Hall Management System (HMS)

System Analysis / System Design (SA / SD) Document

Avikalp Srivastava (14CS10008)

Madhav Datt (14CS30015)

Group 19

Software Engineering Lab

TABLE OF CONTENTS

8.0 **Design Details**

1.0 Introduction

1.1 Purpose

The purpose of this document is to present a detailed system analysis of the Hall Management System (HMS) for IIT Kharagpur. It will explain and analyze the development and features, action sequences, object states etc. of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. This document is intended for both, the stakeholders and the developers of the system.

1.2 Scope

This software system will be a Management System for the HMC, Students, Wardens and Hall Staff of IIT Kharagpur. This system will be designed to maximize the ease of data management by providing tools to assist in automating various book-keeping activities associated with HMC's day to day operations, which would otherwise have to be performed manually.

More specifically, this system is designed to allow HMC to manage and communicate with a group of halls through their wardens and managers to accommodate and manage students of IIT Kharagpur.

The software provides convenient means to manage daily wage based hall staff, such as attendants and gardeners and their payrolls. It also enables students to lodge complaints about issues they face relating to their halls through a portal and get them addressed by their wardens.

1.3 Glossary

Term	**Interpretation/Meaning**
Total Student Dues/Total Student Charges	The total of all charges levied - Room rent, Mess charges and Amenities charges.
Daily Worker Salary/Wage	Total salary payable to daily workers - Gardeners and Attendants. Calculated as (Attendance during a month * Daily wage).

Total Annual Salaries payable by a Hall	Total money payable by a Hall in the form of salaries - calculated as the sum of Daily Worker Salaries and the Clerk Salary.
Annual Grant from the HMC to a Hall	Total yearly grant issued by the HMC for a hall to cover its Total Annual Salaries payable and other miscellaneous expenditures.

1.4 Background Document Analysis for Use Case and Class Diagram

1.4.1 Use Case Derivations

1.4.1.1 Student Use Case Diagram

Use Case	Statement from User Document
Take Admission <includes> Forward Admission Note	"After a student takes admission, he/she presents a note from the admission unit"
Pay Dues <includes> Amenity Dues, Room Charges, Mess charges <includes> Notify Mess Manager	"Whenever a student comes to pay his dues, his total due is computed as the sum of mess charge, amenity charge, and room rent." & "The total amount collected from each student of a hall towards mess charges is handed over to the mess manager every month."
Repair Request, Worker Complaint <extend> Raise Complaint	"The students should be able to raise various types of complaints" & "The complaints can be repair requests such as fused lights, non-functional water taps, nonfunctional water filters, room repair, etc. They can also register complaints regarding the behavior of attendants, mess staff, etc."

1.4.1.2 Warden Use Case Diagram

Use Case	Statement from User Document
Enter Annual Grant Request	"The Wardens of different halls should be able to enter their expenditure details against the allocations."
View Room Occupancy	"The warden of each hall should be able to find out the occupancy of his hall"
File ATR <extends> View Complaint	"He should also be able to view the complaints raised by the students and post his Action Taken Report (ATR) to each complaint."
Hire Worker <includes> Enter Pay, Create Record	"Whenever a new staff is recruited his details including his daily pay is entered."
Fire Worker <includes> Delete Records	"Whenever a staff leaves, it should be possible to delete his records."
Issue Workers' Pay Cheques <extends> View Pay List	"At the end of every month a consolidated list of salary payable to each employee of the hall along with cheques for each employee is printed out"
Print Statement extends View Statement of Accounts	"The warden should be able to view the statement of accounts any time. The warden would take a print out of the annual consolidated statement of accounts,"
Controlling Warden - View Overall Room Occupancy	"The controlling warden should be able to view the overall room occupancy."

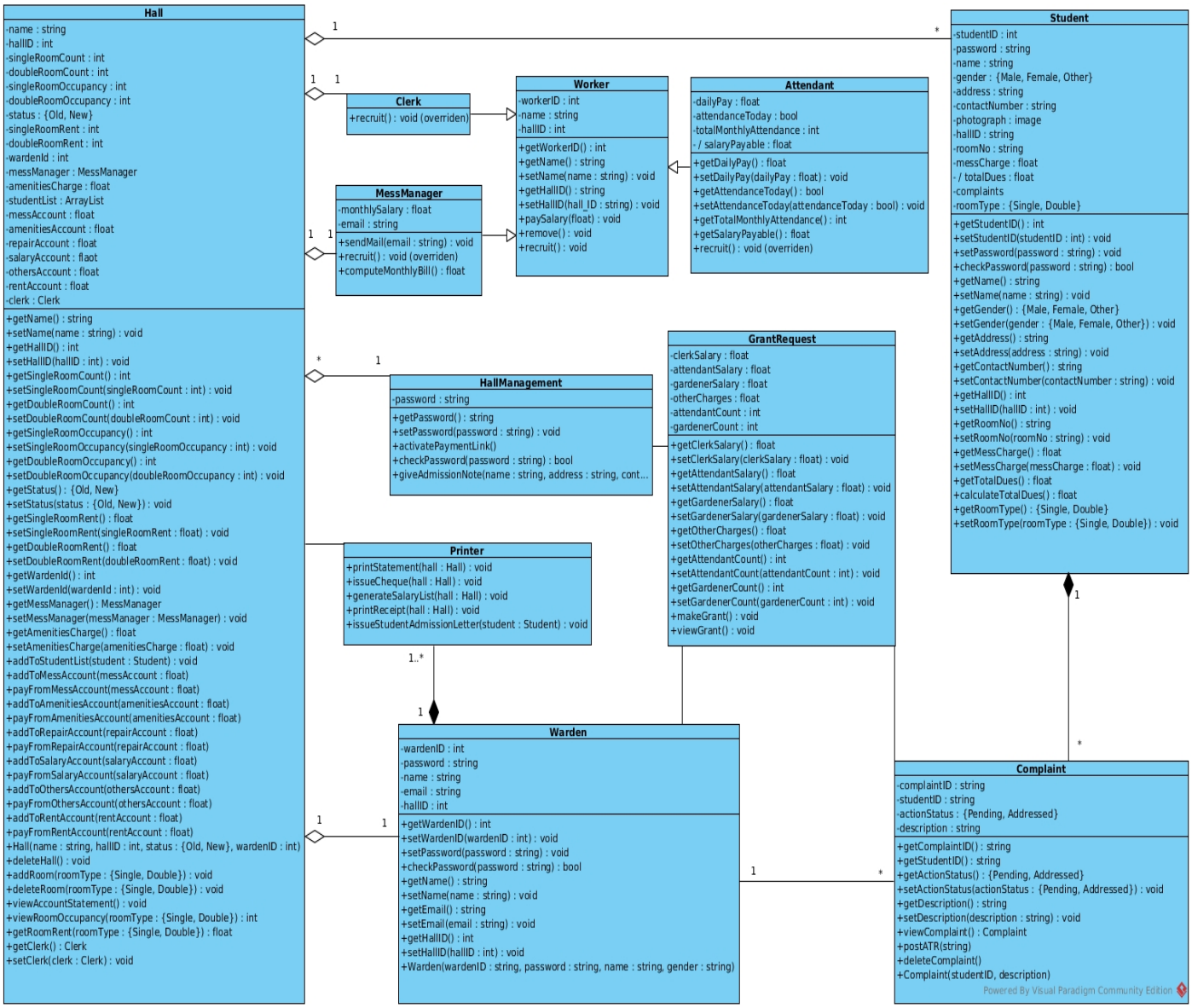
1.4.1.3 Hall Management Center Use Case Diagram

Use Case	Statement from User Document
Initialize Hall of Residence	Not Explicit.
Add Single Room Rent, Double Room Rent, Set Status Old/New etc.	"Each room has a fixed room rent. The newly constructed halls have higher rent compared to some of the older halls. Twin sharing rooms have lower rent"
Activate Payment Option, Alter Hall	Not Explicit.
Issue Annual Grant	"The HMC receives an annual grant from the Institute for staff salary and the upkeep of the halls and gardens. The HMC chairman should be provided support for distribution of the grant among the different halls"
Allot Accommodations <includes> Issue Admission Letter to Student, Update Hall	"He/she is then allotted a hall, and also a specific room number. A letter indicating this allotted room is issued to the concerned student."

1.4.1.4 Miscelleneous Use Case Diagram

Use Case	Statement from User Document
Print Pay Cheques	"At the end of every month a consolidated list of salary payable to each employee of the hall along with cheques for each employee is printed out."
Print Account Statement	"The warden would take a print out of the annual consolidated statement of accounts ..."
Print Student Admission Letter	" A letter indicating this allotted room is issued to the concerned student."
Mark Attendance	"The Hall clerk enters any leave taken by an attendant or a gardener from at the terminal located at the hall office."
Enter Monthly Charges for Students	"The mess manager would input to the software the total charges for each student in a month on mess account"

1.4.2 Refined Class Diagram



1.4.3 Class Diagram Documentation

The Hall Management System consists of the following classes to handle various parts of the functionality. Hall, Warden, Student, HallManagement, MessManager, Clerk, Attendant, Worker, GrantRequest, Printer and Complaint.

The above mentioned classes are related as follows:

- A Hall aggregates a Clerk, MessManager, Warden, and one or more Students and has a HallManagement. Hall is associated with Printer.
- A Warden contains/owns one or more Printers, and is associated with a Hall, a GrantRequest and zero or more Complaints.
- Attendant, MessManager and Clerk inherit from the class Worker. Each Clerk and MessManager is associated with one Hall.
- HallManagement is associated with zero or more Halls and GrantRequests. Only one object of such a class may exist.
- GrantRequest is associated with a Warden, multiple Complaints and HallManagement.
- Each Printer is owned by a Warden and is associated with a Hall.
- A Complaint is owned by a Student and is associated with one Warden and GrantRequest.
- Student is associated with a Hall and is composed of zero or more Complaints lodged.

1.4.3.1 Student Class

Student contains data members studentID, password, name, gender, address, contactNumber, hallID, roomNo, photograph, messCharge, roomType and their respective getter and setter functions.

The totalDues data member will be computed as Total Student Dues/Charges when the calculateTotalDues method is called. The method will take no parameters and return a float equalling the total amount due from the Student. The checkPassword function takes the password entered at login as parameter and returns boolean value true if authentication is successful.

1.4.3.2 Warden Class

Warden contains data members wardenID, password, name, email, hallID and their respective getter and setter functions. The checkPassword function takes the password entered at login as parameter and returns boolean value true if authentication is successful.

A Warden object may be constructed through the Warden constructor with the parameters wardenID, password, name, email and hallID.

1.4.3.3 Hall Class

Hall contains data members name, hallID, singleRoomCount, doubleRoom count, singleRoomOccupancy (number of occupied single rooms), doubleRoomOccupancy (number of occupied double rooms), status (if the hall is old or new), singleRoomRent, doubleRoomRent, wardenID, messManager, clerk, amenitiesCharge, and their respective getter and setter functions.

The studentList data member contains all Student residents of a particular hall in an ArrayList. The addToStudentList function takes Student as parameters and adds passed Student to the studentList.

The messAccount, amenitiesAccount, repairAccount, salaryAccount, othersAccount and rentAccount float type data members keep records of the various expenditure accounts of the Hall. The functions addToMessAccount, addToAmenitiesAccount, addToRepairAccount, addToSalaryAccount, addToOthersAccount and addToRentAccount take the respective account data member as parameter and add amount to the specified account. The functions payFromMessAccount, payFromAmenitiesAccount, payFromRepairAccount, payFromSalaryAccount, payFromOthersAccount and payFromRentAccount take the respective account data member as parameter and pay amount from the specified account, wherever due.

viewRoomOccupancy method of the Hall takes roomType as parameter and returns an integer with the number of rooms of the said type that are occupied. The viewAccountStatement method takes no parameters and displays the details of the various accounts of the Hall.

The addRoom and deleteRoom functions take roomType as parameter and add or delete a room of the specified type to the Hall. A Hall may be constructed using the Hall constructor with the parameters hallID, name, status and wardenID. The deleteHall method deletes an existing Hall.

