First Iteration Final Report:
mycheapfriend.com

COMS 4156: Advanced Software Engineering

Team: CheapSkates

Team Members

Double Trouble: Waseem Ilahi (wki2001@columbia.edu)

Shaoqing Niu (sn2385@columbia.edu)

Dynamic Duo: Michael Glass (mgg2102@columbia.edu)

Huning "David" Dai (hd2210@columbia.edu)

The **Cheapskates** from **MyCheapFriend.com** will lend you our **Final Report,** if you give us feedback next week.

Table of Content

2. Revised Schedule: (OMITTED)

3. Revised Schedule: (OMITTED)

4. Revised Schedule: (OMITTED)

5. Revised Schedule:

We successfully completed all the work before the demo, which took place on the 6th of November. We had the demo earlier, because our system was ready for the demo by then.

The revised schedule is attached at the end of this document. It shows that all the work has been completed for the 1st iteration.

6. High Level Architecture: (OMITTED)

7. Component-Level Design:

Before we go any further, we would like to acknowledge, a couple of referenced to the code that was acquired from the outside sources. First, the code to read the email, in the files; EmailRead.java, EmailSend.java and GmailUtilities.java was acquired from the online blog with the url: http://forums.sun.com/thread.jspa?threadID=5267916 and second the code for the password generator, in the file named, PasswordGenerator.java was acquired from the online blog: http://www.glenmccl.com/tip_010.htm

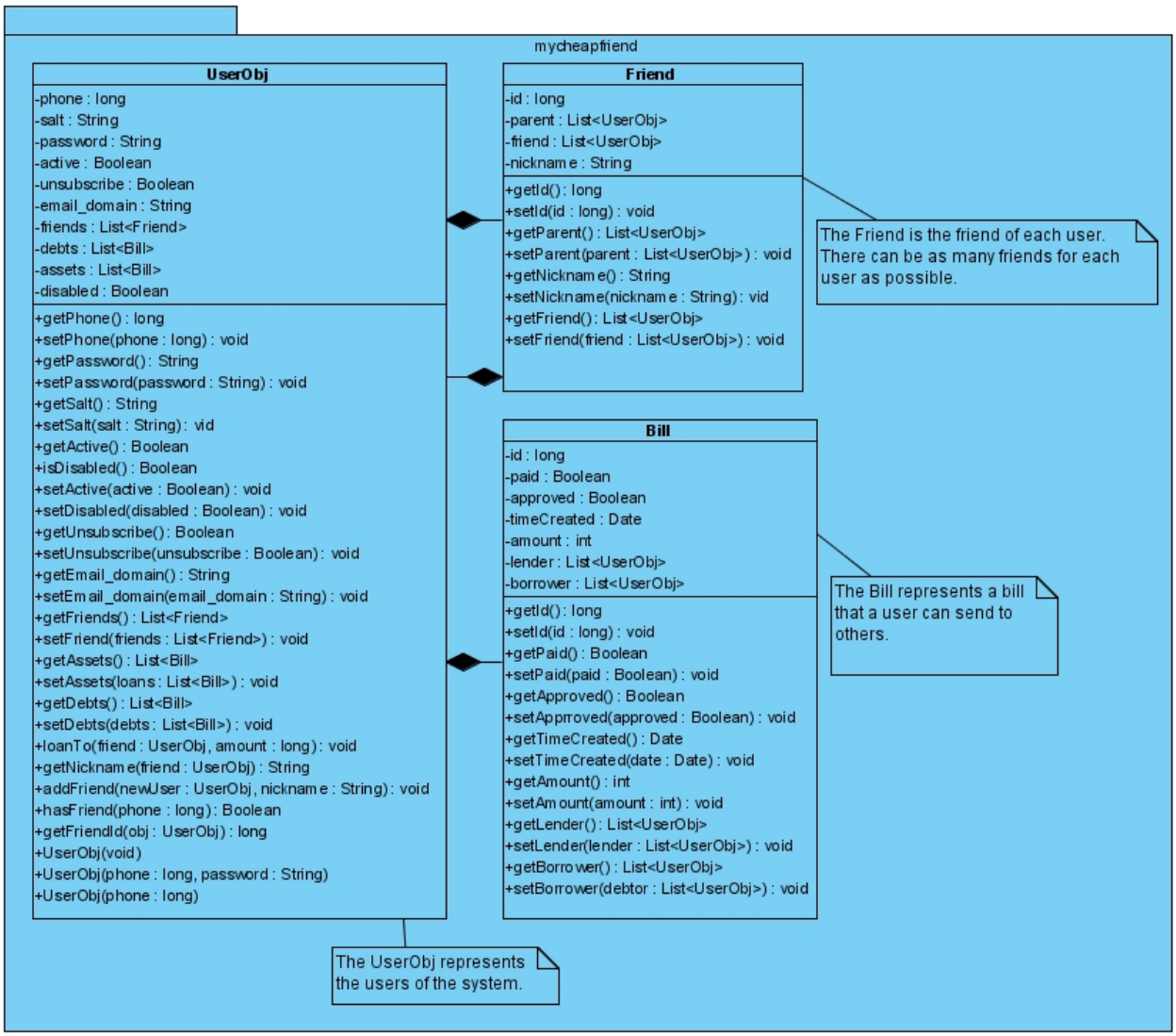
This code was then modified and the additions were made to it to fit our needs.

