Integrated Library Management System

Addis Ababa University Center of Information Technology and Scientific Computing

Software Engineering Department

System Design Specifications (SDS) Document

Made By: 20/03569 Matta Kimani

Table of Contents

[List of Tables I](#_TOC_250014)

[List of Figures ii](#_TOC_250013)

[Acronyms iii](#_TOC_250012)

1. [Introduction 1](#_TOC_250011)
   1. [Purpose 1](#_TOC_250010)
   2. [General Overview 1](#_TOC_250009)
   3. [Development Methods & Contingencies. 4](#_TOC_250008)
2. [System Architecture. 4](#_TOC_250007)
   1. Subsystem decomposition. 4
      1. Layer 1. 4
      2. Layer 2. 5
      3. Layer3. 6
   2. [Hardware/software mapping. 7](#_TOC_250006)
      1. Deployment Diagram. 7
3. [Object Model 8](#_TOC_250005)
   1. [Class Diagram. 8](#_TOC_250004)
   2. [Sequence Diagram 9](#_TOC_250003)
      1. Add Book. 9
      2. Issue Book 10
      3. Register Members. 11
      4. Remove Book 12
      5. Return Book. 13
      6. Review Book. 14
   3. [State Chart Diagram. 15](#_TOC_250002)
4. [Detailed Design. 16](#_TOC_250001)

[References 26](#_TOC_250000)

## List of Tables

*Table 1: Librarian Class 16*

*Table 2: Attribute description of Librarian class 16*

*Table 3: Operation description for Librarian Class 17*

*Table 4: Patron class 8*

*Table 5: Attribute description for Patron class 19*

*Table 6: Operation description for Patron class 18*

*Table 7: Library Admin Class 20*

*Table 8: Attribute description for Library Admin class 20*

*Table 9: Operation description for 7: Library ceases 21*

*Table 10: Book Class*

*Table 11: Attribute description for Book ceases 22*

*Table 12: Operation description for Book Class 23*

*Table 13: Transaction Class 23*

*Table 14: Attribute description for Transaction Class 24*

*Table 15: Operation description for Transaction Class 24*

*Table 16: Book Comment Class 24*

*Table 17: Attribute description for Librarian Class 25*

*Table 18: Operation description for Librarian CIAs 25*

## List of Figures

*Figure 1: Example Diagram* J

*Figure 2: Layer 1 4*

*Figure* 3: *Layer 2. 5*

*Figure 4: Layer 3 6*

*Figure 5: Deployment Diagram.* 7

*Figure 6: ceases Diagram 8*

*Figure 7: Add book. 9*

*Figure 8: Issue book. 10*

*Figure 9: Register Members. 11*

*Figure 10: Remove Books. 12*

*Figure 11: Return Book. 13*

*Figure 12: Review Book. 14*

# Acronyms

OOP — Object Oriented Programming

ILMS — Integrated Library Management System

MVC - Model View Control SE - Standard Edition

PHP — Hypertext Pre—processor SDS - System Design Specification http — hypertext transfer protocol

## Introduction

### Purpose

The purpose of this design document is to present the system design at a level that can be directly traced to the specific system objective along with providing more detailed data, functional and behavioral requirements.

This design document will verify that the current design meets all of the explicit requirements contained in the system model as well as implicit requirements desired.

#### General Overview

In object-oriented programming development, model-view-controller (MVC) is a methodology or design pattern for successfully and efficiently relating the user interface to underlying data models. The model-view-controller pattern proposes three main components or objects to be used:

* + - A Model, which represents the underlying, logical structure of data in a software application and the high-level class associated with it. This object model does not contain any information about the user interface.
    - A View, which is a collection of classes representing the elements in the user interface (all of the things the user can see and respond to on the screen, such as buttons, display boxes, and so forth)
    - A Controller, which represents the classes connecting the model and the view, and is used to communicate between classes in the model and view.

Unnecessary complexity is an adverse to software development. Complexity leads to software that is buggy, and expensive to maintain. One big issue behind code complexity is putting dependencies everywhere. Conversely, removing unnecessary dependencies makes luscious code that is less buggy and easier to maintain and reusable without modification. Where stable code can be reused without introducing new bugs into it.

The system's data model is contained in a persistent storage, a traditional server-side MySQL database. The controller classes are written in JavaScript for the web deployment and java for the desktop. The system's interface is written in HTML/CSS and java. This sounds like MVC, where each component follows a more rigid pattern.

The **Controller** manages the user requests (received as HTTP GET or POST requests) when the user clicks on GUI elements or when a librarian /administrator clicks on the desktop interface to perform actions. Its main function is to call and coordinate the necessary resources/objects needed to perform the user action.

The **Model** is the data and the rules applying to that data, which represent concepts that the application manages. As stated above, ILMS depends on a database to store all the required data such as member data used for account control, book catalog data, user feedback used for rating books, transaction cache and so on. The model gives the controller a data representation of user requests.

The **View** provides different ways to present the data received from the **Model.** It refers to the desktop and web interface of the ILMS system where users can interact with.

The interface presented must be different if the request came from the desktop app or from the web interface. The model returns exactly the same data, the only difference is that the controller will choose a different view to render them. Usually view and controller collaborate, whereas the model initially operates independently of both. With the help of MVC-concept, program elements are largely independent of each other which implies a clear and well-structured program code.

Here is a brief explanation of how MVC-concept works in the ILMS system form the administrator's point of view as an example.

An administrator can add or remove Library Members. Say the administrator clicks on add member’s button. There is a particular control that handles all account-related actions (accountControl.java). A model for handles data and logic related to the items in the ILMS System and a series of views to present, a page to enter new member's info to be exact.

