DESIGN AND DEVELOPMENT OF AN INTERACTIVE ROBOTIC CONVERSATIONAL COMPANION FOR ELDERLY PEOPLE

Gonapinuwala Withanage Malith Manuhara

149357H

Degree of Master of Science

Department of Electrical Engineering

University of Moratuwa
Sri Lanka

April 2019

DESIGN AND DEVELOPMENT OF AN INTERACTIVE ROBOTIC CONVERSATIONAL COMPANION FOR ELDERLY PEOPLE

Gonapinuwala Withanage Malith Manuhara

149357H

Dissertation submitted in partial fulfillment of the requirements for the degree Master of Science

Department of Electrical Engineering

University of Moratuwa
Sri Lanka

April 2019

Declaration of the Candidate & Supervisor

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: Date:

The supervisor should certify the dissertation with the following declaration.

The above candidate has carried out research for the Masters dissertation under my supervision.

Signature of the supervisor: Date:

Abstract

The ageing of population is rapidly accelerating worldwide and as a result countries are facing social and economic challenges. Hence, the majority of the elderly population all around the world is facing difficulties.

The loss of ability is typically associated with ageing and the elders require special attention in both physical and mental concerns. The requirement of a suitable caretaker becomes very important in caring for an elderly person. A human caretaker would be the ideal solution. But the availability of such genuine resource is a very rare luxury in the modern society. Hence the society and the elderly population are in need of a suitable alternate solution. Introduction of service robots has become a very promising development in addressing problems faced by elderly population in the world. This research work proposes a robotic conversational companion capable of vocal interaction with elderly users in human like dialogues, during service assistance.

A Finite State Interaction Module (FSIM) and a regular expression based language identification method have been introduced for facilitating this task. A Knowledge Database (KDB) containing specific data has been designed, implemented and connected with the robot system to enable more meaningful and natural dialogue creation. State transition diagram and event flow diagrams explaining the functionality of the states are presented. The robots performance has been evaluated by user rating.

Experimental results including a selected segment of conversation are presented with an analysis including the change of FSIM states. Human user has been asked to interact with the experimental setup and rate the user experience varying from “Very Bad” to “Excellent”. The evaluation results have indicated a high user satisfaction rate close to “Good”, validating the robots capability to interact in a human friendly manner during service assistance.

Acknowledgement

A great many people have contributed the completion of this dissertation. I owe my sincere gratitude to all those people who have made this dissertation possible. My supervisor Dr. Buddhika Jayasekara has given me the greatest support and guidance throughout the research project. His patience and support helped me in many difficult situations. I am also thankful not only for his technical advice but also for advice on my writings, carefully reading and commenting on revisions of this manuscript. I would also like to acknowledge the support provided by Mr. Viraj Muthugala.

Especially I would like to thank all the academic staff of the Department of Electrical Engineering, Faculty of Engineering, University of Moratuwa for their guidance. I would also like to acknowledge for technical assistant staff of the Department for their support for laboratory work.

None of this would have been possible without the love and patience of my family.

Table of Content

List of Figures

List of Tables

List of Abbreviations

ABNF	Augmented Backus-Naur Form
API	Application Programming Interface
DOF	Degree of Freedom
FSIM	Finite State Interaction Module
HMM	Hidden Markov Model
IDE	Integrated Development Environment
MFCC	Mel-frequency Cepstral Coefficients
NLP	Natural Language Processing
ROS	Robot Operating System
TTS	Text to Speech

CHAPTER 1: **INTRODUCTION**

1.1 **Service Robot for Elderly People**

One of the major problems faced by majority of the elderly population in the world is inability to live independently. Ageing makes it harder for them to cope with day to day activities such as health care, mobility and fulfilling personal needs [1]. Not only these physical difficulties but also difficulties related to maintaining interpersonal interactions & relationships are regularly experienced. Hence, lack of companionship, loneliness, depression and social isolation are common issues to elderly population. Anyhow, caring assistance from a genuine human care taker is not an easy option always [2] [3].

As a result, elders are seeking for some sort of assistance through technology for the activities that they are unable to perform independently [4]. With technological advancements and new research and developments, service robotics is becoming a very promising development in addressing the above issue [5][6]. However, it is important to know the perspective of the society regarding the idea of machine taking care of their loved ones; it shows that the society is well prepared to adopt this new method [7][8]. Further, researches indicate that robots can take care of elderly while improving social relations without making negative effect on elderly person's dignity [9].

1.2 **Human Robot Interaction**

In order to achieve these, robots should be carefully developed with human friendly intelligence. Robots should have better social behavior and special social skills beyond just the functional capabilities [10]. Hence, communication skills of the robot play a critical role in enabling more natural Human Robot Interactions (HRI). HRI is a field with high attention from both academic and industrial community. It

focuses in understanding, designing, and evaluating robotic systems pertaining specially to humans [11].

1.3 **Importance of Natural Language**

An ideal service robot designed for elderly should be capable of conducting natural human like communication since vocal instructions being the most sensed and preferred communication method practiced by humans [12]-[14]. The elderly community is looking forward for technological assistance in their day today service requirements without feeling any difference to having a human caretaker. Therefor human friendly intelligent service robot designed for assisting such elderly community should have the ability of understanding human vocal instructions and appropriately responding with natural human to human like communication.

1.4 **Objective of the Research**

Many researches have been conducted on improving the vocal communication between robots and humans. However, it still remains an unresolved quest of finding the perfection in human robot interactive communication [26]. Keeping that in mind the objectives of this research work are defined as,

- Propose a suitable architecture for developing a service oriented conversational companion robot for elderly people.
- Design and develop a conversational companion robot that is capable of interacting with elderly user with human like interactive dialogues for assisting them during service requests.
- Propose a suitable method for validating the proposed system and validate the developed system to determine the robots capability to interact in a human friendly manner during service assistance.

1.5 **Human – Robot Communication for Service Robots**

Method of developing a human - robot communication model based on multimodal and multisensory human behavior analysis has been presented. The model makes use of both verbal and nonverbal behavioral patterns of elderly to analyze and provide mobility assistance through human interactions. The researchers highlight the essential requirement of an interactive robot system capable of employing service capability improved through communicative actions and to enable a strong and competent communication with the service recipient. Technological fields of speech recognition, gesture interpretation, dialogue management, speech synthesis, and generation, face detection, recognition and detection of human activities have been used to correspond or to mimic human communicative capabilities in the developed communication model. The model considers several communicative functions that incorporates knowledge from natural human-human interactions. The communicative functions are action-directives, information-requests, assertions and meta-communicative actions. The communication model proposed is a mix of a state and frame-based model and the mix is in line of McTear's categorization. Observations from human communication behaviors and patient-care interactions have been abstracted into human-robot communication patterns. Recording scenarios and measurements close to real life situations have been collected for developing the HRI model. [15]

Details on communication model that needs to be integrated into the robotic platform in order to allow a more natural human-robot interaction through a multimodal architecture has been discussed. Natural communication has been implemented based on activity detector and a Hidden Markov Model (HMM). The spoken command recognition system uses activity detector as the first step to detect time segments in the audio stream containing spoken commands. Then, a set of HMM models are trained as the second step. A study has been conducted to identify natural multimodal interactions. These interactions corresponds to core activities of the addressed audience. Information with respect to human to human communications during the performance of day to day activities in between elderly

people and their human caretakers are been analyzed by the study. Improvement to the proposed communication model by addition of enhanced set of audio-gestural commands is suggested as future work. In addition, the researchers suggests to finalize the proposed communication scheme by developing an algorithmic flow diagram of all possible interaction states pertaining to the communication model. Further the researchers highlight the fact that development of safe systems specially designed for indoor mobility assistance must include effective and natural interaction between the particular system and the operator. [16]

Researchers highlight the fact that determining the success in developing assistive robots is considered to be majorly based on user evaluation and acceptance. Implementation has been evaluated with end users and promising results are available for the navigational assistance in real-world scenarios. The evaluation has been performed in two phases. During the first phase, a participants group of 36 patients have been selected according to well-defined criteria. The group members have been selected from various environments under the supervision of a medical institution. In respect of evaluating communicational aspects during the first phase, users native language, German, has been used to repeat a sequence of five commands three. The final phase of evaluation has been conducted with the participation of more than 30 subjects. Audio-gestural human-robot communication and cognitive (navigation) assistance system has been validated by this study of evaluation. Two specific test scenarios has been developed. The developed test scenarios consist of tailored assessment methods and performance metrics of qualitative as well as quantitative analysis. The researchers identify that standard methodology or a standard scale has not been developed for measuring the user satisfaction based on user experience of interacting and receiving services from an assistive robot or any technology similar to that. Hence the researchers categorize the evaluation process as rather difficult. Therefore, a custom questionnaire has been used by the researchers to measure the user satisfaction level. The researchers have failed to use validated scale or a questionnaire resulting limitations in comparing the final results with any other previous work conducted by other researchers. [17]

Robot Operating System (ROS) has been based for designing a robot capable of object sorting. ROS is a distributed processing framework used for development of robots. The robot is capable of communicating with human beings. One out of three embedded computers in the robot system; a microphone and a speaker is connected to one of them in order to implement natural language interaction and reasoning. This implementation of natural language interaction and reasoning aspects act as the brains of the system. The overall system consist of multiple nodes. Out of which the input speech is used by speech recognition node for generating a text string. Following processers are used by speech recognition node

I. Input speech signal noise reduction
II. Mel-frequency cepstral coefficients (MFCC) feature extraction
III. Speech decoding by combining the Hidden Markov Model (HMM) acoustic model and Ngram language model.

The speech synthesis node receives the output text. The speech synthesis node utilizes text-to-speech (TTS) technology. TTS process the recieved text. The researchers propose a human-robot-environment interactive reasoning mechanism. This mechanism is based on traditional CBR-BDI mechanism. Further they have proposed a "dialogue and 3D scene interaction module" to improve the performance of traditional CBR-BDI mechanism. Desire analysis is an additional function achieved by the interaction module in addition to realizing the traditional function of map matching. The robot has been designed in such manner where it is capable of analyzing user's desire. The robot can take the initiative to guide users through dialogue when the user's desire is incomplete and the user's former desire can be replenished by using input information. The guidance base is constructed based on dialogue generating function corresponded by each node. [18]

A conversation architecture based on Grice's maxims has been used for developing chatter bot for customer service applications. Researchers identify that existing well performing conversation generation approaches exchanges pairwise utterances only and doesn't perform well in longer conversations. Further these platforms utilize linguistic and stochastic principles and have low grammatical and structural aspects.

The aim of this research is to develop chatter robots that can perform long conversations with more meaningful human-like behavior. In order for robot to represent the behavior of a human representative, chatter agents capable of continuing longer conversations have been designed and the robots have been utilized with capabilities to perform conversations beyond just question and answer sessions. Chatter bot conversations with more meaningful content and context has been simulated using the proposed architecture by the research team. The conversation development module is made of Knowledge Engine and the Conversation Engine. A speech acts hash table and topics hash table creates the Knowledge Engine where a context map defines each topic. The Conversation Engine is made up with the conversation planner module and automated probabilistic finite state machine. The evaluation mechanism is based on the Grice's cooperative maxims. Grice's cooperative maxims is the most basic idea behind theory of pragmatics. Artificial conversation capabilities of several other existing chatter bots have been compared against actual conversations. The results describes that sample conversations grade well against Grice's four maxims by satisfying quality maxim, quantity maxim, relation maxim and manner maxim. The results exhibits that the chatter bot can hold meaningful conversations and develop long conversations. [19].

An interactive dialog system capable of continuing a conversation with the user for retrieving user's personal details has been developed. Natural Language Processing has been utilized for extracting user's personal information and developing a detailed user dataset. The main goal of natural language processing is to give a computer with the ability to interact with people the way people interact with each other. This user profile has been used to customize and adopt conversations to make the relationship between the user and robot more trustworthy and comfortable. The user profile has been designed to update automatically whenever new knowledge is extracted from dialogues. The proposed system platform has been developed using python. Personal details and keywords are extracted by the Rapid Automatic Keyword Extraction (RAKE) and search for text functions. Both these technologies are python implementations. This developed system has been tested using college

students. Ten students in the age group of 20 to 30 years representing both sexualities. The survey group had been asked to grade the robots performance in terms of relevancy and capability to identify the user. [20] [21]

The research paper presents a review conducted on service robots that requires to engage with uncertain information in natural language vocal interactions. It highlights the use of fuzzy systems as a method of decoding uncertain information. The existing systems developed for decoding uncertain information in vocal interactions could not match the human cognitive capabilities pertaining to decoding uncertain information. [22] A method has been proposed to successfully decode uncertain information pertaining to navigational instructions. The proposed method is effective and it uses robots knowledge in uncertain information. Robot gathers knowledge on uncertain information by analyzing the environment and refereeing user descriptions. Certain functional modules such as Uncertain Information Understanding Module (UIUM), Feedback Evaluation Module (FEM), Interaction Management Module (IMM) and Robot Experience Model (REM) have been introduced in the proposed method. The UIUM module is made up of fuzzy based neural networks. All the information pertaining to robots understanding on surrounding environment, actions being taken place and context are organized in the REM. Language memory is used to analyzed and identify the vocal commands. The language memory consist of various keywords, lexical symbols and basic grammar components. The developed system is capable of updating the REM with the facts learned through interactive conversations. The vocal interactions in between the robot and the user are managed by means of a finite state module. The success of the developed interactive robot system has been measured through a user study. An artificially created indoor environment has been used to conduct the said user study. The performance results of the study is based on an index. Study results based on this "satisfactory level" index shows that the performance of this proposed method is better than existing methods. [23] [24].

An overview of research in human–robot interactive communication has been presented. Research advancements in both verbal and nonverbal aspects have been discussed [25].

In summary, it is evident that Research and developments on Human – Robot Vocal interaction for Service Robots is a live topic at present among academia. Various algorithms and methods have been experimented in developing vocal interaction mechanisms suitable for different applications. Available literature expose service robots with vocal interaction capability designed for different applications such as,

- Navigational Assistant
- Object Sorting
- Customer Service
- Information Extraction

Development of an Interactive Robotic Conversational Companion for Elderly People is a new addition to the above list.

1.6 **Research Methodology**

First step is to clearly identify the inputs, outputs and main components of the desired robot system. Then, a suitable mechanism has to be proposed to design the core of the robot system to cater the service oriented human like communication requirement. There is a very vast scope for service oriented applications and related communication requirements when assisting elders in their day to day activities, hence the development scope of this research required to be limited to satisfy the testing and evaluation process to complete the research within time limitations. Identification and recording of natural human like dialog patterns pertaining to service oriented requests and responses is the next important step. Since the proposed robot system should conduct human like natural communication, it is required to collect sufficient amount of data for identifying natural human like dialog patterns. The Next step is to design the core of the robot system and develop it to operate on

real word hardware with input and output mechanisms. For this, appropriate hardware and software requirements has to be identified and the designed robot system has to be developed on them.

Finally, the developed robot system need to be validated to determine the robots capability to interact in a human friendly manner during service assistance. For this, a rating based evaluation method is to be used. The robot system is put into interact with real human users and the user rating is recorded for final evaluation.

1.7 **Organization of the Dissertation**

This is the introductory chapter explaining the requirement for initiation this research work and the scope of the work being done. This chapter also describes related research work conducted by other researches. It presents background information required for understanding latest advancements of the field and makes familiar with existing developments. The rest of this dissertation is organized as follows.

Chapter Two elaborates on functional overview of the robot system. Implementation of Finite State Interaction Module (FSIM) is explained in Chapter Three. Chapter Four describes special architectures used in the robot system to enable more human like dialogue interactions. Robot hardware and software structure with implementation methodology of the experimental setup is explained in Chapter Five. Finally, the chapter Six concludes the research work with results and conclusion.

CHAPTER 2: **SYSTEM OVERVIEW**

Input and output of the designed robot communication system shall be human voice and machine voice respectively. The elderly user may utter service requests and the robot shall be capable of understanding and providing meaningful responses while developing a natural dialogue.

The complete functional overview of the system is illustrated in Fig. 2.1 in a modular format. The basic function of the illustrated system is to interact with the elderly user through human like dialogues as an assistive conversation companion. The system capacity mainly covers service assistance where it is capable of recognizing and understanding user utterances and responding to assist the elderly user. Recognition and understanding of the user utterance is performed by the Voice Recognition and Understanding module where the user utterance is ultimately converted into a raw text. Matching the user voice input and production of a raw text transcription is performed with the help of Google Speech Recognition Application Programming Interface (API). Recognized raw text is then processed to comply with the Interaction Management Module (IMM) through String Conditioning. All the interactions are managed by the IMM which consist of a Finite State Interaction Module (FSIM) and a Scheduler Engine (SE). SE sub module is responsible for invoking reminders for the user. It enables the system to bring the attention of the elderly user towards scheduled tasks and provide further instructions. A particular recognized text pertaining to a particular user utterance is processed through the IMM and a relevant response is generated in text format. This text is then fed to the Voice Response Generation module where the Voice Output is generated with the help of Python Text to Speech (TTS) library. In order to generate more user friendly dialogues, the system consist of a Knowledge Database (KDB). KDB is designed for storing information related to language patterns, user specific information and object specific information. The three sub databases of KDB are referred by the IMM for generating more user friendly responses. The system is capable of updating the KDB with reference to the user interactions.