1.4.3.4 HallManagement Class

HallManagement contains password as data member with a getter and setter function for the same. The checkPassword function takes the password entered at login as parameter and returns boolean value true if authentication is successful.

The giveAdmissionNote function takes Student details - name, address, contactNumber and photograph as parameters and issues an admission note to the said student, thus creating/adding the Student to the system.

The activatePaymentLink function activates a payment link for the Student, enabling them to pay their Total Student Dues.

1.4.3.5 Clerk Class

The Clerk Class inherits from the Worker Class. The inherited characteristics can be followed under section 1.4.3.7.

The overridden recruit function takes no parameters and recruits/adds a Clerk to the system for a specific Hall.

1.4.3.6 MessManager Class

The MessManager Class inherits from the Worker Class. The inherited characteristics can be followed under section 1.4.3.7.

MessManager contains the data members monthlySalary and email. The computeMonthlyBill function computes the Total Monthly Mess charges payable by resident Students of the Hall.

The overridden recruit function takes no parameters and recruits/adds a MessManager to the system for a specific Hall.

1.4.3.7 Worker Class

Worker Class contains data members workedID, name and hallID with their respective getter and setter functions.

paySalary function pays salary of the worker, computed as the Daily Worker Wage or Monthly Salary, from the Hall's Salary account.

The overridden recruit function takes no parameters and recruits/adds a Worker to the system for a specific Hall. The remove function takes no parameters and removes a worker from the system, implying quitting of work or firing of Worker by Warden.

1.4.3.8 Attendant Class

The Attendant Class inherits from the Worker Class. The inherited characteristics can be followed under section 1.4.3.7.

Attendant can be of the type attendant or gardener and contain the data members dailyPay, attendanceToday with their respective getter and setter functions. totalMonthlyAttendance data member is computed as the sum of number of present days during the said month and is returned by function getTotalMonthlyAttendance, which takes no parameters.

The salaryPayable data member is computed using the getSalaryPayable function which takes no parameters and returns an integer with the Daily Worker Salary/Wage.

The overridden recruit function takes no parameters and recruits/adds an Attendant to the system for a specific Hall.

1.4.3.9 Printer Class

Printer Class contains no data members. The issueStudentAdmissionLetter function takes Student as parameter and prints the said Student's admission letter as a PDF.

The printStatement, issueCheque, generateSalaryList and printReceipt functions, each take Hall as parameters and print Account Statements, issue Salary Payment Cheques, display list of all Worker salaries and print expense receipts for the said Hall respectively.

1.4.3.10 GrantRequest Class

GrantRequest contains data members clerkSalary, attendantSalary, gardenerSalary, otherCharges, attendantCount and gardenerCount with their respective getter and setter functions.

The makeGrant function is used to make/approve an annual grant requested by a Hall. The viewGrant function displays details of a filed Annual Grant Request by a Warden.

1.4.3.11 Complaint Class

Complaint Class contains data members complaintID, studentID and description with their respective getter functions. Data member actionStatus, to indicate if the Complaint has been addressed, with its getter and setter function.

The viewComplaint and deleteComplaint functions take no parameters and are used to display the details of the Complaint or delete lodged Complaint.

The postATR function takes a string ATR as parameter and add details of the Action Taken Reported posted by the respective Warden.

A Complaint object may be constructed using the Complaint constructor with parameters studentID and description.

1.5 Overview of SA Document

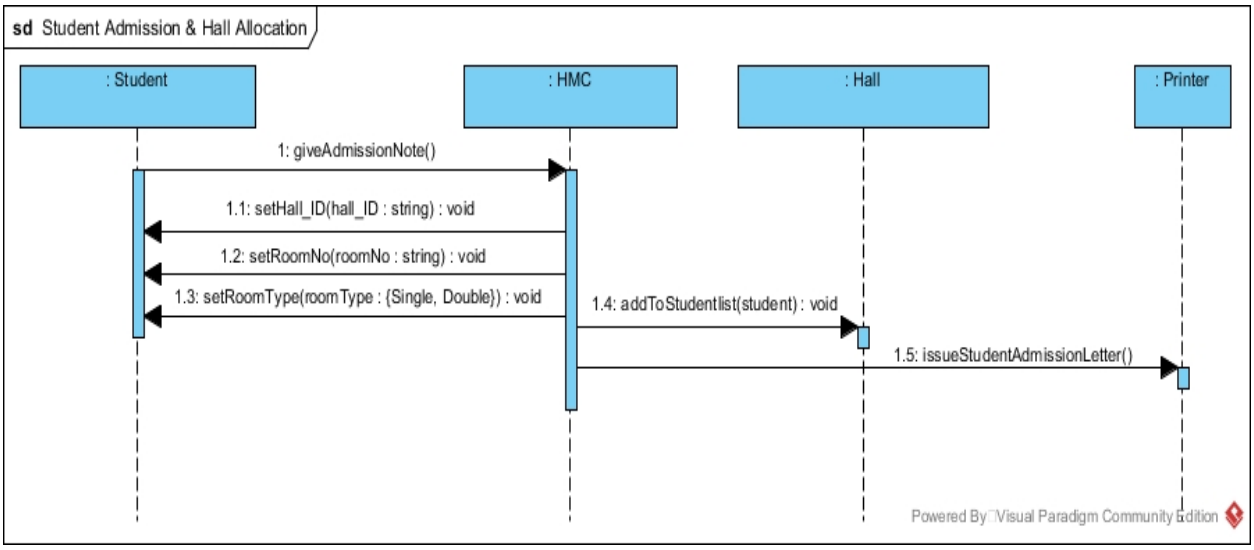
For most part, the rest of the System Analysis document is for the developers working to program the software. The next chapter, Sequence Diagrams show object interactions arranged in time sequence. They depicts the objects and classes involved in the scenario (specifically, various use cases) and the sequence of messages exchanged between the objects needed to carry out the functionality specified.

Chapter 3 contains an illustration of the states (stage in the evolution or behavior of an object) an object can attain as well as the transitions between those states in the UML. Chapter 4 contains Activity Diagrams, to represent control flow from one activity to another and to describe dynamic aspects of the system. The fifth chapter contains Collaboration Diagrams to indicate method call sequences along with the object organization. This is primarily designed for the developers.

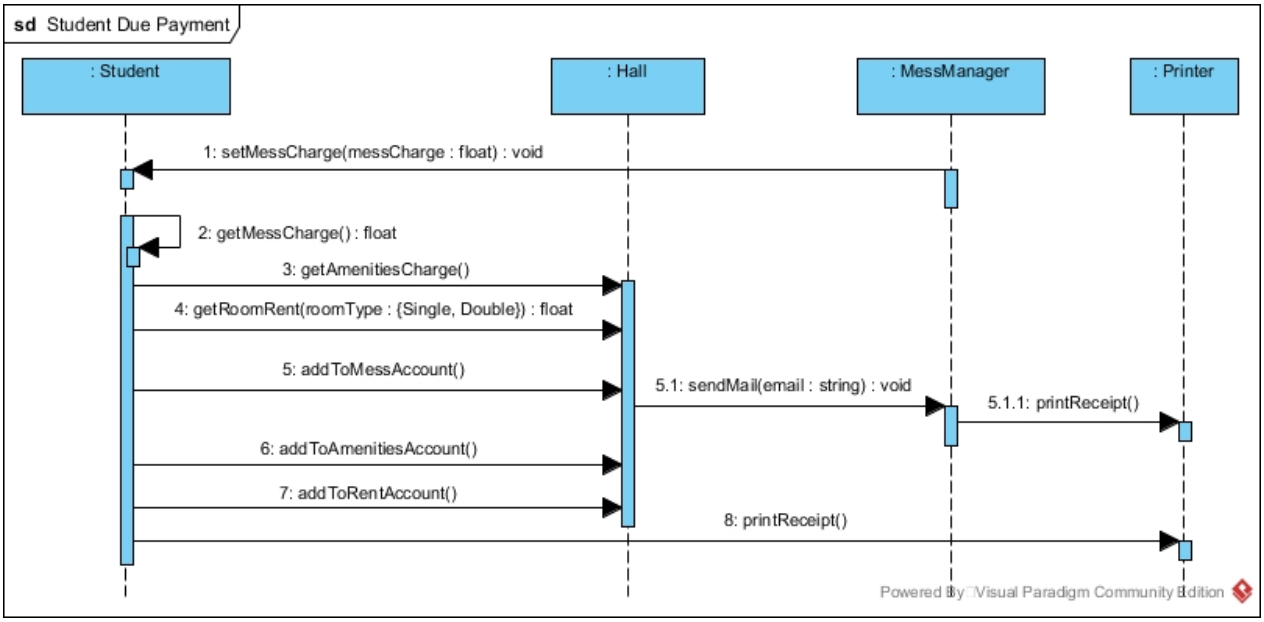
The fourth section of the document consists of System Parameters, followed by certain limitations explained in the last section.

2.0 Sequence Diagrams

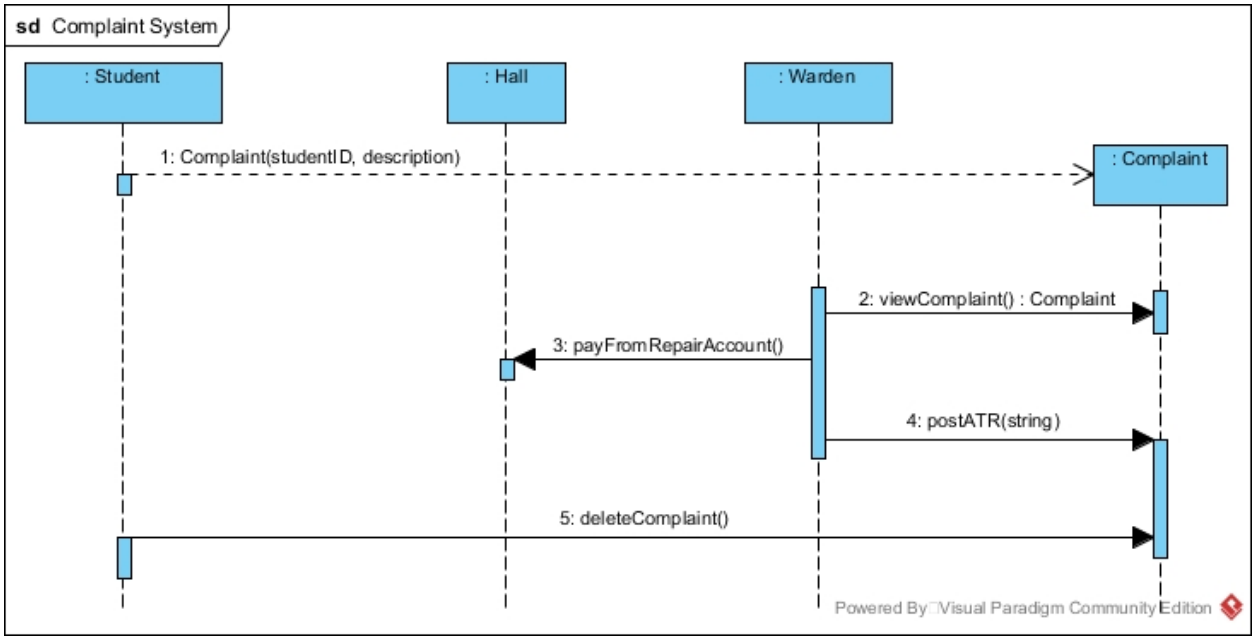
2.1 Student Admission and Hall Allocation