ILMS database

controller

* + - * Account ControI.java

Desktop interface

Figure 1: Example Diagram

MVC Architecture and ILMS System Security

There is a login screen that collects and validates the username and password. This page then directs the user to another screen that allows them to continue in a secure manner. However, as MVC doesn't provide anything that prevents the user from going directly to a page, security is accomplished by including a security check inside the controller object. Because the interface to the customer is driven through the controller object, there is a single-entry point and a single location for the security checks to be performed.

MVC Architecture and Maintainability

The model view controller (MVC) is a three different tier architecture design pattern for the dissociation Data Access logic, user interface and control logic. As stated above the ILMS system is structured in MVC pattern. Model expressing domain knowledge, view representing user interface, control that is used to manage the updates to views. The dissociation property extensively aids in the maintenance of ILMS system components.

MVC architecture provides a better performance than most of the patterns available yet performance of systems can be further enhanced based on two parameters namely response time and throughput. The response time and throughput are improved based on the proposed database search algorithm using B+ tree. (<http://www.ijcaonline.org/archives/volume134/number12/23970-2016908099)>

These are additional reasons why we chose MVC architectural pattern.

Developer specialization and focus: UI developers can focus exclusively on UI, without being hindered by business logic rules or code.

Parallel development:

* + - Business logic developers can build empty classes that allow UI developers to forge ahead before business logic is fully implemented.
    - UI can be reworked as much as required without slowing down the development of code that implements business rules.

Applications having an MVC design are also more easily extensible than others

###### Development Methods & Contingencies

The system used an object orientated approach as well as MVC architecture in development. Since the first step in OOP is to identify all the objects the programmer wants to manipulate and how they relate to each other, an exercise often known as data modelling. Once an object has been identified, it is generalized as a class of objects which defines the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method. Objects communicate with well-defined interfaces called messages. The system used this approach in designing the overall process.

The system has also make use of UML diagrams to provide a standard way to visualize the design of a system in developing and used Java SE in development of the desktop component the logic processing of the web component since these languages is well fitted to OOP.

