CS536 Final Project: IRC Bot Using Pircbot Java Framework

Andrés Alejos
December 13, 2018

1 INTRODUCTION

Internet Relay Chat (IRC) is an application layer network protocol that enables text chat messaging between users, using the client/server model as opposed to a peer-to-peer (P2P) model. IRC facilitates chat rooms that are referred to as channels, where clients can join to begin using the chat services, which will be the focus of my project. IRC does offer a variable range of services other that these chat rooms, such as private text messaging and file-sharing, but I will not do work regarding these features. IRC bots are essentially automated clients that join channels and can participate in different way depending on the behaviors they are programmed to perform.

Many channels will use bots that just sit idle on the channel and can moderate the activity that occurs. They listen to the messages sent across the channel, and can be programmed to react differently depending on what is sent. For example, a bot on a channel might be programmed to listen to the word **Date** on the channel, and whenever **Date** is messaged to the channel, it will respond with the date-time group. These bots are the focus of my project, and I will explore how they work by creating my own bot with the help of the PircBot Java API.

2 BACKGROUND

Chat rooms are becoming more and more ubiquitous across the internet, with new websites adopting chat rooms and adding them into their services. Twitch, the most prevalent online streaming service, uses chat rooms specific to each stream, enabling all of the viewers of the stream to communicate with one another. YouTube added its own live-streaming service

with chat functions as well, and new similar services are constantly popping up, such as Slack, which is targeted more towards business and formal work/organizational groups, and Discord, which is targeted more towards gamers.

Most of these services offer other services other that the chat room, such as Twitch and YouTube with streaming and Discord with Voice-Over-IP (VoIP), however the text chats remain a vital part of the product that they offer. One of the attractions to these chat services is their modularity, and the ability to integrate bots. Across all of these platforms, bots are used for updating members of the chat room to news, updated policies, schedule of events, as well as helping to moderate the room to ensure a welcoming and friendly climate. With these bots being such an integral part of the chat services, and becoming more prevalent, I want to learn not only how they work, but also how they can be leveraged to perform more complicated and intricate tasks. Learning how these IRC bots work and gaining the knowledge of how to write one would allow me to cater my own bot to specific problems, and have more of an appreciated for this often-unnoticed feature.

3 Contributions

I created a bot based off of the Pircbot Java framework, and named my class and bot PU_Bot (Purdue University Bot). Originally, I intended to include five features as described in my project proposal, which were: a date and time function, a filter to ban people who use inappropriate language, a spam filter to stop clients from repeatedly sending too many messages over the channel, a translate feature which uses Google translate to translate a message, a wiki feature which displays the Wikipedia summary of a term to the channel along with a link to the full article. I had to end up changing the features due to the client-side nature of the Pircbot framework.

Since Pircbot supports bots which are clients and simply react to actions taken on the channel, and does not actually see messages before they are sent to the channel, there is no way for my bot to intercept inappropriate messages and filter them before they reach the channel. The best I would be able to achieve with the framework would be to keep track of users who use inappropriate language, and warn them for using it before possibly kicking them from the channel if they persisted. This did not really meet my original intent of replacing foul words with characters such as asterisks, so I abandoned that feature altogether. Instead, my bot has similar features to what I originally intended, as well as a couple different features which cater themselves to the client-side nature of the framework. These features are a help feature, a date and time feature, a translate feature, a language detection feature, a wiki look-up feature, and a news look-up feature.

I chose these feature to leverage available APIs as well as use well-known data formats such as RSS to provide useful functionality to the bot. I will go in depth into each individual command, including the concept behind it, its implementation, and lessons learned from each. Aside from the commands a user can issue to the bot through the channel, the Pircbot framework allows for many other event-oriented features, which the programmer can modify to do different things as reactions to different events.

3.1 |HELP

The help command is invoked by typing **"|help"** into the channel, upon which the bot responds with:

PUBot Commands:
|time – Displays current date and time
|translate (desired language): (string)
|detect : (string you want to know the language of)
|wiki : (topic to search)
|news – lists top 5 current stories

The purpose of this command is to guide the user on how to use the bot. This was the first command I implemented since it simply waits for the command in the form of a string and responds by sending messages containing the help strings.