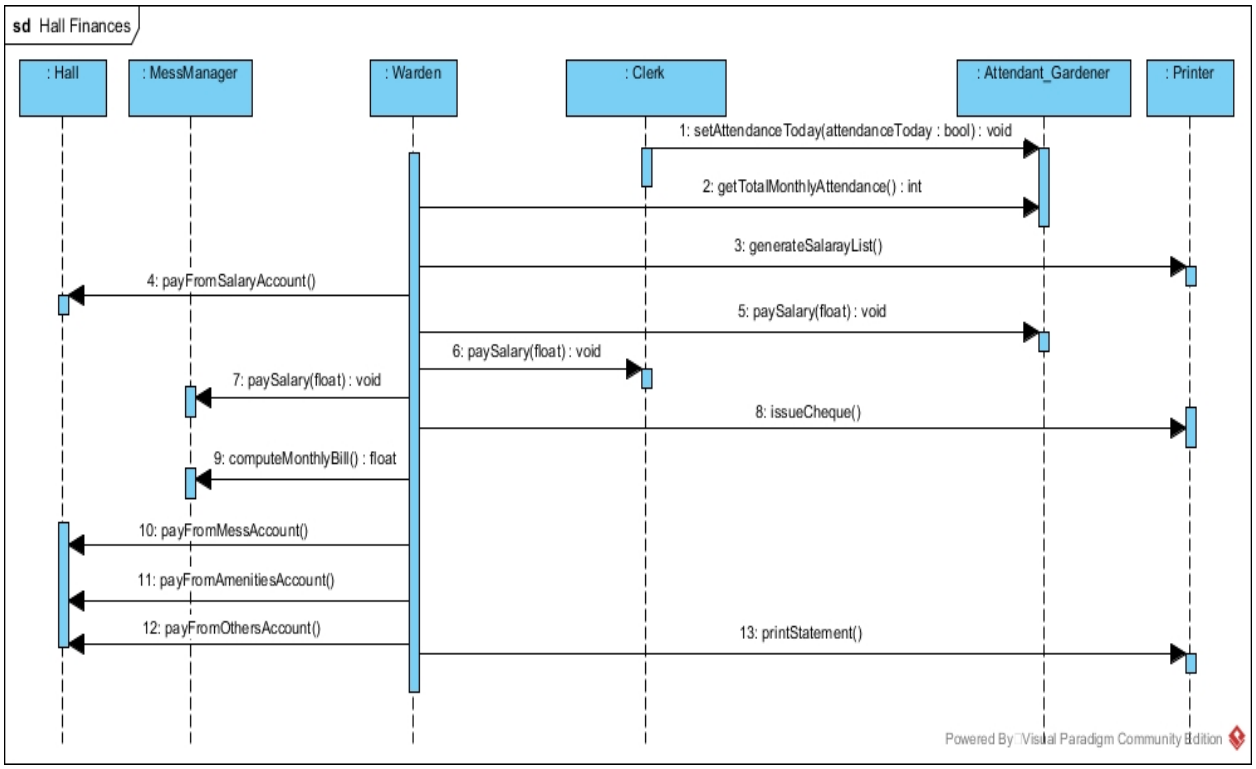
2.2 Student Due Payment



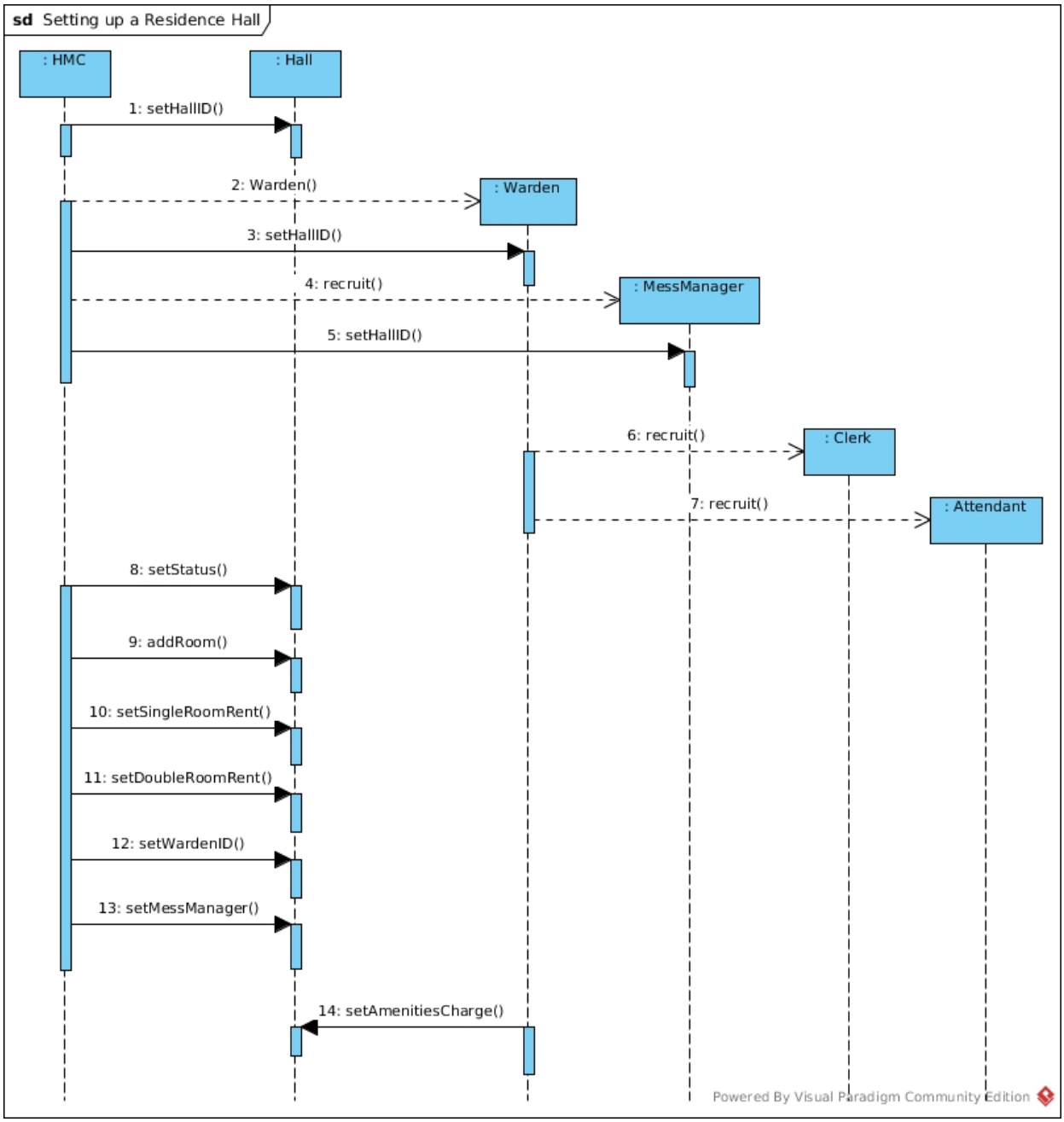
2.3 Complaint System



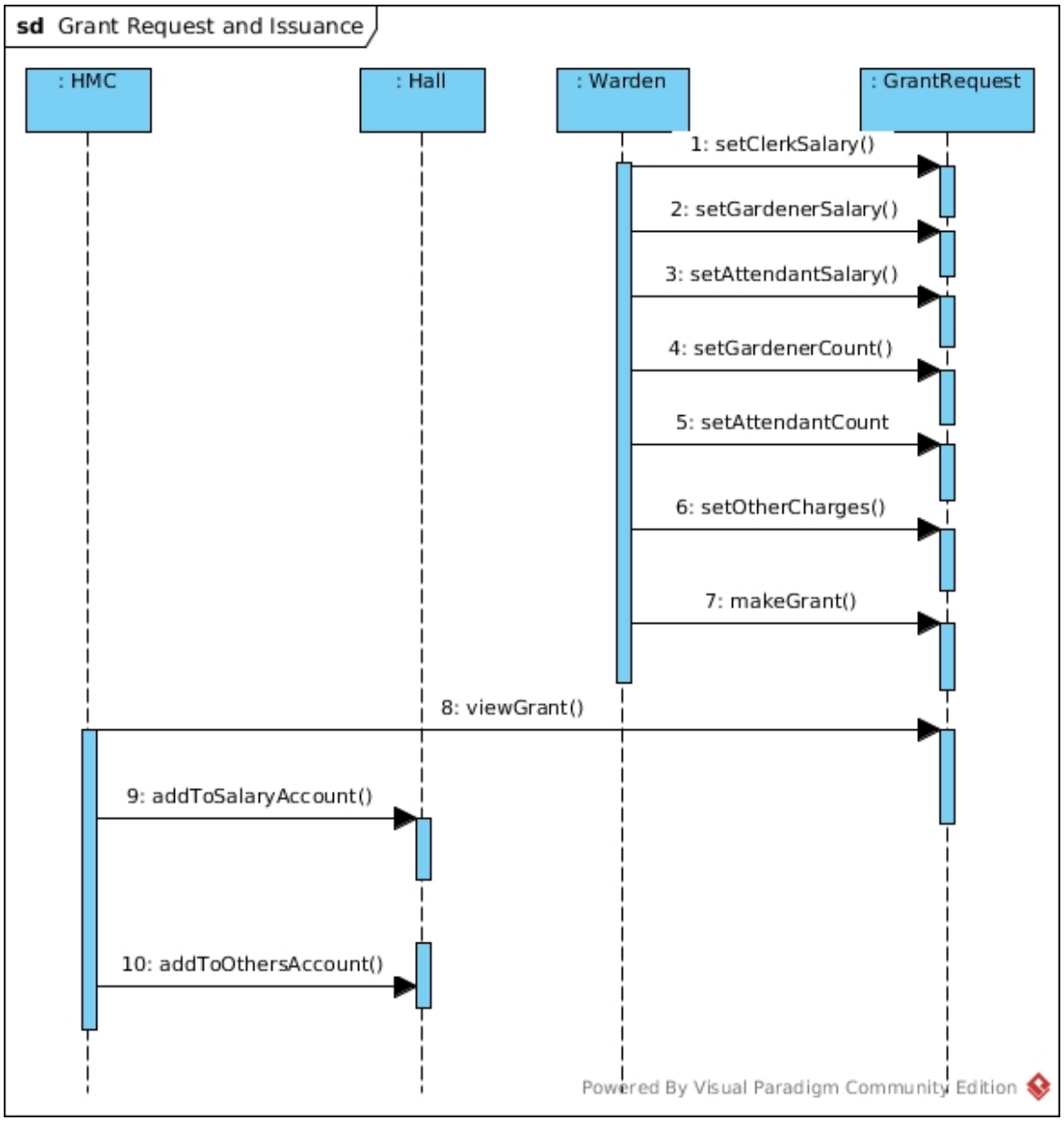
2.4 Hall Finances



2.5 Setting up a Residence Hall

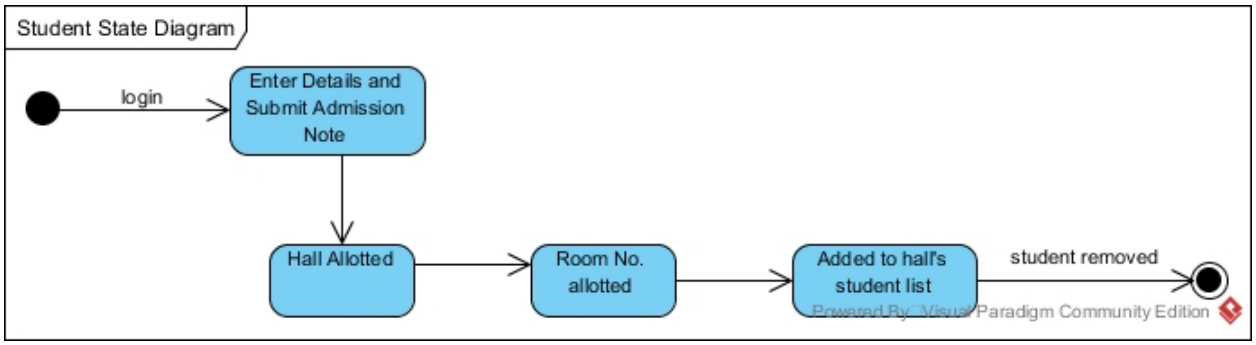


