Living Requirements Document

Team: Voice++

Members: Petros Khachatrian (Lead), Chris Antanasian, Sheng Zheng, Julio Garcia, Belsin Barkhosir

Project name: InvocaBot

Current Revision: 1.3

Revision History:

<table>
<thead>
<tr>
<th>Revision</th>
<th>Description</th>
<th>Date Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>- Initial Version w/ Beta version of Sphinx4</td>
<td>01/29/2015</td>
</tr>
<tr>
<td>1.1</td>
<td>- Upgraded to Prealpha version of Sphinx4</td>
<td>02/17/2015</td>
</tr>
<tr>
<td></td>
<td>- Added Flowroute.com service to FreeSwitch</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>- Added more use cases and user stories</td>
<td>02/24/2015</td>
</tr>
<tr>
<td></td>
<td>- Added acceptance tests for each use case</td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>- Added more use cases and user stories</td>
<td>02/26/2015</td>
</tr>
<tr>
<td></td>
<td>- Added system models and functional requirements</td>
<td></td>
</tr>
</tbody>
</table>

Introduction:

InvocaBot is a hands-free agent for mobile devices and landlines that actively listens for a keyphrase made during a phone call. Upon a hearing a keyphrase, it executes the subsequent voice command. It receives the user's voice commands through Sphinx-4 and our own software written in JRuby. InvocaBot will aid the user by implementing various services provided by the mobile device it is being used on and also other online services. A key feature of InvocaBot is that it is always listening; that is, there is no need to activate it through a keypress. This streamlines online 0services while on a call.

Glossary of Terms:

Flowroute - an international telecommunications service. It is used to route and receive calls over SIP trunking system.

FreeSwitch - a scalable open source cross-platform telephony platform designed to route and interconnect popular communication protocols using audio, video, text or any other form of media.
**JRuby** - JRuby is an implementation of the Ruby programming language atop the Java Virtual Machine, written largely in Java.

**MGCP** - Media Gateway Control Protocol: a standard protocol for handling the signaling and session management needed during a multimedia conference.

**Phoneme** - any of the perceptually distinct units of sound in a specified language that distinguish one word from another, for example *p*, *b*, *d*, and *t* in the English words *pad*, *pat*, *bad*, and *bat*.

**SIP** - Session Initiation Protocol (SIP) is a telecommunications protocol for signaling and controlling multimedia communication sessions. It is commonly used for voice and video calls, as well as instant messaging over IP networks.

**Sphinx-4** - Adjustable and modifiable speech recognition system jointly designed by Carnegie Mellon University, Sun Microsystems laboratories, Mitsubishi Electric Research Labs, and Hewlett-Packard's Cambridge Research Lab.

**Telecom Server** - a server that allows us the enhanced capability to process incoming and outgoing audio through a phone call.

**System Architecture Overview: (UML)**

**High-level Overview**

[Diagram of system architecture]
System models:

In order to be able to help the user with a task, we need to be able to determine which API we will be using, and what task is being performed with that API. For this reason, we have a “TaskManager” class that is in charge of processing the command string recognized by Sphinx-4, and constructing the appropriate version of “APICommand” to be invoked. For example, if it is determined that the Google Calendar API is to be used, the “TaskManager” will create an instance of the “GoogleCalendarAPICommand” that will then be added to the “GoogleCalendarTaskManager”’s CommandQueue for it to be invoked.
Functional Requirements:

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Priority</th>
<th>Time Estimates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phoneme processing functionality</td>
<td>1</td>
<td>4 hours</td>
</tr>
<tr>
<td>Correct phoneme recognition</td>
<td>1</td>
<td>4 hours</td>
</tr>
<tr>
<td>PocketSphinx listening</td>
<td>3</td>
<td>2 hours</td>
</tr>
<tr>
<td>No caller query processing (1-way functionality)</td>
<td>3</td>
<td>2 hours</td>
</tr>
<tr>
<td>Pause recognition</td>
<td>1</td>
<td>3 hours</td>
</tr>
<tr>
<td>Task Description</td>
<td>Priority</td>
<td>Time</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>State switching</td>
<td>1</td>
<td>3 hours</td>
</tr>
<tr>
<td>Command processing</td>
<td>1</td>
<td>4 hours</td>
</tr>
<tr>
<td>Agent response tones</td>
<td>3</td>
<td>2 hours</td>
</tr>
<tr>
<td>Intermediary task manager</td>
<td>2</td>
<td>6 hours</td>
</tr>
<tr>
<td>Integration with Google Calendar API</td>
<td>1</td>
<td>12 hours</td>
</tr>
<tr>
<td>Functionality with multiple APIs</td>
<td>3</td>
<td>24 hours</td>
</tr>
<tr>
<td>Link between Sphinx4 and FreeSwitch</td>
<td>1</td>
<td>12 hours</td>
</tr>
<tr>
<td>Setup audio log for every call</td>
<td>3</td>
<td>3 hours</td>
</tr>
<tr>
<td>Output record of commands during call</td>
<td>2</td>
<td>4 hours</td>
</tr>
</tbody>
</table>

