Design Document for University Applicant Determination and Messaging Transactor

University of Minnesota, Twin Cities

CSci5801: Software Engineering

Revision 2.6

05/02/2016

Prepared by Group #17

Krystal Lee 4339496

Qi Wu 5190447

Yuanchen Lu 4620983

Zixiang Ma 4644999

Document Revision History - Version 1

Rev	**Date**	**Author**	**Change Description**
1.0	03/31/2016	Krystal Lee	Document Creation
1.1	04/07/2016	Krystal Lee	Modified Document Format Added Scenarios: 5.1.1
1.2	04/07/2016	Qi Wu	Added class description: applicant
1.3	04/09/2016	Zixiang Ma	Added 1.1, 1.2, 1.3, 1.4 Added system architecture diagram
1.4	04/09/2016	Yuanchen Lu	Added a UML class diagram
1.5	04/09/2016	Krystal Lee	Added Scenarios: 5.1.2, 5.1.3, 5.1.4, 5.1.5, 5.1.6, 5.1.7 Added Sequence Diagrams for all current scenarios
1.6	4/10/2016	Zixiang Ma	Added 2.1, 2.2, 2.3, 2.4 Modified 1.1, 1.2, 1.3, 1.4 Added system environment diagram
1.7	04/10/2016	Krystal Lee	Added Scenarios: 5.1.8, 5.1.9, 5.1.10 Modified Scenarios: 5.1.2, 5.1.3, 5.1.4, 5.1.5, 5.1.6, 5.1.7 Added Sequence Diagrams for all current scenarios Modified UML Class Diagram
1.8	04/11/2016	Qi Wu	Added class description: user, application
1.9	4/11/2016	Zixiang Ma	Revised system architecture diagram Re-formating document Added 2.4 Revised Non-functional requirement
1.10	4/11/2016	Yuanchen Lu	Added 4.3.6
1.11	4/11/2016	Krystal Lee	Modified Document Format Modified Scenarios: All (updated descriptions) Modified Sequence Diagrams: 5.1.6, 5.1.2, 5.1.1 Modified UML Class Diagram Modified Class Descriptions: User, Applicant, Admin, Review Added: 3.1 and 3.2 including Interface Diagram

Document Revision History - Version 2

Rev	**Date**	**Author**	**Change Description**
2.0	04/19/2016	Krystal Lee	Document Creation
2.1	04/24/2016	Zixiang Ma Yuanchen Lu Qi Wu	Added design pattern descriptions Modified class descriptions
2.2	04/27/2016	Krystal Lee	Modified class diagram (Fig4): added design patterns
2.3	04/28/2016	Krystal Lee	Modified class diagram (Fig4): changed factory pattern
2.4	04/30/2016	Krystal Lee	Modified Document Format Modified class diagram (Fig4): changed interface classes Modified class, attribute, and method descriptions to match new class diagram. Added class diagrams for section 4 subsystems. Modified facade diagram (Fig3): added new classes Modified Scenarios and Sequence Diagrams: 5.1.1 to 5.1.10 Added Scenario and Sequence Diagram: 5.1.11
2.5	04/31/2016	Krystal Lee	Modified section 4.2 design patterns Modified class diagram (Fig 4): added/deleted methods Modified class descriptions: MessageTransactor, Filter, Waitlist, Department, ApplicationFactory, Application, Undergraduate, Graduate, Professional, ReviewBehavior, Faculty, FacultyReview, Admin, College, CollegeReview
2.6	05/02/2016	Krystal Lee	Modified Scenarios and Sequence Diagrams: 5.1.1, 5.1.4, 5.1.9, 5.1.10, 5.1.11 Modified class descriptions: College Modified structural diagrams: Fig 4 & Fig 4.1 Updated sections: 1, 2, 3, 6, 7
2.7	05/02/2016	Zixiang Ma	Re-formating Revised 2.2 2.3

Table of Contents

1 Introduction

1.1 Purpose

The purpose of the system design document is to present a detailed description of U-ADMT system mainly in respect of architecture, structure design, and dynamic design to support further software development process.

1.2 System Overview

The U-ADMT system is designed to facilitate university admission process. Specifically, it will deliver information among users in order to make the admission process more efficient. It does this by storing information and updating information about applications and reviews along with providing utility features such as filtering which allows users to easily access the data. There are four targeted user levels: applicant and users (college, department, and admin). For detailed information please refer to section 4.

1.3 Design Objectives

The U-ADMT system consist of 5 subsystems: Application, Account, Department, Filter, and Message Transactor. The following tables shows the general subsystem structure and classes, which are addressed in detail in section 4.

Subsystem Name	Classes

Application subsystem	1. **ApplicationFactory:** Factory that creates Applications
	2. **Application:** Stores application data, status, and Reviews
	    a. **Undergraduate:** Application type
	    b. **Graduate:** Application type
	    c. **Professional:** Application type
	3. **Review:** Stores scores, recommendations, and comments made by Faculty

Account subsystem	**1.** **Applicant:** Allows applicant to change final decision status on an Application **2.** **User:** Allows users to review and access Applications a. **Faculty:** Belongs to Departments and reviews Applications b. **Admin:** Belongs to Departments and can assign and review Applications c. **College:** Can review Applications, manage Departments, and mange Waitlists 3. **ReviewBehavior:** Strategies that determine how Users can review

	a. **FacultyReview:** Can create a Review to score an application, make a recommendation, and optionally leave a comment b. **AdminReview:** Can collate scores and take majority recommendation for Applications c. **CollegeReview:** Can change decision status for an Application
Department subsystem	1. **Department:** Stores information about different departments including waitlists, max and current seats, and minimum threshold values for screening 2. **Waitlist:** Stores information about the applications on a specific waitlist for a Department and Application type
Filter subsystem	1. **Filter:** Stores a criteria and returns Applications that matches
Message Transactor subsystem	**1. MessageTransactor:** Stores message information, and triggers Email using external service

Table 1: System Components

The Application subsystem attempts to store information and allow manipulation of a submitted application and all data accumulated during the review process pertaining to the status of the application. Most importantly it allows applications to be of different types (Undergraduate, Graduate, or Professional) which facilitates the ability for users to review applications based on different fields. It interacts heavily with the other subsystems and the U-ADMT interface as it is the primary data source for the system.

The Account subsystem attempts to store information about all the different user types (Applicant, Faculty, Admin, and College) and their main functionalities in U-ADMT. It notes that Applicants are different from other Users and separates them from Faculty, Admin, and College users which more actively participate in the review process and have their own reviewing behavior. It relies heavily on the other subsystems because it itself does not manipulate object data directly, but instead calls different object methods as needed.

The Department subsystem attempts to store information about each Department and its Waitlists including seat numbers, and screening criteria. It interacts mainly with College users who both create and manages the Department and Waitlist objects in the system. The Filter subsystem attempts to store all the information a user would need to filter applications. It is created by Users, and destroyed after the filter is applied. Finally the Message Transactor subsystem attempts to store all information a College user would need to send a message to an Applicant through the use of an external Email service. It is created by College users, and only interacts with them.

1.4 References

Group 17, "Requirements Based Test Cases for University Applicant Determination and Messaging Transactor". March 23, 2016.

Group 17, "Software Requirement Specification for University Applicant Determination and Messaging Transactor". March 23, 2016.

Group 17, "Use Case for University Applicant Determination and Messaging Transactor". March 23, 2016.

IEEE Std 830-1998, "IEEE Recommended Practice for Software Requirements Specifications". October 20, 1998.

1.5 Definitions, Acronyms, and Abbreviations

No.	Term	Definition
1	User	The person, or persons, who will interact directly with the product. In the context of this recommended practice the user may be members or applicants to the same University.
2	Application Data	The applicant information sent to the system from a shared database. Information is obtained from an external application software, and contains the following information: ● Personal Info ● Application Info ○ Desired College ○ Desired Department ○ Undergraduate/Graduate/Professional Status ● Educational Background ● Test Scores ● Essays ● Recommendations ● Awards ● Financial Status ● Language
3	Application	An application is made by the system for use by its users. It includes the application data, application type, application number, application status, score, application recommendation, and any comments made.
4	Application Type	An application can be for undergraduates, graduates, or professionals.
5	Application Status	An indicator showing application process. It can be undecided, accepted, rejected, waitlisted, or confirmed. All users with access to an application can see the application status.

6	Score	A score is associated with an application, and used as one of the evaluation criteria made by department level users. It is based on individual criteria scores. College and department users can filter and rank applications by score.
7	Application Recommendation	An evaluation made by faculty level user including accept, reject, or waitlist. It can be overridden by higher level users.
8	Comment	Additional information that users optionally provide for each application to explain how the application recommendation is made. Users may reply to other users comments to explain things like why a recommendation was overrided.
9	Waitlist	A collection of applications that can only be accepted when more opening becomes available.
10	Auto-Reject	A system feature that speeds the automatic removal those who fail to meet the threshold values.
11	Response Deadline	A deadline set by college level users for applicant level users. It is the deadline that applicant level users have to change an application status of accepted to rejected or confirmed.
12	Review Deadline	A deadline set by administrator level users for faculty level users. It is the deadline that faculty users have to complete reviewing by.
13	Acceptance Threshold	A threshold set by college level users for the number of applications allowed to have a status of accepted for each department. In other words, the maximum number of applicant level users to allow.

Table 2. Definitions

No.	Abbreviation	Full name
1	GPA	Grade Point Average
2	GRE	Graduate Record Examinations
3	SAT	Scholastic Aptitude Test
4	ACT	American College Testing
5	U-ADMT	University Applicant Determination Message Transactor

Table 3. Abbreviations/Acronyms

2 Design Overview

2.1 Introduction

Object oriented design methodology is utilized in the U-ADMT design process. It uses a factory design pattern for the Application subsystem allowing for Applications to be created with different application types while hiding the actual creation logic from U-ADMT. It also uses a strategy pattern for the User subsystem which encapsulates the reviewing behavior of users allowing for easy modification of behaviors used by the subclasses of User. These design patterns are further expanded on in Section 4.2.

The system is made of five conceptual classes: user, application, filter, and department, and they are architected based on a shared data repository in order to favor data-sharing operation.

2.2 Environment Overview

The U-ADMT system is designed for four levels of users and they all need to interact with U-ADMT through a graphical user interface system (see Figure 1). U-ADMT accepts valid user requests from all level users and stores necessary information in the data repository. To access the database and other external components including email, application and account services, U-ADMT needs to go through an external interface for higher maintainability purposes.

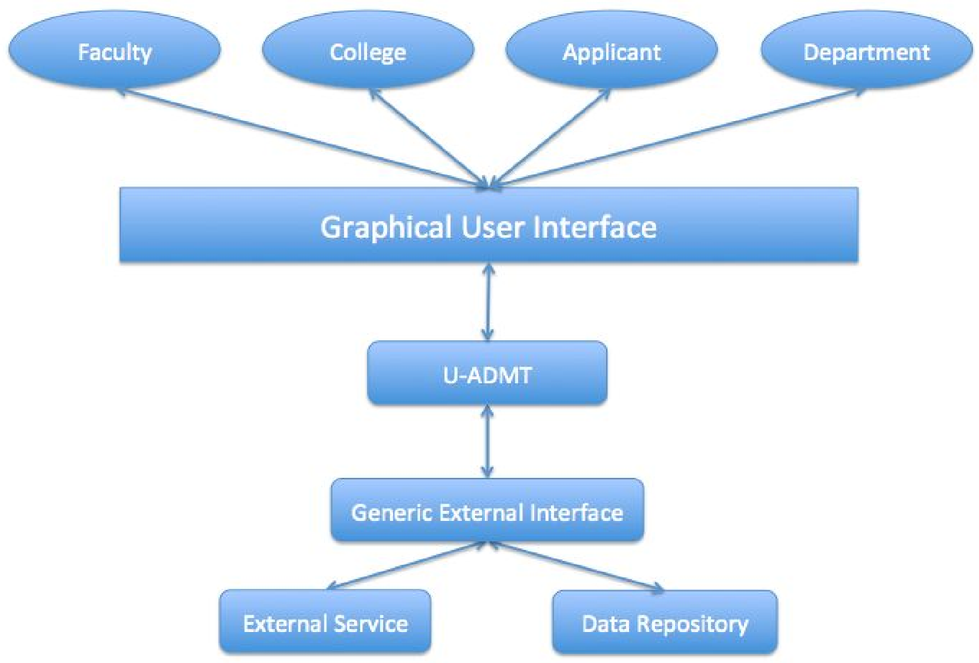

Figure 1: U-ADMT Environment Diagram

2.3 System Architecture

U-ADMT system follows the repository architecture where a shared central repository (see Figure 2) contains all data which can be directly accessed by all other components. The Users component is part of the U-ADMT system and in this diagram represents the classes in the Account subsystem, which is specified in section 4.3. Email, account and application services are considered external services and they also have direct access to the shared data repository.

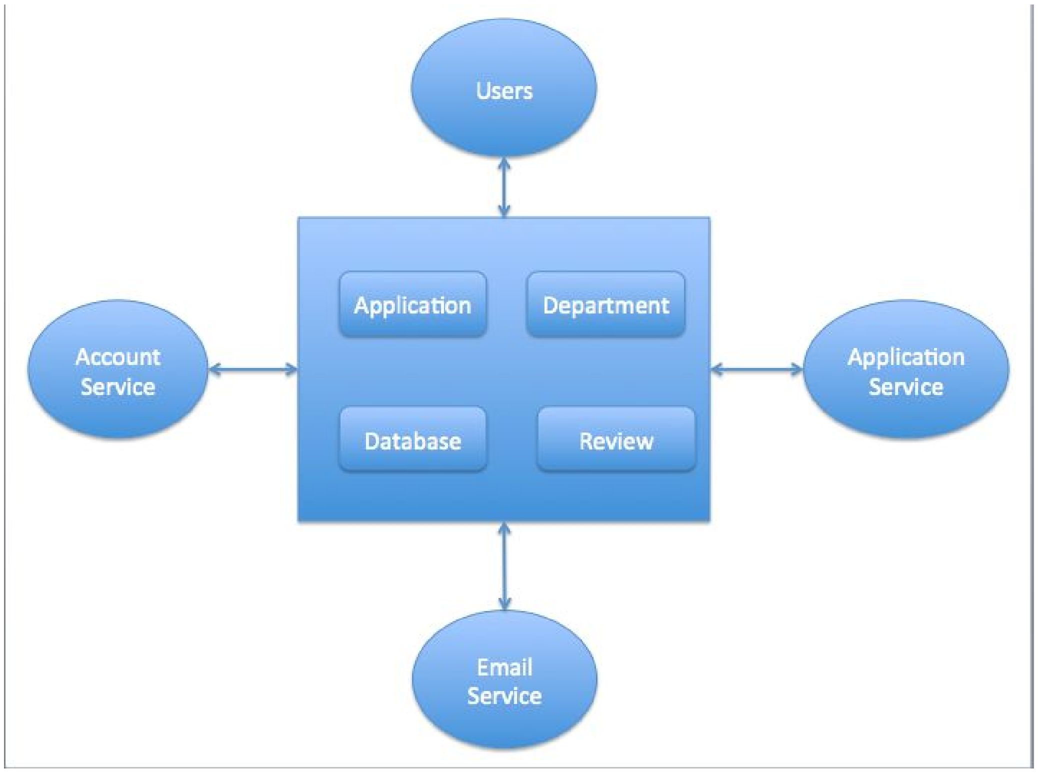

Figure 2: U-ADMT System Architecture

2.4 Constraints and Assumptions

The U-ADMT system is directly constrained by information safety consideration and reliability requirements. Each level user has specified access privilege to information, and system should deny access when user requests unauthorized information. The system should be capable of handling a large amount of users simultaneously. Reliability requirement is also depends on the quality of database design and other external components such as email and account services.

The U-ADMT system is designed and will be operated based on the following assumptions:

- Accounts will be created by users and information on them will be stored in Account service
- Applications are all submitted and data on them will be stored in Application service
- All Applications are submitted with complete and valid data
- Department, major, and number of position openings are defined and already exists in the system

3 Interfaces and Data Stores

3.1 System Interfaces

U-ADMT mainly interacts with 3 other external systems (Application Service, E-mail Service, and Account Service). It exports and imports data to these different services. Since U-ADMT is a complex system it uses a facade, U-ADMT Repository interface, to interact with the external services. Users of the system can access it through a Graphical User Interface (GUI) that interacts with the facade. See **Figure 3** for diagram of the interactions.

3.1.1 Graphical User Interface (GUI)

This interface is used to facilitate user interaction with the system. It interacts with other systems through the U-ADMT Repository interface, and data can be accessed by the software at a fixed memory location.

3.1.2 U-ADMT Repository Interface

This interface is used to review applications and do message transacting. A GUI is provided for users to interact with this repository, and data can be accessed by a software at a fixed memory location in a shared database. This interface acts a facade to more complex system details. See **Figure 3** for more details on what methods are called by this interface to interact with the GUI.

3.1.3 Account Service

This interface is used to create and store information about user accounts such as username, password, account level, and e-mail. A GUI is provided for users to interact with this service allowing users to create and modify the account data that U-ADMT uses. This data is used by User classes in U-ADMT, and the data can be accessed by the software at a fixed memory location in a shared database.

3.1.4 Application Service

This interface is used to create and store information about submitted applications such as grades and test scores. A GUI is provided for applicants to interact with this service allowing applicants to create and submit application data that then gets reviewed by U-ADMT. The application data can be accessed by the software at a fixed memory location in a shared database.

3.1.5 Email Service Interface

This interface is used to send emails for the messaging transactor of U-ADMT. It stores information about email content which can be accessed by the software at a fixed memory location in a shared database.

