Jen Liu, Ranna Zhou, Derek Vaughn
6.005 Spring 2012

Design

Conversation Design

A conversation is a chat room with a unique ID that any user can create and invite other users to join. A conversation is started when a user selects another user on the “Currently Online” list and sends him/her a message. Both users are now automatically part of that conversation, and can add any other user currently online by clicking the “Add user” button (at which point the added user is automatically part of the conversation--if we have time, we may implement invites so that users can decide whether or not they wish to join). The conversation ends when all users exit the chat room.

To implement this, we have a Conversation class which represents the model in our MVC design pattern. Each conversation is identified by a unique integer (id) and maintains a HashSet of users (members) represented as instances of the Client class.

The Server class keeps track of all active conversations (1 or more user present) in a HashMap called activeConvos, which maps each conversation ID to the list of Clients present in the conversation. It also keeps a running convoCount, incremented each time a new conversation is created--the new conversation assumes the current convoCount as its unique integer ID. When a conversation ends, its ID is removed from activeConvos. Finally, the server maintains a List of online users (userList), adding a Client to the list each time a new socket is created.

An instance of the Client class is initialized each time a user signs on and connects to the server. The user is identified by a username and associated with a List of Conversations (currentConvos) that the user either started or was added to by another user.

When a user performs any action (create a conversation, add a user to a conversation, send a message, or exit a conversation), the Client class formats a request using the appropriate method (createChatRequest, addUserRequest, sendMessageRequest, or leaveRequest) and then sends it to the Server. The Server then parses the request through interpretRequest and takes the appropriate action, sending responses to all Clients in the conversation.

Datatypes

class Conversation (id: int, members: SynchronizedSet<Client>) extends DefaultStyledDocument
Conversation extends DefaultStyledDocument which can be passed directly into JTextFields. The Conversation class is our model. It will be modified by the controller(Client) and then updated to the GUI. The conversation holds a set of Clients currently in the conversation.

```
        public Conversation(id: int, username: String)
        public synchronized addMessage(sender: Client, message: String)
                Adds a message to the conversation.
                @param sender: the
        public addMember(user: Client)
        public removeMember(user: Client)
```

class Client(username: String, currentConvos: HashSet<Conversation>)
The Client connects to the server through its ServerSocket. The client gets ListenerEvents from it's current conversations and sends them to the server. When the client receives messages from the server it interprets them and modifies the conversations accordingly.

```
    public Client(username: String)
    public void leaveRequest(convoID: int){
        sendRequest("leave " + convoID.toString);
    }
    public void sendMessageRequest(convoID: int, message: String){
        sendRequest("msg " + convoID.toString + " " + this.username + " " + message +     "/n");
    }
    public void createChatRequest(receipientUsername: String, message: String){
        sendRequest("create " + receipientUsername+ " " + message + "/n");
    }
    public void addUserRequest(ConvoID: int, receipientUsername: String){
        sendRequest("addUser " + convoID.toString + " " + receipientUsername + "/n");
    }
    private sendRequest(message: String)
    private interpretMessage(message: String[])
    public modifyUsers()
    public modifyMessage()
    public createConversation()
```

class Server(convoCount: int, activeConvos: HashMap<convoID: int, userList: List<Client>>, onlineUsers: HashSet<Client>)
The Server is responsible for maintaining the ServerSocket and accepting incoming client connections. As clients connect a new thread is opened for communication between each client and the server. The server accepts requests from the clients, interprets them and sends a message to each client involved. The server stores a HashMap with ConvoIDs that are linked to a List of Clients involved in that conversation.

```
    private interpretRequest(message: String)
    // Computes the Request sent by a client and prepares arguments to sent to sendMessage
    private void sendMessage(message: String convoID: int, username: String)
    // User A sends a createConvo request, then the server sends a message to the recipient
    // client.
        clientB.socket.outputStream(convoID + " " + username + " " + message + "/n")
```

