2/19/2016

Speech Based Assistant for Ubuntu with integration to Gedit

(Software Requirements Specification)

Team 4

CS 243

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to present a detailed description of the Voice Assistant for Ubuntu. It will explain the purpose and features of the software, interfaces of the software, what the software will do, the constraints under which it must operate and restrictions/limitations under which user should deploy this software. This SRS serve as a basis for enhancement and provide a foundation for continued product evaluation. It reduces the effort on recoding, retesting and redesign.

1.2 Scope

This software system is a simple voice assistant for Ubuntu. This system will be designed to maximize the user's productivity by providing features to execute simple tasks such as schedule recurring tasks at a certain time, emulate clicks and keyboard presses, set various settings and system variables, search for files or applications etc.

Search feature of this system outputs results from Google search after listening to voice input. Dictation feature integrated as gedit plugin can be used to write emails, narrate essays and long documents in Gedit without touching the keyboard. It can't dictate special characters and supports only English language. It's not a real time dictation that is it listens a bunch of words and writes it in gedit.

2. The Overall Description

2.1 Product Perspective

When we use Ubuntu (or for that matter any Linux based system), we feel the need for a voice assistant which can do basic tasks for us, write texts for us, all using voice recognition and hence improve our productivity.

The main features of the project that are included:

(a**) Voice Based Assistant**: Can search, rename and delete local files, and modifying system settings. Display Google search results directly into the UI.

(b) **Dictation Tool**: Offers a dictation tool to use with Gedit Text Editor. It can write into a text file while you dictate.

(c**) Tasker**: Schedule time triggered tasks, emulate keyboard keys and mouse clicks, other timed-tasks can be launching applications and power controls.

This system is similar to Voice based assistant Cortana in Windows and also adds many "root" features like tasking that Cortana does not implement but the product does not give as smart results as Cortana because it does not user Machine Learning.

2.1.1 Software Interfaces

This software system consists of:

- *gedit plugin to call sphinx for dictation*
- *cron functionality to set recurring tasks*
- *file browser to show the relevant commands results*
- *web kit functionality to show google search result*

2.1.2 User Interfaces

Voice based interaction:

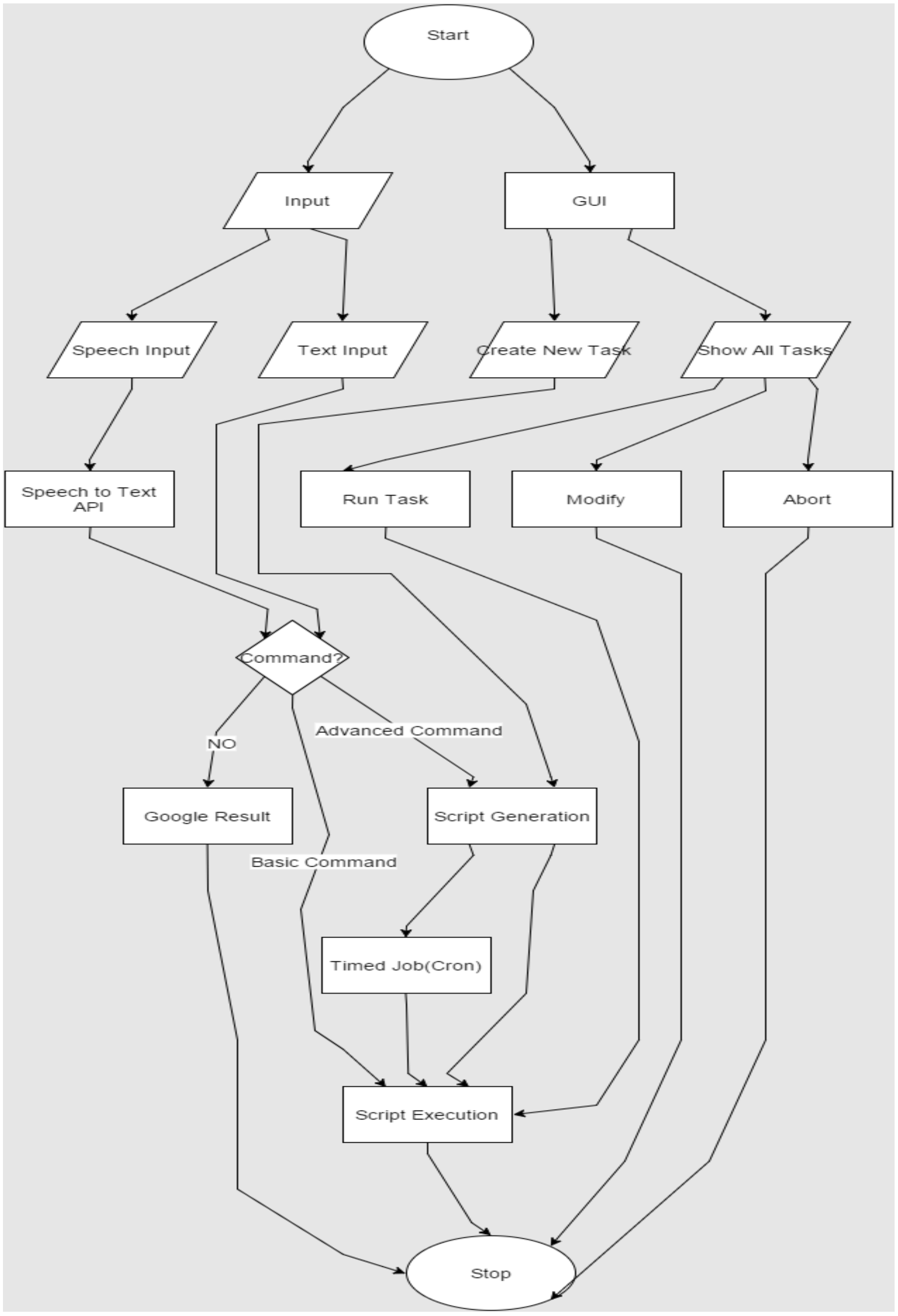
User can speak and input relevant commands. According to command type, the user will get results, or will be asked further to execute tasks.

