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 |

**Team Members:**

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

**Contents of this Document**

Design Issues

Detailed Design Information

Trace of Requirements to Design

1. **Detailed Design Issues**
   1. **Overview**
      1. 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 reach 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.
   2. **Re-Usability**
      1. As mentioned before, since Jarvis’ modules will be developed independently of each other, any programmer can be able to adapt some of them. The action module will not have the flexibility as the other modules since it is very specific for our program.
   3. **Performance**
      1. Publisher-subscriber scales well for such small application, with a small number of publishers and subscriber nodes and low message volume.
      2. 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, 80% accuracy is the minimum. The quality of the face detection must also be developed with a minimum of 70% accuracy.
   4. **Prototype**
      1. 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.
   5. **Prototype Issues**
      1. 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 tight coupling associated with synchronization of all the modules.
      2. All the prototypes required integration through an action module; therefore implementing them will require knowledge of both the publishing module and the subscribing module.
      3. A pub/sub 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**
   1. **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:  
      1. *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. 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. 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.
      2. *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.
      3. *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 that 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.
      4. *User:* Stores identification of each user in addition to a dictionary that the Configuration Manager uses to translate command objects provided by the Speech Recognizer.
      5. *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.
      6. *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 and the Speech Constructor. The Action Manager passes status notifications to the Main Window. Finally, it passes text output from its operations to the Speech Constructor to be processed and verbally expressed to the user.
      7. *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.
      8. *Speech Constructor*: Processes text output from the Action Manager, construct natural-sounding sentences, and verbalizes those sentences to the user.
      9. *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 direct after verbalizing the trigger words is recorded, converted to text, and parses the text 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.
      10. *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](https://msdn.microsoft.com/en-us/library/ee850490%28v=vs.110%29.aspx?f=255&MSPPError=-2147217396).
      11. *IObservable<T> and IObservable<T>:* These are interfaces provided by the .NET Framework used to implement the Observer Pattern.
   2. **Class Diagram:**



* 1. **Sequence and Activity Diagrams**

1. **Detection and Recognition**: This sequence diagram depicts operations taken to process frames from the user’s webcams. During processing, faces in the frame are detected, isolated, and compare 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.



1. **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).

****

1. **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.

****

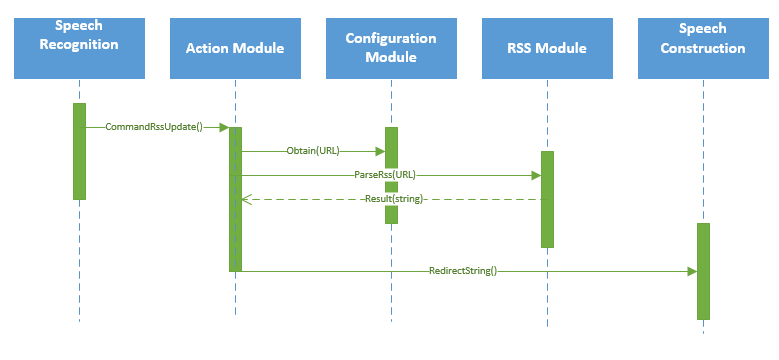
1. **Speech recognition**: The user provides a voice command which is taken in by the audio listener class. It is then passed to the Speech Recognizer class which looks through the speech library and returns its result. 
2. **Taking Picture**: The Speech Recognizer calls the Action module, which accesses 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. 
3. **Open Application**: The Speech Recognizer calls the Action module to open the specified application. The Action module checks Jarvis’s 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. 
4. **Log Out**: The Speech Recognizer calls the Action module the command for logging out. It will then call the log out function, logging the user out of his or her computer.



1. **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.



1. **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.



1. **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.



1. **Trace of Requirements to Design**