Bilkent University

Department of Computer Engineering

Senior Design Project

Project Short Name: Clerk

Low-Level Design Report

Ahmet Malal, Ensar Kaya, Faruk Şimşekli, Muhammed Salih Altun, Samet Demir

Supervisor: Prof. Uğur Doğrusöz

Jury Members: Prof. Varol Akman and Prof. Çiğdem Gündüz Demir

Innovation Expert: Mehmet Surav

Low-Level Design Report
Feb 17, 2020
This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Contents

1 Introduction

Microsoft Word is one of the most popular word-processing programs around the world. It is used primarily to create various types of documents that you can print and publish, such as books, papers and reports. When you open a document in Microsoft Word, it can be edited using various features that Word provides. Currently, these features are accessible with use of some input devices, such as mouse and keyboard.

In this way, it is assumed that people who are going to use Microsoft Word, should able to use these devices in a conventional way. However, some people can't use these devices effectively or at all. Some people might not want to use them. Some people can't use their hands or are visually impaired. There should be a reliable way for these people who also may want to create documents and write reports, poems, and maybe a book.

Let us consider a visually impaired person, for instance. They should be able to create, delete, and save files; type and delete sentences; change the font and type of the text; change the position of the cursor and the alignment of the paragraph; change the color of the words or insert images into the document. Briefly, they should be able to use basic functionalities of a program like Microsoft Word. There is no tool currently available that provides use of major functionalities of Microsoft Word without the use of mouse and keyboard, or similar input devices.

Microsoft allows developers from around the world to develop applications that will extend the functionality of Word. These applications are called "Word add-ins". A Word add-in is an application which is essentially embedded inside of Word and can be launched from within a running Word instance. We want to build our application as an add-in for Word because it is the most popular text editor there is and it fits best with the nature of the work we want to do, which is to extend the conventional text-editor functionalities and make text-editing more accessible for people.

We propose an add-in for Word that will allow users to utilize Word features with voice commands.

1.1 Object design trade-offs

1.1.1 Usability vs Functionality

The users will be able to perform all functionalities of Clerk using voice commands from an audio input device. There will not be a need to use any other input devices except a designated key which distinguishes commands from normal inputs. This is especially important since part of our target users are people who aren't able to use these devices due to visual impairment. Yet, we should have enough functionality to achieve our purpose in the system. Therefore, we aims to balance usability and functionality since they are both important design goals of our system.

1.1.2 Extensibility Modularity vs Performance

Clerk will be sufficiently modular and additional functionality will be added easily. The application will be implemented such that replacing existing modules and adding new ones like different speech recognition API's and libraries will be easy. On the other hand, performance is crucial for our system as well. Performance of Clerk depends highly on the response time for the API calls that will be made for speech recognition. The user should not be waiting for longer than 3 seconds for the system to have acceptable usability, and by extension, the speech recognition service that will be used should not have any delays longer than this period. Another aspect that may be a bottleneck for performance is understanding and executing the commands. The user should not be waiting for longer than 5 seconds from the moment they start talking to see the effect of their speech on the document whether they have given a command or they are just in typing mode. Correctness is another aspect of speech recognition that should be considered as part of the performance of the system. User's speech should be converted to text with at least 90% accuracy. Hence, in our system we favor performance over extensibility and modularity.

1.1.3 Performance vs Cost

The user' should not wait more than 3 seconds when listening mode is active. However; the application performance is highly dependent on Google's Speech-to-Text and Text-to-Speech API's. The application is going to use "Speech Recognition with Data Logging opt-in for enhanced models which will cost \$0.006/15 seconds for Speech-to-Text API and "WaveNet voices" which will cost \$16.0/1 million characters for Text-to-Speech API. We have total \$1500 budget which is giving by Google for usage of the API's for students. In command mode, the application performance is dependent on Microsoft Office365 Word which always give response under 1 seconds. Thus our application should not wait a user more than 1 second when we got the command.

1.1.4 Availability vs Cost

The main processes such as Speech-to-Text and Text-to-Speech is done in a remote server. Therefore, the availability of the application is dependent on Google servers. System uptime should be high to prevent any inconvenience. The system should be highly modular to prevent the application going down when there is an issue or update in one of the sub-modules. Scheduled updates or maintenances of sub-modules processes should be announced at least 24 hours before and should take at most several hours.

