WEB 2.0 COLLABORATION FACEBOOK CHAT

Senyao Du

sd2693@columbia.edu

Requested by and Submitted to

Professor Henning Schulzrinne

DEC 26, 2011

Abstract

We propose a way of integrating social networking tools with online interaction model (e.g. chat, calendar) to form a new collaboration method. This method incorporates a new ideology of using OAuth protocols provided by social networking tools like Facebook and Twitter as user authentication models, as well as using social graphs provided by them as enhanced collaboration methods. These methods leverage the existing infrastructures provided by the social network tools to acquire contact information, and use the same infrastructures to broadcast event invitations or send messages to specific contacts at the same time.

As a final addition to this application, we have provided the user with the much needed video chat function, trying to augment their real time interaction experiences. Using the new assisted P2P multicast model built in Flash Player 10.1, we provide users with a low latency and bandwidth efficient way of carrying out video chat using multicasting and P2P unicast among a number of peers.

The combined result of this online collaboration application is able to provide users with a streamlined experience form signing up using Facebook account, to creating a chatroom, inviting Facebook friends along to participate and doing video chat with them.

Introduction

Traditional approach to chat room collaboration relies on the authentication model of correctly entering a user name and password pair, which is normally associated with a unique email address that requires verification, which requires a fair amount of effort from users to start using the application, for they need to create a new user name, which should not be duplicate with those current in the system, and a password that is compliant with some security standards. On top of those requirements, they are usually forced to enter correctly a CAPTCHA to verify that he is indeed a human, and lastly, get his email address entered verified through clicking a link inside the email sent from the application.

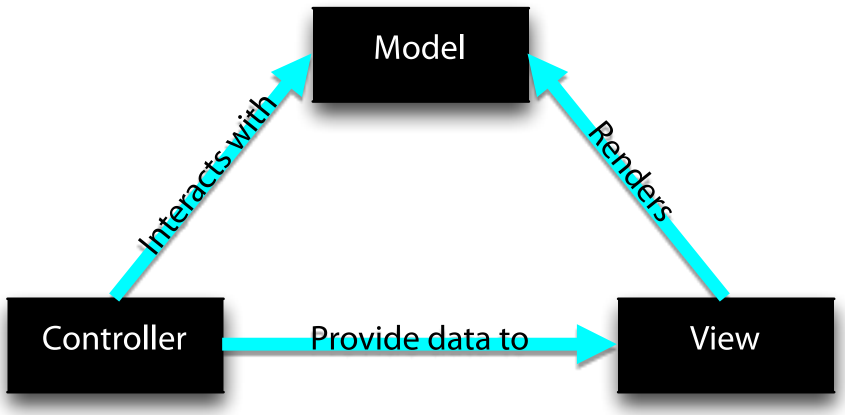
With the flourishing of online social tools like Facebook and Twitter, and the powerful APIs provided by these platforms, we begin to be able to identify a user not through asking for username and password, but instead, have identities providers to tell us who this person is - Facebook and Twitter just do that. On top of merely providing identities, after asking explicitly for user's permission, these platforms give us a plethora of details about who this person is, who his friends are, and provides different means of contacting his friends.

This project focuses on multiuser collaboration and social communication tool integration. The motivation behind the project is people's tendency to use online social tools to facilities daily activities have increased dramatically over the last few years. So, we have decided to use a social approach towards a simple application towards chat. Armed with Facebook and Adobe Flash, we have achieved a solution that is both useful and efficient.In this report, we'll look the details of the implementation of the Facebook Chat web application, struggles we met and finally solution used in working code, which includes different frameworks I used, as well as the data model behind those frameworks.

Initial Framework

We started our project by first choosing the web framework we wanted to use. We have came a long way where JAVA servlet or PHP were the only choices for developing a web application. There are many choices out there today, from the elegant Ruby on Rails 3, to the most clearly defined Python Django, from the old ASP that runs in IIS to the new Node Express that is purely Javascript based.

We initially settled on the free web stack provided by Microsoft: ASP.NET MVC3, Entity Framework 4.1and SQL Server Compact 4.0. Built on top of the traditional ASP.NET web form model, MVC3 framework is built to use more modern paradigm, the model-view-controller pattern.



The model view controller paradigm

Using this pattern we have achieved the separation of concern, instead of making the web application a giant octopus that does everything in a single class file, we try to compartmentalize the application into different philosophical models that include model, view and controller.

