Senior Project Final Report

BatChat - Mobile Communications Application

2014

Cody Martin

The senior project I chose to develop was a mobile application on the very popular Android platform. This application allows users to connect and communicate with one another over the open and standardized XMPP communications protocol. This is a cross-platform network protocol that allows anybody to adopt and build applications on top of. It has already been adopted by many large organizations and products. Both Microsoft and Google both use it for interoperating between different federated chat networks, which makes it the perfect chat medium to bring to mobile devices.

I chose to design and develop this project in collaboration with Adam Jacques. We felt that working as a team would help us to develop skills that could be later used in the professional world. In order to get the most out of the project we wanted to make it as close to a professional work environment as possible. We accomplished this by partaking in code reviews for all code changes to the SVN repository, and using an issue management system to track bugs and to keep our progress. In doing so we were able to prove that we could work as a real-world development team.

Working in collaboration with Adam was one of the largest benefits in developing this project for me. The aspect of making it as real world as possible is something that I did not experience much throughout my undergraduate curriculum and I find it to be very important. I believe that it will help me immensely when I start my career after graduation. The collaboration tools we used were very effective. They made it easy to open bugs/issues against one another's code, review one another's code before it was committed to the codebase, and to keep track of the progress we were making both as a team and as individuals. The tool we used to track our progress was a web based

project management application called Redmine. I added all of the tasks I laid out in the proposal to Redmine. I could then see what I had left to complete, I was also easily able to see anything that had bugs that needed to be fixed. Using Redmine really helped to keep me on track and show me what I still needed to complete. It was a good way to stay organized and always be able to have a task to work on. Before either one of us pushed any code to the main SVN repository, we first submitted all of our code changes to a web-page tool called ReviewBoard. We then requested that the other person would inspect the code changes to catch any clearly broken code or other code style issues. This process step wasn't about actually having the reviewer try to run the code and ensure that it worked, but it was supposed to catch grotesque or other obvious issues to drive a high-quality codebase. We also used this process to question design decisions the other person may be making at this time. This is inline with processes that many large and serious software engineering companies require and served as a practice run for business-oriented application development.

We chose to implement this project because we have used different XMPP clients made for the Android platform and they provided a poor user experience. They had distasteful user interfaces and did not support some features that the XMPP protocol allows and makes easy to implement. Because of the poorly implemented XMPP clients currently in the market, we thought we could create and provide a better product to the currently underserved customers.

Choosing to implement this project on the Android platform allowed me to experience developing an application in an unfamiliar environment. Mobile application development is an important skill in today's world with so many new companies targeting Android, so this project gave me the experience I needed to feel comfortable working on a serious Android project.

Being a two-person project, we decided to divide it into two separate parts that could be easily worked on independently. Then we could later merge the two parts when we reached the point of being compatible. I worked on the user interface portion while Adam worked on the networking stack. Since the user interface depended on the network system in a few places, Adam and I both designed an abstract interface that I could code against until he got around to implementing them. Doing this made it possible for us to work at our own pace and be able to work without one another, as we had different schedules.

