Jarvis Emulator
High Level Design
COP 4331, Fall 2015

Modification History

Version	Date	Who	Comment
v1.0	10/26/2015	Robin Schiro	Created document
v1.1	10/27/2015	Jimmy Lam	Added Design Issues
v1.2	10/28/2015	Manuel Gonzalez	Added High Level Architecture
v1.3	10/29/2015	Robin Schiro	Fixed spelling/grammar mistakes

Team Members:

- Jimmy Lam
- Julian Rojas
- Manuel Gonzalez
- Robin Schiro

Contents of this Document

I) High Level Architecture

a) **Major Components**

The Jarvis Emulator will be composed of 7 modules that will be implementing Microsoft .NET Observer Pattern to interact between them. In this pattern each object is either an observable or an observer, or both. An observable will send a message to all the objects that are currently "observing" its messages and these will respond to this message accordingly. Because the observable does not have any knowledge about its observers (except the fact that they are observers), this pattern allows for high modularization and maintainability.

The 7 modules are named as follows:

i) Face Recognizer Module
ii) Speech Recognizer Module
iii) Configuration Module
iv) Actions Module
v) RSS Feed Module
vi) UI Module
vii) Speech Constructor Module

b) **System Interfaces**

i) **User - System:**

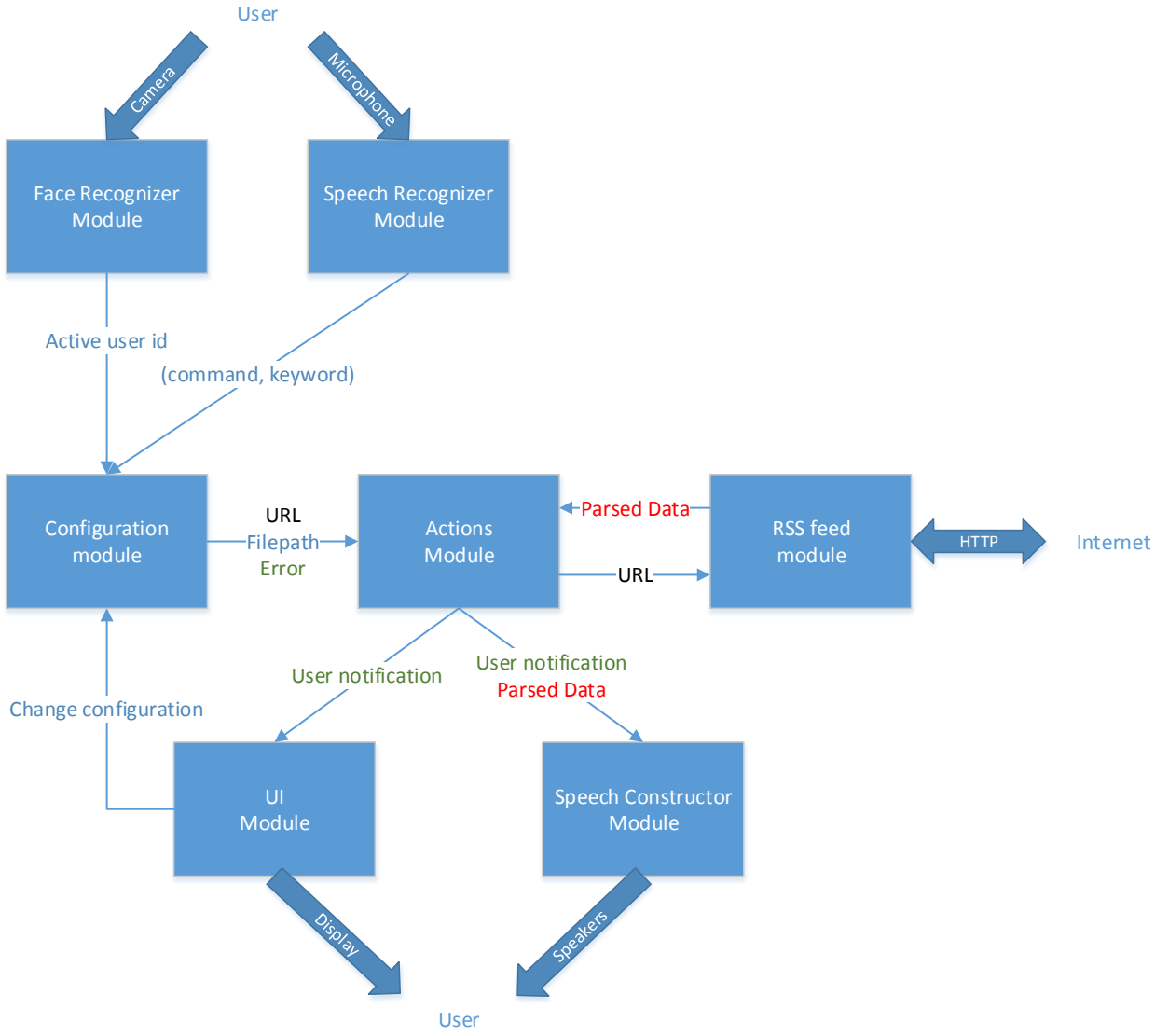
(1) Camera
(2) Microphone

(3) Display
(4) Speakers

ii) **Internet - System**
(1) HTTP

iii) **Module - Module**
(1) Active user id message
(2) Command, keyword pair message
(3) URL message
(4) Filepath message
(5) Error message
(6) Parsed Data message
(7) Change Configuration message
(8) User notification message

The following diagram shows the High-Level Architecture of our system, along with the interfaces between the modules inside the system, the system and the user and the system and the internet.