- The model is independent of any views or controllers. Its primary goal is to provide a consistent data model of the entity we need for the application to function. Usually, it interacts with the database to persist the data in memory, as well as communicating with the controller exchange data.
- The view's primary goal is to renders the model with a predefined HTML template, it could be strongly typed, which ties with only one model, or it could be loosely typed, rendering data from different models. However, it only gets data from the controller, and it never interacts with the model directly.
- The controller interacts with both the model and the view. On one hand, it loads data and make it save changes to the model, on the other hand, it communicates with the view to provide necessary data for rendering of web page, as well as to retrieve any model changes to pass on to model for processing.

The MVC framework provided through Microsoft is convention based, which means it uses same name for all the controller, model and view, but in different directories specified by convention[1]. Comparing with a typical enterprise J2EE solution like Spring Framework[2] with JSF, this approach saves quite a bit amount of efforts by automatically wiring up the models with its corresponding views and controllers. A convention based approach is easier to debug as well, instead of going into XML configuration to identify the culprit that is causing the problem, people could easily identify the problems by browsing the code in controller and view by looking at the name and using convention.

Chat Model

We have researched into the options we have for creating a chatroom that doesn't needs users to refresh manually. There are three basic models for doing it:

- Active Polling: At a fixed time intervals, the application will try to make an AJAX call to the server to retrieve the most recent chat messages since last received.
- Long Polling: Instead of a fixed time intervals, the application will make a blocked AJAX to the server and only returns when there is new messages, and after updating the UI, it will make another same blocked AJAX call to get new messages.
- WebSocket[3]: Using the new IETF standard for communicating with the servers, we can create a web base application behave just like a native C# client, that will communicate with the server using native sockets. However, this standard is not well implemented across the web browsers as of today.

After some deliberation and experimenting with coding for a bit, we have decided on the long polling model. It's well supported across different browsers and most reasonable to implement without excessive calling when scaling up. There are some online examples about using Observables and Reactive framework to implement the server side, which could create a subscriber based push model that fits exactly we what wanted, but in practice, it turns out to be confusing and hard to use. Being a beta product, it is poorly documented, and hard to figure out by oneself how to create simple component for chatting.

Here is where we stuck for the longest of time. Besides the Reactive framework suggested by Google searches, we have looked other ways as well. What we discover is that in the current production version of .NET, Microsoft has little documentation on how to implemented block calling using async methods in C#. There are some progress made in F# and new .NET framework 4.5, but we find it extremely hard to use them to implement what we have in mind correctly.

[1] Unsurprisingly, they are usually called Model, View and Controller

[2] http://www.springsource.org/ a division of VMWare, provides XML based ways of wiring up models, views and controllers.

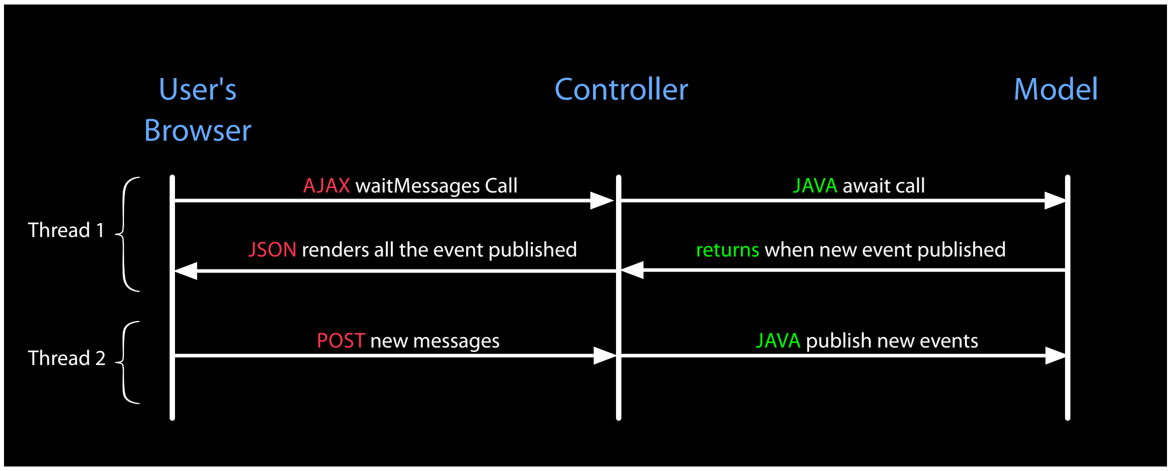
[3] http://tools.ietf.org/html/rfc6455 a new standard for doing two way communication

Framework Switch

We switched to a different framework after the working with C# to no avail. The current application is built on top of the Play framework.[4] This is a JAVA framework, which uses groovy to dynamic interpret the HTML template, which is the view. It uses JAVA classes to make up the model and the controller.

