тест с включване на възел от разпределената система с операционна система MS Windows XP с инсталиран XAMPP 1.6.4 (с MySQL 5.0.45).

Инсталацията се свежда до инсталиране на компонентите осигуряващи работата на трите слоя на приложението. Това включва SQL скриптовете за създаване на работните таблици и потребител (схема) с права за SELECT/INSERT/DELETE операции по възлите на системата (MySQL); конфигурация на приложния сървър (Tomcat осигуряващ работата на презентационния слой) и на системата RMI клиент-сървър. Конфигурацията на приложния и RMI сървърите изисква редактирането на няколко файла съдържащи информация за възлите на разпределената система (имена на сървърите, на схемите и пароли за достъп до базите данни) както и за политиката за java сигурността свързана с използването на мениджъра по сигурността (Security Manager за приложния сървър Tomcat и RMISecurityManager за RMI клиент-сървър системата). Конфигурационните файлове за отделните части на системата са:

- *catalina.policy* – детайлно описание на параметрите (политиките Java Security Policy – трябва да се укажат *java.net.SocketPermission*, *java.io.FilePermission*, *java.lang.RuntimePermission* и

*java.util.PropertyPermission*) на вградения механизъм на java сигурността. Трябва да се разреши достъп на определени портове (80 за http комуникация с RMI сървър при първоначалното изтеглянето на stub бланки от клиента и на 1099 или друг при по-нататъшното взаимодействие), както права за четене/запис върху файловата система (*FilePermission* на определено място). Файла се намира в директорията с конфигурационните файлове на Tomcat (*\$CATALINA\_HOME/conf*) където са и файловете *tomcat-users.xml* (за дефиниране на роли – например администратор на Tomcat), *server.xml* (параметри на контейнера), *web.xml* и други.

- *DBNodes.conf* съдържа информация за възлите на системата – прочита се при инициализация на RMI сървър. Всеки ред от този файл има структура име на сървър-схема (*//hostname/DBname*), име на потребител на базата (guest със съответните му предоставени права **GRANT select,update,insert on tablename** за манипулация на записите от съответните таблици) и парола за достъп до съответната база/схема
- *server.policy* съдържа ограниченията които налага SecurityManager на RMI сървър (*SocketPermission*, *FilePermission*) от вида:

```
grant {  
    permission java.net.SocketPermission "*:1024-","accept, connect";  
    permission java.net.SocketPermission "*:80","accept, connect";  
    permission java.io.FilePermission "/var/www/-"," read, execute";  
    ....  
}
```

- *RMIserver.conf* съдържа информация за RMI сървърите (един или няколко) – има вид подобен на *DBNodes.conf* с разликата, че параметрите са име на сървър (в URL форма която се изисква от системата за именоване *java.rmi.Naming* – а именно **//host:port/name** и се използва от клиента при обръщение към

сървъра посредством *RMI регистъра*) и стойността на параметъра *java.rmi.server.codebase*<sup>10</sup>, която сървъра предоставя на клиентите за да могат да си изтеглят stub бланките

### **Възможни промени (upgrade) на първоначалната система**

По първоначален план проекта UniLib трябваше да обслужва система от различни университети (намиращи се евентуално в различни градове), които са относително доста независими един от друг и са свързани с помощта на Интернет. В последствие обаче първоначалната идея би могла да претърпи промени и да се наложи добавяне или промяна на функционалност на съществуващата система. Промените биха могли да бъдат от най-различно естество – като например отнасящи се до промяна в структурата и съдържанието:

- добавяне на нови видове информационни носители и допълнителни параметри за тях (което води до промяна в структурата на таблиците в базата данни) – така например е възможно добавяне на онлайн ресурси (линкове към html/pdf източници), при които обаче заемането ще отпадне т.е. ще бъде реализирано четене
- добавяне на допълнителни възможности за потребителите на системата – освен сегашните би могло да се добавят интереси на потребителя както и възможност за коментиране (рецензиране) на съдържанието; форум за потребителите; уведомяване по електронната поща и т.н.

При евентуална промяна в съдържанието на базата данни следва да се направят и съответните промени в middleware свързани с добавянето на функционалност към вече съществуващата. При използвания трислоен модел това е лесно за реализация понеже има разделяне на задачите, които всеки слой решава.

Отделно от независимо от промените в базата данни съществуват и варианти свързани с промяна на потребителския интерфейс и функционалността която предлага. Би могло да се добавят нови начини на търсене – например търсене в конкретен възел на системата (при което ще трябва да се визуализират активните възли на системата – примерно с падащ списък в `jsp` и да се добави функционалност за търсене и по този параметър в сървърната част вместо досегашния обход по всички възли – реално в момента RMI сървър има тази възможност чрез метода `search(..,nodeID,..)`, но не и клиентския интерфейс където има прозрачност на достъпа при търсене). Възможни са вероятно и множество други промени било само по вида на презентационния слой, както и на сървърната функционалност.

Потенциален недостатък на първоначалния вариант е липсата на възможност за сортиране на резултатите при търсене (или пък на заетите от потребителя книги), което може да бъде добавено сравнително лесно в презентационния слой (с добавяне на елемент-контрола в `jsp` и промяна на функционалността на `bean` компонента за сортиране на набора от резултати) без никакви изменения в останалите слоеве. Изобщо модифицирането на вида на клиентската част на приложението е доста улеснено вследствие на избраната технология от една страна и липсата на директна обвързаност със следващите слоеве.

### **Тестване на приложението**

Интернет базираните многослойни приложения имат множество възможни точки на провал, които трябва да се вземат под внимание при тестване<sup>11</sup>. Системата трябва да работи добре като цяло, което значи че трябва да се тества работата на отделните слоеве.