GUI based interaction:

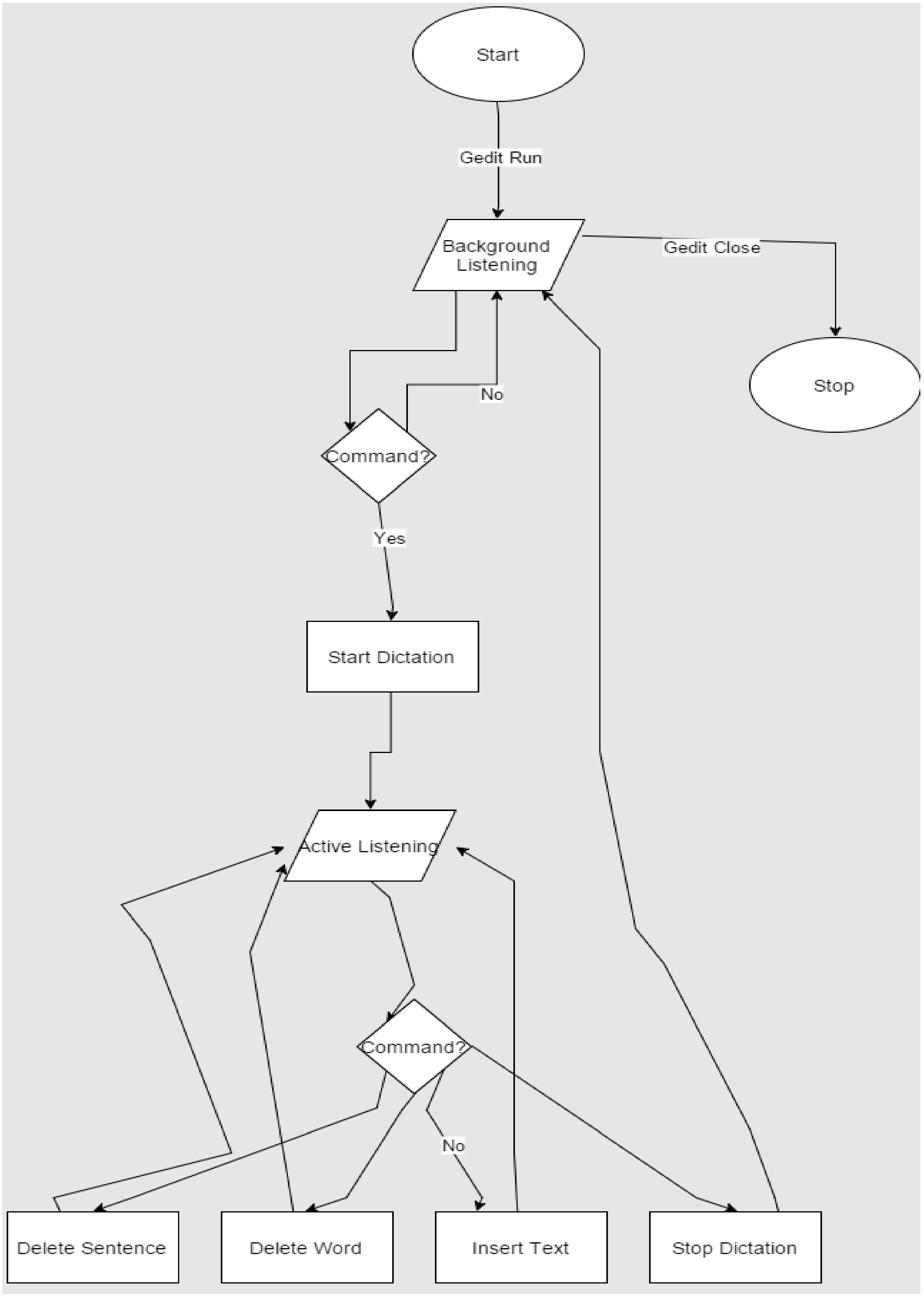
User will be able to click and select whichever tasks he will want to execute.

