OpenBot

Dhananjay J [*], Jayajith J J [*], Meenu Chandran [*], Guided by Jisha Jose [+]
Mar Baselios College Of Engineering and Technology,Trivandrum

Abstract—OpenBot is a chat bot that can interact with the user and can perform action based on user queries which have a specified format. This is integrated with a multi-user chat application which operates on a server program which is custom made and the user interface is common for bot and chat operations. Bot server handles normal messages and displays its reply to the user. If a query is detected, control is passed to appropriate API Servers and required fields are parsed using the JSON Parser and displayed to the user. Multi-user chat facilitates both broadcast and private messaging capabilities. Communication can be facilitated between clients runing in the same machine as server or otherwise.

Index Terms—Chat Bot, XMPP, JSON Parser,LAN Chat

I. Introduction

CHAT Bots are programs that are designed to emulate human conversations to such an extent that it is not readily recognizable, passing the Turing Test. Turing Test only assesses the similarity between a human and a machine conversation but not the actual correctness of the response by itself. High-end chat bots are powered by Artificial Intelligence (AI) and use Natural Language Processing (NLP). But simple chatbots scan the keyboard inputs and a response is elicited with the most matching keywords. OpenBot is a chat bot designed to interact with the user and also to perform actions based on certain user queries of predefined format. OpenBot can be readily worked with by using any standard XMPP supporting clients like Jitsi, Spark etc. It implements the Pandorabot API. The query modules that we wish to implement are integrated to this bot. The user types in a message for the bot to respond. The text is classified either as a command (as it follows a prescribed format and this case starts with a ?) or as a routine message for conversing with the bot. If detected as a command, the control is passed from the bot to the relevant API servers that are responsible for processing the query that the user has typed in. If detected as a normal message, bot module handles the further processing by fetching the response as mandated by its API and returns to the user. The Chat Bot that implements the Pandorabot API has been named OpenBot owing to its Openness. The bot is an implementation of an open source API as is the query modules that are integrated to it. This makes it easy to add new features and scale the bot to suit different operating environments accordingly. The exact query modules and their functions will be dealt upon in the coming sections. The query modules that we have integrated into the bot are for Weather Search (using Yahoo weather API), Dictionary (Pearson dictionary API),Online encyclopedia search (Using the Wikipedia API) and to search for famous sayings on a topic (QuotesDesign API). OpenBot is linked to the UI of the chat client. The chat client-server modules are designed to support multithreaded chat between clients connected to the network which can be wired or wireless. This uses sockets and is based on UDP protocol. The chat application handle broadcast messages (the messages that are sent to every user who is online) and private messages (between two users). It also performs user validation. The client-server module can facilitate comunication between clients running in the localhost as well as between remote system.

II. Problem Analysis

A. Problem Statement

To design a chat bot which can interact with the user as well as perform actions based on user queries of specified format. Within a subsection, content division may again be included as shown below.

B. Problem Description

To choose an open/closed source API or design a new interface to interact with. The choice of interface along with the formatting and data transfer is a topic to be discussed. Compression formats along with data representation formats and choice of protocol has to be done. Integrating the bot with a utility service like instant messaging and file transfer is to be looked into.

C. Features Of the Project

Through open source API Implementations we can imple-ment features like Weather check,Online Encyclopedia and Dictionary reference as well as searching for famous quotes.

The chat bot is in a common interface with a multiuser chat client that can support broadcast and private messages between clients running in the local host or other machines

D. Problem Solving Methodologies

1) Existing System: Most of the chatbots are primarily used for interacting with the user and most often do not go beyond it.

2) Proposed System: Contrary to existing systems, Open-Bot uses a open design which makes it easier to add new features to the bot and provides a common interface with a crucial utility such as instant messaging and file transfer , thus offering a personal assistant styled help and a communication medium under one banner. The bot module can also work on any standard XMPP client.