## System Architecture

* 1. Subsystem **decomposition**

2.1. I Layer 1

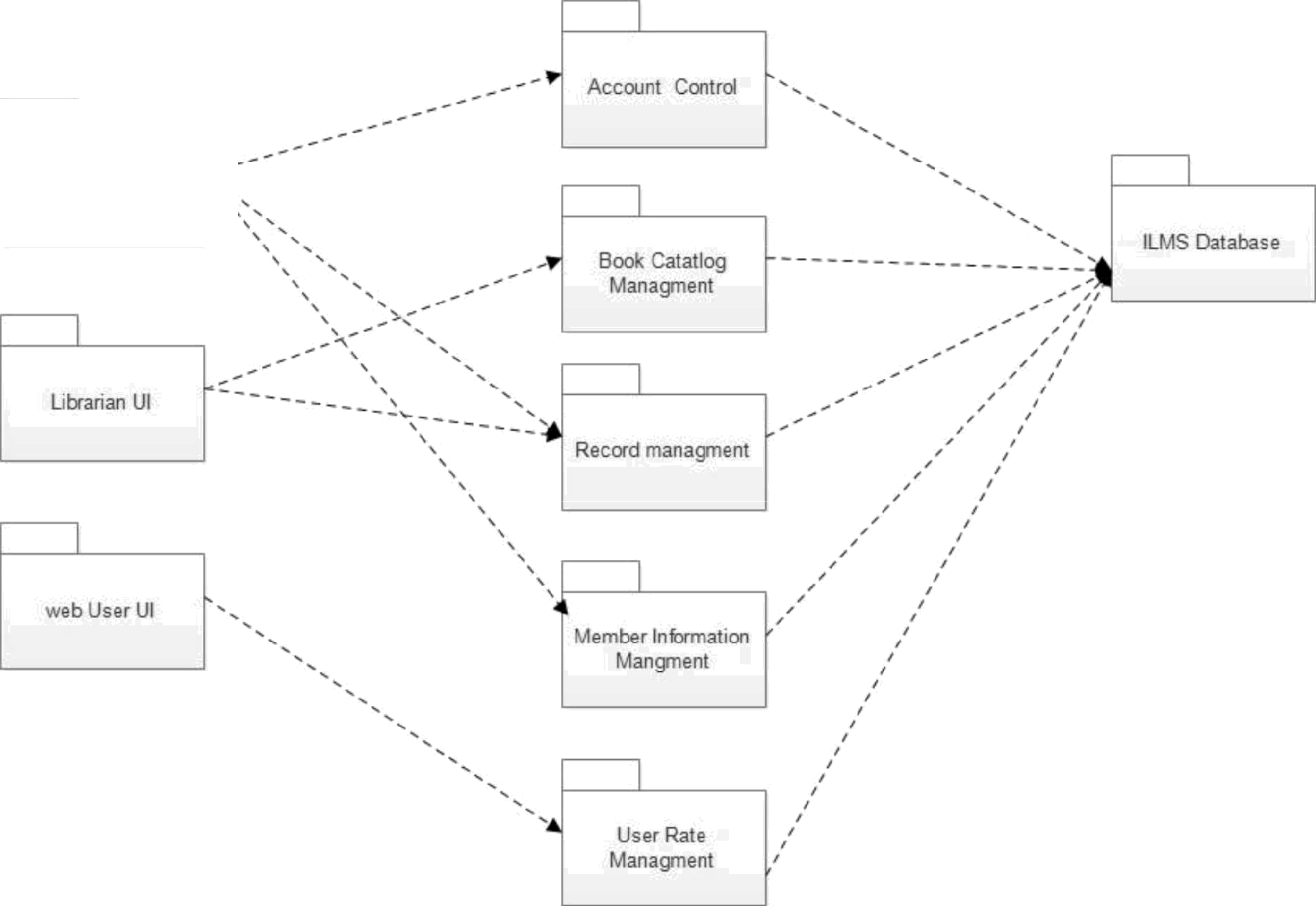
View/presentation

Control

Persistent storage

Figure 2: Layer 1

* + 1. Layer 2



Admin UI

Figure 3: Layer 2 Diagram

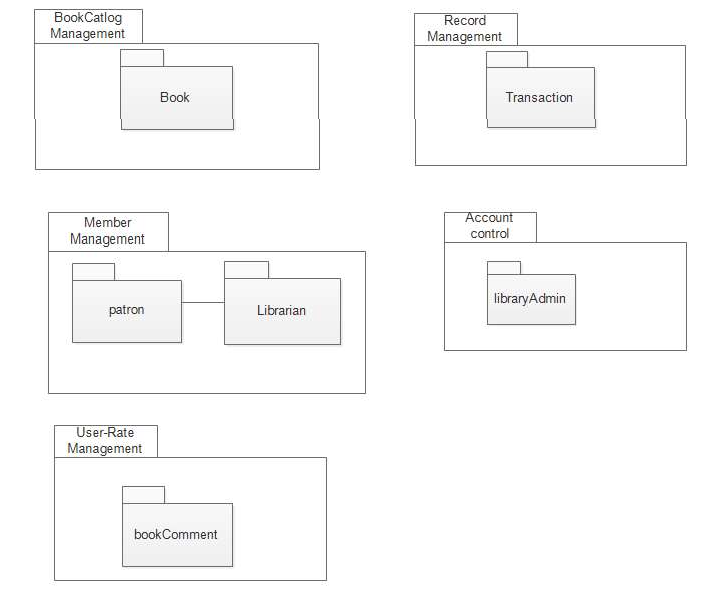
* + 1. Layer3

Figure 4: Layer 3 Diagram

##### Hardware/software mapping

2.2.1 Deployment Diagram

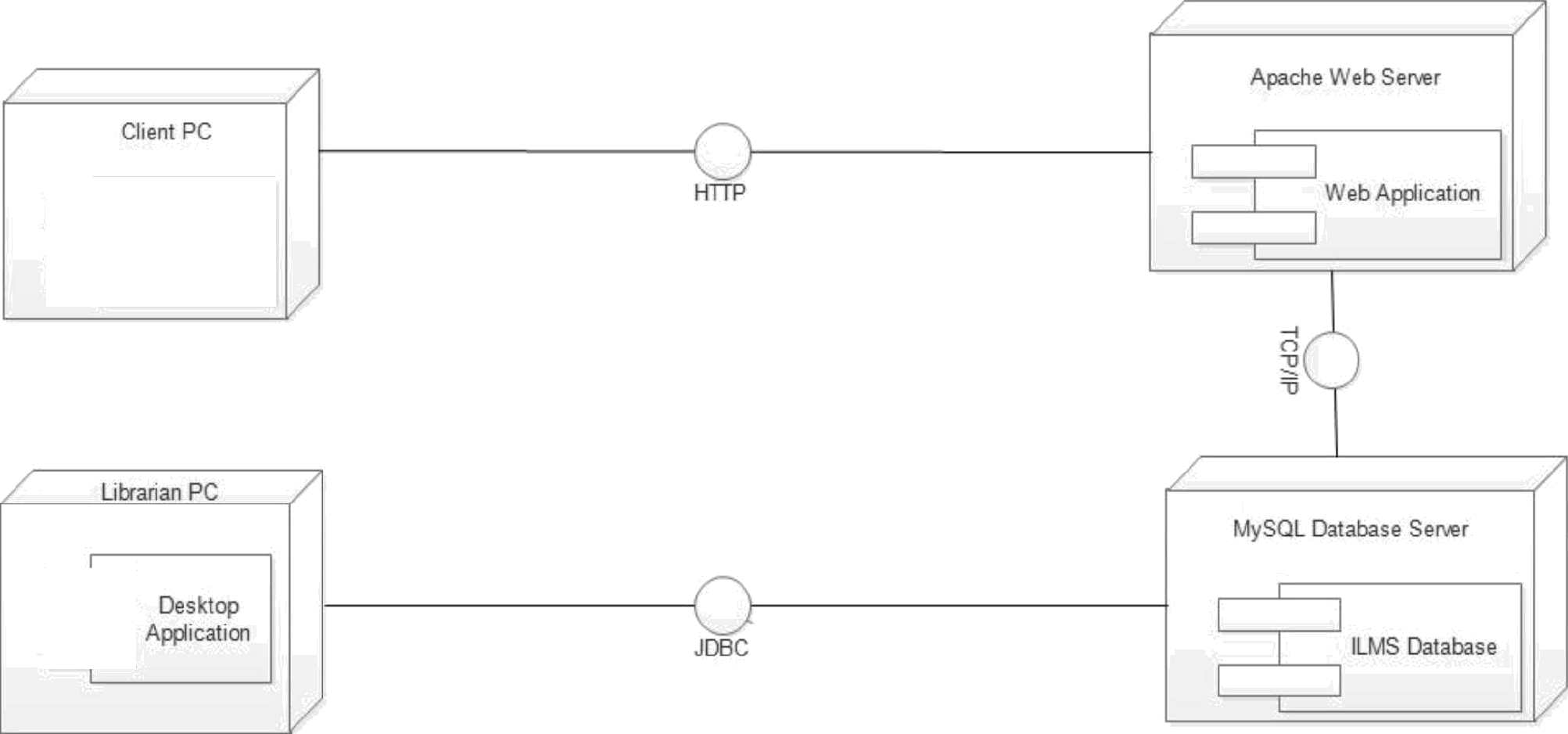
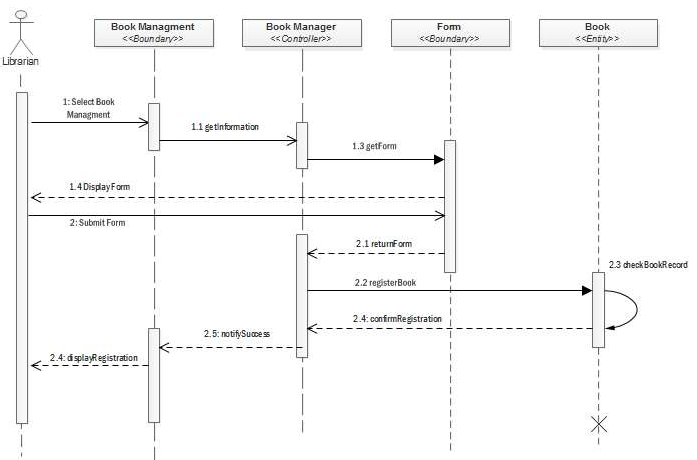


