Domain Specific Intelligent Personal Assistant with Bilingual Voice Command Processing

Saadman Shahid Chowdhury
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
saadman.shahid@gmail.com

Atiar Talukdar
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
swajan.talukdar@gmail.com

Ashik Mahmud
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
ashik.mahmud137@gmail.com

Tanzilur Rahman
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
tanzilur.rahman@northsouth.edu

***Abstract* — Intelligent Personal Assistants (IPA), like Siri and Alexa, are created to assist their users with simple digital tasks. Here, we propose the steps we have used to develop a voice operated IPA which can process direct commands in two languages: English and Bengali, to perform menial tasks for the users. The speech recognition engine of the IPA is constructed with Sphinx-4, and the language processing is performed by a modified finite state automaton. The IPA also takes advantage of the subject/action structure of commands to reduce the size of the word domain, and utilizes a generalization function to ensure that the language processor can understand multiple languages without undergoing major modification - making this approach suitable when training data is limited.**

Keywords — Digital Assistant, Artificial Intelligence, Automated Speech Recognition, Finite State Automaton, Language Processing, Voice User Interface.

I. Introduction

An Intelligent Personal Assistant (IPA) is a computer program with Artificial Intelligence (AI), designed to aid its users with their tasks. The IPA communicates seamlessly with its users while it answers their inquiries or performs actions to satisfy their requests [1]. Modern IPAs can perform a wide variety of tasks - ranging from binary tasks like opening an app or setting an alarm, to more complex tasks like noting down dictations or making phone calls. Notable examples of such IPAs are Google Assistant from Google, Siri from Apple, and Alexa from Amazon.

IPAs do not necessarily need to communicate only through voice, but modern IPAs are pushing towards Voice User Interfacing (VUI), i.e. interacting with users only through voice, without the use of screens or physical interaction [2]. This requires the IPA to: (A) listen to human speech, (B) understand what is being implied, and (C) perform an action or reply with their own synthesized voice [3]. This concept is visualized in Fig.1.

Firstly, for (A), the IPA needs to have Automated Speech Recognition (ASR) capabilities - one such ASR module creation tools is the Sphinx-4 framework, created by Carnegie Mellon University, which uses Hidden Markov Models - a probabilistic approach to classifying audio signals into words [4]. Furthermore, ASR modules require use of linguistic knowledge libraries called acoustic models for each language. In our research, we have used the Sphinx-4 and its aiding software: Sphinx-Train to create a speech recognition model to suit our IPA, which understands domain specific commands in two languages: English and Bengali [5][6].

Secondly, for (B), the IPA needs to use a Natural Language Processing (NLP) module. The NLP's task is to infer what is being implied in a sentence, and decide on the course of action in response. A simple, yet effective, way of creating an NLP module is to use Deterministic Finite State Automatons (FSA) [7] - which are unidirectional graphs with two types of vertices: accepting states and non-accepting states. Through traversal of the graph, if an accepting state is reached, an action takes place; otherwise, a non-accepting state is reached, where no action takes place or an error is thrown [8].

For the purpose of this paper, we have created a functional IPA, which can do menial tasks like turning on/off smartphone applications. It is able to process "audio commands" and perform actions through VUI. This paper is written to assist the readers in replicating our steps to create their own IPA. Finally, to demonstrate how the IPA can be fine-tuned for multiple languages, we have designed our IPA to understand two languages: English and Bengali. In II.Methodology, we describe the steps in detail and give instructions on how to build the ASR and NLP, and how to improve the effectiveness of the IPA by identifying the "subject/action" structure of commands. In III.Limitations, we note what the shortcomings of our approach is. Finally, In IV.Conclusion we provide pointers on how our work can be further improved upon to create a more effective IPA.

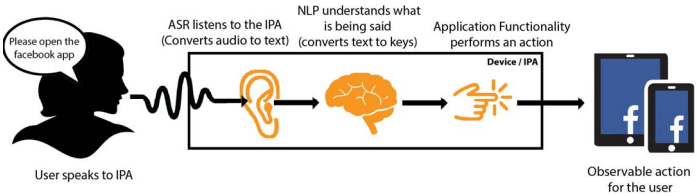

Fig. 1. Concept of IPA

II. Methodology

To make our methodology easier to understand, we have shown the flow of information through the IPA in Fig.2, starting from (1) "User speaks to IPA" to (7) "Observable action". Where (1) is the input to the IPA, and (7) is the output from the IPA. Briefly - from (1) an audio file is produced, which is passed through the ASR module (2),(3) - creating a text file, i.e. the spoken sentence into computer text. The text file is then passed through the NLP module (4),(5) which attempts to infer what is being implied in the sentence - if it cannot comprehend the user's command, it throws an error; but if it can understand what is being implied, it passes a specific key to the next module. Step (6) accepts the key and performs an action depending on the particular key. Each step along the way, from (2) to (7), are described in the Methodology in detail. The interaction between the modules visualized in Fig.1.