Now, we do have our original design, as there was no change made to the design, but we will like to go a little bit into the detail, for the web UI and possibly a couple more classes that we have added. The rest is the same as the previous report.

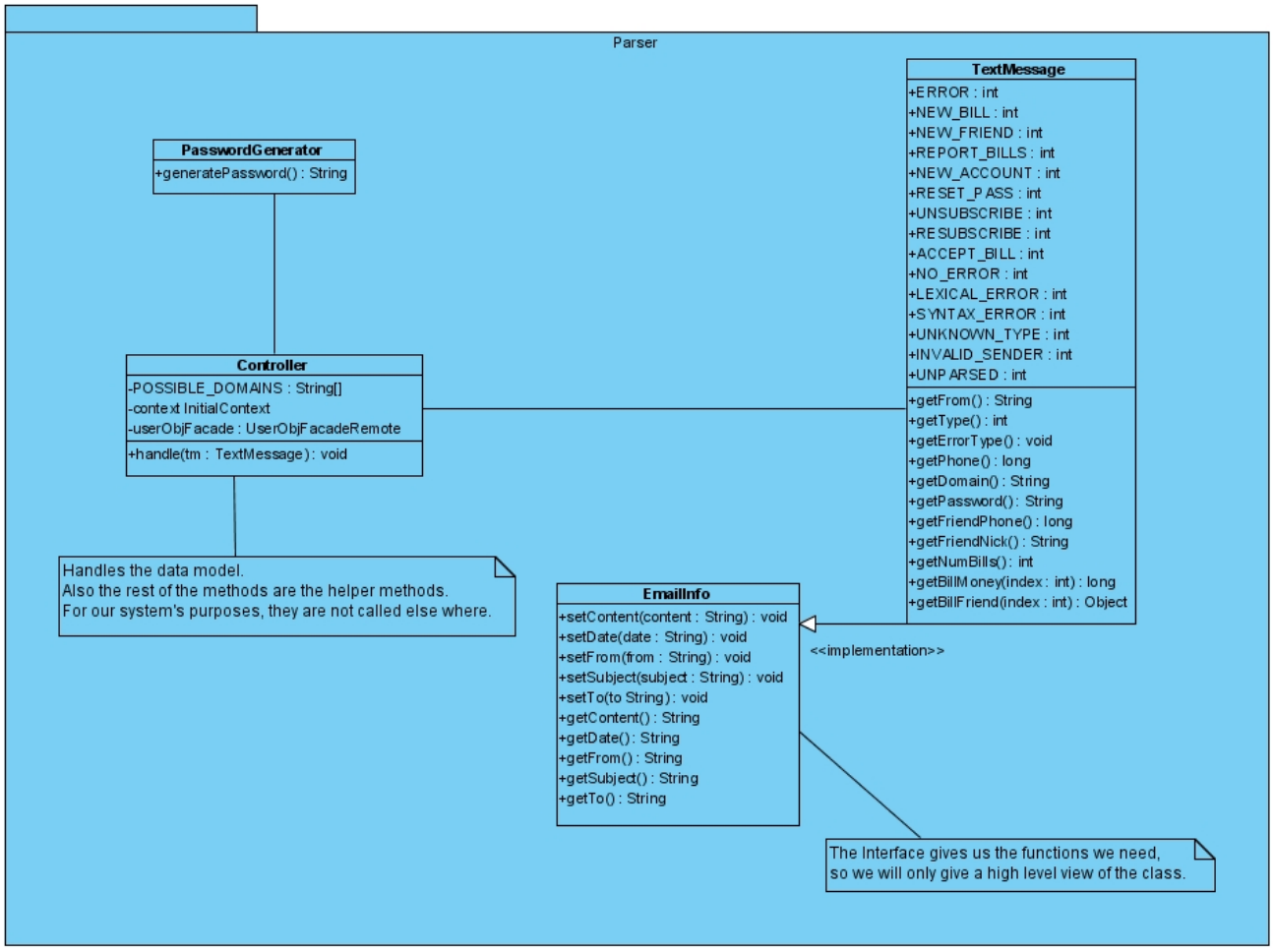
Our system basically has four main components (tasks); the data model (entity setup), the business logic (controller), the E-Mail Wrapper (Email Info….) and the Web UI (admin only).