Figure 5: Deployment Diagram

##### Sequence Diagram

3.2.1



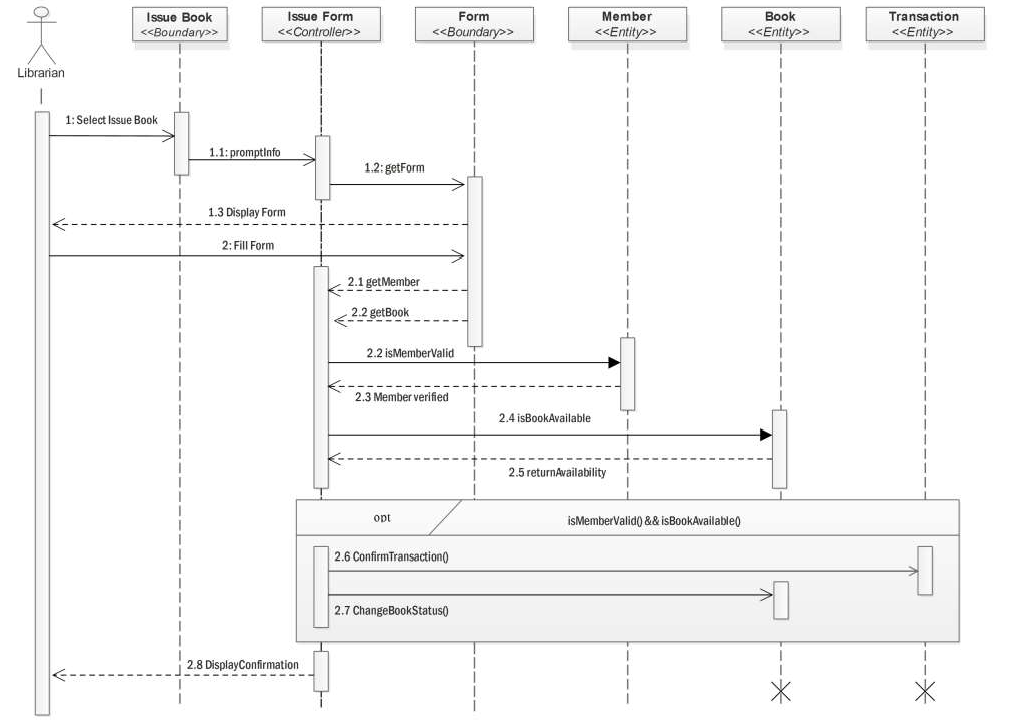
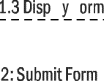
* + 1. Issue Book

Figure 8: Issue book

Register Members



ember



form

*<<Boundary>>*



register

<<Goniro1ier>>



ember Management

*<<Boundary> >*

1: Select Employee

1.I gelifraction

* 1. rent Form

l 2.3chec#Reor0

* 1. register Member

2.4: confirm Registration

2.6 2.5 notify/success

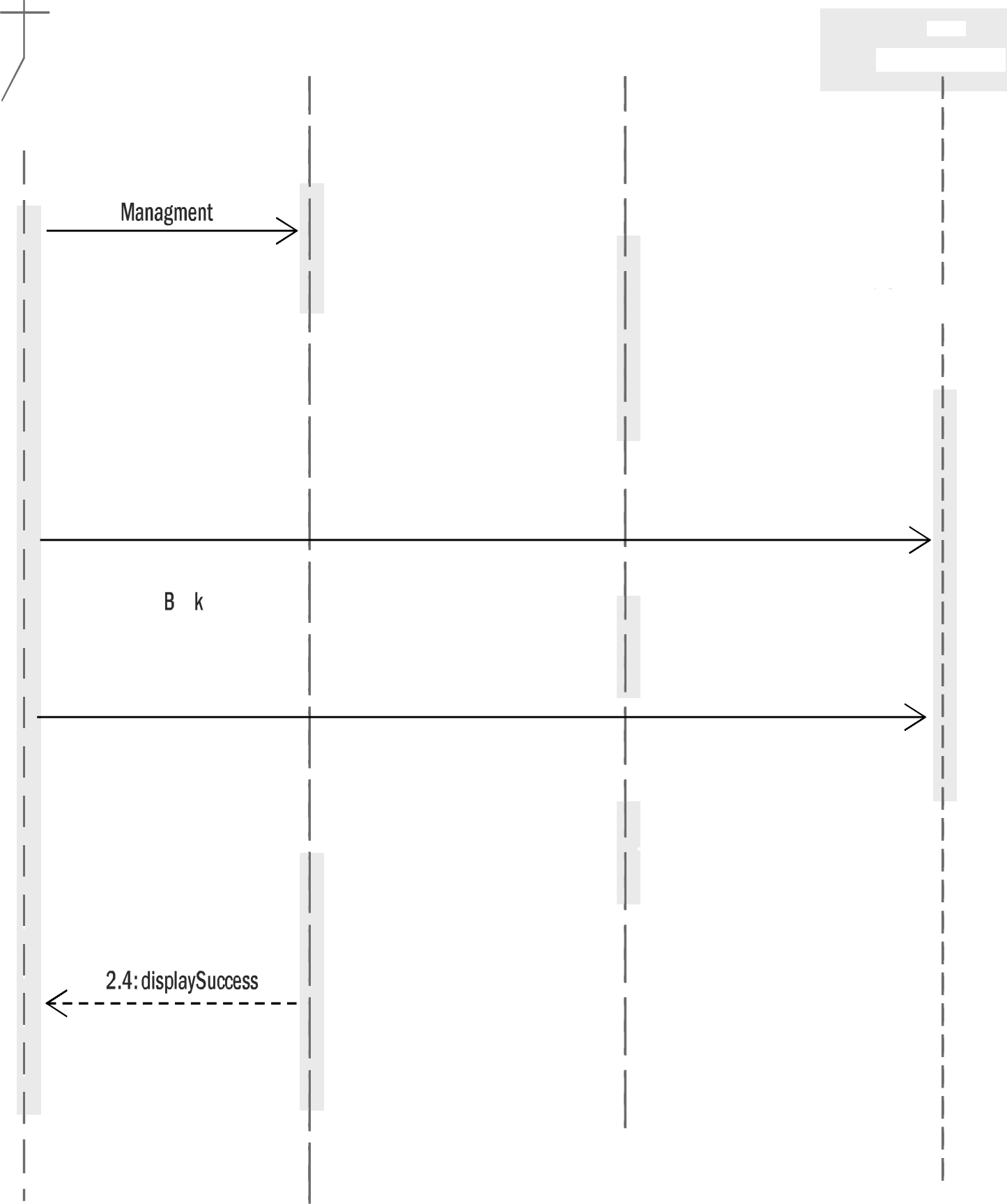
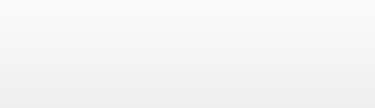
Figure 9: Register Members

