System Design Document

For

Library Management System

Prepared for:

Ardeshir Bagheri

Prepared by:

Peter Abelseth
Ken Anglas
Matthew Hinton
Daniel Huettner
Sardor Isakov
Jeremy Lerner
Carlos Lozano

Term project for CPSC 2301 - Langara College

Fall 2012

Table of Contents

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Carlos Lozano	Initial version of this document sections 0-8	08/11/12
2.0	Daniel Huettner	Corrected section 3 – reworked contents	12/11/12
2.1	Jeremy Lerner	Corrected section 7 – Grammar & logic	12/11/12
2.5	Carlos Lozano	Added Appendixes 1-3	12/11/12
3.0	Carlos Lozano	Corrections	09/12/12

0. Overview of System Design

The overall system design objective is to provide a well-organized, modular design that will reduce system's complexity, simplify change, and result in an easy implementation. This will be accomplished by designing a strong and cohesive system with low coupling. In addition, this we will provide interface design models that are consistent, user friendly, and will represent easy to follow transitions through the different subsystems.

This document is aimed to developers and system designers. It provides a comprehensive architectural overview of the system using a number of different architectural views to depict different aspects of the structure. It is intended to capture and convey the significant architectural decisions made on the system.

The System Design Document is based on the Software Requirements Specifications documents for the Libris - Library Management System. This document describes the goals and trade-offs, the subsystem decomposition, the concurrency identification, the hardware and software mapping, the data management, the global resource handling, the software control techniques, and the boundary conditions for the Library Management System requested by the CICSR Library at the Faculty of Engineering at University of British Columbia.

1. Current System

This is a new system based on our Requirements Specifications Document and ideas from different existing Library Management Systems software used in local libraries in the Metro Vancouver area including public libraries and libraries located at diverse educational institutions including Langara College and The University of British Columbia.

A parallel research led us to diverse software companies that included download trials and videos that helped us understand the organisation of library software and the jargon used by librarians, and gave us a better insight about the problem faced at libraries and what possible solutions were available.

Amongst the existing solutions we found we pay special attention to the Sierra Library System used at Langara College and to ResourceMate from Jaywil Software that was easy to follow and had valuable documentation regarding functionality that should be available in a library management system for a high school and colleges/universities.

2. Definitions, Acronyms and Abbreviations

Browser Screen(s): The front end screen for guests patrons to search for resources.
Catalog: Group of operations concerning creation, edition, import, export and search on full resources catalog.
Circulation: Term used to grouped checkout, check-in, renewal, reservations, reference and fines management.
Client Program: The program running on browsing workstations scattered throughout the library.
Guest: Any external person that visits the library who is not allowed to checkout resources or anyone who searches the catalog and does not log in to the system.
History: A list of all records for a given resource or patron.

Librarian: Any employee of the library with or without access to the system, also known as Staff Member. Librarian serve patrons or help patrons do research tasks.
Librarian Interface: Screen for the librarians to serve patrons or to perform administrative tasks. Depending on their job, different libraries get to see less or more options.
Libris: Latin for books.
LSM: Library Management System
Patron: Anyone using the system who is allowed to checkout resources of the system.
Reserve: A resource can be requested by other patron when it is already checked out.
Reservation: Resources on reserve.
Reference: A resource is placed on reserve for a predetermined period of time by a faculty member. Patrons may only consult reference items while in the library.
Renewal: To extend the period of a resource reaching its due date.
Resource: An item the library carries, i.e.: books, magazines, videos, etc.
Resource copy: A specific copy of a given resource.
Server Program: The application running on the server machine.
Staff Member: A more generic way to identify librarians and other employees of the library that have access to the system.
To-Do: Tasks staff members need to accomplish during a specific date.

3. References and Acknowledgements

Langara College Library. Sierra Tour. September 2012.Tour to the Langara Library's LMS software application. <http://youtu.be/J4H4veWIFsY>

Jaywil Software Development. Resource Mate. October 2012. Library Management System for Schools and Colleges. <http://www.resorucemate.com>

Evergreen. Evergreen Open-source library system. Open source LMS software. <http://www.open-ils.org/>

Our most sincere gratitude to the Langara Library, the Burnaby, the North Delta, and the Vancouver Public Libraries staff for their invaluable information provided to us to help us understand diverse library requirements during the requirements elicitation stage of this project and for clarifying doubts that raised during our design stage.

1. Design Goals

1.1. Definition

Considering the non-functional requirements described in the SRS document (see Appendix 1), the global requirements from the problem statement and the pseudo requirements, we will describe the design goals for Libris - Library Management System.

The programming language selected for this system is Java with support of MySQL as database Management System. Java, as an Object Oriented programming language, allows for rapid development of applications following the iterative software development model. MySQL was selected as it is an Open Source relational database that is reliable, maintainable and easy to implement, and allows multi-user access to the database, which is one of the design goals.

Functional requirements are considered as they influence the way the system should operate. However, not all requirements found during the Requirements Elicitation stage will be taken in consideration as their priority is not high enough to be implemented in the first version of the software; those requirements will be added-value for next versions. All high priority requirements, those the dictate what the system must do, and some medium priority requirements, those that are required eventually but could wait, will be considered for this version (See Appendix 2).

The resulting class diagrams for this system can be seen in Appendix 3.

1.2. Design Goals Classification

The following are our design criteria grouped by the following categories:

Performance Criteria:

Response Time (High Priority)	The maximum response time is 48 seconds, with each request not taking longer than 2 seconds. The server will only be able to process requests one-at-a-time (on a first-come-first-served basis). Response time includes the amount of time needed to process the other requests ahead this request in the first-come-first-served queue. Clients can only send one request at a time (that is, they cannot send another request until a response is received for the previous request), so it is impossible to have more than 24 requests in the server's request queue at any given moment.
Throughput	30 tasks per minute

Dependability Criteria:

Robustness (High Priority)	The system must be able to survive invalid user input. Control objects on the client side must check each value stored in an entity object before sending a request to the server. The same is also done on the control object on the server side.
Reliability	Mean time to failure is once per week
Availability (High Priority)	System should be available to library staff and patrons during opening hours, outside of that window it is up to the administrator to take it down for backups and maintenance.
Security	The system provides both client-side and server-side access control to restrict users to a predefined set of transactions based upon their user class (guest, student, faculty, staff, or admin).

Maintenance Criteria

Extensibility (High Priority)	Since a closed architecture is used, new functionality and classes can be easily added to the system.

Modifiability	Since a closed architecture is used, adding functionality to the system in later revisions may be difficult, especially if the additions were not anticipated.
Adaptability	It is not possible to port this system to another application domain.
Portability (High Priority)	System should be able to run on any OS and hardware that supports JAVA and Java Swing functionality.
Readability	Code should be understandable provided that the user has programming experience and access to the class diagrams
Traceability of requirements (High Priority)	Since a closed architecture is used, the requirements of the system can be easily traced, typically in one vertical slice.

End user criteria

Usability (High Priority)	System should be intuitive and easy to operate. Patrons should require no training or just the help from messages on screen. For Library staff, the system should be simple and should consider correct flow of operations and guide users during complex transactions.

1.3. Trade-offs

Delivery time vs. functionality	If our team runs low on time, we may choose not to implement some of our low-priority trade-offs for this release of our software.

No other trade-off has been found at this stage.

2. Subsystem Decomposition

Overall, for our system we have chosen a three-tier client-server architecture (see Figure 2.1). Our client subsystem is responsible for providing a friendly interface for the user and ensuring that the data it sends to the server is correct. Our server subsystem is responsible for providing access control and security to protect our database from requests sent from malicious client programs. For both our client and server subsystems, we have chosen a closed, layered architecture. Figure 2.1.

Our client subsystem is divided into three distinct layers (see Figure 2.2). The top layer, called the user interface subsystem, provides a friendly interface for the user. The middle layer, called the control layer, provides the application logic. User requests sent from the top layer must pass through the middle layer before it can be sent to the server. The middle layer verifies the correctness of the data from the user then translates the request into one or more SQL queries to be sent to the server. The bottom layer, called the server interface subsystem, provides a means of communication with the server. It is responsible for establishing a socket connection with the server and sending requests. Hence, once a request is verified and translated by the middle layer, it is sent to the server interface subsystem, which sends the request and receives a response from the server.

For the client, the subsystems contained within the middle layer are of particular interest. The configuration management subsystem is responsible for reading from and writing to a simple configuration file, which stores information such as the IP address and port number of the server. The session management subsystem is responsible for server authentication. The error management subsystem is responsible for handling all errors. The data management subsystem is responsible for preparing requests to send to the server for adding, removing, searching, and modifying data. The services offered by the various client subsystems are shown in Figures 2.3 and 2.4.

Similarly, our server subsystem is divided into three distinct layers (see Figure 2.5). The top layer provides an interface for the user as well as an interface for the client. That is, the top layer receives requests from the local user and from each of clients. The middle layer provides the application logic. This layer is responsible for enforcing access control to ensure the user making a request is authorized to do so. The bottom layer provides a means of communicating with the database. The services offered by the various server subsystems are shown in Figures 2.6 and 2.7.

Top Level Component Diagram

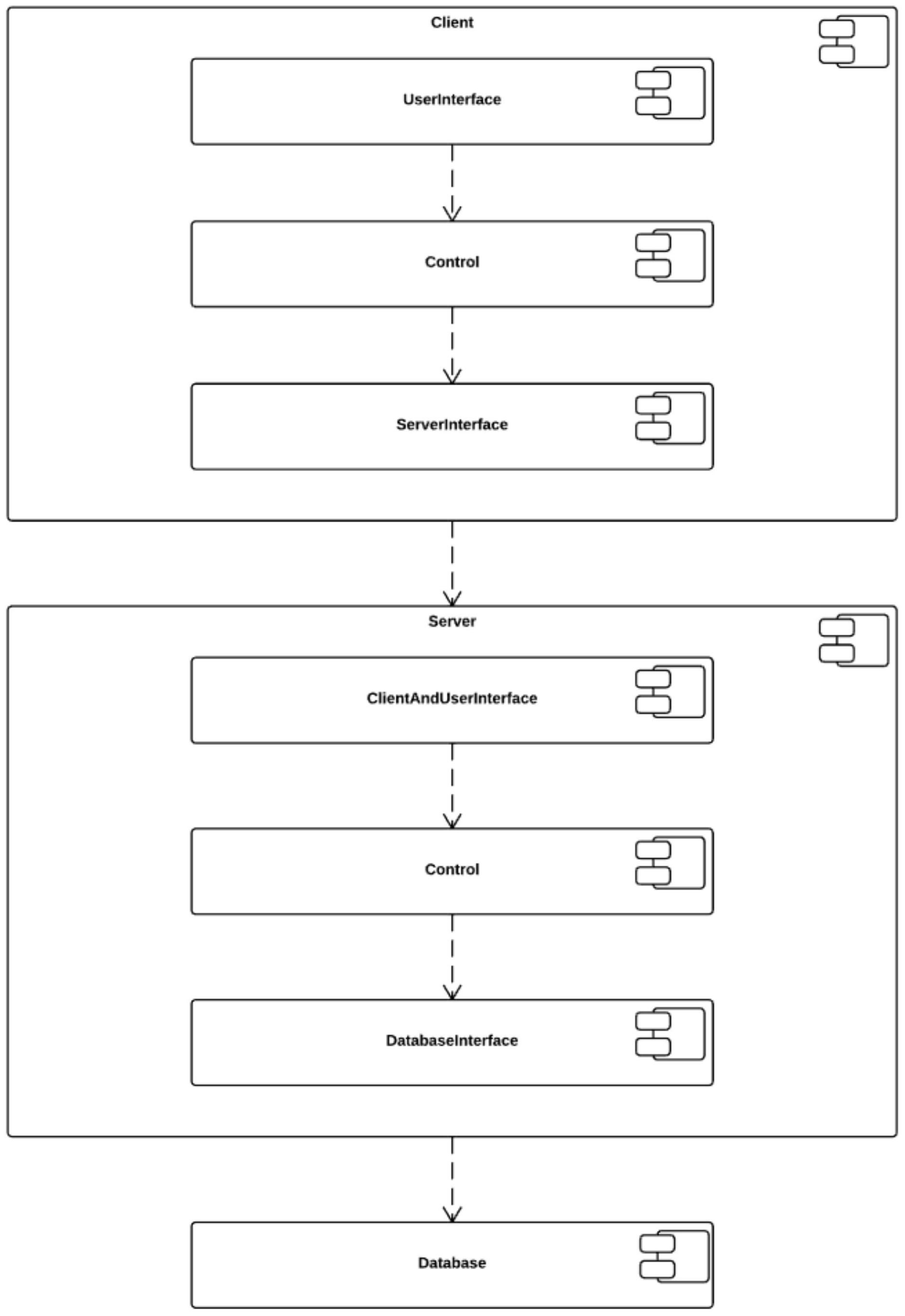


Figure 2.2: Client Subsystem Decomposition

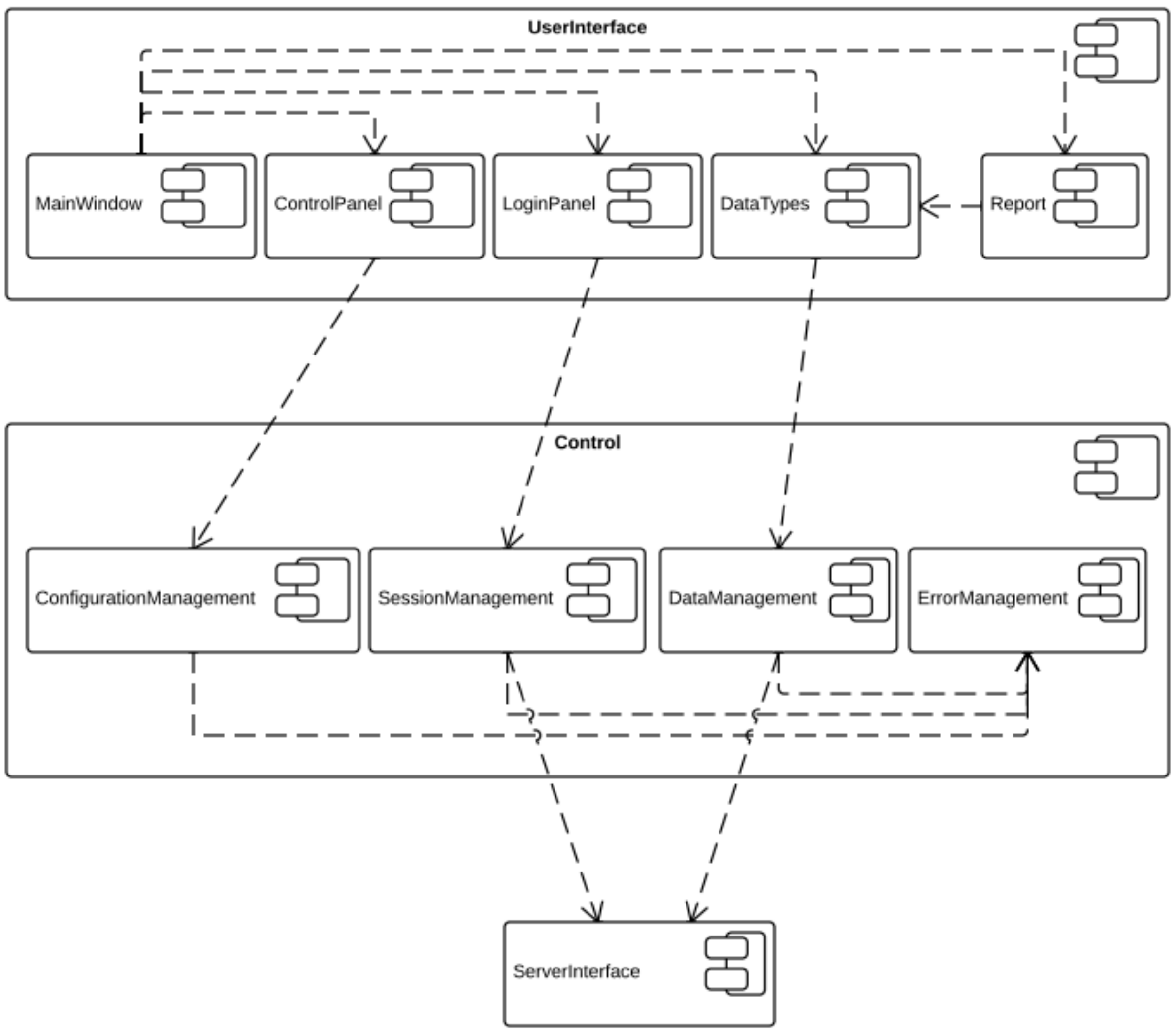


Figure 2.3: Client Subsystem With Services Identified

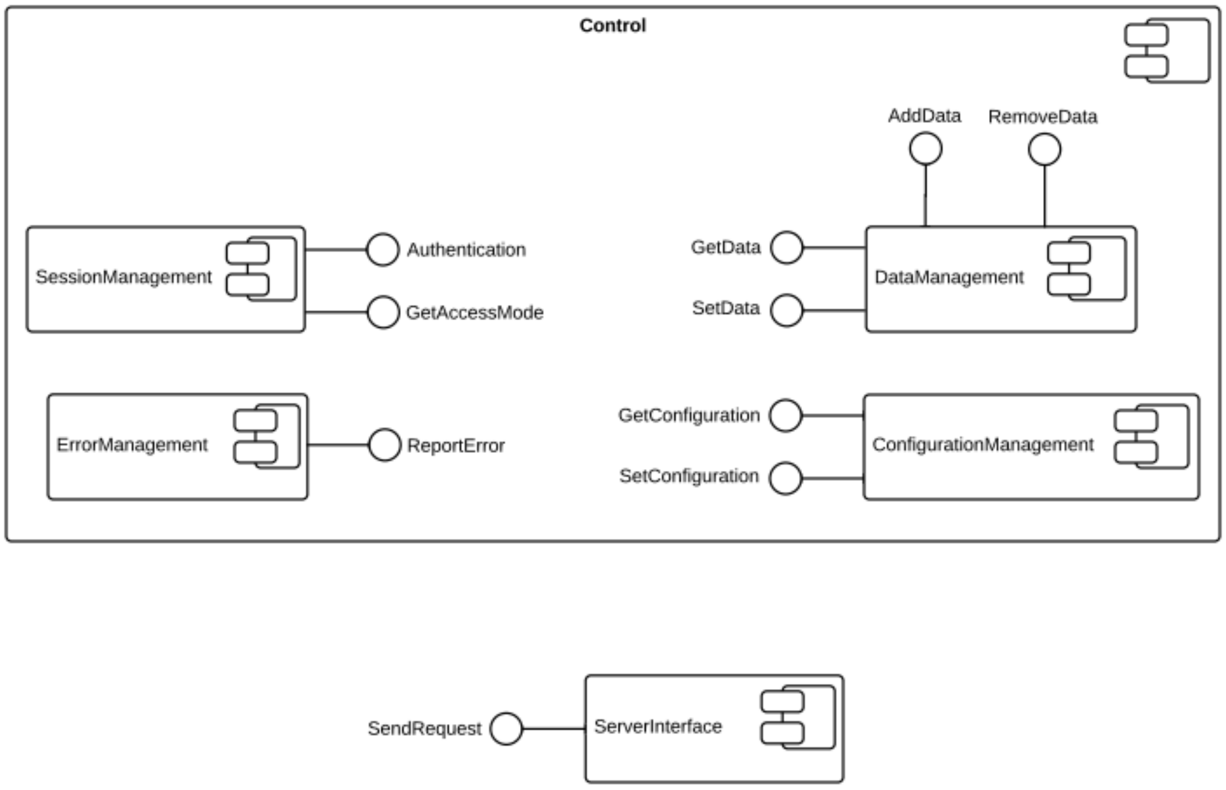
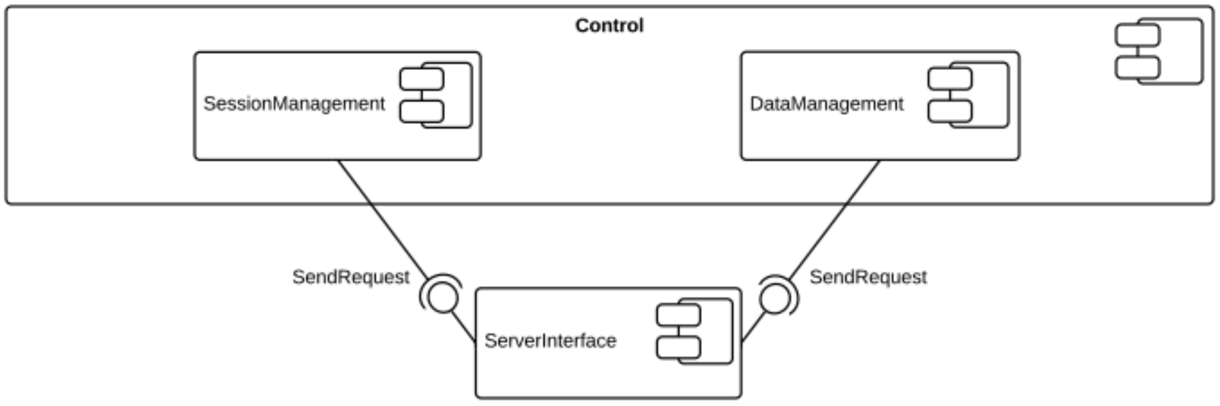