A high level UML structure below will try to explain the four sections.

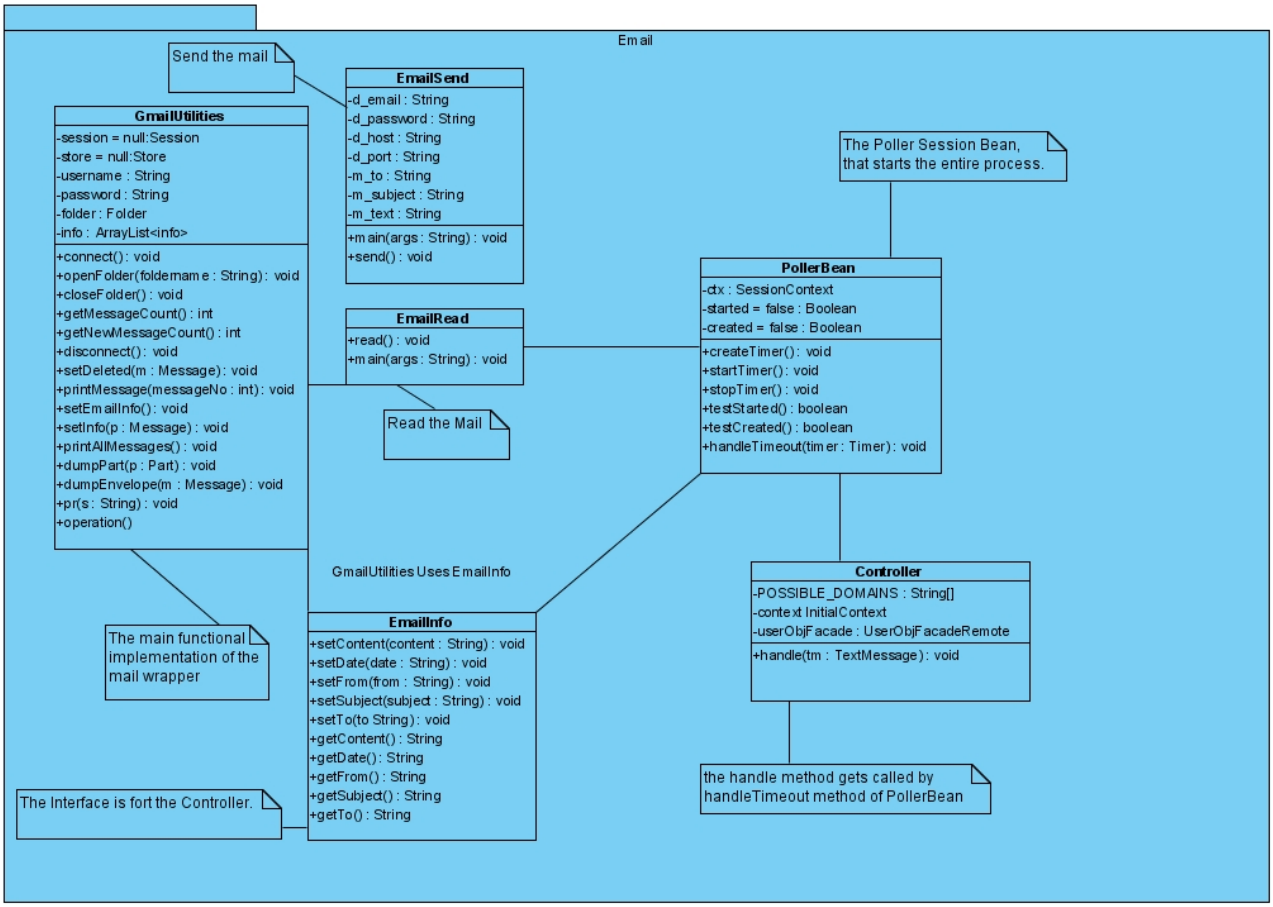
First off, the data model; there are three “tables/entities” basically, the “UserObj”, “Friend” and the “Bill”. To facilitate the communication from the UserObj class and the client side, we have created a “façade/session bean”; UserObjFacate.



