CONCORDIA UNIVERSITY

COEN 6312

MODEL DRIVEN SOFTWARE ENGINEERING

DELIVERABLE 4: State Machines

Team : EmBridge

40053363 - Ishaan Sharma
40035779 - Kishor Tare
40043592 - Nidhi Patel
40048878 - Rashmi Narayan
40071278 - Yassine Jebbar

Guided By:
Dr. Wahab Hamou-Lhadj
Mohammad Reza Rejali

March 22, 2019

Contents

List of Figures

List of Tables

Introduction

This document contains the **State Machine Diagram** of four main classes and **Action Specification** of three operations of the system. The Action Specification Language used is **Java**.

Team Workflow

During this deliverable, the team often made regular meetings to discuss the tasks and align the doubts. The objective of each meeting is described in Table 1. :

Table 1: Team meetings and milestones.

Date	Discussion
March 8	Review on Deliverable Description and Requirements; Upcoming Meetings Schedules.
March 10	Detailed discussion on Deliverable 4; State diagram of classes Account and Post
March 15	State diagram of classes Group Request and Message; Analysis of all state diagrams
March 17	Action Specifications for 3 operations
March 22	Final discussion and modifications on deliverable 4; Submission of Deliverable 4

1 State Machine Diagrams

A state diagram describes the behavior of a single object in response to a series of events in a system.[2] This paper describes the states of classes Account, Post, GroupRequest and Message chosen from the Class Diagram submitted in Deliverable 3. The following state diagrams include various notations such as State, Transition, Events, Decision Node such as Fork and Join and Terminated.

1.1 Account

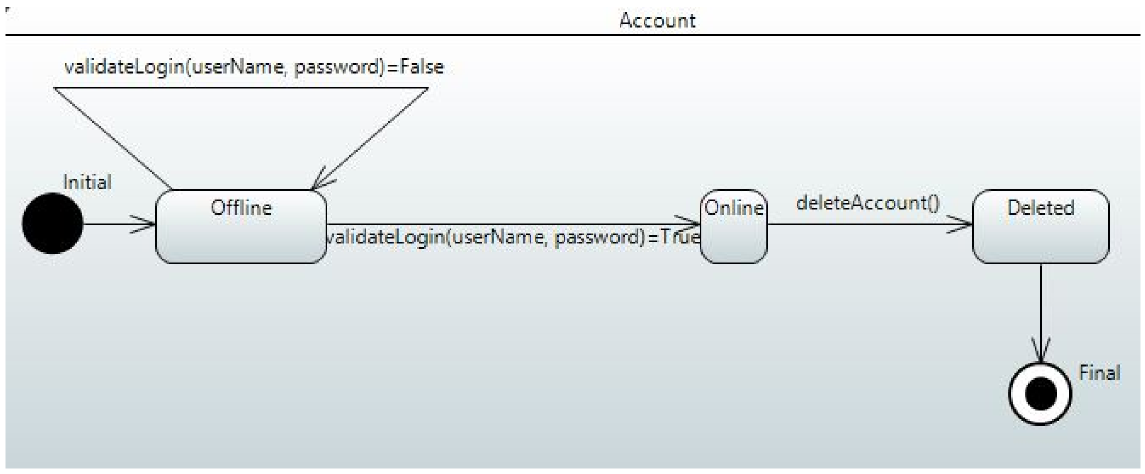
Account
validateLogin(userName, password)=False
Initial
Offline
validateLogin(userName, password)=True
Online
deleteAccount()
Deleted
Final

Figure 1: Account State Diagram

An account has states such as **Offline, Online and Deleted**. It is initially in offline state as soon as it is created. To login and change the state to online, the *validateLogin(userName, password)* method should return True. If it returns False, i.e., the login does not get validated, the account remains in offline state. An account can be deleted by executing *deleteAccount()* method.