Design Issues

a) Design Trade-off

i) We will be using a publish-subscribe architectural design, which will allow us to have low coupling between our modules, as well as high flexibility of how our modules fit together (as they primarily just need to be able to publish or subscribe to one another). However, the tradeoff is that we may have some difficulty in the testing process as we must make sure each message is delivered properly between publishers and subscribers, which may cost us some testing time.

b) Prototyping

i) We will be using rapid prototyping so that we will be able to test out different algorithms and libraries while completing our project along the way, and saving us time. Each member will be able to work on his module by using these prototypes so that he can test his prototype to make sure his module works, and later on integrate his module with other team members through a subscription manager.

c) Technical Difficulties and Solutions

i) One technical difficulty with our project is the speech recognition module. The current library that is being used has about 50-60% accuracy, which isn't good. Various testing shows that the speech recognition is sometimes very off, or confuses user's speech with similar sounding words; for example, if the user says "Hi," the speech recognizer would hear "high" instead. A solution is to find a better speech library that would hopefully improve the speech recognition overall. We will also take into account of all similar sounding words, and add those words to the commands that Jarvis will recognize; for example, make sure Jarvis can recognize "Hi" and "high" as a greeting.

ii) While the base program that we are using has great facial detection, a technical difficulty is that its recognition system doesn't always work properly. For instance, Jarvis might confuse the user for someone else. One solution is to train Jarvis by having Jarvis take more pictures of the user. Another solution is improve the facial recognition algorithm. While the former option is easier, it may be bothersome and time consuming for each user to take so many photos of themselves. The latter solution would help make Jarvis a higher quality program, although much more difficult to implement.

d) Maintainability

i) Using the publish-subscribe design, for the project to run as a whole, each module communicates by published packets of data to each of its observers. For instance, if the user calls for weather updates, the speech recognition will parse the user's voice and publish the command to the action module. This module then calls for the RSS feeds from a weather website. These feeds will be published to the speech construction module to tell the user the weather. If something goes wrong with one module, the whole program might be affected. But since each module's own functions are independent, it should be easy to locate the issue and fix any faults that occur without interfering with other modules.

e) Reusability

i) With a highly decoupled design, each of our modules can function independently of one another. Therefore, each module can be reused in other programs, such as those requiring facial recognition, RSS feeds, speech construction or speech recognition.

f) Testability

i) Our project has a highly decoupled design, so each module can be tested independently from one another. Our project will also be testable when all our modules are integrated. However, since we are using the publish-subscribe design, once our module is integrated, we will test each module to make sure that all subscribers get the messages from the correct publisher, and all publishers send the right messages to the correct subscriber, making the testing process highly involved.

g) Risks

i) Using a publish-subscribe architecture, we may encounter issues in the future, such as if we decide to add new modules. Since the modules are subscribed to each other, we must make sure that future modules are subscribed to whichever modules that they need information from. As a result, we must update the subscription manager to establish communications with the added module so that Jarvis functions properly.

Jarvis Emulator
Detailed Design
COP 4331, Fall 2015

Modification History

Version	Date	Who	Comment
v1.0	10/26/2015	Robin Schiro	Created document
v1.1	10/27/2015	Jimmy Lam	Added my sequence diagrams
v1.2	10/28/2015	Robin Schiro	Added class diagram (with description)
v1.3	10/28/2015	Manuel Gonzalez	Added Answer Question and Greet User Sequence Diagrams
v1.4	10/28/2015	Julian Rojas	Added Detailed Design Issues. Added Sequence Diagrams.
v1.5	10/28/2015	Robin Schiro	Added my sequence/activity diagrams
v1.6	10/28/2015	Robin Schiro	Added Trace of Requirements
v1.7	10/29/2015	Robin Schiro	Performed grammar/spell checking

Team Members:

- Jimmy Lam
- Julian Rojas
- Manuel Gonzalez
- Robin Schiro

Contents of this Document

---

1) Detailed Design Issues

a) **Overview**

i) With the use of the publish-subscribe architecture we are able to increase testability (by isolating the different modules) and increase security (by only allowing subscribers to be able to receive the messages) but at the same time, we increase the complexity of it (implementing the subscription mechanism) and decrease a little bit the performance, since there will be higher latency due to message exchange.

b) **Re-Usability**

i) As mentioned before, since Jarvis' modules will be developed independently of each other, any programmer will be able to adapt some of them. The subscription manager and action module will not have as much flexibility as the other modules since they are very specific for our program.

c) **Performance**
   i) The publish-subscribe architectural style scales well for such a small application, considering the small number of publisher and subscriber nodes and low message volume.
   ii) Jarvis is being developed with performance in mind. There are many modules that require complicated algorithms where high accuracy and running time are top priority. The speech recognition requires a robust library where we are able to obtain precise commands, 70% accuracy is the minimum. The quality of the face recognition must also be developed with a minimum of 70% accuracy.

d) **Prototype**
   i) Prototyping is being done individually for faster developing time and easier implementation of functional requirements. A vast majority of the detailed design is based on the knowledge acquired as we build it.

e) **Prototype Issues**
   i) Since each one of the modules is being developed separately we have to take into account the integration of the publish-subscribe pattern and the data transfer system associated with synchronization of all the modules.
   ii) All of the prototypes required integration through a subscription manager; therefore implementing them will require knowledge of both the publishing module and the subscribing module.
   iii) A publish-subscribe system can't guarantee delivery of messages to any modules that might require such assured delivery or the publisher might send incorrect messages to the wrong subscriber.

