



INSTITUTE FOR ADVANCED
COMPUTING AND
SOFTWARE DEVELOPMENT
AKURDI, PUNE
Documentation On

"Omnipresent buddy - A chatbot which ease your life "

PG-DBDA FEB 2020

Submitted by :

Group No:07

Bony Srivastava 1310

Devender Kumar 1316

Mr. Prashant Karhale
Centre Coordinator

Mr. Akshay Tilekar
Project Guide

INDEX

Table of Contents

System Engineering and Analysis:	5
Software requirement Analysis:	5
Design:	5
Coding:	5
Testing:	5
Limitation of unit testing:	6
Maintenance:	6
What is a chatbot?	7
How do Chatbots work?	8
Building the Bot	10
Pre-requisites	10
NLP	10
NLTK: A Brief Intro	11
Text Pre-Processing	12
Bag of Words	13
TF-IDF Approach	14
Cosine Similarity	15
Importing the necessary libraries	16
Corpus	16
Reading in the data	16
Pre-processing the raw text	17
Keyword Matching	17
Generating Response	17
Roborasa	21
Quick Installation	19
Important Concepts	22
File Structure	23
Conclusion	Error! Bookmark not defined.

ABSTRACT

Chatbots, or conversational interfaces as they are also known, present a new way for individuals to interact with computer systems. Traditionally, to get a question answered by a software program involved using a search engine, or filling out a form. A chatbot allows a user to simply ask questions in the same manner that they would address a human. The most well known chatbots currently are voice chatbots: Alexa and Siri. However, chatbots are currently being adopted at a high rate on computer chat platforms.

Basically, Chatbots can mimic human conversation and entertain users. The technology at the core of the rise of the chatbot is natural language processing ("NLP"). chatbots eliminate the requirement of any manpower during online interaction and are hence seen as a big advantage by companies receiving multiple queries at once. This also presents companies with the opportunity to save on costs while aligning chatbots with their goals and hence presenting customers with a particular type of interaction leading to conversion.

Software Life Cycle Model

In order to make this Project we are going to use Classic LIFE CYCLE MODEL .Classic life cycle model is also known as WATERFALL MODEL. The life cycle model demands a Systematic sequential approach to software development that begins at the system level and progresses through analysis design coding, testing and maintenance.

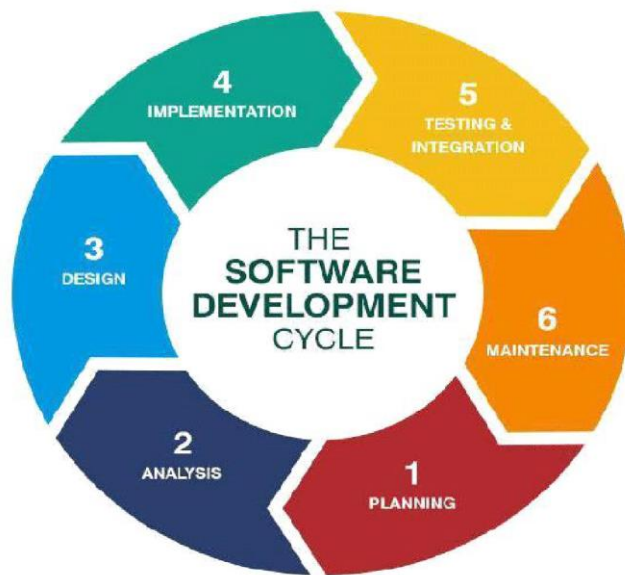


Fig-1

The Classic Life Cycle Model

The waterfall model is a sequential software development process, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of conception initiation, Analysis, Design (validation), construction. Testing and maintained.

System Engineering and Analysis:

Because software is always a part of larger system work. Begins by establishing requirements for all system elements and Then allocating some subset of these requirements to the software system Engineering and analysis encompasses the requirement gathering at the system level with a small amount of top level design and analysis.

Software requirement Analysis:

The requirement gathering process is intensified and focused specifically on the software Requirement for the both system and software are discussed and reviewed with the customer. The customer specifies the entire requirement or the software and the function to be performed by the software.