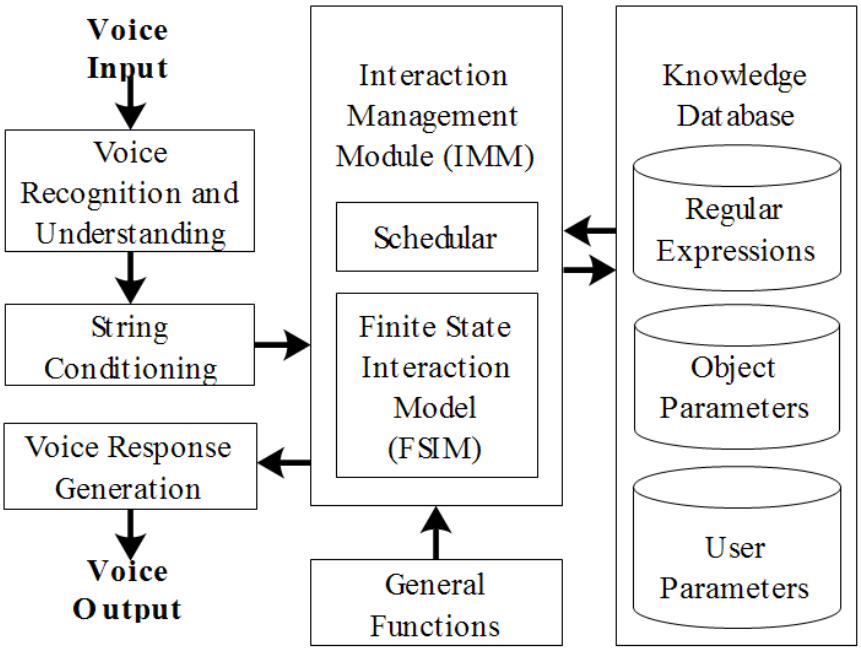


Fig. 2.1 : Functional overview of the system.

2.1 **Software and Tools**

The proposed system has been implemented using Python. Python is a programming language developed by Guido van Rossum available in 1991 for the first time. It is customary in use as a general-purpose programming language where high-level core interpreted to suit the purpose. Python code is readable and use large amount of whitespaces. It is a core design philosophy of this language. Python version 3.5.2 has been used in this development.

Fig. 2.2 : Symbol of Python programming language

As the source of Python, Anaconda has been used. Anaconda is a distribution of Python and R programming languages. It is free and open source. This distribution

of Python and R is mainly used for developing applications requiring data science and machine learning aspects. Since, it simplify deployment and package management. Anaconda version 4.2.0 64 bit version has been used in this development.

Fig. 2.3 : Symbol of Anaconda open source distribution

PyCharm is a source code editor or Integrated Development Environment (IDE). This IDE is customary designed for the Python language by the Czech company JetBrains.

Fig. 2.4 : Symbol of PyCharm IDE

2.2 **Voice Recognition**

Voice or speech recognition plays a key role in this development. The robot should be able to clearly identify and recognize voice content from the elderly user in order to reply with a proper respond. Voice recognition take place in several steps. Ultimate target of these steps is to convert the vocal information into machine understandable and easily processable format, in simple words it is the process of converting spoken words into text. Human user's voice is the first component of the

process. This physical energy is transferred into an electrical signal using a microphone. The result is an analog electrical signal, representing the original voice data. Next the analog signal is digitized to make it a machine understandable and easily processable signal. An analog to digital converter module is used for this conversion. Still the vocal data is available as a digital signal which is a very easily processable format. This digital signal is converted into a text string using several software modules. A text string is a very suitable format for information extraction and manipulation.

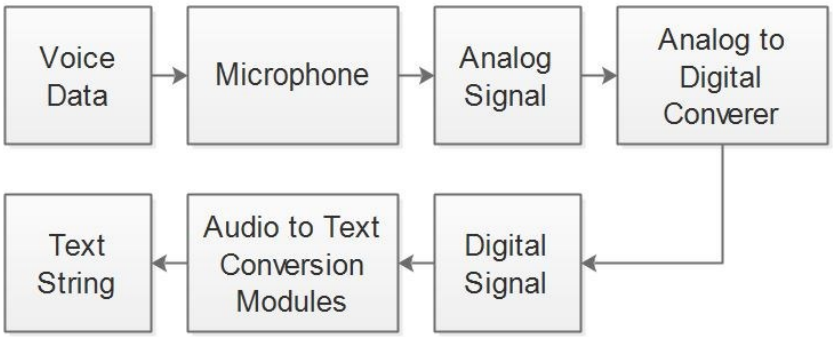


Fig. 2.5 : Voice to text conversion process

A typical digital audio signal to text string conversion process consist of a complex methodology, involving simplification algorithms and efficient computation methods. In general, the process would involve identification of audio segments of utterances by splitting the waveform by audio segments of silences. Then the selected audio portions have to be further sliced to form smaller pieces of voice segments, such as audio wave segments of as long as 10 ms duration. Finally, extract information from the smallest piece to select the best matching combination from a database.

Python supports many speech recognition engines and Application Programming Interfaces (API) such as,

i. Google Speech Recognition API
ii. Google Cloud Speech API
iii. Microsoft Bing Voice Recognition
iv. IBM Speech to Text
v. Wit.ai
vi. Houndify
vii. CMUSphinx
viii. Snowboy Hotword Detection

These different engines and APIs use different technologies and architectures to determine the text equivalent of a vocal signal. Therefore, the final outcome of each engine or API differs with respect to performance (speed, accuracy ect.). Most modern speech recognition systems rely on a model known as Hidden Markov Model. It has been proven to be really practical for speech recognition. However it should be noted that speech to text translation is never 100% accurate due to probabilistic nature of the speech and immense potential exist in that focused area of research and development.

Google Speech Recognition API has been used in this conversation robot development due to its free availability. However, changeover to any other API or Engine does not affect the core development.

2.3 **String Conditioning**

Next stages of the robot system process the speech text generated from the Voice Recognition module. Since, this text is generated from Google Speech Recognition API; it is required to format and generate a suitable text input required by the robot system. Any special characters are removed and letters are converted in to lower case in formatting the speech text.

2.4 **Interaction Management Module (IMM)**

This is the core module of the robot system consisting of astate machine and a scheduler engine. Voice output text or the responses from the robot system are generated based on the voice text input and the knowledge database. General functions that are required for the generation of robot response and understanding of the input voice text are linked to this core module.

2.4.1 Finite State Interaction Module (FSIM)

Finite State Interaction Module consist of a Finite State Machine (FSM) which controls the core functionality of the robot system. A FSM consist of finite number of states, inputs and outputs. The active state can change from one state to another in response to change of external inputs, such change of state is called a state transition. A FSM can be represented by state diagrams as shown in Fig. 2.6. Every state is represented by a circle and an arrow marks the transition from one state to the other based on the current state and the input to the state machine. Transition from one state can happen to one or multiple other states in response to external inputs.

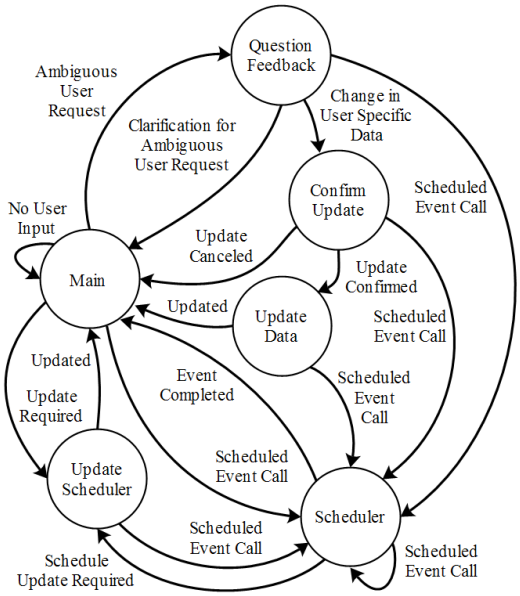


Fig. 2.6 : Example of a State Diagram

The FSM of the robot system is developed in a software environment with six number of states. An instance is created for each state inherited from the state class. Each state has specific function calls which is capable of rendering the assigned functionality.

2.4.2 Scheduler Engine (SE)

Information on routine scheduled events can be inserted in to the system for the invocation of reminders of those events time to time. It enables the system to bring the attention of the elderly user towards scheduled events and provide further instructions. Specific time and an event tag is available as system information. The robot system will continuously check with the current time for any possible occurrence of the event and change the current functional state after informing the elderly user regarding the event. Then the elderly user can further interact with the robot system for taking necessary actions regarding the routine scheduled event that has been reminded of.

2.5 **Knowledge Database (KDB)**

KDB is a predefined information set available for robot system for generating more human like and meaning full responses. It consist of three type of information,

i. Regular Expressions Library
ii. User Parameters
iii. Object Parameters

All these information relevant to the particular elderly user are preloaded to the robot system. But it is possible to modify, insert new or delete any existing record of information later without making any changes to the system code. Further it is

possible for the elderly user to update any particular information while communicating with the robot system.

2.6 **Voice Generation**

The robots vocal output is generated by a Text to Speech (TTS) engine. IMM in combination with the KDB and general functions generates a text string as the robot output. This text is then converted or synthesized into audible voice by this engine. Several such engines are available for developers,

i. Google Text to Speech (gTTS) API
ii. iOS Text to Speech and Speech Recognition.
iii. Microsoft speech engine developed for Microsoft Windows 10 users.
iv. IBM Watson Text to Speech API

In this robot system, Python Text to Speech library "pyttsx3" is used as the TTS engine. It is an open platform TTS library which uses specific speech engines in reference to running operating system as follows.

i. nsss - NS Speech Synthesizer on Mac OS
ii. sapi5 - SAPI5 on Windows
iii. espeak - eSpeak on any distribution or platform that it capable of hosting the shared library

CHAPTER 3: **FINITE STATE INTERACTION MODULE**

3.1 **Finite State Machine**

The proposed system is capable of understanding user utterances and respond. When it comes to service oriented user input, the system is further capable of interacting with the user through human like dialogues to obtain extended clarifications of service request followed by suitable responses. In addition, scheduler functionality is also handled by the system to be more meaningful in elderly assistance. The IMM handles all these functionalities using FSIM and SE modules. FSIM holds the major functional responsibilities. The functionality of FSIM is depicted in Fig. 3.1 as a finite state transition diagram.

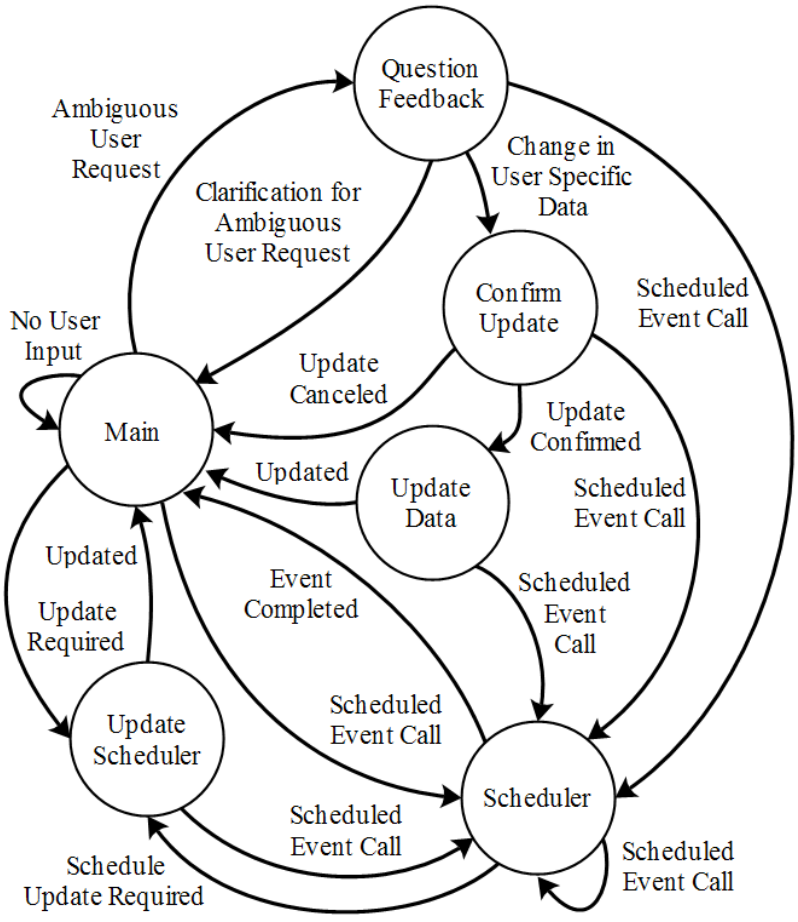


Fig. 3.1 : The finite state transition diagram of finite state interaction module.

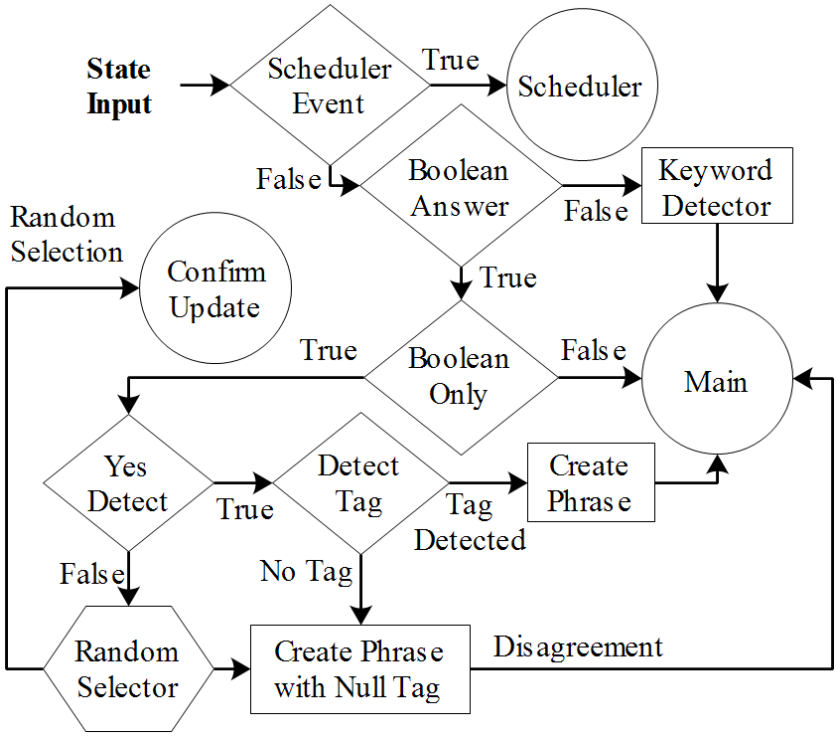


Fig. 3.2 : The event flow of the "Main" state

3.2 **Functionality of States**

The six finite states available in the FSIM is responsible for human like response generation, user interactions pertaining to scheduled events and updating information in the KDB. The default state of the FSIM is Main and transition to Scheduler state is possible from any state. Each and every state has its own unique functionality which has been modeled using event flow diagrams. The resulting response or applicable state change for a particular user input is decided by the event flows.

The functionality inside the "Main" state is depicted by the event flow diagram shown in Fig. 3.2. The state input is first analyzed against the KDB and forwarded for classification. Classification state differentiate between service oriented user inputs and other. If some other state input is identified, then it is again checked for any scheduler event and the state would be transitioned to "Scheduler"

state if detected. The event flow would process any unrecognized event separately and deliver a suitable response. User attempt to update the scheduler activities are also treated as a service oriented request and handled separately after service oriented classification stage. For a specific service oriented request, the event flow includes a random behavior. The event flow would randomly select in between providing a direct response or further extending the interaction between the user and the robot. The extended interaction is for requesting further clarification or providing any comment on the user service request. The action classifier would shift between an extended dialogue requiring further clarifications and an extended dialogue involving system responses with suitable and appropriate comments or feedbacks regarding the user service request. It would enhance the human like interaction nature of the dialogues. If any requirement for clarification is detected, a clarification response is raised followed by a transition to the state "Question Feedback" where the event flow for extended interaction involving service request clarification is handled. A clarification response can be informative, based on the available information in KDB. Since the clarification generator would refer the KDB for gathering of useful information. Otherwise a suitable response is compiled by the feedback generator, referring the KDB.

Fig. 3.3 shows the event flow diagram applicable to the "Question feedback" state. Scheduled events are again handled in a similar manner as in "Main" state. The core functionality of this state is to analyze user responses for the clarification questions and respond accordingly. The desired user answers are identified and categorized in to three sub categories for processing, 1) Answers with Boolean response only 2) Answers with Boolean response and additional information 3) Answer without Boolean response component. User is prompted randomly for an attempt to update KDB when the user disagrees for a certain informative clarification request. If not, all the other responses ultimately cause transition to the "Main" state. This state transition take place with different types of state inputs to the next state. The keyword detector, create phrase and create phrase with null tag stages will generate the appropriate state inputs.