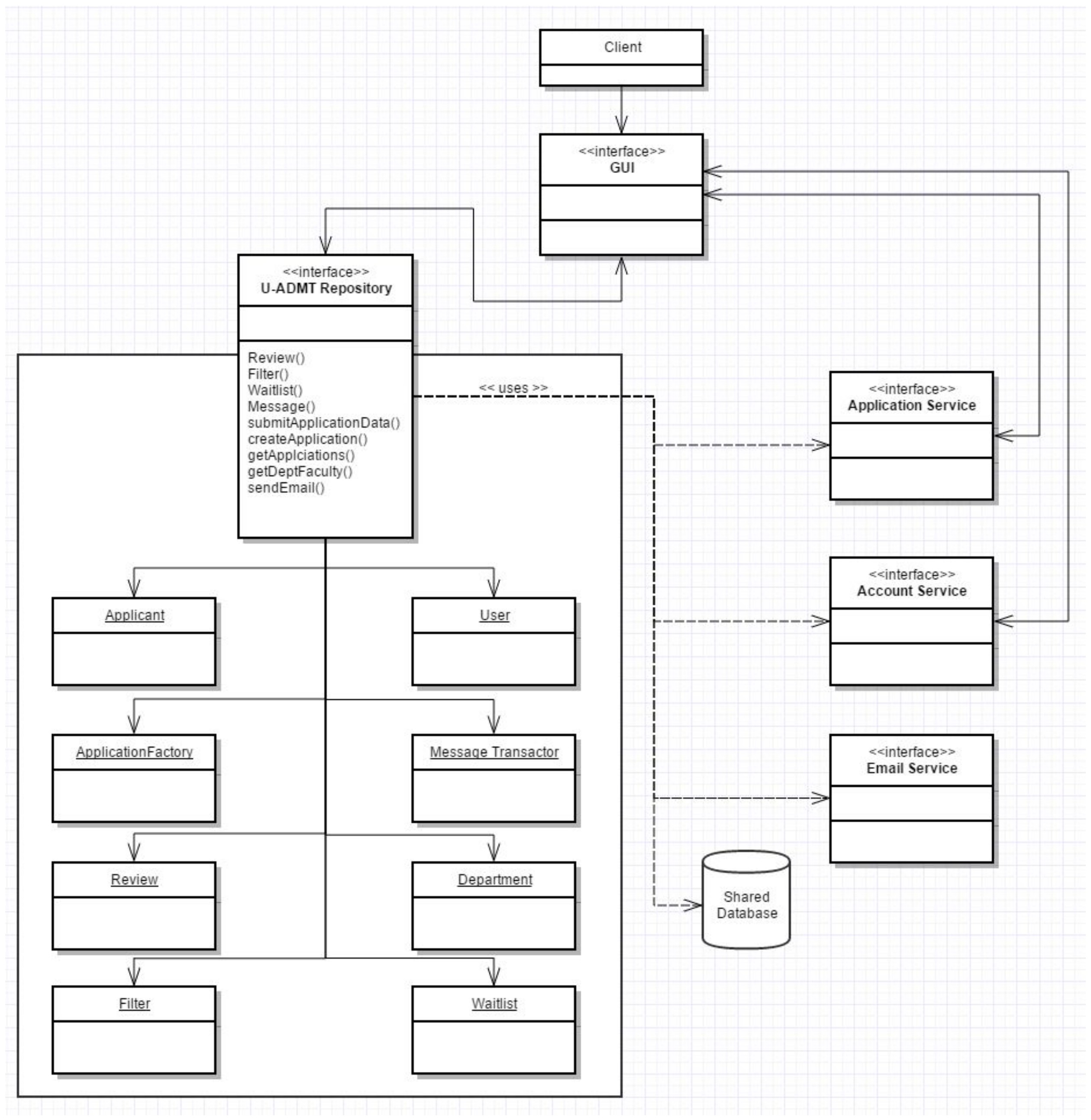

Figure 3: Facade Interface (larger version as Figure3.png)

3.2 Data Stores

U-ADMT creates some data stores as part of the system to help with the review and messaging process. It stores the data in the shared database.

3.2.1 Applications & Reviews

Not only does the system have to keep track of the application data an applicant submits, but also what type of application it is (undergraduate, graduate, or professional), its status in the review process, and the reviews associated with it. Therefore the system must store additional data for applications on top of that received from the external application service including data on the reviews made by users.

3.2.2 Messages

Additionally the system facilitates message transaction through the aid of an email service. The system stores the messages.

3.2.3 Departments

Applications are for specific departments in the university (such as CSE Undergraduate or CLA Professional). Additionally, faculty and administrator level users review applications based on their departments, and if a department is filled an applicant level user can’t be admitted. Therefore it is needed for the system to store data on each department.

3.2.4 Waitlists

For each department there is a waitlist for after it gets filled. Applications can be added and removed from the waitlist as time goes on. So the system must have data stored for each of these created waitlists.

4 Structural Design

4.1 Class Diagram

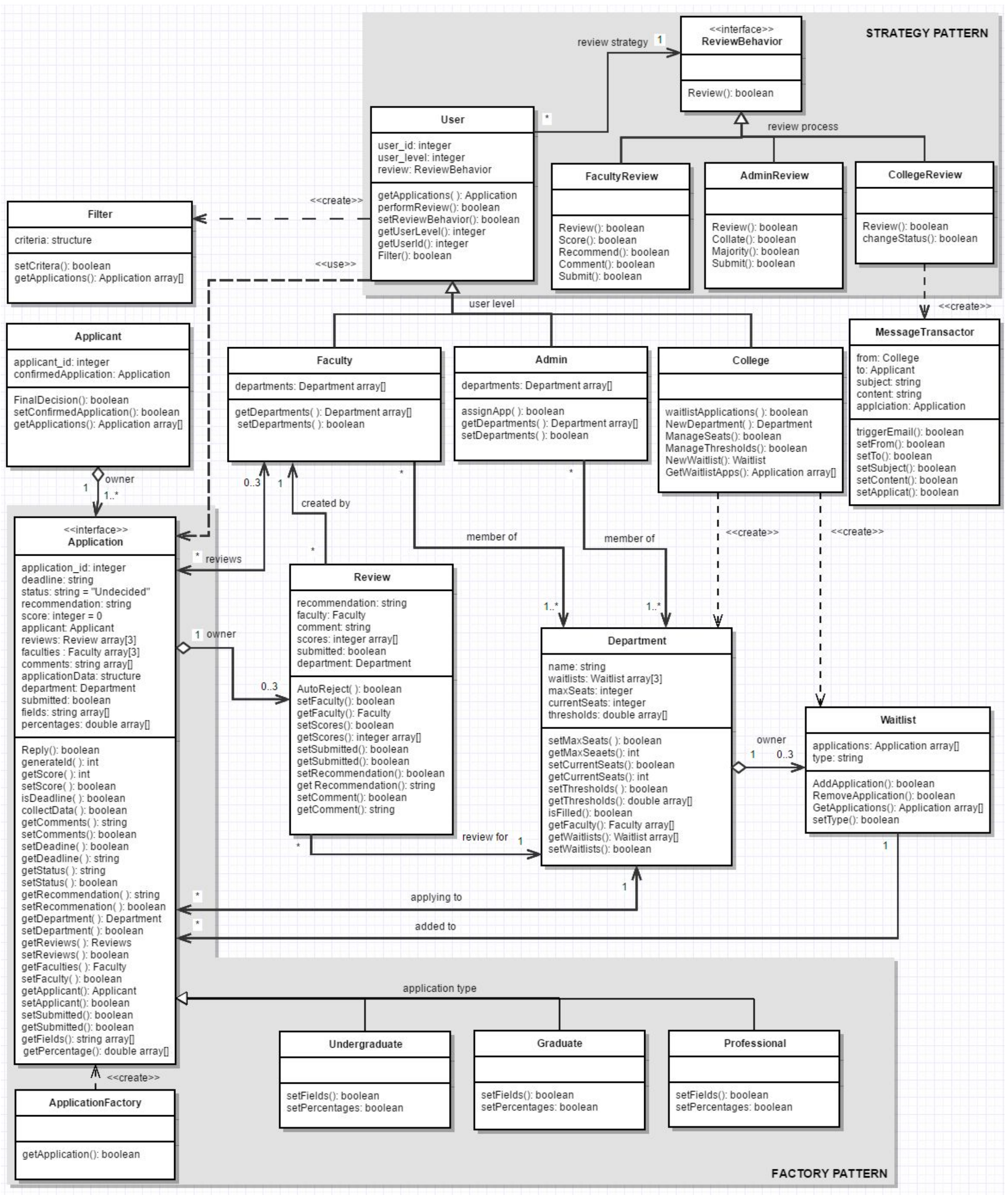

Figure 4: Class Diagram (larger version as Figure4.png)

4.2 Subsystem Design Patterns

4.2.1 Factory Pattern - Application Subsystem

Factory pattern, as a creation pattern, is utilized to create Application objects. The ApplicationFactory class uses internal logic to encapsulate the creation of different Application types. The ApplicationFactory class is used by the U-ADMT Repository interface (Figure 3) to create Applications that U-ADMT can use (Scenario 5.1.1 and Figure 5.1). The factory pattern allows the subclasses (Undergraduate, Graduate, or Professional) decide which type of Application is created. Additionally, using ApplicationFactory to create Applications hides the actual creation logic from U-ADMT Repository. This way if we want to eventually change the Application creation process, or the types of Applications available we can simply replace the Application interface with another one.

4.2.2 Strategy Pattern - User Subsystem

Strategy pattern is utilized for the User subsystem. In strategy pattern, the behavior of a class can be changed at run time allowing creation of objects which use different strategies to show different behaviors. In the User subsystem, each subclass (Admin, College, or Faculty) may vary in terms of their reviewing behavior. Therefore by using strategy pattern with the ReviewBehavior class we encapsulate the review strategies that the different Users use. This means that if we want to change the way one of the Users reviews we only need to change one of the subclasses, and don’t actually have to directly modify the User class or subclasses. Additionally if we wanted to add more types of review strategies we only need to add to the ReviewBehavior interface.

4.3 Class Descriptions

4.4 Classes in the Account Subsystem

The Account subsystem has two main classes: User and Applicant. The Applicant class allows an Applicant to access and make final status decisions on their Applications (Scenario 5.1.7 and Figure 5.7). The User class has three subclasses (Faculty, Admin, and College). Users are granted the ability to filter Applications, and more importantly, review Applications. Each of the different User subclasses has their own reviewing behavior. Thus a strategy pattern is used to define the different behaviors via a ReviewBehavior interface. Faculty users use FacultyReview behavior to create reviews and score, recommend, and comment on Applciations (Scenario 5.1.4 and Figure 5.4). Admin users use AdminReview behavior to collate scores and take majority recommendations on Applications based on their three Reviews (Scenario 5.1.5 and Figure 5.5). Finally, College users use CollegeReview behavior to change the status on Applications (Scenario 5.1.6 and Figure 5.6). College users also have the ability to manage Departments (Scenario 5.1.10 and Figure 5.10) and Waitlists (Scenarios 5.1.9 and 5.1.11 and Figures 5.9 and 5.11).

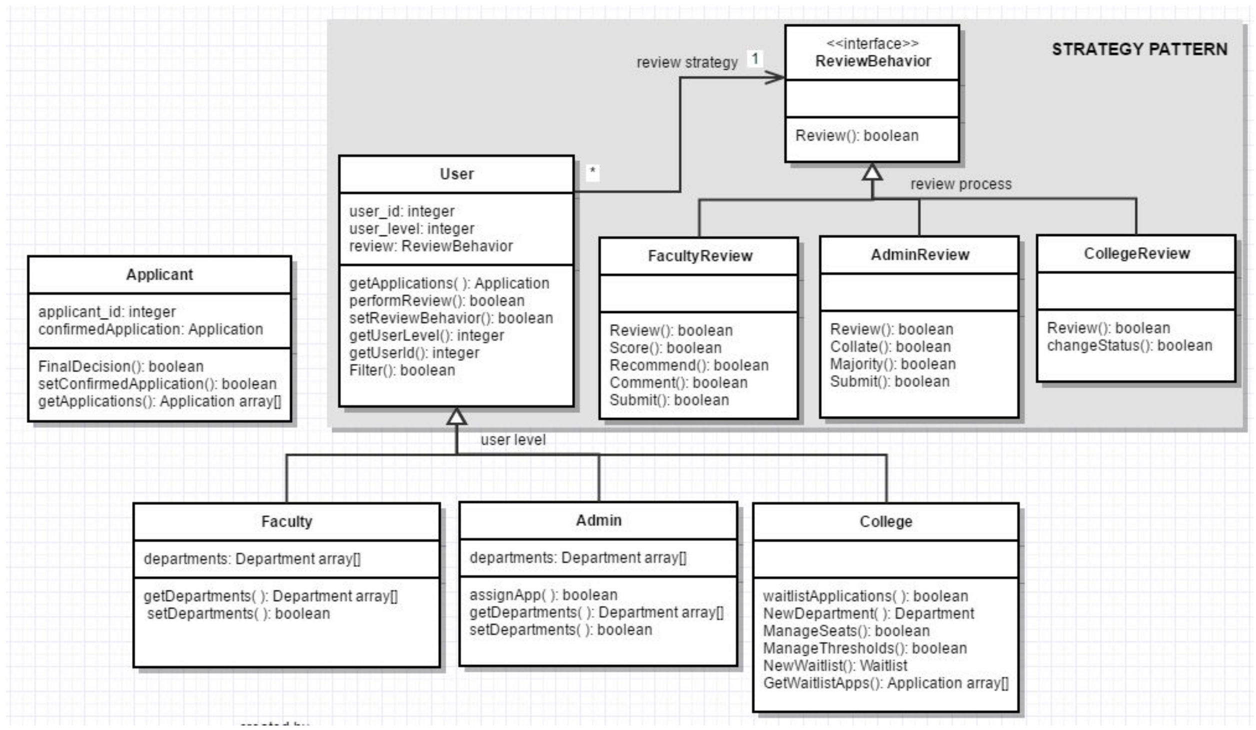

Figure 4.1: Account Subsystem Class Diagram (larger version as Figure4.1.png)

4.4.1 Class: User

- Purpose: To model the relevant aspects of the User
- Constraints: None
- Persistent: No (created at system initialization from other available data)

4.4.1.1 Attribute Descriptions

1. Attribute: user_id
   Type: integer
   Description: Stores the id of the current User
   Constraints: Should be a positive value

2. Attribute: user_level
   Type: integer
   Description: Stores the level of the current User
   Constraints: Should be in the range from 1 to 3 where 1 is Faculty level, 2 is Admin level, and 3 is College level

3. Attribute: review
   Type: ReviewBehavior
   Description: Stores the reviewing behavior of the current User
   Constraints: None

4.4.1.2 Method Descriptions

1. Method: Filter(structure criteria)
   Return Type: list of Applications
   Parameters: criteria - the structure containing the criteria for the filter
   Return value: List of Applications that satisfies the criteria and can be accessed by current User
   Pre-condition: None
   Post-condition: None
   Attributes read/used: user_level
   Methods called:Filter. Filter.setCriteria(), Filter.getApplications().

   Processing logic: Create a Filter object, and use setCritera of Filter object to input criteria then get the output from the Filter object after using getApplications.

   Test case 1: Call Filter with criteria X.
   Expected output is: All applications that satisfies the criteria and can be accessed by current user are returned.

2. Method: getApplications()
   Return Type: list of Applications
   Parameters: None
   Return value: All Applications the current user can access or NULL
   Pre-condition: None
   Post-condition: None
   Attributes read/used: user_level
   Methods called: None

Processing logic:
Use U-ADMT Repository interface to search shared database for Applications that the current user_level can access. Then return all matching applications, otherwise, return NULL.

Test case 1: Call getApplication with correct application_id X, and user_level 1 (faculty).
Expect output is the the application assigned to the current faculty.
Test case 2: Call getApplication with correct application_id X, and user_level 2 or 3 (admin or college).
Expect output is all the application in the current college .
Test case 3: Call getApplication with application_id X which does not exist in the database.
Expect output is NULL.

3. Method: performReview(Application app)
   Return Type: boolean
   Parameters: app - the Application that the User wants to review
   Return value: Success or Failure
   Pre-condition: app is a valid Application
   Post-condition: None
   Attributes read/used: review
   Methods called:Filter. review.Review()

   Processing logic: Use review attribute to get the ReviewBehavior of the User, and then call Review(). The actual review process will differ depending on the ReviewBehavior. If Review() returns Success then return Success. Otherwise return Failure

   Test case 1: Call performReview on a Faculty User with Application X.
   Expected output is: FacultyReview.Review() is performed on X.

4. Method: setReviewBehavior(ReviewBehavior new_behavior)
   Return Type: boolean
   Parameters: new_behavior - the ReviewBehavior to set review
   Return value: Success or Failure
   Pre-condition: new_behavior is a valid ReviewBehavior
   Post-condition: None
   Attributes read/used: review
   Methods called: None

   Processing logic: Set the attribute review to contain the input new_behavior and return Success. Otherwise return Failure.

   Test case 1: Call setReviewBehavior with ReviewBehavior X.
   Expected output is: review attribute is set to X and return Success.

5. Method: getUserLevel()
   Return Type: integer
   Parameters: None
   Return value: user_level, the attribute containing the current set integer, or NULL
   Pre-condition: user_level should be between 1 and 3
   Post-condition: None
   Attributes read/used: user_level
   Methods called: None

   Processing logic: If the attribute user_level exists, return it. Otherwise return NULL.

   Test case 1: Call getUserLevel where User has a user_level attribute = x.
   Expected output is: user_level attribute.
   Test case 2: Call getUserLevel where User does not have a user_level attribute.
   Expected output is: NULL

6. Method: getUserId()
   Return Type: integer
   Parameters: None
   Return value: user_id, the attribute containing the current set integer, or NULL
   Pre-condition: None
   Post-condition: None
   Attributes read/used: user_id
   Methods called: None

   Processing logic: If the attribute user_id exists, return it. Otherwise return NULL.

   Test case 1: Call getUserId where Application has a user_id attribute = x.
   Expected output is: user_id attribute.
   Test case 2: Call getUserId where Application does not have a user_id attribute.
   Expected output is: NULL

4.4.2 Class: ReviewBehavior

· Purpose: To model the relevant aspects of the ReviewBehavior
· Constraints: Interface used by User
· Persistent: No (created at system initialization from other available data)

4.4.2.1 Attribute Descriptions

4.4.2.2 Method Descriptions

1. Method: Review(Application app)

Return Type: boolean
Parameters: app - the Application that the User wants to review
Return value: Success or Failure
Pre-condition: app is a valid Application, and Review should be called by performReview
Post-condition: None
Attributes read/used: None
Methods called: None

Processing logic: The actual review process will differ depending on the User subclass. The default version of Review simply returns Success without changing anything.

Test case 1: Call Review with Application X.
Expected output is: Success.

4.4.3 Class: FacultyReview

· Purpose: To model the relevant aspects of the FacultyReview
· Constraints: subclass of ReviewBehavior
· Persistent: No (created at system initialization from other available data)

4.4.3.1 Attribute Descriptions

4.4.3.2 Method Descriptions

1. Method:Review(Application app)
   Return Type:boolean
   Parameters: app - the Application that the User wants to review
   Return value: Success or Failure
   Pre-condition: app is a valid Application with less than 3 Reviews in attribute reviews
   Post-condition: None
   Attributes read/used: app.department, department.thresholds
   Methods called: app.getDepartment(), department.getThresholds(), review.AutoReject(), Score(), Comment(), Recommend(), and Submit()

   Processing logic: Create a review object, getDepartment from Application and get Thresholds from said Department. If the application data cannot satisfy the threshold, give the option to call review.AutoReject() to set scores to 0 and recommendation to “Reject”. Else give the options for the User to review app by calling Score(), Recommend(), and Comment(). Finally give the user the option to submit the review, and if so call Submit(). If all the processes go without failure return Success else return Failure.

   Test case 1: Call review with application X, score Y, recommendation Z and comment C(satisfy threshold).

Expected output is: a new Review with score Y and recommendation Z and comment C is added to the application X review list.
Test case 2: Call review with application X, score Y, recommendation Z and comment C(not satisfy threshold).
Expected output is: a new Review with score 0 and recommendation rejected and comment C is added to the application X review list.

2. Method: Score(Review rev)
   Return Type: boolean
   Parameters: rev - the Review that the User wants to score
   Return value: Success or Failure
   Pre-condition: rev is a valid Review
   Post-condition: None
   Attributes read/used: rev.scores
   Methods called: rev.getScores(), rev.setScores()

   Processing logic: Get the scores for rev using rev.getScores(), and then allow the User to update the scores with integer values using rev.setScores(integer new_score). If the User doesn't modify any scores or setScores returns Success then return Success. Otherwise return Failure.

   Test case 1: Call Score on Review X and update the second score to 25.
   Expected output is: Value of integer X.scores[1] = 25 and return Success.