Design:

Software design is actually a multi-step process that focuses on distinct attributes of the program data structure, software architecture, procedural detail and interface of the software that can be assessed or quality before coding begins .Like requirement the design is documented and becomes part of the software.

Coding:

The design must be translated into a machine readable form. The coding step performs this task. If design is programmed in a detailed manner, coding can be accomplished mechanically

Testing:

Once code has been generated programmed testing begins. The testing process focuses on the internals of the software ensuring that all statement have been tested and on the functional externals hat is conducting tests to uncover the errors and ensure that defined input will produce the results that agree with the required results.

Unit testing:

In computer programming, Unit testing is software Verification and validation method where the programmer gains confidence that individual units of source

code are fit to use A unit is the smallest testable part of an application. In procedural programming a unit may be an individual programmed, function, procedure, etc. while in object-oriented programming, the smallest Unit is a class, which may belong to a base/superclass abstract class or derived/child class.

Benefits:

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict written contract that the piece of code must satisfy.

Documentation:

Unit testing provides a sort of living documentation of the system. Developers looking to learn what functionality is provided by a unit and how to use it can look at the tests to gain a basic understanding of the unit API.

Limitation of unit testing:

Testing cannot be expected to catch error in the program –It is impossible to evaluate all execution paths for all but the most trivial programs. The same is true for unit testing. Additionally, unit testing only types the functionality of the units themselves.

Maintenance:

Software will undoubtedly undergo change after it is Delivered to the customer .Change will occur because errors have been encountered because the software must be able to accommodate changes in its external environment because the customer requires functional or performance enhancement enhancements. The classic life cycle is the oldest and most widely used paradigm or software engineering.

PURPOSE

What is a chatbot?

A chatbot is an artificial intelligence-powered piece of software in a device , application, website or other networks that try to gauge consumer's needs and then assist them to perform a particular task like a commercial transaction, hotel booking, form submission etc . Today almost every company has a chatbot deployed to engage with the users. Some of the ways in which companies are using chatbots are:

- To deliver flight information
- to connect customers and their finances
- As customer support

The possibilities are (almost) limitless.

History of chatbots dates back to 1966 when a computer program called ELIZA was invented by Weizenbaum. It imitated the language of a psychotherapist from only 200 lines of code.

DEEP DIVE

The Bot Evolution

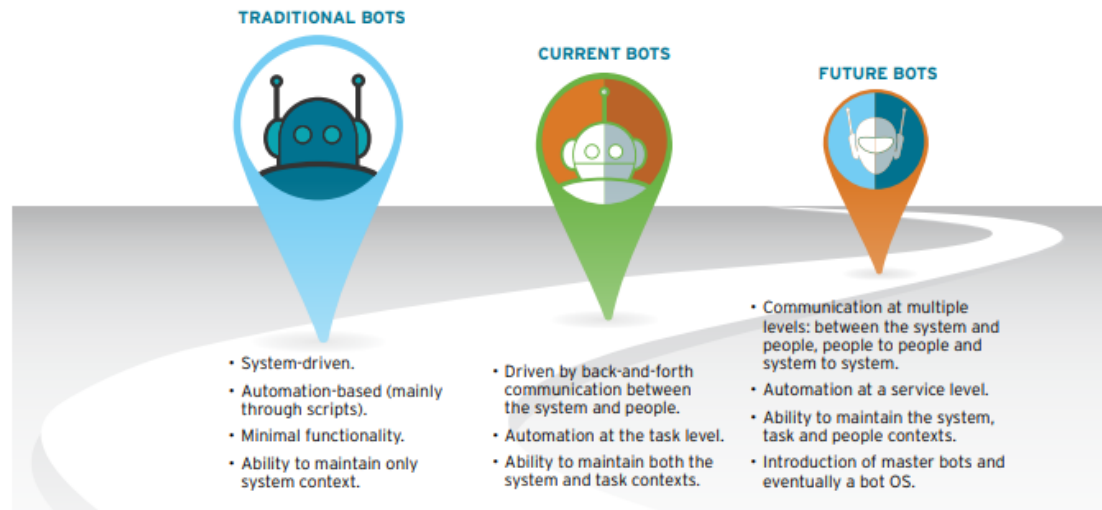


Fig-2

How do Chatbots work?