2.6 Requesting Annual Grant

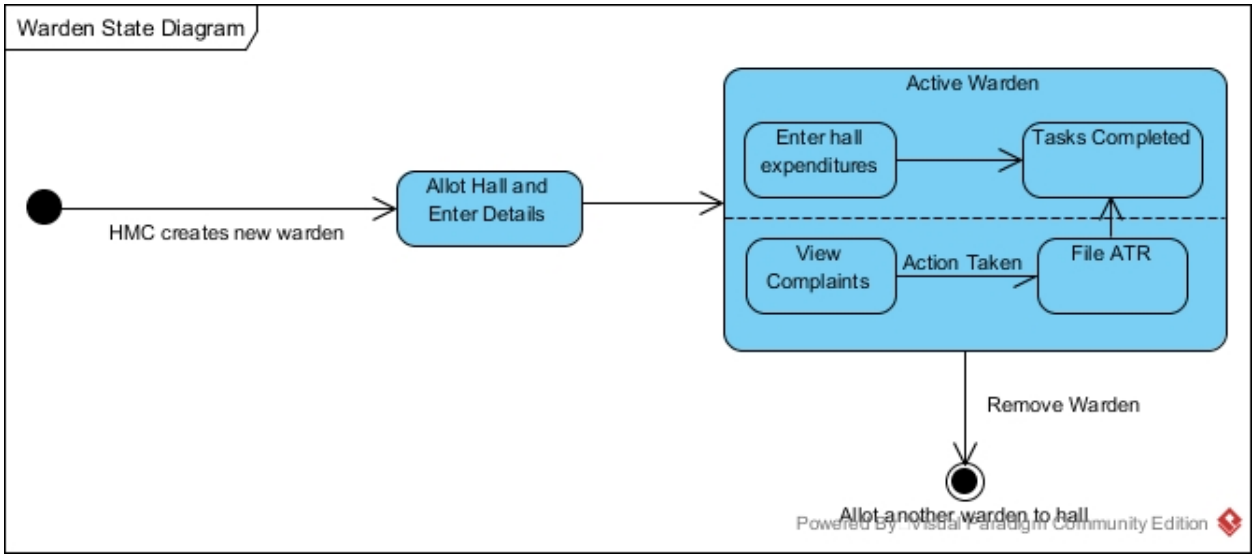


3.0 State Chart Diagrams

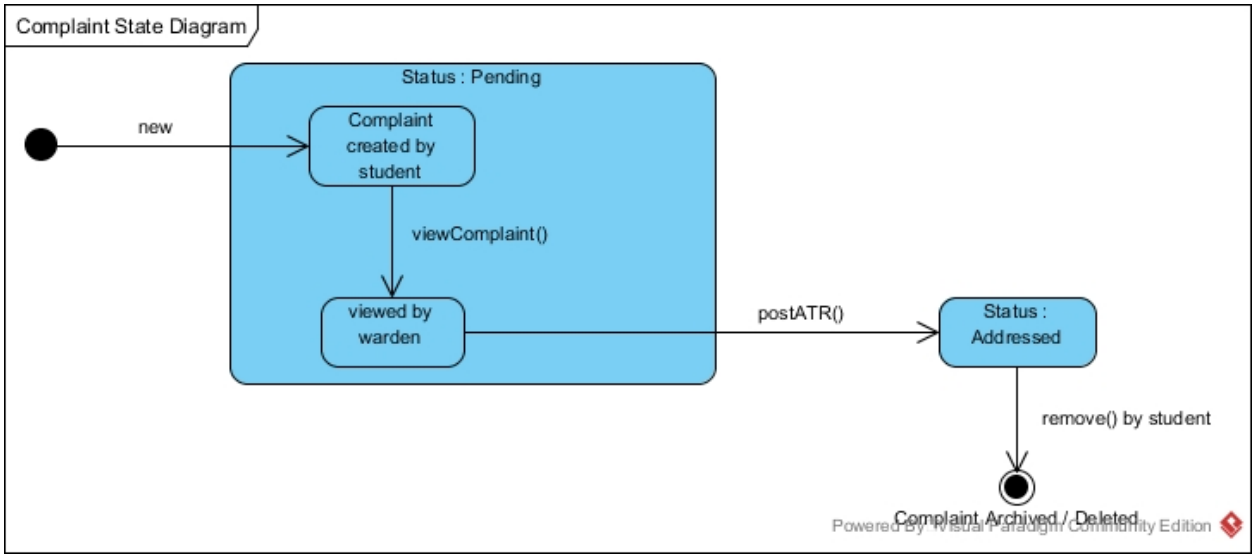
3.1 Student State Diagram



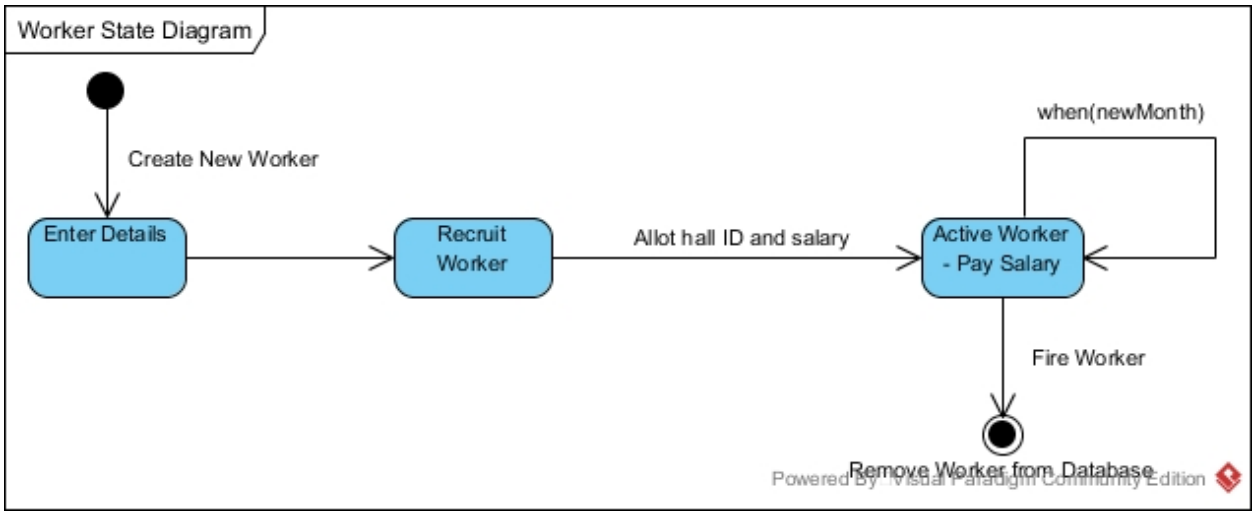
3.2 Warden State Diagram



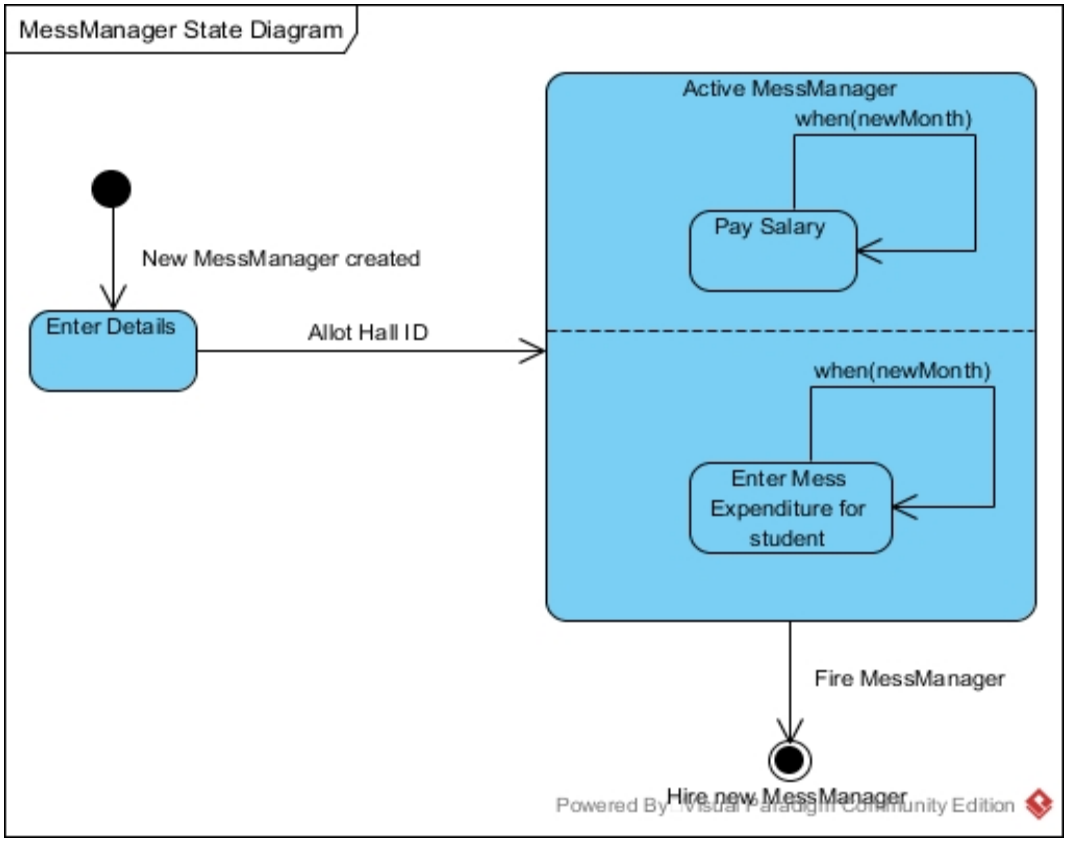
3.3 Complaint State Diagram