* + 1. Remove Book



book

*<<Entity»*



book Management

«8our#a/y>>

book Manager

*«<0ontroIIei>»*

form

*<«Boundary»*

Librarian

1.1 form

1.2 get list

1.3 retvrrBookList

!

1.4 Display List

2: Select/Search

2.1 retvrnBook

22RezRemo eButto

2.3rem eBook

2.4: a mRemo al

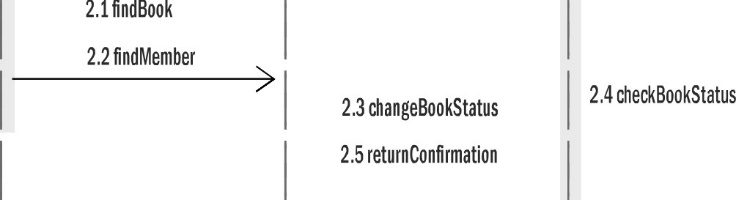
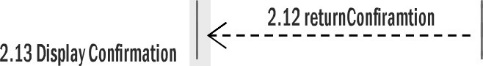
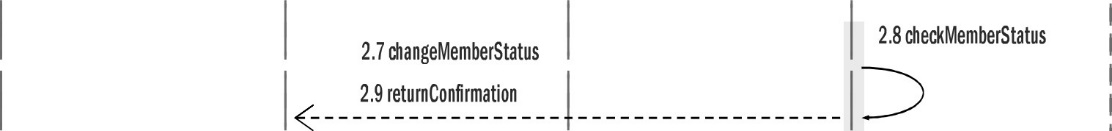
2.5: noiseless