1.1.5 Cost vs Accuracy

Accuracy is one of the most important design goals of Clerk. When we consider the performance of our system, accuracy would have the most importance. Nonetheless, cost is not something we should ignore. When we think about our possible customers, cost side of our system should be appealing to them. Hence, the accuracy of the system will highly depend on the models that we will use in the system. The better the accuracy, more expensive it gets. So, we will try to balance these aspects until we get acceptable accuracy and cost values.

1.2 Interface documentation guidelines

Class	ClassName
This is the class description.	
Attributes	
`attribute`	Attribute description.
Methods	
`method(args1, args2)`	Method description.

1.3 Engineering standards

Unified Modeling Language (UML) standard [1] has been used to model our classes. The Institute of Electrical and Electronics Engineers (IEEE) style [2] has been used to cite our references.

1.4 Definitions, acronyms, and abbreviations

API: Application Programming Interface
UI: User Interface
Speech Recognition: Identifying the content in a particular speech segment(e.g. recognizing the punctuation marks said in a speech).
BCP-47 Language Tag: Tag codes for identifying languages standardized by IETF [3]. For example, the tag "en-US" indicates the English language as used in the United States.

2 Packages

The application is designed according to a client-server architecture. Client side is responsible for communicating with the Word API, recording and sending sound signals to the server, parsing the received transcription in order to apply what the user requests to be done. The server side will handle communications with the speech recognition service.

2.1 Client

2.1.1 View

The view layer contains views and the events that result from user interaction with them.

mainView: The view class for the main screen. Lists the possible interactions such as start&stop listening mode, start&stop command mode, and settings. Shows the status of internet connection. Shows the icon of the add-in on the top.

howToUseView: The view class for the "How To Use" screen. Lists the available command options such as start&stop listening mode, select text, change the font etc. Has a

short tutorial video to increase usability of the add-in. Allows user to give feedback and rate the add-in in Microsoft Add-in Store.

optionsView: The view class for the "Options" screen. Has a scroll bar to change the voice threshold to decrease the effect of unwanted noise. Has an option to allow user to add their most used words to increase reliability of the speech recognition.

listeningView: The view class for the listening mode screen. It basically shows the server connection status and listening status such as when the user speaks a button turns green and when user is silent the button remains red.

commandView: The view class for the command mode screen. It shows the listened command and the outcome of the execution of that command such as successful or failure.

2.1.2 Logic

The logic layer contains the controller and model of our system. This layer will be a bridge between the view layer and the server.

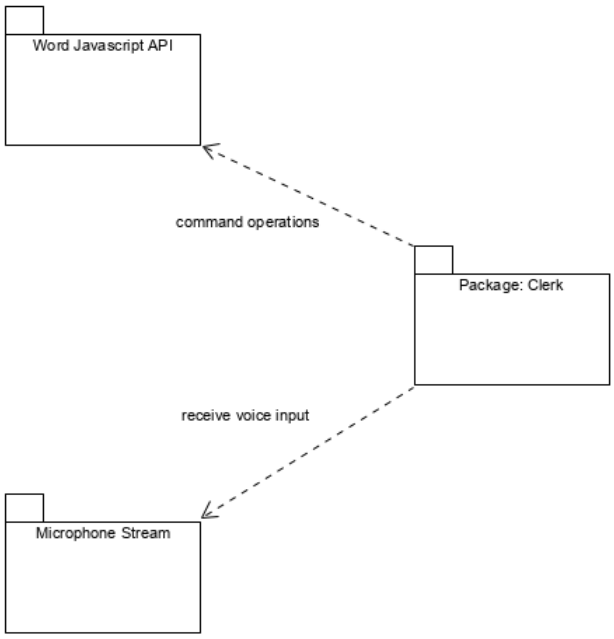

Figure 1: Abstracted view of client side logic. The package Clerk deals with sending voice to the server to transcribe it and parses commands. Communicates with Word Javascript API and the microphone stream on client side.

ClerkController: The class that manages the inputs coming from server and directs it to the view of Clerk and vice versa. It will also provide the state management. For instance, it will change the modes by listening the change mode event. ClerkController will also have instances of model classes and it will direct the speech input to these model by considering the current mode. For example, if the mode is listening mode, then it will direct the input to SpeechParser and then WordAPIService.