There are broadly two variants of chatbots: **Rule-Based** and **Self-learning**.

1. In a **Rule-based approach**, a bot answers questions based on some rules on which it is trained on. The rules defined can be very simple to very complex. The bots can handle simple queries but fail to manage complex ones.
2. Self-learning bots are the ones that use some Machine Learning-based approaches and are definitely more efficient than rule-based bots. These bots can be of further two types: **Retrieval Based** or **Generative**

i) In **retrieval-based models**, a chatbot uses some heuristic to select a response from a library of predefined responses. The chatbot uses the message and context of the conversation for selecting the best response from a predefined list of bot messages. The context can include a current position in the dialogue tree, all previous messages in the conversation, previously saved variables (e.g. username). Heuristics for selecting a response can be engineered in many different ways, from rule-based if-else conditional logic to machine learning classifiers.

ii) **Generative** bots can generate the answers and not always replies with one of the answers from a set of answers. This makes them more intelligent as they take word by word from the query and generates the answers.

Anatomy of a Chatbot

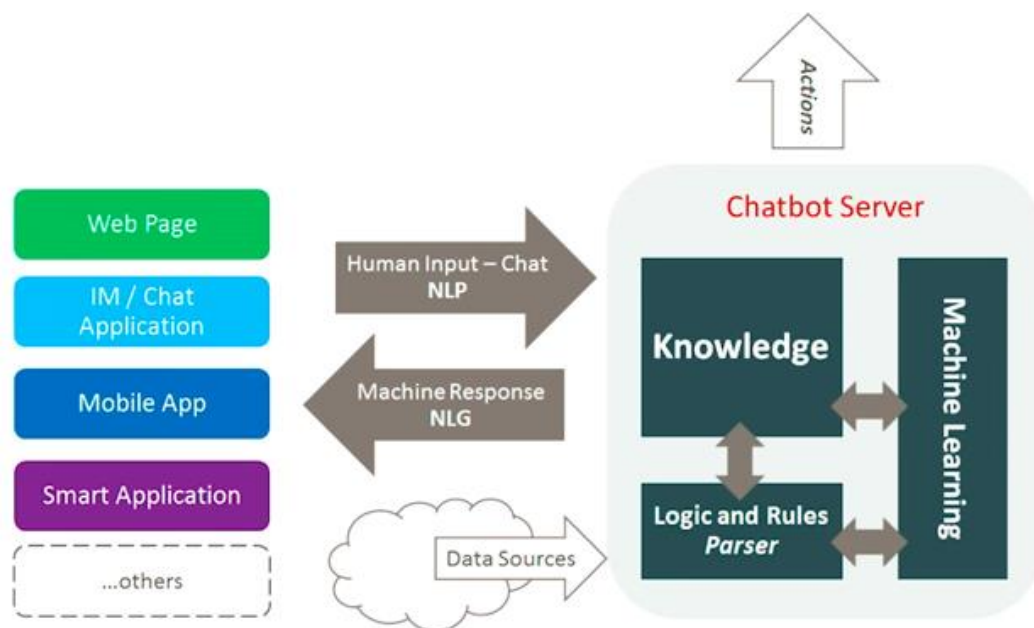


Fig-3

Building the Bot

Pre-requisites

Hands-On knowledge of **scikit** library and **NLTK** is assumed. However, if you are new to NLP, you can still read the article and then refer back to resources.

NLP

The field of study that focuses on the interactions between human language and computers is called Natural Language Processing, or NLP for short. It sits at the intersection of computer science, artificial intelligence, and computational linguistics. NLP is a way for computers to analyze, understand, and derive meaning from human language in a smart and useful way. By utilizing NLP, developers can organize and structure knowledge to perform tasks such as automatic summarization, translation, named entity recognition, relationship extraction, sentiment analysis, speech recognition, and topic segmentation.

NLTK: A Brief Intro

NLTK(Natural Language Toolkit) is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries.