1.2 Post

A Post can have states such as **Empty, Ready, Published and Deleted**. A post is initially empty. Upon executing *addContent(Content Type c)* method, i.e., adding contents such as text, image or video, it will be in ready state. Adding more contents by *addContent(Content Type c)* at this state will hold post in ready state. After adding the post by *addPost()*, it will be published on the wall and visible for others. A post can be edited by *updatePost(Content Type c)* but the state of post will remain as published since its behaviour remians the same. The post can be deleted by *deletePost()* method which will delete the post and then terminate the object.

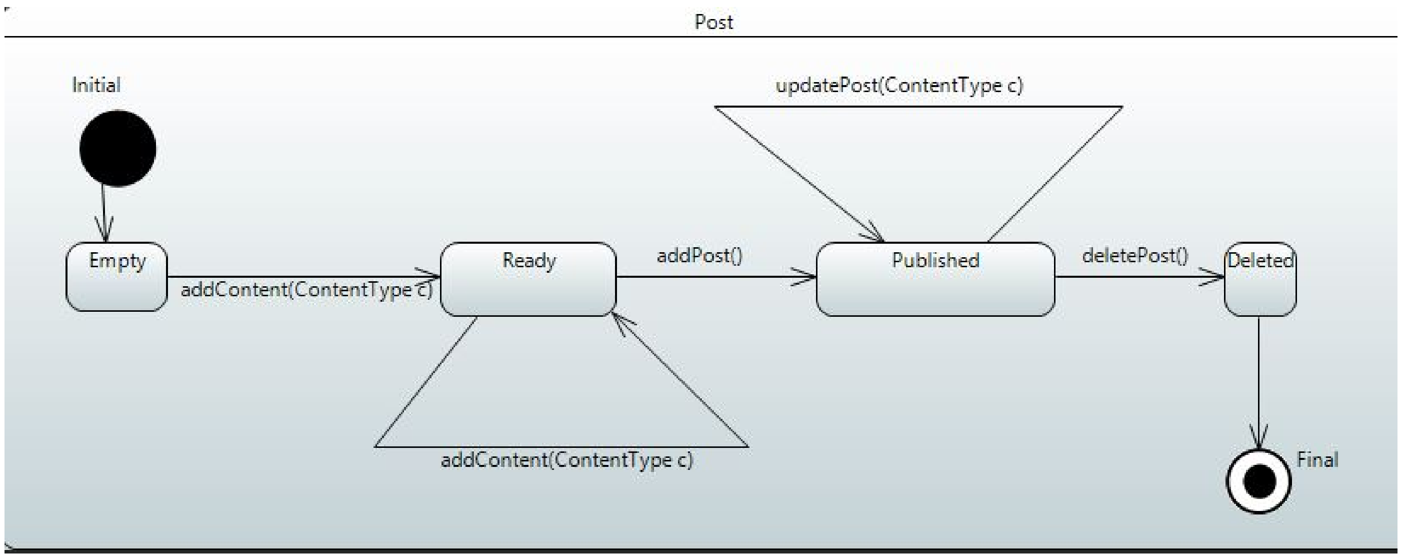
Post
Initial
updatePost(ContentType c)
Empty
addContent(ContentType c)
Ready
addPost()
Published
deletePost()
Deleted
addContent(ContentType c)
Final