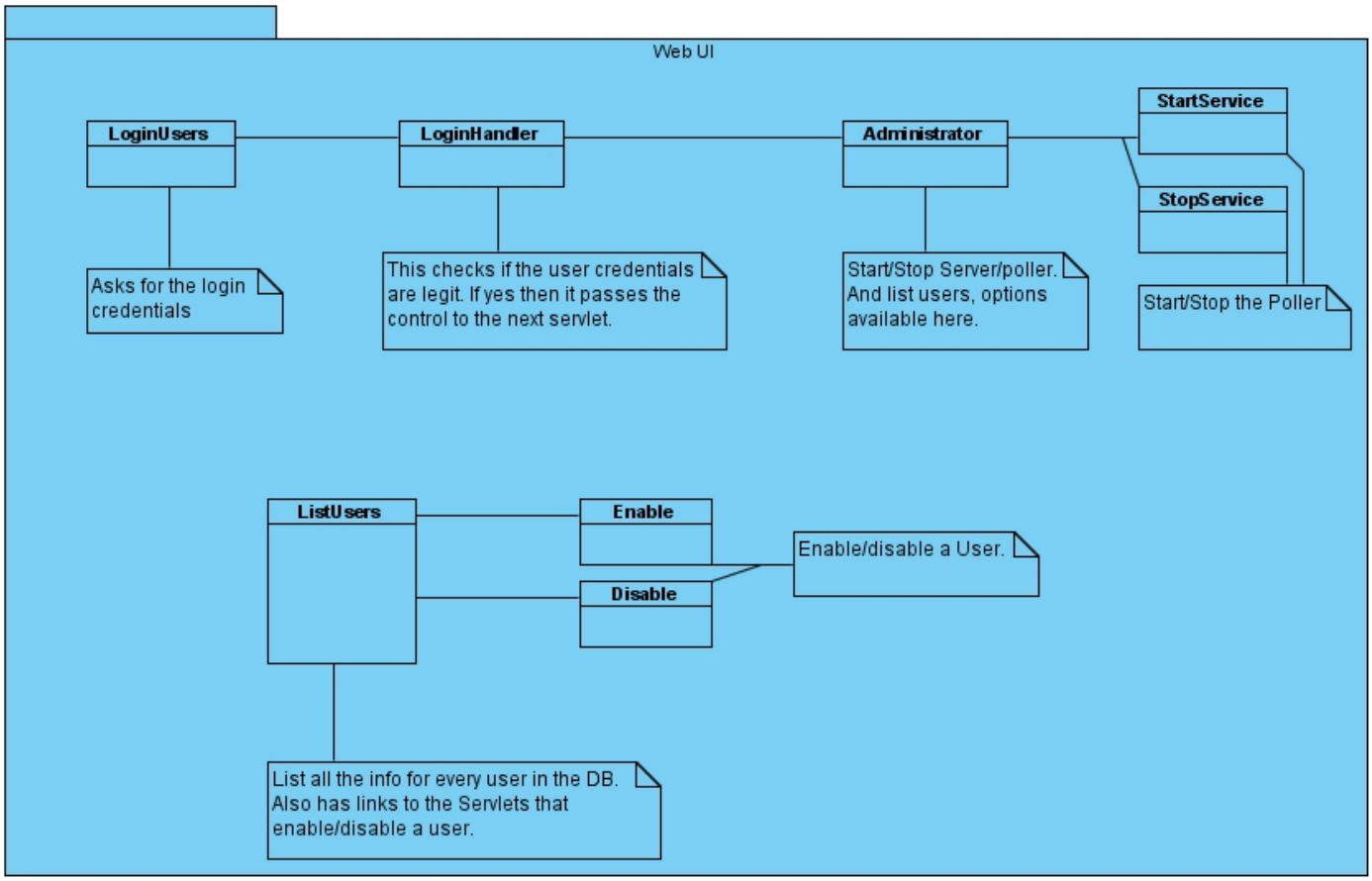
The Business Logic/Controller is basically the Controller.java file. This file has a handle(text) message. It utilizes the text with the help of the TextMessage.java interface. When the poller's timeout method is called, it first reads all the emails from the inbox and then gives the messages to the handle( ) method one by one. Following diagram shows the high level view of the Controller.java file and the TextMessage.java interface.