SpeechParser: This class will format the transcript of the speech. The server returns all the words in lowercase format and it doesn't recognise the punctuation words such as comma and semicolon. Basically, this class will parse the transcript and format it.

CommadParser: This class will convert the transcript of the speech returned from the server to an instance of WordAPIService executable Command instance. To give an example, if the user says "select last sentence", this class will return a Command instance where instruction is "select" and the inputs field is "last sentence".

Command: This class is `WordAPIService` executable object. It is a general class of all the instances of commands such as delete, insert and so on.

MicrophoneStream: This class will be a pre-processing stage of the audio input. When the input comes form microphone, the controller directs it to MicrophoneStream to clean the distortions in the audio input. For example, This distortion might be an unwanted voice such as music running in background.

WordAPIService: This is a wrapper class of MS Word API. It is a simplified version where the instructions that we will be using will be wrapped in this class to ease the usage of MS Word API. For example, some instructions might require a sequence of Word API method calls. This class will wrap these method calls in a single method.

2.2 Server

The server deals with speech sent from the client. When it receives the audio signal, the server first does some clean-up processing on the audio itself and then sends the signal to the external speech recognition service. The speech recognition service sends back the transcription for the audio which is then sent back to the client by the server.

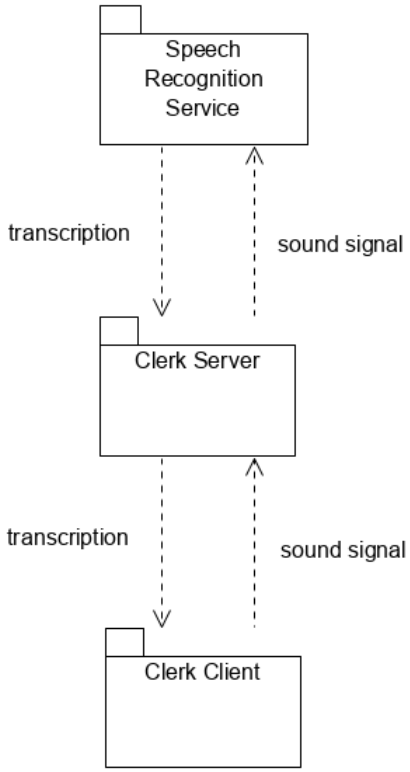

Figure 2: Abstracted view of server and client communications in Clerk. Sound signal travels from client side to the server which sends the signal to the speech recognition service which returns a transcription which is then returned back to the client. The speech recognition services currently considered are Google Cloud Speech and IBM Watson Speech-To-Text.

The server achieves this functionality with a speech client that communicates with the speech recognition service.

2.2.1 Speech Client

The speech client communicates with the speech recognition service. It has some configuration settings specific to the recognition service such as the language and grammar to be used, possibly some phrases that are wanted to be prioritized or more easily recognized; which are parameters to the speech recognition service that can be set.

In the actual application there will be three instances of speech client for each mode - sleeping, listening and command - of Clerk since different modes require different words or word groups to be emphasized. For example, in command mode the speech recognition service should emphasize command keywords over everything else.

3 Class Interfaces

3.1 Client

3.1.1 View

Class	**MainPage**
The class for the main page of the Add-in	
Attributes	
`INTERNET_CONNECTION`	`int` value for checking the status of internet connection
Methods	
`mainView`	The view called by `mainView` method. It shows the main menu of add-in which has an icon and several buttons such as start&stop listening, give command, and options.
`listeningView`	The view called by `listeningView` method. It shows the status of connection to the server during listening.
`commandView`	The view called by `commandView` method. It shows the the listened command.
`commandStatusView`	The view called by `commandStatusView` method. It shows the success or failure of the request command after execution attempt.
`optionsView`	The view called by `optionsView` method. It shows the options menu which consists of voice threshold scroll bar and most used phrases.
`howToUseView`	The view called by `howToUseView` method. It shows the possible commands and has a brief tutorial about usage of the add-in.