III. System Design

The facility in the bot which is provided to answer user queries is due to implementation of open source APIs. Each individual Query module parses the JSON response generated from the API URL and returns the extracted result to the user. The bot is linked to a multiuser chat application which supports messaging and file transfer between clients connected the local network. It uses UDP protocol and a client-server model. Communication is acheived through sockets. Client and server maintain separate set of sockets and I/O objects for messaging and file transfer. Client plays a larger role in file transfer. File transfer is possible for small text based files.

IV. System Implementation

A. Query Modules

1) Dictionary: The module uses an array list data structure. A method to remove spaces from the text input from the user is defined. The response generated from the API URL for Pearson dictionary is fetched fully. The URL is

http://api.pearson.com/v2/dictionaries/ldoce5/entries?headword=”+word

where the *word* part corresponds to the actual word for which the meaning has to be found out. The response is generated in JSON format in the API. Here an input stream object is used to fetch the response and the JSON data is parsed with the help of a JSON Parser. The JSON Parser helps us to traverse the entire generated response and extract only the fields that are displayed.
Part of JSON response when queried to find meaning of the word Apple in Pearson dictionary is as follows:


```
{"status":200,"offset":0,"limit":10,"count":10,"total":11,"url":"/v2/dictio
naries/ldoce5/entries?headword=+apple","results":[{"datasets":["ldoce5","di
ctionary","sandbox"],"headword":"Apple","id":"cqAExNSHvf","senses":[{"defin
ition":["a US computer company whose best-known products are the iPod and
iPhone"]}],"url":"/v2/dictionaries/entries/cqAExNSHvf"},{"datasets":["ldoce
5","dictionary","sandbox"],"headword":"apple","id":"cqAExNC13Q","part_of_sp
eech":"noun","pronunciations":[{"audio":[{"lang":"British
English","type":"pronunciation","url":"/v2/dictionaries/assets/ldoce/gb_pro
n/brelasdeapple.mp3"},{"lang":"American
English","type":"pronunciation","url":"/v2/dictionaries/assets/ldoce/us_pro
n/apple1.mp3"}],"ipa":"ˈæpəl"}]
```


Fig. 1. JSON Response for dictionary query

In this case we are extracting field called senses which in turn is the first field of a list called results. From senses the parser gets the first field in the list structure of senses called definition which in turn has the definition of queried word as its value. So in effect, from the large list that was the JSON response, we have parsed upto the value of the field results.senses.definition which is the text that the bot gives back to the user in the response to the query.

The format of the query is:

?dictionary *word*

2) Weather: The module uses an array list data structure. A method to remove spaces from the text input from the user is defined. The response generated from the API URL for Yahoo Weather is fetched fully. The URL is

https://query.yahooapis.com/v1/public/yql?”+”q=select%20*%20from%20weather.forecast%20where%20woeid%20in%20(select%20woeid%20from%20geo.places(1)”+”%20where%20text=%22”+city+”%22)&format=json&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys;

where the *city* part corresponds to the city name for which the weather forecast has to be found out. The response is generated in JSON format in the API. Here an input stream object is used to fetch the response and the JSON data is parsed with the help of a JSON Parser. The JSON Parser helps us to traverse the entire generated response and extract only the fields that are displayed.
In this case we are extracting field called results which in turn is the first field of a list called query. From query the parser gets the field in the list structure called channel From list structure channel a sublist is reached from where sublist condition and furthermore the sublist forecast is reached. This So in effect, from the large list that was the JSON response, which in turn has the required bot response as its value. We have parsed upto the value of the field query.results.channel.item.condition.forecast which is the text that the bot gives back to the user in the response to the query. The format of the query is:

?weather *city*

3) Wikipedia: The module uses an array list data structure. A method to remove spaces from the text input from the user is defined. The response generated from the API URL for Wikipedia is fetched fully. The URL is
https://en.wikipedia.org/w/api.php?format=json&action=query&prop=extracts&exintro=&explaintext=&titles=”+title Part of JSON response when queried to find meaning of the word Pathology in Wikipedia is as follows:


```
{"batchcomplete":"","query":{"normalized":[{"from":"
pathology","to":"Pathology"}],"pages":{"48791":{"pageid":48791,"ns":0,"titl
e":"Pathology","extract":"Pathology (from the Greek roots of pathos
(\u03c0\u03ac\u03b8\u03bf\u03c2), meaning \"experience\" or \"suffering\",
and -logia (-\u03bb\u03bf\u03b3\u03af\u03b1), \"study of\") is a
significant component of the causal study of disease and a major field in
modern medicine and diagnosis.\nThe term pathology itself may be used
broadly to refer to the study of disease in general, incorporating a wide
range of bioscience research fields and medical practices (including plant
pathology and veterinary pathology), or more narrowly to describe work
within the contemporary medical field of \"general pathology,\"
```


Fig. 2. JSON Response for a wikipedia search query

In this case we are extracting field called pages which in turn is the first field of a list called query. From query the parser gets the field in the list structure called extract which in turn has the required bot response as its value. We have parsed upto the value of the field query.pages.extract which is the text that the bot gives back to the user in the response to the query. If proper match with is not found in the encyclopedia, appropriate error message is returned to the user.

The format of the query is:

?wikipedia *word*

4) Quotes: The module uses an array list data structure. A method to remove spaces from the text input from the user is defined. The response generated from the API URL for QuotesDesign is fetched fully. The URL is

http://quotesondesign.com/wp-json/posts?filter[orderby]=rand
&filter[posts/ per/ page]=1

Part of JSON response when queried to find quotes on the subject Knowledge in QuotesDesign is as follows. In

[{"ID":1270,"title":"Greg Hudson","content":"<p>It is important not to let the perfect become the enemy of the good, even when you can agree on what perfect is. Doubly so when you can’t. As unpleasant as it is to be trapped by past mistakes, you can’t make any progress by being afraid of your own shadow during design. <\/p>\n",

Fig. 3. JSON Response for a QuotesDesign search query

this case we are extracting field called pages which in turn is the first field of a list called title. The title list has three fields which need to process. The first field indicates that the request made by the user was valid, the second field which has the quote wich is response to the user query and the third field is the name of the person whom the quote belongs to.

The format of the query is:

?quotes subject

5) Driver for the query modules: This checks whether user input matches any of the prescribed query format. If so, an object corresponding to relevant query module is created. The parsed field is displayed back to the user if error flags were not set for that particular module. In each of the query modules, a list array data structure is used whose first field indicates whether necessary response has been elicited. The parsed field in each module is placed in the second field of the list array so that upon passing the data structure to the driver module, format check and validity of response can be performed.

B. OpenBot Module

The module contains an interface for creating a session of the chat bot. We have included the implementation of Cleverbot API and the Pandorabot API. Cleverbot was made a paid API in the course of us doing the project, making Pandorabot the API of choice as it preserves the Cleverbots ability to work in any standard XMPP client. A HTTP request is sent to the bot server and the response to the passed user string is processed and returned back to the user. The module uses a HTTP Cookie object that carries state information between server and user agent. Cookie is widely adopted to create stateful sessions. The module include methods to accept user entries in multiple languages and notifies the server accordingly. UTF-8 encoding is used for data transfer. In the user interface for the bot module, when user types in a message, the first position is compared to the character ?. If so it is detected as a command and control is passed to the driver module for the query. Otherwise the control is passed to the bot module which does the further processing.

C. Multiuser chat client

Basic UI for chat client consists of a text area where the messages are displayed, a text field where the user types in the message. Client connects to the server using a port number that is common to both the server and the client. Server IP address is also provided if the client is not running on the localhost. Log-in capability checks whether the username and password provided by the user matches the list of registered user. The message structure maintained by the client is composed of these components : sender, recipient, content of the message and message type (private/broadcast/login/bye).