2) Detailed Design Information

a) **Class Diagram Description:** Each class in the Jarvis Emulator roughly corresponds to the various "modules" outlined in the high level architectural design. The following are descriptions of the functionality of each class and its observation relationship with other classes:
   i) *Face Detector:* Uses the Adaboost algorithm contained within the OpenCV API to determine the locations of faces in each frame of video captured by the webcam. This class is used to pass raw face images to the Face Recognizer for recognition processing. Because detection must always occur before recognition, this is the only class that has access to the Face Recognizer class. This class is observed by the Configuration Manager, which must know who the active user is at all times. It is also observed by the Main Window, which displays the processed feed from the webcam when the user opens the 'Video Feed' tab.

   ii) *Face Recognizer:* Takes input images from the Face Detector in conjunction with the training data loaded as eigen images to determine the active user of the application.

   iii) *Configuration Manager:* Stores details related to the configuration of the application and of each user to a file. Also retrieves this information when it's needed by the other modules. Specifically, the Configuration Manager is observed by the Face Detector, Speech Recognizer, and the Main Window. It provides the path to the training data to the Face Detector (to be used by the Face Recognizer) as well as the guids of the associated users. Additionally, this class sends user specific command objects that it has translated from the Speech Recognizer back to the Speech Recognizer. The Main Window uses the loaded data from the

Configuration Manager to display configuration settings to a user.

iv) *User:* Stores identification information related to each user. Additionally, it stores a dictionary that the Configuration Manager uses to translate command objects provided by the Speech Recognizer (e.g. “reddit” might be translated to “www.reddit.com/funny.rss”).

v) *Main Window*: The visual user interface of the application. The Main Window is used to configure user profiles and the general application settings. It is observed by the Configuration Manager, which saves the configuration settings inputted by the user.

vi) *Action Manager*: The “command executor” of the application. The Action Manager performs various operations based on <command,command object> pairs received from the Speech Recognizer. Specifically, these operations include opening/closing applications, logging out, taking pictures of the user, and retrieving updates from websites relevant to the user (it uses the RSS Manager for this one). It is observed by the Main Window, Speech Constructor and Face Detector. The Action Manager passes status notifications to the Main Window. Additionally, it passes text output from its operations to the Speech Constructor to be processed and verbally expressed to the user. Finally, the Face Detector is an observer of the Action Manager so that it can be used to take snapshots of the active user’s face on command.

vii) *RSS Manager:* Retrieves RSS feeds from the URL specified by the Action Manager and parses the retrieved information. This class is used only by the Action Manager and is therefore not included in the publish-subscribe system. It sends parsed information back to the Action Manager.

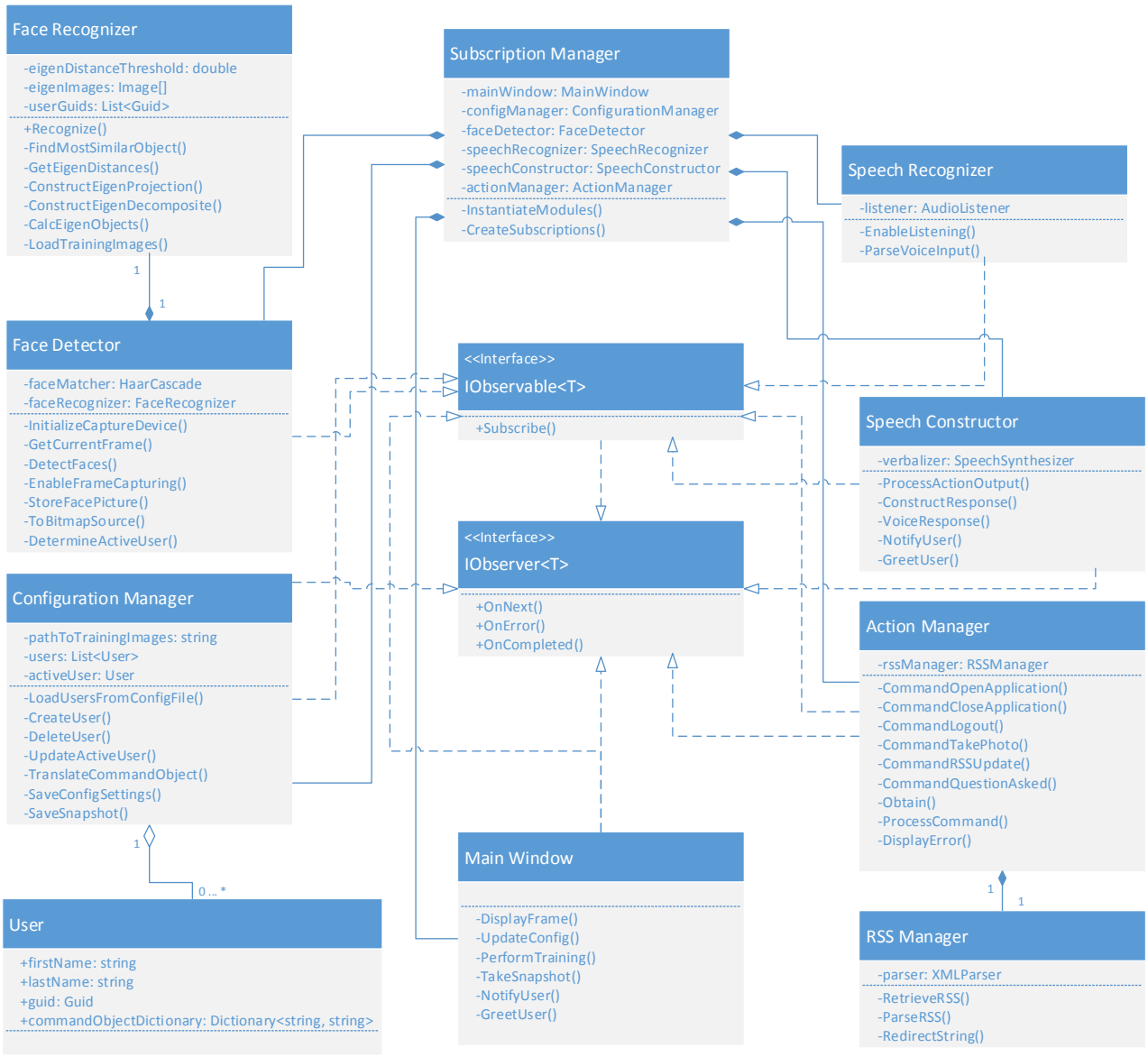
viii) *Speech Constructor*: Processes text output from the Action Manager, constructs natural-sounding sentences, and verbalizes those sentences to the user.