We use JavaMail API to implement the interaction between users and our system. For this purpose, we build a poller that checks for new emails in our robot account (which is now using Gmail) via POP every few seconds. The Poller is a session bean. The poller collects new emails every five seconds and passes them to the email parser for both syntactic and semantic processing. It then calls the handle () method of the controller, which handles the backend functionality. Whenever our system is going to provide outputs, it does so by sending emails to the users' accounts. The sending part is realized using SMTP. For the following diagram, we have assumed that the getters and setters for all the fields are present in the class.



The Web UI is for the administrators only, which currently means us CheapSkates. It will give us, the admins, the web access to the database; a front-end for manipulating tables. There are multiple Servlets, first we login then, and then we have the choice of starting stopping the poller. Finally, we see the list of all the users in the DB. We can enable and disable a user to block them from using the MCF services. A high level view of these Servlets is shown below.



8. Component Model Services:

Our usage of the Component Model Services did not change, since the last report. We are, however, going to include the same description from the last report that explains the utilization of the services.

We are using EJB for persistence (Entity beans) as well as messaging (for queuing). We are also using JavaMail and Servlets as a front-end.

We had hoped that implementing our Entity beans would be as simple as the demo in class had been. When implementing, we had difficulties both creating the nested relations between our entities, and much larger difficulties using complicated (non-standard) primary keys to index our entries in the database. The vastness of the EJB libraries makes it difficult to access the meat of getting a lot of what we want to get done. It seems like we're hitting a nail with a sledge hammer.

That being said, requirements are requirements: We're receiving e-mail, somewhat asynchronously, via POP (checking the server via a poller), sending e-mail via SMTP. Business logic is performed and the messages are queued for entity processing. All of the messages we receive have all of the state information we need for processing, so we need to keep no user state beyond that which we keep in the database, so there is no reason to keep further session information, and thus our business logic is done in a stateless manner.

We use Java Mail, which is often used with JMS, however we use it for user i/o instead.