Figure 2: Post State Diagram

1.3 GroupRequest

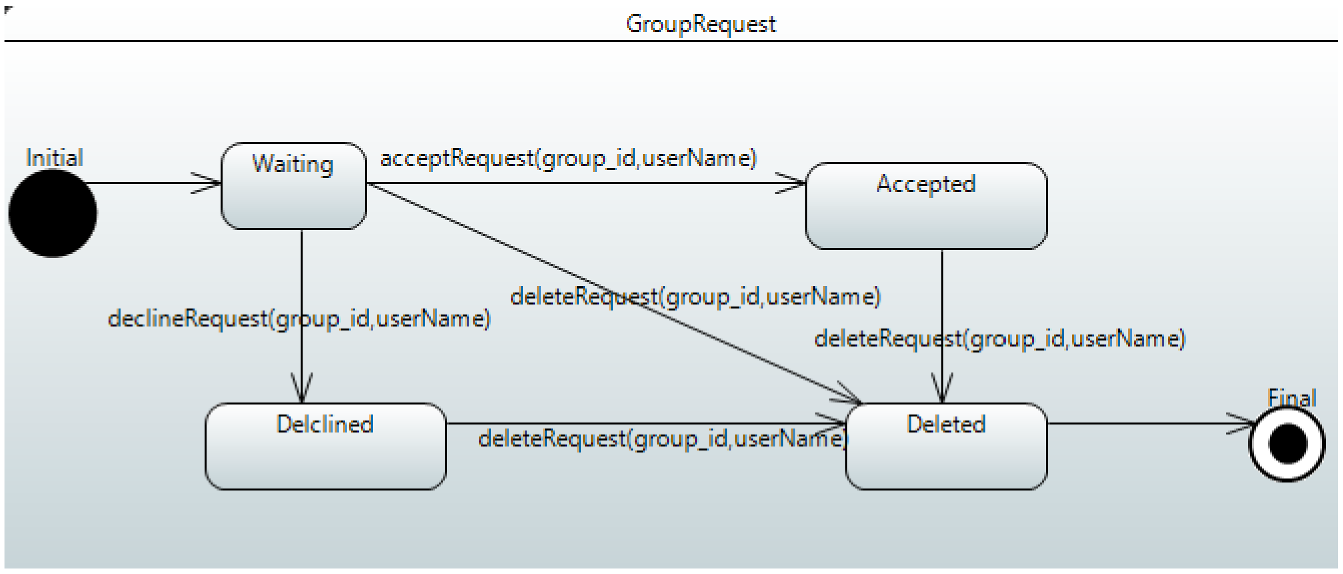
GroupRequest
Initial
Waiting
acceptRequest(group_id,userName)
Accepted
declineRequest(group_id,userName)
deleteRequest(group_id,userName)
deleteRequest(group_id,userName)
Delclined
deleteRequest(group_id,userName)
Deleted
Final

Figure 3: Group State Diagram

A GroupRequest can have states such as **Waiting, Accepted, Declined and Deleted.** A GroupRequest is initially in waiting(state) for requests to be accepted or declined in repository. When *acceptRequest(group_id, userName)* is operated, a User is accepted in a group. Hence, State of the object is Accepted. On the other hand, When *declineRequest(group _id, userName)* is executed, a User is declined to be added in a group which will change the state of the object from

Waiting to Declined. Once the request is accepted or declined, the request should get deleted. *deleteRequest(group_id, userName)* will delete the object of request and the state of the class will change from Accepted or Declined to Deleted. However, object of the request can also be deleted directly when it is in waiting state. After this state, the state of the object is terminated.