ix) *Speech Recognizer:* Listens for commands provided by the active user of the application. The user uses a set of trigger words (e.g. “Ok Jarvis”) to enable the “listening”. Anything the user says directly after verbalizing the trigger words is recorded, converted to text, and parsed for <command, command object> pairs. The Speech Recognizer is observed by the Action Manager, which receives <command, command object> pairs and performs the corresponding actions. It is also observed by the Configuration Manager, which receives the raw converted text and the dictionary of the active user to aid in the parsing.

x) *Subscription Manager*: Ties all of the modules together. This class instantiates the modules and set ups the subscriptions between them as per the Observer Pattern described here.

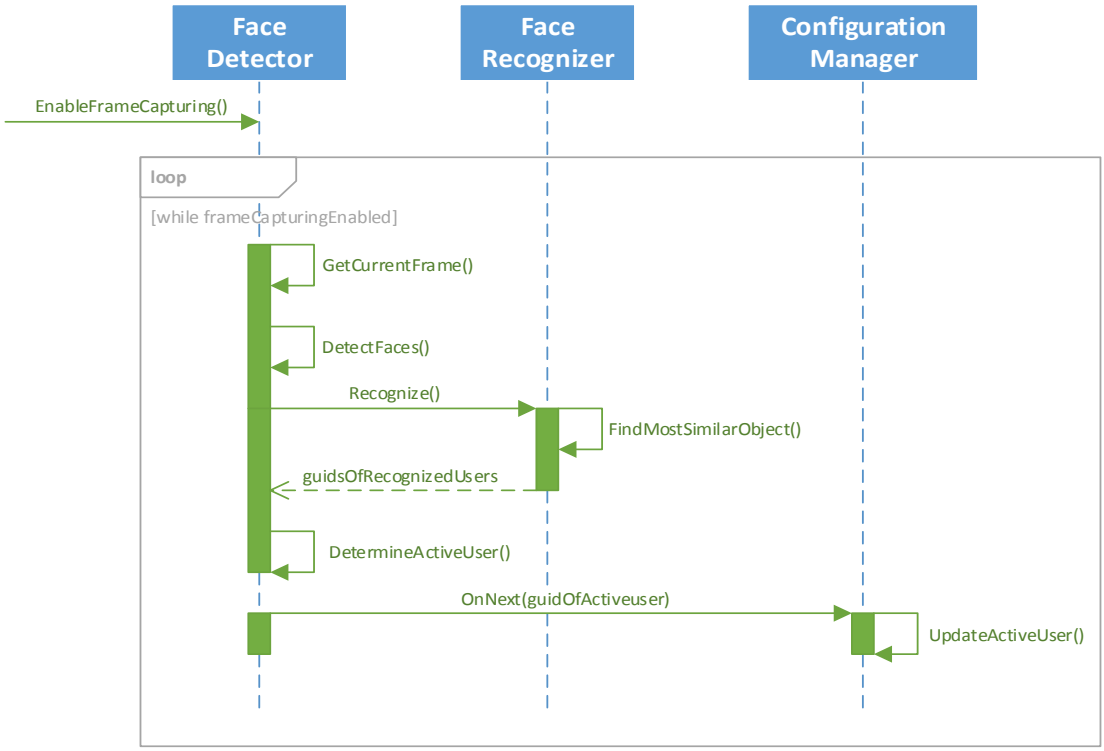
xi) *IObservable<T> and IObservable<T>:* These are interfaces provided by the .NET Framework used to implement the Observer Pattern.

