First Iteration Second 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 **Second Progress Report,** if you give us back feedback next week.

Revised Schedule:

We have split up shared (pair-programmed tasks) to individual programmers now that the requirements have been hammered out. Also we realize we're not going to strictly work on Fridays now that it's coming down to crunch time, so we've added a few extra days during the weekend to get stuff done.

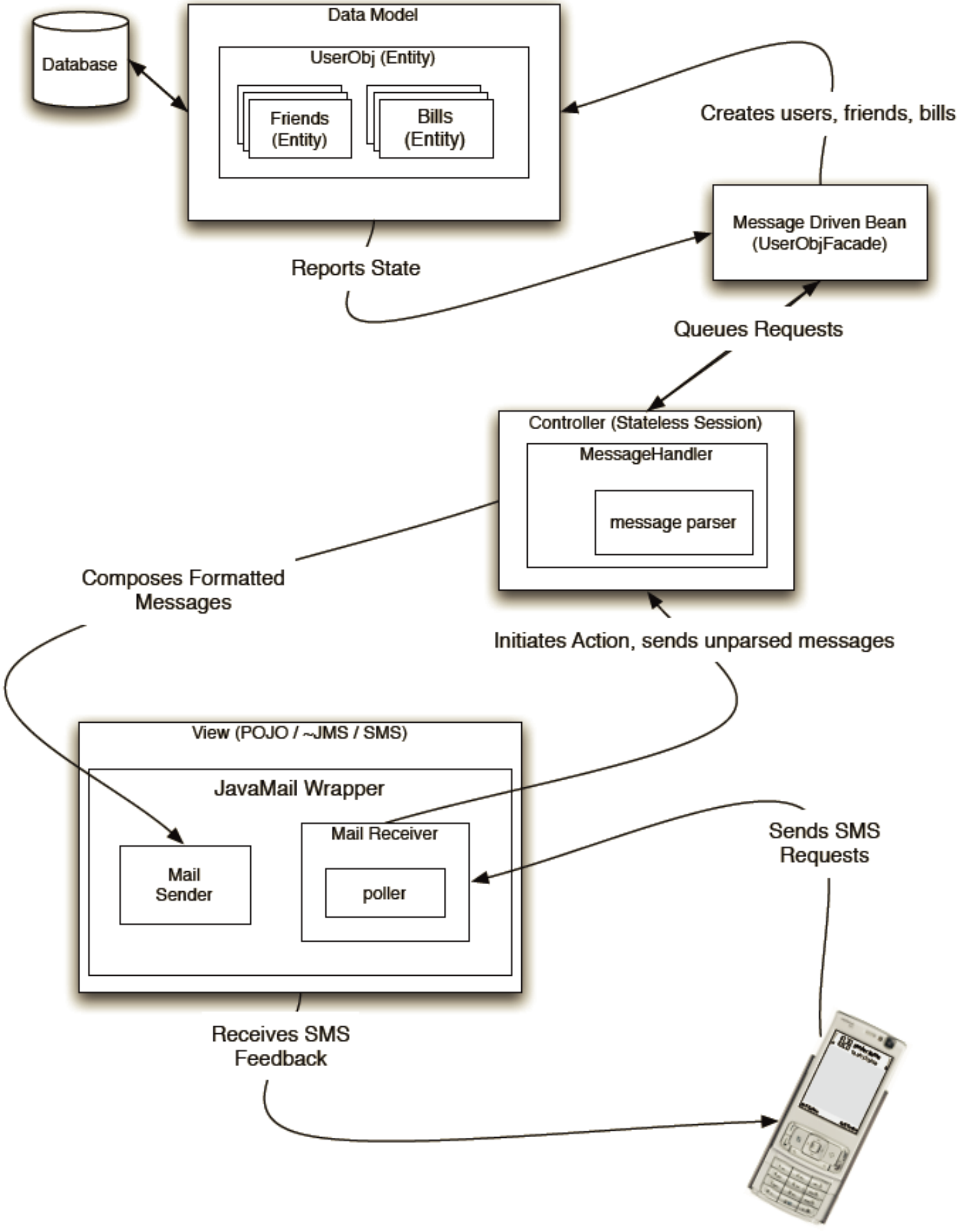
There is some worry of missing our Nov 4th deadline based on the fact that not all team members have been communicating well. We have moved our internal deadline for integration and deployment up to the 28th so we can formulate a contingency plan if everything doesn't come together perfectly.