Figure 2.4: Client Subsystem Relationships



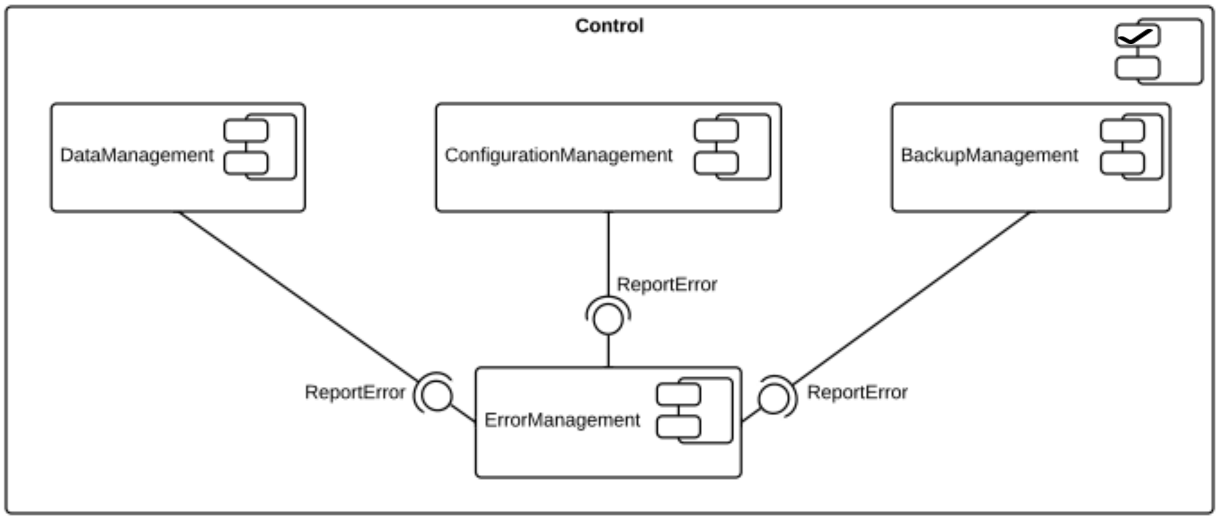


Figure 2.4 Continued

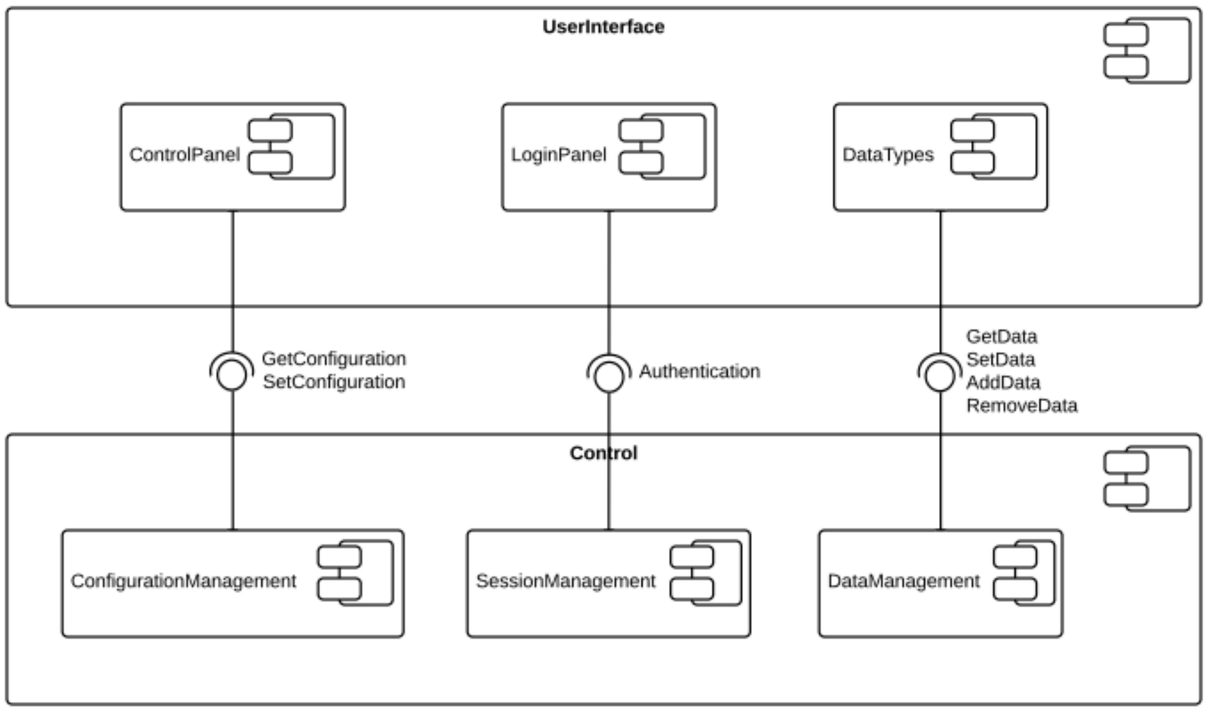


Figure 2.5: Server Subsystem Decomposition

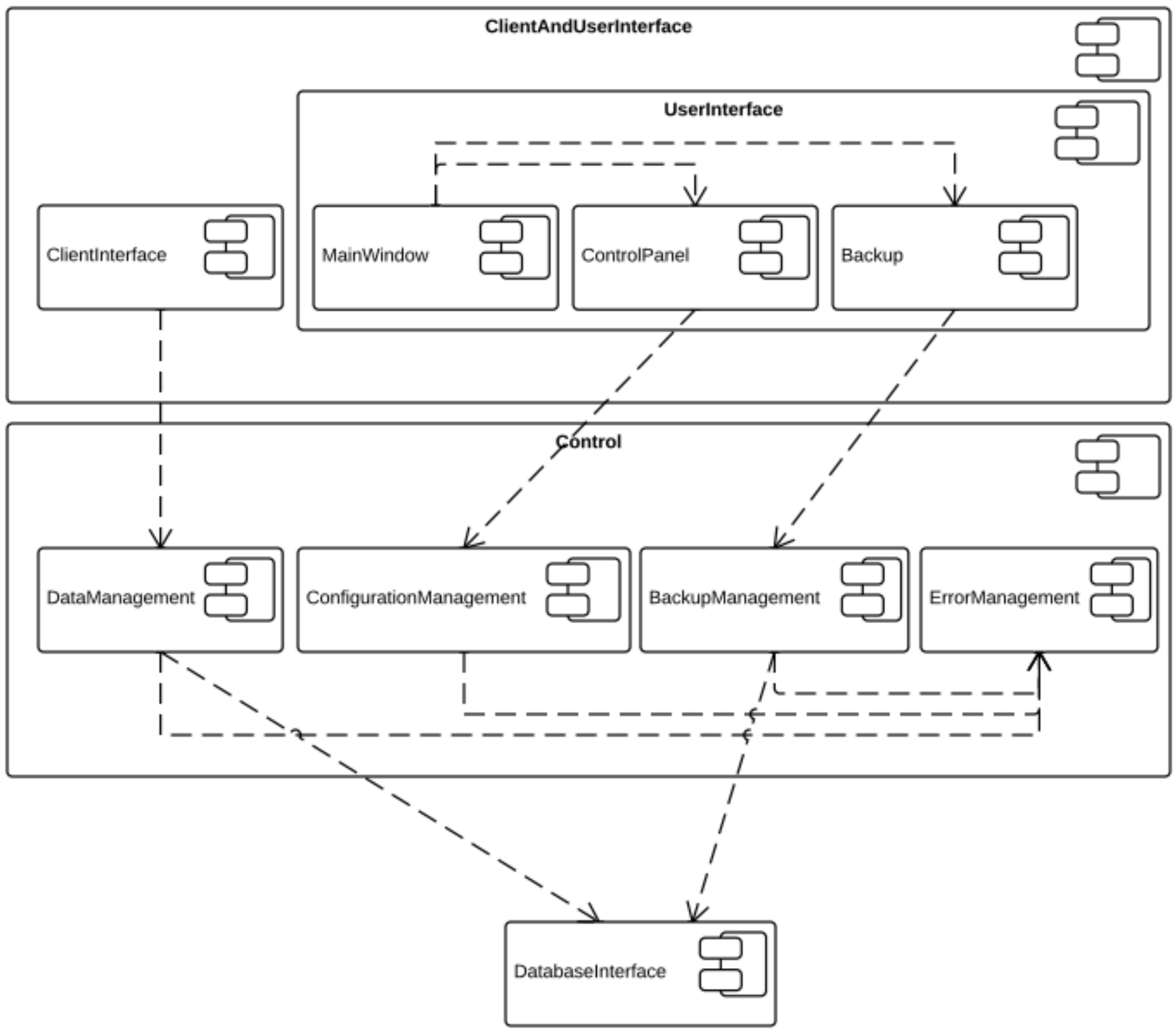


Figure 2.6: Server Subsystem With Services Identified

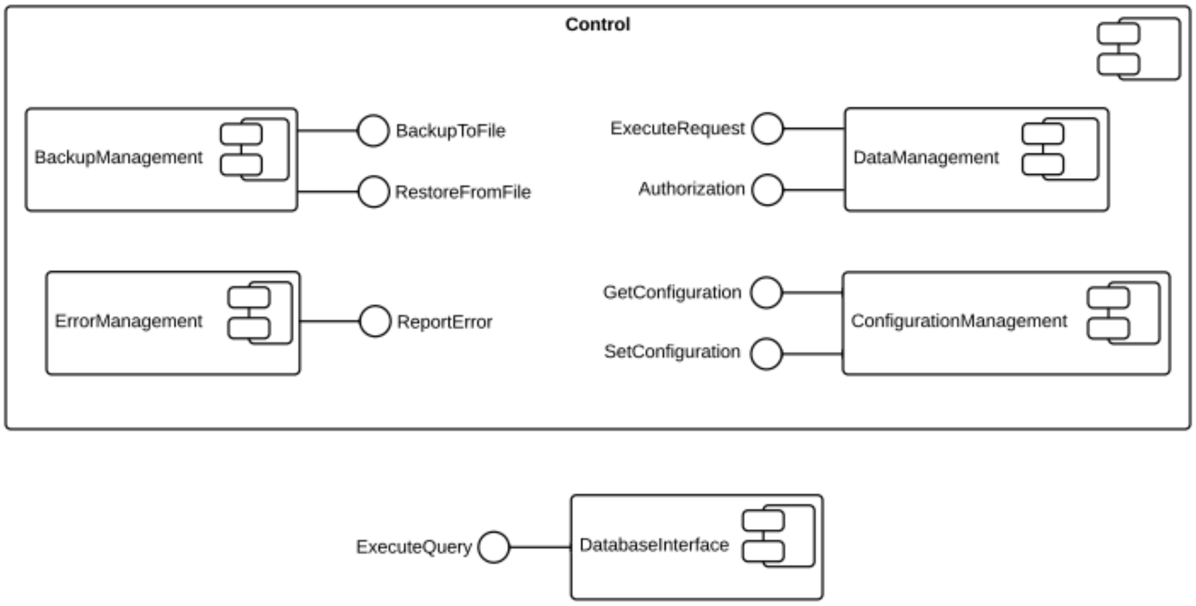
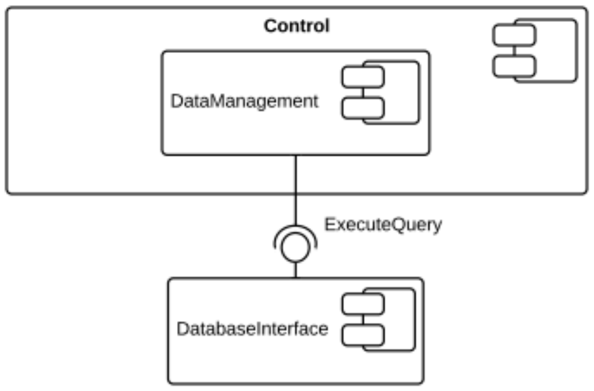


Figure 2.7: Server Subsystem Relationships



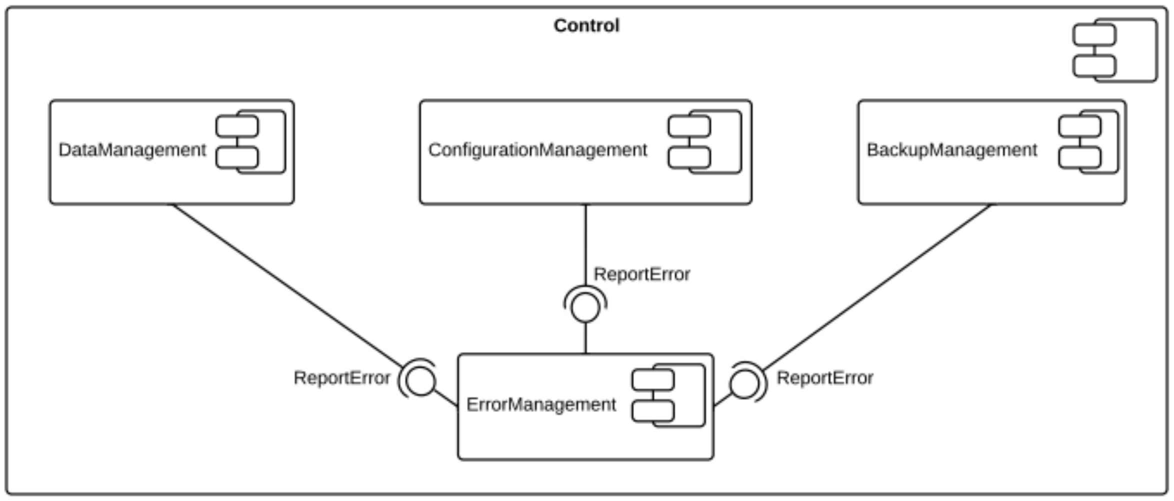
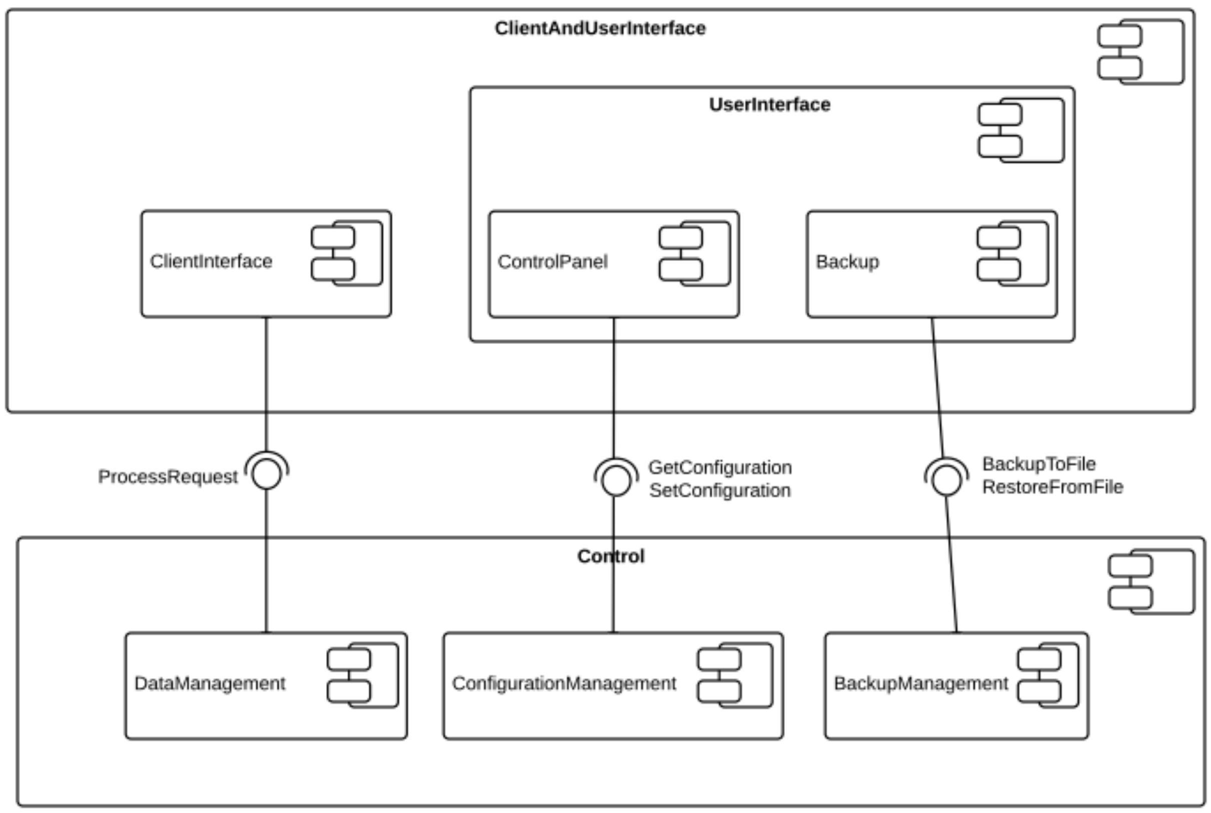


Figure 2.7 Continued



3. Concurrency Identification

Like most modern server programs, the Libris server program is multithreaded. It has a main thread listening for connection requests (ClientConnectionManager class), and handler threads, each for communicating with a single client (ClientConnectionThread class).

Access to the database is serialized, however, to ensure that multiple clients accessing the same data will not corrupt the integrity of the data. That is, only one database request may be executed at a time. Therefore, although the server instantiates multiple threads for concurrent communication with multiple clients, any requests sent from the clients are handled sequentially by the server.

The client program will only be able to send one query at a time and this query will only consist of one query. If the client requires multiple queries, it will serialize its requests and send each request to the server one at a time. This allows the server to not halt and perform a large query sent by one client while the other clients wait for their request to be performed.

4. Hardware/Software Mapping

4.1. Hardware

The system will run on any hardware that is able to run Java runtime programs. As Java programs are not resource demanding we are recommending a minimum configuration for the server and the clients. The database management engine

Server Machine Requirements:

Parameters	Minimum
Processor	Intel Atom N2600 1.6 GHz – One Quad core processor
RAM	4 GB
Monitor Resolution	1024 x 768
Operating System	Windows 2008/R2 – Linux Server with LAMP
Peripheral Devices	Standard 101 Keys Keyboard & Mouse
Video RAM	512 MB
Hard Disk Space	100 GB

Client Machine Requirements:

Parameters	Minimum
Processor	Intel Core i3 2.13 GHz
RAM	1 GB
Monitor Resolution	1024 x 768
Operating System	Windows 95/2000/XP/Vista/7/8
Peripheral Devices	Standard 101 Keys Keyboard/Mouse
Video RAM	512 MB
Hard Disk Space	50 GB

4.2. Software

For the implementation the hardware platform that the software runs on, will be mid-level performance desktop computer with standard hardware components used for fairly modern computers. We chose this because of the availability of desktops and their low price.
We are expecting a fairly high response time between client and server computers, the overhead needed for communication between the client and server is fairly small so our desired response time is pretty high.

The expected transaction rate for our given system has yet to be stress tested to experience under what conditions is our system under heavy load, however we are expecting a minimum of 24 Requests/min.

We do not need any extra piece of hardware to handle the data generation rate.

The response time and information flow rate should not exceed the available communication bandwidth between subsystems. Our client/server shall be implemented within a LAN and the information flow is minimum, so we should not have any problems with that.

As long as the implemented server hardware has sufficient memory to run the server which as nearly all modern computers do, there should be more than enough memory to buffer bursts of request.

For the implementation of our system, the subsystems are mapped on existing hardware/software as follows:

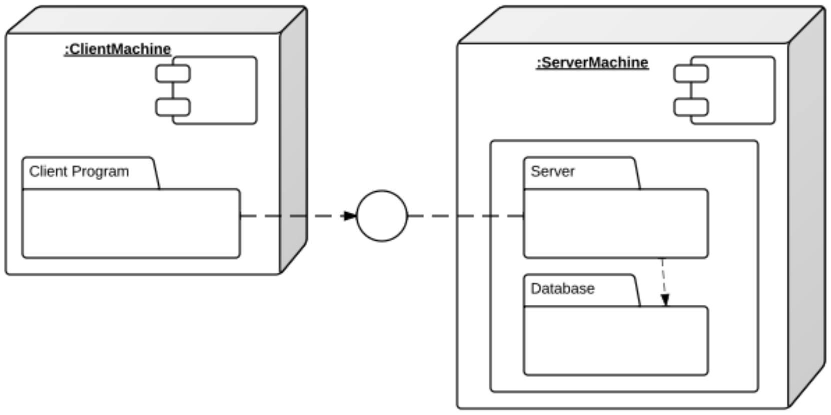


Figure 4.1 – Hardware/Software Mapping

4.3. Allocation of Resources

The computation rate for this small system is not demanding and only requires one processor. Any multithreading that has been implemented on the server, will be handled under java threads to allow execution on a single processor as well as multi core processor.

4.4. Connectivity

For our system, we will be using a star LAN topology for physically connecting the both the client and server environment together.

The following is a diagram of the physical connectivity for our system:

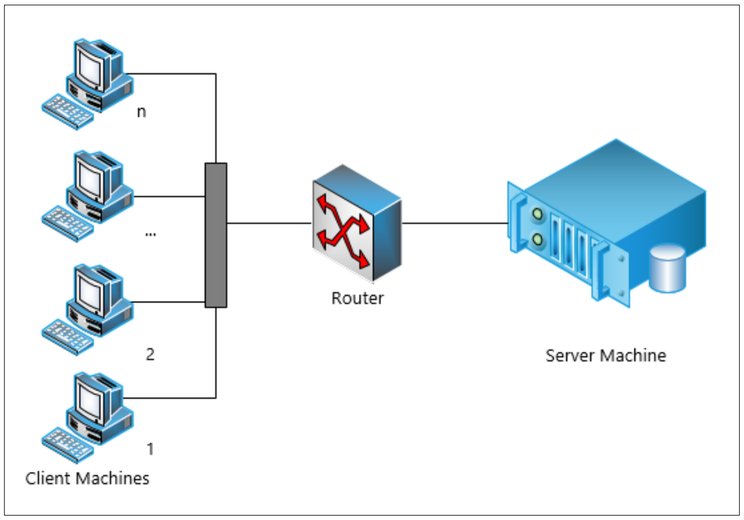


Figure 4.2 – Physical Connectivity

4.5. Network architecture

The transmission media will be an Ethernet cable setup through a particular LAN network. We have yet to test the reliability of wireless communication within our system, however our system is designed to interact within a single LAN setup so wireless communication should not be an issue for our system.

Our system between the client and server will establish a secure TCP connection and maintain that connection for the client's life span. The interaction is asynchronous, much like a regular client/server paradigm, the server waits for incoming requests and handles them accordingly.

The estimated bandwidth requirement is well below 250 Kbytes/sec, any modern LAN setup should be able to handle our system easily.

5. Data Management

5.1. Persistent Data Design

To store the persistent data required by our system, such as resources, users, loans, etc., a relational database is being used. There are several relationships between the data that is being stored so using a relational database makes the most sense to maintain these constraints. The other option is to use a file system to store our persistent data but this method is cumbersome, has low access speed, and is difficult to maintain the required constraints. An object-oriented database is a possibility but none of our persistent data is complex enough to justify using an object-oriented database. Object-oriented databases also are not as mature as relational databases thus tools for using one are scarcer.

It has been decided to use MySQL DBMS to satisfy the need for persistent data storage. MySQL offers concurrency control, scalability, and security options. The database is designed to be centralize to reduce implementation time and keep the system simple. A distributed database is out of scope for this system because the entire system will be running on a local network. The database is not designed around extensibility but it does allow for the possibility to be expanded upon to add more functionality to the system later one.

It is expected that the database will queried at least 20 times a minute during average use and during high use of the system there could be as many as 200 queries a minute. For searching for a resource there could be as many as 100 results returned by the database. On average for searching resource we expect the number of results to be around 20. On average for all other queries we expect there to be only 1 results and at most up to 50.

All queries will be created on the client system and validated on the server. This was decided to simplify implementation and reduce development time. The queries will use the Structured Query Language (SQL) format written for a MySQL database.

To prevent a full loss of data in the event of a database corruption or an external factor such as a fire, the system should support a full backup of all data in the database. The user will be responsible for making sure backups are performed and stored in a secure and off-site location. For the scope of this system, the database will be maintained on the same machine as the server. It will not be hidden but using Java's Java Database Connection (JDBC) it will be possible to deploy the database on a separate machine if necessary.

To connect our application to the MySQL database, JDBC is being used which offers an API to access the MySQL database but is also independent of which database management system(DBMS) is being used. This will allow for a change of DBMS in the future with minimal changes to code.