3. Method: Recommend(Review rev)
   Return Type: boolean
   Parameters: rev - the Review that the User wants to recommend
   Return value: Success or Failure
   Pre-condition: rev is a valid Review
   Post-condition: None
   Attributes read/used: rev.recommendation
   Methods called: rev.getRecommendation(), rev.setRecommendation()

   Processing logic: Get the recommendation for rev using rev.getRecommendation(), and then allow the User to update the recommendation with a string value that must be either “Accept”, “Waitlist”, or “Reject” using rev.setRecommendation(string new_rec). If the User doesn't modify the recommendation or setRecommendation returns Success then return Success. Otherwise return Failure.

   Test case 1: Call Recommend on Review X and update the Recommendation to “Waitlist”.
   Expected output is: Value of string X.Recommendation = “Waitlist” and return Success.

4. Method: Comment(Review rev)
   Return Type: boolean

Parameters: rev - the Review that the User wants to comment on
Return value: Success or Failure
Pre-condition: rev is a valid Review
Post-condition: None
Attributes read/used: rev.comment
Methods called: rev.getComment(), rev.setComment()

Processing logic: Get the comment for rev using rev.getComment(), and then allow the User to update the comment with a new string using rev.setComment(string new_comment). If the User doesn't modify the comment or setComment returns Success then return Success. Otherwise return Failure.

Test case 1: Call Comment on Review X and update the comment to "hello".
Expected output is: Value of string X.comment = "hello" and return Success.

5. Method: Submit(Review review, Application application)
   Return Type: boolean
   Parameters: review - Review to submit
   Application - the Application where review is added
   Return value: Success or Failure
   Pre-condition: review is a valid Review, and application is a valid Application with fewer than 3 objects in review
   Post-condition: review is added to reviews attribute of application
   Attributes read/used: review.submitted, application.reviews
   Methods called: review.setSubmitted(), application.setReviews()
   Processing logic: Use review.setSubmitted to set reivew submitted attribute to true. Then use application.setReviews to add the submitted review to the review list. If all operation success, return Success. Else return Failure.

   Test case 1: Call Submit with application X and Review Y.
   Expected output is: Review Y submitted attribute is set to True. Review Y is added to the reviews attribute in X.

4.4.4 Class: AdminReview

· Purpose: To model the relevant aspects of the AdminReview
· Constraints: subclass of ReviewBehavior
· Persistent: No (created at system initialization from other available data)

4.4.4.1 Attribute Descriptions

4.4.4.2 Method Descriptions

1. Method: Review(Application application)

Return Type: boolean
Parameters: application - Applciation object to be reviewed
Return value: Success or Failure
Pre-condition: None
Post-condition: None
Attributes read/used: application.deadline, review.submitted
Methods called: Collate(), Majority(), Submit(), application.getDeadline(), application.getReviews(), review.AutoReject()

Processing logic: First, get the deadline of the given application, if it is not pass deadline, return Failure. Else get reviews for the Application, if one of the review submitted = false call AutoReject() on the review. Else call Collate() and Majority() to set score and recommendation of the application If all operations are success, return Success, otherwise return Failure.

Test case 1: Call Review with application X with 3 valid reviews and it is past the deadline.
Expected output is: score and recommendation is calculated and return Success.

2. Method: Collate(Application application)
Return Type: boolean
Parameters: application - application to be collated
Return value: Success or Failure
Pre-condition: application has been assigned to 3 faculty
Post-condition: application score has been updated
Attributes read/used: application.reviews, application.department, review.scores
Methods called: application.setScore(), application.getReviews(), review.getScores()

Processing logic: Get Reviews from the Application reviews attribute. Then for each Review, get the scores attribute. Calculate the total scores for the review based on the Application fields and percentages. If a total score is 0, ignore it. Take the average of all scores that aren't 0, and set it as the score attribute of the Application.

Test case 1: Call Collate with Application X.
Expected output is:application X is set to the average score of reviews.

3. Method: Majority(Appplication application)
Return Type:boolean
Parameters: application - application to do majority
Return value: Success or Failure
Pre-condition: application has been assigned to 3 faculty
Post-condition: application recommendation has been updated to "Accept", "Reject", or "Waitlist"
Attributes read/used: application.reviews, review.recommendation

Methods called: application.setRecommendation(), application.getReviews(), review.getRecommendation()

Processing logic: Get Reviews from the Application reviews attribute. Then for each Review, get the recommendation attribute. Set the Application recommendation to the recommendation which appears most times. If there is no majority set recommendation to "Waitlist". Return success if all these can be done. Otherwise return failure.

Test case 1: Call Majority with application X.
Expected output is: the recommendation of X is set to the majority recommendation.

4. Method: Submit(Application application)
   Return Type:boolean
   Parameters: application - application to be submitted
   Return value: Success or Failure
   Pre-condition: application has valid score and recommendation values
   Post-condition: submitted is set to "True"
   Attributes read/used: application.submitted
   Methods called: application.setSubmitted()

   Processing logic: Check if the application recommendation and score is set. If recommendation and score is set, set application attribute submitted to True and return Success. Else remain attribute submitted unchanged and return Failure.

   Test case 1: Call submit with application X which score and recommendation is set.
   Expected output is: submitted attribute of the given application is set to true and return success.
   Test case 2: Call submit with application X which score and recommendation is not set. Expected output is: submitted attribute is not changed and return failure.

4.4.5 Class: CollegeReview

- Purpose: To model the relevant aspects of the CollegeReview
- Constraints: subclass of ReviewBehavior
- Persistent: No (created at system initialization from other available data)

4.4.5.1 Attribute Descriptions

4.4.5.2 Method Descriptions

1. Method: Review(Application application)
   Return Type: boolean
   Parameters: application - Applciation object to be reviewed
   Return value: Success or Failure
   Pre-condition: None

Post-condition: None
Attributes read/used: application.deadline, messageTransactor.to, messageTransactor.from, messageTransactor.application, messageTransactor.subject, messageTransactor.content
Methods called: changeStatus(), application.setDeadline(), application.getApplicant(), messageTransactor.setTo(), messageTransactor.setFrom(), messageTransactor.setApplication(), messageTransactor.setSubject(), messageTransactor.setContent(), messageTransactor.triggerEmail()

Processing logic: First, get the deadline of the given application, if it is not pass deadline, return Failure. Else get reviews for the Application, if one of the review submitted = false call AutoReject() on the review. Else call Collate() and Majority() to set score and recommendation of the application If all operations are success, return Success, otherwise return Failure.

Test case 1: Call Review with application X with 3 valid reviews and it is past the deadline.
Expected output is: score and recommendation is calculated and return Success.

2. Method: changeStatus(Application application)
Return Type: boolean
Parameters: application - Application that status wants to change
Return value: Success or Failure
Pre-condition: application has a valid value for recommendation attribute
Post-condition: application status attribute = recommendation attribute or “Waitlisted”
Attributes read/used: application.recommendation
Methods called: application.getRecommendation(), application.getDepartment(), application.setStatus()

Processing logic: First, get recommendation and Department of the Application. If recommendation is “Accept”, and Department is filled. Then set status to “Waitlisted”. If recommendation is “Accept” and Department is not filled, status set to “Accepted”. If recommendation is not “Accept”, set status to match the recommendation. If all operation is success, return Success. Otherwise, return Failure.

Test case 1: Call changeStatus with application X and recommendation “Accept” with full Department.
Expected output is: status is set to “Waitlisted”.
Test case 2: Call changeStatus with application X and recommendation “Accept” with Department that is not filled.
Expected output is: status is set to “Accepted”.
Test case 3: Call changeStatus with application X and recommendation “Reject”
Expected output is: status is set to “Rejected”.

4.4.6 Class: Admin

· Purpose: To model the relevant aspects of the Admin
· Constraints: subclass of User
· Persistent: No (created at system initialization from other available data)

4.4.6.1 Attribute Descriptions

1. Attribute: departments
   Type: list of Departments
   Description: Departments the Admin is a part of (can be more than one since can have something like CSE and CBS, etc.)
   Constraints: None

4.4.6.2 Method Descriptions

1. Method: setDepartments(Department new_department)
   Return Type: boolean
   Parameters: new_department - the Department to add to departments
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: departments now includes new_department
   Attributes read/used: departments
   Methods called: None

   Processing logic: Set the attribute departments to contain the input new_department and return Success. Otherwise return Failure.

   Test case 1: Call setDepartments with Department X.
   Expected output is: departments attribute is set to contain X and return Success.

2. Method: getDepartments()
   Return Type: list of Departments
   Parameters: None
   Return value: departments, the attribute containing the current set list of Departments, or NULL.
   Pre-condition: None
   Post-condition: None
   Attributes read/used: departments
   Methods called: None

   Processing logic: If the attribute departments exists, return it. Otherwise return NULL

   Test case 1: Call getDepartments where Admin has a departments attribute = x.
   Expected output is: departments attribute.

Test case 2: Call getDepartments where Admin does not have a departments attribute.
Expected output is: NULL.

3. Method: assignApps(Application application)
Return Type :boolean
Parameters: application - the Application to be assigned for review
Return value: Success or Failure
Pre-condition: application exists
Post-condition: application has Faculty user assigned to it
Attributes read/used: application.department
Methods called: application.getDepartment(), department.getFaculty(), application.getFaculties(), application.setFaculties(), application.setDeadline().

Processing logic: Get the Application Department, and all the Facilities in that Department. Set one of the Faculty objects in Application's facilities attribute to one of the Faculty members in that Department as long as there are less than 3 objects in facilities. Set the Application deadline, and return Success. Else return Failure.

Test case 1: Call AssignApps with application X.
Expected output is: X is assigned to a faculty Y where Y is some member of the Department X is applying to.

4.4.7 Class: Faculty

· Purpose: To model the relevant aspects of the Faculty
· Constraints: subclass of User
· Persistent: No (created at system initialization from other available data)

4.4.7.1 Attribute Descriptions

1. Attribute: departments
Type: list of Departments
Description: Departments the Faculty is a part of (can be more than one since can have something like CSE and CBS, etc.)
Constraints: No.

4.4.7.2 Method Descriptions

1. Method: setDepartments(Department new_department)
Return Type: boolean
Parameters: new_department - the Department to add to departments
Return value: Success or Failure
Pre-condition: None
Post-condition: departments now includes new_department
Attributes read/used: departments

Methods called: None

Processing logic: Set the attribute departments to contain the input new_department and return Success. Otherwise return Failure.

Test case 1: Call setDepartments with Department X.
Expected output is: departments attribute is set to contain X and return Success.

2. Method: getDepartments()
Return Type: list of Departments
Parameters: None
Return value: departments, the attribute containing the current set list of Departments, or NULL.
Pre-condition: None
Post-condition: None
Attributes read/used: departments
Methods called: None

Processing logic: If the attribute departments exists, return it. Otherwise return NULL

Test case 1: Call getDepartments where Faculty has a departments attribute = x.
Expected output is: departments attribute.
Test case 2: Call getDepartments where Faculty does not have a departments attribute.
Expected output is: NULL.

4.4.8 Class: College

· Purpose: To model the relevant aspects of the College
· Constraints: subclass of User
· Persistent: No (created at system initialization from other available data)

4.4.8.1 Attribute Descriptions

4.4.8.2 Method Descriptions

1. Method:waitlistApplications(Application array[] apps)
Return Type: boolean
Parameters: apps - list of Applications to waitlist
Return value: Success or Failure
Pre-condition: apps is a valid list of Applications with corresponding valid Departments and Waitlists
Post-condition: All the status of apps is changed to “Waitlisted”, and the Waitlist of the corresponding Application is updated with the Application object.
Attributes read/used: application.department, department.waitlists, application.score

Methods called: application.getDepartment(), department.AddToWaitlist(), waitlist.AddApplication(), application.getScore()

Processing logic: for each application, get its department and score, if the score is lower than the department threshold, call addToWaitlist() to add current application to waitlist. Return success when all the application is checked successfully. Otherwise, return failure.

Test case 1: Call waitlistApplications. Expected output is: All the application with score lower than its department threshold is added to waitlist.

2. Method: NewDepartment(Department new_department)
Return Type: Department
Parameters: new_department - the Department the College user wants to create
Return value: The created Department or NULL
Pre-condition: None
Post-condition: new_department exists
Attributes read/used: None
Methods called: None

Processing logic: If new_department is NULL, i.e. it doesn't exist, create a new Department object and return it. Otherwise return NULL.

Test case 1: Call newDepartments with Department X where X does not exist yet.
Expected output is: X is created as a new Department.
Test case 1: Call newDepartments with Department X where X does exist.
Expected output is: NULL.

3. Method: ManageSeats(Department department)
Return Type: boolean
Parameters: department - the Department the College user wants to manage the seats of
Return value: Success or Failure
Pre-condition: department is a valid Department
Post-condition: None
Attributes read/used: department.maxSeats
Methods called: department.getMaxSeats(), department.setMaxSeats()

Processing logic: Get the maxSeats of department. If the College user wants to update the maxSeats set the maxSeats of department to the new value as long as it is positive and return Success. Otherwise return Failure.

Test case 1: Call ManageSeats with Department X and change maxSeats to a valid positive Y.
Expected output is: maxSeats attribute of X is set to Y and return Success.

4. Method: ManageThresholds(Department department)
   Return Type: boolean
   Parameters: department - the Department the College user wants to manage the thresholds of
   Return value: Success or Failure
   Pre-condition: department is a valid Department
   Post-condition: None
   Attributes read/used: department.thresholds
   Methods called: department.getThresholds(), department.setThresholds()

   Processing logic: Get the thresholds of department. If the College user wants to update the thresholds set the thresholds of department to the new value as long as the given values are positive and return Success. Otherwise return Failure.

   Test case 1: Call ManageThresholds with Department X and change the second thresholds value to a valid positive Y.
   Expected output is: thresholds[1] attribute of X is set to Y and return Success.

5. Method: NewWaitlist(Department department, string app_type)
   Return Type: Waitlist
   Parameters: department - the Department the College user wants to create the Waitlist in
   app_type - the Application type the Waitlist should accept
   Return value: The created Waitlist or NULL
   Pre-condition: None
   Post-condition: None
   Attributes read/used: department.waitlists, waitlist.type
   Methods called: department.getWaitlists(), waitlist.setType(), department.setWaitlists()

   Processing logic: Get all the waitlists for department. As long as department has less than 3 waitlists and none of the waitlists have the same type as app_type, create a new Waitlist object and set its type = app_type. Then set the new Waitlist as one of department's Waitlists and return it. Otherwise return NULL.

   Test case 1: Call newWaitlist with Department X and string "Undergraduate" where X has 2 Waitlists and neither of them are of type "Undergraduate".
   Expected output is: Y, a Waitlist with type="Undergraduate", and returned. Y is set as a Waitlist for X.
   Test case 2: Call newWaitlist with Department X and string "Undergraduate" where X has 3 Waitlists.
   Expected output is: NULL.

6. Method: getWaitlistApps(Waitlist waitlist, integer num)
   Return Type: list of Applications
   Parameters: waitlist - the Waitlist to get Applications from

num - the integer of Applications to return
Return value: list of Applications that is greater than or equal to num in size
Pre-condition: None
Post-condition: None
Attributes read/used: waitlist.applications
Methods called: waitlist.GetApplications()

Processing logic: As long as the waitlist exists, call waitlist.GetApplications(num), and return the results.

Test case 1: Call getWaitlistApps with waitlist X and integer Y, and the size of applications in X is greater than Y.
Expected output is: Y Applications are returned, and they are the Y Applications from the top of the Waitlist X.

4.4.9 Class: Applicant

- Purpose: To model the relevant aspects of the Applicant
- Constraints: None
- Persistent: No (created at system initialization from other available data)

4.4.9.1 Attribute Descriptions

1. Attribute: applicant_id
   Type: integer
   Description: Stores the id of the current Applicant
   Constraints: Should be positive

2. Attribute: confirmedApplication
   Type: Application
   Description: Stores the confirmed application of the current Applicant
   Constraints: None

4.4.9.2 Method Descriptions

1. Method: FinalDecision(Application application)
   Return Type: boolean
   Parameters: application - the Application that the Applicant wants to give a final decision to
   Return value: Success or Failure
   Pre-condition: Current Application status is “Accepted”
   Post-condition: Application status is now “Confirmed” or “Rejected”
   Attributes read/used: confirmedApplication, application.department, application.status
   Methods called: application.setStatus(), application.getDepartment(), department.currentSeats()

Processing logic: Get the Application status from the Application. If the Application status is not "Accepted", Applicant cannot make final decision, make no change and return Failure. If the Application status is "Accepted", and Applicant wants to change it to "Rejected" or "Confirmed", change the application status, update the confirmedApplication attribute, and return Success. If the status is changed to "Confirmed" also update the corresponding currentSeats of the Application's department.

Test case 1: Call FinalDecision with decision X and application Y.
Expected output is: status of application Y is set to X and return true if the input is valid. Else status is not modified and return false

2. Method: setConfirmedApplication(Application new_application)
   Return Type: boolean
   Parameters: new_application – the new confirmed Application to be set for currentApplication.
   Return value: Success or Failure.
   Pre-condition: None
   Post-condition: confirmedApplication is set to new_application
   Attributes read/used: confirmedApplication
   Methods called: None

   Processing logic: If confirmed application is NULL, set confirmedApplication to the input new_application and return Success. If there is already an Application, X, set as currentApplication change the status of X to "Accepted", and then set currentApplication to new_application and return Success. Otherwise, return Failure.

   Test case 1: Call setConfirmedApplication with application X when confirmedApplication is NULL.
   Expect output is the confirmedApplication set to X and return true.
   Test case 2: Call setConfirmedApplication with application X when confirmedApplication is Y.
   Expect output is the confirmedApplication set to X and Y's status is "Accepted" and return true.

3. Method: getApplications()
   Return Type: list of Applications
   Parameters: None
   Return value: All Applications the current Applicant can access or NULL
   Pre-condition: None
   Post-condition: None
   Attributes read/used: user_level, applicant_id
   Methods called: None

   Processing logic:

Use U-ADMT Repository interface to search shared database for Applications that the current user_level can access. Then return all matching applications, with applicant = applicant_id otherwise, return NULL.

Test case 1: Call getApplication with correct applicant_id = X.
Expect output is all the Applications with applicant = X.

4.5 Classes in the Application Subsystem

The Application subsystem uses a factory pattern to create Application objects to be one of three different subclasses (Undergraduate, Graduate, or Professional) that represent different Application types. The factory is represented by the ApplicationFactory class which creates objects using the Application interface, and associates each Application with an Applicant user (Scenario 5.1.1 and Figure 5.1). Additionally, each Application can have upwards of 3 Reviews associated with it. Each Review is created by a Faculty user, and is used to assess an Application based on score, recommendations, and comments (Scenario 5.1.4 and Figure 5.4).