The revised schedule is attached at the end of this document.

High Level Architecture:

We have added a state diagram on the next page, to show our system's architecture. This diagram also shows the component framework services that each part of the system corresponds to. As you can see, there are multiple steps before the message reaches from the user to the database. We use Message-Driven Bean to communicate between the database and the Controller that handles the message parsing. Then we have the JavaMail wrapper that communicates with the mobile device at the front end.

P – T – O →

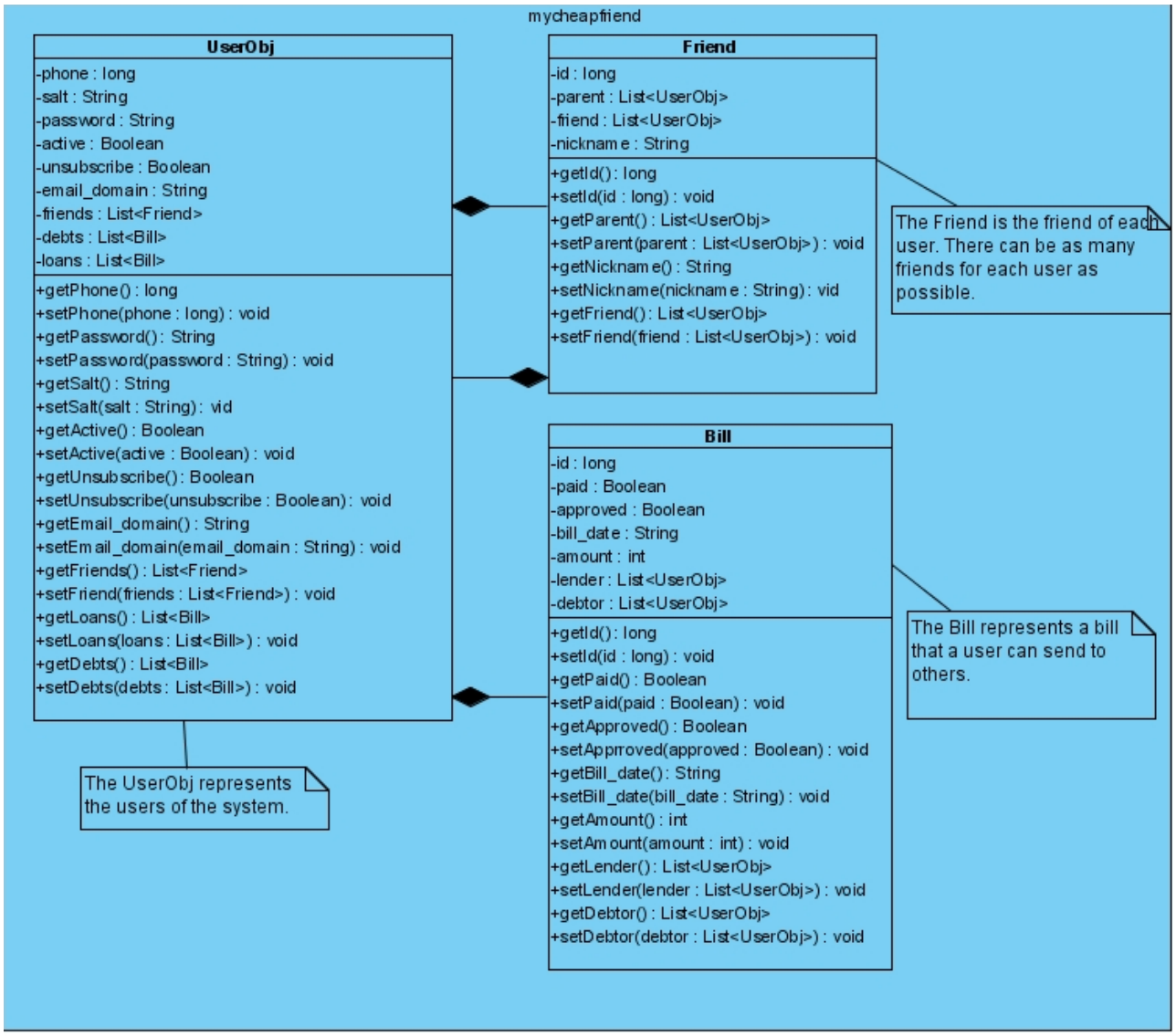

Database