5.2. Entity Relationship Diagram

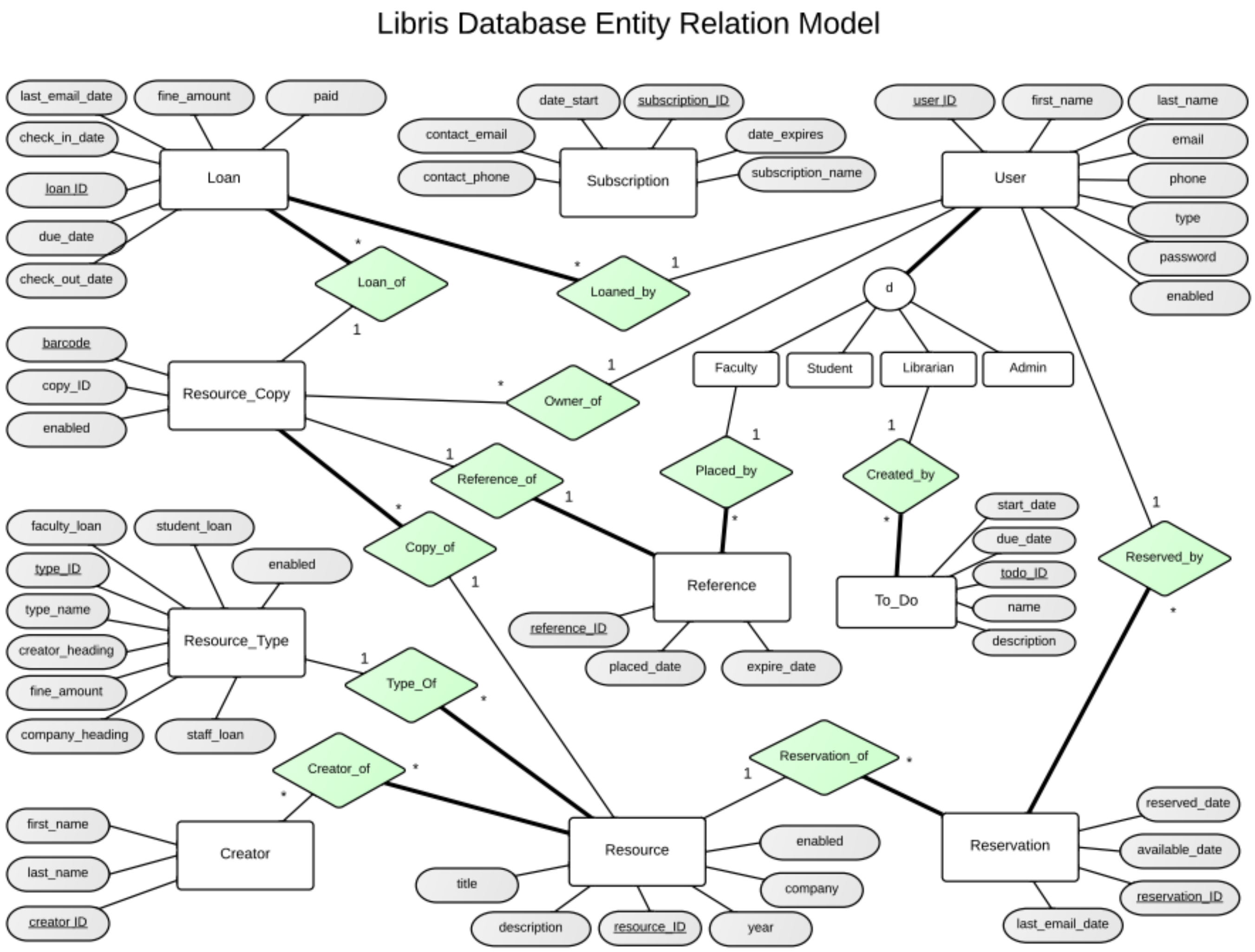


Figure 5.1- ERD for Libris Database

Notes:

- The user_ID of a User is auto generated by the database and given to the user upon adding the user to the database.
- The type of a User must be either "student", "faculty", "librarian", "admin".
- The User password should be encrypted.

- The enabled attribute of a User denotes whether the User is active and must be either true or false.
- The permissions of a User and what information they are able to access is determined by their type.
- A fine for a Loan cannot be partially paid, it must be paid in full or not at all.
- A reservation can be made only when a resource is not available to check out.
- A resource owned by a user can never be checked out.
- Only a user of type librarian, admin, or faculty may place a resource on reference.
- Only a user of type librarian or admin may create a To-Do.
- The enabled attribute of a ResourceCopy determines if the copy is in active circulation or not and must be either true or false.
- The enabled attribute of a Resource determines if the Resource is listed or not and must be either true or false.
- The enabled attribute of a ResourceType determines if the type is being used or not and must be either true or false.

5.3. Description of Database Tables

T1: User (user_ID, first_name, last_name, password, email, phone, type, enabled)

- Represents:
  - Strong entity set User
- Primary Key (user_ID)
- Constraints:
  - user_ID cannot be null
  - first_name, last_name cannot be null
  - email cannot be null
  - type must be either “student”, ”faculty”, ”librarian”, or “admin”
  - enabled must be either true or false

T2: To_Do (todo_ID, name, description, due_date, start_date, user_ID)

- Represents:
  - Strong entity set To_Do
  - Relationship set Created_by
- Primary Key (todo_ID)
- Foreign Key (user_ID) references User(user_ID)
- Constraints:
  - todo_ID cannot be null
  - due_date must be after start_date
  - user_ID cannot be null

T3: Resource_Type (type_ID, type_name, creator_heading, company_heading, student_loan, faculty_loan, staff_loan, fine_amount, enabled)

- Represents:
  - Strong entity set Resource_Type
- Primary Key (type_ID)

- Constraints:
    - type_ID cannot be null
    - type_name cannot be null
    - student_loan, faculty_loan, staff_loan cannot be null.
    - fine_amount must be greater than or equal to zero.
    - enabled cannot be null

T4: Subscription (subscription_ID, subscription_name, start_date, expire_date, contact_phone, contact_email)

- Represents:
    - Strong entity set Subscription
- Primary Key (subscription_ID)
- Constraints:
    - subscription_ID cannot be null
    - subscription_name cannot be null

T5: Resource (resource_ID, title, year, description, company, enabled, type_ID)

- Represents:
    - Strong entity set Resource
    - Relationship set Type_of
- Primary Key (resource_ID)
- Foreign Key (type_ID) references Resource_Type(type_ID)
- Constraints:
    - resource_ID cannot be null
    - title cannot be null
    - type_ID cannot be null
    - enabled cannot be null

T6: Resource_Copy (barcode, copy_ID, enabled, resource_ID, user_ID)

- Represents:
    - Strong entity set Resource_Copy
    - Relationship set Copy_of
    - Relationship set Owned_by
- Primary Key (barcode)
- Foreign Key (resource_ID) references Resource(resource_ID)
- Foreign Key (user_ID) references User(user_ID)
- Constraints:
    - barcode cannot be null
    - resource_ID cannot be null
    - copy_ID cannot be null
    - enabled cannot be null

T7: Reservation (reservation_ID, reserved_date, available_date, last_email_date, resource_ID, user_ID)

- Represents:
    - Strong entity set Reservation
    - Relationship set Reserved_by
    - Relationship set Reservation_of
- Primary Key (reservation_ID)
- Foreign Key (resource_ID) references Resource(resource_ID)

- Foreign Key (user_ID) references User(user_ID)
- Constraints:
    - reservation_ID cannot be null
    - reserved_date cannot be null
    - resource_ID cannot be null
    - user_ID cannot be null

T8: Reference (reference_ID, placed_date, expire_date, barcode, user_ID)

- Represents:
    - Strong entity set Reference
    - Relationship set Placed_by
    - Relationship set Reference_of
- Primary Key (reference_ID)
- Foreign Key (barcode) references Resource_Copy(barcode)
- Foreign Key (user_ID) references User(user_ID)
- Constraints:
    - reference_ID cannot be null
    - placed_date cannot be null
    - user_ID cannot be null
    - barcode cannot be null

T9: Loan (loan_ID, check_out_date, check_in_date, due_date, last_email_date, fine, fine_paid, user_ID, barcode)

- Represents:
    - Strong entity set Loan
    - Relationship set Loaned_by
    - Relationship set Loan_of
- Primary Key (loan_ID)
- Foreign Key (user_ID) references User(user_ID)
- Foreign Key (barcode) references Resource_Copy(barcode)
- Constraints:
    - loan_ID cannot be null
    - check_out_date cannot be null
    - due_date cannot be null
    - user_ID cannot be null
    - barcode cannot be null

T10: Creator (creator_ID, first_name, last_name)

- Represents:
    - Strong entity set Creator
- Primary Key (creator_ID)
- Constraints:
    - creator_ID cannot be null
    - last_name cannot be null

T11: Creator_has_Resource (creator_ID, resource_ID)

- Represents:
    - Relationship set Creator_of
- Primary Key (creator_ID, resource_ID)
- Foreign Key (creator_ID) references Creator(creator_ID)

- Foreign Key (resource_ID) references Resource(resource_ID)
- Constraints:
  - creator_ID cannot be null
  - resource_ID cannot be null

Libris Database Relational Diagram

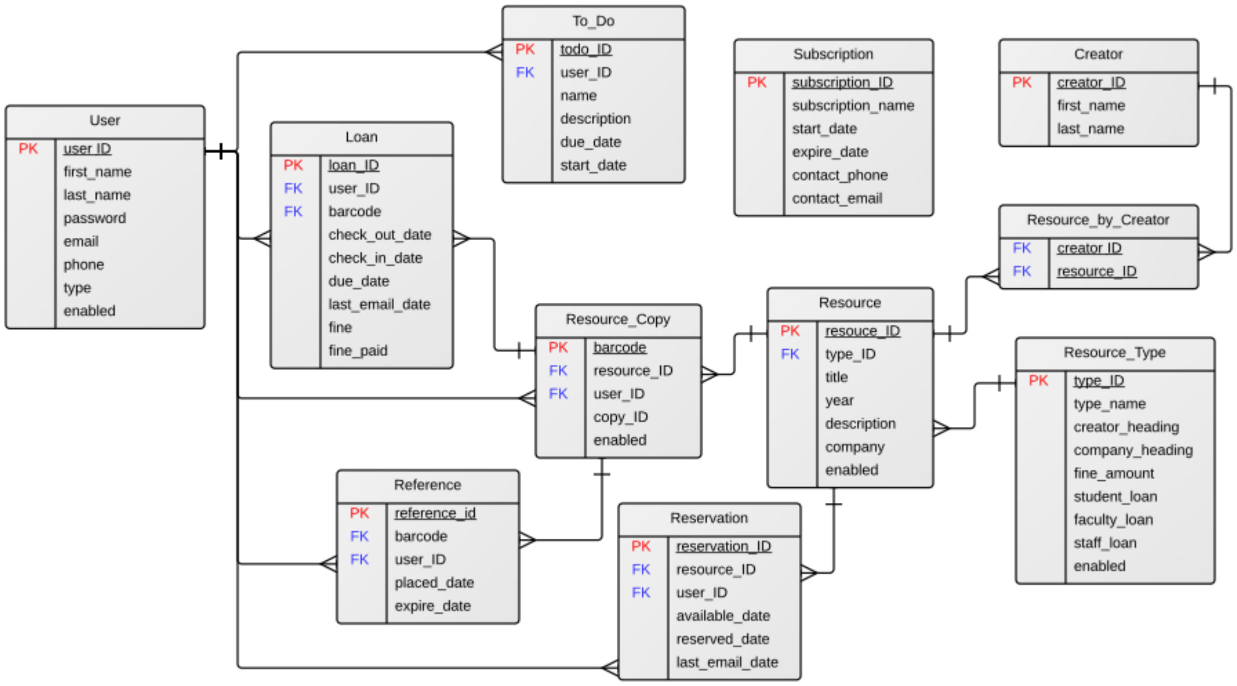


Figure 5.2 – Database Tables and Relationships

5.4. Functional Dependencies

T1: User (user_ID, first_name, last_name, password, email, phone, type, enabled)
FD1: user_ID → first_name, last_name, password, email, phone, type, enabled (Key)

T2: To_Do (todo_ID, name, description, due_date, start_date, user_ID)
FD2: todo_ID → name, description, due_date, start_date, user_ID (Key)

T3: Resource_Type (type_ID, type_name, creator_heading, company_heading, student_loan, faculty_loan, staff_loan, fine_amount, enabled)
FD3: type_ID → type_name, creator_heading, company_heading, student_loan, faculty_loan, staff_loan, fine_amount, enabled (Key)

T4: Subscription (subscription_ID, subscription_name, start_date, expire_date, contact_phone, contact_email)
FD4: subscription_ID → subscription_name, start_date, expire_date, contact_phone, contact_email (Key)

T5: Resource (resource_ID, title, year, description, company, enabled, type_ID)

FD5: resource_ID → title, year, description, company, enabled, type_ID (Key)

T6: Resource_Copy (barcode, copy_ID, enabled, resource_ID, user_ID)
FD6: barcode → copy_ID, enabled, resource_ID, user_ID (Key)

T7: Reservation (reservation_ID, reserved_date, available_date, last_email_date, resource_ID, user_ID)
FD7: reservation_ID → reserved_date, available_date, last_email_date, resource_ID, user_ID (Key)

T8: Reference (reference_ID, placed_date, expire_date, barcode, user_ID)
FD8: reference_ID → placed_date, expire_date, barcode, user_ID (Key)

T9: Loan (loan_ID, check_out_date, check_in_date, due_date, last_email_date, fine, fine_paid, user_ID, barcode)
FD9: loan_ID → ccheck_out_date, check_in_date, due_date, last_email_date, fine, fine_paid, user_ID, barcode (Key)

T10: Creator (creator_ID, first_name, last_name)
FD10: creator_ID → first_name, last_name (Key)

T11: Creator_has_Resource (creator_ID, resource_ID)
FD11: creator_ID, resource_ID (Key)

5.5. Database Normalization:

After reviewing the functional dependencies of the tables it is observed that for each table, the key is a super key and determines every other attribute of the table. Therefore, all of the tables are in Boyce-Codd Normal Form (BCNF).

5.6. Database Sample Queries

```
-- --------------------------------------------------------
-- Used for a general search of a Resource
-- --------------------------------------------------------
SELECT
    IF(title LIKE '%SEARCHTERM%', 2,
        IF(description LIKE '%SEARCHTERM%', 1,0)
    ) AS title_first,
    resource.resource_ID, resource.title, resource.year,
resource.description, resource.company, resource.resourceType_type_ID,
creatorHasResource.creator_creator_ID
FROM
    resource, creatorHasResource
WHERE
    resource.enabled = TRUE AND
    (resource.title LIKE '%SEACHTERM%' OR
    resource.description LIKE '%SEARCHTERM%') AND
    creatorHasResource.resource_resource_ID = resource.resource_ID
ORDER BY
    title_first;
```

```
-- -----------------------------------------------------
-- Used for a specific search of a resource
-- -----------------------------------------------------
SELECT
    resource.resource_ID, resource.title, resource.year,
resource.description, resource.company, resource.resourceType_type_ID,
creatorHasResource.creator_creator_ID
FROM
    resource, creatorHasResource, creator, resourceType
WHERE
    resource.enabled = TRUE AND
    resource.resource_ID = creatorHasResource.resource_resource_ID AND
    resource.title LIKE '%TITLESEARCH%' AND
    resource.description LIKE '%DESCSEARCH%' AND
    resource.resourceType_type_ID = 999 AND
    creator.creator_ID = creatorHasResource.creator_creator_ID AND
    creator.first_name LIKE '%FIRSTNSEARCH%' AND
    creator.last_name LIKE '%LASTNSEARCH%';


-- -----------------------------------------------------
-- Used for logging a user in
-- -----------------------------------------------------
SELECT
    user_ID, first_name, last_name, email, phone, type
FROM
    User
WHERE
    enabled = TRUE AND
    user_ID = 999999999 AND
    password = MD5('MyPassword');


-- -----------------------------------------------------
-- Used for getting all current loans of a user
-- -----------------------------------------------------
SELECT
    loan.loan_ID, loan.check_out_date, loan.due_date,
loan.check_in_date, loan.fine, loan.fine_paid, resource.title
FROM
    user, loan, resourceCopy, resource
WHERE
    user.user_ID = 999999999 AND
    user.user_ID = loan.user_user_ID AND
    loan.resourceCopy_barcode = resourceCopy.barcode AND
    resourceCopy.resource_resource_ID = resource.resource_ID;
```

```
-- -----------------------------------------------------
-- Used for getting the total fines from a user
-- -----------------------------------------------------
SELECT
    user.user_ID, SUM(loan.fine)
FROM
    user, loan
WHERE
    user.user_ID = 999999999 AND
    user.user_ID = loan.user_user_ID AND
    loan.fine_paid = false;
```

6. Global Resource Handling

6.1. Access Control and Security

A number of restrictions are imposed on different user classes and the operations they are permitted to perform. Each user class has the permissions of the one listed above it:

- Guests can search and view information for resources.
- Students can place resources on reserve and have them checked out under their accounts.
- Faculties can have resources put on reference under their accounts, including resources foreign to the library.
- Staff members can check in, check out, place on reference, place on reserve, add, remove, and edit resources (and resource copies), and add, remove and edit users' information
- Administrators can add and remove resource types, waive fines, and perform database maintenance functions.

These restrictions are implemented at multiple layers. When a user logs into the Client application, they input a user ID and password. These are sent to the server, which authenticates the account and sends back an integer value corresponding to that user's permissions. Meanwhile, the server spawns a thread to handle the connection, which stores the user's permissions.

Subsequently, the client receives the integer and adjusts its GUI to show only the options available to the user. Every time the client attempts to access the database, the server-side thread checks its permissions. In this way, security is implemented on both the client and server sides.

The connections between client and server use Java's Socket API. All the communications are encrypted.

User Permissions Definition:

ReadSelf (bit 0): Permission to read the specified field in the specified table for an entry that is linked to the user currently logged in.
WriteSelf (bit 1): Permission to change the specified field in the specified table for an entry that is linked to the user currently logged in.

AddSelf (bit 2): Permission to add a new entry to the specified table that links to the user currently logged in.
RemoveSelf (bit 3): Permission to remove the specified entry from the specified table that links to the user currently logged in.
ReadOther (bit 4): Permission to read the specified field in the specified table for an entry that is not linked to the user currently logged in. This may be linked to another user of equal or lower user class.
WriteOther (bit 5): Permission to change the specified field in the specified table for an entry that is not linked to the user currently logged in. This may be linked to another user of equal or lower user class.
AddOther (bit 6): Permission to add a new entry to the specified table that links to a user other than the user currently logged in. This may be linked to another user of equal or lower user class.
RemoveOther (bit 7): Permission to remove the specified entry from the specified table that links to a user other than the user currently logged in. This may be linked to another user of equal or lower user class.

	Guest	Patron	Staff	Administrator
UserBasic	00000000	00000011	11110011	11111111
UserExtended	00000000	00000001	11010001	11111111
Resource	00010000	00010000	11110000	11111111
ResourceCopy	00010000	00010000	11110000	11111111
Subscription	00000000	00000000	11111111	11111111
ToDo	00000000	00000000	11111111	11111111
Loan	00000000	00000001	11111111	11111111
Reference	00000001	00000001	11111111	11111111
Reservation	00000000	00001101	11011101	11011101
Creator	00010000	00010000	11110000	11111111

Permissions Matrix

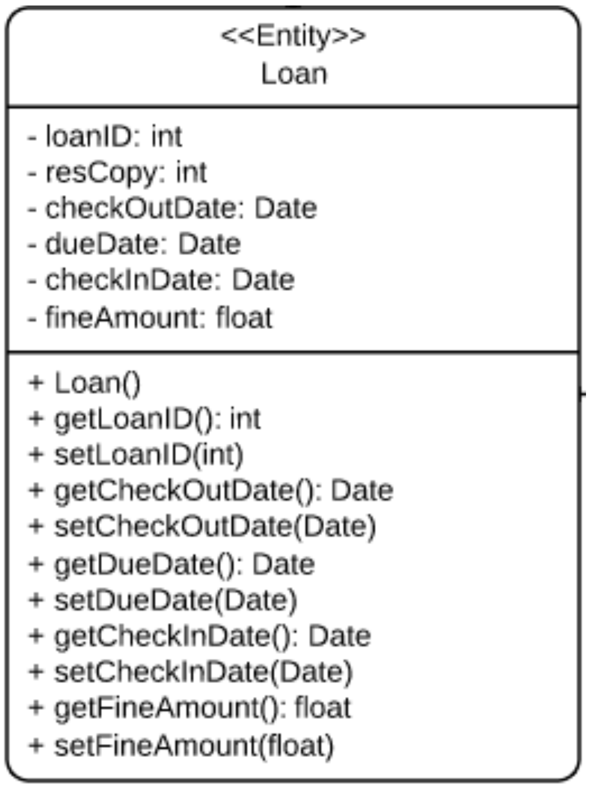
7. Software Control

In Libris Library Management System, we decided to use a combination of three possible control flow mechanisms in designing the system's global control flow. Event-driven control and thread mechanisms will be used the most, throughout the system.

For example, threads will be mostly used on server side of the system. The server will only create a new thread as a new client connects. Once the client disconnects the thread will be destroyed. Each Thread will respond to a client connection. First, the server authenticates the client and assigns a thread, which will handle all of the client’s requests.

To ensure a robust design with respect to concurrency, we define the following strategy for dealing with concurrent accesses to shared data:

1. Entity objects should not provide direct access to their fields. Instead, access to the object's state should be done through methods.



```
public class Loan {
        private int loanID;
        private int resCopy;
        private Date checkOutDate;
        private Date dueDate;
        ...

        public Loan() {
        }

        public int getLoanID {
                return loanID;
        }

        public void setLoanID(int
laonID) {
                this.loabID;
        }
        // . . .
}
```

Figure 7.1

2. Control objects should not be shared among threads. This should be mandatory when control objects survive the processing of a single request.

3. Method synchronization will be used with threads. When threads need to call the same method, we will use synchronization to ensure that the method is not accessed simultaneously by both threads.

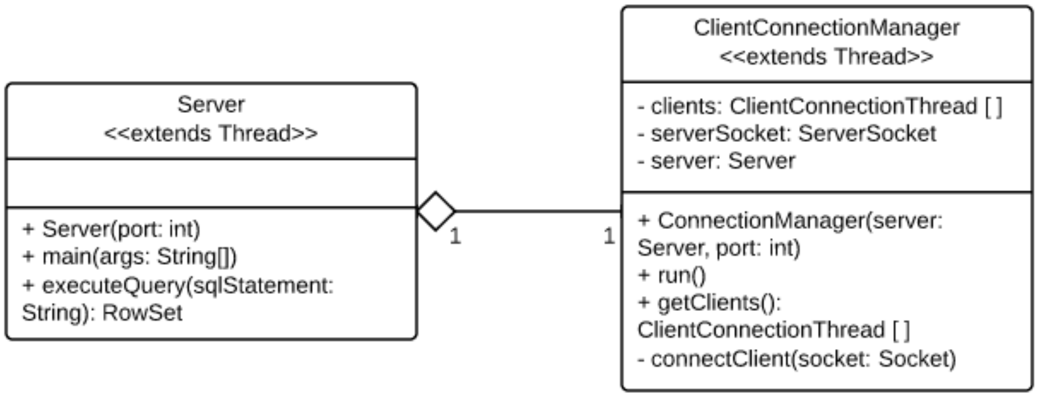

Server
<<extends Thread>>
+ Server(port: int)
+ main(args: String[])
+ executeQuery(sqlStatement: String): RowSet
1
1
ClientConnectionManager
<<extends Thread>>
- clients: ClientConnectionThread [ ]
- serverSocket: ServerSocket
- server: Server
+ ConnectionManager(server: Server, port: int)
+ run()
+ getClients(): ClientConnectionThread [ ]
- connectClient(socket: Socket)

Figure 7.2

4. Singleton objects will be used. This is useful when exactly one object is needed to coordinate actions across the system.

```
public class Singleton {
    singleton Singleton;

    private Singleton() {
    }

    public Singleton getReference() {
        if (singleton == null)
            singleton = new Singleton();
        return singleton;
    }

    public synchronized void execute (String arg) {
        //... perform operation here ...
     }
 }
```

Figure 7.3 – Singleton Class

The DBinterface class will be Singleton, it will ensure that thread accessing Database Interface controller will go through one instance (see figure 7.4).

```
package server.databaseinterface;

import java.sql.Connection;

public class DBInterface {
      DBInterface singleton;

      private static Connection connect = null;
      private static String mysql_username = "root";
      private static String mysql_password = "pass";
      private static String server_ip = "localhost";

      private DBInterface() {
      }

      public DBInterface getReference() {
            if (dbInterface == null)
                dbInterface = new DBInterface();
            return dbInterface;
      }

      /**
      *@param String sql statement
      */
      public synchronized void insertRecord(CachedRowSet rowSet)
throws Exception {
            Class.forName("com.mysql.jdbc.Driver");
            connect =
      DriverManager.getConnection("jdbc:mysql://localhost/librisDB …
      }
}
```

Figure 7.4 – DBInterface – Singleton Class

5. Regarding authentication, when user launches the program the system will show form for entering username and password. The system will grab username and password and send it to server to match it with the existing records in database. If existing record is found in database, server will send back corresponding database tuple. Client system then display appropriate information.