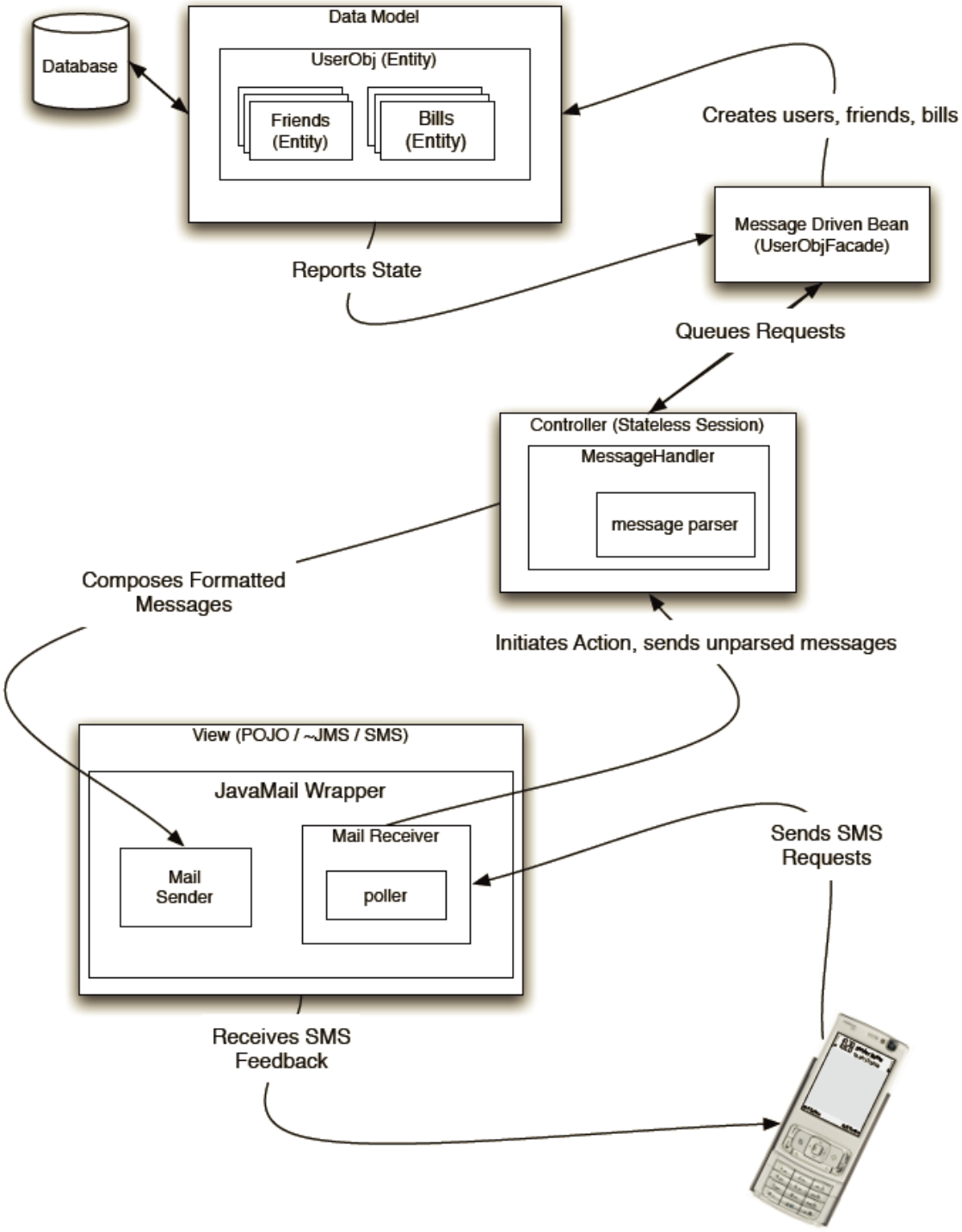
All the messages received are manipulated by a Message Handler as a general Controller which is a stateless session. When sending a message, the Message Handler composes formatted messages and forwards it to the Mail Sender. When receiving a message, the message receives data generated by a message parser. Here, message parser is used to check the type of a message and tell whether the message is legal. The message parser also converts an incoming message to a standardized form so that it could be handled by Message Handler easier.

The Message handler also controls communication between the mail system and our data model. All the interfaces with data model are integrated in the UserObjFacade and a message driven bean will be used. Related commands sending to the data model include "Create a user", "Create a friend", "Request a bill" and so on. There are also data sending from data model to the message handler as the response of commands.

The data model is implemented with entities (User Obj) since we need to keep record of users' information. In this object, there are entities for friends and bills. A user may have several friends but a friend belongs to only one user. Bill is a transaction between a user and his friend. Creation of friends and bills are done according to the messages sent by the Message Handler.

To sum up, java Mail serves as the frontend while Entity Beans serves for the backend. Stateless session and message handler are used for communication.