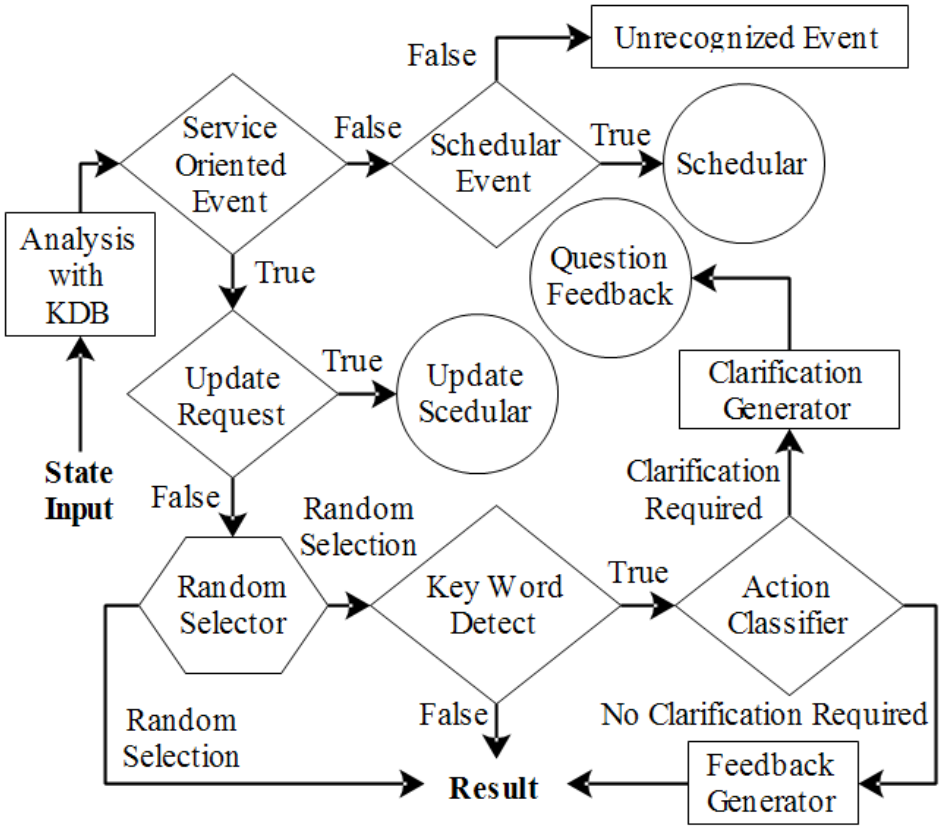


Fig. 3.3 : The event flow of the "Question Feedback" state

"Update" and "Confirm Update" states are responsible for updating the KDB as per the user requirement. It is important to bring out the attention of the elderly user towards scheduled events such as medicine intake, food intake etc. Event flows pertaining to "Scheduler" and "Update Scheduler" states as shown in Fig. 3.4 handles interactions with the elderly user for communicating these pre-defined scheduled tasks and performing any amendments for available scheduled events as well. When the system reminds of a particular scheduled activity, user can either acknowledge it by interacting with the. System or make an amendment for the current scheduled event. It is very important to make sure that these elderly users specifically follow their routine courses such as medicine and food intake. Hence, if either of those acknowledgements to a scheduled event has not been recorded, the warning alert stage will force the user to act accordingly.

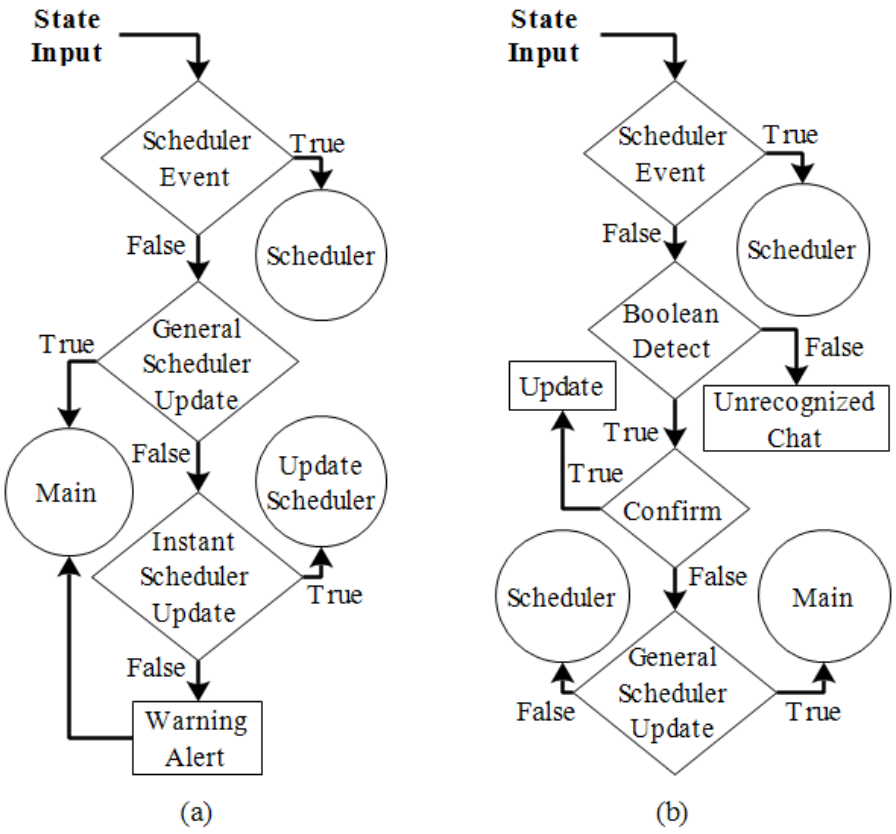


Fig. 3.4 : (a) The event flow of the “Scheduler” state (b) The event flow of the “Update Scheduler” state

CHAPTER 4: **USER INTERACTION**

One of the key aspects of this effort is to develop a conversation companion that is capable of conducting service oriented conversations that are human like as much as possible. In order to accomplish that requirement, several approaches has followed. However, first we have to accurately recognize the user input. Voice input recognition and categorization takes place based on a Regular Expression Based Language Pattern matching method. The combined result of all of the following approaches are set to render a more human like conversation capability by the service robot.

4.1 **Regular Expression Based Language Patterns**

Voice input from the user is converted into a text by the speech recognition mechanism. This text is then matched with a set of regular expressions to recognize the content. Regular expressions are basically a miniature but highly specialized programming language that can be embedded and used inside other programming languages. Regular Expressions specify the rules for a set of possible strings that we want to match. Rules can be defined using Regular Expressions to match specific string patterns such as telephone numbers, email addresses and any other sentence with special characters or pattern. Regular Expressions use a set of ordinary characters in a special purpose, these special characters as named metacharacters. Fig. 4.1 shows the metacharacters of Regular Expression Language.

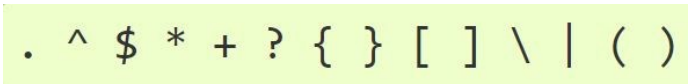


Fig. 4.1 : Mettacharacters of Regular Expression Language

These metacharacters are used when defining rules and each of these character has a defined functionality. For example, [^a] will match any character except ‘a’ and [a-c] will match any character ‘a’, ‘b’, ‘c’. The backslash character is used to

escape metacharacters. Other special characters preceded by the backslash mean unique signals resulting escape of metacharacters. Following are sample code segments of such regular expressions available in programme code.

i. (?:can you |could you |)(?:please |)(turn|switch) off (a |an |the |that |this |)(.*)
ii. i would like to (?:have|take)(?: my|) ([a-z]+) ([0-9]{1,2}) (minutes|hours) later(?: every day|)
iii. (i would like to |lets)(have |take)(?:a |an |the |this |that |some |my |)(.*)
iv. i would like to (?:have|take)(?: my|) ([a-z]*medicine) (?:at |by |)([0-9]{1,2}) ([0-9]{1,2})(?: every day|)

User voice text is matched with one of such expressions and then relevant information is extracted for providing a suitable response. These regular expressions are available in combination with meaningful direct responses and the value pairs are stored in dictionary data structure.

Regular Expression : (?:bye|goodbye|cheerio)
Responses : "bye", "goodbye", "cheerio", "take care", "bye take care", "see you later", "catch you later"

```
regex_pair = (

(r'(?:bye|goodbye|cheerio)',

("bye", "goodbye", "cheerio", "take care", "bye take care",
"see you later", "catch you later")),


(r'(?:can you |could you |)(?:please |)(turn|switch) (a |an
|the |that |this |)(.*) off',

("ok", "sure", "sure i will", "ok give me a minute", "sure
give me a minute", "give me a minute ill %1 off %2%3", "just
give me a minute, I will turn off")),
```

```
(r'(?:how are you|how do you do)',
("i am fine", "i am fine thank you", "i am doing great", "i am doing fine")),
)
```

A dictionary is a mapping of unique key value pairs. Further, it is an indexed, unordered data structure. In this dictionary structure, the value component is a data array consisting of multiple entities. An array is another type of data structure which can store multiple values.

```
robot_responses = pattern.match(user_input)
selected_response = random.choice(robot_responses)
```

The user input is matched with a regular expression pattern and the corresponding set of responses are selected. One response out of the set is selected randomly for generating a default robot feedback. Responding with a default feedback is a basic level capability of the robot system. More methods of developing meaningful and more human friendly responses are discussed below.

4.2 **Information Source**

It is required to possess certain amount of subject information so that a person can actively participate and add value to an ongoing dissuasion. It would be more or less the similar when developing a response for continuing a service oriented conversation. Hence, the robot system should have access and should be able to properly handle certain amount of user specific information and object specific information for it to participate in a more natural human friendly conversation. Fig. 4.2 and 4.3 depicts the terminal output during a live conversation. The shown conversations carry special characteristic which will be discussed in next two subsections.

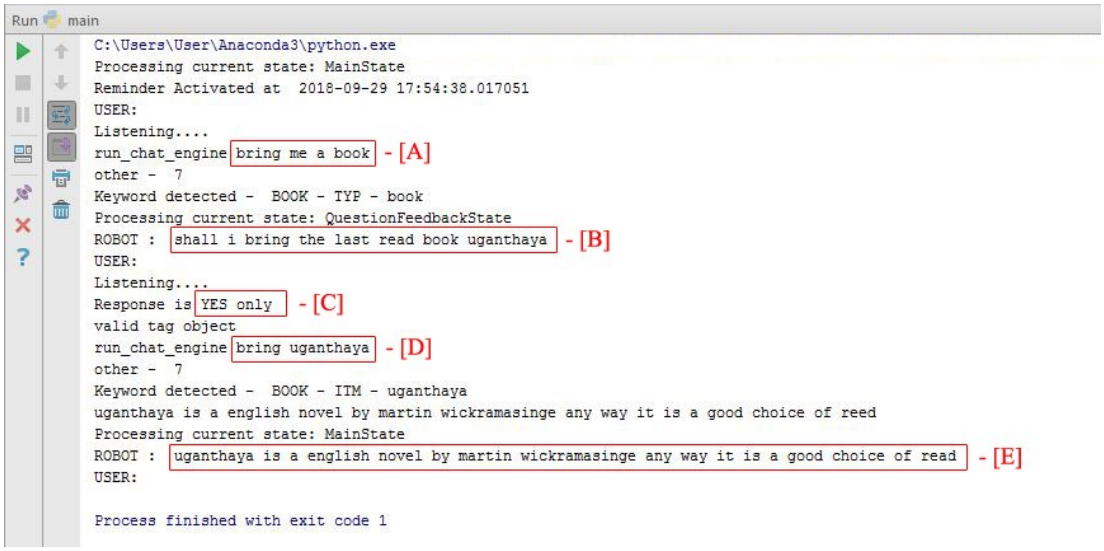


Fig. 4.2 : Terminal output during a live conversation between the robot system and the human user

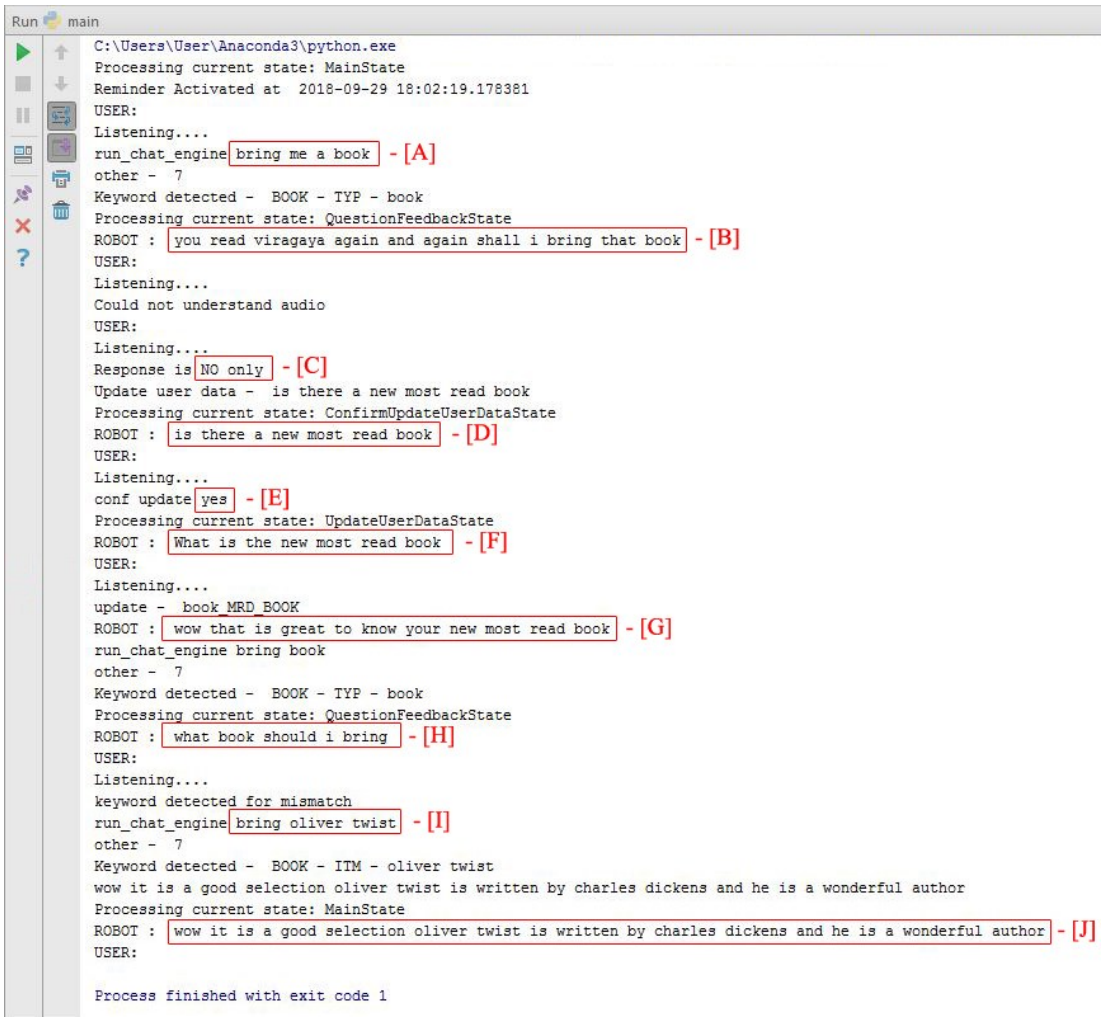


Fig. 4.3 : Terminal output during a live conversation between the robot system and the human user

4.2.1 Question Feedback

The robot system may respond the user with a question, when a feedback is required with further clarification to the initial service request. For example as shown in Fig. 4.2; the robot will ask back a question [B] to clarify the service request [A]. In order to ask the specific question and receive the confirmation [C], the robot should possess certain knowledge pertaining to the user preferences or behaviors. These information is available to the robot system through KDB as follows. These parameters are preset, but provisions for altering based on any fresh knowledge is possible during a conversation.

```
TAG_DIC["novel_LRD_BOOK"]          = "oliver twist"
TAG_DIC["magazine_MRD_BOOK"]       = "ieee"
TAG_DIC["food_FAV_FOOD"]           = "bread"
TAG_DIC["drink_MEA_FOOD"]          = "coffee"
TAG_DIC["fruit_FAV_FOOD"]          = "apple"
```

4.2.2 General Feedback

Referring [E] of Fig 4.2 and [J] of Fig. 4.3, indicates that the robot is capable of providing general comments to the user. Robot system would detect a Key Word from the user response and search the KDB for any information that can be shared with the user. KDB consists of such information as follows.

```
TAG_DIC["oliver twist_AUT_BOOK"]   = "charles dickens"
TAG_DIC["oliver twist_LAN_BOOK"]   = "english"
TAG_DIC["oliver twist_TYP_BOOK"]   = "novel"

TAG_DIC["apple_GOD_FOOD"]          = "health"
TAG_DIC["apple_BAD_FOOD"]          = "significantly nothing"
TAG_DIC["apple_NUT_FOOD"]          = "vitamin"
```

4.3 **User Reminders**

Robot system is capable of making reminders on preset event times. Fig. 4.4 presents an instance where the robot is making a reminder of dinner event. However the user can either act accordingly or make adjustments to the reminded time. Programme code represents the events in a data array as follows.

```
REMIND_DIC = [[7, 31, "breakfast", 1], [12, 31, "lunch", 1],
[17, 51, "dinner", 1], [8, 30, "morning medicine", 1], [13,
30, "after noon medicine", 1], [18, 30, "evening medicine",
1]]
```

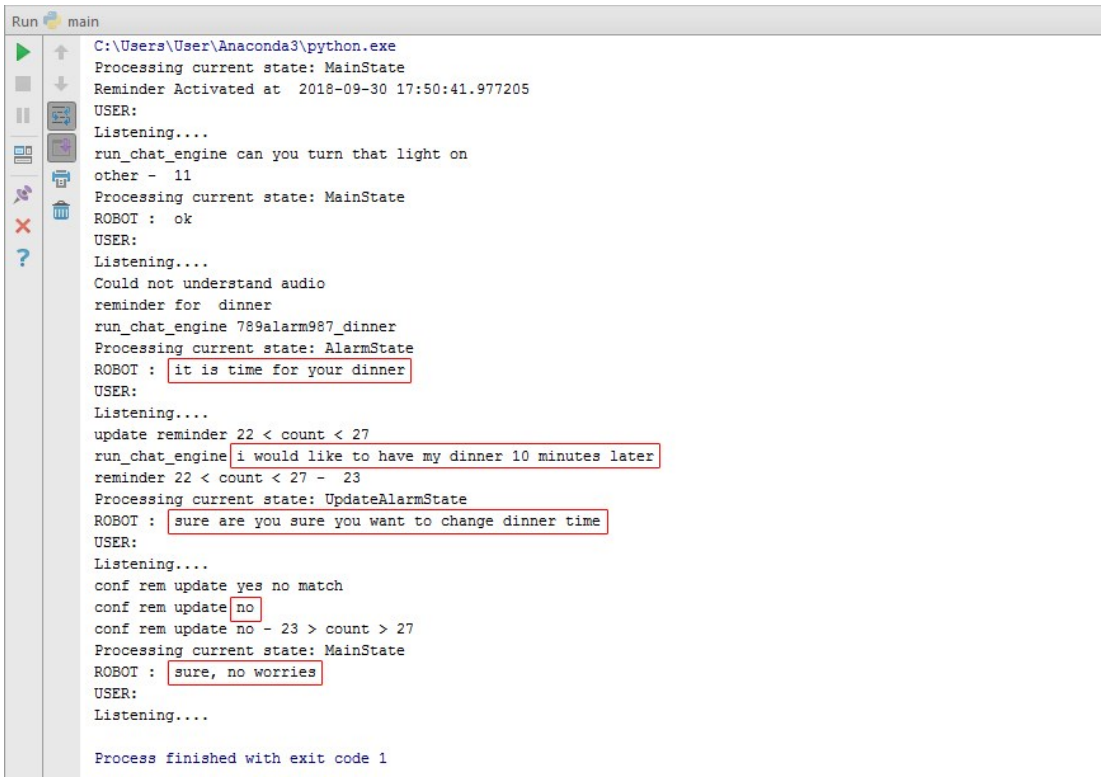


Fig. 4.4 : Terminal output during a reminder execution

CHAPTER 5: **THE EXPERIMENTAL SETUP AND RESULTS**

The proposed system has been implemented for supporting two types of services

1) A service requiring bringing an object
2) A service requiring reading some material

and three object categories

1) Food
2) Reading Material
3) Medicine

However, the implemented system is still capable of responding to other service types and object categories without involving in extended human to human like dialogue interactions. In addition, general interaction phrases such as greetings are understood and responded by the implemented system.