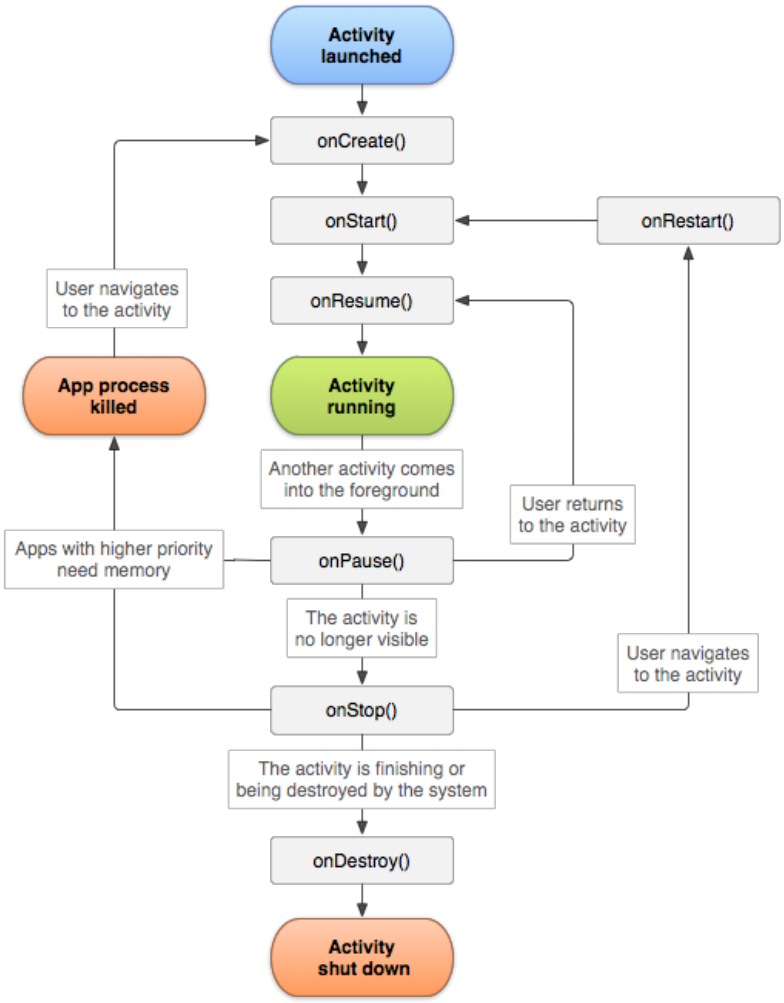
The first thing I did in the project was create a few basic user interface layouts for the login and home screen. I created the original layouts using Eclipse's built in user interface editor. To use this, I dragged and dropped items to their desired location and then selected different options such as the layout_gravity, size, and ID. Once I started creating more layouts, I found that writing the XML by hand seemed to be a lot more efficient and I was able to get better looking results because I was not constrained by the expectations of the editors. Not only was I able to make the user interface look better with this process, but I often times ran into problems using Eclipse's built in UI editor. The Eclipse plugins would get into a corrupted state and fail to render anything causing my entire process to be blocked until I could resolve the problem. Many times Eclipse would only show a "Java Null Pointer Exception" when loading the UI editor and it could no longer build or run the application. The only fix I could find that consistently worked and resolved the problem was deleting the .android folder. This folder contains your Android Virtual Devices and other configuration settings forcing me to reconfigure them several times.

As part of the application side that I worked on, I also implemented the Android service that was responsible for running in the background and facilitating all communication between the user interface, local setting files, and the networking subsystem. This service is called the Connection Manager and is heart of the whole

application. It controls the network subsystem and is the center of communication for the application. Once the user is logged in, the Connection Manager service takes over control of the whole open XMPP session and callbacks are registered with the networking subsystem to receive notifications of any and all interesting events that happen, including but not limited to authentication failures, incoming messages, presence updates, and subscription requests. When a callback event occurs, the Connection Manager will appropriately handle it. Some responses include notifying the user that something happened which is currently handled by triggering a standard Android notification and posting it to the system-wide notification center. The Connection Manager is also responsible for posting events to the different app-owned activities and user interface controls so that they can update accordingly.

In order to effectively communicate between the Connection Manager and the Android activities, I had to learn how the Android activity life cycle functioned and always be aware of the state it was in for any activity that the Connection Manager needed to communicate with. The reason the state is necessary for this communication is because the only time an activity can change what is being displayed in its user interface is when that activity is in the foreground. Before I understood this, the service would attempt to post events to activities that were not the currently displayed activity causing the entire application to crash. I later learned that this was because I was attempting to change the user interface when the activity was paused or possibly not even created yet. Attempting to change a user interface of an activity that is not currently in the foreground will cause the application to crash. Using the different methods Android provides you are able to determine what part of the lifecycle that each activity is currently in to know whether or not you are able to update its user interface. Figure 1 below shows the lifecycle of an activity from creation to activity shutdown.

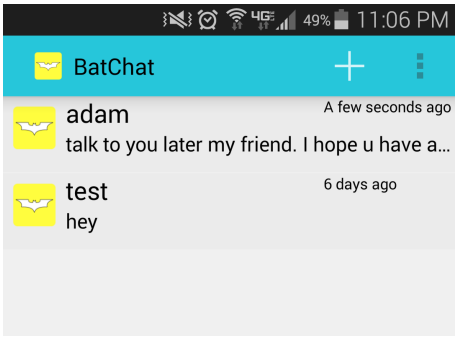
Figure 1: Android Activity Life cycle



Understanding the Android activity lifecycle was also important in other aspects of the application. Often times adding code to the onResume() and onPause() callbacks was necessary in order for the activities to be in the desired state. Things such as broadcast receivers needed to be registered and unregistered when the activity is paused and resumed. I also used onResume() to render and new user interface updates that may have happened while that activity was paused. Using the provided callbacks I was able to determine whether or not the activity was currently paused.