A. Step 2: Audio Processing

In step (1), the user speaks into the microphone of the device, which creates an audio file. The audio file is an array of integers containing information on the audio signal. Depending on the recording device, the audio file might have characteristics that cannot be processed by speech recognition engines made using Sphinx-4. Hence, the audio file's properties must first be converted to Mono Channels, 16000Hz Sampling Rate, and 32-bit Sample Size. This standardizes all the audio inputs to the speech recognition engine.

B. Step 3: Speech Recognition Engine

The Sphinx framework requires 3 files to operate: Language Model (LM), Acoustic Model (AM), and the Dictionary. The Dictionary is simply a list of words which our ASR is able to classify; the full list of words in our dictionary is in Fig.3. The LM is the phonetic breakdown of the words in the Dictionary – it contains the sequence of phonetic units (phonemes) that make up those words; the LM can be acquired by uploading the Dictionary to the Sphinx Knowledge Base Tool website. The AM is a file that contains probabilistic values which are used to identify phonemes from the values in audio files – through the use of Hidden Markov Models; the AM can be created using the SphinxTrain tool – which accepts the speech corpus: audio AM training files and transcript files. The transcript is simply just the text representations of the audio files. Our training corpus for SphinxTrain is 2.5 Hours long and has recordings of 120 speakers, speaking 35 different words, 10 times each in different pitch and accent – this is sufficient for domain specific speech recognizers.

The IPA has been designed with efficiency in mind. We have identified a structure in direct verbal commands, where there is always atleast one "subject" and atleast one action that needs to be performed on that "action" - all the other words are usually redundant. By only identifying the subject words and action words, and discarding the rest of the words in the command, it is possible to understand what the IPA needs to do. e.g. in the instruction: "could you please turn on the facebook app?", by discarding majority of the words, and only identifying the subject word: "facebook", and action word: "on" - the IPA can deduce what do next. Because fewer words are used, the need of processing is less and the success rate is higher - yet, to the user - the effect is the same.