Data Model
UserObj (Entity)
Friends (Entity)
Bills (Entity)
Creates users, friends, bills
Message Driven Bean (UserObjFacade)
Reports State
Queues Requests
Controller (Stateless Session)
MessageHandler
message parser
Composes Formatted Messages
Initiates Action, sends unparsed messages
View (POJO / ~JMS / SMS)
JavaMail Wrapper
Mail Receiver
Mail Sender
poller
Sends SMS Requests
Receives SMS Feedback

Component-Level Design:

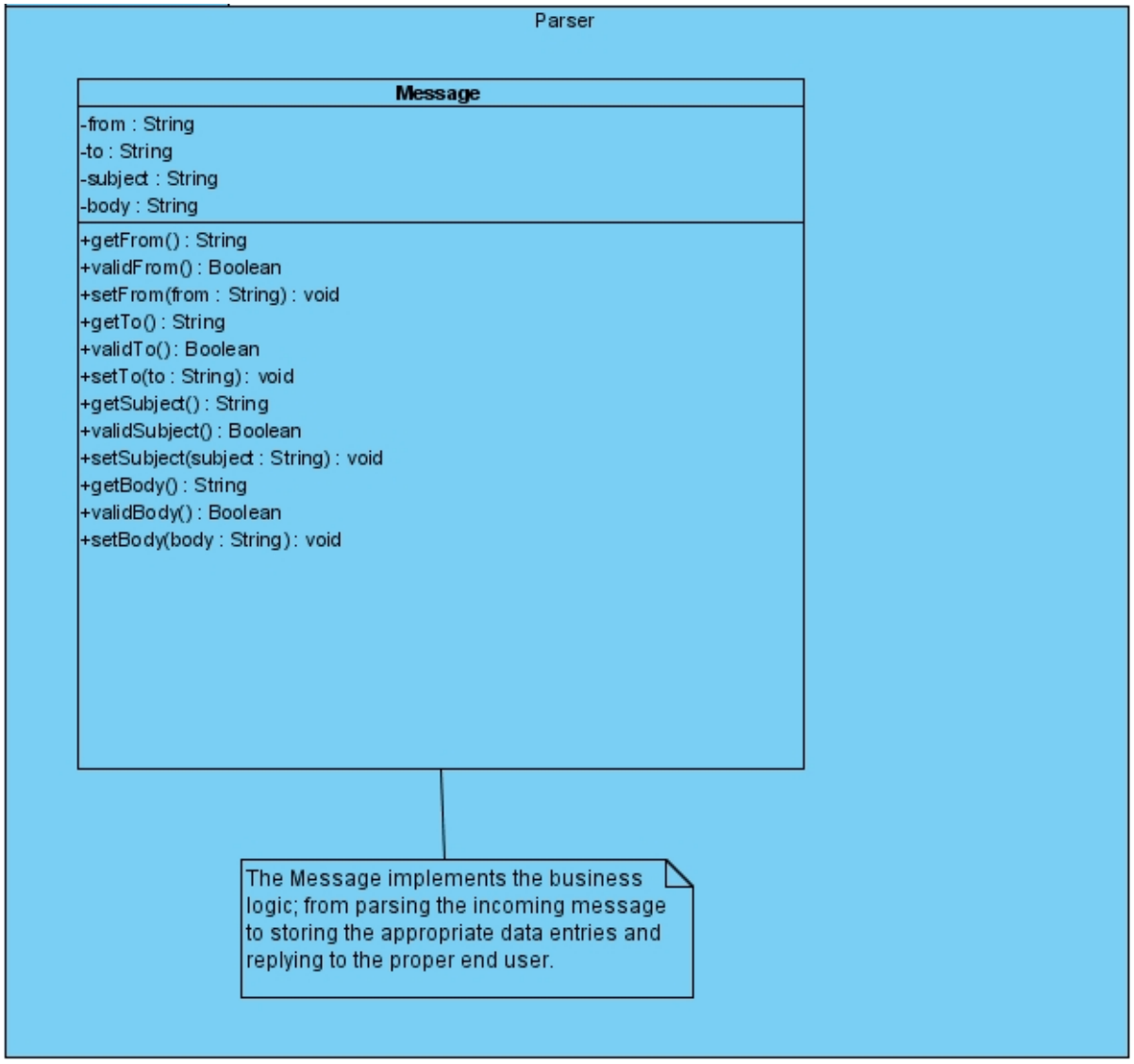
Our system basically has four main components (tasks). The data model (entity setup), the business logic (controller), the E-Mail Wrapper and the Web UI (last task to be done).