In between Google Speech Recognition API and Python TTS library, a literal text processing task is performed. The textural format of the recognized speech is identified against its context and fed into the FSIM for generating a suitable response. A regular expression language based architecture is proposed and it is implemented in Python programming environment for semantic identification. The proposed method is capable of identifying multiple language patterns with respect to a specific meaning. An example for the regular expression based recognition is given below in Augmented Backus-Naur Form (ABNF) syntax.

$command1 = $action1 $object;
$command2 = $action2 $object;
$action1 = bring;
$action2 = read;
$object = [the | a | some] (book | water | bread | medicine);

The $command1 represents several language patterns such as bring a book, bring the medicine, bring some water etc. similarly the $command2 represents language patterns pertaining to “read” action. This proposed method is implemented using regular expression syntax and a data structure with complex regular expressions and responses is referred in identifying user expressions. It is important to find correct and most suitable language patterns applicable in service-oriented tasks. Hence, a survey is conducted to identify language patterns practically applicable for human interactions in accomplishing service-oriented tasks. The participants were asked to fill out a form requesting basic information consisting gender, age and proficiency in English. Proficiency in English has been specially taken into validation through a 1-5 scale input. This scales enable the users to record their proficiency level varying from “Very Bad” - “Excellent” respectively. English not being the first language of most of the target participants, collected data would have a direct effect by the survey participants’ capacity of handling English language. Participants were asked to complete the form with dialogues that would take place in between a human caretaker and an elderly person pertaining to a various service oriented scenarios. Here, the participants were asked to imagine and respond as both the human caretaker and the elderly person having the conversation. This allows identifying human like language patterns applicable in both understanding and responding service oriented dialogues. Data has been collected from a population size of 20 varying in an age range of 25y – 55y, male population percentage of 61.11% and mean English proficiency rating of 3.65. It would have been better if more elderly community been participated for the survey to widen the age variation, but altogether this can be identified as a promising data set for identifying language patterns used in service oriented dialogues.

The proposed robotic design has been deployed with Moratuwa Intelligent Robot (MIRob) [25]. The robot is running Microsoft Windows based operating system with Python version 3.5. The inbuilt speaker system is accessed for voice output and voice input been captured through a wireless microphone attached to the user for minimizing the environmental noise mixing with the voice input.

5.1 **Moratuwa Intelligent Robot (MIRob)**

Moratuwa Intelligent Robot (MIRob) has been developed as a domestic service robot. The robot is made up of Pioneer 3DX mobile robot platform and Cyton Gamma 300 manipulator. The mobile robot platform is capable of moving with a maximum speed of 1.2 ms^{-1} and carrying a maximum weight of 17 kg. It is a differential drive mobile robot with 500-tick encoders. The robot is equipped with sonar sensor arrays, inbuilt gyroscope, a binocular vision system, front and rear facing bumpers, an aluminum structure to facilitate other equipment and necessary height requirement.

Fig. 5.1 : The system being tested with Moratuwa Intelligent Robot (MIRob)

An embedded PC with 2GB RAM and Intel Core 2 Duo 2.26 GHz processor has been integrated into the robot base for performing high level processing and controlling such as, image processing, voice recognition and understanding, tasks planning and decision making. A WLAN adapter capable of using IEEE 802.11a/b/g

is available for communication. In addition to that, the robot consists with stereo speakers and microphones to play sound and voice recognition.

5.2 **Experiments and Results**

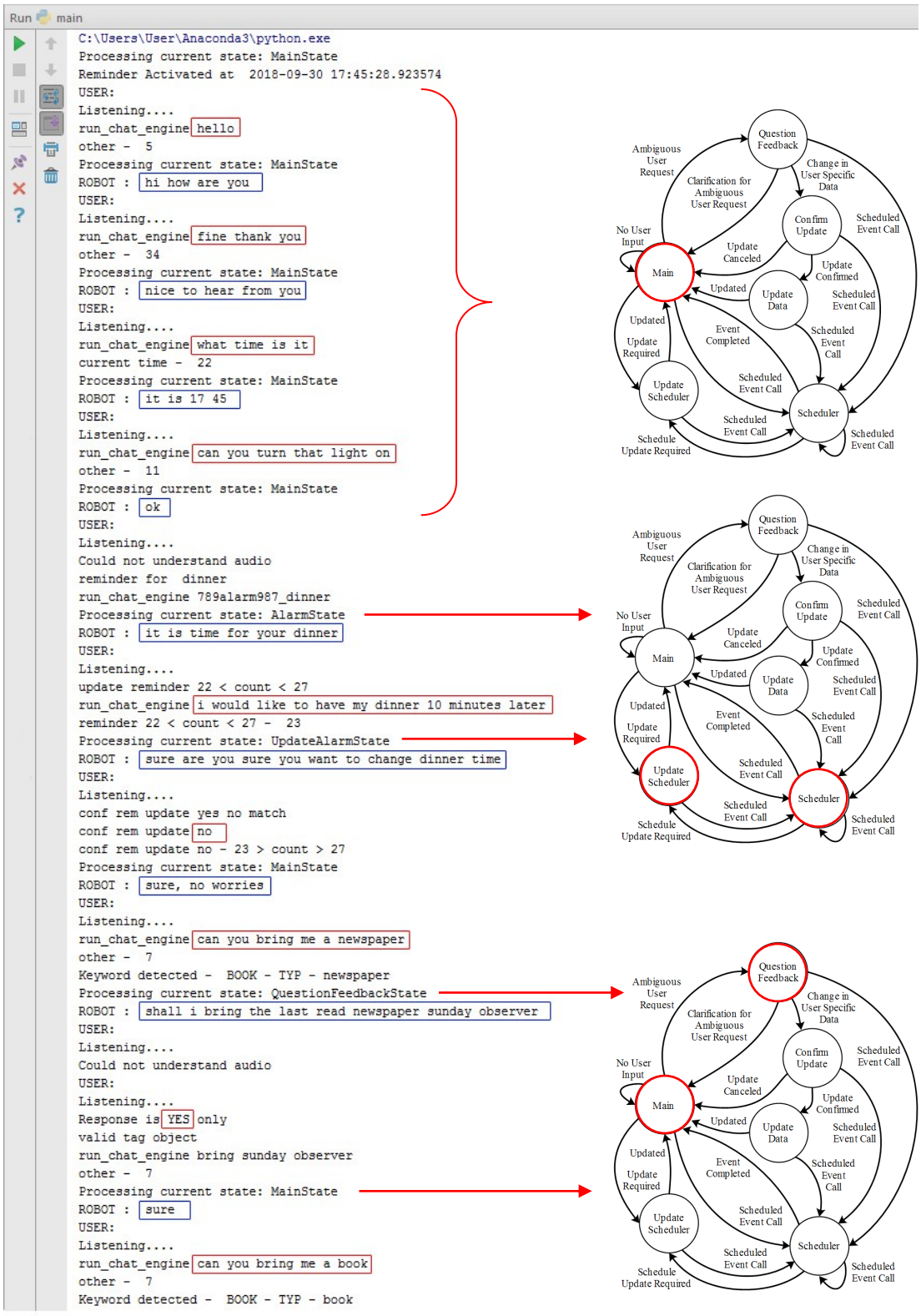
The experimental setup implemented on Moratuwa Intelligent Robot has been used for obtaining experimental results. As the experiment, a human user was asked to interact with the experimental setup and provide an overall rating in 1 – 5 scale for overall satisfaction. The 1 – 5 scales enable the user to express his or her user experience varying from "Very Bad" - "Excellent" respectively. The MIRob and a participant interacting with MIRob during the experiment is shown in Fig. 5.1. The user inputs and robot responses were recorded for analysis. The human user details consisting age, gender and proficiency in English has been recorded along with the rating. A selected conversation segment between the human user and the robot is given below. This dialogue segment shows various scenarios and language patterns handled by the robot at the experimental setup. The PyCharm terminal output is shown in Fig. 5.2. Terminal output indicates how the user input is read by the voice recognition engine and the responses are generated by the FSIM. Change of FSIM states through the dialogue segments is also illustrated in Table 5.1.

1) User : Hello
2) Robot : Hi how are you?
3) User : I am fine thank you
4) Robot : Nice to hear from you
5) User : What time is it?
6) Robot : It is 17 45
7) User : Can you turn that light on
8) Robot : Ok
9) Robot : It is time for your dinner
10) User : I would like to have my dinner 10 minutes later
11) Robot : Sure, Are you sure you want to change dinner time

12) User : No
13) Robot : Sure, no worries
14) User : Can you bring me a newspaper
15) Robot : Shall I bring the last read newspaper “Sunday Observer”?
16) User : Yes
17) Robot : Sure
18) User : Can you bring me a book?
19) Robot : What book should I bring?
20) User : Bring “Oliver Twist”
21) Robot : Wow it is a god selection. “Oliver Twist” is written by “Charles Dickens” and he is a wonderful author!
22) User : Can you bring me a book?
23) Robot : You read “Viragaya” again and again, shall I bring that book?
24) User : No
25) Robot : Is there a new most read book?
26) User : Yes
27) Robot : What is the new most read book?
28) User : “Gamperaliya”
29) Robot : Wow it is good to know your new most read book!
30) Robot : Shall I bring the last read book “Uganthaya”?
31) User : Yes
32) Robot : Wow it is a good selection! “Uganthaya” is written by “Martin Wickramasinghe” and he is a wonderful author!
33) User : Can you read me the book?
34) Robot : Give me a minute I will read

Initially the robot was at the “Main” state. First eight lines of the conversation have no impact for state change which consists of robot - user interactions with direct responses to service oriented requests. A scheduled event had occurred and the robot had stated “It is time for your dinner” in dialogue line number nine. Hence the state has changed to “Schedular”. At this point, the user must acknowledge the scheduled event. The user had acknowledged the scheduled event by requesting to postpone the

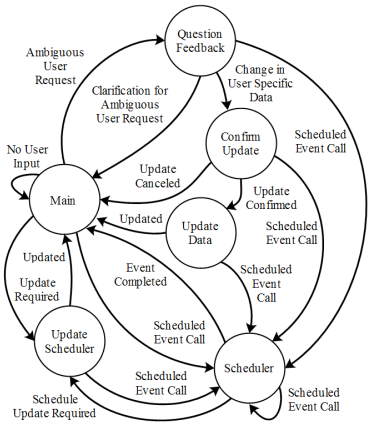
event by dialogue line number ten. Robot had requested the confirmation by dialogue line number 11 and had initiated the update process by changing the state to “Update Schedular”. At the end of the update process, the state had moved back to “Main” expecting a fresh user input. Robot has acknowledged the user input. Further, user had made another service request at the “Main” state by stating “Can you bring me a newspaper?” The request was not well specified and the robot was unable to decide on what book the user was referring. Hence, the robot had moved to “Question Feedback” state for getting clarifications from the user. With the users answer, the robot had commented on the user’s selection by referring to the KDB. It is common to add some comment or explain our own experiences during a conversation. This makes the conversation more interesting and natural for participants. Then the robot had again moved to “Question Feedback” state with the dialogue line number 19. Robot has again requested for a clarification. During the dialog line number 23, the state has again changed to “Question Feedback”. Here the robot had requested a clarification by referring to a specific user parameter in the KDB “Most Read Book”. With the user’s response in dialogue line number 24, the robot had identified a user intent for updating a user parameter. The state had moved to “Confirm Update” for obtaining an update confirmation. Robot had obtained the new value for the user parameter and update the KDB by moving to “Update” state. The sample dialogue has again moved to a “Question Feedback” at dialogue line umber 30 to confirm the user preference and then at dialogue line umber 32 to the “Main” state.


Run main
C:\Users\User\Anaconda3\python.exe
Processing current state: MainState
Reminder Activated at 2018-09-30 17:45:28.923574
USER:
Listening....
run_chat_engine hello
other - 5
Processing current state: MainState
ROBOT : hi how are you
USER:
Listening....
run_chat_engine fine thank you
other - 34
Processing current state: MainState
ROBOT : nice to hear from you
USER:
Listening....
run_chat_engine what time is it
current time - 22
Processing current state: MainState
ROBOT : it is 17 45
USER:
Listening....
run_chat_engine can you turn that light on
other - 11
Processing current state: MainState
ROBOT : ok
USER:
Listening....
Could not understand audio
reminder for dinner
run_chat_engine 789alarm987_dinner
Processing current state: AlarmState
ROBOT : it is time for your dinner
USER:
Listening....
update reminder 22 < count < 27
run_chat_engine i would like to have my dinner 10 minutes later
reminder 22 < count < 27 - 23
Processing current state: UpdateAlarmState
ROBOT : sure are you sure you want to change dinner time
USER:
Listening....
conf rem update yes no match
conf rem update no
conf rem update no - 23 > count > 27
Processing current state: MainState
ROBOT : sure, no worries
USER:
Listening....
run_chat_engine can you bring me a newspaper
other - 7
Keyword detected - BOOK - TYP - newspaper
Processing current state: QuestionFeedbackState
ROBOT : shall i bring the last read newspaper sunday observer
USER:
Listening....
Could not understand audio
USER:
Listening....
Response is YES only
valid tag object
run_chat_engine bring sunday observer
other - 7
Processing current state: MainState
ROBOT : sure
USER:
Listening....
run_chat_engine can you bring me a book
other - 7
Keyword detected - BOOK - TYP - book
Question Feedback
Ambiguous User Request
Change in User Specific Data
Clarification for Ambiguous User Request
Confirm Update
Scheduled Event Call
No User Input
Update Canceled
Update Confirmed
Main
Updated
Update Data
Scheduled Event Call
Updated
Update Required
Event Completed
Scheduled Event Call
Update Scheduler
Scheduled Event Call
Scheduler
Scheduled Event Call
Schedule Update Required
Scheduled Event Call

(a)

```
Processing current state: QuestionFeedbackState
ROBOT : what book should i bring
USER:
Listening....
Could not understand audio
USER:
Listening....
Could not understand audio
USER:
Listening....
keyword detected for mismatch
run_chat_engine bring oliver twist
other -  7
Keyword detected -  BOOK - ITM - oliver twist
wow it is a good selection oliver twist is written by charles dickens and he is a wonderful author
Processing current state: MainState
ROBOT : wow it is a good selection oliver twist is written by charles dickens and he is a wonderful author
USER:
Listening....
run_chat_engine can you bring me a book
other -  7
Keyword detected -  BOOK - TYP - book
Processing current state: QuestionFeedbackState
ROBOT : you read viragaya again and again shall i bring that book
USER:
Listening....
Could not understand audio
USER:
Listening....
Response is NO only
Update user data -  is there a new most read book
Processing current state: ConfirmUpdateUserDataState
ROBOT : is there a new most read book
USER:
Listening....
conf update yes
Processing current state: UpdateUserDataState
ROBOT :  What is the new most read book
USER:
Listening....
update -  book_MRD_BOOK
ROBOT : wow that is great to know your new most read book
run_chat_engine bring book
other -  7
Keyword detected -  BOOK - TYP - book
Processing current state: QuestionFeedbackState
ROBOT : shall i bring the last read book uganthaya
USER:
Listening....
Response is YES only
valid tag object
run_chat_engine bring uganthaya
other -  7
Keyword detected -  BOOK - ITM - uganthaya
wow it is a good selection uganthaya is written by martin wickramasinge and he is a wonderful author
Processing current state: MainState
ROBOT : wow it is a good selection uganthaya is written by martin wickramasinge and he is a wonderful author
USER:
Listening....
Could not understand audio
USER:
Listening....
run_chat_engine can you read the book
other -  19
Processing current state: MainState
ROBOT : give me a minute i will read
USER:
Listening....
Could not understand audio
USER:

Process finished with exit code 1
```



(b)

Fig. 5.2 : (a) and (b) Terminal output of the conversation indicating transfer of states

The setup has been experimented with a user population. All the participants volunteered for the evaluation task were asked to interact with the robotic system for a sufficient time and respond to a questioner containing following questions. The participant's feedback has been recorded. Prior to the user interaction; the users were beforehand informed regarding the limitations of the developed system.