- Потребителите на приложението имат различни умения, браузери и операционни системи. Разпределената система от университетски библиотеки UniLib е тествана под операционни

системи Linux и Window с браузъри – Internet Explorer 6.0 и 7.0, Firefox 2.0.x и Opera 9.x. Приложението работи стабилно. Размерът, вида и цвета на шрифтовете е един и същ в различните браузери. Графиките са с коректната резолюция и размер. Прегледана е цялостната навигация при преминаване между отделните страници на потребителския интерфейс, което е свързано с валидация на линковете и използването на сесийни променливи.

- Очаква се потребителите да достъпват приложението с различна скорост зависеща от техните интернет връзки и това би могло да бъде причина за допълнително забавяне.
- Различните потребители използват различни езици и кодировки (локализация). В нашето приложение, ние сме използвали UTF-8 кодировка, което би следвало да е универсално решение.
- Тъй като приложението е отворено към света, трябва да бъде осигурена сигурността на работата му. Проблема със сигурността има много различни страни: от вида на използвания софтуер (в случая основата е java базирания механизъм за сигурност както за клиентската така и за сървърната част); използваната операционна система и допълнителни средства за защита като firewall свързани с администрирането на сървърите на системата. При първоначалната реализация на приложението се използва Linux-базиран уеб сървър с firewall.

Приложението е тествано със следните хардуерни конфигурации – RAM памет 1-2 GB и процесори 1 – 2.5 GHz. Системата се държи стабилно. Препоръчително е да се използват по-мощни конфигурации (двупроцесорна сървърна система с 2-4 GB памет) за постигане на по добра производителност, особено ако системата бъде използвана от голям брой потребители.

Тестването на работата на базата данни (крайния слой) също

изключително важно.

- Време за отговор (Response Time) – оценка на бързината на DML операциите в базата данни (SELECT, INSERT, UPDATE). Когато потребителя въведе дадена заявка, това е времето за нейното изпълнение без да се отчита времето на работа на уеб сървъра за визуализиране на резултата. При тестваната система с 4 възела и резултат от търсене (от вида *SELECT from books*) в базата от порядъка на няколкостотин записа времето е незначително – по-малко от секунда. Предполага се че това е случая в който времето за отговор е най-голямо, понеже заемането и връщането са свързани с по-малък обем данни за обработвани а и са асоциирани към конкретен възел (а не към всички както е при търсенето). При използване едновременно от много потребители и увеличаване обема на данните в разпределената база, това време би могло да нарасне значително.
- Интегритет на данните – тестването на интегритет на данните е процеса на намиране на некоректни данни в таблиците от базата данни. Тук става дума не за валидацията която извършват презентационния и бизнес слоя, а за работата на крайния слой за достъп до базата данни. В случая става дума за тестване на крайния резултат (в базата данни) от действието на регистрационната форма, както формите за заемане и връщане на книги (INSERT в случая на регистрация и заемане и DELETE в случая на връщане – съответно в таблиците **users** и **loans**).

## Заклучение

В настоящата дипломна работа беше направена реализация на разпределена система от университетски библиотеки с онлайн достъп. Потребителите на системата след като се регистрират могат да използват редица услуги като:

- да осъществят просто и разширено търсене (по комбинация от няколко различни параметъра – автор, ключови думи, издателство и т.н.) в голямо разнообразие от материали и източници (книги, периодични издания и др.)
- да заемат и връщат намерените налични издания от звената на библиотеката
- да следят текущия си статус – заетите вече материали
- да получават детайлна информация за предлаганите изданията

Приложението е лесно и интуитивно за използване. В следствие на избора на java технологии за реализацията на системата тя е преносима и мултиплатформена както и лесно мащабируема по отношение на броя на включените звена и обема на предлаганото съдържание. Изискванията към клиента са единствено да има интернет и браузър (поддържащ *JavaScript* и *CSS*) – Internet Explorer, Firefox или Opera без значение вида на операционната система.

Приложението е проектирано така, че да предполага допълнително надграждане съобразно евентуалните нужди на потребителите. Има няколко насоки за бъдещо развитие за настояща дипломна работа свързани с различни възможности за промяна на вида на презентационния уеб слой и промяна или добавяне на функционалност на разпределения middleware съобразно конкретните нужди на потребителите или добавяне на ново (по обем и тип) съдържание в разпределената база данни.



## Използвана литература

---

- 1 Andrew S. Tanenbaum, Distributed Operating Systems, Prentice Hall, 1995
- 2 Силвия Илиева, Лекции по middleware, ФМИ 2006
- 3 [http://en.wikipedia.org/wiki/Distributed\\_computing](http://en.wikipedia.org/wiki/Distributed_computing)
- 4 [http://en.wikipedia.org/wiki/Web\\_server](http://en.wikipedia.org/wiki/Web_server)
- 5 Java Server Pages – O'Reilly 2<sup>nd</sup> edition, 2002
- 6 <http://www.apachefriends.org/en/xampp.html>
- 7 Дик Стефлик, Прашант Сридхаран - Java за мрежови приложения 2 издание, Prentice Hall, 2000
- 8 Светлин Наков и колектив – Програмиране за .NET Framework 1 том
- 9 [http://en.wikipedia.org/wiki/Database\\_model](http://en.wikipedia.org/wiki/Database_model)
- 10 <http://java.sun.com/j2se/1.4.2/docs/guide/rmi/codebase.html>
- 11 Glenford J. Myers et. al., The Art of Software Testing, 2004