Figure 10: Remove Books

* + 1. Return Book



Librarian



1:feleURe(r

1.2 getketrrlForm

I.4:Disyhy Farm

I.3:iet4rzF rm

2.11(#)#frmTra#sa%o#

Figure 11: Return Book



eturn Window

*«Boundag»>*



00k

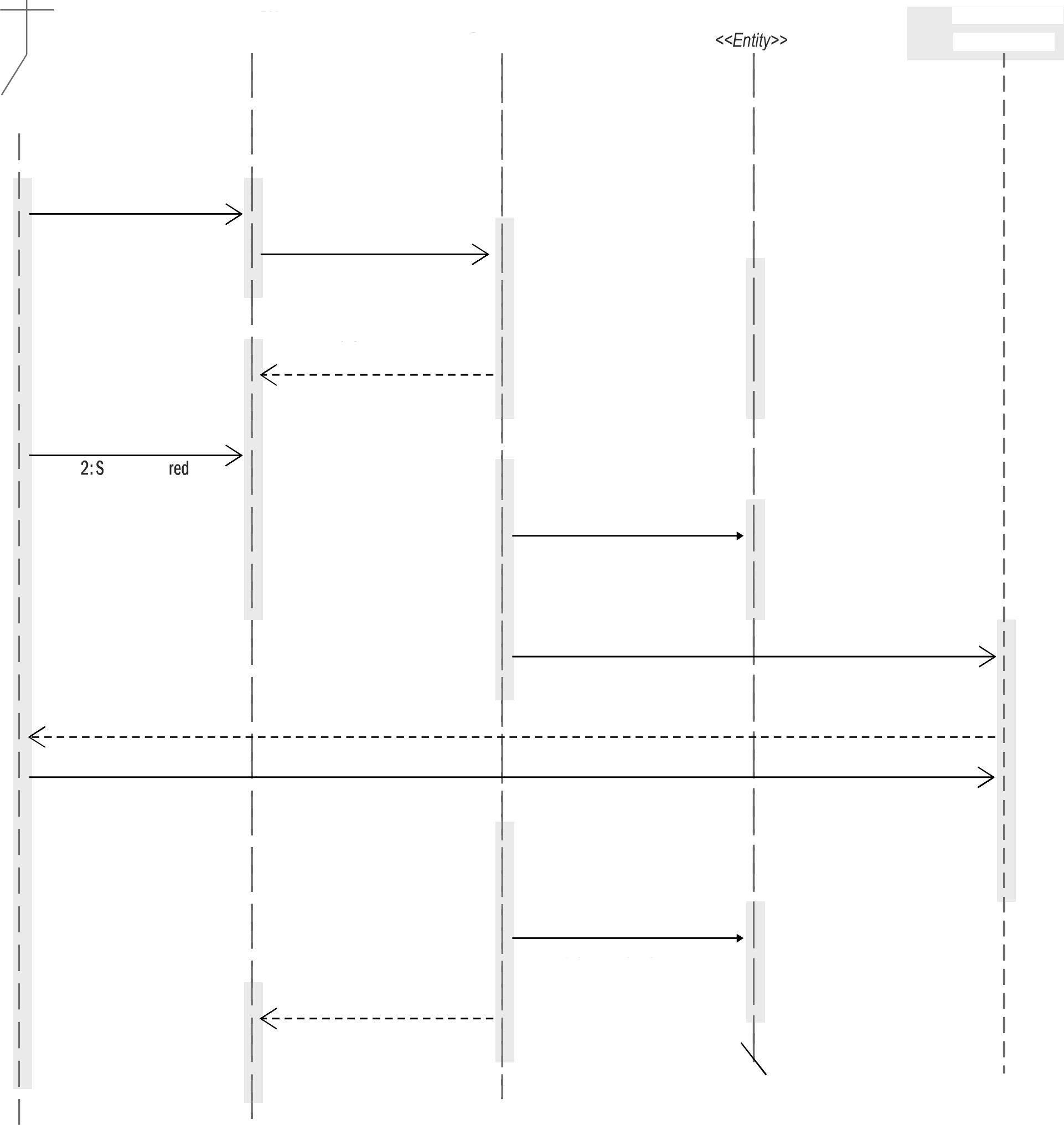
*«Entifi/»*



*«Entily>»*



* + 1. Review Book



ook Manager

*«Sontrollei>>*

omment Form

*«Boundary>»*

Patron

1: Searcä Book

1.1: searcäBook

* 1. : getBooküst
  2. : fetvrrBooküst

1.4 ret0rrRes0lt

1.5 displayResvlt

ele¢t Piefe

Book

2.1 findBook

* 1. yetBook
  2. retvrnBook

.4 getCommentFofm

2.5 displayComfrentForri

3: svbfritForm

3.1 etvr Fœm

* 1. appe dComme t
  2. fettlrI1C0flIifITIüti0I1

3.4 retvrnComment

3.5 displayComment

ook

isplay

«8oundary»

Figure 12: Review Book

##### 3.3 State Chart Diagram

State Chart for Book Object

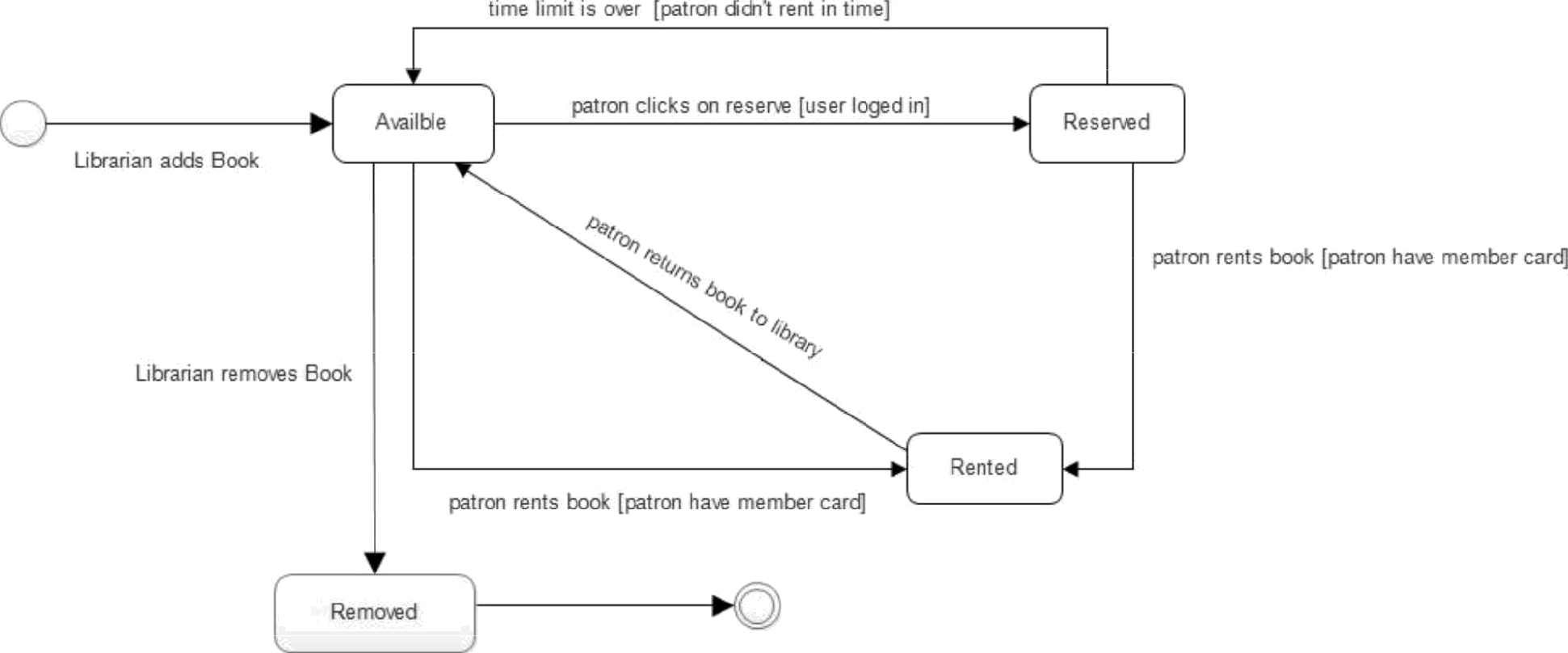


Figure 13: State chart Diagram

#### Detailed Design

Table 1: Librarian Class

|  |
| --- |
| Librarian |
| * name: String * employed: String * IoglnPassword: String |
| addBook(title:String, isbn:String, bookID:int, publisher:String, author:String, edition:String, category:String): void removeBook(bookID:int): void  updateBook(bookID:int, title:String, isbn:String, publisher:String, author:String, edition:String,category:String): void searchBook(key:String): void  IendBook(b:Book, p:Patron): void returnBook(b:Book, p:Patron): void removeReservation(b:Book): void searchPatron(key:String): void generateReport():void changePass(new:String): void |