Q1. What is your gender?
Q2. How old are you?
Q3. How would you rate your proficiency in English language?
Q4. How would you rate the overall performance of the Robotic Companion?

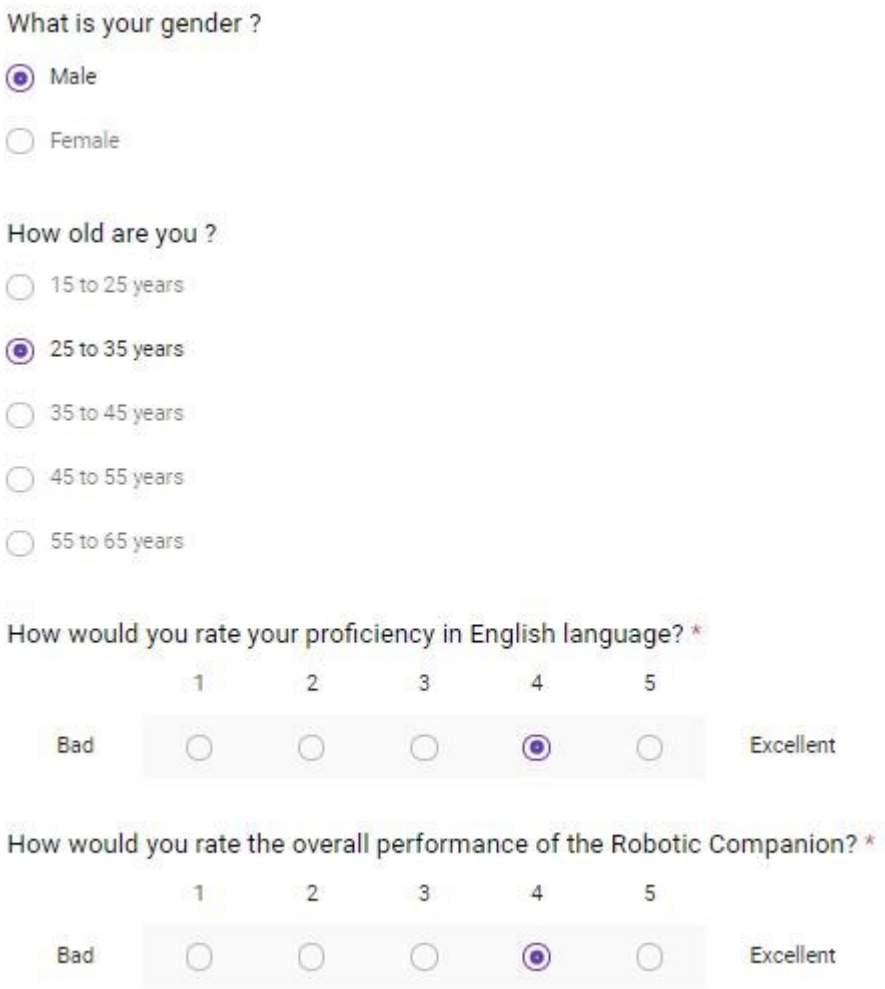


Fig. 5.3 : Graphical view of the evaluation form

A user population size of 24 varying in an age range of 24y – 55y, male population percentage of 66.67% and average English proficiency rating of 3.625 for a 1 – 5 scale has participated the evaluation. The statistics of the participated user population can be represented in graphical forma as depicted in Fig. 5.4 and 5.5. Evaluation results of the experiment (i.e., user rating) are illustrated in Fig.5.6 and Fig. 5.7.

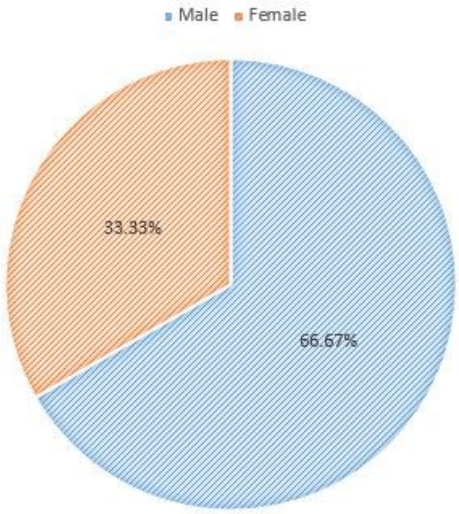


Fig. 5.4 : Gender variation of the population volunteered for the survey

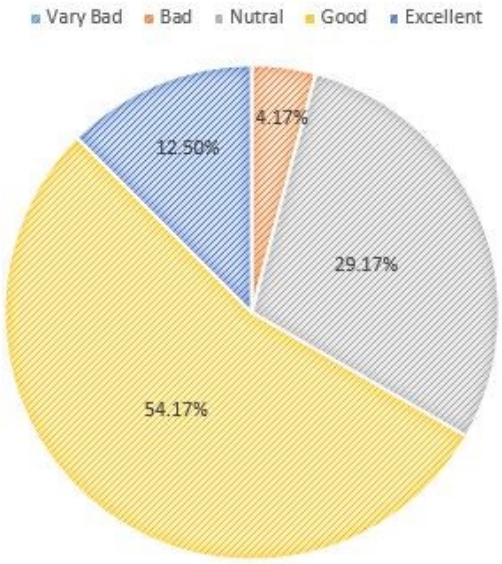


Fig. 5.5 : English proficiency variation of the population volunteered for the survey

The average user rating was 3.75 (σ = 0.72) and the median was 4.0. This implies that the overall user rating stand close to "Good". Considering the experimented implementation where the services are limited to only two types, the received average value lies in the positive side. Therefore this validates the proposed robots capability to successfully interact with humans and express human like dialogs during a service assistance scenario with high user satisfaction rate.

Table 5.1 : Change of state through the dialogue segment

Dialogue No.	**Corresponding FSIM state**
1	"Main"
2	
3	
4	
5	
6	
7	
8	
9	"Schedular"
10	
11	"Update Schedular"
12	
13	"Main"
14	
15	"Question Feedback"
16	
17	"Main"
18	
19	"Question Feedback"
20	

Dialogue No.	Corresponding FSIM state
21	"Main"
22	
23	"Question Feedback"
24	
25	"Confirm Update"
26	
27	"Update Data"
28	
29	
30	"Question Feedback"
31	
32	"Main"
33	
34	

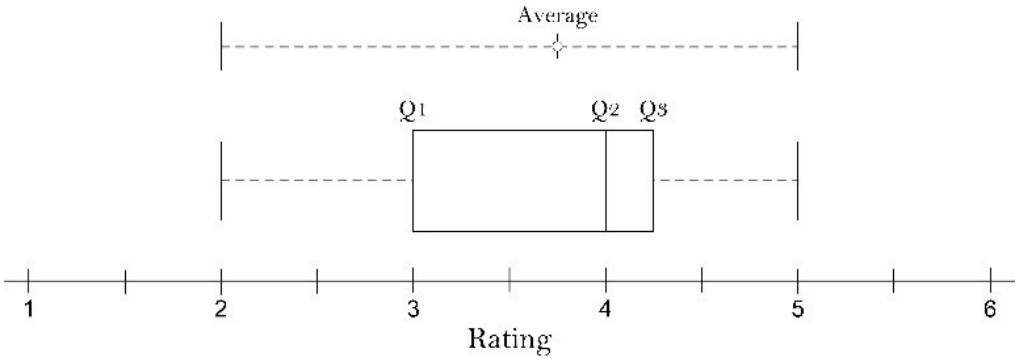


Fig. 5.6 : User evaluation results obtained during the experiment. The box plot has the standard notation.

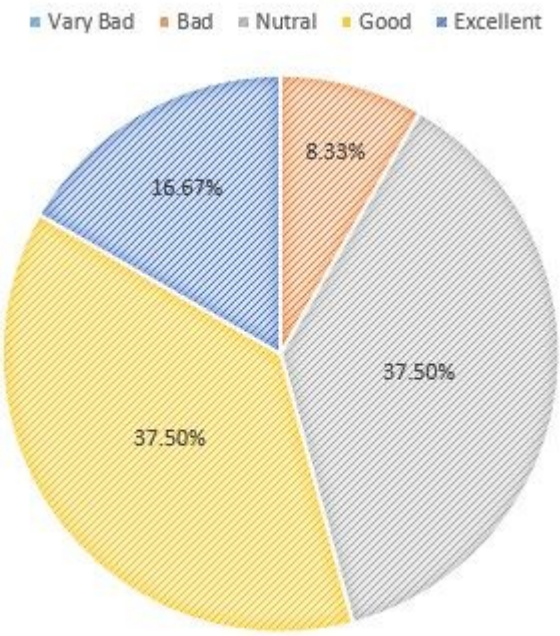


Fig. 5.7 : Rating given by the population volunteered for the survey

CHAPTER 6: **CONCLUSION AND FUTURE WORKS**

6.1 **Conclusion**

A Finite State Interaction Module (FSIM), Knowledge Database (KDB) and a regular expression based language identification method have been proposed for implementing the Interaction Management Module (IMM). The FSIM consist of number of finite states for handling different communication scenarios. Information available in the Knowledge Database (KDB) is accessed by the IMM for interacting with the user. Furthermore, the system is capable of updating the KDB as per user requirements. Python is the main platform used in developing this robot system. Further, Google Speech Recognition API has been used for voice to text conversion and Python Text to Speech library has been used for text to speech conversion. Regular expressions are used for identifying the voice inputs.

The systems has been tested with the experimental setup implemented on MIRob. The overall user rating over a 1 – 5 scale has been recorded form an experiment conducted using a sample user population in order to validate the performance of the developed system. The experiment shows very promising results indicating high user satisfaction rate, which validates the robots capability to interact in a human friendly manner during service assistance.

Hence it concludes that, the proposed architecture including FSIM, KDB and IMM is suitable for developing a service oriented conversational companion robot for elderly people. The designed and developed conversational companion robot is capable of interacting with elderly user with human like interactive dialogues for assisting them during service requests. Proposed method for validating the system is acceptable and it has validated the developed system in terms of the robots capability to interact in a human friendly manner during service assistance.

6.2 **Future Works**

The proposed system has been implemented with a Knowledge Database (KDB) only supporting two types of services and three object categories.

Supported Services

1. A service requiring bringing an object
2. A service requiring reading some material

Supported Object Categories

1. Food
2. Reading Material
3. Medicine for testing.

Hence, the user interaction scope would be limited. It is required to improve the KDB to support more service types and object categories.

The self-learning capability of the robot can be improved. Current capability addresses updating specific data in the KDB. The system can be improved to identify and learn new language patterns and corresponding responses as well. Further, it can be developed to update existing language patterns and corresponding responses with new variations.

References

[1] S. Bedaf et al., “Which activities threaten independent living of elderly when becoming problematic: inspiration for meaningful service robot functionality,” Disability and Rehabilitation: Assistive Technology, vol. 9, no. 6, pp. 445–452, Nov. 2014.

[2] Jennifer Piatt et al., “Companionship with a Robot? Therapists’ Perspectives on Socially Assistive Robots as Therapeutic Interventions in Community Mental Health for Older Adults,” American Journal of Recreation Therapy, vol. Vol 15, No 4, 2016.

[3] World Health Organization, Ed., World report on ageing and health. Geneva: WHO, 2015.

[4] S. Bedaf, G. J. Gelderblom, and L. de Witte, “Overview and Categorization of Robots Supporting Independent Living of Elderly People: What Activities Do They Support and How Far Have They Developed,” Assistive Technology, vol. 27, no. 2, pp. 88–100, Apr. 2015.

[5] T. Vandemeulebroucke, B. Dierckx de Casterlé, and C. Gastmans, “The use of care robots in aged care: A systematic review of argument-based ethics literature,” Archives of Gerontology and Geriatrics, vol. 74, pp. 15–25, Jan. 2018.

[6] R. Yu et al., “Use of a Therapeutic, Socially Assistive Pet Robot (PARO) in Improving Mood and Stimulating Social Interaction and Communication for People With Dementia: Study Protocol for a Randomized Controlled Trial,” JMIR Research Protocols, vol. 4, no. 2, p. e45, May 2015.

[7] R. M. Agrigoroaie and A. Tapus, “Developing a healthcare robot with personalized behaviors and social skills for the elderly,” in ACM/IEEE Int. Conf. on Human-Robot Interaction (HRI), pp. 589–590, Mar. 2016.

[8] T. Vandemeulebroucke, B. D. de Casterlé, and C. Gastmans, “How do older adults experience and perceive socially assistive robots in aged care: a systematic review of qualitative evidence,” Aging & Mental Health, vol. 22, no. 2, pp. 149–167, Feb. 2018.

[9] J. Hudson, M. Orviska, and J. Hunady, “People’s Attitudes to Robots in Caring for the Elderly,” Int. Journal of Social Robotics, vol. 9, no. 2, pp. 199–210, Apr. 2017.

[10] S. Bedaf, P. Marti, and L. De Witte, “What are the preferred characteristics of a service robot for the elderly? A multi-country focus group study with older adults and caregivers,” Assistive Technology, pp. 1–11, Nov. 2017.

[11] M. A. Goodrich and A. C. Schultz, “Human-Robot Interaction: A Survey,” Foundations and Trends® in Human-Computer Interaction, vol. 1, no. 3, pp. 203–275, 2007.

[12] T. Fong, I. Nourbakhsh, and K. Dautenhahn, “A survey of socially interactive robots,” Robotics and Autonomous Systems, vol. 42, no. 3–4, pp. 143–166, Mar. 2003.

[13] S. Huang, T. Tanioka, R. Locsin, M. Parker, and O. Masory, “Functions of a caring robot in nursing,” in Int. Conf. on Natural Language Processing andKnowledge Engineering, pp. 425–429, 2011.

[14] P. Menzel and F. D’Aluisio, Robo Sapiens: evolution of a new species. Cambridge (Mass): MIT, 2001.

[15] S.-E. Fotinea et al., “The MOBOT human-robot communication model,” in Int. Conf. on Cognitive Infocommunications, pp. 201–206, 2015.

[16] E. Efthimiou et al., “The MOBOT Platform – Showcasing Multimodality in Human-Assistive Robot Interaction,” in Universal Access in Human-Computer Interaction. Interaction Techniques and Environments, vol. 9738, M. Antona and C. Stephanidis, Eds. Cham: Springer International Publishing, 2016, pp. 382–391.

[17] E. Efthimiou et al., “The MOBOT rollator human-robot interaction model and user evaluation process,” in IEEE Symposium Series on Computational Intelligence, pp. 1–8, 2016.

[18] Y. Lin, H. Min, H. Zhou, and F. Pei, “A Human-Robot-Environment Interactive Reasoning Mechanism for Object Sorting Robot,” IEEE Transactions on Cognitive and Developmental Systems, pp. 1–1, 2017.

[19] C. Chakrabarti and G. F. Luger, “A semantic architecture for artificial conversations,” in Int. Conf. on Soft Computing and Intelligent Systems, pp. 21–26, 2012.

[20] I. A. Hameed, “Using natural language processing (NLP) for designing socially intelligent robots,” in IEEE Int. Conf. on Development and Learning and Epigenetic Robotics, pp. 268–269, 2016.

[21] J. Shruthi, S. Swamy, and C. G. Sarika, “Survey on - Socially Intelligent Robots by using NLP,” Inte. Journal of Computer Applications, vol. 171, no. 1, pp. 19–21, Aug. 2017.

[22] M. A. V. J. Muthugala, and A. G. B. P. Jayasekara, “A Review of Service Robots Coping With Uncertain Information in Natural Language Instructions,” IEEE Access, vol. 6, pp. 12913 - 12928, Feb. 2018.

[23] M. A. V. J. Muthugala and A. G. B. P. Jayasekara, “MIRob: An intelligent service robot that learns from interactive discussions while handling uncertain information in user instructions,” In Moratuwa Engineering Research Conf., 2016, pp. 397–402.

[24] M. A. V. J. Muthugala, and A. G. B. P. Jayasekara, “Enhancing User Satisfaction by Adapting Robot’s Perception of Uncertain Information Based on Environment and User Feedback,” IEEE Access, vol. 5, pp. 26435 - 26447, Nov. 2017.

[25] N. Mavridis, “A review of verbal and non-verbal human–robot interactive communication,” Robotics and Autonomous Systems, vol. 63, pp. 22–35, Jan. 2015.

Publications

[1] G. W. M. Manuhara, M. A. V. J. Muthugala, and A. G. B. P. Jayasekara, "Design and Development of an Interactive Service Robot as a Conversational Companion for Elderly People," in Moratuwa Engineering Research Conf., pp. 378-383, 2018.