Client creates a socket with IP Address of the server and the port number. InputStream and OutputStream objects are created for each client with which it can send messages to the server and receives acknowledgement when the client is accepted. Each client runs on a separate thread. If message entered by user is of type message, the message content is displayed on the text area along with the sender and receiver information. The bot is integrated to the client interface. When the button is clicked, the bot user interface is called and that performs the functions mentioned earlier. Each client maintains a history log which is in the form of a table that has all the messages sent and received by the client after the log was active. File transfer was implemented in the client to enable transfer of files between clients.

D. Multiuser chat server

The server asks for the path where the database file is located. This is an XML file that has the list of users registered which includes their usernames and password information. At the time of log-in, client sends the username and password to the server and the server parses the database file to find a match and performs validation. Server also maintains a message format that is consistent with the client. Server creates a socket with the common port number and actively listens for incoming connection. Appropriate error messages are generated if the server cannot bind to the port. Server is responsible for maintaining the list of users who are online. When a client logs out by closing the chat window, a log-out event is generated and the server updates the list by removing the username of that particular client. Similar operation is performed when a new registered user is online. List is updated by adding the particular username to it. The server dispatches the message to the destination client through a socket that was created in the server thread for which there is only once only instance. In case of a broadcast message, the server iterates through every client in the online list and sends the socket to all. In the same way by which texts are exchanged, files can also be sent.

E. File Transfer

For uploading a FileInputStream and FileOutputStream ob-jects are created. A socket is created for file transfer using port number and IP address of server. On the client thread that is sending a file, the user can browse for a file he/she wants to send and the path is stored. This path is used

to create the fileinputstream for this particular thread. The file is accessed and its content is written to a buffer and length of the file is noted down in a variable. For download two sockets are created. One is passive. That is to listen for file transfer requests and is created using server IP and port number. The other socket is for the actual transfer of data. It accepts the requests from the passive socket. Using getRemoteSocketAddress() method, this active socket can get the IP address of requesting client from the passive socket. The fileinputstream object that was used in the sender side is now copied to the fileoutputstream of the receiving side. In this system, server module is not involved in the actual file transfer. Passive socket takes that role. But the server has a message type in it called ”uploadres” and ”uploadreq” which are just a way to display confirmation dialogue on the client side.

V. Limitations of the project

As of any system, this system has its limitations.

- Communication between systems is restricted to those connected within a local network
- The only mode of input to the system is through text.
- Cannot send large files. Limited to small text based files. System is not compatible with mobile devices

VI. Future Enhancement

- To make the system compatible with mobile devices.
- Introducing new modes of input. Eg. VOIP for instance Include other formats and higher file size for transfer
- Adding more APIs for the bot to make it even more useful.

VII. Conclusion

OpenBot is a chat bot that is designed to work with any standard XMPP supporting client. Bot Server deals with the normal interaction with the user whereas the query modules integrated to it are handled by the relevant API servers. The query modules extract the required fields after parsing the entire JSON response generated by the API URL. The Bot is linked to a multithreaded chat supporting client-server system. Communication can be facilitated between systems connected to a local network. Client has to provide servers IP address and a common port number (shared between client and the server) to log-in. Validation is performed by parsing the XML database file. Chat history log is maintained for each client after the client chooses a log file to save into. File transfer is possible for small text based files. Clients have a passive socket especially for this purpose, minimizing the role played by the server.

References

[1] Arun Gopi, Shobana Devi P, Sajini T, Jose Stephen, Bhadhran VK,”Multilingual Speech to Speech MT based chat system”, Conference on Computing and Network Communications (CoCoNet’15), Dec.2015

[2] Jeesoo Bang, Hyungjong Noh, Yonghee Kim, and Gary Geunbae Lee,”Example-based Chat-oriented Dialogue System with Personalized Long-term Memory”,International Conference on Big Data and Smart Computing (BIGCOMP),2015

[3] Ibrahim Muhammad Abba,Norshakirah Ab. Aziz,Umapathy Eaganathan,Janet Gabriel,”LAN Chat Messenger (LCM) using Java Programming with VOIP”International Conference on Research and Innovation in Information Systems(ICRIIS’13),Nov. 2013