b) **Class Diagram:**

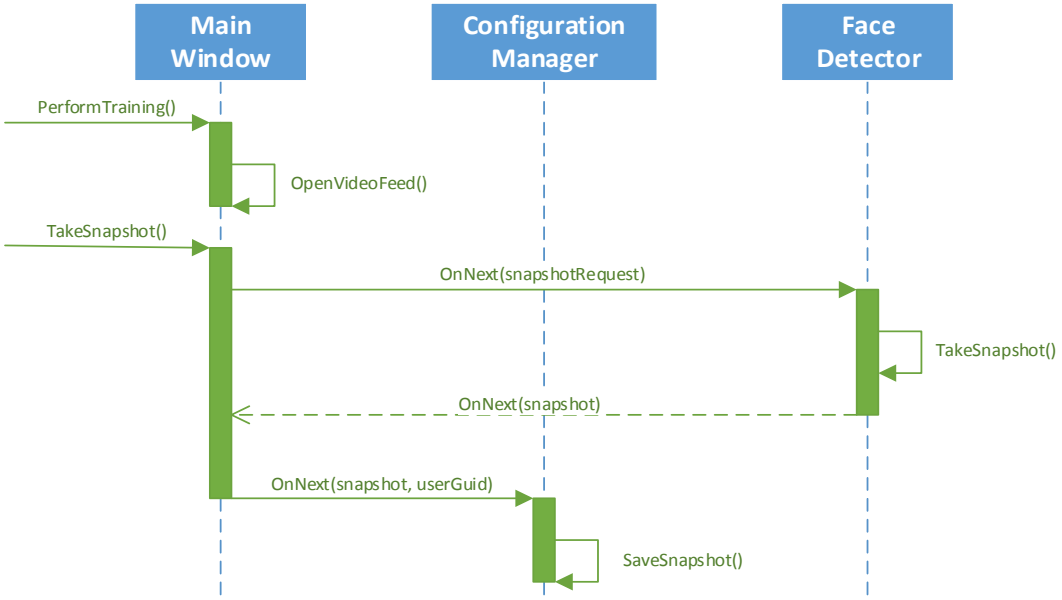


c) **Sequence and Activity Diagrams**

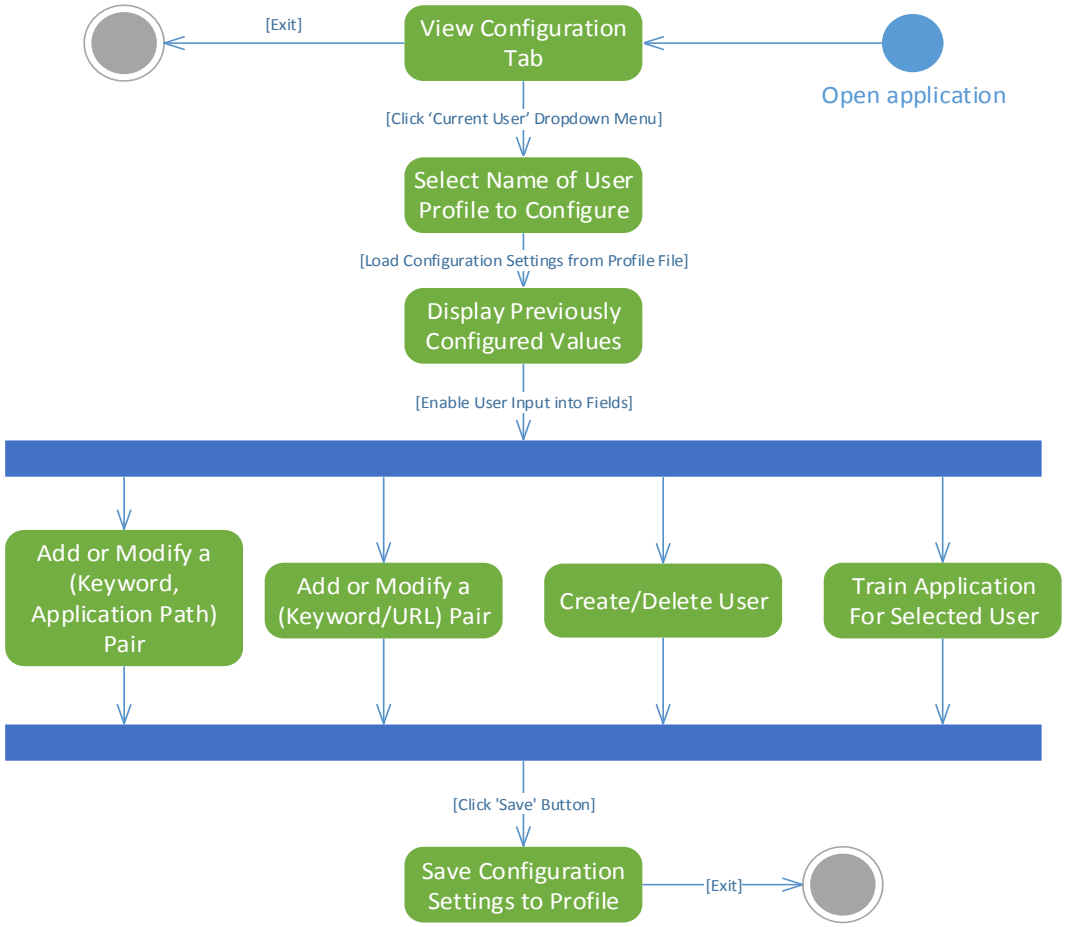
1) **Detection and Recognition**: This sequence diagram depicts the operations taken to process frames from the user's webcam. During processing, faces in the frame are detected, isolated, and compared against a database of training data to perform recognition. The active user is determined based on the result of the call to Recognize(). This information is relayed to the Configuration Manager.