3.4 Worker State Diagram

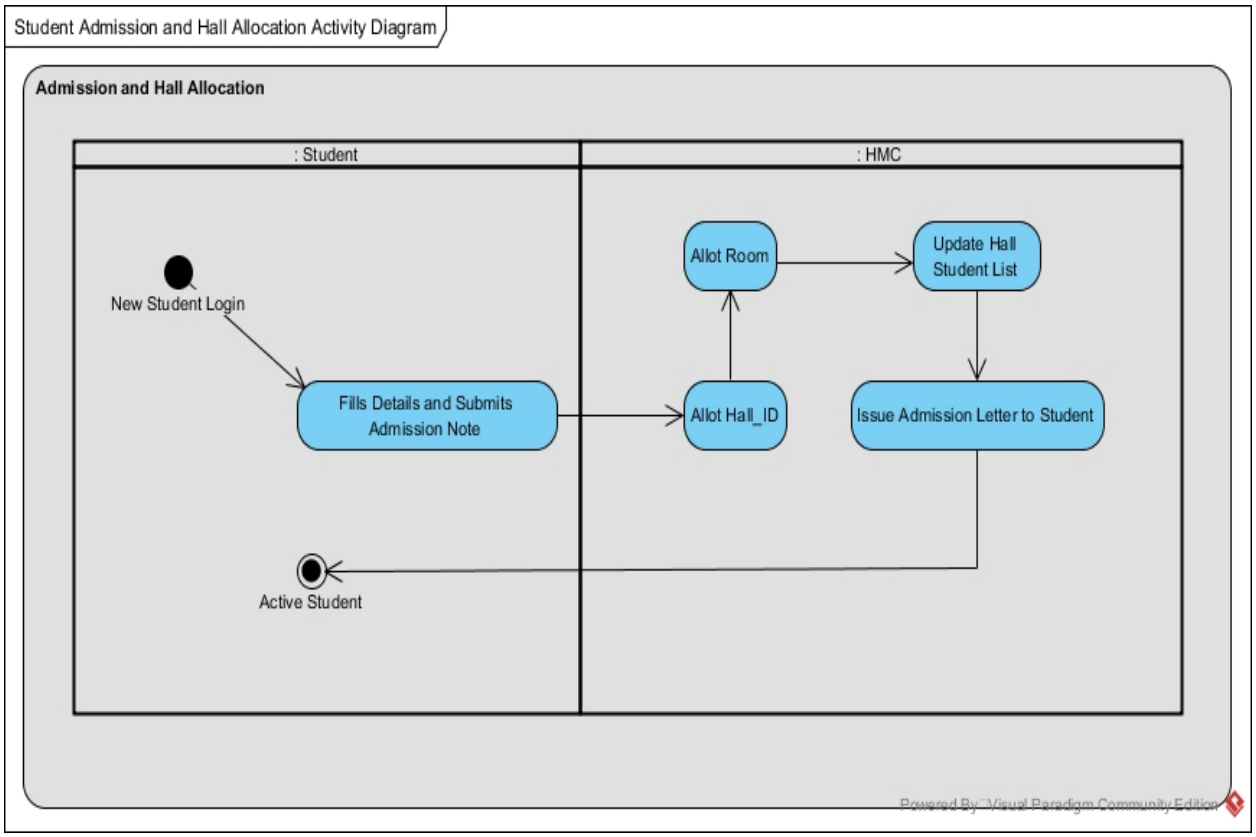


3.5 Mess Manager State Diagram

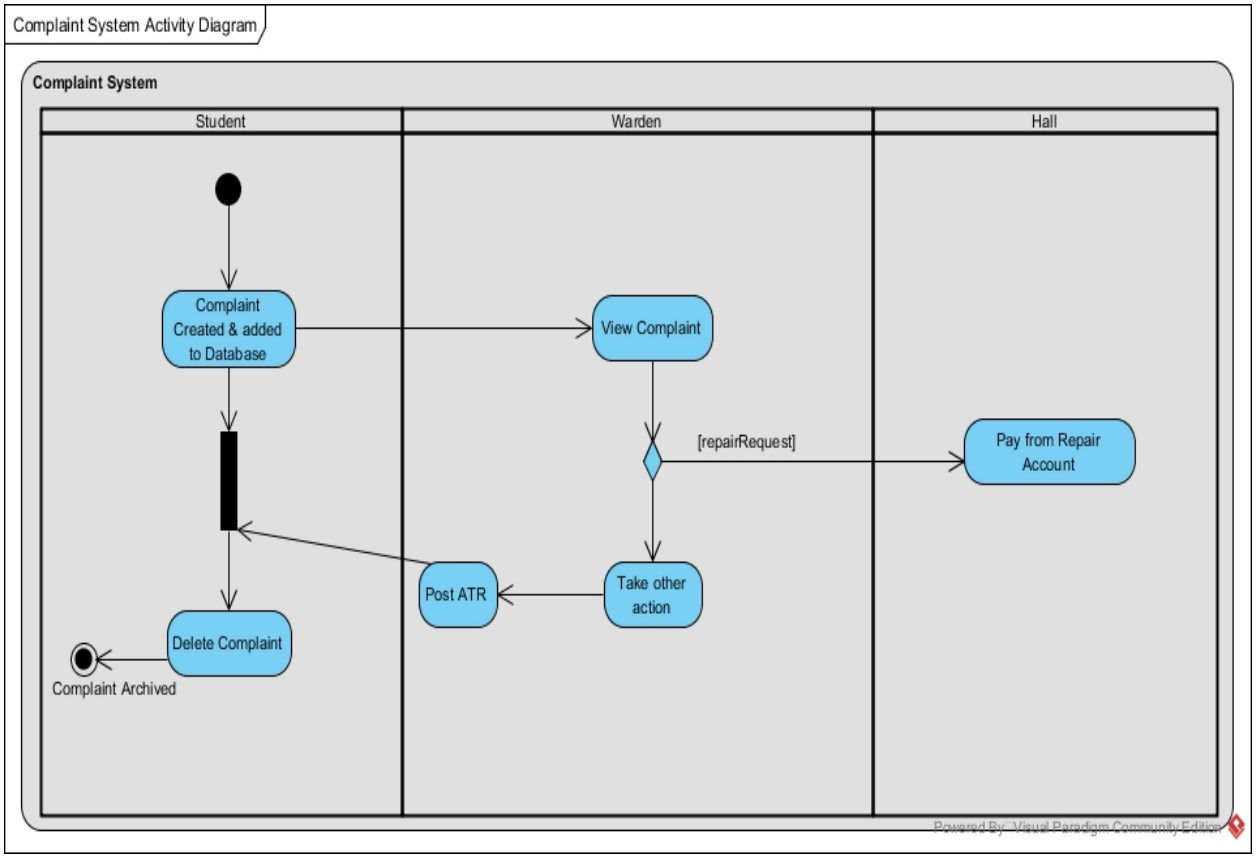


4.0 Activity Diagram Analysis

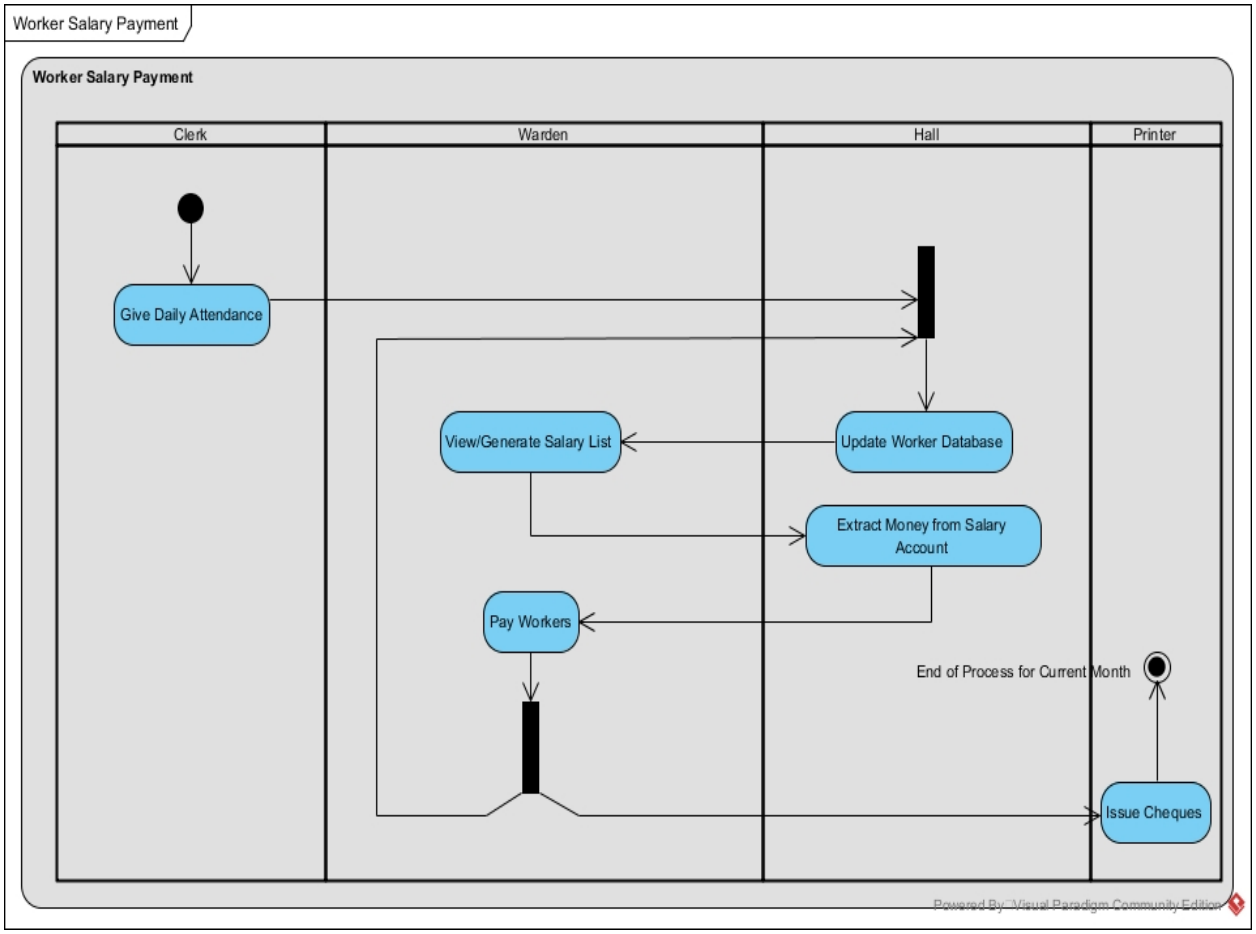
4.1 Student Admission and Hall Allocation Process