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

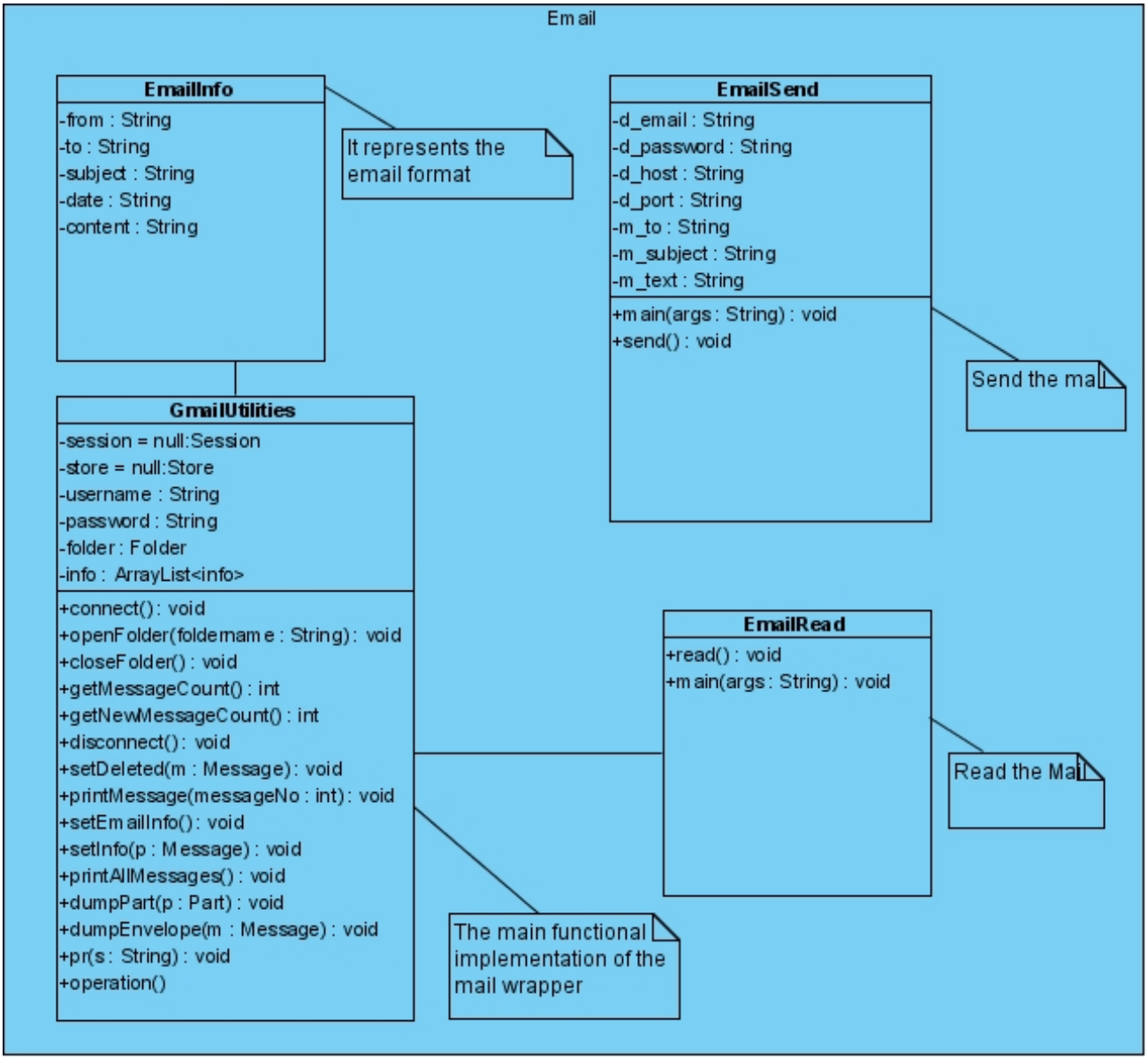
First off, the data model; there are three “tables/entities” basically, the “UserObj”, “Friend” and the “Bill”. To facilitate the communication from the UserObj class and the client side, we have created a “façade/session bean”; UserObjFacate.