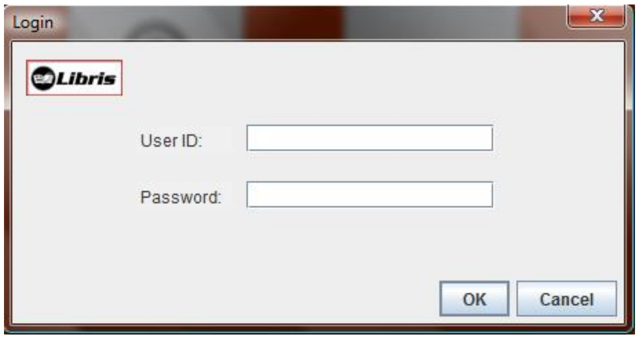


Figure 7.5 –Login Window

The server system does not have to know of the Client who is connecting to server. The client-side must know server's IP address and port number in order to connect. There is no network-wide name server.

8. Boundary Conditions

The following use cases are the ones concerning initialization, termination and failure for the client and server programs:

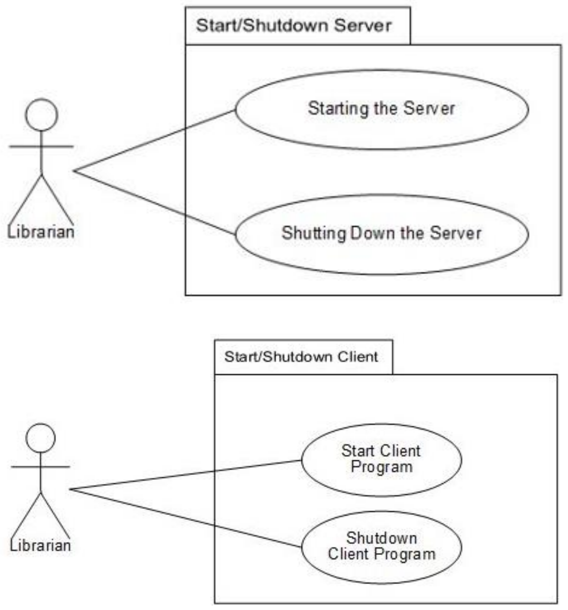


Figure 8.1 – Boundary Conditions UML Use Cases

Name	**StartServer**
Participating Actors	Librarian
Entry Conditions	Librarian has access to server room
Exit Conditions	Server program is up and running
Flow of Events	1. Librarian presses button to start server 2. Server loads OS and starts server services including the database management system. 3. Librarian launches the executable that starts the server program 4. Server program runs initial verification of dependencies including connectivity to the database and networking services. 5. Librarian verifies that server started correctly.
Special Requirements	Database did not connect: Librarian must check that the database management system stars and should contact a System Administrator if it does not. Networking subsystem did not start: Librarian should check that the server connects to other devices and contacts a System Administrator if it does not.

Name	**ShutdownServer**
Participating Actors	Librarian
Entry Conditions	Librarian has access to server room or to a remote console Server program is running
Exit Conditions	Server program has stopped
Flow of Events	1. Librarian checks that no other users are currently using the system 2. Librarian checks that no batch processes are running. 3. Librarian starts shutdown process. 4. Server closes connection to network resources and closes connection to the database. 5. Server stops services and starts shutdown of the operating system if necessary.
Special Requirements	Users Logged in: A broadcast is sent asking users to log out. Batch process is running: librarian waits or stop the process if it can be restarted.

Name	**StartClient**
Participating Actors	Librarian
Entry Conditions	None
Exit Conditions	Client program is up and running
Flow of Events	1. Librarian presses button to start client machine 2. Machine loads OS and starts basic services. 3. Librarian launches the executable that starts the client program 4. Client program runs initial verification of dependencies including connectivity to the database and networking services. 5. Librarian verifies that client program started correctly.
Special Requirements	Database did not connect: Librarian must check that the database management system started and should contact a System Administrator if it does not. Networking subsystem did not start: Librarian should check that the client machine connects to other devices and contacts a System Administrator if it does not.

Name	**ShutdownClient**
Participating Actors	User
Entry Conditions	Client program is running
Exit Conditions	Client program is stopped
Flow of Events	1. User chooses the option to exit program 2. Program closes all connections and terminates. 3. The system shutdowns services and the OS
Special Requirements	None.

Name	**OpenDatabaseConnection**
Participating Actors	ServerProgram, Database Management System
Entry Conditions	Server program is running
Exit Conditions	Server Program talks to the database
Flow of Events	1. System opens connectivity to the database 2. Database responds to request from server program
Special Requirements	Connection Down: Database does not respond request and needs to be restarted.

Name	**DatabaseConnectionFails**
Participating Actors	Administrator, Database Management System
Entry Conditions	Database connection is down
Exit Conditions	Server Program talks to the database
Flow of Events	1. Administrator troubleshoots database management system 2. Administrator restart database management system services. 3. System opens connectivity to the database 4. Database responds to request from server program
Special Requirements	Connection Down: Database does not respond request and needs to be restarted.

Name	**NetworkLinkFails**
Participating Actors	Administrator
Entry Conditions	Network link is down
Exit Conditions	Link is established
Flow of Events	1. Client program reports loss of connectivity 2. Administrator checks that TCP/IP is running and that physical link (including Ethernet cable and switches) is not operational and fixes issue. 3. Administrator retries connectivity to the network 4. Client program connects to the network
Special Requirements	Damaged Physical link: Administrator replaces damage cable or device. TCP/IP is not running: Administrator restart network services.

8.1. Initialization

System will be started at the beginning of the week or when needed during the weeks. During start-up the server program will check that the database and the networking subsystem have started or are functional and will start the necessary Java runtime programs.

Programs and databases will be checked for consistency and that all required files are available and have the correct version.

The first time the client program starts it will verify that the configuration file exist, if not the configuration file will be created with input given by the installer giving the IP address of the server program.

The server program will start as a console where no information is displayed unless the verbose mode is enabled. The client program will start on the Guest mode.

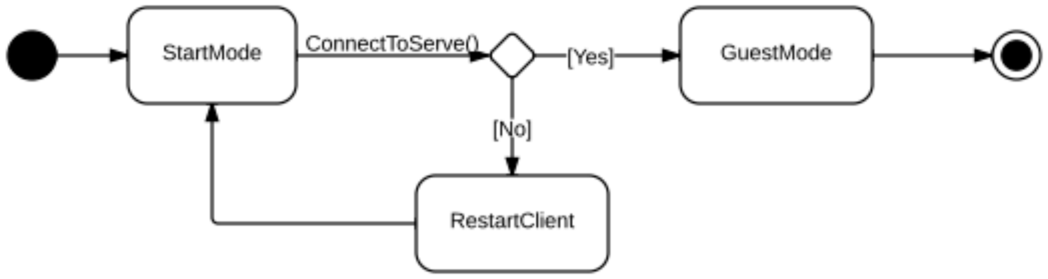


Figure 8.2 Client Program Initialization – State Diagram

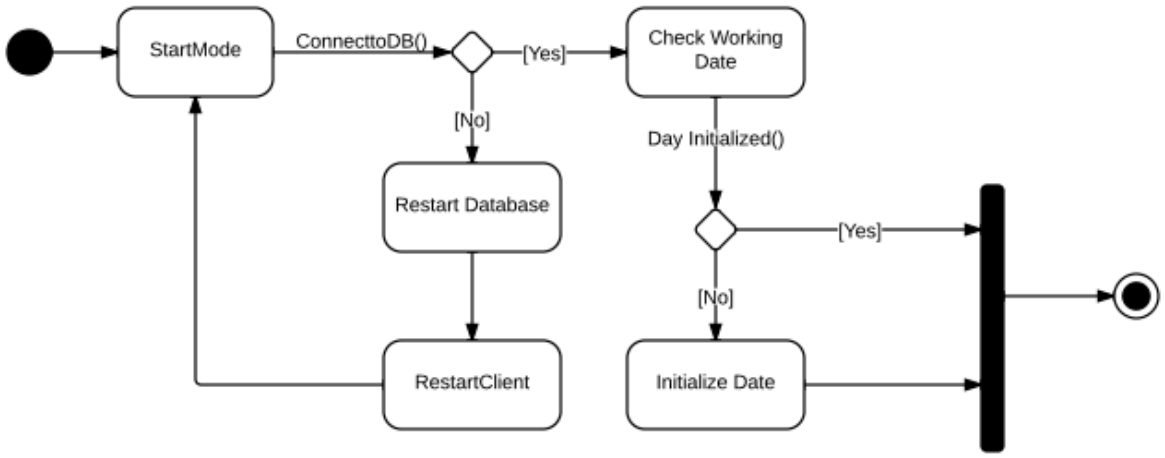


Figure 8.3 Server Program Initialization – State Diagram

8.2. Termination

Only administrators will be allowed to shut down the server program and only when it is necessary to fix a failure or to do system maintenance. The server program does not need to be shut down during the week unless the administrator plans to shut it down.

For standard shutdown the system should guarantee that all connections are closed and that no batch processes are still running. Once all connections are closed, including the connection to the database management system, the server it shut down.

A full database backup is recommended before shutting down the server for maintenance.

The client program can be terminated at any time without any consequence to the server program or to any of the other clients.

8.3. Failure

A global routine will be started when the server starts with the only purpose to be in charge of error handling. Any error found during the execution of the program must be handled by the routine that will display an error message and then return control to the current function if the error is recoverable. For unrecoverable errors (i.e. Database link broken) a clear message will be displayed indicating the error condition that should be a good clue for the administrator to troubleshoot problem.

An error during the execution of the client program should be handled by the same global routine. Input errors or missing information should be displayed as messages to the user so that they can correct errors and retry the operation.
When a communication link fails on the client program a message is displayed to indicate that program is executing in offline mode and that users should contact as member of the staff to help them.

There would be no backup communications link, .unless management at the library decides to add a redundant link on each client (i.e. wireless link).

During a major unrecoverable failure on the server program, a message will be displayed on the librarian workstations so that they could troubleshoot or contact the administrator. If a system restart is needed, initialization processes that happen only at the start of the day will be validated and ignored. There is no major difference between a normal conditions start up and a re-initialization after failure. Consistency checks should be run during each initialization regardless of the previous status.

Hardware failures will not be contemplated in this design, although it is recommended that the administrator of the system implements redundant storage units with RAID 5 or 6 and ideally enable replication to a redundant server that could be used in case of hardware failure. With a good Disaster Recovery plan, the risk of losing data is minimized and recovery time should be very short.

Hardware failure on client machines should be dealt with the administrator who can troubleshoot and repair failure as soon as possible. Out of order client should affect library patrons as there would be less stations available for their research.

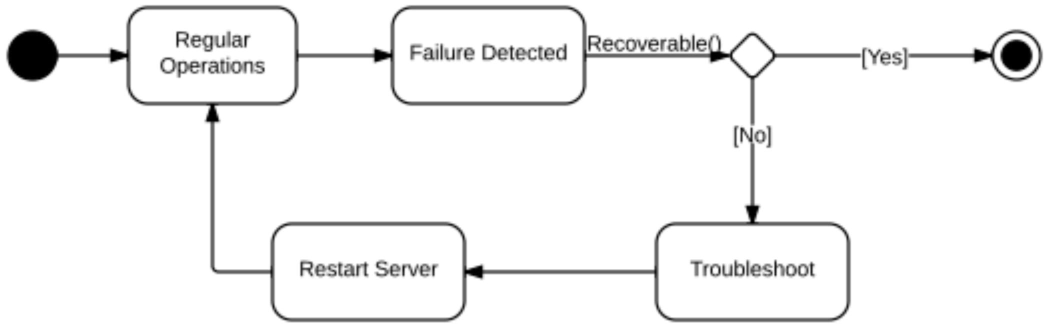


Figure 8.4 Server Program Error Handling – State Diagram

Appendix 1

Revised set of Non-Functional-Requirements:

#	FURPS Type	Non-Functional Requirement
101.	Performance (Response Time)	The client should take no longer than 5 seconds to respond to a simple requests and no longer than 10 second seconds to start displaying results for more complex requests that fetch multiple results, not exceeding 48 seconds to complete request.
102.	Performance (Concurrency)	System shall be able to perform several transactions at a given time, but only one user shall be able to change a database record at a time.
103.	Usability (Consistent User Interface)	System shall have clear, friendly and intuitive user interfaces adaptable to the type of user.
104.	Performance (Concurrency)	System shall be able to support up to 24 users at the same time.
105.	Usability (Human factor)	Guest and unidentified patrons shall not need training to browse the catalog.
106.	Performance (Throughput)	System shall be able to support more than 24 tasks per minute.
107.	Reliability (Robustness)	System shall be able to survive invalid user input.
108.	Reliability (Mean Time to Failure)	System shall be reliable having mean time to failure of one a week.
109.	Performance (Availability)	System should be available to library staff and patrons during regular opening hours to a rate of not less than 95% of the time.
110.	Supportability (Extensibility)	System shall be able to allow the creation of new type of resources not available at the time of implementation. i.e. Microchip Books
111.	Supportability (Modifiability)	System should be able to be modified to correct unexpected functionality and to include new requirements that do not force a change of system architecture.
112.	Supportability (Portability)	System shall be able to run on any OS and hardware that supports JAVA and Java Swing functionality.
113.	Supportability (Traceability of Requirements)	System requirements shall be able to be traced to make sure functional requirements are satisfied.

Appendix 2

Revised set of Functional Requirements:

#	Functional Requirement	Priority
	Catalog	
R1.	A staff member shall be able to add/remove resources	High
R1.1.	A resource cannot be removed if it is currently checked out or on reference	High
R1.2.	A resource cannot be added if it is a duplicate of a resource already present	High
R2.	A staff member must be able to view/change resource information	High
R2.1.	A resource's information can be edited by only one staff member at a time	High
R3.	A staff member must be able to add/remove resource copies	High
R3.1.	A resource cannot be removed if any of its copies are checked out	High
R3.2.	A resource cannot be removed if any of its copies are on reference	High
R4.	A staff member must be able to add and remove resource copies from Reference	High
R4.1.	A resource copy must be placed on reference under a faculty member's name or a staff member's name	High
R5.	A staff member must be able to add/remove/renew subscriptions	High
R5.1.	No magazine subscriptions may be duplicates of each other	High
R6.	A user must be able to search for resources using a search criteria	High
R6.1.	User must be able to see if the resource is available or on reference	High
R7.	A staff member must be able to view a resource subscription	High
R7.1.	The staff member must be able to see when the subscription was last renewed	High
R7.2.	The staff member must be able to see when the subscription expires	High
R7.3.	The staff member must be able to see when the subscription was first bought	Med.
R8.	A staff member shall be able to view/change resource copy information	High
	Circulation	
R9.	A staff member shall be able to view/change fines from patrons	High
R9.1.	A patron's information can be edited to remove fines from account	High
R9.2.	A payment will be marked as paid	High
R9.3.	A record of fines and paid fines will be kept	High
R10.	A staff member shall be able to view currently checked out resources of a user	High
R10.1.	A history will be kept for resources checked out	High
R11.	A staff member should be able to see the history of checked out resources of any user.	Med.
R11.1.	History will be given for a minimum of last 5 actions	Med.
R11.2.	Actions include checked out material, fines, and notifications	Med.
R12.	A staff member shall be able to renew/extend due date of checked out material not requested by another patron.	High
R12.1.	Resource will be marked if requested by another patron	High
R12.2.	It is not possible to perform function of renew if its requested	High
R13.	A staff member shall be able to issue/check-out material for patrons	High

R13.1.	Material shall not be on reference or reserved for another patron	High
R14.	A patron shall not be able to exceed a limit of checked-out resources	High
R14.1.	Limit will be checked upon request for checkout	High
R14.2.	No more than 30 resources may be checked	High
R15.	Only available resources can be checked-out	High
R15.1.	Reference material will not have option to check out	High
R16.	Reference material cannot be checked-out (or removed from library).	High
R16.1.	Referenced material will be marked on the resource information	High
R17.	Patrons cannot check-out more than one copy of same resource	High
R17.1.	Checkout will reject loan of two copies of same resource	High
R18.	A staff member shall be able to check-in returned material	High
R18.1.	Staff member can check-in material with or without a patron	High
R19.	Staff member shall be able to put a resource on reserve for another user	High
R19.1.	User could be patron, faculty, or another staff member	High
R20.	A patron shall be able to reserve material already checked-out	High
R20.1.	Patron will be notified once material has been checked-in	High
R20.2.	Patrons reserve on a resource expires after 2 weeks	High
R21.	Patron shall be able to check his reservation records and his position in the queue for materials that the patron currently has on reserve	High
R21.1.	Any patron shall be able to recall list of items put on hold.	High
R21.2.	System should be able to display reservation details for that patron.	High
R22.	System shall able to update the reserve queues once resources are returned	High
R22.1.	Once a book is checked in, system should be able to check for reservations	High
R22.2.	If a hold is found, system should be able update the queue for that book	High
R23.	Librarian shall be able to cancel reservations	High
R23.1.	Librarian shall be able to see patron’s reservation records	High
R23.2.	Librarian should be able to select items to cancel	High
R23.3.	System shall ask librarian to proceed or to cancel removal	High
R24.	A Patron shall be able to view a list of their own checked-out resources	High
R25.	A Patron shall be able to view a summary of their current outstanding fines	High
R25.1.	Patron shall be able to view what money is owed in fines	Med.
R26.	A staff member shall be able to view patrons’ reserved resources	High
R27.	Patrons shall be able to cancel their own reservations	High
R27.1.	Patrons shall be able to list reservations	High
R27.2.	Patron shall be able to select reservations to cancel	High
R27.3.	Patron shall be able to cancel selected reservations	High
R28.	A staff member shall be able to put/remove a resource on reference for a faculty member	High
R28.1.	Librarians shall be able to open the reference master to add or remove reference items.	High
R28.2.	Librarian shall be able to identify the faculty member adding or removing item to/from the reference master.	High

R29.	A staff member shall be able view the list of reserved resources(from Resource Record)	High
R29.1.	A staff shall be able to a list of reserved items	High
R30.	A staff member shall be able to view resource history and resource copy history	High
R30.1.	A staff member shall be able to open history of any resource to see who checkout resource in the past.	High
R31.	System shall be able to remove an expired reservation	High
R31.1.	System shall be able to drop reservations not filled by 2 weeks after due date of return.	High
R32.	Patrons are fined $0.25 per day that resources are overdue, to a maximum of $5.00 per overdue item	High
R32.1.	Staff members do not pay fines	High
R32.2.	Fines are no longer increased when a resource is returned	High
R33.	Student can check out a book for 4 weeks	High
R34.	Faculty member may check out a book for 3 months	High
R35.	Staff member may check out a book for 1 year	High
R36.	Resources other than books may be checked out for 1 week at a time, regardless of the user class	High
User Management		
R37.	Administrator shall be able to change a user class	High
R37.1.	An administrator cannot change their own user class	High
R37.2.	An Administrator can change the user class of another Administrator	High
R38.	A staff member shall able to add and disable students and faculty members from the system	High
R38.1.	A warning message is given if the account has unpaid fines or resources check out	High
R38.2.	A staff member shall be able to view student/faculty information	High
R39.	A staff member shall be able to view/change patrons' information (name, ID number, etc.)	High
R40.	A staff member shall be able to disable an account that currently has no outstanding fines	High
R40.1.	A staff member shall be able to view what resources and outstanding fines the user has	High
R40.2.	The account shall not be the staff member himself who is deleting the account	High
R41.	An administrator shall be able to import a list of new patrons to the database	High
R41.1.	A patron shall not already be in the system	High
Reporting		
R42.	System shall be able to send email reminders of materials that is nearly due and overdue	High
R42.1.	The system shall keep up to date records of which resources are overdue	High
R42.2.	The system shall be able to send emails to users	High
R43.	System shall be able to send an email message to a patron when a resource he/she has on reserve becomes available.	Med.
R43.1.	The system shall be able to detect when a resource on reserve is check in	Med.

R43.2.	The system shall be able to clearly notify the staff member when they have checked in a resource that has been reserved.	Med.
R44.	A staff member shall be able to view resources that are currently checked out and overdue.	High
R44.1.	The system shall keep an up to date record of currently checked out and overdue resources.	High
R44.2.	The system shall allow staff members to view checked out resources and overdue resources.	High
R45.	A staff member shall be able to view all outstanding fines for all users	Med.
R45.1.	The system shall keep up to date records of fines for a user and for which loan the fines were incurred	Med.
R46.	System shall be able to create TO-DO to renew a subscription	Low
R46.1.	The system shall be able to determine if a subscription needs renewing	Low
	Browsing & Guest Access	
R47.	User shall sign in to a terminal using a username and password (a terminal is a computer running the client software)	High
R47.1.	The system shall be able to verify a user's username and password	High
R47.2.	The system shall be able to determine the access level of a user	High
R48.	Patrons shall be able to search catalog for resources	High
R48.1.	The system shall maintain up to date records of all resources.	High
R48.2.	The system shall be able to determine if a resource is available.	High
R49.	By default, if no one is logged in, the client is in a "guest" mode	High
R49.1.	The system shall support a public interface where anyone can search for resources	High
R50.	In guest mode, the only commands available are: "search for resource", "login"	High
	Application Management	
R51.	A staff member shall be able to start/shutdown the server program	High
R51.1.	Any authorized staff member shall be able start/shutdown the server from their computer	High
R51.2.	A staff member issues "shutdown server" command from the client computer	High
R51.3.	Staff member clicks start the server button, system issues "start server" action	High
R51.4.	Server system receives the "shutdown server" command and stops all running programs by killing all processes	High
R51.5.	Server system receives command and executes server program	High
R52.	A staff member shall be able to start/shutdown the client	High
R52.1.	Client program can be started by clicking on start	High
R52.2.	A staff member can stop the client by closing the client program	High
R52.3.	Client program start and connects to server program which hosted on remote machine	High
R52.4.	Starting the client shall be restricted to authorized users listed on a management access control list	High
R52.5.	Client disconnects all connection with the server	High
R52.6.	Close client shall be restricted to authorized users listed on a management access control list	High

R53.	Any user shall be able to sign into the client and sign out	High
R53.1.	User will be prompted to enter her/his username, password and click “enter”	High
R53.2.	The system shall authenticated user by checking correctness of the login and looking up user’s table from database	High
R53.3.	The system shall automatically determine user role and credentials in the system	High
R53.4.	The system shall open correct user mode according user’s role and credentials	High
R53.5.	The system can be sign out by clicking sign out button	High
R54.	System shall be able to create TO-DO for expired reference resources	Low
R55.	At the end of the day a staff member shall be able to create/restore a backup of database	Med.
R55.1.	To create a backup of database, user authorized and listed on a management access control list	Med.
R55.2.	A staff member clicks “backup database” button, system prompts for user’s password	Med.
R55.3.	The system checks user credentials, if access granted, the system starts backing up the database	Med.
R55.4.	A staff member, whose user credentials has been approved, can perform restore backup of database.	Med.
R55.5.	The system shall be able to restore backup of database, when user issues restore command	Med.