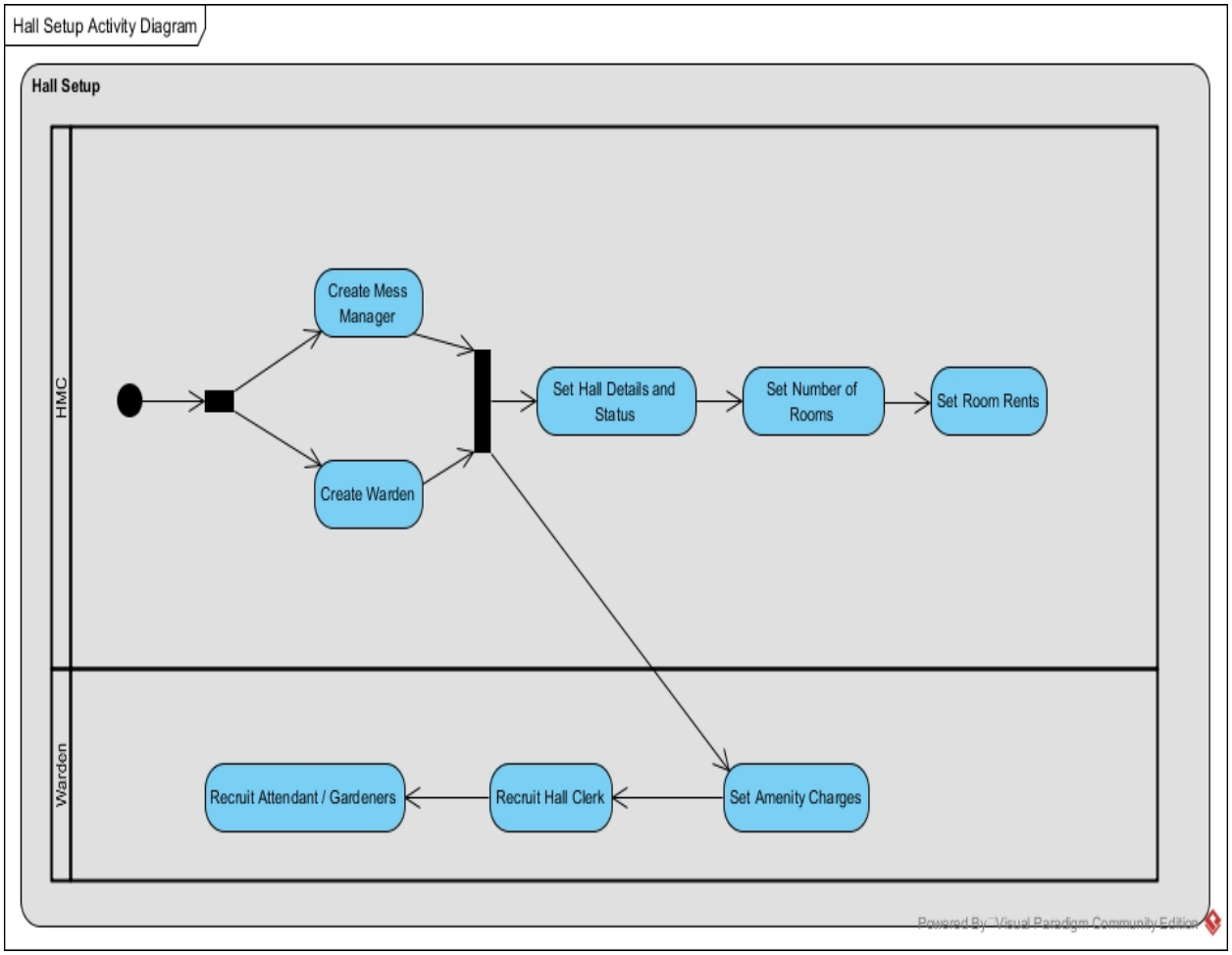
4.2 Complaint System



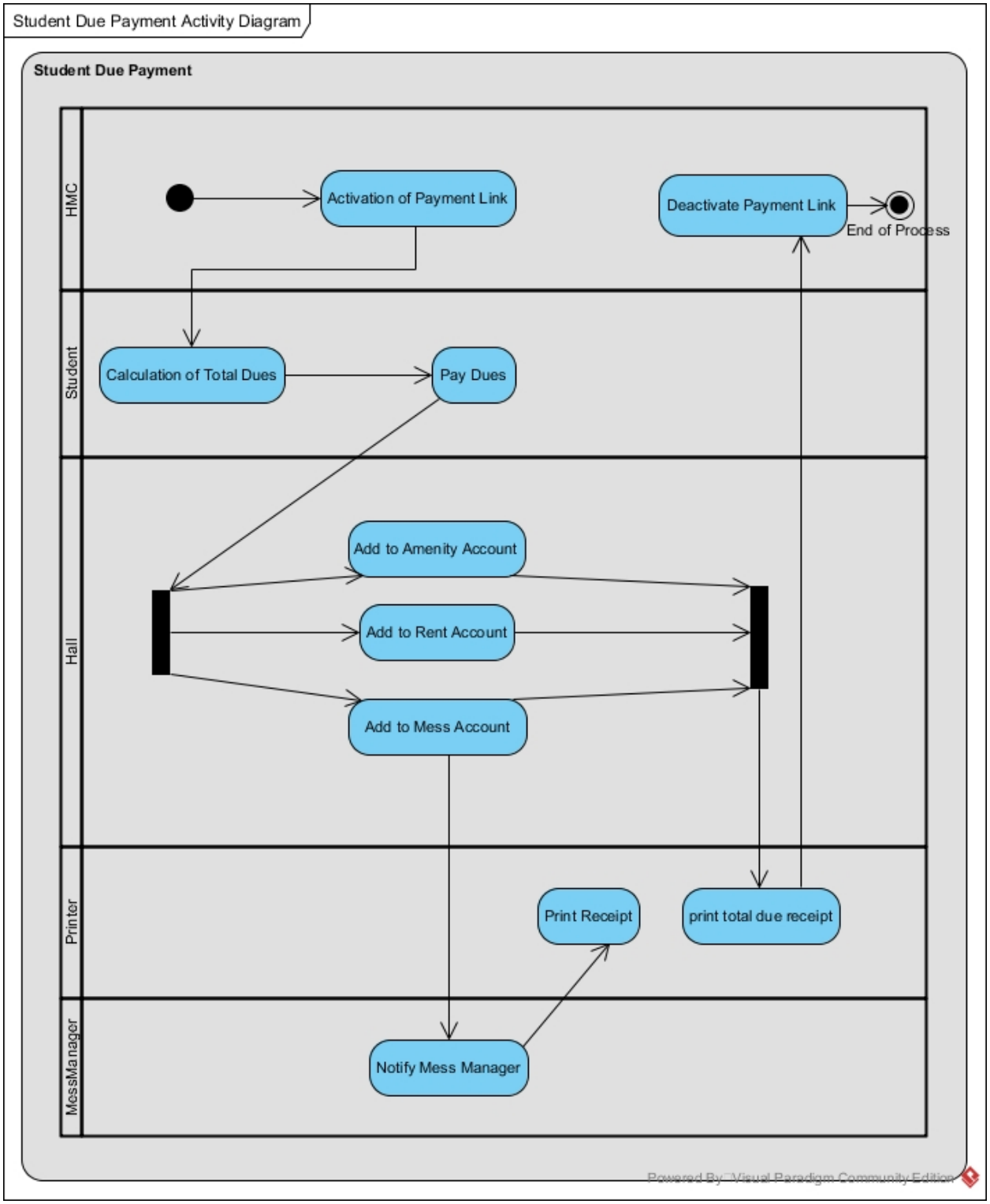
4.3 Worker Salary Payment Process



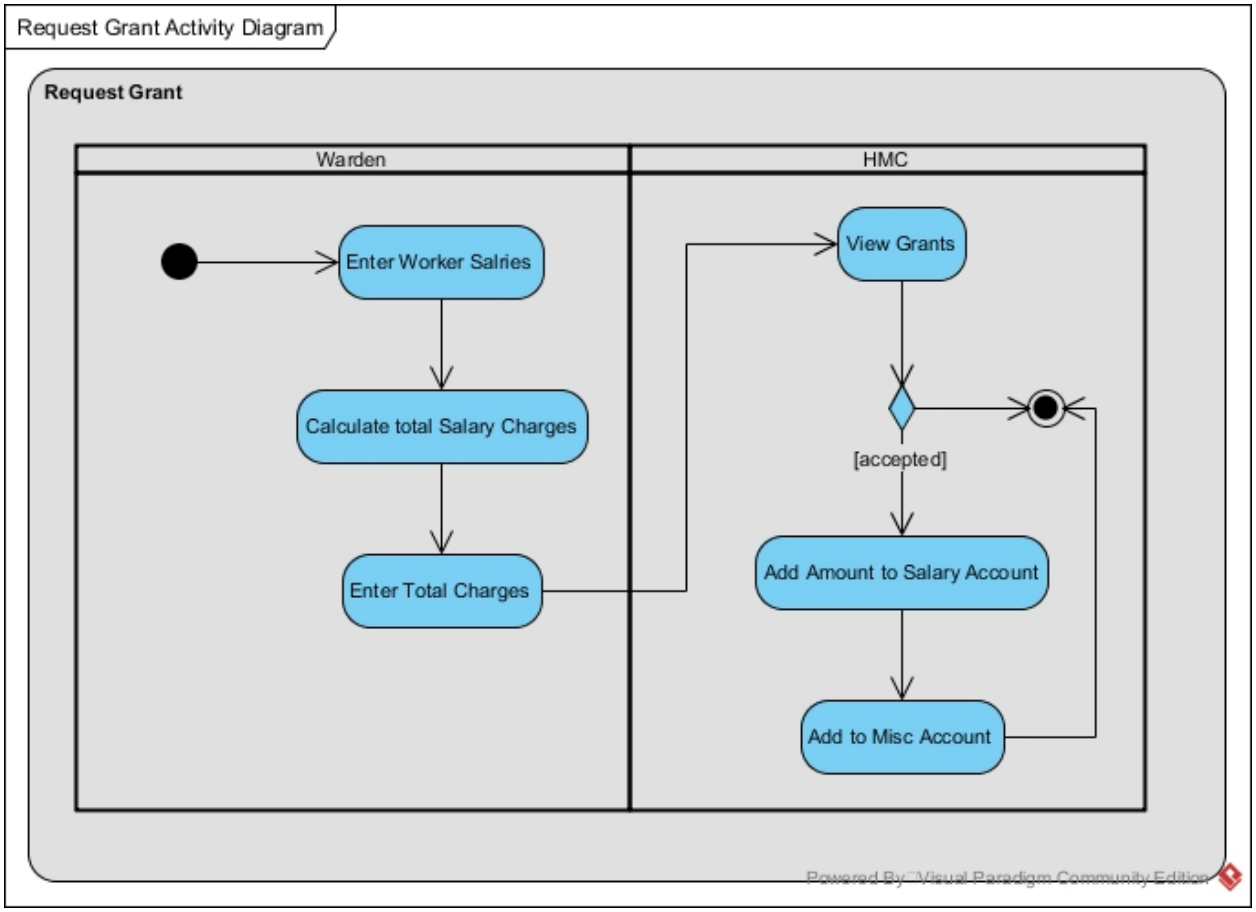
4.4 Hall Setup Activity Diagram



4.5 Student Due Payment

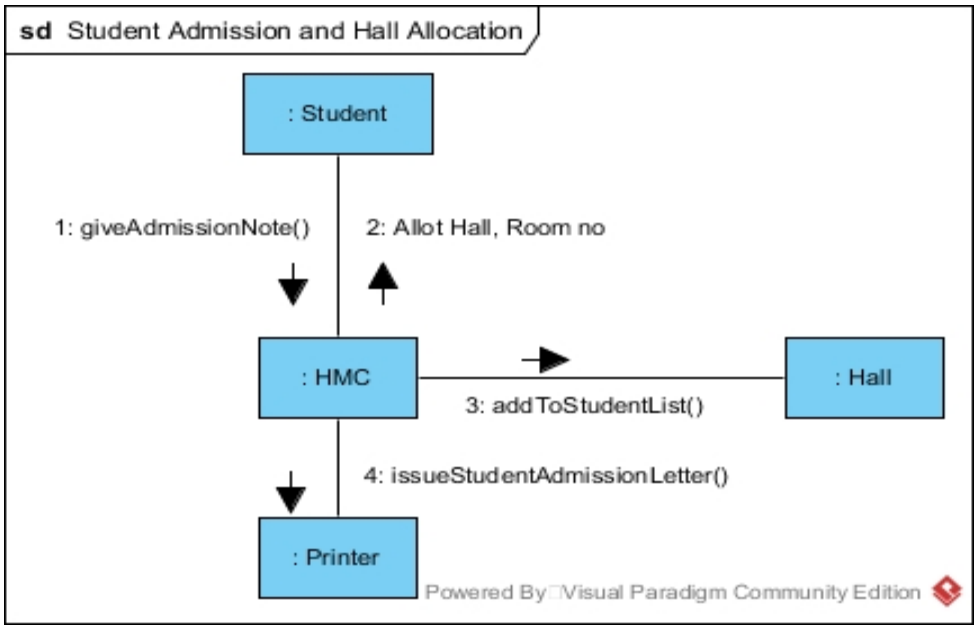


4.6 Request Grant Process

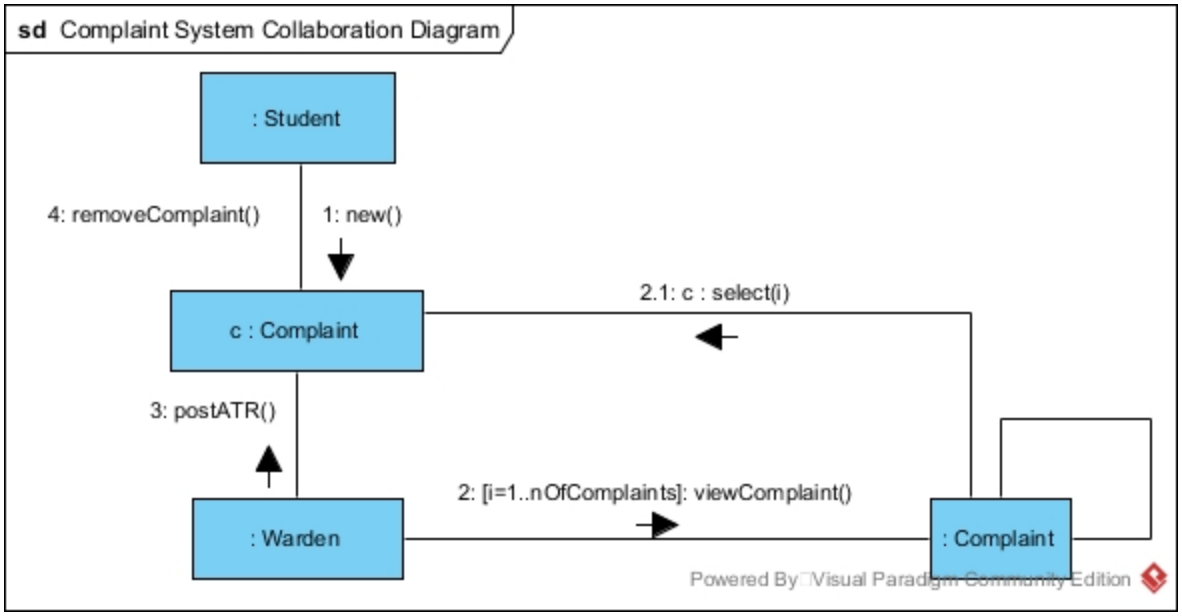


5.0 Collaboration Diagram Analysis

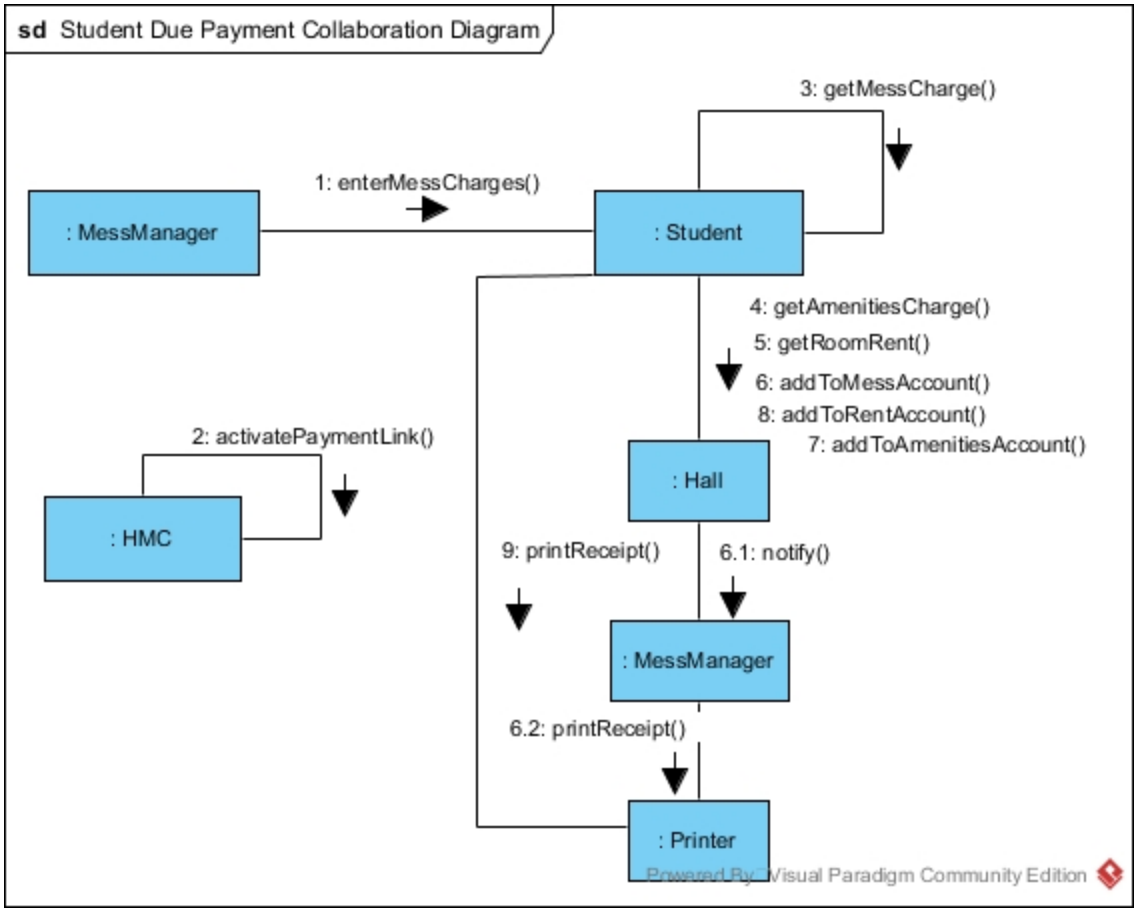
5.1 Student Admission and Hall Allotment



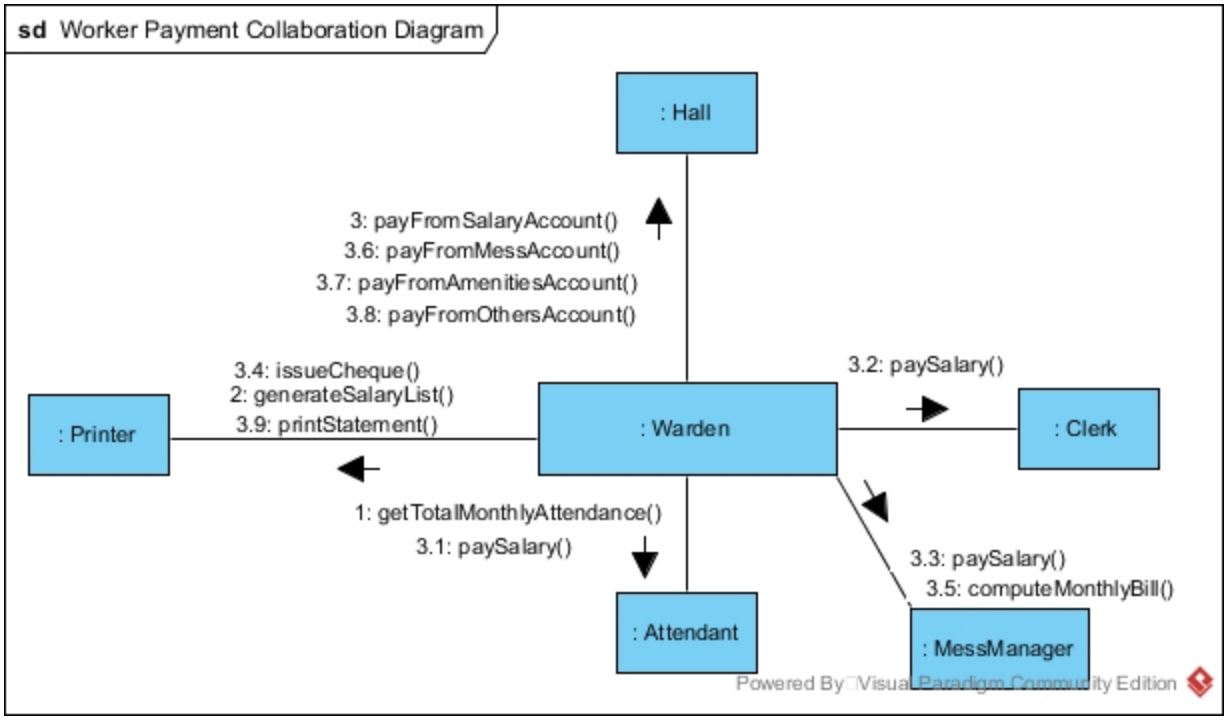
5.2 Complaint System Collaboration Diagram



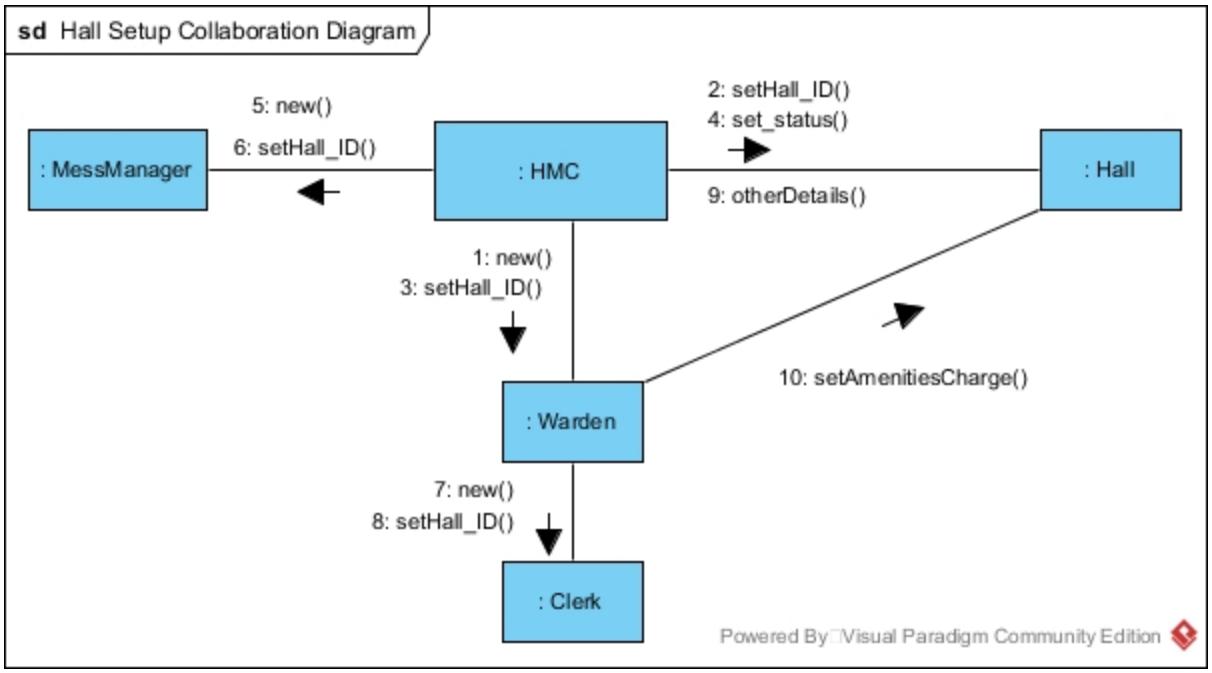
5.3 Student Due Payment Collaboration Diagram



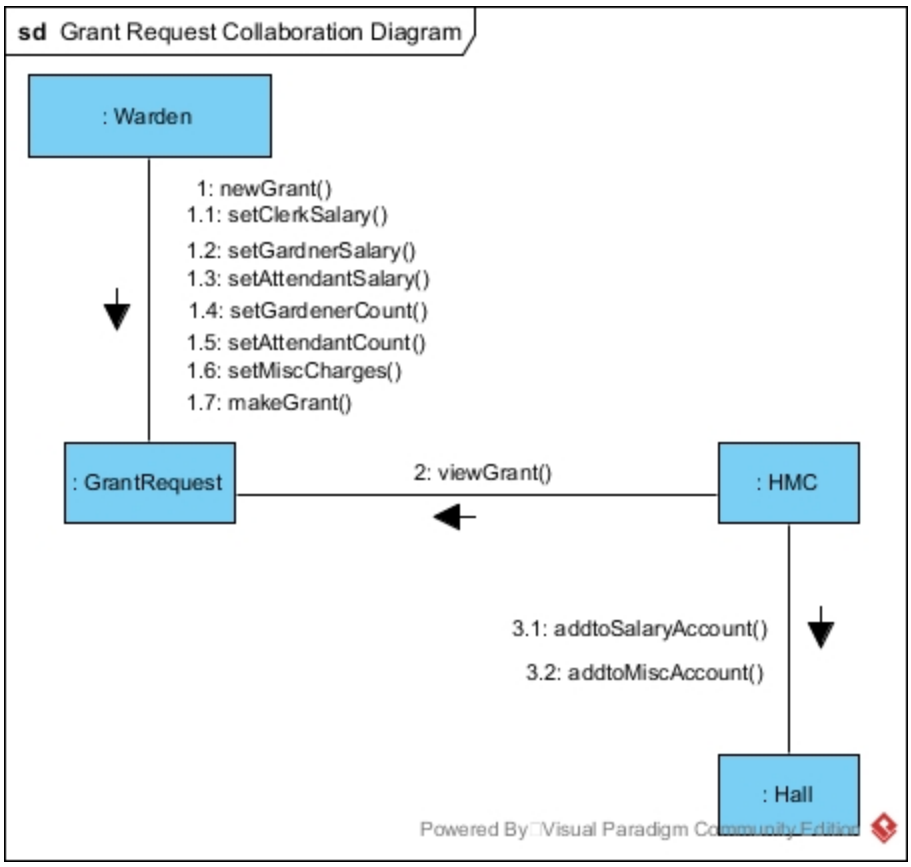
5.4 Worker Payment Collaboration Diagram



5.5 Hall Setup Collaboration Diagram



5.6 Grant Request Collaboration Diagram



6.0 System Parameters

6.1 Global System Architecture

The overall system architecture is a 2-tier architecture which includes client at one end and the database at the server side. Both the MySQL Database server and the client interface of the software and its various components are hosted on the same system.

6.2 Platform, Language, Build System, Libraries

The Hall Management System Software, though is meant to be cross-platform, is designed for systems running Linux operating systems. The software will be written in Python 2.7

with the open source MySQL being used as the relational database to store Hall, Student, Warden, Worker etc. data. The MySQL Connector API will be used to mediate between the Python program and the database. Interfaces will be made using TkInter.

External libraries used include MySQL Python Libraries, pyPDF2, AuthKit 0.4.5, pyUnit unittest Unit Testing Framework, pylint for debugging and warnings etc. All these libraries are either free and open-source, or free for non-commercial use. For RegEx matching, re library from the Python Standard Libraries is used.