It still uses the similar convention based MVC model from what we had in mind for our initial framework decision, but what makes it uniquely fit our project is the functional programming library it provides, as well as a good amount of documentation.

With the functional library, we are able to make client side blocking, server side non-blocking calls like await and promise to facilitate the broadcasting of chatroom message. Each client side AJAX call to the server for new messages will be translated into a server await call for more messages as they become published.



Interaction between browser, controller and model when chatting

Authentication & Authorization

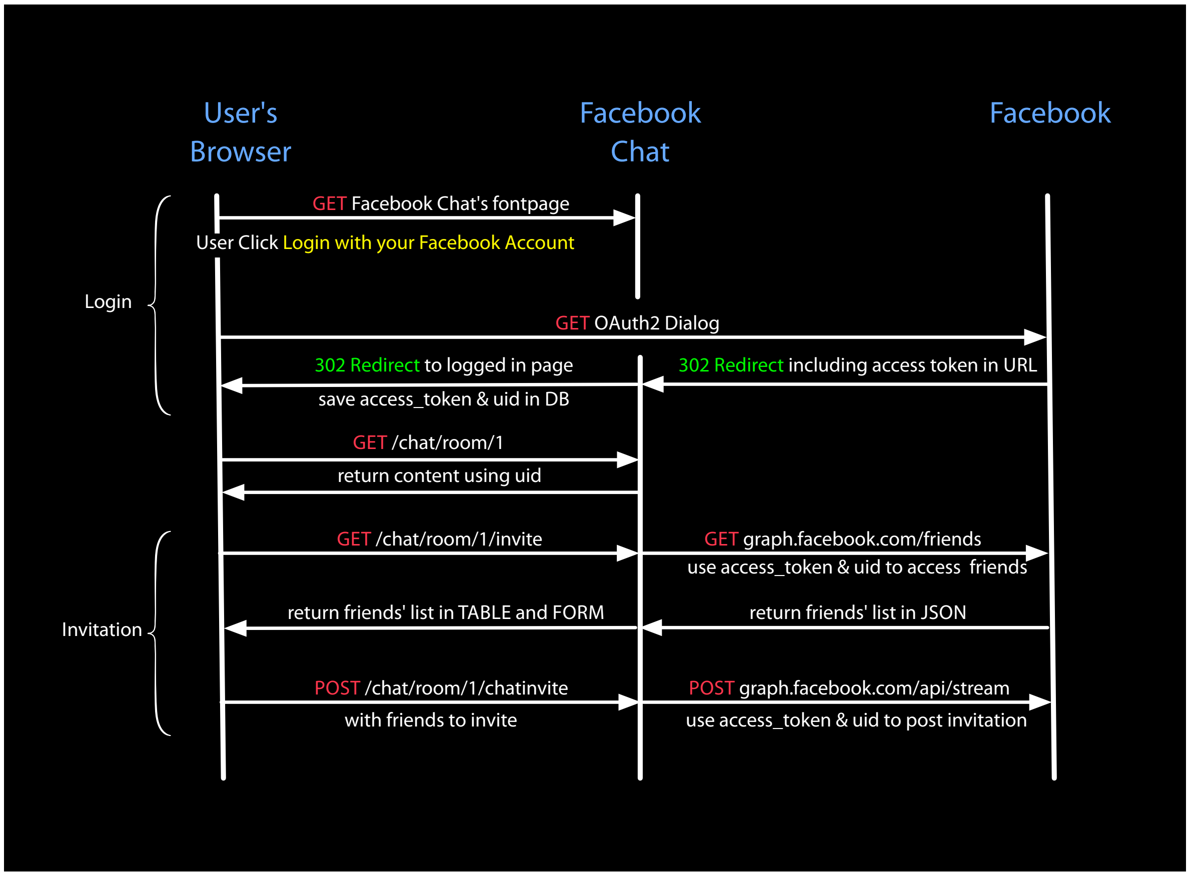
We uses Facebook for authentication in our main application[5], as it provides us with the name and a unique ID (UID) for each user. By leveraging both, we could differentiate different users in the application using the UID, and tag various chat messages accordingly.

The same uid is for authorization as well. When a new chat room is created, it is only authorized for the creator to participate. When the creator invites more people through Facebook, their UIDs are added to the same chatroom, thus allowing them to participate.