Table 2: Attribute description of Librarian class

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Visibility | Invariant |
| name | String | Private | name <> NULL and must contain first, middle and last name and  shouldn't contain special characters and integers. |
| employeelD | String | Private | employeelD *<>* NULL |
| IoglnPassword | String | Private | IoglnPassword <>NULL must be at least 6 characters and at most  25 characters |

Table 3: Operation description for Librarian Class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation | Visibility | Return  type | Argument | Pre-Condition | Post Condition |
| addBook | Public | void | title:String,  isbn:String, bookID:int, publisher:String, author:String, edition:String,  category:String | The book should  not exist | The book should  exist |
| removeBook | Public | void | bookID:int | The book should  exist and must not  be reserved or rented | The book should  not exist |
| updateBook | Public | void | bookID:int,  title:String, isbn:String, publisher:String, author:String, edition:String, category:String | Book information  was not updated | Book information  should be updated |
| searchBook | Public | void | key:String |  | Searched book  results should be displayed to the user |
| lendBook | Public | void | **b:Book**  p:Patron | Book to be lend  must exist and is available | Book should be  lent to the patron and should not be available until  returned |
| returnBook | Public | void | b:Book  p:Patron | Book should be  rented to Patron | Book should be  returned and become available for rent again |
| removeReservation | Public | void | b: Book | Book should be  reserved | Book reservation  must be cancelled |
| searchPatron | Public | void | key:String | The user has  logged in | Searched patron  results should be displayed to the user |
| generateReport | Public | void |  | There should be a  transaction stored  in the database | Report should be  generated |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| changePass | Public | void | newPass:String | Previous password  should exist | New password should replace previous  password |

Table 4: Patron class

|  |
| --- |
| Patron |
| * name: string * id: string * allBooksRented: Book[\*] * currentBooks: Book[\*] * IastBookrentDate: Date[\*] * scaduledReturnDate: Date[\*] * IoglnPassword: string |
| + changePass(new: String): void  + reserveBook(b:Book): void  + rateBook(b:Book, r:Int): void  + commentBook(b:Book, reviewer:Patron, com:String): void  + searchBook(key:String): Book  + changeMemberStatus(key:String): void |

Table 5:Operation description for Patron class

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Visibility | Invariant |
| name | String | Private | name<> NULL must contain first, middle and Last name  and no special caracters |
| id | String | Private | id <> NULL |
| allBooksRented | Book | Private | allBooksRented <> LL |
| currentBooks | Book | Private | currentBooks <> NULL |
| IastBookrentDate | Date | Private | IastBookrentDate <> NULL |
| scaduledReturnDate | Date | Private | scaduledReturnDate <> ALL |
| IoglnPassword | String | Private | Category <> NULL must be more than 8 characters long |
| currentRenter | Array | Private | currentRenter <> NULL must contain array of patron  object id values |

Figure 6: Operation description for patron class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation | Visibility | Return  type | Argument | Pre-Condition | Post Condition |
| changePass | Public | void | - new: String | Existing password  in database | The existing  password should |
|  |  |  |  |  | be replaced with the new password |
|  |  |  |  |  | passed as |
|  |  |  |  |  | argument |
| reserveBook | Public | Void | -b:Book | Book's object | The passed |
|  |  |  |  | passed reserved | book's object |
|  |  |  |  | attribute is true | ‘reserved’ |
|  |  |  |  |  | attribute should |
|  |  |  |  |  | be set to false to |
|  |  |  |  |  | denote |
|  |  |  |  |  | reservation |
| rateBook | Public | Void | * b: Book * r: Int | Book's object rate  attribute is has | Book's object  rate attribute will |
|  |  |  |  | certain float value | be calculated |
|  |  |  |  |  | adding the passed |
|  |  |  |  |  | ‘r’ int argument |
| searchBook | Public | Book | - key:String | The librarian has  logged into the system. | The required  result will be returned. |
| changeMemberStatus | Public | void | void | The selected member  should exist with a given status. | The member's  status should change. |

Table 7:Library Admin Class

|  |
| --- |
| LibraryAdmin |
| * name: string * employeelD: string * IoglnPassword: string |
| + addPatron(name:String, id:String): void  + removePatron(id String): void  + updatePatronRecord(id String): void  + changePass(string new): void |

Table 8: Attribute description for LibraryAdmin class

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Visibility | Invariant |
| name | String | Private | title <> NULL must contain full name |
| employeelD | String | Private | employeelD <> NULL |
| IoglnPassword | String | Private | booklD <>NULL must be more than 8 characters long |

Table 9: Operation description for LibraryAdmin class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation | Visibility | Return  type | Argument | Pre-Condition | Post Condition |
| addPatron | Public | void | - name:String | The patron | The new patron |
|  |  |  | - id:String | should not exist | should be added |
|  |  |  |  |  | to database with |
|  |  |  |  |  | the passed |
|  |  |  |  |  | attributes |
| removePatron | Public | Void | - id String | A patron with the | The patron and |
|  |  |  |  | passed id should | all its values with |
|  |  |  |  | exist | the passed id |
|  |  |  |  |  | should be |
|  |  |  |  |  | removed from |
|  |  |  |  |  | the database |
| updatePatronRecord | Public | Void | - id: String | Patron with the  passed id should exist in database | Updates the  patron's record. |
| changePass | Public | VOid | - new:String | Existing | The existing |
|  |  |  |  | password in database | password should  be replaced with the new |
|  |  |  |  |  | password passed |
|  |  |  |  |  | as argument |