3.2 |TIME

As the help feature instructs, the user can invoke this command by simply typing **"|time"** into the chat. The bot responds by sending a message into the channel with the date and time. This was second command I included, since it is the most straightforward command that included more work than simply outputting a string. The bot simply looks up the date and time and formats it into a string, and send a message to the channel with the result.

3.3 |TRANSLATE AND |DETECT

The translate command was the first complex feature I attempted to add to the bot. I was planning on using Google Translate somehow to fulfill this feature. I was not sure how I would go about it, but I initially thought I could use the website and make HTTP GET requests while dynamically filling in the fields within the web-page. There were many unofficial Java libraries and APIs I looked at initially when trying to complete this section, but none of them seemed to show any promise since they were all deprecated. This was due to Google monetizing these features within the past few years by providing their own paid APIs through their Google Cloud Platform. Since Google provides a 12-month free trial of their APIs, I decided to use their official APIs to complete this feature, which proved to be the correct decision.

It took a while to get the development environment that Google requires to use their APIs set up, but once I did it made the rest of the feature very easy to program. This was a very good learning experience for me since it required that I consult the API documentation and learn how to use a professional-grade API. Additionally, while consulting the Google Translate API I realized that it would not tak too much extra code to add the language detection command either, so I proceeded to use the same API to enable that feature. The API is structured such that you can call a Translator object, as well as a Detector object, which made it so that I could not only add the text detection command, but also allowed the translate command to not require that the user input what their initial language was, since I could use the API to detect which language they are writing in. Another hurdle I had to overcome, however, was that the Google API uses ISO 639-1 two-digit language codes to encode their languages. Since I did not want the users to have to know the language codes for each language, I had to use another

Java library that I found from Github to enable to conversion between the natural language and the codes. This package did not have the best documentation, so it required that I read through the source code to determine how to best use the package for what I wanted my bot to do. This, again, taught me good lessons on how to trace other developers' source code and the benefits of good documentation.

3.4 |Wiki

I next implemented the Wiki function of my bot, after having seen this feature on many bots across the internet. As opposed to Google' Cloud Platform, Wikipedia's API is open to the public free of charge, and there is a dedicated page on the Wikimedia (the parent company of Wikipedia) regarding how to use it. Although it is open to the public, it is not necessarily the easiest API to work with, as it is not clear how it is organized, and required that I dig through and experience some trial and error to figure out which queries lead to the results I expected. I knew that the functionality that I wanted was the ability for the user to search a term and have the bot respond with a brief summary of the term along with a link to the Wikipedia article.

I started off by referencing the Wikipedia API page I mentioned earlier, where it lists a link where you can append whatever keyword you want at the end of the link, and it will reference a URL with a JSON of all of the Wikipedia search results relevant to the item you searched. For example, if the word was **banana**, then the search result JSON would include results for items such as **banana**, **banana split**, **banana republic**, and other similar keywords. The way I implemented the feature was to just use the first result of whatever keyword the user gave, since the first result is the closest match to whatever is searched. This JSON happened to just be a JSON Array, not actually a JSON object, which since I have not had much experience working with the format, took me a while to understand how to actually parse the data.

This particular page actually could redirect you to another JSON object for each search result with the entirety of the article contents in many different formats, but I did not actually have to go to any other pages, since the first result also had a field for the page summary as well as the link to the page. So my final implementation consisted of getting the keyword to search from the user, appending it to the URL provided on the API page, fetch the JSON Array from the URL, and parse the array to retrieve the summary and article link to the first item in the array. Once I had those strings, I simply sent them back to the channel after formatting.