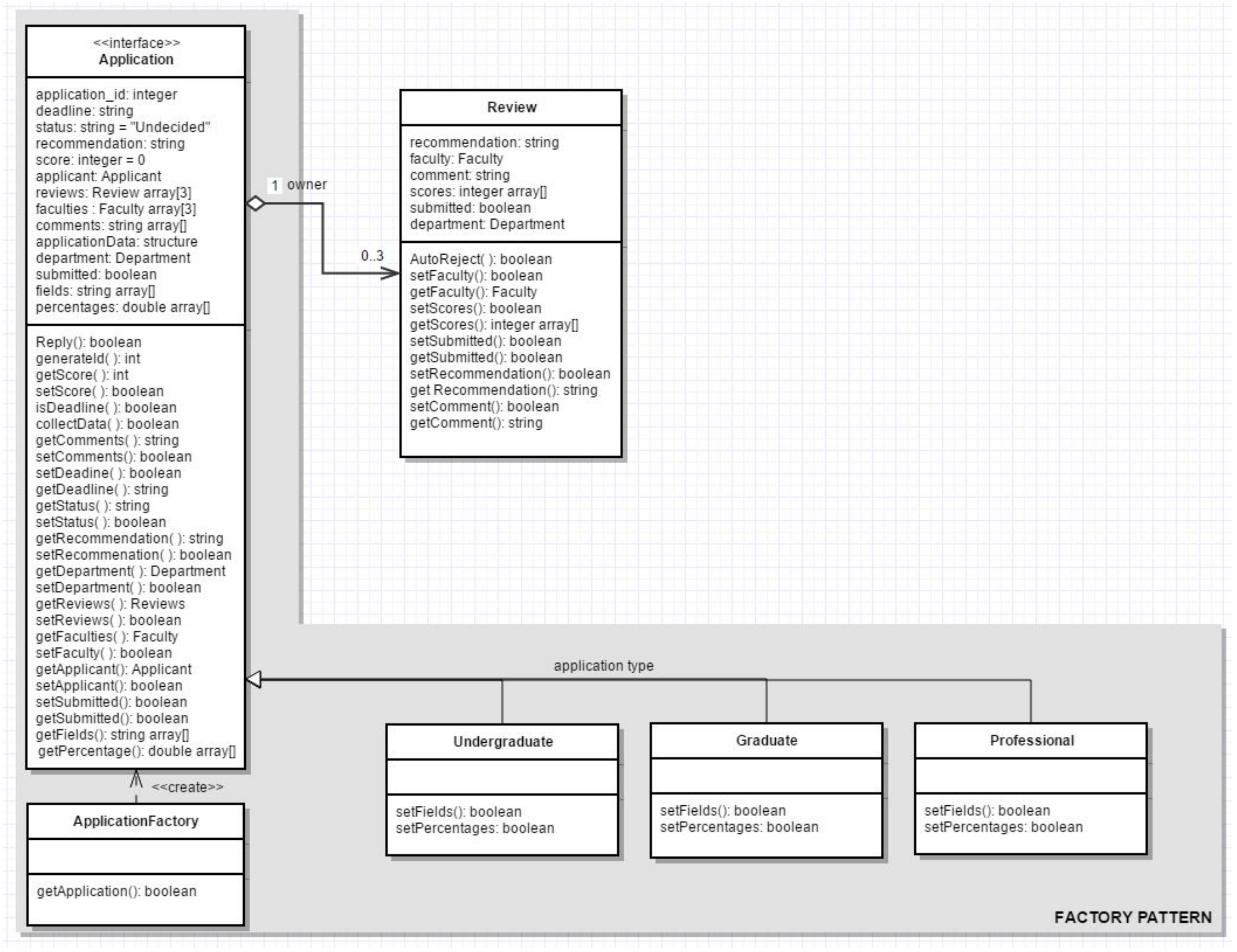

Figure 4.2: Application Subsystem Class Diagram (larger version as Figure4.2.png)

4.5.1 Class: ApplicationFactory

- Purpose: To model the relevant aspects of the ApplicationFactory
- Constraints: None
- Persistent: No (created at system initialization from other available data)

4.5.1.1 Attribute Descriptions

4.5.1.2 Method Descriptions

1. Method: getApplication(string application_type)
   Return Type: Application
   Parameters: application_type - the string indicating the type of Application to create (e.g. Graduate)
   Return value: the created Application
   Pre-condition: None
   Post-condition: An Application based on application_type is created
   Attributes read/used: None
   Methods called:

   Processing logic: Create a new Application based on applciation_type and return it. Otherwise return NULL

   Test case 1: Call getApplication() with string "Undergraduate"
   Expected output is: A new Undergraduate Application.

4.5.2 Class: Application

- Purpose: To model the relevant aspects of the Application
- Constraints: Must be created by ApplicationFactory
- Persistent: Yes

4.5.2.1 Attribute Descriptions

1. Attribute: application_id
   Type: integer
   Description: Stores the id of the current Application
   Constraints: Should be a positive value

2. Attribute: score
   Type: integer
   Description: Stores the score of the current Application
   Constraints: Should be a value between 0 and 100

3. Attribute: status

Type: string
Description: Stores the Application status
Constraints:Should be one of the following cases: “Undecided”, “Accepted”, “Waitlisted”, or “Rejected”

4. Attribute: recommendation
   Type: string
   Description: Stores the recommended status for the current Application
   Constraints: Should be one of the following cases: “Accept”, “Waitlist”, or “Reject”

5. Attribute: reviews
   Type: list of Reviews
   Description: Stores the all Faculty Review data for the current Application
   Constraints: Must contain three or less Reviews

6. Attribute: faculties
   Type: list of Faculty
   Description: Stores the assigned Faculty for the current Application
   Constraints: Must contain three or less Faculty

7. Attribute: deadline
   Type: string
   Description: Stores the deadline for the current application
   Constraints: Should be in date format “month-day-year hh:mm:ss timezone”

8. Attribute: applicant
   Type: Applicant
   Description: Stores the application owner
   Constraints: Should be valid Applicant object

9. Attribute: department
   Type: Department
   Description: Stores the Department which the Application is applying to
   Constraints: Should be valid Department object

10. Attribute: applicationData
    Type: structure
    Description: Stores the Application Data from the shared database (contains stuff like the application type & stats like GPA, etc.)
    Constraints: None

11. Attribute: comments
    Type: list of strings

Description: Stores comments to the Application
Constraints: None

12. Attribute: submitted
Type: boolean
Description: Stores whether the Application has been submitted by an Admin user or not
Constraints: Should be "True" or "False

13. Attribute: fields
Type: list of strings
Description: Stores the fields the Application will be reviewed on based on the application type
Constraints: Should either be ["GPA", "Extra", "Recommendation", "ACT/SAT", "Essay"] for Undergraduate applications. ["GPA", "Research", "Recommendation", "GRE", "Essay"] for Graduate applications, or ["GPA", "Work", "Recommendation", "Essay"] for Professional applications

14. Attribute: percentages
Type: list of doubles
Description: Stores the percentages the Application scores will be weighted on based on the application type and corresponding fields
Constraints: Should either be [0.25, 0.20, 0.15, 0.25, 0.15] for Undergraduate applications, [0.30, 0.20, 0.20, 0.15, 0.15] for Graduate applications, or [0.25, 0.25, 0.25, 0,25] for Professional applications

4.5.2.2 Method Descriptions

1. Method: setScore(integer new_score)
Return Type: boolean
Parameters: new_score - the integer object to set
Return value: Success or Failure
Pre-condition: new_socre is between 0 and 100
Post-condition: score is now set to new_score
Attributes read/used: score
Methods called: None

   Processing logic: Set the attribute score to the input new_score and return Success. Otherwise return Failure.

   Test case 1: Call setScore with integer X.
Expected output is: score attribute is set to X and return Success.

2. Method: getScore()
Return Type: integer
Parameters: None

Return value: score, the attribute containing the current set integer, or NULL
Pre-condition: None
Post-condition: None
Attributes read/used: score
Methods called: None

Processing logic: If the attribute score exists, return it. Otherwise return NULL.

Test case 1: Call getScore where Application has a score attribute = x.
Expected output is: score attribute.
Test case 2: Call geScores where Application does not have a scores attribute.
Expected output is: NULL

3. Method: setComments(string new_comment)
   Return Type: boolean
   Parameters: new_comment - the string to add to comments
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: comments now includes new_comment
   Attributes read/used: comments
   Methods called: None

   Processing logic: Set the attribute comments to contain the input new_comment and return Success. Otherwise return Failure.

   Test case 1: Call setComments with string X.
   Expected output is: comments attribute is set to contain X and return Success.

4. Method: getComments()
   Return Type: list of strings
   Parameters: None
   Return value: comments, the attribute containing the current set list of strings, or NULL
   Pre-condition: None
   Post-condition: None
   Attributes read/used: comments
   Methods called: None

   Processing logic: If the attribute comments exists, return it. Otherwise return NULL.

   Test case 1: Call getComments where Application has a comments attribute = x.
   Expected output is: comments attribute.
   Test case 2: Call getComments where Application does not have a comments attribute.
   Expected output is: NULL

5. Method: Reply(string new_reply)
   Return Type: boolean
   Parameters: new_reply - the string to add as a reply to comments
   Return value: Success or Failure
   Pre-condition: comments is not empty
   Post-condition: comments now includes new_reply as a reply to the last comment
   Attributes read/used: comments
   Methods called: setComments()

   Processing logic: Set the attribute comments to contain the input new_reply as a reply to the last string in the comments attribute and return Success. Otherwise return Failure

   Test case 1: Call Reply with string X.
   Expected output is: comments attribute is set to contain X as a reply to the previously last comment and return Success.

6. Method: setReviews(Review new_review)
   Return Type: boolean
   Parameters: new_review - the Review to add to reviews
   Return value: Success or Failure
   Pre-condition: reviews has less than 3 Reviews in it
   Post-condition: reviews now includes new_review
   Attributes read/used: reviews
   Methods called: None

   Processing logic: Set the attribute reviews to contain the input new_review and return Success. Otherwise return Failure.

   Test case 1: Call setReviews with Review X.
   Expected output is: reviews attribute is set to contain X and return Success.

7. Method: getReviews()
   Return Type: list of Reviews
   Parameters: None
   Return value: reviews, the attribute containing the current set list of Reviews, or NULL
   Pre-condition: None
   Post-condition: None
   Attributes read/used: reviews
   Methods called: None

   Processing logic: If the attribute reviews exists, return it. Otherwise return NULL.

Test case 1: Call getReviews where Application has a reviews attribute = x.
Expected output is: reviews attribute.
Test case 2: Call geReviews where Application does not have a reviews attribute.
Expected output is: NULL

8. Method: setFaculty(Faculty new_faculty)
   Return Type: boolean
   Parameters: new_faculty - the Faculty to add to faculties
   Return value: Success or Failure
   Pre-condition: faculties has less than 3 Faculty in it
   Post-condition: faculties now includes new_faculty
   Attributes read/used: faculties
   Methods called: None

   Processing logic: Set the attribute faculties to contain the input new_faculty and return Success. Otherwise return Failure.

   Test case 1: Call setFaculty with Faculty X.
   Expected output is: faculties attribute is set to contain X and return Success.

9. Method: getFaculities()
   Return Type: list of Faculty
   Parameters: None
   Return value: faculties, the attribute containing the current set list of Faculty, or NULL
   Pre-condition: None
   Post-condition: None
   Attributes read/used: faculties
   Methods called: None

   Processing logic: If the attribute faculties exists, return it. Otherwise return NULL.

   Test case 1: Call getFaculties where Application has a faculties attribute = x.
   Expected output is: faculties attribute.
   Test case 2: Call geFaculties where Application does not have a faculties attribute.
   Expected output is: NULL

10. Method: setStatus(string new_status)
    Return Type: boolean
    Parameters: new_status - the string to set
    Return value: Success or Failure
    Pre-condition: new_status is “Undecided”, “Accepted”, “Waitlisted”, or “Rejected”
    Post-condition: status is now set to new_status
    Attributes read/used: status

Methods called: None

Processing logic: Set the attribute status to the input new_status and return Success. Otherwise return Failure.

Test case 1: Call setStatus with string X.
Expected output is: status attribute is set to X and return Success.

11. Method: getStatus()
    Return Type: string
    Parameters: None
    Return value: status, the attribute containing the current set string, or NULL
    Pre-condition: None
    Post-condition: None
    Attributes read/used: status
    Methods called: None

    Processing logic: If the attribute status exists, return it. Otherwise return NULL.

    Test case 1: Call getStatus where Application has a status attribute = x.
    Expected output is: status attribute.
    Test case 2: Call getStatus where Application does not have a status attribute.
    Expected output is: NULL

12. Method: setRecommendation(string new_recommendation)
    Return Type: boolean
    Parameters: new_recommendation - the string to set
    Return value: Success or Failure
    Pre-condition: new_recommendation is “Accept”, “Waitlist”, or “Reject”
    Post-condition: recommendation is now set to new_recommendation
    Attributes read/used: recommendation
    Methods called: None

    Processing logic: Set the attribute recommendation to the input new_recommendation and return Success. Otherwise return Failure.

    Test case 1: Call setRecommendation with string X.
    Expected output is: recommendation attribute is set to X and return Success.

13. Method: getRecommendation()
    Return Type: integer
    Parameters: None
    Return value: recommendation, the attribute containing the current set string, or NULL

Pre-condition: None
Post-condition: None
Attributes read/used: recommendation
Methods called: None

Processing logic: If the attribute recommendation exists, return it. Otherwise return NULL.

Test case 1: Call getRecommendation where Application has a recommendation attribute = x.
Expected output is: recommendation attribute.
Test case 2: Call getRecommendation where Application does not have a recommendation attribute.
Expected output is: NULL

14. Method: setDepartment(Department new_department)
    Return Type: boolean
    Parameters: new_department - the Department to set
    Return value: Success or Failure
    Pre-condition: new_department is valid Department
    Post-condition: department is now set to new_department
    Attributes read/used: department
    Methods called: None

    Processing logic: Set the attribute department to the input new_department and return Success. Otherwise return Failure.

    Test case 1: Call setDepartment with Department X.
    Expected output is: department attribute is set to X and return Success.

15. Method: getDepartment()
    Return Type: Department
    Parameters: None
    Return value: department, the attribute containing the current set Department, or NULL
    Pre-condition: None
    Post-condition: None
    Attributes read/used: department
    Methods called: None

    Processing logic: If the attribute department exists, return it. Otherwise return NULL.

    Test case 1: Call getDepartment where Application has a department attribute = x.
    Expected output is: department attribute.
    Test case 2: Call getDepartment where Application does not have a department attribute.

Expected output is: NULL

16. Method: setApplicant(Applicant new_applicant)
    Return Type: boolean
    Parameters: new_applicant - the Applicant to set
    Return value: Success or Failure
    Pre-condition: new_applicant is valid Applicant
    Post-condition: applicant is now set to new_applicant
    Attributes read/used: applicant
    Methods called: None

    Processing logic: Set the attribute applicant to the input new_applicant and return Success. Otherwise return Failure.

    Test case 1: Call setApplicant with Applicant X.
    Expected output is: applicant attribute is set to X and return Success.

17. Method: getApplicant()
    Return Type: Applicant
    Parameters: None
    Return value: applicant, the attribute containing the current set Applicant, or NULL
    Pre-condition: None
    Post-condition: None
    Attributes read/used: applicant
    Methods called: None

    Processing logic: If the attribute applicant exists, return it. Otherwise return NULL.

    Test case 1: Call getApplicant where Application has a applicant attribute = x.
    Expected output is: applicant attribute.
    Test case 2: Call getApplicant where Application does not have a applicant attribute.
    Expected output is: NULL

18. Method: setSubmitted(boolean new_submitted)
    Return Type: boolean
    Parameters: new_submitted - the boolean to set
    Return value: Success or Failure
    Pre-condition: None
    Post-condition: submitted is now set to new_submitted
    Attributes read/used: submitted
    Methods called: None

    Processing logic: Set the attribute submitted to the input new_submitted and return Success.

Otherwise return Failure.

Test case 1: Call setSubmitted with boolean X.
Expected output is: submitted attribute is set to X and return Success.

19. Method: getSubmitted()
Return Type: boolean
Parameters: None
Return value: submitted, the attribute containing the current set boolean, or NULL
Pre-condition: None
Post-condition: None
Attributes read/used: submitted
Methods called: None

Processing logic: If the attribute submitted exists, return it. Otherwise return NULL.

Test case 1: Call getSubmitted where Application has a submitted attribute = x.
Expected output is: submitted attribute.
Test case 2: Call getSubmitted where Application does not have a submitted attribute.
Expected output is: NULL

20. Method: setDeadline(string new_deadline)
Return Type: boolean
Parameters: new_deadline - the string to set
Return value: Success or Failure
Pre-condition: None
Post-condition: deadline is now set to new_deadline
Attributes read/used: deadline
Methods called: None

Processing logic: Set the attribute deadline to the input new_deadline and return Success.
Otherwise return Failure.

Test case 1: Call setDeadline with string X.
Expected output is: deadline attribute is set to X and return Success.

21. Method: getDeadlinee()
Return Type: integer
Parameters: None
Return value: deadline, the attribute containing the current set string, or NULL
Pre-condition: None
Post-condition: None
Attributes read/used: deadline

Methods called: None

Processing logic: If the attribute deadline exists, return it. Otherwise return NULL.

Test case 1: Call getDeadline where Application has a deadline attribute = x.
Expected output is: deadline attribute.
Test case 2: Call getDeadline where Application does not have a deadline attribute.
Expected output is: NULL

22. Method: isDeadline()
Return Type: boolean
Parameters: None
Return value: Whether the current time has passed the deadline or not - True or False
Pre-condition: deadline attribute exists
Post-condition: None
Attributes read/used: deadline
Methods called: None

Processing logic:Get the system time, compare with deadline attribute. If the system time is equal or larger than deadline, return true. Else return false.

Test case 1: Current time is X, deadline is Y(Y<=X), call isDeadline().
Expected output is: Return true.
Test case 1: Current time is X, deadline is Y(Y>X), call isDeadline().
Expected output is: Return false.

23. Method: collectData()
Return Type: boolean
Parameters: None
Return value: Success or Failure
Pre-condition: There exists Application Data for the current Application
Post-condition: applicationData is now filled with valid data
Attributes read/used: applicationData
Methods called: collectData()

Processing logic: Get data for an Application by using U-ADMT Repository to get data from the shared database, and add to applicationData structure attribute, and return Success. Else return Failure.

Test case 1: Call collectData.
Expected output is: applicationData is added as applicationData attribute.

24. Method: generateId()

Return Type:boolean
Parameters: None
Return value: Success or Failure
Pre-condition: None
Post-condition: application_id is now a unique integer
Attributes read/used: application_id
Methods called: None

Processing logic: When application_id attribute is empty, set a new application_id to application_id attribute that is an uniquely generated integer.

Test case 1: Call generateId when application_id is empty.
Expected output is: application_id set to new integer X.
Test case 2: Call generateId when application_id is not empty.
Expected output is: application_id is not changed.

25. Method: getFields()
Return Type: list of strings
Parameters: None
Return value: fields, the attribute containing the current set list of strings, or NULL
Pre-condition: None
Post-condition: None
Attributes read/used: fields
Methods called: None

Processing logic: If the attribute fields exists, return it. Otherwise return NULL.

Test case 1: Call getFields where Application has a fields attribute = x.
Expected output is: fields attribute.
Test case 2: Call getFields where Application does not have a fields attribute.
Expected output is: NULL

26. Method: getPercentages()
Return Type: list of doubles
Parameters: None
Return value: percentages, the attribute containing the current set list of doubles, or NULL
Pre-condition: None
Post-condition: None
Attributes read/used: percentages
Methods called: None

Processing logic: If the attribute percentages exists, return it. Otherwise return NULL.