Command based interaction:

User will be able to execute tasks by directly inputting commands by typing them into the textbox.



User Interface of software system



User Interface of Gedit Plugin

2.1.3 Hardware Interfaces

This system has no specific hardware interface requirement except microphone to get voice input from user.

2.2 User Characteristics

Any user with basic education and expertise will be able to use this application. The user mush have a good grasp in English language (spoken and written) because the application's default language is English.

American English accent is mandatory considering our application's limitation to not properly recognize other accents. We recommend the user to try Americanize his/her accent to make good use of the software. Basic technical expertise and experience with computers is also required from the User to understand the functionalities that the application is providing are.

2.3 Product Functions

Use Case Scenario 1 – Execute a task (at present time or later)

Purpose	User want to execute a task.
Input	Name of the task (and time of execution) to be executed and as voice/text input
Consequence	Task will be executed (scheduled to run at execution time)
Pre-conditions	Software system is running.
Basic Flow	Sphinx takes voice as input and outputs name of the task, which is searched in the database and corresponding script will be executed.

Use Case Scenario 2 – Dictation

Purpose	User want to dictate some text.
Input	Voice input should be “Start Dictation” then text to be dictated.
Output	Dictated text in gedit document.
Pre-conditions	Gedit is running.
Basic Flow	Sphinx takes voice as input and outputs text which will be written in gedit doc.

Use Case Scenario 3 – Search

Purpose	User want to search a file locally or search on web.
Input	Text (or File name) to search as voice/text input
Output	Search result
Pre-conditions	Software system is running
Basic Flow	Sphinx takes voice as input and outputs text. This text will be searched on google and local files and result will be displayed in web view and gtk UI.

2.4 Constraints

1. *Hardware Limitations:*

Most inbuilt microphones are very limited in recording good quality audio, and generally are not capable enough to suppress noise.

2. *Software limitations:*

The voice based assistant is a standalone application and the gedit plugin is not a feature of it. Direct integration will force development into gedit application which is not desirable.

3. *Safety and security considerations:*

The application has root access and can wreck whole systems if checks are not in place and there is a user mistake.

2.5 Assumptions and Dependencies

Assumptions:

- *User is installing and using this application on Ubuntu operating system.*
- *The default audio system is Pyaudio (recommended for Ubuntu).*
- *The internet is secure.*

3. Specific Requirements

3.1 External Interfaces

The system need voice to perform operations. Sphinx base gets the user input as voice converts voice to text and output it to be used by other modules. In some cases the user is asked for addition confirmation like when deleting a file or changing sensitive settings. The search output is displayed to user in a web view. On either the execution of any command or failure of the command appropriate message is displayed to user. At the same time error.log and history.log file is being maintained. Error.log records all the error of the software. History.log keep record of the different tasks that got executed by the software like some_file.pdf deleted at 3am 19 Feb 2016.

3.2 Functional Requirement

Tasker via GUI

Brief Description: The User will access the application and select the “Set new task” or “Show all tasks” option. Depending on the choice, new page of the application will be launched.

If the user selects to set a new task, then new page with a form to create new task will be display where he’ll be prompt to enter/select task and it’s time of execution. Tasker will save the task and its relevant information in Database and execute it at its time of execution.