**User Stories/Cases:**

*Key:*

0. User Story
   a. User Case #1
   b. User Case #2
      i. Acceptance test

1. When the user begins a phone call, the system will connect to the user and begin listening:
   a. Agent has phoneme processing functionality.
      i. User starts talking and agent properly outputs what he/she said.
   b. Agent will always be “listening” for correct sequences of phonemes.
      i. User starts talking and agent properly outputs everything what he/she said.
   c. Implement “listening” to be done via PocketSphinx since it is lightweight and low power.
      i. At any moment after the user talks, the agent will begin processing.
      i. Other sequences of phonemes will not trigger the agent to activate processing.
   e. Agent must transfer audio processing to fully-fledged Sphinx4 for more accuracy in processing commands.
i. Agent begins recording and saving text after keyword until pause
   ■ Input: “hello everyone. Invocabot set meeting *pause*”
   ■ Output: “*blip*” sets meeting
f. The agent does not process queries from the caller.
   i. The caller talks and the agent does not process at all.
g. Enable agent to recognize pause.
   i. Give queries with random pauses and check if the queries are
      separated wherever pauses occurred.

2. The user gives a command to the agent and awaits a response:
   a. Sphinx4 should be tailing the audio output of the FreeSwitch.
      i. Give Sphinx4 a few queries with pauses in between, and check if output
         is occurring fast enough.
   b. The agent will begin caching all phonemes until a decently sized pause in
      sound is detected.
      i. Print out the buffer after each pause, and ensure it is the same as what
      has been said.
   c. When a pause in sound is detected, make agent play confirmation blip.
      i. Sound is played immediately after speech is processed.
   d. Sphinx4 should not be running at full power until the keyphrase “INVOCABOT”
      is detected.
      i. After the keyphrase “INVOCABOT” is detected, Sphinx4 should then
      begin to operate at full power to handle the incoming query.

3. The user hears the blip and awaits the agent to perform the given commands:
   a. Agent will convert speech to text.
      i. User starts talking and agent properly outputs what he/she said.
   b. Depending on the command, the Java implementation should sort out what
      happens using a case statement.
      i. User makes a query and the case statement will have the different API’s
      to handle the different queries.
   c. Our application should parse the data and hand it over to the API in the correct
      format.
      i. Check what format, for example, the Google Calendar API accepts, and
      pass the query in that format.

4. Agent should be able to freely swap between states.
   a. Agent may leave listening state and go to recording state.
   b. Enable agent to revert to listening state
   c. Agent will return to “listening” mode after the “function-complete” tone plays.
      i. If successful: “Hello Petros, nice day today. Invocabot, setup a meeting.
         *pause* Yeah, the weather was great. Invocabot, what is the temperature? *pause*” The agent will accept these commands and
         these commands only:
5. After receiving a command, the agent uses the proper API to do proper functionality.
   a. The TaskManager class will parse the text.
      i. It will output the keywords and ignore the fluff in the commands.
   b. The TaskManager class will determine the API after parsing.
      i. Given the keywords, it will see which API makes the most sense to use.
      ii. For example, if the user said “Set a meeting tomorrow at 9:00 AM,” the TaskManager will see the keywords meeting, tomorrow, and 9:00AM and see that the Calendar API makes the most sense.
   c. After determining the proper API to use, the TaskManager will create APICommand classes corresponding with the API.
      i. Creates a GoogleCalendarAPICommand class for a meeting.
   d. Passes off the command to proper API’s TaskManager class.
      i. Every GoogleCalendarAPICommand will be passed off to the GoogleCalendarTaskManager class.
   e. The proper API’s TaskManager class performs the functionality.
      i. User’s command should be properly done.
      ii. The meeting should be set for 9:00AM tomorrow and the user will be able to see it on screen.

6. The user is finished with a phone call.
   a. The agent should be closed when the call ends.
      i. If below output is clear, then end of agent is successful
   b. The applications enabled during the phone call will also be closed when the call ends.
      i. The applications are closed after the phone call.
   c. The agent should output the summary of activities on the phone screen during the conversation when the call ends: which apps have been used and what has been edited.
      i. Call Duration: 23 min 23 sec
         12:20 PM Start call, agent enabled.
         12:30 PM Calendar: Party at 10 tonight.
         12:32 PM Google Map: Search “43rd street”.
         12:43 PM End call, agent disabled.
   d. The activities of Invocabot should be saved in logs.
      i. Call with Chris(805-111-1234) 2/26/2015 12:20PM
         Call Duration - 10 min 3 sec
         12:20 PM Start call, agent enabled.
         12:24 PM Calendar: Party at 10 tonight.
         12:30 PM End call, agent disabled.

Call with Julio(805-222-1234) 2/27/2015 2:20PM
Call Duration - 4 min 3 sec
2:20 PM Start call, agent enabled.
2:22 PM Calendar: Meeting at 5 PM tomorrow.
2:24 PM End call, agent disabled.

e. The FreeSwitch should have an audio log of the conversation that can be downloaded by the user
   i. If the audio log does not exist, the FreeSwitch is not behaving correctly.

Prototyping Code + Test Cases:
https://github.com/pkhachatrian/voicePlusPlus

Technologies Employed:
- Sphinx-4 Voice Recognition API (Prealpha)
- JRuby 1.7.18
- Amazon Web Services (AWS)
- CloudOps private cloud
- FreeSwitch
- Flowroute