1.4 Message

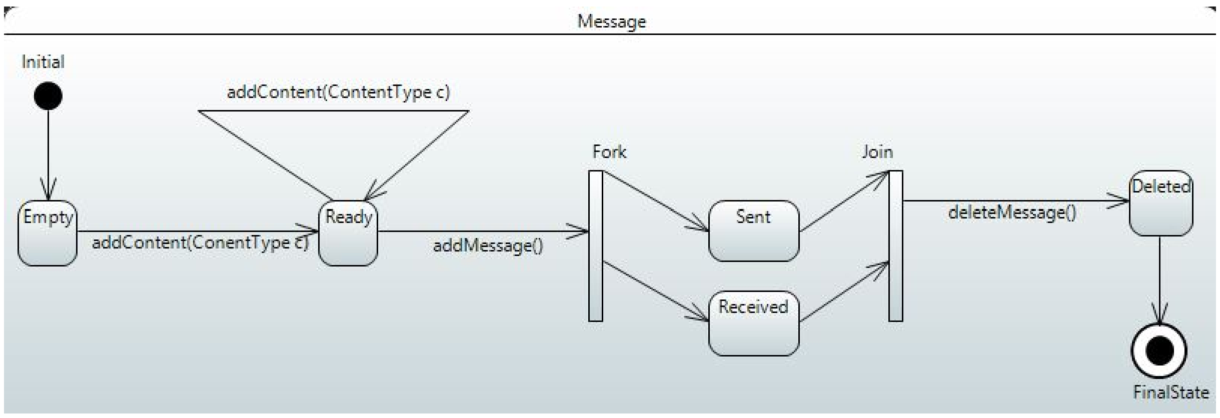

Figure 4: Message State Diagram

A message can have states such as **Empty, Ready, Sent, Received and Deleted**. A message is initially empty. Upon executing *addContent(Content Type c)* method, i.e., adding contents such as text, image or video, it will be in ready state. Adding more contents by *addContent(Content Type c)* at this state will hold message in ready state. After adding the message by *addMessage()* method, the message object will be sent, at the same time received by the receiver. Therefore, this parallel existence of states is represented by **fork** and **join**. The message can be deleted by *deleteMessage()* method after which it gets terminated.

2 Action Specifications

2.1 validateLogin

This method is used to verify username and password when account object is trying to access logIn Session. To do so, we need to fetch account object's userName from AccountDao class using findByName(String userName) which will return and check respective object's username exists or not. Then, validateLogin(String userName, String password) will compare the password associated with the username with the one entered. If the password matches, it returns True. Else, it returns False.

```
/**********************
 * locate user by its username
 *
 * @return user object
 **********************/

public User findByName(String userName) {
    return accountDao.findByUsername(userName);

}

/*************************
 * Validate authentication of user account.
 *
 * @param userName of type string is a unique username of an * employee.
 * @param password of type string is password related to an  * employee account.
 **************************/

public boolean validateLogin(String userName, String password) {
    AccountDao accountDao = new AccountDao();
    User user = accountDao.findByUsername(userName);
    if (user.password.equals(password)) {
        return true;
    }
    else {
        return false;
    }
 }
```

2.2 createPost

This method creates the post and publishes it at the same time using the class PostBuilder. It also instantiates post object by adding the content such as text,image and video. It also displays the content once added.

```
public void createPost(){
    PostBuilder postbuilder = new PostBuilder();
    Post textPost = postBuilder.createTextPost();
    Post imagePost = postBuilder.createImagePost();
    Post videoPost = postBuilder.createVideoPost();
    textPost.showContent();
    imagePost.showContent();
    videoPost.showContent();
}
```

2.3 acceptRequest

This is the action taken by admin to accept the group join request. The function first fetches the group id to which the request is sent and the username of account that sent the request. The username is then added to the list of members and the request is deleted.

groupDao is a data access object class with **Runtime Declaration** presented as ”repository interface” used mainly for ”CRUD Operation”. [1]**CrudRepository** is a Spring data interface, and to use it, we need to create our interface by extending CrudRepository for a specific type.[1] Spring provides CrudRepository implementation class automatically at runtime.”[1] It contains methods such as **save, findById , delete , count** etc.

```
/*************************
 * @param userName is a user name from Account
 * @param group\_id is unique group id
*************************/

  public Group acceptRequest(string userName, long group_Id) {
      Group group = groupDao.findById(group_id);
      User user = groupDao.findByName(userName);
      group.getRequests().remove(user);
      group.getMembers().add(user);
      return groupDao.save(group);
  }
```

References

[1] ”Spring Data CrudRepository Example. https://www.concretepage.com/spring-5/spring-data-crudrepository-example.

[2] ”State Diagrams - Everything to Know about State Charts”. https://www.smartdraw.com/state-diagram/.