Appendix 3

Class Diagrams for Libris:

Figure A3.1: Package: client.userinterface.datatypes

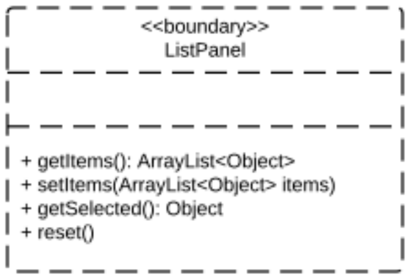

<<boundary>>
ListPanel
+ getItems(): ArrayList<Object>
+ setItems(ArrayList<Object> items)
+ getSelected(): Object
+ reset()

Figure A3.2: Package: client.userinterface.mainwindow

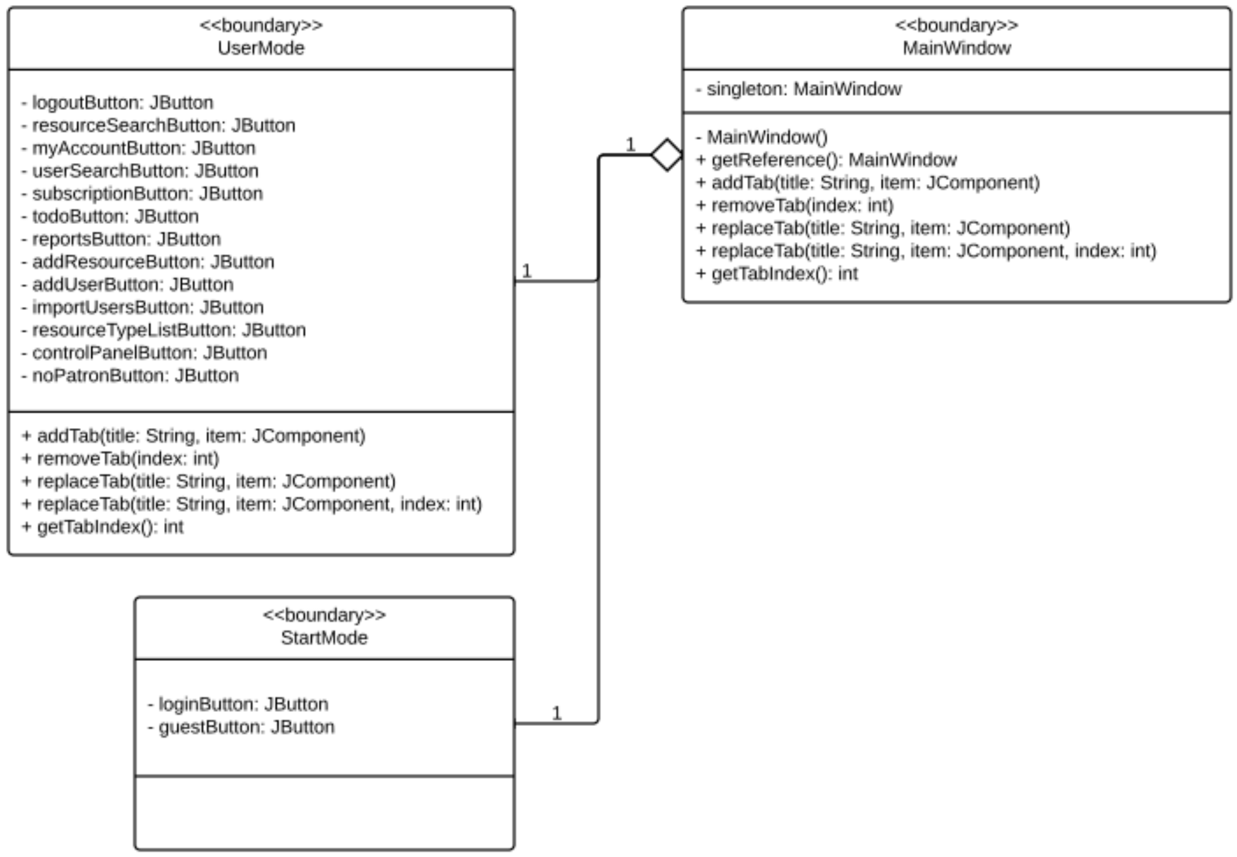

<<boundary>>
UserMode
- logoutButton: JButton
- resourceSearchButton: JButton
- myAccountButton: JButton
- userSearchButton: JButton
- subscriptionButton: JButton
- todoButton: JButton
- reportsButton: JButton
- addResourceButton: JButton
- addUserButton: JButton
- importUsersButton: JButton
- resourceTypeListButton: JButton
- controlPanelButton: JButton
- noPatronButton: JButton
+ addTab(title: String, item: JComponent)
+ removeTab(index: int)
+ replaceTab(title: String, item: JComponent)
+ replaceTab(title: String, item: JComponent, index: int)
+ getTabIndex(): int
<<boundary>>
MainWindow
- singleton: MainWindow
- MainWindow()
+ getReference(): MainWindow
+ addTab(title: String, item: JComponent)
+ removeTab(index: int)
+ replaceTab(title: String, item: JComponent)
+ replaceTab(title: String, item: JComponent, index: int)
+ getTabIndex(): int
<<boundary>>
StartMode
- loginButton: JButton
- guestButton: JButton
1
1
1

Figure A3.3: Package: client.userinterface.controlpanel

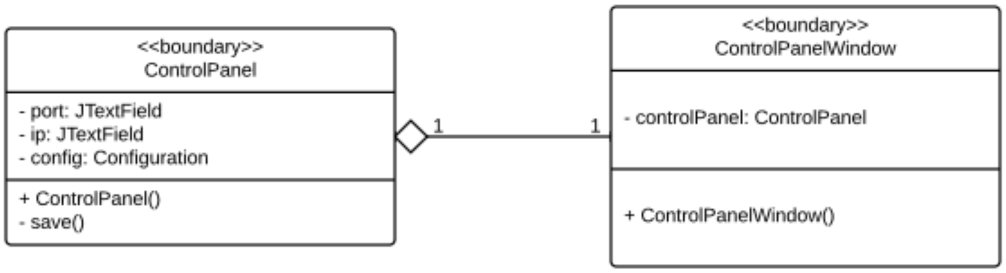


Figure A3.4: Package: client.userinterface.datatypes.report

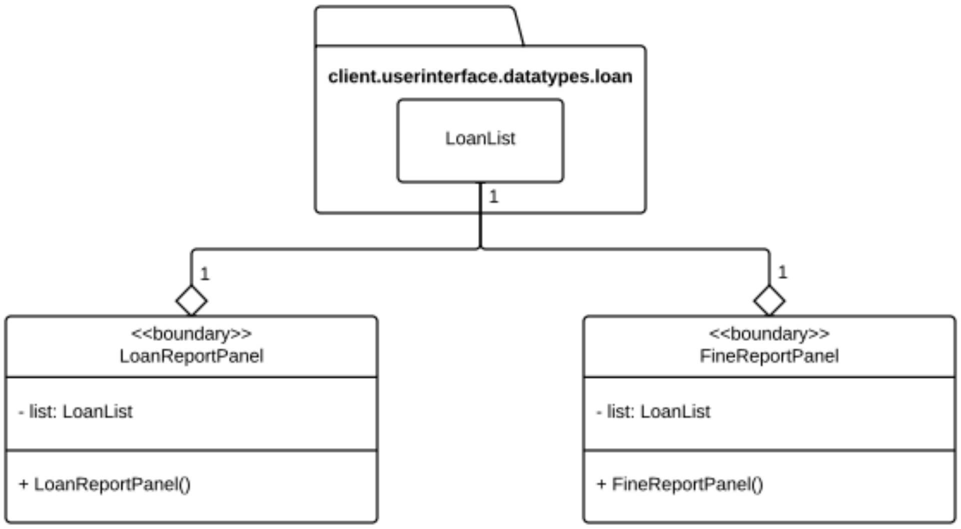


Figure A3.5: Package: client.userinterface.login

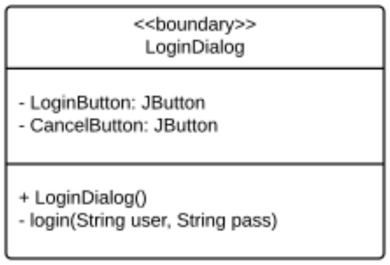


Figure A3.6: Package: client.userinterface.datatypes.user

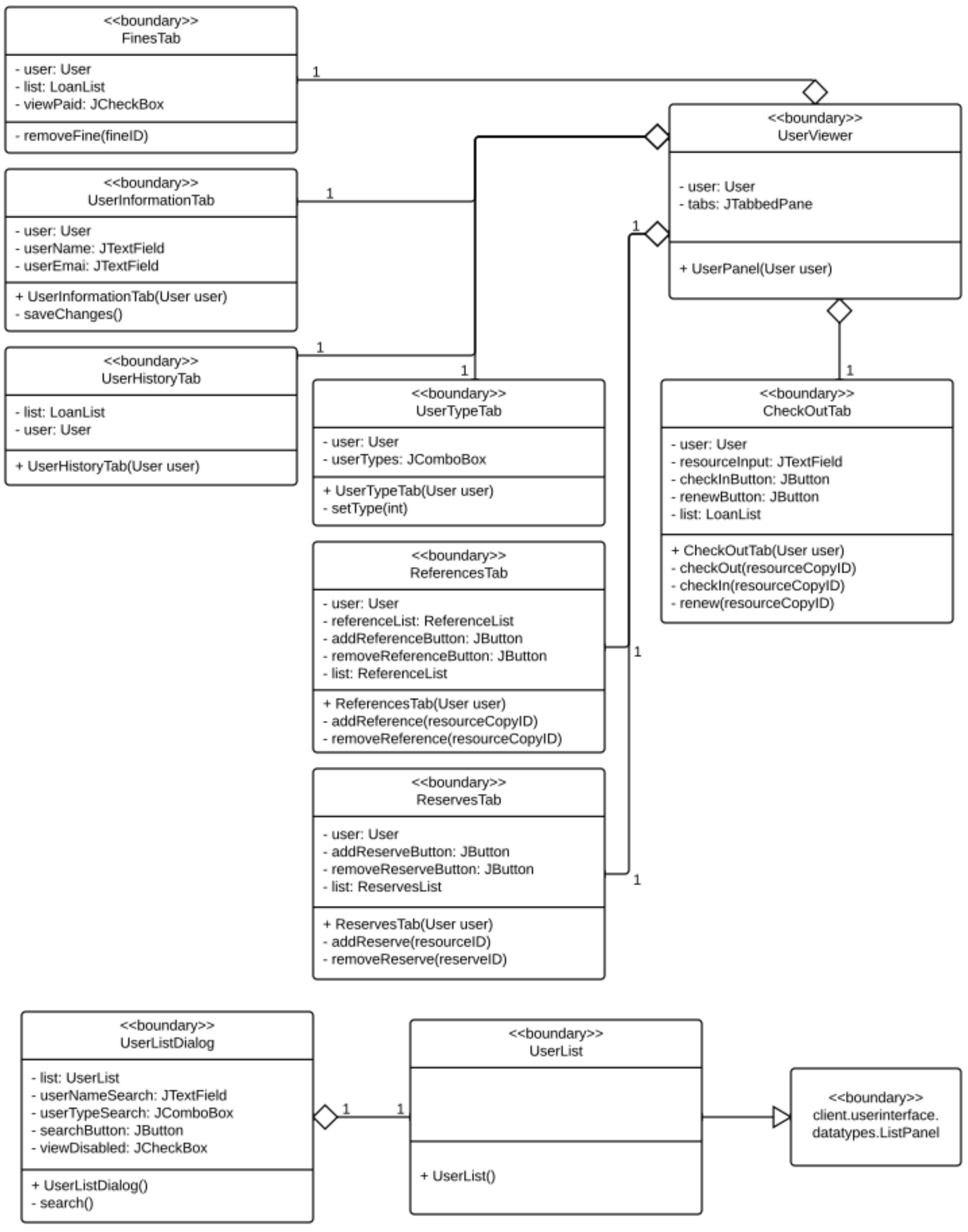


Figure A3.7: Package: client.userinterface.datatypes.resourcecopy

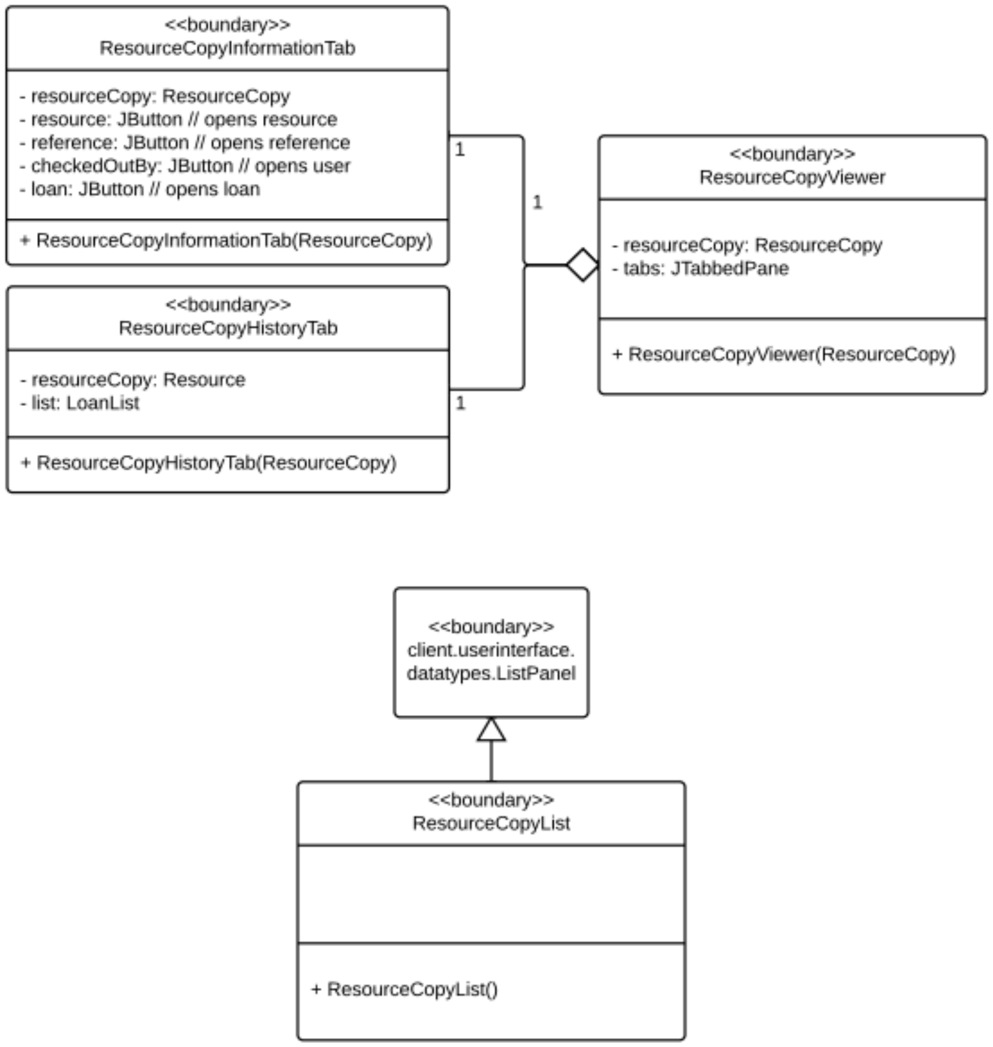

<<boundary>>
ResourceCopyInformationTab
- resourceCopy: ResourceCopy
- resource: JButton // opens resource
- reference: JButton // opens reference
- checkedOutBy: JButton // opens user
- loan: JButton // opens loan
+ ResourceCopyInformationTab(ResourceCopy)
<<boundary>>
ResourceCopyHistoryTab
- resourceCopy: Resource
- list: LoanList
+ ResourceCopyHistoryTab(ResourceCopy)
1
1
1
<<boundary>>
ResourceCopyViewer
- resourceCopy: ResourceCopy
- tabs: JTabbedPane
+ ResourceCopyViewer(ResourceCopy)
<<boundary>>
client.userinterface.
datatypes.ListPanel
<<boundary>>
ResourceCopyList
+ ResourceCopyList()