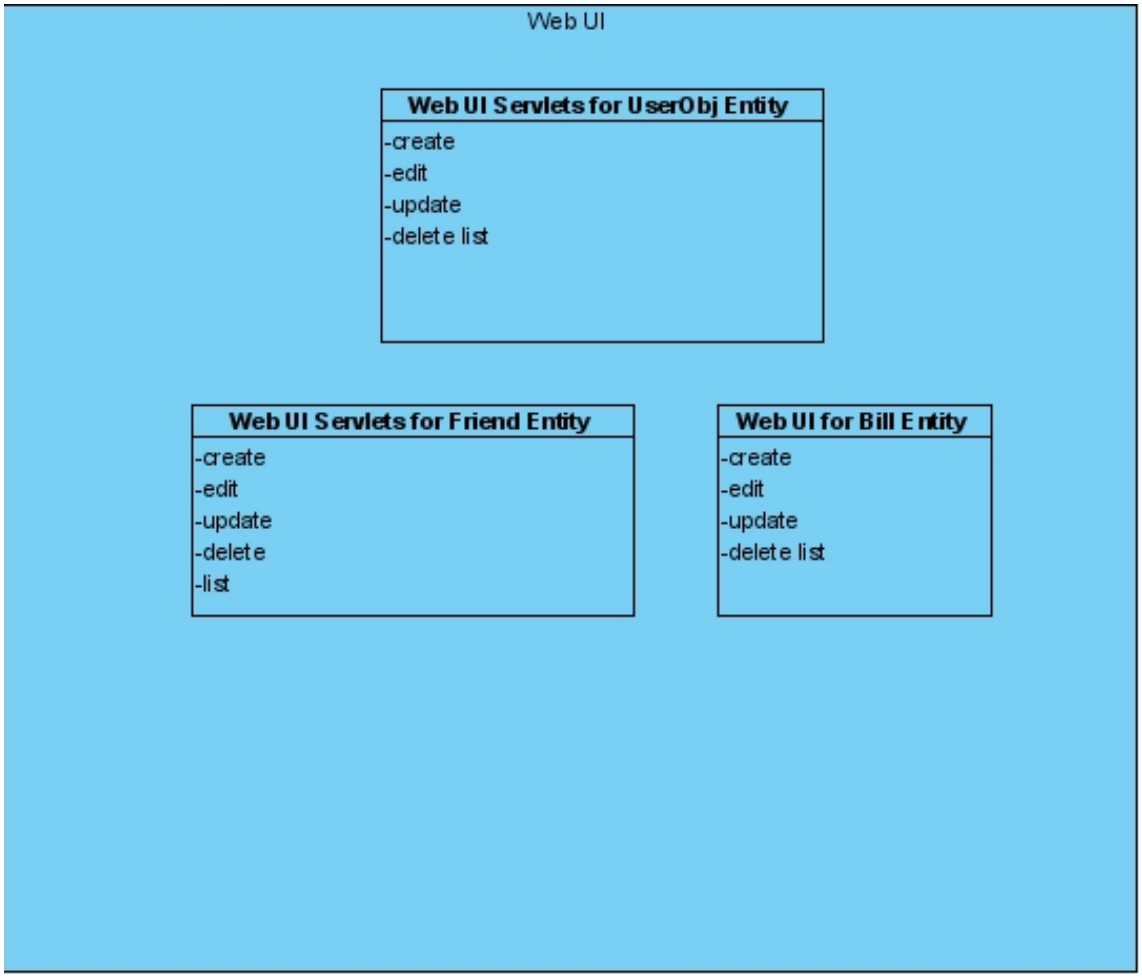
The Business Logic/Controller is basically a class “Message” that parses the message received from the user/mail wrapper and puts it in the appropriate table and vice versa. The following diagram shows how much we have so far.