NLTK has been called “a wonderful tool for teaching and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

Natural Language Processing with Python provides a practical introduction to programming for language processing. I highly recommend this book to people beginning in NLP with Python.

Downloading and installing NLTK

1. Install NLTK: run `pip install nltk`
2. Test installation: run `python` then type `import nltk`

Installing NLTK Packages

import NLTK and run `nltk.download()`. This will open the NLTK downloader from where you can choose the corpora and models to download. You can also download all packages at once.

Text Pre- Processing with NLTK

The main issue with text data is that it is all in text format (strings). However, Machine learning algorithms need some sort of numerical feature vector in order to perform the task. So before we start with any NLP project we need to pre-process it to make it ideal for work. Basic **text pre-processing** includes:

- Converting the entire text into **uppercase or lowercase**, so that the algorithm does not treat the same words in different cases as different
- **Tokenization**: Tokenization is just the term used to describe the process of converting the normal text strings into a list of tokens i.e words that we actually want. Sentence tokenizer can be used to find the list of sentences and Word tokenizer can be used to find the list of words in strings.

The NLTK data package includes a pre-trained Punkt tokenizer for English.

- Removing **Noise** i.e everything that isn't in a standard number or letter.
- Removing **Stop words**. Sometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called *stop words*

Stemming: Stemming is the process of reducing inflected (or sometimes derived) words to their stem, base or root form — generally a written word form. Example if we were to stem the following words: “Stems”, “Stemming”, “Stemmed”, “and Stigmatization”, the result would be a single word “stem”.

Lemmatization: A slight variant of stemming is lemmatization. The major difference between these is that stemming can often create non-existent words, whereas lemmas are actual words. So, your root stem, meaning the word you end up with, is not something you can just look up in a dictionary, but you can look up a lemma. Examples of Lemmatization are that “run” is a base form for words like “running” or “ran” or that the word “better” and “good” are in the same lemma so they are considered the same.

Bag of Words: After the initial preprocessing phase, we need to transform the text into a meaningful vector (or array) of numbers. The bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

- A vocabulary of known words.
- A measure of the presence of known words.

Why is it called a “*bag*” of words? A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things: A vocabulary of known words. A measure of the presence of known words.

The intuition behind the Bag of Words is that documents are similar if they have similar content. Also, we can learn something about the meaning of the document from its content alone.

For example, if our dictionary contains the words {Learning, is, the, not, great}, and we want to vectorize the text “Learning is great”, we would have the following vector: (1, 1, 0, 0, 1).

TF-IDF Approach

A problem with the Bag of Words approach is that highly frequent words start to dominate in the document (e.g. larger score), but may not contain as much “informational content”. Also, it will give more weight to longer documents than shorter documents.

One approach is to rescale the frequency of words by how often they appear in all documents so that the scores for frequent words like “the” that are also frequent across all documents are penalized. This approach to scoring is called **Term Frequency-Inverse Document Frequency**, or **TF-IDF** for short, where:

Term Frequency: is a scoring of the frequency of the word in the current document.

Inverse Document Frequency: is a scoring of how rare the word is across documents.

$IDF = 1 + \log(N/n)$, where, N is the number of documents and n is the number of documents a term t has appeared in.

Example:

Consider a document containing 100 words wherein the word 'phone' appears 5 times.

*The term frequency (i.e., tf) for phone is then $(5 / 100) = 0.05$. Now, assume we have 10 million documents and the word phone appears in one thousand of these. Then, the inverse document frequency (i.e., IDF) is calculated as $\log(10,000,000 / 1,000) = 4$. Thus, the Tf-IDF weight is the product of these quantities: $0.05 * 4 = 0.20$.*

Tf-IDF can be implemented in scikit learn as:

from sklearn.feature_extraction.text import TfidfVectorizer

Cosine Similarity

TF-IDF is a transformation applied to texts to get two real-valued vectors in vector space. We can then obtain the **Cosine** similarity of any pair of vectors by taking their dot product and dividing that by the product of their norms. That yields the cosine of the angle between the vectors. **Cosine similarity** is a measure of similarity between two non-zero vectors.