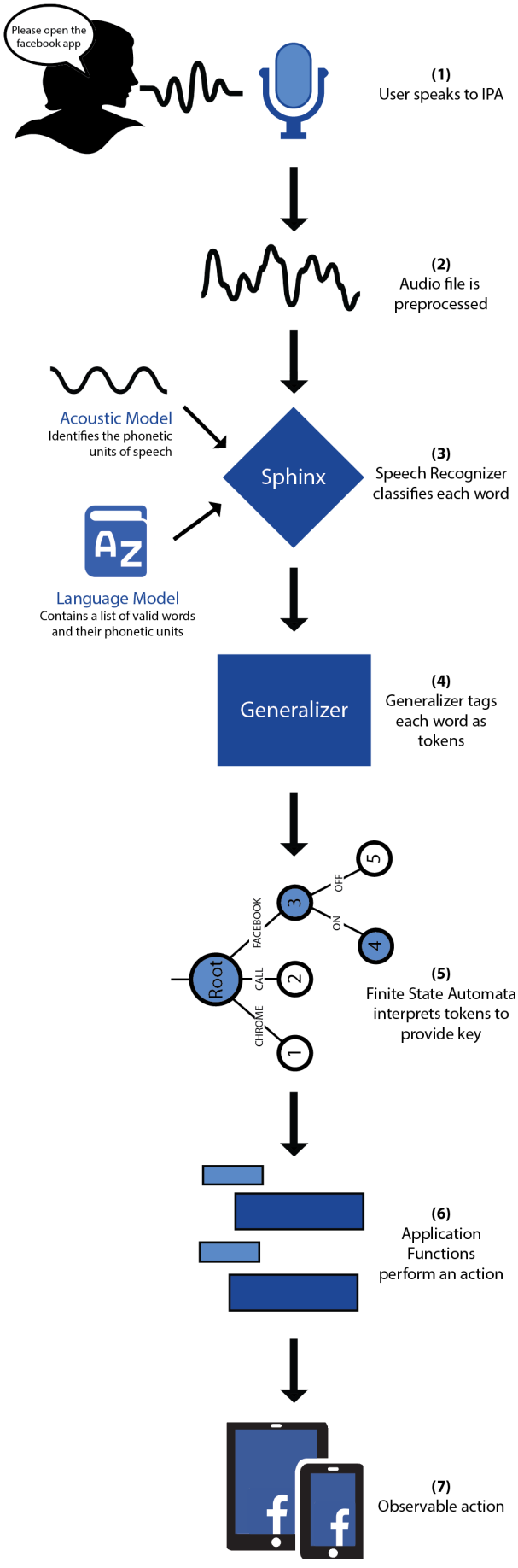

Fig. 2. Flow of information through the IPA

C. Step 4: Generalizer

The Generalizer allows the IPA to be multilingual. It is a function that takes each word from step (3) as input and then outputs a token or a null value for each word. An outputted token is a word which is a more "general definition" of the inputted word. e.g.

Generalize ("start") = "ON".

Here, the input word: "start", is generalized to the output token: "ON", which is a more general definition of "start".

The generalizer is important because there are multiple words (usually synonyms of each other), which imply the same meaning in a certain context. e.g., similar to "start" -

Generalize("open") = "ON"

Generalize("on") = "ON"

Similarly, words in different languages which are also synonymous can be generalized to the same token, e.g.

Generalize ("kholo") = "ON"

Generalize ("chalu") = "ON"

The above Bengali words: "kholo" (Meaning, "To open"), and "chalu" (Meaning, "Begin"), are synonymous - and can be Generalized to the token "ON". Also, any word that is not in the Generalizer is considered redundant and returns a null value. E.g., the word "please" is redundant, as it holds no meaning when delivering commands. Here lies the importance of the generalizer, by converting the input words to generalized tokens, the complexity of the FSA in step (5) is reduced greatly, and the FSA is able to operate in multiple languages as well.

TABLE I: LIST OF WORD/TOKEN IN THE GENERALIZER

Token	Word	Token	Word
ON	On	**OFF**	Off
	Open		Close
	Begin		Stop
	Initiate		Bondho
	Start		Thamao
	Launch	**CHROME**	Chrome
	Charo		Browser
	Kholo	**FACEBOOK**	Facebook
	Chalu		Ef Be
	Chalao		
	Jalao		

Table 1 contains a list of all the tokens used in our IPA. The following are some examples of the generalizer in action:

Generalize (" please open the facebook app ") = "ON", "FACEBOOK"

Generalize ("turn off facebook") = "OFF", "FACEBOOK"

Similarly, the two below sentences are direct translations of the above English sentences. Note, the "sequence of tokens" are not the same as above because "subjects" are placed before "actions" in Bengali (this is accounted for in Step 5).

Generalize (" doya kore facebook chalu koro ") = "FACEBOOK", "ON"

Generalize ("facebook bondho koro") = "FACEBOOK", "OFF"

D. Step 5: Finite State Automata

In the deterministic finite state automaton, traversal between the current state and an adjacent state can only take place - if the "required token" of one of the outgoing edges of the current state equals the "input token". When a "sequence of tokens", from step (6), is inputted to the FSA - depending on the sequence, the FSA is traversed and a particular end state is reached - and if the end state is an accepting state, the FSA sends instructions to the next step to perform an action depending on the specific end state.

We have designed our FSA to perform actions in response to the user's commands. In step (6), the Generalizer deduces the what the user is saying - and based on that information - in step (7) the FSA deduces what the IPA needs to do. Our design of the FSA is shown in Fig.3.

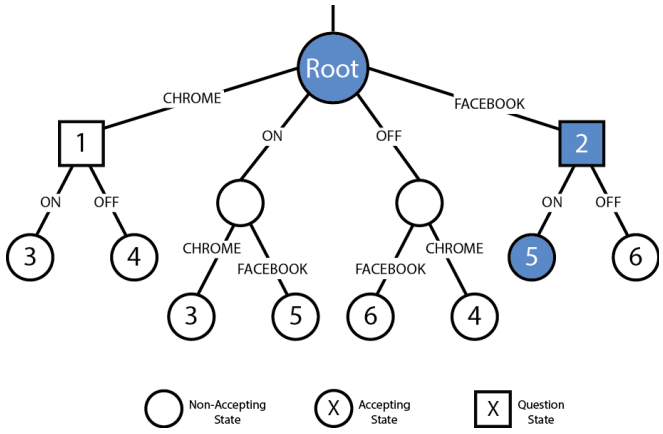

Fig. 3. The design of the Finite State Automaton

Furthermore, we have upgraded our FSA to have a third type of state: "Question State". This state is reached when the IPA cannot reach an accepting state because it does not have full information or did not understand the user's question - so it sends instructions to the output interface to ask the user for more information. As an example, consider from Fig.4, if the user says: "The facebook app" - the token, "FACEBOOK", is inputed to the FSA and state 2 is reached. Here, the FSA does not know what to do with the facebook app: "should the app be opened or closed". Hence state 2 is a Question State. After the user responds with an answer, the FSA starts traversal from the Question State (state 2, here), and either reaches an end state or is unable to comply.