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 a few seconds. The poller collects new emails as user inputs and passes them to the email parser for both syntactic and semantic processing. 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 the CheapSkates. It will give us the admins the read/write access to the database; a front-end for manipulating tables. There are fifteen Servlets, five for each entity in the data model; for create/edit/update/delete and list. A high level view of these classes is shown below.



Component Model 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 JavaMail, which is often used with JMS, however we use it for user i/o instead.

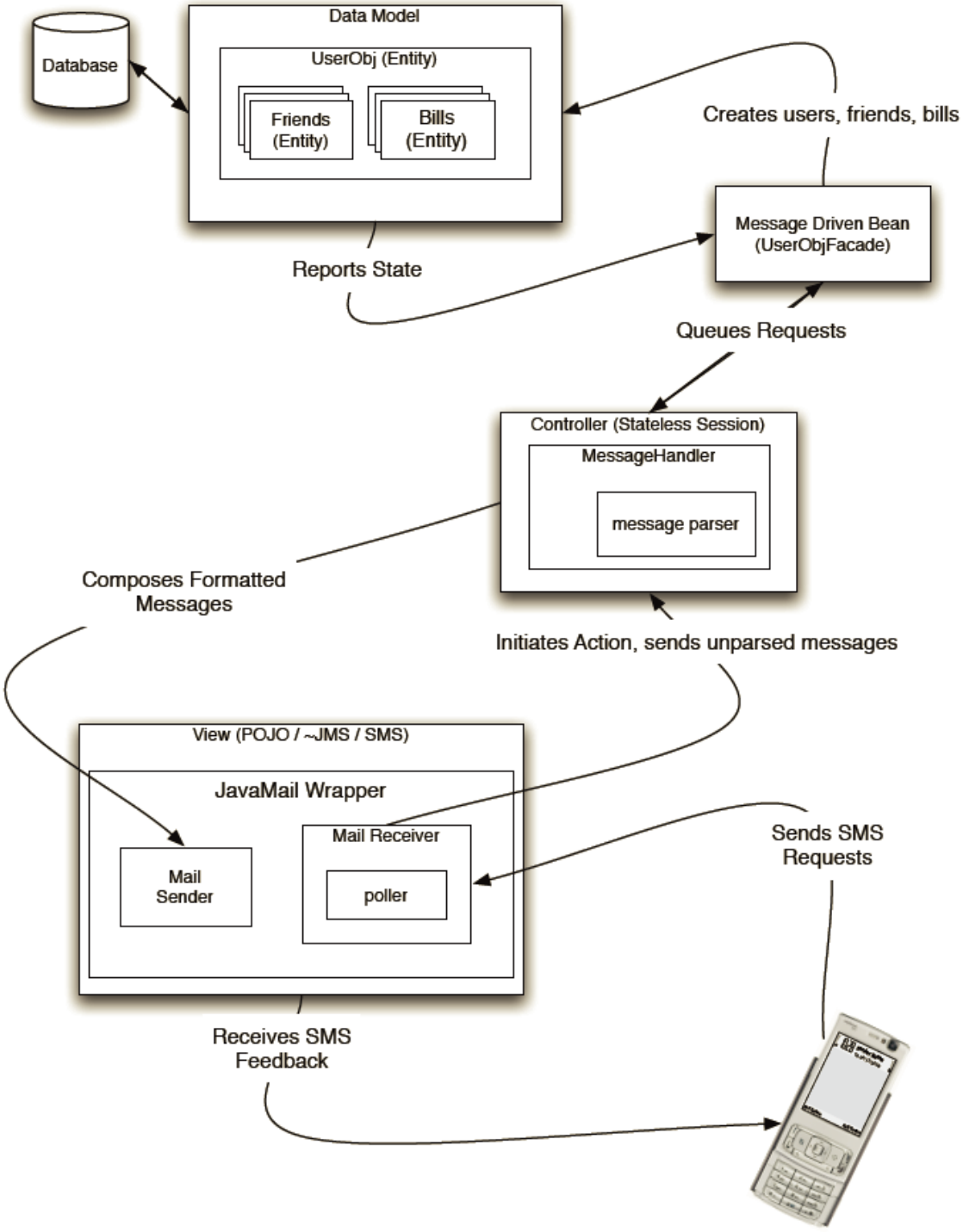
All the messages received are manipulated by a Message Handler as a general Controller which is a stateless session. When sending a message, the Message Handler composes formatted messages and forwards it to the Mail Sender. When receiving a message, the message receives data generated by a message parser. 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.