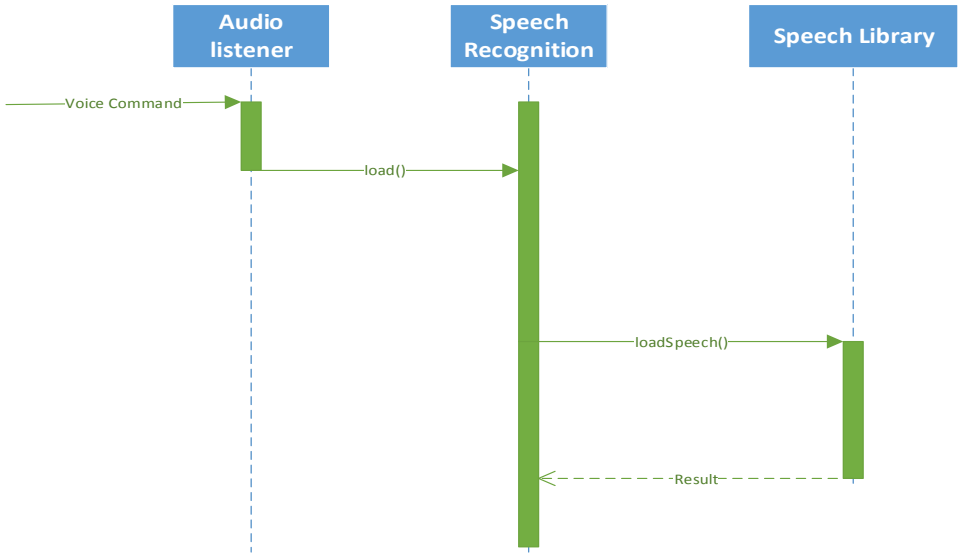
2) **Training the Jarvis Emulator for a User**: This sequence diagram shows how a user would "train" the Jarvis Emulator to recognize his/her face. In the Configuration tab, he clicks the 'Train' button and proceeds to take snapshots of himself (recommended total of at least 50).



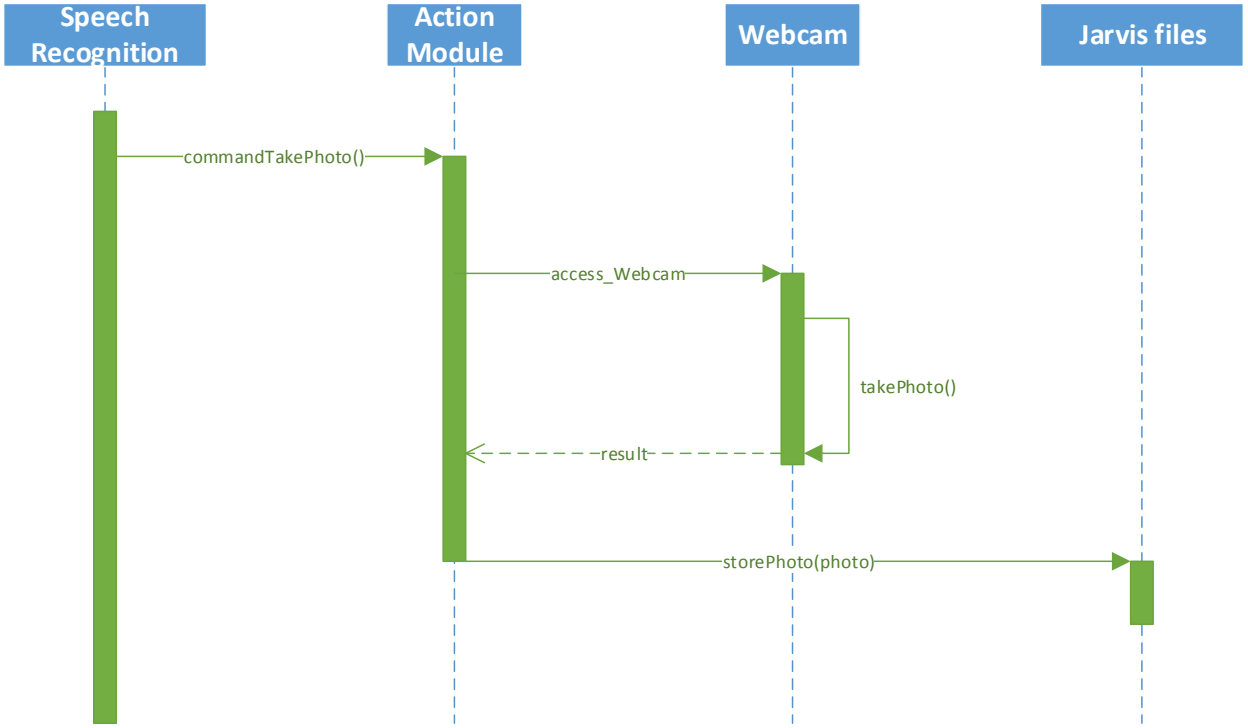
3) **Modify Configuration**: This activity diagram shows how the user can input information into the Configuration tab of the Main Window, how this information is saved, and how it is displayed.