[4] http://www.playframework.org/ a new lean framework for creating scalable JAVA web applications

[5] There is a separate administrative backend that uses a different authentication method, i.e. the traditional username and password model.

Facebook uses OAuth2 as the means for application to make sure users get authenticated, as well as providing the application the needed access_token to acquire further information from the server. We have all interactions between the user, the application and Facebook illustrated as follows:


User's Browser
Facebook Chat
Facebook
Login
Invitation
GET Facebook Chat's fontpage
User Click Login with your Facebook Account
GET OAuth2 Dialog
302 Redirect to logged in page
save access_token & uid in DB
302 Redirect including access token in URL
GET /chat/room/1
return content using uid
GET /chat/room/1/invite
GET graph.facebook.com/friends
use access_token & uid to access friends
return friends' list in TABLE and FORM
return friends' list in JSON
POST /chat/room/1/chatinvite
with friends to invite
POST graph.facebook.com/api/stream
use access_token & uid to post invitation

Application interactions with user and Facebook

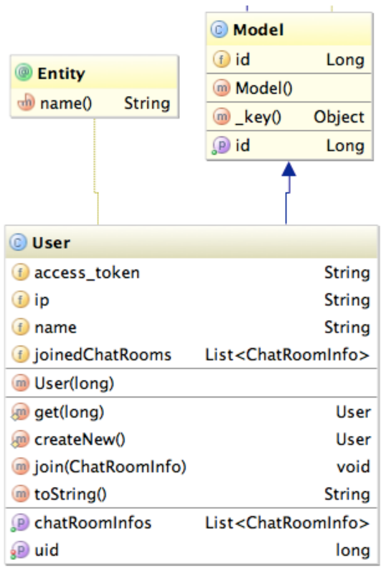
The whole process includes the following:

1. When user clicks the Facebook login button, he is redirected to Facebook for login
2. Upon successful login, the browser is redirected to the landing page, the access_token is passed to the controller for storing in the database.
3. Controller tries to retrieve more information from Facebook, including uid and name when rendering the page for a user.
4. Controller renders the number of chatrooms that are associate with that uid.
5. User enters a chatroom, clicks invite button, and then his friends list is retrieved from Facebook for choosing whom to invite. The list is rendered a form with multiple checkbox in a table.
6. When the user picks the users to invite, with user's consent, application acts on user's behave to send message to selected friends to invite them into the chatroom.

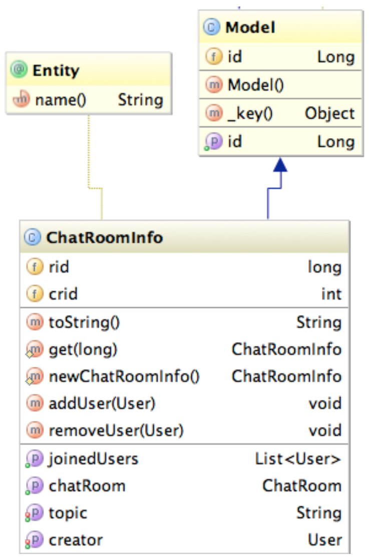
Data Model and Data Persistence

Play framework provides us with a method of persisting data into database by leveraging the new JAVA persistent API (JPA). By extending the generic Model class provided by the framework, we are able to save the entity that is annotated with the @entity symbol into database with some simple function calls which in turn hook into JPA manager to save those entity into database.