Figure A3.8: Package: client.userinterface.datatypes.loan

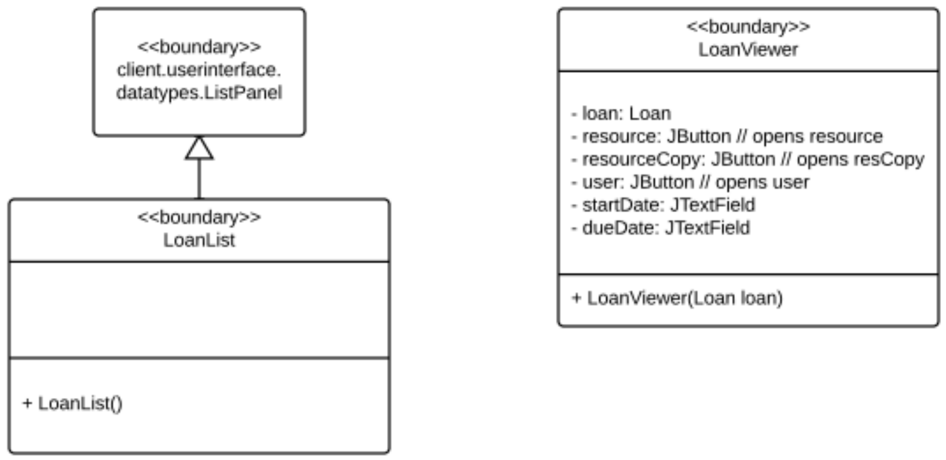


Figure A3.9: Package: client.userinterface.datatypes.reserve

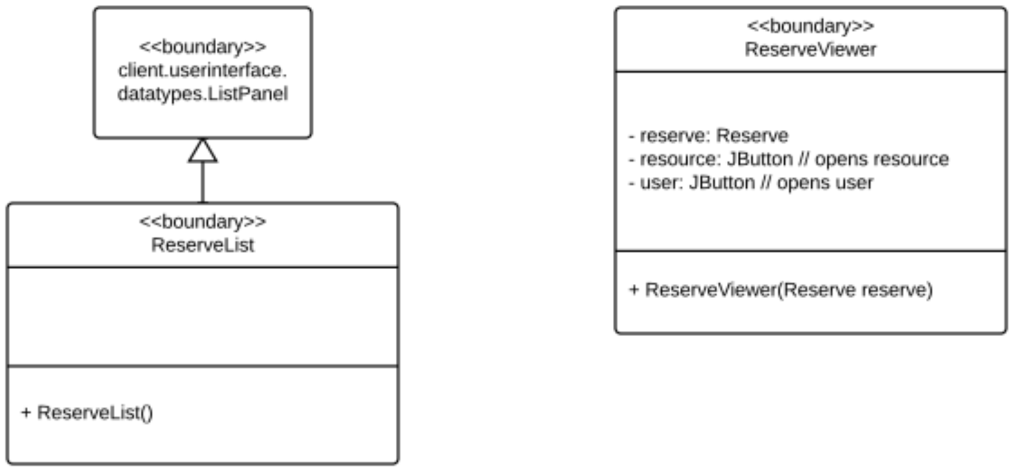


Figure A3.10: Package: client.userinterface.datatypes.reference

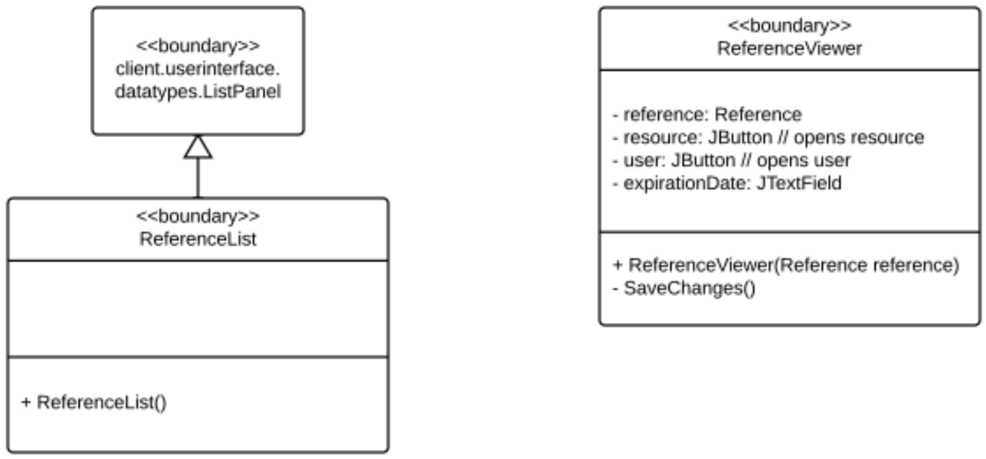


Figure A3.11: Package: client.userinterface.datatypes.nopatron

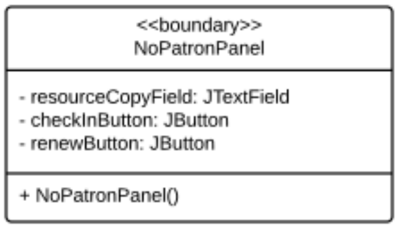


Figure A3.12: Package: client.userinterface.datatypes.resource

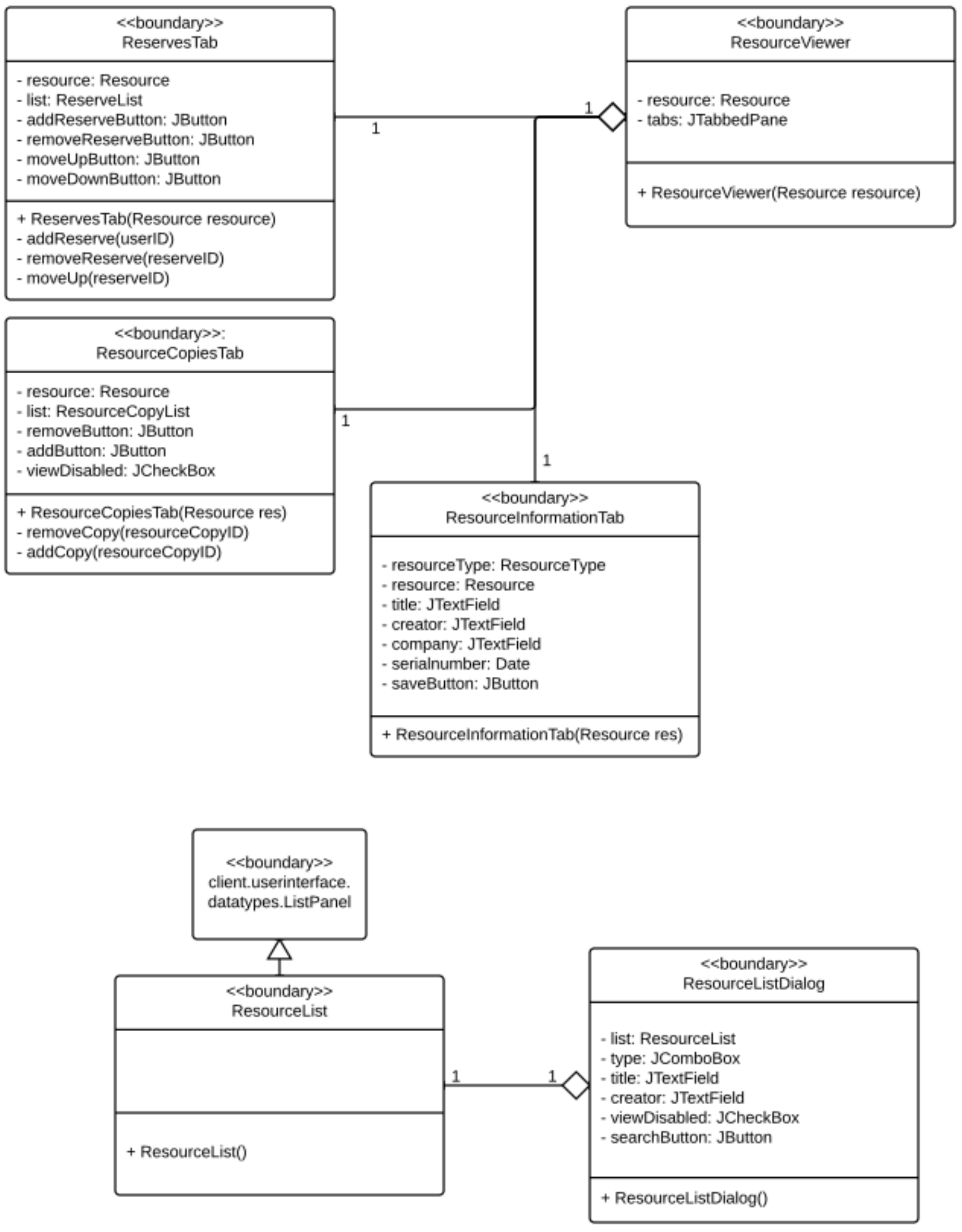


Figure A3.13 Package: client.userinterface.datatypes.subscription

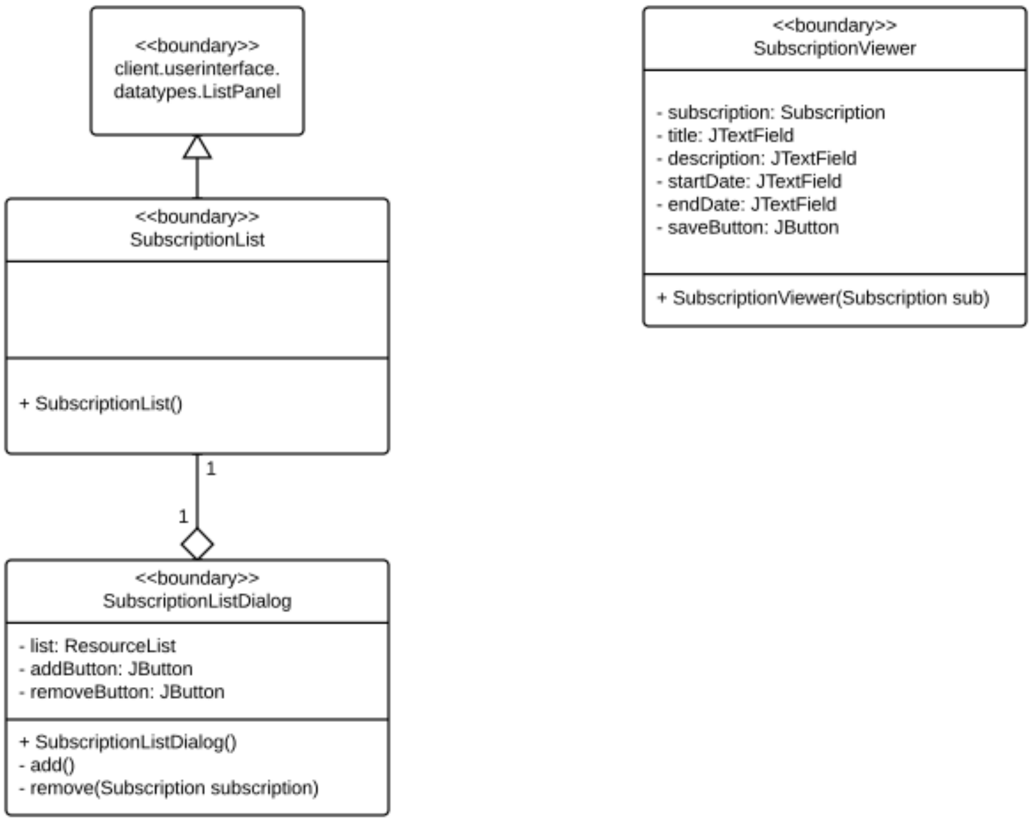


Figure A3.14: Package: client.userinterface.datatypes.todo

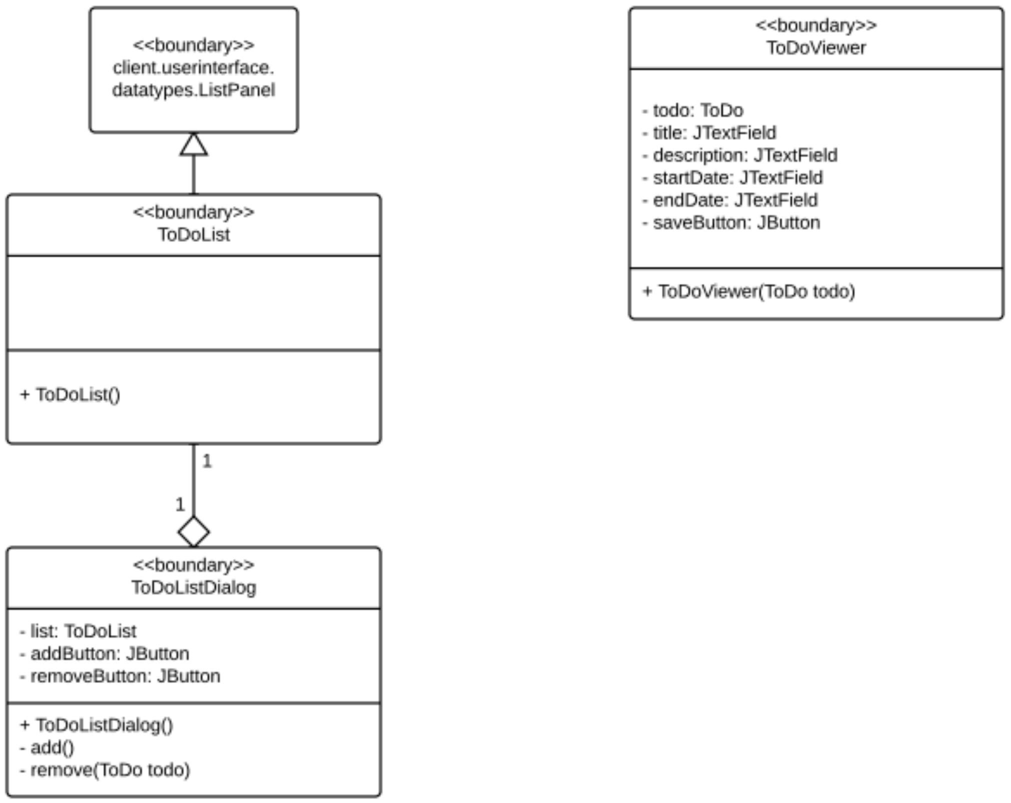


Figure A3.15: Package: client.userinterface.datatypes.resourcetype

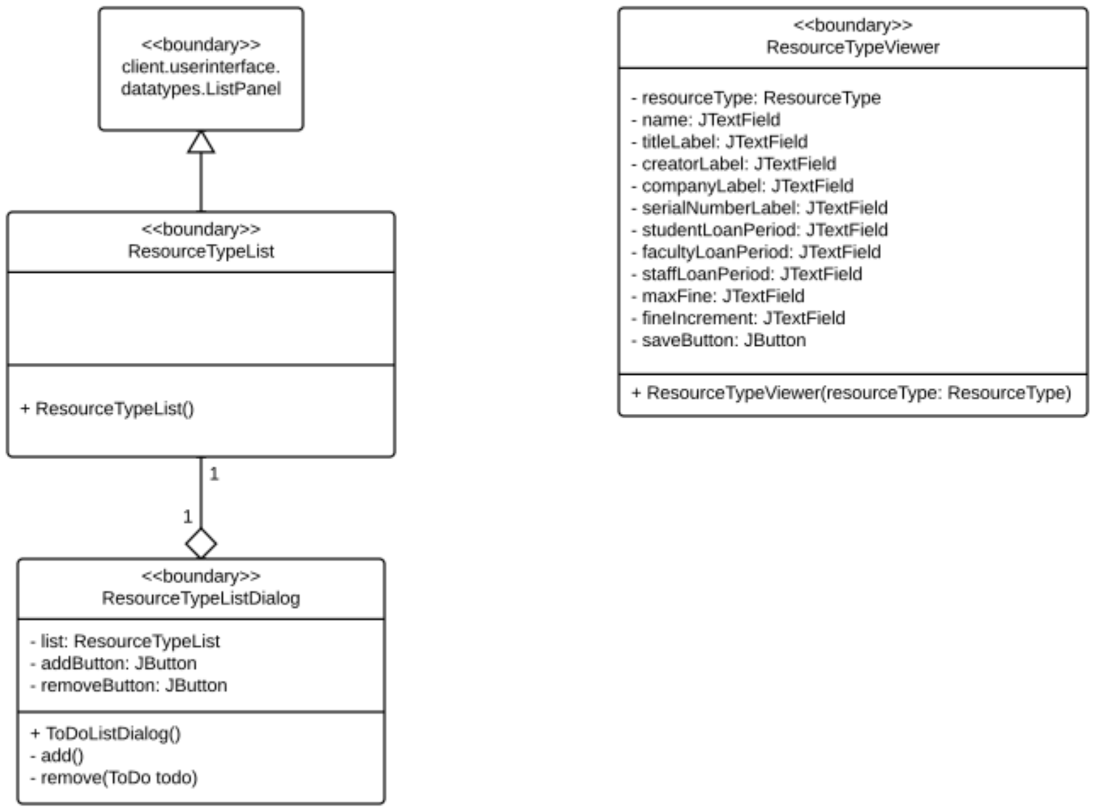


Figure A3.16: Package: server.clientanduserinterface.userinterface

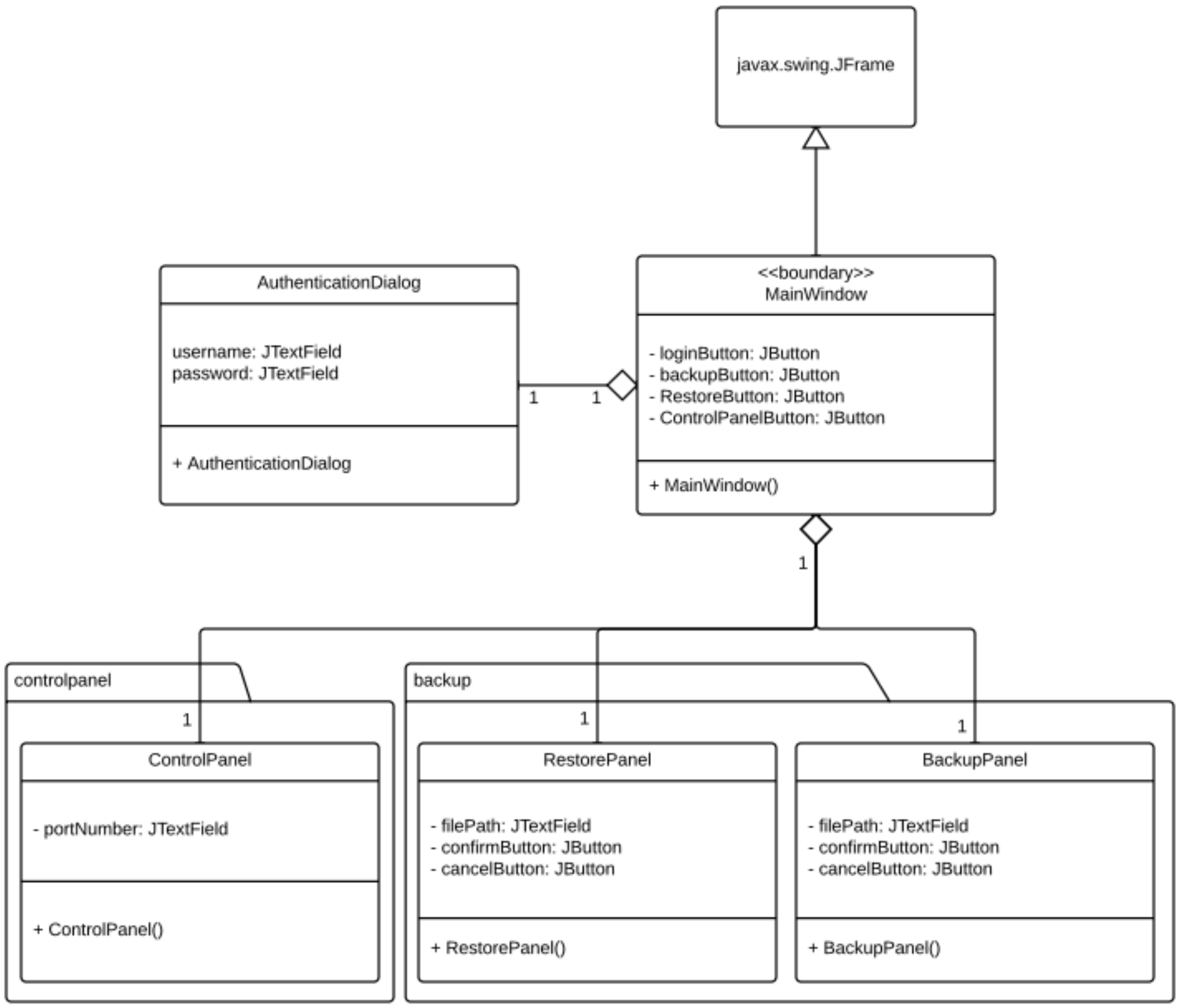


Figure A3.17

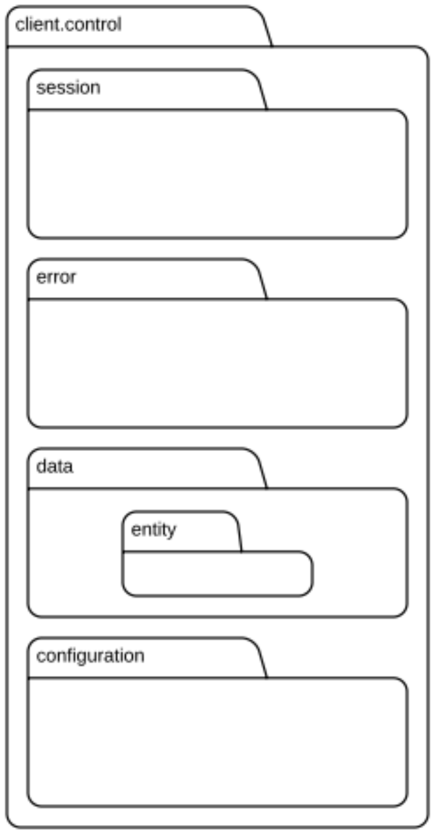


Figure A3.18

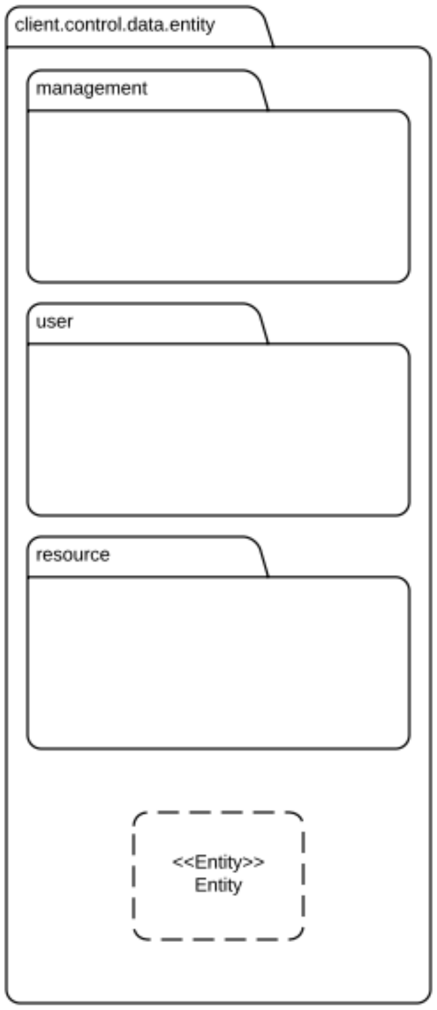

client.control.data.entity
management
user
resource
<<Entity>>
Entity

Figure A3.19
Package: client.control.data.entity.

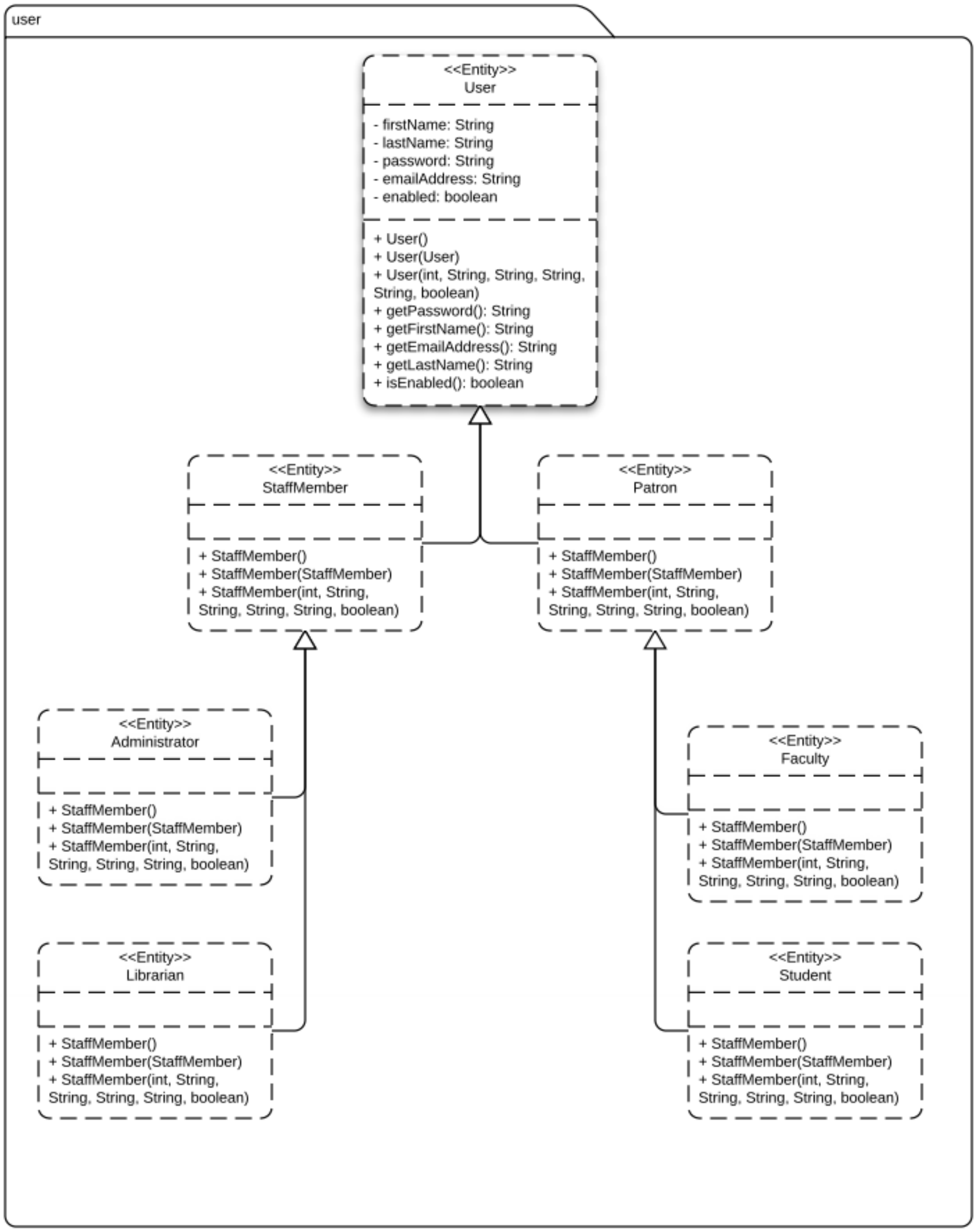

user
<<Entity>>
User
- firstName: String
- lastName: String
- password: String
- emailAddress: String
- enabled: boolean
+ User()
+ User(User)
+ User(int, String, String, String, String, boolean)
+ getPassword(): String
+ getFirstName(): String
+ getEmailAddress(): String
+ getLastName(): String
+ isEnabled(): boolean
<<Entity>>
StaffMember
+ StaffMember()
+ StaffMember(StaffMember)
+ StaffMember(int, String, String, String, String, boolean)
<<Entity>>
Patron
+ StaffMember()
+ StaffMember(StaffMember)
+ StaffMember(int, String, String, String, String, boolean)
<<Entity>>
Administrator
+ StaffMember()
+ StaffMember(StaffMember)
+ StaffMember(int, String, String, String, String, boolean)
<<Entity>>
Faculty
+ StaffMember()
+ StaffMember(StaffMember)
+ StaffMember(int, String, String, String, String, boolean)
<<Entity>>
Librarian
+ StaffMember()
+ StaffMember(StaffMember)
+ StaffMember(int, String, String, String, String, boolean)
<<Entity>>
Student
+ StaffMember()
+ StaffMember(StaffMember)
+ StaffMember(int, String, String, String, String, boolean)