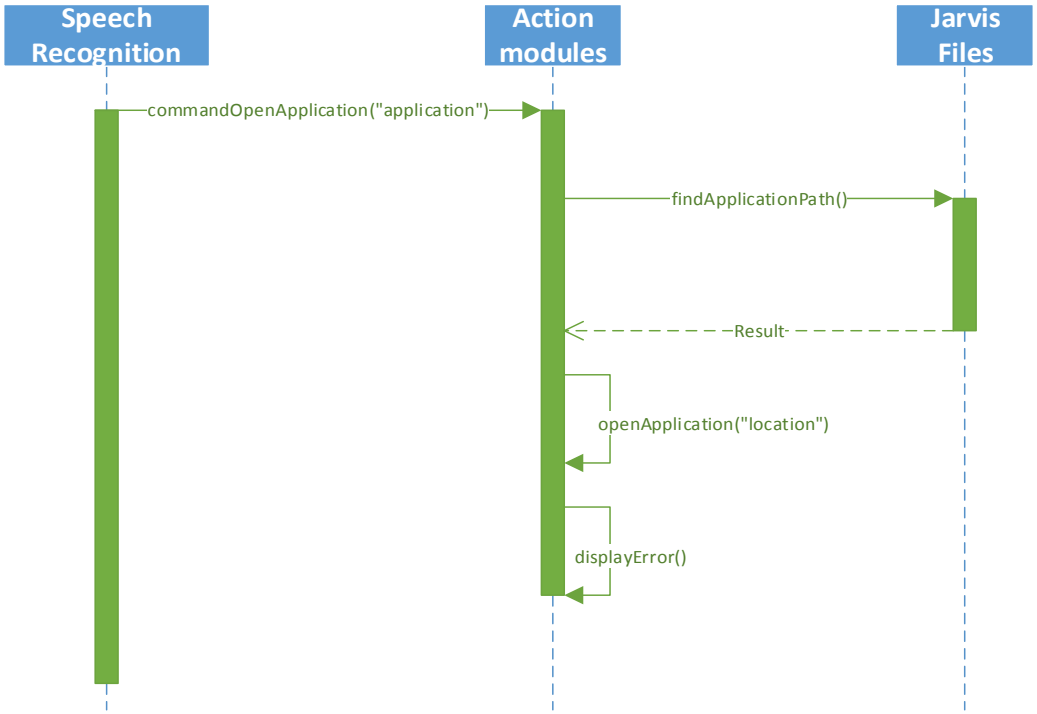
4) **Speech Recognition**: The user provides a voice command which is taken in by the Audio Listener class. It is then passed to the Speech Recognizer class which looks through the speech library and returns its result.



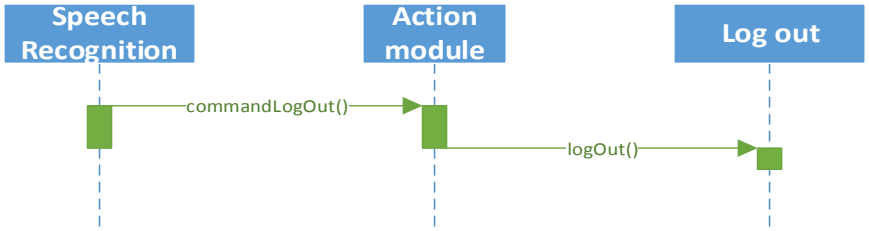
5) **Taking Picture**: The Speech Recognizer calls the Action Module, which accesses the user's webcam, taking a picture, and returning the result to the Action Module, where it will proceed to store the picture in Jarvis's file folder.



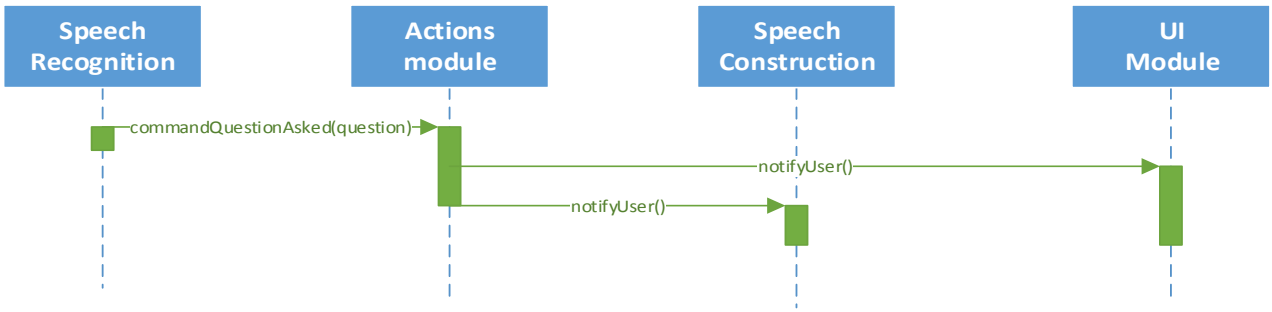
6) **Open Application**: The Speech Recognizer publishes a message to the Action Module to open the specified application. The Action Module checks Jarvis' files to find the location of the application (given by the user), and returns the location for the Action Module to open. Otherwise, if the application was not specified, an error will be displayed.