Date of Conference: 30 May-1 June 2018
Date Added to IEEE Xplore: 31 July 2018
ISBN Information:
Electronic ISBN: 978-1-5386-4417-1
Print on Demand (PoD) ISBN: 978-1-5386-4418-8
INSPEC Accession Number: 17962776
DOI: 10.1109/MERCon.2018.8421933
Publisher: IEEE
Conference Location: Moratuwa, Sri Lanka

Appendix A: Python Programme Code

File Name : main.py

```
import speech_recognition as sr
import datetime
from chatbot import Chatbot
from chatOut import chat_speak
from functionalParameters import REMIND_DIC

r = sr.Recognizer()
CB = Chatbot()

t_now = datetime.datetime.now()
print("Reminder Activated at ", t_now)

while 1:

    # Check time for reminder

    t_now = datetime.datetime.now()

    for i in REMIND_DIC:
        if t_now.hour == i[0] and t_now.minute == i[1] and i[3] == 1:
            # current alarm is disabled for restricting one time
occurrence
            i[3] = 0
            print("reminder for ", i[2])
            chat_out = CB.run("789alarm987" + "_" + i[2])
            chat_speak(chat_out)

        elif t_now.hour == 0 and t_now.minute == 0:
            # enable all alarms at 12 midnight
            i[3] = 1


    # get audio from the microphone

    with sr.Microphone() as source:
        #r.adjust_for_ambient_noise(source)
        #r.energy_threshold = 3000
        print("USER:")
        audio = r.listen(source)

    try:
        print("Listening....")
        audio_text = r.recognize_google(audio)
        chat_out = CB.run(str(audio_text))
        chat_speak(chat_out)
    except sr.UnknownValueError:
        print("Could not understand audio")
    except sr.RequestError as e:
        print("Could not request results; {0}".format(e))
```

File Name : chatbot.py

```
"""
    ############################################################
    chatbot
"""

from generalFunctions import make_lower
from stateMachine import StateMachine


"""
    ################################################################
    initiate chatbot
"""


class Chatbot(object):

    SM = StateMachine()

    def run(self, chat_in):
        chat_out = self.SM.event(make_lower(chat_in))
        return chat_out
```

File Name : chatOut.py

```
import pyttsx3

def chat_speak(chat_out):

    print("ROBOT : ", chat_out)

    engine = pyttsx3.init()
    rate = engine.getProperty('rate')
    engine.setProperty('rate', rate - 50)
    volume = engine.getProperty('volume')
    engine.setProperty('volume', volume - 0.25)
    voices = engine.getProperty('voices')
    engine.setProperty('voice', voices[0].id)
    engine.say(chat_out)
    engine.runAndWait()
```

File Name : generalFunctions.py

```
"""
    ############################################################
    Common functions required by other modules
"""

import re
from flask import flash
from regexPairs import regex_pairs
from regexPairs import keyword_regex_pairs

regex_pairs = [(re.compile(x, re.IGNORECASE), y) for (x, y) in
regex_pairs]
keyword_regex_pairs = [(re.compile(x, re.IGNORECASE), y) for (x, y) in
keyword_regex_pairs]


"""
    ################################################################
    Classify chat input
"""


def classifier(chat_in):
    count = 0
    for (pattern, response) in regex_pairs:
        match = pattern.match(chat_in)
        count += 1
        if match:
            return match, response, count

    return "no-match-available", "no-response-available", "no-count-
available"


"""
    ################################################################
    Detect key words
"""


def keyword_detector(match_group_1):
    for (pattern, response) in keyword_regex_pairs:
        match = pattern.match(match_group_1)
        if match:
            # Assign the only response as keyword_type_category
            keyword_type_category = response
            keyword_type, keyword_category =
keyword_type_category.split("_")
            return match.group(1), keyword_type, keyword_category
    return "no-key-word-available", "NULL", "NULL"


"""
    ################################################################
    Condition regular expression respond
        Replace wild cards with block of user input
"""


def set_wildcard_strings(response, match):
```

```
    # find the position of the first regex replacement pattern
    pos = response.find('%')
    # check for multiple patterns
    while pos >= 0:
        # convert position of first regex replacement pattern to numerical
value
        num = int(response[pos + 1:pos + 2])
        # find the substitution from the match and store
        regex_obj_str = match.group(num)
        response = response[:pos] + regex_obj_str + response[pos + 2:]
        # find the position of the next regex replacement pattern
        pos = response.find('%')
    return response


"""
    ################################################################
    Convert chat input to lower case
"""


def make_lower(chat_in):
    chat_in_lower = chat_in.lower()
    return chat_in_lower


"""
    ################################################################
    Tokenize chat input
"""


def tokenize(chat_in):
    chat_in_tokenize = chat_in.split(" ")
    return chat_in_tokenize
```

File Name : stateMachine.py

```
"""
    ################################################################
    We define the state machine.
"""

from states import MainState

"""
    ################################################################
    A simple state machine
"""


class StateMachine(object):

    next_state_data = "null-action", "null-object", "null-tag"

    def __init__(self):
        """
        Initialize the components.
        """
```

```
        # Start with a default state.
        self.state = MainState()

    def event(self, event):
        """
        Incoming chats are delegated to the given states which then handle
the event. The result is
        then assigned as the new state. resulting chat out is returned.
        """

        # The next state will be the result of the on_event function.
        next_state, self.next_state_data = self.state.event(event,
self.next_state_data)
        chat_out = self.state.chat_out
        self.state = next_state
        return chat_out
```

File Name : regexPairs.py

```
"""
    ############################################################
    Regular expressions response pairs matching language patterns and
responses
"""


"""
    ###################################################################
    Identifying keywords for initiating extended chat
"""

keyword_regex_pairs = (
    (r'(gamperaliya|oliver twist|harry potter|madol doova|uganthaya)',
     "BOOK_ITM"),

    (r'(book|story book|novel|short story|magazine|newspaper)',
     "BOOK_TYP"),

    (r'(apple|tea|bread|water|coke|chocolate|ice cream)',
     "FOOD_ITM"),

    (r'(food|drink|sweet|fruit|candy|desert|fruit)',
     "FOOD_TYP"),

    (r'(paracitamol|piriton|zitrecine|metfomin)',
     "MEDI_ITM"),

    (r'(medicine|drugs)',
     "MEDI_TYP")

)


"""
    ###################################################################
    Extended chat feedback responses
"""

feedback_regex_pairs = (
```

```
    (r'BRING_FOOD_ITM_(.*)',
     ("it is a really good choice %1 contains <NUT_FOOD> it is good for
you to have them",
      "%1 is good <GOD_FOOD> you should have them",
      "%1 is not good <BAD_FOOD> you should not have them")),

    (r'BRING_BOOK_ITM_(.*)',
     ("wow it is a good selection %1 is written by <AUT_BOOK> and he is a
wonderful author",
      "%1 is a <LAN_BOOK> <TYP_BOOK> by <AUT_BOOK> any way it is a good
choice of read")),

    (r'BRING_MEDI_ITM_(.*)',
     ("it seems you are not feeling well today",
      "%1 will make you feel better",
      "you better rest and take care of your self")),

    (r'READ_BOOK_ITM_(.*)',
     ("wow it is a good selection %1 is written by <AUT_BOOK> and he is a
wonderful author",
      "%1 is a good <LAN_BOOK> <TYP_BOOK> by <AUT_BOOK> any way it is a
nice choice of read"))

)


"""
    ##################################################################
    Extended chat question responses
"""

question_regex_pairs = (

    (r'BRING_FOOD_TYP_(.*)',
     ("shall i bring your favorite %1 <FAV_FOOD> ",
      "you regularly eat <MEA_FOOD> shall i bring some",
      "what %1 should i bring",
      "sure it is good to have <FAV_FOOD> now"
      "sure it is good to have <MEA_FOOD> now"
      "shall i bring <MEA_FOOD>")),


    (r'BRING_BOOK_TYP_(.*)',
     ("what %1 should i bring",
      "shall i bring your favorite %1 <FAV_BOOK>",
      "you read <MRD_BOOK> again and again shall i bring that %1",
      "shall i bring the last read %1 <LRD_BOOK>")),

    (r'BRING_MEDI_TYP_(.*)',
     ("what medicine should i bring",
      "how are you feeling now shall i bring all the medicine",
      "are you getting better now what medicine shall i bring ")),

    (r'READ_BOOK_TYP_(.*)',
     ("what %1 should i read",
      "shall i read <FAV_BOOK>",
      "you like <MRD_BOOK> shall i read that %1",
      "shall i read <LRD_BOOK> the last read %1"))

)


"""
    ##################################################################
```

```
    Chat inputs and responses
"""

regex_pairs = (
    # 1
    (r'(?:good morning)',
     ("good morning", "good morning to you")),
    # 2
    (r'(?:good afternoon)',
     ("good afternoon", "good afternoon to you")),
    # 3
    (r'(?:good evening)',
     ("good evening", "good evening to you")),
    # 4
    (r'(?:good night)',
     ("good night", "good night to you")),
    # 5
    (r'(?:hi|hey|hello|how are you|hello how are you|hi how are you|hey
how are you)',
     ("hello", "hi", "hey", "hello how are you", "hi how are you", "hey
how are you")),
    # 6
    (r'(?:bye|goodbye|cheerio)',
     ("bye", "goodbye", "cheerio", "take care", "bye, take care", "see you
later", "catch you later")),
    # 7
    (r'(?:can you |could you |)(?:please |)bring (?:me |)(?:a |an |the
|this |that |some |my |)(.*)',
     ("ok", "sure", "sure i will", "just give me a minute, i will bring",
"ok give me a minute", "sure give me a minute",
      "give me a minute i will bring", "i will be right back")),
    # 8
    (r'(?:can you |could you |)(?:please |)(turn|switch) off (a |an |the
|that |this |)(.*)',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute", "give me a minute i will %1 off %2%3",
      "just give me a minute, i will turn off")),
    # 9
    (r'(?:can you |could you |)(?:please |)(?:turn|switch) on (a |an |the
|that |this |)(.*)',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute", "give me a minute i will %1 on %2%3",
      "just give me a minute, i will turn on")),
    # 10
    (r'(?:can you |could you |)(?:please |)(turn|switch) (a |an |the |that
|this |)(.*) off',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute", "give me a minute i will %1 off %2%3",
      "just give me a minute, i will turn off")),
    # 11
    (r'(?:can you |could you |)(?:please |)(?:turn|switch) (a |an |the
|that |this |)(.*) on',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute", "give me a minute i will %1 on %2%3",
      "just give me a minute, i will turn on")),
    # 12
    (r'(?:can you |could you |)(?:please |)move (?:the|this|that) (.*)',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute", "give me a minute i will move",
      "just give me a minute, i will move")),
    # 13
    (r'(?:can you |could you |)(?:please |)call (?:the |this |that
|)(.*)',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
```

```
minute", "give me a minute i will call",
      "just give me a minute, i will call")),
    # 14
    (r'(?:can you |could you |)(?:please |)(take|walk) me to
(?:a|an|the|this|that) (.*)',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute", "give me a minute i will %1 you",
      "just give me a minute, i will %1 you")),
    # 15
    (r'(?:can you |could you |)(?:please |)wash (?:a|an|the|this|that)
(.*)',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute", "give me a minute i will wash",
      "just give me a minute, i will wash")),
    # 16
    (r'(?:can you |could you |)(?:please |)clean (?:a|an|the|this|that)
(.*)',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute", "give me a minute i will clean",
      "just give me a minute, i will clean")),
    # 17
    (r'(?:can you |could you |)(?:please |)brush my (.*)',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute", "give me a minute i will brush",
      "just give me a minute, i will brush")),
    # 18
    (r'(?:can you |could you |)(?:please |)comb (?:the |this |that |my
|)(.*)',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute", "give me a minute i will comb",
      "just give me a minute, i will comb")),
    # 19
    (r'(?:can you |could you |)(?:please |)read (?:me
|)(?:a|an|the|this|that|)(.*)',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute", "give me a minute i will read",
      "just give me a minute, i will read")),
    # 20
    (r'(?:can you |could you |)(?:please |)rub (?:the |this |that |some
|)(.*) on (?:the |this |that |my |)(.*)',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute", "give me a minute i will rub",
      "just give me a minute, i will rub")),
    # 21
    (r'(?:can you |could you |)(?:please |)help (?:me |)to (.*)',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute")),
    # 22
    (r'(?:can you |could you |what is |what |do you know |)(?:please
|)(?:tell |)(?:me |)(?:the |)time(?: now| is it| is it now|)',
     ("ok", "sure")),
    # 23
    (r'i would like to (?:have|take)(?: my|) ([a-z]+) ([0-9]{1,2})
(minutes|hours) later(?: every day|)',
     ("ok are you sure you want to change %1 time", "sure are you sure you
want to change %1 time")),
    # 24
    (r'i would like to (?:have|take)(?: my|) ([a-z]+) (?:at |by |)([0-
9]{1,2}) ([0-9]{1,2})(?: every day|)',
     ("ok are you sure you want to change %1 time", "sure are you sure you
want to change %1 time")),
    # 25
    (r'i would like to (?:have|take)(?: my|) ([a-z ]*medicine) ([0-
9]{1,2}) (minutes|hours) later(?: every day|)',
```

```
     ("ok are you sure you want to change %1 time", "sure are you sure you
want to change %1 time")),
    # 26
    (r'i would like to (?:have|take)(?: my|) ([a-z ]*medicine) (?:at |by
|)([0-9]{1,2}) ([0-9]{1,2})(?: every day|)',
     ("ok are you sure you want to change %1 time", "sure are you sure you
want to change %1 time")),
    # 27
    (r'i would like to (?:have|take) my ([a-z]+)(?: now|)',
     ("ok", "sure", "i will bring your %1", "sure i will bring your %1")),
    # 28
    (r'(?:how are you|how do you do)',
     ("i am fine", "i am fine thank you", "i am doing great", "i am doing
fine")),
    # 29
    (r'(i would like to |lets )(have |take )(?:a |an |the |this |that
|some |my |)(.*)',
     ("ok", "sure", "sure i will", "just give me a minute, i will bring",
"ok give me a minute", "sure give me a minute",
      "give me a minute i will bring", "i will be right back")),
    # 30
    (r'(?:can you |could you |what is |do you know |what |)(?:please
|)(?:tell |)(?:me |)(?:the |todays |)date(?: today|is today|)',
     ("ok", "sure")),
    # 31
    (r'(please|ok|okay|sure|of course|certainly)',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute")),
    # 32
    (r'(?:can you |could you |)(?:please |)charge (a |an |the |that |this
|my |)(.*)',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute", "give me a minute i will charge %2%3",
      "just give me a minute, i will charge")),
    # 33
    (r'(?:can you |could you |)(?:please |)give (?:me |)(?:a |an |the
|this |that |some |my |)(.*)',
     ("ok", "sure", "sure i will", "ok give me a minute", "sure give me a
minute"))
)
```

File Name : states.py

```
"""
    ############################################################
    We define states based on the state object with unique functionality
"""

from stateObject import State
from chatEngine import run_chat_engine
from questionFeedbackEngine import run_question_feedback_engine
from confirmUpdateUserDataEngine import
run_confirm_update_user_data_engine
from updateUserDataEngine import run_update_user_data_engine
from alarmEngine import run_alarm_engine
from updateAlarmEngine import run_update_alarm_engine
"""
    ##############################################################
    Main state contain all the basic chat functionality.
"""
```

```
class MainState(State):

    chat_out = "This is the default chat out for MainState"

    def event(self, chat_in, next_state_data):
        self.chat_out, next_sate_id, next_state_data =
run_chat_engine(chat_in, next_state_data)
        if next_sate_id == 0:
            return MainState(), next_state_data
        elif next_sate_id == 1:
            return QuestionFeedbackState(), next_state_data
        elif next_sate_id == 2:
            return ConfirmUpdateUserDataState(), next_state_data
        elif next_sate_id == 3:
            return UpdateUserDataState(), next_state_data
        elif next_sate_id == 4:
            return AlarmState(), next_state_data
        elif next_sate_id == 5:
            return UpdateAlarmState(), next_state_data
        else:
            return self


"""
    ################################################################
    Requiring a feedback for a question
"""


class QuestionFeedbackState(State):

    chat_out = "This is the default chat out for QuestionFeedbackState"

    def event(self, chat_in, next_state_data):
        self.chat_out, next_sate_id, next_state_data =
run_question_feedback_engine(chat_in, next_state_data)
        if next_sate_id == 0:
            return MainState(), next_state_data
        elif next_sate_id == 1:
            return QuestionFeedbackState(), next_state_data
        elif next_sate_id == 2:
            return ConfirmUpdateUserDataState(), next_state_data
        elif next_sate_id == 3:
            return UpdateUserDataState(), next_state_data
        elif next_sate_id == 4:
            return AlarmState(), next_state_data
        elif next_sate_id == 5:
            return UpdateAlarmState(), next_state_data
        else:
            return self


"""
    ################################################################
    Requiring to confirm to initiate the update
"""


class ConfirmUpdateUserDataState(State):

    chat_out = "This is the default chat out for
ConfirmUpdateUserDataState"
```

```
    def event(self, chat_in, next_state_data):
        self.chat_out, next_sate_id, next_state_data =
run_confirm_update_user_data_engine(chat_in, next_state_data)
        if next_sate_id == 0:
            return MainState(), next_state_data
        elif next_sate_id == 1:
            return QuestionFeedbackState(), next_state_data
        elif next_sate_id == 2:
            return ConfirmUpdateUserDataState(), next_state_data
        elif next_sate_id == 3:
            return UpdateUserDataState(), next_state_data
        elif next_sate_id == 4:
            return AlarmState(), next_state_data
        elif next_sate_id == 5:
            return UpdateAlarmState(), next_state_data
        else:
            return self


"""
    ################################################################
    Update user data
"""


class UpdateUserDataState(State):

    chat_out = "This is the default chat out for UpdateUserDataState"

    def event(self, chat_in, next_state_data):
        self.chat_out, next_sate_id, next_state_data =
run_update_user_data_engine(chat_in, next_state_data)
        if next_sate_id == 0:
            return MainState(), next_state_data
        elif next_sate_id == 1:
            return QuestionFeedbackState(), next_state_data
        elif next_sate_id == 2:
            return ConfirmUpdateUserDataState(), next_state_data
        elif next_sate_id == 3:
            return UpdateUserDataState(), next_state_data
        elif next_sate_id == 4:
            return AlarmState(), next_state_data
        elif next_sate_id == 5:
            return UpdateAlarmState(), next_state_data
        else:
            return self


"""
    ################################################################
    Alarm
"""


class AlarmState(State):

    chat_out = "This is the default chat out for Alarm state"

    def event(self, chat_in, next_state_data):
        self.chat_out, next_sate_id, next_state_data =
run_alarm_engine(chat_in, next_state_data)
        if next_sate_id == 0:
            return MainState(), next_state_data
```

```
        elif next_sate_id == 1:
            return QuestionFeedbackState(), next_state_data
        elif next_sate_id == 2:
            return ConfirmUpdateUserDataState(), next_state_data
        elif next_sate_id == 3:
            return UpdateUserDataState(), next_state_data
        elif next_sate_id == 4:
            return AlarmState(), next_state_data
        elif next_sate_id == 5:
            return UpdateAlarmState(), next_state_data
        else:
            return self


"""
    ################################################################
    Update Alarm
"""


class UpdateAlarmState(State):

    chat_out = "This is the default chat out for UpdateAlarmState"

    def event(self, chat_in, next_state_data):
        self.chat_out, next_sate_id, next_state_data =
run_update_alarm_engine(chat_in, next_state_data)
        if next_sate_id == 0:
            return MainState(), next_state_data
        elif next_sate_id == 1:
            return QuestionFeedbackState(), next_state_data
        elif next_sate_id == 2:
            return ConfirmUpdateUserDataState(), next_state_data
        elif next_sate_id == 3:
            return UpdateUserDataState(), next_state_data
        elif next_sate_id == 4:
            return AlarmState(), next_state_data
        elif next_sate_id == 5:
            return UpdateAlarmState(), next_state_data
        else:
            return self
```