Figure A3.20
Package: client.control.data.entity

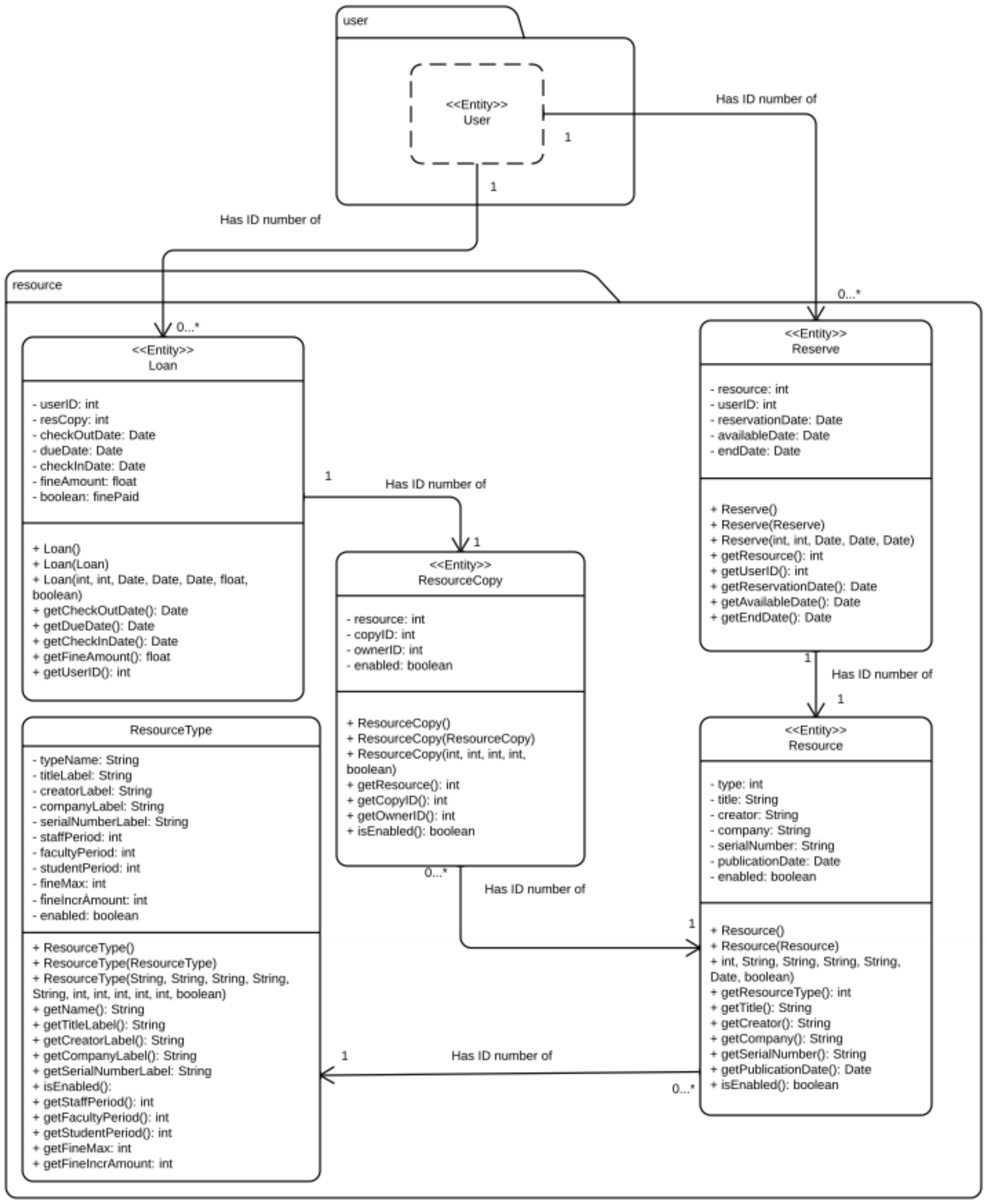


Figure A3.21
Package: client.control.data.entity

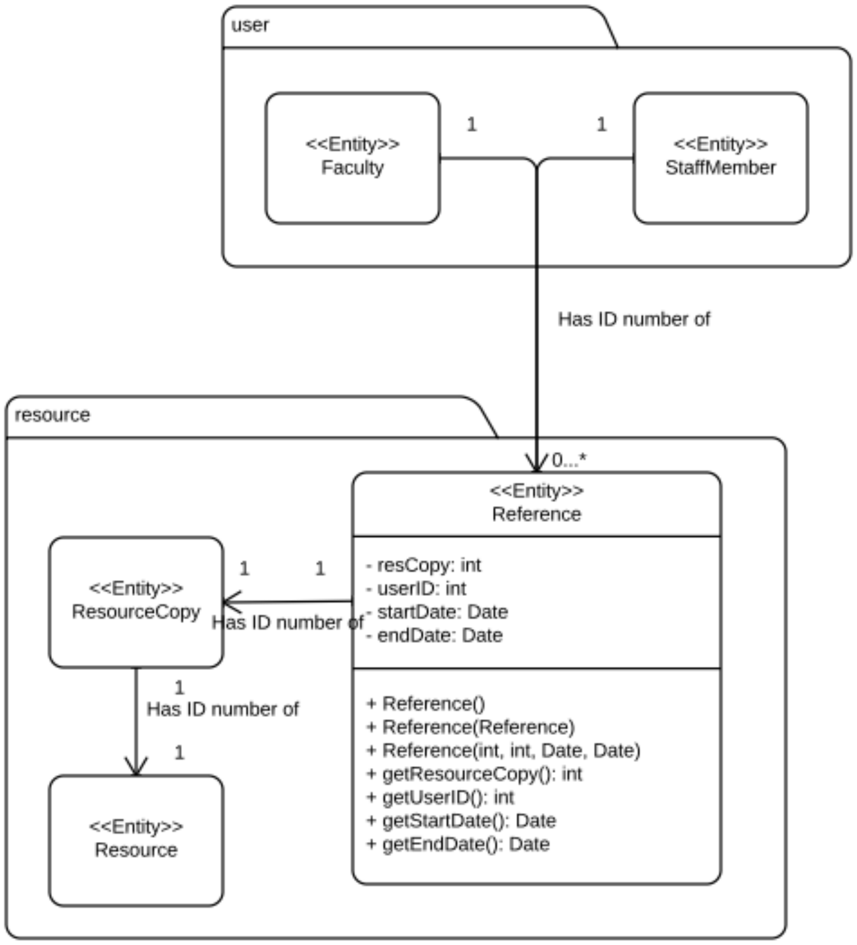


Figure A3.22
Package: client.control.data.entity

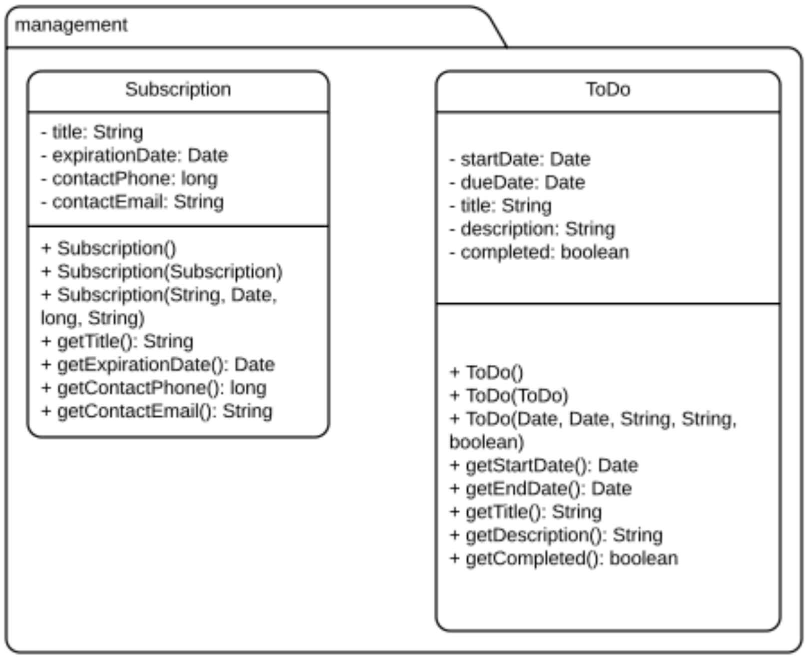


Figure A3.23

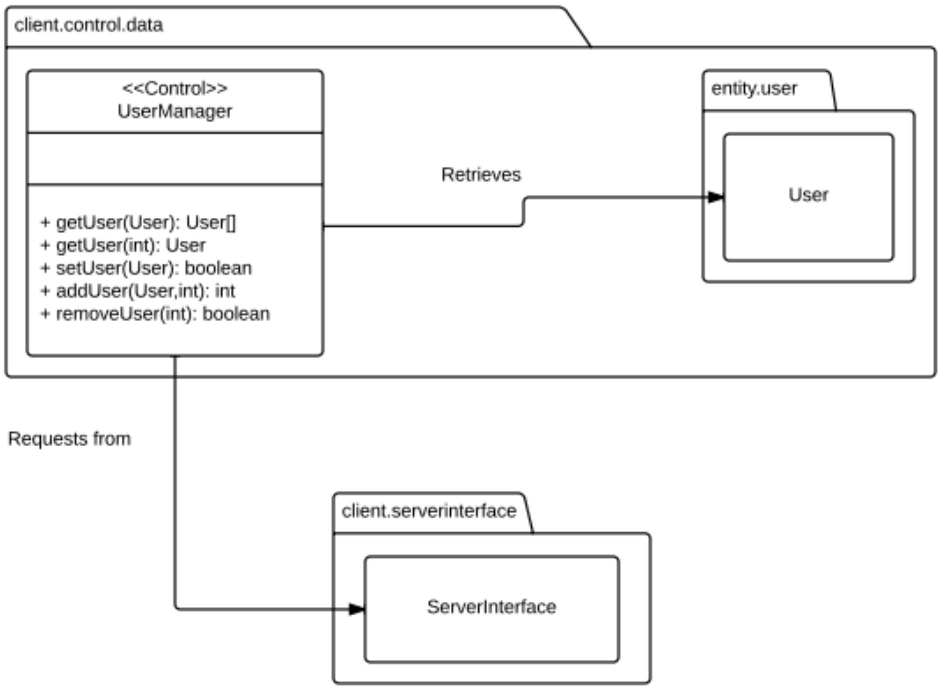


Figure A3.24

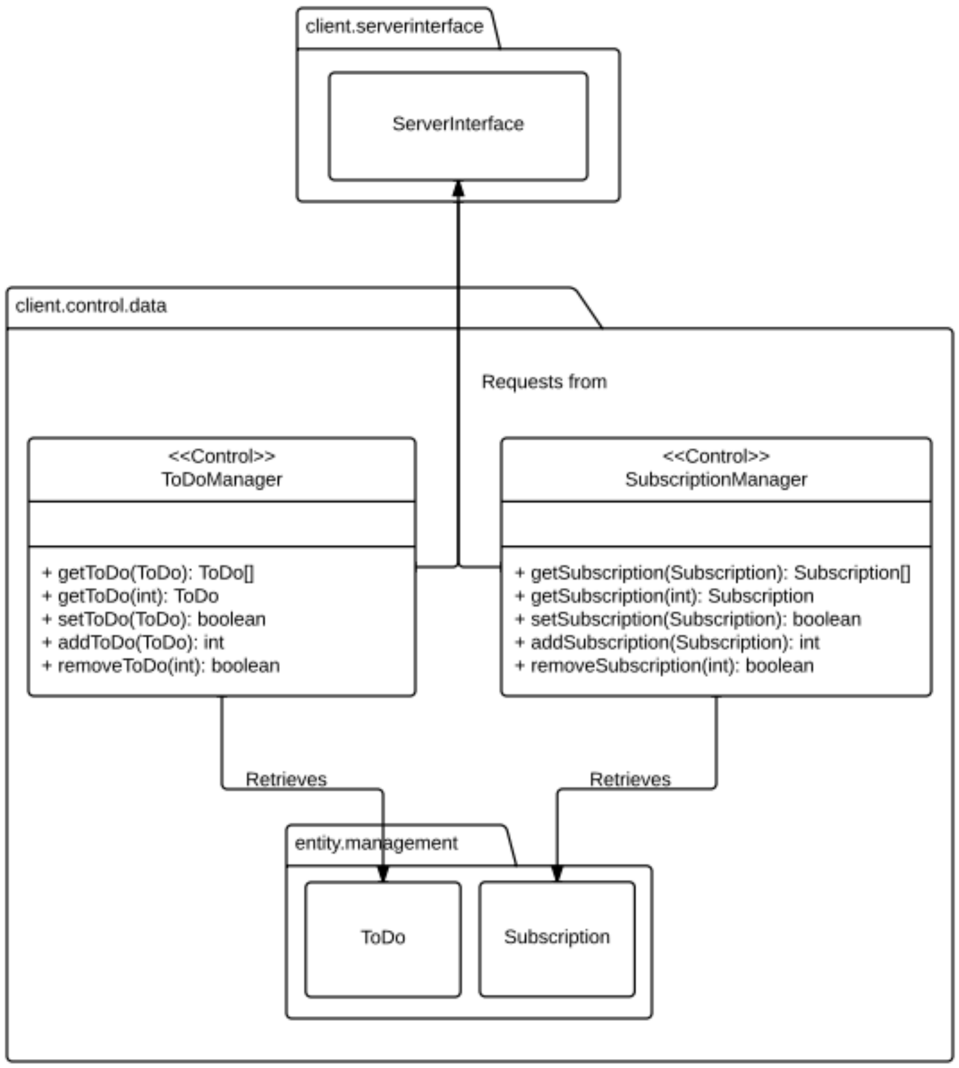

client.serverinterface
ServerInterface
client.control.data
Requests from
<<Control>>
ToDoManager
+ getToDo(ToDo): ToDo[]
+ getToDo(int): ToDo
+ setToDo(ToDo): boolean
+ addToDo(ToDo): int
+ removeToDo(int): boolean
<<Control>>
SubscriptionManager
+ getSubscription(Subscription): Subscription[]
+ getSubscription(int): Subscription
+ setSubscription(Subscription): boolean
+ addSubscription(Subscription): int
+ removeSubscription(int): boolean
Retrieves
Retrieves
entity.management
ToDo
Subscription

Figure A3.25

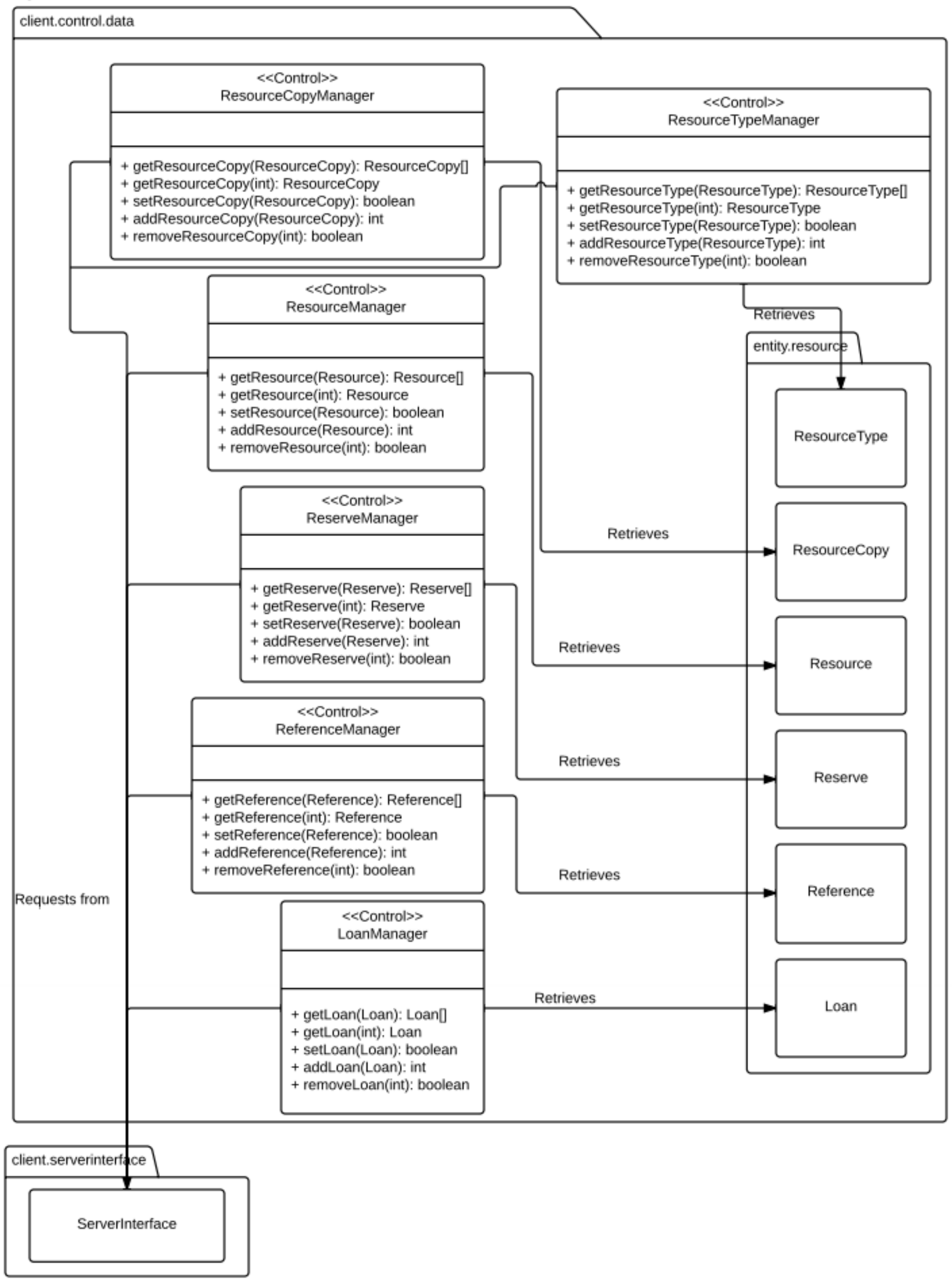

client.control.data
<<Control>>
ResourceCopyManager
+ getResourceCopy(ResourceCopy): ResourceCopy[]
+ getResourceCopy(int): ResourceCopy
+ setResourceCopy(ResourceCopy): boolean
+ addResourceCopy(ResourceCopy): int
+ removeResourceCopy(int): boolean
<<Control>>
ResourceTypeManager
+ getResourceType(ResourceType): ResourceType[]
+ getResourceType(int): ResourceType
+ setResourceType(ResourceType): boolean
+ addResourceType(ResourceType): int
+ removeResourceType(int): boolean
<<Control>>
ResourceManager
+ getResource(Resource): Resource[]
+ getResource(int): Resource
+ setResource(Resource): boolean
+ addResource(Resource): int
+ removeResource(int): boolean
<<Control>>
ReserveManager
+ getReserve(Reserve): Reserve[]
+ getReserve(int): Reserve
+ setReserve(Reserve): boolean
+ addReserve(Reserve): int
+ removeReserve(int): boolean
<<Control>>
ReferenceManager
+ getReference(Reference): Reference[]
+ getReference(int): Reference
+ setReference(Reference): boolean
+ addReference(Reference): int
+ removeReference(int): boolean
<<Control>>
LoanManager
+ getLoan(Loan): Loan[]
+ getLoan(int): Loan
+ setLoan(Loan): boolean
+ addLoan(Loan): int
+ removeLoan(int): boolean
Retrieves
entity.resource
ResourceType
ResourceCopy
Resource
Reserve
Reference
Loan
Requests from
client.serverinterface
ServerInterface

Figure A3.26

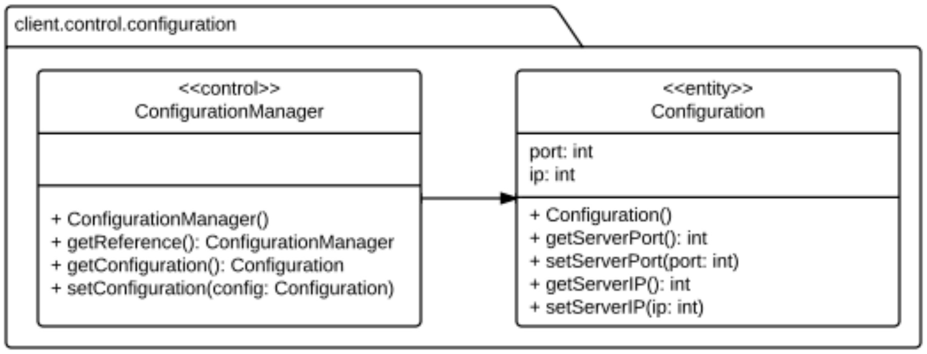


Figure A3.27

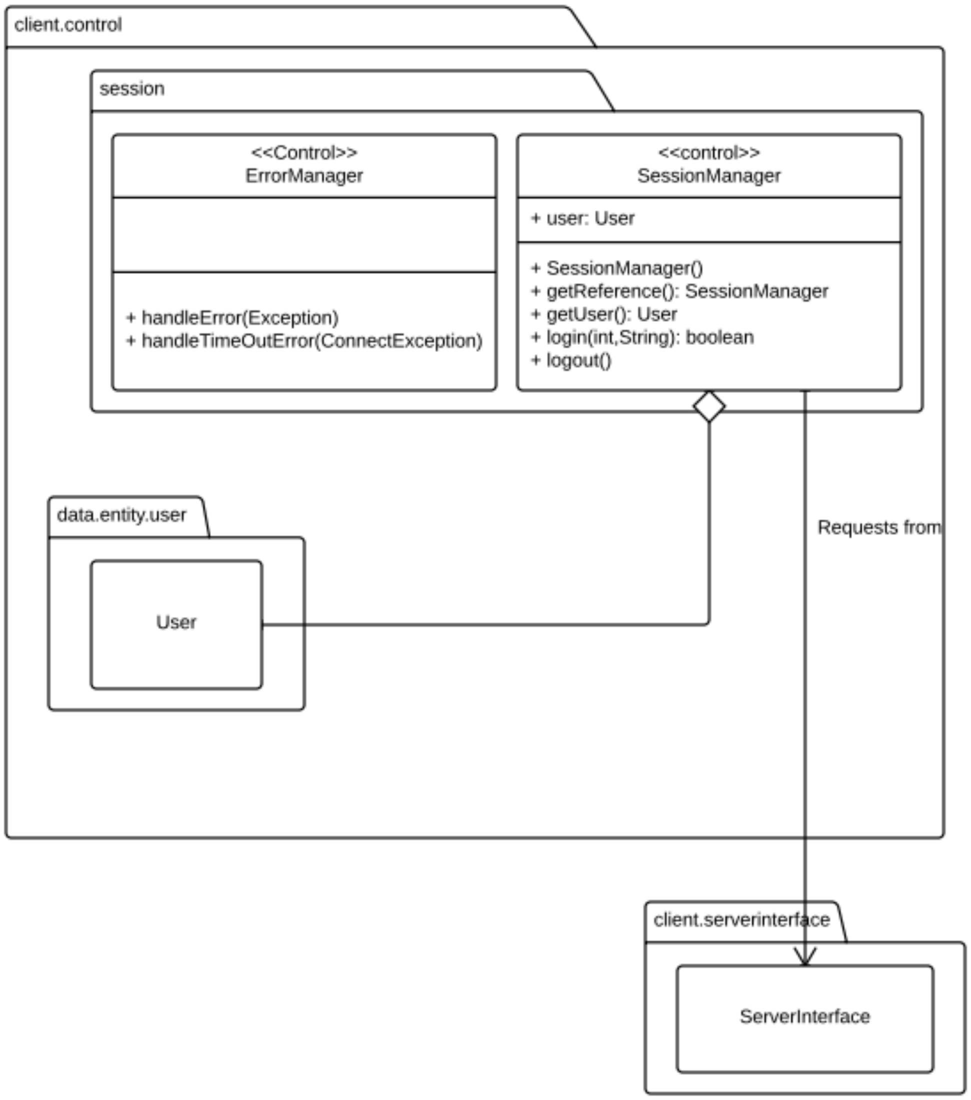


Figure A3.28

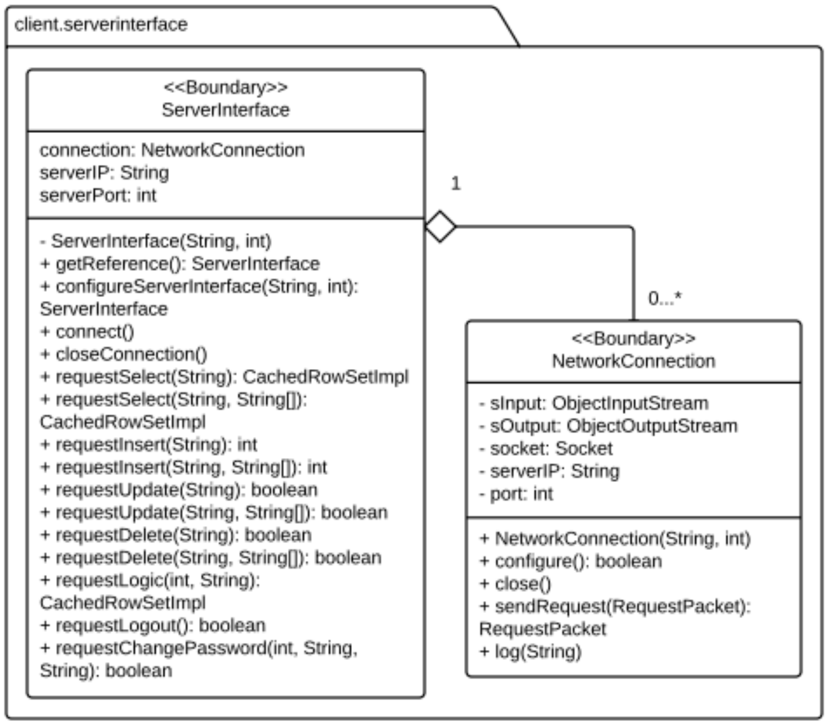

client.serverinterface
<<Boundary>>
ServerInterface
connection: NetworkConnection
serverIP: String
serverPort: int
- ServerInterface(String, int)
+ getReference(): ServerInterface
+ configureServerInterface(String, int): ServerInterface
+ connect()
+ closeConnection()
+ requestSelect(String): CachedRowSetImpl
+ requestSelect(String, String[]): CachedRowSetImpl
+ requestInsert(String): int
+ requestInsert(String, String[]): int
+ requestUpdate(String): boolean
+ requestUpdate(String, String[]): boolean
+ requestDelete(String): boolean
+ requestDelete(String, String[]): boolean
+ requestLogic(int, String): CachedRowSetImpl
+ requestLogout(): boolean
+ requestChangePassword(int, String, String): boolean
1
0...*
<<Boundary>>
NetworkConnection
- sInput: ObjectInputStream
- sOutput: ObjectOutputStream
- socket: Socket
- serverIP: String
- port: int
+ NetworkConnection(String, int)
+ configure(): boolean
+ close()
+ sendRequest(RequestPacket): RequestPacket
+ log(String)