Table 10: Book Class

|  |
| --- |
| Book |
| * title: string * isbn: int * booklD: String * publisher: string * author: string * edition: string * category: string * rate: float * reservation: boolean * rented: boolean * renters: Patron[\*] * currentRenter: Patron |
| + getBookList(name:String): Book[\*]  + removePatron(id String): void  + addPatron(name:String, id:String): void  + removePatron(id String): void |

Table 11: Attribute description for Book class

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Visibility | Invariant |
| title | String | Private | title <> NULL |
| isbn | Integer | Private | isbn <> NULL must be 10 digits |
| booklD | String | Private | booklD <>NULL |
| publisher | String | Private | publisher <> NULL |
| author | String | Private | Author <> NULL and must contain first and last name |
| edition | string | Private | Edition <> NULL |
| category | String | Private | Category <> NULL |
| rate | float | Private | Rate <> NULL |
| reservation | boolean | Private | Reservation <> NULL |
| rented | boolean | Private | rented <> NULL |
| renters | Array | Private | Renters <> NULL must contain array of patron object id  values |
| currentRenter | Array | Private | currentRenter <> NULL must contain array of patron  object id values |

Table 12: Operation description for Book Class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation | Visibility | Return  type | Argument | Pre-Condition | Post  Condition |
| getBookList | public | Book[\*] | Book ID / Title | An information  about the required book should be provided. | A list of  related books should be returned. |
| removeBook | Public | void | Void | There should exist  at least one book with the provided  detail. | A book must  be eliminated. |
| changeBookStatus | public | void | Integer | The indicated book  should exist with a  Defined status. | The Book's  status should  change. |
| appendComment | public | void | MemberlD  Comment Date | The selected book  should exist. | A comment  concerning the book is appended to it's comment  list. |

Table 13: Transaction Class

|  |
| --- |
| Transaction |
| * transactionNumber: int * transaction Date: Date * transactionType: String * actorPatronlD: String * transactedBooklD: int |
| + updateTransaction(): void  + getTransaction(transactionNumber: int): void |

Table 14: Attribute description for Transaction Class

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Visibility | Invariant |
| transactionNumber | int | Private | transactionNumber <> NULL |
| transactionDate | Date | Private | transactionDate <> NULL |
| transactionType | String | Private | transactionType <>NULL must have the values of either  “rent”, “return”, “reserve”, “reserveCancellation”, “addBook”, “removeBook”, “addPatron” or “removePatron” |
| actorPatronID | String | Private | actorPatronID <> NULL |
| transactedBookID | int | Private | transactedBookID <> NULL |

Table 15: Operation description for Transaction Class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation | Visibility | Return  type | Argument | Pre-Condition | Post Condition |
| addTransaction | Public | void | transaction Number: int, transactionDate: Date, transactionType: String,  actorPatronlD: String, transactedBooklD | The transaction  should not exist | The transaction  should be added and should exist |
| getTransaction | Public | void | transaction Number: int, | The transaction  should exist | The required  transaction should be displayed to the user |

Table 16: BookComment Class

|  |
| --- |
| **BookComment** |
| * commentlD: int * date: Date * patronlD: String * booklD: int * comment: String |
| + delete(): void  + addTransaction(): void |

Table 17: Attribute description for Librarian Class

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Visibility | Invariant |
| commentID | int | Private | commentID <> NULL |
| date | Date | Private | date <> NULL |
| patronID | String | Private | patronID <>NULL |
| comment | String | Private | comment <>NULL |

Table 18: Operation description for Librarian Class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation | Visibility | Return  type | Argument | Pre-Condition | Post Condition |
| delete | Public | void |  | The comment  should exist | The comment  should not exist anymore |
| addTransaction | Public | void |  | An authorized  transaction must  occur. | The transaction's  information should  be recorded. |

Table 19: Operation description for Book Manager Class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation | Visibility | Return  type | Argument | Pre-Condition | Post Condition |
| findBook | public | Book | Book ID | Book ID should be  provided. | A book with the  given id is returned. |
| findMember | public | Member | Member ID | Member ID should  be provided. | A member with the  stated id is returned |
| searchBook | Public | Book | Name | Book Name should  be provided. | A list of books  related to the provided name are returned. |

## References

**Websites**

MVC architecture. "Understanding MVC architecture". 16 July 2013.MVCarch.web .15 November 2022< [http://www.understandingMVC.com](http://www.understandingMVC.com/) >.

Wiki. "Detail Design". 15 July 2012. Wikipedia. [15 November 2022. +http://wikipeida.com/detailDesignDescription.html](http://wikipeida.com/detailDesignDescription.html) >.

2022. +. "Object oriented programming".2013. 2022. +, Inc. Web. 15 November 2022.http:// m.wikihow.com/OOP >

Worcester polytechnic institute. "How to draw state chart diagram".2014. Worcester polytechnic. 15 November 2022. +http:// [https://www.wpi.edu/academics/ugradstudies/statechart.html](http://www.wpi.edu/academics/ugradstudies/statechart.html) +.

Prakash Dhansak. "Software Design Specification"" March 21, 2007. LinkedIn corporation. 15 November 2022. < [https://www.s1ideshare.net/mobi1e/ramPrakash1989/sds-explanation](http://www.s1ideshare.net/mobi1e/ramPrakash1989/sds-explanation) >

Sensi. "Sample design document". 17 March 2009. Sensi org. 15 November 2022. +http:// www.sensi org /.../LUT\_LS\_RDv1\_1.doc >.