File Name : stateObject.py

```
"""
    ################################################################
    Define a state object with general utility functions for the
    individual states.
"""


class State(object):

    chat_out = "This is the default chat out for state object"

    def __init__(self):
        print('Processing current state:', str(self))

    def event(self, read_in):
        """
```

```
        Handle events that are delegated to this State.
        """
        pass

    def __repr__(self):
        """
        Leverages the __str__ method to describe the State.
        """
        return self.__str__()

    def __str__(self):
        """
        Returns the name of the State.
        """
        return self.__class__.__name__
```

File Name : chatEngine.py

```
"""
    ##############################################################
    Functionality supporting the MainState
"""

import random
import datetime
from functionalParameters import *
from generalFunctions import classifier
from generalFunctions import set_wildcard_strings
from generalFunctions import keyword_detector
from actionClassifier import action_classifier
from unrecognizedChat import reply_unrecognized_chat


"""
    ##################################################################
    Chat Engine
"""


def run_chat_engine(chat_in, current_state_data):

    print("run_chat_engine", chat_in)

    next_state_data = "null-action", "null-object", "null-tag", "null-tag-
object"

    match, response, count = classifier(chat_in)
    # No matching regular expression found for the chat input
    if match == "no-match-available" and response == "no-response-
available":
        # Lookout for alarms
        alarm_tag = chat_in.split("_")
        if alarm_tag[0].strip() == "789alarm987":
            alarm_event = str(alarm_tag[1].strip())
            next_state_id = 4
            next_state_data = "null-action", alarm_event, "null-tag",
"null-tag-object"
            chat_out = "it is time for your " + alarm_event
            return chat_out, next_state_id, next_state_data
        else:
```

```
            print("unrecognized")
            next_state_id = 0
            chat_out = reply_unrecognized_chat()
            return chat_out, next_state_id, next_state_data
    else:
        if 22 < count < 27:
            print("reminder 22 < count < 27 - ", count)
            next_state_id = 5
            next_state_data = match, "null-object", count, "null-tag-
object"
            selected_response = random.choice(response)
            chat_out = set_wildcard_strings(selected_response, match)
            return chat_out, next_state_id, next_state_data
        elif count == 22:
            print("current time - ", count)
            next_state_id = 0
            t_now = datetime.datetime.now()
            if 3 < t_now.hour < 5:
                chat_out = "it is " + str(t_now.hour) + " " +
str(t_now.minute) + " too early in the morning"
            elif 5 < t_now.hour < 7:
                chat_out = "it is " + str(t_now.hour) + " " +
str(t_now.minute) + " good day to you"
            elif 11 < t_now.hour < 12:
                chat_out = "it is " + str(t_now.hour) + " " +
str(t_now.minute) + " almost noon"
            elif 21 < t_now.hour < 23:
                chat_out = "it is " + str(t_now.hour) + " " +
str(t_now.minute) + " time to sleep now"
            elif 23 < t_now.hour < 24:
                chat_out = "it is " + str(t_now.hour) + " " +
str(t_now.minute) + " in the midnight"
            else:
                chat_out = "it is " + str(t_now.hour) + " " +
str(t_now.minute)
            return chat_out, next_state_id, next_state_data
        elif count == 30:
            print("current date - ", count)
            next_state_id = 0
            t_now = datetime.datetime.now()
            chat_out = "it is " + str(t_now.strftime("%A")) + " " +
str(t_now.day) + " " + str(t_now.strftime("%B")) + " " +
str(t_now.strftime("%Y"))
            return chat_out, next_state_id, next_state_data
        else:
            selected_response = random.choice(response)
            print("other - ", count)
            # Random check for initiating a extended chat
            # Do not initiate an extended chat
            if random.randrange(0, 9) > RANDOM_CHECK:
                chat_out = set_wildcard_strings(selected_response, match)
                next_state_id = 0
                return chat_out, next_state_id, next_state_data
            # Initiate an extended chat
            else:
                # Check if known keywords are found
                try:
                    keyword, keyword_type, keyword_category =
keyword_detector(match.group(1))
                except IndexError:
                    keyword = "no-key-word-available"
                # No known key word found
                if keyword == "no-key-word-available":
                    chat_out = set_wildcard_strings(selected_response,
```

```
match)
                    next_state_id = 0
                    return chat_out, next_state_id, next_state_data
                # Known key word found
                else:
                    print("Keyword detected - ", keyword_type, "-",
keyword_category, "-", keyword)
                    action_classifier_phrase = ACTION_DIC[count] + "_" +
keyword_type + "_" + keyword_category + "_" + keyword
                    # Classify and execute chat respond based on feedback
or questing
                    chat_out, next_state_id, next_state_data =
action_classifier(action_classifier_phrase)
                    return chat_out, next_state_id, next_state_data
```

File Name : questionFeedbackEngine.py

```
"""
    ############################################################
    Provide the response for the reply given for question asked by the
    robot.
    This function is executed in QuestionFeedbackState
"""

import re
import random
from functionalParameters import RANDOM_CHECK
from functionalParameters import TAG_EXPANDER
from chatEngine import run_chat_engine
from generalFunctions import keyword_detector
from generalFunctions import make_lower
from functionalParameters import YES_TAG
from functionalParameters import NO_TAG


def run_question_feedback_engine(chat_in, current_state_data):

    next_state_data = "null-action", "null-object", "null-tag", "null-tag-
object"
    yes_no_pattern = re.compile(r'(yes|sure|ok|okay|no|nope|dont|do not|)(
.*|)')
    match = yes_no_pattern.match(chat_in)

    alarm_tag = chat_in.split("_")
    if alarm_tag[0].strip() == "789alarm987":
        alarm_event = str(alarm_tag[1].strip())
        next_state_id = 4
        next_state_data = "null-action", alarm_event, "null-tag", "null-
tag-object"
        chat_out = "it is time for your " + alarm_event
        return chat_out, next_state_id, next_state_data
    else:
        if match:
            # Response is YES/NO only
            if match.group(2) == "":
                # Response is YES only
                if match.group(1) in YES_TAG:
                    print("Response is YES only")
                    if current_state_data[3] == "null-tag-object":
                        print("null-tag-object")
```

```
                        chat_in_phrase =
str(make_lower(current_state_data[0])) + " " + str(current_state_data[1])
                        chat_out, next_state_id, next_state_data =
run_chat_engine(chat_in_phrase, next_state_data)
                        return chat_out, next_state_id, next_state_data
                    else:
                        print("valid tag object")
                        chat_in_phrase =
str(make_lower(current_state_data[0])) + " " + str(current_state_data[3])
                        chat_out, next_state_id, next_state_data =
run_chat_engine(chat_in_phrase, next_state_data)
                        return chat_out, next_state_id, next_state_data
                # Response is NO only
                elif match.group(1) in NO_TAG:
                    print("Response is NO only")
                    # Random check for updating user data or not
                    # Update user data
                    if random.randrange(0, 9) < RANDOM_CHECK and
current_state_data[3] != "null-tag-object":
                        update_request = "is there a new " +
str(TAG_EXPANDER[current_state_data[2]])
                        print("Update user data - ", update_request)
                        chat_out = update_request
                        next_state_id = 2
                        return chat_out, next_state_id, current_state_data
                    # Not update user data
                    else:
                        print("Not update user data")
                        chat_in_phrase =
str(make_lower(current_state_data[0])) + " " + str(current_state_data[1])
                        chat_out, next_state_id, next_state_data =
run_chat_engine(chat_in_phrase, next_state_data)
                        return chat_out, next_state_id, next_state_data
                else:
                    return pattern_mismatched(chat_in, current_state_data)
            # Response is YES/NO ++++
            else:
                # Response is YES ++++
                if match.group(1) in YES_TAG:
                    print("Response is YES ++++")
                    chat_out, next_state_id, next_state_data =
run_chat_engine(match.group(2).strip(), next_state_data)
                    return chat_out, next_state_id, next_state_data
                # Response is NO ++++
                elif match.group(1) in NO_TAG:
                    print("Response is NO ++++")
                    chat_out, next_state_id, next_state_data =
run_chat_engine(match.group(2).strip(), next_state_data)
                    return chat_out, next_state_id, next_state_data
                else:
                    return pattern_mismatched(chat_in, current_state_data)
        else:
            print("No match found")
            return pattern_mismatched(chat_in, current_state_data)

"""
    ################################################################
    If pattern mismatched
"""


def pattern_mismatched(chat_in, current_state_data):

    next_state_data = "null-action", "null-object", "null-tag"
```

```
    if keyword_detector(chat_in):
        # Create response tag
        print("keyword detected for mismatch")
        chat_in_phrase = str(make_lower(current_state_data[0])) + " " +
str(chat_in)
        chat_out, next_state_id, next_state_data =
run_chat_engine(chat_in_phrase, next_state_data)
        return chat_out, next_state_id, next_state_data
    else:
        print("no keyword detected for mismatch")
        chat_out, next_state_id, next_state_data =
run_chat_engine(chat_in, next_state_data)
        return chat_out, next_state_id, next_state_data
```

File Name : confirmUpdateUserDataEngine.py

```
"""
    ############################################################
    Confirm Update user data
"""

import re
from functionalParameters import YES_TAG
from functionalParameters import NO_TAG
from functionalParameters import TAG_EXPANDER
from chatEngine import run_chat_engine
from generalFunctions import make_lower
from chatOut import chat_speak


def run_confirm_update_user_data_engine(chat_in, current_state_data):

    next_state_data = "null-action", "null-object", "null-tag", "null-tag-
object"
    yes_no_pattern = re.compile(r'(yes|sure|ok|okay|no|nope|dont|do not|)(
.*|)')
    match = yes_no_pattern.match(chat_in)

    alarm_tag = chat_in.split("_")
    if alarm_tag[0].strip() == "789alarm987":
        alarm_event = str(alarm_tag[1].strip())
        next_state_id = 4
        next_state_data = "null-action", alarm_event, "null-tag", "null-
tag-object"
        chat_out = "it is time for your " + alarm_event
        return chat_out, next_state_id, next_state_data
    else:
        if match:
            if match.group(1) in YES_TAG:
                print("conf update yes")
                chat_out = "What is the new " +
str(TAG_EXPANDER[current_state_data[2]])
                next_state_id = 3
                return chat_out, next_state_id, current_state_data
            elif match.group(1) in NO_TAG:
                print("conf update no")
                chat_speak("ok that is fine")
                chat_in_phrase = str(make_lower(current_state_data[0])) +
" " + str(current_state_data[1])
```

```
                chat_out, next_state_id, next_state_data =
run_chat_engine(chat_in_phrase, next_state_data)
                return chat_out, next_state_id, next_state_data
            else:
                print("conf update else, cant find any yes no input")
                chat_out, next_state_id, next_state_data =
run_chat_engine(chat_in, next_state_data)
                return chat_out, next_state_id, next_state_data
        else:
            print("conf update else")
            chat_out, next_state_id, next_state_data =
run_chat_engine(chat_in, next_state_data)
            return chat_out, next_state_id, next_state_data
```

File Name : updateUserDataEngine.py

```
"""
    ############################################################
    Update user data
"""

from chatEngine import run_chat_engine
from generalFunctions import make_lower
from functionalParameters import TAG_EXPANDER
from functionalParameters import TAG_DIC
from chatOut import chat_speak


def run_update_user_data_engine(chat_in, current_state_data):

    next_state_data = "null-action", "null-object", "null-tag", "null-tag-
object"

    alarm_tag = chat_in.split("_")
    if alarm_tag[0].strip() == "789alarm987":
        alarm_event = str(alarm_tag[1].strip())
        next_state_id = 4
        next_state_data = "null-action", alarm_event, "null-tag", "null-
tag-object"
        chat_out = "it is time for your " + alarm_event
        return chat_out, next_state_id, next_state_data
    else:
        tag_string = current_state_data[1] + "_" + current_state_data[2]
        print("update - ", tag_string)
        TAG_DIC[tag_string] = chat_in
        chat_speak("wow that is great to know your new " +
str(TAG_EXPANDER[current_state_data[2]]))
        chat_in_phrase = str(make_lower(current_state_data[0])) + " " +
str(current_state_data[1])
        chat_out, next_state_id, next_state_data =
run_chat_engine(chat_in_phrase, next_state_data)
        return chat_out, next_state_id, next_state_data
```

File Name : alarmEngine.py

```
"""
    ############################################################
    Alarm
"""


import re
from chatEngine import run_chat_engine
from generalFunctions import classifier


def run_alarm_engine(chat_in, current_state_data):

    next_state_data = "null-action", "null-object", "null-tag", "null-tag-
object"
    update_alarm_pattern = re.compile(r'(?:can you |could you |)(?:please
|)remind (?:me |)(?:in |by |)([0-9]{1,2}) (minutes|hours)(?: later|)(?:
every day|)(?: please|)')
    # Alarm update is requested

    update_alarm_pattern_match = update_alarm_pattern.match(chat_in)

    alarm_tag = chat_in.split("_")
    if alarm_tag[0].strip() == "789alarm987":
        alarm_event = str(alarm_tag[1].strip())
        next_state_id = 4
        next_state_data = "null-action", alarm_event, "null-tag", "null-
tag-object"
        chat_out = "it is time for your " + alarm_event
        return chat_out, next_state_id, next_state_data
    elif update_alarm_pattern_match:
        # Alarm acknowledged
        print("update reminder direct")
        next_state_id = 5
        next_state_data = update_alarm_pattern_match,
str(current_state_data[1]), "null-tag", "null-tag-object"
        chat_out = "are you sure you want to chang " +
str(current_state_data[1]) + " time"
        print("are you sure you want to change " +
str(current_state_data[1]) + " time")
        return chat_out, next_state_id, next_state_data
    else:
        match, response, count = classifier(chat_in)

        if match == "no-match-available" and response == "no-response-
available":
            # force acknowledge
            next_state_id = 0
            chat_out = "you must take your " + str(current_state_data[1])
+ " on time"
            print("you must take your " + str(current_state_data[1]) + "
on time")
            return chat_out, next_state_id, next_state_data
        else:
            if count == 7:
                # Alarm acknowledged
                print("bring me some food")
                chat_out, next_state_id, next_state_data =
run_chat_engine(chat_in, next_state_data)
                return chat_out, next_state_id, next_state_data
```

```
            elif 22 < count < 27:
                # Alarm acknowledged
                # next_state_id = 5
                # next_state_data = "null-action",
str(current_state_data[1]), "null-tag", "null-tag-object"
                # chat_out = "are you sure you want to update " +
str(current_state_data[1]) + " alarm time"
                # print("are you sure you want to update " +
str(current_state_data[1]) + " alarm time")
                # return chat_out, next_state_id, next_state_data
                print("update reminder 22 < count < 27")
                chat_out, next_state_id, next_state_data =
run_chat_engine(chat_in, next_state_data)
                return chat_out, next_state_id, next_state_data
            else:
                # force acknowledge
                next_state_id = 0
                chat_out = "you must take your " +
str(current_state_data[1]) + " on time"
                print("you must take your " + str(current_state_data[1]) +
" on time")
                return chat_out, next_state_id, next_state_data
```