Test case 1: Call getPercentages where Application has a percentages attribute = x.
Expected output is: percentages attribute.
Test case 2: Call getPercentages where Application does not have a percentages attribute.
Expected output is: NULL

4.5.3 Class: Undergraduate

· Purpose: To model the relevant aspects of the Undergraduate Application
· Constraints: subclass of Application
· Persistent: No (created at system initialization from other available data)

4.5.3.1 Attribute Descriptions

4.5.3.2 Method Descriptions

1. Method: setFields()
   Return Type: boolean
   Parameters: None
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: Application fields is set to [“GPA”, “Extra”, “Recommendation”, “ACT/SAT”, “Essay”]
   Attributes read/used: Application.fields
   Methods called: None

   Processing logic: Set the fields attribute to [“GPA”, “Extra”, “Recommendation”, “ACT/SAT”, “Essay”] and return Success. Else return Failure.

   Test case 1: Call setFields.
   Expected output is: fields attribute is set to [“GPA”, “Extra”, “Recommendation”, “ACT/SAT”, “Essay”] and return Success.

2. Method: setPercentages()
   Return Type: boolean
   Parameters: None
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: Application percentages is set to [0.25, 0.20, 0.15, 0.25, 0.15]
   Attributes read/used: Application.percentages
   Methods called: None

   Processing logic: Set the percentages attribute to [0.25, 0.20, 0.15, 0.25, 0.15] and return Success. Else return Failure.

Test case 1: Call setPercentages.
Expected output is: fields attribute is set to [0.25, 0.20, 0.15, 0.25, 0.15] and return Success.

4.5.4 Class: Graduate

- Purpose: To model the relevant aspects of the Graduate Application
- Constraints: subclass of Application
- Persistent: No (created at system initialization from other available data)

4.5.4.1 Attribute Descriptions

4.5.4.2 Method Descriptions

1. Method: setFields()
   Return Type: boolean
   Parameters: None
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: Application fields is set to [“GPA”, “Research”, “Recommendation”, “GRE”, “Essay”]
   Attributes read/used: Application.fields
   Methods called: None

   Processing logic: Set the fields attribute to [“GPA”, “Research”, “Recommendation”, “GRE”, “Essay”] and return Success. Else return Failure.

   Test case 1: Call setFields.
   Expected output is: fields attribute is set to [“GPA”, “Research”, “Recommendation”, “GRE”, “Essay”] and return Success.

2. Method: setPercentages()
   Return Type: boolean
   Parameters: None
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: Application percentages is set to [0.30, 0.20, 0.20, 0.15, 0.15]
   Attributes read/used: Application.percentages
   Methods called: None

   Processing logic: Set the percentages attribute to [0.30, 0.20, 0.20, 0.15, 0.15] and return Success. Else return Failure.

Test case 1: Call setPercentages.
Expected output is: fields attribute is set to [0.30, 0.20, 0.20, 0.15, 0.15] and return Success.

4.5.5 Class: Professional

- Purpose: To model the relevant aspects of the Professional Application
- Constraints: subclass of Application
- Persistent: No (created at system initialization from other available data)

4.5.5.1 Attribute Descriptions

4.5.5.2 Method Descriptions

1. Method: setFields()
   Return Type: boolean
   Parameters: None
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: Application fields is set to [“GPA”, “Work”, “Recommendation”, “Essay”]
   Attributes read/used: Application.fields
   Methods called: None

   Processing logic: Set the fields attribute to [“GPA”, “Work”, “Recommendation”, “Essay”] and return Success. Else return Failure.

   Test case 1: Call setFields.
   Expected output is: fields attribute is set to [“GPA”, “Work”, “Recommendation”, “Essay”] and return Success.

2. Method: setPercentages()
   Return Type: boolean
   Parameters: None
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: Application percentages is set to [0.25, 0.25, 0.25, 0,25]
   Attributes read/used: Application.percentages
   Methods called: None

   Processing logic: Set the percentages attribute to [0.25, 0.25, 0.25, 0,25] and return Success. Else return Failure.

   Test case 1: Call setPercentages.
   Expected output is: fields attribute is set to [0.25, 0.25, 0.25, 0,25] and return Success.

4.5.6 Class: Review

- Purpose: To model the relevant aspects of an Review
- Constraints: Must be created by Faculty user
- Persistent: Yes

4.5.6.1 Attribute Descriptions

1. Attribute: recommendation
   Type: string
   Description: Stores the recommendation given to the Review
   Constraints: Has to be either "Accept", "Waitlist", or "Reject"

2. Attribute: faculty
   Type: Faculty
   Description: Stores the Faculty user who created the review
   Constraints: Must be valid Faculty object

3. Attribute: comment
   Type: string
   Description: Stores an optional comment
   Constraints: None

4. Attribute: scores
   Type: list of integers
   Description: Stores the scores given to the Review
   Constraints: Must be between non-negative

5. Attribute: submitted
   Type: boolean
   Description: Stores whether the Review has been submitted or not
   Constraints: Must be initialized to False

6. Attribute: department
   Type: Department
   Description: Stores the Department the review, and thus corresponding scores, is for
   Constraints: Must be valid Department object

4.5.6.2 Method Descriptions

1. Method: AutoReject()
   Return Type: boolean
   Parameters: None
   Return value: Success or Failure

Pre-condition: submitted is False
Post-condition: scores is an array of 0s and recommendation is “Reject”
Attributes read/used: scores, recommendation
Methods called: None

Processing logic: Set each value in scores to 0, and set recommendation to “Reject”.

Test case 1: Call AutoReject on an unsubmitted Review object X.
Expected output is: scores is an array of 0s, and recommendation is “Reject”, return Success.
Test case 2: Call AutoReject on a submitted Review object X.
Expected output is: nothing is changed, return Failure.

2. Method: setFaculty(Faculty new_faculty)
   Return Type: boolean
   Parameters: new_faculty - the Faculty object to set
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: faculty is now set to new_faculty
   Attributes read/used: faculty
   Methods called: None

   Processing logic: Set the attribute faculty to the input new_faculty and return Success. Otherwise return Failure.

   Test case 1: Call setFaculty with Faculty X.
   Expected output is: faculty attribute is set to X and return Success.

3. Method: getFaculty()
   Return Type: Faculty
   Parameters: None
   Return value: faculty, the attribute containing the current set Faculty object, or NULL
   Pre-condition: None
   Post-condition: None
   Attributes read/used: faculty
   Methods called: None

   Processing logic: If the attribute faculty exists, return it. Otherwise return NULL.

   Test case 1: Call getFaculty where Review has a faculty attribute = x.
   Expected output is: faculty attribute.
   Test case 2: Call getFaculty where Review does not have a faculty attribute.
   Expected output is: NULL

4. Method: setScores(integer new_score)
   Return Type: boolean
   Parameters: new_score - the integer score to add
   Return value: Success or Failure
   Pre-condition: new_score must be non-negative
   Post-condition: scores is now includes new_score
   Attributes read/used: scores
   Methods called: None

   Processing logic: Set the attribute scores to include the input new_scores and return Success. Otherwise return Failure.

   Test case 1: Call setScores with integer X.
   Expected output is: scores attribute is set to include X and return Success.

5. Method: getScores()
   Return Type: list of integers
   Parameters: None
   Return value: scores, the attribute containing the current set list of integers, or NULL
   Pre-condition: None
   Post-condition: None
   Attributes read/used: scores
   Methods called: None

   Processing logic: If the attribute scores exists, return it. Otherwise return NULL.

   Test case 1: Call getScores where Review has a scores attribute = x.
   Expected output is: scores attribute.
   Test case 2: Call geScores where Review does not have a scores attribute.
   Expected output is: NULL

6. Method: setSubmitted(boolean new_submitted)
   Return Type: boolean
   Parameters: new_submitted - the boolean to set
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: submitted is now set to new_submitted
   Attributes read/used: submitted
   Methods called: None

   Processing logic: Set the attribute submitted to the input new_submitted and return Success. Otherwise return Failure.

Test case 1: Call setSubmitted with boolean X.
Expected output is: submitted attribute is set to X and return Success.

7. Method: getSubmitted()
   Return Type: boolean
   Parameters: None
   Return value: submitted, the attribute containing the current set boolean, or NULL
   Pre-condition: None
   Post-condition: None
   Attributes read/used: submitted
   Methods called: None

   Processing logic: If the attribute submitted exists, return it. Otherwise return NULL.

   Test case 1: Call getSubmitted where Review has a submitted attribute = x.
   Expected output is: submitted attribute.
   Test case 2: Call getSubmitted where Review does not have a submitted attribute.
   Expected output is: NULL

8. Method: setRecommendation(string new_recommendation)
   Return Type: boolean
   Parameters: new_recommendation - the string to set
   Return value: Success or Failure
   Pre-condition: new_recommendation must be "Accept", "Reject", or "Waitlist"
   Post-condition: recommendation is now set to new_recommendation
   Attributes read/used: recommendation
   Methods called: None

   Processing logic: Set the attribute recommendation to the input new_recommendation and return Success. Otherwise return Failure.

   Test case 1: Call setRecommendation with string X.
   Expected output is: recommendation attribute is set to X and return Success.

9. Method: getRecommendation()
   Return Type: string
   Parameters: None
   Return value: recommendation, the attribute containing the current set string, or NULL
   Pre-condition: If recommendation exists it must be either "Accept", "Reject", or "Waitlist"
   Post-condition: None
   Attributes read/used: recommendation
   Methods called: None

Processing logic: If the attribute recommendation exists, return it. Otherwise return NULL.

Test case 1: Call getRecommendation where Review has a recommendation attribute = x.
Expected output is: recommendationattribute.
Test case 2: Call getRecommendation where Review does not have a recommendation attribute.
Expected output is: NULL

10. Method: setComment(string new_comment)
Return Type: boolean
Parameters: new_comment - the string to set
Return value: Success or Failure
Pre-condition: None
Post-condition: comment is now set to new_comment
Attributes read/used: comment
Methods called: None

Processing logic: Set the attribute comment to the input new_comment and return Success. Otherwise return Failure.

Test case 1: Call setComment with string X.
Expected output is: comment attribute is set to X and return Success.

11. Method: getComment()
Return Type: string
Parameters: None
Return value: comment, the attribute containing the current set string, or NULL
Pre-condition: None
Post-condition: None
Attributes read/used: comment
Methods called: None

Processing logic: If the attribute comment exists, return it. Otherwise return NULL.

Test case 1: Call geComment where Review has a comment attribute = x.
Expected output is: comment attribute.
Test case 2: Call getComment where Review does not have a comment attribute.
Expected output is: NULL

4.6 Classes in the Department Subsystem

The Department subsystem consists of two classes: Department and Waitlist. Both classes are created and managed by the College class in the User subsystem (Scenarios 5.1.10 and 5.1.11 with Figures 5.10 and 5.11). The Department class contains data on the current and maximum seats as well as threshold screening values for some department at the university. A Department can have upwards of 3 different Waitlists associated with it; one for Undergraduate, one for Graduate, and one for Professional Applications. Faculty and Admin classes in the User subsystem are associated with one or more Departments (Scenario 5.1.3 and Figure 5.3). Additionally, Application and Review objects are associated with only one Department and Waitlist (Scenario 5.1.9 and Figure 5.9).

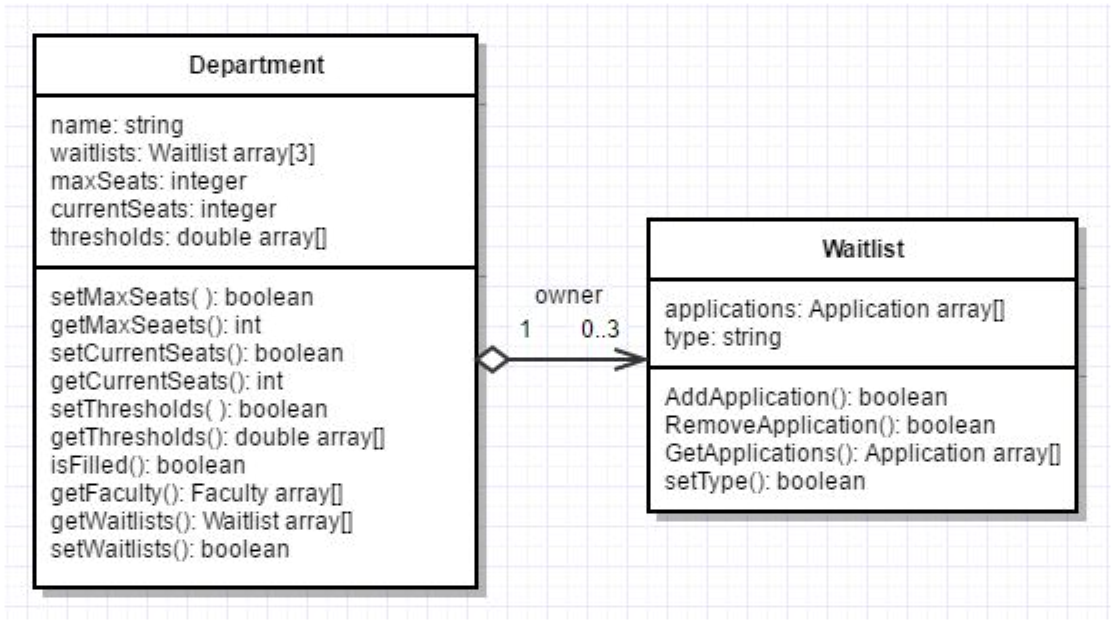
Department
name: string
waitlists: Waitlist array[3]
maxSeats: integer
currentSeats: integer
thresholds: double array[]
setMaxSeats( ): boolean
getMaxSeaets(): int
setCurrentSeats(): boolean
getCurrentSeats(): int
setThresholds( ): boolean
getThresholds(): double array[]
isFilled(): boolean
getFaculty(): Faculty array[]
getWaitlists(): Waitlist array[]
setWaitlists(): boolean
owner
1
0..3
Waitlist
applications: Application array[]
type: string
AddApplication(): boolean
RemoveApplication(): boolean
GetApplications(): Application array[]
setType(): boolean

Figure 4.3: Department Subsystem Class Diagram

4.6.1 Class: Department

- Purpose: To model the relevant aspects of the Department
- Constraints: Must be created by College user
- Persistent: Yes

4.6.1.1 Attribute Descriptions

1. Attribute: name
   Type: string
   Description: Stores the name of the Department (e.g. CSE, CLA, etc.)
   Constraints: None

2. Attribute: waitlists
   Type: list of Waitlists
   Description: Stores the Waitlists associated with the Department
   Constraints: Maximum of 3 Waitlists

3. Attribute: maxSeats
   Type: integer
   Description: Stores max number of available admission seats

Constraints: None

4. Attributes: currentSeats
   Type: integer
   Description: Stores current number of admitted seats
   Constraints: None

5. Attributes: thresholds
   Type: list of doubles
   Description: Stores minimum values of criteria for admitting Applications
   Constraints: None

4.6.1.2 Method Descriptions

1. Method: setMaxSeats(integer new_max)
   Return Type: boolean
   Parameters: new_max - the integer to set
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: maxSeats is now set to new_max
   Attributes read/used: maxSeats
   Methods called: None

   Processing logic: Set the attribute maxSeats to the input new_max and return Success. Otherwise return Failure.

   Test case 1: Call setMaxSeats with integer X.
   Expected output is: maxSeats attribute is set to X and return Success.

2. Method: getMaxSeats()
   Return Type: integer
   Parameters: None
   Return value: maxSeats, the attribute containing the current set integer, or NULL.
   Pre-condition: None
   Post-condition: None
   Attributes read/used: maxSeats
   Methods called: None

   Processing logic: If the attribute maxSeats exists, return it. Otherwise return NULL

   Test case 1: Call geMaxSeats where Department has a maxSeats attribute = x.
   Expected output is: maxSeats attribute.
   Test case 2: Call getMaxSeats where Department does not have a maxSeats attribute.
   Expected output is: NULL

4. Method: setCurrentSeats(integer new_current)
   Return Type: boolean
   Parameters: new_current - the integer to set
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: currentSeats is now set to new_current
   Attributes read/used: currentSeats
   Methods called: None

   Processing logic: Set the attribute currentSeats to the input new_comment and return Success. Otherwise return Failure.

   Test case 1: Call setComment with integer X.
   Expected output is: currentSeats attribute is set to X and return Success.

5. Method: getCurrentSeats()
   Return Type: integer
   Parameters: None
   Return value: currentSeats, the attribute containing the current set integer, or NULL.
   Pre-condition: None
   Post-condition: None
   Attributes read/used: currentSeats
   Methods called: None

   Processing logic: If the attribute currentSeats exists, return it. Otherwise return NULL

   Test case 1: Call geMaxSeats where Department has a currentSeats attribute = x.
   Expected output is: currentSeats attribute.
   Test case 2: Call getMaxSeats where Department does not have a currentSeats attribute.
   Expected output is: NULL

6. Method: setThresholds(double new_threshold)
   Return Type: boolean
   Parameters: new_threshold - the double to add to thresholds
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: thresholds now includes new_threshold
   Attributes read/used: thresholds
   Methods called: None

   Processing logic: Set the attribute thresholds to include the input new_threshold and return Success. Otherwise return Failure.

Test case 1: Call setThresholds with double X.
Expected output is: thresholds attribute is set to include X and return Success.

7. Method: getThresholds()
Return Type: list of doubles
Parameters: None
Return value: thresholds, the attribute containing the current set list of doubles, or NULL.
Pre-condition: None
Post-condition: None
Attributes read/used: thresholds
Methods called: None

   Processing logic: If the attribute thresholds exists, return it. Otherwise return NULL

   Test case 1: Call geThresholds where Department has a thresholds attribute = x.
Expected output is: thresholds attribute.
Test case 2: Call getThresholds where Department does not have a thresholds attribute.
Expected output is: NULL

8. Method: setWaitlists(Waitlist new_waitlist)
Return Type: boolean
Parameters: new_waitlists - the Waitlist to add to waitlists
Return value: Success or Failure
Pre-condition: waitlists has less than 3 Waitlists in it
Post-condition: waitlists now includes new_waitlist
Attributes read/used: waitlists
Methods called: None

   Processing logic: Set the attribute waitlists to contain the input new_waitlist and return Success. Otherwise return Failure.

   Test case 1: Call setWaitlists with Waitlist X.
Expected output is: waitlists attribute is set to contain X and return Success.

9. Method: getWaitlists()
Return Type: list of Waitlists
Parameters: None
Return value: waitlists, the attribute containing the current set list of Waitlists, or NULL.
Pre-condition: None
Post-condition: None
Attributes read/used: waitlists
Methods called: None

Processing logic: If the attribute waitlists exists, return it. Otherwise return NULL

Test case 1: Call getWaitlists where Department has a waitlists attribute = x.
Expected output is: waitlists attribute.
Test case 2: Call getWaitlists where Department does not have a waitlists attribute.
Expected output is: NULL

10. Method:isFilled()
Return Type: boolean
Parameters: None
Return value: whether the department available seat is filled - True or False
Pre-condition: None
Post-condition: None
Attributes read/used: maxSeats, currentSeats
Methods called: None

Processing logic: If currentSeats greater or equal to maxSeat return true. If currentSeats smaller than maxSeat, return false.