The diagram from the "High Level Architecture" shows different framework components in action. We are going to paste it here again.

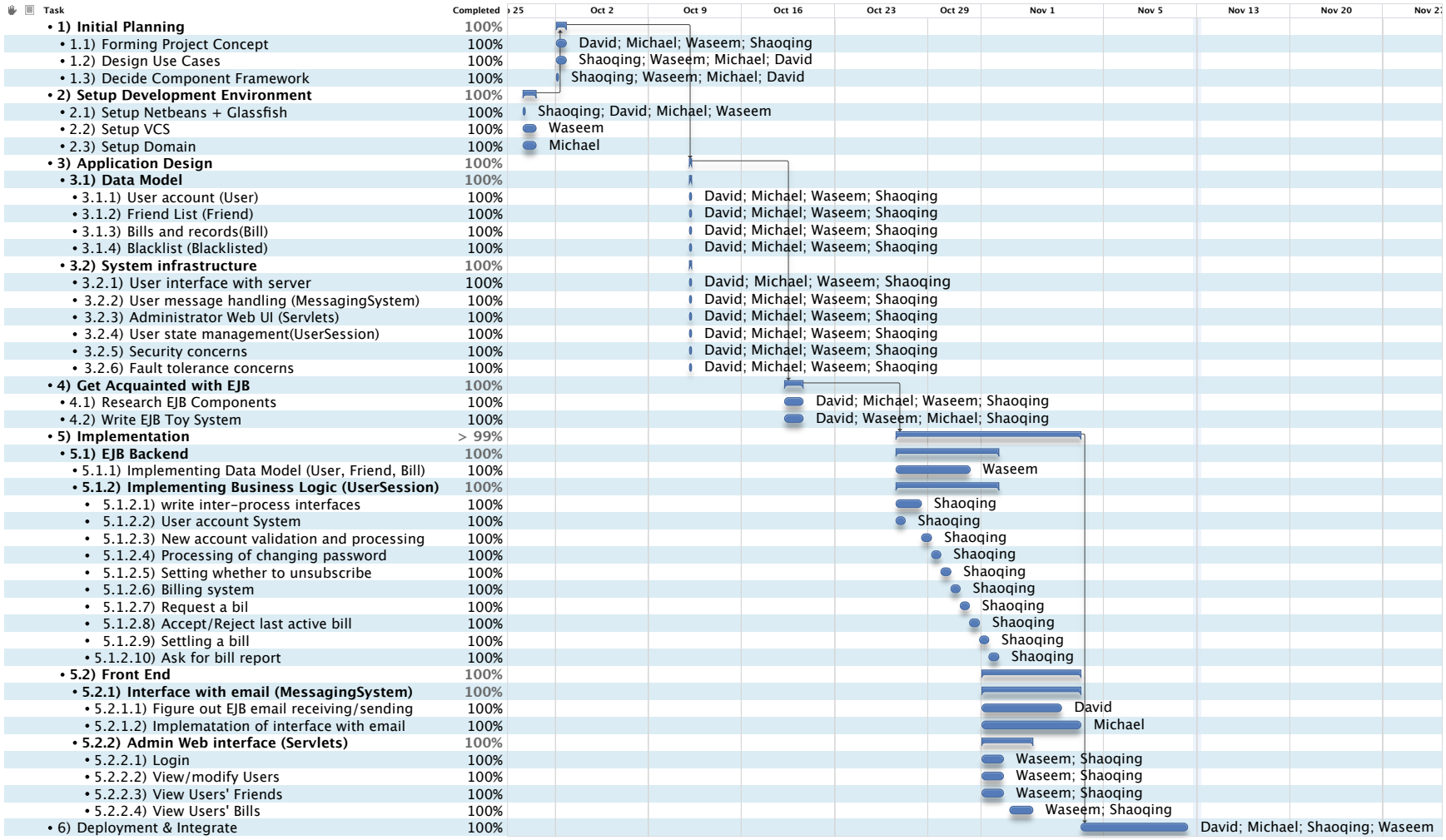


9. Response to Demo Concerns:

Regarding the demo, the TA was quite satisfied with our system, since it all the requirements. However, there were two concerns. First, the TA was confused about the way of refusing a bill. This is because we modified the requirements in the second progress report, now if a bill is not accepted after a day (24hrs), then the bill cannot be accepted and marked as a dead bill. Second, the TA mentioned that it would be better for our system to generate different reply-to addresses for different bills. The way it works now is the user has to reply several 'y' to robot@mycheapfriend.com one for each bill, and it requires him to remember how many and the order of bills that are waiting for acceptance. We found this advice very helpful and are planning to update the system in the second iteration.