Importing the necessary libraries

```
#import necessary libraries
import io # dealing with various types of I/O
import random #to generate random values in a given sequence
import string # to process standard python strings
import warnings #useful to alert the user of some condition
import numpy as np
```

Corpus

For our example, we will be using the Wikipedia page for chatbots as our corpus. Copy the contents from the page and place it in a text file named 'chatbot.txt'. However, you can use any corpus of your choice.

Reading in the data

We will read in the corpus.txt file and convert the entire corpus into a list of sentences and a list of words for further pre-processing.

```
# uncomment the following only the first time
nltk.download('punkt') # first-time use only
nltk.download('wordnet') # first-time use only

#Reading in the corpus
with open('/content/coronanewinfo.txt', 'r', encoding='utf8', errors='ignore') as fin:
    raw = fin.read().lower()
```

Let see an example of the sent_tokens and the word_token


```
#Tokenisation
sent_tokens = nltk.sent_tokenize(raw)# converts to list of sentences
word_tokens = nltk.word_tokenize(raw)# converts to list of words
```

Pre-processing the raw text

We shall now define a function called LemTokens which will take as input the tokens and return normalized tokens.

```
# Preprocessing
lemmer = WordNetLemmatizer()
def LemTokens(tokens):
    return [lemmer.lemmatize(token) for token in tokens]
remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
```

Keyword matching

Next, we shall define a function for a greeting by the bot i.e if a user's input is a greeting, the bot shall return a greeting response. ELIZA uses a simple keyword matching for greetings. We will utilize the same concept here.

```
# Keyword Matching
GREETING_INPUTS = ("hello", "hi", "greetings", "sup", "what's up","hey",)
GREETING_RESPONSES = ["hi", "hey", "**nods**", "hi there", "hello", "I am glad! You are talking to me"]

def greeting(sentence):
    """If user's input is a greeting, return a greeting response"""
    for word in sentence.split():
        if word.lower() in GREETING_INPUTS:
            return random.choice(GREETING_RESPONSES)
```

Generating Response

To generate a response from our bot for input questions, the concept of document similarity will be used. So we begin by importing the necessary modules.

From scikit learn library, import the TFidf vectorizer to convert a collection of raw documents to a matrix of TF-IDF features.

Also, import `cosine` similarity module from scikit learn library.

We define a function **response** which searches the user's utterance for one or more known keywords and returns one of several possible responses. If it doesn't find the input matching any of the keywords, it returns a response: "I am sorry! I don't understand you"

```
# Generating response
def response(user_response):
    robo_response=''
    sent_tokens.append(user_response)
    TfidfVec = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english')
    tfidf = TfidfVec.fit_transform(sent_tokens)
    vals = cosine_similarity(tfidf[-1], tfidf)
    idx=vals.argsort()[0][-2]
    flat = vals.flatten()
    flat.sort()
    req_tfidf = flat[-2]
    if(req_tfidf==0):
        robo_response=robo_response+"I am sorry! I don't understand you"
        return robo_response
    else:
        robo_response = robo_response+sent_tokens[idx]
        return robo_response
```

Finally, we will feed the lines that we want our bot to say while starting and ending a conversation depending upon the user's input.

```
flag=True
print("ROBO: My name is Robo. I will answer your queries about coronavirus.")
while(flag==True):
    user_response = input()
    user_response=user_response.lower()
    if(user_response!='bye'):
        if(user_response=='thanks' or user_response=='thank you' ):
            flag=False
            print("ROBO: You are welcome..")
        else:
            if(greeting(user_response)!=None):
                print("ROBO: "+greeting(user_response))
            else:
                print("ROBO: ",end="")
                print(response(user_response))
                sent_tokens.remove(user_response)
    else:
        flag=False
        print("ROBO: Bye! take care..")
```

ROBORASA

What is RASA?

Rasa is an open source machine learning framework for automated text and voice-based conversations. Understand messages, hold conversations, and connect to messaging channels and APIs.