Test case 1: call isFilled.(currentSeats equals to maxSeats).
Expected output is: Return true.

Test case 2: call isFilled.(currentSeats is smaller than maxSeats).
Expected output is: Return false.

11. Method: getFaculty()
Return Type: list of Faculty
Parameters: None
Return value: a list of Faculty that are associated with the Department or NULL
Pre-condition: None
Post-condition: None
Attributes read/used: Faculty.departments
Methods called: getDeptFaculty()

Processing logic: Use U-ADMT Repository to access the shared database, and get Faculty users with the Department in their departments attribute. Add each Faculty user to the return list, and then return.

Test case 1: call GetFaculty on Department X
Expected output is: Returns the list of Faculty associated with X. If there is no Faculty, return NULL.

4.6.2 Class: Waitlist

- Purpose: To model the relevant aspects of the Filter
- Constraints: Must be created by College user
- Persistent: Yes

4.6.2.1 Attribute Descriptions

1. Attribute: applications
   Type: list of Applications
   Description: Stores the Applications for the current Waitlist
   Constraints: None

2. Attribute: type
   Type: string
   Description: Stores the type of the Applications the Waitlist accepts
   Constraints: None

4.6.2.2 Method Descriptions

1. Method: setType(string new_type)
   Return Type: boolean
   Parameters: new_type - the string to be set
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: type is now set to new_type
   Attributes read/used: type
   Methods called: None

   Processing logic: Set the attribute type to the input new_type and return success. Otherwise return failure.

   Test case 1: Call setType with string X.
   Expected output is: type attribute is set to X and return success.

2. Method: AddApplication(Application new_application)
   Return Type: boolean
   Parameters: new_application - the Application to be added to the Waitlist
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: applications now includes new_application
   Attributes read/used: applications, Application.score
   Methods called: Application.getScore()

Processing logic: Get the score of new_application. Then add new_application to applications so that applications will have all Applications in order of decreasing score, and then return success. Otherwise return failure.

Test case 1: Call AddApplication with Application X.
Expected output is: X is in applications with all the scores in decreasing order and return success.

3. Method: RemoveApplication(Application remove_application)
   Return Type: boolean
   Parameters: remove_application - the Application to be removed from the Waitlist
   Return value: Success or Failure
   Pre-condition: remove_application is in applications
   Post-condition: applications now excludes remove_application
   Attributes read/used: applications
   Methods called: None

   Processing logic: Remove remove_application from applications and return success. Otherwise return failure.

   Test case 1: Call RemoveApplication with Application X which is in applications.
   Expected output is: applications attribute no longer contains X and return success.

4. Method: GetApplications(integer num)
   Return Type: list of Applications
   Parameters: num - the integer of Applications to return
   Return value: list of Applications that is greater than or equal to num in size
   Pre-condition: None
   Post-condition: None
   Attributes read/used: applications
   Methods called: None

   Processing logic: As long as there is an Application in the applications attribute, and the return list hasn't exceeded num, add an Application from the top of the Waitlist to the return list. After either attributes is empty or the number of Applications in the return list is equal to num, return the list of desired Applications.

   Test case 1: Call GetApplications with integer X, and the size of applications is greater than X.
   Expected output is: X Applications are returned, and they are the X Applications from the top of the Waitlist.

4.7 Classes in the Filter Subsystem

The Filter subsystem only consists of one class: Filter. This class is used by the User subsystem classes to create and apply Filters to Applications. The Filter class utilizes the U-ADMT Repository to access the shared database to obtain, and return the desired Applications (Scenario 5.1.8 and Figure 5.8)

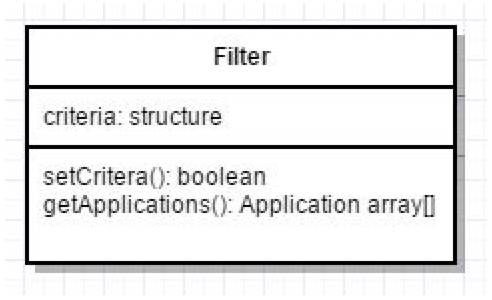

Figure 4.4: Filter Subsystem Class Diagram

4.7.1 Class: Filter

- Purpose: To model the relevant aspects of the Filter
- Constraints: Must be created by User
- Persistent: Yes

4.7.1.1 Attribute Descriptions

1. Attribute: criteria
   Type: structure
   Description: Stores the criteria for the current filter
   Constraints: None

4.7.1.2 Method Descriptions

5. Method: setCriteria(Criteria new_criteria)
   Return Type: boolean
   Parameters: new_criteria - the structure to be set
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: criteria is now set to new_criteria
   Attributes read/used: criteria
   Methods called: None

   Processing logic: Set the attribute criteria to the input structure and return success. Otherwise return failure.

   Test case 1: Call setCriteria with structure X.
   Expected output is: criteria attribute is set to X and return success.

6. Method: getApplications()
   Return Type: list of Application
   Parameters: None

Return value: list of Application that satisfies the criteria
Pre-condition: None
Post-condition: None
Attributes read/used: criteria
Methods called: getApplications()

Processing logic: Get all the Applications from the shared database through U-ADMT Repository by checking one by one if the Application satisfies the criteria. If yes, add to the result list. After checking, return the result list.

Test case 1: Call getApplication with criteria X.
Expected output is: all the application which satisfies the criteria in one result list.

4.8 Classes in the Message Transactor Subsystem

The Message Transactor subsystem only consists of one class: MessageTransactor. This class is used by the College class in the User subsystem to create and send messages from College to Applicant Users through the use of an external Email Service. This usually happens when the College User changes the status of an Application (Scenario 5.1.6 and Figure 5.6).

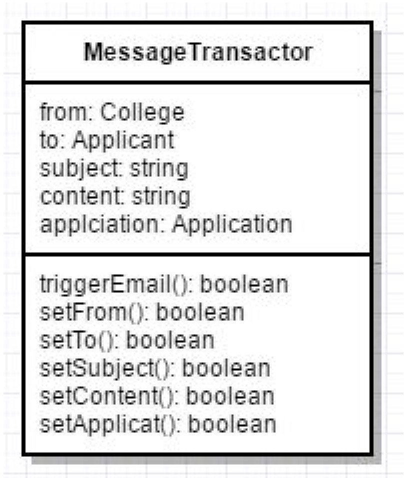

Figure 4.5: Message Transactor Subsystem Class Diagram

4.8.1 Class: MessageTransactor

· Purpose: To model the relevant aspects of the Message Transactor
· Constraints: Must be created by College user
· Persistent: Yes

4.8.1.1 Attribute Descriptions

1. Attribute: from
   Type: College

Description: Stores the College user the message is from
Constraints: Should be College object that created the MessageTransactor

2. Attribute: to
Type: Applicant
Description: Stores the Applicant user the message is to
Constraints: Should be Applicant object corresponding to application

3. Attribute: subject
Type: String
Description: Stores message subject
Constraints: None

4. Attribute: content
Type: String
Description: Stores message content
Constraints: None

5. Attribute: application
Type: Application
Description: Stores which application the message belongs to
Constraints: None

4.8.1.2 Method Descriptions

1. Method: triggerEmail()
Return Type:boolean
Parameters: None
Return value: Success or Failure
Pre-condition: from, to, and application are not NULL
Post-condition: None
Attributes read/used: from, to, subject, content, application
Methods called: sendEmail()

   Processing logic: Send e-mail to Applicant user using the U-ADMT Repository to facilitate sending the email through an external Email Service. Return whether the operation is successful.

   Test case 1: Call triggerEmail() with all attributes filled out.
Expected output is: an e-mail is sent to Applicant.

2. Method: setFrom(College new_college)
Return Type: boolean
Parameters: new_college - the College object to set
Return value: Success or Failure

Pre-condition: None
Post-condition: from is now set to new_college
Attributes read/used: from
Methods called: None

Processing logic: Set the attribute from to the input new_college and return Success. Otherwise return Failure.

Test case 1: Call setFrom with College X.
Expected output is: from attribute is set to X and return Success.

3. Method: setTo(Applicant new_applicant)
   Return Type: boolean
   Parameters: new_applicant - the Applicant object to set
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: to is now set to new_applicant
   Attributes read/used: to
   Methods called: None

   Processing logic: Set the attribute to to the input new_applicant and return Success. Otherwise return Failure.

   Test case 1: Call setTo with Applicant X.
   Expected output is: to attribute is set to X and return Success.

4. Method: setSubject(String new_subject)
   Return Type: boolean
   Parameters: new_subject - the string to set
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: subject is now set to new_subject
   Attributes read/used: subject
   Methods called: None

   Processing logic: Set the attribute subject to the input new_subject and return Success. Otherwise return Failure.

   Test case 1: Call setSubject with string X.
   Expected output is: subject attribute is set to X and return Success.

5. Method: setContent(String new_content)
   Return Type: boolean

Parameters: new_content - the string to set
Return value: Success or Failure
Pre-condition: None
Post-condition: content is now set to new_content
Attributes read/used: content
Methods called: None

Processing logic: Set the attribute content to the input new_content and return Success. Otherwise return Failure.

Test case 1: Call seContent with string X.
Expected output is: content attribute is set to X and return Success.

6. Method: setApplication(Application new_application)
   Return Type: boolean
   Parameters: new_application - the Application object to set
   Return value: Success or Failure
   Pre-condition: None
   Post-condition: faculty is now set to new_application
   Attributes read/used: application
   Methods called: None

   Processing logic: Set the application faculty to the input new_application and return Success. Otherwise return Failure.

   Test case 1: Call setApplication with Application X.
   Expected output is: application attribute is set to X and return Success.

5 Dynamic Model

5.1 Scenarios

5.1.1 Application Creation in U-ADMT

An applicant user submits application data to the external Application Service. As long as the application data is valid, U-ADMT must create application(s) to store the application data. To do that, it creates a new application for each department the applicant user applies to using the ApplicationFactory to specify the type of the Application. The application generates an unique id for itself, collects the application data from the shared database through U-ADMT Repository, and associates itself with the submitting applicant user.

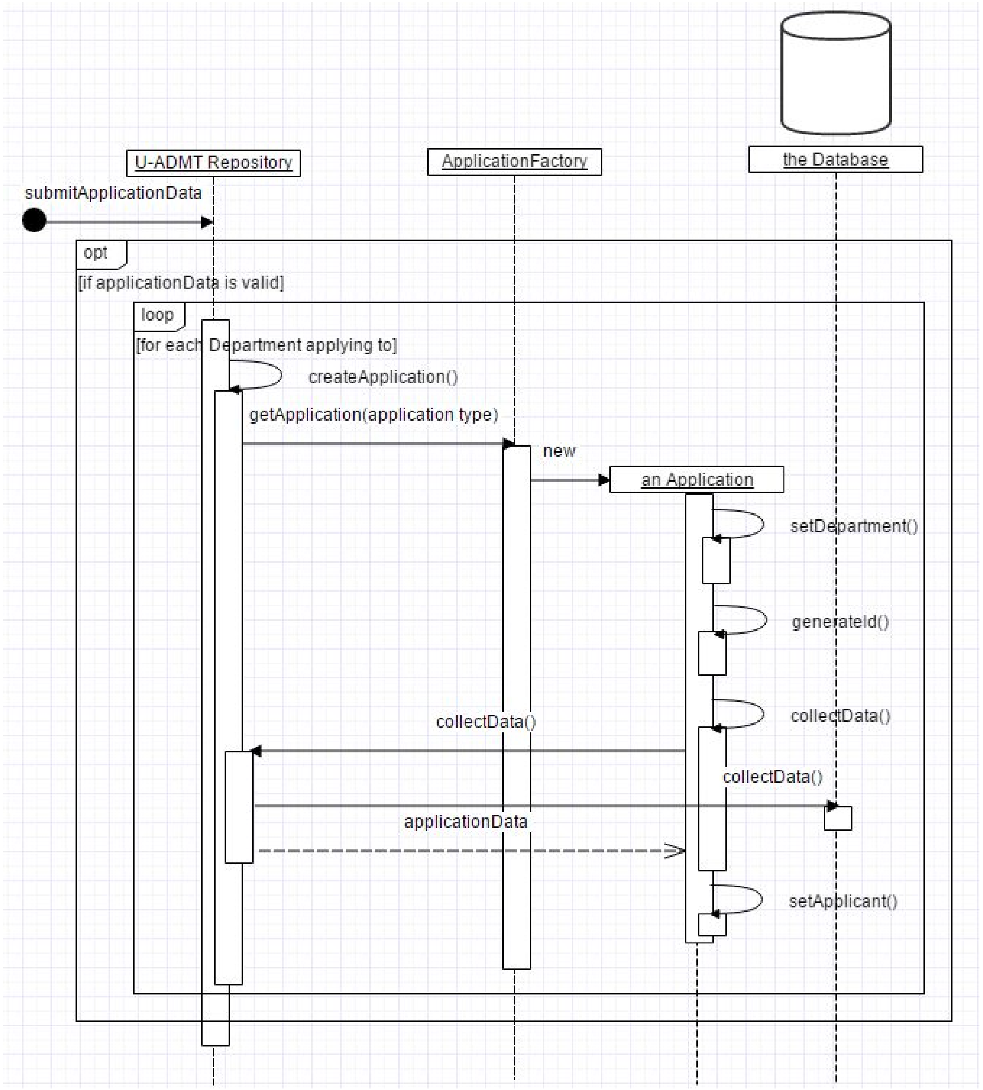

Figure 5.1: Sequence Diagram of 5.1.1 Application Creation in U-ADMT (larger version as Figure5.1.png)

5.1.2 Access Application(s)

An user requests applications. U-ADMT then returns all the applications the user has access to. To do this the system first checks the user level. This matters since different level users may have access to different submitted applications. Finally it uses U-ADMT Repository to access the shared database to return each application to the user.

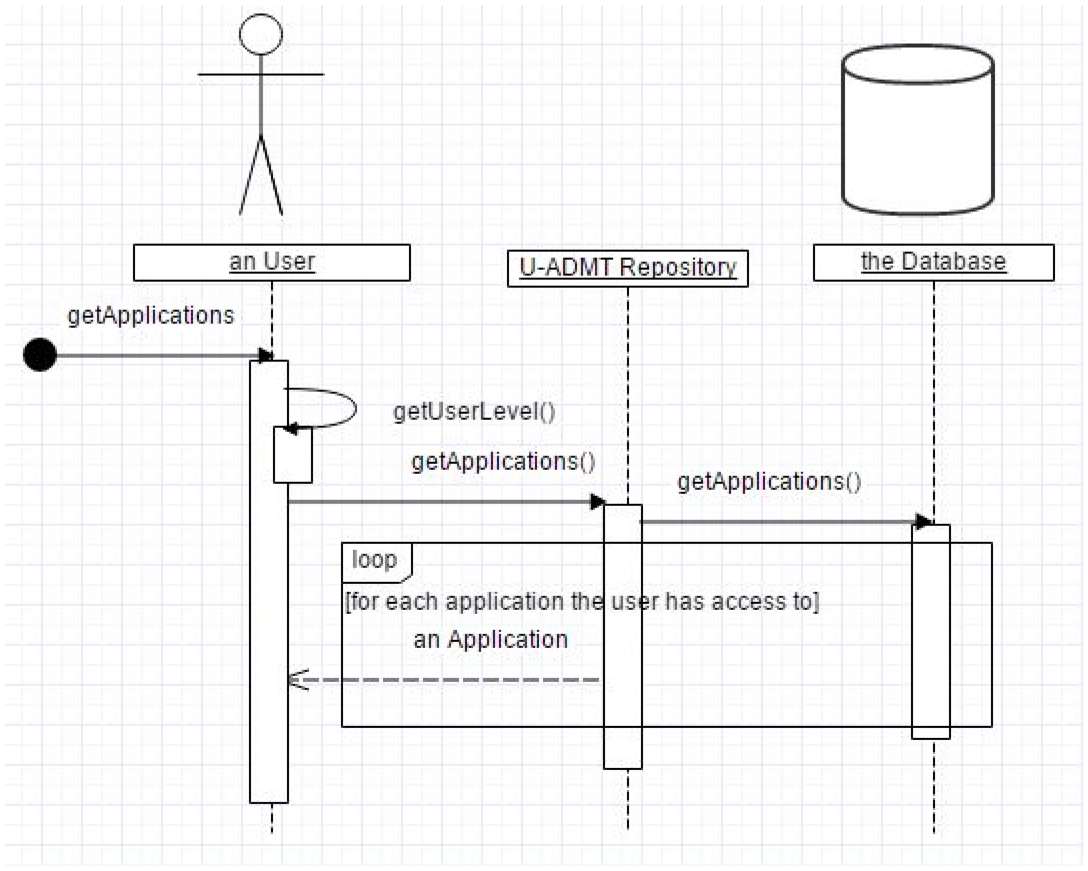

Figure 5.2: Sequence Diagram of 5.1.2 Access Application(s)

5.1.3 Assign Application to Faculty

An Admin user assigns a Faculty user to an Application for reviewing. This is done by first getting the Department associated with the Application, and then getting the Faculty from the matching Department. U-ADMT Repository is used to access the shared database to get each matching Faculty user. If the Application can be assigned to another Faculty user (i.e. there are less than 3 currently assigned Faculty to the Application) the Admin user then assigns a Faculty user to the Application along with a deadline.