When the FSA reaches an accepting state or question state, a "key" with a specific ID is generated and inputted to Step 6. This "key" acts as a signal to the Application Functionality on what action to perform.

E. Step 6: Application Functionality

The application functionality is the program which performs observable actions for the user on the user interface (which can be visual or audio), or can either observe actions taking place, or receive any response/errors from the IPA. Here, we simply created a switch-case where each case calls a different function that communicates with the hardware. The hardware then performs the action programmed in the particular function leading to Step (7): User observes action.

Using Fig.2, as a roadmap, the following is an example of a successful execution of a command -

Step 1: the user says: “please open the facebook app”.

Step 2: the audio file is preprocessed.

Step 3: The ASR classifies the action word: “open” and subject word: “facebook”.

Step 4: The Generalizer, from Table 1, generates the tokens: “ON”, “FACEBOOK”.

Step 5: The FSA, in Fig.3, reaches accepting state 5, and passes key_5 forward.

Step 6: The application functionality calls the function associated with key_5 to instruct the device’s operating system to initiate the Facebook App.

Step 7: The user observes the Facebook App being opened.

III. Limitations

The Sphinx-4 framework has its own audio noise reduction system, but it is limited in its abilities, and from our experience, attempting to add our own noise cancelling module increases the word error rate of existing acoustic models. The FSA needs to be designed by the developer - hence, when creating an FSA to process more complex commands, the developer needs to ensure that all possible cases are accounted for. However, for simple and direct commands and when datasets are not available, the FSA with Generalizer is a much more cost effective solution than machine learning. As for the Generalizer, two words may have the same spelling but have different meanings, e.g. “bat” can be a playing stick or a flying mammal. This ambiguity is not prevalent in domain specific applications, but may be an issue when scaling up.

IV. Conclusion

The approach specified in this paper is suitable for creating an Intelligent Personal Assistant which is capable of understanding domain specific direct voice commands from users. By selecting only the subject words and action words for training, the size required for the corpus is smaller. The generalizer and FSA allows the creation of an efficient solution to natural language processing when developers do not have access to sufficient training data. Although, this approach has limitations (i.e. it is useful only when the IPA needs to be domain specific), it can be further improved. Some recommended improvements: by using a larger audio/speech corpus for training the CMU Sphinx, the success rate can be increased significantly, the Generalizer can also be designed to be context sensitive to lessen the issue of ambiguity, and the addition of a voice synthesizer after the application functionality step can turn the IPA into a complete VUI.

References

[1] J. Kurpansky, "What Is an Intelligent Digital Assistant?", Medium, 2017.

[2] J. Iso-Sipila, M. Moberg, and O. Viikki, "Multi-lingual speaker-independent voice user interface for mobile devices," in ICASSP 2006. IEEE, 2006, vol. I, pp. 1081-1084.

[3] S. Springenberg, "Intelligent Personal Assistants. In: Speech Technology Seminar, Institute of Computer Science Hamburg, 2016, pp. 5-7

[4] M. Vasilache, J. Iso-Sipilä and O. iikki, "On a Practical Design of a Low Complexity Speech Recognition ngine", in Proc. Int. Conf. on Acoustics, Speech and Signal Processing, Montreal, Quebec, Canada, 2004, vol. 5, pp. 113-116.

[5] M. M. H. Nahid, M. A. Islam and M. S. Islam, "A noble approach for recognizing Bangla real number automatically using CMU Sphinx4," 2016 5th International Conference on Informatics, Electronics and Vision (ICIEV), Dhaka, 2016, pp. 844-849.

[6] P. K. Sahu and D. S. Ganesh, "A study on automatic speech recognition toolkits," 2015 International Conference on Microwave, Optical and Communication Engineering (ICMOCE), Bhubaneswar, 2015, pp. 365-368.

[7] R. Rangra and Madhusudan, "Natural language parsing: Using finite state automata," 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, 2016, pp. 456-463.

[8] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, Introduction to Automata Theory Languages and Computation.