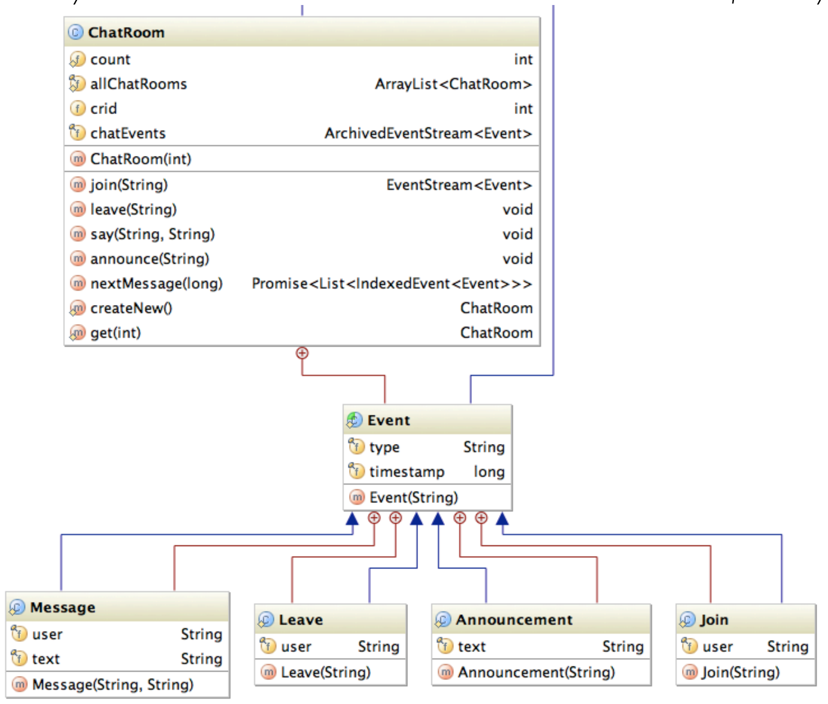
With the mechanics of manipulating database out of our way, we have more time to think through clearly what need to define as entity that has to be persisted into the database as well the transitory memory model for what needed to enable the chat application.



User Entity



ChatroomInfo Entity



ChatRoom Model

As we can see from the entity diagram. Both user and the meta information about a chatroom extends the Model, and thus managed by JPA to be persisted in the database. The actual model for chatroom, however, is transient with latest 100 events. As it is more complex and costly to store its data structure in the database.

However, what we could do to mitigate the issue is, instead persisting AchivedEventStream on the server, that we allow the user to save a copy of transcript by storing the JSON output from the server into user's own personal chat history record as blob data structure, and upon opening a historical chatroom, directly sending the JSON into the view to be rendered.

Video Chat

Video chat is a key component of online collaboration. Seeing face to face and talking directly looking into each other's eyes makes a difference. Thus we have decided to include video chat as a part our collaboration tools independent of Facebook.

We use Adobe Media Streaming service as the backend server, and create a chat client for a P2P accelerated multicasting video chat solution, whose infrastructure has been baked into the current version of Flash Player. Using Adobe's Flex framework, we created our video chat application:

- It first by establishing two connections with server to setup two streams with multicast group parameters for sending and receiving packets, and then it joins those groups.
- Then the client grabs the camera and microphone, attaching those into the outgoing stream and start publishing the stream with the name of his own UID.
- Meanwhile, it tries to play the stream with the receiving UID on a different stream.

With what has been built into Flash Client, it could create a peer assisted multicast network. After some initial setup time, if all the clients are within the range of sending and receiving multicast packets, we can see a big reduction in latency, for it eliminates the need for sending the video stream into in the server and retrieving the video stream from the server, instead, clients are sending the video directly towards each other.

Admin Panel

Finally, using the what Play framework has provided us with, a CRUD module[6] and a security module[7], we have created a simple management tool for managing all the chatroom as well as the user information.

By extending the CRUD controllers, we create management tools for User and Chatroom meta info entities, so that admin of the application could choose to remove a chatroom or delete a user from the application as needed.

[6] A web framework used for managing entities that are managed by the JPA framework

[7] Simple authentication just by looking at two fields named Username and Password

Conclusion & Further Work

In this report, we have taken a look the rationales of how to implement the chatroom, the initial choice of a MVC framework, the change of choices for frameworks to facilitate the implementation, the overall architecture of chat alone, the authentication, authorization using Facebook, augmented social interaction by exchanging data with Facebook, the entities and model used for chat, the implementation of the video chat portion and the swift implementation of a admin module.

Due to time constraint, wee believe the application could be further developed to incorporate the following features:

Multi-chat Google Hangout Style

We can include more features into the program to better facilitate a video conference, which includes more people and more features, like "passing the mic" function as well as the ability to add in more users dynamically

Integration with Google, Twitter and other OAuth Provider

Google and Twitter could both be used for login. Depending the user's permission, we could choose to send chat or collaboration invitation to his specifically invited friends or a general list he has pre-made. We want to allow people from different social networks to come together for collaboration instead of limiting to a specific social network. Ideally, we want to eliminate the need for signing up and logging into our system, instead, we choose to opt to use online provider for different identities.

Better functional modules

We want to provide the user with more options with the ability to integrate into existing services like Google Calendar and Facebook event. So some of the actions online could be translated into a Google Calendar entry or a new Facebook event. Further more, we want the user to add new modules into his collaboration group, for example, using Microsoft's Docs for Facebook to edit word documents together or creating a wiki entry to share with collaborators.