10. Controversies:

There is no problem among the team.


Task
Completed
Sep 25
Oct 2
Oct 9
Oct 16
Oct 23
Oct 29
Nov 1
Nov 5
Nov 13
Nov 20
• 1) Initial Planning 100%
• 1.1) Forming Project Concept 100% David; Michael; Waseem; Shaoqing
• 1.2) Design Use Cases 100% Shaoqing; Waseem; Michael; David
• 1.3) Decide Component Framework 100% Shaoqing; Waseem; Michael; David
• 2) Setup Development Environment 100%
• 2.1) Setup Netbeans + Glassfish 100% Shaoqing; David; Michael; Waseem
• 2.2) Setup VCS 100% Waseem
• 2.3) Setup Domain 100% Michael
• 3) Application Design 100%
• 3.1) Data Model 100%
• 3.1.1) User account (User) 100% David; Michael; Waseem; Shaoqing
• 3.1.2) Friend List (Friend) 100% David; Michael; Waseem; Shaoqing
• 3.1.3) Bills and records(Bill) 100% David; Michael; Waseem; Shaoqing
• 3.1.4) Blacklist (Blacklisted) 100% David; Michael; Waseem; Shaoqing
• 3.2) System infrastructure 100%
• 3.2.1) User interface with server 100% David; Michael; Waseem; Shaoqing
• 3.2.2) User message handling (MessagingSystem) 100% David; Michael; Waseem; Shaoqing
• 3.2.3) Administrator Web UI (Servlets) 100% David; Michael; Waseem; Shaoqing
• 3.2.4) User state management(UserSession) 100% David; Michael; Waseem; Shaoqing
• 3.2.5) Security concerns 100% David; Michael; Waseem; Shaoqing
• 3.2.6) Fault tolerance concerns 100% David; Michael; Waseem; Shaoqing
• 4) Get Acquainted with EJB 100%
• 4.1) Research EJB Components 100% David; Michael; Waseem; Shaoqing
• 4.2) Write EJB Toy System 100% David; Waseem; Michael; Shaoqing
• 5) Implementation > 99%
• 5.1) EJB Backend 100%
• 5.1.1) Implementing Data Model (User, Friend, Bill) 100% Waseem
• 5.1.2) Implementing Business Logic (UserSession) 100%
• 5.1.2.1) write inter-process interfaces 100% Shaoqing
• 5.1.2.2) User account System 100% Shaoqing
• 5.1.2.3) New account validation and processing 100% Shaoqing
• 5.1.2.4) Processing of changing password 100% Shaoqing
• 5.1.2.5) Setting whether to unsubscribe 100% Shaoqing
• 5.1.2.6) Billing system 100% Shaoqing
• 5.1.2.7) Request a bil 100% Shaoqing
• 5.1.2.8) Accept/Reject last active bill 100% Shaoqing
• 5.1.2.9) Settling a bill 100% Shaoqing
• 5.1.2.10) Ask for bill report 100% Shaoqing
• 5.2) Front End 100%
• 5.2.1) Interface with email (MessagingSystem) 100%
• 5.2.1.1) Figure out EJB email receiving/sending 100% David
• 5.2.1.2) Implematation of interface with email 100% Michael
• 5.2.2) Admin Web interface (Servlets) 100%
• 5.2.2.1) Login 100% Waseem; Shaoqing
• 5.2.2.2) View/modify Users 100% Waseem; Shaoqing
• 5.2.2.3) View Users' Friends 100% Waseem; Shaoqing
• 5.2.2.4) View Users' Bills 100% Waseem; Shaoqing
• 6) Deployment & Integrate 100% David; Michael; Shaoqing; Waseem