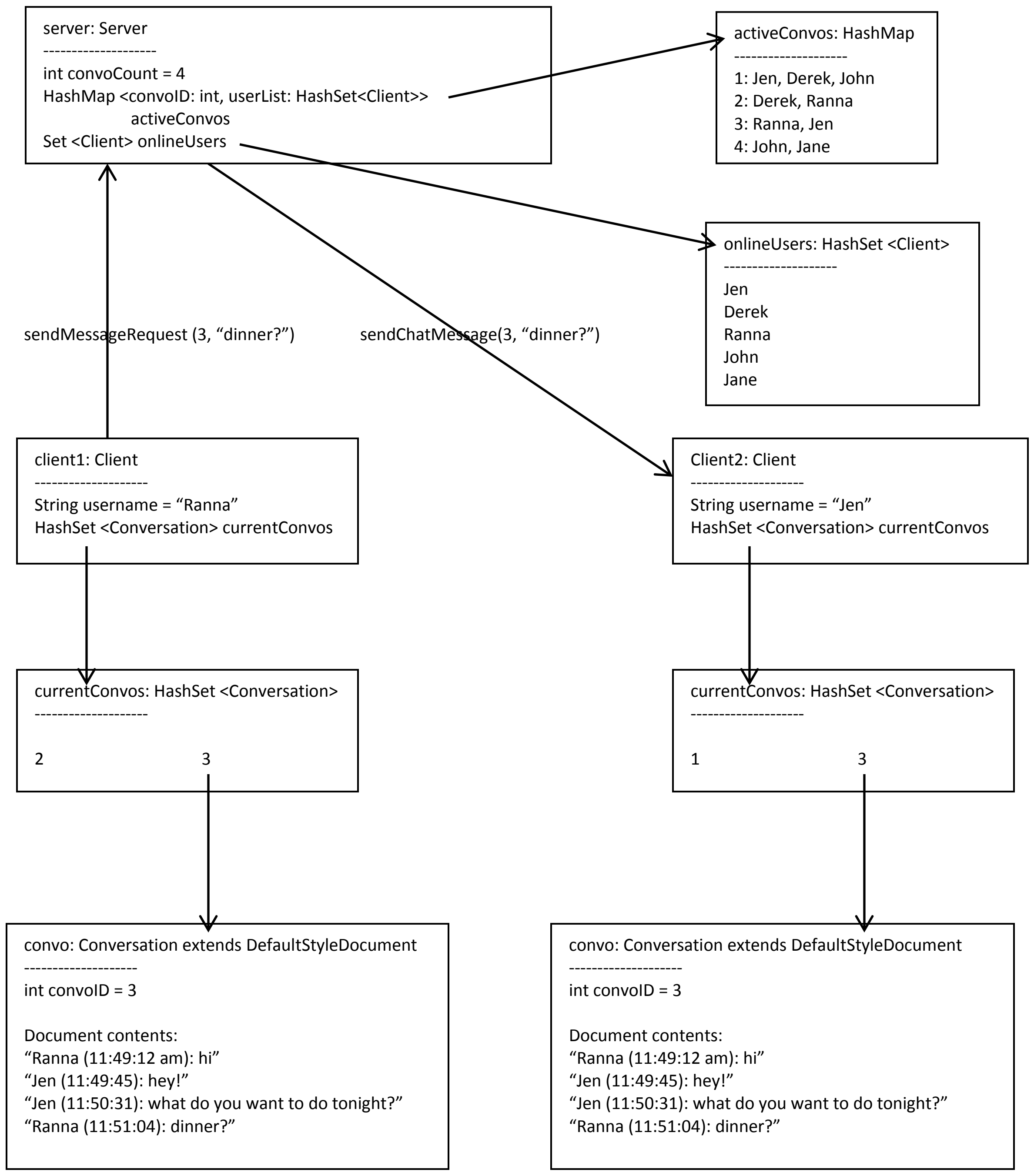
Client-Server Protocol

Client-to-Server:
MESSAGES ::= (CREATECONVO | ADDUSER | LEAVECONVO | MESSAGE)
CREATECONVO ::= “create” SPACE USERNAME SPACE MESSAGE NEWLINE
ADDUSER ::= “addUser” SPACE CONVOID SPACE USERNAME NEWLINE
LEAVECONVO ::= “leave” SPACE CONVOID NEWLINE
MESSAGE ::= “msg” SPACE CONVOID SPACE USERNAME SPACE TEXT SPACE NEWLINE
CONVOID ::= [\d]+
USERNAME ::= [a-zA-Z[\d]]+
TEXT ::= [^NEWLINE]+
NEWLINE ::= “\n”
SPACE ::= “ “

onLeaveConvo, if members.length == 0, send TERMINATECONVO message to server so that server removes the CONVOID from ACTIVECONVOS list.

Server-to-Client:
MESSAGES ::= (NAMECONFLICT | MESSAGE | USERLIST)
NAMECONFLICT ::= USERNAME SPACE “is already taken” NEWLINE
USERLIST ::= “userList” (SPACE USERNAME)+ NEWLINE
MESSAGE ::= CONVOID SPACE USERNAME SPACE TEXT SPACE NEWLINE
CONVOID ::= [\d]+
USERNAME ::= [a-zA-Z[\d]]+
TEXT ::= [^NEWLINE]+
NEWLINE ::= “\n”
SPACE ::= “ “

Conversation Example



server: Server
----
int convoCount = 4
HashMap <convoID: int, userList: HashSet<Client>>
activeConvos
Set <Client> onlineUsers
activeConvos: HashMap

1: Jen, Derek, John
2: Derek, Ranna
3: Ranna, Jen
4: John, Jane
onlineUsers: HashSet <Client>