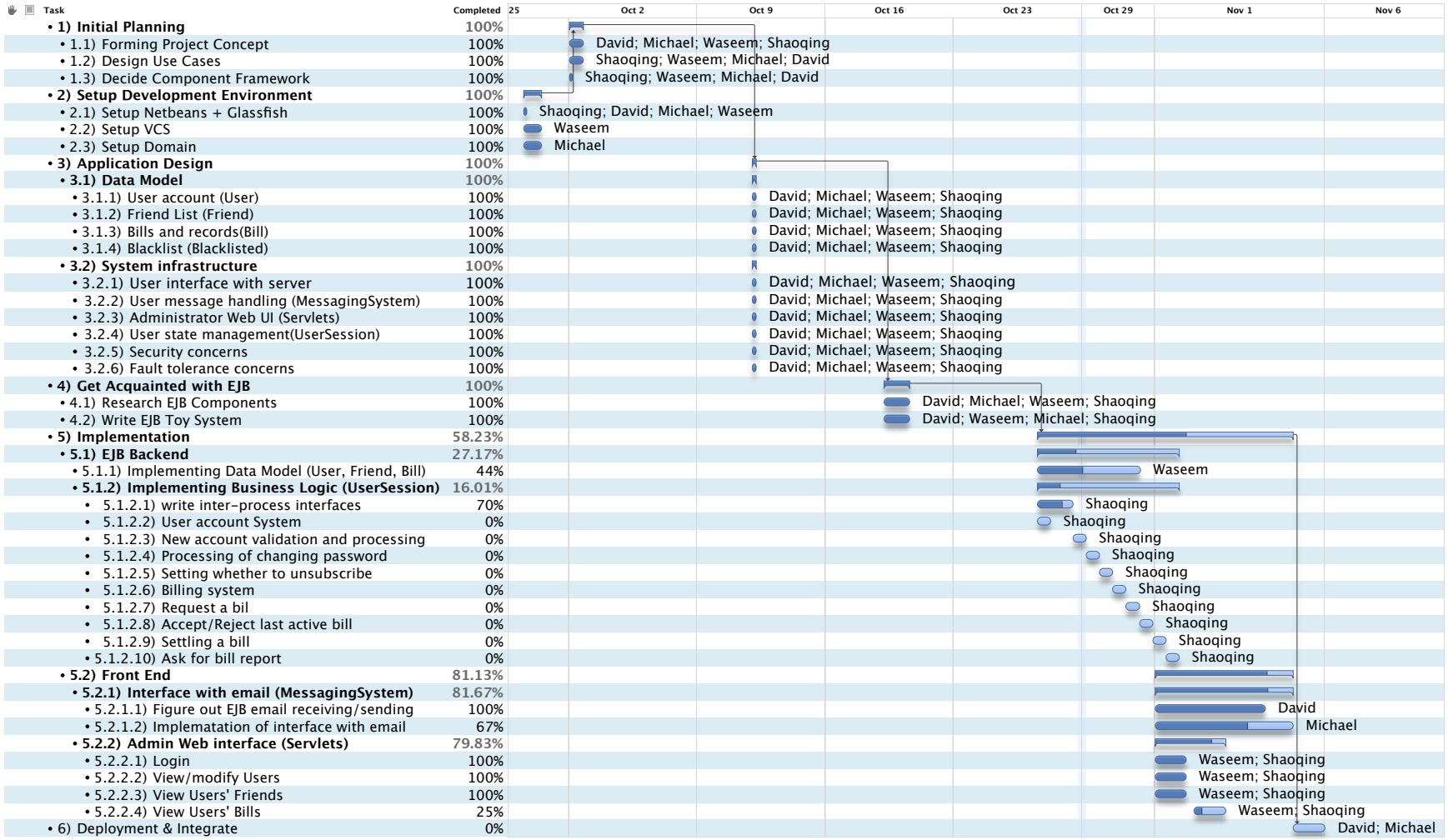
To sum up, javaMail serves as the frontend while Entity Beans serves for the backend. Stateless session and message handler are used for communication.

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



Controversies

There are no controversies among the team at this point.


Task
Completed
25
Oct 2
Oct 9
Oct 16
Oct 23
Oct 29
Nov 1
Nov 6
• 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 58.23%
• 5.1) EJB Backend 27.17%
• 5.1.1) Implementing Data Model (User, Friend, Bill) 44% Waseem
• 5.1.2) Implementing Business Logic (UserSession) 16.01%
• 5.1.2.1) write inter-process interfaces 70% Shaoqing
• 5.1.2.2) User account System 0% Shaoqing
• 5.1.2.3) New account validation and processing 0% Shaoqing
• 5.1.2.4) Processing of changing password 0% Shaoqing
• 5.1.2.5) Setting whether to unsubscribe 0% Shaoqing
• 5.1.2.6) Billing system 0% Shaoqing
• 5.1.2.7) Request a bil 0% Shaoqing
• 5.1.2.8) Accept/Reject last active bill 0% Shaoqing
• 5.1.2.9) Settling a bill 0% Shaoqing
• 5.1.2.10) Ask for bill report 0% Shaoqing
• 5.2) Front End 81.13%
• 5.2.1) Interface with email (MessagingSystem) 81.67%
• 5.2.1.1) Figure out EJB email receiving/sending 100% David
• 5.2.1.2) Implematation of interface with email 67% Michael
• 5.2.2) Admin Web interface (Servlets) 79.83%
• 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 25% Waseem; Shaoqing
• 6) Deployment & Integrate 0% David; Michael