3.5 |News

The last feature I decided to work with was integrating RSS parsing into the bot. I was inspired by one of the bots mentioned in the Pircbot website's list of bots made with the Pircbot framework, where it mentions a bot called Newsbot, which simply updates whatever channels it is in with summaries and articles whenever a new article appears on a news site. I wanted to do something similar, but much simpler, so I decided I just wanted to list off the top five headlines at the time the command is entered, along with the links to each full article. After looking into how to accomplish this, it became clear that RSS (Real Simple Syndication/Rich Site Summary) would be the easiest approach, since RSS already handles all of the formatting of news articles, and many news sites support their own RSS feeds. After perusing through a

few different RSS feeds including Reddit and Google News' RSS feeds, I decided to use Reuters' RSS feed since it seemed the most straightforward and easiest to work with.

Next I had to learn how to parse RSS feeds using Java, which contrary to what I had anticipated, is not a trivial task. RSS can just be parsed as a series of strings, but that would be discarding the benefits of using RSS, so I did not want to proceed with that option. I ended up referencing a tutorial from Vogella, a tech website and blog, where they provide a library to parse RSS, although it was not presented as a library or even a Git repository, so I just had to copy and paste the code. Their library was divided into three different Java classes: Feed, FeedMessage, and RSSFeedParser. The first two classes format the contents of the URL you provide, where the Feed class stores a whole RSS feed, each FeedMessage represents one RSS message, and the RSSFeedParser actually parses the feed and messages. Thus, the pipeline works by providing the URL to the desired RSS feed as a string parameter to a new RSSFeedParser. Then I can create a new Feed using the parser to read from the provided URL.

Finally, I can get a list of each individual messages in the Feed through a **getMessages** method in the Feed class, where I then just iterate through the first five messages in the list to get the headlines and links to the top five headlines. The code provided from the library does not work with every RSS feed due to different formats, so some tinkering might be necessary for different site, but this taught me how RSS feeds work and how to work with URLs in Java and how to work with input streams. The benefit of using RSS for this feature is that the bot will fetch whatever is on the feed at the time of the command, so the site maintains the actual content, while the bot can simply pull from the feed. The last touch I added was formatting all of my commands such that the links appear in blue so that they are distinguished from other text.

4 Future Work

The main realizations I cam to throughout this project are the inherent limitations of using a framework such as Pircbot to create IRC bots, due to having to work within the boundaries of the framework, and due to it only being a client-side bot. This caused two of my desired features to basically be impossible, and caused me to look for other options. I was under the impression that I would be able to implement these features, such as a profanity feature and spam filter, since I had seen them in services such as Twitch, but after reading on how people implement their bots on Twitch, it appears as though those are services and APIs that Twitch offers to their streamers and bot developers, so even those who are using client-side bots have access to certain server-side operations. In the future, if I were to continue this work, I might look into implementing this bot into a specific service such as Twitch or Discord, in order to get access to some of those APIs.

Additionally, I would want to develop more features outside of explicit commands, such as how to kick, ban, or promote users in the channel. I strayed away from these sorts of features for the scope of this project due to it being more difficult to test these features than it is to test whether commands work, and with this being my first real experience working with IRC, I wanted to try features that could provide more instant feedback and which were easier to troubleshoot. This could mean either using other frameworks for developing a bot, or not

using a framework at all. The Pircbot framework is very helpful to understand how IRC bots work, and to help the first development cycle from being too overwhelming, but to make the most powerful bot possible, I would probably have to get away from using a framework.

5 Conclusions

This project helped me learn many different concepts that I might not typically have exposure to. This included:

1. Coding more in Java, which is a language I am somewhat comfortable and familiar with, but not entirely so
2. Using the Google Cloud Platform which included obtaining API keys and reading from their documentation
3. Using IRC with an IRC client, which is foreign to me as I have never worked with bare IRC before
4. Learning to parse JSON Arrays and Objects
5. Learning how RSS is formatted and how to parse it
6. Learning how to use Gradle to build a project and manage dependencies

These lessons have not only help expand my knowledge of networking and programming concepts, but also help me learn faster in the future. I enjoyed having the autonomy to create a project of my own desire with my own parameters, and go about seeking how to complete the project using a range of resources.