File Name : updateAlarmEngine.py

```
"""
    ############################################################
    Update Alarm
"""


import re
from unrecognizedChat import reply_unrecognized_chat
from functionalParameters import YES_TAG
from functionalParameters import REMIND_DIC


def run_update_alarm_engine(chat_in, current_state_data):

    next_state_data = "null-action", "null-object", "null-tag", "null-tag-
object"
    yes_no_pattern = re.compile(r'(yes|sure|ok|okay|no|nope|dont|do not)(
.*|)')
    match = yes_no_pattern.match(chat_in)

    alarm_tag = chat_in.split("_")
    if alarm_tag[0].strip() == "789alarm987":
        alarm_event = str(alarm_tag[1].strip())
        next_state_id = 4
        next_state_data = "null-action", alarm_event, "null-tag", "null-
tag-object"
        chat_out = "it is time for your " + alarm_event
        return chat_out, next_state_id, next_state_data
    elif match:
        print("conf rem update yes no match")
        if match.group(1) in YES_TAG:
            print("conf rem update yes")
            if current_state_data[1] == "null-object":
                print("conf rem update yes - 23 > count > 27")
                update_reminder_parameter(current_state_data[0],
```

```
current_state_data[1], current_state_data[2])
                next_state_id = 0
                chat_out = "sure"
                return chat_out, next_state_id, current_state_data
            else:
                print("conf rem update yes - alarm direct update")
                update_reminder_parameter(current_state_data[0],
current_state_data[1], current_state_data[2])
                next_state_id = 0
                chat_out = "sure"
                return chat_out, next_state_id, current_state_data

        else:
            print("conf rem update no")
            if current_state_data[1] == "null-object":
                print("conf rem update no - 23 > count > 27")
                next_state_id = 0
                chat_out = "sure, no worries"
                return chat_out, next_state_id, current_state_data
            else:
                print("conf rem update no - alarm direct update")
                next_state_id = 4
                chat_out = "sure, no worries"
                return chat_out, next_state_id, current_state_data
    else:
        next_state_id = 0
        chat_out = reply_unrecognized_chat()
        return chat_out, next_state_id, next_state_data


"""
    ##############################################################
    Update Alarm Parameter
"""


def update_reminder_parameter(match, reminder_event, count):

    if reminder_event == "null-object":
        print("update request initiated from chat engine")
        for i in REMIND_DIC:
            if count == 23 or count == 25:
                if i[2] == str(match.group(1)) and str(match.group(3)) ==
"minutes":
                    print("match found 23 25 m")
                    i[1] += int(match.group(2))
                    i[3] = 1
                    print(REMIND_DIC)
                    return
                elif i[2] == str(match.group(1)) and str(match.group(3))
== "hours":
                    print("match found 23 25 h")
                    i[0] += int(match.group(2))
                    i[3] = 1
                    print(REMIND_DIC)
                    return
                else:
                    print("no match found 23 25")
            if count == 24 or count == 26:
                if i[2] == str(match.group(1)):
                    print("match found 24 26")
                    i[0] = int(match.group(2))
                    i[1] = int(match.group(3))
                    i[3] = 1
```

```
                    print(REMIND_DIC)
                    return
                else:
                    print("no match found 24 26")
            else:
                print("no match found")

    else:
        print("update request initiated from alarm")
        for i in REMIND_DIC:
            if i[2] == reminder_event and str(match.group(2)) ==
"minutes":
                print("match found")
                i[1] += int(match.group(1))
                i[3] = 1
                print(REMIND_DIC)
                return
            elif i[2] == reminder_event and str(match.group(2)) ==
"hours":
                print("match found")
                i[0] += int(match.group(1))
                i[3] = 1
                print(REMIND_DIC)
                return
            else:
                print("no match found")

    return
```

File Name : functionalParameters.py

```
"""
    ############################################################
    Parameters required for stating functional behaviors
"""

"""
    ################################################################
    Random check for executing an extended chat with question or feedback
"""

RANDOM_CHECK = 9


"""
    ################################################################
    YES NO identifiers
"""


YES_TAG = ("yes", "sure", "ok", "okay")
NO_TAG = ("no", "nope", "dont", "do not")


"""
    ################################################################
    TAG Expander
"""
```

```
TAG_EXPANDER = {}
TAG_EXPANDER["FAV_BOOK"] = "favorite book"
TAG_EXPANDER["LRD_BOOK"] = "last read book"
TAG_EXPANDER["MRD_BOOK"] = "most read book"
TAG_EXPANDER["FAV_FOOD"] = "favorite food"
TAG_EXPANDER["MEA_FOOD"] = "most eat food"


"""
    ############################################################
    <TAG> Identification for feedback response generation
"""


TAG_DIC = {}

TAG_DIC["book_FAV_BOOK"] = "gamperaliya"
TAG_DIC["book_LRD_BOOK"] = "uganthaya"
TAG_DIC["book_MRD_BOOK"] = "viragaya"

TAG_DIC["novel_FAV_BOOK"] = "madoldoowa"
TAG_DIC["novel_LRD_BOOK"] = "oliver twist"
TAG_DIC["novel_MRD_BOOK"] = "harry potter"

TAG_DIC["short story_FAV_BOOK"] = "the red"
TAG_DIC["short story_LRD_BOOK"] = "the last one"
TAG_DIC["short story_MRD_BOOK"] = "geckos"

TAG_DIC["newspaper_FAV_BOOK"] = "sunday times"
TAG_DIC["newspaper_LRD_BOOK"] = "sunday observer"
TAG_DIC["newspaper_MRD_BOOK"] = "daily mirror"

TAG_DIC["magazine_FAV_BOOK"] = "iet"
TAG_DIC["magazine_LRD_BOOK"] = "airport review"
TAG_DIC["magazine_MRD_BOOK"] = "ieee"

TAG_DIC["food_FAV_FOOD"] = "bread"
TAG_DIC["food_MEA_FOOD"] = "cereals"

TAG_DIC["drink_FAV_FOOD"] = "tea"
TAG_DIC["drink_MEA_FOOD"] = "coffee"

TAG_DIC["sweet_FAV_FOOD"] = "chocolate"
TAG_DIC["sewwt_MEA_FOOD"] = "toffee"

TAG_DIC["candy_FAV_FOOD"] = "chocolate"
TAG_DIC["candy_MEA_FOOD"] = "toffee"

TAG_DIC["desert_FAV_FOOD"] = "cake"
TAG_DIC["desert_MEA_FOOD"] = "ice cream"

TAG_DIC["fruit_FAV_FOOD"] = "apple"
TAG_DIC["fruit_MEA_FOOD"] = "mango"

TAG_DIC["gamperaliya_AUT_BOOK"] = "martin wickramasinge"
TAG_DIC["gamperaliya_LAN_BOOK"] = "sinhala"
TAG_DIC["gamperaliya_TYP_BOOK"] = "novel"

TAG_DIC["oliver twist_AUT_BOOK"] = "charles dickens"
TAG_DIC["oliver twist_LAN_BOOK"] = "english"
TAG_DIC["oliver twist_TYP_BOOK"] = "novel"

TAG_DIC["harry potter_AUT_BOOK"] = "j k rowling"
TAG_DIC["harry potter_LAN_BOOK"] = "english"
```

```
TAG_DIC["harry potter_TYP_BOOK"] = "novel"

TAG_DIC["madol doova_AUT_BOOK"] = "martin wickramasinge"
TAG_DIC["madol doova_LAN_BOOK"] = "sinhala"
TAG_DIC["madol doova_TYP_BOOK"] = "novel"

TAG_DIC["uganthaya_AUT_BOOK"] = "martin wickramasinge"
TAG_DIC["uganthaya_LAN_BOOK"] = "english"
TAG_DIC["uganthaya_TYP_BOOK"] = "novel"

TAG_DIC["apple_GOD_FOOD"] = "health"
TAG_DIC["apple_BAD_FOOD"] = "significantly nothing"
TAG_DIC["apple_NUT_FOOD"] = "vitamin"

TAG_DIC["tea_GOD_FOOD"] = "for health"
TAG_DIC["tea_BAD_FOOD"] = "for diabetes"
TAG_DIC["tea_NUT_FOOD"] = "minerals"

TAG_DIC["bread_GOD_FOOD"] = "to eat with curry"
TAG_DIC["bread_BAD_FOOD"] = "for diabetes and cholesterol"
TAG_DIC["bread_NUT_FOOD"] = "minerals"

TAG_DIC["water_GOD_FOOD"] = "for thirsty"
TAG_DIC["water_BAD_FOOD"] = "when taken too much"
TAG_DIC["water_NUT_FOOD"] = "minerals"

TAG_DIC["coke_GOD_FOOD"] = "for thirsty"
TAG_DIC["coke_BAD_FOOD"] = "for diabetes as it contain lots of sugar"
TAG_DIC["coke_NUT_FOOD"] = "lots of sugar"

TAG_DIC["chocolate_GOD_FOOD"] = "as an energizer"
TAG_DIC["chocolate_BAD_FOOD"] = "for diabetes as it contain lots of sugar"
TAG_DIC["chocolate_NUT_FOOD"] = "calories"

TAG_DIC["ice cream_GOD_FOOD"] = "during warm days"
TAG_DIC["ice cream_BAD_FOOD"] = "for diabetes as it contain lots of sugar"
TAG_DIC["ice cream_NUT_FOOD"] = "calories"


"""
    ################################################################
    Action dictionary
"""


ACTION_DIC = {}
ACTION_DIC[1] = "SPEAK"
ACTION_DIC[2] = "SPEAK"
ACTION_DIC[3] = "SPEAK"
ACTION_DIC[4] = "SPEAK"
ACTION_DIC[5] = "SPEAK"
ACTION_DIC[6] = "SPEAK"
ACTION_DIC[7] = "BRING"
ACTION_DIC[8] = "SWITCH_OFF"
ACTION_DIC[9] = "SWITCH_ON"
ACTION_DIC[10] = "SWITCH_OFF"
ACTION_DIC[11] = "SWITCH_ON"
ACTION_DIC[12] = "MOVE"
ACTION_DIC[13] = "CALL"
ACTION_DIC[14] = "WALK"
ACTION_DIC[15] = "WASH"
ACTION_DIC[16] = "CLEAN"
ACTION_DIC[17] = "BRUSH"
ACTION_DIC[18] = "COMB"
```

```
ACTION_DIC[19] = "READ"
ACTION_DIC[20] = "RUB"
ACTION_DIC[21] = "HELP"
ACTION_DIC[22] = "SPEAK"
ACTION_DIC[23] = "SPEAK"
ACTION_DIC[24] = "SPEAK"
ACTION_DIC[25] = "SPEAK"
ACTION_DIC[26] = "SPEAK"
ACTION_DIC[27] = "BRING"
ACTION_DIC[28] = "SPEAK"
ACTION_DIC[29] = "BRING"
ACTION_DIC[30] = "SPEAK"
ACTION_DIC[31] = "SPEAK"
ACTION_DIC[32] = "CHARGE"
ACTION_DIC[33] = "GIVE"

"""
    ############################################################
    Reminders
"""


REMIND_DIC = [[8, 20, "breakfast", 1], [18, 29, "lunch", 1], [17, 51,
"dinner", 1], [8, 30, "morning medicine", 1], [13, 30, "after noon
medicine", 1], [20, 30, "evening medicine", 1]]
```

File Name : actionClasifier.py

```
"""
    ############################################################
    Classify the next action based on input type; next action requires a
clarification
    or providing a feedback
"""

import re
import random
from regexPairs import feedback_regex_pairs
from regexPairs import question_regex_pairs
from functionalParameters import TAG_DIC

# Create regex keyword pairs object
feedback_regex_pairs = [(re.compile(x, re.IGNORECASE), y) for (x, y) in
feedback_regex_pairs]
question_regex_pairs = [(re.compile(x, re.IGNORECASE), y) for (x, y) in
question_regex_pairs]

"""
    ############################################################
    Classify based on keyword type
"""


def action_classifier(action_classifier_phrase):

    action, keyword_type, keyword_category, keyword =
action_classifier_phrase.split("_")

    if keyword_category == "ITM":
        chat_out, next_state_id, next_state_data =
```

```
generate_respond_feedback(action_classifier_phrase)

    elif keyword_category == "TYP":
        chat_out, next_state_id, next_state_data =
generate_question_feedback(action_classifier_phrase)

    return chat_out, next_state_id, next_state_data


"""
    ################################################################
    Generate a feedback response for the chat input
"""


def generate_respond_feedback(action_classifier_phrase):

    # check matching feedback response pattern
    for (pattern, response) in feedback_regex_pairs:
        match = pattern.match(action_classifier_phrase)

        if match:
            selected_response = random.choice(response)
            chat_out, tag, tag_object = expand_tags(selected_response,
match)
            print(chat_out)
            next_state_id = 0
            next_state_data = "null-action", "null-object", "null-tag"
            return chat_out, next_state_id, next_state_data

    # If no matching feedback response pattern found
    next_state_id = 0
    next_state_data = "null-action", "null-object", "null-tag", "null-tag-
object"
    return "feedback response - " + str(action_classifier_phrase),
next_state_id, next_state_data


"""
    ################################################################
    Generate a question response for the chat input
"""


def generate_question_feedback(action_classifier_phrase):
    # check matching feedback response pattern
    for (pattern, response) in question_regex_pairs:
        match = pattern.match(action_classifier_phrase)

        if match:
            selected_response = random.choice(response)
            chat_out, tag, tag_object = expand_tags(selected_response,
match)
            # action_classifier_phrase =
            # ACTION_DIC[count] + "_" + keyword_type + "_" +
keyword_category + "_" + keyword
            split_action_classifier_phrase =
action_classifier_phrase.split("_")
            next_state_id = 1
            next_state_data = split_action_classifier_phrase[0],
split_action_classifier_phrase[3], tag, tag_object
            return chat_out, next_state_id, next_state_data

    # If no matching question response pattern found
```

```
    next_state_id = 1
    next_state_data = "null-action", "null-object", "null-tag", "null-tag-
object"
    return "feedback response - " + str(action_classifier_phrase),
next_state_id, next_state_data


"""
    ###############################################################
    Condition the matched action_classifier_phrase
"""


def expand_tags(response, match):
    # Replace wild cards with block of user input
    wildcard_response, wildcard_object = set_wildcard_strings(response,
match)
    if wildcard_object == "no-wildcard_strings-available":
        return response, "null-tag", "null-tag-object"
    else:
        tagged_response, tag, tag_object = set_tags(wildcard_response,
wildcard_object)
        return tagged_response, tag, tag_object


"""
    ###############################################################
    find and replace feedback <> tags
"""


def set_tags(wildcard_resp, wildcard_object):
    # find the position of the tag
    pos = wildcard_resp.find('<')
    # check for multiple tags
    tag = "null-tag"
    tag_object = "null-tag-object"
    while pos >= 0:
        # select the tag without <>
        tag = wildcard_resp[pos + 1:pos + 9]
        # create the response with substituted tags
        tag_object = substitute_tags(tag, wildcard_object)
        wildcard_resp = wildcard_resp[:pos] + tag_object +
wildcard_resp[pos + 10:]
        pos = wildcard_resp.find('<')
    return wildcard_resp, tag, tag_object

"""
    ###############################################################
    Replace <> tags with suitable content available
"""


def substitute_tags(tag, wildcard_object):
    tag_string = wildcard_object + "_" + tag
    return TAG_DIC[tag_string]


"""
    ###############################################################
    Condition regular expression respond
        Replace wild cards with block of user input
"""
```

```
def set_wildcard_strings(response, match):
    # find the position of the first regex replacement pattern
    pos = response.find('%')
    # check for multiple patterns
    if pos >= 0:
        # convert position of first regex replacement pattern to numerical
value
        num = int(response[pos + 1:pos + 2])
        # find the substitution from the match and store
        regex_obj_str = match.group(num)
        response = response[:pos] + regex_obj_str + response[pos + 2:]
        return response, regex_obj_str
    else:
        return response, "no-wildcard_strings-available"
```

File Name : unrecognizedChat.py

```
"""
    ############################################################
    Reply for unrecognized chat input
"""


def reply_unrecognized_chat():
    chat_out = "pardon me"
    return chat_out
```

File Name : reminderBot.py

```
"""
    ############################################################
    Handles all the reminders
"""

import datetime
from functionalParameters import REMIND_DIC


def reminder_bot():

    TEMP_REMIND_DIC = [12, 12, "temp_reminder", 0]

    t_now = datetime.datetime.now()

    for i in REMIND_DIC:

        if t_now.hour == i[0] and t_now.minute == i[1] and i[3] == 1:
            # current alarm is disabled for restricting one time
occurrence
            i[3] = 0
            # Store the current reminder time in temporary location.
            # Fix reminders are passed to temp register and then alarmed
from tmp register
            TEMP_REMIND_DIC[0] = i[0]
            TEMP_REMIND_DIC[1] = i[1]
```

```
            TEMP_REMIND_DIC[2] = i[2]
            TEMP_REMIND_DIC[3] = 1

            print("reminder for ", i[2], " @ Hr ", i[0], " Mn ", i[1])

        elif t_now.hour == 0 and t_now.minute == 0:
            # enable all alarms at 12 midnight
            i[3] = 1

        # tmp register
        if t_now.hour == TEMP_REMIND_DIC[0] and t_now.minute ==
TEMP_REMIND_DIC[1] and TEMP_REMIND_DIC[3] == 1:
            # current alarm is disabled for restricting one time
occurrence
            TEMP_REMIND_DIC[3] = 0
            # alarm
            alarm_string = str(TEMP_REMIND_DIC[2])
            print(alarm_string)

            return alarm_string

    return "no-alarm"
```