The home page of our application holds all ongoing conversations. You are able to open new conversations by hitting the “+” button in the action bar at the top of the screen. Each conversation in this activity is displayed to the user interface with a fragment. “A fragment represents a behavior or a portion of a user interface in an activity.” One reason fragments are nice is because they act like an activity within their host activity. You can add a tag to each fragment so that you are able to get a handle on it later. Doing so allows you to change what it is displaying when new data comes and needs to be updated. Every fragment’s lifecycle is directly affected by the host activities lifecycle. For example, if the activity is paused, you cannot update any fragments inside of the activity because all the fragments are also paused. However, when the activity is resumed you can manipulate all of the fragments at will. The reason I decided to use fragments in the user interface was so it was easier to manipulate smaller portions of the user interface at a time. Being able to separate each conversation to hold its own data like a class instead of requiring the activity to remember the state of each conversation. Figure 2 shows part of a screenshot of the home activity. You are able to see how the fragments are used to separate the information based on the conversation.

Figure 2: Home Activity



In order to provide a consistent user experience throughout the entire application, I designed the new conversation activity in a very similar way to the home page. Using fragments to show all of the friends currently on your roster and to also show pending

subscription requests. If I were to design this page again I would change the way I did this. Using fragments in this situation gives no advantage in the user interface because I never update or change any information. The only piece of information on each fragment is the contacts username. Instead of using fragments, a cleaner and more concise way would have been to use an Android listview. I could have had the same desired user experience with more concise code.

In the action bar of the new conversation activity I also added the functionality to send a subscription request. Instead of opening a new activity or window inside the current activity, I added an EditText to the action bar with a search icon next to it. You can then enter a XMPP client handle into this text box and press the search button. If it isn't a valid XMPP client handle, an Android toast will pop up and tell you so. Otherwise it will send the subscription request to that user and they will have the opportunity to accept or reject the request. My reasoning behind adding the subscription request to the action bar was to make as few user interface screens as possible. I did not want to make the user navigate from screen to screen and get lost in all of the commotion. Adam and I had discussed that the simpler the user interface is, the better the user experience would be.

Developing for the Android environment proved to be a difficult process as I encountered several problems while working on this project. The IDE would often get into an inconsistent state or have random errors. The fix could range from requiring re-installs to just restarting the IDE. This greatly impacted my progress on the project, at times, because it required me to stop coding and fix my environment.

There are only two development environments to use for targeting the Android platform and for this project we ended up using Eclipse. We chose to use Eclipse because we weren't sure if Android Studio would be feature complete enough for our project. Many of the different Scala and intermediate plugins that were used to compile

and transform the generated code were recently developed which means that they did not fully support Android Studio, causing us to use Eclipse. Eclipse is currently the most widely used and is the officially accepted development environment for Android which meant that it was expected to be a safe choice for developing in, but is currently on a plan to be replaced by a new IDE. Android Studio is the new environment that is built on a completely new base IDE that was designed to fix many of the problems that Eclipse-users suffer from. If we were to do this project over again, we probably would have tried using Android Studio. While it may not have been fully supported by all of our plugins, we believe that we could have avoided spending a significant portion of our time fixing corrupted environments.

At first I thought this project would be fairly easy considering it was a lot of user interface work. However, as I progressed into the project it proved to be more difficult than I had first anticipated. I discovered interacting with the framework caused problems that took a more advanced design to solve. Also, running into problems with Eclipse slowed my progress and made working with Android very frustrating at times. In the end it was an excellent way for me to learn android. It was also a great way for me to use a school assignment to learn the real world development skills required to work in a team environment. Working with Adam also really improved my code reviewing abilities. Adam has had prior experience with doing serious code reviews and having him there reviewing my code really helped me to see how reviews can improve the codebase and decrease the number of regressions. Using Redmine to manage our progress in the project also helped me to see what I still needed to accomplish and fix, which helped to keep me on track. Overall this project ended up a success. It served as a good experience to end my undergraduate degree and start my career as a software engineer.