6 Sample Demonstration

In this demonstration, I walk through all of the commands I mention in this report, and show how the PU_Bot responds to each. The user with the nickname PU_Bot_2 is a user from my local IRC client, from where I am invoking the commands.

```
<PUB_Bot> Welcome, PU2_Bot! Type '|help' if you want to know what I can do
<PU2_Bot> |help
<PUB_Bot> PUBot Commands:
<PUB_Bot> |time -- Displays current date and time
<PUB_Bot> |translate {desired language}: {string}
<PUB_Bot> |detect : {string you want to know the language of}
<PUB_Bot> |wiki : {topic to search}
<PUB_Bot> |news -- lists top 5 current stories
<PU2_Bot> |time
<PUB_Bot> PU2_Bot: The time is now Sat Dec 08 15:25:03 EST 2018
<PU2_Bot> |translate spanish : can you help me pass my exams?
<PUB_Bot> Translated Text: ¿Puedes ayudarme a pasar mis exámenes?
<PU2_Bot> |translate english : ¿Puedes ayudarme a pasar mis exámenes?
<PUB_Bot> Translated Text: Can you help me pass my exams?
<PU2_Bot> |detect : je m'apelle John
<PUB_Bot> Detected Language: French
<PU2_Bot> |detect : que hora es
<PUB_Bot> Detected Language: Spanish
<PU2_Bot> |wiki : Purdue
<PUB_Bot> Purdue University is a public research university in West Lafayette, Indiana, and the flagship campus of
  the Purdue University system.
<PUB_Bot> Link to full article: https://en.wikipedia.org/wiki/Purdue_University
<PU2_Bot> |news
<PUB_Bot> 1. U.N. climate negotiators sweat over detail and divides (Link: http://feeds.reuters.com/~r/reuters/topNew
  s/~3/3XsEIQjraTA/u-n-climate-negotiators-sweat-over-detail-and-divides-idUSKBN1O70LS)
<PUB_Bot> 2. France's 'yellow vests' clash with police in Paris (Link: http://feeds.reuters.com/~r/reuters/topNews/~3
  /8Z__zPLq3a8/frances-yellow-vests-clash-with-police-in-paris-idUSKBN1O700H)
<PUB_Bot> 3. Many U.S.-bound caravan migrants disperse as asylum process stalls (Link: http://feeds.reuters.com/~r/re
  uters/topNews/~3/xx55bZQgwFI/many-u-s-bound-caravan-migrants-disperse-as-asylum-process-stalls-idUSKBN1O62IF)
<PUB_Bot> 4. Afghan 'Messi boy' forced to flee home (Link: http://feeds.reuters.com/~r/reuters/topNews/~3/NzmU2BTmSY8
  /afghan-messi-boy-forced-to-flee-home-idUSKBN1O70CX)
<PUB_Bot> 5. Trump says Kelly will leave chief of staff job at end of year (Link: http://feeds.reuters.com/~r/reuters
  /topNews/~3/92fRYXLAcso/trump-says-kelly-will-leave-chief-of-staff-job-at-end-of-year-idUSKBN1O70R1)
```

6.1 Git Repository

Below is a link to the Git repository where I have saved my work. Refer to the README file on how to compile the project. I used Gradle to build the project. Since I used Google Cloud Platform in this project, you might have to get your own API keys to be able to use the translate and detect features, but feel free to just refer to the code to see how it works.

https://github.com/NikoTatopolous/Purdue/tree/master/AY19-1/Pircbot

REFERENCES

[1] Cloud Translation API documentation,
https://cloud.google.com/translate/docs/

[2] Wikipedia API Main Page,
https://www.mediawiki.org/wiki/API:Main_page

[3] Lars Vogel: RSS feeds with Java - Tutorial,
http://www.vogella.com/tutorials/RSSFeed/article.html

[4] ISO 639-1 Language Code Support,
https://github.com/TakahikoKawasaki/nv-i18n

[5] PircBot Java IRC Bot,
http://www.jibble.org/pircbot.php