Quick Install

Download Anaconda

Download Visual Studio C distributable

Create folder roborasa

Conda create --name botchat venv python==3.7.6

Conda activate botched venv

Conda install ujson==2.0.3

Conda install tensorflow==2.1.0

Pip install rasa==1.10.0

Rasa init

IMPORTANT CONCEPTS:

INTENTS:

Consider it as the main or target of the user input. If a user says "hey", "hello", here the intent would be greeting

The Verb in your dialog.

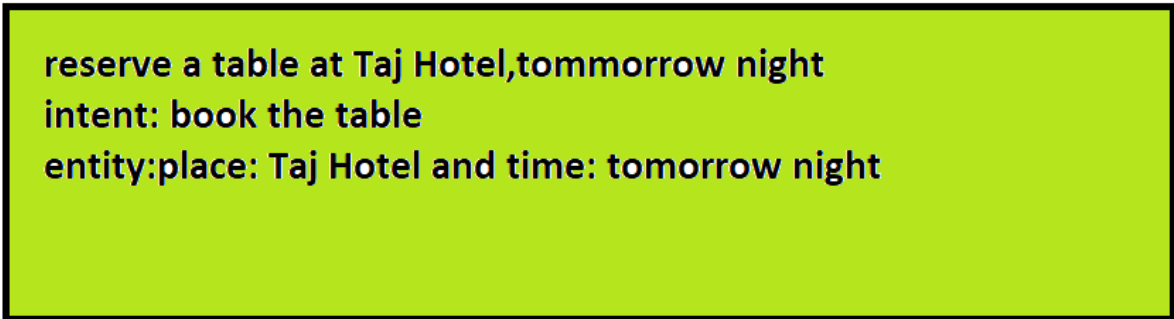


HI

ENTITY:

Consider it as the information from user input that can be extracted. "I want to order a book". Here "book" is an entity.

The Nouns are in your dialog.



reserve a table at Taj Hotel, tomorrow night
intent: book the table
entity: place: Taj Hotel and time: tomorrow night

ACTIONS:

As the name suggests it is an operation which can be performed by the bot. It could be replying in return, querying a database

Action could be just a hard copy reply or some API response.

STORIES:

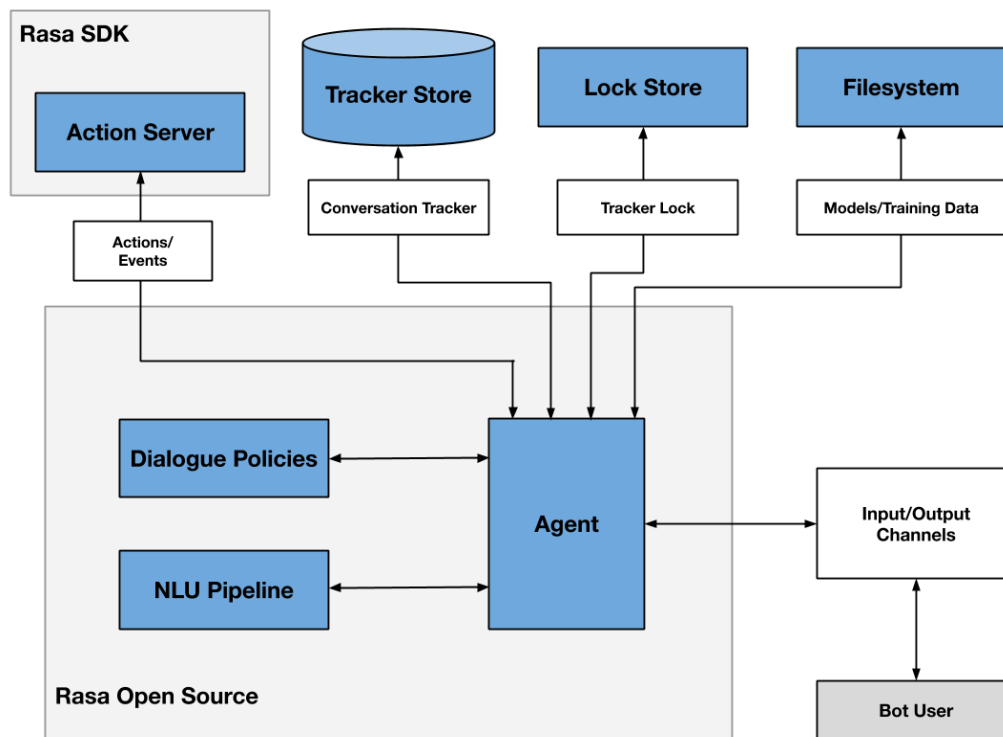
These are sample interactions between the user and the bot, defined in terms of intents captured and actions performed