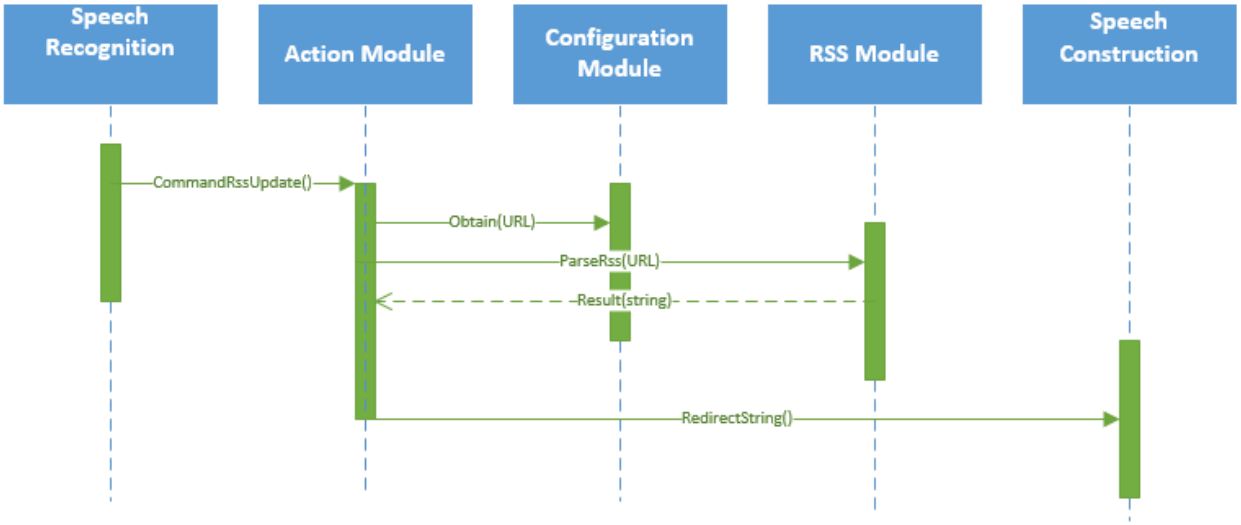
7) **Log Out**: The Speech Recognizer publishes a command to the Action Module for logging out. It will then call the log out function, logging the user out of his or her computer.



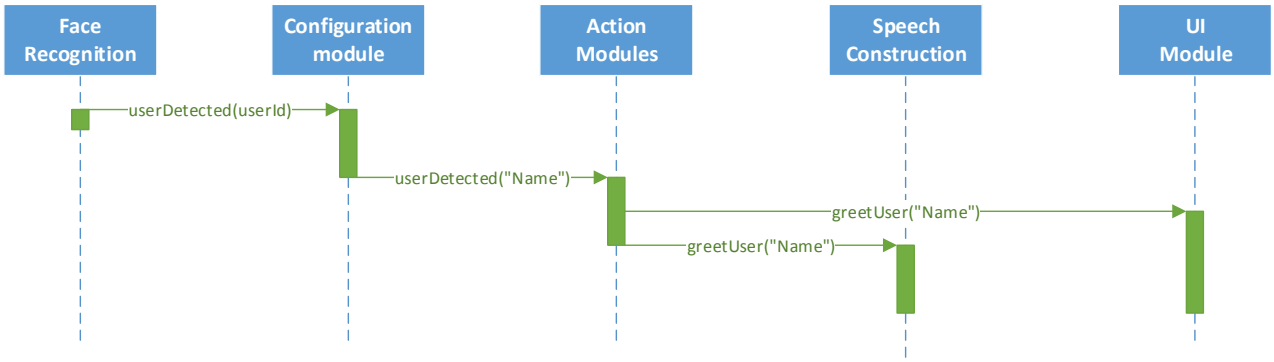
8) **Answer Question:** The Speech Recognizer recognizes one of the supported user questions, calls the Action Module, which in turn will issue a notification to the speech constructor and UI modules so they output the answer to user.



9) **RSS Feed:** The Speech Recognizer calls the Action Module. The Action Module will obtain the URL needed (previously given by the user in the Configuration Module) and send it to the RSS class. This class will then parse the RSS Feed and get the information needed. The RSS Module will return a string to the Action Module where it will be directed it to the Speech Constructor so that it can be outputted to the user.



10) **Greet User:** The Face Recognizer recognizes a registered user and calls the Configuration Module, which calls the Action Module, which issues a notification to the speech constructor and UI so they output a greeting to the user.



3) Trace of Requirements to Design

Number	SRS Functional Requirement	Designed Implementation
1	The frames of the feed are processed under eigenanalysis using the OpenCV library.	Sequence Diagram (1): Detection and Recognition
2	The application can track the position of the user's face.	Sequence Diagram (1): Detection and Recognition
3	The user interface allows the user to "train" the application for facial recognition.	Sequence Diagram (2): Training the Jarvis Emulator for a User
4	The user interface allows the user to save and update set of configuration settings based on selections made in the 'Configuration' tab.	Activity Diagram (3): Modify Configuration
5	Jarvis shall recognize voice commands of the user with the window's speech library and should have 70% accuracy.	Sequence Diagram (4): Speech Recognition
6	Jarvis shall open other applications based on user command.	Sequence Diagram (6): Open Application
7	Jarvis shall log the user out of their computer based on user command.	Sequence Diagram (7): Log Out
8	Jarvis shall take a picture of the user for the user and store the photo.	Sequence Diagram (5): Taking Picture
9	Given specifically formatted data, the application should generate human language speech that summarizes and describes the data.	Sequence Diagram (8): Answer Question
10	The application should answer basic user questions.	Sequence Diagram (8): Answer Question
11	The application should greet the user through the speakers upon user recognition.	Sequence Diagram (10): Greet User
12	Jarvis should be able to subscribe to Website RSS Feeds.	Sequence Diagram (9): RSS Feed