If he selects to see all tasks, then new page displayed will contain list of all the tasks which will be executed in future. These are the tasks user would have already set in past. Each task will show:

- it’s description (i.e. what it’ll do)
- time of execution
- option to change the time of execution
- option to modify the task
- option to abort the task

<u>Execute tasks via speech reorganization or command</u>

Brief description: The user will access the application and say “Start”. This will invoke sphinx to listen the commands/text and do necessary actions. Else user can manually enter the command/text just by typing.

After listening to text Tasker will respond. If the text contains a keyword (predefined word), then it will execute the corresponding action. E.g User can use this feature to launch the application “Firefox” by launching application and speaking “Start” >> “Open Firefox”.

If output text received from Sphinx is not a keyword and doesn’t exist in database (where tasks to be executed is listed), then Tasker will search the text on web and local file system. Tasker can start dictation in gedit. It’ll open gedit and append output of Sphinx inside gedit doc.

Dictation in Gedit

Brief description: The user will open gedit and after getting voice input “Start Dictation”, dictation will be started. Now sphinx will listen each sentence and append the output in gedit doc. There are some reserved commands like Delete, Delete word, Delete line. Delete command deletes a character at a time. Delete word deletes the closest word to the cursor position. Delete line deletes the line in which the cursor is placed. On end of each dictation the file should be auto saved and reloaded so that user need not to press reload button again and again and the text is rendered smoothly.

3.3 Performance Requirements

- User should be good in English (American accent) then our software easy to use for user.
- As software is operating system orientated so it is for single user.

3.4 Logical Database Requirements

Database stores all the information about tasks that Tasker will execute at certain time.

Relational database will store the following data in its table:

- Name of the task (Visible to user)
- Time of execution
- Path to the python/bash script which will execute the task

Database will be accessed every time when user:

- Set a new task
- Modify/abort scheduled task.
- Search something by inputting voice or text.

3.5 Design Constraints

Standard Python and Bash should be used to develop and extend the software. Readability must be ensured while developing the software. Use test cases after any modification to code. Write comments to code in specific format. Use variable names that are intuitive and specific to the vale it indicates.

3.6 Software System Attributes

3.6.1 Reliability & Availability

The application will run on user demand.

Handling the crashes:

1. Crash of the standalone application will cause the loss of the task currently being handled and will not affect gedit plugin or timed tasks.
2. Gedit plugin might crash gedit along with it, this case is handled by auto-backup feature of gedit which saves files at specific intervals of time.
3. Time triggered tasks are saved in default crone scheduler of the host's system, and will not be destroyed.

3.6.2 Security

To protect the software from accidental or malicious access, use, modification, destruction these features are present:

1. *Confirmation for sensitive tasks like deletion, setting system variables and other root tasks.*
2. *History and logs are saved with every tasks that is executed.*
3. *Looping tasks are also limited to a fixed number so there are no infinite loops.*
4. *Task limit is enforced so that the user cannot overload the system.*

5. Standalone application, gedit plugin and crone timed tasks are all handled by different modules and hence work independently.
6. Functions local to a module are chained together such that the next function will only be called if the previous one is successful.

3.6.3 Maintainability

- Software's dependencies are open-sourced and the code is well documented and commented, and hence the software is highly maintainable.
- Speech recognition library and Sphinx are both maintained and updated by their respective communities.
- Python, Bash, Cron are generally backwards compatible and hence any updates to them will not break any outdated software.

3.6.4 Portability

The core application is based on python and bash and the voice recognition library uses Pyaudio (recommended audio system for Ubuntu) as it's audio system for microphone recording.

3.7 Legal Requirement:

Software is based on following tools and their legal aspects:

Tools:

1. Linux Kernel: The Linux kernel is released under the GNU General Public License (GPL).So it is a free software and gives freedoms to run, study, share (copy), and modify the software.
2. Gedit: Gedit is free and open-source software subject to the requirements of the GNU General Public License (GPL).
3. MySQL: For developers of Free Open Source Software ("FOSS") applications MySQL is under the GPL that want to combine and distribute their application under FOSS.

Languages:

1. Python: The Python software Foundation License (PSFL) is a **BSD-style**, permissive free software license which is compatible with the GNU General Public License (GPL).

2. Bash: Bash also released under the GNU General Public License (GPL).
3. MySQL: For developers of Free Open Source Software ("FOSS") applications MySQL is under the GPL that want to combine and distribute their application under FOSS.

Libraries:

1. CMU Sphinx: CMU Sphinx is a permissive free software under BSD**-style** License, The only restrictions placed on users of software released under a typical BSD license are that if they redistribute such software in any form, with or without modification, they must include in the redistribution the original copyright notice.
2. Python Speech recognition: This program is also made available under the BSD license.