Figure A3.29 Comminication Class Diagram

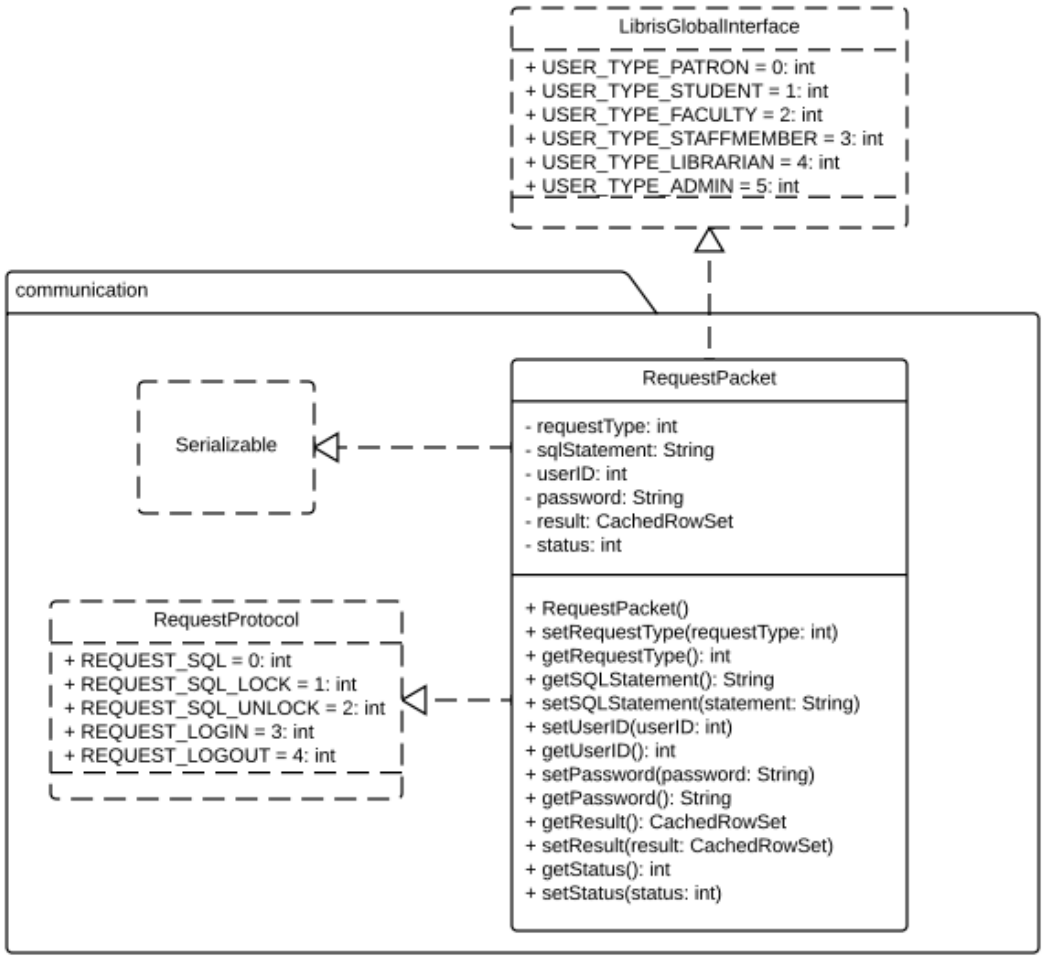

LibrisGlobalInterface
+ USER_TYPE_PATRON = 0: int
+ USER_TYPE_STUDENT = 1: int
+ USER_TYPE_FACULTY = 2: int
+ USER_TYPE_STAFFMEMBER = 3: int
+ USER_TYPE_LIBRARIAN = 4: int
+ USER_TYPE_ADMIN = 5: int
communication
Serializable
RequestPacket
- requestType: int
- sqlStatement: String
- userID: int
- password: String
- result: CachedRowSet
- status: int
+ RequestPacket()
+ setRequestType(requestType: int)
+ getRequestType(): int
+ getSQLStatement(): String
+ setSQLStatement(statement: String)
+ setUserID(userID: int)
+ getUserID(): int
+ setPassword(password: String)
+ getPassword(): String
+ getResult(): CachedRowSet
+ setResult(result: CachedRowSet)
+ getStatus(): int
+ setStatus(status: int)
RequestProtocol
+ REQUEST_SQL = 0: int
+ REQUEST_SQL_LOCK = 1: int
+ REQUEST_SQL_UNLOCK = 2: int
+ REQUEST_LOGIN = 3: int
+ REQUEST_LOGOUT = 4: int

Figure A3.30 Server Class Diagrams

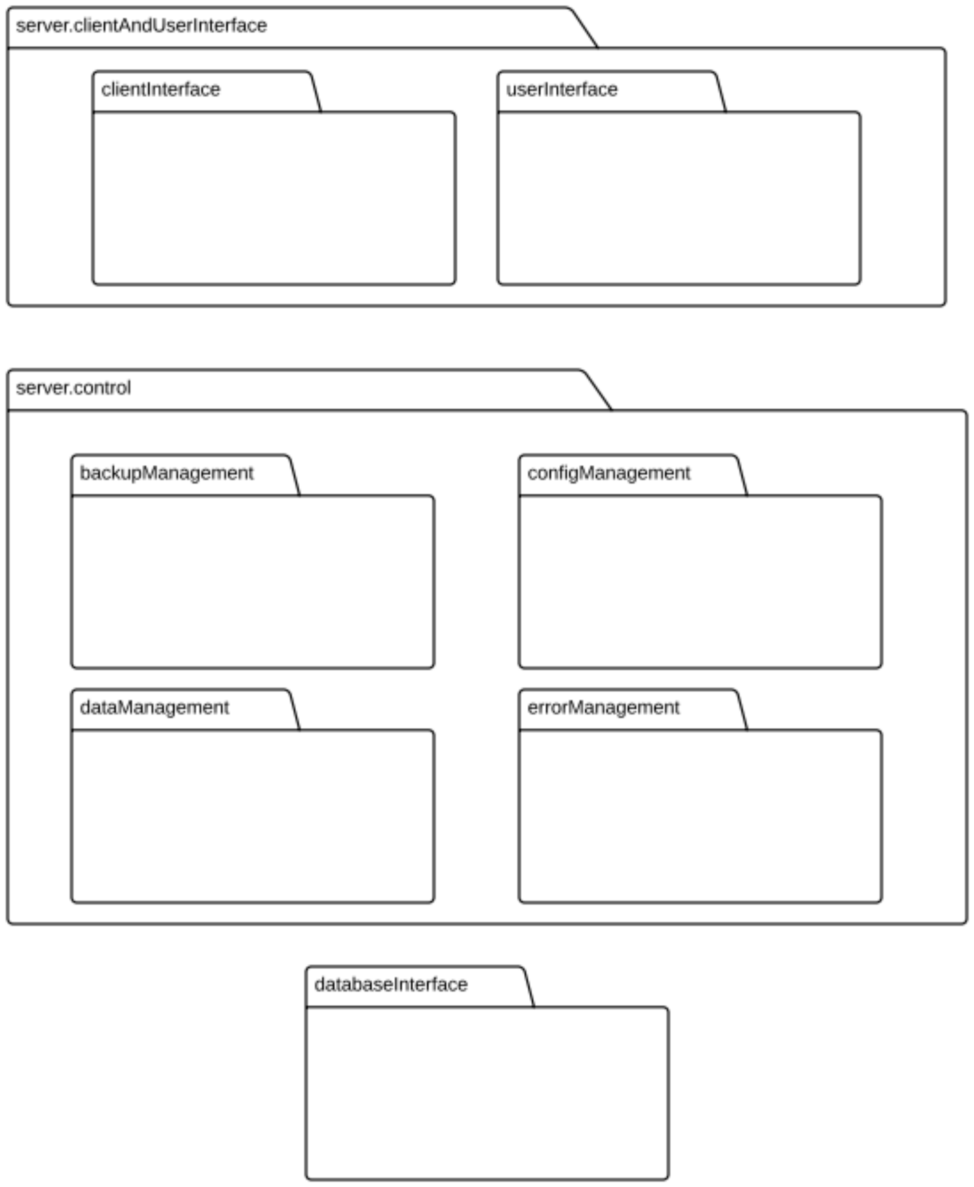


Figure A3.31 Server Class Diagrams (Cont.)

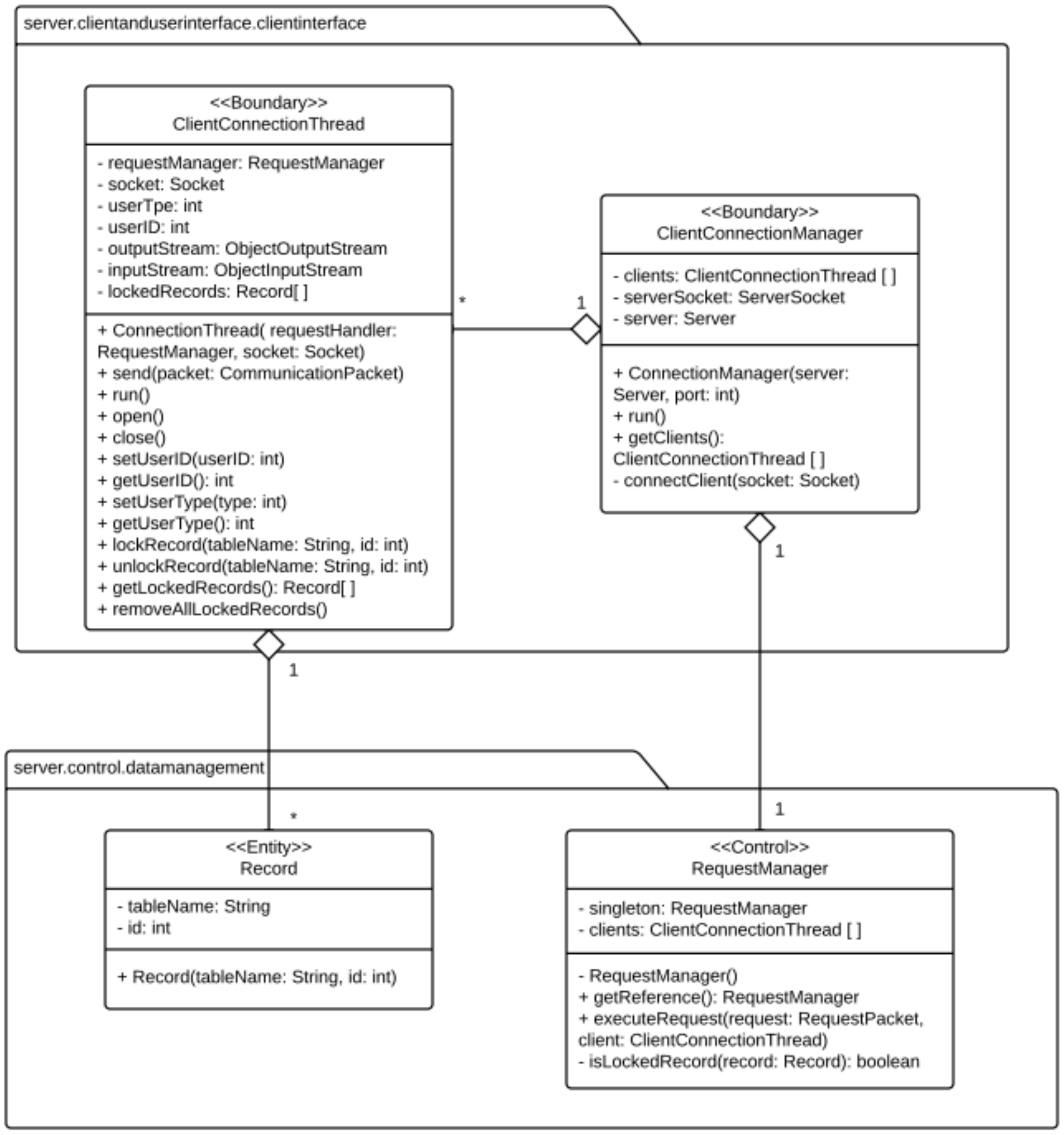

server.clientanduserinterface.clientinterface
<<Boundary>>
ClientConnectionThread
- requestManager: RequestManager
- socket: Socket
- userTpe: int
- userID: int
- outputStream: ObjectOutputStream
- inputStream: ObjectInputStream
- lockedRecords: Record[ ]
+ ConnectionThread(requestHandler: RequestManager, socket: Socket)
+ send(packet: CommunicationPacket)
+ run()
+ open()
+ close()
+ setUserID(userID: int)
+ getUserID(): int
+ setUserType(type: int)
+ getUserType(): int
+ lockRecord(tableName: String, id: int)
+ unlockRecord(tableName: String, id: int)
+ getLockedRecords(): Record[ ]
+ removeAllLockedRecords()
*
1
<<Boundary>>
ClientConnectionManager
- clients: ClientConnectionThread [ ]
- serverSocket: ServerSocket
- server: Server
+ ConnectionManager(server: Server, port: int)
+ run()
+ getClients(): ClientConnectionThread [ ]
- connectClient(socket: Socket)
1
1
server.control.datamanagement
*
1
<<Entity>>
Record
- tableName: String
- id: int
+ Record(tableName: String, id: int)
<<Control>>
RequestManager
- singleton: RequestManager
- clients: ClientConnectionThread [ ]
- RequestManager()
+ getReference(): RequestManager
+ executeRequest(request: RequestPacket, client: ClientConnectionThread)
- isLockedRecord(record: Record): boolean

Figure A3.32 Server Class Diagrams (Cont.)

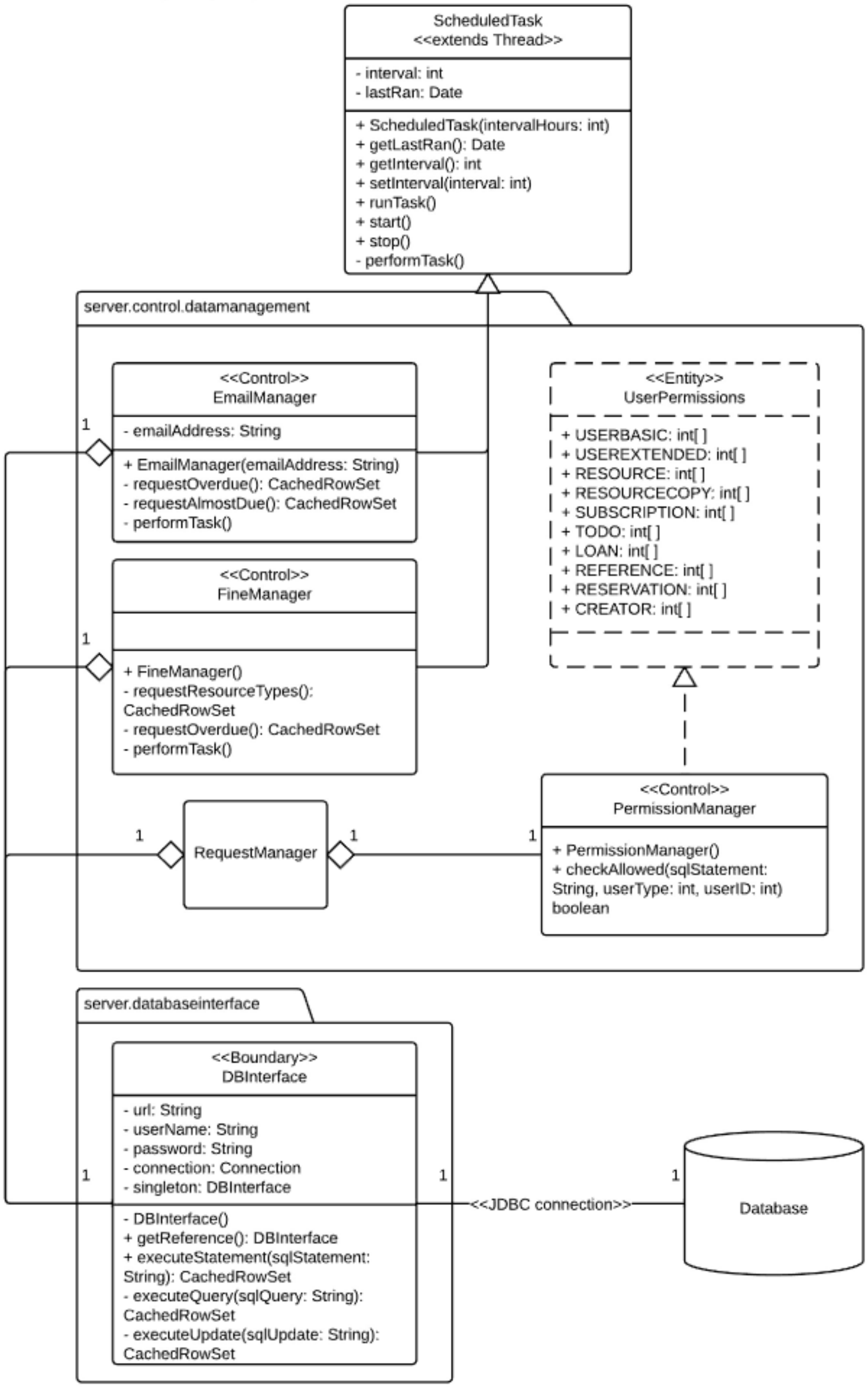
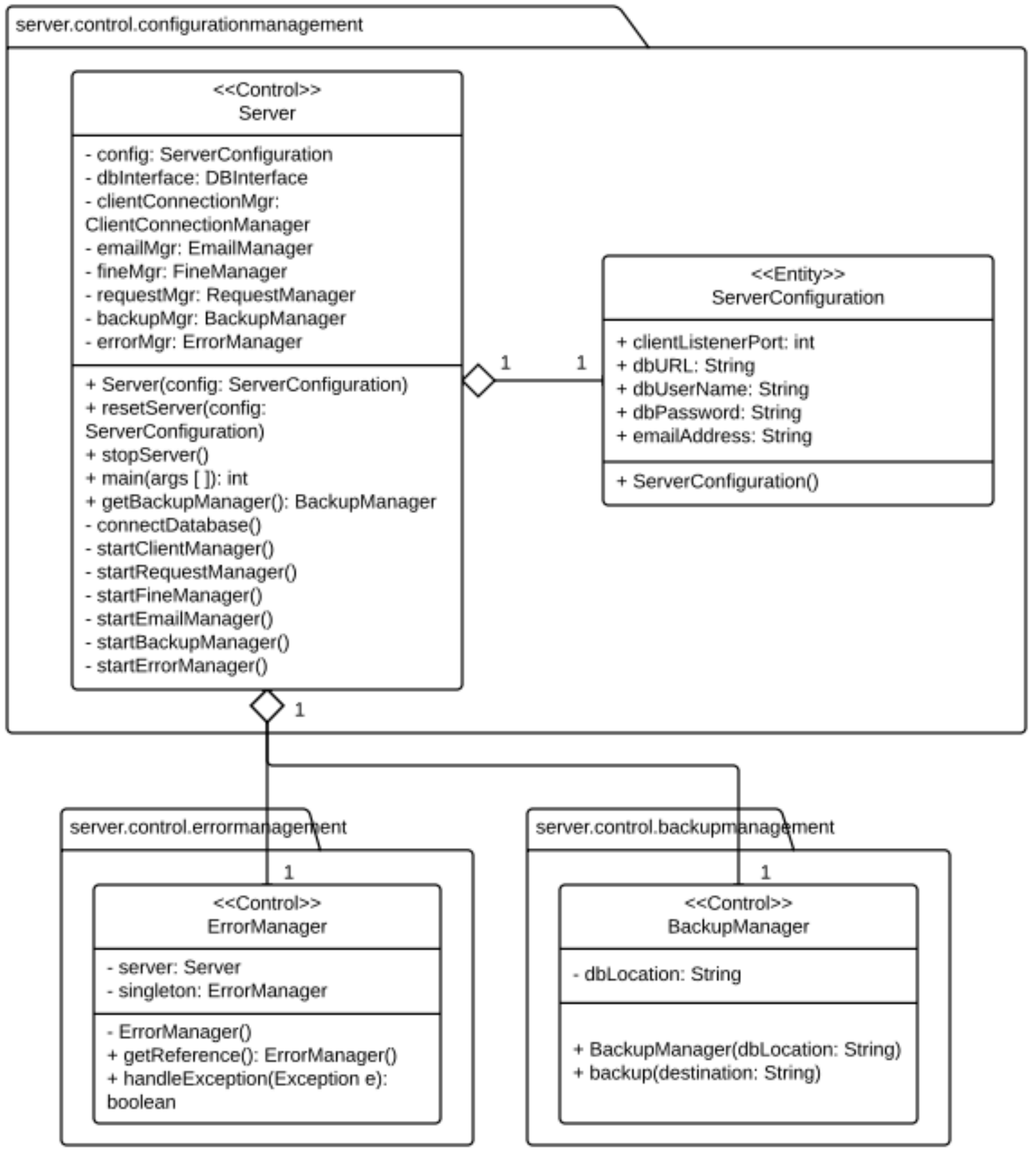


Figure A3.33 Server Class Diagrams (Cont.)


server.control.configurationmanagement
<<Control>>
Server
- config: ServerConfiguration
- dbInterface: DBInterface
- clientConnectionMgr: ClientConnectionManager
- emailMgr: EmailManager
- fineMgr: FineManager
- requestMgr: RequestManager
- backupMgr: BackupManager
- errorMgr: ErrorManager
+ Server(config: ServerConfiguration)
+ resetServer(config: ServerConfiguration)
+ stopServer()
+ main(args [ ]): int
+ getBackupManager(): BackupManager
- connectDatabase()
- startClientManager()
- startRequestManager()
- startFineManager()
- startEmailManager()
- startBackupManager()
- startErrorManager()
1
1
<<Entity>>
ServerConfiguration
+ clientListenerPort: int
+ dbURL: String
+ dbUserName: String
+ dbPassword: String
+ emailAddress: String
+ ServerConfiguration()
1
server.control.errormanagement
1
<<Control>>
ErrorManager
- server: Server
- singleton: ErrorManager
- ErrorManager()
+ getReference(): ErrorManager()
+ handleException(Exception e): boolean
server.control.backupmanagement
1
<<Control>>
BackupManager
- dbLocation: String
+ BackupManager(dbLocation: String)
+ backup(destination: String)