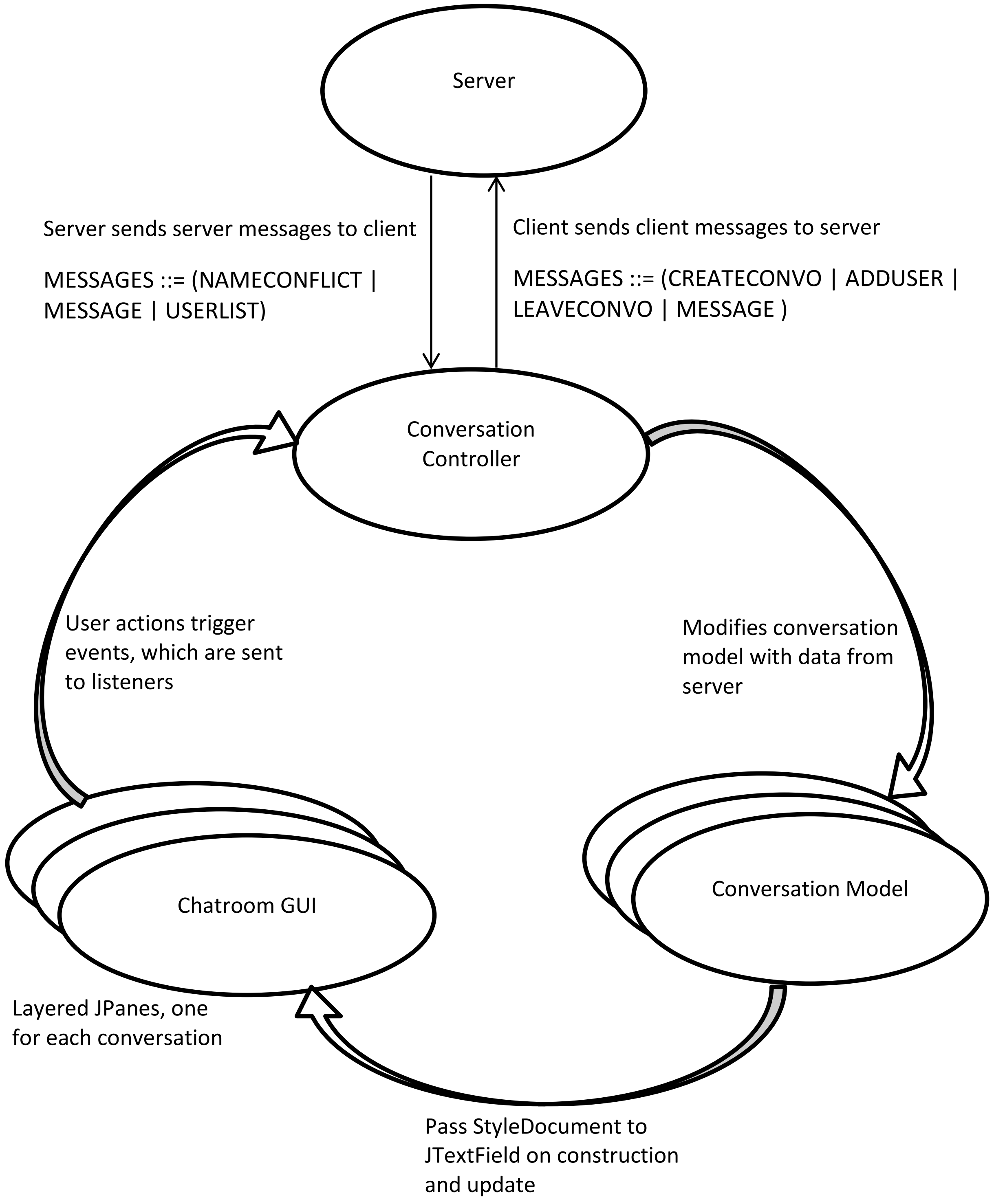
Jen
Derek
Ranna
John
Jane
sendMessageRequest (3, “dinner?”)
sendChatMessage(3, “dinner?”)
client1: Client

String username = “Ranna”
HashSet <Conversation> currentConvos
Client2: Client

String username = “Jen”
HashSet <Conversation> currentConvos
currentConvos: HashSet <Conversation>

2
3
currentConvos: HashSet <Conversation>

1
3
convo: Conversation extends DefaultStyleDocument
----
int convoID = 3
Document contents:
“Ranna (11:49:12 am): hi”
“Jen (11:49:45): hey!”
“Jen (11:50:31): what do you want to do tonight?”
“Ranna (11:51:04): dinner?”
convo: Conversation extends DefaultStyleDocument
----
int convoID = 3
Document contents:
“Ranna (11:49:12 am): hi”
“Jen (11:49:45): hey!”
“Jen (11:50:31): what do you want to do tonight?”
“Ranna (11:51:04): dinner?”


Server
Server sends server messages to client
MESSAGES ::= (NAMECONFLICT | MESSAGE | USERLIST)
Client sends client messages to server
MESSAGES ::= (CREATECONVO | ADDUSER | LEAVECONVO | MESSAGE)
Conversation Controller
User actions trigger events, which are sent to listeners
Modifies conversation model with data from server
Chatroom GUI
Conversation Model
Layered JPanes, one for each conversation
Pass StyleDocument to JTextField on construction and update