3.1.2 Logic

Class	**ClerkController**
Manages the communication between microphone and MS Word	
Attributes	
`mode`	It keeps the state information of Clerk system. It can be `speechMode`, `commandMode` or `sleepMode`.
Methods	
`setMode(mode)`	It is a set method of mode
`startListening()`	Clerk will start listening and the mode will be set to `speechMode`
`stopListening()`	Clerk will stop listening and the mode will be set to `sleepMode`
`execute(command)`	It executes the given command using `WordAPIService`
`processInCommandMode(text)`	It processes the given text using `CommandParser`
`processInSpeechMode(text)`	It processes the given text using `SpeechParser`

Class	**SpeechParser**
It parses the speech and corrects the punctuation and lowercase and uppercase characters	
Attributes	
`mode`	It keeps the state information of Clerk system. It can be `speechMode`, `commandMode` or `sleepMode`.
Methods	
`parse(text)`	This method will parse the given text and it will detect the sentences, names, and punctuation marks. Then it will call format method to format the text.
`format(text)`	This method will format the given text. It will check the special punctuation words like "comma", the names that needs to be in uppercase and it will format the text accordingly.

Class	**CommandParser**
It parses the command and returns a `Command` instance to `ClerkController`	
Methods	
`parse(text)`	This method will parse the given text and it will detect which kind of command it is and return the required command instance.

Class	**Command**
It is a model class of commands ordered by the user	
Attributes	
`executed`	It keeps the information of whether the command has been executed or not
`header`	It is the header of the command such as "delete" in "delete first paragraph" command
`params`	It is the header of the command such as "first paragraph" in "delete first paragraph" command
Methods	
`isExecuted()`	It is a get method of `executed`
`setExecuted(executed)`	It is a set method of `executed`

Class	**MicrophoneStream**
The class for signal processing before sending the data to `Server`	
Attributes	
`threshold`	The threshold of the signal. This variable will be used to get rid of the noise in background.
Methods	
`filter(signal)`	This method will filter the signal by the threshold and return the filtered signal.

Class	**WordAPIService**
This is a wrapper class for MS Word API operations.	
Methods	
`delete()`	Delete the selected text.
`delete(position)`	Delete the given position.
`deleteSentence(index)`	Delete the sentence with the given index.
`deleteParagraph(index)`	Delete the paragraph with the given index.
`insertText(from, to)`	Inserts the given text to cursor position.
`insertParagraph()`	Insert an empty paragraph to where the cursor is.
`select(from, to)`	Select the text from "from" position to "to" position.
`cut()`	Cut the selected text.
`paste()`	Paste to where the cursor is.
`makeBold()`	Make the selected text bold.
`makeItalic()`	Make the selected text italic.

3.2 Server

3.2.1 Speech Client

Class	**SpeechClient**
Communicates with the speech recognition service.	
Attributes	
`speechRecognitionConfig`	An instance of `SpeechRecognitionConfig` class which envelopes the configuration settings for the speech recognition service, such as the language and encoding style.
`audioStream`	An instance of `AudioStream` class which receives and processes the incoming audio from the client before sending it to the speech recognition service.
Methods	
`start(config)`	Initiliazes the client for speech recognition with the given configuration and starts sending back the transcription for incoming text.
`stop()`	Stops sending data to the speech recognition service.

Class	**SpeechRecognitionConfig**
Wrapper for configuration settings for the speech recognition service.	
Attributes*	
`API_KEY`	The API key for speech recognition service.
`encoding`	The encoding of the audio file used.
`sampleRate`	The sampling rate of the audio file used, in Hertz.
`languageCode`	The BCP-47 code of the language to be used in speech recognition.
`profanityFilter`	Binary value for whether a profanity filter should be used.
`speechContexts`	Phrases set to be more easily understood by the speech recognition service.

*: All attributes will have setter and getter methods which are omitted for the purposes of avoiding clutter.

Class	**AudioStream**
Used in processing incoming audio from client.	
Methods	
`process(byte[])`	Method for modifying the incoming signal.

4 Glossary

References

[1] “Unified Modeling Language,” http://www.uml.org/, Accessed: 2020-02-16.

[2] “IEEE Citation Guidelines,” https://ieeedataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf, Accessed: 2020-02-16.

[3] “BCP 47 - Tags for Identifying Languages,” https://tools.ietf.org/html/bcp47, Accessed: 2020-02-16.