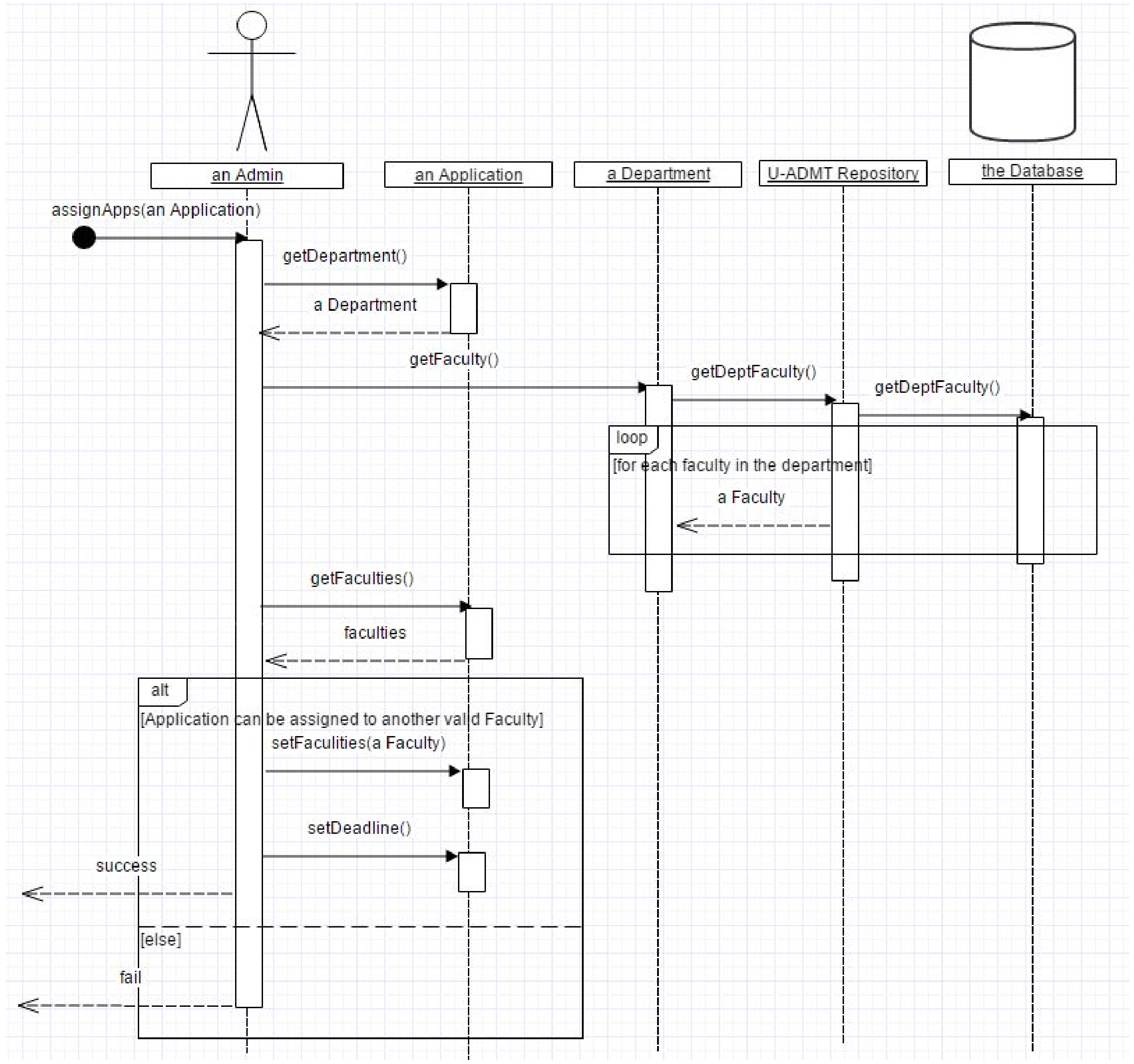

Figure 5.3: Sequence Diagram of 5.1.3 Assign Application to Faculty (larger version as Figure5.3.png)

5.1.4 Faculty Review

A Faculty user reviews an Application. To do this the system first creates a new Review. It then checks the Department the Application is for, and the minimum threshold values set in said Department. If the Application has values below the thresholds the system provides the user with option to Auto-Reject the application which sets the score to 0 and recommendation to "Reject". Otherwise, the Faculty user reviews the application by giving a score to each criteria, giving a recommendation, and optionally leaving a comment. Finally, if the user wants to submit the Application the system links the Review to the Application.
Sequence Diagram as Figure 5.4 on pg.48 (larger version as Figure5.4.png)

5.1.5 Admin Review

An Admin user reviews an Application by collating the scores, and taking the majority of the recommendations. To do this the system first checks the deadline of an Application, and verifies if it has passed. If it has it then grabs the Reviews for the application. There should, ideally, be 3 in total that have been submitted. However, if any have not been submitted the system changes the Review to an Auto-Reject. When the Admin user collates the scores the system gets the scores for each review, and sets the average as the score for the application. When the user takes the majority of the recommendations the system gets the recommendation for each review, and sets the majority as recommendation for the application. If there is no majority it sets the recommendation to "Waitlist". Additionally, if the Admin wants to submit a reply to the comments the system facilitates this by updating the comments in the Application. Finally, if the user wants to submit the Application the system does this by updating the Application information.
Sequence Diagram as Figure 5.5 on pg.49 (larger version as Figure5.5.png)

5.1.6 College Review

A College user reviews an Application by changing its status from "Undecided" to match the recommendation of the Application. To do this the system first gets the recommendation for the Application. If it is "Accepted" it checks if the Department the Application is applying to is filled. If it is then Application status is instead changed to "Waitlisted", otherwise it is changed to "Accepted". If the recommendation is "Waitlisted" or "Rejected" then the system changes the Application status to match. Finally the system updates the corresponding Applicant user of the decision by using MessageTransactor to trigger an email with the appropriate information through U-ADMT Repostiory.
Sequence Diagram as Figure 5.6 on pg.50 (larger version as Figure5.6.png)

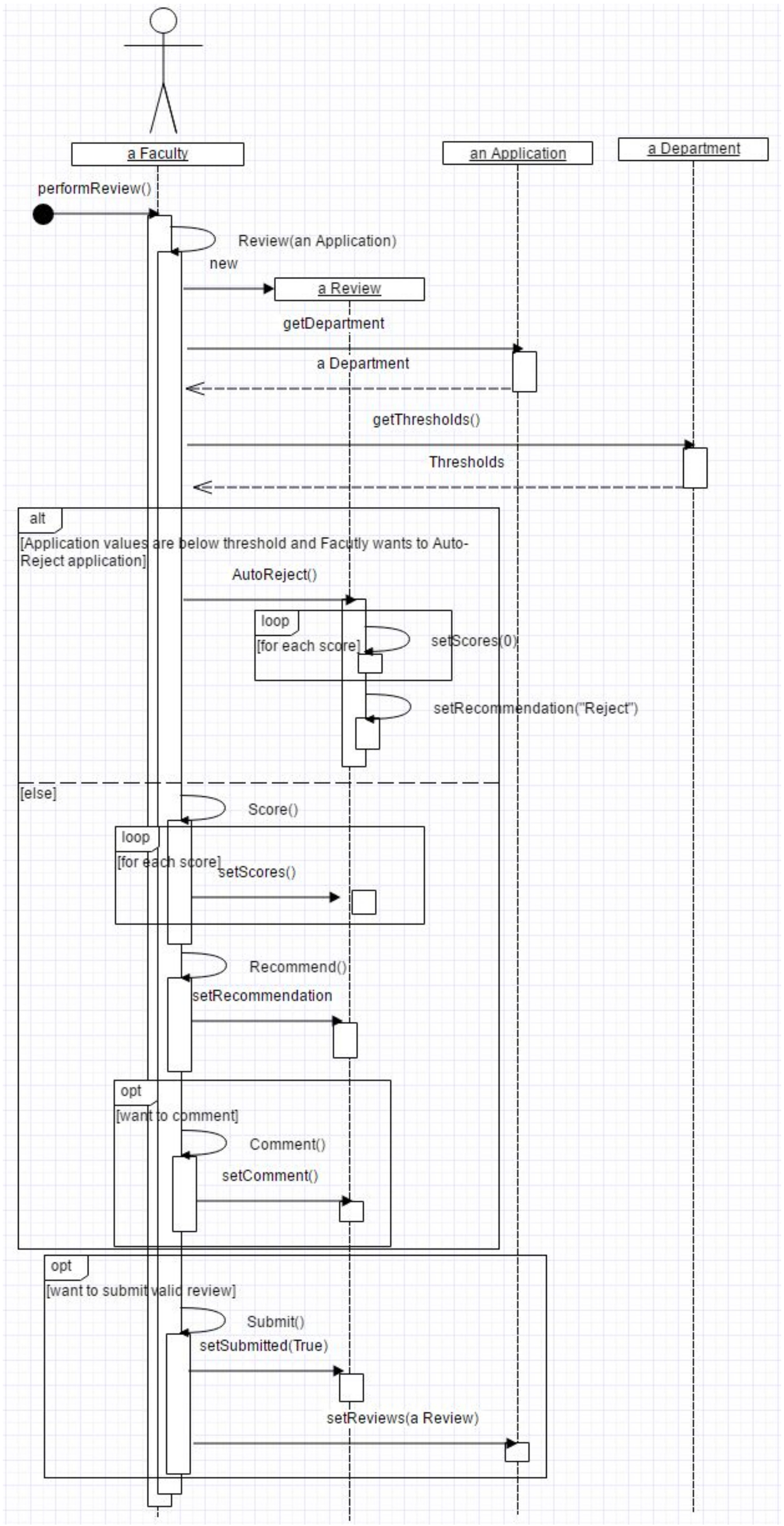

Figure 5.4: Sequence Diagram of 5.1.4 Faculty Review (larger version as Figure5.4.png)

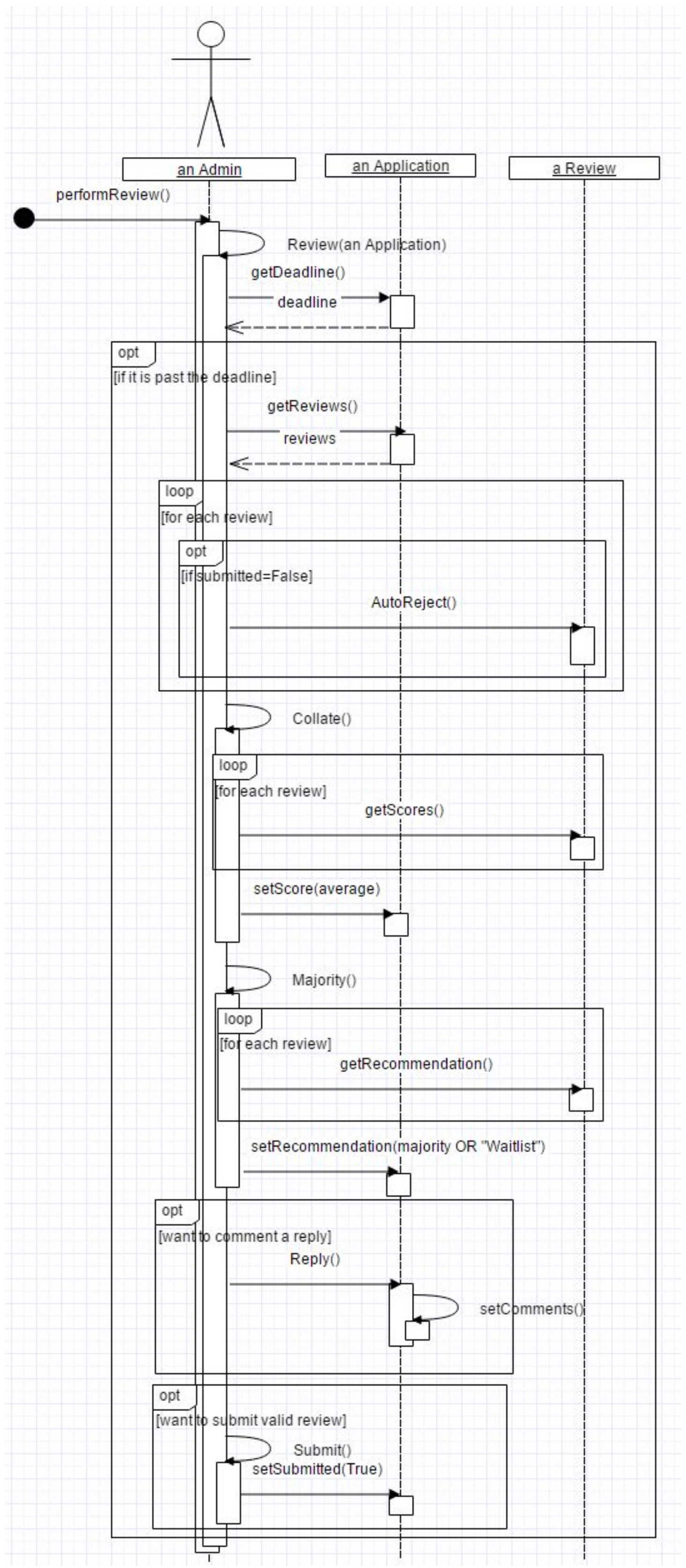

Figure 5.5: Sequence Diagram of 5.1.5 Admin Review (larger version as Figure5.5.png)

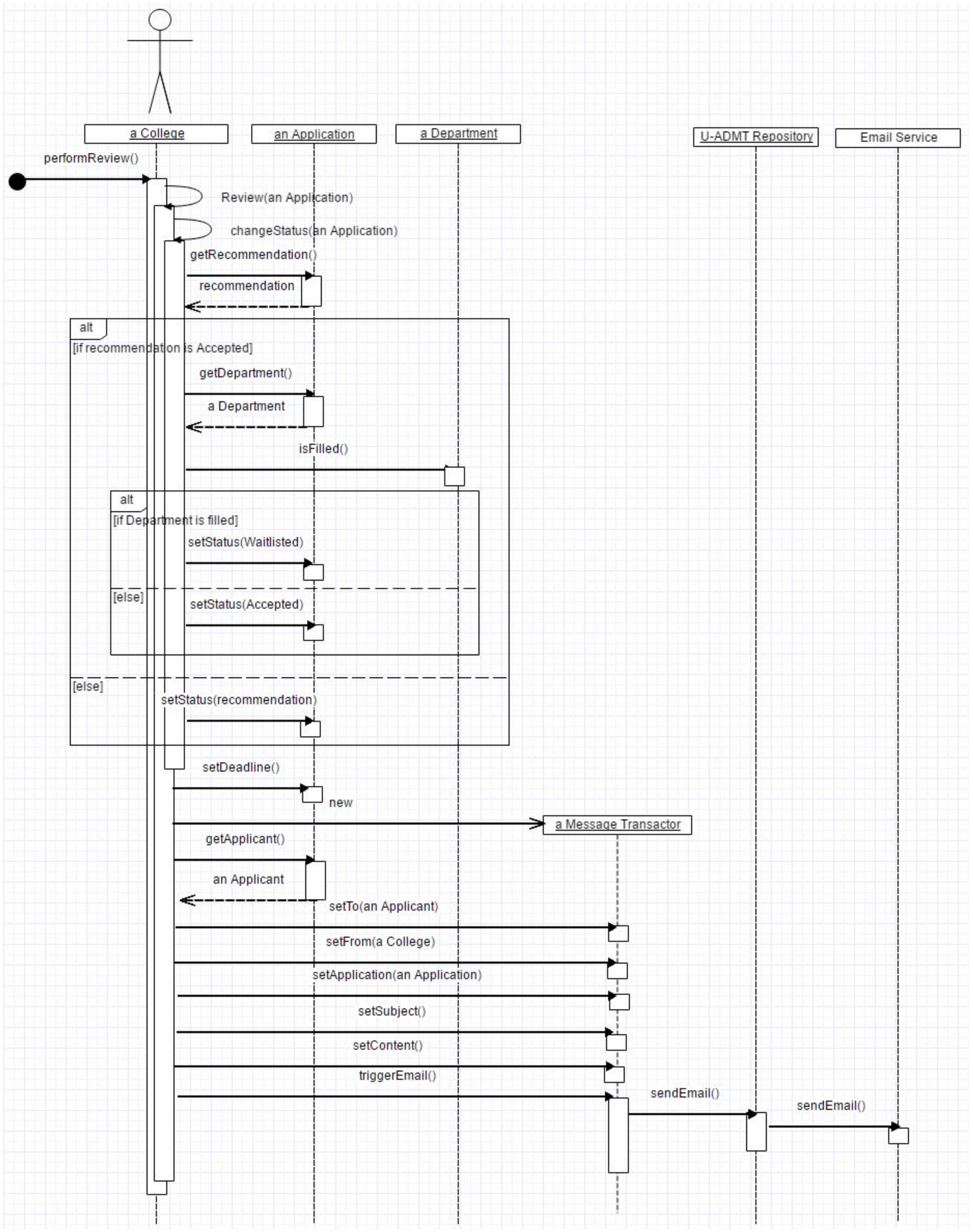

Figure 5.6: Sequence Diagram of 5.1.6 College Review (larger version as Figure5.6.png)

5.1.7 Applicant Final Decision

An Applicant user makes a final decision on an Application by confirming or rejecting an Accepted Application. To do this the system first checks that it is not past the deadline for making a final decision. If it is not, and the Applicant confirms the acceptance then the system updates the current seats of the department the Applicant has confirmed to. Additionally, an Applicant can only have one confirmed Application on their account so the system keeps track of this. If the Applicant rejects then the system deletes the Application

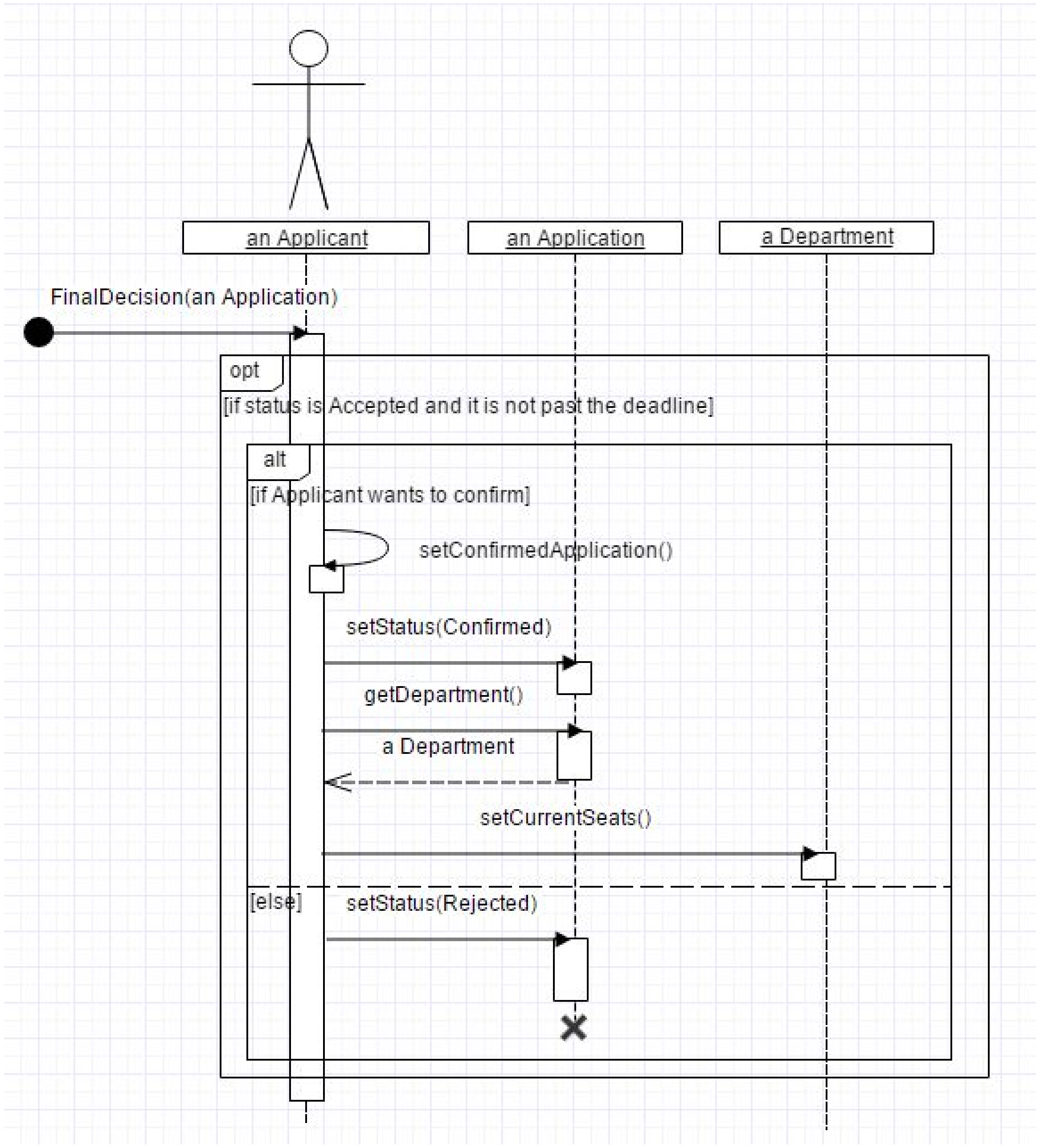
an Applicant
an Application
a Department
FinalDecision(an Application)
opt
[if status is Accepted and it is not past the deadline]
alt
[if Applicant wants to confirm]
setConfirmedApplication()
setStatus(Confirmed)
getDepartment()
a Department
setCurrentSeats()
[else]
setStatus(Rejected)

Figure 5.7: Sequence Diagram of 5.1.7 Applicant Final Decision

5.1.8 Filter Applications

A user filters applications based on some criteria. To do this the system creates a Filter based on the criteria the user input. It then gets the applications matching the criteria via U-ADMT Repository which interacts with the shared database.