The following are required to run and use the software:

- Python should be installed on all computers (preferably Python 2.7) running the software and its various components.
- MySQL Database should be installed on the server/computer running the software and its various components.
- MySQL Connector to connect to the database.
- pyPDF2 to create PDF files for material to be printed.
- Latest version of the Google Chrome browser to access the Web Portal for lodging complaints.

6.3 Sizing, Security and Performance

Considering that about 10,000 students live in hostels, the response time of the web site for students to file complaints, should be acceptable even under 1000 simultaneous clicks.

Since the software deals with hall accounts, wage payments to workers, and various payables by students, it should be very secure to prevent the possibility of various types of frauds, mismanagement of money and financial irregularities.

For added security and system protection, hashed passwords will be stored in the database after obtaining hash value from a secure SHA - 2 or bcrypt algorithm. A random and unique “salt” is used with the one-way function that hashes the password, to prevent Rainbow Table attacks. This renders brute forcing impossible too.

The software should be responsive and have excellent performance. The program is expected to give an output/complete the action initiated by the user in 100 milliseconds for most actions and

tasks in the software and its various components. Some actions such as printing are expected to take multiple seconds.

6.4 Software Architecture

Object-oriented architecture forms the basis of the HMS software.

In this style, data representations and their associated primitive operations are encapsulated in an abstract data type or object. The components of this style are the objects or instances of the abstract data types. Objects interact through function and procedure invocations. The two important aspects of this style are as follows:

- An object is responsible for preserving the integrity of its representation (usually by maintaining some invariant over it)
- The representation is hidden from other objects. This kind of abstraction makes the software easier to develop.

7.0 Limitations

7.1 Limitations on Queries

The HMS software has to be continuously optimized to minimize the number of MySQL database queries needed for regular operation. Heavy traffic can increase the amount of MySQL usage needed to run defined operations. In high-traffic situations, many simultaneous database connections can cause excessive strain on the server. An incomplete connection to a server causes the "Connection timed out" response in the visitor's browser.

7.2 Restrictions on Table Size

The effective maximum table size for MySQL databases is usually determined by operating system constraints on file sizes, and the program may lead to a table-full error in some of the following circumstances and situations:

- The disk might be full.

- The InnoDB storage engine maintains InnoDB tables within a tablespace that can be created from several files. This enables a table to exceed the maximum individual file size.
- You are using the MEMORY (HEAP) storage engine; in this case you need to increase the value of the max_heap_table_size system variable.

8.0 Design Details

8.1 Database Design

The Relational Database Management System (RDBMS) MySQL is being used. The schema with data attributes for various Tables of data used in the system is as follows:

8.1.1 Table: student

studentID : UNSIGNED INT (Primary Key), password : VARCHAR(200), name : VARCHAR(50), gender : CHAR, address : VARCHAR(50), contactNumber : UNSIGNED INT, hallID : UNSIGNED INT (Foreign Key), roomNo : VARCHAR(5), messCharge : FLOAT(10,2), roomType : CHAR

8.1.2 Table: warden

wardenID : UNSIGNED INT (Primary Key), password : VARCHAR(200), name : VARCHAR(50), email : VARCHAR(20), hallID : UNSIGNED INT (Foreign Key)

8.1.3 Table: hall

hallID : UNSIGNED INT (Primary Key), name : VARCHAR(50), clerkID : UNSIGNED INT (Foreign Key), messManagerID : UNSIGNED INT (Foreign Key), status : CHAR, singleRoomCount : UNSIGNED INT, doubleRoomCount : UNSIGNED INT, singleRoomOccupancy : UNSIGNED INT, doubleRoomOccupancy : UNSIGNED INT, singleRoomRent : UNSIGNED INT, doubleRoomRent : UNSIGNED INT, amenitiesCharge : UNSIGNED INT, messAccount : FLOAT(10,2), amenitiesAccount : FLOAT(10,2), repairAccount : FLOAT(10,2), salaryAccount : FLOAT(10,2), othersAccount : FLOAT(10,2), rentAccount : FLOAT(10,2)

8.1.4 Table: worker

workerID : UNSIGNED INT (Primary Key), name : VARCHAR(50), hallID : UNSIGNED INT (Foreign Key), email : VARCHAR(20) NULL, monthlySalary : FLOAT(10,2) NULL, dailyWage : FLOAT(10,2) NULL

8.1.5 Table: attendance

workerID : UNSIGNED INT (Primary Key), monthlyAttendance : UNSIGNED INT

8.1.6 Table: complaint

complaintID : UNSIGNED INT (Primary Key), studentID : UNSIGNED INT (Foreign Key), actionStatus : CHAR, description : BLOB

8.2 I/O Procedures & Interfaces

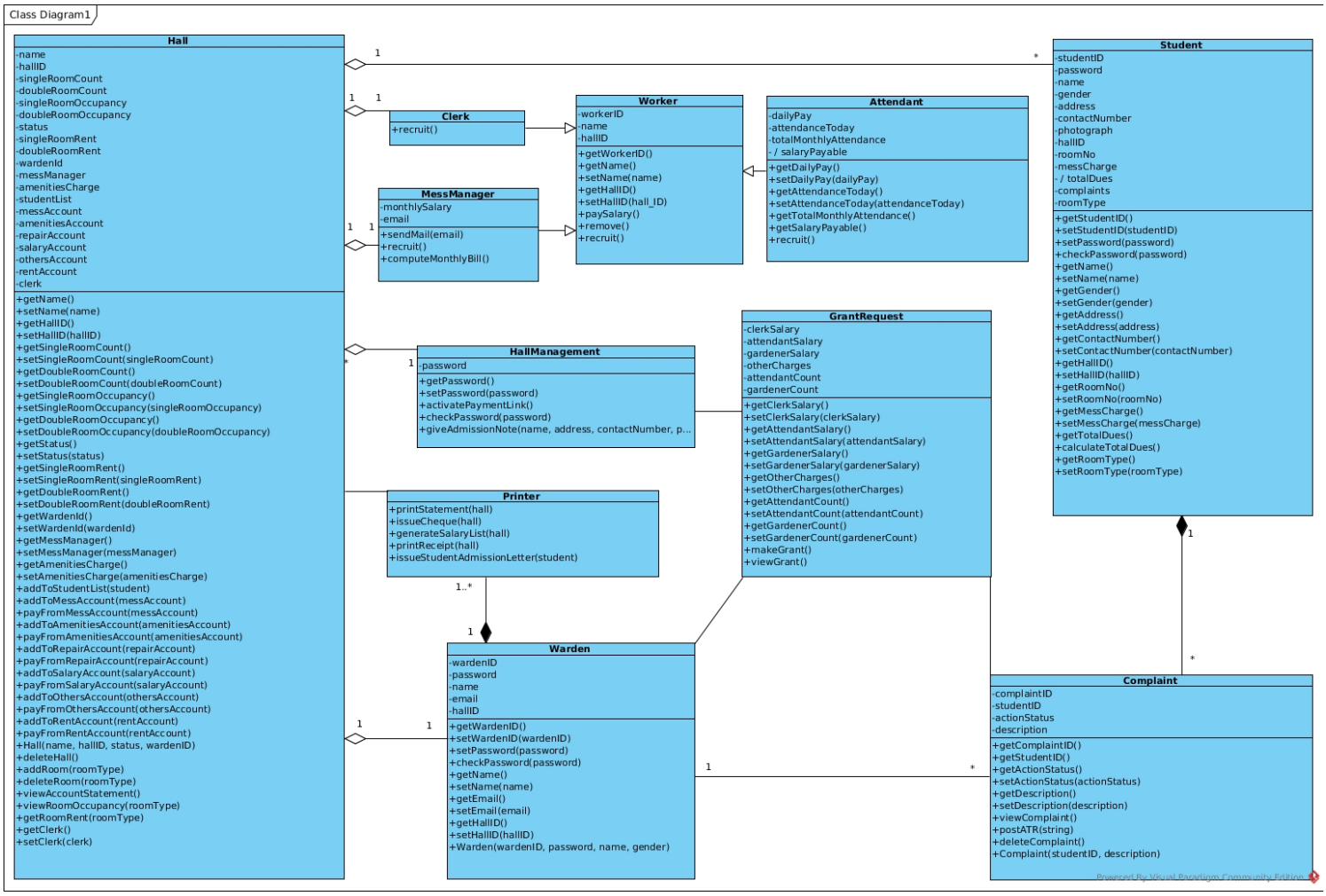
Across interfaces and components of the system, inputs will only be provided through the keyboard and via mouse clicks.

The outputs for various operations may be given on the screen, in the form of updated details, a confirmation message, a success/failure dialog box or the required data value. Certain operations, such as those involving printing of admission letters, printing account statements, cheques etc. will result in output in the form of a PDF file with the required information or a failure dialog box.

To navigate between parts of the interface or functionality tabs, mouse clicks or the keyboard shortcut Alt + Mnemonic_Key can be used. Inputs, depending on their type will be accepted through text fields, drop-down menus and radio buttons. Login ID and password will always be accepted through a text field and a specialized password field.

Output will be displayed in specific output areas that are part of the software interface or in the form of PDF documents.

8.3 Class Diagram in Target Language



8.4 Identified Reuse

Certain components of the software system have been identified for reuse. These will be implemented either using existing solutions or already written, available code and off-the-shelf components. These identified reuses are as follows:

8.4.1 Singleton Classes

For all the classes that are required to be singleton, a metaclass will be used for implementation as per the following example:

```
class Singleton(type):
    _instances = {}
    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            cls._instances[cls] = super(Singleton, cls).__call__(*args, **kwargs)
        return cls._instances[cls]


class Logger(object):
    __metaclass__ = Singleton
```

8.4.2 Model-View-Controller Design Pattern

To manage the calls to the database, the various queries made etc. the Model-View-Controller Design Pattern will be used.

- Controller would first get a query from the user. It would know that the query is for viewing a data field of a particular type of object. Accordingly it would choose the appropriate Model.
- If the query is for a particular data field, Controller calls the getter method of that specific Model, passing the objectID (hallID, studentID, wardenID, workerID etc.) as the argument, for returning the data field required.
- If the user query consists of a component name, Controller calls getter method of that specific Model, passing the component name as the argument, for returning the required data field list for the given component. Then the Controller displays the response to the user.

8.4.3 Inheritance in Classes using Worker

The Worker Class and its characteristics are designed to be inherited by the Clerk, MessManager and Attendant Classes. They share the same common attributes, such as, a worker ID, the Hall they work at, their name etc.

8.5 Coding Guidelines

8.5.1 Naming Convention

All sorts of naming in the code will be written as specified by the Google Python Style Guide.

`module_name`, `package_name`, `ClassName`, `method_name`, `ExceptionName`, `function_name`, `GLOBAL_CONSTANT_NAME`, `global_var_name`, `instance_var_name`, `function_parameter_name`, `local_var_name`

8.5.2 Coding Conventions

- Do not terminate lines with semicolons and do not use semicolons to put two commands on the same line.
- Maximum line length is 80 characters.
- Do not use parenthesis in return statements or conditional statements unless using parentheses for implied line continuation.
- Indent your code blocks with 4 spaces. Align wrapped elements vertically.
- Two blank lines between top-level definitions, one blank line between method definitions.
- No whitespace inside parentheses, brackets or braces. No whitespace before a comma, semicolon, or colon. Use whitespace after a comma, semicolon, or colon except at the end of the line.
- Surround binary operators with a single space on either side for assignment, comparisons, arithmetic operators and Booleans.
- Explicitly close files and sockets when done with them.
- Even a file meant to be used as a script should be importable and a mere import should not have the side effect of executing the script's main functionality. The main functionality should be in a main() function.

8.5.3 Coding Standards

All code will be written as per the Google Python Style Guide and Pylint and is covered under the GNU General Purpose License 2.0.

8.6 Library Usage

In addition to built-in types and the Python Standard Library, the following libraries will be used - MySQL Python Libraries, pyPDF2, AuthKit 0.4.5, pyUnit unittest Unit Testing Framework, pylint for debugging and warnings etc.

All other open source libraries covered by the MIT, GNU General Purpose or the Apache Software License may be used. Proprietary APIs, free for non-commercial use may also be used.