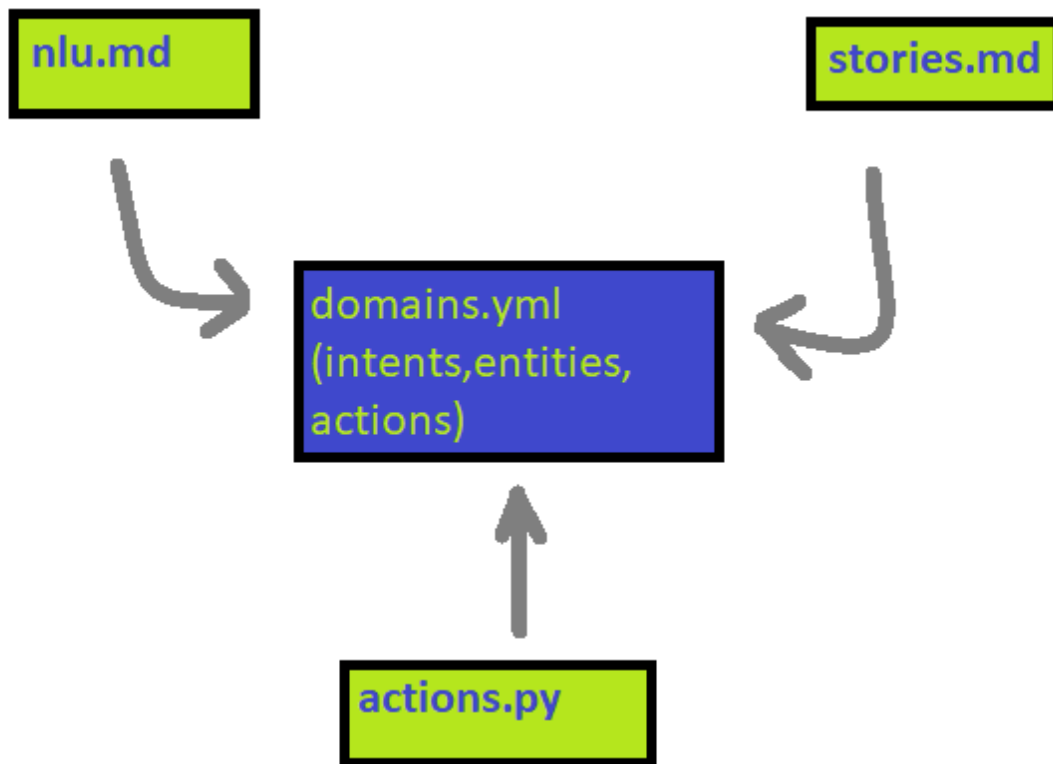
DOMAIN:

Domain knowledge is required to reply for any user with the help of intents, actions, responses and entities.

FILE STRUCTURE:

"OmniPresent buddy - A chatbot which ease your life"





Rasa important feature are:

1. rasa nlu
2. rasa core

<https://github.com/devenderkumar1605/ROBOJI-ChatBot/tree/ROBORASA>

Conclusion

Though it is a very simple bot with hardly any cognitive skills, its a good way to get into NLP and get to know about chatbots. Though 'ROBO' responds to user input. It won't fool your friends, and for a production system you'll want to consider one of the existing bot platforms or frameworks, but this example should help you think through the design and challenge of creating a chatbot. From my perspective, chatbots or smart assistants with artificial intelligence are dramatically changing businesses. There is a wide range of chatbot building platforms that are available for various enterprises, such as e-commerce, retail, banking, leisure, travel, healthcare, and so on.

Chatbots can reach out to a large audience on messaging apps and be more effective than humans. They may develop into a capable information-gathering tool in the near future.