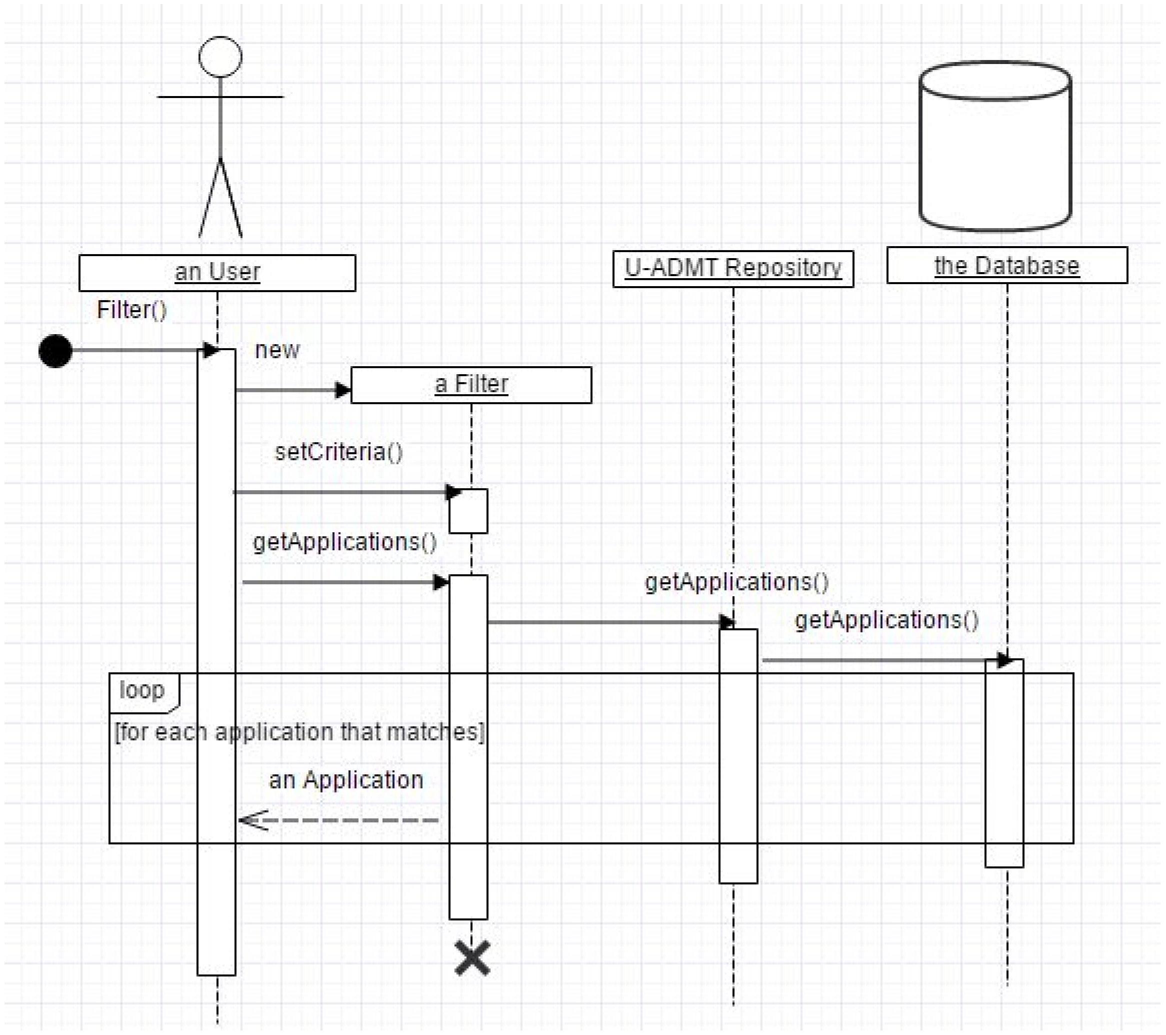

Figure 5.8: Sequence Diagram of 5.1.8 Filter Applications

5.1.9 Waitlist Applications

A college user waitlists applications. To do this the system gets the Department for each Application, and requests the corresponding waitlists for the Department. It then finds the Waitlist with the same type as the Application type. It then adds the Application to the correct Waitlist based on its score.

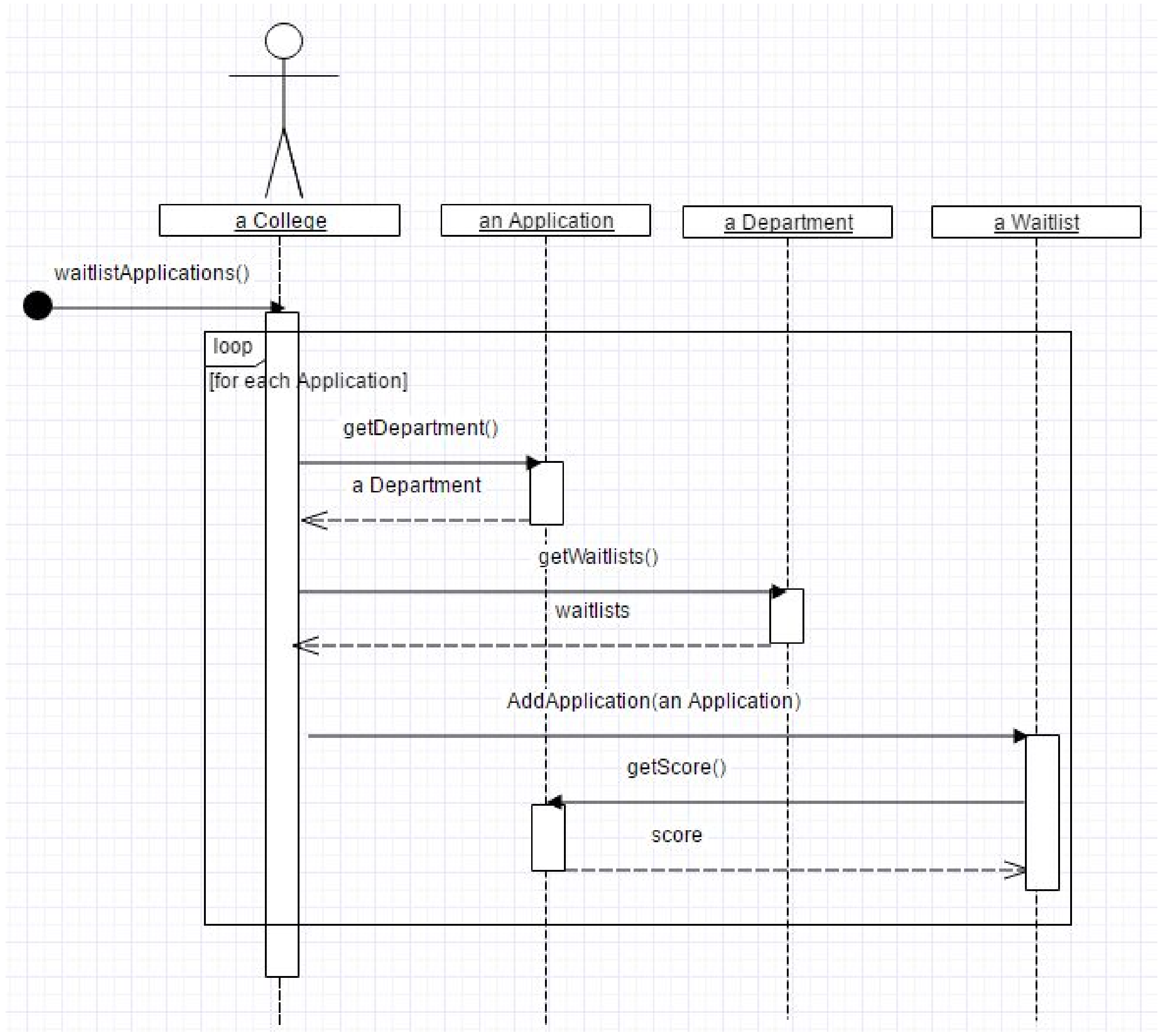

Figure 5.9: Sequence Diagram of 5.1.9 Waitlist Applications

5.1.10 Manage Department

A college user manages a Department. To do this if the Department does not exist yet the College user can create it. Additionally the College user can update the maximum seats and AutoReject thresholds of a Department by choosing to manage the seats and thresholds of a Department. The system facilitates this by getting and setting the maximum seats and thresholds for the Department.

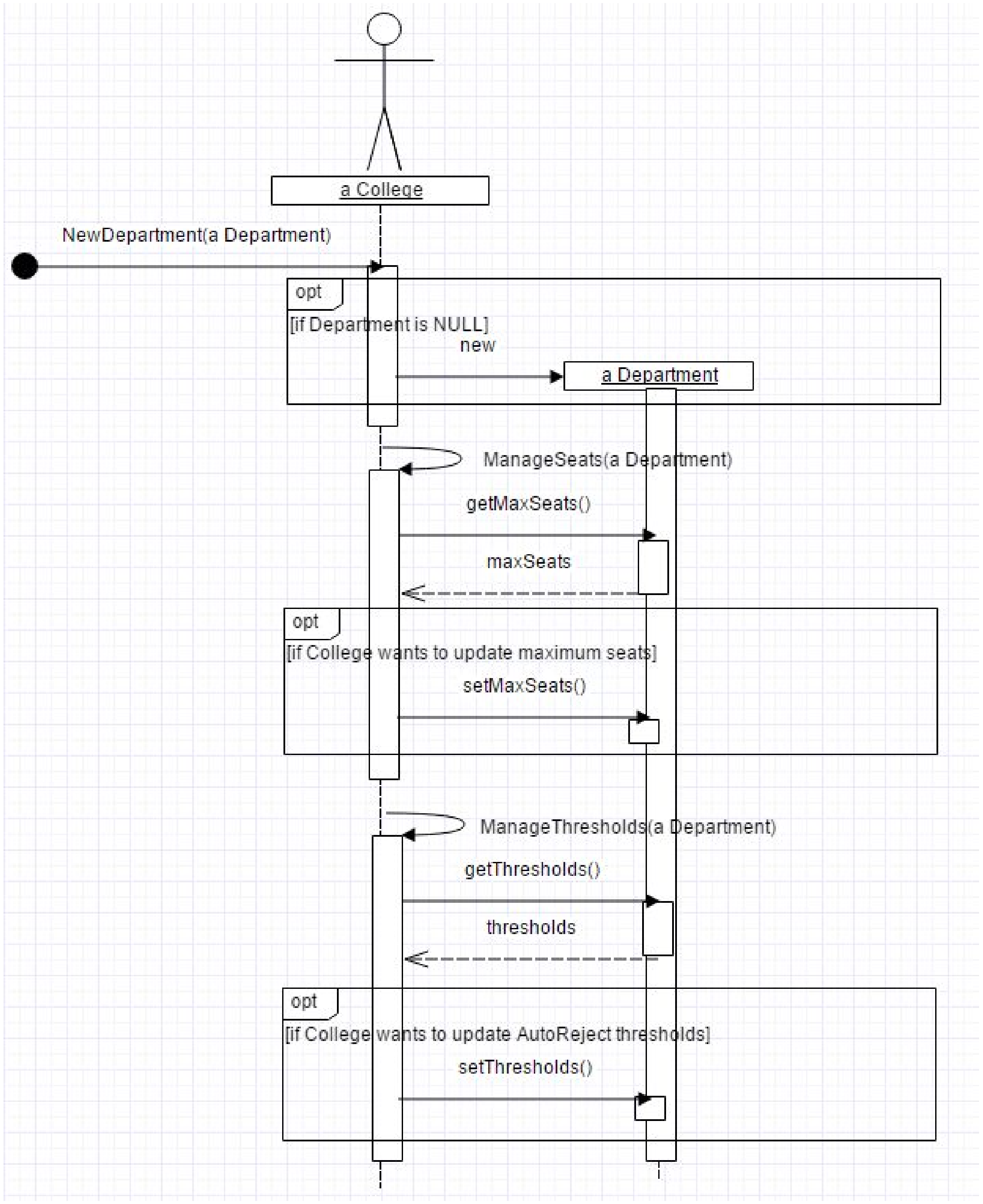
a College
NewDepartment(a Department)
opt
[if Department is NULL]
new
a Department
ManageSeats(a Department)
getMaxSeats()
maxSeats
opt
[if College wants to update maximum seats]
setMaxSeats()
ManageThresholds(a Department)
getThresholds()
thresholds
opt
[if College wants to update AutoReject thresholds]
setThresholds()

Figure 5.10: Sequence Diagram of 5.1.10 Manage Department

5.1.11 Manage Wait-list

A college user manages Waitlists for a Department. To do this if the College user wants to create a new Waitlist they must specify the desired Application type the Waitlist will accept. As long as the Department has less than 3 Waitlists and none of them are of the same type as the one the College user wants to create then the system will make a new Waitlist and associate it with the corresponding Department. Additionally, if the College user requests a certain number of Applications, X, from the Waitlist the system returns the appropriate Applications by going through the Waitlist from the top and returning X number of Applications as long as there are still Applications to return.

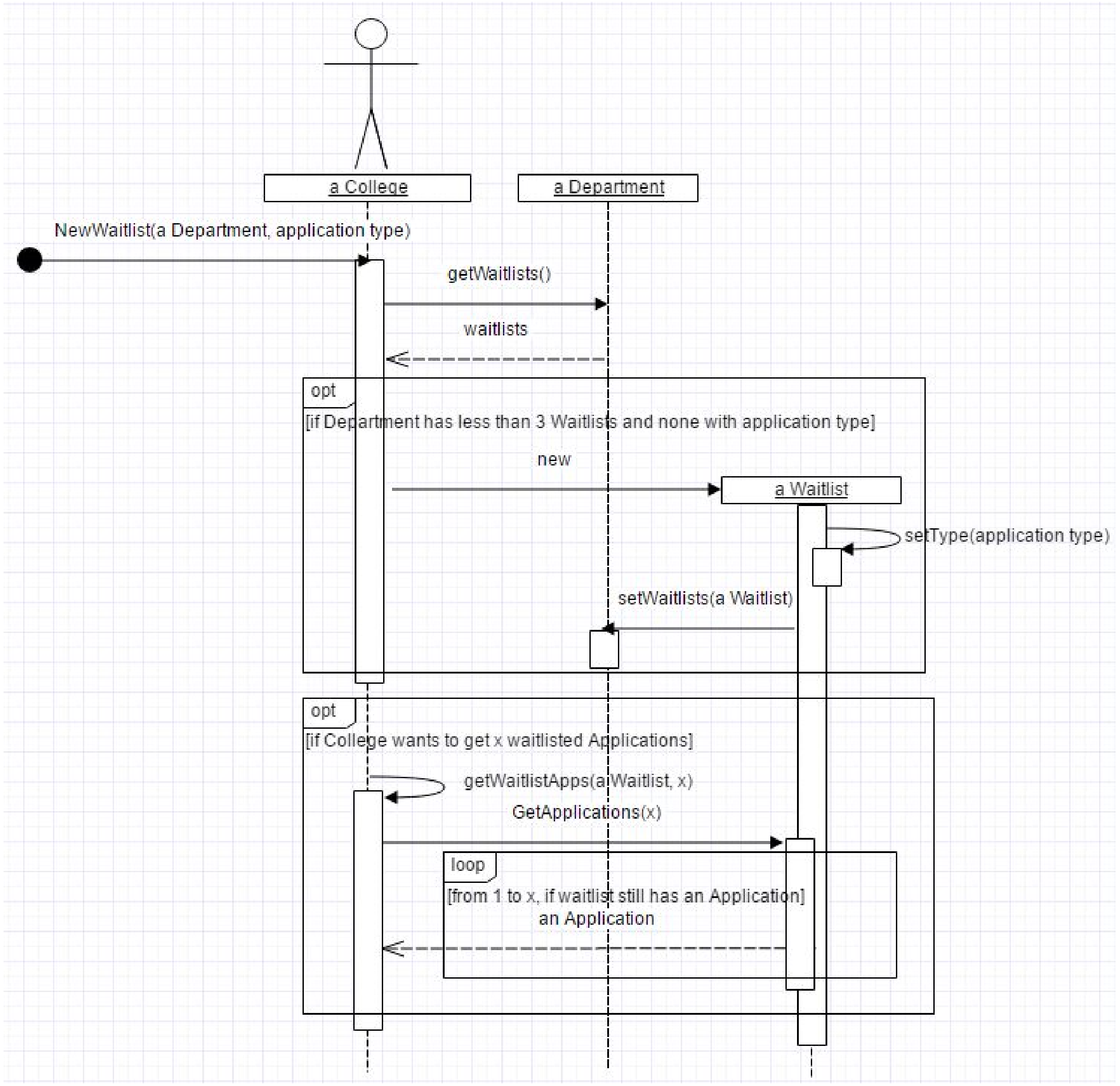
a College
a Department
NewWaitlist(a Department, application type)
getWaitlists()
waitlists
opt
[if Department has less than 3 Waitlists and none with application type]
new
a Waitlist
setType(application type)
setWaitlists(a Waitlist)
opt
[if College wants to get x waitlisted Applications]
getWaitlistApps(a Waitlist, x)
GetApplications(x)
loop
[from 1 to x, if waitlist still has an Application]
an Application

Figure 5.11: Sequence Diagram of 5.1.11 Manage Waitlist

6 Non-functional requirements

The U-ADMT system should be always keep information safe, which includes restricting access for each level of user and not leaking information to unauthorized users. This is addressed in the design by keeping track of attributes for many classes that would be useful for security purposes such as User levels and ids. This is also addressed by keeping Applicants as a separate class to the other User levels meaning it is designed to not have access to higher level information contained in system classes such as Filter, Department, and Waitlist.

It is also important that U-ADMT be flexible to future changes. This is addressed in the use of design patterns. For example, if another user level is desired it can be easily added as a subclass of User with its review behavior either being the same as an already defined one or added as a subclass of ReviewBehavior. Additionally, if another Applciation type is desired it can simply be added as a subclass of Application that gets created by ApplicationFactory. Additionally the use of multiple subsystems makes it so that changes in one area shouldn't too greatly impact other areas.

Portability and ability to interact with other systems is also important to U-ADMT. This is addressed in the architecture of the system. By using a facade to facilitate interaction between the U-ADMT Repository interface and external interfaces allows the U-ADMT information to be encapsulated from the overall system.

The system changeability, stability and response time will heavily rely on database and other external components.

7 Supplementary Documentation

The reader may find it useful to use the included larger versions of the following figures:

- Figure 3
- Figure 4
- Figure 4.1
- Figure 4.2
- Figure 5.1
- Figure 5.3
- Figure 5.4
- Figure 5.5
- Figure 5.6